To the Graduate Council:

I am submitting herewith a thesis written by Stephen W. Parker entitled "Microcomputers—Hardware, Software, and Applications." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Robert E. Bodenheimer
Major Professor

We have read this thesis and recommend its acceptance:

R. C. Gonzalez

R C W Rochelle

Accepted for the Council:

Vice Chancellor
Graduate Studies and Research

MICROCOMPUTERS—HARDWARE, SOFTWARE, AND APPLICATIONS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee

Stephen W. Parker

August 1975

1265300

ACKNOWLEDGMENTS

The author would like to thank Dr. Robert E. Bodenheimer for the time he took to help develop and proofread this thesis. Appreciation is extended to Dr. R. C. Gonzalez and Dr. R. E. Rochelle for serving on the author's committee. Gratitude is extended to B. A. Fittes for his help in developing the applications. Appreciation is also extended to the Naval Weapons Laboratory for supporting the author during his entire Masters Degree program. Finally, the author would like to thank his wife, Melissa, for her patience and help during the development of this thesis.

ABSTRACT

A description of the various types of microcomputers is presented in order to compare the features of the microcomputers presently available. The microcomputers are grouped into the following categories: (1) Calculator-oriented, (2) Fixed instruction-set, and (3) Microprogrammable microcomputers. Both the architecture and the software of the microcomputers are presented to develop the comparison. Three applications are described to exemplify the use of microcomputers in different situations.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
II.	ARCHITECTURE	7
	Calculator Oriented Microprocessors	7
	Rockwell PPS-4	7
	Fairchild PPS-25	11
	Intel MCS-40	14
	Fixed Instruction Microprocessors	24
	Fairchild F-8	26
	Motorola 6800	29
	Intel 8080	43
	Microprogrammable Microcomputers	60
	National IMP-16C	60
	Phase 1 control bits	67
	Phase 3 control bits	67
	Phase 5 control bits	67
	Phase 7 control bits	67
	Summary	70
III.	SOFTWARE	73
	Data Movement	73
	Addressing modes	75
	Immediate addressing	75
	Indexed addressing	75

CHAPTER PAGE

Indirect addressing 75
Registers used 76
Data Manipulation 79
Decision and Control 83
Conditional instructions 85
Subroutines 85
Four-bit control instructions 88
Unique control features 89
Input/Output 90
Microcomputer Coding 92
Assemblers 92
Editors 99
Loaders 102
Compilers 104
Microprogramming 104
IV. APPLICATIONS 108
Desk Calculator Application 108
Traffic Controller 121
ADC Application 135
V. CONCLUSION 142
LIST OF REFERENCES 143
APPENDICES 146
A. INSTRUCTION SET OF THE INTEL 8080 MICROCOMPUTER 147
B. INSTRUCTION SET OF THE MOTOROLA 6800 MICROPROCESSOR . . . 155

CHAPTER PAGE

C. INSTRUCTION SET OF THE NATIONAL IMP-16C MICROCOMPUTER . . 158

D. ASSEMBLER ERROR MESSAGES USED BY THE INTEL 8080 160

E. EDITOR COMMANDS FOR THE INTEL 8080 162

F. ASSEMBLED PROGRAM OF THE DESK CALCULATOR APPLICATION . . . 167

G. ASSEMBLED PROGRAM OF THE TRAFFIC CONTROLLER APPLICATION . 183

H. ASSEMBLED PROGRAM OF THE ADC APPLICATION 188

VITA 190

LIST OF TABLES

TABLE		PAGE
1.	Example of PPS-25 Nine-Digit Floating Point Arithmetic	15
2.	Functional Description of Each Pin of the Intel 4004	20
3.	Functional Description of Each Pin of the Motorola 6800	32
4.	Functional Description of Each Pin of the PIA	37
5.	Selection of PIA Locations	42
6.	Control Register Format	42
7.	Control of the Interrupt Input Signals	44
8.	Control of Interrupt Inputs, CRA-5 is "0"	45
9.	Control of Bits Three, Four and Five	46
10.	Functional Description of Each Pin of the Intel 8080	51
11.	Description of Status Information Bits of the Intel 8080	54
12.	ALU and Control Function Bits	68
13.	Comparison of the Architectures of Various Microcomputers	71
14.	Intel 8080 Instructions to Perform Immediate Addressing	75
15.	An Example of Indexed Addressing Using The IMP-16C Instruction Set	76
16.	Intel 8080 and National IMP-16C Instructions to Perform Indirect Addressing	77

TABLE PAGE

17. Data Manipulation Capabilities of Microprocessors 80

18. Example of Operations of a DAA Instruction 81

19. Examples of the Logic Operations, Complement, Shift, and Rotate Instructions 82

20. Examples of Increment and Decrement Instructions 84

21. Control Instructions Which Change the Order in Which the Program is Executed 86

22. Nesting of Subroutines 87

23. Unconditional Jump Created by Transferring Registers . . 90

24. An Assembly Language Program Illustrating the Basic Features of Assemblers, and the Program Written Using the Intel 8080 Assembly Language 96

25. An Example of the Intel 8080 Assembler Converting a Source Tape, to a List Tape, to an Absolute Loader Tape 97

26. An Example of a Macro Using the Intel 8080 Instruction Set 100

27. Examples of the Error Detection Capabilities of the Intel 8080 Assembler 101

28. A Comparison of a Compiler Program (PL/M) and an Assembly Language Program, Both Written from the Intel 8080 Languages 105

29. Applications of Microcomputers 109

TABLE PAGE

30. Example of Addition and Subtraction Operations Performed by the Desk Calculator Program 122

31. Typical Operation and Results as Observed on the Teletype 123

32. Control of Each Side of the Traffic Light is Accomplished with Two Bits 125

33. Example of Successive Approximator as an ADC 141

LIST OF FIGURES

FIGURE		PAGE
1.	Basic Microcomputer Structure	2
2.	Block Diagram of Rockwell PPS-4 Microcomputer	8
3.	Rockwell PPS-4 Microprocessor	10
4.	Block Diagram of Fairchild PPS-25 Microcomputer	12
5.	Fairchild PPS-25 Microprocessor	13
6.	Block Diagram of Intel MCS-40 Microcomputer	16
7.	Timing Diagram of MCS-40 System	18
8.	Intel 4004 Microprocessor	21
9.	Intel 4040 Microprocessor	25
10.	Block Diagram of Fairchild F-8 Microcomputer	27
11.	Fairchild F-8 Microprocessor	28
12.	Block Diagram of Motorola 6800 Microcomputer	30
13.	Motorola 6800 Microprocessor	36
14.	Block Diagram of Intel 8080 Microcomputer	48
15.	Instruction Cycle of the Intel 8080 Microcomputer	56
16.	Intel 8080 Microprocessor	58
17.	Block Diagram of National IMP-16C Microcomputer	61
18.	RALU Block Diagram	63
19.	Block Diagram of CROM	65
20.	Timing Diagram of IMP-16C System	66
21.	Data Movement Within the Major Storage Areas is Shown by Dashed Lines	74

FIGURE PAGE

22. Preparation of a Program Using Assemblers, Compilers, and Loaders 93
23. Typical Loader Data for the Intel 8080 Microcomputer . . . 103
24. IMP-16C Microinstruction Word Formats 107
25. Block Diagram of Desk Calculator Application 111
26. Schematic of Desk Calculator 112
27. Minus (-) Sign Schematic 114
28. Error (E) Indicator Schematic 115
29. Flowchart for Desk Calculator Applications 117
30. Circuit Developed to Control Each Side of the Traffic Lights 124
31. Random Signal (Car) Generator Schematic 126
32. Flowchart of Traffic Controller Application 128
33. Block Diagram of ADC Application 136
34. Flowchart of ADC Application 138

CHAPTER I

INTRODUCTION

Advancements in digital circuits have progressively increased large scale integrated (LSI) technology. In the past few years a single LSI chip has progressed to contain over 14,000 transistors. This tremendous chip density has caused a revolution in the applications for digital hardware. The most important development with these LSI devices is the microcomputer. A microcomputer contains LSI devices which are used as the central processing unit (CPU) of the computer. Core or semiconductor memory is used to store instructions and data (Figure 1). Most microcomputer CPUs are organized about a microprocessor. The microprocessor is a single LSI device which interprets instruction strings and controls hardware registers, an arithmetic logic unit (ALU), and input/output control logic. However, some microcomputers are developed which use more than one LSI device as the CPU. For example, the National Semiconductor series of microcomputers use a parallel combination of four-bit devices plus read-only-memory to develop the CPU of eight- and sixteen-bit microcomputers.

The advantages of using a microcomputer are: quick design cycles, low costs, and flexibility of designs. Microcomputers are used in place of hard-wired logic designs. Consequently, design board layouts are simplified. Complex interconnections required for large numbers of conventional integrated circuits (ICs) are replaced by the

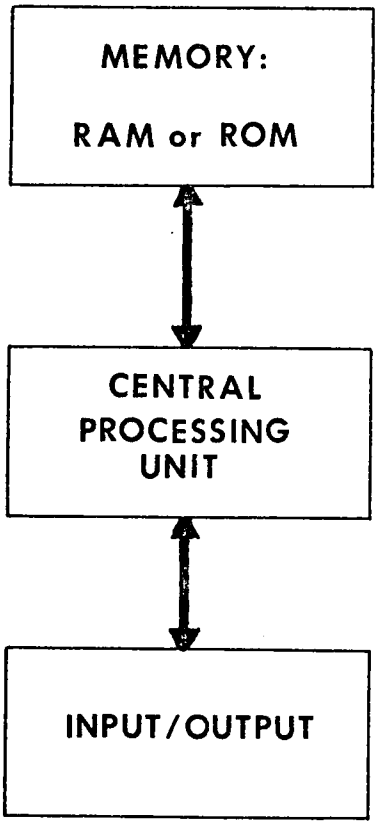


Figure 1. Basic Microcomputer Structure

read-only-memory (ROM) in the microcomputer. Therefore, the only interconnect wiring on the design boards runs between the address and data buses and the input/output devices of the microcomputer. The logic design becomes simply the manipulation of functional blocks, with the control sequence being stored and executed as a software program. All the breadboarding that is required consists of interconnecting the few LSI packages of the microcomputer. No hardwire logic must be simulated, optimized, or breadboarded.

LSI microprocessor technology has enabled combinational circuit designers to decrease component count and cost. Cost savings are not limited to direct circuit component cost but extend to other system hardware costs. Connectors are decreased in number, cabling is simplified, and the card cage is reduced in size. Indirect costs are also reduced since assembly of the design takes less time, documentation is simpler, and maintenance is easier.

Expansion and flexibility within a microcomputer system design is simply that of modifying the control program. In contrast, modifications of a hardwired logic design would require redesigning and laying out the new circuit diagram, utilizing additional components, and rewiring the layout. Flexibility within a microcomputer is described by the fact that random data collecting, complex logic decisions, and arithmetic computations can be performed by software techniques. Generating these functions by hardwired logic would require many IC components (1, 2).

At the present stage of microcomputer development, the only disadvantage of microcomputers is the speed at which a microprocessor operates. Single instruction executions vary from two microseconds to sixty microseconds, depending upon the microprocessor being used. For any high speed or minicomputer application, the designer must consider the execution times of the microcomputer to decide whether or not the processor can be used. Although speed is a limitation of microcomputers, the many advantages of microcomputers overshadow this limitation.

Three basic types of microprocessors are available: (1) Calculator-oriented machines, (2) fixed instruction-set machines, and (3) microprogrammable machines. Calculator-oriented processors are 4-bit machines which handle binary-coded-decimal (BCD) numbers easily. For arithmetic operations, these calculator processors operate on a serial string of four bits (digits) which correspond to decimal numbers. However, some general-purpose computer features are included in the 4-bit machines since manufacturers desired applications of a wider range than just calculators. Typical four-bit microprocessors which are described are the: Intel 4004, Fairchild PPS-25, and Rockwell PPS-4.

Current development is presently centered on the fixed instruction-set microprocessors. These processors have a word length of eight bits, i.e., their data buses and registers hold eight bits of information. Fixed instruction processors have both architectures and instruction sets similar to minicomputers. Typical fixed

instruction-set microprocessors which are described are the: Fairchild F-8, Motorola 6800, and the Intel 8080.

Few microprocessors emerging today are microprogrammable, but the significance of the machines which are microproprogrammable is evident.

A microprogramming approach is used to implement a microcomputer's CPU functions. This implementation is accomplished by replacing fixed conventional control logic with memory. Each address in the memory represents unique states in the conventional control logic, and each memory output is connected to control lines from the conventional logic. Stored in the memory are basic instructions called microinstructions. These instructions include the fundamental control, testing, branching, and moving operations upon which all functions of a computer are based. For the microcomputer to perform high level operations, sequences of these microinstructions corresponding to high-level functions are stored in ROM and are called when desired. These sequences are called macroinstructions. These macroinstructions develop the instruction set of the microcomputer. Using the capability of the microprogramming feature allows the programmer to develop any instruction desired in a particular application. The National IMP-16C microprocessor is the microprogrammable processor described.

To properly evaluate the microcomputer all aspects, architecture, software, and applicability have been researched. Chapter II evaluates the architectural structures of the most prominent

microcomputers. Timing, data buses, registers, and input/output capabilities are developed for the 4-, 8-, and 16-bit microprocessors.

Software capabilities of microcomputers are developed in Chapter III. Data movement, data manipulation, decision and control, and input/output instructions are explained, with examples. Also, ways in which a programmer can write a program and store the program into memory are developed: assemblers, editors, compilers, loaders, and microprogramming.

A large variety of industrial and communication designs can be performed by microprocessors: process control systems, terminal devices, automobile control systems, and traffic control systems are just a few. In actuality a microcomputer can be used for any application in which random logic control requires optimization. Specific applications of the Intel 8080 microcomputer are described in Chapter IV. These applications, desk calculator and traffic controller, have been designed and developed to demonstrate actual applications of a microcomputer. The desk calculator performs basic add and subtract functions and can be developed further to perform all arithmetic functions. The traffic controller has been developed to simulate any type of traffic light presently being used.

CHAPTER II

ARCHITECTURE

In order to determine the particular microcomputer needed for an application, a designer must understand the structure of the hardware within the microcomputer. This structure, the architecture, is discussed by separating the microprocessors into the three groups developed in Chapter I: calculator oriented, fixed instruction, and microprogrammable. The main areas of research were concentrated on the architecture of the Intel MCS-40, the Intel 8080, the Motorola 6800, and the National IMP-16C microcomputers. However, other machines are described to give an overall comparison of microcomputer architectures.

I. CALCULATOR ORIENTED MICROPROCESSORS

Rockwell PPS-4

To present the calculator-oriented processors, three systems are described: Rockwell PPS-4, Fairchild PPS-25, and Intel MCS-4. The Rockwell PPS-4 microcomputer system (Figure 2) consists of a four-bit central processing unit (CPU), a 200-kilohertz clock, read-only-memory (ROM) and random-access-memory (RAM) storage devices, and input/output (I/O) facilities. Up to 16,384 bytes (eight bits per byte) of ROM storage capacity and 8,192 bytes of RAM data capacity are available in the PPS-4. I/O facilities consist of:

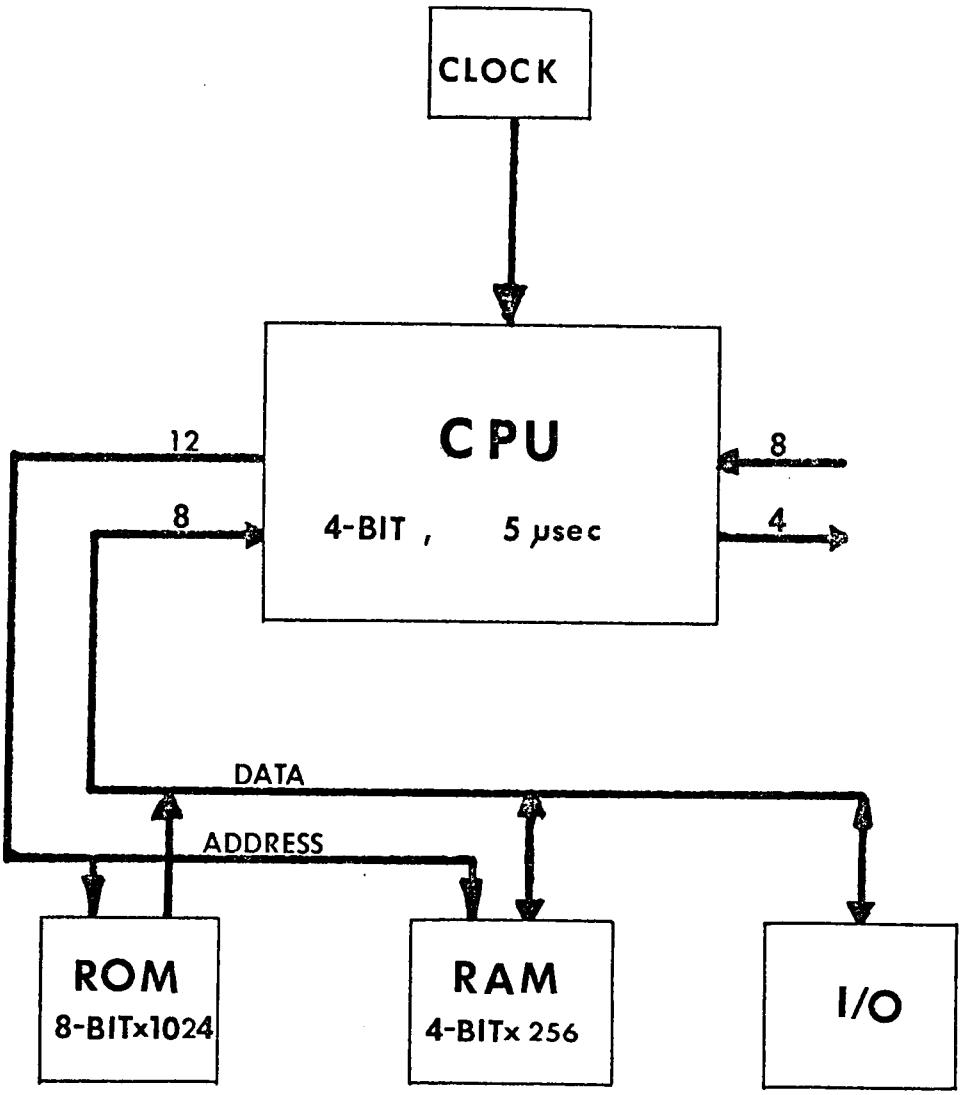


Figure 2. Block Diagram of Rockwell PPS-4 Microcomputer

1. General I/O with three four-bit I/O groups.
2. Keyboard/Display circuit which handles 64 keys and a sixteen digit seven-segment display.
3. Printer controller.
4. 1200 bit per second Universal Asynchronous Receiver Transmitter.
5. Bus interface circuits for general needs.

Although the keyboard/display circuit suggests that Rockwell is developing a desk calculator, the remaining I/O devices suggest further applications.

As most other microcomputers, the PPS-4 operates with a fixed, nonvarying basic machine cycle. Rather than accomplishing the fetch-execute by time subdivision inside one machine cycle, the PPS-4 implements the fetch and execute in alternating cycles. Thus, the PPS-4 overlaps the fetch cycle with the execute cycle of the previous instruction. This unique operation increases the speed of the Rockwell machine instruction execution time.

The CPU of the PPS-4, the microprocessor, is organized as a single accumulator computer (Figure 3). Four-bit digits are operated upon by the arithmetic logic unit (ALU) with the results being sent to the accumulator. Storage references are made with a twelve-bit RAM address register which can be loaded with the four-bit digits of the accumulator. The index register (X register) provides an additional arithmetic register which can be used to modify the RAM address: using indexing techniques. Few program accessible registers exist,

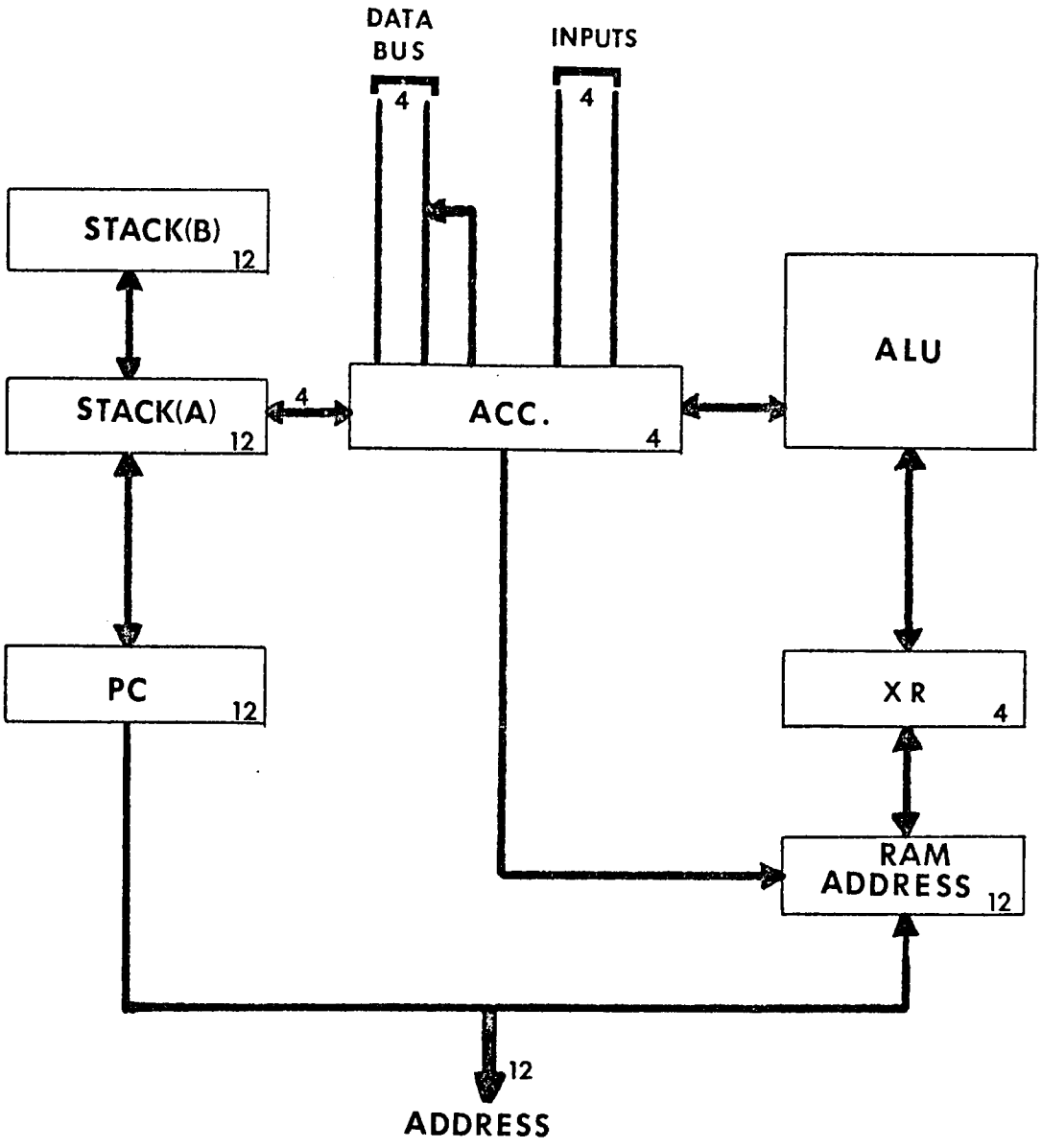


Figure 3. Rockwell PPS-4 Microprocessor

therefore most data reference instructions require storage. Two stack positions are available for subroutine return address saving. One of the stack positions can be passed into the accumulator then to memory, increasing the stack depth (4, 5).

Fairchild PPS-25

One of the most unique architectures of any microprocessor is contained within the Fairchild PPS-25. This microcomputer includes a two-chip CPU, ROM, RAM, and I/O chips (Figure 4). Internally the PPS-25 differs in that the RAM module is organized as just three 25-digit registers. This organization is odd, but is appropriate for numeric processing applications, such as navigation systems or calculators. Because only a small percentage of applications require 25 digits of precision, the PPS-25 allows programmer selection of different sections of the 25-digit register.

Data within the PPS-25 travels between RAM and the CPU on a four-bit bus. During a 62.5 microsecond machine cycle there are 25 minicycles of 2.5 microseconds each. During the minicycle a new digit is presented to the CPU from the selected RAM register: with 25 serial sequential operations the old digit in the CPU is replaced by this new digit.

A 6 × 25 bit ROM which represents six masks, each 25-bits long, is in the PPS-25 CPU (Figure 5). Each bit in a mask specifies whether the corresponding digit from the RAM register is to be operated upon by the current instruction. "Ones" define the data to be used. As an

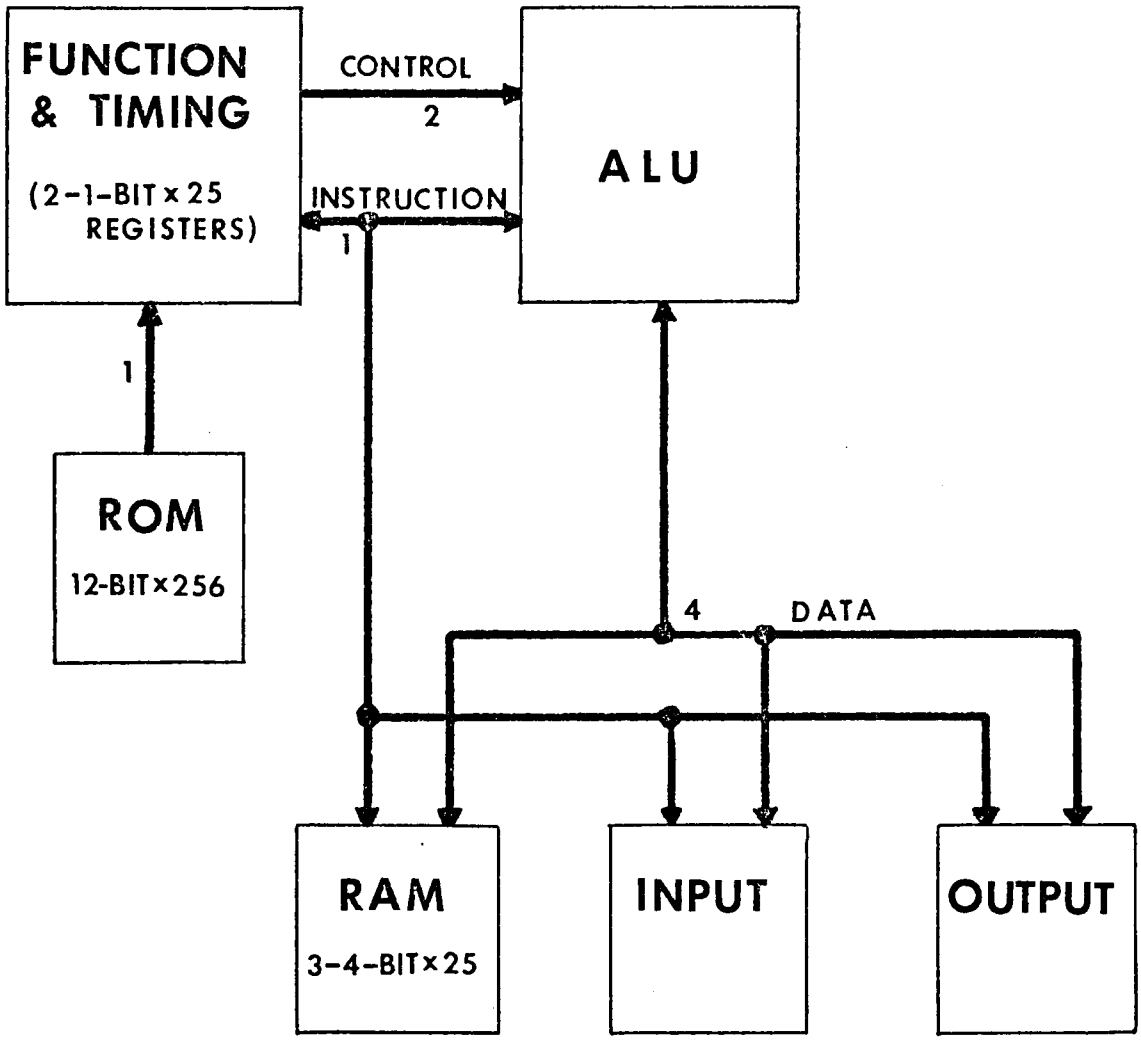


Figure 4. Block Diagram of Fairchild PPS-25 Microcomputer

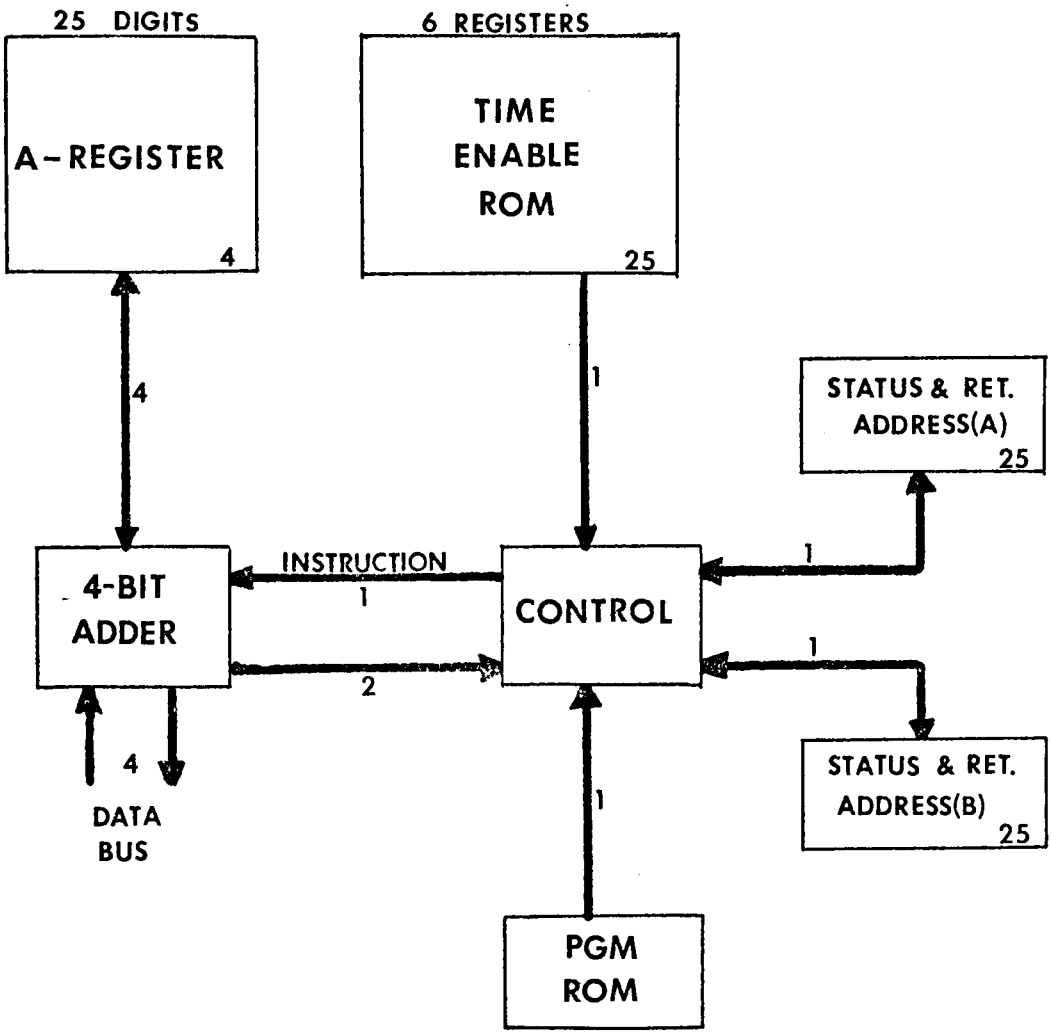


Figure 5. Fairchild PPS-25 Microprocessor

example, an application is shown which uses nine-digit floating point arithmetic (Table 1). Two of the masks select the significant digits, and two masks select the exponent digits. Manipulation of data not covered by the ROM masks is provided by a single register in the PPS-25. This register can be programmed to hold a number in the range of 0 to 24 thus, allowing any single digit to be selected and operated upon. Every PPS-25 instruction has a three-bit code which specifies one of the six masks to define the data format, and the seventh code (the preprogrammed register) selects one of the 25 digits.

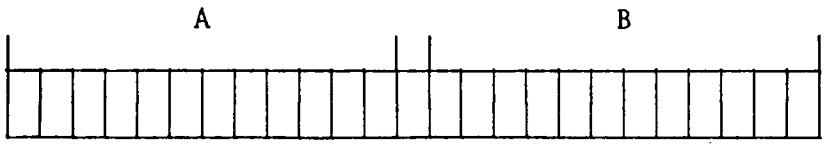
An instruction fetch uses the same 25 minicycles. As execution of one instruction is being performed, the instruction is being fetched, i.e., as the 25 minicycles are moving data between the RAM and CPU, the minicycles are also being used to send an address to the ROM for the next instruction. This instruction fetch sequence is performed on a one-bit wide path. During the fourth through twelfth time slots, the ROM instruction address is sent out from the CPU. During the fourteenth through twenty-fifth time slots, the instruction is serially transmitted from the ROM to the CPU (4, 5).

Intel MCS-40

The Intel MCS-40 microcomputer system consists of either an Intel 4004 or an Intel 4040 as the CPU, ROM and RAM as memory devices, and I/O ports to interface to external devices (Figure 6). All inter-element communication within the system transpires over a four-bit data bus. The data bus transfers information from the CPU to

TABLE 1

EXAMPLE OF PPS-25 NINE-DIGIT FLOATING POINT ARITHMETIC



"A" Digit Mask	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
"B" Digit Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0		
"A" Exponent Mask	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0		
"B" Exponent Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

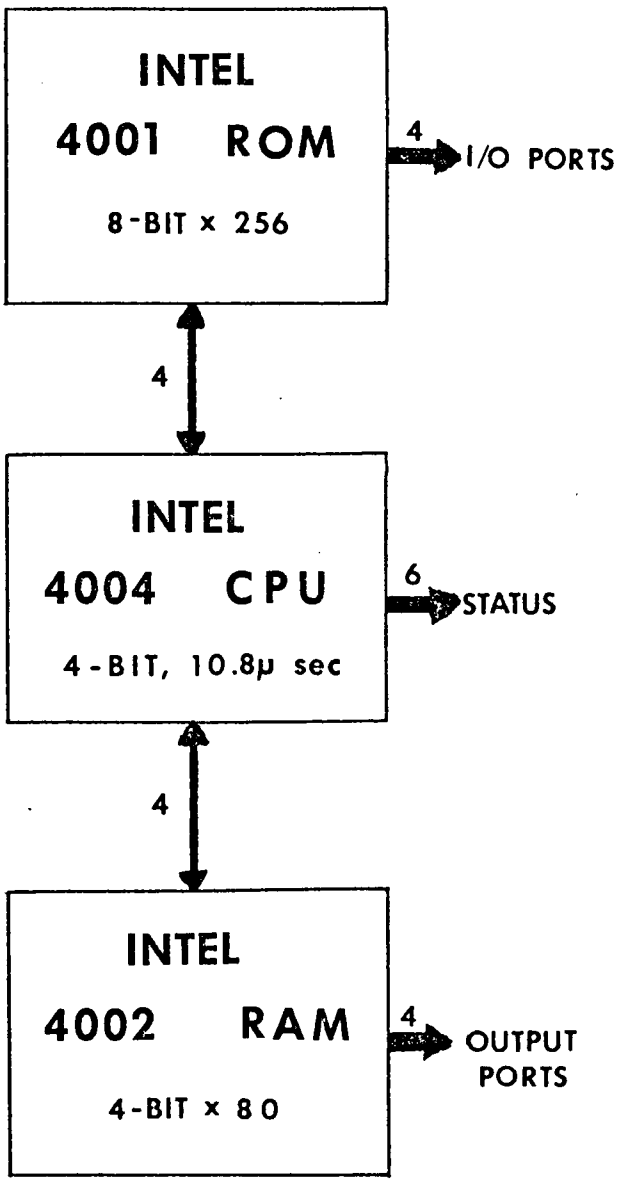


Figure 6. Block Diagram of Intel MCS-40 Microcomputer

memory and I/O devices. Information on the data bus is contained within one instruction cycle. This instruction cycle is divided into equal time segments, each segment being one clock period. A twelve-bit instruction address is presented to memory during the first three time segments. An eight-bit instruction is provided during the fourth and fifth segments, and the instruction is decoded during the sixth segment. Program execution is performed during the remaining two segments (Figure 7). Instructions which require two bytes are executed in sixteen cycles which are partitioned into eight cycle segments. During the first cycle the instruction is fetched, and during the second cycle the operand, the second byte of the instruction, is fetched and the instruction executed.

As defined, the CPU of the MCS-40 microcomputer system can be either the Intel 4004 or the Intel 4040 microprocessor. The more powerful of the two machines being the Intel 4040. A description of the 4004 is given followed by an explanation of the increased capabilities of the 4040. Communication among the 4004 and other memobers of the microcomputer is via a four-bit data bus. The 4004 contains five command control lines; four are used to control the RAM chips, and one is used to control the ROM chip. Instructions and their format, discussed in Chapter III, of the 4004 consist of the following:

1. Sixteen machine instructions.
2. Fourteen accumulator instructions.
3. Sixteen input/output instructions.

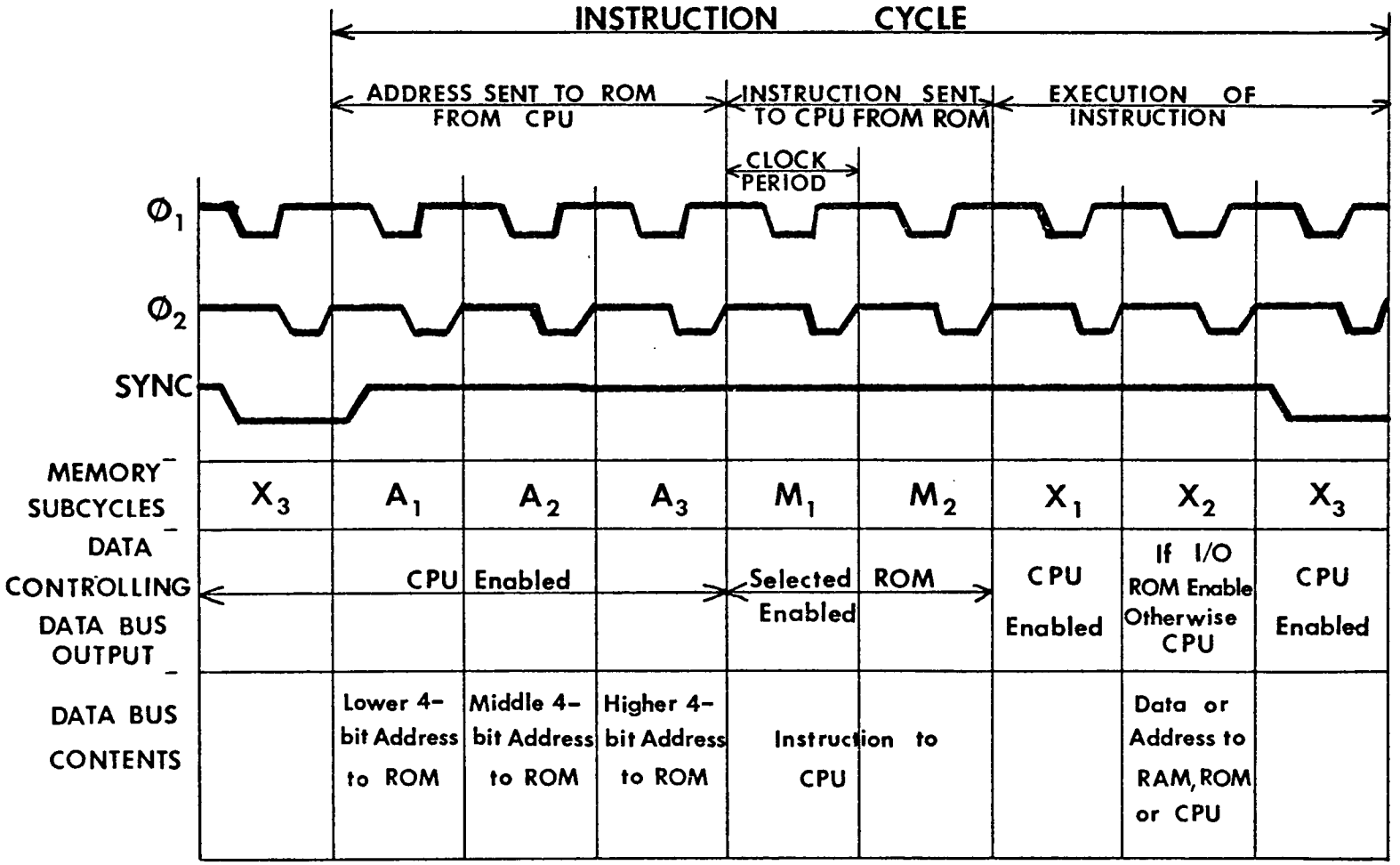


Figure 7. Timing Diagram of MCS-40 System

A sixteen pin LSI package contains the 4004 CPU. Table 2 gives a functional description of each pin.

Timing within the MCS-40 system is supplied by two nonoverlapping clock phases ϕ_1, ϕ_2. A synchronization signal, SYNC, is generated every eight clock periods and is used as the beginning of an instruction cycle. External timing is developed by the MCS-40 components using the SYNC, ϕ_1, ϕ_2 signals (Figure 7).

An MCS-40 machine cycle begins with the 4004 sending a SYNC pulse to the ROM and RAM chips. At this time twelve-bits of the ROM address, the instruction address, are sent to the data bus. This address is then incremented by one and stored in the program counter. The selected ROM then sends back eight-bits of instruction or data, which is stored as the instruction. Execution of the instruction is then performed.

The functional blocks of the 4004 shown in Figure 8 are as follows:

1. Address register and address incrementer.
2. Index register (sixteen words of four-bits each).
3. Four-bit adder.
4. Instruction register, decoder, and control.
5. Peripheral circuitry.

The address register contains one level to store the next instruction address (program counter) and three levels to stack subroutine calls (see Chapter III). An address is stored in an address buffer, after being read, and is demultiplexed, into three four-bit words, to the

TABLE 2

FUNCTIONAL DESCRIPTION OF EACH PIN
OF THE INTEL 4004

Pin Number	Designation	Description of Function
1-4	D_0-D_3	Bidirectional data bus. All address and data communication between the processor and the RAM and ROM chips is handled by way of these 4 lines.
5	V_{SS}	Most positive supply voltage.
6-7	ϕ_1-ϕ_2	Nonoverlapping clock signals which determine processor timing.
8	SYNC	SYNC output. Synchronization signal generated by the processor and sent to ROM and RAM chips. Indicates beginning of instruction cycle.
9	RESET	RESET input. A "1" level applied to this pin clears all flag and status flip-flops and forces the program counter to 0. To completely clear all of the address and index registers, RESET must be applied for 64 clock cycles (8 machine cycles).
10	TEST	TEST input. The logical state of this input can be examined with the JCN instruction.
11	CM-ROM	This pin enables a ROM bank, which can contain up to 4K words of program using 4001, 4308, or 4289.
12	V_{DD}	Main supply voltage to the processor. Value must be V_{SS} -15.0V ±5%.
13-16	CM-RAM_0- CM-RAM_3	CM-RAM outputs. These outputs act as bank select signals for the 4402 RAM chips in the system.

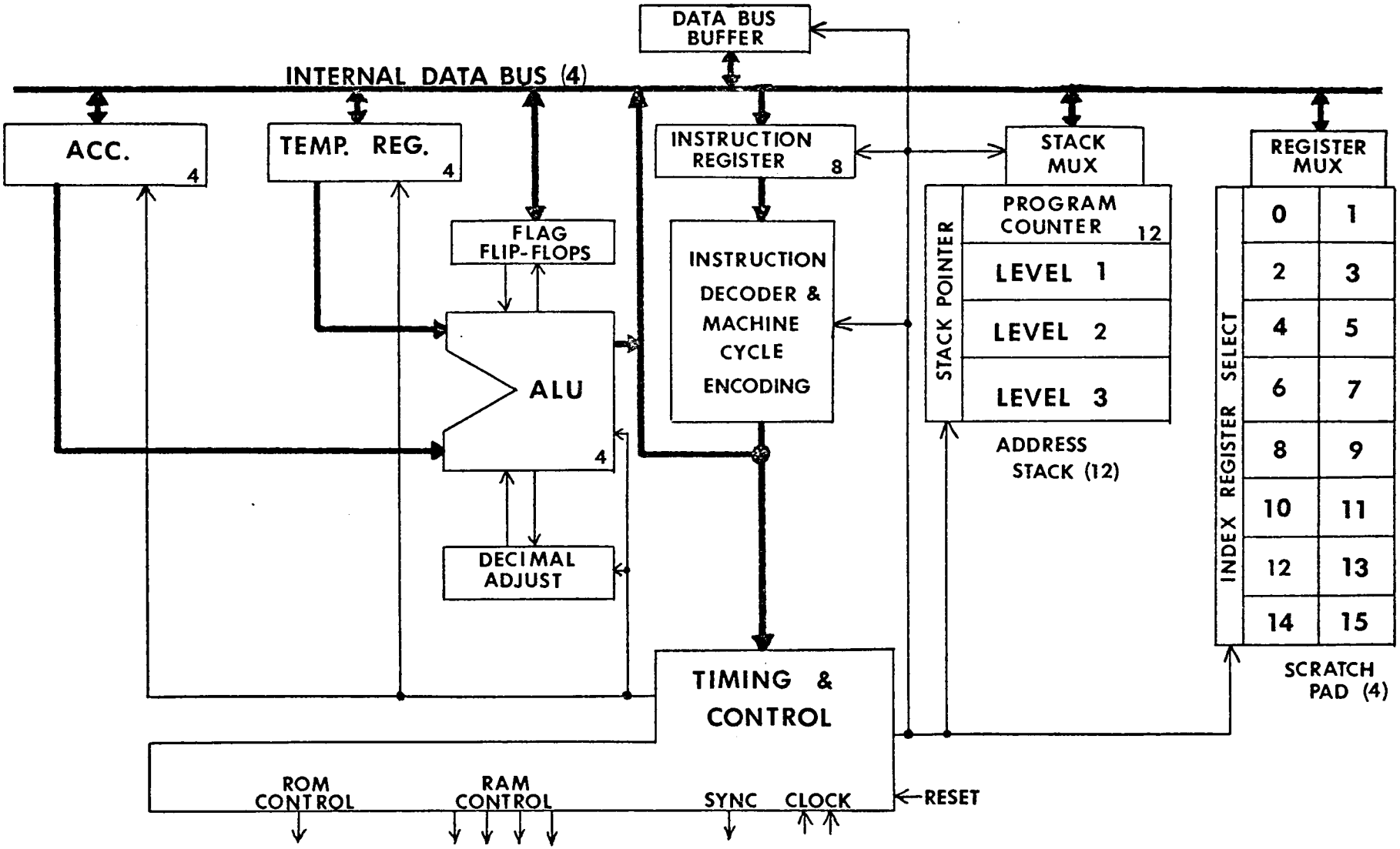


Figure 8. Intel 4004 Microprocessor

address register bus during cycles A_1, A_2, and A_3. This current address is incremented by a four-bit carry look ahead circuit (address incrementer) after each four bits are placed on the data bus. The incremented address is then transferred to the address buffer and stored in the program counter.

Two modes of operation are available with the index register. In one mode, the index register provides sixteen addressable storage locations for intermediate calculation and control. In the other mode, the index register provides eight pairs of addressable locations for addressing RAM and ROM memory locations. Also, in this second mode a storage location is provided for data fetched from a ROM. The address of the index register is provided by the internal bus and the refresh counter, then the address is multiplexed into the index register decoder. Contents of the index register are transferred to the internal bus through the multiplexer. Writing into the register is accomplished by transferring the contents of the internal bus into a temporary register, then to the index register.

The four-bit adder is a ripple-through carry type. One term of the addition comes from a register which communicates with the internal bus, and a register which can transfer data or complemented data to the adder. The other term comes from the accumulator and the carry flip-flop. Answers from the adder are then placed back into the accumulator and the carry flip-flop. Rotate, both right and left, are accomplished by the accumulator which contains a shifter. Communication between the accumulator and the command and control register,

special ROM's, the condition flip-flops, and the internal bus is accomplished.

The instruction register is loaded with the contents of the internal bus at M_1 and M_2 of the instruction cycle. After loading, the instruction register contains the instruction fetched from a ROM. Instructions are then decoded by the instruction decoder and appropriately gated with timing signals. This gating provides control signals used within the MCS-40 system. Double length instructions set a double cycle flip-flop so that two machine cycles are used for execution. These double length instructions are sixteen-bits wide and are stored in two successive locations of ROM.

Peripheral circuitry includes the data bus input/output buffers communicating between data paths and the internal bus. Also included are a timing and SYNC generator, one ROM command control, and four RAM command control buffers, and a reset flip-flop. During reset all RAM's and static flip-flops are cleared, and the data bus is set to zero. After reset, program control returns to location zero. Therefore, if the reset signal is applied long enough for the index register refresh counter to scan all locations of memory, all registers and RAM locations will be cleared (6).

Several improvements in both the architecture and the instruction set are developed into the Intel 4040 microprocessor. These improvements were enabled by the experience Intel gained in developing and using the 4004. Hardware improvements are as follows:

1. Single level interrupts.
2. The address stack increased from 4 × 12 bits to 8 × 12 bits, allowing up to seven levels of subroutines.
3. The index register array increased to 24 four-bit registers.
4. Implementation of the capability to single step the operation of programs.
5. The capability of addressing up to 8K × 8 words of ROM.

Software improvements include fourteen additional instructions which are the following:

1. Halt.
2. Logical operations.
3. Interrupt enable and disable functions.
4. ROM and index register bank switching.

Figure 9 is the block diagram of the 4040 microprocessor. Comparing Figures 8 (page 21) and 9, both the equivalent and improved functions of the microprocessors are visible (6).

II. FIXED INSTRUCTION MICROPROCESSORS

Current development in microprocessors is being put into eight-bit processors in which registers and buses typically hold eight bits of information. These microprocessors incorporate features which allow more freedom to a designer, such as, large stacks for subroutines and interrupt capabilities. Typical of these eight-bit microprocessors are the Fairchild F-8, the Motorola 6800, and the Intel 8080.

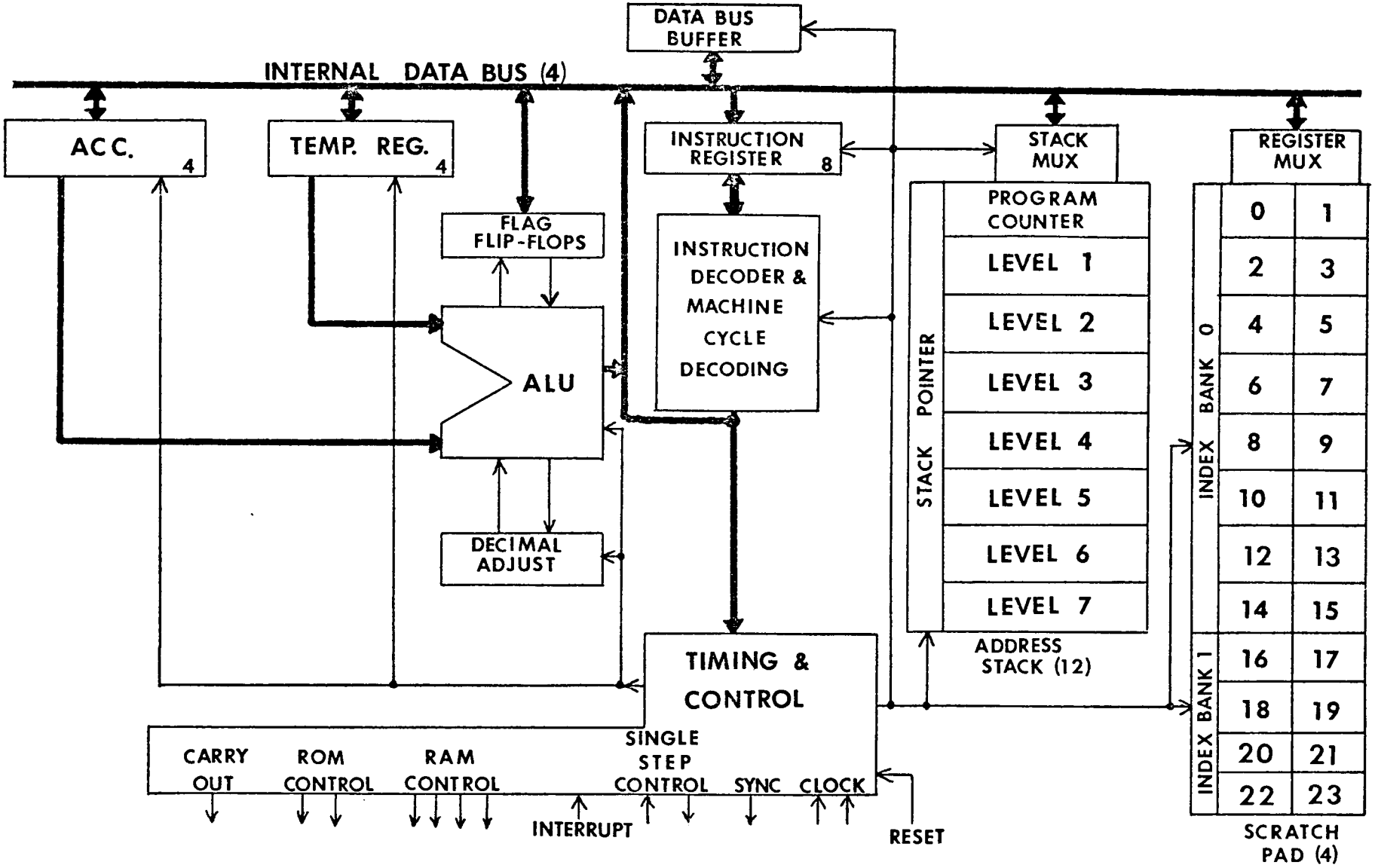


Figure 9. Intel 4040 Microprocessor

Fairchild F-8

The Fairchild F-8 microcomputer set consists of a CPU, a RAM, a mask-programmed ROM, and a general memory interface chip (Figure 10). A RAM and a ROM integrated circuit provide input and output paths. The ROM provides two bidirectional eight-bit channels whereas the RAM provides one. Addresses are programmed into ROM I/O ports when the ROM is programmed. All memory chips of the F-8 microcomputer accept one interrupt signal, and each chip connects to two neighbors thus, chaining the interrupt priorities. All memory devices also have two program location counters, which allow for fast interrupt and single-level subroutine servicing. The memory devices also contain a data address register. Reference to data in RAM and ROM devices is made through the sixteen-bit data counter. Both the program counters and the address register can be read into registers in the CPU, modified, and returned to the memory devices.

The F-8 CPU has two bidirectional input/output ports, a 64-word register file, and a master clock, but uniquely, the CPU does not contain a program counter or storage address. Also, an accumulator, a W register to hold the ALU and interrupt status, and an indirect scratch-pad address register (ISAR) are within the single CPU chip (Figure 11). The ISAR register is used to point to one of the 64 registers in a file. That register pointed to and the lowest twelve registers in the file are accessible from instructions. Registers nine through fifteen have access to the W register status bits and the

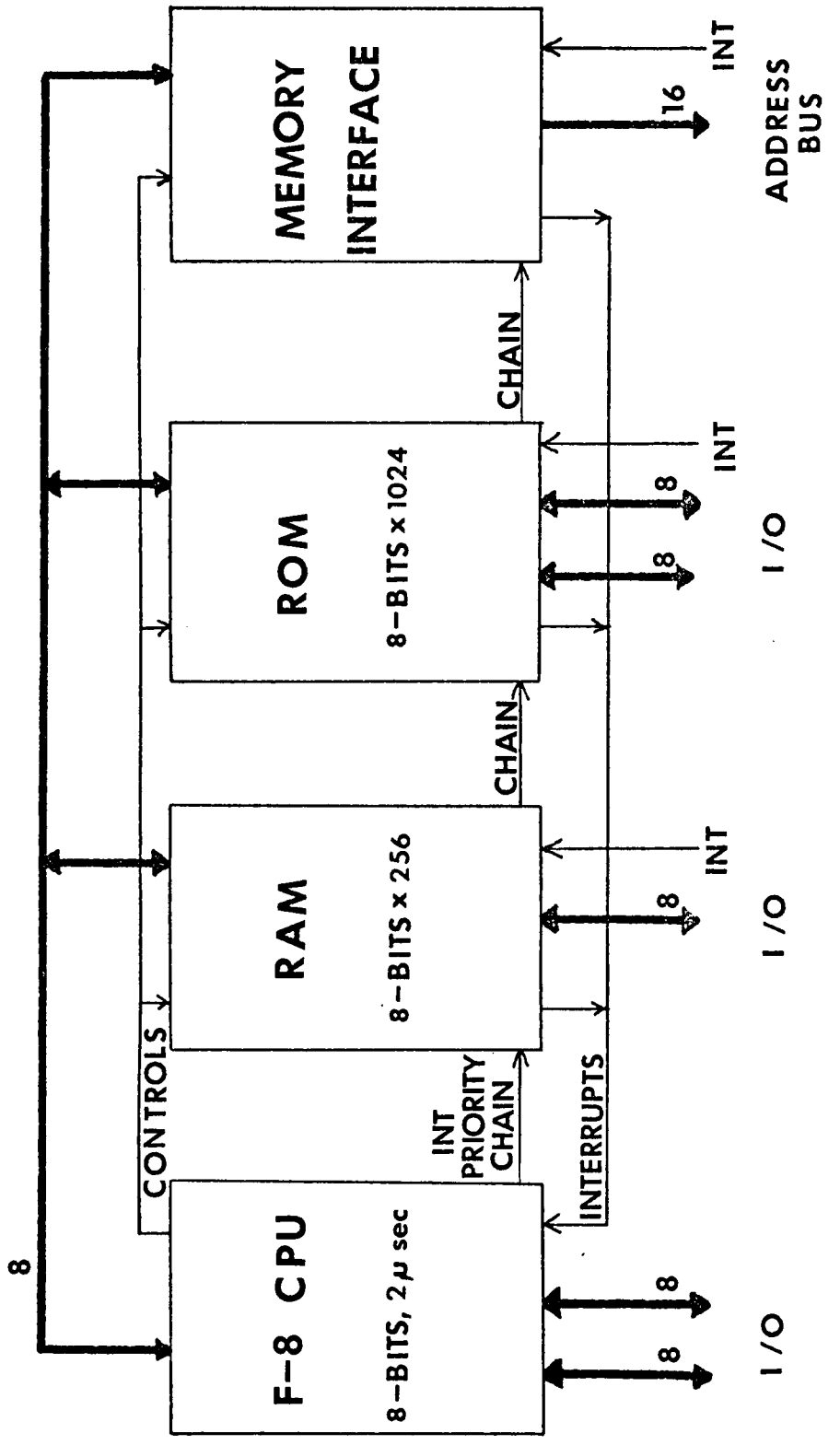


Figure 10. Block Diagram of Fairchild F-8 Microcomputer

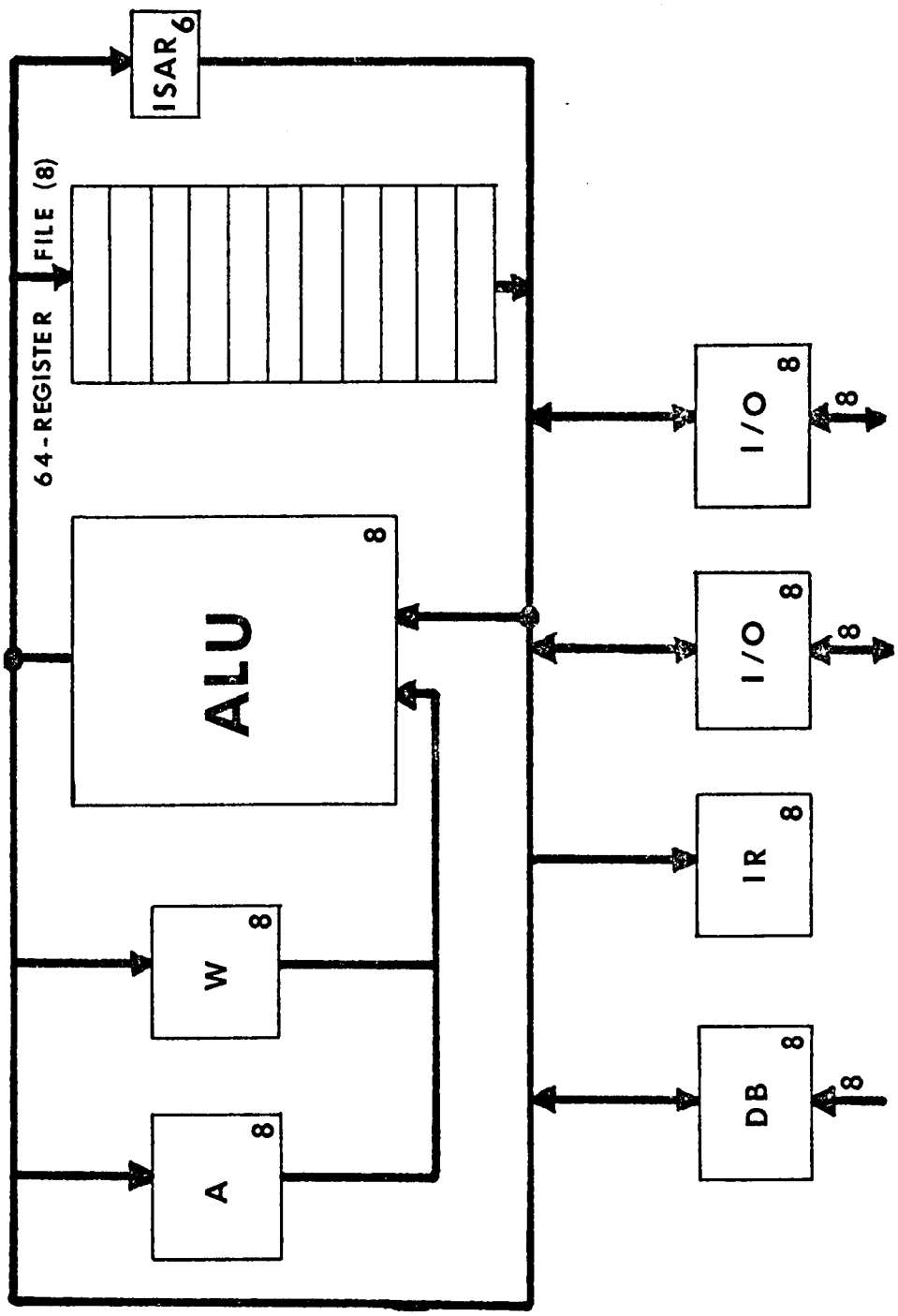


Figure 11. Fairchild F-8 Microprocessor

memory program counters. Because of this condensed CPU package some small applications can be performed with no external RAM (5).

Motorola 6800

The Motorola 6800 microcomputer set consists of a set of specially designed integrated circuit chips. Specifically, the microcomputer includes an MC6800 microprocessor, an MCM6830 ROM, an MCM6810 RAM, an MC6820 PIA, and an MC6850 ACIA (Figure 12). The Peripheral Interface Adapter (PIA) offers bidirectional control of two eight-bit buses and control and interrupt lines to and from external devices. The Asynchronous Communications Interface Adapter (ACIA) performs parallel/serial and serial/parallel conversion to allow the 6800 microcomputer to interface to a teletype.

A microbus structure is developed within the 6800 system in which both memory and I/O interfaces exist in the same address space. Selection of the memory or I/O interface is accomplished by the state of the address bus. Interfaces on the microbus with ROM and RAM memories are connected without buffers. This interface technique is allowed because all components for the system operate at the same voltage level and with the same drive capability. Chip selects are used within the system to distinguish the integrated circuits, thus eliminating the need for address decoding logic. No input/output lines exist on the system bus because all devices connected to the processor have similar interfaces.

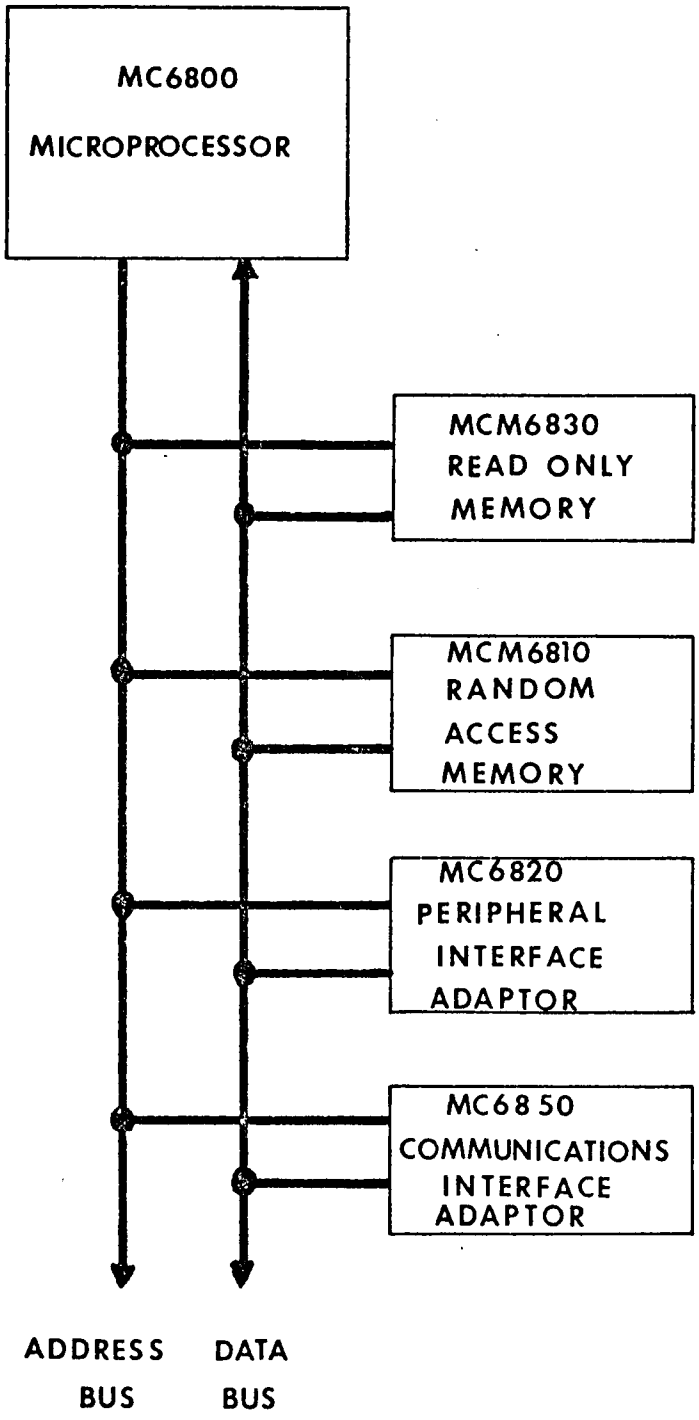


Figure 12. Block Diagram of Motorola 6800 Microcomputer

A forty pin microprocessor package contains the MPU 6800, the CPU of the 6800 system. Table 3 gives a functional description of each pin.

The MPU, or CPU, is organized with an eight-bit data/instruction path. The CPU contains two accumulators, a program counter, a stack pointer, and an index register (Figure 13). Two eight-bit accumulators are used to hold operands and results from the ALU. A two byte (sixteen bits) program counter points to the current program address. The stack pointer, normally a RAM location, is a two byte register which contains the address of the next location to be used during a stack operation. Another two byte register, the index register, is used to store data or a memory address during the indexed memory addressing mode (Chapter III).

The most unique device in the 6800 microcomputer system is the PIA. This PIA is used to provide sixteen bits of external interface and four control lines of addressable locations in memory. Each I/O bit is individually programmable to act as either input or output data. Operating characteristics of an interface are established by writing from the processor to the data direction and control registers of the PIA. Register select lines of the PIA serve the same purpose as address lines do in memory. Also, the PIA can be used to detect incoming interrupt signals on any of the control lines.

A forty pin integrated circuit package contains the PIA. Table 4 gives a functional description of each pin.

TABLE 3

FUNCTIONAL DESCRIPTION OF EACH PIN
OF THE MOTOROLA 6800

Pin	Designation	Description of Function
1 & 21	V_{SS}	Ground Reference
2	HALT	Halt. When this input is in the low state, all activity in the machine will be halted. In the halt mode, the machine will stop at the end of an instruction, Bus Available will be at a one level, Valid Memory Address will be at a zero, and all other three-state lines will be in the three-state mode.
3 & 37	Clocks Phase One & Phase Two (ϕ1, ϕ2)	Two pins are used for a two-phase nonoverlapping clock that runs at the V_{CC} voltage level.
4	$\overline{IRQ}$	$\overline{\text{Interrupt Request}}$. This level sensitive input requests that an interrupt sequence be generated within the machine. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. At that time, if the interrupt mask bit in the Condition Code Register is not set, the machine will begin an interrupt sequence. The Index Register, Program Counter, Accumulators, and Condition Code Register are stored away on the stack. Next the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations FFF8 and FFF9. An address loaded at these locations causes the MPU to branch to an interrupt routine in memory.
5	VMA	Valid Memory Address. This output indicates to peripheral devices that there is a valid address on the address bus. In normal operation, this signal should be utilized for enabling peripheral interfaces such as the PIA and ACIA. This signal is not three-

TABLE 3 (continued)

Pin	Designation	Description of Function
		state. One standard TTL load and 30 pF may be directly driven by this active high signal.
6	$\overline{NMI}$	Nonmaskable Interrupt. A low-going edge on this input requests that a nonmask-interrupt sequence be generated within the processor. As with the Interrupt Request signal, the processor will complete the current instruction that is being executed before it recognizes the $\overline{NMI}$ signal. The interrupt mask bit in the Condition Code Register has no effect on $\overline{NMI}$. The Index Register, Program Counter, Accumulators and Condition Code Register are stored away on the stack. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations FFFC and FFFD. An address loaded at these locations causes the MPU to branch to a nonmaskable interrupt routine in memory.
7	BA	Bus Available. The Bus Available signal will normally be in the low state; when activated, it will go to the high state indicating that the microprocessor has stopped and that the address bus is available. This will occur if the $\overline{Halt}$ line is in the low state or the processor is in the WAIT state as a result of the execution of a WAIT instruction. At such time, all three-state output drivers will go to their off state and other outputs to their normally inactive level. The processor is removed from the WAIT state by the occurrence of a maskable (mask bit 1 = 0) or nonmaskable interrupt. This output is capable of driving one standard TTL load and 30 pF.
8	V_{CC}	Supply Voltage

TABLE 3 (continued)

Pin	Designation	Description of Function
9-20 22-25	A0-A15	Address Bus. Sixteen pins are used for the address bus. The outputs are three-state bus drivers capable of driving one standard TTL load and 130 pF. When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications.
25-33	D0-D7	Data Bus. Eight pins are used for the data bus. It is bidirectional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers capable of driving one standard TTL load and 130 pF.
34	R/W	Read/Write. This TTL compatible output signals the peripherals and memory devices whether the MPU is in a Read (high) or Write (low) state. The normal standby state of this signal is Read (high). Three-State Control going high will turn Read/Write to the off (high impedance) state. This output is capable of driving one standard TTL load and 130 pF.
36	DBE	Data Bus Enable. This input is the three-state control signal for the MPU data bus and will enable the bus drivers when in the high state. This input is TTL compatible; however in normal operation, it would be driven by the phase two clock. During an MPU read cycle, the data bus drivers will be disabled internally. When it is desired that another device control the data bus such as in Direct Memory Access (DMA) applications, DBE should be held low.
39	TSC	Three-State Control. This input causes all of the address lines and the Read/Write line to go into the off or high impedance state. This state will occur 500 ns after TSC = 2.4V. The Valid Memory Address and Bus Available signals will be forced low. The

TABLE 3 (continued)

Pin	Designation	Description of Function
		data bus is not affected by TSC and has its own enable (Data Bus Enable). In DMA applications, the Three-State Control line should be brought high on the leading edge of the Phase One Clock. The $\phi 1$ clock must be held in the high state and the $\phi 2$ in the low state for this function to operate properly. The address bus will then be available for other devices to directly address memory. Since the MPU is a dynamic device, it can be held in this state for only 5.0 us or destruction of data will occur in the MPU.
40	$\overline{\text{Reset}}$	$\overline{\text{Reset}}$. This input is used to reset and start the MPU from a power down condition, resulting from a power failure or an initial start-up of the processor. If a positive edge is detected on the input, this will signal the MPU to begin the restart sequence. This will start execution of a routine to initialize the processor from its reset condition. All the higher order address lines will be forced high. For the restart, the last two (FFFE, FFFF) locations in memory will be used to load the program that is addressed by the program counter. During the restart routine, the interrupt mask bit is set and must be reset before the MPU can be interrupted by $\overline{\text{IRQ}}$.

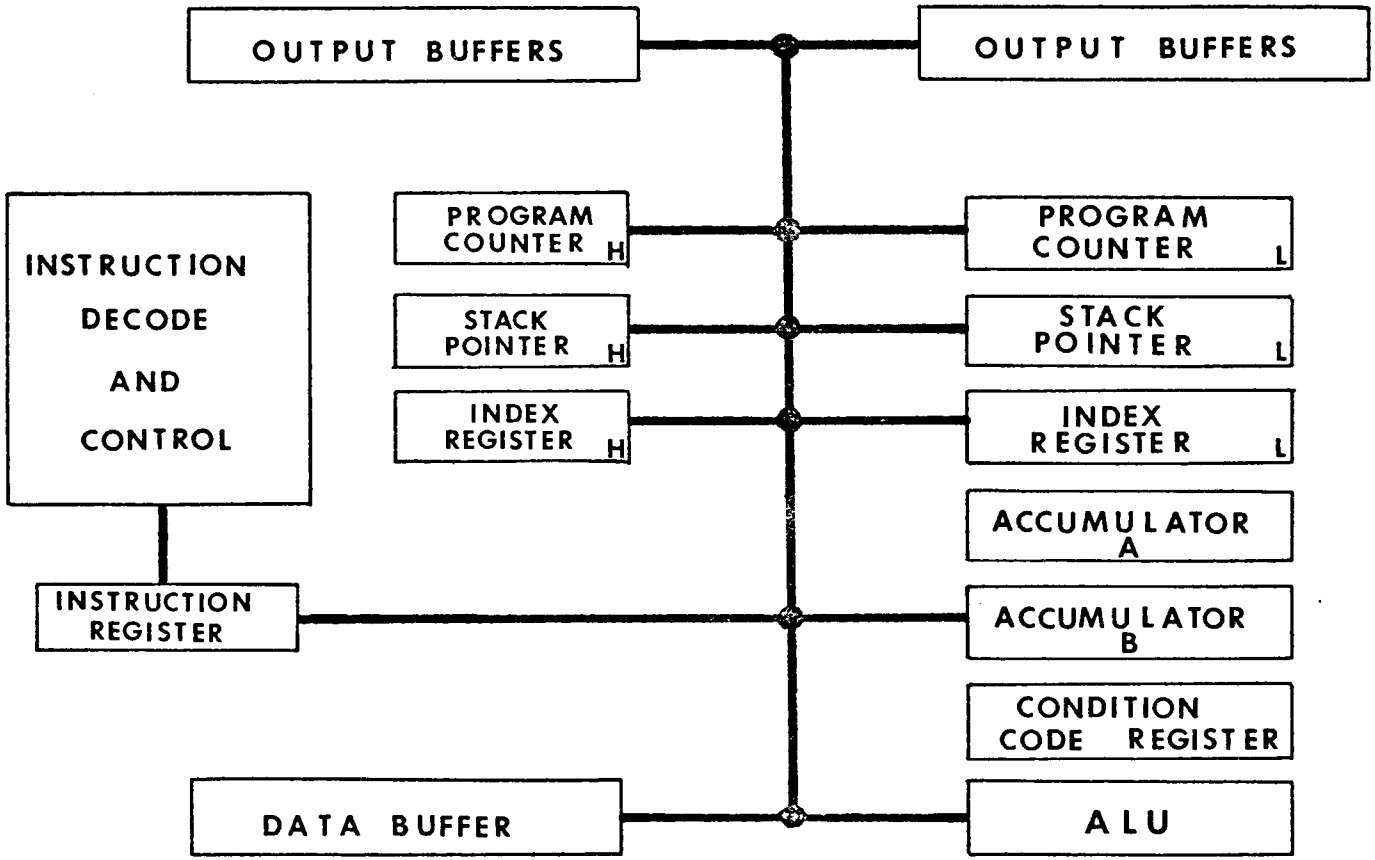


Figure 13. Motorola 6800 Microprocessor

TABLE 4

FUNCTIONAL DESCRIPTION OF EACH PIN OF THE PIA

Pin	Designation	Description of Function
1	V_{SS}	Ground Reference
2-9	PA0-PA7	Section A Peripheral Data. Each of the peripheral data lines can be programmed to act as an input or output. This is accomplished by setting a "1" in the corresponding Data Direction Register bit for those lines which are to be outputs. A "0" in a bit of the Data Direction Register causes the corresponding peripheral data line to act as an input. During an MPU Read Peripheral Data Operation, the data on peripheral lines programmed to act as inputs appears directly on the corresponding MPU Data Bus lines. In the input mode the internal pullup resistor on these lines represents a maximum of one standard TTL load.
10-17	PB0-PB7	Section B Peripheral Data. The peripheral data lines in the B Section of the PIA can be programmed to act as either inputs or outputs in a similar manner to PA0-PA7. However, the output buffers driving these lines differ from those driving lines PA0-PA7. They have three-state capability, allowing them to enter a high impedance state when the peripheral data line is used as an input. In addition, data on the peripheral data lines PB0-PB7 will be read properly from those lines programmed as outputs even if the voltages are below 2.0 volts for a "high." As outputs, these lines are compatible with standard TTL and may also be used as a source of up to 1 milliampere at 1.5 volts to directly drive the base of a transistor switch.
8 & 40	CB1,CA1	Interrupt Input. Peripheral Input lines CA1 and CB1 are input only lines that set the interrupt flags of the control registers.

TABLE 4 (continued)

Pin	Designation	Description of Function
		The active transition for these signals is also programmed by the two control registers.
19	CB2	Peripheral Control. Peripheral Control line CB2 may also be programmed to act as an interrupt input or peripheral control output. As an input, this line has high input impedance and is compatible with standard TTL. As an output it is compatible with standard TTL and may also be used as a source of up to 1 milliampere at 1.5 volts to directly drive the base of a transistor switch. This line is programmed by Control Register B.
20	V_{CC}	Supply Voltage
21	R/W	PIA Read/Write. This signal is generated by the MPU to control the direction of data transfers on the Data Bus. A low state on the PIA Read/Write line enables the input buffers and data is transferred from the MPU to the PIA on the E signal if the device has been selected. A high on the Read/Write line sets up the PIA for a transfer of data to the bus. The PIA output buffers are enabled when the proper address and the enable pulse E are present.
22-24	CS0, $\overline{CS2}$, CS1	PIA Chip Select. These three input signals are used to select the PIA. CS0 and CS1 must be high and $\overline{CS2}$ must be low for selection of the device. Data transfers are then performed under the control of the Enable and Read/Write signals. The chip select lines must be stable for the duration of the E pulse.
25	E	PIA Enable. The enable pulse, E, is the only timing signal that is supplied to the PIA. Timing of the other signals is referenced to the leading and trailing edges of the E pulse. This signal will normally be a

TABLE 4 (continued)

Pin	Designation	Description of Function
		derivative of the MC6800 $\phi 2$ Clock. The E pulse is used to condition the interrupt control lines CA1, CA2, CB1, and CB2. At least one E pulse can occur from the inactive edge to the active edge of the input signal to set the interrupt flag, when the lines are used as inputs.
26-33	D0-D7	PIA Bidirectional Data. The bidirectional data lines (D0-D7) allow the transfer of data between the MPU and the PIA. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs a PIA read operation. The Read/Write line is in the Read (high) state when the PIA is selected for a Read operation.
34	$\overline{\text{Reset}}$	$\overline{\text{Reset}}$. The active low $\overline{\text{Reset}}$ line is used to reset all register bits in the PIA to a logical zero (low). This line can be used as a power-on reset and as a master reset during system operation.
35-36	RS1, RS0	PIA Register Select. The two register select lines are used to select the various registers inside the PIA. These two lines are used in conjunction with Internal Control Registers to select a particular register that is to be written or read. The Register select lines should be stable for the duration of the E pulse while in the read or write cycle.
37-38	$\overline{\text{IRQB}}$, $\overline{\text{IRQA}}$	$\overline{\text{Interrupt Request}}$. The active low $\overline{\text{Interrupt Request}}$ lines ($\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$) act to interrupt the MPU either directly or through interrupt priority circuitry. These lines are "open source" (no load device on the chip) and are capable of sinking a current of 1.6 mA from an external source. This permits all interrupt request lines to be tied together in a wire-OR configuration.

TABLE 4 (continued)

Pin	Designation	Description of Function
		Each Interrupt Request line has two internal interrupt flag bits that will cause the Interrupt Request line to go low. Each flag bit is associated with a particular peripheral interrupt line. Also four interrupt enable bits are provided in the PIA which may be used to inhibit a particular interrupt from a peripheral device. Servicing an interrupt by the MPU may be accomplished by a software routine that, on a prioritized basis, sequentially reads and tests the two control registers in each PIA for interrupt flag bits that are set. The Interrupt Flag is cleared (zeroed) as a result of an MPU Read Peripheral Data Operation.
39	CA2	Peripheral Control (CA2). The peripheral control line CA2 can be programmed to act as an interrupt input or as a peripheral control output. As an output, this line is compatible with standard TTL; as an input the internal pullup resistor on this line represents one standard TTL load. The function of this signal line is programmed with Control Register A.

Six locations in the PIA are accessible to the MPU data bus; two peripheral registers, two data direction registers, and two control registers. Table 5 shows the method in which these locations are selected and controlled by inputs RS0 and RS1 and control register bit two (7).

Control of the direction of data through each corresponding peripheral data line is accomplished by the CPU by means of the data direction registers. A data direction register bit set to "0" configures the peripheral data line as an input, and a bit set to "1" sets the peripheral line as an output.

The control registers (CRA and CRB) allow the CPU of the 6800 system to control operation of the four peripheral control lines CA1, CA2, CB1, and CB2. Also, these registers allow the CPU to enable the interrupt lines and monitor the status of the interrupt flags. Table 6 shows the format of the control registers. Bits zero through five may be either read or written by the CPU, as the proper programming occurs. Bits six and seven are read only and are modified by external interrupts occurring on any of the four control lines.

Bit two, the data direction access control bit, allows selection of either a peripheral interface register or the data direction register. Bits six and seven, the interrupt flag bits, are set by active transitions of signals on the four interrupt and peripheral status lines. These bits cannot be set by the CPU, but are reset indirectly by a read peripheral data operation. The two lowest order bits, zero and one, are used to control the interrupt lines CA1 and

TABLE 5

SELECTION OF PIA LOCATIONS

RS1	RS0	Control Register Bit CRA-2	Control Register Bit CRB-2	Location Selected
0	0	1	X	Peripheral Register A
0	0	0	X	Data Direction Register A
0	1	X	X	Control Register A
1	0	X	1	Peripheral Register B
1	0	X	0	Data Direction Register B
1	1	X	X	Control Register B

X = Don't care.

TABLE 6

CONTROL REGISTER FORMAT

	7	6	5	4	3	2	1	0
CRA	IRQA1	IRQA2	CA2 Control			DDRA Access	CA1 Control	
	7	6	5	4	3	2	1	0
CRB	IRQB1	IRQB2	CB2 Control			DDRB Access	CB1 Control	

CB1. Bit zero is used to enable the CPU interrupt signals IRQA and IRQB. Bit one determines the active transition of the interrupt signals. For example, if CRA-0 is a "1" and CRA-1 is a "0" an interrupt flag is set, CRA-7, when CA1 becomes a "0." However, if CRA-0 were a "1" and CRA-1 a "1," then the interrupt flag is set when CA1 becomes a "1" (Table 7). Bits three, four, and five of the control registers are used to control CA2 and CB2 peripheral control lines. These bits determine if the control lines are used for an interrupt input or an output control signal. As bit five becomes a "1," bits three through five become output lines which are used to control peripheral data transfers. Tables 8 and 9 describe the characteristics of bits three through five of both control registers (7).

As previously discussed, these functions of the PIA are merely software devices which the programmer can use to define the I/O features of a particular application. A likely possibility would be to use the control register lines as both output signals and interrupt devices in one program. All this technique would require are a few instructions to convert the control register bits to the correct sequence desired.

Intel 8080

The Intel 8080 microcomputer set consists of an 8080 microprocessor as the CPU, RAM and/or ROM memory, an I/O module, and if desired, an output module. These functional units, shown in Figure 14, are buffered and are interconnected by the following six buses:

TABLE 7

CONTROL OF THE INTERRUPT INPUT SIGNALS

CRA-1 (CRB-1)	CRA-0[a] (CRB-0)	Interrupt Input CA1 (CB1)	Interrupt Flag[b] CRA-7 (CRB-7)	MPU Interrupt Request IRQA (IRQB)
0	0	↓[c] Active	Set high on ↓ of CA1 (CB1)	Disabled—IRQ remains high
0	1	↓ Active	Set high on ↓ of CA1 (CB1)	Goes low when the interrupt flag bit CRA-7 (CRB-7) goes high
1	0	↑[d] Active	Set high on ↑ of CA1 (CB1)	Disabled—IRQ remains high
1	1	↑ Active	Set high on ↑ of CA1 (CB1)	Goes low when the interrupt flag bit CRA-7 (CRB-7) goes high

[a]If CRA-0 (CRB-0) is low when an interrupt occurs (interrupt disabled) and is later brought high, IRQA (IRQB) occurs on the positive transition of CRA-0 (CRB-0).

[b]The Interrupt flag bit CRA-7 is cleared by an MPU Read of the A Data Register, and CRB-7 is cleared by an MPU Read of the B Data Register.

[c]↓ indicates negative transition (high to low).

[d]↑ indicates positive transition (low to high).

TABLE 8

CONTROL OF INTERRUPT INPUTS, CRA-5 IS "0"

CRA-5 (CRB-5)	CRA-4 (CRB-4)	CRA-3[a] (CRB-3)	Interrupt Input CA2 (CB2)	Interrupt Flag[b] CRA-6 (CRB-6)	MPU Interrupt Request IRQA (IRQB)
0	0	0	↓[c] Active	Set high on ↓ of CA2 (CB2)	Disabled—IRQ remains high
0	0	1	↓ Active	Set high on ↓ of CA2 (CB2)	Goes low when the interrupt flag bit CRA-6 (CRB-6) goes high
0	1	0	↑[d] Active	Set high on ↑ of CA2 (CB2)	Disabled—IRQ remains high
0	1	1	↑ Active	Set high on ↑ of CA2 (CB2)	Goes low when the interrupt flag bit CRA-6 (CRB-6) goes high

[a]If CRA-3 (CRB-3) is low when an interrupt occurs (Interrupt Disabled) and is later brought high, IRQA (IRQB) occurs on the positive transition of CRA-3 (CRB-3).

[b]The Interrupt flag bit CRA-6 is cleared by an MPU Read of the A Data Register and CRB-6 is cleared by an MPU Read of the B Data Register.

[c]↓ indicates negative transition (high to low).

[d]↑ indicates positive transition (low to high).

TABLE 9

CONTROL OF BITS THREE, FOUR AND FIVE

CRA-5	CRA-4	CRA-3	Cleared	Set
			Control of CA2	
1	0	0	Low on negative transition of E after an MPU Read "A" Data operation.	High on an active transition of the CA1 signal.
1	0	1	Low immediately after an MPU Read "A" Data operation.	High on the negative edge of the next "E" pulse.
1	1	0	Low when CRA-3 goes low as a result of an MPU Write in Control Register "A."	Always low as long as CRA-3 is low.
1	1	1	Always high as long as CRA-3 is high.	High when CRA-3 goes high as a result of a Write in Control Register "A."
			Control of CB2	
1	0	0	Low on the positive transition of the first E pulse following an MPU Write "B" Data Register operation.	High when the interrupt flag bit CRB-7 is set by an active transition of the CB1 signal.
1	0	1	Low on the positive transition of the first E pulse following an MPU Write "B" Data Register operation.	High on the positive transition of the next "E" pulse.
1	1	0	Low when CRB-3 goes low as a result of an MPU Write in Control Register "B."	Always low as long as CRB-3 is low. Will go high on an MPU Write in Control Register "B" that changes CRB-3 to "one."

TABLE 9 (continued)

CRA-5	CRA-4	CRA-3	Cleared	Set
1	1	1	Always high as long as CRB-3 is high. Will be cleared when an MPU Write Control Register "B" results in clearing CRB-3 to "zero."	High when CRB-3 goes high as a result of an MPU Write into Control Register "B."

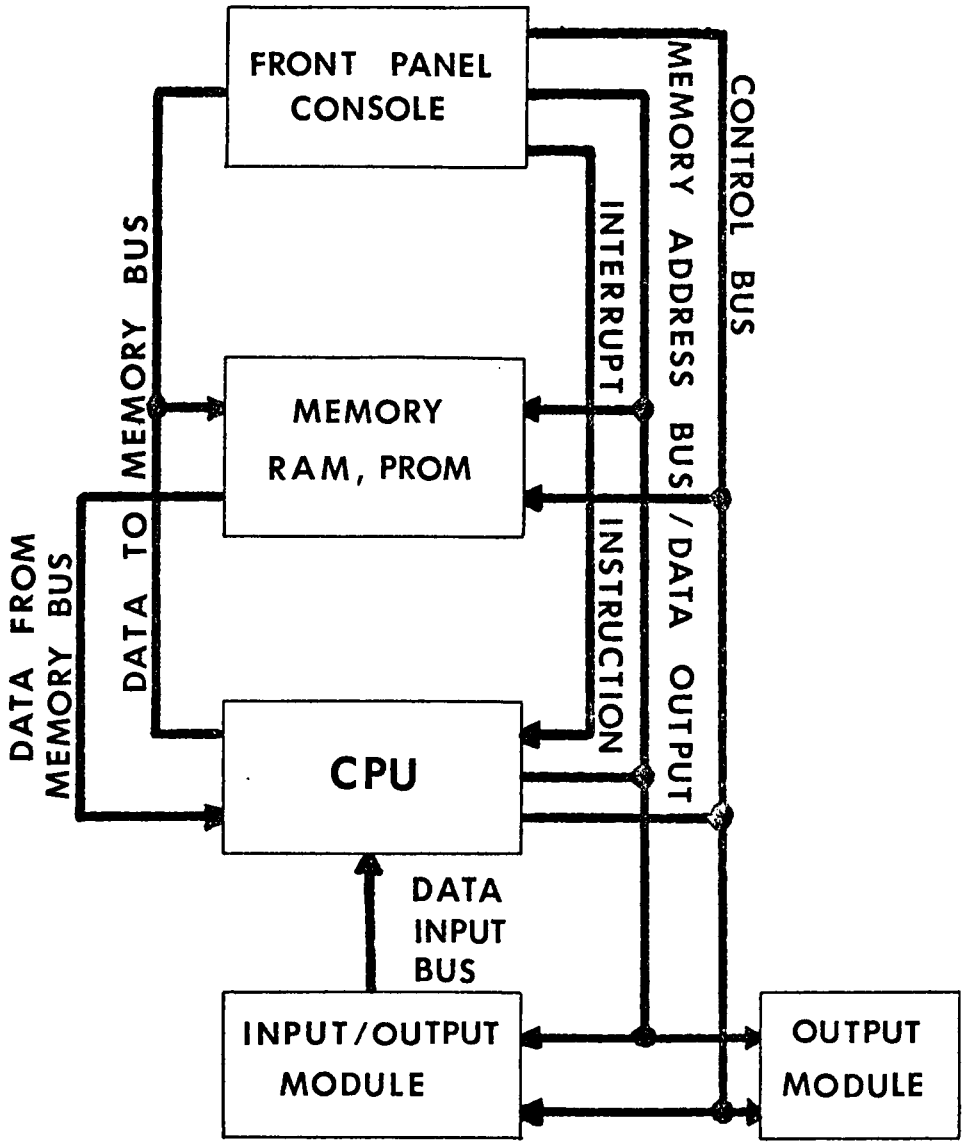


Figure 14. Block Diagram of Intel 8080 Microcomputer

1. Memory address bus which carries memory addresses from the CPU to memory, I/O, and output modules.
2. Memory bus which carries data from memory to the CPU.
3. Data input bus which carries data from the input ports to the CPU.
4. Interrupt instruction bus which allows the 8080 to receive an interrupt signal.
5. Output data bus which carries data from the CPU to memory, I/O, and output modules.
6. Control bus which controls instruction execution.

Memory operations, both read and write, are performed to either obtain data from a memory location and put that data in the CPU or to send data from the CPU to a memory location. A memory read in the 8080 is performed when the CPU sends a memory address to the memory devices. Then the memory device sends the data in the memory location to the CPU. A memory write operation is accomplished when the CPU sends a memory address to the memory device. The CPU then sends the data to be stored in the memory location. Finally, the CPU sends a control signal to the memory device to strobe the data into memory.

Input/output operations performed by the Intel 8080 are accomplished in order to obtain or send data from or to an external device. An input is accomplished when the CPU sends an I/O address, which specifies the device to be used for an input operation, to the I/O module. The I/O modules respond by inputting the data which is present on the selected input port. Output operations are accomplished

when the CPU sends an I/O address, specifying the device to be used for an output operation, to the I/O and output modules. Then the CPU sends the data which is to be output and an I/O write command to the modules. Finally, the I/O module latches the data and sends the data, the CPU has supplied, to the selected external device.

Interrupt operations are performed whenever an external device, which requires servicing, sends an interrupt signal to the CPU. CPU actions are to interrupt the normal sequencing of instructions, perform the operations required by the external device, and return to the point at which the normal sequence was interrupted. An interrupt operation is performed when the external device sends the interrupt signal to the CPU. The CPU acknowledges the interrupt sent by the external device and receives an interrupt instruction provided by this interrupting device. Then the CPU executes the interrupt instruction (8).

A forty pin LSI chip contains the Intel 8080 microprocessor, the CPU of the Intel 8080 microcomputer. Table 10 contains a list of the control signals on the 8080 microprocessor. Eight-bits of status information are also supplied by the 8080 at the beginning of each machine cycle (Table 11). These eight-bits inform the microcomputer of the status of the processor.

Instructions in the 8080 require from one to three bytes of data and from one to five machine cycles (designated M1, M2, M3, M4, and M5). In addition, each machine cycle requires from three to five states (designated T1, T2, T3, T4, and T5). Each state has the

TABLE 10

FUNCTIONAL DESCRIPTION OF EACH PIN OF THE INTEL 8080

Pin	Designation	Description of Function
2	V_{SS}	Ground Reference
3-10	D_7-D_0 (Input/ output) tri-state	DATA BUS; the data bus provides bidirectional communication between memory and I/O devices for instructions and data transfers. D_0 is the least significant bit.
11	V_{bb}	-5 volts (substrate bias)
12	RESET (input)	RESET; while the RESET signal is activated, the content of the program counter is cleared and the instruction register is set to 0. After RESET, the program will start at location 0 in memory. The INTE and HLDA flip-flops are also reset.
13	HOLD (input)	HOLD; the HOLD signal requests the CPU to enter the HOLD state. The HOLD state allows an external device to gain control of the 8080 address and data bus as soon as the 8080 has completed its use of these buses for the current machine cycle. It is recognized under the following conditions: •the CPU is in the HALT state •the CPU is in the T2 or TW state and the READY signal is active. As a result of entering the HOLD state the CPU ADDRESS BUS (A_{15}-A_0) and DATA BUS (D_7-D_0) will be in their high impedance state. The CPU acknowledges its state with the HOLD ACKNOWLEDGE (HLDA) pin.
14	INT (input)	INTERRUPT REQUEST; the CPU recognizes an interrupt request on this line at the end of the current instruction or while halted. If the CPU is in the HOLD state or if the Interrupt Enable flip-flop is reset it will not honor the request. The CPU acknowledges acceptance of an interrupt by sending out the INTA (Interrupt Acknowledge) status signal at SYNC time. During the next instruction fetch

TABLE 10 (continued)

Pin	Designation	Description of Function
		cycle the program counter is not advanced and a 1 byte instruction (usually RESTART) can be inserted.
15 and 22	ϕ_1 & ϕ_2	Clock inputs.
16	INTE (output)	INTERRUPT ENABLE; indicates the content of the internal interrupt enable flip/flop. This flip/flop may be set or reset by the EI and DI instructions and inhibits interrupts from being accepted by the CPU if it is reset. It is automatically reset (disabling further interrupts) at time T1 of the instruction fetch cycle (M1) when an interrupt is accepted and is also reset by the RESET signal.
17	DBIN (output)	DATA BUS IN; the DBIN signal indicates to external circuits that the data bus is in the input mode. This signal should be used to enable the gating of data onto the 8080 data bus from memory or I/O.
18	$\overline{WR}$ (output)	WRITE; the $\overline{WR}$ signal is used for memory WRITE or I/O output control. The data on the data bus is stable while the WR signal is active ($\overline{WR}$ = 0).
19	SYNC (output)	SYNCHRONIZING SIGNAL; the SYNC pin provides a signal to indicate the beginning of each machine cycle. (Instructions can be executed in 1, 2, 3, 4 or 5 machine cycles, and the status information of each machine cycle is sent to external latches at SYNC time.)
20	V_{CC}	+5 volts.
21	HLDA (output)	HOLD ACKNOWLEDGE; the HLDA signal appears in response to the HOLD signal and indicates that the data and address bus will go to the high impedance state. The HLDA signal begins at: T3 for Read memory or input. The Clock

TABLE 10 (continued)

Pin	Designation	Description of Function
		Period following T3 for WRITE memory or OUTPUT operation. In either case, the HLDA signal appears after the rising edge of ϕ_1 and high impedance occurs after the rising edge of ϕ_2.
23	READY (input)	READY; the READY signal indicates to the 8080 that valid memory or input data is available on the 8080 data bus. This signal is used to synchronize the CPU with slower memory or I/O devices. If after sending an address out the 8080 does not receive a READY input, the 8080 will enter a WAIT state for as long as the READY line is low.
24	WAIT (output)	WAIT; the WAIT signal acknowledges that the CPU is in a WAIT state.
28	V_{DD}	+12 volts.
1, 25-27, 29-40	A_{15}-A_0 (output tri-state)	ADDRESS BUS; the address bus provides the address to memory (up to 64K 8-bit words) or denotes the I/O device number for up to 256 input and 256 output devices. A_0 is the least significant address bit.

TABLE 11

DESCRIPTION OF STATUS INFORMATION BITS
OF THE INTEL 8080

Designation	Data Bus Bit	Description of Status Bit
HLTA	D_3	Acknowledge signal for HALT instruction.
INTA*	D_0	Acknowledge signal for INTERRUPT request. Signal should be used to gate a restart instruction onto the data bus when DBIN is active.
INP*	D_6	Indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when DBIN is active.
OUT	D_4	Indicates that the address bus contains the address of an output device and the data bus will contain the output data when $\overline{WR}$ is active.
MEMR*	D_7	Designates that the data bus will be used for memory read data.
M_1	D_5	Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction.
STACK	D_2	Indicates that the address bus holds the pushdown stack address from the Stack Pointer.
$\overline{WO}$	D_1	Indicates that the operation in the current machine cycle will be a WRITE memory or OUTPUT function ($\overline{WO}$ = 0). Otherwise, a READ memory or INPUT operation will be executed.

*These three status bits can be used to control the flow of data onto the 8080 data bus.

duration of 0.5 microseconds. Cycle M1 is the operation code fetch and lasts four or five states. Cycles M2 through M5 last from three to five clock periods, states, each.

Instruction state requirements range from a minimum of four states for nonmemory referencing instructions up to a maximum of eighteen states for the most complex instructions. Since the maximum clock frequency is two megahertz, assembly language instructions can be executed by the 8080 in two to nine microseconds.

Figure 15 shows the basic instruction cycle and is used to develop the operation of the 8080 microprocessor. During state T1 of M1 the contents of the program counter are sent to the address bus, SYNC is true, and the data bus contains status information pertaining to the cycle that is being initialized. State T1 is followed by T2 during which the READY, HOLD, and HALT acknowledge signals are tested. If READY is true, T3 is entered, otherwise the 8080 will go into the wait state (Tw) and stay there until READY becomes true. This allows the 8080 to be synchronized to memory with any access time and to any I/O device. By controlling the READY line a programmer can single step through any program. During state T3, data from memory is available on the data bus and is transferred into the instruction register. The instruction decoder and control sections then generate the signals to control internal data transfers, timing, and machine cycle requirements of the instructions encountered. After the cycle is complete, either at the end of T4 or T5, the 8080 returns to T1 and enters machine cycle M2, unless the instruction required only one

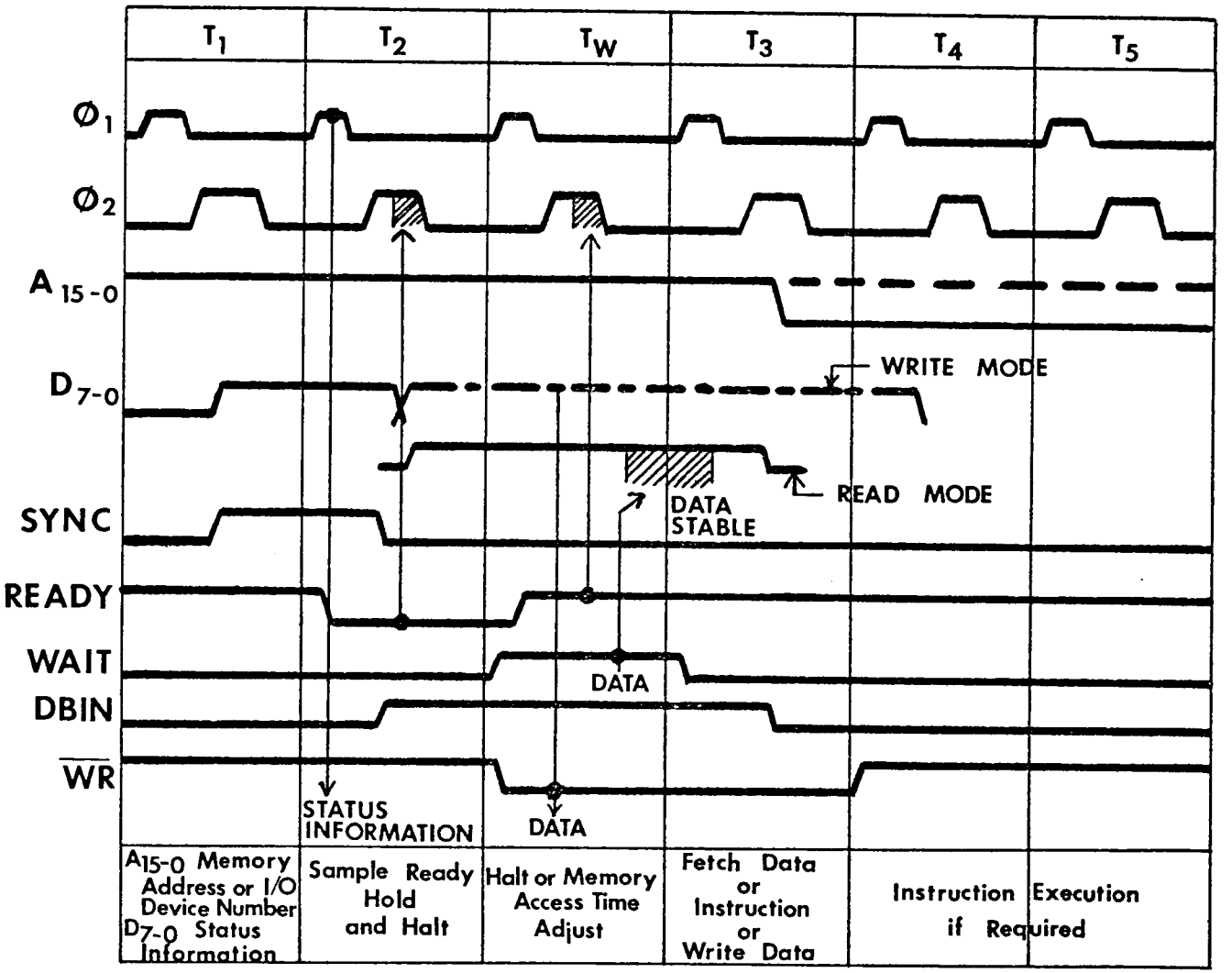


Figure 15. Instruction Cycle of the Intel 8080 Microcomputer

machine cycle for execution. As many cycles and states as are required for completion of the instruction are performed at which time the 8080 enters a new M1 cycle.

Only during the last machine cycle of an instruction is the interrupt request line tested. If an interrupt has been requested, a special M1 cycle is entered in which no program counter incrementing takes place. Also during this special M1 cycle an INTERRUPT ACKNOWLEDGE status signal is sent to the interrupting device. Then one of eight possible single byte calls is sent to the 8080 (8).

Figure 16 shows the block diagram of the 8080 microprocessor. The 8080 consists of the following functional units: register array and address logic, arithmetic logic unit, instruction register and control section, and a bidirectional tri-state data bus buffer. Six sixteen-bit registers are contained within the register section; a program counter, a stack pointer, six eight-bit general purpose registers arranged in pairs, and a temporary register pair. As in other microprocessors, the program counter contains the memory address of the current program instruction. The stack pointer register differs in that any portion of RAM memory can be used as this register. All six general purpose registers can be used as either single registers or register pairs, and the temporary register is used for internal execution and is not program accessible.

Transfer of eight-bit data bytes is accomplished between the internal bus and the register array via the register select multiplexer. Sixteen-bit transfers proceed between the register array

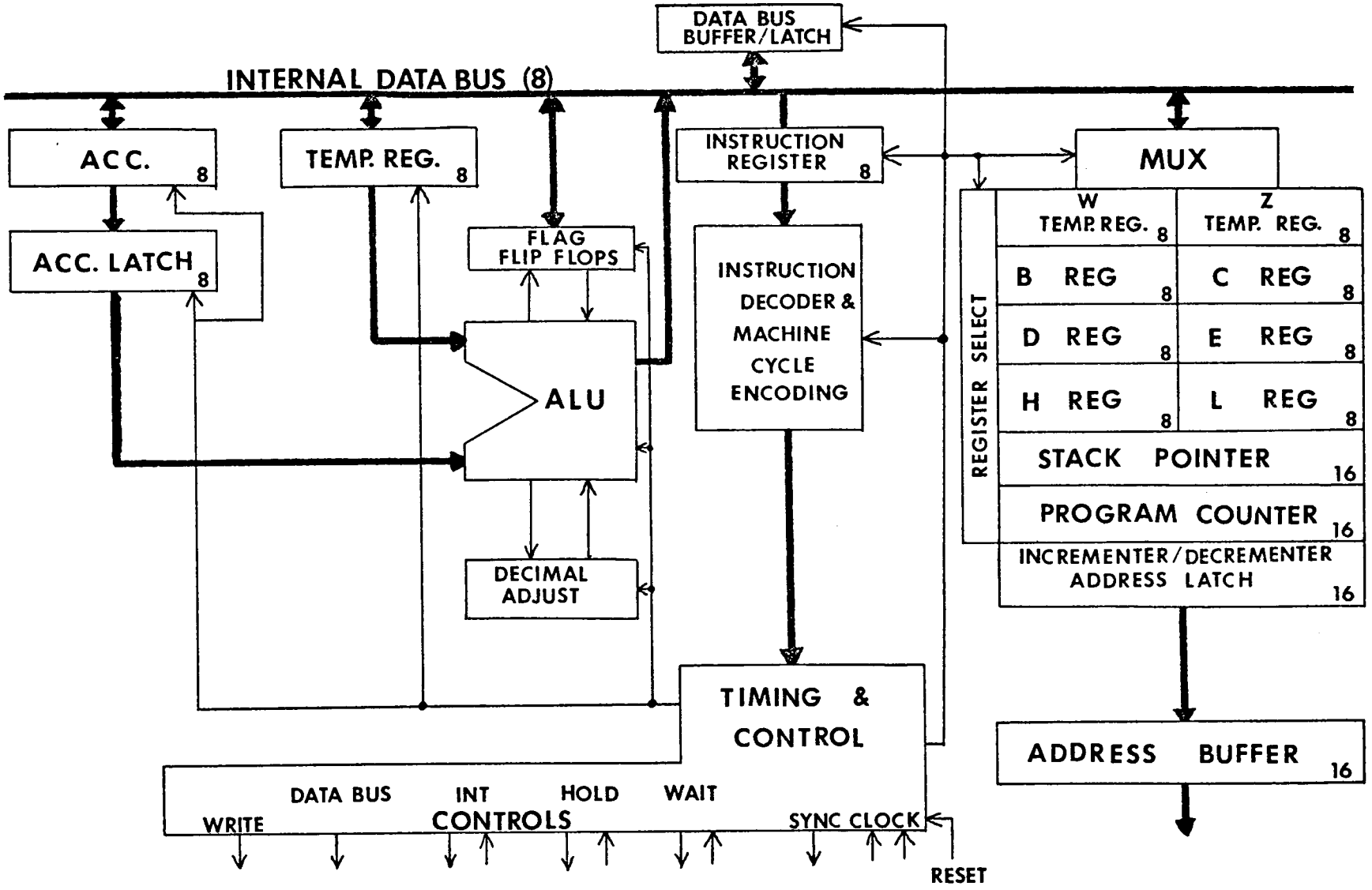


Figure 16. Intel 8080 Microprocessor

and address latch or the incrementer-decrementer circuit. This incrementer-decrementer circuit receives data from the address latch and sends the new data, which can be incremented, decremented, or left alone, to the register array.

The ALU contains an eight-bit accumulator and a temporary carry flip-flop, a five-bit flag register (zero, carry, sign, parity, and auxiliary carry), and an eight-bit temporary register. Arithmetic, logical, and rotate operations are performed in the ALU. Results can be transferred to either the internal bus or the flag register. The temporary register receives data from the internal bus and sends all or portions of the data to the ALU, the flag register, or the internal bus. The accumulator loads data from the ALU and the internal bus and transfers data to the temporary register and the internal bus (8).

Input/output operations performed by the Intel 8080 are two byte instructions, which require three machine cycles. The first byte contains the operation to be performed, and the second byte contains the eight-bit address of the external device selected. Execution of the instruction, performed during the third cycle, varies depending upon whether an input or output has been designated. For an input operation, an INP status bit is developed. This status bit is sent to the I/O module connector to select and enable the appropriate input port. Input data is then forwarded to the main bus of the processor, and the data is received and stored in the accumulator. Output operations differ from input operations only in that the accumulator data is placed on the main bus and routed to the appropriate port.

III. MICROPROGRAMMABLE MICROCOMPUTERS

National IMP-16C

The National IMP-16C microcomputer is available in a variety of configurations: as a system, as a CPU card, and as a component kit. A discussion of the CPU card, the IMP-16C, microcomputer is undertaken in the following paragraphs. The IMP-16C is unique in the microcomputer field for two reasons. First, there is no single chip microprocessor which performs as the CPU of the microcomputer. The IMP-16C consists of four four-bit Register and Arithmetic Logic Units (RALUs) and a Control and Read-Only Memory (CROM) device which control the operations of the computer. Second, the IMP-16C is microprogrammable, i.e., a programmer has the capability of devising a unique instruction set applicable only to a certain application (11, 12).

A block diagram of the IMP-16C is given in Figure 17. Addresses and data are both transferred from the data buffer over the sixteen-bit buffered data out (BDO) bus. This bus is used for transferring data to peripheral units and external devices. The on card memory receives data directly from the BDO bus. Addresses are loaded into the address latches and brought out to external devices. Addresses are also routed directly to the on card memory from the address latches (9).

Four RALUs, a four-bit control bus, and either one or two CROMs comprise the CPU. Both the RALUs and the CROMs are individual LSI devices. An RALU is a four-bit unit, and four RALUs in parallel form

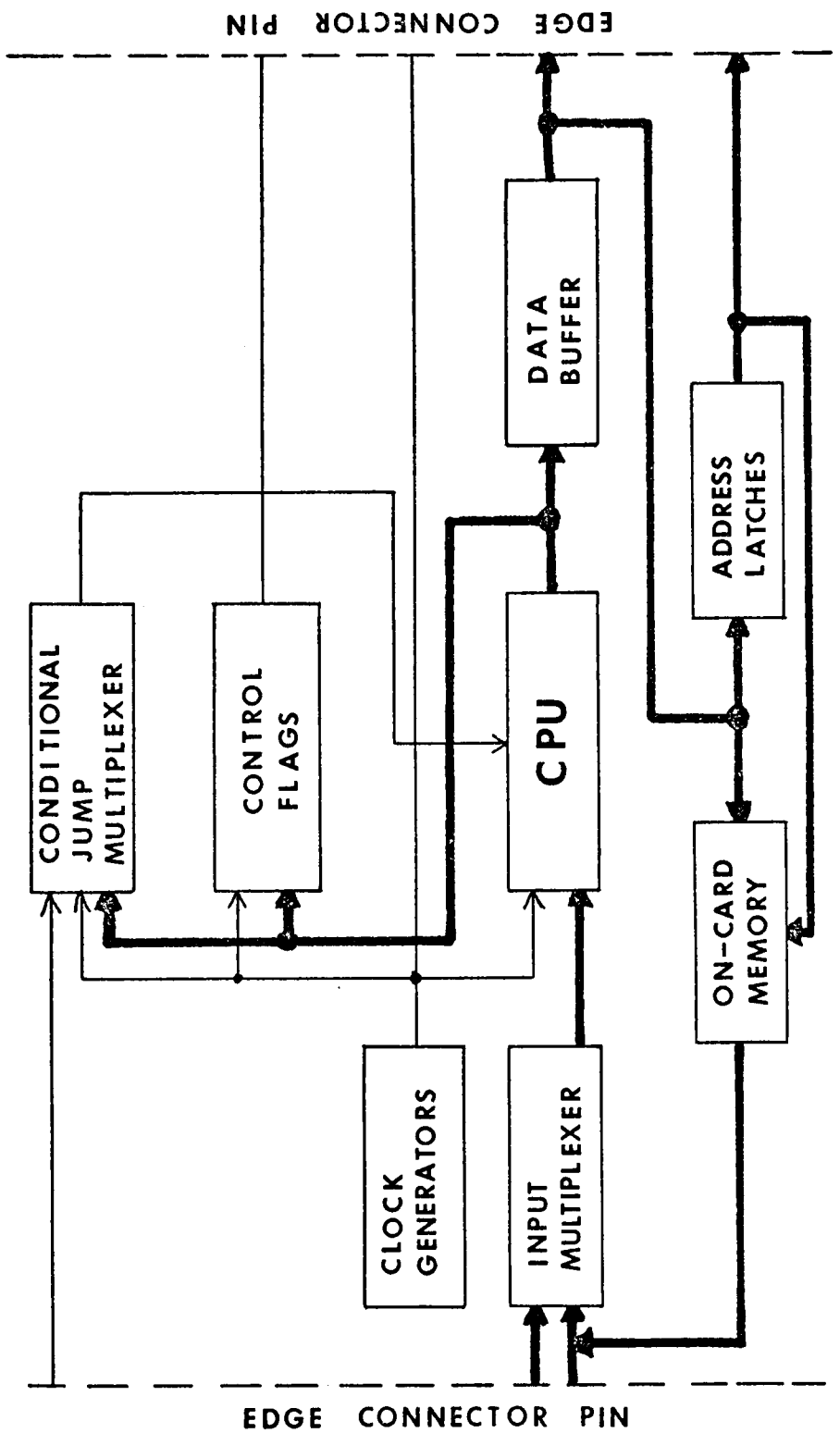


Figure 17. Block Diagram of National IMP-16C Microcomputer

the IMP-16C sixteen bit unit. The RALUs are controlled by the microinstructions stored in the CROMs.

Figure 18 shows the arithmetic section comprised with the four RALUs. Seven general purpose registers are provided and any one of them can be loaded onto the A- or B- bus for processing by the ALU. One register is a program counter, one a memory data register, one a memory address register, and four are accumulators. The program counter holds the address of the next instruction to be executed. The program counter is incremented by one immediately following the fetching of each instruction. Data transferred from memory to the processor is stored in the memory data register. For example, when fetching data, the address is placed in the memory address register and the fetch instruction causes the data work to be transferred from the designated memory location to the memory data register. All four accumulators can hold operands or results of data manipulation instructions during arithmetic and logical operations. An accumulator must be specified by the programmer in order to take part in any operation.

A sixteen word last in, first out stack is provided by the RALUs. This stack can be accessed over both the A- and the R- buses. As the stack becomes full, a stack full signal is provided as status information. Other status information is provided by a sixteen-bit status register. Link, Overflow, Carry, and Flag indicators are available in the sixteen-bits. Bit fifteen, the Link flag, can be included in the shift operations. Bits fourteen and thirteen, the

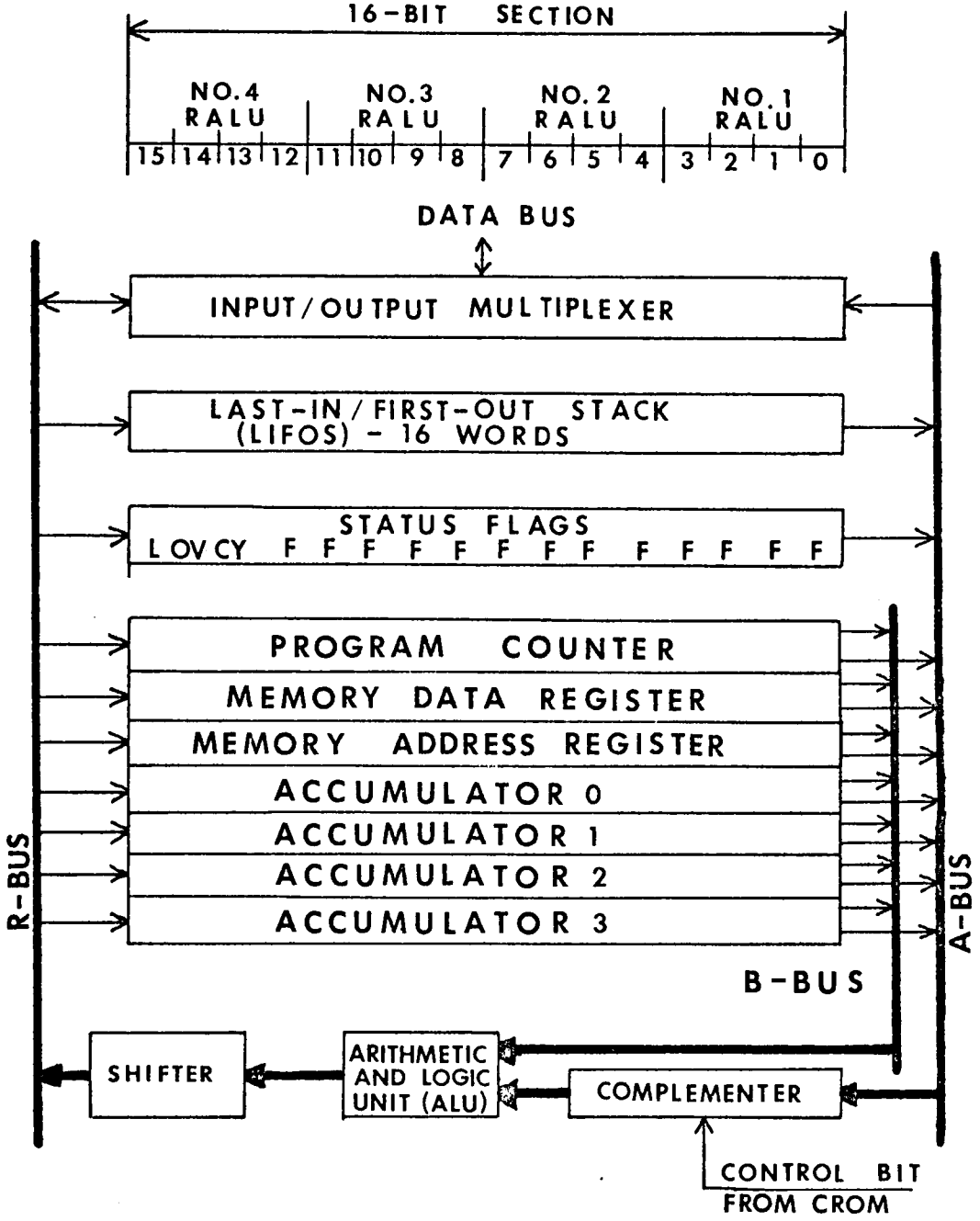


Figure 18. RALU Block Diagram

Overflow and Carry, respectively, are functions of the results of ADD operations. All three of these flags can be disabled, allowing the flag positions to be used for general purpose applications. Bits zero through twelve, Flag bits, provide a flag for use when the programmer wants to save status bits on the stack (10).

The CROM provides read-only microprogram storage and control logic for use with the RALU. Storage for 100 microinstructions each 23-bits long is provided in the CROM. Also, circuitry is provided for sequencing subroutine execution and circuitry for translation of microinstructions into RALU commands. Figure 19 is a block diagram of the CROM. Primary control of the RALU is accomplished over a four-bit control bus. This control bus is time-multiplexed to transfer four-bit words of control information each machine cycle.

During an instruction fetch the nine most significant bits of the instruction word are transferred to the CROM. The nine bits contain the opcode and other control information of the instruction work. All instruction bits are decoded, then the ROM address control in the CROM directs the control sequence to an entry point in the microprogram. This sequence continues until the execution of the fetched instruction is completed. Another fetch cycle is then performed by the CROM, thus repeating the instruction decoding process (11).

The basic machine cycle of the IMP-16C consists of the execution of a single microprogram step. Eight time intervals (T1, T2, . . . T8) are comprised within each time period. As shown in Figure 20, clock

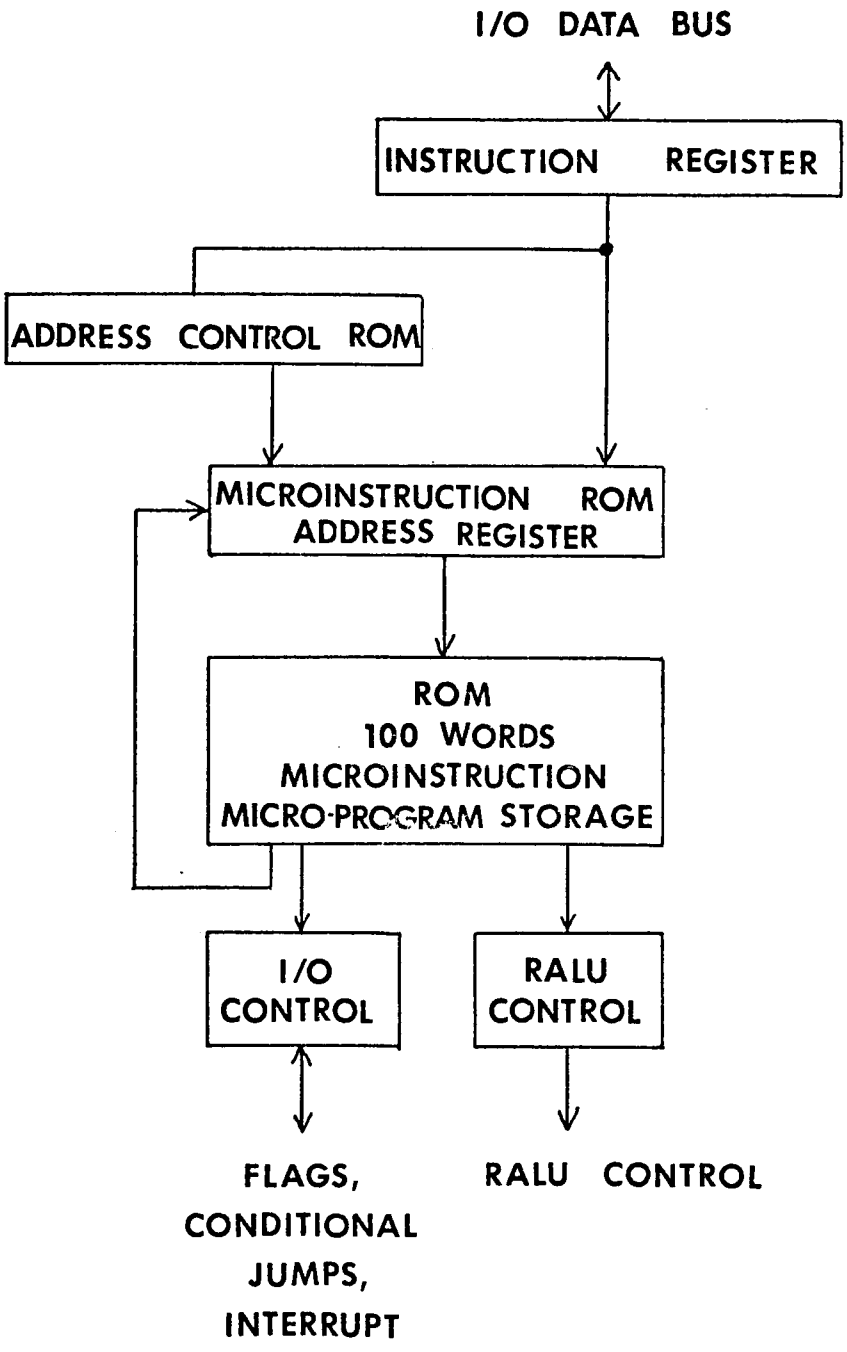


Figure 19. Block Diagram of CROM

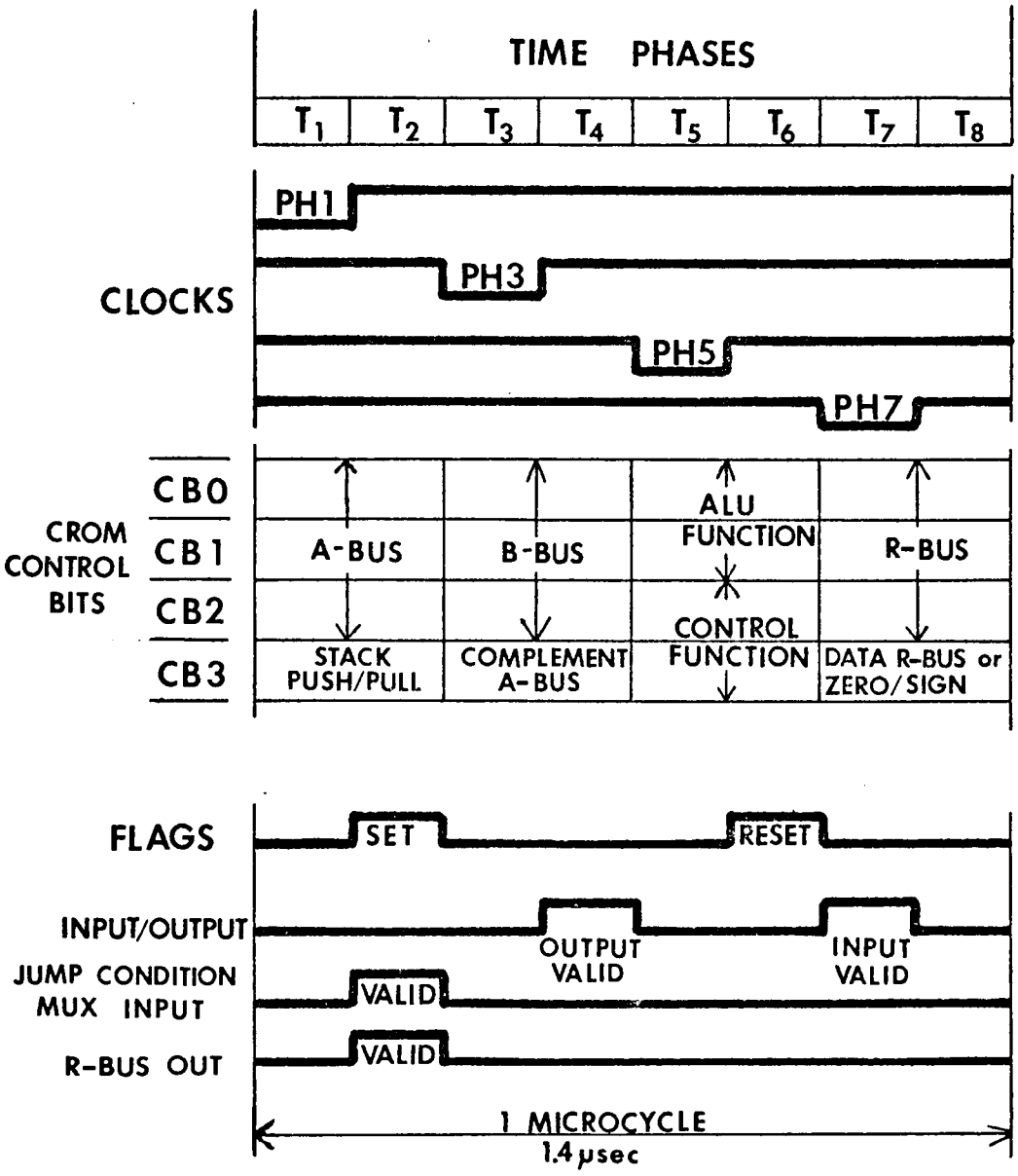


Figure 20. Timing Diagram of IMP-16C System

pulses occur in the four clock lines at respective odd-time intervals, T1, T3, T5, and T7. During these phases control information is passed from the CROM to the RALU. Encoding of the control bits, of Figure 19, between the RALU and the CROM, is described for each of these odd-time phases.

Phase 1 control bits. If the three-bit A- bus address field is nonzero, the contents of the RALU register designated by the macroinstruction are gated onto the A- bus. If the A- bus field is zero and the push-pull control bit is "1," the contents of the top of the stack are gated onto the A- bus. If the A- bus field is zero and the push-pull control bit is "0," the value zero is gated onto the A- bus.

Phase 3 control bits. If the three-bit B- bus address field is nonzero, the contents of the register designated in the macroinstruction are gated onto the B- bus. If the B- bus field is zero, the value zero is gated onto the B- bus. The complement A- bus causes the A input to the ALU to be complemented.

Phase 5 control bits. The ALU bits and control functions bits designate a function as specified in Table 12.

Phase 7 control bits. If the three-bit R- bus field is nonzero the contents of the R- bus are gated onto the register addressed by this field. If the control function bits of Phase 5 are not 01 then CB3 specifies the source of the data gated onto the R- bus. If CB3 is "0," the source is the output of the ALU, and if CB3 is "1," the data

TABLE 12

ALU AND CONTROL FUNCTION BITS

Code	Function
	ALU Function Bits
00	AND
01	XOR
10	OR
11	ADD
	Control Function Bits
00	No-Op (no operation for control bits only)
01	R—0/SIGN (zero or sign propagation)
10	LSH (Left Shift)
11	RSH (Right Shift)

comes from an external source. If 01 is specified in Phase 5, then CB3 specifies whether zero or the sign is propagated throughout the more significant byte. If the R- bus field is nonzero and the push-pull control bit was active during Phase 1, a data word is pushed onto the stack from the R- bus (9).

Figure 20 (page 66) also indicates the means of controlling the flag flip-flops. The IMP-16C has a unique flag address for each machine cycle. A flag can be set during T2 and reset during T6 of the same cycle. For example, if a conditional jump is performed, the condition is tested during T2. If the condition were not tested at T2, the jump might not occur since the condition is reset at T6.

Data transfers between the RALU and the data bus can occur at three times during each machine cycle. During T4, the data word presently on the A- bus is gated onto the data bus. This information is either ouput data or address value used for loading the address register latch. At T2 the contents of the R- bus are gated onto the data bus. This data provides inputs, for condition instructions, to permit testing of the results of the previous instruction. During T7 data to be transferred to the RALU is on the data bus. This data can be gated onto the R- bus and subsequently stored in one of the accumulators.

Input/output operations are performed with register in and register out macroinstructions. These instructions simply address a particular device and initiate data exchanges. The address of the I/O device is determined by the sum of the contents of accumulator three

and the seven-bit control field of the instruction. One scheme used for programming the sixteen-bits available for the address and command code is to use the low order three-bits to define the I/O order. These three order bits permit eight different operations for each I/O device, i.e., an order of read data, write data, read status, reset, rewind, or backspace could be given. The next three-bits are used to define the device address. Each external device must then decode the address field of the I/O instruction to determine which peripheral is to respond. Peripheral devices are then accessed by read peripheral and write peripheral control flags in conjunction with the peripheral address.

IV. SUMMARY

Table 13 summarizes the architecture of the microcomputers discussed. Architecture of the four-bit machines differs greatly from that of the eight- and sixteen-bit microcomputers. Four-bit microcomputers do not allow much, if any at all, I/O interfacing with external devices. The eight- and sixteen- bit machines are designed specifically for interfacing to external components. However, different manufacturers create different I/O capabilities. Somewhat similar processors, as far as I/O capabilities are concerned, are the Intel 8080 and National IMP-16C. Both of these microcomputers provide data to an external device via an I/O port. The Motorola 6800 microcomputer, however, has been developed to allow programming techniques to supply I/O information. This I/O technique is

TABLE 13

COMPARISON OF THE ARCHITECTURES OF VARIOUS MICROCOMPUTERS

Microcomputer	Parts Family: Clock Driver, I/O Interface, UART/USRT/ACIA, RAM, ROM/PROM, Interface	Classification	Interrupts, One-chip CPU, Microprogrammed, Accessible Stack, BCD Arithmetic	Word Size (Data/Instruction)	Address Capacity	Clock (usec/Phases)	Register Add Time (usec)	Number of Registers: ALU	XR	GP	Stack Size (Nr × Bits)	Voltage Required
Rockwell PPS-4	X X X X X	4-bit CPU	X X	4/8	4K	5/2	5	1		1	2 × 12	+17
Fairchild PPS-25	X X X	4-bit CPU	X	4×25/12	6656	2.5/2	62.5	1			4 × 12	-9, +5
Intel 4004	X X X X X	4-bit CPU	X X	4/8	4K	1.35/2	10.8	1		16	3 × 12	+15, or -10, +5
Intel 4040	X X X X X	4-bit CPU	X X X	4/8	4K	1.35/2	10.8	1		24	7 × 12	+15, or -10, +5
Fairchild F-8	X X X X X X	8-bit CPU	X X X X	8/8	64K	0.5/2	2	65			RAM	+5, +12
Motorola 6800	X X X X	8-bit CPU	X X X X	8/8	64K	1/2	2	2	1		RAM	+5
Intel 8080	X X X X X X	8-bit CPU	X X X X	8/8	64K	0.5/2	2	1		8	RAM	±5, -12
National IMP-16C		16-bit CPU	X X X	16/16	64K	1.4/4	4.6	2	2		16 × 16	+5, -12

accomplished by the PIA device which is tied directly to the data bus. The PIA is program accessible and can function as either an input port, an output port, or both.

Although the CPU architecture is similar in all of the processors described, a difference is seen once the instruction set is defined. Chapter III develops the software capabilities of microcomputers, with examples. In developing these capabilities a greater understanding of the architectures just described is established.

CHAPTER III

SOFTWARE

Using microprocessors in digital circuit designs requires the knowledge of programming techniques. Designers must consider the use of computational algorithms, subroutines, indirect addressing and other software oriented factors to replace hardware components. A comparison of the instruction sets of various microprocessors establishes the capabilities of microprocessor software. Appendices A, B, and C contain the complete instruction sets of the Intel 8080, Motorola 6800, and National IMP 16-C and are used to develop examples.

Microprocessor instruction sets are described in four categories: (1) Date movement, (2) data manipulation, (3) decision and control, and (4) input/output.

I. DATA MOVEMENT

Data movement instructions are used to move information between internal locations. Figure 21 shows the overall diagram of the way data is manipulated. In any processor, an instruction word consists of an operation code and an operand code. Different techniques used for data movement are: immediate addressing, indexed addressing, and indirect addressing.

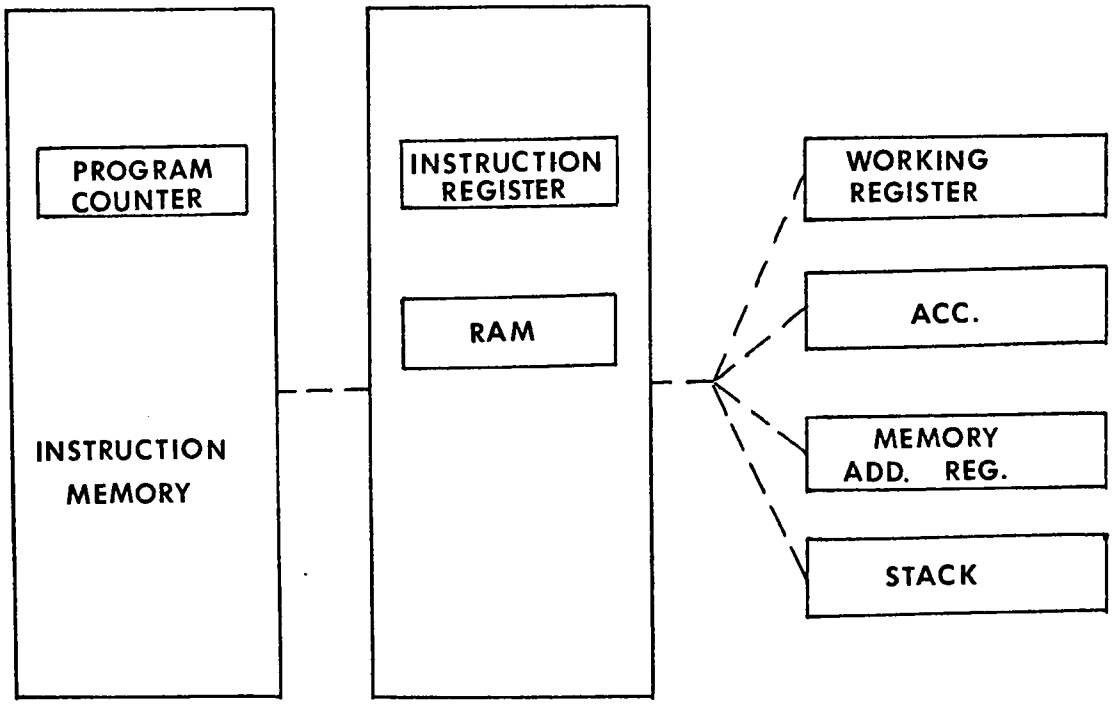


Figure 21. Data Movement Within the Major Storage Areas is Shown by Dashed Lines

Addressing Modes

Immediate addressing. Immediate addressing defines the data to be read or written as part of the instruction being executed. The data may be either a memory location or simply data. A typical software program for the Intel 8080 immediate addressing is shown in Table 14.

TABLE 14

INTEL 8080 INSTRUCTIONS TO PERFORM IMMEDIATE ADDRESSING

```
MVI     A, 8H       ;LOADS THE ACCUMULATOR
                    ;WITH 8 HEXADECIMAL
LDA     100H        ;LOADS THE ACCUMULATOR WITH
                    ;THE CONTENTS OF MEMORY
                    ;LOCATION 100 HEXADECIMAL
```

Indexed addressing. Indexed addressing generates a memory location without providing all sixteen bits in the current instruction. The memory location is obtained from the instruction adding the immediate data, designated by the second byte of the instruction, to an index register. A typical instruction using the National IMP-16C instruction set for indexed addressing is shown in Table 15.

Indirect addressing. Indirect addressing is accomplished by computing a memory address, called the direct address, using the address of the immediate data. Using index addressing is another way to obtain the direct address. The direct address is then used as the

TABLE 15

AN EXAMPLE OF INDEXED ADDRESSING USING
THE IMP-16C INSTRUCTION SET

```
AC 2 contains 01
AC 3 contains 2E
LD      AC0, NDATA (AC3)        ;LOADS INTO ACCUMULATOR AC0
                                ;THE VALUE STORED IN 012E
                                ;PLUS THE VALUE OF NDATA
```

memory location to be read or written. Indirect addressing allows any memory location to be the memory address register, and the contents of the memory location can be used to point to any word in memory (13). Typical examples, using the Intel 8080 and National IMP-16C instruction sets, are shown in Table 16.

Registers Used

Although microprocessors generally have several working registers, the essential register is the accumulator. The accumulator has access to read/write memory by instructions performing immediate addressing and data manipulation between the accumulator and memory. Being able to use the accumulator and the working registers reduces access time and increases bit efficiency. Less time is required by the user requesting a working register rather than memory location, and fewer bits are required to specify the working register rather than a memory location. As described in Chapter II, the Intel 8080 has an accumulator and six working registers, the Motorola 6800 two accumulators and an index register, and the National IMP-16C has four accumulators.

TABLE 16

INTEL 8080 AND NATIONAL IMP-16C INSTRUCTIONS TO PERFORM INDIRECT ADDRESSING

Intel 8080 Indirect Addressing			National IMP-16C Indirect Addressing	
LXI	B, 400H	;LOADS THE B AND C ;WORKING REGISTERS WITH ;0400 HEXADECIMAL.	LD ACO, @ 100 D(AC2)	;LOADS THE ACCUMULATOR ;ACO WITH THE CONTENTS ;OF MEMORY LOCATION
LDAX	B	;LOADS THE ACCUMULATOR ;WITH THE CONTENTS OF ;MEMORY LOCATION 0400 ;HEXADECIMAL.		;100 DECIMAL PLUS THE ;CONTENTS OF AC2.

The applicability of the working registers is determined by the software capability of the registers. One might ask, does the software allow for each working register to be:

1. Tested for zero?
2. Used for index addressing?
3. Loaded directly from memory?
4. Used for arithmetic or logic operations?

For example, all six registers of the Intel 8080 can be tested for zero, used for immediate addressing, and used in arithmetic and logic operations; therefore, during programming, these six working registers are very helpful. Since the Motorola and National microprocessors have two and four accumulators, respectively, greater flexibility exists for programming because all operations desired can be accomplished with any accumulator.

Most microprocessors have stack-oriented registers which can only be used on a last-in, first-out process. In order to use this register, programming algorithms must be developed such that the required data is always on top of the stack. Because the stack instruction refers only to the top register of the stack, these stack registers are more numerous than the individual registers and do not require instruction bits for addressing.

One of the most important registers in the microprocessor is the program counter. This counter points to the next location memory for an instruction-fetch operation. Being able to load or store the

program counter (PC) is extremely important, since a programmer can modify the instruction sequence by changing the PC.

II. DATA MANIPULATION

Data manipulation capabilities of microprocessors are usually limited to addition, subtraction, and some logical operations (Table 17). The Fairchild PPS-25 is the only microprocessor that features decimal arithmetic; all others perform binary operations and include a special instruction to convert binary numbers to binary-coded-decimal (BCD) numbers. Through mask-programmed options the Fairchild PPS-25 has the ability to specify one of the six fields upon which arithmetic is to occur. Since the words are organized with a maximum of 25 decimal, BCD, four-bit fields, part of the register can be treated as an exponent and part as the number. Program selection arithmetic can then be performed on these fields.

All other processors operate on words varying from four to sixteen bits, therefore multiple-word arithmetic is often required. Within the four bit calculator machines, the Intel 8080, and the Motorola 6800, a decimal adjust (DAA) instruction is provided. This DAA instruction takes four bits of binary data and formats the four bits into the corresponding four bits of BCD data, i.e., any four-bit binary number greater than nine is incremented by six to give the correct BCD number (Table 18).

Calculator oriented machines usually do not allow for logic operations, i.e., AND, OR, or EXCLUSIVE-OR, whereas the fixed

TABLE 17

DATA MANIPULATION CAPABILITIES OF MICROPROCESSORS

Arithmetic Functions	
add	with or without carry between accumulator and register, memory or immediate data multiple precision with skip if carry out
subtract	with or without carry between accumulator and register, memory or immediate data
increment/decrement	
multiply divide	by subroutine or special hardware
Logic Functions	
complement	
rotate shift	with or without carry
AND OR EXCLUSIVE OR	
compare accumulator	with register, memory or immediate data

TABLE 18

EXAMPLE OF OPERATIONS OF A DAA INSTRUCTION

```
0000 1011          ;BINARY NUMBER GREATER THAN NINE
     0110          ;INCREMENTED BY SIX
0001 0001 (11)     ;GIVES ELEVEN THE CORRECT BCD NUMBER
```

instruction and microprogrammable processors do (Appendices A, B, and C). However, nearly all microprocessors permit some form of complement, shift, and rotate operations. Usually the shift or rotate is by one bit, but the National IMP-16C allows rotation by an arbitrary number of bits, eliminating many instruction bits. Table 19 gives some examples of the logic operations, the complement, and shift or rotate instructions.

Increment and decrement arithmetic functions are accomplished along with test and conditional functions in order to control program loops. For example, the Rockwell PPS-4 has a one byte instruction that adds a four-bit immediate field to the accumulator. If the carry out is set (the accumulator has reached its maximum value), the next instruction is skipped, but the carry flip-flop is not disturbed. The National IMP-16C has an analogous one-word instruction; Add Immediate and Skip if Zero (AISZ). The Intel 4004 processor allows incrementing any one of sixteen four-bit registers and, if the result is zero, the next instruction sequence is performed. If the result is not zero, a jump occurs to an immediate location designated by the second byte of the instruction. Another powerful capability of the National IMP-16C

TABLE 19

EXAMPLES OF THE LOGIC OPERATIONS, COMPLEMENT, SHIFT, AND ROTATE INSTRUCTIONS

Intel 4004	
RAR	;ROTATE THE ACCUMULATOR RIGHT ;THROUGH THE CARRY BIT.
Motorola 6800	
RORA	;ROTATE THE A ACCUMULATOR RIGHT ;THROUGH THE CARRY BIT.
EORA 400D	;EXCLUSIVE-OR THE A ACCUMULATOR ;WITH THE CONTENTS OF LOCATION ;400 DECIMAL AND PUT THE RESULT ;IN THE A ACCUMULATOR.
National IMP-16C	
SHR 16D	;SHIFT THE CONTENTS OF THE ;ACCUMULATOR RIGHT 16 BITS.
CAI 3D	;COMPLEMENT THE ACCUMULATOR ;AND ADD THE IMMEDIATE DATA ;3 DECIMAL TO THE ACCUMULATOR.

is incrementing or decrementing a memory location and skipping the next instruction if the contents of the memory location become zero. This feature allows memory locations to be used as counters for control purposes. The Intel 8080 also permits a memory location to be incremented or decremented, but a conditional instruction must follow the increment/decrement to test the location for zero. Table 20 gives examples of the incrementing and decrementing capabilities of microprocessors.

III. DECISION AND CONTROL

Decision and control instructions direct the microprocessor away from the standard sequencing of instructions by changing the value of the program counter. Such instructions, JUMP, CALL, or BRANCH, change the PC either conditionally or unconditionally. The difference between a JUMP or BRANCH and a CALL has to do with whether or not the PC is saved. JUMP and BRANCH instructions cause the PC to change values but never save the PC and CALL instructions store the PC. Once the called program has been executed (a RETURN instruction is processed) the program counter is restored to point at the instruction following the last CALL. Another difference between control instructions is that CALLS may be nested, i.e., a series of CALLS is performed transferring the program counter to different instruction blocks. By executing a series of RETURNS (the same number as CALLS), the PC will return to the original instruction block.

TABLE 20

EXAMPLES OF INCREMENT AND DECREMENT INSTRUCTIONS

Intel 4004

```
ISZ     10H, LOOP       ;THE CONTENT OF INDEX REGISTER 10H
                        ;IS INCREMENTED BY ONE.  IF RESULT = 0
                        ;NEXT INSTRUCTION IS EXECUTED.  IF
                        ;RESULT ≠ 0, PC IS LOADED WITH ADDRESS
                        ;FOR LOOP.
```

Intel 8080

```
INR     0400H           ;INCREMENT THE CONTENTS OF MEMORY
                        ;LOCATION 0400 HEXADECIMAL.

JZ      0100H           ;IF THE CONTENT OF 0400H IS ZERO,
                        ;JUMP TO LOCATION 0100 HEXADECIMAL.
```

National IMP-16C

```
AISZ    3D              ;ADD IMMEDIATE DATA 3 DECIMAL TO THE
                        ;ACCUMULATOR AND SKIP IF THE ACCUMULATOR
                        ;BECOMES ZERO.

DSZ     300D            ;DECREMENT THE CONTENTS OF MEMORY
                        ;LOCATION 300 DECIMAL AND SKIP THE
                        ;NEXT INSTRUCTION IF THE CONTENTS
                        ;BECOME ZERO.
```

Conditional Instructions

Using conditional CALLS and BRANCHES or JUMPS is vital to any programming algorithm. Table 21 describes the various conditional instructions obtainable from microprocessors. These control instructions allow the program to respond to inputs rather than to continuously respond in the same manner. Each program must perform a different function if:

1. A carry has been generated.
2. A zero has been computed.
3. An interrupt has been issued.

Therefore, conditional control instructions are essential.

All microprocessors do not permit branching on every condition which develops within a program, but this absence can be overcome by using extra program instructions. Commonly, a JUMP or CALL occurs if a condition is "TRUE," but if the JUMP or CALL does not exist for the condition being "FALSE," an extra unconditional JUMP or CALL overcomes the problem.

Subroutines

CALLS, conditional and unconditional, allow the programmer to use one copy of a sequence of instructions, a subroutine, and enter it from different routines. For example, a specific input routine can be called whenever this input is required in an instruction sequence. The last instruction of the subroutine, a RETURN, causes the program counter to return to the instruction immediately following the CALL.

TABLE 21

CONTROL INSTRUCTIONS WHICH CHANGE THE ORDER IN WHICH THE PROGRAM IS EXECUTED

JUMP BRANCH SKIP CALL RETURN	conditional or unconditional

Intel 8080 JUMP

JUMP

Location	Instruction	
1	C3 = JUMP	;JUMP TO 3800
2	00	
3	38 = 3800	;PC LOADED WITH 3800

Intel 8080 CALL

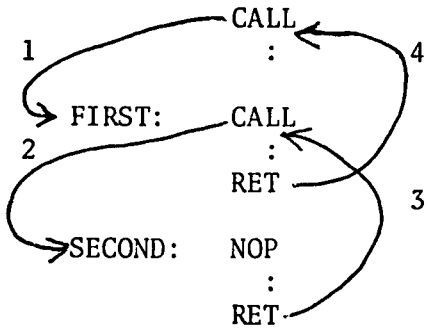
CALL

Location	Instruction	
1	CD = CALL	;CALL 0300
2	00	
3	03 = 0300	;PC IS SAVED SO THAT ;WHEN RETURN IS ;PERFORMED, PROGRAM ;RETURNS TO LOCATION 4.

Nesting of the CALLS allows subroutines to call on the other subroutines to gather information or perform operations (Table 22).

TABLE 22

NESTING OF SUBROUTINES




```
        FIRST       ;(1)THE PROGRAM CALLS THE
                    ;SUBROUTINE FIRST
                    ;(2)SUBROUTINE FIRST CALLS
        SECOND      ;SUBROUTINE SECOND
                    ;(3)AFTER SECOND HAS BEEN
                    ;COMPLETED PROGRAM RETURNS
                    ;TO THE INSTRUCTION
                    ;IMMEDIATELY FOLLOWING THE
                    ;CALL IN SUBROUTINE FIRST
                    ;(4)AFTER FIRST HAS BEEN
                    ;COMPLETED PROGRAM RETURNS
                    ;TO MAIN ROUTINE
                    ;(PC POINTS TO INSTRUCTION
                    ;FOLLOWING CALL FIRST)
```

A sixteen-bit register called a stack pointer is used by some microprocessors. The stack pointer is automatically decremented by a RETURN instruction. Once a CALL has been fetched the current program counter is stored in a RAM location, and the RETURN instruction loads the PC from this RAM location. The address of the RAM location is given by the contents of the stack pointer register. Nesting of subroutines is accomplished by storing the current PC and decrementing the stack pointer each time a CALL instruction is encountered, and as the RETURN instruction is performed, the stack pointer is incremented and the PC reloaded from the address provided by the stack pointer

register. The Intel 8080 and Motorola 6800 processors use memory locations as the stack register, allowing arbitrary depth of nesting. Limitation of these stacks is only because memory locations are being used. The National IMP-16C has a sixteen level stack.

Each microprocessor differs in some way in the set of conditions for JUMPS, BRANCHES, or CALLS. The Motorola 6800 and Intel 8080 instruction sets are more powerful than other microprocessors on conditional control instructions. From Appendix B the large number of conditional BRANCHES within the Motorola instruction set is seen. Appendix A shows the instruction set of the Intel 8080 and included are the conditional CALLS and RETURNS, which are not found in any other microprocessor system. The CALL uses a fourteen bit immediate address, thus a three byte instruction. Intel does have a single byte unconditional CALL however, which stores the current PC and replaces the PC with all zeroes except for bit positions three through five. Eight subroutines can be located in the lower order 64 locations of memory using this call. If needed, a JUMP to a higher memory location may be performed at one of these called locations allowing a much longer program to be processed. The Intel 8080 accepts and executes such an instruction upon the receipt of an interrupt signal, allowing external devices to direct program control.

Four-Bit Control Instructions

Considering control features of the four bit machines, the Rockwell PPS microprocessor allows unconditional jumps to one of 64

locations specified by six data bits in a twelve-bit JUMP instruction. "Long" jumps are accomplished by an immediate twelve-bit address in two successive words of the instruction. The Fairchild PPS-25 microprocessor has a flexible structure for control techniques. Unconditional jumps are made only within the same ROM and are specified by immediate data in the JUMP instruction. The immediate data is interpreted as a signed 2's-complement number which is added to the present ROM location, allowing both forward and backward jumping within the same ROM. A separate instruction allows the JUMP to go to a new ROM. CALLS within the Fairchild PPS are performed by storing the PC plus one in one of two fields of a status register, then unconditionally or conditionally jumping. A RETURN is performed by reloading the PC with the stored status register, after incrementing by one. By incrementing the PC twice the JUMP instruction which followed the original storage instruction is skipped (13).

Unique Control Features

Other control features which are important for a programmer to be familiar with are found within the National IMP-16C and the Intel 8080. The IMP-16C generates flexibility within the address used for unconditional JUMPS or CALLS. The PC is loaded with an address computed from eight bits of the immediate field added to the sixteen-bit contents of the index register. If an indirect mode is selected, this computed address is used as the memory location whose contents become the PC. Within the 8080 there is an instruction which

transfers the contents of two working registers into the PC, causing an unconditional JUMP. This same feature is provided by the IMP-16C with the jump address being located in the index register (Table 23), or the address can be based on the contents of one of the lower RAM locations by using the indirect memory reference.

TABLE 23

UNCONDITIONAL JUMP CREATED BY TRANSFERRING REGISTERS

Intel 8080		
PCHL		;JUMP INDIRECT TO LOCATION ;SPECIFIED BY H AND L REGISTERS
National IMP-16C		
JMPP	40D	;JUMP INDIRECT TO THE MEMORY ;LOCATION SPECIFIED BY THE SUM ;OF THE CONTENTS OF LOCATION ;100H AND 40DECIMAL

IV. INPUT/OUTPUT

Microcomputer interfaces and input/output instructions vary considerably. The scheme within the Intel 8080 and the National IMP-16C is to provide bits on the address bus for both input and output instructions. A particular I/O device is selected by an appropriate INPUT or OUTPUT pulse, then the microprocessor either outputs the accumulator contents or inputs the data into the accumulator. In the

Intel 8080, the address bus bits are within the current instruction word, and in the IMP-16C, the address bus bits are formed by adding the immediate data of the instruction to the internal working register.

The Motorola 6800 processor has one of the most sophisticated I/O developments available in microcomputers (Chapter II). A special Peripheral Interface Adapter (PIA) provides means for interfacing equipment to the 6800 system. Two eight-bit bidirectional buses and four control lines are programmed to respond as either input or output devices and various control modes, respectively. Greater flexibility, due to the programming required, is given with the PIA than has been given with either the Intel 8080 or the National IMP-16C.

I/O devices of four-bit machines are more special purpose than the eight- and sixteen-bit processors. The Rockwell PPS system uses immediate data for device selection, but provides a bidirectional data exchange in the same cycle, i.e., four-bits in the accumulator go out on four-bits of the bus. I/O ports of the Intel 4004 are associated with ROM and RAM, selected by separated instructions, devices of the MCS-4 system. Both reading and writing of the I/O ROM and the latched RAM output is accomplished by an eight-bit instruction. A set of I/O instructions which control special I/O devices of the Fairchild system are used by the Fairchild PPS-25. Also, a direct eight-bit serial input to a ROM register which is added to the ROM register is available with the PPS-25. A special PPS-25 I/O instruction loads serial data into the active status register, allowing each bit to be interrogated as an individual flag (13).

V. MICROCOMPUTER CODING

Once programming techniques have been learned the designer must convert an algorithm of instructions into machine language which can be loaded directly into microcomputer memory. Microcomputer coding, developing the algorithm into a machine language program, has the tools: (1) assemblers, (2) editors, (3) loaders, (4) compilers, (5) microprogramming. An editor program may be used to prepare, modify, and store the microcomputer program. Assemblers produce an object code which is either absolute or relocatable. Absolute programs must reside in memory locations assigned before assembly, and relocatable programs may reside anywhere in RAM memory. Both absolute and relocatable programs are placed in memory by loaders. Compilers and microprogramming techniques are developed for more advanced types of programming. Figure 22 shows the function of assemblers, editors, loaders, and compilers (14).

Assemblers

Assembly language is the most common language used for microcomputer programming. Assembly language programming has the features; symbolic codes (mnemonics), labels which refer to memory locations, and symbolic names for operands (registers, flip-flops, or test conditions). For example, in the Intel 8080 the instruction

$$(A) \longleftarrow (A) + (r)$$

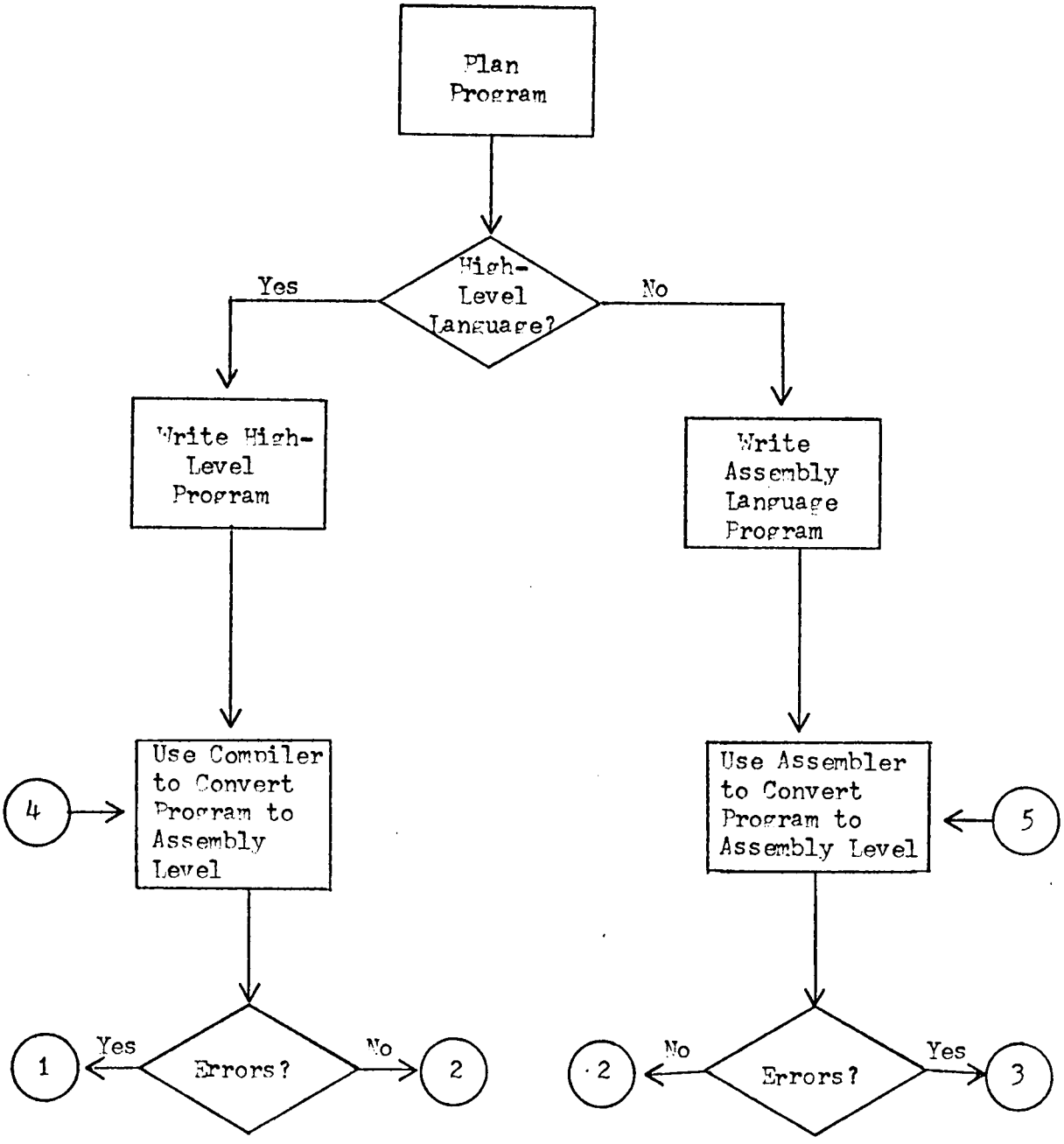


Figure 22. Preparation of a Program Using Assemblers, Compilers, and Loaders

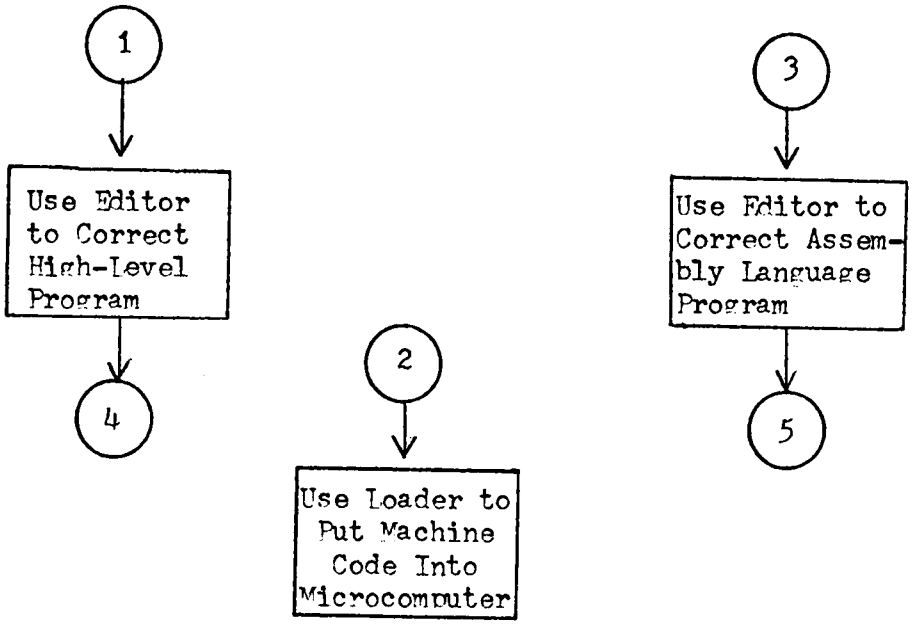


Figure 22 (continued)

replaces the contents of the accumulator with the sum of the contents of the accumulator and the contents of register r. The Intel assembly language equivalent is

```
ADD     B
```

where B represents the register r. This register may be the accumulator itself, one of the working registers, or a memory location.

The assembler reads a source tape with statements written in symbolic assembly language (Table 24) and constructs the program tables. Then the assembler produces an object tape with binary or hexadecimal format for the microcomputer (Table 25).

An assembler is a program that must be loaded and run on a computer. The Intel 8080 microcomputer set has a resident assembler, i.e., the assembler program is loaded into the 8080 microcomputer RAM for assembling programs written for the 8080 microcomputer. This resident assembler requires three readings of the program source tape. The first pass sets up the symbol tables for the program, the second pass outputs the assembled listing of the program, and the third pass outputs the object tape for the program. Another type of assembler, a cross-assembler, is one which is programmed in Fortran and run on a computer which has the ability to compile a Fortran program. Therefore, the source tape is assembled, to obtain the object tape, on some computer other than the microcomputer. This allows for storing the

TABLE 24

AN ASSEMBLY LANGUAGE PROGRAM ILLUSTRATING THE BASIC FEATURES OF ASSEMBLERS, AND THE PROGRAM WRITTEN USING THE INTEL 8080 ASSEMBLY LANGUAGE

Address Label	Symbolic Op-Code	Operand Name	Comments
	ORIGIN =	600	;PSEUDO CODE DEFINING
	•		;INITIAL PROGRAM ADDRESS
	•		
	DECREMENT	B REGISTER	
	JUMP ZERO	OVER	;IF RESULT = 0 JUMP TO
	•		;INSTRUCTION LABELED OVER
OVER:	ADD	10H	;ADD 10 HEXADECIMAL TO
	•		;ACCUMULATOR
	END		;PSEUDO CODE
	ORG	600	
	•		
	DCR	B	
	JZ	OVER	
	•		
OVER:	ADD	10H	
	•		
	END		

TABLE 25

AN EXAMPLE OF THE INTEL 8080 ASSEMBLER CONVERTING A SOURCE TAPE, TO A LIST TAPE, TO AN ABSOLUTE LOADER TAPE

Lines of a Source Tape

```
ADD     B
JUMP    300H
```

Lines of a List Tape

```
600     80          ADD  B
601     C30003      JUMP 300H
```

location of instruction | hexadecimal representation of machine word | assembly language statement written by programmer

Lines of an Object Tape

```
1000  0000   ADD statement
1100  0011
0000  0000   JUMP 300H statement
0000  0011      (3 bytes)
```

assembler on peripherals not interfaced to the microcomputer; tape units or disk units.

Pseudo-operations that do not assemble into microcomputer instructions but control the assembly of instructions which are assembled are provided by assemblers. A few of the more commonly used pseudo-instructions are: number system, origin, comment, equal, and data statements. Number system instructions are defined by placing a letter H, B, O, or D after each number in the program. An H defines a hexadecimal number, a B binary numbers, an O octal numbers, and a D decimal numbers. Origin statements cause the following instruction to be located at the operand location specified. For example, ORIGIN 100H locates the next instruction at location 100 hexadecimal. Consecutive locations of the program being assembled are used until another origin statement appears. Comments are used to explain the source file containing assembly language with the English text. A symbol such as ":," ";," ",," or "/" and everything to the right of the symbol is ignored by the assembler, however, the program listing produces all symbols and the comments. Equal statements are used to make labels equal to specific data. A statement such as DATA = 10B causes the contents of DATA to be 10 binary. Data statements create successive storage locations in memory, with the first location being labeled (14).

A macro facility is very useful when certain sections of the program are duplicates. The macro consists of a sequence of codes which are defined by data values, addresses, labels, or instructions.

A single statement is all that is required for the assembler to produce an expansion of the macro. At the location the macro is to begin a list of the values to be assigned appears. The assembler then completes the expansion of the macro as requested. An example of a macro using Intel 8080 instructions is given in Table 26 (15).

An important feature of assemblers is to detect and point to a variety of errors in source statements. Table 27 is an example of some typical errors found by assembling a program on the Intel 8080 microcomputer. Appendix D is a complete list of all assembler errors detectable with the 8080 assembler. These errors do not include logic errors or subtle program errors but do include duplicate address labels, undefined labels, undefined mnemonics, undefined operand field names, wrong number of operands, and invalid numbers in the number system chosen. Although all errors are not detectable, the errors that are detected are of great help to the programmer.

Editors

Editors are interactive systems or programs that allow designers to prepare a program and to make changes or correct errors with simple commands. Programs can be prepared and assembled. Then any errors, additions, deletions, or documentation can be added using the editor. The edited source program can then be output to paper tape creating a new source file.

With the Intel 8080 microcomputer an editor program is read into RAM memory. From this program a source tape can be read or created

TABLE 26

AN EXAMPLE OF A MACRO USING THE INTEL 8080 INSTRUCTION SET

Label	Op-Code	Operand	Comments
MAC1	MACRO	P1, P2	
	ADC	P1	
	INR	P2	
	SUB	P2	
	ENDM		

defines the MACRO

The reference:

```
MAC1   B, E
```

is assembled by the assembler as

```
ADC    B
INR    E
SUB    E
```

within the program.

TABLE 27

EXAMPLES OF THE ERROR DETECTION CAPABILITIES
OF THE INTEL 8080 ASSEMBLER

```
    0000                ORG     100H
U   0100  3E00          MVI     A, F7H      ;1.  UNDEFINED IDENTIFIER
E   0102  4C444942      LDI     B, 300H     ;2.  EXPRESSION ERROR
E   0106  0003
    0108  OA            LDAX    B
U   0109  CA0000        JZ      NEXT        ;3.  UNDEFINED IDENTIFIER
    010C  210006        LXI     H, 600H
R   010F  OA            LDAX    H           ;4.  REGISTER ERROR
    0110  86            ADD     M
B   0111  OE37          MVI     C, '7       ;5.  BALANCE ERROR
R   0113  C5            PUSH    A           ;6.  REGISTER ERROR
    0000                END
```

Appendix D contains a description of each error shown in this table.

and all changes made by using the instructions of Appendix E. The editor program stores the input data in a memory buffer and uses a special register, the buffer pointer. This register is used to point to the location of the workspace upon which operations are to be performed. Single characters are then used to perform the operations desired by the programmer.

Loaders

An elementary loader reads successive words on an object tape and writes them into successive RAM memory locations. The loader starts at a fixed origin, defined by the ORIGIN statement. A relocating loader, if available, uses a special object tape and the desired origin data to adjust the program addresses and load the resulting instruction. Linking is another capability of more advanced loaders. Program segments with undefined labels or names are loaded with the loader supplying the missing cross references between the separate routines. For most microcomputers only a basic loader exists and reassembling must be performed to change origins or define undefined labels.

Looking at the Intel 8080 again, the loader is a basic loader in that the 8080 will only load tapes in successive locations. The difference with the 8080 is that hexadecimal data is output to the object tape, with a checksum to check for reader errors, as shown in Figure 23.

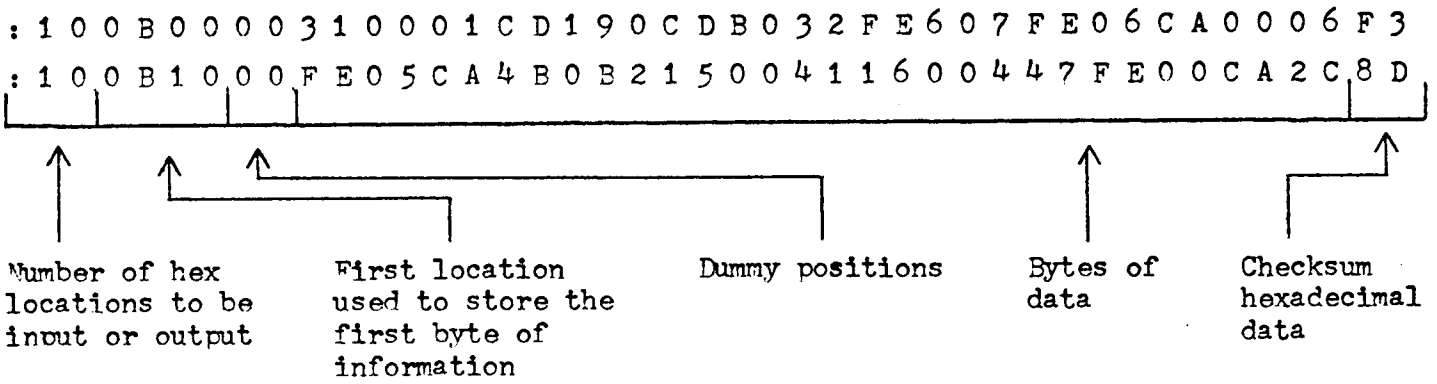


Figure 23. Typical Loader Data for the Intel 8080 Microcomputer

Compilers

A compiler is a program which accepts as input data another program, written in a higher-level language, and outputs another program written in either assembly or machine language. Compilers make a program easier to write because they provide a language which requires less instruction statements. Compilers also eliminate the need to write detailed codes to control loops, access data structures, and program arithmetic computations (16).

The only available high-level language for microcomputers is PL/M, a language developed for the Intel machines. Using PL/M may reduce errors because of the decreased detail needed to describe a program, but failure to understand conventions within the compiler cause new problems. Relying too heavily on the compiler can reduce the capabilities and efficiency of the microcomputer. By comparing the compiler source program listing and the assembly listing any loss of time, capability, and efficiency can readily be found. An example of a PL/M program for the Intel 8080 microcomputer is shown in Table 28 (17).

Microprogramming

The National IMP-16C can be tailored to design requirements by using a mask-programmed control ROM (CROM), and letting the designer develop the basic machine language instruction set by writing a microprogram. A microprogram is a particular sequence of control words necessary to produce a particular program. Described in

TABLE 28

A COMPARISON OF A COMPILER PROGRAM (PL/M) AND AN ASSEMBLY LANGUAGE PROGRAM, BOTH WRITTEN FROM THE INTEL 8080 LANGUAGES

PL/M Program

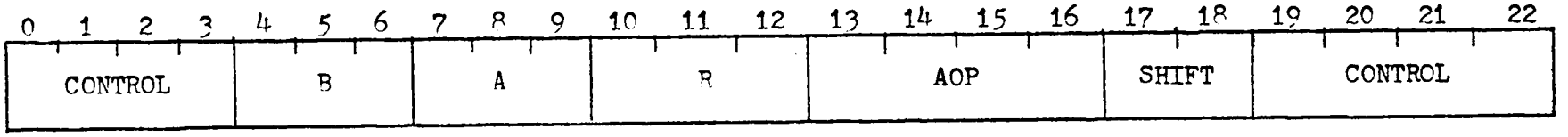
```
DECLARE  (X, Y, Z) BYTE;
X = 2;
Y = 6;
DO I = 0 TO 10;
     IF X > Y THEN
     Z = X + Y - 10
     X = X + 1;
     ELSE  Z = X - Y + 10
     X = X + 1;

END
```

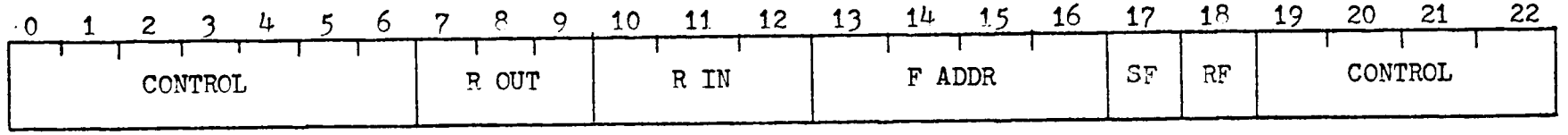
and an Equivalent Assembly Program

```
          LXI     H, 500H          ;SET LOCATIONS FOR ANSWER
          MVI     B, 2H            ;B REG. CONTAINS X
          MVI     C, 6H            ;C REG. CONTAINS Y
          MVI     E, 10D           ;E REG. IS COUNT
START:    MOV     A, B             ;PUT X IN ACCUMULATOR
          CMP     C                ;IS X > Y
          JP      THEN
ELSE:     SUB     C                ;X - Y
          ADI     10D              ;X - Y + 10
STORE:    MOV     M, A             ;STORE RESULT
          INR     B                ;INCREMENT X
          INX     H                ;INCREMENT STORAGE LOCATION
          DCR     E                ;DECREMENT COUNT
          JZ      FINISH           ;IF ZERO PROGRAM FINISHED
          JMP     START            ;PERFORM ANOTHER OPERATION
THEN:     ADC     C                ;X + Y
          SUI     10D              ;X + Y - 10
          JMP     STORE            ;STORE RESULT
```

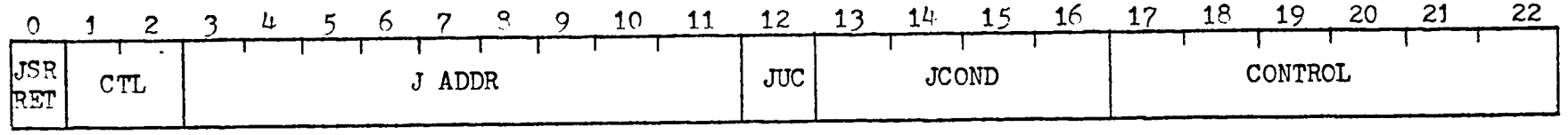
Chapter II were the architectural features developed to create the microprogramming feature of the IMP-16C (RALU, CROM). Definition must now be made of the microprogram word format which controls the RALU. The CROM bit functions have been divided into three classes (Figure 24). Control fields are defined to develop the storage control required. The arithmetic class of fields allows specification of the registers to be loaded on the A- or B-bus of the RALU, the ALU operation to be performed (AOP), a shift operation, and the register to be loaded from the R-bus. I/O class fields provide for addressing a register for data output and a register for data input and setting (SF) and/or resetting (RF) one of sixteen external control flags to indicate the type of I/O transfer. The microinstruction JUMP class provides a nine-bit address field, selection of up to sixteen jump conditions (JCOND), an unconditional jump command (JUC) and specification of a subroutine jump or return (JSR/RET). Each of these fields provide the programmer with the capability of performing nearly any operation needed for a certain application.



Arithmetic Instructions



I/O Instructions



JUMP Instructions

Figure 24. IMP-16C Microinstruction Word Formats

CHAPTER IV

APPLICATIONS

Before the introduction of microcomputers, logic systems used discrete and random logic to perform desired applications. A new direction, in the design of logic systems, was launched with the emergence of the microcomputer. These microcomputers are being applied in computational and control applications. The selection of a microcomputer based system depends heavily on the application. Table 29 presents a list of applications which are dependent upon the word-length of the microcomputer.

In this chapter three types of microcomputer applications are discussed. Applications using the microcomputer as a desk calculator, as a traffic controller, and as an analog-to-digital converter (ADC) are developed. Both the desk calculator and traffic controller have been designed and developed, whereas the ADC application has only been designed. Each application is developed using the Intel 8080 microcomputer system as the controlling device.

I. DESK CALCULATOR APPLICATION

An add and subtract calculator machine was developed using the Intel 8080 microcomputer. Before designing the system some considerations were given to the following questions:

TABLE 29

APPLICATIONS OF MICROCOMPUTERS

Applications		
Four-bit Processors	Eight-bit Processors	Sixteen-bit Processors
1. BCD display control or calculation	1. Intelligent terminals	1. Data acquisition systems
2. Electronic cash registers	2. On-board computers (automobile)	2. Process monitoring
3. Business and accounting machines	3. Process and machine control	3. Navigational systems
4. Credit card verification	4. Traffic control	4. Automatic test systems
5. Appliances	5. Education	5. Peripheral control
6. Game machines	6. Medical electronics	
	7. Measurement systems	

1. Should software or hardware be used to control the multiplexed LED idsplay?
2. What type of notation should be used for entering, adding, and subtracting data?

The final decisions made were to use hardware to control the display and to use algebraic notations.

The decision for controlling the display with hardware was based upon the fact that by developing a hardware controller the following would be accomplished:

1. The computer would not be tied up with continually updating the display information.
2. Complexity of the software would be decreased.
3. Amount of memory required for the program storage would be decreased.

Figures 25 and 26 show the hardware approach taken. The circuit consists of a clock, seven four-bit data latches, four 8-to-1 multiplexers, a BCD-to-seven segment decoder, and drivers for the LED's.

A 74123 monostable, in which the two one-shots are tied together to perform as an astable, is used as the clock. Both the anode drivers and the multiplexers are clocked at a 140 kilohertz rate such that the correct LED is "ON" when the corresponding data is selected. With this 140 kilohertz rate, no flickering of the LED display occurs.

Data which is output for display purposes is latched into the four-bit latches. These latches are strobed by the 8080 such that information is shifted from the LSB position to the MSB position.

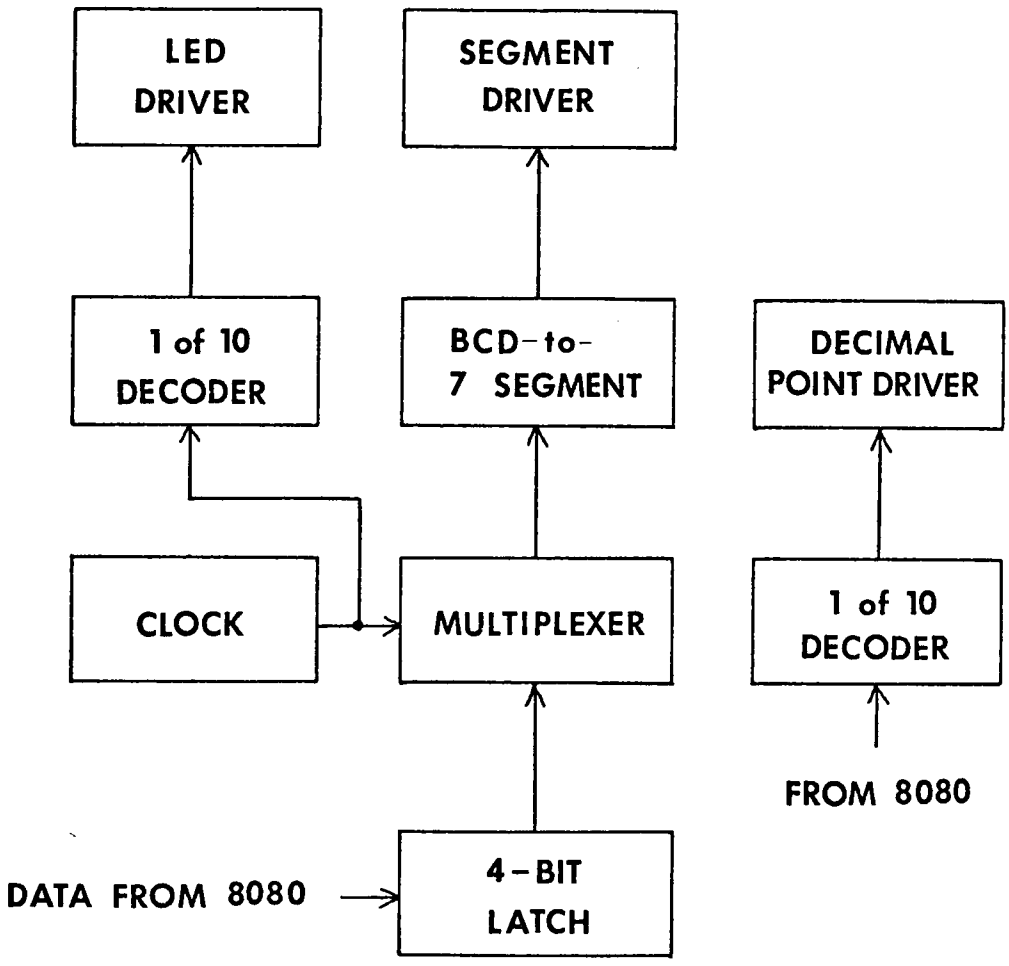


Figure 25. Block Diagram of Desk Calculator Application

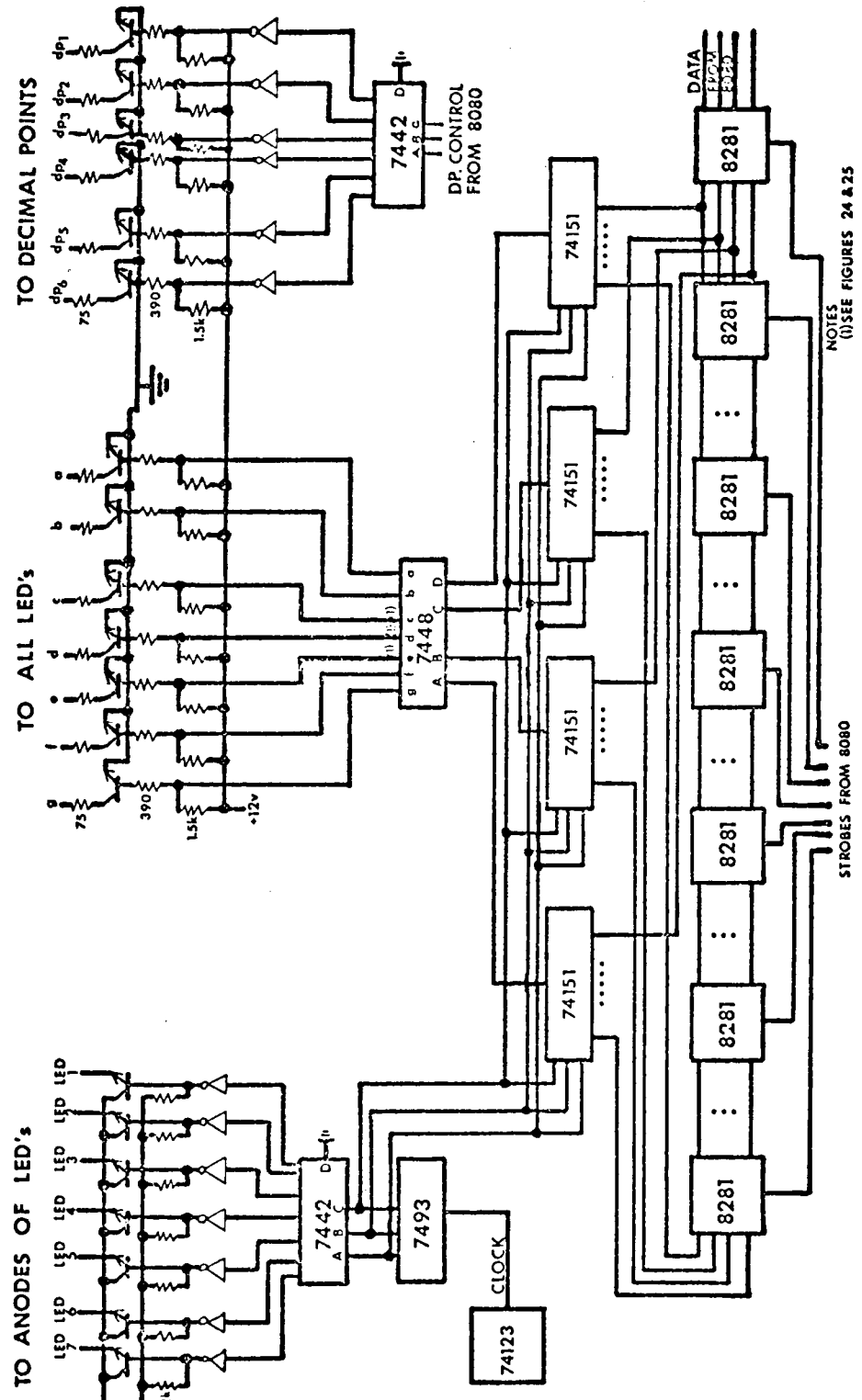


Figure 26. Schematic of Desk Calculator

Because only one BCD-to-seven segment decoder (7448) is used to drive all seven displays, multiplexing is required. 74151 multiplexers are used to select one of the seven inputs provided. During each clock period, one of the inputs is selected along with the corresponding LED driver. Four multiplexers are used to provide the BCD number to the 7448. As each BCD number is input, the corresponding seven segment information is clocked to the appropriate LED.

MPS 6531 transistors are used as the drivers for the LED segments and decimal points. When a particular line is selected, the transistor saturates allowing twenty milliamps to be drawn through the segment or decimal point. Selection of the correct decimal point is made using a one out of ten decoder (7442). No decimal point is output until the BCD code of one is input to the 7442. The BCD code is then incremented each time the decimal point is to move. Drivers for the LED anodes differ from the LED segment drivers only in that rather than sinking current, five volts is output through the transistor to the selected line.

In order to display negative numbers or overflow errors, hardware modifications were made to the output of the 7448 decoder (Figures 27 and 28). To select the minus indicator, an eleven (decimal) is output to the display. From Figure 27, both the c and d segments of the LED are turned "OFF" leaving merely the g segment or a -. To select the error indication, a thirteen (decimal) is output to the display. From Figure 28, segment e is turned "ON" giving an E on the display.

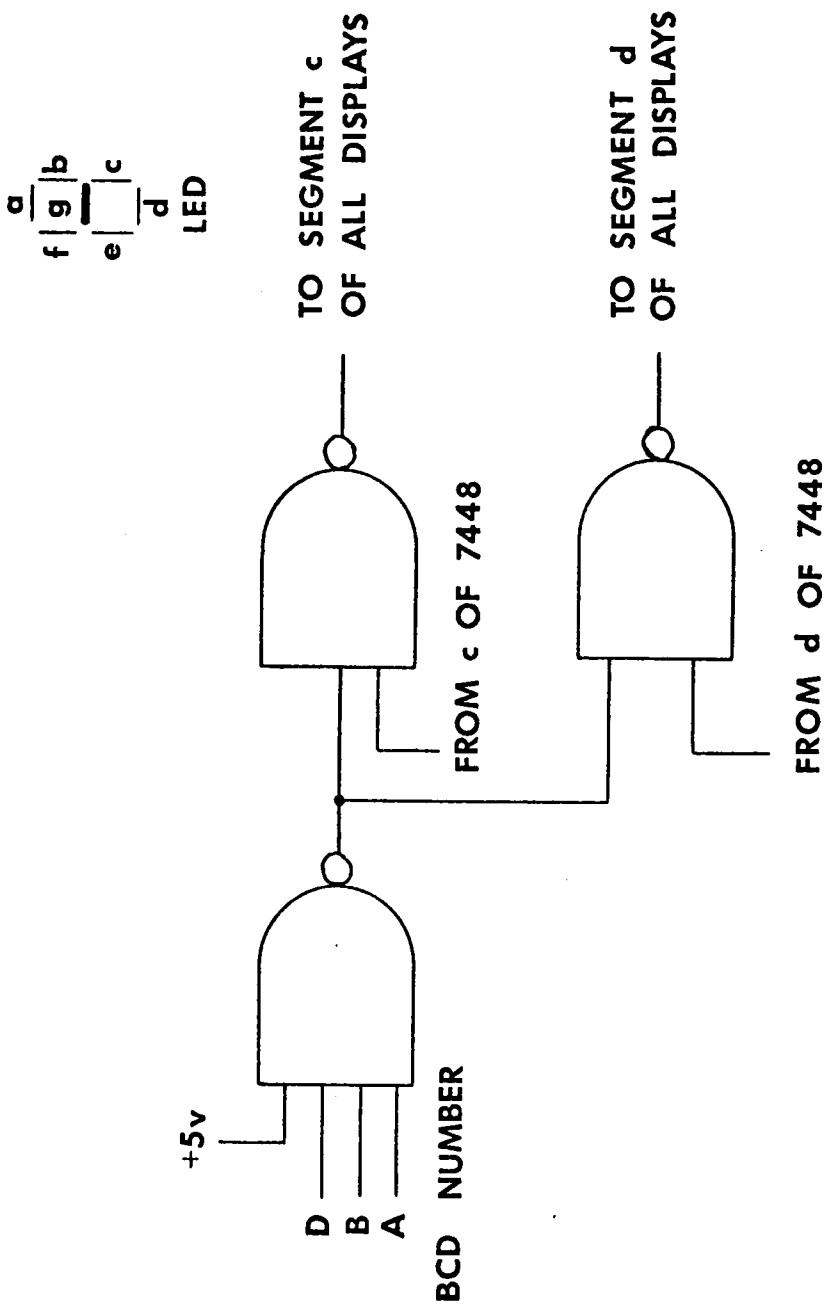


Figure 27. Minus (-) Sign Schematic

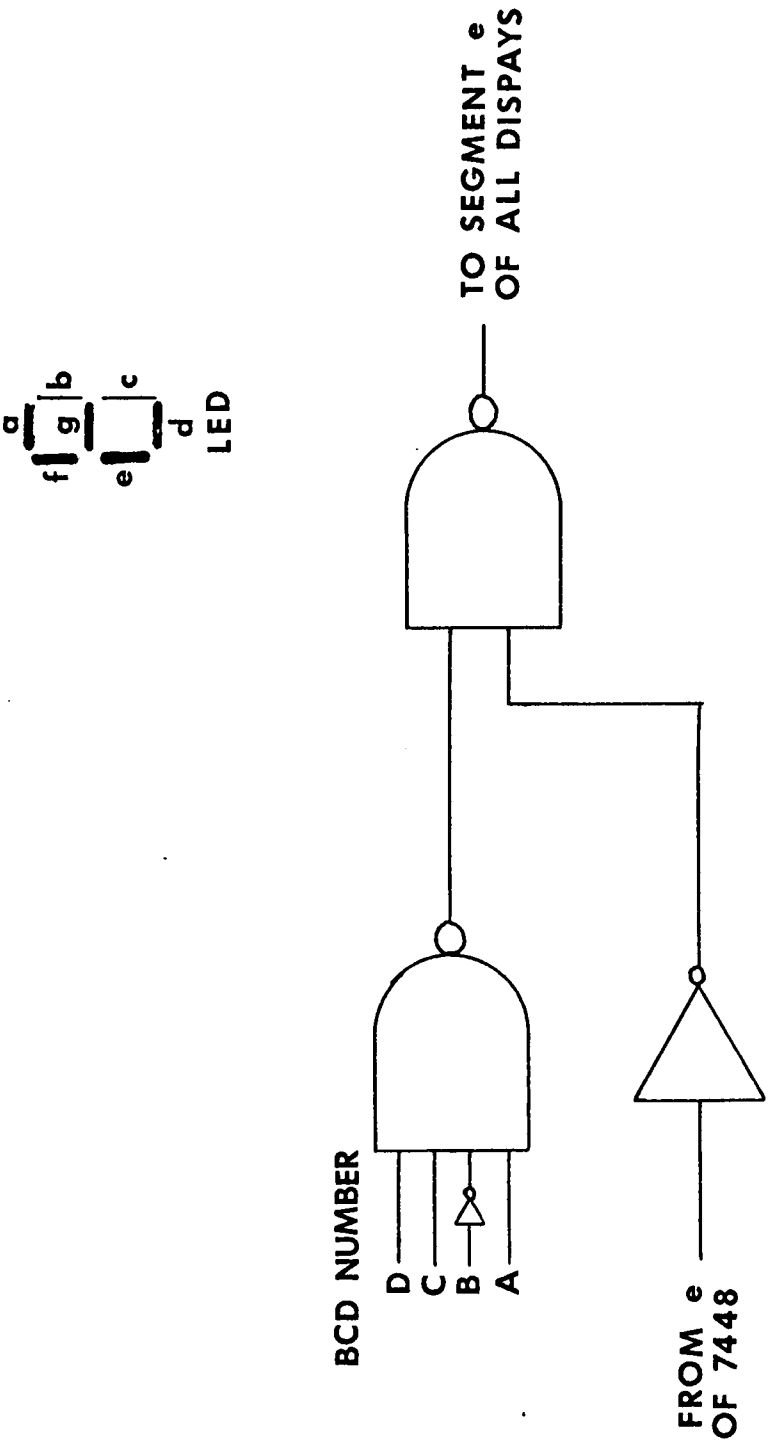


Figure 28. Error (E) Indicator Schematic

The basic idea used in the software development was that all numbers input and all answers would be displayed on both the LED's and a teletype. General requirements included using an ASR-33 teletype as the keyboard input device and outputting all numbers to the LED display unit. Addition and subtraction of at least seven numbers was required. However, the program developed is capable of operating on ten integer numbers and ten decimal numbers. In other words, an operation such as

```
 1042769132.7913100042
+8876567421.0002101399
```

can be performed. Also, the program cycles so that continuous operations can be performed.

An overall flowchart is shown in Figure 29 and the program is shown in Appendix F. As a number or character is input, appropriate action is taken. If the first number is being input, this number is stored in locations 500H to 513H. The second number is stored in locations 400H to 413H. A decimal point (.) sets the storage locations for the decimal numbers. Plus (+) and minus (-) inputs set appropriate flags for the corresponding numbers. An S input changes the sign of the number, a "rubout" clears the last entry, a C clears everything and restarts the program cycle, and an "=" causes some operation to be performed. Any other character which is entered will display an error indication.

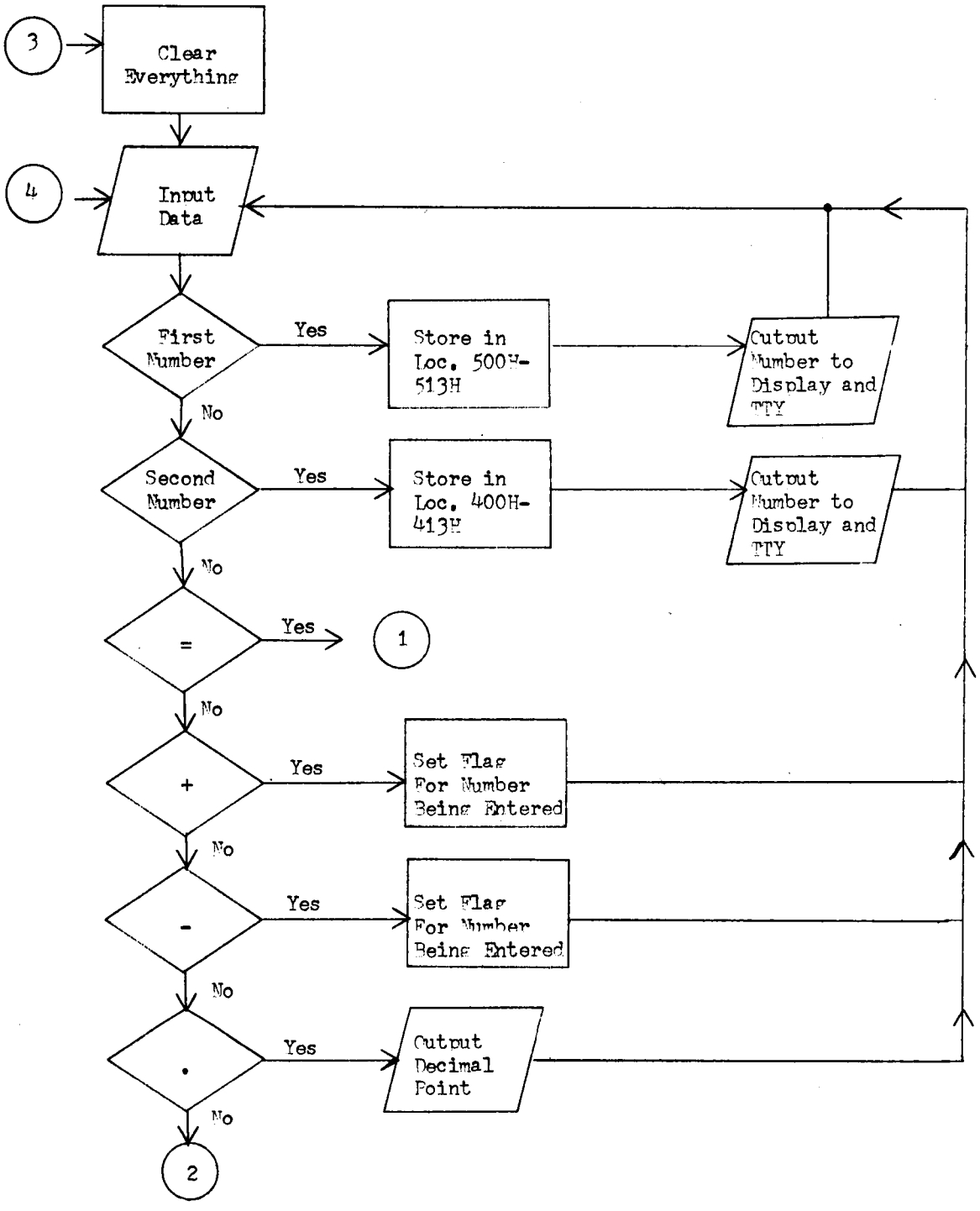


Figure 29. Flowchart for Desk Calculator Applications

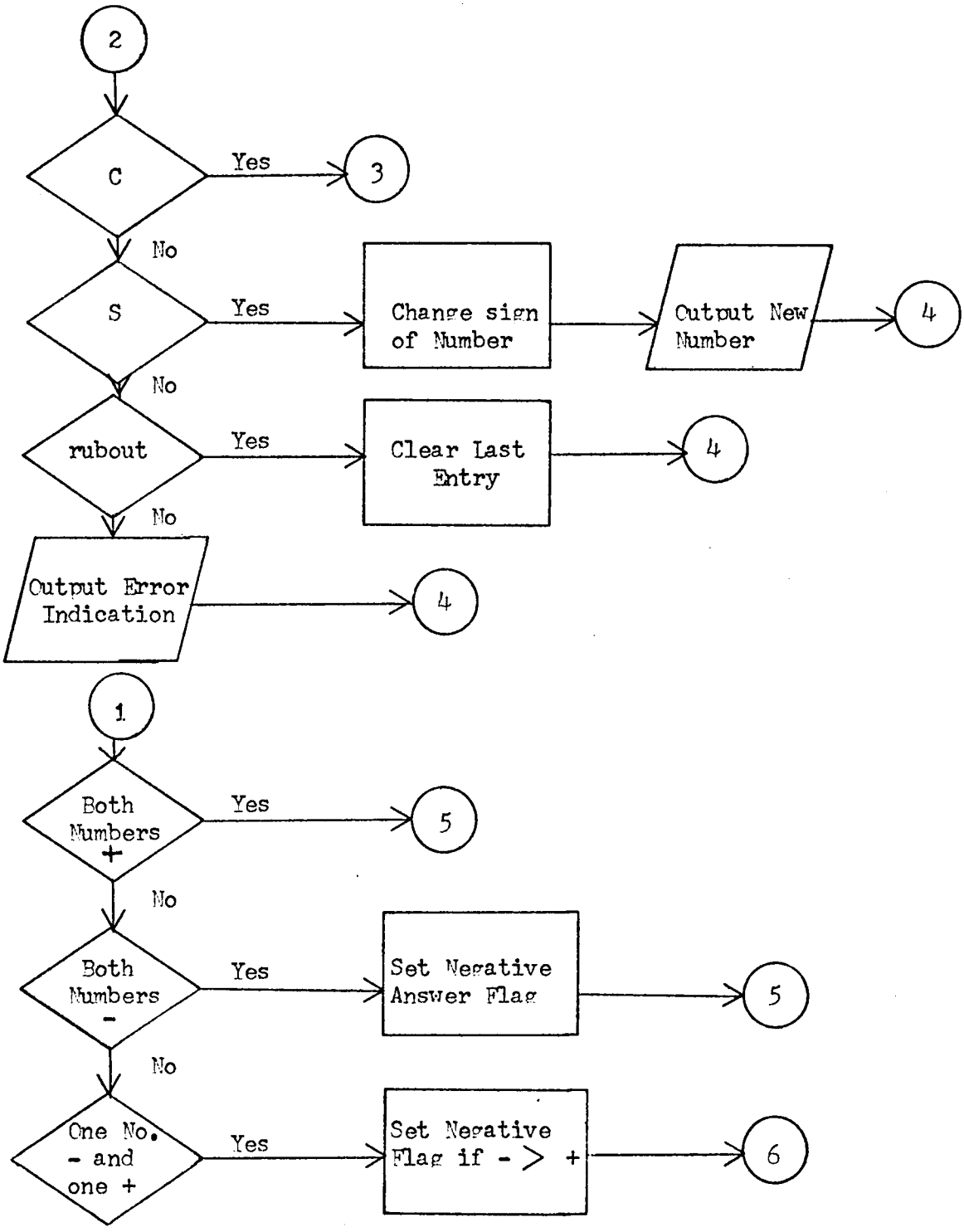
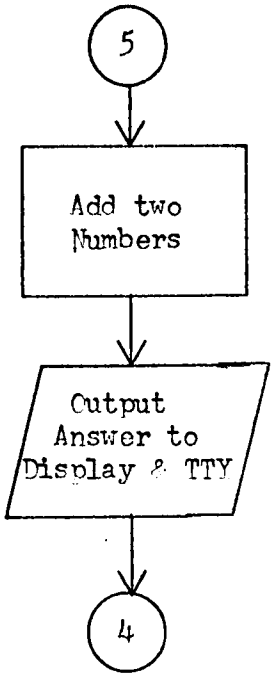


Figure 29 (continued)



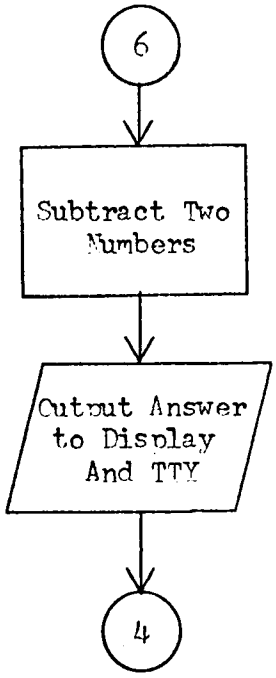


Figure 29 (continued)

As the number is being entered, the integer numbers are arranged from 509H to 500H, with 509H being the LSB. Decimal numbers are entered, after the decimal point has been input, from 509H to 513H. Each byte of the memory locations contains only one of the BCD numbers, i.e., entering 49.356 is stored as:

... 505	506	507	508	509	50A	50B	50C	50D	50E ...
00	00	00	04	09	03	05	06	00	00

Once an "=" is input the program checks the sign flags of the numbers. If both numbers are positive, an addition is performed. If both numbers are negative, an addition is performed, and a minus sign is output with the answer. However, if one number is positive and one number negative, the largest number is stored in locations 500H to 513H. After the larger number is stored, the program checks the sign flags of the numbers and sets the flag for a negative answer if the larger number is negative. Then, a subtraction of the two numbers is performed.

Addition and subtraction are accomplished by operating on bytes of data. Addition is performed by adding two bytes and performing the DAA instruction on the result. If the fourth bit is set, the carry bit is set, and another addition of bytes is performed. Subtraction is accomplished by using either the nine's or ten's complement, depending on whether or not a borrow is needed. Addition of the complement to the minuend and using the DAA instruction to adjust the

number finishes the subtraction operation. Table 30 is an example of both addition and subtraction operations. Results of these operations are always stored in locations 500H to 513H.

Output of the answer is accomplished by first, skipping all leading zeros and outputting the integer numbers MSB to LSB, respectively. Once location 509H has been output, a decimal point is output. All trailing zeros are then skipped, and the decimal numbers output.

After completion of an add or subtract cycle, the program waits for a new input. Addition or subtraction to the previous answer can be performed or an entirely new operation can be entered. Table 31 shows a typical example and the input and output observed on the teletype.

II. TRAFFIC CONTROLLER

Two basic types of traffic lights are implemented in this microcomputer application—fixed-time controllers and actuated controllers. Fixed-time controllers are the most common type of traffic controllers. These fixed-time controllers are ones which give the normal green-red-yellow sequence, flash red and yellow at night, or give advanced and delay green lights for left-turn traffic. Actuated controllers operate based on the traffic flow.

Figure 30 shows the design used to control the lights on each side of the traffic light. Two bits output from the Intel 8080 control each side. With the arrangement of the two bits shown in

TABLE 30

EXAMPLE OF ADDITION AND SUBTRACTION OPERATIONS PERFORMED BY THE DESK CALCULATOR PROGRAM

Addition

5	0101	
+6	0110	
11	1011	
	0110	DAA
	1 0001	

Since Bit 4 is set, the carry bit is set.

Subtraction

8	1000	
-3	+ 1	carry bit since no
5	1001	borrow is needed
	+1001	form ten's complement
	1 0010	
	+ 0011	add minuend (3)
	1 0101	answer is 5 with Bit 4 set to one which means no borrow was needed.

TABLE 31

TYPICAL OPERATION AND RESULTS AS OBSERVED
ON THE TELETYPE

```
23.4 + 432.16 = 455.56

+12.4 = 467.96

-167.06 = 300.9

17.2 - 187.3 = -170.1

S + 10 = 180.1

C

100.3 - .367 = 99.933
```

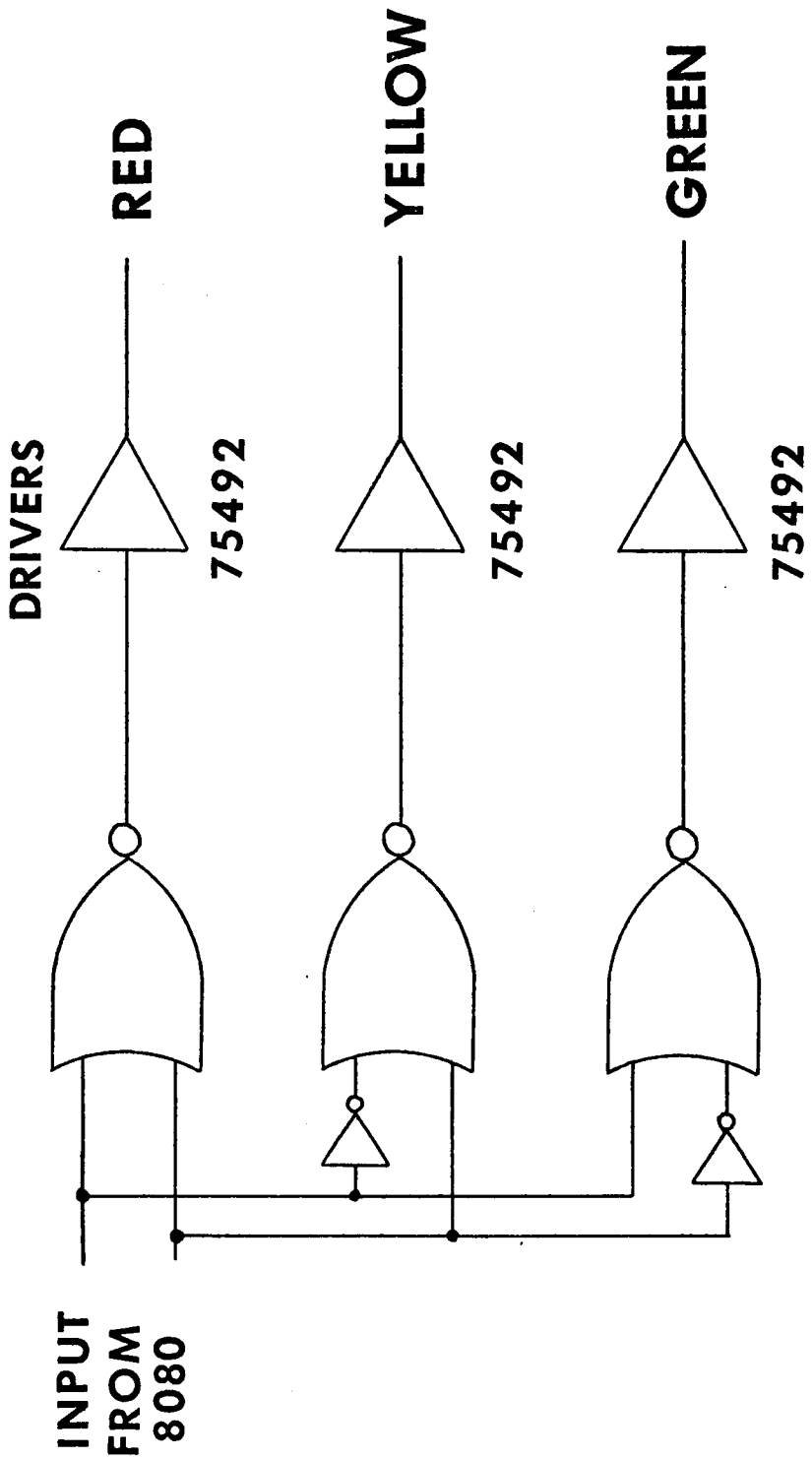


Figure 30. Circuit Developed to Control Each Side of the Traffic Lights

Table 32, a total of eight bits were used to control the entire light. The lights used are incandescent bulbs which are rated 200 milliamps at six volts. A 75492 driver, which sinks up to 250 milliamps, is used to drive each light.

TABLE 32

CONTROL OF EACH SIDE OF THE TRAFFIC LIGHT IS ACCOMPLISHED WITH TWO BITS

Bits	Output
11	OFF
10	YELLOW
00	RED
01	GREEN

In order to simulate an actuated controller some type of random car generator was required. Figure 31 shows the random sequence generator developed. The nine JK flip-flops tied together and clocked simultaneously creates the randomness. Two 74121 one-shots tied together to operate as an astable clock are used as the system clock. By tying a potentiometer to one of the one-shots the clock period was made variable. Setting the clock period to one second allows the random generator to repeat itself every 512 clock pulses. (Nine flip-flops clocked once a second yields 2^9 or 512 clock pulses.)

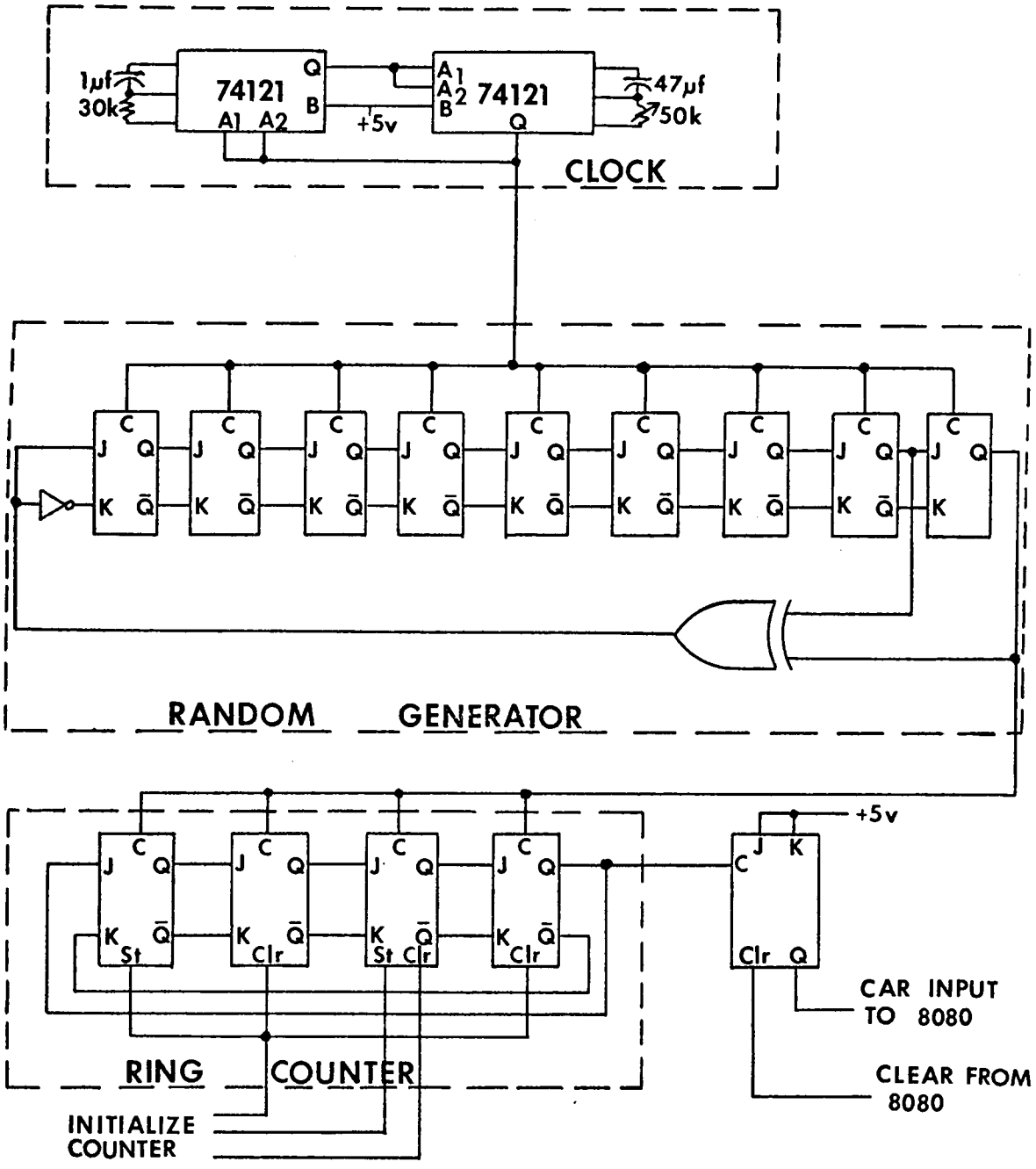


Figure 31. Random Signal (Car) Generator Schematic

The output of this random generator has the characteristic that over the 512 time periods, an equal number of "1's" and "0's" are output. In order to decrease the number of "1's" generated, the random generator pulses are used to clock a ring counter (Figure 31). With only one "1" in the ring counter, the interval between "cars" is increased. The output of the ring counter is then used to strobe a latch. This latch is sensed through an input port of the 8080. Also, the latch is cleared from the 8080, allowing each "car" to be counted only once.

In general, however, one of the streets of an intersection is more heavily travelled than the other. To account for this, the capability of presetting the ring counter is provided. The two possibilities are the following:

1. For the less travelled street, flip-flop 1 is set while the other three are cleared.
2. For the more travelled street, flip-flops 1 and 3 are set while the other two are cleared.

By setting two flip-flops, twice as many "cars" are generated. With the addition of more flip-flops to the ring counter, this setting and clearing of flip-flops can be expanded to obtain the desired response.

Figure 32 is a flowchart of the program developed and Appendix G is the program. Sixteen possible states are included in the fixed time sequence. Time sequences are developed using a one second subroutine. Each state consists of a bit pattern that specifies the light configurations during that state. Also, the state consists

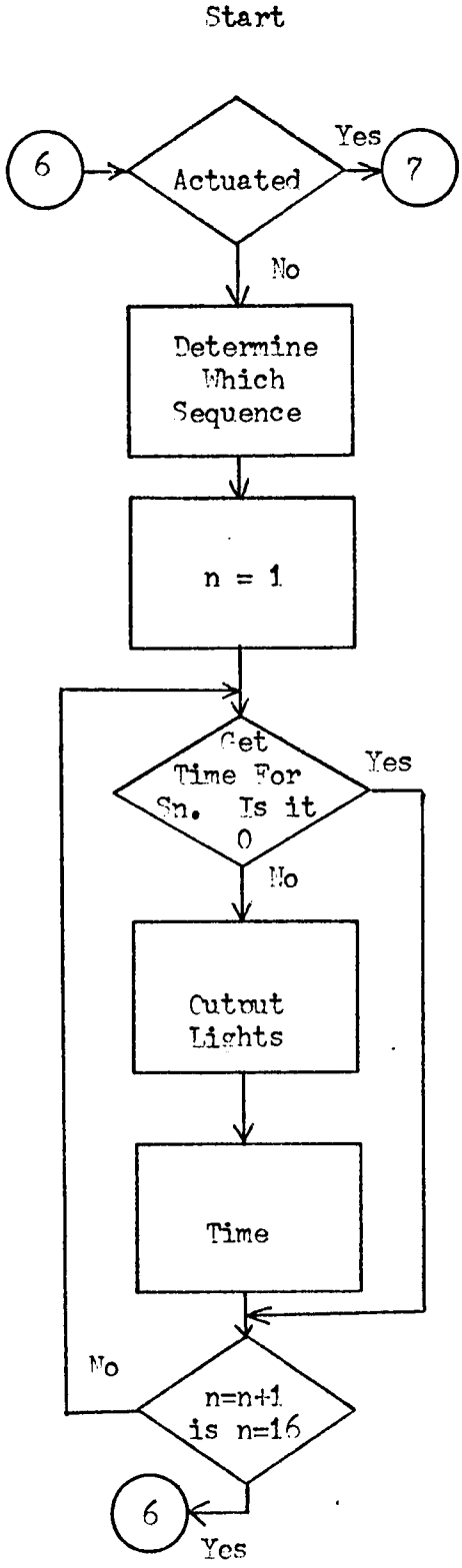


Figure 32. Flowchart of Traffic Controller Application

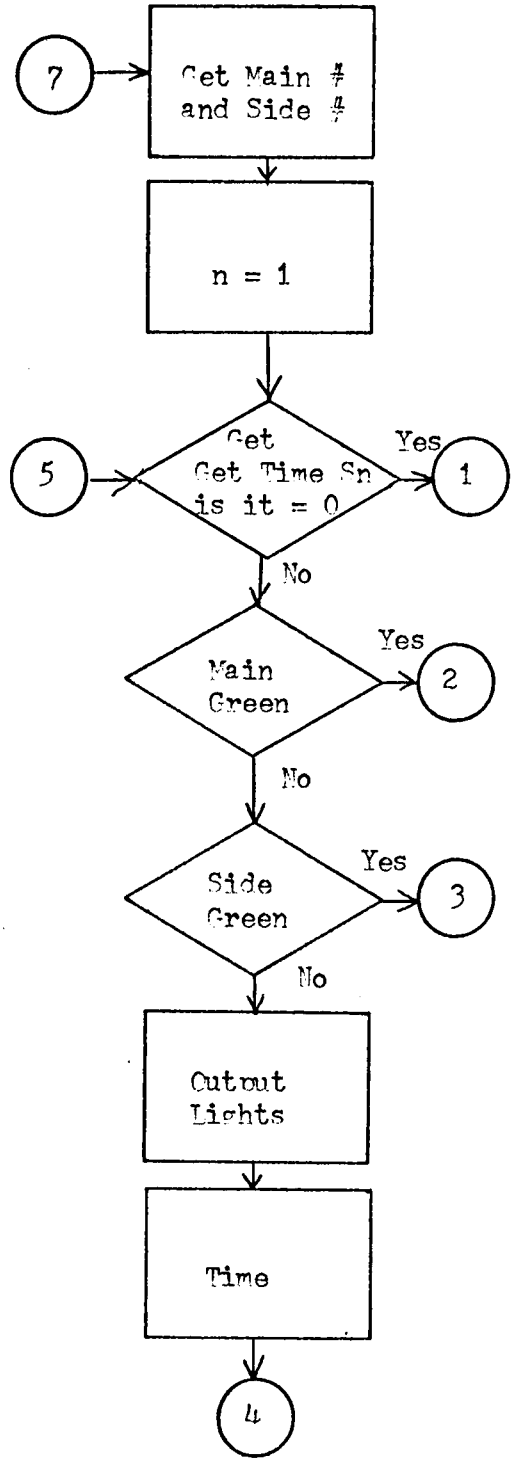


Figure 32 (continued)

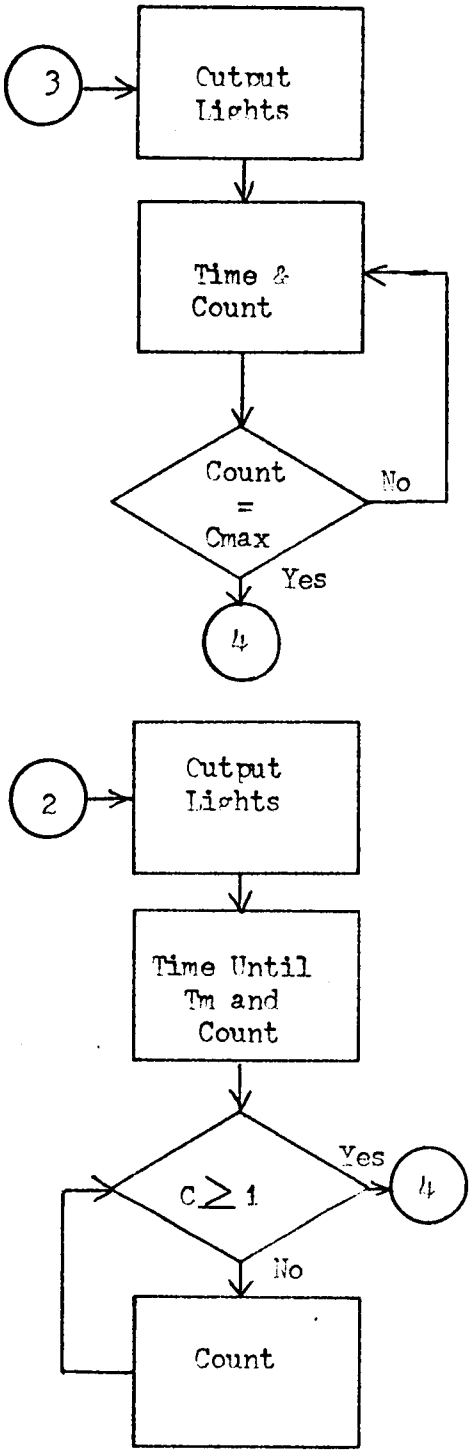


Figure 32 (continued)

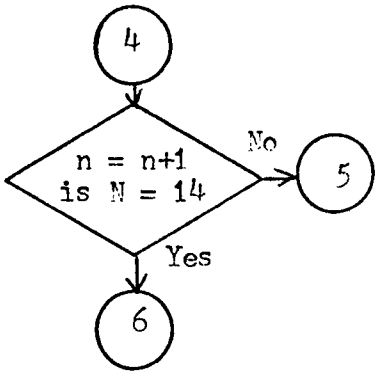


Figure 32 (continued)

of the length of time each particular state is to last. An external switch selects one of five sequences to operate. Four of the sequences are fixed-time and one is the actuated sequence. Therefore, sets of thirty-two bytes completely specify the sequences developed.

An algorithm was developed to generate the actuated sequence application. This algorithm assumes the two streets of an intersection are Main and Side. The algorithm then proceeds as follows:

1. If the light is green on Main, the cars are counted on Side. This light remains green on Main a minimum time, TM. If, after TM seconds, at least one car has been counted on Side, the lights will change to green for Side. However, if no cars are on Side, the lights stay green on Main until a car appears on Side.

2. If the light is green on Side, the cars are counted on Main. This light remains green on Side a maximum time, TS. If the car count reaches a maximum before TS, the lights change to green for Main. If the car count does not reach the maximum, the lights change to green for Main at TS anyway.

Again, thirty-two bytes are used to specify the sequence, with only fourteen states allowed. Of the extra four bytes, one is not used, one contains the maximum car count, and the other two contain the state number of the Main green and the Side green. Only during these last two states are cars counted.

For demonstration purposes five typical sequences are developed. All times were acquired from the Knoxville Traffic Engineering Department. The first sequence is fixed-time and represents the

timing at the intersection of Cumberland Avenue and Seventeenth Street. Timing consists of 55 seconds of green on Cumberland, followed by a four second yellow. Twenty-seven seconds of green on Seventeenth are followed by a four second yellow. The second sequence demonstrates the Cumberland and Seventeenth intersection after 11:00 p.m., i.e., the lights blink red toward Seventeenth and yellow toward Cumberland. Timing for these lights is one second "ON" and one second "OFF."

Simulation of the intersection at Sixteenth Street and Cumberland Avenue is performed by the third sequence. This sequence starts with a 32 second green on Sixteenth Street followed by a four second yellow. Then westbound traffic on Cumberland gets a seven second advanced green. After the seven seconds, the light turns green for eastbound traffic. Both lights remain green for 38 seconds, at the end of which, westbound traffic gets a four second yellow, then red. As the westbound light turns red, the eastbound traffic gets a nine second delayed green followed by a four second yellow. As the eastbound light turns red, Sixteenth Street lights turn green, and the cycle repeats.

An example of how mechanical traffic lights are sometimes inoperable is developed in the fourth fixed-time sequence. This sequence simulates the Alcoa Highway and Kingston Pike intersection and begins with a 32 second green on Kingston Pike. This green is followed by a four second yellow and then red. As Kingston Pike turns

red, the Alcoa Highway side turns green for one second, yellow for one second, and back to red.

The actuated sequence is the fifth sequence developed. No information was available from the Knoxville Traffic engineer, therefore timing sequences are also developed. Specified times of 32 seconds are set for both Main and Side green. The maximum number of cars to be counted on Main before switching occurred is set to five. Because the random generator always generated at least one car in 32 seconds, the requirement that either 32 seconds or 3 cars, whichever occurred last, was set for the light to turn green to Side. Also, the car count developed for each side is displayed on the LED's of the desk calculator application.

As implemented, the program can easily be stored in less than 256 bytes of memory. By storing the program in a 256 × 8 ROM and the light patterns and times in a PROM, the entire routine would require only two integrated circuits. However, if this were to be implemented in a real situation, a few changes would be made. First, a 24-hour clock would be connected to the interrupt line so that the controller would change sequences during the denser traffic periods. Second, the presence of a car would generate an interrupt. This would delete the processor having to look for a car constantly. With these changes and this traffic controller algorithm, this application could very well replace the mechanical controllers presently used.

III. ADC APPLICATION

Using a microcomputer as an ADC develops one application of a microprocessor as a data acquisition device. The type of ADC designed is an eight-bit successive approximation ADC. A successive approximation ADC begins by setting an initial digital output. This digital output is converted to an analog voltage and compared to the voltage to be converted (V_0). The initial digital setting is either incremented or decremented, depending on whether the initial output was less than or greater than V_0. Repetition of this cycle is performed until the two voltages are the same. Rather than incrementing or decrementing by a single digit, the ADC developed halves or doubles the digital values. Faster conversion times are obtained by this halving and doubling technique.

Figure 33 is a block diagram of the converter application. In the diagram, the digital-to-analog converter (DAC) takes the 8080 output, changes this digital output to an analog voltage, and inputs this voltage to the comparator. A comparison of the original voltage and the 8080 converted voltage is then made. If the 8080 voltage is greater than the original voltage, the comparator outputs a "1." If, however, the 8080 voltage is less than or equal to the original voltage, the comparator outputs a "0." Inputs and outputs of the 8080 inform the peripheral devices the state of the microcomputer. For example, a Busy Signal informs the peripheral devices the 8080 is currently in the conversion process. (Other I/O signals are discussed during the software discussion.)

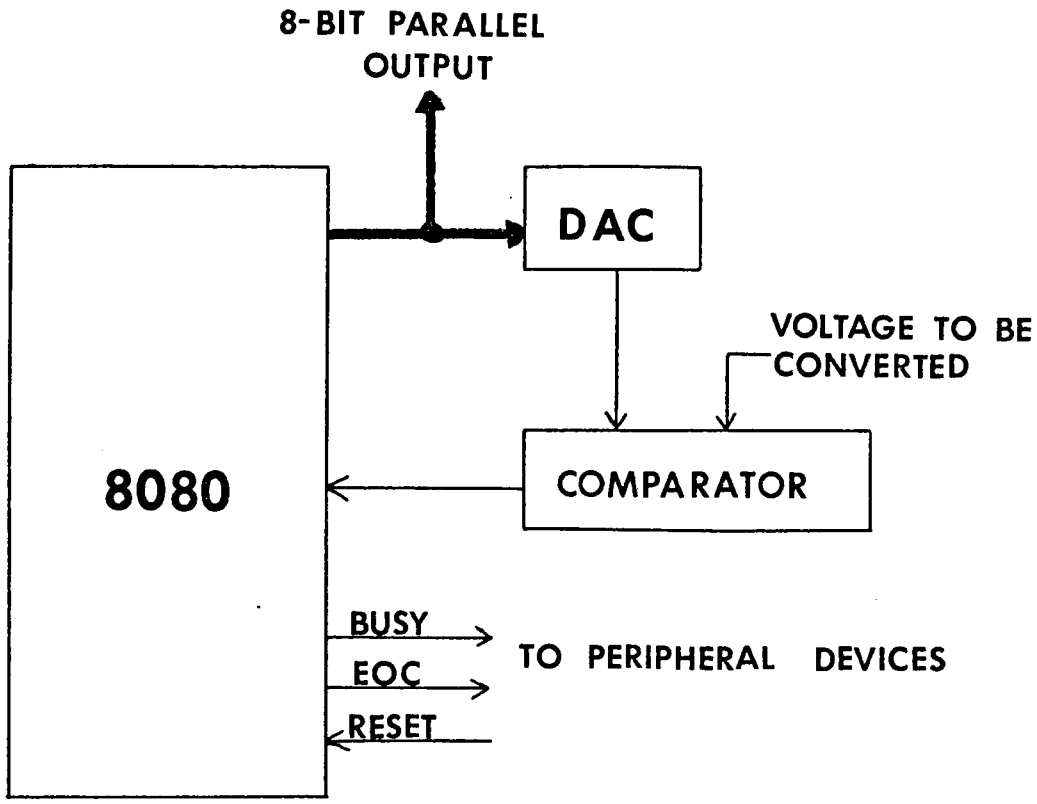


Figure 33. Block Diagram of ADC Application

The software for this ADC application is included in flowchart form in Figure 34 and as a program in Appendix H. As the program begins the accumulator is initialized to 80H, one-half of the largest number available in eight-bits. Also, a pointer, which is used to halve or double the accumulator value, is set to 80H. As the comparator inputs a "1," the program halves the contents of the accumulator. As the comparator inputs a "0," the program doubles the contents of the accumulator. Each time an operation is performed on the accumulator, the interval of values which the accumulator can become is decreased by one-half. For example, if the accumulator contains 80H and a "1" is input from the comparator, the program decreases the contents of the accumulator by one-half (to 40H). The next operation performed causes the values of the accumulator to fall between the values of 0 to 40H or from 40H to 80H; one-half of the values previously. Therefore, a successive approximator is created. This new digital voltage is then output to the DAC, and the sequence begins again. Once the least significant bit (LSB) is set, the 8080 outputs an end of conversion (EOC) signal. This EOC signal informs the peripheral devices that the eight-bits being output from the 8080 are the ADC converted bits. Another conversion is performed once a reset signal is input to the 8080.

An example of this sequence is given in Table 33. The conversion time for this particular sequence is 620.5 microseconds, which yields a 1.6 kilohertz conversion rate. Resolution of the eight-bit converter is 1 out of 256. Because the converted data is latched into an 8080 output port the data remains stable.

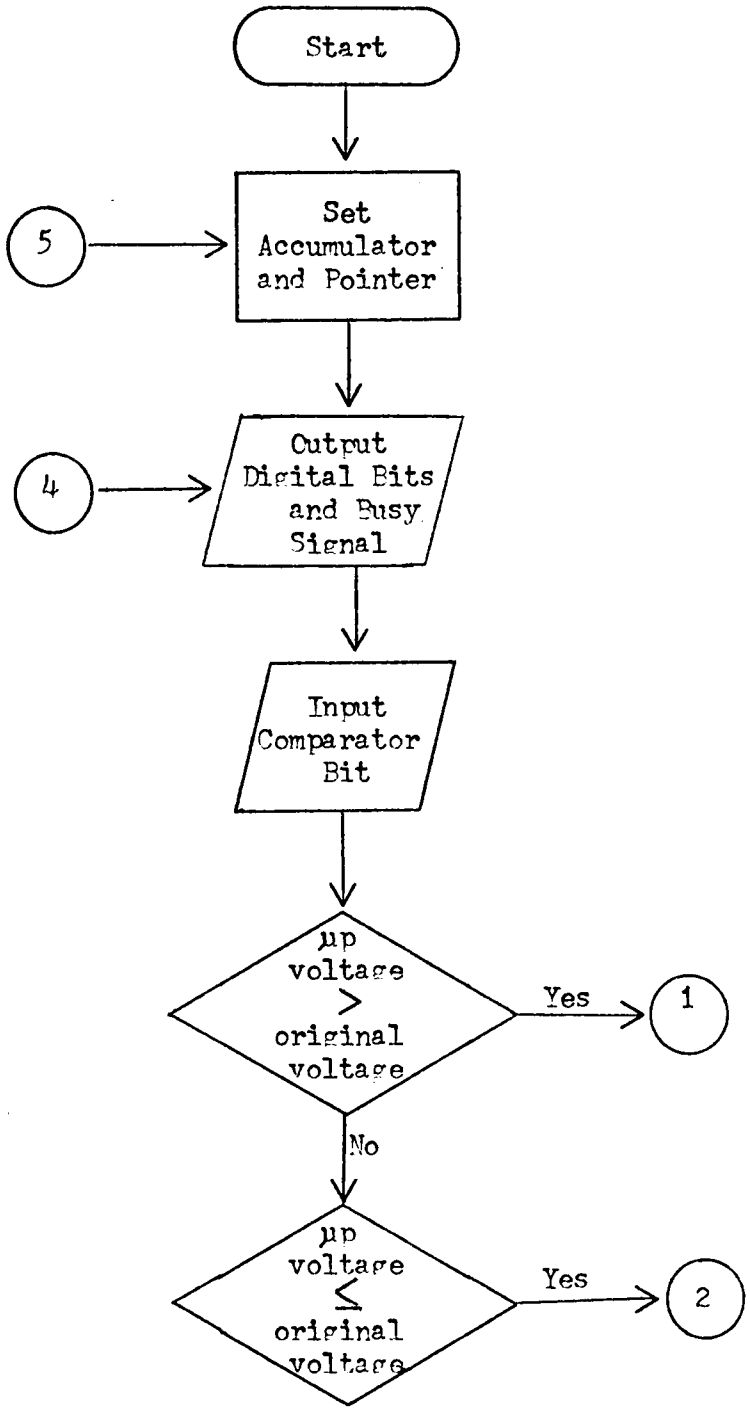


Figure 34. Flowchart of ADC Application

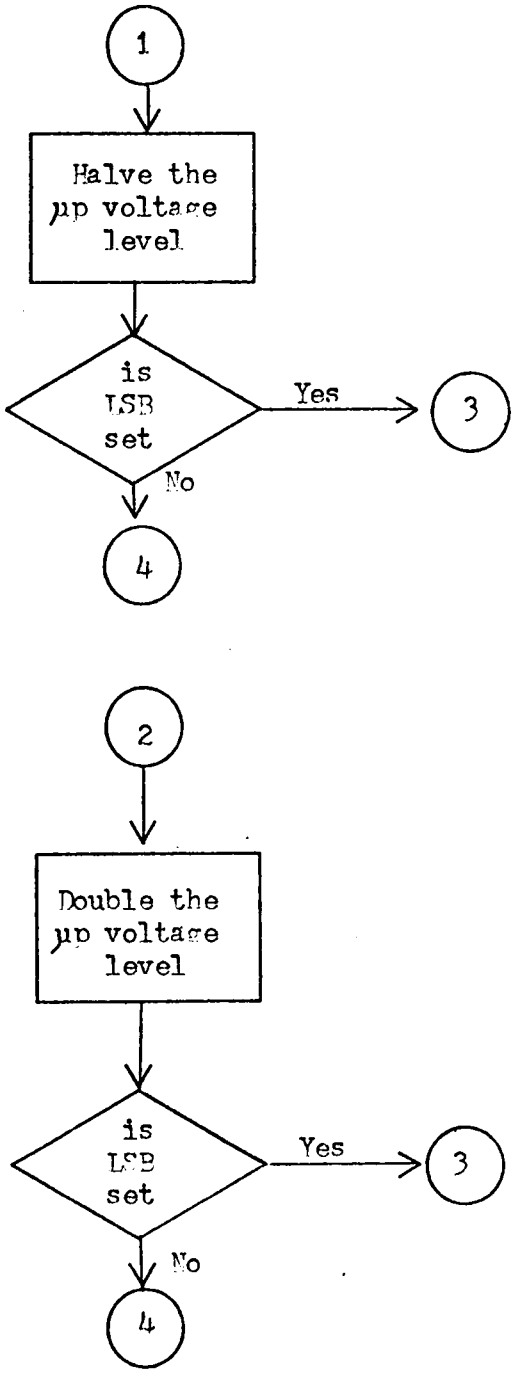


Figure 34 (continued)

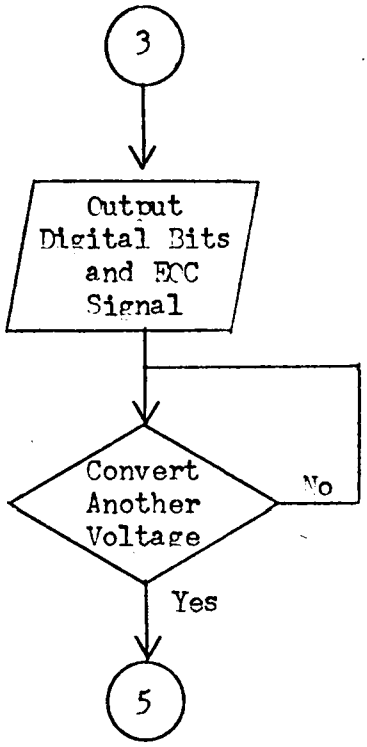
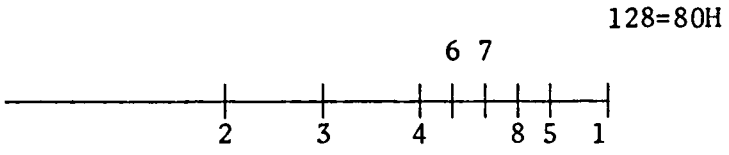


Figure 34 (continued)

TABLE 33

EXAMPLE OF SUCCESSIVE APPROXIMATOR AS AN ADC

Voltage to be converted = 117



Microprocessor Digital Output	Microprocessor		Original Voltage
(1) 1 0 0 0 0 0 0 0	"	>	"
(2) 0 1 0 0 0 0 0 0	"	<	"
(3) 0 1 1 0 0 0 0 0	"	<	"
(4) 0 1 1 1 0 0 0 0	"	<	"
(5) 0 1 1 1 1 0 0 0	"	>	"
(6) 0 1 1 1 0 1 0 0	"	<	"
(7) 0 1 1 1 0 1 1 0	"	>	"
(8) 0 1 1 1 0 1 0 1	"	<	"
After (8)			
0 1 1 1 0 1 0 1	LSB is set and ADC signal is 117.		

CHAPTER V

CONCLUSION

A comparison of the various types of microcomputers presently available has been developed. Many manufacturers are currently advertising or designing new microcomputers. These new microcomputer systems have been developed using some of the traits of the microcomputers presented. However, differences do appear, such as, twelve-bit machines are being developed and integrated injection logic is being used.

Future developments of microcomputers are being concentrated on speed and device density. With increased speed a microcomputer can challenge the minicomputer. As device density increases, so will the functional performance of the microcomputer. Typical of future microcomputers is a 500 nanosecond cycle time and an instruction set containing 500 instructions. With these advanced developments, microcomputers will not only be replacing random logic designs but will also be replacing minicomputers, in many applications.

LIST OF REFERENCES

LIST OF REFERENCES

1. Gladstone, Bruce. "Designing with Microprocessor Instead of Wired Logic Asks More of Designers," Electronics (October 11, 1973).

2. Lewis, Donald R., and Siena, Ralph W. "Microprocessor or Random Logic," Electronic Design 18 (September 1, 1973).

3. Shultz, Gaymond W., Holt, Raymond M., and McFarland, Harold, Jr. "A Guide to Using LSI Microprocessors," Computer (June 1973), 13-19.

4. Odgin, Jerry L. "Survey of Microprocessors Reveals Limitless Variety," EDN (April 20, 1974), 38-43.

5. "New Logic Notebook," Microcomputer Technique, Inc. (September 1974).

6. "Intel MCS-40 User's Manual for Logic Designers," Intel Corporation (November 1974).

7. "M6800 Systems Reference and Data Sheets," Motorola Semiconductor Corporation (1974).

8. "Intel 8080 Microcomputer System," Intel Corporation (1974).

9. "IMP-16C Application Manual," National Semiconductor Corporation (January 1974).

10. "IMP-00A/520 MOS/LSI Register and Arithmetic Logic Unit (RALU)," National Semiconductor Corporation (1974).

11. Ogdin, Jerry L. "Survey of Microprogrammable uP's Reveals Ultimate Software Flexibility," EDN (July 20, 1974), 69-74.

12. Vandling, Gilbert C., and Waldecker, Donald E. "The Microprogram Control Technique for Digital Logic Design," Computer Design (August 1969), 44-51.

13. Weiss, C. Dennis. "Software for MOS/LSI Microprocessors," Electronic Design 7 (April 1, 1974), 50-57.

14. Weiss, C. Dennis. "MOS/LSI Microcomputer Coding," Electronic Design 8 (April 12, 1974), 66-71.

15. "Programming Manual for the 8080 Microcomputer," Volume 2, Intel Corporation (May 1974).

16. Martinez, Ralph. "A Look at Trends in Microprocessor/Microcomputer Software Systems," Computer Design (June 1975), 51-57.

17. "PL/M Programming Manual," Intel Corporation (1974).

APPENDICES

APPENDIX A

INSTRUCTION SET OF THE INTEL 8080 MICROCOMPUTER

Mnemonic	Instruction
MOV r_1, r_2	Load register r_1 with the contents of r_2.
MOV r, M	Load register r with the content of the memory location addressed by the contents of registers H and L.
MOV M, r	Load byte two of the instruction into the memory location addressed by the contents of registers H and L.
MVI r $\langle B_2 \rangle$	Load byte two of the instruction into register r.
MVI M $\langle B_2 \rangle$	Load the memory location addressed by the contents of registers H and L with the content of register r.
INR r	The content of register r is incremented by one.
DCR r	The content of register r is decremented by one.
ADD r	Add the content of register r to the content of register A and place the result into register A.
ADC r	Add the content of register r and the contents of the carry flip-flop to the content of the A register and place the result into register A.
SUB r	Subtract the content of register r from the content of register A and place the result into register A.
SBB r	Subtract the content of register r and the content of the carry flip-flop from the content of register A and place the result into register A.
ANA r	Place the logical product of the register A and register r into register A.
XRA r	Place the "exclusive - or" of the content of register A and register r into register A.

Mnemonic	Instruction
ORA r	Place the "inclusive - or" of the content of register A and register r into register A.
CMP r	Compare the content of register A with the content of register r.
ADD M	ADD
ADC M	ADD with carry
SUB M	SUBTRACT
SBB M	SUBTRACT with borrow
ANA M	Logical AND
XRA M	Exclusive OR
ORA M	Inclusive OR
CMP M	COMPARE
ADI $<B_2>$	ADD
ACI $<B_2>$	ADD with carry
SUI $<B_2>$	SUBTRACT
SBI $<B_2>$	SUBTRACT with borrow
ANI $<B_2>$	Logical AND
XRI $<B_2>$	Exclusive OR
ORI $<B_2>$	Inclusive OR
CPI $<B_2>$	COMPARE
RLC	Rotate the content of register A left one bit.

Mnemonic	Instruction
RRC	Rotate the content of register A right one bit.
RAL	Rotate the content of Register A left one bit.
RAR	Rotate the content of register A right one bit.
JMP <B_2> <B_3>	Jump unconditionally to the instruction located in memory location addressed by byte two and byte three.
HLT	On receipt of the Halt Instruction, the activity of the processor is immediately suspended in the STOPPED state.
CALL <B_2> <B_3>	Jump unconditionally to the instruction located in memory location addressed by byte two and byte three of the instruction.

Conditional JUMP Instructions

JUMP:

JC <B_2> <B_3>	If (carry) = 1
JNC <B_2> <B_3>	If (carry) = 0
JZ <B_2> <B_3>	If (zero) = 1
JNZ <B_2> <B_3>	If (zero) = 0
JP <B_2> <B_3>	If (sign) = 0

Mnemonic	Instruction
JM <B_2> <B_3>	If (sign) = 1
JPE <B_2> <B_3>	If (parity) = 1
JPO <B_2> <B3>	If (parity) = 0

Conditional CALL Instructions

CALL:

CC <B_2> <B_3>	If (carry) = 1
CNC <B_2> <B_3>	If (carry) = 0
CZ <B_2> <B_3>	If (zero) = 1
CNZ <B_2> <B_3>	If (zero) = 0
CP <B_2> <B_3>	If (sign) = 0
CM <B_2> <B_3>	If (sign) = 1

Mnemonic	Instruction
CPE <B_2> <B_3>	If (parity) = 1
CPO <B_2> <B_3>	If (parity) = 0
RET	Return to the instruction in the memory location addressed by the last values shifted into the push-down stack addressed by SP.

Conditional RETURN Instruction

RETURN:

RC	If (carry) = 1
RNC	If (carry) = 0
RZ	If (zero) = 1
RNZ	If (zero) = 0
RP	If (sign) = 0
RM	If (sign) = 1
RPE	If (parity) = 1
RPO	If (parity) = 0
RST	Interrupt Instruction
IN <B_2>	Byte two of the instruction, which denotes the I/O device number, is sent to the I/O device through the address lines. New data for the accumulator is loaded from the data bus when DBIN control signal is active.
OUT <B_2>	Byte two of the instruction, which denotes the I/O device number, is sent to the I/O device through the address lines. The content of the accumulator is made available on the data bus when the $\overline{WR}$ control signal is 0.

Mnemonic	Instruction
LXI B <B_2> <B_3>	Load byte two of the instruction into C. Load byte three of the instruction into B.
LXI D <B_2> <B_3>	Load byte two of the instruction into E. Load byte 3 of the instruction into D.
LXI H <B_2> <B_3>	Load byte two of the instruction into L. Load byte three of the instruction into H.
LXI SP <B_2> <B_3>	Load byte two of the instruction into the lower order 8-bit of the stack pointer and byte three into the higher order 8-bit of the stack pointed.
PUSH PSW	Save the contents of A and F (5-flags) into the pushdown stack addressed by the SP register.
PUSH B	Save the contents of B and C registers.
PUSH D	Save the contents of D and E registers.
PUSH H	Save the contents of H and L registers.
POP PSW	Restore the last values in the pushdown stack addressed by SP into A and F.
POP B	Restore the last values in the pushdown stack addressed by SP into B and C.
POP D	Restore the last values in the pushdown stack addressed by SP into D and E.
POP H	Restore the last values in the pushdown stack addressed by SP into H and L.
STA <B_2> <B_3>	Store the accumulator content into the memory location addressed by byte two and byte three of the instruction.
LDA <B_2> <B_3>	Load the accumulator with the content of the memory location addressed by byte two and byte three of the instruction.

Mnemonic	Instruction
XCHG	Exchange the contents of registers H and L and registers D and E.
XTHL	Exchange the contents of registers H, L and the last values in the pushdown stack addressed by registers SP.
SPHL	Transfer the contents of registers H and L into register SP.
PCHL	JUMP INDIRECT
DAD SP	Add the content of register SP to the content of registers H and L and place the result into registers H and L.
DAD B	Add the content of registers B and C to the content of registers H and L and place the result into registers H and L.
DAD H	Add the content of registers H and L to the content of registers H and L and place the result into registers H and L.
DAD D	Add the content of registers D and E to the content of registers H and L and place the result into registers H and L.
STAX B	Store the accumulator content in the memory location addressed by the content of registers B and C.
STAX D	Store the accumulator content into the memory location addressed by the content of register D and E.
LDAX B	Load the accumulator with the content of the memory location addressed by the content of registers B and C.
LDAX D	Load the accumulator with the content of memory location addressed by the content of register D and E.
INX B	The content of register pair B and C is incremented by one.
INX H	The content of register H and L is incremented by one.

Mnemonic	Instruction
INX D	The content of register D and E is incremented by one.
INX SP	The content of register SP is incremented by one.
DCX B	The content of register B and C is decremented by one.
DCX H	The content of register H and L is decremented by one.
DCX D	The content of register D and E is decremented by one.
DCX SP	The content of register SP is decremented by one.
CMA	The content of accumulator is complemented.
STC	Set the carry flip-flop to 1.
CMC	The content of carry is complemented.
DAA	Decimal Adjust Accumulator
SHLD $\langle B_2 \rangle$ $\langle B_3 \rangle$	Store the contents of registers H and L into the memory location addressed by byte two and byte three of the instructions.
LHLD $\langle B_2 \rangle$ $\langle B_3 \rangle$	Load the registers H and L with the contents of the memory location addressed by byte two and byte three of the instruction.
EI	Interrupt System Enable
DI	Interrupt System Disable
INR M	The content of memory designated by registers H and L is incremented by one.
DCR M	The content of memory designated by registers H and L is decremented by one.

APPENDIX B

INSTRUCTION SET OF THE MOTOROLA 6800 MICROPROCESSOR

Accumulator and Memory Instructions

Mnemonic	Instructions
ADDA	Add
ADDB	
ABA	Add Accumulators
ADCA	Add with Carry
ADCB	
ANDA	And
ANDB	
BITA	Bit Test
BITB	
CLR	Clear
CLRA	
CLRB	
CMPA	Compare
CMPB	
CBA	Compare Accumulators
COM	Complement, 1's
COMA	
COMB	
NEG	Complement, 2's
NEGA	(Negate)
NEGB	
DAA	Decimal Adjust, A
DEC	Decrement
DECA	
DECB	
EORA	Exclusive OR
EORB	
INC	Increment
INCA	
INCB	
LDAA	Load Accumulator
LDAB	
ORAA	Or, Inclusive
ORAB	
PSHA	Push Data
PSHB	
PULA	Pull Data

Mnemonic	Instructions
PULB	
ROL	Rotate Left
ROLA	
ROLB	
ROR	Rotate Right
RORA	
RORB	
ASL	Shift Left, Arithmetic
ASLA	
ASLB	
ASR	Shift Right, Arithmetic
ASRA	
ASRB	
LSR	Shift Right, Logic
LSRA	
LSRB	
STAA	Store Accumulator
STAB	
SUBA	Subtract
SUBB	
SBA	Subtract Accumulator
SBCA	Subtract with Carry
SBCB	
TAB	Transfer Accumulators
TBA	
TST	Test, Zero or Minus
TSTA	
TSTB	

Index Register and Stack Manipulation Instructions

Mnemonic	Instruction
CPX	Compare Index Register
DEX	Decrement Index Register
DES	Decrement Stack Pointer
INX	Increment Index Register
INS	Increment Stack Pointer
LDX	Load Index Register
LDS	Load Stack Pointer
STX	Store Index Register
STS	Store Stack Pointer
TXS	Index Register $\longrightarrow$ Stack Pointer
TSX	Stack Pointer $\longrightarrow$ Index Register

Jump and Branch Instructions

Mnemonic	Instructions
BRA	Branch Always
BCC	Branch If Carry Clear
BCS	Branch If Carry Set
BEQ	Branch If = Zero
BGE	Branch if $\geq$ Zero
BGT	Branch If > Zero
BHI	Branch If Higher
BLE	Branch If $\leq$ Zero
BLS	Branch If Lower or Same
BLT	Branch If < Zero
BMI	Branch If Minus
BNE	Branch If Not Equal Zero
BVC	Branch If Overflow Clear
BVS	Branch If Overflow Set
BPL	Branch If Plus
BSR	Branch To Subroutine
JMP	Jump
JSR	Jump to Subroutine
NOP	No Operation
RTI	Return From Interrupt
RTS	Return From Subroutine
SWI	Software Interrupt
WAI	Wait For Interrupt

Condition Code Register Manipulation Instructions

Mnemonic	Instruction
CLC	Clear Carry
CLI	Clear Interrupt Mask
CLV	Clear Overflow
SEC	Set Carry
SEI	Set Interrupt Mask
SEV	Set Overflow
TAP	Accumulator A $\longrightarrow$ CCR
TPA	CCR $\longrightarrow$ Accumulator A

APPENDIX C

INSTRUCTION SET OF THE NATIONAL IMP-16C MICROCOMPUTER

Memory Reference Instructions

Mnemonic	Instruction
LD	Load
LD	Load Indirect
ST	Store
ST	Store Indirect
ADD	Add
SUB	Subtract
JMP	Jump
JMP	Jump Indirect
JSR	Jump to Subroutine
JSR	Jump to Subroutine Indirect
ISZ	Increment and Skip If Zero
DSZ	Decrement and Skip If Zero
SKAZ	Skip If AND Is Zero
SKG	Skip If Greater
SKNE	Skip If Not Equal
AND	And
OR	Or

Register Reference Instructions

Mnemonic	Instruction
PUSH	Push on to Stack Register
PULL	Pull from Stack
AISZ	Add Immediate, Skip if Zero
LI	Load Immediate
CAI	Complement and Add Immediate
RCPY	Register Copy
XCHRS	Exchange Register and Top of Stack
RXCH	Exchange Registers
RAND	Register And
RXOR	Register Exclusive Or
RADD	Register Add
SHL	Shift Left
SHR	Shift Right
ROL	Rotate Left
ROR	Rotate Right

Input/Output, Flag, and Halt Instructions

Mnemonic	Instruction
SFLG	Set Flag
PFLG	Pulse Flag
PUSHF	Push Flags On Stack
PULLF	Pull Flags From Stack
RIN	Register In
ROUT	Register Out
HALT	Halt

Transfer of Control Instructions

Mnemonic	Instruction
BOC	Branch-On Condition
RTS	Return from Subroutine
RTI	Return from Interrupt
JSRI	Jump to Subroutine Implied

IMP-16C Extended Instruction Set

Mnemonic	Instruction
MPY	Multiply
DIV	Divide
DADD	Double Precision Add
DSUB	Double Precision Subtract
LDB	Load Byte
STB	Store Byte
SETST	Set Status Flag
CLRST	Clear Status Flag
SKSTF	Skip If Status Flag True
SETBIT	Set Bit
CLRBIT	Clear Bit
CMPBIT	Complement Bit
SKBIT	Skip If Bit True
ISCAN	Interrupt Scan
JINT	Jump Indirect to Level Zero Interrupt
JMPP	Jump Through Pointer
JSRP	Jump to Subroutine Through Pointer

APPENDIX D

ASSEMBLER ERROR MESSAGES USED BY THE INTEL 8080

A ADDRESS ERROR:

Description: This error indicates that the address referenced by a JUMP or CALL instruction is not in the range 0 to 65535.

B BALANCE ERROR:

Description: This error indicates that the parentheses in an expression are unbalanced, or that the quotes in an ASCII string are unbalanced.

E EXPRESSION ERROR:

Description: This error indicates a badly constructed expression. It usually occurs due to a missing operator, omitted comma, or a misspelled opcode.

F FORMAT ERROR:

Description: This error indicates an error in the format of a statement. It is usually caused by a missing operand or an extraneous operand.

I ILLEGAL CHARACTER:

Description: This error indicates that an invalid ASCII character is present in the statement. It is also caused by a numeric character which is too big for the base of the number in which it occurs.

M MULTIPLE DEFINITION:

Description: This error indicates that two or more labels exist which are identical or not unique in the first five characters.

N NESTING ERROR:

Description: This error indicates that an IF, ENDIF, MACRO, or ENDM statement is improperly nested.

P PHASE ERROR:

Description: This error indicates that the value of an element being defined changed between pass one and pass two of the assembly.

Q QUESTIONABLE SYNTAX:

Description: This error is usually caused by omitting an opcode.

R REGISTER ERROR:

Description: This error indicates that a register specified for an operation is invalid for that operation.

S STACK OVERFLOW:

Description: This error indicates that the assembler's internal expression evaluation stack became too large and overflowed the memory available to the assembler. It may be caused by using extremely long character strings, too many nested macros, too many nested IF statements, or expressions which are too complex.

T TABLE OVERFLOW:

Description: This error indicates that the assembler's symbol table space has been exhausted. This may be caused by using too many symbols, overly lengthly macro definitions, or too many macro references in an assembly.

U UNDEFINED IDENTIFIER:

Description: This error indicates that a symbol used in an operand field was never defined by appearing in the label field of another instruction.

V ILLEGAL VALUE:

Description: This error indicates that the value of an operand or expression exceeds the range required for a particular operation.

APPENDIX E

EDITOR COMMANDS FOR THE INTEL 8080

A COMMAND (APPEND):

The format of the Append command is:

A$$

Description: The Append command causes text to be read from the assigned system READER device and to be appended to the workspace until one of the following conditions are met.

1. End of tape
2. End of file character (control-Z) read
3. Workspace full
4. 50 lines read

I COMMAND (INSERT):

The format of the Insert command is:

I inserted text $$

Description: The Insert command causes text to be input from the CONSOLE device keyboard into the workspace. Text is inserted at the location pointed to by the buffer pointer and the buffer pointer is positioned after the last character of the inserted text.

B COMMAND (BEGINNING):

The format of the Beginning command is:

B$$

Description: The B, or Beginning, command moves the buffer pointer to the beginning of the workspace. This is useful when data is to be inserted at the front of the workspace or in order to type out the entire contents of the workspace.

Z COMMAND (END OF WORKSPACE):

The format of the End of Workspace command is:

Z$$

Description: The Z command is used to position the buffer pointer to the end of the workspace. This command is used before appending text to the end of the workspace.

C COMMAND (CHARACTER):

The format of the Character command is:

nC$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The C or Character command moves the buffer pointer forward or backward the number of characters specified by n. A negative argument causes the buffer pointer to move backward, and a positive argument causes the buffer pointer to move forward. If n is greater than the number of characters between the buffer pointer and the end (or beginning) of the workspace, the buffer pointer is moved to the end (or beginning) of the workspace. If n is 0, no movement occurs.

L COMMAND (LINE):

The format of the Line command is:

nL$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The L or Line command moves the buffer pointer the number of lines specified by n. A positive argument causes the buffer pointer to move forward, and a negative argument causes the buffer pointer to move backward. An argument of 0 causes the buffer pointer to move backward to the first previous LINE FEED, i.e., the beginning of the current line. If n is greater than the number of lines between the buffer pointer and the end (or beginning) of the workspace, the buffer pointer is moved to the end (or beginning) of the workspace.

T COMMAND (TYPE)

The format of the Type command is:

nT$$

n is a decimal number from -254 to +255. If n is out of range, it is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The T, or Type command causes the number of lines specified by n to be output to the system CONSOLE device. Typing starts at the current location of the buffer pointer. n indicates the number of lines to be typed, if positive, or the number of previous lines to be typed if negative. If 0, the characters from the previous LINE FEED to the current buffer pointer location will be typed. If n is greater than the number of lines existing after (or before) the buffer pointer, only the existing lines are typed.

W COMMAND (WRITE):

The format of the Write command is:

nW$$

n is a decimal number from -254 to +255. A negative number is treated as if it were positive by this command. n is evaluated modulo 256. If n is not present, it is assumed to be positive 1.

Description: The W, or Write command causes the specified number of lines to be output to the system PUNCH device. If the Teletype is assigned as the PUNCH device, the lines will be printed as well as punched.

K COMMAND (KILL):

The format of the Kill command is:

nK$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The K, or Kill command performs the same operation as the D command, except that it works on a line-by-line basis instead of a character-by-character basis. If n is 0, the characters from the buffer pointer to the first previous line feed will be deleted. If n is greater than the number of lines between the buffer pointer and the

end (or beginning) of the workspace, the lines will be deleted and the buffer pointer will be at the end (or beginning) of the workspace.

E COMMAND (END):

The format of the End command is:

E$$

Description: The E, or End command causes the entire workspace to be output to the system PUNCH device, the workspace to be cleared, the remaining input on the input tape to be copied onto the PUNCH device, and the Editor to be reinitialized. If the Teletype is assigned as the system PUNCH device, the output will be printed as well as punched.

N COMMAND (NULL):

The format of the Null command is:

N$$

Description: The N, or Null command causes sixty null characters to be output to the system PUNCH device. This command may be used to punch leader or trailer paper tape.

D COMMAND (DELETE):

The format of the Delete command is:

nD$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The D, or Delete command causes a specified number of characters to be deleted from the workspace. The numeric argument which precedes the command determines the number of characters to be deleted, and its sign determines which direction is taken. A negative argument deletes characters preceding the buffer pointer, and a positive argument deletes characters following the buffer pointer. If n is 0, no action occurs. If n is greater than the number of characters between the buffer pointer and the end (or beginning) of the workspace, the appropriate characters are deleted and the buffer pointer is left at the end (or beginning) of the workspace.

F COMMAND (FIND):

The format of the Find command is:

Ftext$$

text is a string of length 16 characters or less, including any characters except ESCAPE, ALT MODE, or BREAK.

Description: The F, or Find command, causes the Editor to search for the first occurrence of a character string in the workspace. The string to be searched out appears as an alphanumeric argument after the initial F. The Editor will begin its search at the present location of the buffer pointer and will conclude upon reaching either of the following points:

(1) A match is found. The buffer pointer is then positioned immediately after the matched character string.

(2) The end of workspace is encountered. Text Editor then types the message "CANNOT FIND" and the search string on the system CONSOLE device. The buffer pointer is not affected. If the string to be found is greater than 16 characters, a match will never occur.

S COMMAND (SUBSTITUTE):

The format of the Substitute command is:

S search string $ substitute string $$

search string and substitute string are strings of characters including any characters except ALT MODE, ESCAPE or BREAK.

Description: The S, or Substitute command searches the workspace for the first occurrence of a specified character string of up to 16 characters and substitutes another string for it, if the search is successful. The search begins at the present location of the buffer pointer, and ends either when the string is found, or at the end of the workspace, in which case the message "CANNOT FIND" and the search string is typed on the system CONSOLE device. If the search is successful, the buffer pointer is positioned after the substituted string. If the search is unsuccessful, the buffer pointer is unaffected.

APPENDIX F

ASSEMBLED PROGRAM OF THE DESK CALCULATOR APPLICATION

```
;
;       PROGRAM WHICH ADDS AND SUBTRACTS TWO NOS.
;       USING TTY AS I/O AND DISPLAYS TO THE LEDS.
;
;       PROGRAM BEGINS AT LOC Ø6ØØ
;
        ORG     6ØØH
;
;       CLEAR STORAGE LOCS.
;
BEGIN:  LXI     SP,ØØ6ØH
        MVI     H,1ØD
        LXI     B,1ØØH          ;CLEAR LOCATIONS
CLE2:   LDAX    B               ;100H-10AH  THESE
        XRA     A               ;ARE USED AS FLAG
        STAX    B               ;REGISTERS
        INX     B
        DCR     H
        JNZ     CLE2
        MVI     H,2ØD
        CALL    OUTPC           ;CLEAR DISPLAYS
        LXI     B,5ØØH
        LXI     D,4ØØH
CLEAR:  LDAX    B
        XRA     A
        STAX    B
        LDAX    D
        XRA     A
        STAX    D
        INX     B
        INX     D
        DCR     H
        JNZ     CLEAR
;
;       PLACE THE NOS. FROM
;       TTY INTO MEMORY
        MVI     C,ØDH
        CALL    38Ø9H
        MVI     C,ØAH
        CALL    38Ø9H
        LXI     H,5Ø1H
        MVI     D,ØH
```

```
PROC1:  MVI     E,ØEH
PROC2:  LDA     1Ø6H
        CPI     8H
        JZ      ERROR
        CALL    38Ø3H      ;INPUT DIGIT, MONITOR ROUTINE
        MOV     C,A
        CALL    38Ø9H      ;DISPLAY DIGIT
        CMA
        ANI     7FH
        PUSH    PSW
        LDA     1Ø7H
        CPI     ØH
        JZ      CON6
        CALL    OUTPC
        MVI     A,ØH
        STA     1Ø7H
CON6:   POP     PSW
        CPI     'C'        ;CLEAR ALL VALUES
        JZ      6ØØH
        CPI     7FH        ;CLEAR LAST ENTRY(RUBOUT)
        JZ      CLENT
        CPI     'S'        ;CHANGE SIGN
        JNZ     EQUL
        CALL    CHGSN
        JMP     PROC2
EQUL:   CPI     '='        ;HAS = BEEN INPUT
        JNZ     MINUS
        CALL    SUBT1
        JMP     ADSUB
MINUS:  CPI     '-'        ;MINUS ENTERED
        JNZ     PLUS
MINU1:  CALL    OTMIN
        MVI     A,1H
        STA     1Ø6H
        LDA     1Ø4H
        CPI     3H
        JZ      TSUB       ;IF FIRST NO. BEING
        MVI     A,2H       ;ENTERED LOAD FLAG
        STA     1ØØH       ;OF 2 IN LOC 1ØØH, IF
        JMP     PROC2      ;SECOND NO. LOAD FLAG
TSUB:   MVI     A,3H       ;OF 3 IN LOC 1Ø1H.
        STA     1Ø1H
        JMP     SECND
PLUS:   CPI     '+'        ;HAS + BEEN INPUT
        JNZ     DEC
PLUS1:  MVI     A,ØH
        STA     1Ø6H
        LDA     1Ø4H
        CPI     3H
```

```
        JZ      TAD       ;IF FIRST NO. BEING
        MVI     A,ØH      ;ENTERED LOAD FLAG OF Ø
        STA     1ØØH      ;IN 1ØØH , IF SECOND NO.
        JMP     PROC2     ;LOAD FLAG OF 1 IN LOC
TAD:    MVI     A,1H      ;1Ø1H
        STA     1Ø1H
        JMP     SECND
DEC:    CPI     '.'       ;DEC. POINT ENTERED
        JNZ     CONT
        MOV     A,E
        CPI     ØEH       ;IF DEC POINT IS ALREADY
        JM      PROC2     ;IN THE NO. IGNORE THE DEC
        DCR     E
        DCR     E
        MOV     A,E
        OUT     1
        JMP     SUBD
CONT:   CPI     'Ø'       ;IF ANY ERRONEOUS VALUE
        JM      ERROR     ;HAS BEEN ENTERED OUTPUT
        CPI     'A'       ;AN ERROR MESSAGE
        JP      ERROR
CONT2:  PUSH    PSW
        MVI     A,3H
        STA     1Ø4H
        LDA     1Ø6H
        INR     A
        STA     1Ø6H
        MOV     A,E
        CPI     ØEH       ;HAS . BEEN INPUT
        JZ      INTGR
        POP     PSW
        SUI     3ØH
        CALL    OUTP1
        CALL    DECENT
        JMP     PROC2     ;GET ANOTHER DIGIT
INTGR:  POP     PSW
        SUI     3ØH
        CALL    OUTP1
        MOV     C,A
        CALL    ENTER
        JMP     PROC2     ;GET ANOTHER DIGIT
SECND:  NOP
        LDA     1Ø6H
        CPI     ØH
        JNZ     MINS
        MVI     A,1H
        STA     1Ø7H
        NOP
        NOP
```

```
          NOP
          NOP
MINS:     LXI     H,4Ø1H    ;SET MEM LOC FOR NEXT NO.
          MVI     D,Ø1H     ;SET CHECK FOR 2ND ENTRY
          JMP     PROC1
SUBD:     NOP
          NOP
          MOV     A,D
          CPI     Ø1H
          JZ      SEDEC
          LXI     H,5ØAH    ;SET MEM LOC FOR DEC NO.
          JMP PROC2
SEDEC:    NOP
          NOP
          LXI     H,4ØAH    ;SET MEM LOC FOR 2ND DEC NO.
          JMP     PROC2
;
; THIS ROUTINE DETERMINES WHETHER AN
; ADDITION OR SUBTRACTION SHOULD BE PERFORMED
; BY CHECKING THE + AND - FLAGS. IF ONLY +'S
; ARE ENTERED -- ADDITION. IF TWO -'S, ADDITION
; WITH - OUTPUT. IF ONE + AND ONE - -- SUBTRACTION
;
ADSUB:    LDA     1ØØH      ; 1ST NO. POS OR NEG
          CPI     ØH
          JZ      AD1
          LDA     1Ø1H      ; 2ND NO. POS OR NEG
          CPI     1H
          JZ      AD2
          MVI     A,2H      ; IF BOTH ARE NEG
          STA     1Ø5H      ; SET SIGN LOC TO - AND ADD
          JMP     ADD1
AD2:      LDA     1Ø5H      ; 1ST NEG AND 2ND POS CHECK
          CPI     2H        ; TO SEE IF THEY WERE XCHGED AND
          JZ      PLSN      ; IF SIGN FLAG SET TO MINUS. IF
          MVI     A,2H      ; SO RESET SIGN TO +. OTHERWISE
          STA     1Ø5H      ; SET TO - AND SUBTRACT
          JMP     SUBT
PLSN:     MVI     A,ØH      ; RESET SIGN FLAG TO +
          STA     1Ø5H
          JMP     SUBT
AD1:      LDA     1Ø1H      ; 1ST NO POS AND 2ND POS - ADD
          CPI     1H
          JZ      ADD1
          JMP     SUBT      ; 1ST POS AND 2ND NEG - SUBTRACT
;
;         ADD TWO NUMBERS. LOCS 5ØØ-513 AND 4ØØ-413
;         CONTAIN THE NOS. THE ANSWER WILL BE
;         STORED IN LOC 5ØØ-513.
```

```
;
ADD1:   XRA     A
        MVI     B,14H
        LXI     D,513H    ;LOAD ALL NOS.
        LXI     H,413H
        NOP
        NOP
        NOP
AGAIN:  LDAX    D         ;LOAD ACC FROM D,E
        ADC     M         ;ADD ACC TO MEM REG
        DAA
        CPI     1ØH       ;SHOULD CARRY BE SET
        JM      GOON
        ANI     ØFH
        STC
        JMP     GOON1
GOON:   CMC
GOON1:  STAX    D         ;STORE RESULT
        DCR     B
        JZ      OVER2
        DCX     D         ;GET SECOND SET OF DIGITS
        DCX     H
        JMP     AGAIN
;
; SUBTRACT TWO NOS. LOC 5ØØ CONTAINS THE
; MINUEND AND LOC 4ØØ CONTAINS THE SUBTRAHEND
; THE ANSWER WILL REPLACE THE MINUEND
;
SUBT:   MVI     C,14H
        LXI     D,513H    ; D CONTAINS DEC MINUEND
        LXI     H,413H    ; H CONTAINS DEC SUBTRAHEND
LOP1:   STC
LOP2:   MVI     A,Ø9H     ; GET THE 1Ø'S COMPLEMENT BY
        ACI     ØH        ; LOADING ACC WITH Ø9H AND ADD
        SUB     M         ; WITH CARRY.  SUB FROM
        XCHG              ; THIS SUBTRAHEND
        ADD     M         ; ADD MINUEND
        DAA               ; DECIMAL ADJUST
        CPI     1ØH
        JM      MINT      ; SHOULD BORROW OCCUR
        ANI     ØFH
        MOV     M,A       ; STORE RESULT
        XCHG
        DCR     C
        JZ      OVER2     ; DO ALL INTEGERS
        DCX     D
        DCX     H
        JMP     LOP1
MINT:   ANI     ØFH
```

```
        MOV     M,A
        XCHG
        DCR     C
        JZ      OVER2
        DCX     D
        DCX     H
        JMP     LOP2
;
;       WRITE THE ANSWER TO TTY AND LEDS.
;
OVER2:  CALL    OUTPC
        MVI     B,ØH
        LDA     1Ø5H        ;IF - FLAG SET OUTPUT
        CPI     ØH          ;A MINUS SIGN
        JZ      NMIN
        MVI     C,'-'
        CALL    38Ø9H
        INR     B
        CALL    OTMIN
NMIN:   MVI     D,ØAH       ;OUTPUT MSB
        LXI     H,5ØØH
WRITE:  NOP
        JMP     ZERO1       ;SKIP LEADING ZEROS
OT1:    CALL    ASCSUB      ;CONVERT TO ASCII
        INR     B
        MOV     A,B
        CPI     Ø8H         ;IF INTEGERS > 7
        JZ      ERROR       ;OUTPUT ERROR MESSAGE
        DCR     D
        JZ      DEC2
        INX     H
        MOV     A,M
        NOP
        NOP
        JMP     OT1
DEC2:   NOP
        DCR     E
        DCR     E
        MOV     A,E
        OUT     1
        NOP
        MVI     C,'.'       ;OUTPUT .
        CALL    38Ø9H
        MVI     D,ØAH       ;SKIP TRAILING ZEROS
        LXI     H,513H
        JMP     ZERO2
WRIT3:  LXI     H,5ØAH
WRIT2:  MOV     A,M         ;OUTPUT DEC NOS.
        CALL    ASCSUB
```

```
        INR     B
        MOV     A,B
        CPI     Ø8H
        JZ      ERROR
        DCR     D
        JZ      RETP
        INX     H
        JMP     WRIT2
;
; SKIP ALL LEADING ZEROES
;
ZERO1:  MOV     A,M
        CPI     ØH          ;IS CONTENTS = 0
        JNZ     OT1
        INX     H
        DCR     D
        JNZ     ZERO1
        MVI     C,'Ø        ; IF NO INTEGERS PUT A Ø
        CALL    38Ø9H
        MVI     A,ØH
        CALL    OUTP1
        INR     B
        JMP     DEC2
;
; SKIP ALL TRAILING ZEROES
;
ZERO2:  MOV     A,M
        CPI     ØH
        JNZ     WRIT3
        DCX     H
        DCR     D
        JNZ     ZERO2
        JMP     RETP        ; IF NO DEC. NOS. OUTPUT NONE
;
; PROGRAM GIVES A CARRIAGE RETURN, LINE FEED
; BEFORE THE NEXT NO. IS INPUT. AFTER THE CRLF
; A NEW ADD OR SUB MAY BE DONE TO THE ANSWER
; PREVIOUSLY CALCULATED OR AN ENTIRELY NEW
; ROUTINE MAY BE ENTERED.
;
RETP:   LDA     1Ø5H        ;IS ANSWER + OR -
        CPI     ØH
        JZ      POSAN
        MVI     A,2H        ;IF -, PUT FLAGS OF
        STA     1ØØH        ;2 AND 3 IN LOCS.
        MVI     A,3H        ; 100H AND 101 H.
        STA     1Ø1H
        JMP     RETCP
POSAN:  MVI     A,ØH        ;IF +, PUT FLAGS OF
```

```
        STA     1ØØH      ;0 AND 1 IN LOCS.
        MVI     A,1H      ;100H AND 101 H.
        STA     1Ø1H
RETCP:  MVI     A,ØH
        STA     1Ø5H
        LXI     D,4ØØH    ; CLEAR SEC. LOC BUT SAVE ANSWER
        MVI     B,14H
CL2:    LDAX    D
        XRA     A
        STAX    D
        INX     D
        DCR     B
        JNZ     CL2
        MVI     C,ØDH     ;CR
        CALL    38Ø9H
        MVI     C,ØAH     ;LF
        CALL    38Ø9H
INP6:   CALL    38Ø3H     ; LOOK FOR INPUT
        MOV     C,A
        CALL    38Ø9H
        CMA
        ANI     7FH
        MVI     D,ØH
        MVI     E,ØEH
        CPI     'C'
        JZ      BEGIN
        CPI     7FH       ; RUBOUT
        JZ      CLENT
        CPI     'S'
        JNZ     EQUL1
        CALL    CHGSN
        JMP     INP6
EQUL1:  CPI     '-'       ;- ENTERED, IF YES SUB NEXT
        JZ      MINU1     ;ENTRY FROM PREV. ANSWER
        CPI     '+'       ; + ENTERED, IF YES ADD NEXT
        JZ      PLUS1     ; ENTRY TO PREVIOUS ANSWER
        CPI     'Ø'
        JM      ERROR     ; CHECK FOR ILLEGAL INPUT
        CPI     'A'
        JP      ERROR
        PUSH    PSW
        MVI     B,2ØD
        LXI     H,1ØØH
        LXI     D,5ØØH    ; OK, CLEAR THE PREV INPUT
CL3:    LDAX    D         ; AND BEGIN NEW OPERATION
        XRA     A
        STAX    D
        MOV     A,M
        XRA     A
```

```
          MOV     M,A
          INX     D
          INX     H
          DCR     B
          JNZ     CL3
          CALL    OUTPC
          LXI     H,5Ø1H
          MVI     D,ØH
          MVI     E,ØEH
          POP     PSW
          JMP     CONT2
;
; OUTPUT AN ERROR MESSAGE WHEN AN
; INCORRECT DIGIT HAS BEEN DEPRESSED
;
ERROR:    MVI     C,'Y'
          CALL    38Ø9H
          MVI     C,'O'
          CALL    38Ø9H
          MVI     C,'U'
          CALL    38Ø9H
          MVI     C,' '
          CALL    38Ø9H
          MVI     C,'G'
          CALL    38Ø9H
          MVI     C,'O'
          CALL    38Ø9H
          MVI     C,'O'
          CALL    38Ø9H
          MVI     C,'F'
          CALL    38Ø9H
          MVI     C,'E'
          CALL    38Ø9H
          MVI     C,'D'
          CALL    38Ø9H
          MVI     C,ØFH
ER3:      MVI     A,ØDØH    ;OUTPUT E'S TO DISPLAY
          OUT     1
          MVI     A,ØFFH
          OUT     3
          CALL    ONESEC    ;CALL ONESEC ROUTINE
          DCR     C
          JZ      PROC1
          CALL    OUTPC
          CALL    ONESEC
          MVI     A,ØH
          STA     1Ø6H
          JMP     ER3
;
```

```
; CLEAR THE ENTRY WHICH HAS JUST BEEN INPUT
;
CLENT:  CALL    OUTPC
        MVI     B,14H
        MOV     A,D         ; 2ND NO. BEING ENTERED
        CPI     Ø1H         ; CLEAR 4ØØ-413
        JZ      ENT2
        LXI     H,5ØØH      ; 1ST NO. BEING ENTERED
NO2:    MOV     A,M         ; CLEAR 5ØØ-513
        XRA     A
        MOV     M,A
        INX     H
        DCR     B
        JNZ     NO2
        LXI     H,5Ø1H
CRLF:   MVI     C,ØDH       ; OUTPUT CRLF
        CALL    38Ø9H
        MVI     C,ØAH
        CALL    38Ø9H
        JMP     PROC2       ; START OVER
ENT2:   LXI     H,4ØØH
NO3:    MOV     A,M
        XRA     A
        MOV     M,A
        INX     H
        DCR     B
        JNZ     NO3
        LXI     H,4Ø1H
        JMP     CRLF
;
;       THIS SUBROUTINE
;       ENTERS THE NOS IN THE
;       CORRECT ORDER
;
ENTER:  NOP
        NOP
        NOP
        MVI     B,9D
REPT:   MOV     A,M         ;SHIFT THE NOS.
        DCX     H           ;TO THE RIGHT
        MOV     M,A         ;AS THEY ARE
        INX     H           ;ENTERED SO THAT
        INX     H           ; THE NUMBER JUST
        DCR     B           ; ENTERED WILL BE IN
        JNZ     REPT        ; THE LSB
        DCX     H
        MOV     A,C
        MOV     M,A
        MVI     B,8D
```

```
NEW:    DCX     H
        DCR     B
        JNZ     NEW
        RET


;       THIS SUBROUTINE
;       ENTERS THE DEC NOS.
;
DECENT: MOV     M,A       ; STORE THE DEC NOS.
        INX     H         ; AS THEY ARE ENTERED
        RET


;THIS SUBROUTINE
;OUTPUTS TO TTY
;
ASCSUB: NOP
        CALL    OUTP1     ; OUTPUT TO DISPLAY
        NOP
        NOP
HXD:    ANI     ØFH
        CPI     1ØD
        JM      CNO
        ADI     Ø7H
CNO:    ADI     3ØH       ; CHANGE TO ASCII
        MOV     C,A
        CALL    38Ø9H
        RET


; THIS SUBROUTINE CHECKS THE NOS.
; IF FIRST NO. IS SMALLER THAN SECOND
; NO. AND A SUBTRACTION IS TO OCCUR THE NOS.
; WILL BE SWITCHED IN LOCS. AND A FLAG FOR
; MINUS SIGN WILL BE SET
;
SUBT1:  LDA     1ØØH      ; IS FIRST NO. POS. OR NEG.
        CPI     ØH
        JZ      T2
        LDA     1Ø1H      ; IF FIRST NO. NEG. IS SEC NO.
        CPI     3H
        RZ
        JMP     GO2
T2:     LDA     1Ø1H      ; IF  FIRST NO. POS. IS SEC. NO.
```

```
        CPI     1H
        RZ              ; IF BOTH POS. DO NOT XCHNG
GO2:    MVI     B,14H
        LXI     D,5ØØH
        LXI     H,4ØØH
TRY:    MOV     C,M     ; LOAD C WITH CONTENTS IN MEM REG
        LDAX    D       ; LOAD ACC WITH CONTENTS OF D, E
        SUB     C       ; SUB A FROM C (FIRST-SECOND)
        JZ      TRY1
        RP              ; IF A>C DO NOT XCHNG
        JM      INVRT   ; IF A<C XCHNG
TRY1:   INX     D       ; IF A=C CHECK MSB TO LSB FOR A< C
        INX     H
        DCR     B       ; IF A=C FOR ALL BYTES DO
        RZ              ; NOT XCHNG NOS.
        JMP     TRY
INVRT:  MVI     A,2H    ; SET LOC 1Ø5H AS THE SIGN FLAG
        STA     1Ø5H
        MVI     B,14H
        LXI     D,5ØØH
        LXI     H,4ØØH
TEC:    MOV     C,M     ; LOAD C WITH M
        LDAX    D       ; LOAD ACC WITH D,E
        MOV     M,A     ; STORE ACC IN M REG
        MOV     A,C     ; PUT C INTO ACC
        STAX    D       ; STORE ACC IN D
        INX     D       ; GET NEXT BYTE FROM M
        INX     H
        DCR     B
        RZ              ; RETURN WHEN ALL BYTES XCNHGD
        JMP     TEC


;       THIS SUBROUTINE
;       CLEARS DISPLAYS
;
OUTPC:  MVI     E,ØEH   ; CLEAR DEC. POINT
        MVI     A,ØFEH
        OUT     1
        MVI     A,ØFFH  ; STROBE DISPLAYS
        OUT     3
        MVI     A,ØH
        OUT     3
        RET
```

```
;       THIS SUBROUTINE
;       OUTPUTS NUMBERS TO DISPLAY
;
OUTP1:  PUSH    PSW
        PUSH    PSW
        MVI     A,4ØH     ; STROBE EACH DISPLAY
        OUT     3         ; SEPERATELY, MSB TO LSB
        MVI     A,2ØH
        OUT     3
        MVI     A,1ØH
        OUT     3
        MVI     A,Ø8H
        OUT     3
        MVI     A,Ø4H
        OUT     3
        MVI     A,Ø2H
        OUT     3
        POP     PSW
        RLC
        RLC
        RLC
        RLC
        ANI     ØFØH
        PUSH    PSW
        MOV     A,E
        CPI     ØEH
        JNZ     DEC9      ; SHOULD DEC. POINT
        POP     PSW       ; BE STROBED
        ORA     E
        JMP     CON9
DEC9:   DCR     E
        DCR     E
        POP     PSW
        ORA     E
CON9:   OUT     1         ; OUTPUT NUMBER
        MVI     A,Ø1H
        OUT     3
        MVI     A,ØØH     ; PUT ALL STROBES HIGH
        OUT     3
        POP     PSW
        RET


; THIS SUBROUTINE
; OUTPUTS A MINUS SIGN
;
OTMIN:  MIV     A,ØBH
        CALL    OUTP1
        RET
```

```
; THIS SUBROUTINE
; CHANGES THE SIGN OF THE NUMBER
;
CHGSN:  CALL    OUTPC     ; CLEAR DISPLAY
        MOV     A,D
        CPI     ØH        ; FIRST OR SECOND NO.
        JZ      FNUM
        LDA     1Ø1H
        CPI     1H        ; SECOND NO. POS.
        JNZ     SPOS
        LDA     1Ø6H      ; IF SO OUTPUT A
        CPI     7H
        JZ      ERROR     ; MINUS SIGN
        CALL    OTMIN
        MVI     A,3H
        STA     1Ø1H
REDY:   LXI     H,4ØØH    ; OUTPUT THE SECOND NO.
        CALL    CHON
        LXI     H,413H
        CALL    CHZER
        MOV     A,C
        CPI     ØH
        JZ      NXNUM
        MOV     A,E       ; SHOULD DEC. POINT BE
        CPI     ØEH       ; OUTPUT
        CZ      DNUM
        LXI     H,4ØAH
        CALL    CHDEC
NXNUM:  RET
SPOS:   MVI     A,1H      ; SECOND NO. NEG.
        STA     1Ø1H      ; OUTPUT SECOND NO.
        JMP     REDY
FNUM:   LDA     1ØØH      ; FIRST NO. POS.
        CPI     ØH
        JNZ     FPOS
        LDA     1Ø6H
        CPI     7H
        JZ      ERROR
        CALL    OTMIN     ; OUTPUT MINUS SIGN
        MVI     A,2H
        STA     1ØØH
REDY2:  LXI     H,5ØØH    ; OUTPUT FIRST NO.
        CALL    CHON
        LXI     H,513H
        CALL    CHZER
        MOV     A,C
        CPI     ØH
        JZ      NXNM2
        MOV     A,E       ; SHOULD DEC. POINT
```

```
          CPI     ØEH       ; BE OUTPUT
          CZ      DNUM
          LXI     H,5ØAH
          CALL    CHDEC
NXNM2:    RET
FPOS:     MVI     A,ØH      ; FIRST NO. NEG.
          STA     1ØØH      ; OUTPUT FIRST NO.
          JMP     REDY2


; THIS SUBROUTINE
; SKIP TRAILING ZEROS
; OF THE CHANGE SIGN ROUTINE
CHZER:    MVI     C,ØAH
SKP2:     MOV     A,M
          CPI     ØH        ; IS MEM. LOC. ZERO
          RNZ
          DCX     H
          DCR     C
          JNZ     SKP2
          RET


; THIS SUBROUTINE
; CHANGES DEC NOS. FOR
; CHANGE SIGN ROUTINE
CHDEC:    MOV     A,M
          CALL    OUTP1     ;OUTPUT NO.
          DCR     C
          RZ
          INX     H
          JMP     CHDEC


; THIS SUBROUTINE
; CHANGES INTEGERS FOR
; CHANGE SIGN ROUTINE
CHON:     MVI     C,ØAH
SKP1:     MOV     A,M
          CPI     ØH        ; IS MEM. LOC. ZERO
          JNZ     WRTN
          INX     H
          DCR     C
          JNZ     SKP1
          RET
WRTN:     CALL    OUTP1     ; OUTPUT NO.
```

```
        INX     H
        MOV     A,M
        DCR     C
        JNZ     WRTN
        RET


; THIS SUBROUTINE OUTPUTS
; A DEC. FOR CHANGE SIGN ROUTINE
;
DNUM:   MVI     E,ØEH
        DCR     E
        DCR     E
        MOV     A,E       ; OUTPUT DEC. POINT.
        OUT     1
        RET
```

APPENDIX G

ASSEMBLED PROGRAM OF THE TRAFFIC CONTROLLER APPLICATION

```
          ; THIS PROGRAM SIMULATES A TRAFFIC
          ; CONTROLLER.  IT ALLOWS FOR 5 FIXED TIME
          ; SEQUENCES AND 1 ACTUATED SEQUENCE.
          ;
          ORG     ØBØØH
START:    LXI     SP,1ØØH         ; INIT SP
LP1:      CALL    OUTPC           ; CLEAR DISPLAY
          IN      3               ; INPUT SWR
          CMA
          ANI     7H
          CPI     6H              ; TST PGR SWT
          JZ      6ØØH
          CPI     5H              ; ACTUATED?
          JZ      ACT             ; YES
          LXI     H,45ØH          ; INIT VECTORS
          LXI     D,46ØH
          MOV     B,A
          CPI     ØH              ; WHICH SEQ?
          JZ      LP3
LP2:      MVI     A,2ØH           ; BUMP VECTORS
          ADD     L
          MOV     L,A
          ADI     1ØH
          MOV     E,A
          DCR     B
          JNZ     LP2
LP3:      MVI     B,ØFH           ; 16 STATES IN FIXED
SETUP:    LDAX    D               ; GET TIME
          PUSH    D               ; SAVE ADDRESS
          CPI     ØH              ; ZERO?
          JZ      REST            ; YES
          PUSH    PSW             ; SAVE TIME
          MOV     A,M             ; GET LIGHTS
          OUT     1               ; OUTPUT
          POP     PSW             ; RESTORE TIME
TIME:     CALL    ONESEC          ; TIMER LOOP
          DCR     A
          JNZ     TIME
REST:     INR     L               ; BUMP POINTERS
          POP     D               ; GET ADDRESS
          INR     E
          DCR     B
```

```
        JNZ     SETUP           ; FINISHED CYCLE?
        JMP     LP1             ; START OVER
        ; THIS SECTION PERFORMS THE ACTUATED SEQUENCE
        ;
ACT:    LXI     H,Ø5DEH         ; ACTUATED SEQUENCE
        MOV     A,M             ; GET MAIN #
        INR     L
        MOV     D,A             ; SAVE MAIN #
        MOV     A,M             ; GET SIDE #
        MOV     E,A             ; SAVE SIDE #
ACT1:   MVI     B,ØH            ; 14 STATES MAX
        LXI     H,Ø5DØH         ; SET VECTOR
ACT2:   MOV     A,M             ; GET TIME
        CPI     ØH              ; ZERO
        JZ      REST3
        PUSH    PSW             ; SAVE TIME
        MOV     A,B
        CMP     D               ; MAIN?
        JZ      MAIN
        CMP     E               ; SIDE?
        JZ      SIDE
        MOV     A,L             ; GET PATTERN
        ANI     ØEFH
        MOV     L,A
        MOV     A,M
        OUT     1               ; OUTPUT
        POP     PSW             ; GET TIME
        PUSH    D               ; SAVE MAIN AND SIDE
ACT3:   CALL    ONESEC          ; TIMER LOOP
        DCR     A
        JMZ     ACT3
        JMP     REST1
MAIN:   MOV     A,L             ; MAIN GREEN
        ANI     ØEFH            ; GET PATTERN
        MOV     L,A
        MOV     A,M
        OUT     1               ; OUTPUT
        MVI     A,1CH           ; SLOW CARS DOWN
        OUT     3
        XRA     A
        OUT     3
        MVI     C,ØH            ; CLEAR COUNTER
        POP     PSW             ; GET TIME
        PUSH    D               ; SAVE MAIN AND SIDE
MAIN1:  CALL    ONESEC          ; TIMER LOOP
        PUSH    PSW             ; SAVE TIME
        IN      3               ; LOOK FOR CAR
        ANI     1ØH
        ADD     C
```

```
        MOV     C,A             ; SAVE CAR COUNT IN C
        CALL    OUTP2           ; OUTPUT CARS
MAIN2:  POP     PSW             ; RESTORE TIME
        DCR     A
        JNZ     MAIN1
        MOV     A,C
MAIN3:  CPI     4ØH             ; WAIT ON CARS
        JP      REST1
        CALL    ONESEC
        IN      3
        ANI     1ØH
        ADD     C
        MOV     C,A
        PUSH    PSW
        CALL    OUTP2
        POP     PSW
        JMP     MAIN3
SIDE:   MOV     A,L             ; SIDE GREEN
        ANI     ØEFH            ; GET PATTERN
        MOV     L,A
        MOV     A,M
        OUT     1               ; OUTPUT
        MVI     A,2H            ; SPEED UP CARS
        OUT     3
        XRA     A
        OUT     3
        MVI     C,ØH            ; CLEAR COUNTER
        POP     PSW             ; GET TIME
        PUSH    D               ; SAVE MAIN AND SIDE
SIDE1:  CALL    ONESEC          ; TIMER LOOP
        PUSH    PSW             ; SAVE TIME
        IN      3               ; LOOK FOR CAR
        ANI     1ØH
        ADD     C               ; UPDATE COUNT
        MOV     C,A
        LDA     Ø5CEH
        CMP     C               ; MAX CARS?
        JZ      REST2           ; YES
        MOV     A,C             ; OUTPUT COUNT
        CALL    OUTP2           ; OUTPUT CARS
SIDE2:  POP     PSW             ; RESTORE TIME
        DCR     A
        JNZ     SIDE1
        JMP     REST1
REST2:  CALL    OUTP2
        POP     PSW             ; CLEAR STACK
REST1:  POP     D               ; RESTORE POINTERS
REST3:  MOV     A,L
        ORI     1ØH
```

```
        INR     A
        MOV     L,A
        INR     B               ; INC STATE #
        MVI     A,ØEH
        CMP     B               ; CYCLE FINISHED?
        JNZ     ACT2            ; NO
        JMP     LP1             ; YES


; THIS SUBROUTINE TIMES ONE SECOND
;
ONESEC: MVI     E,ØC8H          ; TIMES 1 SECOND
LOOP1:  MVI     D,ØFFH
LOOP2:  NOP
        NOP
        NOP
        NOP
        DCR     D
        JNZ     LOOP2
        DCR     E
        JNZ     LOOP1
        RET


;
; THIS SUBROUTINE CLEARS LATCHES AND
; OUTPUTS THE CAR COUNT
;
OUTP2:  MOV     A,C             ; PUT CARS IN A
        OUT     1
        MVI     A,Ø4H           ; CLEAR LATCH
        OUT     3
        INR     A               ; STROBE DISPLAY
        OUT     3
        XRA     A
        OUT     3
        MOV     A,M             ; OUTPUT LIGHTS
        OUT     1
        RET


;
; THIS SUBROUTINE CLEARS THE DISPLAY
;
OUTPC:  MVI     E,ØEH
        MVI     A,ØFEH          ; OUTPUT F TO
```

```
OUT     1               ; CLEAR DISPLAYS
MVI     A,ØFFH          ; OUTPUT STROBES
OUT     3
MVI     A,ØH
OUT     3
RET
END
```

APPENDIX H

ASSEMBLED PROGRAM OF THE ADC APPLICATION

```
        ; THIS PROGRAM PERFORMS THE ADC
        ; APPLICATION OF MICROCOMPUTERS.
        ;
        ORG     1ØØH
START:  IN      3               ; RESET INPUT ?
        CPI     ØH
        JZ      START
BEGIN:  MVI     A,2H            ; OUTPUT BUSY SIGNAL
        OUT     1
        MVI     A,8ØH           ; SET INITIAL VALUE
        MVI     B,8ØH           ; SET POINTER
CMPIN:  IN      3               ; GET COMPARATOR BIT
        ANI     1H              ; IS BIT SET TO
        CPI     ØH              ; >= OR <
        JZ      GRTHN
LESTN:  PUSH    PSW             ; IF < SET DOUBLE DIGITAL LEVEL
        MOV     A,B             ; ROTATE POINTER
        RAR                     ; TO THE RIGHT
        MOV     B,A
        POP     PSW
        ADD     B               ; ADD POINTER TO ACC.
        PUSH    PSW
        ANI     1H
        CPI     1H              ; IS LSB SET
        JZ      RESULT          ; IF SO OUTPUT ANSWER
        POP     PSW
        OUT     3
        JMP     CMPIN           ; COMPARE LEVELS AGAIN
GRTHN:  PUSH    B               ; IF >= SET HALVE
        PUSH    PSW             ; THE DIGITAL LEVEL
        MOV     A,B             ; ROTATE POINTER
        RAR                     ; TO THE RIGHT
        MOV     B,A
        POP     PSW
        ADD     B               ; ADD POINTER TO ACC.
        MOV     C,B
        POP     B
        XRA     B
        MOV     B,C
        PUSH    PSW
        ANI     1H
        CPI     1H              ; IS LSB SET
```

```
        JZ      RESULT          ; IF SO OUTPUT ANSWER
        POP     PSW
        OUT     3
        JMP     CMPIN           ; COMPARE LEVELS AGAIN
RESULT: POP     PSW
        OUT     3               ; OUTPUT A/D LEVEL
        MVI     A,1H
        OUT     1               ; OUTPUT EOC SIGNAL
        JMP     START
        END
```

VITA

Stephen Wayne Parker was born on 1 September 1949 in Kingsport, Tennessee. He graduated from Dobyns-Bennett High School in 1967. In 1971 he received the Bachelor of Science degree in Electrical Engineering from The University of Tennessee. At this time, he began working for the Naval Weapons Laboratory in Dahlgren, Virginia. In May 1974 he was accepted to a one year full time advanced study program, provided by the Naval Weapons Laboratory, and returned to The University of Tennessee. In August 1975 he received his Master of Science degree in Electrical Engineering from The University of Tennessee.

He is married to the former Mary Melissa Thompson of Knoxville, Tennessee.