MICROCOMPUTER DESCRIPTION AND SOFTWARE DEVELOPMENT

by

Richard S. Gogesch

Submitted in Partial Fulfillment of the Requirements
for the Degree of
Master of Science in Engineering
in the
Electrical Engineering
Program

Professor Samuel J. Skarote 6/8/76
Adviser Date

Dean of the Graduate School 6/9/76
Date

YOUNGSTOWN STATE UNIVERSITY

June, 1976

ABSTRACT

MICROCOMPUTER DESCRIPTION
AND SOFTWARE DEVELOPMENT

Richard S. Gogesch
Master of Science in Engineering
Youngstown State University, 1976

Basic features of some microprocessors are discussed. Internal operation of a typical microcomputer was investigated.

Special software features of a 4040 microprocessor are discussed. Part of the family (MCS-40) of peripheral devices that can be employed with the 4040 microprocessor are presented. A software system (assembly language + simulator) was developed. The test procedure to verify the operation of the software is given and a typical application follows.

The software developed serves to decrease program development time of a 4040 based microcomputer system.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to William L. Spetz, Applications Engineer, Intel Corporation. Mr. Spetz was a source of professional advice. He also obtained several integrated circuits for use in my thesis.

I would also like to express my sincere thanks to Marlin Rubright, Assistant Field Sales Manager, Pioneer/Cleveland. Mr. Rubright helped in the programming of two programmable read-only memories.

Sincere thanks are also due to Anna Mae Serrecchio for her fine work typing this thesis. Thank you, Professor Samuel J. Skarote for the excellent job you have done as my adviser.

TABLE OF CONTENTS

LIST OF SYMBOLS

SYMBOL	DEFINITION
CPU	Central processing unit
DMA	Direct memory access
I/0	Input or output
BCD	Binary coded decimal
ALU	Arithmetic Logic unit
MOS	Metal oxide semiconductor
DIP	Dual inline package
CMOS	Complementary metal oxide semiconductor
RAM	Random access memory
ROM	Read only memory
CM-ROM	Command line for read only memory
CM-RAM	Command line for random access memory
CY	Carry flip-flop
SYNC	Syncronization signal produced by central processing unit
OPR	High order 4 bits of machine instruction
OPA	Lower order 4 bits of machine instruction
MCS-40	Trade mark INTEL Corporation 4040 microcomputer system
TTL	Transistor-transistor logic

LIST OF FIGURES

I. INTRODUCTION

The objective of this thesis is to develop an assembly language and a simulator for the Intel 4040 microcomputer. The Intel 4040 microcomputer was chosen because documentation for this microcomputer was readily available.

Machine language programming is a very time-consuming and tedious process. An alternative to machine language programming lies in the use of assembly languages, which are software systems that allow the use of mnemonic operation codes and mnemonic statement labels. With the use of assembly language, the programmer can program more easily and quickly, however, once written, there is no assurance that the program will work properly. One method of testing a program is to actually run it on the microcomputer. An alternative to this approach is through simulation of the microcomputer. Simulation is a very popular method of testing programs, since simulation does not require the acquisition of an expensive microcomputer development system. The use of assembly languages and simulators, therefore, represent a cost effective means for microcomputer program development.

Chapter II describes the functions of a typical microcomputer. This chapter gives the reader background information on microcomputers. Chapter III describes the features of the Intel 4040 microcomputer. Chapter III also gives a detailed instruction description for the 4040 instruction set. Chapter IV gives a basic description of a few of the MCS-40

system components. Chapter V is a discussion of the assembly language and simulator developed. Chapter VI explains how the assembly language and simulator were tested. Chapter VI also presents a sample application of the use of the assembly language and simulator. The conclusions which were reached are found in Chapter VII. Appendix A contains a program listing of the cross-assembler. Sample output of the cross assembler is seen in Appendix B. Appendix C is a program listing of the simulator. Sample output of the simulator is seen in Appendix D. Appendix E contains a sample program used to test the assembler. Appendix F contains an example program developed to implement motor speed control.

A microcomputer is an integrated system of miniaturized electronic devices capable of performing the functions normally associated with random logic, minicomputers, and larger central processing units (CPU's).[1] A microcomputer replaces random logic by storing program sequences in memory, rather than implementing these logic functions with gates and flip-flops. The advent of the microcomputer has revolutionized many product fields and is being designed into many other areas.

Once a microcomputer has been chosen as an integral part of a design, the design engineer faces the problem of choosing the best microcomputer for the given application. With more than 100 microcomputers available this becomes a tedious process. The design engineer can, however, minimize this task by choosing some criteria to eliminate several microcomputers from consideration.

[1]"Put a complete Microprocessor in your System for less than $30", INTEL Corporation, 1975, p. 1-4

Some major features of microcomputers are:
Interrupt Structure,
One-Chip CPU Packaging,
Microprogrammability,
DMA (Direct Memory Access) Ability
Arithmetic Modes,
Speed,
Word Size,
and Power Requirements.[2]

These factors, while not the only important features of microcomputers, are important enough to be amoung the first to be examined.

Interrupts can be defined as the ability of a microprocessor to respond to an external event. Interrupts may or may not be an important feature, depending upon two factors: 1) does the application require real time quick response to external events, and 2) does the software design strategy encourage the use of interrupts? Many applications will not require interrupts, even some real time applications do not require interrupts. An alternative to interrupts is software organization such that external events are monitored frequently enough to guarantee service.

The next factor to be considered is one-chip CPU packaging. One-chip CPU packaging has a substantial effect on assembly and repair cost, as well as affecting the size of the finished product. Even among one-chip CPUs there are a tremendous variety in the number of integrated circuits which must be added to achieve a working computer. A CPU which multiplexes data and addresses through the same pins will require additional hardware support. Some other areas fre-

[2]Ogdin, J. and Phillips, S., "Processor Selection", New Logic Notebook, September, 1974, p. 3.

quently requiring substantial hardware support are state decoding and input/output (I/0) control. While the cost of this hardware support circuitry is small, it can double or triple the manufacturing costs of the computer.

Microprogrammability allows the fine structure of the microcomputer to be changed while the overall structure remains the same. An example of this would be the number of registers and set of instructions available to the programmer. They can be changed within limited realms, however, gross changes are not possible. Microprograms are usually stored in read only memories, either on the computer's control chip or externally in standard read only memories connected to the control logic. Microprogrammability becomes important when a microcomputer is designed to emulate some other computer, or to implement a specialized set of instructions. Common software routines such as multiply and divide can easily be implemented in a microprogram. Virtually all programs can be reduced to a micro-program.

DMA is an abbreviation for direct memory access. DMA lets high-speed peripheral devices gain direct access to main storage without disturbing the CPU; the alternative approach requires that the CPU read or write every word between memory and the peripheral device. In order for DMA to work the CPU must be prevented from interfering with the block of main storage in which DMA is in progress. Most microcomputers with DMA capability suspend CPU cycles when DMA is in progress. Some microcomputers with more sophisticated architectures

allow the CPU to continue operation while DMA is in progress unless I/O is attempted in core where DMA is occuring. This more sophisticated architecture, while more efficient, also requires substantial hardware support.

Arithmetic is usually performed in two's compliment form on microcomputers. In addition, some processors have special instructions which are designed for handling binary coded decimal (BCD) numbers. Whether these instructions are important or not depends upon whether data must be transferred in BCD form. If BCD data is presented to the computer, or required from it, BCD arithmetic is an essential feature of the microcomputer system.

Speed of a microcomputer may become important if real-time events must be handled within specified time limits. Word size is simply defined as the number of bits with which the microcomputer can directly perform arithmetic. Word size is not an important factor unless speed is required, since computers with smaller work lengths can perform functions of a larger microcomputer in an increased number of machine cycles. Power requirements are also important, since some technologies require several supply voltages, thereby increasing the cost of the system.

Another important consideration when selecting a microcomputer is vendor commitment; that is, will the vendor continue making the microcomputer for the entire applications life. Generally a good measure of vendor commitment is software support. If the vendor has a considerable investment in

software support, then chances are the microcomputer will be around for some time to come. Documentation is another measure of vendor commitment. A product with skimpy, vague, and ambiguous specifications usually connotes a management disinterest in the product. Products such as these should be avoided.

If large systems are dealt with, software support is essential. Minimal software support should include:

1. An assembler that allows use of symbolic operation codes and symbolic statement labels.
2. An editor that allows the programmer to easily change source programs for re-assembly.
3. A simulator that executes machine code for the microprocessor on a larger computer or minicomputer.[3]

[3]Ogdin, J. and McPhillips, S., p. 5.

II. FUNCTIONS OF A TYPICAL MICROCOMPUTER

A typical digital computer consists of the following:

1. A central processing unit (CPU).
2. A memory.
3. Input/Output (I/O) ports.[4]

Primarily the program memory serves as a place to store instructions, which are the coded pieces of data that direct all of the activities of the CPU. A group of instructions logically arranged in program memory is referred to as a program. The CPU fetches each instruction from memory in a logically determinate sequence, and uses it to initiate processing actions.

The data memory is used to store the data to be manipulated. The CPU can access all data in the data memory bank. At times there is not enough data memory to store all of the data required for the application. The solution to this problem lies in the use of input ports. The CPU can address these input ports. Data can be contained in these input ports. Another feature of the input ports is that they allow the CPU to receive information from peripheral devices.

Most computers have output ports in addition to input ports. These output ports allow the computer to transmit data outside the computer. The output may go to a display, to a peripheral device that produces a "hard copy", such as

[4]Raphael, Howard A., INTEL MCS-40 User's Namual for Logic Designers (Santa Clara, CA: INTEL Corporation, 1974) p. vii

a line printer, to a peripheral storage device, such as a magnetic tape unit, or the output may constitute process control signals, such as in an automated assembly line.

The CPU ties the system together. It controls all of the functions performed by the other devices. The CPU must be able to fetch instructions from memory, decode the binary instructions, and execute them. It must be able to reference memory and I/O ports. In addition to these functions, some CPUs can respond to certain control signals. Two examples of these control signals would be interrupt and stop.

A typical CPU consists of the following interconnected units.

1. Registers.
2. An Arithmetic/Logic Unit (ALU).
3. Control Circuitry.[5]

Registers are temporary storage units within the CPU. Some registers, such as the program counter and instruction register, have dedicated uses. Other registers, such as the accumulator, are for more general purpose use. As a result, these registers are usually referred to as general purpose registers.

The accumulator usually stores operands (numbers) which are to be manipulated by the arithmetic logic unit. A typical instruction might direct the ALU to addl to the accumulator and store the result in the accumulator. This is

[5]Raphael, Howard A., MCS-40, p. viii

an example where the accumulator is both a source (operand) and a destination (result) register.

Program instructions are stored in the program memory. The CPU must examine the contents of memory in order to determine which action is appropriate. This means the CPU must know which memory location contains the next instruction. Each of the memory locations is numbered to distinguish it from all other locations in memory. The number identifying the memory location is called an address.

The CPU contains a counter which contains the address of the next program instruction. This register is called the program counter. The CPU updates the counter by adding "1" to the counter each time it fetches an instruction. This process assures that the counter is always current.

The programmer therefore, stores instructions in numerically adjacent addresses, such that the instructions in lower addresses will be executed before instructions in the higher addresses. The only time this general rule is violated is when a "jump" instruction is executed.

A jump instruction contains the address of the instruction which is to follow it. The next instruction can be stored in any address, as long as the jump instruction specifies the correct address. During the process of a jump instruction, the CPU replaces the contents of the program counter with the destination address of the jump.

A special kind of jump occurs when the program "branches" to a subroutine. This kind of jump requires that the CPU remember the contents of the program counter before

the jump occurs. This process enables the CPU to resume execution of the main program when the last instruction in the subroutine has been executed.

A subroutine is a program within a program. It is usually a set of instructions which must be executed repeatedly within a program. Routines which calculate the square, the sine, or the logarithm of a variable are good examples of subroutines. Other examples might be programs designed for inputting or outputting data to a peripheral device.

The processor handles subroutines in a special way. When the processor receives a jump to subroutine instruction, it increments the program counter and stores the result in a memory known as the stack. The processor then stores the address specified in the jump to subroutine instruction in its program counter. Therefore, the next instruction executed will be the first instruction of the subroutine.

The last instruction in a subroutine will be a branch back instruction. When the processor receives a branch back instruction, it replaces the contents of the program counter with the address of the top of the stack.

Subroutines are often nested; that is, one subroutine will sometimes call a second subroutine. This is an acceptable procedure, as long as the CPU has enough capacity to store the return addresses, and the logical provision for implementation. Therefore, the maximum level of subroutine nesting is determined by the depth of the stack. If the stack has space for storing seven return addresses, then seven levels of subroutine

nesting may be accomodated.

Every computer has a word length. A computer's word length is determined by the size of its internal storage elements and data busses. A computer whose registers and busses can store and transfer 8 bits of information, has a word length of 8 bits. The characteritic 8 bit field is usually referred to as a byte. A 4 bit field is referred to as a nibble.

Each operation a computer can perform is specified by a unique binary code. An 8 bit word used as an instruction can refer to a maximum of 256 alternative actions. This is more than adequate for most processors. The eight bits stored in the instruction register selectively activate one of a number of output lines. In this case a maximum of 256 output lines. Each line represents a set of activities associated with execution of a particular instruction code. Timing pulses develop electrical signals which are used to initiate specific actions. The translation of binary code into a specific action is performed by the instruction decoder and associated control circuitry.

A CPU may use a register or a register pair to store the address of a memory location. If the address register(s) is programmable, then the programmer can change the contents of the register prior to execution of a memory reference instruction.

All processors contain an arithmetic/logic unit. The ALU must contain an adder capable of combining the contents of two registers in accordance with the rules of binary

arithmetic. This capability allows the processor to perform arithmetic manipulations on data it obtains from memory and other inputs.

Using only the basic adder, a programmer can write routines which will subtract, multiply, and divide. This gives the machine complete arithmetic capabilities. In practice, however, most ALUs provide other built-in functions, including hardware subtraction, boolean logic operations, and shift operations.

The ALU usually contains flag bits which register certain conditions which arise during arithmetic manipulations. Generally, it is possible to program jumps which are conditionally dependent upon one or more flag bits. Thus, for example, the program may be designed to jump to a special routine, if the carry bit is zero. The presence of a carry generally indicates an overflow in the accumulator.

The control circuitry is the primary functional unit within a CPU. The control circuitry maintains the proper sequence of events required for any processing task. Some processors have control circuitry capable of responding to external signals, such as an interrupt request. An interrupt request causes the control circuitry to temporarily interrupt main program execution, jump to a special routine to service the interrupting device, then automatically return to the main program.

III. THE INTEL 4040 CENTRAL PROCESSING UNIT

The Intel 4040 is a single chip 4 bit parallel metal oxide semiconductor (MOS) central processor. The 4040 is packaged in a 24 pin dual inline package (DIP). The pin configuration is shown in Figure 1. A brief functional description of each pin is given in Figure 2.

Circuit timing for all clocked CPUs is critical; the 4040 is not an exception. Figure 3 shows a timing diagram for the 4040. Circuit timing is accomplished by a two phase non-overlapping clock. The start of an instruction cycle is signaled by a SYNC signal, which is generated by the processor. The SYNC signal is sent to the various read only memory (ROM), and random access memory (RAM), and peripheral chips in the system. An instruction cycle consists of the following operations:

1. The 12 bit contents of the program counter is sent out to the ROM chips in three 4 bit groups during A_1, A_2, A_3.

2. The 8 bit instruction or data from the addressed ROM location is received by the processor at M_1, and M_2 at which time the instruction is decoded.

3. Instruction execution occurs during X_1, X_2, and X_3. Data or address information may be sent to output ports or RAM chips; data may be received from input ports or RAM chips; or data may be operated on within the processor.[6]

Figure 4 is a block diagram of the 4040 indicating the major circuit blocks and their interconnections. The following major functional blocks are contained in the 4040:

[6]Raphael, Howard A., MCS-40, p. 1-6

4040 – Central Processor Unit

- Instructions (60 total) including Logical Operations and Read Program Memory
- Large number of family devices
- 10.8 microsecond instruction cycle standard
- 2-phase dynamic operation
- Instruction set includes conditional branching, jump to subroutine and indirect fetching
- Logical instructions
- Binary and decimal arithmetic modes
- CPU directly compatible with MCS-4 ROMs and RAMs
- Unlimited number of input and output lines
- Interrupt capability
- Single step operation
- 8K byte memory addressing capability and up to 5120 bits of RAM
- 24 index registers
- Subroutine nesting to 7 levels

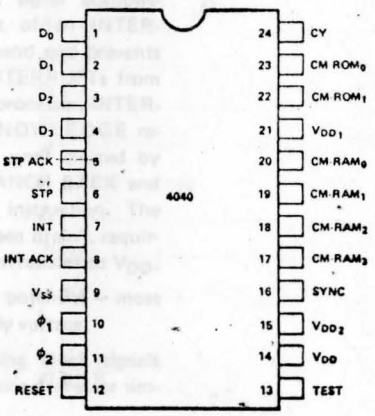
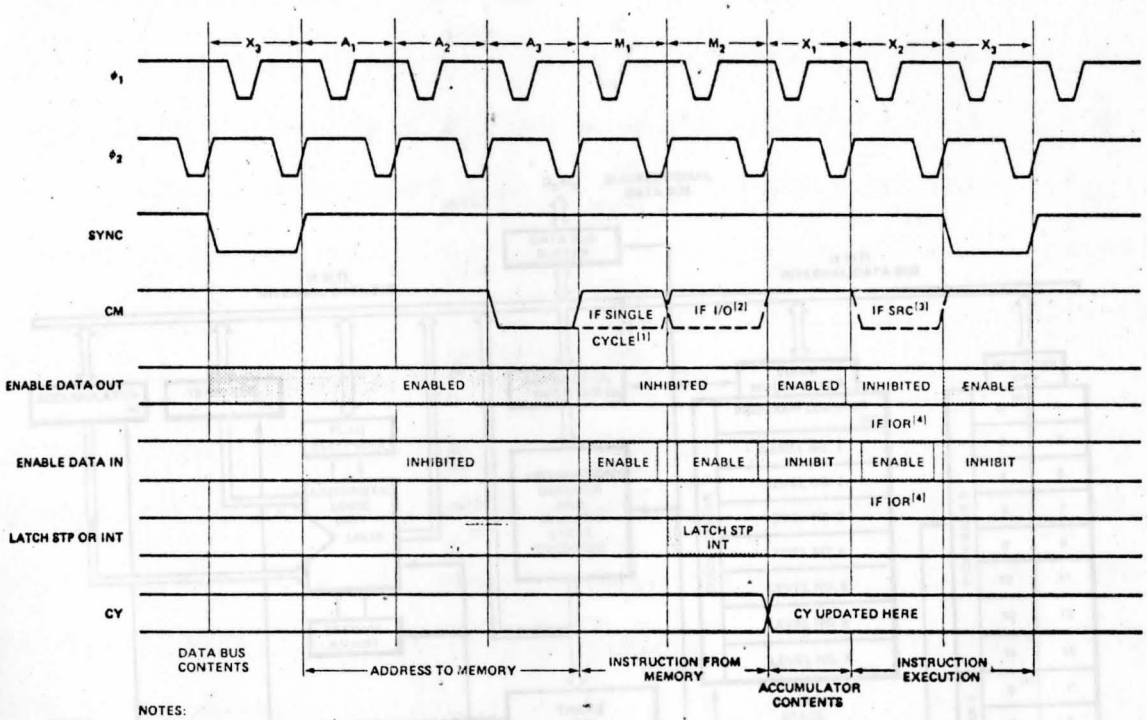


FIGURE 1. 4040 PIN CONFIGURATION

Pin Description

Pin No.	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	STPA	STOP ACKNOWLEDGE output. This signal acknowledges that the processor has entered the stop mode. Output is "open drain" requiring pull-down resistor to V_{DD}.
6	STP	STOP input signal. A logic "1" level at this input causes the processor to enter the STOP mode.
7	INT	INTERRUPT input signal. A logic "1" level at this input causes the processor to enter the INTERRUPT mode.
8	INTA	INTERRUPT ACKNOWLEDGE output. This signal acknowledges receipt of an INTERRUPT command and prevents additional INTERRUPTs from entering the processor. INTERRUPT ACKNOWLEDGE remains active until cleared by the new BRANCH BACK and SRC (BBS) instruction. The output is "open drain", requiring a pull-down resistor to V_{DD}.
9	V_{SS}	Circuit GND potential – most positive supply voltage.
10-11	ϕ_1-ϕ_2	Non-overlapping clock signals which determine processor timing.
12	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 96 clock cycles(12 machine cycles).
13	TEST	TEST input. The logical state of this input can be examined with the JCN instruction.
14	V_{DD}	Main supply voltage to the processor. Value must be V_{SS} -15.0V ±5%.
15	V_{DD2}	Supply voltage for output buffers. May be varied depending on interface conditions.
16	SYNC	SYNC output. Synchronization signal generated by the processor and sent to ROM and RAM chips. Indicates beginning of instruction cycle.
17-20	CM-RAM$_0$-CM-RAM$_3$	CM-RAM outputs. These outputs act as bank select signals for the 4002 RAM chips in the system.
21	V_{DD1}	Supply voltage for timing circuit. Value must be V_{SS} -15.0V ±5%. Allows low power standby operation. Only SYNC will be generated when this pin is the only active V_{DD}.
22-23	CM-ROM$_0$-CM-ROM$_1$	CM-ROM outputs. These outputs act as bank select signals for the ROM chips in the system.
24	CY	CARRY output buffer. The state of the CY flip-flop is presented at this output and is updated at X_1. The output is "open drain" requiring a pull-down resistor to V_{DD}.

FIGURE 2. 4040 FUNCTIONAL PIN DESCRIPTION



NOTES:

1. CM-ROM, RAM SIGNALS WILL BE PRESENT AT M_1 FOR ANY SINGLE CYCLE INSTRUCTION OR FOR THE FIRST CYCLE OF A DOUBLE CYCLE INSTRUCTION.
2. CM-ROM, RAM SIGNALS WILL BE PRESENT AT M_2 FOR ANY OF THE SIXTEEN I/O GROUP INSTRUCTIONS.
3. CM-ROM, RAM SIGNALS WILL BE PRESENT AT X_2 DURING EXECUTION OF AN SRC INSTRUCTION.
4. IOR MEANS ONE OF THE I/O READ INSTRUCTIONS: SBM, ROM, RDR, ADM, RDϕ, RD1, RD2, RD3.

FIGURE 3. 4040 CIRCUIT TIMING

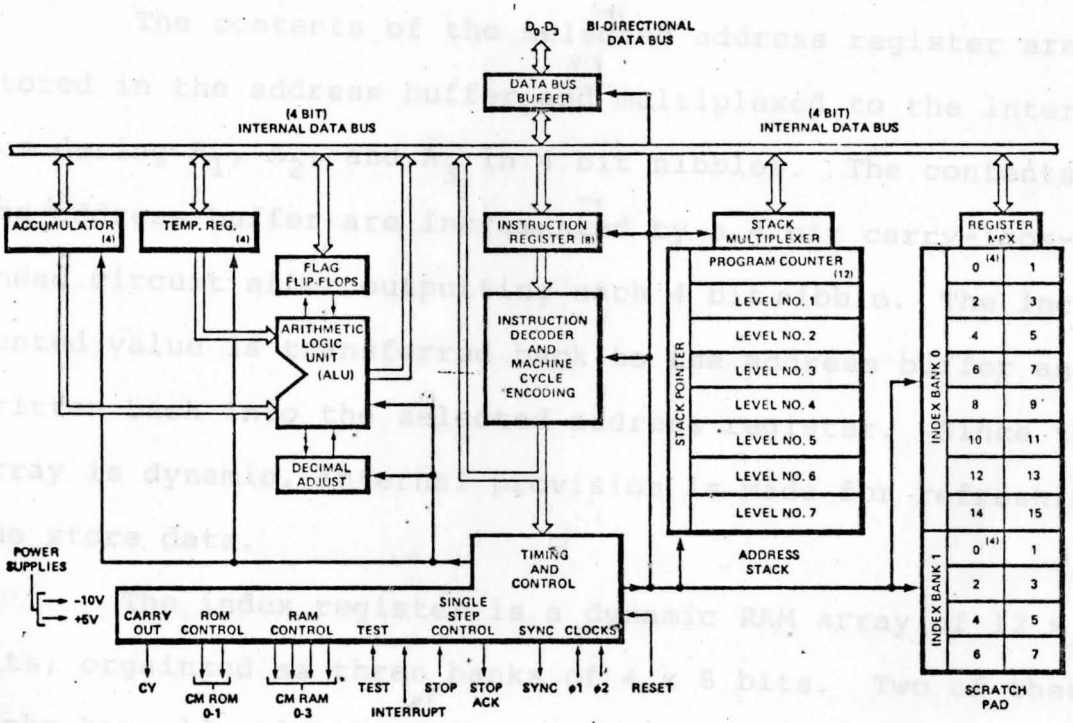


FIGURE 4. 4040 BLOCK DIAGRAM

1. Address register stack and address incremeter.
2. Index register array.
3. Instruction register/decoder and control logic.
4. 4 bit adder/accumulator.
5. Hardware interrupt and stop control.
6. Peripheral circuits for controlling timing and external communication.[7]

The address register is a dynamic RAM array of 8 x 12 bits, operating as a push-down stack. One level of this array is used to store the effective address. This leaves seven levels available for subroutine nesting.

The contents of the selected address register are stored in the address buffer and multiplexed to the internal bus during A_1, A_2, and A_3 in 4 bit nibbles. The contents of the address buffer are incremented by a 4 bit carry-look-ahead circuit after outputting each 4 bit nibble. The incremented value is transferred back to the address buffer and written back into the selected address register. Since the array is dynamic, internal provision is made for refreshing the store data.

The index register is a dynamic RAM array of 12 x 8 bits, orgainzed as three banks of 4 x 8 bits. Two of these banks have identical address locations, so these banks must be individually selected with program instructions. The third bank has unique addresses; therefore it is always available for use.

The index registers can be used two different ways. They can be used to store 4 bits data for computation. They can also be used in pairs for addressing ROM, RAM, and I/O

[7]Raphael, Howard A., MCS-40, p. 1-7.

ports or for storing fetched data from ROM.

Index register addressing is provided by the internal data bus. The addresses are multiplexed to the array decoder. The content of a selected index register is stored in a temporary register and multiplexed to the internal bus.

The 4 bit adder used in the 4040 is the ripple-through carry type. The adder buffer/register communicates with the internal data bus on one side and can transfer the data or one's complemented data to the adder. The other term of addition comes from the accumulator and carry flip-flop. The output of the adder is transferred to the accumulator and carry flip-flop. The accumulator has the capability to implement shift-right and shift-left instructions. The accumulator can communicate with the command register, with special ROMs, with the condition logic, and with the internal bus. The command register contains a 3 bit code used for CM-RAM line switching and one bit used for CM-ROM switching. The special ROMs communicate with the internal bus. The condition logic senses when an addition yields a zero result, when the accumulator is zero, the state of the carry flip-flop, and the state of an external signal (TEST). These conditions can be used to implement jump-on-condition and increment and skip if zero instructions.

The instruction register is loaded with the content of the internal bus at M_1 and M_2. The instructions are decoded in the instruction decoder and gated with timing signal to provide the control signals for various functional blocks.

The 4040 has interrupt and stop controls which override normal processor operation. The interrupt logic detects and acknowledges presence of an external interrupt signal and forces the processor to execute a jump-to-subroutine to location 003 hexadecimal.

The stop control logic detects and acknowledges the presence of a stop signal. The processor is forced to execute a no-operation instruction until the stop signal is removed.

The CPU command lines (CM-ROM, CM-RAM) are used to control the ROMs and RAMs by indicating how to interpret the data bus content at any given time.

The command lines allow the implementation of RAM bank, chip, register, and character addressing. The command lines also control ROM chip addressing. Operation of the command control lines is seen in Figure 5.

The 4040 has a set of sixty instructions. The instructions can by divided into four groups as follows:

1. Machine Instructions - This group of 16 instructions are designated by an OPR code of 0000-1101. Within this group is contained a second group which is designated the supplemental group.

2. 4040 Group - This group of 14 instructions is designated by an OPR code of 0000 and an OPA code of 0001-1110. These are the new instructions which have been added to the 4040.

3. I/O Group - This group is designated by an OPR code of 1110. This group of 16 instructions is used for transferring data between the processor and the RAM chips or I/O circuits.

4. Accumulator Group - This group of 14 instructions is designated by an OPR code of 1111 and operates only on the accumulator/carry flip-flop, the special ROMS and the

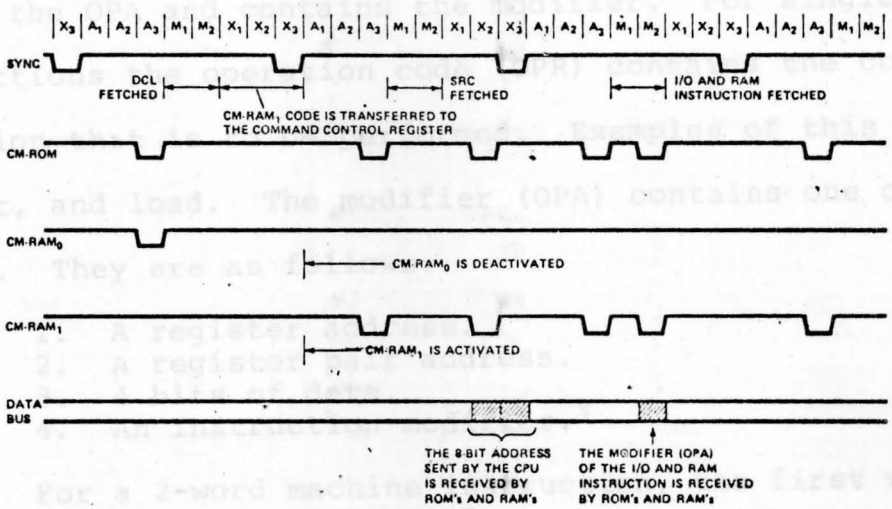


FIGURE 5. OPERATION OF THE COMMAND CONTROL LINES

command register.[8]

There are 2 types of instructions; they are 1 word instructions and 2 word instructions. A 1 word instruction is 8 bits wide and requires 8 clock periods (1 instruction cycle). A 2 word instruction is 16 bits wide and requires 16 clock periods (2 instruction cycles). Each instruction word is divided into 4 bit nibbles. The upper 4 bits is called the OPR and contains the operation code. The lower 4 bits is called the OPA and contains the modifier. For single word instructions the operation code (OPR) contains the code of the operation that is to be performed. Examples of this are add, subract, and load. The modifier (OPA) contains one of 4 things. They are as follows:

1. A register address.
2. A register pair address.
3. 4 bits of data.
4. An instruction modifier.[9]

For a 2-word machine instruction, the first word is similar to that of the 1-word instruction, however, the modifier contains one of 4 things. They are as follows:

1. A register address.
2. A register pair address.
3. The upper portion of another ROM address.
4. A condition for jumping.[10]

The second word contains either the middle portion (in OPR) and the lower portion (in OPA) of another ROM address or 8 bits of data. Instruction formats are shown in Figure 6.

[8]Raphael, Howard A., MCS-40, p. 1-18.
[9]Raphael, Howard A., MCS-40, p. 1-18.
[10]Raphael, Howard A., MCS-40, p. 1-18.

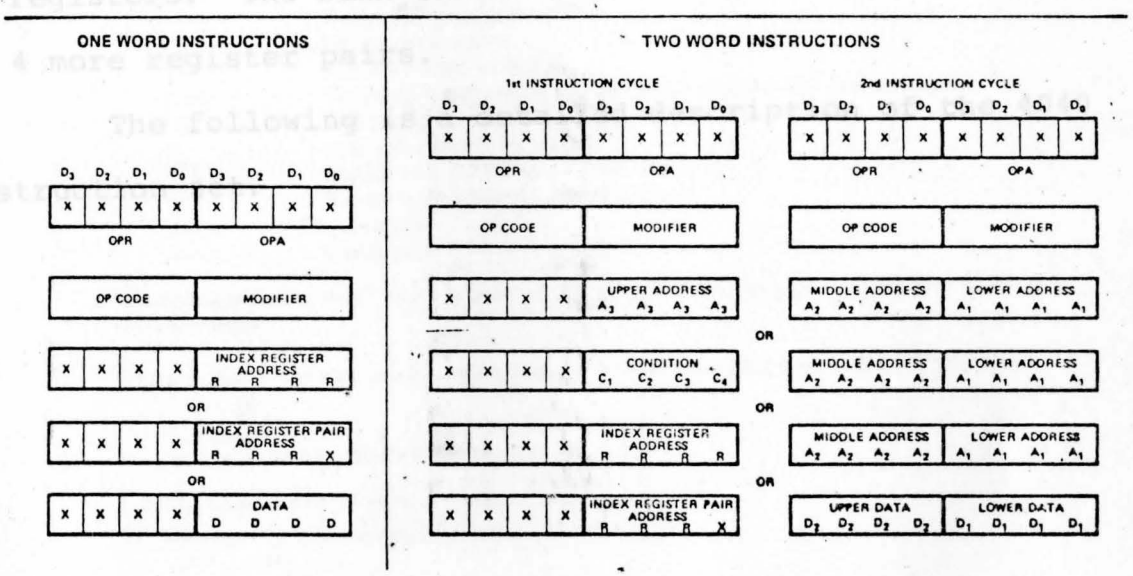


FIGURE 6. 4040 INSTRUCTION FORMATS

The upper 4 bits of an instruction will always be fetched before the lower 4 bits of instruction.

Index registers can be addressed in two ways. The OPA code of an instruction may specify 1 of 16 possible locations. The bank switch of the 4040 will allow access to 8 more registers. The second way to access registers is by specifying a pair of registers with the higher order 3 bits of the OPA code. This will allow direct addressing of 8 pairs of registers. The bank switch of the 4040 will allow access to 4 more register pairs.

The following is a detailed description of the 4040 instruction set.

DETAILED INSTRUCTION DESCRIPTION

A. Symbols and Abbreviations

The following symbols and abbreviations will be used throughout the next few sections:

SRCR	SRC Register
( )	the content of is transferred to
ACC	Accumulator (4 bit)
CY	Carry Flip-Flop
ACBR	Accumulator Buffer Register (4 bit)
RRRR	Index register address
RRR	Index register pair address
P_L	Low order program counter Field (4 bit)
P_M	Middle order program counter Field (4 bit)
P_H	High order program counter Field (4 bit)
a_i	Order i content of the accumulator
CM_i	Order i content of the command register
M	RAM main character location
M_{si}	RAM status character i
DB (T)	Data bus content at time T
Stack	The 3 or 7 registers in the address register other than the program counter.
CR	Command register
IE	Interrupt enable
RB0	Register bank 0 $RRRR_0$ – RRRR, enable
RB1	Register bank 1 $RRRR_0$ – RRRR, enable
V	Logical OR
Λ	Logical AND

Throughout the text "page" means a block of 256 instructions whose address differs only on the most significant 4 bits.

Example: page 7 means all locations having addresses between 0111 0000 0000 and 0111 1111 1111

B. Format for Describing Each Instruction

Each instruction will be described as follows:

(1) Mnemonic symbol and meaning
(2) OPR and OPA code
(3) Symbolic representation of the instruction
(4) Description of the instruction (if necessary)
(5) Example and/or exceptions (if necessary)

C. One Word Machine Instructions

Mnemonic: NOP (No Operation)
OPR OPA: 0000 0000
Symbolic: Not applicable
Description: No operation performed

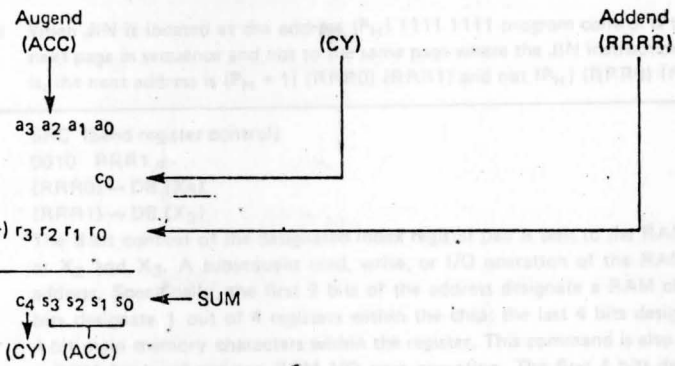
Mnemonic: LDM (Load Data to Accumulator)
OPR OPA: 1101 DDDD
Symbolic: DDDD → ACC
Description: The 4 bits of data, DDDD stored in the OPA field of instruction word are loaded into the accumulator. The previous contents of the accumulator are lost. The carry/link bit is unaffected.

Mnemonic: LD (Load index register to Accumulator)
OPR OPA: 1010 RRRR
Symbolic: (RRRR) → ACC
Description: The 4 bit content of the designated index register (RRRR) is loaded into the accumulator. The previous contents of the accumulator are lost. The 4 bit content of the index register and the carry/link bit are unaffected.

Mnemonic: XCH (Exchange index register and accumulator)
OPR OPA: 1011 RRRR
Symbolic: (ACC) → ACBR, (RRRR) → ACC, (ACBR) → RRRR
Description: The 4 bit content of the designated index register is loaded into the accumulator. The prior content of the accumulator is loaded into the designated register. The carry/link bit is unaffected.

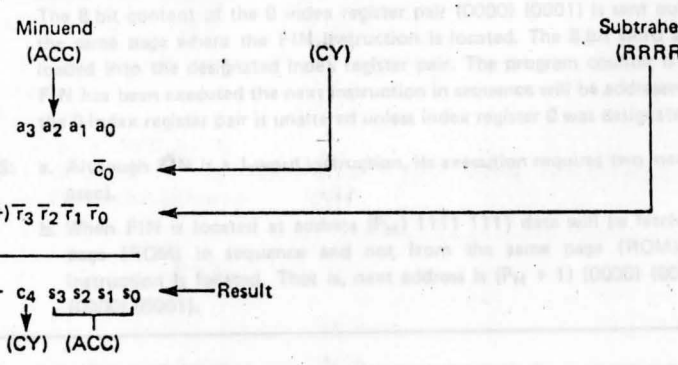
Mnemonic: ADD (Add index register to accumulator with carry)
OPR OPA: 1000 RRRR
Symbolic: (RRRR) + (ACC) + (CY) → ACC, CY
Description: The 4 bit content of the designated index register is added to the content of the accumulator with carry. The result is stored in the accumulator. The carry/link is set to 1 if a sum greater than 15_{10} was generated to indicate a carry out; otherwise, the carry/link is set to 0. The 4 bit content of the index register is unaffected.

Example:



Mnemonic: SUB (Subtract index register from accumulator with borrow)
OPR OPA: 1001 RRRR
Symbolic: (ACC) + (RRRR) + (CY) → ACC, CY
Description: The 4 bit content of the designated index register is complemented (ones complement) and added to content of the accumulator with borrow and the result is stored in the accumulator. If a borrow is generated, the carry bit is set to 0; otherwise, it is set to 1. The 4 bit content of the index register is unaffected.

Example:



Mnemonic: INC (Increment index register)
OPR OPA: 0110 RRRR
Symbolic: (RRRR) +1 → RRRR
Description: The 4 bit content of the designated index register is incremented by 1. The index register is set to zero in case of overflow. The carry/link is unaffected.

Mnemonic: BBL (Branch back and load data to the accumulator)
OPR OPA: 1100 DDDD
Symbolic: (Stack) → P_L, P_M, P_H; DDDD → ACC
Description: The program counter (address stack) is pushed down one level. Program control transfers to the next instruction following the last jump to subroutine (JMS) instruction. The 4 bits of data DDDD stored in the OPA portion of the instruction are loaded to the accumulator. BBL is used to return from subroutine to main program.

Mnemonic: JIN (Jump indirect)
OPR OPA: 0011 RRR1
Symbolic: (RRR0) → P_M
(RRR1) → P_L; P_H unchanged
Description: The 8 bit content of the designated index register pair is loaded into the low order 8 positions of the program counter. Program control is transferred to the instruction at that address on the same page (same ROM) where the JIN instruction is located. The 8 bit content of the index register is unaffected.

EXCEPTIONS: When JIN is located at the address (P_H) 1111 1111 program control is transferred to the next page in sequence and not to the same page where the JIN instruction is located. That is, the next address is (P_H + 1) (RRR0) (RRR1) and not (P_H) (RRR0) (RRR1)

Mnemonic: SRC (Send register control)
OPR OPA: 0010 RRR1
Symbolic: (RRR0) → DB (X_2)
(RRR1) → DB (X_3)
Description: The 8 bit content of the designated index register pair is sent to the RAM address register at X_2 and X_3. A subsequent read, write, or I/O operation of the RAM will utilize this address. Specifically, the first 2 bits of the address designate a RAM chip; the second 2 bits designate 1 out of 4 registers within the chip; the last 4 bits designate 1 out of 16 4 bit main memory characters within the register. This command is also used to designate a ROM for a subsequent ROM I/O port operation. The first 4 bits designate the ROM chip number to be selected. The address in ROM or RAM is not cleared until the next SRC instruction is executed. The 8 bit content of the index register is unaffected.

Mnemonic: FIN (Fetch indirect from ROM)
OPR OPA: 0011 RRR0
Symbolic: (P_H) (0000) (0001) → ROM address
(OPR) → RRR0
(OPA) → RRR1
Description: The 8 bit content of the 0 index register pair (0000) (0001) is sent out as an address in the same page where the FIN instruction is located. The 8 bit word at that location is loaded into the designated index register pair. The program counter is unaffected; after FIN has been executed the next instruction in sequence will be addressed. The content of the 0 index register pair is unaltered unless index register 0 was designated.

EXCEPTIONS:
a. Although FIN is a 1-word instruction, its execution requires two memory cycles (21.6 μsec).
b. When FIN is located at address (P_H) 1111 1111 data will be fetched from the next page (ROM) in sequence and not from the same page (ROM) where the FIN instruction is located. That is, next address is (P_H + 1) (0000) (0001) and not (P_H) (0000) (0001).

Mnemonic: HLT
OPR OPA: 0000 0001
Symbolic: 1 → HALT 1 → STOP
Description: The processor sets the HALT and STOP flip-flops. Program counter incrementer and data input buffers are inhibited. The processor executes NOP continuously; continuation can occur by means of STOP or INTERRUPT control.

In this mode, the Program Counter + 1 is gated out at A_1, A_2, and A_3, times on the data bus. M_1, M_2 times will contain the addressed ROM instruction on the data bus. X_1, the 4 bit Accumulator contents, X_2 and X_3 will contain the 8 bit SRC register.

Mnemonic: BBS
OPR OPA: 0000 0010
Symbolic: (Stack → P_L, P_M, P_H;)
SRCR0 → DB(X2)
SRCR1 → DB(X3)
Description: This instruction is a combination of BRANCH BACK and SRC. The effective address counter is decremented and program control is returned to the location saved by the forced JMS which occurred at the beginning of the interrupt routine. In addition, the content of the SRC register is sent out at X_2 and X_3 of the instruction cycle, thus restoring the I/O port selection. This instruction will also turn off the INTA line re-enabling the CPU for Interrupt.

The previously selected Index register bank will also be restored during this instruction.

Mnemonic: LCR
OPR OPA: 0000 0011
Symbolic: (CR) → ACC
Description: The 4 bit contents of the COMMAND REGISTER are transferred to the ACCUMULATOR. This allows saving the command register values before processing the interrupt.

Mnemonic: OR4
OPR OPA: 0000 0100
Symbolic: ($RRRR_4$) V (ACC) → ACC
Description: The 4 bit contents of index register #4 are logically "OR-ed" with the ACCUMULATOR. The result is placed in the ACCUMULATOR and the CARRY flip-flop is unaffected.

Examples:

(ACC)	0101
($RRRR_4$)	1001
ACC	1101

(ACC)	0000
($RRRR_4$)	1000
ACC	1000

Mnemonic: OR5
OPR OPA: 0000 0101
Symbolic: ($RRRR_5$) V (ACC) → ACC
Description: The 4 bit contents of index register #5 are logically "OR-ed" with the ACCUMULATOR. Carry flip-flop is unaffected.

Mnemonic: AN6
OPR OPA: 0000 0110
Symbolic: $(RRRR_6) \wedge (ACC) \rightarrow ACC$
Description: The 4 bit contents of index register #6 are logically "AND-ed" with the ACCUMULATOR. The result is placed in the ACCUMULATOR and the CARRY is unaffected.

Examples:

(ACC)	0110
$(RRRR_6)$	0100
ACC	0100
(ACC)	1111
$(RRRR_6)$	0001
ACC	0001

Mnemonic: AN7
OPR OPA: 0000 0111
Symbolic: $(RRRR_7) \wedge (ACC) \rightarrow ACC$
Description: The 4 bit contents of index register #7 are logically "AND-ed" with the ACCUMULATOR. Carry flip-flop is unaffected.

Mnemonic: DB0
OPR OPA: 0000 1000
Symbolic: Enable $\rightarrow$ CM-ROM_0
Description: DESIGNATE ROM BANK 0. The most significant bit of the COMMAND REGISTER, CR_3, is reset. On the third instruction cycle following its execution, it causes CM-ROM_0 to be activated. This Bank is selected with reset.

Mnemonic: DB1
OPR OPA: 0000 1001
Symbolic: Enable $\rightarrow$ CM-ROM_1
Description: DESIGNATE ROM BANK 1. The most significant bit of the COMMAND REGISTER, CR_3, is set. On the third instruction cycle following its execution, it causes CM-ROM_1 to be activated.

Mnemonic: SB0
OPR OPA: 0000 1010
Symbolic: $1 \rightarrow RB0$, $0 \rightarrow RB1$
Description: SELECT INDEX REGISTER BANK 0. The index register bank select flip-flop is reset. Index registers 0 - 7, 8 - 15 will be available for program use. This bank is to be selected with a Reset.

Mnemonic: SB1
OPR OPA: 0000 1011
Symbolic: $0 \rightarrow RB0$ $1 \rightarrow RB1$
Description: SELECT INDEX REGISTER BANK 1. The index register bank select flip-flop is set. Index registers 0* - 7*, 8 - 15 will be available for program use.

Mnemonic: RPM
OPR OPA: 0000 1110
Symbolic: (1111) (SRC) $\rightarrow$ ROM/RAM address
(DDDD) $\rightarrow$ ACC
Description: READ PROGRAM MEMORY. This instruction can be used only with the 4289 Standard Memory and I/O Interface Chip. The contents of the previously selected nibble of R/W Program Memory are transferred to the 4040 and loaded to the ACCUMULATOR.

Mnemonic: EIN
OPR OPA: 0000 1100
Symbolic: 1 → IE
Description: ENABLE INTERRUPT. Internal interrupt detection logic is enabled.

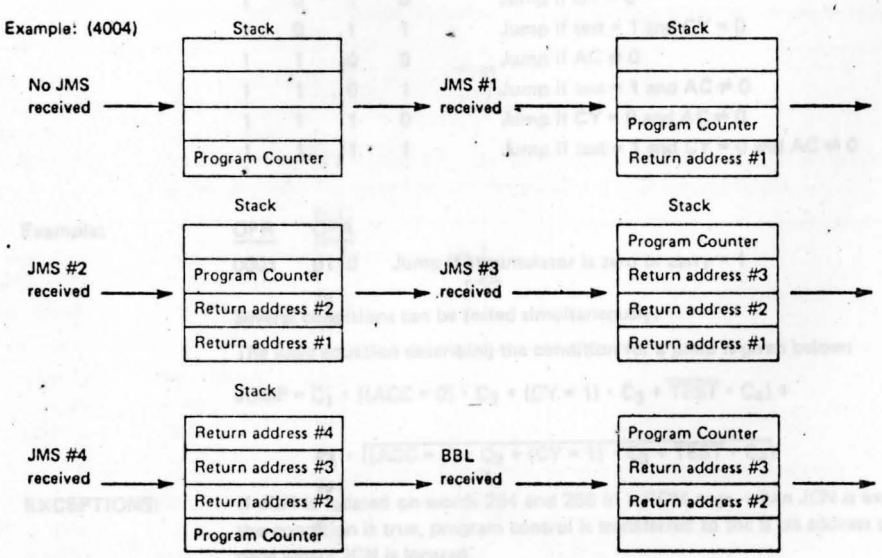
Mnemonic: DIN
OPR OPA: 0000 1101
Symbolic: 0 IE
Description: DISABLE INTERRUPT. Internal interrupt detection logic is disabled.

D. Two Word Machine Instruction

Mnemonic: JUN (Jump unconditional)
1st word OPR OPA: 0100 A_3 A_3 A_3 A_3
2nd word OPR OPA: A_2 A_2 A_2 A_2 A_1 A_1 A_1 A_1
Symbolic: A_1 A_1 A_1 A_1 → P_L, A_2 A_2 A_2 A_2 → P_M, A_3 A_3 A_3 A_3 → P_H
Description: Program control is unconditionally transferred to the instruction locator at the address A_3 A_3 A_3 A_3, A_2 A_2 A_2 A_2, A_1 A_1 A_1 A_1

Mnemonic: JMS (Jump to Subroutine)
1st word OPR OPA: 0101 A_3 A_3 A_3 A_3
2nd word OPR OPA: A_2 A_2 A_2 A_2 A_1 A_1 A_1
Symbolic: $(P_H, P_M, P_L + 2)$ → Stack
A_1 A_1 A_1 A_1 → P_L, A_2 A_2 A_2 A_2 → P_M, A_3 A_3 A_3 A_3 → P_H
Description: The address of the next instruction in sequence following JMS (return address) is saved in the push down stack. Program control is transferred to the instruction located at the 12 bit address ($A_3A_3A_3A_3A_2A_2A_2A_2A_1A_1A_1A_1$). Execution of a return instruction (BBL) will cause the saved address to be pulled out of the stack, therefore, program control is transferred to the next sequential instruction after the last JMS.

The push down stack has 4 registers (8 registers in 4040). One of them is used as the program counter, therefore nesting of JMS can occur up to 3 levels (7 levels in the 4040).



The deepest return address is lost.

Mnemonic: JCN (Jump conditional)

1st word OPR OPA: 0001 $C_1\ C_2\ C_3\ C_4$

2nd word OPR OPA: $A_2\ A_2\ A_2\ A_2$ $A_1\ A_1\ A_1\ A_1$

Symbolic: If $C_1\ C_2\ C_3\ C_4$ is true, $A_2\ A_2\ A_2\ A_2 \rightarrow P_M$
$A_1\ A_1\ A_1\ A_1 \rightarrow P_L$, P_H unchanged
if $C_1\ C_2\ C_3\ C_4$ is false,
$(P_H) \rightarrow P_H$, $(P_M) \rightarrow P_M$, $(P_L + 2) \rightarrow P_L$

Description: If the designated condition code is true, program control is transferred to the instruction located at the 8 bit address $A_2\ A_2\ A_2\ A_2$, $A_1\ A_1\ A_1\ A_1$ on the same page (ROM) where JCN is located.

If the condition is not true the next instruction in sequence after JCN is executed.

The condition bits are assigned as follows:

$C_1 = 0$ Do not invert jump condition
$C_1 = 1$ Invert jump condition
$C_2 = 1$ Jump if the accumulator content is zero
$C_3 = 1$ Jump if the carry/link content is 1
$C_4 = 1$ Jump if test signal (pin 10 on 4004) is zero.

C_X Condition Table for JCN Instruction

C_1	C_2	C_3	C_4	
0	0	0	0	NO OPERATION
0	0	0	1	Jump if test = 0 (High)
0	0	1	0	Jump if CY = 1
0	0	1	1	Jump if test = 0 or CY = 1
0	1	0	0	Jump if AC = 0
0	1	0	1	Jump if test = 0 or AC = 0
0	1	1	0	Jump if CY = 1 or AC = 0
0	1	1	1	Jump if test = 0 or CY = 1 or AC = 0
1	0	0	0	Jump Unconditionally
1	0	0	1	Jump if test = 1 (Low)
1	0	1	0	Jump if CY = 0
1	0	1	1	Jump if test = 1 and CY = 0
1	1	0	0	Jump if AC ≠ 0
1	1	0	1	Jump if test = 1 and AC ≠ 0
1	1	1	0	Jump if CY = 0 and AC ≠ 0
1	1	1	1	Jump if test = 1 and CY = 0 and AC ≠ 0

Example:

OPR	OPA	
0001	0110	Jump if accumulator is zero or carry = 1

Several conditions can be tested simultaneously.

The logic equation describing the condition for a jump is given below:

$$JUMP = \overline{C_1} \cdot ((ACC = 0) \cdot C_2 + (CY = 1) \cdot C_3 + \overline{TEST} \cdot C_4) +$$

$$C_1 \cdot \overline{((ACC = 0) \cdot C_2 + (CY = 1) \cdot C_3 + \overline{TEST} \cdot C_4)}$$

EXCEPTIONS: If JCN is located on words 254 and 255 of a ROM page, when JCN is executed and the condition is true, program control is transferred to the 8 bit address on the next page where JCN is located.

Mnemonic: ISZ (Increment index register skip if zero)
1st word OPR OPA: 0111 RRRR
2nd word OPR OPA: $A_2\ A_2\ A_2\ A_2$ $A_1\ A_1\ A_1\ A_1$
Symbolic: (RRRR) + 1 → RRRR, if result = 0
$(P_H) \rightarrow P_H$, $(P_M) \rightarrow P_M$, $(P_L + 2) \rightarrow P_L$:
if result ≠ 0 $(P_H) \rightarrow P_H$,
$A_2\ A_2\ A_2\ A_2 \rightarrow P_M$, $A_1\ A_1\ A_1\ A_1 \rightarrow P_L$
Description: The content of the designated index register is incremented by 1. The accumulator and carry/link are unaffected. If the result is zero, the next instruction after ISZ is executed. If the result is different from 0, program control is transferred to the instruction located at the 8 bit address $A_2\ A_2\ A_2\ A_2$, $A_1\ A_1\ A_1\ A_1$ on the same page (ROM) where the ISZ instruction is located.

EXCEPTIONS: If ISZ is located on words 254 and 255 of a ROM page, when ISZ is executed and the result is not zero, program control is transferred to the 8 bit address located on the next page in sequence and not on the same page where ISZ is located.

Mnemonic: FIM (Fetched immediate from ROM)
1st word OPR OPA: 0010 RRR0
2nd word OPR OPA: $D_2\ D_2\ D_2\ D_2$ $D_1\ D_1\ D_1\ D_1$
Symbolic: $D_2\ D_2\ D_2\ D_2 \rightarrow$ RRR0
$D_1\ D_1\ D_1\ D_1 \rightarrow$ RRR1
Description: The 2nd word represents 8 bits of data which are loaded into the designated index register pair.

E. Input/Output Instructions

The following I/O instructions are described as they relate to ROM and RAM devices. These same instructions (mnemonics) can be redefined for devices other than ROM and RAM.

Mnemonic: RDM (Read RAM character)
OPR OPA: 1110 1001
Symbolic: (M) → ACC
Description: The content of the previously selected RAM main memory character is transferred to the accumulator. The carry/link is unaffected. The 4 bit data in memory is unaffected.

Mnemonic: RDO (Read RAM status character 0)
OPR OPA: 1110 1100
Symbolic: $(M_{S0}) \rightarrow$ ACC
Description: The 4 bits of status character 0 for the previously selected RAM register are transferred to the accumulator. The carry/link and the status character are unaffected.

Mnemonic: RD1 (Read RAM status character 1)
OPR OPA: 1110 1101
Symbolic: $(M_{S1}) \rightarrow$ ACC

Mnemonic: RD2 (Read RAM status character 2)
OPR OPA: 1110 1110
Symbolic: $(M_{S2}) \rightarrow$ ACC

Mnemonic: RD3 (Read RAM status character 3)
OPR OPA: 1110 1111
Symbolic: $(M_{S3}) \rightarrow$ ACC

Mnemonic: RDR (Read ROM port)
OPR OPA: 1110 1010
Symbolic: (ROM input lines) → ACC
Description: The data present at the input lines of the previously selected ROM chip is transferred to the accumulator. The carry/link is unaffected.

If the I/O option has both inputs and outputs within the same 4 I/O lines, the user can choose to have either "0" or "1" transferred to the accumulator for those I/O pins coded as outputs, when an RDR instruction is executed.

Example: Given a port with I/O coded with 2 inputs and 2 outputs, when RDR is executed the transfer is as shown below:

$I_3\ O_2\ O_1\ I_0$ (ACC)

1 X X 0 ⟹ 1 (1 or 0) (1 or 0) 0

Input Data — User can choose

Mnemonic: WRM (Write accumulator into RAM character)
OPR OPA: 1110 0000
Symbolic: (ACC) → M
Description: The accumulator content is written into the previously selected RAM main memory character location. The accumulator and carry/link are unaffected.

Mnemonic: WR0 (Write accumulator into RAM status character 0)
OPR OPA: 1110 0100
Symbolic: (ACC) → M_{S0}
Description: The content of the accumulator is written into the RAM status character 0 of the previously selected RAM register. The accumulator and the carry/link are unaffected.

Mnemonic: WR1 (Write accumulator into RAM status character 1)
OPR OPA: 1110 0101
Symbolic: (ACC) → M_{S1}

Mnemonic: WR2 (Write accumulator into RAM status character 2)
OPR OPA: 1110 0110
Symbolic: (ACC) → M_{S2}

Mnemonic: WR3 (Write accumulator into RAM status character 3)
OPR OPA: 1110 0111
Symbolic: (ACC) → M_{S3}

Mnemonic: WRR (Write ROM port)
OPR OPA: 1110 0010
Symbolic: (ACC) → ROM output lines
Description: The content of the accumulator is transferred to the ROM output port of the previously selected ROM chip. The data is available on the output pins until a new WRR is executed on the same chip. The ACC content and carry/link are unaffected. (The LSB bit of the accumulator appears on I/O_0.) No operation is performed on I/O lines coded as inputs.

Mnemonic: WMP (Write memory port)
OPR OPA: 1110 0001
Symbolic: (ACC) → RAM output register
Description: The content of the accumulator is transferred to the RAM output port of the previously selected RAM chip. The data is available on the output pins until a new WMP is executed on the same RAM chip. The content of the ACC and the carry/link are unaffected. (The LSB bit of the accumulator appears on O_0, Pin 16, of the 4002.)

Mnemonic: ADM (Add from memory with carry)
OPR OPA: 1110 1011
Symbolic: (M) + (ACC) + (CY) → ACC, CY
Description: The content of the previously selected RAM main memory character is added to the accumulator with carry. The RAM character is unaffected.

Mnemonic: SBM (Subtract from memory with borrow)
OPR OPA: 1110 1000
Symbolic: $(\overline{M}) + (ACC) + (\overline{CY}) \rightarrow ACC, CY$
Description: The content of the previously selected RAM character is subtracted from the accumulator with borrow. The RAM character is unaffected.

F. Accumulator Group Instructions

Mnemonic: CLB (Clear both)
OPR OPA: 1111 0000
Symbolic: 0 → ACC, 0 → CY
Description: Set accumulator and carry/link to 0.

Mnemonic: CLC (Clear carry)
OPR OPA: 1111 0001
Symbolic: 0 → CY
Description: Set carry/link to 0

Mnemonic: CMC (Complement carry)
OPR OPA: 1111 0011
Symbolic: $(\overline{CY}) \rightarrow CY$
Description: The carry/link content is complemented

Mnemonic: STC (Set carry)
OPR OPA: 1111 1010
Symbolic: 1 → CY
Description: Set carry/link to a 1

Mnemonic: CMA (Complement Accumulator)
OPR OPA: 1111 0100
Symbolic: $\overline{a_3}\ \overline{a_2}\ \overline{a_1}\ \overline{a_0} \rightarrow ACC$
Description: The content of the accumulator is complemented. The carry/link is unaffected.

Mnemonic: IAC (Increment accumulator)
OPR OPA: 1111 0010
Symbolic: (ACC) + 1 → ACC
Description: The content of the accumulator is incremented by 1. No overflow sets the carry/link to 0; overflow sets the carry/link to a 1.

Mnemonic: DAC (decrement accumulator)
OPR OPA: 1111 1000
Symbolic: (ACC) - 1 → ACC
Description: The content of the accumulator is decremented by 1. A borrow sets the carry/link to 0, no borrow sets the carry/link to a 1.

Example:

(ACC)

$a_3\ a_2\ a_1\ a_0$

+) 1 1 1 1

$C_4\ S_3\ S_2\ S_1\ S_0$

CY ACC

Mnemonic: RAL (Rotate left)
OPR OPA: 1111 0101
Symbolic: $C_0 \rightarrow a_0$, $a_i \rightarrow a_{i-1}$, $a_3 \rightarrow CY$
Description: The content of the accumulator and carry/link are rotated left.

Mnemonic: RAR (Rotate right)
OPR OPA: 1111 0110
Symbolic: $a_0 \rightarrow CY$, $a_i \rightarrow a_{i-1}$, $C_0 \rightarrow a_3$
Description: The content of the accumulator and carry/link are rotated right.

Mnemonic: TCC (Transmit carry and clear)
OPR OPA: 1111 0111
Symbolic: $0 \rightarrow ACC$, $(CY) \rightarrow a_0$, $0 \rightarrow CY$
Description: The accumulator is cleared. The least significant position of the accumulator is set to the value of the carry/link. The carry/link is set to 0.

Mnemonic: DAA (Decimal adjust accumulator)
OPR OPA: 1111 1011
Symbolic: (ACC) + 0000 → ACC
or
0110
Description: The accumulator is incremented by 6 if either the carry/link if 1 or if the accumulator content is greater than 9. The carry/link is set to a 1 if the result generates a carry, otherwise it is unaffected.

Mnemonic: TCS (Transfer carry subtract)
OPR OPA: 1111 1001
Symbolic: 1001 → ACC if (CY) = 0
1010 → ACC if (CY) = 1
0 → CY
Description: The accumulator is set to 9 if the carry/link is 0.
The accumulator is set to 10 if the carry/link is a 1.
The carry/link is set to 0.

Mnemonic: KBP (Keyboard process)
OPR OPA: 1111 1100
Symbolic: (ACC) → KBP ROM → ACC
Description: A code conversion is performed on the accumulator content, from 1 out of n to binary code. If the accumulator content has more than one bit on, the accumulator will be set to 15 (to indicate error). The carry/link is unaffected. The conversion table is shown below.

(ACC) before KBP	(ACC) after KBP
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 0
0 1 0 0	0 0 1 1
1 0 0 0	0 1 0 0
0 0 1 1	1 1 1 1
0 1 0 1	1 1 1 1
0 1 1 0	1 1 1 1
0 1 1 1	1 1 1 1
1 0 0 1	1 1 1 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 1
1 1 0 0	1 1 1 1
1 1 0 1	1 1 1 1
1 1 1 0	1 1 1 1
1 1 1 1	1 1 1 1

Mnemonic: DCL (Designate command line)
OPR OPA: 1111 1101
Symbolic: $a_0 \rightarrow CM_0$, $a_1 \rightarrow CM_1$, $a_2 \rightarrow CM_2$
Description: The content of the three least significant accumulator bits is transferred to the command control register within the CPU.

This instruction provides RAM bank selection when multiple RAM banks are used. (If no DCL instruction is sent out, RAM Bank number zero is automatically selected after application of at least one RESET). DCL remains latched until it is changed.

The selection is made according to the following truth table.

(ACC)	CM - RAM_i Enabled	Bank No.
X 0 0 0	CM - RAM_0	Bank 0
X 0 0 1	CM - RAM_1	Bank 1
X 0 1 0	CM - RAM_2	Bank 2
X 1 0 0	CM - RAM_3	Bank 3
X 0 1 1	CM - RAM_1, CM - RAM_2	Bank 4
X 1 0 1	CM - RAM_1, CM - RAM_3	Bank 5
X 1 1 0	CM - RAM_2, CM - RAM_3	Bank 6
X 1 1 1	CM - RAM_1, CM - RAM_2, CM - RAM_3	Bank 7

IV. MCS-40 SYSTEM COMPONENTS

A complete microcomputer is composed of several integrated circuits. A working knowledge of the microcomputer system is not possible unless a basic understanding of the function of the system components is achieved. The system components are the set of all integrated circuits designed to set system timing, store data, perform input/output, and perform all functions which are not contained within the CPU chip.

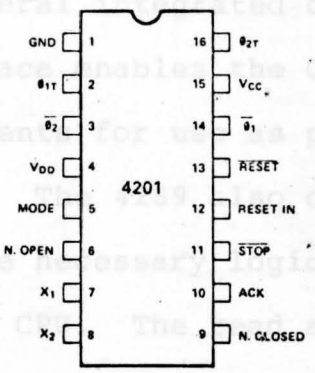
The first of these peripheral integrated circuits is the Intel 4201 clock generator. The 4201 is a complementary metal-oxide-semiconductor (CMOS) integrated circuit. The integrated circuit is designed to fill the clock requirements of the MCS-40[11] microcomputer set. The 4201 contains a crystal controlled oscillator (external), clock generation circuitry, and both two-phase MOS and transistor-transistor-logic (TTL) level clock driver circuits.

The 4201 also performs the reset function required by the MCS-40 components. It also provides the stop and single-step function of the 4040 central processing unit.

The 4201 is packaged in a single 16 pin DIP. The pin configuration and a functional description of each pin is seen in Figure 7.

[11]Raphael, Howard A., MCS-40, p. iii

12	RESET IN	Input to which RC network is connected to provide power-on reset timing.
13	$\overline{\text{RESET}}$	Reset signal output which directly connects to all MCS 40 reset inputs.
14	$\overline{\phi 1}$	Phase 1 MOS level clock output. Directly drives all MCS 40 clock inputs.
15	V_{SS}	Circuit reference potential — most positive supply voltage.
16	$\phi 2T$	Phase 1 TTL level clock output.



Pin Description

Pin No.	Designation	Description of Function
1	GND	Circuit ground potential. This pin can be left floating for low power application. MOS clock output will be operative, TTL clock outputs will not.
2	$\phi 1T$	Phase 1 TTL level clock output.
3	$\overline{\phi 2}$	Phase 2 MOS level clock output. Directly drives all MCS 40 components.
4	V_{DD}	Main Power Supply Pin. $V_{DD} = V_{SS} - 15V \pm 5\%$.
5	MODE	Counter mode control pin. Determines whether counter divides basic frequency by 8 or 7. Mode 1 = V_{SS} Mode 2 = V_{DD}
6	N. OPEN	Input of single step circuitry to which normally open contact of SPDT switch is connected.
7	X1	External Crystal Connection
8	X2	External Crystal Connection
9	N. CLOSED	Input of single step circuitry to which normally closed contact of SPDT switch is connected.
10	ACK	Acknowledge input to single step circuitry normally connected to stop acknowledge output of 4040.
11	$\overline{\text{STOP}}$	Stop output of single step circuitry normally connected to stop input of 4040.

FIGURE 7. 4201 CLOCK GENERATOR

Another peripheral integrated circuit is the Intel 4002-320 bit RAM and 4 bit output port. The 4002 performs two functions. As a RAM it stores 320 bits arranged in 4 registers of twenty 4 bit characters each (16 main memory characters and 4 status characters). As an output port, the 4002 is provided with 4 output lines and associated control logic to perform output operations. The 4002 is a p-channel MOS device and is compatible with all MCS-40 components.

The 4002 is packaged in a single 16 pin Dip. The pin configuration and a function description of each pin is shown in Figure 8. A block diagram of the 4002 is shown in Figure 9.

The Intel 4289 standard memory interface is another peripheral integrated circuit. The 4289 standard memory interface enables the CPU devices to utilize standard memory components for use as program data memory.

The 4289 also contains a 4 bit bi-directional I/0 bus and the necessary logic to multiplex a host of I/O sources to the CPU. The read and write program memory instructions implemented with the 4289, allow the user to store data and modify program memory. The device directly addresses 4K bytes of program memory. The address is obtained sequentially during A_1 - A_3 of the instruction cycle. The 8 bit instruction is presented to the CPU during M_1 and M_2 of the instruction cycle via the four bit data bus.

The 4289 is packaged in a single 40 pin DIP. The pin configuration and functional description of each pin is shown in Figure 10.

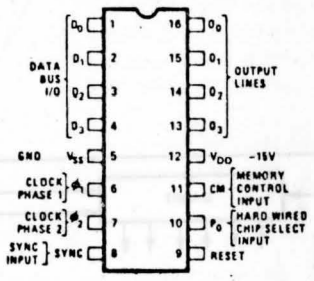


Figure 4-14. 4002 Pin Configuration.

Pin Description

Pin No.	Designation	Description of Function
1-4	D_0-D_3	Bidirectional data bus. All address, instruction and data communication between processor and the RAM MEMORY or the output port is transmitted on these 4 pins.
5	V_{SS}	Circuit GND potential; most positive supply voltage.
6-7	ϕ_1-ϕ_2	Non-overlapping clock signals which are used to generate the basic chip timing.
8	SYNC	Synchronization input signal driven by SYNC output of processor.
9	RESET	RESET input. A logic "1" level applied to the chip, will cause a clear of all output and control static flip-flops and will clear the RAM array. To completely clear the memory, RESET must be applied for at least 32 instruction cycles (256 clock periods) to allow the internal refresh counter to scan the memory. During RESET the data bus output buffers are inhibited (floating condition).
10	Po	For chip selection, the 4002 is available in two metal options, 4002-1 and 4002-2. An external pin, P_0 is also available for chip selection. The chip number is assigned as follows:

Chip No.	4002 Option	P_0	D_3 D_2 @ X_2
0	4002-1	GND	0 0
1	4002-1	V_{DD}	0 1
2	4002-2	GND	1 0
3	4002-2	V_{DD}	1 1

11	CM	Command input driven by CM-RAM output of processor. Used for enabling the device during the decoding SRC and instructions.
12	V_{DD}	Main power supply pin. Value must be V_{SS} - 15V ± 5%.
13-16	O_3-O_0	Four bit output port used for transferring data from the CPU to the users system. The outputs are buffered and data remains stable after the port has been loaded. This port can be made TTL compatible.

FIGURE 8. 4002-320 BIT RAM AND 4 BIT OUTPUT PORT

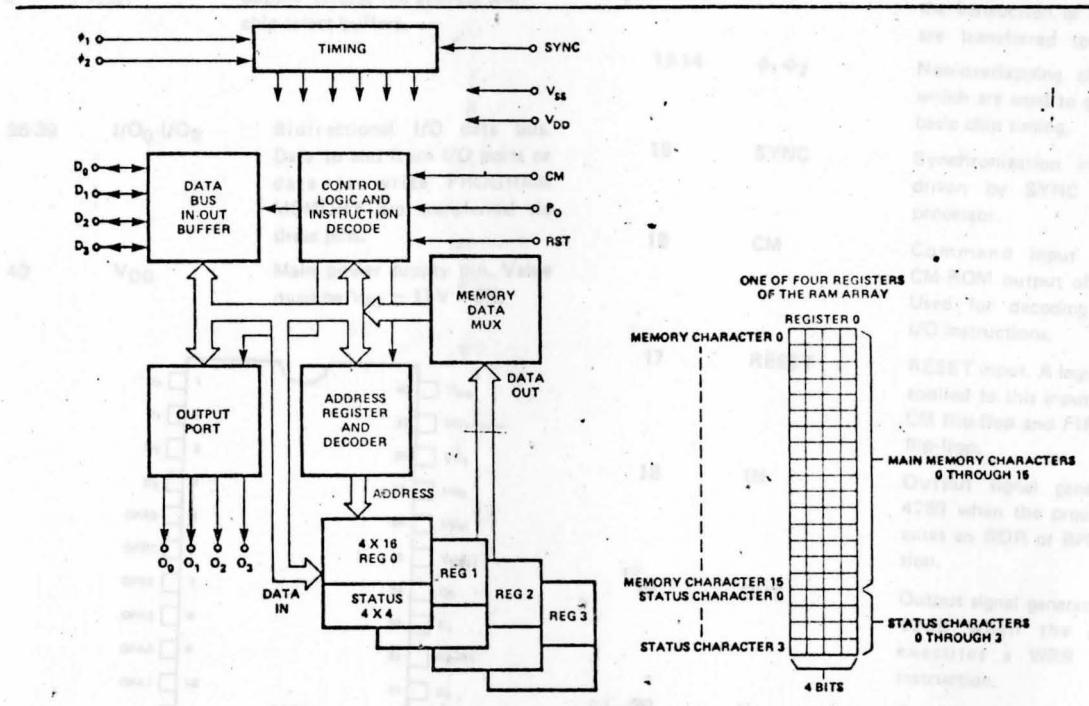
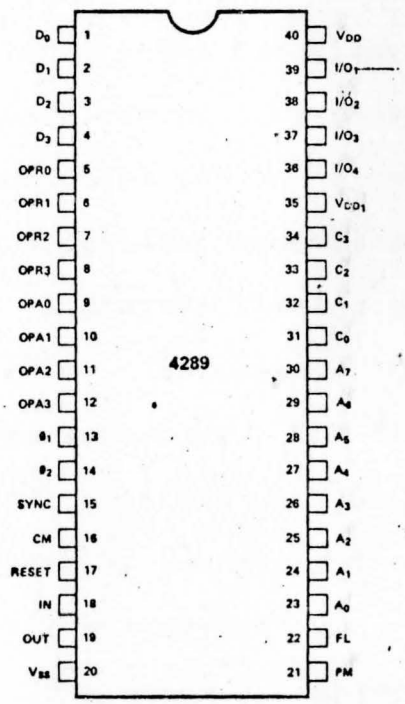


FIGURE 9. 4002 BLOCK DIAGRAM

31-34	C_0-C_3	Chip select output buffers. The address data generated by the processor at A_3, or during an SRC are transferred here.
35	V_{DD1}	Supply voltage for address and chip select buffers.
36-39	I/O_0-I/O_3	Bidirectional I/O data bus. Data to and from I/O ports or data to write PROGRAM MEMORY are transferred via these pins.
40	V_{DD}	Main power supply pin. Value must be V_{SS} - 15V ± 5%.



Pin Description

Pin No.	Designation	Description of Function
1-4	D_0-D_3	Bidirectional data bus. All address, instruction and data communication between processor and the PROGRAM MEMORY or I/O ports is transmitted on these 4 pins.
5-8	OPR_0-OPR_3	The high order 4 bits (OPR) of the instruction or data (RPM) from the PROGRAM MEMORY are transferred to the 4289 on these pins.
9-12	OPA_0-OPA_3	The low order 4 bits (OPA) of the instruction or data (RPM) are transferred to the 4289.
13-14	ϕ_1-ϕ_2	Non-overlapping clock signals which are used to generate the basic chip timing.
15	SYNC	Synchronization input signal driven by SYNC output of processor.
16	CM	Command input driven by CM-ROM output of processor. Used for decoding SRC and I/O instructions.
17	RESET	RESET input. A logic "1" level applied to this input resets the CM flip-flop and FIRST/LAST flip-flop.
18	IN	Output signal generated by 4289 when the processor executes an RDR or RPM instruction.
19	OUT	Output signal generated by the 4289 when the processor executes a WRR or WPM instruction.
20	V_{SS}	Circuit Reference potential, most positive supply voltage
21	PM	Output signal generated by the 4289 when the processor executes an RPM or WPM instruction.
22	F/L	Output signal generated by the 4289 to indicate which half byte of PROGRAM MEMORY is to be operated on.
23-30	A_0-A_7	Address output buffers. The demultiplexed address value generated by the 4289 from the address data supplied by the processor at A_1 and A_2.

FIGURE 10. 4289 STANDARD MEMORY INTERFACE

In the following chapter, a simulation program is developed based only upon the components presented in this chapter. As new components become available, the simulation program may require minor modification. However, the basic procedure would remain essentially unchanged.

V. SOFTWARE AND SIMULATION

Programming of microcomputers is primarily done in machine language. This means that programs are written as sequences of binary numbers. Writing and debuging programs written in machine code is a tedious process. An assembly language is a program language which allows mnemonic operation codes to be used in place of the binary operation codes. The assembler translates these mnemonics into binary code.

As a part of this thesis a cross-assembler was written for the Intel 4040 central processing unit. The program is written in Fortran. A cross-assembler is an assembly language for a computer which executes on another computer. The cross-assembler written executes on any IBM computer. Cross-assemblers have become very popular for microcomputers, since most microcomputers can not directly address enough program memory to use a resident assembler.

A program listing of the cross-assembler is seen in Appendix A. The cross-assembler is of the type known as a two-pass assembler. The first pass assigns an address to each of the labels used in the program. The second pass of the assembler generates the machine code for the microcomputer. All syntactical errors are flaged in the second pass.

The cross-assembler is a fixed column assembler. This means that all operands must reside within certain columns on the input record. There are six input fields that the assembler recognizes. These input fields are seen in

Figure 11. The function of the label parameter is to identify the address of the assembler statement. The label parameter is an alphanumeric symbol which can be up to 3 characters long. The operation code parameter is also an alphanumeric symbol which can be up to three characters long. The arguments which can be used as an operation code are the mnemonic machine instructions which appear in Chapter III. One additional instruction may appear in this field. The instruction is "end". The instruction "end" tells the assembler that this is the last instruction in the program. The next field is designated OPT. OPT is a single hexadecimal argument. OPT contains one of four things. They are as follows:

1. A register address.
2. A register pair address.
3. 4 bits of data.
4. A condition for jumping.

The next field is designated branch label. The field contains a label which is the same as a label appearing in the label field. This field is used for branch instructions. If a branch occurs, the program will be directed to the address of the statement which has a label field identical to the branch label of the instruction being executed. The last two fields, ARG1 and ARG2, are only used for the FIM instruction. These two fields contain data in the form of 2 hexadecimal digits.

Input to the cross-assembler must be in the form specified in Figure 11. The cross-assembler will produce a program listing and the machine code equivalent of the input program. The machine code produced is written as 2 hexadecimal digits in the first 2 columns of a record. Therefore, a

LABEL	OPERATION CODE	OPT1	BRANCH LABEL	ARG1	ARG2

FIGURE 11. CROSS-ASSEMBLER INPUT FORMAT

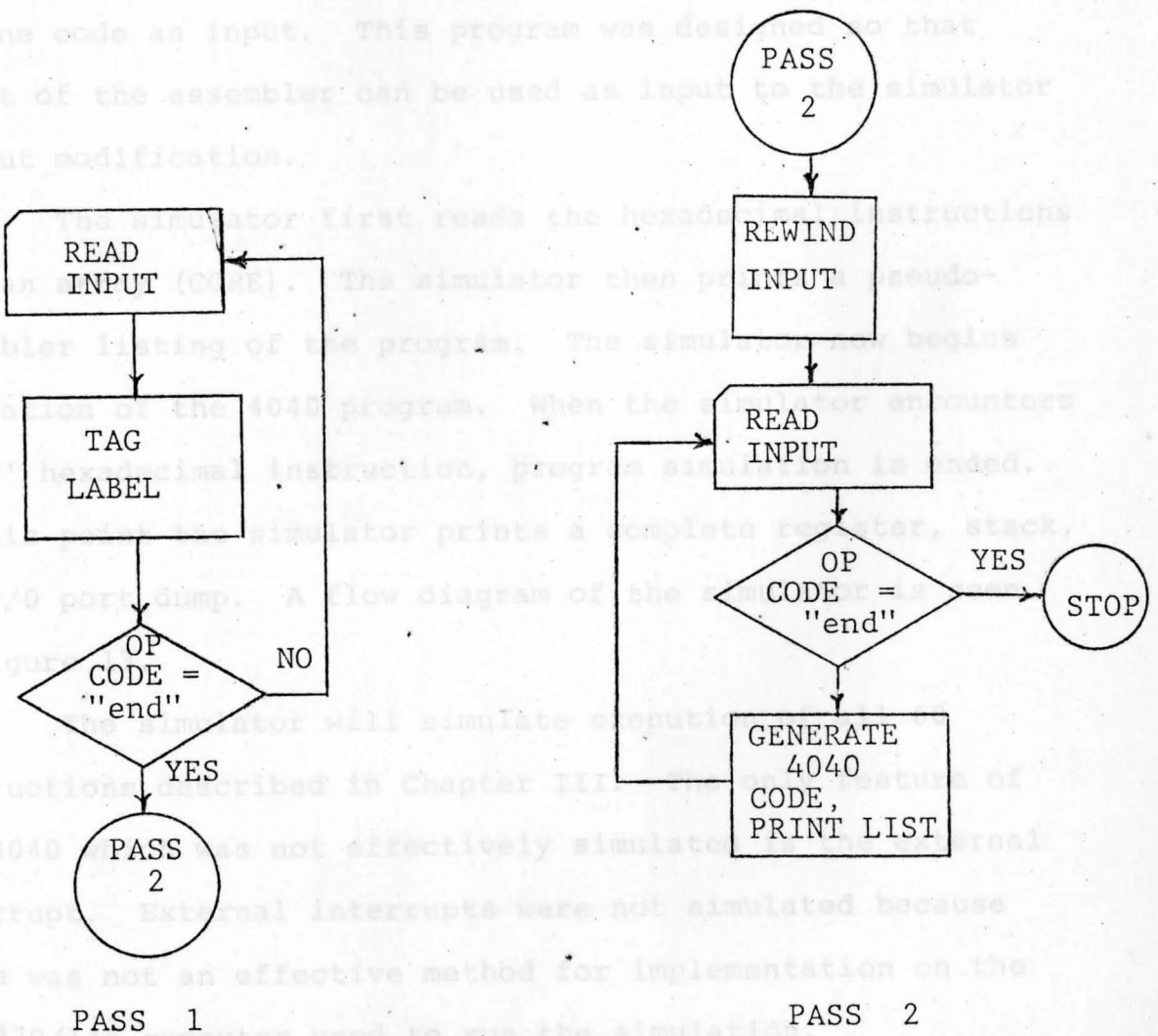


FIGURE 12. CROSS-ASSEMBLER FLOW CHART

1-word instruction will require 1 record, while a 2-word instruction will require 2 records. A flow chart of the cross-assembler logic is seen in Figure 12.

Once an assembler program has been written there is no assurance that the program will work properly. An effective method of program testing is through simulation. As a part of this thesis a 4040 system simulator was developed. The simulator is written in Fortran, and will simulate any 4040 microcomputer system utilizing the MCS-40 system components discussed in Chapter IV. The simulator program listing is seen in Apprendix C. The simulator requires hexadecimal machine code as input. This program was designed so that output of the assembler can be used as input to the simulator without modification.

The simulator first reads the hexadecimal instructions into an array (CORE). The simulator then prints a pseudo-assembler listing of the program. The simulator now begins simulation of the 4040 program. When the simulator encounters a 'OF' hexadecimal instruction, program simulation is ended. At this point the simulator prints a complete register, stack, and I/O port dump. A flow diagram of the simulator is seen in Figure 13.

The simulator will simulate execution of all 60 instructions described in Chapter III. The only feature of the 4040 which was not effectively simulated is the external interrupt. External interrupts were not simulated because there was not an effective method for implementation on the IBM 370/145 computer used to run the simulation.

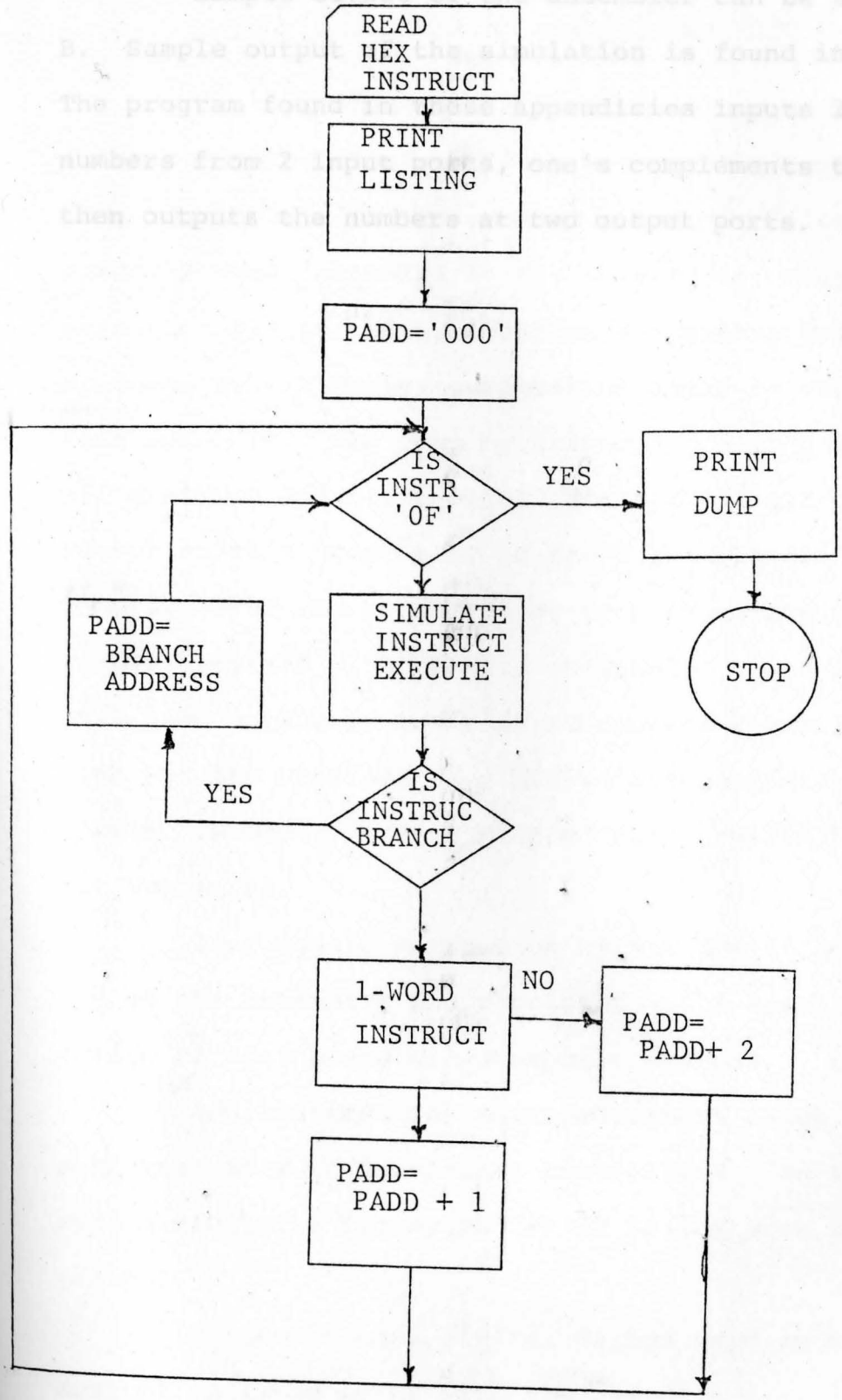


FIGURE 13. SIMULATOR FLOW DIAGRAM

Sample output of the assembler can be seen in Appendix B. Sample output of the simulation is found in Appendix D. The program found in these appendicies inputs 2 four bit numbers from 2 input ports, one's complements the numbers, then outputs the numbers at two output ports.

VI. TESTING AND APPLICATIONS

Testing of assemblers and simulators is necessary. If an assembler or simulator does not work properly, then a program developed with the assembler and simulator will not work properly when installed on the actual microcomputer. The assembler was tested by checking the binary operation codes produced by all mnemonic operation codes in the 4040 instruction set. This was done by assembling a program containing all mnemonic operation codes in the 4040 instruction set. The output of this program can be found in Appendix E. Testing of the simulator was performed by comparing simulator output of sample programs with results obtained by hand calculations. This hand method is inferior to comparing simulator output with results obtained on a MCS-40 microcomputer development system. However, MCS-40 microcomputer system hardware was not available.

Thorough investigation of the MCS-40 components and use of the assembler and simulator, will simplify system design of 4040 based microcomputer systems.

Applications for microprocessors range from automated cash registers to industrial controllers. An interesting application of a microcomputer is in the area of motor speed control.

A conventional digital method used in motor speed control is implemented with the use of a digital phase-lock loop. This method is seen in Figure 14. The conventional digital phase-lock loop makes the voltage controlled oscilla-

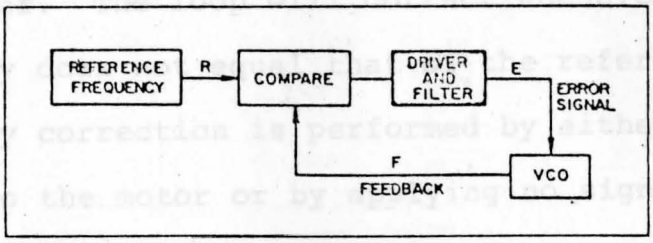


FIGURE 14. CONVENTIONAL DIGITAL PHASE-LOCK LOOP

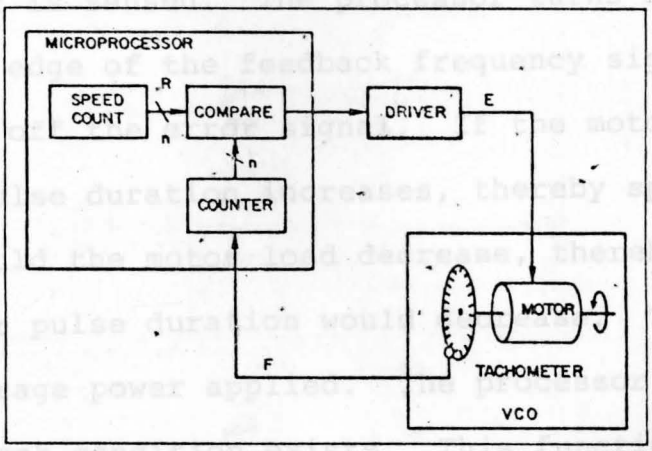


FIGURE 15. COMPUTERIZED DIGITAL PHASE-LOCK LOOP

tor frequency track that of the reference. In motor speed control, the voltage controlled oscillator is actually a motor tachometer. The loop will correct whenever the feedback frequency does not equal that of the reference. Phase and frequency correction is performed by either applying an error signal to the motor or by applying no signal at all.

With the microcomputer approach, a dedicated series of micro-instructions implements the strategy of phase and frequency correction.[12] Figure 15 shows how a microcomputer becomes an integral part of a digital phase-lock loop, and Figure 16 is a flow chart describing one method of microcomputer driven motor speed control. As can be seen in Figure 16, when the leading edge of the reference frequency is encountered an interrupt is sensed. The processor turns on an error signal. The leading edge of the feedback frequency signals the processor to turn off the error signal. If the motor speed decreases, the error pulse duration increases, thereby speeding up the motor. Should the motor load decrease, thereby increasing speed, error pulse duration would decrease. This would result in less average power applied. The processor now checks if an out-of-lock condition exists. This function is performed by producing a count indicating the feedback frequency. This is compared to the reference frequency. If a gross difference exists, application or removal of the error pulse will bring the motor into lock. Figure 17 shows the overall system

[12]Raphael, Howard A., "Motor Control by PLL", ELECTRONIC DESIGN, April 26, 1975, p. 54-57

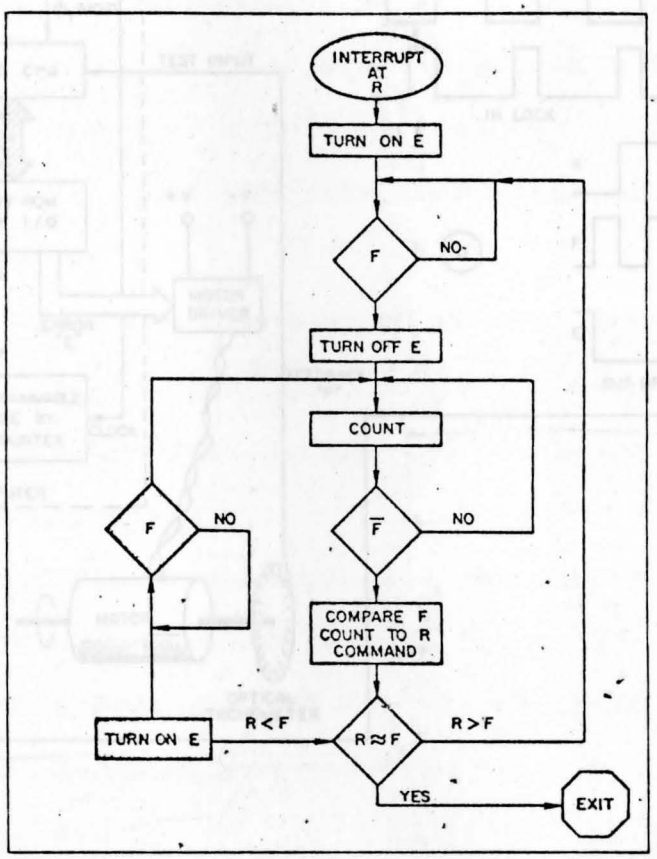


FIGURE 16. FLOW CHART OF COMPUTERIZED MOTOR CONTROL

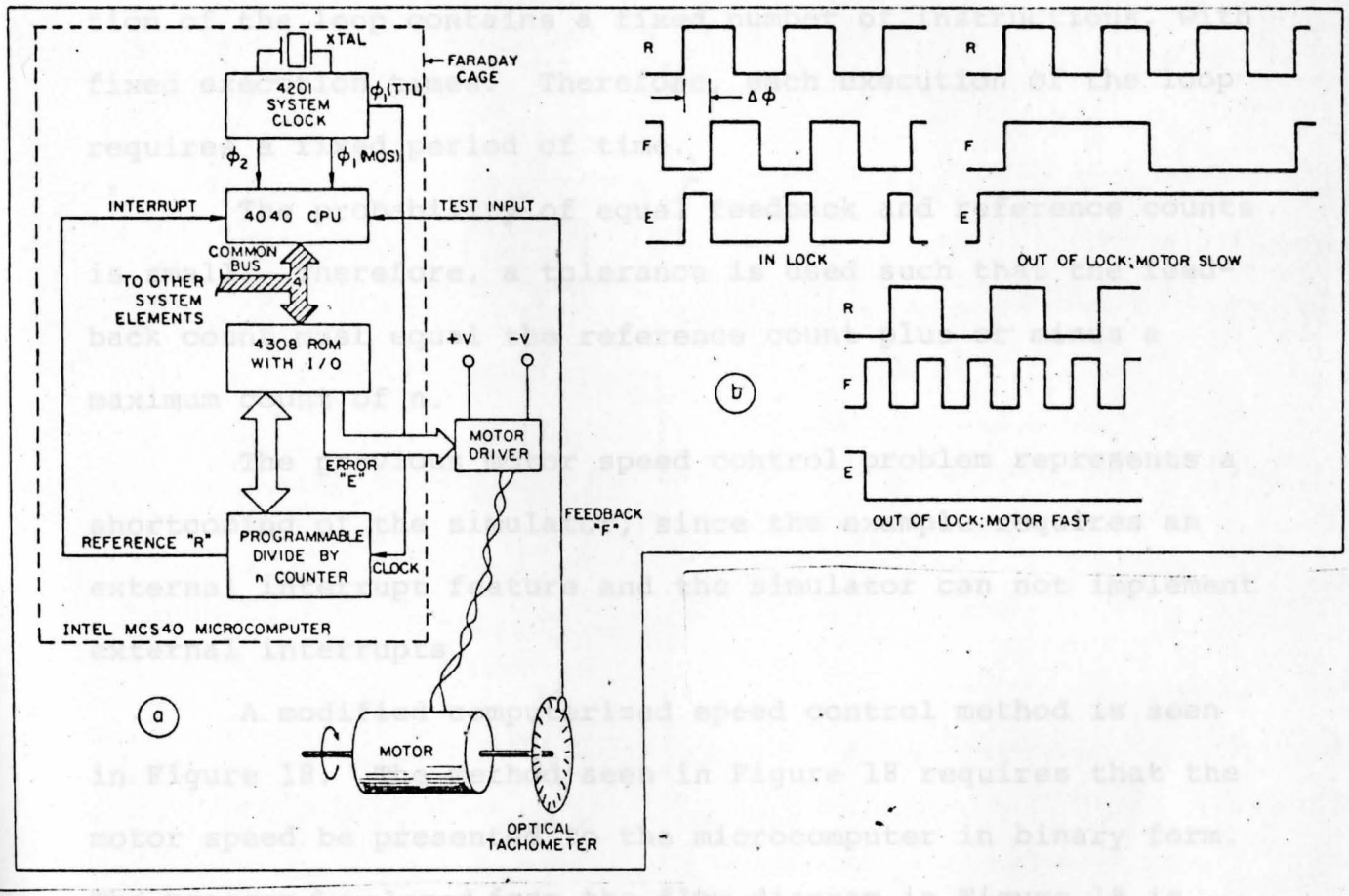


FIGURE 17. SYSTEM CONFIGURATION WITH SAMPLE LOCK CONDITIONS

design with some examples of lock conditions.

To calculate the count, the processor executes a software loop while the feedback signal is present. Each execution of the loop contains a fixed number of instructions, with fixed execution times. Therefore, each execution of the loop requires a fixed period of time.

The probability of equal feedback and reference counts is small. Therefore, a tolerance is used such that the feedback count must equal the reference count plus or minus a maximum count of n.

The previous motor speed control problem represents a shortcoming of the simulator, since the example requires an external interrupt feature and the simulator can not implement external interrupts.

A modified computerized speed control method is seen in Figure 18. The method seen in Figure 18 requires that the motor speed be presented to the microcomputer in binary form. The program developed from the flow diagram in Figure 18 is seen in Appendix F.

The program in Appendix F first reads a 4 bit (scaled) desired motor speed. The program then checks if the motor speed is greater than the desired speed. If the motor speed is greater than the desired speed, the error pulse is turned off. This results in less average power applied and a reduction in speed. The error pulse is a single bit. The error pulse is off when the output bit is "0". The error pulse is on when the output bit is "1". If the motor speed is less

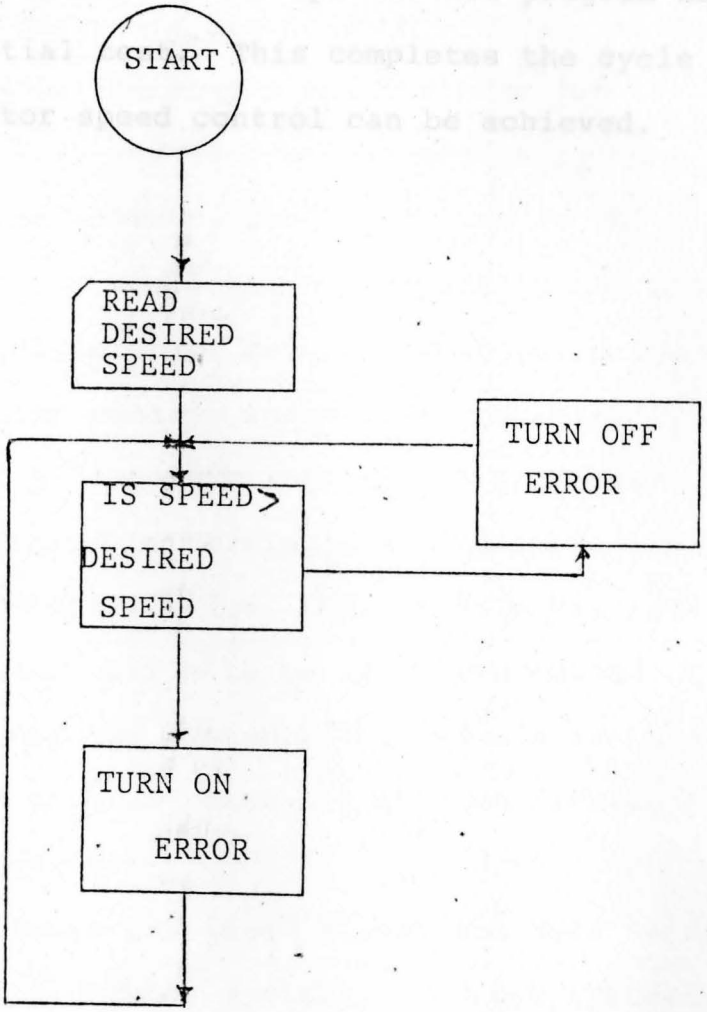


FIGURE 18. ALTERNATIVE COMPUTERIZED MOTOR SPEED CONTROL

than the desired speed, the error pulse is turned on. This will result in an increase in speed. The program now loops back to the initial test. This completes the cycle with which computerized motor speed control can be achieved.

VII. CONCLUSIONS

The software developed in this thesis is functional. The cross-assembler generates machine code for the 4040. The simulator simulates any 4040 microcomputer system. The only weak point of the software developed lies in the simulator. External interrupts could not be adapted to the simulator.

Microcomputers are becoming integral parts of cash registers, display panels, industrial controllers, and many other devices. All applications of microcomputers require that a micro-program be developed to implement the strategy required for the application. This thesis has provided the 4040 microcomputer user with an effective method of programming the microcomputer system. This thesis has also provided for inexpensive program check-out through simulation of 4040 microcomputer systems.

The software developed allows the user to design around 4040 microcomputer systems. This may represent a limitation to the design engineer, since other microcomputers may be better for a given application. This limitation is due to the fact that the software developed is not directly applicable to other microcomputer systems. The developed software, however, can be modified such that it will work for another microcomputer. This modification would mean completely rewriting all of the software developed, but the rewriting process would be relatively inexpensive since the program structure remains basically the same. Only the features of the microcomputers will differ.

Development of higher order languages is a suggested extension to this thesis. A higher order language such as Fortran or PL/1 could be developed. The advantage of a high order language is that it allows the programmer to write a program in fewer source statements then a similar program written in assembly language. This represents a reduction in development time and a cost savings.. The software developed in this thesis should be used to develop the high order languages, since it will simplify development of the high order languages.

In conclusion optimal microcomputer system performance is only as good as the software developed by the engineer.

APPENDIX A

CROSS-ASSEMBLER PROGRAM LISTING

```
FILE: ASSEMBLE FORTRAN  A1          YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

      IMPLICIT INTEGER (A - Z)                                          ASSC0010
      DIMENSION LABEL(2000),LABADD(2000),SYMBOL(100)                    ASSC0020
      DIMENSION NJCL(72)                                                ASSC0030
C                                                                       ASSC0040
C                                                                       ASSC0050
C     LABEL IS AN ARRAY CONTAINING LABELS                               ASSC0060
C                                                                       ASSC0070
C     LABADD IS AN ARRAY CONTAINING ADDRESSES OF LABELS                 ASSC0080
C                                                                       ASSC0090
C     SYMBOL IS AN ARRAY CONTAINING ASSEMBLER MNEUMONIC SYMBOLS         ASSC0100
C                                                                       ASSC0110
C     PACD IS A VARIABLE CONTAINING THE PROGRAM ADDRESS                 ASSC0120
C                                                                       ASSC0130
C                                                                       ASSC0140
C                                                                       ASSC0150
      DATA END/'END'/                                                   ASSC0160
      DATA BLANK /'    '/                                               ASSC0170
      DATA LABEL /2000*'    '/                                          ASSC0180
      DATA LABADD /2000*0/                                              ASSC0190
      DATA SYMBOL/'NOP','HLT','BBS','LCR','OR4','OR5','AN6','AN7',     ASSC0200
     *'DB0','DB1','SBC','SB1','EIN','DIN','RPM','JCN','FIM','SRC',     ASSC0210
     *'FIN','JIN','JUN','JMS','INC','ISZ','ADD','SUB','LD','XCH','BBL',ASSC0220
     *'LDM','WRM','WMP','WRR','WPM','WR0','WR1','WR2','WR3',            ASSC0230
     *'SBM','RDM','RDR','ADM','RD0','RD1','RD2','RD3','CLB',           ASSC0240
     *'CLC','IAC','CMC','CMA','RAL','RAR','TCC','DAC',                 ASSC0250
     *'TCS','STC','DAA','KBP','DCL','END'/                             ASSC0260
      BANK=1                                                            ASSC0270
      INFILE=1                                                          ASSC0280
      CFILE=7                                                           ASSC0290
C                                                                       ASSC0300
C                                                                       ASSC0310
C                                                                       ASSC0320
9     PACD=-1                                                           ASSC0330
      LABKNT=1                                                          ASSC0340
7     FORMAT(A3,2X,A3,2X,Z1,A3,2Z1)                                     ASSC0350
1     CONTINUE                                                          ASSC0360
      READ(INFILE,7,END=4)RLAB,RSYMB,OPT1,ARSYMB,ARG1,ARG2              ASSC0370
C                                                                       ASSC0380
C                                                                       ASSC0390
      PACD=PACD + 1                                                     ASSC0400
      IF(RLAB.EQ.BLANK)GO TO 2                                          ASSC0410
      LABEL(LABKNT)= RLAB                                               ASSC0420
      LABADD(LABKNT)=PADD                                               ASSC0430
      LABKNT=LABKNT + 1                                                 ASSC0440
C                                                                       ASSC0450
2     IF(RSYMB.EQ.SYMBOL(16))PACD = PACD + 1                            ASSC0460
      IF(RSYMB.EQ.SYMBOL(17))PACD = PACD + 1                            ASSC0470
      IF(RSYMB.EQ.SYMBOL(21))PACD = PACD + 1                            ASSC0480
      IF(RSYMB.EQ.SYMBOL(22))PACD = PACD + 1                            ASSC0490
      IF(RSYMB.EQ.SYMBOL(24))PADD = PADD + 1                            ASSC0500
      IF(RSYMB.NE.END)GO TO 1                                           ASSC0510
C                                                                       ASSC0520
C                                                                       ASSC0530
C                                                                       ASSC0540
C     THE FOLLOWING SECTION GENERATES THE CODE                          ASSC0550
```

```
FILE: ASSEMBLE FORTRAN  A1          YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

C                                                                       ASSC0560
C                                                                       ASSC0570
4       REWIND INFILE                                                   ASSC0580
        PCOUNT=0                                                        ASSC0590
        PAGE=1                                                          ASSC0600
        STMT=0                                                          ASSC0610
        PADD=-1                                                         ASSC0620
        WRITE(6,90)                                                     ASSC0630
        WRITE(6,901)PAGE                                                ASSC0640
        WRITE(6,902)BANK                                                ASSC0650
        WRITE(6,903)                                                    ASSC0660
        GO TO 51                                                        ASSC0670
59      WRITE(OFILE,17)CAT1                                             ASSC0680
17      FORMAT(Z2)                                                      ASSC0690
        IF(TWODAT.EQ.1)WRITE(OFILE,17)DAT2                              ASSC0700
51      READ(INFILE,7)RLAB,RSYMB,OPT1,ARSYMB,ARG1,ARG2                  ASSC0710
C                                                                       ASSC0720
C                                                                       ASSC0730
        TWODAT=0                                                        ASSC0740
        PADD= PADD + 1                                                  ASSC0750
903     FORMAT(38X,'ADDRESS   STMT   STATEMENT')                        ASSC0760
901     FORMAT(110X,'PAGE ',I3)                                         ASSC0770
902     FORMAT(110X,'BANK ',I3)                                         ASSC0780
904     FORMAT(40X,Z3,5X,I4,5X,72A1)                                    ASSC0790
        CPAGE=MOD(PADD,256)                                             ASSC0800
        IF(CPAGE.NE.0)GO TO 50                                          ASSC0810
        CPAGE=PADD/256                                                  ASSC0820
        WRITE(6,899)CPAGE                                               ASSC0830
899     FORMAT(/,10X,'CORE PAGE',I3,/)                                  ASSC0840
        PCOUNT=PCOUNT + 3                                               ASSC0850
50      IF(PCOUNT.LE.55)GO TO 83                                        ASSC0860
        PCOUNT=0                                                        ASSC0870
        WRITE(6,90)                                                     ASSC0880
90      FORMAT(1H1)                                                     ASSC0890
        PAGE=PAGE + 1                                                   ASSC0900
        WRITE(6,901)PAGE                                                ASSC0910
        WRITE(6,902)BANK                                                ASSC0920
        WRITE(6,903)                                                    ASSC0930
83      PCOUNT=PCOUNT + 1                                               ASSC0940
        STMT=STMT + 1                                                   ASSC0950
        BACKSPACE INFILE                                                ASSC0960
        READ(INFILE,84)NJCL                                             ASSC0970
84      FORMAT(72A1)                                                    ASSC0980
        WRITE(6,904)PADD,STMT,NJCL                                      ASSC0990
        DO 53 J=1,100                                                   ASSC1000
        IF(RSYMB.EQ.SYMBOL(J)) GO TO 54                                 ASSC1010
53      CONTINUE                                                        ASSC1020
        WRITE(6,941)                                                    ASSC1030
941     FORMAT('          *** SYNTAX ***')                              ASSC1040
        GO TO 99                                                        ASSC1050
C                                                                       ASSC1060
C                                                                       ASSC1070
C                                                                       ASSC1080
54      GO TO (100,100,100,100,100,100,100,100,100,100,100,100,100,    ASSC1090
     *100,100,115,116,117,118,119,120,121,122,123,124,125,126,          ASSC1100
```

```
     *127,128,129,131,131,131,131,131,131,131,131,131,131,131,      ASS01110
     *131,131,131,131,131,131,131,131,131,131,131,131,131,131,      ASS01120
     *131,131,131,131,131,161,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,  ASS01130
     *8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8),J                 ASS01140
C                                                                   ASS01150
C                                                                   ASS01160
100  DAT1= J-1                                                      ASS01170
     GO TO 99                                                       ASS01180
131   DAT1=J + 193                                                  ASS01190
     GO TO 99                                                       ASS01200
116   TWODAT=1                                                      ASS01210
C    ASSEMBLER INSTRUCTION IS ... FIM                               ASS01220
     DAT1=32 + OPT1                                                 ASS01230
     DAT2= 16* ARG1 + ARG2                                          ASS01240
     PACD= PACD + 1                                                 ASS01250
     GO TO 99                                                       ASS01260
117  CONTINUE                                                       ASS01270
C    ASSEMBLER INSTRUCTION IS ... SRC                               ASS01280
     DAT1= OPT1 + 33                                                ASS01290
     GO TO 99                                                       ASS01300
118  DAT1=OPT1 + 48                                                 ASS01310
C    ASSEMBLER INSTRUCTION IS ... FIN                               ASS01320
     GO TO 99                                                       ASS01330
119  DAT1= OPT1 + 49                                                ASS01340
C    ASSEMBLER INSTRUCTION IS ... JIN                               ASS01350
     GO TO 99                                                       ASS01360
115  TWODAT=1                                                       ASS01370
     DAT1= 16 + OPT1                                                ASS01380
C    ASSEMBLER INSTRUCTION IS ... JCN                               ASS01390
     CALL SUBFND (SUADDR,LABEL,ARSYMB,LABKNT)                       ASS01400
     NUM=LABACD(SUADDR)                                             ASS01410
     A=NUM/256                                                      ASS01420
     B=PACD/256                                                     ASS01430
     IF(A.NE.B)WRITE(6,777)ARSYMB                                   ASS01440
777  FORMAT(' LABEL ',A3,' CAUSED PAGING-ERROR JCN INSTRUCTION')    ASS01450
     DAT2=MOD(NUM,256)                                              ASS01460
     PACD=PACD + 1                                                  ASS01470
     GO TO 99                                                       ASS01480
120  CALL SUBFND(SUADDR,LABEL,ARSYMB,LABKNT)                        ASS01490
     NUM=LABACD(SUADDR)                                             ASS01500
C    ASSEMBLER INSTRUCTION IS ... JUN                               ASS01510
     DAT1= 64 + NUM/256                                             ASS01520
     TWODAT=1                                                       ASS01530
     DAT2=MOD(NUM,256)                                              ASS01540
     PACD=PACD + 1                                                  ASS01550
     GO TO 99                                                       ASS01560
121  CALL SUBFND(SUADDR,LABEL,ARSYMB,LABKNT)                        ASS01570
C    ASSEMBLER INSTRUCTION IS ... JMS                               ASS01580
     NUM=LABACD(SUADDR)                                             ASS01590
     DAT1= 80 + NUM/256                                             ASS01600
     TWODAT=1                                                       ASS01610
     DAT2=MOD(NUM,256)                                              ASS01620
     PACD=PACD + 1                                                  ASS01630
     GO TO 99                                                       ASS01640
122  DAT1= OPT1 + 96                                                ASS01650
```

```
FILE: ASSEMBLE FORTRAN  A1          YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

C       ASSEMBLER INSTRUCTION IS ... INC                                ASSC1660
        GO TO 99                                                        ASS01670
123     CONTINUE                                                        ASSC1680
        CAT1= 112 + CPT1                                                ASS01690
C       ASSEMBLER INSTRUCTION IS ... ISZ                                ASSC1700
        TWOCAT=1                                                        ASSC1710
        CALL SUBEND (SUADDR,LABEL,ARSYMB,LABKNT)                        ASSC1720
        NUM=LABADD(SUADDR)                                              ASSC1730
        CAT2=MOD(NUM,256)                                               ASSC1740
888     FORMAT(' LABEL ',A3,' CAUSED PAGING ERROR IN ISZ INSTRUCTION')  ASSC1750
        A=NUM/256                                                       ASSC1760
        B=PADD/256                                                      ASSC1770
        IF(A.NE.B)WRITE(6,888)ARSYMB                                    ASSC1780
        GO TO 99                                                        ASSC1790
124     CAT1=128 + OPT1                                                 ASSC1800
C       ASSEMBLER INSTRUCTION IS ... ADD                                ASSC1810
        GO TO 99                                                        ASSC1820
125     CAT1= 144 + CPT1                                                ASSC1830
C       ASSEMBLER INSTRUCTION IS ... SUB                                ASSC1840
        GO TO 99                                                        ASSC1850
126     CAT1=160 + OPT1                                                 ASSC1860
C       ASSEMBLER INSTRUCTION IS ... LD                                 ASSC1870
        GO TO 99                                                        ASSC1880
127     CAT1=176 + OPT1                                                 ASSC1890
C       ASSEMBLER INSTRUCTION IS XCH                                    ASSC1900
        GO TO 99                                                        ASSC1910
128     CAT1=192 + OPT1                                                 ASSC1920
C       ASSEMBLER INSTRUCTION IS ... BBL                                ASSC1930
        GO TO 99                                                        ASSC1940
129     CAT1= 208 + CPT1                                                ASSC1950
C       ASSEMBLER INSTRUCTION IS ... LDM                                ASSC1960
        GO TO 99                                                        ASSC1970
161     CONTINUE                                                        ASSC1980
C                                                                       ASSC1990
C       THIS SECTION IS WHERE END IS ENCOUNTERED                        ASSC2000
        IF(BANK.EQ.2) GO TO 1982                                        ASSC2010
        BANK=2                                                          ASSC2020
        INFILE=2                                                        ASSC2030
        OFILE=8                                                         ASSC2040
        GO TO 9                                                         ASSC2050
8       CONTINUE                                                        ASSC2060
1982    STOP                                                            ASSC2070
        END                                                             ASSC2080
C                                                                       ASSC2090
        SUBROUTINE SUBEND(SUADDR,LABEL,ARSYMB,LABKNT)                   ASSC2100
        IMPLICIT INTEGER (A - Z)                                        ASSC2110
        DIMENSION LABEL(2000)                                           ASSC2120
        DO 1 J=1,LABKNT                                                 ASSC2130
        IF(ARSYMB.EQ.LABEL(J)) GO TO 2                                  ASSC2140
1       CONTINUE                                                        ASSC2150
        WRITE(6,987)ARSYMB                                              ASSC2160
987     FORMAT(' *** ERROR *** ',A3,' NOT FOUND FOR BRANCH DESTINATION') ASSC2170
2       SUADDR=J                                                        ASSC2180
        RETURN                                                          ASSC2190
        END                                                             ASSC2200
```

APPENDIX B

CROSS-ASSEMBLER SAMPLE OUTPUT

```
TOP   FIM   2     00
      SRC   2
      RDR
      CMA
      WPR
      XCH   0
      LDM   1
      XCH   3
      SRC   2
      RDR
      CMA
      WPR
      XCH   1
STP   END

R;
```

```
                    ADDRESS   STMT   STATEMENT

CORE PAGE  0

                      000        1       TOP   EIM   2     00
                      002        2             SPC   2
                      003        3             PPP
                      004        4             CMA
                      005        5             UPP
                      006        6             XCH   0
                      007        7             LDM   1
                      008        8             XCH   3
                      009        9             SPC   2
                      00A       10             PPP
                      00B       11             CMA
                      00C       12             UPP
                      00D       13             XCH   1
                      00E       14       STP   END
```

```
22
00
23
EA
F4
F2
B0
D1
B3
23
EA
F4
F2
B1

R;
```

APPENDIX C

INTEL 4040 SIMULATOR PROGRAM LISTING

```
FILE: SIM      FORTRAN  A1          YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

      IMPLICIT INTEGER (A - Z)                                          SIMCC010
      REAL TME                                                          SIMCC020
      DIMENSION LOCATE(7),INDEX(2,16),CORE(2,2,4096),                   SIMCC030
     *DATRAM(4,4,4,20),DATOUT(4,4)                                      SIMCC040
      DIMENSION B(2)                                                    SIMCC050
      COMMON /SUB/CORE                                                  SIMCC060
C                                                                       SIMCC070
C     LOCATE(7) IS AN ARRAY CONTAINING 7 ADDRESSES FOR A SUBROUTINE     SIMCC080
C     WHICH CAN BE CALLED                                               SIMCC090
C                                                                       SIMCC100
C     INDEX(2,16) IS A 2 DIMENSIONAL ARRAY CONTAINING THE 4 BIT VALUES  SIMCC110
C     WHICH ARE FOUND IN THE INDEX REGISTERS                            SIMCC120
C                                                                       SIMCC130
C     CORE(2,2,4096) IS A 3 DIMENSIONAL ARRAY CONTAINING INSTRUCTIONS TOSIMCC140
C     BE EXECUTED                                                       SIMCC150
C     1ST POSITION IS CORE BANK NUMBER                                  SIMCC160
C     2ND POSITION IS THE NUMBER OF THE HALF BYTE POSITION              SIMCC170
C     3RD POSITION IS THE ADDRESS OF THE INSTRUCTION                    SIMCC180
C                                                                       SIMCC190
C     DATRAM(4,4,4,20) IS A 4 DIMENSIONAL ARRAY CONTAINING DATA IN      SIMCC200
C     THE DATARAM                                                       SIMCC210
C     1ST POSITION SIGNIFIES DATARAM BANK                               SIMCC220
C      2ND POSITION SIGNIFIES DATARAM NUMBER                            SIMCC230
C     3RD POSITION SIGNIFIES ROW OF DATARAM                             SIMCC240
C     4TH POSITION SIGNIFIES COLUMN NUMBER, LAST 4 NUMBERS ARE          SIMCC250
C     STATUS POSITIONS                                                  SIMCC260
C                                                                       SIMCC270
C     DATOUT(4,4) IS A 2 DIMENSIONAL ARRAY CONTAINING OUTPUT OF THE     SIMCC280
C     16 DATARAM CHIPS,  FIRST POSITION IS BANK NUMBER, SECOND          SIMCC290
C     POSITION IS CHIP NUMBER                                           SIMCC300
C                                                                       SIMCC310
C     SCYCLE IS A VARIABLE THAT TELLS IF THE BANKS ARE TO BE            SIMCC320
C     SWITCHED, SWITCH OCCURS IF SCYCLE IS EQUAL TO 3                   SIMCC330
C                                                                       SIMCC340
C     SBANK IS THE VARIABLE TELLING WHICH BANK IS TO BE SWITCHED TO,    SIMCC350
C     CB0 MEANS SBANK=1, CB1 MEANS SBANK=2                              SIMCC360
C                                                                       SIMCC370
C     INTEN IS A VARIABLE TELLING IF INTERRUPT IS ENABLED               SIMCC380
C     IF INTEN=1 INTERRUPT IS ENABLED,IF INTEN=0 INTERRUPT              SIMCC390
C     IS DISABLED                                                       SIMCC400
C                                                                       SIMCC410
C     CHIP IS A VARIABLE USED FOR SRC ADDRESSING RAM CHIP NUMBER        SIMCC420
C                                                                       SIMCC430
C     REGSTR IS A VARIABLE USED FOR SRC ADDRESSING RAM REGISTER NUMBER  SIMCC440
C                                                                       SIMCC450
C     CHAR IS A VARIABLE USED FOR SRC ADDRESSING RAM CHARCTER NUMBER    SIMCC460
C                                                                       SIMCC470
C     PAGE IS A VARIABLE TELLING WHICH PAGE PROGRAM IS EXECUTING IN     SIMCC480
C                                                                       SIMCC490
C     THE FOLLOWING DATA STATEMENTS WILL INITIALIZE ARRAYS              SIMCC500
      DATA LOCATE /7*0/                                                 SIMCC510
      DATA INDEX /32*0/                                                 SIMCC520
      DATA DATRAM /1280*0/                                              SIMCC530
      DATA DATOUT /16*0/                                                SIMCC540
C                                                                       SIMCC550
```

FILE: SIM FORTRAN A1 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
C                                                                       SIMC0560
C                                                                       SIMC0570
C     THE FILLOWING SEGMENT LOADS THE HEXADECIMAL PROGRAM INTO CORE     SIMC0580
C     THE PROGRAM IS LOCATED ON RECORDS WITH THE FOLLOWING FORMAT       SIMC0590
C     FORMAT(32(2Z1,1X))                                                SIMC0600
C                                                                       SIMC0610
C                                                                       SIMC0620
C     THIS SECTION NEEDS WRITTEN                                        SIMC0630
      CALL CORLD                                                        SIMC0640
C                                                                       SIMC0650
C                                                                       SIMC0660
C     PROGRAM IS NOW LOADED INTO CORE                                   SIMC0670
C                                                                       SIMC0680
C                                                                       SIMC0690
C     PSIZE IS A VARIABLE CONTAINING PROGRAM SIZE                       SIMC0700
C                                                                       SIMC0710
C     THE FOLLOWING SUBROUTINE CALL WILL PRINT OUT THE ASSEMBLER LISTINGSIMC0720
C                                                                       SIMC0730
      CALL ASMLST                                                       SIMC0740
C                                                                       SIMC0750
C                                                                       SIMC0760
C                                                                       SIMC0770
C     THE NEXT PART OF THE PROGRAM WILL SIMULATE RUNNING THE PROGRAM    SIMC0780
C     THE 4040 SYSTEM                                                   SIMC0790
C                                                                       SIMC0800
C     IRBANK IS A VARIABLE DESIGNATING INDEX REGISTER BANK SELECTED     SIMC0810
C     BANK IS A VARAABLE WHICH SIGNIFIES WHICH CORE BANK IS ACTIVATED   SIMC0820
C     CYCLE IS A VARIABLE WHICH KEEPS TRACK OF HOW MANY MACHINE CYCLES  SIMC0830
C     THAT IT TAKES TO RUN THE PROGRAM                                  SIMC0840
C                                                                       SIMC0850
C     PADD IS A VARIABLE WHICH DESIGNATES THE ADDRESS THE MICRO IS      SIMC0860
C     PRESENTLY EXECUTING                                               SIMC0870
C     NOBANK IS A VARIABLE TELLING WHICH INDEX REGISTER                 SIMC0880
C     IS NOT ACTIVATED                                                  SIMC0890
C                                                                       SIMC0900
      WRITE(6,189)                                                      SIMC0910
189   FORMAT('   ENTER THE STARTING POSITION FOR RAM IN BANK1 THEN BANK SIMC0920
     *2')                                                               SIMC0930
      READ(5,*)B                                                        SIMC0940
      FLIP=1                                                            SIMC0950
      IRBANK=1                                                          SIMC0960
      NOBANK=2                                                          SIMC0970
      BANK=1                                                            SIMC0980
      CYCLE=0                                                           SIMC0990
      PADD=0                                                            SIMC1000
      INTEN=0                                                           SIMC1010
      SBANK=1                                                           SIMC1020
      SCYCLE=3                                                          SIMC1030
      COMLIN=1                                                          SIMC1040
9999  CONTINUE                                                          SIMC1050
      PADD=PADD+1                                                       SIMC1060
      CYCLE=CYCLE+1                                                     SIMC1070
      SCYCLE=SCYCLE+1                                                   SIMC1080
      IF(SCYCLE.EQ.3)BANK=SBANK                                         SIMC1090
C                                                                       SIMC1100
```

```
ILE: SIM      FORTRAN  A1          YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

      INSTR IS A VARIABLE CONTAINING THE FIRST 4 BITS OF THE INSTRUCTIONSIMO1110
      TO BE EXECUTED                                                   SIMO1120
                                                                       SIMO1130
      INSTR=CORE(BANK,1,PADD) + 1                                      SIMO1140
      GO TO (100,101,102,103,104,105,106,107,108,109,110,111,112,113,  SIMO1150
     *114,115), INSTR                                                  SIMO1160
C                                                                      SIMO1170
C     THE ABOVE GO TO BRANCHES TO THE PART WHERE THE FIRST 4 BITS OF   SIMO1180
C     THE INSTRUCTION ARE                                              SIMO1190
C                                                                      SIMO1200
100   COR2=CORE(BANK,2,PADD) + 1                                       SIMO1210
      GO TO (200,201,202,203,204,205,206,207,208,209,210,211,212,213,  SIMO1220
     *214,215), COR2                                                   SIMO1230
200   CONTINUE                                                         SIMO1240
C     ASSEMBLER INSTRUCTION IS ... NOP                                 SIMO1250
      GO TO 9999                                                       SIMO1260
201   PADD=3                                                           SIMO1270
C     ASSEMBLER INSTRUCTION IS ... HLT                                 SIMO1280
      WRITE(6,91)                                                      SIMO1290
91    FORMAT('   HLT INSTRUCTION ENCOUNTERED, INTERRUPT PROCESSED')    SIMO1300
      GO TO 9999                                                       SIMO1310
202   CONTINUE                                                         SIMO1320
C     ASSEMBLER INSTRUCTION IS ... BBS                                 SIMO1330
      PADD=LOCATE(1)                                                   SIMO1340
      DO 123 KKK=1,6                                                   SIMO1350
123   LOCATE(KKK)=LOCATE(KKK+1)                                        SIMO1360
      LOCATE(7)=99999                                                  SIMO1370
      GO TO 9999                                                       SIMO1380
203   CONTINUE                                                         SIMO1390
C     ASSEMBLER INSTRUCTION IS ... LCR                                 SIMO1400
      ACC=(COMLIN-1)*2 + (BANK-1)                                      SIMO1410
      GO TO 9999                                                       SIMO1420
204   CONTINUE                                                         SIMO1430
C     ASSEMBLER INSTRUCTION IS ... OR4                                 SIMO1440
      CALL BIT(B4,B3,B2,B1,INDEX(IRBANK,5))                            SIMO1450
81    CALL BIT(C4,C3,C2,C1,ACC)                                        SIMO1460
      IF(B4.EQ.1 .OR. C4.EQ.1)C4=1                                     SIMO1470
      IF(B3.EQ.1 .OR. C3.EQ.1)C3=1                                     SIMO1480
      IF(B2.EQ.1 .OR. C2.EQ.1)C2=1                                     SIMO1490
      IF(B1.EQ.1 .OR. C1.EQ.1)C1=1                                     SIMO1500
      ACC=C4*8 + C3*4 + C2*2 + C1                                      SIMO1510
      GO TO 9999                                                       SIMO1520
205   CONTINUE                                                         SIMO1530
C     ASSEMBLER INSTRUCTION IS ... OR5                                 SIMO1540
      CALL BIT(B4,B3,B2,B1,INDEX(IRBANK,6))                            SIMO1550
      GO TO 81                                                         SIMO1560
206   CONTINUE                                                         SIMO1570
C     ASSEMBLER INSTRUCTION IS ... AN6                                 SIMO1580
      CALL BIT(B4,B3,B2,B1,INDEX(IRBANK,7))                            SIMO1590
82    CALL BIT(C4,C3,C2,C1,ACC)                                        SIMO1600
      IF(B4.EQ.0 .OR. C4.EQ.0)C4=0                                     SIMO1610
      IF(B3.EQ.0 .OR. C3.EQ.0)C3=0                                     SIMO1620
      IF(B2.EQ.0 .OR. C2.EQ.0)C2=0                                     SIMO1630
      IF(B1.EQ.0 .OR. C1.EQ.0)C1=0                                     SIMO1640
      ACC= C4*8 + C3*4 + C2*2 + C1                                     SIMO1650
```

```
      GO TO 9999                                                        SIM01660
207   CONTINUE                                                          SIM01670
C     ASSEMBLER INSTRUCTION IS ... AN7                                  SIM01680
      CALL BIT(B4,B3,B2,B1,INDEX(IRBANK,8))                             SIM01690
      GO TO 82                                                          SIM01700
208   SBANK=1                                                           SIM01710
      SCYCLE=0                                                          SIM01720
C     ASSEMBLER INSTRUCTION IS ... DB0                                  SIM01730
      GO TO 9999                                                        SIM01740
209   SBANK=2                                                           SIM01750
C     ASSEMBLER INSTRUCTION IS ... DB1                                  SIM01760
      SCYCLE=0                                                          SIM01770
      GO TO 9999                                                        SIM01780
210   IRBANK=1                                                          SIM01790
C     ASSEMBLER INSTRUCTION IS ... SB0                                  SIM01800
      NOBANK=2                                                          SIM01810
      GO TO 9999                                                        SIM01820
211   IRBANK=2                                                          SIM01830
C     ASSEMBLER INSTRUCTION IS ... SB1                                  SIM01840
      NOBANK=1                                                          SIM01850
      GO TO 9999                                                        SIM01860
212   INTEN=1                                                           SIM01870
C     ASSEMBLER INSTRUCTION IS ... EIN                                  SIM01880
      GO TO 9999                                                        SIM01890
213   INTEN=0                                                           SIM01900
C     ASSEMBLER INSTRUCTION IS ... DIN                                  SIM01910
      GO TO 9999                                                        SIM01920
214   CONTINUE                                                          SIM01930
C     ASSEMBLER INSTRUCTION IS ... RPM                                  SIM01940
      ADDRES=256*DATOUT(1,1) + 16*((CHIP-1)*4 +REGSTR-1)                SIM01950
     *+(CHAR-1) + 1                                                     SIM01960
      ACC=CORE(BANK,FLIP,ADDRES)                                        SIM01970
      IF(FLIP.EQ.1)F=2                                                  SIM01980
      IF(FLIP.EQ.2)F=1                                                  SIM01990
      FLIP=F                                                            SIM02000
      GO TO 9999                                                        SIM02010
215   CONTINUE                                                          SIM02020
      WRITE(6,9876)                                                     SIM02030
9876  FORMAT('1',///,40X,' REGISTER DUMP')                              SIM02040
      TME=CYCLE * 10.8E-6                                               SIM02050
      WRITE(6,1987)CYCLE,TME                                            SIM02060
1987  FORMAT(///,1X,' SIMULATION REQUIRED',I9,'CYCLES',/,               SIM02070
     *1X,' THIS IS APPROXIMATELY', E15.7,' SECONDS CPU TIME')           SIM02080
9875  FORMAT(///,' REGISTER  0 1 2 3 4 5 6 7 8 9 A B C D E F')         SIM02090
      WRITE(6,9875)                                                     SIM02100
      WRITE(6,9874)(INDEX(1,J),J=1,16)                                  SIM02110
9874  FORMAT(' BANK 1    ',16Z2)                                        SIM02120
      WRITE(6,9873)(INDEX(2,J),J=1,16)                                  SIM02130
9873  FORMAT(' BANK 2    ',16Z2)                                        SIM02140
      WRITE(6,9872)ACC                                                  SIM02150
9872  FORMAT(///,' THE VALUE OF THE ACCUMULATOR IS ...',Z2)             SIM02160
      WRITE(6,98711)CARRY                                               SIM02170
98711  FORMAT(//,' THE VALUE OF THE CARRY FLIP-FLOP IS ...',Z2)         SIM02180
      WRITE(6,9871)                                                     SIM02190
9871  FORMAT('1',///,40X,' RAM OUTPUT DUMP')                            SIM02200
```

FILE: SIM FORTRAN A1 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
      WRITE(6,9870)                                                    SIM02210
9870  FORMAT(//,' CHIP NUMBER ',' 0 1 2 3')                            SIM02220
      DO 9868 K=1,4                                                    SIM02230
      KK=K-1                                                           SIM02240
      WRITE(6,9869)KK,(DATOUT(K,J),J=1,4)                              SIM02250
9869  FORMAT(' COMMAND LINE',I2,'  ',4Z2)                              SIM02260
9868  CONTINUE                                                         SIM02270
      WRITE(6,9860)                                                    SIM02280
9860  FORMAT('1',///,35X,'ADDRESS REGISTER DUMP')                      SIM02290
      DO 9861 K=1,7                                                    SIM02300
      WRITE(6,9859)K,LOCATE(K)                                         SIM02310
9859  FORMAT(' ADDRESS STACK LEVEL NUMBER ',I2,' IS ACCRESS ',Z3)      SIM02320
9861    CONTINUE                                                       SIM02330
      WRITE(5,9858)                                                    SIM02340
9858  FORMAT(' DO YOU WANT A DATA RAM DUMP?? 1=YES,0=NO')              SIM02350
      READ(5,*)CON                                                     SIM02360
      IF(CON.NE.1)STOP                                                 SIM02370
      DO 9840 COMLIN=1,4                                               SIM02380
      WRITE(6,985C)                                                    SIM02390
9850  FORMAT('1',40X,'DATA RAM DUMP')                                  SIM02400
      WRITE(6,9849)                                                    SIM02410
9849  FORMAT(///,' RAM ADDRESS       0 1 2 3 4 5 6 7 8 9 A B C D E F') SIM02420
      COM=COMLIN-1                                                     SIM02430
      WRITE(6,9839)COM                                                 SIM02440
9839  FORMAT(' COMMAND LINE NUMBER',I2)                                SIM02450
      DO 9840  CHIP=1,4                                                SIM02460
      CH=CHIP-1                                                        SIM02470
      DO 9840 REGIST=1,4                                               SIM02480
      RE=REGIST-1                                                      SIM02490
      WRITE(6,98391)CH,RE,(DATRAM(COMLIN,CHIP,REGIST,I),I=1,16)        SIM02500
98391  FORMAT(' CHIP',I2,' REGISTER',I2,16Z2)                          SIM02510
9840  CONTINUE                                                         SIM02520
      DO 9820 COMLIN=1,4                                               SIM02530
      WRITE(6,9850)                                                    SIM02540
      WRITE(6,9810)                                                    SIM02550
9810  FORMAT(' STATUS CHARACTER   0 1 2 3')                            SIM02560
      COM=COMLIN-1                                                     SIM02570
      WRITE(6,9819)COM                                                 SIM02580
9819  FORMAT(' COMMAND LINE NUMBER',I2)                                SIM02590
      DO 9820 CHIP=1,4                                                 SIM02600
      CH=CHIP-1                                                        SIM02610
      DO 9820 REGIST=1,4                                               SIM02620
      RE=REGIST-1                                                      SIM02630
      WRITE(6,9809)CH,RE,(DATRAM(COMLIN,CHIP,REGIST,I),I=17,20)        SIM02640
9809  FORMAT(' CHIP',I2,' REGISTER',I2,1X,4Z2)                         SIM02650
9820  CONTINUE                                                         SIM02660
      STOP                                                             SIM02670
101   CONTINUE                                                         SIM02680
C     ASSEMBLER INSTRUCTION IS ... JCN                                 SIM02690
      PAGE =MOD((PADD+2),256)                                          SIM02700
      ADDRES=256*PAGE + 16*CORE(BANK,1,PADD+1) + CORE(BANK,2,PADD+1)   SIM02710
      CYCLE=CYCLE + 1                                                  SIM02720
      SCYCLE=SCYCLE +1                                                 SIM02730
      CORD=CORE(BANK,2,PADD) + 1                                       SIM02740
      PADD=PADD+1                                                      SIM02750
```

FILE: SIM FORTRAN A1 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
43    FORMAT(' INPUT A VALUE FOR TEST')                                  SIM02760
      GO TO (500,501,502,503,504,505,506,507,508,509,510,511,512,513,    SIM02770
     *514,515),CORD                                                      SIM02780
500   GO TO 9999                                                         SIM02790
501   WRITE(6,43)                                                        SIM02800
      READ(5,*)TEST                                                      SIM02810
      IF(TEST.EQ.0)PADD=ADDRES                                           SIM02820
      GO TO 9999                                                         SIM02830
502   IF(CARRY.EQ.1)PADD=ADDRES                                          SIM02840
      GO TO 9999                                                         SIM02850
503   WRITE(6,43)                                                        SIM02860
      READ(5,*)TEST                                                      SIM02870
      IF(TEST.EQ.0 .OR. CARRY.EQ.1)PADD=ADDRES                           SIM02880
      GO TO 9999                                                         SIM02890
504   IF(ACC.EQ.0)PADD=ADDRES                                            SIM02900
      GO TO 9999                                                         SIM02910
505   WRITE(6,43)                                                        SIM02920
      READ(5,*)TEST                                                      SIM02930
      IF(TEST.EQ.0 .OR. ACC.EQ.0)PADD=ADDRES                             SIM02940
      GO TO 9999                                                         SIM02950
506   IF(ACC.EQ.0 .OR. CARRY.EQ.1)PADD=ADDRES                            SIM02960
      GO TO 9999                                                         SIM02970
507   WRITE(6,43)                                                        SIM02980
      READ(5,*)TEST                                                      SIM02990
      IF(ACC.EQ.0 .OR. CARRY.EQ.1 .OR. TEST.EQ.0)PADD=ADDRES             SIM03000
      GO TO 9999                                                         SIM03010
508   PADD=ADDRES                                                        SIM03020
      GO TO 9999                                                         SIM03030
509   WRITE(6,43)                                                        SIM03040
      READ(5,*)TEST                                                      SIM03050
      IF(TEST.EQ.1)PADD=ADDRES                                           SIM03060
      GO TO 9999                                                         SIM03070
510   IF(CARRY.EQ.0)PADD=ADDRES                                          SIM03080
      GO TO 9999                                                         SIM03090
511   WRITE(6,43)                                                        SIM03100
      READ(5,*)TEST                                                      SIM03110
      IF(TEST.EQ.1 .AND. CARRY.EQ.0)PADD=ADDRES                          SIM03120
      GO TO 9999                                                         SIM03130
512   IF(ACC.NE.0)PADD=ADDRES                                            SIM03140
      GO TO 9999                                                         SIM03150
513   WRITE(6,43)                                                        SIM03160
      READ(5,*)TEST                                                      SIM03170
      IF(ACC.NE.0 .AND. TEST.EQ.1)PADD=ADDRES                            SIM03180
      GO TO 9999                                                         SIM03190
514   IF(ACC.NE.0 .AND. CARRY.EQ.0)PADD=ADDRES                           SIM03200
      GO TO 9999                                                         SIM03210
515   WRITE(6,43)                                                        SIM03220
      READ(5,*)TEST                                                      SIM03230
      IF(ACC.NE.0 .AND. CARRY.EQ.0 .AND. TEST.EQ.1)PADD=ADDRES           SIM03240
      GO TO 9999                                                         SIM03250
102   CONTINUE                                                           SIM03260
C     ASSEMBLER INSTRUCTION IS ... FIM OR SRC                            SIM03270
      COMPUT = PADD+1                                                    SIM03280
      REG=CORE(BANK,2,PADD)                                              SIM03290
      DECIDE=MOD(CORE(BANK,2,PADD),2)                                    SIM03300
```

```
      IF(DECIDE.NE.0)GO TO 83                                          SIM03310
C     ASSEMBLER INSTRUCTION IS ... FIM                                 SIM03320
      INDEX(IRBANK,REG+1)=CORE(BANK,1,COMPUT)                          SIM03330
      INDEX(IRBANK,REG+2)=CORE(BANK,2,COMPUT)                          SIM03340
      CYCLE=CYCLE+1                                                    SIM03350
      SCYCLE=SCYCLE+1                                                  SIM03360
      PADD=PADD+1                                                      SIM03370
      IF((REG+1).LE.8)GO TO 9999                                       SIM03380
      INDEX(NOBANK,REG+1)=INDEX(IRBANK,REG+1)                          SIM03390
      INDEX(NOBANK,REG+2)=INDEX(IRBANK,REG+2)                          SIM03400
      GO TO 9999                                                       SIM03410
C     ASSEMBLER INSTRUCTION IS ... SRC                                 SIM03420
83    CHIP=INDEX(IRBANK,REG)/4 + 1                                     SIM03430
      REGSTR=MOD(INDEX(IRBANK,REG),4) + 1                              SIM03440
      CHAR= INDEX(IRBANK,REG+1) + 1                                    SIM03450
      GO TO 9999                                                       SIM03460
103   CONTINUE                                                         SIM03470
C     ASSEMBLER INSTRUCTION IS ... FIN OR JIN                          SIM03480
      DECIDE=MOD(CORE(BANK,2,PADD),2)                                  SIM03490
      REGIST=CORE(BANK,2,PADD)                                         SIM03500
      IF(DECIDE.EQ.1)GO TO 84                                          SIM03510
      REGIST=REGIST + 1                                                SIM03520
C     ASSEMBLER INSTRUCTION IS ... FIN                                 SIM03530
      PAGE=PADD/256                                                    SIM03540
      ADDRES=256*PAGE + 16*INDEX(IRBANK,1) + INDEX(IRBANK,2) + 1       SIM03550
      INDEX(IRBANK,REGIST)= CORE(BANK,1,ADDRES)                        SIM03560
      INDEX(IRBANK,REGIST+1)=CORE(BANK,2,ADDRES)                       SIM03570
      CYCLE=CYCLE+1                                                    SIM03580
      SCYCLE=SCYCLE+1                                                  SIM03590
      IF(REGIST.LE.8)GO TO 9999                                        SIM03600
      INDEX(NOBANK,REGIST)=INDEX(IRBANK,REGIST)                        SIM03610
      INDEX(NOBANK,REGIST+1)=INDEX(IRBANK,REGIST+1)                    SIM03620
      GO TO 9999                                                       SIM03630
84    PAGE=(PADD + 1)/256                                              SIM03640
C     ASSEMBLER INSTRUCTION IS ... JIN                                 SIM03650
      ADDRES=256*PAGE + 16*INDEX(IRBANK,REGIST) + INDEX(IRBANK,REGIST+1)SIM03660
      PADD=ADDRES                                                      SIM03670
      GO TO 9999                                                       SIM03680
104   CONTINUE                                                         SIM03690
C     ASSEMBLER INSTRUCTION IS ... JUN                                 SIM03700
      CYCLE=CYCLE+1                                                    SIM03710
      SCYCLE=SCYCLE+ 1                                                 SIM03720
      PADD=CORE(BANK,2,PADD)*256+CORE(BANK,1,PADD+1)*16+CORE(BANK,2,PADDSIM03730
     *+1)                                                              SIM03740
      GO TO 9999                                                       SIM03750
105   CONTINUE                                                         SIM03760
C     ASSEMBLER INSTRUCTION IS ... JMS                                 SIM03770
      LOCATE(7)=LOCATE(6)                                              SIM03780
      LOCATE(6)=LOCATE(5)                                              SIM03790
      LOCATE(5)=LOCATE(4)                                              SIM03800
      LOCATE(4)=LOCATE(3)                                              SIM03810
      LOCATE(3)=LOCATE(2)                                              SIM03820
      LOCATE(2)=LOCATE(1)                                              SIM03830
      LOCATE(1)=PADD+1                                                 SIM03840
      PADD=CORE(BANK,2,PADD)*256 + CORE(BANK,1,PADD+1)*16 +            SIM03850
```

```
      *CORE(BANK,2,PADD+1)                                                  SIM03860
      CYCLE=CYCLE+1                                                         SIM03870
      SCYCLE=SCYCLE+1                                                       SIM03880
      GO TO 9999                                                            SIM03890
106   REGIST=CORE(BANK,2,PADD) + 1                                          SIM03900
C     ASSEMBLER INSTRUCTION IS ... INC                                      SIM03910
      INDEX(IRBANK,REGIST)=INDEX(IRBANK,REGIST) + 1                         SIM03920
      IF(INDEX(IRBANK,REGIST).EQ.16)INDEX(IRBANK,REGIST)=0                  SIM03930
      IF(REGIST.LE.8)GO TO 9999                                             SIM03940
      INDEX(NOBANK,REGIST)=INDEX(IRBANK,REGIST)                             SIM03950
      GO TO 9999                                                            SIM03960
107   CONTINUE                                                              SIM03970
C     ASSEMBLER INSTRCUTION IS ... ISZ                                      SIM03980
      CYCLE=CYCLE+1                                                         SIM03990
      SCYCLE=SCYCLE+1                                                       SIM04000
      REGIST=CORE(BANK,2,PADD) + 1                                          SIM04010
      INDEX(IRBANK,REGIST)=INDEX(IRBANK,REGIST)+1                           SIM04020
      IF(REGIST.GT.8)INDEX(NOBANK,REGIST)=INDEX(IRBANK,REGIST)              SIM04030
      IF(INDEX(IRBANK,REGIST).EQ.16)INDEX(IRBANK,REGIST)=0                  SIM04040
      IF(REGIST.GT.8)INDEX(NOBANK,REGIST)=INDEX(IRBANK,REGIST)              SIM04050
      PADD=PADD+1                                                           SIM04060
      IF(INDEX(IRBANK,REGIST).EQ.0)GO TO 9999                               SIM04070
      PAGE=(PADD + 1)/256                                                   SIM04080
      ADDRES=256*PAGE + 16*CORE(BANK,1,PADD) + CORE(BANK,2,PADD)            SIM04090
      PADD=ADDRES                                                           SIM04100
      GO TO 9999                                                            SIM04110
108   REGIST=CORE(BANK,2,PADD) + 1                                          SIM04120
C     ASSEMBLER INSTRUCTION IS ... ADD                                      SIM04130
      ACC=ACC + INDEX(IRBANK,REGIST) + CARRY                                SIM04140
      IF(ACC.GE.16)CARRY=1                                                  SIM04150
      IF(ACC.GE.16)ACC=ACC-16                                               SIM04160
      GO TO 9999                                                            SIM04170
109   CONTINUE                                                              SIM04180
C                                                                           SIM04190
C     ASSEMBLER INSTRUCTION IS ... SUB                                      SIM04200
      REGIST=CORE(BANK,2,PADD) + 1                                          SIM04210
      CINDX=15 - INDEX(IRBANK,REGIST)                                       SIM04220
      IF(CARRY.EQ.0)CHEK=1                                                  SIM04230
      IF(CARRY.EQ.1)CHEK=0                                                  SIM04240
      ACC=ACC + CHEK +CINDX                                                 SIM04250
      CARRY=0                                                               SIM04260
      IF(ACC.GE.16)CARRY=1                                                  SIM04270
      ACC=MOD(ACC,16)                                                       SIM04280
      GO TO 9999                                                            SIM04290
110   REGIST=CORE(BANK,2,PADD) + 1                                          SIM04300
C     ASSEMBLER INSTRUCTION IS ... LD                                       SIM04310
      ACC=INDEX(IRBANK,REGIST)                                              SIM04320
      GO TO 9999                                                            SIM04330
111   REGIST=CORE(BANK,2,PADD) + 1                                          SIM04340
C     ASSEMBLER INSTRUCTION IS ... XCH                                      SIM04350
      IHOLD=ACC                                                             SIM04360
      ACC=INDEX(IRBANK,REGIST)                                              SIM04370
      INDEX(IRBANK,REGIST)=IHOLD                                            SIM04380
      IF(REGIST.GT.8)INDEX(NOBANK,REGIST)=INDEX(IRBANK,REGIST)              SIM04390
      GO TO 9999                                                            SIM04400
```

```
)
  112   ACC=CORE(BANK,2,PADD)                                          SIM04410
  C     ASSEMBLER INSTRUCTION IS ... BBL                               SIM04420
        PADD=LOCATE(1)                                                 SIM04430
        DO 1401 JJJ=1,6                                                SIM04440
  1401  LOCATE(JJJ)=LOCATE(JJJ+1)                                      SIM04450
        GO TO 9999                                                     SIM04460
  113   ACC=CORE(BANK,2,PADD)                                          SIM04470
  C     ASSEMBLER INSTRUCTION IS ... LDM                               SIM04480
        GO TO 9999                                                     SIM04490
  114   CORD=CORE(BANK,2,PADD) + 1                                     SIM04500
        GO TO (300,301,302,302,304,305,306,307,308,309,310,311,312,313,SIM04510
       *314,315),CORD                                                  SIM04520
  115   CORD=CORE(BANK,2,PADD) + 1                                     SIM04530
        GO TO (400,401,402,403,404,405,406,407,408,409,410,411,412,413,SIM04540
       *414,415),CORD                                                  SIM04550
  400   ACC=0                                                          SIM04560
  C     ASSEMBLER INSTRUCTION IS ... CLB                               SIM04570
        CARRY=0                                                        SIM04580
        GO TO 9999                                                     SIM04590
  401   CARRY=0                                                        SIM04600
  C     ASSEMBLER INSTRUCTION IS ... CLC                               SIM04610
        GO TO 9999                                                     SIM04620
  402   ACC=ACC+1                                                      SIM04630
  C     ASSEMBLER INSTRUCTION IS ... IAC                               SIM04640
        IF(ACC.LE.15)GO TO 9999                                        SIM04650
        ACC=0                                                          SIM04660
        CARRY=1                                                        SIM04670
        GO TO 9999                                                     SIM04680
  403   IF(CARRY.EQ.1)C=0                                              SIM04690
  C     ASSEMBLER INSTRUCTION IS ... CMC                               SIM04700
        IF(CARRY.EQ.C)C=1                                              SIM04710
        CARRY=C                                                        SIM04720
        GO TO 9999                                                     SIM04730
  404   ACC=15-ACC                                                     SIM04740
  C     ASSEMBLER INSTRUCTION IS ... CMA                               SIM04750
        GO TO 9999                                                     SIM04760
  405   C=ACC                                                          SIM04770
  C     ASSEMBLER INSTRUCTION IS ... RAL                               SIM04780
        ACC=MOD(ACC,8)*2 + CARRY                                       SIM04790
        CARRY=C/8                                                      SIM04800
        GO TO 9999                                                     SIM04810
  406   C=MOD(ACC,2)                                                   SIM04820
  C     ASSEMBLER INSTRUCTION IS ... RAR                               SIM04830
        ACC=ACC/2 + CARRY*8                                            SIM04840
        CARRY=C                                                        SIM04850
        GO TO 9999                                                     SIM04860
  407   ACC=CARRY                                                      SIM04870
  C     ASSEMBLER INSTRUCTION IS ... TCC                               SIM04880
        CARRY=0                                                        SIM04890
        GO TO 9999                                                     SIM04900
  408   CONTINUE                                                       SIM04910
  C     ASSEMBLER INSTRUCTION IS ... DAC                               SIM04920
        ACC=ACC + 15                                                   SIM04930
        CARRY=0                                                        SIM04940
        IF(ACC.GE.16)CARRY=1                                           SIM04950
```

```
      ACC=MOD(ACC,16)                                                   SIM04960
      GO TO 9999                                                        SIM04970
  409 CONTINUE                                                          SIM04980
C     ASSEMBLER INSTRUCTION IS ... TCS                                  SIM04990
      IF(CARRY.EQ.0)ACC=9                                               SIM05000
      IF(CARRY.EQ.1)ACC=10                                              SIM05010
      CARRY=0                                                           SIM05020
      GO TO 9999                                                        SIM05030
  410 CARRY=1                                                           SIM05040
C     ASSEMBLER INSTRUCTION IS ... STC                                  SIM05050
      GO TO 9999                                                        SIM05060
  411 IF(CARRY.NE.1 .AND.ACC.LE.9)GO TO 9999                            SIM05070
C     ASSEMBLER INSTRUCTION IS ... DAA                                  SIM05080
      ACC=ACC-10                                                        SIM05090
      IF(ACC.GE.6)CARRY=1                                               SIM05100
      IF(ACC.LT.0)ACC=ACC+16                                            SIM05110
      GO TO 9999                                                        SIM05120
  412 C=ACC                                                             SIM05130
C     ASSEMBLER INSTRUCTION IS ... KBP                                  SIM05140
      ACC=15                                                            SIM05150
      IF(C.EQ.0)ACC=0                                                   SIM05160
      IF(C.EQ.1)ACC=1                                                   SIM05170
      IF(C.EQ.2)ACC=2                                                   SIM05180
      IF(C.EQ.4)ACC=3                                                   SIM05190
      IF(C.EQ.8)ACC=4                                                   SIM05200
      GO TO 9999                                                        SIM05210
  413 IF(ACC.GE.8)COMLIN=ACC-7                                          SIM05220
C     ASSEMBLER INSTRUCTION IS ... DCL                                  SIM05230
      IF(ACC.LT.8)COMLIN=ACC + 1                                        SIM05240
      GO TO 9999                                                        SIM05250
  414 CONTINUE                                                          SIM05260
  415 WRITE(6,95)                                                       SIM05270
   95 FORMAT('   ILLEGAL COMMAND HEXIDECIMAL FE OR FF, EXECUTION TERMINASIM05280
     *TING')                                                            SIM05290
      STOP                                                              SIM05300
  300 DATRAM(COMLIN,CHIP,REGSTR,CHAR)=ACC                               SIM05310
C     ASSEMBLER INSTRUCTION IS WRM                                      SIM05320
      GO TO 9999                                                        SIM05330
  301 DATOUT(COMLIN,CHIP)=ACC                                           SIM05340
      WRITE(6,7899)COMLIN,CHIP,DATOUT(COMLIN,CHIP)                      SIM05350
 7899 FORMAT(' THE VALUE FOR RAM PORT # ',I2,' COMLIN # ',I2,' IS ',I2) SIM05360
C     ASSEMBLER INSTRUCTION IS WMP                                      SIM05370
      GO TO 9999                                                        SIM05380
  302 CONTINUE                                                          SIM05390
C     ASSEMBLER INSTRUCTION IS ... WRR                                  SIM05400
      ADDRES=16*((CHIP-1)*4 + REGSTR-1) +CHAR-1                         SIM05410
      WRITE(6,21)ADDRES,ACC                                             SIM05420
   21 FORMAT('   THE OUTPUT VALUE FOR PORT#',I4,' IS',I3)              SIM05430
      GO TO 9999                                                        SIM05440
  303 CONTINUE                                                          SIM05450
C     ASSEMBLER INSTRUCTION IS ... WPM                                  SIM05460
      ADDRES=256*DATOUT(1,1) + 16*((CHIP-1)*4 + REGSTR-1) + CHAR-1+1    SIM05470
      CHEK=ADDRES-1                                                     SIM05480
      IF(B(BANK).GT.CHEK)GO TO 33                                       SIM05490
      CORE(BANK,FLIP,ADDRES)=ACC                                        SIM05500
```

```
)
      IF(FLIP.EQ.1)F=2                                                  SIM05510
      IF(FLIP.EQ.2)F=1                                                  SIM05520
      FLIP=F                                                            SIM05530
      GO TO 9999                                                        SIM05540
 33   WRITE(6,34)BANK,CHEK                                              SIM05550
 34   FORMAT('   PROGRAM BOMBED WPM TRIED TO WRITE IN ROM BANK',I2,' AD SIM05560
     *DRESS ',Z3)                                                       SIM05570
      STOP                                                              SIM05580
 304  DATRAM(COMLIN,CHIP,REGSTR,17)=ACC                                 SIM05590
 C    ASSEMBLER INSTRUCTION IS ... WR0                                  SIM05600
      GO TO 9999                                                        SIM05610
 305  DATRAM(COMLIN,CHIP,REGSTR,18)=ACC                                 SIM05620
 C     ASSEMBLER INSTRUCTION IS WR1                                     SIM05630
      GO TO 9999                                                        SIM05640
 306  DATRAM(COMLIN,CHIP,REGSTR,19)=ACC                                 SIM05650
 C    ASSEMBLER INSTRUCTION IS ... WR2                                  SIM05660
      GO TO 9999                                                        SIM05670
 307  DATRAM(COMLIN,CHIP,REGSTR,20)=ACC                                 SIM05680
 C    ASSEMBLER INSTRUCTION IS ... WR3                                  SIM05690
      GO TO 9999                                                        SIM05700
 308  CONTINUE                                                          SIM05710
 C    ASSEMBLER INSTRUCTION IS ... SBM                                  SIM05720
      CINDX= 15 - DATRAM(COMLIN,CHIP,REGSTR,CHAR)                       SIM05730
      IF(CARRY.EQ.0)CHEK=1                                              SIM05740
      IF(CARRY.EQ.1)CHEK=0                                              SIM05750
      ACC=ACC + CHEK + CINDX                                            SIM05760
      CARRY=0                                                           SIM05770
      IF(ACC.GE.16)CARRY=1                                              SIM05780
      ACC=MOD(ACC,16)                                                   SIM05790
      GO TO 9999                                                        SIM05800
 309  ACC=DATRAM(COMLIN,CHIP,REGSTR,CHAR)                               SIM05810
 C    ASSEMBLER INSTRUCTION IS ... RDM                                  SIM05820
      GO TO 9999                                                        SIM05830
 310  CONTINUE                                                          SIM05840
 C    ASSEMBLER INSTRUCTION IS RDR                                      SIM05850
      ADDRES=16*((CHIP-1)*4 + REGSTR-1) +CHAR-1                         SIM05860
      WRITE(6,20)ADDRES                                                 SIM05870
 20   FORMAT('   INPUT A VALUE FOR INPUT PORT# ',I3)                    SIM05880
      READ(5,*)ACC                                                      SIM05890
      GO TO 9999                                                        SIM05900
 311  ACC=ACC +DATRAM(COMLIN,CHIP,REGSTR,CHAR) + CARRY                  SIM05910
 C    ASSEMBLER INSTRUCTION IS ... ADM                                  SIM05920
      IF(ACC.GE.16)CARRY=1                                              SIM05930
      IF(ACC.GE.16)ACC=ACC-16                                           SIM05940
      GO TO 9999                                                        SIM05950
 312  ACC=DATRAM(COMLIN,CHIP,REGSTR,17)                                 SIM05960
 C    ASSEMBLER INSTRUCTION IS ... RD0                                  SIM05970
      GO TO 9999                                                        SIM05980
 313  ACC=DATRAM(COMLIN,CHIP,REGSTR,18)                                 SIM05990
 C    ASSEMBLER INSTRUCTION IS ... RD1                                  SIM06000
      GO TO 9999                                                        SIM06010
 314  ACC=DATRAM(COMLIN,CHIP,REGSTR,19)                                 SIM06020
 C    ASSEMBLER INSTRUCTION IS ... RD2                                  SIM06030
      GO TO 9999                                                        SIM06040
 315  ACC=DATRAM(COMLIN,CHIP,REGSTR,20)                                 SIM06050
```

FILE: SIM . FORTRAN A1 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
C     ASSEMBLER INSTRUCTION IS ... RD3                                  SIM06060
      GO TO 9999                                                        SIM06C70
      END                                                               SIM06C80
C                                                                       SIM06090
      SUBROUTINE BIT(B4,B3,B2,B1,PASS)                                  SIM06100
      IMPLICIT INTEGER (A - Z)                                          SIM06110
C                                                                       SIM06120
C     THIS SUBROUTINE DISASSEMBLES A HEXADECIMAL DIGIT INTO 4 BITS      SIM06130
C     B4 BEING THE MOST SIGNIFICANT                                     SIM06140
C                                                                       SIM06150
      B4=0                                                              SIM06160
      B3=0                                                              SIM06170
      B2=0                                                              SIM06180
      B1=0                                                              SIM06190
      IF((PASS-7).GE.1)B4=1                                             SIM06200
      CH=PASS - B4*8                                                    SIM06210
      IF((CH-3).GE.1)B3=1                                               SIM06220
      CH=CH -B3*4                                                       SIM06230
      IF((CH-1).GE.1)B2=1                                               SIM06240
      B1=CH-B2*2                                                        SIM06250
      RETURN                                                            SIM06260
      END                                                               SIM06270
```

FILE: ASMLST FORTRAN A1 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
      SUBROUTINE ASMLST                                                 ASMC0010
      IMPLICIT INTEGER (A - Z)                                          ASMC0020
      COMMON /SUB/CORE                                                  ASMC0030
      DIMENSION CORE(2,2,4096)                                          ASMC0040
      DIMENSION LABEL(60)                                               ASMC0050
      DATA LABEL/'NOP','HLT','BBS','LCR','OR4','OR5','AN6','AN7',      ASMC0060
     *'DBC','DB1','SBC','SB1','EIN','DIN','RPM','JCN','FIM','SRC',     ASMC0070
     *'FIN','JIN','JUN','JMS','INC','ISZ','ADD','SUB','LD','XCH','BBL',ASMC0080
     *'LDM','WRM','WMP','WRR','WPM','WR0','WR1','WR2','WR3',            ASMC0090
     *'SBM','RDM','RDR','ADM','RDC','RD1','RD2','RD3','CLB',            ASMC0100
     *'CLC','IAC','CMC','CMA','RAL','RAR','TCC','DAC',                  ASMC0110
     *'TCS','STC','DAA','KBP','DCL'/                                    ASMC0120
 900  FORMAT(38X,'ADDRESS   STMT  STATEMENT')                           ASMC0130
 901  FORMAT(110X,'PAGE ',I3)                                           ASMC0140
 902  FORMAT(110X,'BANK ',I3)                                           ASMC0150
 903  FORMAT(40X,Z3,5X,I4,5X,A3)                                        ASMC0160
 904  FORMAT(40X,Z3,5X,I4,5X,A3,2X,'RP',I2)                             ASMC0170
 905  FORMAT(40X,Z3,5X,I4,5X,A3,2X,'R ',I2)                             ASMC0180
 906  FORMAT(40X,Z3,5X,I4,5X,A3,2X,Z1,'H')                              ASMC0190
 907  FORMAT(40X,Z3,5X,I4,5X,A3,Z1,1X,2Z1)                              ASMC0200
 908  FORMAT(40X,Z3,5X,I4,5X,A3,2X,'RP',I2,',A''',2Z1,'''')             ASMC0210
 909  FORMAT(40X,Z3,5X,I4,5X,A3,2X,3Z1)                                 ASMC0220
 910  FORMAT(40X,Z3,5X,I4,5X,A3,2X,'R ',I2,',A''',2Z1,'''')             ASMC0230
 911  FORMAT(40X,'*** ERROR ***')                                       ASMC0240
      LASADD=0                                                          ASMC0250
      ERRORS=0                                                          ASMC0260
      PADD=0                                                            ASMC0270
      BANK=1                                                            ASMC0280
      STMT=0                                                            ASMC0290
      ADDRES=-1                                                         ASMC0300
      PKOUNT=0                                                          ASMC0310
      PAGE=1                                                            ASMC0320
 6666 CHEKK=CORE(BANK,1,1)*16 + CORE(BANK,2,1)                          ASMC0330
      IF(CHEKK.EQ.15)GO TO 215                                          ASMC0340
      WRITE(6,901)PAGE                                                  ASMC0350
      WRITE(6,902)BANK                                                  ASMC0360
      WRITE(6,900)                                                      ASMC0370
 9999 CONTINUE                                                          ASMC0380
      BRA=0                                                             ASMC0390
      CPAGE=MOD(PADD,256)                                               ASMC0400
      IF(CPAGE.EQ.0 .OR. (CPAGE.EQ.1 .AND. LASADD.EQ.1))BRA=1           ASMC0410
      LASADD=0                                                          ASMC0420
      IF(BRA.EQ.0)GO TO 50                                              ASMC0430
      CPAGE=PADD/256                                                    ASMC0440
      WRITE(6,899)CPAGE                                                 ASMC0450
  899 FORMAT(/,10X,'CORE PAGE',I3,/)                                    ASMC0460
      PCOUNT=PCOUNT + 3                                                 ASMC0470
   50 IF(PCOUNT.LE.55)GO TO 83                                          ASMC0480
      WRITE(6,90)                                                       ASMC0490
   90 FORMAT(1H1)                                                       ASMC0500
      PAGE=PAGE + 1                                                     ASMC0510
      WRITE(6,901)PAGE                                                  ASMC0520
      WRITE(6,902)BANK                                                  ASMC0530
      WRITE(6,900)                                                      ASMC0540
      PCOUNT=0                                                          ASMC0550
```

```
 83     PCOUNT=PCOUNT + 1                                                ASMC0560
        ADDRES=ADDRES+1                                                  ASMC0570
        PADD=PADD+1                                                      ASMC0580
        IF(PADD.GE.4097)GO TO 215                                        ASMC0590
        STMT=STMT+1                                                      ASMC0600
        CORD=CORE(BANK,1,PADD)+1                                         ASMC0610
        GO TO (100,101,102,103,104,105,106,107,108,109,110,111,112,113,  ASMC0620
       *114,115),CORD                                                    ASMC0630
 100    CORD=CORE(BANK,2,PADD) + 1                                       ASMC0640
        GO TO (200,200,200,200,200,200,200,200,200,200,200,200,200,200,200ASMC0650
       *,215),CORD                                                       ASMC0660
 200    WRITE(6,903)ADDRES,STMT,LABEL(CORD)                              ASMC0670
        GO TO 9999                                                       ASMC0680
 215    IF(CHEKK.EQ.15)GO TO 54                                          ASMC0690
        WRITE(6,977)                                                     ASMC0700
 977    FORMAT(59X,'*** END OF JOB ***')                                 ASMC0710
        WRITE(6,53)ERRORS                                                ASMC0720
 53     FORMAT(5X,I3,' ERRORS WERE DETECTED THIS PASS')                  ASMC0730
 54     IF(BANK.EQ.2)GO TO 7777                                          ASMC0740
        BANK=2                                                           ASMC0750
        STMT=0                                                           ASMC0760
        ADDRES=-1                                                        ASMC0770
        PCOUNT=0                                                         ASMC0780
        PAGE=1                                                           ASMC0790
        PADD=0                                                           ASMC0800
        ERRORS=0                                                         ASMC0810
        GO TO 6666                                                       ASMC0820
 101    LASADD=1                                                         ASMC0830
        WRITE(6,907)ADDRES,STMT,LABEL(16),CORE(BANK,2,PADD),CORE(BANK,1  ASMC0840
       *,PADD+1),CORE(BANK,2,PADD+1)                                     ASMC0850
        PADD=PADD+1                                                      ASMC0860
        ADDRES=ADDRES+1                                                  ASMC0870
        GO TO 9999                                                       ASMC0880
 102    CHK=MOD(CORE(BANK,2,PADD),2)                                     ASMC0890
        RP=(CORE(BANK,2,PADD)/2)*2                                       ASMC0900
        IF(CHK.EQ.1)GO TO 85                                             ASMC0910
        WRITE(6,908)ADDRES,STMT,LABEL(17),RP,CORE(BANK,1,PADD+1),        ASMC0920
       *CORE(BANK,2,PADD+1)                                              ASMC0930
        ADDRES=ADDRES+1                                                  ASMC0940
        PADD=PADD+1                                                      ASMC0950
        LASADD=1                                                         ASMC0960
        GO TO 9999                                                       ASMC0970
 85     WRITE(6,904)ADDRES,STMT,LABEL(18),RP                             ASMC0980
        GO TO 9999                                                       ASMC0990
 103     CHK=MOD(CORE(BANK,2,PADD),2)                                    ASMC1000
        RP=(CORE(BANK,2,PADD)/2)*2                                       ASMC1010
        IF(CHK.EQ.1)GO TO 86                                             ASMC1020
        WRITE(6,904)ADDRES,STMT,LABEL(19),RP                             ASMC1030
        GO TO 9999                                                       ASMC1040
 86     WRITE(6,904)ADDRES,STMT,LABEL(20),RP                             ASMC1050
        GO TO 9999                                                       ASMC1060
 104    WRITE(6,909)ADDRES,STMT,LABEL(21),CORE(BANK,2,PADD),CORE(BANK,1, ASMC1070
       *PADD+1),CORE(BANK,2,PADD+1)                                      ASMC1080
        PADD=PADD+1                                                      ASMC1090
        ADDRES=ADDRES+1                                                  ASMC1100
```

FILE: ASMLST FORTRAN A1 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
      LASACD=1                                                          ASM01110
      GO TO 9999                                                        ASM01120
105   WRITE(6,909)ACDRES,STMT,LABEL(22),CORE(BANK,2,PACD),CORE(BANK,1,  ASM01130
     *PACD+1),CORE(BANK,2,PACD+1)                                       ASM01140
      PACD=PACD+1                                                       ASM01150
      ACDRES=ACDRES+1                                                   ASM01160
      LASACD=1                                                          ASM01170
      GO TO 9999                                                        ASM01180
106   WRITE(6,905)ACDRES,STMT,LABEL(23),CORE(BANK,2,PACD)               ASM01190
      GO TO 9999                                                        ASM01200
107   WRITE(6,910)ACDRES,STMT,LABEL(24),CORE(BANK,2,PACD),CORE(BANK,1   ASM01210
     *,PACD+1),CORE(BANK,2,PACD+1)                                      ASM01220
      PACD=PACD+1                                                       ASM01230
      ACDRES=ACDRES+1                                                   ASM01240
      LASACD=1                                                          ASM01250
      GO TO 9999                                                        ASM01260
108   CONTINUE                                                          ASM01270
109   CONTINUE                                                          ASM01280
110   CONTINUE                                                          ASM01290
111   WRITE(6,905)ACDRES,STMT,LABEL(CORD+16),CORE(BANK,2,PACD)          ASM01300
      GO TO 9999                                                        ASM01310
112   WRITE(6,906)ACDRES,STMT,LABEL(29),CORE(BANK,2,PACD)               ASM01320
      GO TO 9999                                                        ASM01330
113   WRITE(6,906)ACDRES,STMT,LABEL(30),CORE(BANK,2,PACD)               ASM01340
      GO TO 9999                                                        ASM01350
114   CORD=CORE(BANK,2,PADD)+31                                         ASM01360
      WRITE(6,903)ACDRES,STMT,LABEL(CORD)                               ASM01370
      GO TO 9999                                                        ASM01380
115   IF(CORE(BANK,2,PADD).LT.14)GO TO 116                              ASM01390
      WRITE(6,911)                                                      ASM01400
      ERRORS=ERRORS+1                                                   ASM01410
      GO TO 9999                                                        ASM01420
116   CORD=CORE(BANK,2,PACD)+47                                         ASM01430
      WRITE(6,903)ACDRES,STMT,LABEL(CORD)                               ASM01440
      GO TO 9999                                                        ASM01450
7777  RETURN                                                            ASM01460
      END                                                               ASM01470
```

FILE: CORLD . FORTRAN A1 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
      SUBROUTINE CORLC                                                  CORC0010
      IMPLICIT INTEGER (A - Z)                                          CORC0020
      COMMON/SUB/ CORE                                                  CORC0030
      DIMENSION CORE(2,2,4096)                                          CORC0040
      DO 1 K=1,2                                                        CORC0050
      DO 1 J=1,4096                                                     CORC0060
      CORE(K,1,J)=0                                                     CORC0070
1     CORE(K,2,J)=15                                                    CORC0080
      WRITE(6,2)                                                        CORC0090
2     FORMAT(' ENTER UNIT NUMBER WHERE PROGRAM IS TO BE LOADED')        CORC0100
      READ(5,*)INFILE                                                   CORC0110
      DO 3 J=1,2                                                        CORC0120
      DO 4 K=1,4096                                                     CORC0130
      READ(INFILE,5,END=3)CORE(J,1,K),CORE(J,2,K)                       CORC0140
5     FORMAT(2Z1)                                                       CORC0150
4     CONTINUE                                                          CORC0160
3     INFILE=INFILE + 1                                                 CORC0170
      RETURN                                                            CORC0180
      END                                                               CORC0190
```

APPENDIX D

SIMULATOR SAMPLE OUTPUT

```
                        ADDRESS    STMT  STATEMENT

CORE PAGE  0

                           000        1      EIM   PP 2,A'00'
                           002        2      SPC   PP 2
                           003        3      PDP
                           004        4      CMA
                           005        5      WRP
                           006        6      XCH   R  0
                           007        7      IDM   7H
                           008        8      XCH   R  3
                           009        9      SPC   PP 2
                           00A       10      PDP
                           00B       11      CMA
                           00C       12      WRP
                           00D       13      XCH   R  1
                                                  *** END OF JOB ***
0 ERRORS WERE DETECTED THIS PASS
```

```
 ENTER THE STARTING POSITION FOR RAM IN BANK1 THEN BANK 2
?
0 0
 INPUT A VALUE FOR INPUT PORT#   0
?
3
 THE OUTPUT VALUE FOR PORT#   0 IS 12
 INPUT A VALUE FOR INPUT PORT#   1
?
4
 THE OUTPUT VALUE FOR PORT#   1 IS 11 .
```

REGISTER DUMP

SIMULATION REQUIRED 15CYCLES
THIS IS APPROXIMATELY 0.16200000E-03 SECONDS CPU TIME

REGISTER 0 1 2 3 4 5 6 7 8 9 A B C D E F
BANK 1 0C0B0001000000000000000000000000
BANK 2 00000000000000000000000000000000

THE VALUE OF THE ACCUMULATOR IS ...00

THE VALUE OF THE CARRY FLIP-FLOP IS ...00

RAM OUTPUT DUMP

CHIP NUMBER	0	1	2	3
COMMAND LINE 0	00000000			
COMMAND LINE 1	00000000			
COMMAND LINE 2	00000000			
COMMAND LINE 3	00000000			

```
                                        ADDRESS REGISTER DUMP
ADDRESS STACK LEVEL NUMBER  1 IS ADDRESS 000
ADDRESS STACK LEVEL NUMBER  2 IS ADDRESS 000
ADDRESS STACK LEVEL NUMBER  3 IS ADDRESS 000
ADDRESS STACK LEVEL NUMBER  4 IS ADDRESS 000
ADDRESS STACK LEVEL NUMBER  5 IS ADDRESS 000
ADDRESS STACK LEVEL NUMBER  6 IS ADDRESS 000
ADDRESS STACK LEVEL NUMBER  7 IS ADDRESS 000

 DO YOU WANT A DATA RAM DUMP??  1=YES,0=NO
?
1
```

DATA RAM DUMP

```
RAM ADDRESS          0 1 2 3 4 5 6 7 8 9 A B C D E F
COMMAND LINE NUMBER 0
CHIP 0 REGISTER 0000000000000000000000000000000000
CHIP 0 REGISTER 1000000000000000000000000000000000
CHIP 0 REGISTER 2000000000000000000000000000000000
CHIP 0 REGISTER 3000000000000000000000000000000000
CHIP 1 REGISTER 0000000000000000000000000000000000
CHIP 1 REGISTER 1000000000000000000000000000000000
CHIP 1 REGISTER 2000000000000000000000000000000000
CHIP 1 REGISTER 3000000000000000000000000000000000
CHIP 2 REGISTER 0000000000000000000000000000000000
CHIP 2 REGISTER 1000000000000000000000000000000000
CHIP 2 REGISTER 2000000000000000000000000000000000
CHIP 2 REGISTER 3000000000000000000000000000000000
CHIP 3 REGISTER 0000000000000000000000000000000000
CHIP 3 REGISTER 1000000000000000000000000000000000
CHIP 3 REGISTER 2000000000000000000000000000000000
CHIP 3 REGISTER 3000000000000000000000000000000000
```

APPENDIX E

ASSEMBLER TEST OUTPUT

```
                    ADDRESS   STMT   STATEMENT
ORE PAGE  0

                      000       1    TOP   NOP
                      001       2          JCN   ATOP
                      003       3          FIM   0    C0
                      005       4          SRC   2
                      006       5          FIM   0
                      008       6          FIN   0
                      009       7          JIN   2
                      00A       8          JUN    TOP
                      00C       9          JMS    TOP
                      00E      10          INC   F
                      00F      11          ISZ   ETOP
                      010      12          ADD   5
                      011      13          SUB   7
                      012      14          LD    9
                      013      15          XCH   7
                      014      16          BBL   1
                      015      17          LDM   9
                      016      18          CLB
                      017      19          CLC
                      018      20          IAC
                      019      21          CMC
                      01A      22          CMA
                      01B      23          RAL
                      01C      24          RAR
                      01D      25          TCC
                      01E      26          DAC
                      01F      27          TCS
                      020      28          STC
                      021      29          DAA
                      022      30          KBP
                      023      31          DCL
                      024      32          HLT
                      025      33          BBS
                      026      34          LCR
                      027      35          OR4
                      028      36          OR5
                      029      37          AN6
                      02A      38          AN7
                      02B      39          DB0
                      02C      40          DB1
                      02D      41          SB0
                      02E      42          SB1
                      02F      43          EIN
                      030      44          DIN
                      031      45          RPM
                      032      46          WRM
                      033      47          WMP
                      034      48          WRR
                      035      49          WPM
                      036      50          WR0
                      037      51          WR1
                      038      52          WR2
                      039      53          WR3
```

ADDRESS	STMT	STATEMENT
03A	54	SBM
03B	55	RDM
03C	56	RDR
03D	57	ADM
03E	58	RD0
03F	59	RD1
040	60	RD2
041	61	RD3
042	62	END

```
TOP   NOP
      JCN   ATOP
      FIM   0    C0
      SRC   2
      FIM   0
      FIN   0
      JIN   2
      JUN    TOP
      JMS    TOP
      INC   F
      ISZ   ETOP
      ADD   5
      SUB   7
      LD    9
      XCH   7
      BBL   1
      LDM   9
      CLB
      CLC
      IAC
      CMC
      CMA
      RAL
      RAR
      TCC
      DAC
      TCS
      STC
      DAA
      KBP
      DCL
      HLT
      BBS
      LCR
      OR4
      OR5
      AN6
      AN7
      DB0
      DB1
      SB0
      SB1
      EIN
      DIN
      RPM
      WRM
      WMP
      WRR
      WPM
      WR0
      WR1
      WR2
      WR3
      SBM
      RDM
```

```
RDR
ACM
RD0
RD1
RD2
RD3
END
```

FILE: DUMMY DU A

C0
1A
C0
20
C0
23
20
C0
30
33
40
C0
50
C0
6F
7E
C0
85
97
A9
B7
C1
D9
F0
F1
F2
F3
F4
F5
F6
F7
F8
F9
FA
FB
FC
FD
01
C2
03
04
C5
C6
C7
C8
09
CA
0B
0C
0D
0E
E0
E1
E2
E3

FILE: DUMMY DU A

E4
E5
E6
E7
E8
E9
EA
EB
EC
ED
EE
EF

APPENDIX F

MOTOR SPEED CONTROL SAMPLE PROGRAM

FILE: DATA FILE A

```
TOP   FIM  0   C0
      SRC  0
      RDR
      XCH  0
      FIM  2   01
      SRC  2
LOP   CLB
      RDR
      SUB  0
      CMC
      TCC
      WRR
      JUN   LOP
      END
```

```
                ADDRESS    STMT    STATEMENT

CORE PAGE  0

                000          1     TOP   FIM   0     C0
                002          2           SRC   0
                003          3           RDR
                004          4           XCH   0
                005          5           FIM   2     01
                007          6           SRC   2
                008          7     LOP   CLB
                009          8           RDR
                00A          9           SUB   0
                00B         10           CMC
                00C         11           TCC
                00D         12           WRR
                00E         13           JUN   LOP
                010         14           END
```

```
 INPUT A VALUE FOR INPUT PORT#   0
?
5
 INPUT A VALUE FOR INPUT PORT#   1
?
4
 THE OUTPUT VALUE FOR PORT#   1 IS  1
 INPUT A VALUE FOR INPUT PORT#   1
?
4

 THE OUTPUT VALUE FOR PORT#   1 IS  1
 INPUT A VALUE FOR INPUT PORT#   1
?
6

 THE OUTPUT VALUE FOR PORT#   1 IS  0
 INPUT A VALUE FOR INPUT PORT#   1
?
6

 THE OUTPUT VALUE FOR PORT#   1 IS  0
 INPUT A VALUE FOR INPUT PORT#   1
?
7

 THE OUTPUT VALUE FOR PORT#   1 IS  0
 INPUT A VALUE FOR INPUT PORT#   1
?
?

 THE OUTPUT VALUE FOR PORT#   1 IS  1
 INPUT A VALUE FOR INPUT PORT#   1
?
```

BIBLIOGRAPHY

BOOKS

Raphael, Howard A. Intel MCS-40 USER's Manual for Logic Designs, Santa Clara, CA: Intel Corporation, 1974.

"4040 Assembly Language Programming Manual", Intel Corporation, 1974.

"8080 PLM Compiler Operators Manual", Intel Corporation, 1976.

ARTICLES

Ogdin J. and McPhillips S., "Processor Selection", New Logic Notebook, September, 1974.

Raphael, Howard A., "Motor Control by PLL", ELECTRONIC DESIGN, April 26, 1975.

"Put a Complete Microcomputer in your System for Less than "$30.", Intel Corporation, 1975.

Raphael, Howard A., "Microcomputers, Computers on a Chip", Intel Corporation, 1975.

Kildall, Gary A., "High-level Language Simplifies Microcomputer Programming", ELECTRONICS, June 27, 1974.