

To the Graduate Council:

I am submitting herewith a thesis written by Mrugesh Popatlal Trivedi entitled "Diagnostics of VLSI Floating-Point Processors." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin
Donald W. Bouldin, Major Professor

We have read this thesis and
recommend its acceptance:

Robert E. Bodenstein
Robert W. Rochelle

Accepted for the Council:

Levminkel
The Graduate School

DIAGNOSTICS OF VLSI FLOATING-POINT PROCESSORS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Mrugesh Popatlal Trivedi

August 1984

DEDICATION

This thesis is dedicated to my parents without whose love and encouragement I would have never been able to pursue my studies in this country and to complete my thesis. I can never forget the sacrifice my wife has given to spare me for the thesis work when she needed me the most. This is a small token of my appreciation for those who were the constant source of inspiration for this accomplishment.

ACKNOWLEDGMENTS

The author is grateful to Dr. Donald W. Bouldin, the major advisor, for his continuous encouragement and assistance throughout the research and writing of this thesis. Appreciation is expressed to Dr. Robert Rochelle and Dr. Robert Bodenheimer for serving on my thesis committee.

Special acknowledgment is given to Mr. Seiko Harimoto for his patience in explaining the DX10 operating system which not only helped this author understand the system but also helped accelerate the research.

ABSTRACT

The purpose of this project was to develop a strategy for testing the 32-bit operation of three types of single-chip floating-point processors (adder/subtractor, multiplier and divider) and subsequently to implement a procedure successfully.

The primary design goal was to build a testing circuit with minimum external hardware so as to minimize the sources of errors and to enable the user to pinpoint the faulty chip immediately. The second goal was to keep the test hook-up small and handy, so that the user can carry out the testing easily.

The test hook-up developed employs the TM 990/U89 microcomputer from Texas Instruments, an interfacing circuit with SN74S373 latches, and a floating-point chip. Uniformly distributed random test data are applied to the input of a chip and the output obtained is compared with the desired values to verify the operation of the chip.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	1
II. HEWLETT-PACKARD FLOATING-POINT PROCESSORS	3
Data Format	5
Socket Diagram and Pin Assignment	5
Floating-Point Add/Subtract Chip	5
Functional Sequence	12
Floating-Point Multiply Chip	14
Functional Sequence	17
Floating-Point Divide Chip	19
Functional Sequence	22
III. HARDWARE	25
Description of Hardware	27
IV. SOFTWARE	33
Development of Software	34
Development of Master Program	37
General Purpose Programs	44
Floating-Point Processor Programs	45
Adder	45
Fixed to floating conversion	45
Add operation	47
Multiplier	48
Divider	50
User's Instructions for Emulator of TI 990	50
Results of Experiments	52
V. SUMMARY	54
Future Modifications	55
LIST OF REFERENCES	56
APPENDIXES	58
Appendix A.1	59
Appendix A.2	60
Appendix A.3	63
Appendix A.4	65

PAGE

Appendix A.5	67
Appendix A.6	76
Appendix A.7	78
Appendix A.8	80
Appendix A.9	82
Appendix A.10	84
Appendix A.11	86
Appendix A.12	89
Appendix A.13	91
Appendix A.14	93
Appendix A.15	96
VITA	98

LIST OF TABLES

TABLE	PAGE
2.1. Pin Assignments of HP Chip	8
3.1. Wiring Scheme of CRU Bits	29
3.2. Wiring Scheme of Latches	31
3.3. Wiring Scheme of 40-Pin Socket Connector	32
4.1. Histogram for Adder	36
4.2. Histogram for Multiplier	36
4.3. Function Sequence for 32-Bit Add Operation	38
4.4. Alternate Schemes for CRU Byte Assignment	39
4.5. CRU Byte Assignment	40
4.6. Microprogramming for Adder	41
4.7. Summary of Programs	43
4.8. Function Sequence for Fixed to Floating Conversion	46
4.9. Function Sequence for Multiply Operation	49
4.10. Function Sequence for Divide Operation	51

LIST OF FIGURES

FIGURE	PAGE
2.1. Common Block Diagram	6
2.2. Socket Diagram and Pin Assignment	7
2.3. Microphotograph of Add/Subtract Chip	9
2.4. Block Diagram of Add/Subtract Chip	10
2.5. Function Sequence for Add Operation	13
2.6. Microphotograph of Multiply Chip	15
2.7. Block Diagram of Multiply Chip	16
2.8. Function Sequence for Multiply Operation	18
2.9. Microphotograph of Divide Chip	20
2.10. Block Diagram of Divide Chip	21
2.11. Function Sequence for Divide Operation	23
3.1. Basic Wiring Diagram	28

CHAPTER I

INTRODUCTION

Advances in integrated circuit technology have led to the development of devices containing tens and even hundreds of thousands of transistors on a single chip. Along with this increase in complexity there has been an increase in the testing requirements of these circuits. The functional density and complexity of most digital equipment is at a point where testing at a logic gate level is not permissible. Various instruments like Signature Analyzers, Waveform Digitizers, and Logic Analyzers are in use for testing such circuits.

Sometimes when enough data are not available for testing the circuits with these instruments, an individualized approach is preferred, as was the case for this project. Hewlett-Packard had sent some 64-bit floating-point processors to the Electrical Engineering Department for which much performance data were not available. It was known that they had some masking error and that 64-bit operation was not guaranteed. However, 32-bit operation still might be satisfactory. The interest was to check the 32-bit operation of these chips, since the department is planning to build a 32-bit floating-point array processor using these chips. This project was undertaken mainly for this reason.

Chapter II briefly describes the operation of HP floating-point processors and their floating-point format. The discussion

also includes the function sequences and the timing diagrams for the Adder, Multiplier, and Divider.

Chapter III deals with the hardware part of the project and discusses the design goals. It also states what other schemes were considered and why this particular one was selected.

Chapter IV is a very important chapter dealing with the software part of the project. Step by step development of the software is presented in this chapter which also covers the criterion of testing, reasons for not choosing exhaustive testing, selecting uniform distributed random data and their histograms, flow diagrams of different software and the user's instructions.

Chapter V, the concluding chapter, presents a summary and conclusions. Further research suggestions are also given in this chapter.

CHAPTER II

HEWLETT-PACKARD FLOATING-POINT PROCESSORS

Hewlett-Packard has recently developed a set of three monolithic floating-point processor chips (an adder/subtractor, a multiplier, and a divider) that can be used by a variety of computer and instrument systems [1,2].

One major application of the chip set consists of a floating-point processor for a general purpose minicomputer system. The chip set is placed in parallel to the primary data path of the CPU and controlled by a microprogram. The add and multiply chip can perform 32-bit scalar operations in about 750 ns.

It is also possible to operate multiple chip sets in an interleaved manner, and thereby create a pipelined processor capable of executing vector (multiple operation) instructions in the minicomputer system. The add and multiply chips are able to overlap operations, and each provides a 32-bit processing bandwidth of two MFLOPs (millions of floating-point operations per second).

The chip set provides the system designer with simple building blocks which may be used to construct complex processing systems. Pipelined processors can be customized for the data flow of specific algorithms, maximizing the hardware efficiency for specialized applications. Such applications include signal and image processing, graphics processing, and linear system decomposition. Virtually any

structured algorithm can benefit from the chip set in the implementation of high performance floating-point systems.

The primary design goal for the chip set was maximum scalar performance, which was achieved by using combinational data path logic. It was also necessary to perform three-chip partitioning so that each of the fundamentally different floating-point operations could be optimized with its own data path. The chip set has about 20 to 30 times the performance of commercially-available 64-bit floating-point processors.

The second design goal was to maximize the ease-of-use of the chip set. The chip set has the following features: identical pin assignments for input output, power and ground connections (64 pins/chip), simple control requirements, static, single-clock edge requirement, TTL compatible inputs and outputs, power consumption of about 400 mW per chip, three 16-bit, 12-MHZ data buses, and tristate output pins. The chip set handles the 32-bit and 64-bit floating point and 32-bit fixed point data types. Operations provided include add, subtract, data type conversion, multiply, divide, and N-bit shifting.

The 32- and 64-bit operands and results must be time multiplexed for loading and unloading. Operands and the function opcode are loaded into a chip on two or four clock edges. The data propagate through combinational data path logic, and they may be directly unloaded in another two or four clock cycles. Results may alternatively be passed through a pipeline register allowing two operations to be overlapped. Unloading this register requires two

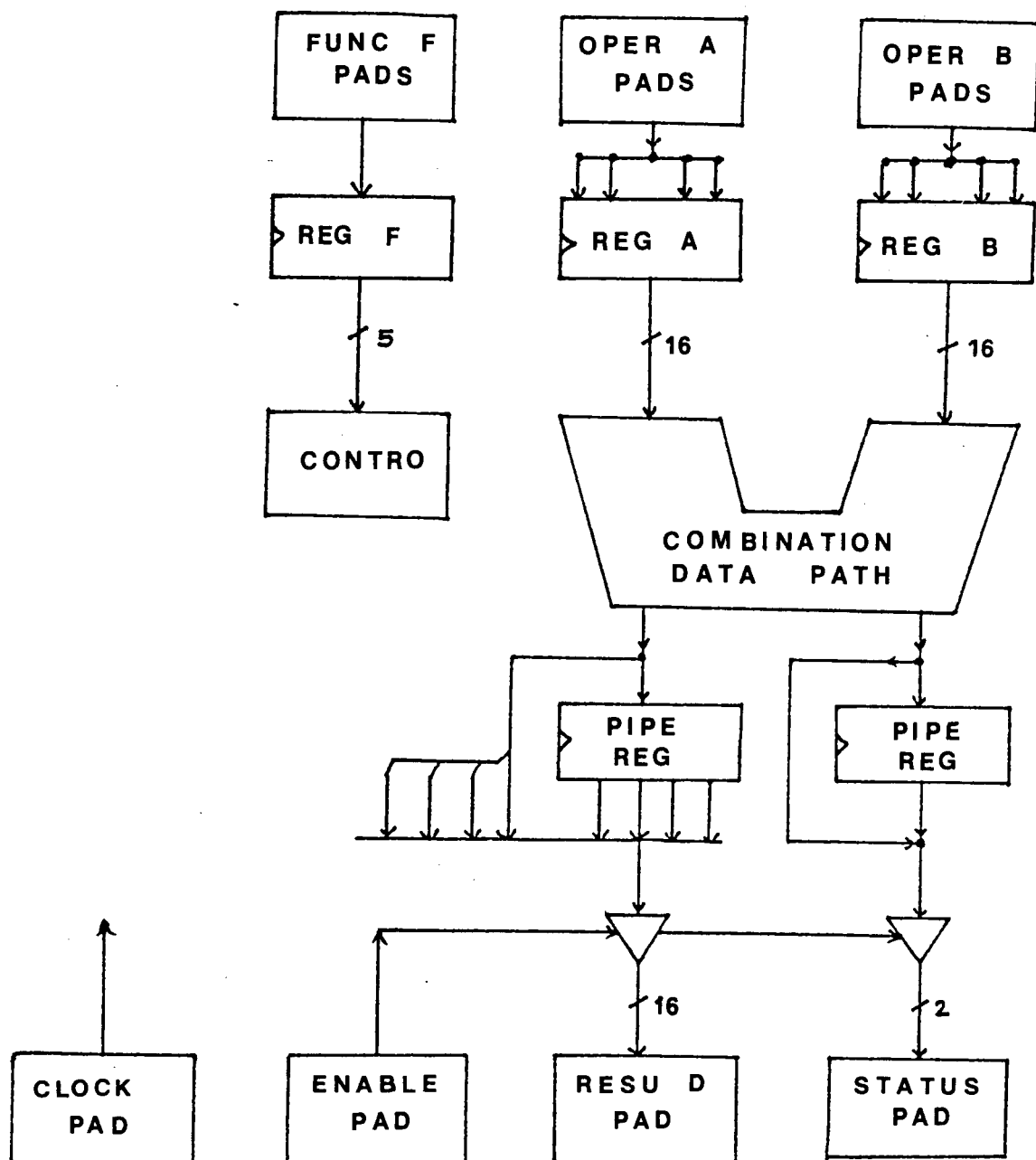
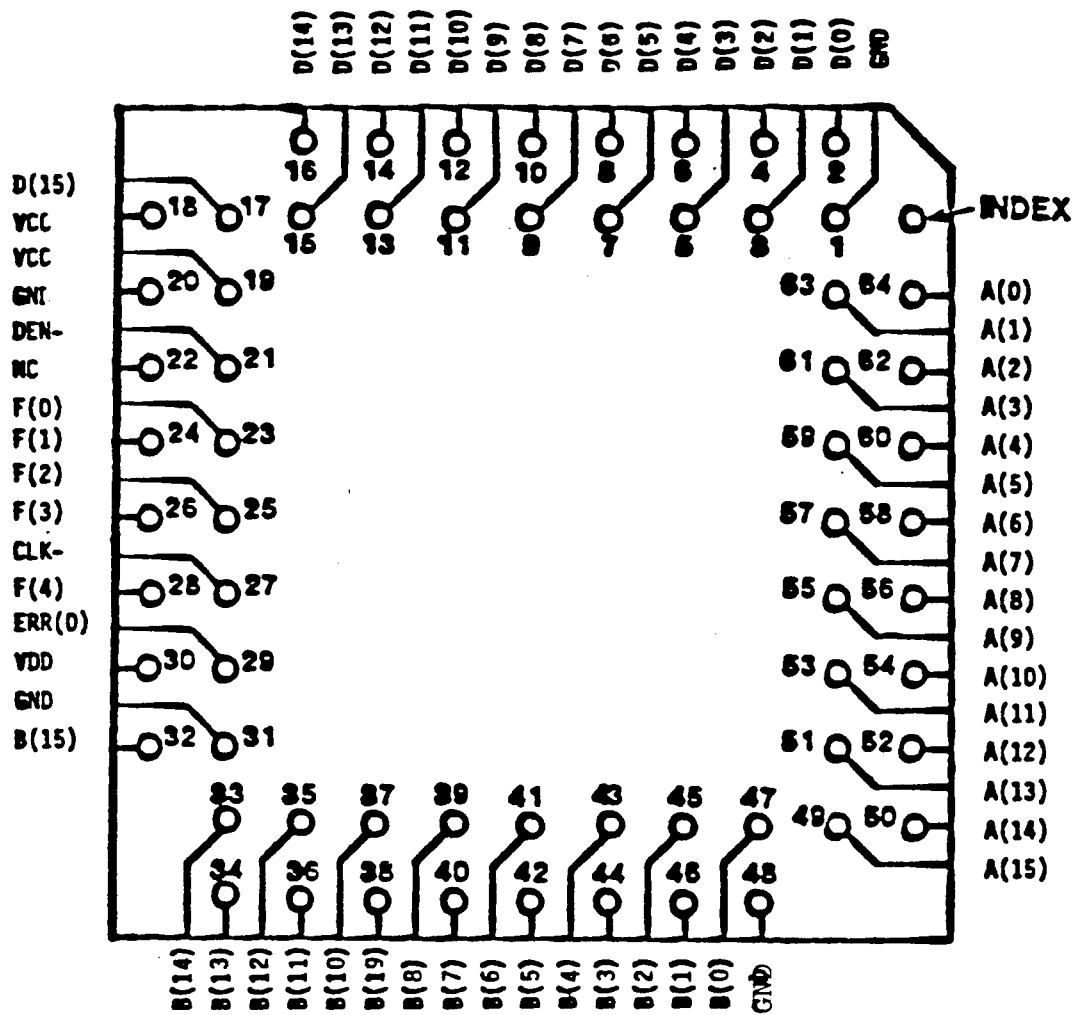


Figure 2.1. Common block diagram.



SOCKET/BOARD TOP VIEW

Figure 2.2. Socket diagram and pin assignment.

Table 2.1
Pin Assignments of HP Chip

Pin #	Type	Name	Pin #	Type	Name	Pin #	Type	Name
1	Power	GND	23	Input	F0	45	Input	B2
2	Output	D0	24	Input	F1	46	Input	B1
3	Output	D1	25	Input	F2	47	Input	B0
4	Output	D2	26	Input	F3	48	Power	GND
5	Output	D3	27	Input	CLK	49	Input	A15
5	Output	D4	28	Input	F4	50	Input	A14
7	Output	D5	29	Output	ERO	51	Input	A13
8	Output	D6	30	Power	Vdd	52	Input	A12
9	Output	D7	31	Power	GND	53	Input	A11
10	Output	D8	32	Input	B15	54	Input	A10
11	Output	D9	33	Input	B14	55	Input	A9
12	Output	D10	34	Input	B13	56	Input	A8
13	Output	D11	35	Input	B12	57	Input	A7
14	Output	D12	36	Input	B11	58	Input	A6
15	Output	D13	37	Input	B10	59	Input	A5
16	Output	D14	38	Input	B9	60	Input	A4
17	Output	D15	39	Input	B8	61	Input	A3
18	Power	Vcc	40	Input	B7	62	Input	A2
19	Power	Vcc	41	Input	B6	63	Input	A1
20	Power	GND	42	Input	B5	64	Input	A0
21	Input	DEN	43	Input	B4			
22	Not Used		44	Input	B3			

Power requirements:

Vcc : 5.0 Volt
 Vdd : 12.0 Volt
 Icc : 40 mA
 Power: 400 mW

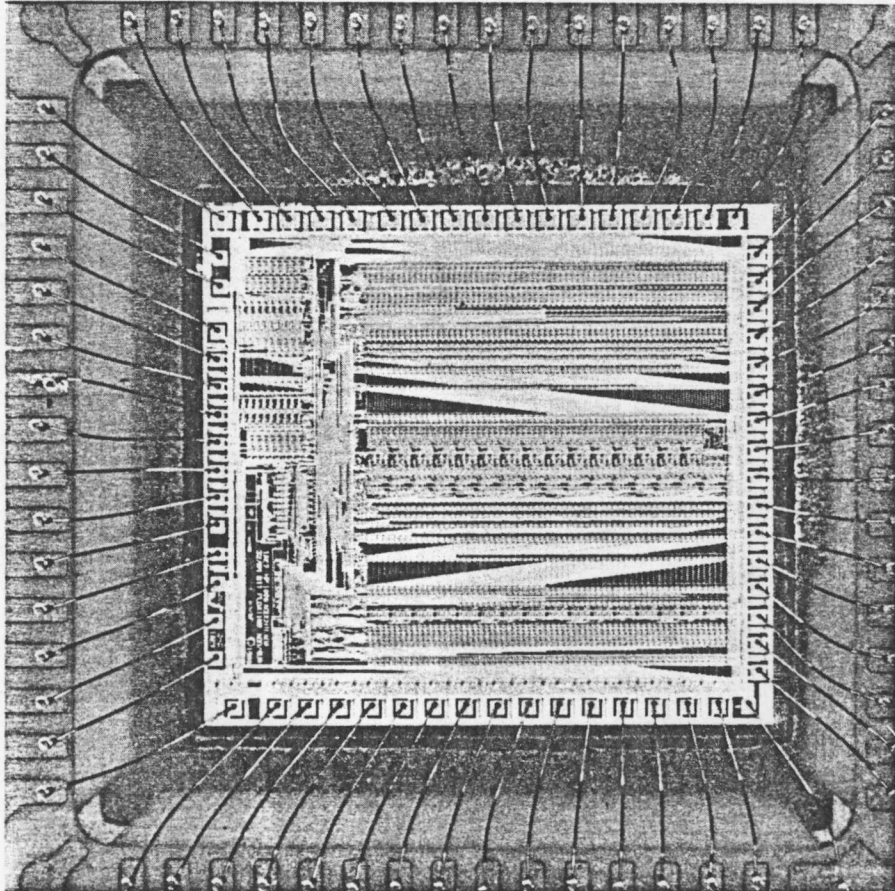


Figure 2.3. Microphotograph of add/subtract chip.

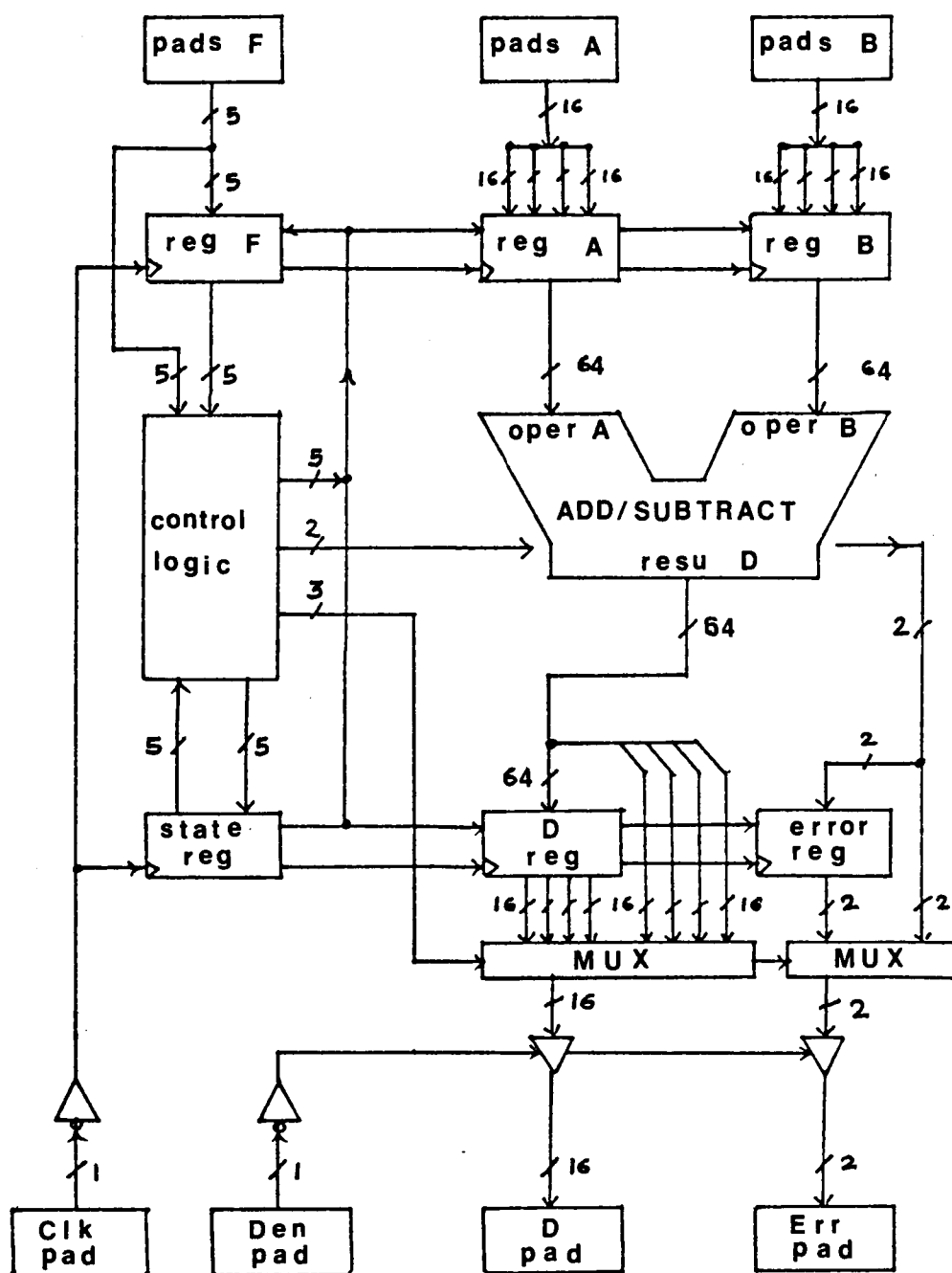


Figure 2.4. Block diagram of add/subtract chip.

structures in this path are operand registers, right shifters, adder and priority encoder, left shifter and incrementer, and output register. The exponent data path is in the upper left and the control logic in the lower left. Operand input pads are along the top and right side, and the result pads are along the left side.

When two operands are first loaded, their exponents are compared and the larger one is passed on. The fraction with the smaller exponent is right shifted by N bits, where N is the difference of the two exponents. The right shift logic consists of six two-to-one multiplexers/bit, and the wiring to perform power-of-two shifts.

Next, the addition or subtraction of fractions is performed exactly as with fixed-point operands, except that at least three extra bits of precision must be included to provide accuracy of the least significant bit. When the two fractions have the same sign, a fraction overflow may occur. If this happens, the fraction is right shifted once to put it back into the range, and the exponent is incremented.

If the operand fractions have different signs, there may be cancellation of leading significant bits, yielding an unnormalized result. Normalization is performed with a priority encoder and an N -bit left shifter. The priority encode logic uses a look-ahead circuit to detect the position of the first significant bit from the left. This is encoded into a 6-bit value that is subtracted from the exponent, and is also used as the shift amount by the left shifter.

The extra bits of precision that were carried are now used to perform rounding of the normalized result fraction. An incrementer conditionally adds one LSB to the fraction. If the fraction overflows, the exponent is incremented and the fraction right shifted one place.

The final step is to check the exponent for overflow or underflow. If either occurs, the constants are generated for the exponent and fraction of the result and the status bits are set.

Functional Sequence

A typical function sequence for 32-bit Add operation is shown in Figure 2.5 [4]. The first two clock edges of the scaler operation load the function and operand registers. (MSW of the operands are loaded first.) At clock edge two, the operands begin propagating through the data path logic, the MSW of the result becomes valid during cycle six. Time for the asynchronous propagation is considered about three clock cycle times. However, the cycle time may be varied over a wide range without affecting the propagation time, and thus the number of cycles for propagation time will vary depending upon the cycle time. The result which is valid at cycle six must propagate through logic external to the chip and set an external register or RAM which will be clocked on edge six. Simultaneously, the NEXT.OUT function is loaded into the function inputs of the chip on edge six, causing the next word of the result to be driven out. A time TOUT (100 ns) after edge six the LSW of the result becomes

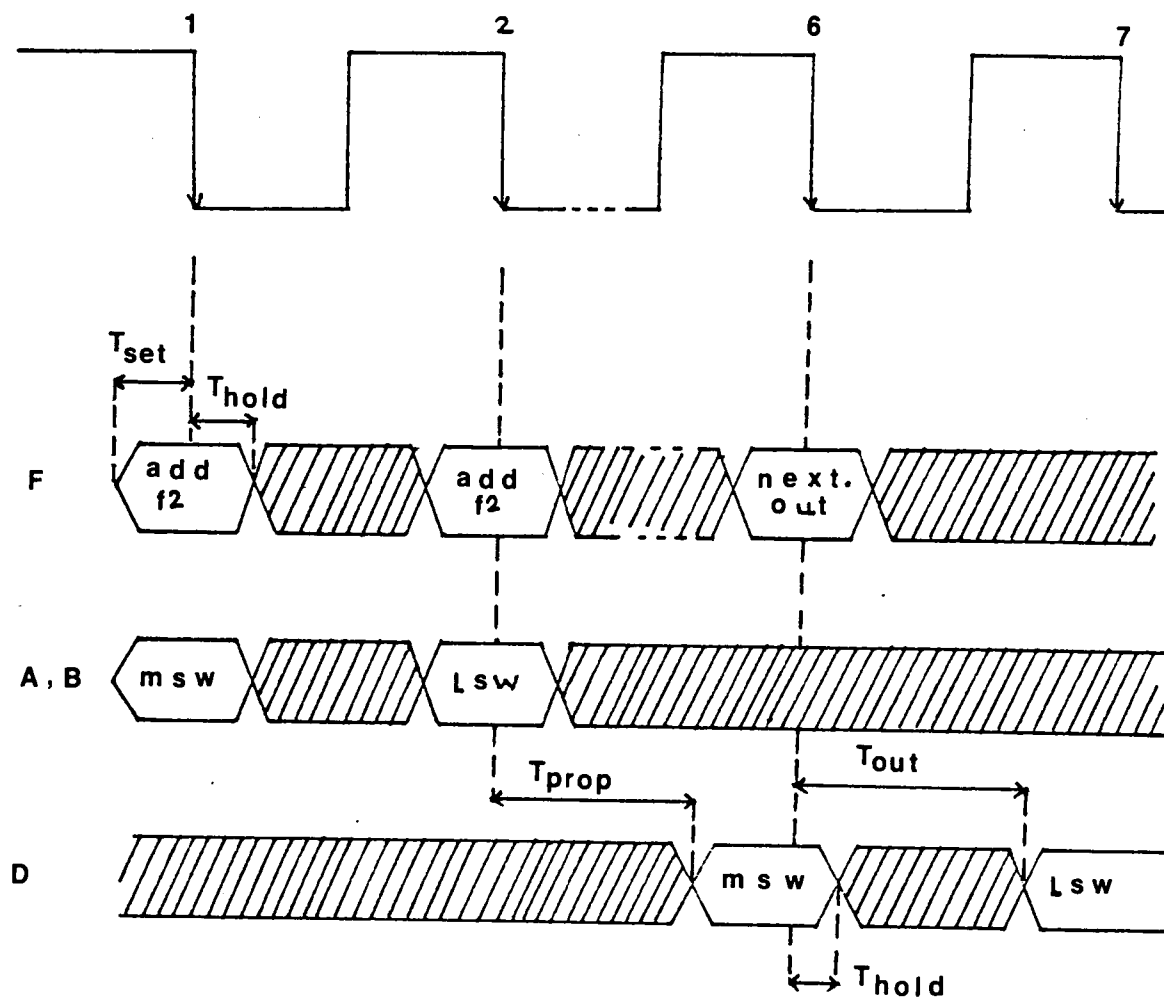


Figure 2.5. Function sequence for add operation.

valid and sets up an external register RAM to be clocked on edge seven. The chip itself needs no function specification on edge seven as long as the DEN-input remains asserted to the end of the cycle.

It should be noted that the operand loading and result unloading is performed synchronously with respect to the external system. Propagation through the combinational data path logic is asynchronous, and the necessary time delay is provided by an integral number of system clocks. The propagation delay is in the range of 400 to 600 ns, depending upon the operation.

Floating-Point Multiply Chip

Floating-point multiplication is relatively simple compared to the floating-point add/subtract chip. Figures 2.6 and 2.7 are the microphotograph and block diagram of this chip. Most of the chip area is used to perform multiplication of the mantissa parts (center and right). The circuitry at the lower left of the chip contains the exponent data path. The multiplier is roughly 245 X 280 mils, and contains about 60,000 devices.

When the operands are first loaded, the two exponents are added and the fractions are masked to the proper precision. One operand is encoded as previously described, and conventional integer multiplication is performed. The integer product (in carry propagate form) is normalized with a zero or one bit left shift, and then rounded to the final position. The result exponent is incremented if there were a zero bit shift or if the fraction overflowed after

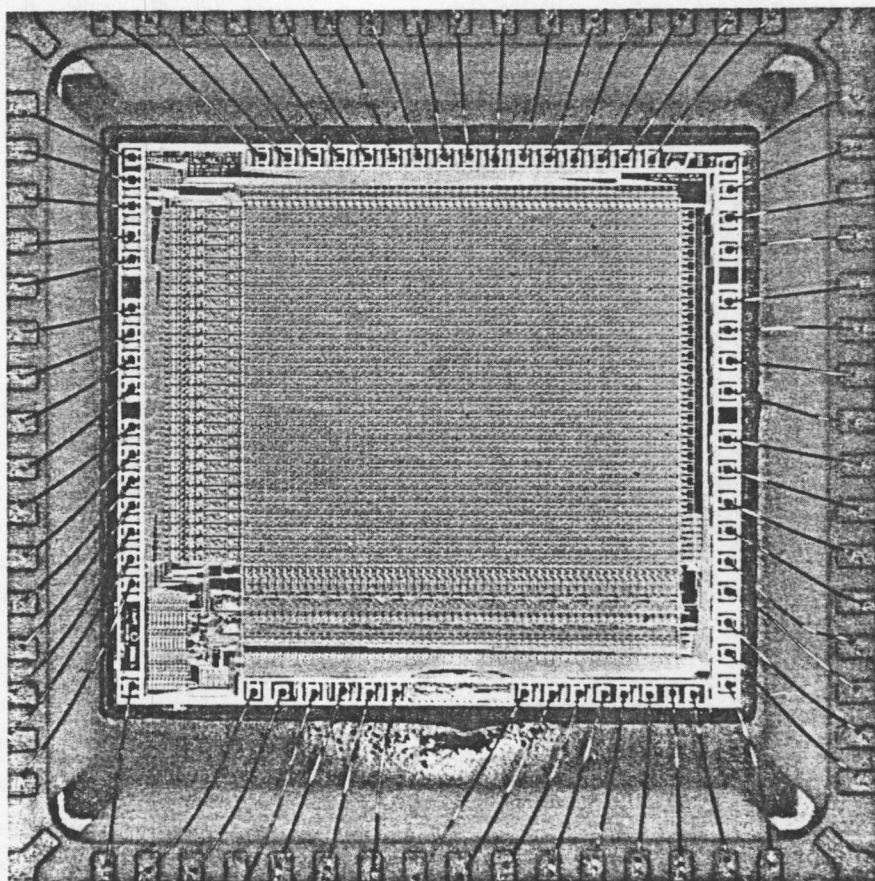


Figure 2.6. Microphotograph of multiply chip.

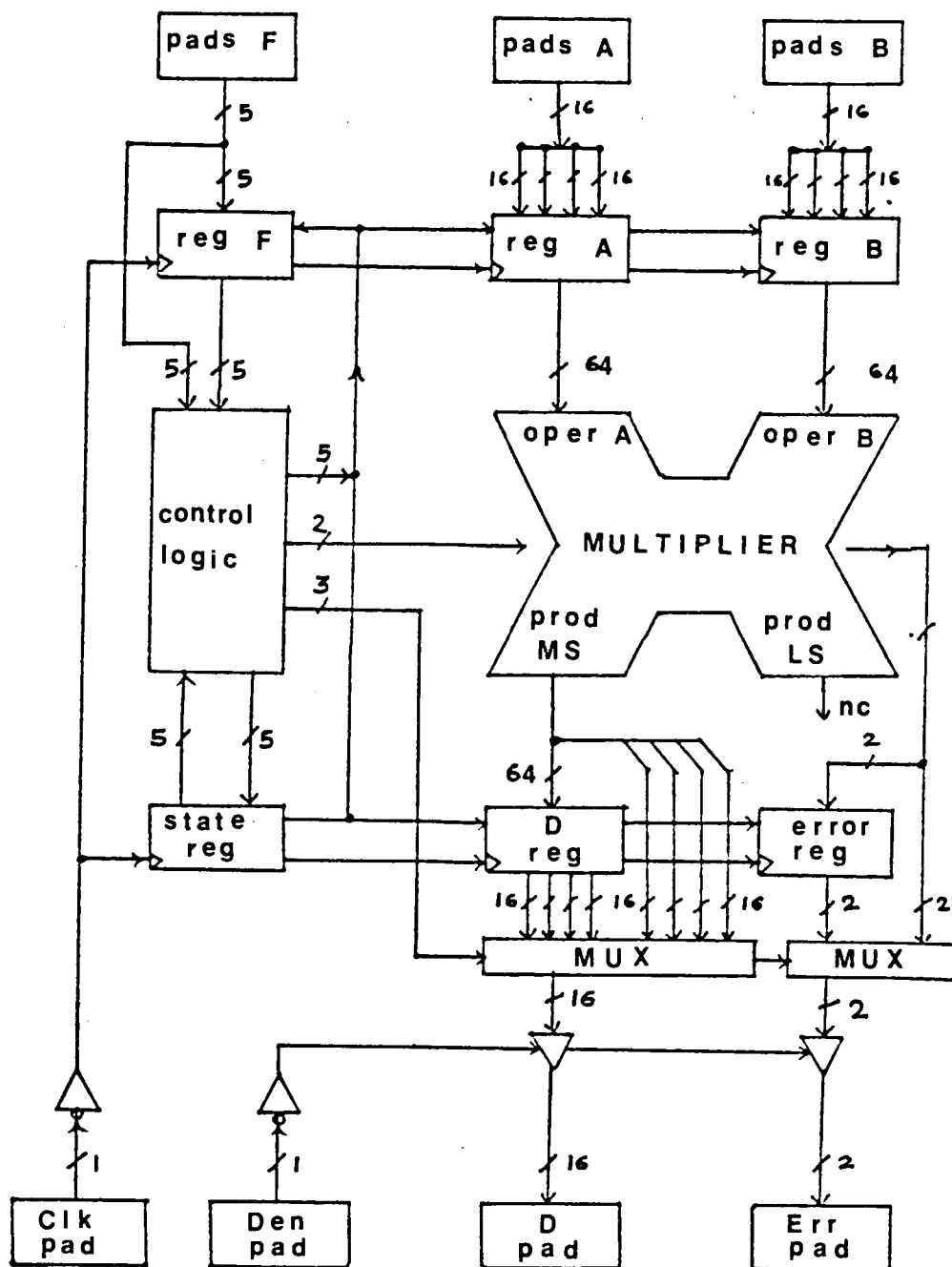


Figure 2.7. Block diagram of multiply chip.

rounding. The exponent is checked for exponent overflow or underflow and if either occurs, the appropriate constants are forced and the status outputs set.

Functional Sequence

A typical function sequence for 32-bit Multiply operation is shown in Figure 2.8 [5]. The first two clock edges of the scalar operation load the function and operand registers (MSW of the operands are loaded first). At clock edge two, the operands begin propagating through the data path logic, the MSW of the result becomes valid during cycle six. Time for the asynchronous propagation is considered about three clock cycles times. However, the cycle time may be varied over a wide range without affecting the propagation time, and thus the number of cycles for propagation time will vary depending upon the cycle time. The result which is valid at cycle six must propagate through logic external to the chip and set up an external register or RAM which will be clocked on edge six. Simultaneously, the NEXT.OUT function is loaded into the function inputs of the chip on edge six, causing the next word of the result to be driven out. A time TOUT (100 ns) after edge six, the LSW of the result becomes valid and sets up an external register or RAM to be clocked on edge seven. The chip itself needs no function specification on edge seven as long as the DEN-input remains asserted to the end of the cycle. The propagation delay is in the range of 400 to 600 ns, depending upon the operation.

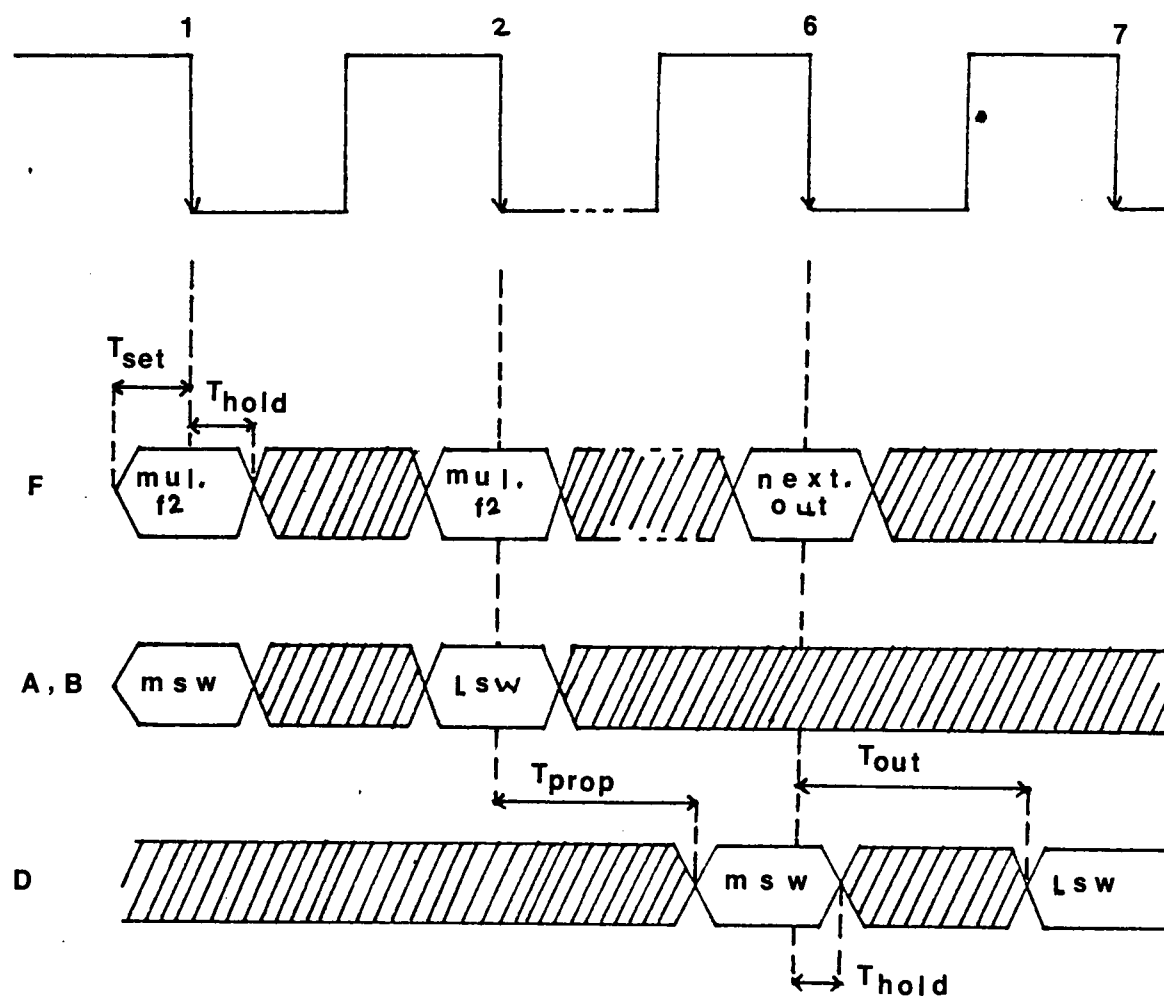


Figure 2.8. Function sequence for multiply operation.

Floating-Point Divide Chip

Floating-point division, like multiplication, is not much more complicated than its fixed point counterpart. The microphotograph and block diagram of the divide chip are shown in Figures 2.9 and 2.10. The divide chip has a small exponent path in the upper and lower left. Most of the chip area is devoted to the mantissa data path in the center and right of the chip. The chip is about 210 X 290 mils in size, and contains about 35,000 devices.

When the operands are first loaded, the two exponents are subtracted and the fractions masked to the proper precision. Conditional two's complement negation is performed so the dividend fraction is positive and the divisor fraction negative. The first seven integral multiples of the divisor are generated combinatorially with three carry propagate adders and four left shifters. An extra load cycle is required at this point to transfer the dividend fraction into the remainder register.

The following sequence of events is repeated anywhere from nine to twenty times depending upon the fraction precision. The remainder fraction is passed to an array of seven carry propagate adders which add it to the seven negative divisor multiples to produce seven new remainders. The smallest positive remainder is passed back to the remainder register, and encoded three bit position is shifted into the quotient register.

This technique is inherently faster than the method of using a full combinational array of 64-bit carry propagate adders [6]-[8]

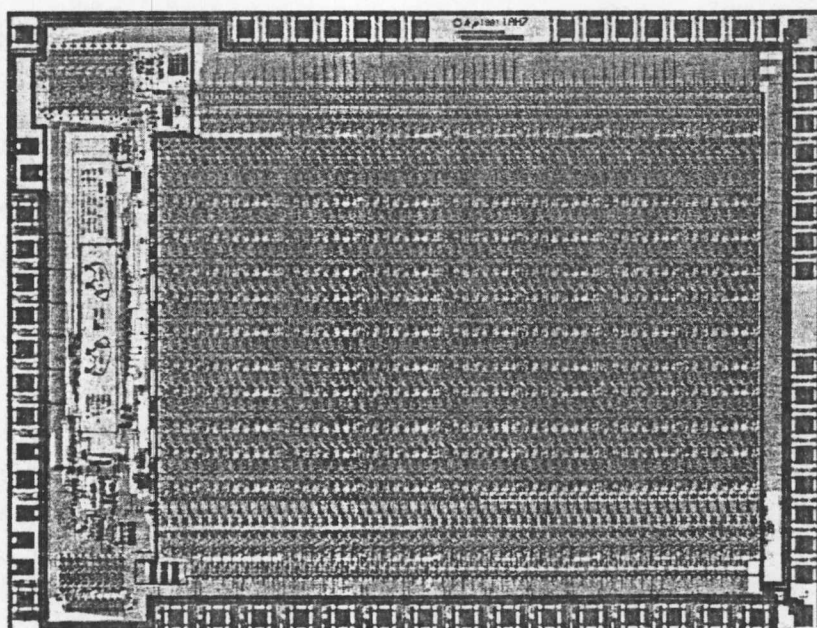


Figure 2.9. Microphotograph of divide chip.

The first technique generates three bits in the time to perform one carry propagation, to drive horizontal enable lines, and to drive the remainder into and out of the array. The combinational technique generates one bit in time to perform one carry propagation and drive horizontal lines. The first technique is thus about two to three times faster. When the quotient is completely loaded, it is normalized with a zero or one left shift and then rounded to its final precision. The result exponent is incremented if there were a zero bit shift or if the fraction overflowed after rounding. The exponent is checked for overflow or underflow and if either occurs the appropriate constants are forced and the status outputs set. The divider chip, unlike the add and multiply chips, must use a portion of its data path repeatedly (the subtract array is cycled anywhere from nine to twenty times). The overall internal delay ranges from 1200 to 2400 ns.

Functional Sequence

Figure 2.11 shows a divide function sequence [6]. The first two edges load the functions and the operands. On the third edge the function LREMAIN loads the dividend into the remainder register. On edges four through twelve, QBIT3 function subtracts a multiple of the divisor from the remainder and three additional quotient bits, which are shifted into the quotient register. At clock edge twelve, a time TPROP (max 300 ns) must elapse to allow the normalization, rounding,

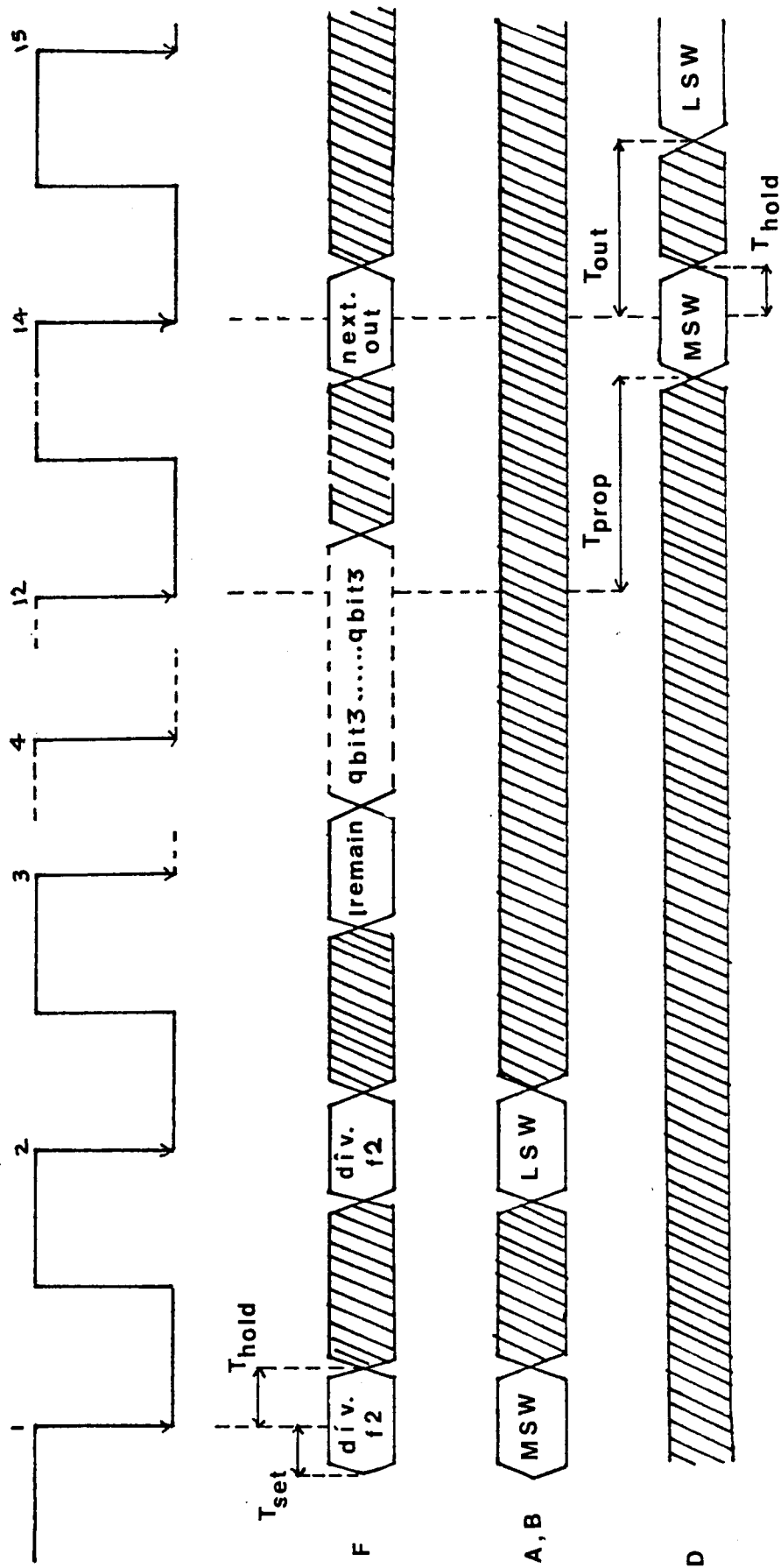


Figure 2.11. Function sequence for divide operation.

and exponent handling to take place. Edges fourteen and fifteen load the two result words into an external register or RAM.

The next chapter discusses the hardware part of the project.

CHAPTER III

HARDWARE

Although the production of very large scale integration circuits (VLSI) is precise, it is not perfect. Thorough testing is required to weed out defective components. Electrical testing is performed at various stages in semiconductor manufacturing and is often repeated by equipment manufacturers.

Electrical testing particularly focuses on two areas. The first is functionality. A functional test verifies that all transistors and interconnecting circuits are intact and operating properly. This is done by providing series of stimuli on input pins while simultaneously monitoring the output pins to verify that it is giving the appropriate response. Each input stimulus and output response is called a "Test Vector."

Verifying all data sheet electrical specifications is the other focus in electrical testing. This includes both DC voltages and currents, as well as AC timing specifications. This is done by forcing a current or voltage on a pin or set of pins. Then the tester measures the current or voltage at another pin or set of pins.

In this project, only the functional testing was done since the electrical specifications were certified as valid by the manufacturer. The primary design goal of this project was to build a testing circuit with minimum external hardware so as to minimize the sources of errors and to enable the user to pinpoint the faulty

chip immediately. The second goal was to keep the test hook-up small and handy so that the user can sit on a small bench in his room and carry out the testing without requiring big instruments [7,8].

As discussed earlier, in functional testing one needs to give a known input stimulus and check whether the output response is appropriate or not. This can be done in various ways using PROMs, EPROMs, counters, registers, latches, and comparators. For instance, one PROM can be programmed with the known input data and the other one with desired output data. Driven by counters, the first PROM can feed input to the test circuit. The output from the circuit can be fed to a comparator whose second input is from the output PROM. The comparator will compare the test result with the standard result and will indicate the error through some display device like an LED. Different sets of PROMs can be used for different input data and respective output data. In another approach, a microprocessor or a microcomputer can be used to fill an EPROM with the required input data. On the other hand, the test output can be stored in another EPROM and fed to the microcomputer again where it can be compared with the desired one. Whenever there is a difference between the test data and the desired data, an appropriate audio or video display can be made.

In the first approach, one will require several hardware components and in case of an error he has to first make sure that everything else other than the circuit under test is O.K. This may be tedious and annoying. Also, one might have to maintain sets of

input and output PROMs, and in case of a very large number of input and output data, the total numbers and cost of these PROMs may be prohibitive.

In the second approach (which is better than the first one) one still needs EPROMs and counters. Also, the scheme will not be easy and fast because first the EPROM will have to be filled by a microprocessor or a microcomputer and then fed to a test circuit; similarly, the case on the output side. So the scheme is neither easy nor fast.

The third idea came to the author: "How about eliminating a EPROM all together from the second scheme?" The question was, then, "What can take the place of the EPROM?" The answer was registers or latches.

Among the computer facilities available within the Department of Electrical Engineering, TI 990 was chosen because of its 16-bit I/O port. The VAX computer which has a 32-bit I/O port was not yet commissioned.

Description of Hardware

Figure 3.1 shows the basic wiring diagram of the circuit. Since A and B input data and D output data are 16-bit wide, the 8-bit latches are used for each of them. Only one 8-bit latch is needed for the 5-bit wide F function input. The 16 bits of the communication register unit (CRU) of the TI 990 [9] are as shown in Table 3.1.

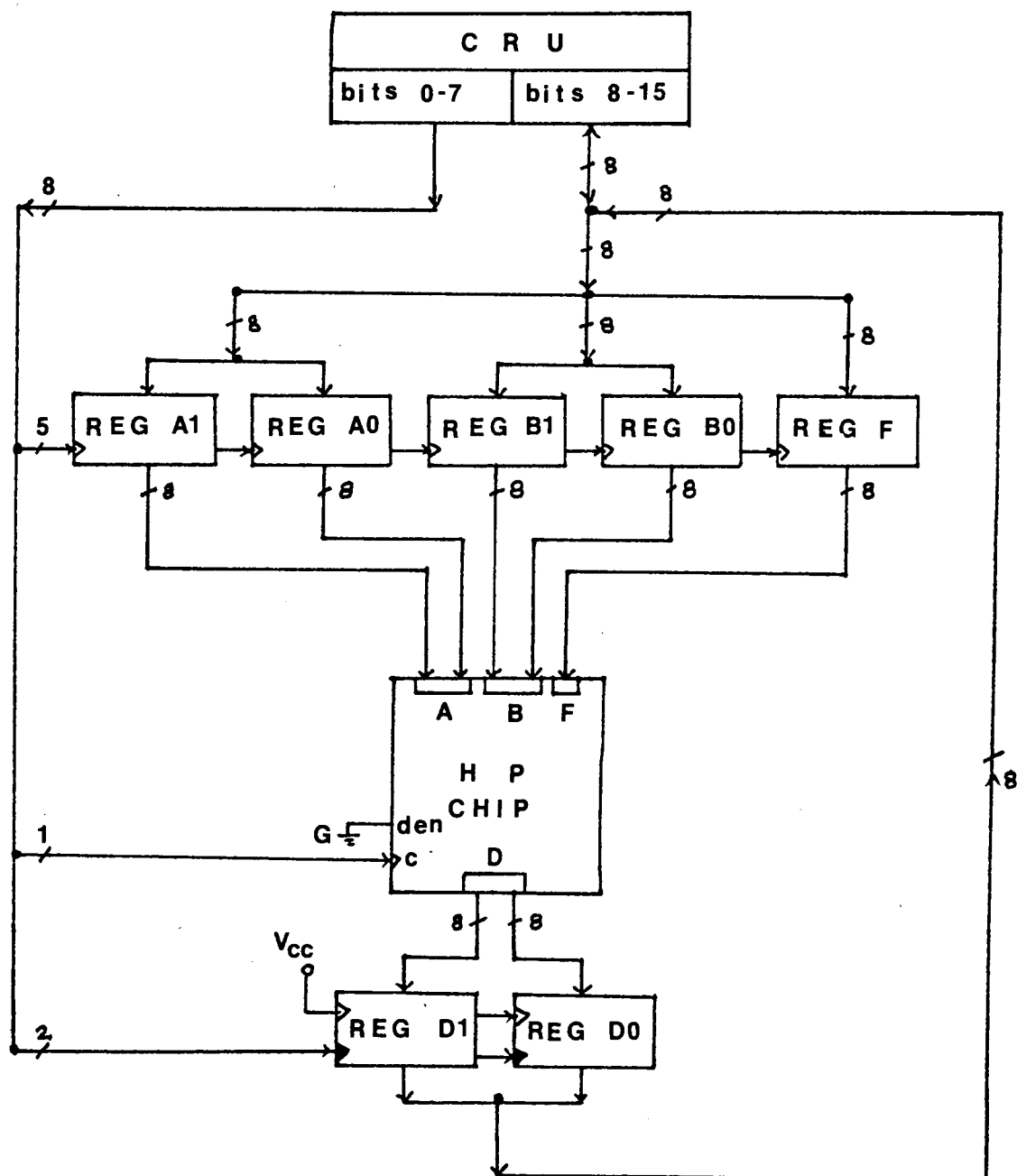


Figure 3.1. Basic wiring diagram.

Table 3.1
Wiring Scheme of CRU Bits

Bit	Assignment	Bit	Assignment							
			F	B1	B0	A1	A0	D1	D0	
0	Clock to the chip	8	D0	D0	D0	D0	D0	Q0	Q0	
1	Enable to F Latch	9	D1	D1	D1	D1	D1	Q1	Q1	
2	Enable to B1 Latch	10	D2	D2	D2	D2	D2	Q2	Q2	
3	Enable to B0 Latch	11	D3	D3	D3	D3	D3	Q3	Q3	
4	Enable to A1 Latch	12	D4	D4	D4	D4	D4	Q4	Q4	
5	Enable to A0 Latch	13	D5	D5	D5	D5	D5	Q5	Q5	
6	Output to D1 Latch	14	D6	D6	D6	D6	D6	Q6	Q6	
7	Output to D0 Latch	15	D7	D7	D7	D7	D7	Q7	Q7	

It should be noted from the above table that Bits 0 to 7 are used as the clock bits for the chip and latches, and Bits 8 to 15 are used as a "Bidirectional Data Bus." In one direction they feed to the input latches A1, A0, B1, B0, and F, and in the other direction they are fed from the output latches D1 and D0.

Outputs of the input latches A1, A0, B1, B0 and F are connected to the respective A, B and F pins of the HP chip. Output of the output latches D1 and D0 are connected to the D pins of the HP chip. It may also be noted that latches A0, B0, and D0 are connected to pins 0 to 7, and latches A1, B1, and D1 are connected to pins 8 to 15 of the A, B, and D pads of the HP chip.

The output controls of the input latches are tied to ground and those of the output latches are microprogrammed controlled. The enables of the input latches are microprogrammed controlled and those of the output latches are tied to Vcc.

Tables 3.2 and 3.3 show the wiring of the latches and the 40-pin socket connector, respectively.

One will appreciate that the hardware scheme presented in this chapter is very simple and yet very efficient. The total number of components is very small which enables the user to trouble-shoot the hardware problems easily.

The software part of the project is discussed in the next chapter.

Table 3.2
Wiring Scheme of Latches

Pin	I/O	D0	D1	A0	A1	B0	B1	F
1	OUT	S33	S37	GND	GND	GND	GND	GND
2	Q0	S29	S29	H64	H56	H47	H39	H23
3	D0	H2	H10	S29	S29	S29	S29	S29
4	D1	H3	H11	S28	S28	S28	S28	S28
5	Q1	S28	S28	H63	H55	H46	H38	H24
6	Q2	S27	S27	H62	H54	H45	H37	H25
7	D2	H4	H12	S27	S27	S27	S27	S27
8	D3	H5	H13	S26	S26	S26	S26	S26
9	Q3	S26	S26	H61	H53	H44	H36	H26
10	GND	GND	GND	GND	GND	GND	GND	GND
11	EN	Vcc	Vcc	S39	S38	S35	S31	S11
12	Q4	S25	S25	H60	H52	H43	H35	H28
13	D4	H6	H14	S25	S25	S25	S25	S25
14	D5	H7	H15	S19	S19	S19	S19	---
15	Q5	S19	S19	H59	H51	H42	H34	A21*
16	Q6	S17	S17	H58	H50	H41	H33	M21*
17	D6	H8	H16	S17	S17	S17	S17	---
18	D7	H9	H17	S15	S15	S15	S15	---
19	Q7	S15	S15	H57	H49	H40	H32	D21*
20	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc

* Indicates future modification connections for adder, multiplier, and divider.

H#. Pin numbers on HP chip.

S#. Pin numbers on 40-pin socket.

Table 3.3
Wiring Scheme of 40-Pin Socket Connector

Pin	Bit	Connector	Pin	Bit	Connector
1			2		
3			4		
5			6		
7			8		
9	0	H27	10		
11	1	F11	12		
13			14		
15	15	L18	16		
17	14	L17	18		
19	13	L14	20		
21			22		
23			24		
25	12	L13	26	11	L8
27	10	L7	28	9	L4
29	8	L3	30		
31	2	B1.11	32		
33	7	D0.1	34		
35	3	B0.11	36		
37	6	D1.1	38	4	A1.11
39	5	A0.11	40	GND	GND

L#. Pin number for any latch in general.

CHAPTER IV

SOFTWARE

Today's very large scale integration (VLSI) testers use a single-user software development environment in which test engineers must write programs that often exceed 25,000 lines of code. Without adequate software tools, they must do object-level patching and debugging, as well as manually analyzing test results.

The complexity of testing VLSI devices has made test program development the number one testing cost. It has forced the test engineer to write programs in terms of the specific, register level characteristics of the hardware.

Structured programming encourages the sharing of debugged files and test patterns, and thus compile time for individual modules is shorter. Hence, the debugging, fine tuning, and maintenance of large test programs become easier and faster when employing structured techniques.

In development of software for this project, these criteria were kept in mind. A general purpose master program was first developed and individual downstream programs for the Adder, Multiplier, and Divider were subsequently derived from it. Several self-debugging programs were also developed to check the correctness of the program itself. Various display formats like "Add..Good," "Mul..Bad.5," "Flt..Done" are used on the kit to notify the user of the end of the test operation and the result.

Development of Software

As discussed in the previous chapter, only the functional test was performed in this project since electrical specifications were certified by the manufacturer.

In functional testing, a set of known input stimuli is applied to the test circuit, and the output response is checked [10]. For HP floating-point processors there are two input operands A and B, each 32-bit wide. There is a control input, F, which is 5-bit wide, and output, D, which is 32-bit wide.

For an exhaustive functional test, it will be necessary to input all possible combinations of operands A and B, and to verify the respective outputs. There are 2^{32} combinations for each of A and B operands, and so the total combinations of both the operands is 2^{64} .

Taking the simplest case of an Adder or Multiplier: one add or multiply operation takes at least 750 ns. This means for 2^{64} operations the total time required will be 750×2^{64} ns which is 1.38×10^{22} ns, or 1.38×10^{13} sec, or 3.84×10^9 hours, or 1.6×10^8 days, or 438,706.81 years. In other words, exhaustive testing is impossible.

Now that exhaustive testing is not possible, what is the best alternate and how much testing time can be allowed? Let us take a relatively small number, say five minutes. It was calculated that in this much time, 4×10^8 operations could be carried out (assuming 750 ns for one operation). Let us now examine the memory requirement

for this: it was calculated to be 10^5 K bytes. No system available within the Department has such a large memory. Extra memory, of course, can be added, but not without sacrificing the size of the test hook-up.

Finally, it was decided that random sampling is the ONLY choice. The size of sampling can be determined using the actual testing time per operation or the available memory space of the system. An actual testing experiment revealed that only 2500 operations per test of Addition, Multiplication, or Division could be performed using 100% of the TI 990 emulator memory (>0146 to >2FFF) and running the CPU at 97.5% capacity [11]. The total time taken for each test set is about 10 seconds.

Regarding the nature of random sampling, the criterion was the best possible distribution over the range, >0000 to >FFFF. A program was developed to check the distribution of these random numbers in eight equal segments, 0000-1FFF, 2000-2FFF, 3000-3FFF, 4000-5FFF, 6000-7FFF, 8000-9FFF, A000-BFFF, C000-DFFF, and E000-FFFF (all numbers are in hexadecimal). After trying several seeds for the random number generations, the best possible set was selected which has even distribution over the entire range.

In Tables 4.1 and 4.2, such a distribution is shown for both the Adder and Multiplier chips. Input operands A and B are the same for both of the chips. D1, D2, D3, and D4 are the outputs obtained during four different versions of the test. TOT D is sum of D1 to D4.

Table 4.1
Histogram for Adder

Range	A	B	D1	D2	D3	D4	TOT D
0000-1FFF	91	75	344	302	336	341	1323
2000-3FFF	75	52	259	323	282	288	1152
4000-5FFF	190	170	939	1130	948	1025	4042
6000-7FFF	170	169	893	854	782	880	3409
8000-9FFF	173	182	761	759	809	742	3071
A000-BFFF	183	218	942	861	1028	956	3787
C000-DFFF	60	50	255	309	308	295	1167
E000-FFFF	58	84	407	262	307	273	1249

Table 4.2
Histogram for Multiplier

Range	A	B	D1	D2	D3	D4	TOT D
0000-1FFF	91	75	308	320	296	347	1271
2000-3FFF	75	52	292	287	299	304	1182
4000-5FFF	190	170	1044	1002	994	964	4004
6000-7FFF	170	169	790	782	729	768	3069
8000-9FFF	173	182	755	771	822	807	3155
A000-BFFF	183	218	1005	1023	1066	981	4075
C000-DFFF	60	50	277	295	276	328	1176
E000-FFFF	58	84	329	320	318	301	1268

In coming paragraphs, the master program is discussed at length from which all other programs for the Adder, Multiplier, and Divider were derived. The individual programs with user's instructions are briefly discussed thereafter.

Development of Master Program

Initially the task was looking so simple that all that was to be done was to send 16 bits of A input, 16 bits of B input, and 5 bits of F input to the test circuit, and clock them together into the chip, and once the operation is performed in 750 ns, read back the 16-bit D output. Furthermore, use of 16-bit CRU port permits straightforward programming. But that wasn't the case.

Take for instance, the 32-bit Add function sequence shown in Table 4.3. Note that A, B, and F inputs have to be present simultaneously on their respective pads before clock #1 can occur on the chip. Now if the system CRU port has only 16 bits, one will have to store these data one by one in a pair of latches or registers connected to the chip pads so that they all will be present when the clock occurs on the chip. This means at least four additional bits are required to clock the A, B, F, and D latches. Where to get these bits? This is not possible with TI 990. Thus, the apparently simple scheme was not feasible.

It was, then, decided to divide a 16-bit word into two 8-bit words. Two latches are used for each of A, B, and D operands, and one latch is used for F. Out of 16 bits from the CRU, seven bits are

Table 4.3
Function Sequence for 32-Bit Add Operation

Clock #	F Input 5 Bits	A Input 16 Bits	B Input 16 Bits	D Output 16 Bits
1	CLEAR	--	--	--
2	ADD.F2	A1	B1	--
3	ADD.F2	A0	B0	--
4	--	--	--	--
5	--	--	--	--
6	--	--	--	--
7	NEXT.OUT	--	--	D1
8	--	--	--	D0

A0, B0, D0 correspond to bits 0-15.

A1, B1, D1 correspond to bits 16-31

required for clocking these seven latches and the eighth bit is required for the chip clock. The remaining 8 bits are used as a Bidirectional Data Bus. Now the question was which byte can be used for "clocking" and which one for "data"? The fact that a "clock word" is 8 bits wide and a "data word" is 16-bit long, and all byte instructions handle only the most significant byte, leaves only one choice that the least significant byte can be used for the "clock word," and the most significant byte for the "data word." This results in some operation time saving as symbolically illustrated in Table 4.4. As mentioned before, the most significant byte of the CRU is used as a Bidirectional Data Bus which implies that bits 8-15 have to be software reset before programming them for input from the output. During this process, bits 0-7 also are reset and go high. This unwanted action creates lots of problems,

Table 4.4

Alternate Schemes for CRU Byte Assignment

Sequence of Operation	
MSB- Data Word (DW)	MSB- Clock Word (CW)
MOV CW+,R1] MSB OF DW	MOV DW,R1] LSB OF DW
MOVB DW+,R1] IS SENT	MOVB CW+,R1] IS SENT
LDCR R1,0]	LDCR R1,0]
MOV CW+,R1 [LSB OF DW	SWPB DW -
MOVB DW+,R1 [IS SENT	MOV DW,R1 [MSB OF DW
LDCR R1,0 [	MOVB CW+,R1 [IS SENT
	LDCR R1,0 [
	INCT DW - For next DW

particularly for the "clock word" because the latches start latching all 1's. Although this does not offer many problems for A, B, or F latches, since that data is not clocked into the chip (it can be clocked only at negative edge); but for the D latches this does create serious problems because they will latch output data from the chip at wrong times. To avoid this problem, instead of enable lines, the output control lines of the D latches were connected to the "clock word" which puts the tristate bus in a high impedance state during software reset. Further, the DEN line of the chip was also tied to ground (Logic 0) so that the tristate output bus is always enabled, and one does not have to bother programming it. Bit assignment of the "clock word" and "data word" is shown in Table 4.5.

Table 4.6 shows typical microprogramming of "clock word" and "data word" for the Adder. Once the microprogramming was figured out, the rest of the job was not difficult. Each data from the A input is combined with every data from the B input and fed to the chip.

Table 4.5
CRU Byte Assignment

LSB			MSB	
Clock Word			Data Word	
0	Chip	Clock	8	Bit 0
1	F	Clock	9	Bit 1
2	B1	Clock	10	Bit 2
3	B0	Clock	11	Bit 3
4	A1	Clock	12	Bit 4
5	A0	Clock	13	Bit 5
6	D1	Output	14	Bit 6
7	D0	Output	15	Bit 7

Table 4.6
Microprogramming for Adder

<-A,B,F OR D-> <-15 TO 8---->	HEX NO.	D0 7	D1 6	A0 5	A1 4	B0 3	B1 2	F 1	CHP 0	HEX NO.
F CLEAR	00	1	1	0	0	0	0	1	1	C3
F CLEAR	00	1	1	0	0	0	0	1	0 *	C2
A1 MSB	--	1	1	0	1	0	0	0	1	D1
A1 LSB	--	1	1	1	0	0	0	0	1	E1
B1 MSB	--	1	1	0	0	0	1	0	1	C5
B1 LSB	--	1	1	0	0	1	0	0	1	C9
F ADD.F2	18	1	1	0	0	0	0	1	1	C3
F ADD.F2	18	1	1	0	0	0	0	1	0 *	C2
A0 MSB	--	1	1	0	1	0	0	0	1	D1
A0 LSB	--	1	1	1	0	0	0	0	1	E1
B0 MSB	--	1	1	0	0	0	1	0	1	C5
B0 LSB	--	1	1	0	0	1	0	0	1	C9
F ADD.F2	18	1	1	0	0	0	0	1	1	C3
F ADD.F2	18	1	1	0	0	0	0	1	0 *	C2
S O F T W A R E R E S E T										
D1 MSB	--	1	0	0	0	0	0	0	1	81
D1 LSB	--	0	1	0	0	0	0	0	1	41
F NEXT.OUT	1F	1	1	0	0	0	0	1	1	C3
F NEXT.OUT	1F	1	1	0	0	0	0	1	0 *	C2
S O F T W A R E R E S E T										
D0 MSB	--	1	0	0	0	0	0	0	1	81
D0 LSB	--	0	1	0	0	0	0	0	1	41

*Clock occurrence to chip.

This process goes on until all data from the A input is combined with all data from the B input. As mentioned earlier in this chapter, a maximum 2500 operations can be performed in one test. This sets a limit on the size of the A and B input data. At first glance 50 X 50 looks like very convenient figures; however, after a little thought it is realized 50 X 50 is not the best choice because it will test each of the A and B input pads for only 50 different inputs. In two tests, the pads can be tested for only 100 inputs. On the other hand, if the size is selected as 240 X 10 and 10 X 240, the pads can be tested for 250 inputs and the total number of input combinations is still 4800. One might be tempted, then, to select 500 X 5 and 5 X 500. In fact this scheme will not work because of the need for more memory space to be occupied by the input (memory occupied by the output does not change). Four different sets of A and B inputs were applied to each chip for testing, so that the total number of A and B inputs was 500 each, and the total test combinations were 10,000.

Using known "good" chips, a set of "correct" or "desired" outputs was obtained for each set of A and B inputs. Output of the chip under test was compared with the desired output obtained from the "good" chip. In case of a bad chip, an error subroutine in the program allows the user to inspect the Error Table starting from the specified memory location to find out an error output vs. desired output. An error counter was also introduced in the program to count the number of errors. In the program, the error counter was set to 8 since this number is sufficiently large to reject a chip.

However, the user can select another number and allot sufficient memory space in the Error Table. A display like "Add..Good" or "Add.Bad.#" is used on the kit to notify the user of the end of the test and its result where # is the number of errors.

The various display formats and user's instructions are discussed in the next paragraphs. Table 4.7 summarizes all these programs.

The discussion of individual programs will better be done in two groups, one for general purpose programs and one for floating-point processor programs.

Table 4.7
Summary of Programs

No.	Name	Purpose	For
1	RAND	To generate random # in regular format	Gen. Pur.
2	RFC	To generate random # in HP format	Gen. Pur.
3	DIS	To check the distribution of random #	Gen. Pur.
4	FLT1	To generate sequential # in HP format	Adder.Flt
5	FLT2	To test the program itself	Adder.Flt
6	FLT	To generate desired output	Adder.Flt
7	FT1	To test the Adder for floating conv.	Adder.Flt
8	ADD1	to test the program itself	Adder.Add
9	ADDT	To generate desired output	Adder.Add
10	AT	To test the Adder for add operation	Adder.Add
11	MTO	To test the program itself	Multiplier
12	MULT	To generate desired output	Multiplier
13	MT	To test the Multiplier	Multiplier
14	DIV	To test the program and the Divider	Divider

General Purpose Programs

Three general purpose programs were developed in this project:

(i) to generate the random numbers in regular format, (ii) to generate the random numbers in HP 1000 floating-point format, and (iii) to check the distribution of the random numbers generated in previous programs.

(i) The program name is "HP.RAND." This program generates a set of random numbers up to 2500. N, the total number of "random numbers" and S1 and S0, the seeds are to be selected by the user. The numbers generated are in the OUT table and the end of it is indicated by zeroes. The display, "Rand..Done," is used to indicate the end of the operation. The program is shown in Appendix A.1.

(ii) The program name is "HP.RFC" (Random to Floating Conversion). This program generates the random numbers in HP 1000 floating-point format. The maximum numbers that can be generated are 2500. HP 1000 format is already discussed elsewhere in this report. N is the total number, and S1 and S0 are the seeds to be determined by the user. Like the previous program, the output is stored in the OUT table and zeros indicate the end of it. The display format used is "RND.FLT.CON." Appendix A.2 shows this program.

(iii) The program name is "HP.DIS." This program counts the distribution of DATA in eight equal ranges from >0000 to >FFFF. The DATA can be AIN, BIN, or OUT. Counter N can be set to any number (max. 2500). The end of the operation is indicated by the "DONE"

display. The user can inspect registers R4 through R11 for the distribution result. The program is shown in Appendix A.3.

Floating-Point Processor Programs

The discussion of these programs is divided under three individual sections, one for each chip: Adder, Multiplier, and Divider.

Adder

For Adder, two basic types of tests were developed, one for Fixed to Floating-Point Conversion, and one for Add Operation.

Fixed to floating conversion. Table 4.8 shows the function sequence for this operation. In this category four different types of programs were developed: (i) for generating the sequential numbers in floating-point format, (ii) for testing the program itself and also the chip with known inputs and "calculated" output, (iii) for generating "desired" output for known input using GOOD chip, and (iv) for testing ANY chip with known random input and generated "desired" output from previous program.

(i) The program is "HP.FLT1." Here starting from any number, sequential numbers up to 2500 can be generated in floating-point format. Counter N is set for this number, and the starting number is entered in A1 and A0 (e.g., if the starting number is 5, then A1 has 0000 and A0 has 0005; if the number is 50,000, A1 has 0005 and A0 has 0000). The display format used for this program is "Flt..Done"

Table 4.8
Function Sequence for Fixed to Floating Conversion

Clock #	F Input 5 Bits	A Input 16 Bits	B Input 16 Bits	D Output 16 Bits
1	CLEAR	--	--	--
2	FT.I2.F2	I1	--	--
3	FT.I2.F2	I0	--	--
4	--	--	--	--
5	--	--	--	--
6	--	--	--	--
7	NEXT.OUT	--	--	D1
8	--	--	--	D0

to indicate the end of the operation. The program is shown in Appendix A.4. Numbers 1 to 1000 in floating point format are shown in Appendix A.5.

(ii) The program name is "HP.FLT2." Here the known input is entered in AIN and the hand calculated output is entered in OUT. Result of the test is displayed by "Flt..Good" or "Flt..Bad." The user can also test the program itself by inspecting SENDT (send table) which should have the sequence 00C3, 00C2, --D1, --E1, 12C3, 12C2, --D1, --E1, 12C3, 12C2, 8100, 4100, 1FC3, 1FC2, 8100, 4100. The program is shown in Appendix A.6.

(iii) The program name is "HP.FLT." Here the known input is entered in AIN and the desired output using GOOD chip is obtained at OUT. Counter A can be set up to 2500. The display format used is "Flt..Done." The program is shown in Appendix A.7.

(iv) The program name is "HP.FT1." Here the random data generated by "HP.RAND" is entered in AIN and the desired output

generated by "HP.FLT" is entered in OUT. The result is displayed by "Flt..Good" or "Flt.Bad.#" the number of errors. The counter A can be set up to 2500. The program is illustrated in Appendix A.8.

Add operation. Table 4.3 (p. 38) shows the function sequence for this operation. Three types of programs were developed for this purpose: (i) for testing the program itself and also the chip with the help of known inputs and "calculated" outputs, (ii) for generating the "desired" output for known random inputs using a GOOD chip, and (iii) for testing ANY chip with random inputs and generated "desired" outputs from the previous program.

(i) The program name is "HP.ADD1." Here two known inputs for A and B are entered in AIN and BIN, and corresponding hand calculated outputs are entered in OUT. The result of the test is indicated by displaying "Add..Good" or "Add..Bad" on the kit. The user can test the program itself by inspecting SENDT—send table—which should have the following sequence of data: 00C3, 00C2, --Di, --Ei, --C5, --C9, 18C3, --D1, --E1, --C5, --C9, 18C3, 13C2, 8100, 4100, 1FC3, 1FC2, 8100, 4100. The program is shown in Appendix A.9.

(ii) The program name is "HP.ADDT." Here random data generated by "HP.RFC" is entered in AIN and BIN, and the desired output is obtained at OUT. A and B counters are set at 240×10 and 10×240 in two different versions of the test. However, they can be set to any value as desired by the user. The display format used here is "Add..Done." Appendix A-10 shows the program.

(iii) The program name is "HP.AT." Here, the random data generated by "HP.RFC" is entered in AIN and BIN, and the desired output generated by "HP.ADDT" is entered in OUT. There are four versions of this test: AT1 through AT4. A and B counters are set at 240×10 in AT1 and AT3, and at 10×240 in AT2 and AT4. However, they can be set to any number according to user's convenience. The display format used is "Add..Good" or "Add.Bad.#" where # is the number of errors. The user can inspect ERROR for the desired output versus error output. Appendix A.11 refers to this program.

Multiplier

Table 4.9 shows the function sequence for this operation. Unlike an Adder here only one "multiply" operation is checked. Three types of programs were developed for this purpose: (i) for testing the program itself and also the chip with the known input and "calculated" output, (ii) for generating the "desired" output for known random input using a GOOD chip, and (iii) for testing ANY chip with random input and generated "desired" outputs from previous programs.

(i) The program name is "HP.MTO." Here two known inputs for A and B are entered in AIN and BIN, and corresponding hand calculated outputs are entered in OUT. The result of the test is indicated by displaying "Mul..Good" or "Mul..Bad" on the kit. The user can test the program itself by inspecting SENDT (send table) which should have the sequence, 00C3, 00C2, --D1, --E1, --C5, --C9, 16C3, 16C2, --D1,

--E1, --C5, --C9, 16C3, 16C2, 8100, 4100, 1FC3, 1FC2, 8100, 4100.

The program is shown in Appendix A.12.

Table 4.9
Function Sequence for Multiply Operation

Clock #	F Input 5 Bits	A Input 16 Bits	B Input 16 Bits	D Output 16 Bits
1	CLEAR	--	--	--
2	MUL.F2	A1	B1	--
3	MUL.F2	A0	B0	--
4	--	--	--	--
5	--	--	--	--
6	--	--	--	--
7	NEXT.OUT	--	--	D1
8	--	--	--	D0

(ii) The program name is "HP.MULT." Here random data generated by "HP.RFC" is entered in AIN and BIN, and desired output is obtained at OUT. A and B counters are set at 240×10 and 10×240 in two different versions of the test; however, they can be set to any value as desired by the user. The display format used here is "Mul..Done." The program is shown in Appendix A.13.

(iii) The program name is "HP.MT." Here random data generated by "HP.RFC" is entered in AIN and BIN, and the desired data generated by "HP.MULT" is entered in OUT. There are four versions of this test: MT1, MT2, MT3 and MT4. A and B counters are set at 240×10 in AT1 and AT3, and at 10×240 in AT2 and AT4. They can be set to any numbers according to the user's convenience. The display format used is "Mul..Good" or "Mul.Bad.#" where # is the number of

errors. The user can inspect ERROR for the desired output versus error output. The program is shown in Appendix A.14.

Divider

Table 4.10 shows the function sequence for this operation. In case of the divider, only two chips were available within the Department, and one of them was definitely bad, so there was no point in developing the detailed programs for the Divider. Only one program was developed. The program name is "HP.DIV." Here two known inputs for A and B are entered in AIN and BIN, and hand calculated corresponding outputs are entered in OUT. The result of the test is indicated by displaying "Div..Good" or "Div..Bad" on the kit. The user can test the program itself by inspecting SENDT (send table) which should have the sequence of data: 00C3, 00C2, --D1, --E1, --C5, --C9, 16C3, 16C2, --D1, --E1, --C5, --C9, 16C3, 16C2, 18C3, 18C2, 12C3, 12C2, 12C3, 12C2, 12C3, 12C2, 12C3, 12C2, 12C2, 12C3, 12C2, 12C3, 12C2, 13C3, 12C2, 8100, 4100, 1FC3, 1FC2, 8100, 4100. The program is illustrated in Appendix A.15.

After discussing various programs for the Adder, Multiplier, and Divider, at the end the user's instructions for Emulator and the results of the experiments are discussed.

User's Instructions for Emulator of TI 990

Two special display formats have been developed for emulator usage: (i) "HP.ET"- This is used to set the sequential numbers in

Table 4.10

Function Sequence for Divide Operation

Clock #	F Input 5 Bits	A Input 16 Bits	B Input 16 Bits	D Output 16 Bits
1	CLEAR	--	--	--
2	DIV.F2	A1	B1	--
3	DIV.F2	A0	B0	--
4	LREMAIN	--	--	--
5	QBIT3	--	--	--
.	.	.	.	.
.	.	.	.	.
12	QBIT3	--	--	--
13	--	--	--	--
14	NEXT.OUT	--	--	D0
15	--	--	--	D1

floating point format (HP.FLT1) with corresponding decimal numbers attached to them, like

1- >4000	>0002	2- >4000	>0004
3- >6000	>0004	4- >4000	>0006

To obtain the output in this form, the user will have to first enter into the emulator mode with "AMPLDX" and then respond to the prompt as follows.

```
? RSTR('HP.ET')      (Ret)
? ET                  (Ret)
```

Then the user should load the object file in the normal way. Whenever the user wants to read the contents of a particular memory space, e.g., from >01A3 to >01B3, he will have to type,

```
? DISP
```

The screen will return with,

SPECIAL DISPLAY FORMAT

```
START ADDRESS  =
END ADDRESS    =
```

Here the user will type the corresponding addresses, and the display will be in the above form. This display is very useful for HP.FLT1.

(ii) "HP.ET1"- This display format is extremely useful while working with HP.RAND, HP.RFC, HP.FLT, HP.ADDT, and HP.MULT. The user can obtain the output from these programs in a form that can be directly used to attach as the input table or output table to other programs like HP.FLT, HP.ADDT, HP.AT, HP.MULT, and HP.MT.

Here the user, after entering into the emulator mode (AMPLDX), will respond to the prompt as follows.

```
? RSTR ('HP.ET1')    (Ret)
? ET                  (Ret)
```

Then he loads the object file in a normal way. Like the previous case, he will enter the START ADDRESS and END ADDRESS to inspect the output, and the display format will be,

```
DATA  >01E4,>8DE3,>3BCD,>1473
DATA  >236C,>14EE,>A3C2,>767F
```

Results of Experiments

The previously discussed tests were carried out on 18 Adder chips, 18 Multiplier chips, and 2 Divider chips. Nine Adder chips, 16 Multiplier chips, and 1 Divider chip were found to be fully capable of 32-bit operation. The remaining chips failed one or more

tests. Sometimes the particular chip passed the particular test several times and failed the same tests several times. Cases were also noted where the particular chip passed a few of the tests consistently and failed the others consistently. Most of the time the bad chips failed all the tests. A separate record has been prepared for these test results.

The last chapter summarizes this work and presents the conclusions.

CHAPTER V

SUMMARY

In this project, a strategy was developed for testing the 32-bit operation of three types of VLSI floating-point processors (adder/subtractor, multiplier and divider). A testing circuit with minimum external hardware was implemented. Major consideration was given to the size and complexity of the test hook-up to facilitate trouble shooting by the operator.

Exhaustive testing was not possible because of the enormous time required for it. Random sampling was the only alternative. The best possible set of random data, the one with the most nearly uniform distribution over the entire range, was selected for inputting the data. Self-testing programs were developed to verify the correctness of the test programs. Initial testing of the chips was done with known input and hand calculated output. For the actual testing, a set of known random input data was first applied to several known "good" chips and the "desired" output was obtained. Subsequently, this input and output data were used to test any unknown chips.

These tests were executed on 38 different chips, and 12 faulty chips were ascertained. Some of the bad chips passed several tests and failed the others, and some of them passed the specific tests sometimes and failed the same tests at other times.

The fact that the tests developed in this project detected some bad chips proves the effectiveness of the tests.

Future Modifications

Since the F input is 5 bits wide, only five input/output pins of the F latch are used in the system. The remaining three, however, can be connected to the DEN lines of the HP chips in case the user wants to mount all the three types of chips (the adder, multiplier, and divider) on the board simultaneously. Space is already provided on the board for this purpose. A wiring scheme was suggested in Table 3.2 (p. 31). Except for the DEN lines, all other pins of the additional chips will have to be connected in parallel with the existing socket.

The software developed in this project is very flexible. The user can generate his own set of random inputs selecting the proper seeds in the programs. He can also increase the size of the test table above 2500 just by arranging for additional memory and selecting the proper N.

LIST OF REFERENCES

LIST OF REFERENCES

1. Ware, F. A., W. H. McAllister, J. R. Carlson, D. K. Sun and R. J. Vlach, "64 Bit Monolithic Floating Point Processors," IEEE Journal of Solid-State Circuits, Vol. SC-17, No. 5, pp. 898-907, October, 1982.
2. McAllister, W. H. and J. R. Carlson, "Floating-Point Chip Set Speeds Real-Time Computer Operation," Hewlett-Packard Journal, pp. 17-23, February, 1984.
3. Ware, F. A., "A 64b Floating Point Processor," 1982 IEEE International Solid State Circuits Conference, Digest of Technical Papers, pp. 24-26, February, 1982.
4. Hewlett-Packard Co., "1AE7 Floating Point Adder Data Sheet, A-1AE7-9001-1," December 22, 1982.
5. Hewlett-Packard Co., "1AH4 Floating Point Multiplier Data Sheet, A-1AH4-9001-1," January 27, 1983.
6. Hewlett-Packard Co., "1AH7 Floating Point Divider Data Sheet, A-1AH7-9001-1," October 18, 1982.
7. Williams, T., "System Development and Testing," Computer Design, April 5, 1983.
8. Lawson, L., "Develop VLSI Software Quickly," Computer Design, November, 1983.
9. Texas Instruments, TM 990/U89 Microcomputer User's Guide, 1981.
10. Eichelberger, E. B. and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," IBM J. Res. Develop., Vol. 27, No. 3, pp. 265-272, May, 1983.
11. Texas Instruments, Model 990 Computer AMPL Microprocessor Prototyping Laboratory Operation Guide (946275,76,77-9701), 1980.

APPENDIXES

APPENDIX A.1

```

IDT  RAND  *
ADRG >150  *
BSS  >20   *
LWPI >150  *
LI   5,##  * SEED S1
LI   6,##  * SEED S0
LI   3,##  * NUMBER N
LI   7,OUT  *
RAND  LI   0,2  *
      MOV  5,1  *
      MOV  6,2  *
LP     A    6,2  *
      JNC  OK   *
      INC  1    *
OK     A    5,1  *
      DEC  0    *
      JNE  LP   *
      A    6,1  *
      CI   2,0  *
      JNE  RTN  *
      CI   1,0  *
      JNE  RTN  *
RTN    LI   2,>3039 *
      MOV  1,5  *
      MOV  2,6  *
      MOV  5,*7+ *
      MOV  6,*7+ *
      DEC  3    *
      JNE  RAND * IF ALL NUMBERS NOT DONE,GO AGAIN
      LI   5,0  *) IF DONE, LOAD ZEROS TO THE
      LI   6,0  *) OUTPUT TABLE, AND GO TO DISPLAY
      MOV  5,*7+ *) ROUTINE.
      MOV  6,*7+ *)
      B    @DSPL *)
*****
DSPL  LI   12,>420 * DISPLAY ROUTINE FOR 'RAND..DONE'
      LI   0,>20  *
GO     LI   5,DSP1 *
      LI   2,10  *
AGAIN  LDCR *5+,13 *
      SBZ  12    *
      MOV  0,8   *
LOOP   DEC  8    *
      JNE  LOOP  *
      DEC  2     *
      JNE  AGAIN *
      JMP  GO    *
*****
DSP1  DATA >1CE0,>1881,>1C82,>1A13
      DATA >17F4,>17F5,>1A16,>1C07
      DATA >1A16,>1C88,>1869
OUT    BSS  >###
*****

```

APPENDIX A.2

```

IDT / RFC /
*
*PART I - RANDOM NUMBER GENERATED IS
*STORED IN AIN WHICH IS INPUT FOR PART II.
*
      AORG >150      *
      BSS >20        *
      LWPI >150      *
      LI 5,###      * SEED S1.
      LI 6,###      * SEED S0.
      LI 3,###      * NUMBER N.
RAND  LI 7,AIN      * O/P FOR P-I AND I/P FOR P-II.
      LI 0,2        *
      MOV 5,1       *
      MOV 6,2       *
LP    A 6,2         *
      JNC OK        *
      INC 1         *
OK    A 5,1         *
      DEC 0         *
      JNE LP        *
      A 6,1         *
      CI 2,0        *
      JNE RTN       *
      CI 1,0        *
      JNE RTN       *
RTN   LI 2,>3039    *
      MOV 1,5       *
      MOV 2,6       *
      MOV 5,*7+     *
      MOV 6,*7+     *
      DEC 3         *
      JEQ ZERO      *
      BLWP @FLTVT   *
      JMP RAND      *
ZERO  DECT 7        *
      DECT 7        *
      LI 0,2        *
      LI 5,0        * LOAD ZEROS TO THE END OF
ZERO1 MOV 5,*7+     * THE TABLE.
      DEC 0         *
      JNE ZERO1     *
      BLWP @FLTVT   *
      JMP DSPR      *
*
*PART II - CONVERTING RANDOM NUMBERS IN
*HP 1000 FLOATING POINT FORMAT. AIN IS
*INPUT.
      AORG >200      *
      BSS >20        *
      DATA FLTWP    *
      DATA FLTCON   *
FLTCON LI 0,AIN      * @ OF A INPUT
      MOV @COUT,4    * @ OF OUTPUT
      LI 7,CLK       * @ OF CLOCK INPUTS
      LI 8,FIN       * @ OF F INPUTS
      MOV 8,15       * F.CLEAR
      BL @SEND       *
      BL @SEND       *
      MOV 15,8       *
      LI 1,2         * SET COUNTER TO 2

```

```

FLT      MOV    0,15      * I1          IO
          BL     @SEND    *
          BL     @SEND    *
          MOV    15,0     *
          MOV    8,15     * FT.I2.F2   FT.I2.F2
          BL     @SEND    *
          BL     @SEND    *
          MOV    15,8     *
          DEC    1        * I1 AND IO TAKEN?
          JEQ    NEXT     * IF YES, GO TO NEXT.OUT
NEXT      B      @FLT     * IF NOT, TAKE IO
          BL     @GET     * GET D1
          BL     @SEND    *
          BL     @SEND    *
          MOV    15,8     *
          BL     @GET     * GET DO
          RTWP    *
*****
SEND      MOV    *7+,10   * SUB ROUTINE FOR SENDING
          MOV    *15+,10  * DATA TO TEST CIRCUIT.
          LI     12,>20   *
          LDCR   10,0     *
          B      *11     *
*****
SEND1     MOV    *7+,10   * SUB ROUTINE FOR SENDING
          SWPB   10       * DATA DURING 'GET' SUB
          LI     12,>20   * ROUTINE.
          LDCR   10,8     *
          B      *11     *
*****
GET        LI     12,0    * SUB ROUTINE FOR SOFTWARE
          SBZ    0        * RESET AND GETTING DATA
          SBZ    15       * FROM TEST CIRCUIT.
          MOV    11,@STORE *
          BL     @SEND1   *
          LI     12,>30   *
          STCR   9,8      *
          SWPB   9        *
          BL     @SEND1   *
          MOV    @STORE,11 *
          LI     12,>30   *
          STCR   9,8      *
          MOV    9,*4+    *
          MOV    4,@COUT  *
          B      *11     *
*****
DSPR      LI     12,>420   * SUB ROUTINE FOR
          LI     0,>20     * DISPLAYING
GO         LI     5,DSP1   * 'RND.FT.CON'
          LI     2,10     *
AGAIN     LDCR   *5+,13   *
          SBZ    12       *
          MOV    0,8      *
LOOP      DEC    8        *
          JNE    LOOP     *
          DEC    2        *
          JNE    AGAIN    *
          JMP    GO       *
*****
CLK        DATA >00C3,>00C2,>00D1,>00E1
          DATA >00C3,>00C2,>00D1,>00E1
          DATA >00C3,>00C2,>0081,>0041

```

```
AIN      DATA >00C3,>00C2,>0081,>0041
          DATA >0000,>1212,>1212,>1F1F
          DATA >1F1F
DSP1     DATA >1CE0,>1C81,>1A12,>17F3
          DATA >18E4,>18F5,>17F6,>1C67
          DATA >1C08,>1C89
STORE    BSS    >2
AIN      BSS    >8
COUT     DATA  OUT
OUT       BSS    >###
*****
```

APPENDIX A.3

```

IDT      / DIS /      *
ACRG     >150        *
BSS      >20         *
LWPI     >150        *
LI       0,DATA      * INPUT TABLE AIN, BIN OR OUT.
LI       3,##        * NUMBER N
LI       4,0         * RESET COUNTERS, >0000->1FFF,
LI       5,0         * >2000->3FFF,
LI       6,0         * >4000->5FFF,
LI       7,0         * >6000->7FFF,
LI       8,0         * >8000->9FFF,
LI       9,0         * >A000->BFFF,
LI      10,0         * >C000->DFFF,
LI      11,0         * >E000->FFFF.
LOOP     MOV      *0+,1 *
          CI      1,>1FFF *
          JLE     LOOP1  *
          CI      1,>3FFF *
          JLE     LOOP2  *
          CI      1,>5FFF *
          JLE     LOOP3  *
          CI      1,>7FFF *
          JLE     LOOP4  *
          CI      1,>9FFF *
          JLE     LOOP5  *
          CI      1,>BFFF *
          JLE     LOOP6  *
          CI      1,>DFFF *
          JLE     LOOP7  *
          CI      1,>FFFF *
          JLE     LOOP8  *
LOOP0    DEC      3      *
          JNE     LOOP   * END OF TABLE? NO, GO OVER
          JMP     DSPR    * YES, GO TO DISPLAY
LOOP1    INC      4      * INC COUNTER 1
          JMP     LOOP0
LOOP2    INC      5      * INC COUNTER 2
          JMP     LOOP0
LOOP3    INC      6      * INC COUNTER 3
          JMP     LOOP0
LOOP4    INC      7      * INC COUNTER 4
          JMP     LOOP0
LOOP5    INC      8      * INC COUNTER 5
          JMP     LOOP0
LOOP6    INC      9      * INC COUNTER 6
          JMP     LOOP0
LOOP7    INC     10      * INC COUNTER 7
          JMP     LOOP0
LOOP8    INC     11      * INC COUNTER 8
          JMP     LOOP0  * GO TO CHECK END OF TABLE
*****
DSPR     LI      12,>420 * SUBROUTINE TO DISPLAY
          LI      0,>20   * DONE
GO       LI      1,DSP   *
          LI      2,10    *
AGAIN    LDCR    *1+,13  *
          SBZ     12      *
          MOV     0,13    *
LP       DEC     13      *
          JNE     LP      *
          DEC     2       *
          JNE     AGAIN   *

```

```
        JMP      GO          *
*****
DSF      DATA  >1FF0,>1FF1,>1FF2,>1FF3
          DATA  >1A14,>1C05,>1C86,>1867
          DATA  >1FF8,>1FF9
DATA      BSS    >###
*****
```

APPENDIX A.4

```

IDT  FLT1  *
AORG >150  *
BSS   >20  *
LWPI  >150  *
LI    0,A0  * @ OF A0 STARTING NUMBER
LI    1,A1  * @ OF A1 STARTING NUMBER
LI    2,##  * NUMBER N
LI    3,OUT  * @ OF OUTPUT TABLE
CLEAR LI    7,CLK * @ OF CLOCK INPUTS
LI    8,FIN  * @ OF F INPUTS
MOV    8,15  *)SEND 'F.CLEAR'
BL     @SEND  *)
BL     @SEND  *)
MOV    15,8  *)
FLT   MOV    1,15  *(SEND I1
BL     @SEND  *(
BL     @SEND  *(
MOV    15,1  *(
DECT   1      *
MOV    8,15  *)SEND 'FT.I2.F2'
BL     @SEND  *)
BL     @SEND  *)
MOV    15,8  *)
MOV    0,15  *(SEND IO
BL     @SEND  *(
BL     @SEND  *(
MOV    15,0  *(
DECT   0      *
INC    *0     * NEXT SEQUENTIAL NUMBER
MOV    8,15  *)SEND 'FT.I2.F2'
BL     @SEND  *)
BL     @SEND  *)
BL     @GET   * GET 'D1
BL     @SEND  *(SEND 'NEXT.OUT'
BL     @SEND  *(
MOV    15,8  *(
BL     @GET   * GET DO
DEC    2      * DECREMENT A COUNTER
JNE    CLEAR * IF NOT ZERO, GO ALL OVER
DSPG  LI    1,DSP1 * IF NOT,DISPLAY 'FLT..DONE'
      B     @DSFR *
*****
SEND  MOV    *7+,10 * SUB-ROUTINE FOR SENDING
      MOVB   *15+,10 * DATA TO TEST CIRCUIT
      LI     12,>20  *
      LDCR   10,0    *
      B      *11     *
*****
SEND1 MOV    *7+,10 * SUB-ROUTINE FOR SENDING
      SWPB   10      * DATA DURING 'GET' SUBROUTINE
      LI     12,>20  *
      LDCR   10,8    *
      B      *11     *
*****
GET   LI     12,0    * SUB-ROUTINE FOR SOFTWARE
      SBO    0       * RESET AND GETTING DATA
      SBZ    15      * FROM TEST CIRCUIT.
      MOV    11,@STORE *
      BL     @SEND1  *
      LI     12,>30  *
      STCR   9,8     *
      SWPB   9       *

```

```

        BL      @SEND1      *
        MOV     @STORE,11   *
        LI      12,>30      *
        STCR    9,8         *
        MOV     9,*3+       *
        B       #11         *
*****
DSPR     LI      12,>420     * SUB-ROUTINE FOR
        LI      0,>20       * DISPLAYING
GO       MOV     1,5        * 'FLT..DONE'
        LI      2,10       *
AGAIN    LDCR    #5+,13     *
        SBZ     12         *
        MOV     0,8        *
LOOP     DEC     8          *
        JNE     LOOP       *
        DEC     2          *
        JNE     AGAIN      *
        JMP     GO         *
*****
A0       DATA  >####
A1       DATA  >####
CLK      DATA  >00C3,>00C2,>00D1,>00E1
        DATA  >00C3,>00C2,>00D1,>00E1
        DATA  >00C3,>00C2,>0081,>0041
        DATA  >00C3,>00C2,>0081,>0041
AIN      DATA  >0000,>1212,>1212,>1F1F
        DATA  >1F1F
DSP1     DATA  >18E0,>1C71,>18F2,>17F3
        DATA  >17F4,>1A15,>1C06,>1C87
        DATA  >1868,>1FF9
STORE    BSS    >2
OUT      BSS    >###
*****

```

APPENDIX A.5

FIXED TO FLOATING CONVERSION

1	-	>4000	>0002	2	-	>4000	>0004
3	-	>6000	>0004	4	-	>4000	>0006
5	-	>5000	>0006	6	-	>6000	>0006
7	-	>7000	>0006	8	-	>4000	>0008
9	-	>4800	>0008	10	-	>5000	>0008
11	-	>5800	>0008	12	-	>6000	>0008
13	-	>6800	>0008	14	-	>7000	>0008
15	-	>7800	>0008	16	-	>4000	>000A
17	-	>4400	>000A	18	-	>4800	>000A
19	-	>4C00	>000A	20	-	>5000	>000A
21	-	>5400	>000A	22	-	>5800	>000A
23	-	>5C00	>000A	24	-	>6000	>000A
25	-	>6400	>000A	26	-	>6800	>000A
27	-	>6C00	>000A	28	-	>7000	>000A
29	-	>7400	>000A	30	-	>7800	>000A
31	-	>7C00	>000A	32	-	>4000	>000C
33	-	>4200	>000C	34	-	>4400	>000C
35	-	>4600	>000C	36	-	>4800	>000C
37	-	>4A00	>000C	38	-	>4C00	>000C
39	-	>4E00	>000C	40	-	>5000	>000C
41	-	>5200	>000C	42	-	>5400	>000C
43	-	>5600	>000C	44	-	>5800	>000C
45	-	>5A00	>000C	46	-	>5C00	>000C
47	-	>5E00	>000C	48	-	>6000	>000C
49	-	>6200	>000C	50	-	>6400	>000C
51	-	>6600	>000C	52	-	>6800	>000C
53	-	>6A00	>000C	54	-	>6C00	>000C
55	-	>6E00	>000C	56	-	>7000	>000C
57	-	>7200	>000C	58	-	>7400	>000C
59	-	>7600	>000C	60	-	>7800	>000C
61	-	>7A00	>000C	62	-	>7C00	>000C
63	-	>7E00	>000C	64	-	>4000	>000E
65	-	>4100	>000E	66	-	>4200	>000E
67	-	>4300	>000E	68	-	>4400	>000E
69	-	>4500	>000E	70	-	>4600	>000E
71	-	>4700	>000E	72	-	>4800	>000E
73	-	>4900	>000E	74	-	>4A00	>000E
75	-	>4B00	>000E	76	-	>4C00	>000E
77	-	>4D00	>000E	78	-	>4E00	>000E
79	-	>4F00	>000E	80	-	>5000	>000E
81	-	>5100	>000E	82	-	>5200	>000E
83	-	>5300	>000E	84	-	>5400	>000E
85	-	>5500	>000E	86	-	>5600	>000E
87	-	>5700	>000E	88	-	>5800	>000E
89	-	>5900	>000E	90	-	>5A00	>000E
91	-	>5B00	>000E	92	-	>5C00	>000E
93	-	>5D00	>000E	94	-	>5E00	>000E
95	-	>5F00	>000E	96	-	>6000	>000E
97	-	>6100	>000E	98	-	>6200	>000E
99	-	>6300	>000E	100	-	>6400	>000E
101	-	>6500	>000E	102	-	>6600	>000E
103	-	>6700	>000E	104	-	>6800	>000E
105	-	>6900	>000E	106	-	>6A00	>000E
107	-	>6B00	>000E	108	-	>6C00	>000E
109	-	>6D00	>000E	110	-	>6E00	>000E
111	-	>6F00	>000E	112	-	>7000	>000E
113	-	>7100	>000E	114	-	>7200	>000E
115	-	>7300	>000E	116	-	>7400	>000E

```

117 - >7500 >000E
119 - >7700 >000E
121 - >7900 >000E
123 - >7B00 >000E
125 - >7D00 >000E
127 - >7F00 >000E
129 - >4080 >0010
131 - >4180 >0010
133 - >4280 >0010
135 - >4380 >0010
137 - >4480 >0010
139 - >4580 >0010
141 - >4680 >0010
143 - >4780 >0010
145 - >4880 >0010
147 - >4980 >0010
149 - >4A80 >0010
151 - >4B80 >0010
153 - >4C80 >0010
155 - >4D80 >0010
157 - >4E80 >0010
159 - >4F80 >0010
161 - >5080 >0010
163 - >5180 >0010
165 - >5280 >0010
167 - >5380 >0010
169 - >5480 >0010
171 - >5580 >0010
173 - >5680 >0010
175 - >5780 >0010
177 - >5880 >0010
179 - >5980 >0010
181 - >5A80 >0010
183 - >5B80 >0010
185 - >5C80 >0010
187 - >5D80 >0010
189 - >5E80 >0010
191 - >5F80 >0010
193 - >6080 >0010
195 - >6180 >0010
197 - >6280 >0010
199 - >6380 >0010
201 - >6480 >0010
203 - >6580 >0010
205 - >6680 >0010
207 - >6780 >0010
209 - >6880 >0010
211 - >6980 >0010
213 - >6A80 >0010
215 - >6B80 >0010
217 - >6C80 >0010
219 - >6D80 >0010
221 - >6E80 >0010
223 - >6F80 >0010
225 - >7080 >0010
227 - >7180 >0010
229 - >7280 >0010
231 - >7380 >0010
233 - >7480 >0010
235 - >7580 >0010
237 - >7680 >0010
239 - >7780 >0010

```

```

118 - >7600 >000E
120 - >7800 >000E
122 - >7A00 >000E
124 - >7C00 >000E
126 - >7E00 >000E
128 - >4000 >0010
130 - >4100 >0010
132 - >4200 >0010
134 - >4300 >0010
136 - >4400 >0010
138 - >4500 >0010
140 - >4600 >0010
142 - >4700 >0010
144 - >4800 >0010
146 - >4900 >0010
148 - >4A00 >0010
150 - >4B00 >0010
152 - >4C00 >0010
154 - >4D00 >0010
156 - >4E00 >0010
158 - >4F00 >0010
160 - >5000 >0010
162 - >5100 >0010
164 - >5200 >0010
166 - >5300 >0010
168 - >5400 >0010
170 - >5500 >0010
172 - >5600 >0010
174 - >5700 >0010
176 - >5800 >0010
178 - >5900 >0010
180 - >5A00 >0010
182 - >5B00 >0010
184 - >5C00 >0010
186 - >5D00 >0010
188 - >5E00 >0010
190 - >5F00 >0010
192 - >6000 >0010
194 - >6100 >0010
196 - >6200 >0010
198 - >6300 >0010
200 - >6400 >0010
202 - >6500 >0010
204 - >6600 >0010
206 - >6700 >0010
208 - >6800 >0010
210 - >6900 >0010
212 - >6A00 >0010
214 - >6B00 >0010
216 - >6C00 >0010
218 - >6D00 >0010
220 - >6E00 >0010
222 - >6F00 >0010
224 - >7000 >0010
226 - >7100 >0010
228 - >7200 >0010
230 - >7300 >0010
232 - >7400 >0010
234 - >7500 >0010
236 - >7600 >0010
238 - >7700 >0010
240 - >7800 >0010

```

241	-	>7880	>0010	242	-	>7900	>0010
243	-	>7980	>0010	244	-	>7A00	>0010
245	-	>7A80	>0010	246	-	>7B00	>0010
247	-	>7B80	>0010	248	-	>7C00	>0010
249	-	>7C80	>0010	250	-	>7D00	>0010
251	-	>7D80	>0010	252	-	>7E00	>0010
253	-	>7E80	>0010	254	-	>7F00	>0010
255	-	>7F80	>0010	256	-	>4000	>0012
257	-	>4040	>0012	258	-	>4080	>0012
259	-	>40C0	>0012	260	-	>4100	>0012
261	-	>4140	>0012	262	-	>4180	>0012
263	-	>41C0	>0012	264	-	>4200	>0012
265	-	>4240	>0012	266	-	>4280	>0012
267	-	>42C0	>0012	268	-	>4300	>0012
269	-	>4340	>0012	270	-	>4380	>0012
271	-	>43C0	>0012	272	-	>4400	>0012
273	-	>4440	>0012	274	-	>4480	>0012
275	-	>44C0	>0012	276	-	>4500	>0012
277	-	>4540	>0012	278	-	>4580	>0012
279	-	>45C0	>0012	280	-	>4600	>0012
281	-	>4640	>0012	282	-	>4680	>0012
283	-	>46C0	>0012	284	-	>4700	>0012
285	-	>4740	>0012	286	-	>4780	>0012
287	-	>47C0	>0012	288	-	>4800	>0012
289	-	>4840	>0012	290	-	>4880	>0012
291	-	>48C0	>0012	292	-	>4900	>0012
293	-	>4940	>0012	294	-	>4980	>0012
295	-	>49C0	>0012	296	-	>4A00	>0012
297	-	>4A40	>0012	298	-	>4A80	>0012
299	-	>4AC0	>0012	300	-	>4B00	>0012
301	-	>4B40	>0012	302	-	>4B80	>0012
303	-	>4BC0	>0012	304	-	>4C00	>0012
305	-	>4C40	>0012	306	-	>4C80	>0012
307	-	>4CC0	>0012	308	-	>4D00	>0012
309	-	>4D40	>0012	310	-	>4D80	>0012
311	-	>4DC0	>0012	312	-	>4E00	>0012
313	-	>4E40	>0012	314	-	>4E80	>0012
315	-	>4EC0	>0012	316	-	>4F00	>0012
317	-	>4F40	>0012	318	-	>4F80	>0012
319	-	>4FC0	>0012	320	-	>5000	>0012
321	-	>5040	>0012	322	-	>5080	>0012
323	-	>50C0	>0012	324	-	>5100	>0012
325	-	>5140	>0012	326	-	>5180	>0012
327	-	>51C0	>0012	328	-	>5200	>0012
329	-	>5240	>0012	330	-	>5280	>0012
331	-	>52C0	>0012	332	-	>5300	>0012
333	-	>5340	>0012	334	-	>5380	>0012
335	-	>53C0	>0012	336	-	>5400	>0012
337	-	>5440	>0012	338	-	>5480	>0012
339	-	>54C0	>0012	340	-	>5500	>0012
341	-	>5540	>0012	342	-	>5580	>0012
343	-	>55C0	>0012	344	-	>5600	>0012
345	-	>5640	>0012	346	-	>5680	>0012
347	-	>56C0	>0012	348	-	>5700	>0012
349	-	>5740	>0012	350	-	>5780	>0012
351	-	>57C0	>0012	352	-	>5800	>0012
353	-	>5840	>0012	354	-	>5880	>0012
355	-	>58C0	>0012	356	-	>5900	>0012
357	-	>5940	>0012	358	-	>5980	>0012
359	-	>59C0	>0012	360	-	>5A00	>0012
361	-	>5A40	>0012	362	-	>5A80	>0012
363	-	>5AC0	>0012	364	-	>5B00	>0012

365	-	>5B40	>0012	366	-	>5B80	>0012
367	-	>5BC0	>0012	368	-	>5C00	>0012
369	-	>5C40	>0012	370	-	>5C80	>0012
371	-	>5CC0	>0012	372	-	>5D00	>0012
373	-	>5D40	>0012	374	-	>5D80	>0012
375	-	>5DC0	>0012	376	-	>5E00	>0012
377	-	>5E40	>0012	378	-	>5E80	>0012
379	-	>5EC0	>0012	380	-	>5F00	>0012
381	-	>5F40	>0012	382	-	>5F80	>0012
383	-	>5FC0	>0012	384	-	>6000	>0012
385	-	>6040	>0012	386	-	>6080	>0012
387	-	>60C0	>0012	388	-	>6100	>0012
389	-	>6140	>0012	390	-	>6180	>0012
391	-	>61C0	>0012	392	-	>6200	>0012
393	-	>6240	>0012	394	-	>6280	>0012
395	-	>62C0	>0012	396	-	>6300	>0012
397	-	>6340	>0012	398	-	>6380	>0012
399	-	>63C0	>0012	400	-	>6400	>0012
401	-	>6440	>0012	402	-	>6480	>0012
403	-	>64C0	>0012	404	-	>6500	>0012
405	-	>6540	>0012	406	-	>6580	>0012
407	-	>65C0	>0012	408	-	>6600	>0012
409	-	>6640	>0012	410	-	>6680	>0012
411	-	>66C0	>0012	412	-	>6700	>0012
413	-	>6740	>0012	414	-	>6780	>0012
415	-	>67C0	>0012	416	-	>6800	>0012
417	-	>6840	>0012	418	-	>6880	>0012
419	-	>68C0	>0012	420	-	>6900	>0012
421	-	>6940	>0012	422	-	>6980	>0012
423	-	>69C0	>0012	424	-	>6A00	>0012
425	-	>6A40	>0012	426	-	>6A80	>0012
427	-	>6AC0	>0012	428	-	>6B00	>0012
429	-	>6B40	>0012	430	-	>6B80	>0012
431	-	>6BC0	>0012	432	-	>6C00	>0012
433	-	>6C40	>0012	434	-	>6C80	>0012
435	-	>6CC0	>0012	436	-	>6D00	>0012
437	-	>6D40	>0012	438	-	>6D80	>0012
439	-	>6DC0	>0012	440	-	>6E00	>0012
441	-	>6E40	>0012	442	-	>6E80	>0012
443	-	>6EC0	>0012	444	-	>6F00	>0012
445	-	>6F40	>0012	446	-	>6F80	>0012
447	-	>6FC0	>0012	448	-	>7000	>0012
449	-	>7040	>0012	450	-	>7080	>0012
451	-	>70C0	>0012	452	-	>7100	>0012
453	-	>7140	>0012	454	-	>7180	>0012
455	-	>71C0	>0012	456	-	>7200	>0012
457	-	>7240	>0012	458	-	>7280	>0012
459	-	>72C0	>0012	460	-	>7300	>0012
461	-	>7340	>0012	462	-	>7380	>0012
463	-	>73C0	>0012	464	-	>7400	>0012
465	-	>7440	>0012	466	-	>7480	>0012
467	-	>74C0	>0012	468	-	>7500	>0012
469	-	>7540	>0012	470	-	>7580	>0012
471	-	>75C0	>0012	472	-	>7600	>0012
473	-	>7640	>0012	474	-	>7680	>0012
475	-	>76C0	>0012	476	-	>7700	>0012
477	-	>7740	>0012	478	-	>7780	>0012
479	-	>77C0	>0012	480	-	>7800	>0012
481	-	>7840	>0012	482	-	>7880	>0012
483	-	>78C0	>0012	484	-	>7900	>0012
485	-	>7940	>0012	486	-	>7980	>0012
487	-	>79C0	>0012	488	-	>7A00	>0012

489	-	>7A40	>0012	490	-	>7A80	>0012
491	-	>7AC0	>0012	492	-	>7B00	>0012
493	-	>7B40	>0012	494	-	>7B80	>0012
495	-	>7BC0	>0012	496	-	>7C00	>0012
497	-	>7C40	>0012	498	-	>7C80	>0012
499	-	>7CC0	>0012	500	-	>7D00	>0012
501	-	>7D40	>0012	502	-	>7D80	>0012
503	-	>7DC0	>0012	504	-	>7E00	>0012
505	-	>7E40	>0012	506	-	>7E80	>0012
507	-	>7EC0	>0012	508	-	>7F00	>0012
509	-	>7F40	>0012	510	-	>7F80	>0012
511	-	>7FC0	>0012	512	-	>4000	>0014
513	-	>4020	>0014	514	-	>4040	>0014
515	-	>4060	>0014	516	-	>4080	>0014
517	-	>40A0	>0014	518	-	>40C0	>0014
519	-	>40E0	>0014	520	-	>4100	>0014
521	-	>4120	>0014	522	-	>4140	>0014
523	-	>4160	>0014	524	-	>4180	>0014
525	-	>41A0	>0014	526	-	>41C0	>0014
527	-	>41E0	>0014	528	-	>4200	>0014
529	-	>4220	>0014	530	-	>4240	>0014
531	-	>4260	>0014	532	-	>4280	>0014
533	-	>42A0	>0014	534	-	>42C0	>0014
535	-	>42E0	>0014	536	-	>4300	>0014
537	-	>4320	>0014	538	-	>4340	>0014
539	-	>4360	>0014	540	-	>4380	>0014
541	-	>43A0	>0014	542	-	>43C0	>0014
543	-	>43E0	>0014	544	-	>4400	>0014
545	-	>4420	>0014	546	-	>4440	>0014
547	-	>4460	>0014	548	-	>4480	>0014
549	-	>44A0	>0014	550	-	>44C0	>0014
551	-	>44E0	>0014	552	-	>4500	>0014
553	-	>4520	>0014	554	-	>4540	>0014
555	-	>4560	>0014	556	-	>4580	>0014
557	-	>45A0	>0014	558	-	>45C0	>0014
559	-	>45E0	>0014	560	-	>4600	>0014
561	-	>4620	>0014	562	-	>4640	>0014
563	-	>4660	>0014	564	-	>4680	>0014
565	-	>46A0	>0014	566	-	>46C0	>0014
567	-	>46E0	>0014	568	-	>4700	>0014
569	-	>4720	>0014	570	-	>4740	>0014
571	-	>4760	>0014	572	-	>4780	>0014
573	-	>47A0	>0014	574	-	>47C0	>0014
575	-	>47E0	>0014	576	-	>4800	>0014
577	-	>4820	>0014	578	-	>4840	>0014
579	-	>4860	>0014	580	-	>4880	>0014
581	-	>48A0	>0014	582	-	>48C0	>0014
583	-	>48E0	>0014	584	-	>4900	>0014
585	-	>4920	>0014	586	-	>4940	>0014
587	-	>4960	>0014	588	-	>4980	>0014
589	-	>49A0	>0014	590	-	>49C0	>0014
591	-	>49E0	>0014	592	-	>4A00	>0014
593	-	>4A20	>0014	594	-	>4A40	>0014
595	-	>4A60	>0014	596	-	>4A80	>0014
597	-	>4AA0	>0014	598	-	>4AC0	>0014
599	-	>4AE0	>0014	600	-	>4B00	>0014
601	-	>4B20	>0014	602	-	>4B40	>0014
603	-	>4B60	>0014	604	-	>4B80	>0014
605	-	>4BA0	>0014	606	-	>4BC0	>0014
607	-	>4BE0	>0014	608	-	>4C00	>0014
609	-	>4C20	>0014	610	-	>4C40	>0014
611	-	>4C60	>0014	612	-	>4C80	>0014

613	-	>4CA0	>0014	614	-	>4CC0	>0014
615	-	>4CE0	>0014	616	-	>4D00	>0014
617	-	>4D20	>0014	618	-	>4D40	>0014
619	-	>4D60	>0014	620	-	>4D80	>0014
621	-	>4DA0	>0014	622	-	>4DC0	>0014
623	-	>4DE0	>0014	624	-	>4E00	>0014
625	-	>4E20	>0014	626	-	>4E40	>0014
627	-	>4E60	>0014	628	-	>4E80	>0014
629	-	>4EA0	>0014	630	-	>4EC0	>0014
631	-	>4EE0	>0014	632	-	>4F00	>0014
633	-	>4F20	>0014	634	-	>4F40	>0014
635	-	>4F60	>0014	636	-	>4F80	>0014
637	-	>4FA0	>0014	638	-	>4FC0	>0014
639	-	>4FE0	>0014	640	-	>5000	>0014
641	-	>5020	>0014	642	-	>5040	>0014
643	-	>5060	>0014	644	-	>5080	>0014
645	-	>50A0	>0014	646	-	>50C0	>0014
647	-	>50E0	>0014	648	-	>5100	>0014
649	-	>5120	>0014	650	-	>5140	>0014
651	-	>5160	>0014	652	-	>5180	>0014
653	-	>51A0	>0014	654	-	>51C0	>0014
655	-	>51E0	>0014	656	-	>5200	>0014
657	-	>5220	>0014	658	-	>5240	>0014
659	-	>5260	>0014	660	-	>5280	>0014
661	-	>52A0	>0014	662	-	>52C0	>0014
663	-	>52E0	>0014	664	-	>5300	>0014
665	-	>5320	>0014	666	-	>5340	>0014
667	-	>5360	>0014	668	-	>5380	>0014
669	-	>53A0	>0014	670	-	>53C0	>0014
671	-	>53E0	>0014	672	-	>5400	>0014
673	-	>5420	>0014	674	-	>5440	>0014
675	-	>5460	>0014	676	-	>5480	>0014
677	-	>54A0	>0014	678	-	>54C0	>0014
679	-	>54E0	>0014	680	-	>5500	>0014
681	-	>5520	>0014	682	-	>5540	>0014
683	-	>5560	>0014	684	-	>5580	>0014
685	-	>55A0	>0014	686	-	>55C0	>0014
687	-	>55E0	>0014	688	-	>5600	>0014
689	-	>5620	>0014	690	-	>5640	>0014
691	-	>5660	>0014	692	-	>5680	>0014
693	-	>56A0	>0014	694	-	>56C0	>0014
695	-	>56E0	>0014	696	-	>5700	>0014
697	-	>5720	>0014	698	-	>5740	>0014
699	-	>5760	>0014	700	-	>5780	>0014
701	-	>57A0	>0014	702	-	>57C0	>0014
703	-	>57E0	>0014	704	-	>5800	>0014
705	-	>5820	>0014	706	-	>5840	>0014
707	-	>5860	>0014	708	-	>5880	>0014
709	-	>58A0	>0014	710	-	>58C0	>0014
711	-	>58E0	>0014	712	-	>5900	>0014
713	-	>5920	>0014	714	-	>5940	>0014
715	-	>5960	>0014	716	-	>5980	>0014
717	-	>59A0	>0014	718	-	>59C0	>0014
719	-	>59E0	>0014	720	-	>5A00	>0014
721	-	>5A20	>0014	722	-	>5A40	>0014
723	-	>5A60	>0014	724	-	>5A80	>0014
725	-	>5AA0	>0014	726	-	>5AC0	>0014
727	-	>5AE0	>0014	728	-	>5B00	>0014
729	-	>5B20	>0014	730	-	>5B40	>0014
731	-	>5B60	>0014	732	-	>5B80	>0014
733	-	>5BA0	>0014	734	-	>5BC0	>0014
735	-	>5BE0	>0014	736	-	>5C00	>0014

737	-	>5C20	>0014	738	-	>5C40	>0014
739	-	>5C60	>0014	740	-	>5C80	>0014
741	-	>5CA0	>0014	742	-	>5CC0	>0014
743	-	>5CE0	>0014	744	-	>5D00	>0014
745	-	>5D20	>0014	746	-	>5D40	>0014
747	-	>5D60	>0014	748	-	>5D80	>0014
749	-	>5DA0	>0014	750	-	>5DC0	>0014
751	-	>5DE0	>0014	752	-	>5E00	>0014
753	-	>5E20	>0014	754	-	>5E40	>0014
755	-	>5E60	>0014	756	-	>5E80	>0014
757	-	>5EA0	>0014	758	-	>5EC0	>0014
759	-	>5EE0	>0014	760	-	>5F00	>0014
761	-	>5F20	>0014	762	-	>5F40	>0014
763	-	>5F60	>0014	764	-	>5F80	>0014
765	-	>5FA0	>0014	766	-	>5FC0	>0014
767	-	>5FE0	>0014	768	-	>6000	>0014
769	-	>6020	>0014	770	-	>6040	>0014
771	-	>6060	>0014	772	-	>6080	>0014
773	-	>60A0	>0014	774	-	>60C0	>0014
775	-	>60E0	>0014	776	-	>6100	>0014
777	-	>6120	>0014	778	-	>6140	>0014
779	-	>6160	>0014	780	-	>6180	>0014
781	-	>61A0	>0014	782	-	>61C0	>0014
783	-	>61E0	>0014	784	-	>6200	>0014
785	-	>6220	>0014	786	-	>6240	>0014
787	-	>6260	>0014	788	-	>6280	>0014
789	-	>62A0	>0014	790	-	>62C0	>0014
791	-	>62E0	>0014	792	-	>6300	>0014
793	-	>6320	>0014	794	-	>6340	>0014
795	-	>6360	>0014	796	-	>6380	>0014
797	-	>63A0	>0014	798	-	>63C0	>0014
799	-	>63E0	>0014	800	-	>6400	>0014
801	-	>6420	>0014	802	-	>6440	>0014
803	-	>6460	>0014	804	-	>6480	>0014
805	-	>64A0	>0014	806	-	>64C0	>0014
807	-	>64E0	>0014	808	-	>6500	>0014
809	-	>6520	>0014	810	-	>6540	>0014
811	-	>6560	>0014	812	-	>6580	>0014
813	-	>65A0	>0014	814	-	>65C0	>0014
815	-	>65E0	>0014	816	-	>6600	>0014
817	-	>6620	>0014	818	-	>6640	>0014
819	-	>6660	>0014	820	-	>6680	>0014
821	-	>66A0	>0014	822	-	>66C0	>0014
823	-	>66E0	>0014	824	-	>6700	>0014
825	-	>6720	>0014	826	-	>6740	>0014
827	-	>6760	>0014	828	-	>6780	>0014
829	-	>67A0	>0014	830	-	>67C0	>0014
831	-	>67E0	>0014	832	-	>6800	>0014
833	-	>6820	>0014	834	-	>6840	>0014
835	-	>6860	>0014	836	-	>6880	>0014
837	-	>68A0	>0014	838	-	>68C0	>0014
839	-	>68E0	>0014	840	-	>6900	>0014
841	-	>6920	>0014	842	-	>6940	>0014
843	-	>6960	>0014	844	-	>6980	>0014
845	-	>69A0	>0014	846	-	>69C0	>0014
847	-	>69E0	>0014	848	-	>6A00	>0014
849	-	>6A20	>0014	850	-	>6A40	>0014
851	-	>6A60	>0014	852	-	>6A80	>0014
853	-	>6AA0	>0014	854	-	>6AC0	>0014
855	-	>6AE0	>0014	856	-	>6B00	>0014
857	-	>6B20	>0014	858	-	>6B40	>0014
859	-	>6B60	>0014	860	-	>6B80	>0014

861	-	>6BA0	>0014	862	-	>6BC0	>0014
863	-	>6BE0	>0014	864	-	>6C00	>0014
865	-	>6C20	>0014	866	-	>6C40	>0014
867	-	>6C60	>0014	868	-	>6C80	>0014
869	-	>6CA0	>0014	870	-	>6CC0	>0014
871	-	>6CE0	>0014	872	-	>6D00	>0014
873	-	>6D20	>0014	874	-	>6D40	>0014
875	-	>6D60	>0014	876	-	>6D80	>0014
877	-	>6DA0	>0014	878	-	>6DC0	>0014
879	-	>6DE0	>0014	880	-	>6E00	>0014
881	-	>6E20	>0014	882	-	>6E40	>0014
883	-	>6E60	>0014	884	-	>6E80	>0014
885	-	>6EA0	>0014	886	-	>6EC0	>0014
887	-	>6EE0	>0014	888	-	>6F00	>0014
889	-	>6F20	>0014	890	-	>6F40	>0014
891	-	>6F60	>0014	892	-	>6F80	>0014
893	-	>6FA0	>0014	894	-	>6FC0	>0014
895	-	>6FE0	>0014	896	-	>7000	>0014
897	-	>7020	>0014	898	-	>7040	>0014
899	-	>7060	>0014	900	-	>7080	>0014
901	-	>70A0	>0014	902	-	>70C0	>0014
903	-	>70E0	>0014	904	-	>7100	>0014
905	-	>7120	>0014	906	-	>7140	>0014
907	-	>7160	>0014	908	-	>7180	>0014
909	-	>71A0	>0014	910	-	>71C0	>0014
911	-	>71E0	>0014	912	-	>7200	>0014
913	-	>7220	>0014	914	-	>7240	>0014
915	-	>7260	>0014	916	-	>7280	>0014
917	-	>72A0	>0014	918	-	>72C0	>0014
919	-	>72E0	>0014	920	-	>7300	>0014
921	-	>7320	>0014	922	-	>7340	>0014
923	-	>7360	>0014	924	-	>7380	>0014
925	-	>73A0	>0014	926	-	>73C0	>0014
927	-	>73E0	>0014	928	-	>7400	>0014
929	-	>7420	>0014	930	-	>7440	>0014
931	-	>7460	>0014	932	-	>7480	>0014
933	-	>74A0	>0014	934	-	>74C0	>0014
935	-	>74E0	>0014	936	-	>7500	>0014
937	-	>7520	>0014	938	-	>7540	>0014
939	-	>7560	>0014	940	-	>7580	>0014
941	-	>75A0	>0014	942	-	>75C0	>0014
943	-	>75E0	>0014	944	-	>7600	>0014
945	-	>7620	>0014	946	-	>7640	>0014
947	-	>7660	>0014	948	-	>7680	>0014
949	-	>76A0	>0014	950	-	>76C0	>0014
951	-	>76E0	>0014	952	-	>7700	>0014
953	-	>7720	>0014	954	-	>7740	>0014
955	-	>7760	>0014	956	-	>7780	>0014
957	-	>77A0	>0014	958	-	>77C0	>0014
959	-	>77E0	>0014	960	-	>7800	>0014
961	-	>7820	>0014	962	-	>7840	>0014
963	-	>7860	>0014	964	-	>7880	>0014
965	-	>78A0	>0014	966	-	>78C0	>0014
967	-	>78E0	>0014	968	-	>7900	>0014
969	-	>7920	>0014	970	-	>7940	>0014
971	-	>7960	>0014	972	-	>7980	>0014
973	-	>79A0	>0014	974	-	>79C0	>0014
975	-	>79E0	>0014	976	-	>7A00	>0014
977	-	>7A20	>0014	978	-	>7A40	>0014
979	-	>7A60	>0014	980	-	>7A80	>0014
981	-	>7AA0	>0014	982	-	>7AC0	>0014
983	-	>7AE0	>0014	984	-	>7B00	>0014

985 - >7B20 >0014
987 - >7B40 >0014
989 - >7BA0 >0014
991 - >7BE0 >0014
993 - >7C20 >0014
995 - >7C60 >0014
997 - >7CA0 >0014
999 - >7CE0 >0014

986 - >7B40 >0014
988 - >7B80 >0014
990 - >7BC0 >0014
992 - >7C00 >0014
994 - >7C40 >0014
996 - >7C80 >0014
998 - >7CC0 >0014
1000 - >7D00 >0014

APPENDIX A.6

```

IDT  / FLT2  / *
AORG >150    *
BSS  >20     *
LWPI >150    *
LI   0,AIN   * @ OF A INPUTS
LI   1,###   * A COUNTER
LI   2,OUT   * @ OF DESIRED OUTPUTS
LI   3,8     * ERROR COUNTER
LI   13,SENDT * @ OF SEND TABLE
CLEAR LI 7,CLK * @ OF CLOCK INPUTS
LI   8,FIN   * @ OF F INPUTS
MOV  8,15    *)SEND 'F.CLEAR'
BL   @SEND   *)
BL   @SEND   *)
MOV  15,8    *)
LI   14,2    * SET COUNTER TO 2
FLT  MOV 0,15 *)SEND I1- IO
BL   @SEND   *)
BL   @SEND   *)
MOV  15,0    *)
MOV  8,15    *)SEND 'FT.I2.F2'-'FT.I2.F2'
BL   @SEND   *)
BL   @SEND   *)
MOV  15,8    *)
DEC  14      * I1 AND IO TAKEN?
JEQ  NEXT    * IF YES, GET OUTPUT D
NEXT B @FLT   * IF NOT, TAKE IO
BL   @GET    * GET D1
MOV  8,15    *)SEND 'NEXT.OUT'
BL   @SEND   *)
BL   @SEND   *)
MOV  15,8    *)
BL   @GET    * GET DO
CI   3,0     * IS ERR COUNT 0 ?
JEQ  DSPB    * IF YES, DISPLAY BAD
DEC  1       * DECREMENT A COUNTER
JNE  CLEAR   * IF NOT ZERO, GO ALL OVER
CI   3,8     * IF ZERO,CHECK ERRORS
JNE  DSPB    * IF ERRORS,DISPLAY BAD
DSPG LI 1,DSP1 * IF NOT,DISPLAY GOOD
B    @DSFR   *
DSPB LI 1,DSP2 *
B    @DSFR   *
*****
SEND  MOV *7+,10 * SUB ROUTINE FOR SENDING
      MOVB *15+,10 * DATA TO TEST CIRCUIT
      MOV  10,*13+ * AND ALSO TO STORE IN SEND
      LI   12,>20 * TABLE FOR PROGRAM TESTING
      LDCR 10,0   *
      B    *11    *
*****
SEND1 MOV *7+,10 * SUBROUTINE FOR SENDING
      SWPB 10     * DATA DURING 'GET'
      MOV  10,*13+ * SUBROUTINE
      LI   12,>20 *
      LDCR 10,8   *
      B    *11    *
*****
GET   LI 12,0    * SUB ROUTINE FOR SOFTWARE
      SBO  0     * RESET AND GETTING DATA
      SBZ  15    * FROM TEST CIRCUIT
      MOV  11,@STORE *

```

```

        BL      @SEND1      *
        LI      12,>30      *
        STCR    9,8         *
        SWPB    9           *
        BL      @SEND1      *
        MOV     @STORE,11   *
        LI      12,>30      *
        STCR    9,8         *
        C       9,*2+       * IS TEST O/P EQUAL DESIRED O/P ?
        JNE     ERR         * NO, DECREMENT ERROR COUNTER
        B       #11         * YES, GET NEXT O/P
ERR      DEC    3           * ERROR COUNTER
        B       #11         *
*****
DSPR     LI     12,>420     * SUB ROUTINE FOR
        LI     0,>20       * DISPLAYING
GO        MOV    1,5       * 'FLT..GOOD' OR 'FLT..BAD'
        LI     2,10       *
AGAIN    LDCR   #5+,13    *
        SBZ    12         *
        MOV    0,8        *
LOOP     DEC    8         *
        JNE    LOOP       *
        DEC    2          *
        JNE    AGAIN      *
        JMP    GO         *
*****
AIN      BSS    >####
OUT      DATA  >4000,>0002,>4000,>0004
        DATA  >6000,>0004,>4000,>0006
        DATA  >0000,>0000,>0000,>0000
        DATA  >0000,>0000,>0000,>0000
CLK      DATA  >00C3,>00C2,>00D1,>00E1
        DATA  >00C3,>00C2,>00D1,>00E1
        DATA  >00C3,>00C2,>00C3,>00C2
        DATA  >00C3,>00C2,>0081,>0041
        DATA  >00C3,>00C2,>0081,>0041
FIN      DATA  >0000,>1212,>1212
        DATA  >1212,>1F1F,>1F1F
DSP1     DATA  >18E0,>1C71,>18F2,>17F3
        DATA  >17F4,>1C25,>1A36,>1A37
        DATA  >1A18,>1FF9
DSP2     DATA  >18E0,>1C71,>18F2,>17F3
        DATA  >17F4,>1835,>1886,>1A17
        DATA  >1FF8,>1FF9
STORE    BSS    >2
SENDT    BSS    >###
*****

```

APPENDIX A.7

```

IDT  FLT  *
AORG >150  *
BSS  >20   *
LWPI >150  *
LI   0,AIN * @ OF A INPUTS
LI   1,###  * A COUNTER
LI   4,OUT  * @ OF OUTPUT TABLE
CLEAR LI 7,CLK * @ OF CLOCK INPUTS
LI   8,FIN  * @ OF F INPUTS
MOV  8,15   *) SEND 'F.CLEAR'
BL   @SEND  *)
BL   @SEND  *)
MOV  15,8   *)
LI   14,2   * SET COUNTER TO 2
FLT  MOV  0,15 *) SEND I1- IO
BL   @SEND  *)
BL   @SEND  *)
MOV  15,0   *)
MOV  8,15   *) SEND 'FT.I2.F2'-'FT.I2.F2'
BL   @SEND  *)
BL   @SEND  *)
MOV  15,8   *)
DEC  14     * I1 AND IO TAKEN?
JEQ  NEXT   * IF YES, GET OUTPUT D
B    @FLT   * IF NOT, TAKE IO
NEXT BL @GET * GET D1
MOV  8,15   *) SEND 'NEXT.OUT'
BL   @SEND  *)
BL   @SEND  *)
MOV  15,8   *)
BL   @GET   * GET IO
DEC  1      * DECREMENT A COUNTER
JNE  CLEAR  * IF NOT ZERO, GO ALL OVER
DSPG LI 1,DSP1 * IF NOT, DISPLAY 'DONE'
B    @DSPR  *
*****
SEND MOV *7+,10 * SUB-ROUTINE FOR SENDING
MOV  *15+,10 * DATA TO TEST CIRCUIT
LI   12,>20  *
LDCR 10,0   *
B    *11    *
*****
SEND1 MOV *7+,10 * SUB-ROUTINE FOR SENDING
SWPB 10     * DATA DURING 'GET'
LI   12,>20  * SUB-ROUTINE
LDCR 10,8   *
B    *11    *
*****
GET  LI 12,0 * SUB-ROUTINE FOR SOFTWARE
SBO  0      * RESET AND GETTING DATA
SBZ  15     * FROM TEST CIRCUIT
MOV  11,@STORE *
BL   @SEND1 *
LI   12,>30  *
STCR 9,8    *
SWPB 9       *
BL   @SEND1 *
MOV  @STORE,11 *
LI   12,>30  *
STCR 9,8    *
B    *11    *
*****

```

```

DSPR    LI    12,>420      * SUB ROUTINE FOR
        LI    0,>20        * DISPLAYING
GO       MOV   1,5         * 'FLT..DONE'
        LI    2,10        *
AGAIN    LDCR  *5+,13      *
        SBZ   12          *
        MOV   0,8         *
LOOP     DEC   8           *
        JNE   LOOP        *
        DEC   2           *
        JNE   AGAIN       *
        JMP   GO          *

```

```

*****
CLK      DATA >00C3,>00C2,>00D1,>00E1
        DATA >00C3,>00C2,>00D1,>00E1
        DATA >00C3,>00C2,>0081,>0041
        DATA >00C3,>00C2,>0081,>0041
FIN      DATA >0000,>1212,>1212,>1F1F
        DATA >1F1F
DSP1     DATA >18E0,>1C71,>18F2,>17F3
        DATA >17F4,>1A15,>1C06,>1C87
        DATA >1868,>1FF9
STORE    BSS   >2
AIN      BSS   >###
OUT      BSS   >###
*****

```

APPENDIX A.8

```

IDT  FT1  *
AORG >150  *
BSS  >20   *
LWPI >150  *
LI   0,AIN * @ OF A INPUTS
LI   1,##  * A COUNTER
LI   2,OUT * @ OF DESIRED OUTPUTS
LI   3,8   * ERROR COUNTER
LI   4,ERROR * @ OF ERROR TABLE
LI   13,ECD * @ OF OUTPUT TABLE
CLEAR LI 7,CLK * @ OF CLOCK INPUTS
LI 8,FIN * @ OF F INPUTS
MOV 8,15 * )SEND 'F.CLEAR'
BL @SEND * )
BL @SEND * )
MOV 15,8 * )
LI 14,2 * SET COUNTER TO 2
FLT  MOV 0,15 * )SEND I1- IO
BL @SEND * )
BL @SEND * )
MOV 15,0 * )
MOV 8,15 * (SEND 'FT.I2.F2'-'FT.I2.F2'
BL @SEND * (
BL @SEND * (
MOV 15,8 * (
DEC 14 * I1 AND IO TAKEN?
JEQ NEXT * IF YES, GET OUTPUT D
B @FLT * IF NOT, TAKE IO
NEXT BL @GET * GET D1
MOV 8,15 * )SEND 'NEXT.OUT'
BL @SEND * )
BL @SEND * )
MOV 15,8 * )
BL @GET * GET DO
DEC 1 * DECREMENT A COUNTER
JNE CLEAR * IF NOT ZERO, GO ALL OVER
CI 3,8 * IF ZERO,CHECK ERRORS
JNE DSPB * IF ERRORS,DISPLAY BAD
DSPG LI 1,DSP1 * IF NOT,DISPLAY GOOD
B @DSPR *
DSPB LI 1,DSP2 *
MOV *13,@DSP3 *
B @DSPR *
*****
SEND MOV *7+,10 * SUB-ROUTINE FOR SENDING
MOVB *15+,10 * DATA TO TEST CIRCUIT
LI 12,>20 *
LDCR 10,0 *
B *11 *
*****
SEND1 MOV *7+,10 * SUB-ROUTINE FOR SENDING
SWPB 10 * DATA DURING 'GET'
LI 12,>20 * SUB-ROUTINE
LDCR 10,8 *
B *11 *
*****
GET LI 12,0 * SUB-ROUTINE FOR SOFTWARE
SBO 0 * RESET AND GETTING DATA
SBZ 15 * FROM TEST CIRCUIT
MOV 11,@STORE *
BL @SEND1 *
LI 12,>30 *

```

```

      STCR 9,8      *
      SWPB 9        *
      BL   @SEND1   *
      MOV  @STORE,11 *
      LI   12,>30    *
      STCR 9,8      *
      C    9,*2+     * IS TEST O/P EQUAL DESIRED O/P ?
      JNE  ERR       * NO, DECREMENT ERROR COUNTER
      B    *11       * YES, GET NEXT O/P
*****
ERR    DECT 2        * SUB-ROUTINE FOR STORING
      MOV  *2+,*4+    * DESIRED DATA VS ERROR
      MOV  9,*4+     * DATA IN CASE OF ERROR
      DEC  3         * DATA FROM TEST CIRCUIT
      INCT 13        *
      CI   3,0       *
      JEQ  DSPB      *
      B    *11       *
*****
DSPR   LI   12,>420   * SUB-ROUTINE FOR
      LI   0,>20      * DISPLAYING
GO      MOV  1,5      * 'FLT..GOOD' OR 'FLT.BAD.#'
      LI   2,10      * WHERE # IS NUMBER OF ERRORS
AGAIN   LDCR *5+,*13  *
      SBZ  12        *
      MOV  0,6       *
LOOP    DEC  8        *
      JNE  LOOP      *
      DEC  2         *
      JNE  AGAIN     *
      JMP  GO        *
*****
CLK     DATA >00C3,>00C2,>00D1,>00E1
      DATA >00C3,>00C2,>00D1,>00E1
      DATA >00C3,>00C2,>00C3,>00C2
      DATA >00C3,>00C2,>0081,>0041
      DATA >00C3,>00C2,>0081,>0041
FIN     DATA >0000,>1212,>1212,>1212
      DATA >1F1F,>1F1F
DSP1    DATA >18E0,>1C71,>18F2,>17F3
      DATA >17F4,>1C25,>1A36,>1A37
      DATA >1A18,>1FF9
DSP2    DATA >18E0,>1C71,>18F2,>17F3
      DATA >17F4,>1835,>1886,>1A17
      DATA >17F8
DSP3    DATA >1FF9
ECD     DATA >1C09,>1F99,>1A49,>1B09
      DATA >1999,>1929,>1829,>1F89
      DATA >1809,>1909
ERROR   BSS   >28
STORE   BSS   >2
AIN     BSS   >###
OUT     BSS   >###
*****

```

APPENDIX A.9

```

IDT      < ADD1      *
AORG     >150        *
BSS      >20         *
LWPI     >150        *
LI       0,AIN       * @ OF A INPUTS
LI       1,##        * A COUNTER
LI       2,OUT       * @ OF DESIRED OUTPUTS
LI       3,8         * ERROR COUNTER
LI       13,SENDT    * @ OF SEND TABLE
START    LI       5,BIN * @ OF B INPUTS
CLEAR    LI       6,## * B COUNTER
          LI       7,CLK * @ OF CLOCK INPUTS
          LI       8,FIN * @ OF F INPUTS
          MOV      8,15 * )SEND 'F.CLEAR'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          LI       14,2 * SET COUNTER TO 2
ADD       MOV      0,15 * )SEND A1- A0
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,0 * )
          MOV      5,15 * (SEND B1- B0
          BL       @SEND * (
          BL       @SEND * (
          MOV      15,5 * (
          MOV      8,15 * )SEND 'ADD.F2'-'ADD.F2'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          DEC      14   * (AB)1 AND (AB)0 ADDED ?
          JEQ      NEXT * IF YES, GET OUTPUT D
          B        @ADD * IF NOT, ADD (AB)0
NEXT      BL       @GET * GET D1
          MOV      8,15 * )SEND 'NEXT.OUT'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          BL       @GET * GET D0
          CI       3,0  * IS ERR COUNT 0 ?
          JEQ      DSPB * IF YES, DISPLAY BAD
          DEC      6    * IF NOT, DECREMENT B COUNTER
          JEQ      CHECK * IF B IS ZERO, CHECK A
          DECT     0    * )IF NOT ZERO, RETAIN FIRST
          DECT     0    * )VALUE OF A AND APPLY
          JMP      CLEAR * )ANOTHER VALUE OF B
CHECK     DEC      1    * DECREMENT A COUNTER
          JNE      START * IF NOT ZERO, GO ALL OVER
          CI       3,8  * IF BOTH ZERO, CHECK ERRORS
          JNE      DSPB * IF ERRORS, DISPLAY BAD
DSPG      LI       1,DSP1 * IF NOT, DISPLAY GOOD
          B        @DSPR *
DSPB      LI       1,DSP2 *
          B        @DSPR *
*****
SEND      MOV      *7+,10 * SUB-ROUTINE FOR SENDING
          MOVB     *15+,10 * DATA TO TEST CIRCUIT
          MOV      10,*13+ * AND ALSO TO STORE IN SEND
          LI       12,>20 * TABLE FOR PROGRAM TESTING
          LDCR     10,0   *
          B        *11   *
*****

```

```

SEND1  MOV    *7+,10      * SUB-ROUTINE FOR SENDING
      SWPB    10          * DATA TO TEST CIRCUIT DURING
      MOV     10,*13+     * 'GET' SUBROUTINE
      LI      12,>20      *
      LDCR    10,8        *
      B       *11         *
*****
GET     LI      12,0       * SUB-ROUTINE FOR SOFTWARE
      SBO      0          * RESET AND GETTING DATA
      SBZ      15         * FROM TEST CIRCUIT
      MOV     11,@STORE   *
      BL      @SEND1      *
      LI      12,>30      *
      STCR    9,8         *
      SWPB    9          *
      BL      @SEND1      *
      MOV     @STORE,11   *
      LI      12,>30      *
      STCR    9,8         *
      C       9,*2+      * IS TEST O/P EQUAL DESIRED O/P ?
      JNE     ERR        * NO, DECREMENT ERROR COUNTER
      B       *11         * YES, GET NEXT O/P
ERR     DEC     3         *
      B       *11         *
*****
DSPR    LI      12,>420    * SUB ROUTINE FOR
      LI      0,>20      * DISPLAYING
GO      MOV     1,5       * 'Add..Good' OR 'Add..Bad'
      LI      2,10       *
AGAIN   LDCR    *5+,13    *
      SBZ      12         *
      MOV     0,8         *
LOOP    DEC     8         *
      JNE     LOOP       *
      DEC     2          *
      JNE     AGAIN      *
      JMP     GO         *
*****
AIN     BSS     >####
BIN     BSS     >####
OUT     BSS     >####
CLK     DATA  >00C3,>00C2,>00D1,>00E1
      DATA  >00C5,>00C9,>00C3,>00C2
      DATA  >00D1,>00E1,>00C5,>00C9
      DATA  >00C3,>00C2,>0081,>0041
      DATA  >00C3,>00C2,>0081,>0041
FIN     DATA  >0000,>1818,>1818,>1F1F
      DATA  >1F1F
DSP1    DATA  >1880,>1A11,>1A12,>17F3
      DATA  >17F4,>1C25,>1A36,>1A37
      DATA  >1A18,>1FF9
DSP2    DATA  >1880,>1A11,>1A12,>17F3
      DATA  >17F4,>1835,>1886,>1A17
      DATA  >1FF8,>1FF9
STORE   BSS     >2
SENDT   BSS     >###
*****

```

APPENDIX A.10

```

IDT      / ADDT/      *
AORG     >150        *
BSS      >20         *
LWPI     >150        *
LI       0,AIN        * @ OF A INPUTS
LI       1,##        * A COUNTER
LI       2,OUT        * @ OF OUTPUT TABLE
START    LI       5,BIN * @ OF B INPUTS
CLEAR    LI       6,##  * B COUNTER
LI       7,CLK        * @ OF CLOCK INPUTS
LI       8,FIN        * @ OF F INPUTS
MOV      8,15        *) SEND 'F.CLEAR'
BL       @SEND        *)
BL       @SEND        *)
MOV      15,8        *)
LI       14,2        * SET COUNTER TO 2
ADD      MOV      0,15 *) SEND A1- A0
BL       @SEND        *)
BL       @SEND        *)
MOV      15,0        *)
MOV      5,15        *) SEND B1, B0
BL       @SEND        *)
BL       @SEND        *)
MOV      15,5        *)
MOV      8,15        *) SEND 'ADD.F2'--'ADD.F2'
BL       @SEND        *)
BL       @SEND        *)
MOV      15,8        *)
DEC      14          * (AB)1 AND (AB)0 ADDED ?
JEQ      NEXT        * IF YES, GET OUTPUT D
B        @ADD         * IF NOT, ADD (AB)0
NEXT     BL       @GET  * GET D1
MOV      8,15        *) SEND 'NEXT.OUT'
BL       @SEND        *)
BL       @SEND        *)
MOV      15,8        *)
BL       @GET        * GET D0
DEC      6           * DECREMENT B COUNTER
JEQ      CHECK       * IF IT IS ZERO,CHECK A
DECT     0           *) IF NOT ZERO, RETAIN FIRST
DECT     0           *) VALUE OF A AND APPLY
JMP      CLEAR       *) ANOTHER VALUE OF B
CHECK    DEC      1   * DECREMENT A COUNTER
JNE      START       * IF NOT ZERO, GO ALL OVER
B        @DSPR        * DISPLAY 'ADD..DONE'
*****
SEND     MOV      *7+,10 * SUB-ROUTINE FOR
MOV      *15+,10      * SENDING DATA TO TEST CIRCUIT
LI       12,>20       *
LDCR     10,0         *
B        *11          *
*****
SEND1    MOV      *7+,10 * SUB-ROUTINE FOR
SWPB     10          * SENDING DATA DURING
LI       12,>20       * 'GET' SUB-ROUTINE
LDCR     10,8         *
B        *11          *
*****
GET      LI       12,0 * SUB-ROUTINE FOR SOFTWARE
SBO      0          * RESET AND GETTING DATA
SBZ      15         * FROM TEST CIRCUIT
MOV      11,@STORE  *

```

```

        BL      @SEND1      *
        LI      12,>30      *
        STCR    9,8         *
        SWPB    9           *
        BL      @SEND1      *
        MOV     @STORE,11   *
        LI      12,>30      *
        STCR    9,8         *
        MOV     9,*2+       *
        B       *11         *
*****
DSPR    LI      12,>420     * SUB ROUTINE FOR
        LI      0,>20       * DISPLAYING
GO       LI      5,DSP1     * 'ADD..DONE'
        LI      2,10       *
AGAIN    LDCR    *5+,13     *
        SBZ     12         *
        MOV     0,8        *
LOOP     DEC     8          *
        JNE     LOOP       *
        DEC     2          *
        JNE     AGAIN      *
        JMP     GO         *
*****
CLK      DATA  >00C3,>00C2,>00D1,>00E1
        DATA  >00C5,>00C9,>00C3,>00C2
        DATA  >00D1,>00E1,>00C5,>00C9
        DATA  >00C3,>00C2,>0081,>0041
        DATA  >00C3,>00C2,>0081,>0041
FIN      DATA  >0000,>1818,>1818,>1F1F
        DATA  >1F1F
DSP1     DATA  >1880,>1A11,>1A12,>17F3
        DATA  >17F4,>1A15,>1C06,>1C87
        DATA  >1868,>1FF9
ECD      DATA  >1C09,>1F99,>1A49,>1B09
        DATA  >1999,>1929,>1829,>1F89
        DATA  >1809,>1909
STORE    BSS     >2
AIN      BSS     >###
BIN      BSS     >###
OUT      BSS     >###
*****

```

APPENDIX A.11

```

IDT      ' AT '      *
AORG     >150        *
BSS      >20         *
LWPI     >150        *
LI       0,AIN       * @ OF A INPUTS
LI       1,##        * A COUNTER
LI       2,OUT       * @ OF DESIRED OUTPUTS
LI       3,8         * ERROR COUNTER
LI       4,ERROR     * @ OF ERROR TABLE
LI       13,ECD      * @ OF OUTPUT TABLE
START    LI       5,BIN * @ OF B INPUTS
CLEAR    LI       6,## * B COUNTER
          LI       7,CLK * @ OF CLOCK INPUTS
          LI       8,FIN * @ OF F INPUTS
          MOV      8,15 * )SEND 'F.CLEAR'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          LI       14,2 * SET COUNTER TO 2
          ADD      MOV 0,15 * )SEND A1- A0
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,0 * )
          MOV      5,15 * (SEND B1- B0
          BL       @SEND * (
          BL       @SEND * (
          MOV      15,5 * (
          MOV      8,15 * (SEND 'ADD.F2'-'ADD.F2'
          BL       @SEND * (
          BL       @SEND * (
          MOV      15,8 * (
          DEC      14  * (AB)1 AND (AB)0 ADDED ?
          JEQ      NEXT * IF YES, GET OUTPUT D
          B        @ADD * IF NOT, ADD (AB)0
NEXT      BL       @GET * GET D1
          MOV      8,15 * )SEND 'NEXT.OUT'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          BL       @GET * GET D0
          DEC      4   * IF NOT, DECREMENT B COUNTER
          JEQ      CHECK * IF IT IS ZERO, CHECK A
          DECT     0   * )IF NOT ZERO, RETAIN FIRST
          DECT     0   * )VALUE OF A AND APPLY
          JMP      CLEAR * )ANOTHER VALUE OF B
CHECK     DEC      1   * DECREMENT A COUNTER
          JNE      START * IF NOT ZERO, GO ALL OVER
          CI       3,8 * IF BOTH ZERO, CHECK ERRORS
          JNE      DSPB * IF ERRORS, DISPLAY BAD
DSPG      LI       1,DSP1 * IF NOT, DISPLAY GOOD
          B        @DSPR *
DSPB      LI       1,DSP2 *
          MOV      *13,@DSP3 *
          B        @DSPR *
*****
SEND      MOV      *7+,10 * SUB-ROUTINE FOR
          MOVB     *15+,10 * SENDING DATA TO TEST CIRCUIT
          LI       12,>20 *
          LDCR     10,0  *
          B        *11  *
*****
SEND1     MOV      *7+,10 * SUB-ROUTINE FOR SENDING

```

```

        SWPB    10          * DATA DURING 'GET'
        LI      12,>20      * SUB-ROUTINE
        LDCR    10,8        *
        B       *11         *
*****
GET      LI      12,0        * SUB-ROUTINE FOR SOFTWARE
        SBO      0          * RESET AND GETTING DATA
        SBZ      15         * FROM TEST CIRCUIT
        MOV      11,@STORE  *
        BL       @SEND1     *
        LI      12,>30      *
        STCR     9,8        *
        SWPB     9          *
        BL       @SEND1     *
        MOV      @STORE,11  *
        LI      12,>30      *
        STCR     9,8        *
        C        9,*2+      * IS TEST O/P EQUAL DESIRED O/P ?
        JNE      ERR        * NO, GO TO ERROR ROUTINE
        B       *11         * YES, GET NEXT O/P
*****
ERR      DECT     2          * SUB-ROUTINE FOR STORING
        MOV      *2+,*4+    * DESIRED DATA VS ERROR
        MOV      9,*4+     * DATA IN CASE OF ERROR
        DEC      3          * DATA FROM TEST CIRCUIT
        INCT     13         *
        CI       3,0        *
        JEQ      DSPB       *
        B       *11         *
*****
DSPR     LI      12,>420     * SUB-ROUTINE FOR
        LI      0,>20       * DISPLAYING
GO        MOV      1,5       * 'ADD..GOOD' OR 'ADD.BAD.#'
        LI      2,10        * WHERE # IS NUMBER OF ERRORS.
AGAIN    LDCR     *5+,*13    *
        SBZ      12         *
        MOV      0,8        *
LOOP     DEC      8          *
        JNE      LOOP       *
        DEC      2          *
        JNE      AGAIN      *
        JMP      GO         *
*****
CLK      DATA    >00C3,>00C2,>00D1,>00E1
        DATA     >00C5,>00C9,>00C3,>00C2
        DATA     >00D1,>00E1,>00C5,>00C9
        DATA     >00C3,>00C2,>0081,>0041
        DATA     >00C3,>00C2,>0081,>0041
FIN      DATA     >0000,>1818,>1818,>1F1F
        DATA     >1F1F
DSP1     DATA     >1880,>1A11,>1A12,>17F3
        DATA     >17F4,>1C25,>1A36,>1A37
        DATA     >1A18,>1FF9
DSP2     DATA     >1880,>1A11,>1A12,>17F3
        DATA     >17F4,>1835,>1886,>1A17
        DATA     >17F8
DSP3     DATA     >1FF9
ECD      DATA     >1C09,>1F99,>1A49,>1B09
        DATA     >1999,>1929,>1829,>1F89
        DATA     >1809,>1909
ERROR    BSS      >28
STORE    BSS      >2

```

```
AIN    BSS    >###  
BIN    BSS    >###  
OUT    BSS    >###
```

```
*****
```

APPENDIX A.12

```

IDT      < MTO      *
AORG     >150        *
BSS      >20         *
LWPI     >150        *
LI       0,AIN        * @ OF A INPUTS
LI       1,##         * A COUNTER
LI       2,OUT        * @ OF DESIRED OUTPUTS
LI       3,8          * ERROR COUNTER
LI       13,SENDT     * @ OF SEND TABLE
START    LI       5,BIN * @ OF B INPUTS
CLEAR    LI       6,##  * B COUNTER
LI       7,CLK        * @ OF CLOCK INPUTS
LI       8,FIN        * @ OF F INPUTS
MOV      8,15         *)SEND 'F.CLEAR'
BL       @SEND        *)
BL       @SEND        *)
MOV      15,8         *)
LI       14,2         * SET COUNTER TO 2
MUL      MOV      0,15 *)SEND A1- AO
BL       @SEND        *)
BL       @SEND        *)
MOV      15,0         *)
MOV      5,15         *)SEND B1- B0
BL       @SEND        *)
BL       @SEND        *)
MOV      15,5         *)
MOV      8,15         *)SEND 'MUL.F2'-'MUL.F2'
BL       @SEND        *)
BL       @SEND        *)
MOV      15,8         *)
DEC      14           * (AB)1 AND (AB)0 MULTIPLIED ?
JEQ      NEXT        * IF YES, GET OUTPUT D
B         @MUL        * IF NOT, MULTIPLY (AB)0
NEXT     BL       @GET * GET D1
MOV      8,15         *)SEND 'NEXT.OUT'
BL       @SEND        *)
BL       @SEND        *)
MOV      15,8         *)
BL       @GET        * GET D0
CI       3,0          * IS ERR COUNT 0 ?
JEQ      DSPB        * IF YES, DISPLAY BAD
DEC      6            * IF NOT, DECREMENT B COUNTER
JEQ      CHECK       * IF B IS ZERO, CHECK A
DECT     0            *)IF NOT ZERO, RETAIN FIRST
DECT     0            *)VALUE OF A AND APPLY
JMP      CLEAR       *)ANOTHER VALUE OF B
CHECK    DEC      1    * DECREMENT A COUNTER
JNE      START       * IF NOT ZERO, GO ALL OVER
CI       3,8          * IF BOTH ZERO, CHECK ERRORS
JNE      DSPB        * IF ERRORS, DISPLAY BAD
DSPG     LI       1,DSP1 * IF NOT, DISPLAY GOOD
B         @DSPR       *
DSPB     LI       1,DSP2 *
B         @DSPR       *
*****
SEND     MOV      *7+,10 * SUB-ROUTINE FOR SENDING
MOV      *15+,10 * DATA TO TEST CIRCUIT
MOV      10,*13+ * AND ALSO TO STORE IN SEND
LI       12,>20 * TABLE FOR PROGRAM TESTING
LDCCR   10,0        *
B        *11        *
*****

```

```

SEND1  MOV    *7+,10      * SUB-ROUTINE FOR SENDING
      SWPB    10          * DATA TO TEST CIRCUIT DURING
      MOV     10,*13+     * 'GET' SUBROUTINE
      LI      12,>20      *
      LDCR    10,8        *
      B       *11         *
*****
GET     LI      12,0       * SUB-ROUTINE FOR SOFTWARE
      SBO      0          * RESET AND GETTING DATA
      SBZ      15         * FROM TEST CIRCUIT
      MOV     11,@STORE   *
      BL      @SEND1      *
      LI      12,>30      *
      STCR    9,8         *
      SWPB    9           *
      BL      @SEND1      *
      MOV     @STORE,11   *
      LI      12,>30      *
      STCR    9,8         *
      C        9,*2+      * IS TEST O/P EQUAL DESIRED O/P ?
      JNE     ERR        * NO, DECREMENT ERROR COUNTER
      B       *11         * YES, GET NEXT O/P
ERR     DEC     3          * ERROR COUNTER
      B       *11         *
*****
DSPR    LI      12,>420    * SUB ROUTINE FOR
      LI      0,>20      * DISPLAYING
GO      MOV     1,5       * 'MUL..GOOD' OR 'MUL..BAD'
      LI      2,10       *
AGAIN   LDCR    *5+,13    *
      SBZ      12         *
      MOV     0,8         *
LOOP    DEC     8         *
      JNE     LOOP       *
      DEC     2           *
      JNE     AGAIN      *
      JMP     GO          *
*****
CLK     DATA   >00C3,>00C2,>00D1,>00E1
      DATA   >00C5,>00C9,>00C3,>00C2
      DATA   >00D1,>00E1,>00C5,>00C9
      DATA   >00C3,>00C2,>0081,>0041
      DATA   >00C3,>00C2,>0081,>0041
FIN     DATA   >0000,>1616,>1616,>1F1F
      DATA   >1F1F
DSP1    DATA   >1AA0,>1C11,>1C72,>17F3
      DATA   >17F4,>1C25
      DATA   >1A36,>1A37,>1A18,>1FF9
DSP2    DATA   >1AA0,>1C11,>1C72,>17F3
      DATA   >17F4,>1835,>1886,>1A17
      DATA   >17F8,>1FF9
STORE   BSS     >2
AIN     BSS     >###
BIN     BSS     >###
OUT     BSS     >###
SENDT   BSS     >###
*****

```

APPENDIX A.13

```

IDT      / MULT /      *
AORG     >150          *
BSS      >20           *
LWPI     >150          *
LI       0,AIN         * @ OF A INPUTS
LI       1,##          * A COUNTER
LI       2,OUT         * @ OF OUTPUT TABLE
START    LI           5,BIN * @ OF B INPUTS
CLEAR    LI           6,##  * B COUNTER
LI       7,CLK         * @ OF CLOCK INPUTS
LI       8,FIN         * @ OF F INPUTS
MOV      8,15          *)SEND 'F.CLEAR'
BL       @SEND         *)
BL       @SEND         *)
MOV      15,8          *)
LI       14,2          * SET COUNTER TO 2
MUL      MOV          0,15 *)SEND A1- A0
BL       @SEND         *)
BL       @SEND         *)
MOV      15,0          *)
MOV      5,15          *)SEND B1- B0
BL       @SEND         *)
BL       @SEND         *)
MOV      15,5          *)
MOV      8,15          *)SEND 'MUL.F2'-'MUL.F2'
BL       @SEND         *)
BL       @SEND         *)
MOV      15,8          *)
DEC      14            * (AB)1 AND (AB)0 MULTIPLIED?
JEQ      NEXT          * IF YES, GET OUTPUT D
NEXT     BL            @MUL * IF NOT, MULTIPLY (AB)0
MOV      8,15          *)GET D1
BL       @SEND         *)SEND 'NEXT.OUT'
BL       @SEND         *)
MOV      15,8          *)
BL       @GET          * GET D0
DEC      6             * DECREMENT B COUNTER
JEQ      CHECK         * IF IT IS ZERO,CHECK A
DECT     0             *)IF NOT ZERO, RETAIN FIRST
DECT     0             *)VALUE OF A AND APPLY
CHECK    JMP          CLEAR *)ANOTHER VALUE OF B.
DEC      1             * DECREMENT A COUNTER
JNE      START         * IF NOT ZERO, GO ALL OVER
B        @DSPR         * DISPLAY 'ADD..DONE'
*****
SEND     MOV          *7+,10 * SUB-ROUTINE FOR
MOV      *15+,10       * SENDING DATA TO TEST CIRCUIT
LI       12,>20         *
LDCR     10,0          *
B        *11           *
*****
SEND1    MOV          *7+,10 * SUB-ROUTINE FOR SENDING
SWPB     10            * DATA DURING 'GET'
LI       12,>20         * SUB-ROUTINE
LDCR     10,8          *
B        *11           *
*****
GET      LI           12,0 * SUB-ROUTINE FOR SOFTWARE
SBO      0             * RESET AND GETTING DATA
SBZ      15            * FROM TEST CIRCUIT
MOV      11,@STORE    *

```

```

        BL      @SEND1      *
        LI      12,>30      *
        STCR    9,8         *
        SWPB    9           *
        BL      @SEND1      *
        MOV     @STORE,11   *
        LI      12,>30      *
        STCR    9,8         *
        MOV     9,*2+       *
        B       *11         *
*****
DSPR    LI      12,>420     * SUB-ROUTINE FOR
        LI      0,>20      * DISPLAYING
GO      LI      5,DSP1     * 'MUL..DONE'
        LI      2,10      *
AGAIN   LDCR    *5+,13     *
        SBZ     12         *
        MOV     0,8        *
LOOP    DEC     8          *
        JNE     LOOP       *
        DEC     2          *
        JNE     AGAIN      *
        JMP     GO         *
*****
CLK     DATA   >00C3,>00C2,>00D1,>00E1
        DATA   >00C5,>00C9,>00C3,>00C2
        DATA   >00D1,>00E1,>00C5,>00C9
        DATA   >00C3,>00C2,>0081,>0041
        DATA   >00C3,>00C2,>0081,>0041
FIN     DATA   >0000,>1616,>1616,>1F1F
        DATA   >1F1F
DSP1    DATA   >1AA0,>1C11,>1C72,>17F3
        DATA   >17F4,>1A15,>1C06,>1C87
        DATA   >1868,>1FF9
STORE   BSS     >2
AIN     BSS     >###
BIN     BSS     >###
OUT     BSS     >###
*****

```

APPENDIX A.14

```

IDT      < MT      *
AORG     >150      *
BSS      >20       *
LWPI     >150      *
LI       0,AIN     * @ OF A INPUTS
LI       1,##      * A COUNTER
LI       2,OUT     * @ OF DESIRED OUTPUTS
LI       3,8       * ERROR COUNTER
LI       4,ERROR   * @ OF ERROR TABLE
LI       13,ECD    * @ OF ERR COUNT DSPLY
START    LI       5,BIN * @ OF B INPUTS
CLEAR    LI       6,## * B COUNTER
          LI       7,CLK * @ OF CLOCK INPUTS
          LI       8,FIN * @ OF F INPUTS
          MOV      8,15 * )SEND 'F.CLEAR'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          LI       14,2 * SET COUNTER TO 2
MUL       MOV      0,15 * )SEND A1- A0
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,0 * )
          MOV      5,15 * (SEND B1- B0
          BL       @SEND * (
          BL       @SEND * (
          MOV      15,5 * (
          MOV      8,15 * )SEND 'MUL.F2'-'MUL.F2'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          DEC      14   * (AB)1 AND (AB)0 MULTIPLIED ?
          JEQ      NEXT * IF YES, GET OUTPUT D
          B        @MUL * IF NOT, MULTIPLY (AB)0
NEXT      BL       @GET * GET D1
          MOV      8,15 * )SEND 'NEXT.OUT'
          BL       @SEND * )
          BL       @SEND * )
          MOV      15,8 * )
          BL       @GET * GET D0
          DEC      6     * IF NOT, DECREMENT B COUNTER
          JEQ      CHECK * IF IT IS ZERO, CHECK A
          DECT     0     * )IF NOT ZERO, RETAIN FIRST
          DECT     0     * )VALUE OF A AND APPLY
          JMP      CLEAR * )ANOTHER VALUE OF B
CHECK     DEC      1     * DECREMENT A COUNTER
          JNE      START * IF NOT ZERO, GO ALL OVER
          CI       3,8   * IF BOTH ZERO, CHECK ERRORS
          JNE      DSPB  * IF ERRORS, DISPLAY BAD
DSPG      LI       1,DSP1 * IF NOT, DISPLAY GOOD
          B        @DSPR *
DSPB      LI       1,DSP2 *
          MOV      *13,@DSP3 *
          B        @DSPR *
*****
SEND      MOV      *7+,10 * SUB-ROUTINE FOR
          MOVB     *15+,10 * SENDING DATA TO TEST BOARD
          LI       12,>20 * TABLE FOR VERIFICATION
          LDCR     10,0   *
          B        *11    *
*****
SEND1     MOV      *7+,10 * SUB-ROUTINE FOR SENDING

```

```

        SWPB    10          * DATA DURING 'GET'
        LI      12,>20      * SUB-ROUTINE.
        LDCR    10,8        *
        B       *11         *
        NOP                     *
*****
GET      LI      12,0        * SUB-ROUTINE FOR SOFTWARE
        SBO      0          * RESET AND GETTING DATA
        SBZ      15         * FROM TEST BOARD.
        MOV      11,@STORE  *
        BL       @SEND1     *
        LI      12,>30      *
        STCR     9,8        *
        SWPB     9          *
        BL       @SEND1     *
        MOV      @STORE,11  *
        LI      12,>30      *
        STCR     9,8        *
        C        9,*2+      * IS TEST O/P EQUAL DESIRED O/P ?
        JNE      ERR        * NO,GO TO ERROR ROUTINE
        B       *11         *
*****
ERR      DECT     2          * SUB-ROUTINE FOR STORING
        MOV      *2+,*4+    * DESIRED DATA VS ERROR
        MOV      9,*4+     * DATA IN CASE OF ERROR
        DEC      3          * DATA FROM TEST BOARD.
        INCT     13         *
        CI       3,0        * ERROR COUNT ZERO?
        JEQ      DSPB       * DISPLAY, 'BAD'
        B       *11         *
*****
DSPR     LI      12,>420     * SUB-ROUTINE FOR
        LI      0,>20       * DISPLAYING
GO        MOV      1,5       * 'MUL..GOOD' OR 'MUL.BAD.#'
        LI      2,10        * WHERE # IS NUMBER OF ERRORS.
AGAIN     LDCR    *5+,13    *
        SBZ      12         *
        MOV      0,8        *
LOOP      DEC      8         *
        JNE      LOOP       *
        DEC      2          *
        JNE      AGAIN      *
        JMP      GO         *
*****
CLK      DATA >00C3,>00C2,>00D1,>00E1
        DATA >00C5,>00C9,>00C3,>00C2
        DATA >00D1,>00E1,>00C5,>00C9
        DATA >00C3,>00C2,>0081,>0041
        DATA >00C3,>00C2,>0081,>0041
FIN       DATA >0000,>1616,>1616
        DATA >1F1F,>1F1F
DSP1      DATA >1AA0,>1C11,>1C72,>17F3
        DATA >17F4,>1C25
        DATA >1A36,>1A37,>1A18,>1FF9
DSP2      DATA >1AA0,>1C11,>1C72,>17F3
        DATA >17F4,>1835,>1886,>1A17
        DATA >17F8
DSP3      DATA >1FF9
ECD       DATA >1C09,>1F99,>1A49,>1B09
        DATA >1999,>1929,>1829,>1F89
        DATA >1809,>1909
ERROR     BSS     >28

```

```
STORE  BSS  >2  
AIN    BSS  >###  
BIN    BSS  >###  
OUT    BSS  >###  
*****
```

APPENDIX A.15

	IDT	< DIV	*
	AORG	>150	*
	BSS	>20	*
	LWFI	>150	*
	LI	0,AIN	* @ OF A INPUTS
	LI	1,##	* A COUNTER
	LI	2,OUT	* @ OF DESIRED OUTPUTS
	LI	3,8	* ERROR COUNTER
	LI	13,SENDT	* @ OF OUTPUT TABLE
START	LI	5,BIN	* @ OF B INPUTS
	LI	6,##	* B COUNTER
CLEAR	LI	7,CLK	* @ OF CLOCK INPUTS
	LI	8,FIN	* @ OF F INPUTS
	MOV	8,15	*)SEND 'F.CLEAR'
	BL	@SEND	*)
	BL	@SEND	*)
	MOV	15,8	*)
	LI	14,2	*
DIV	MOV	0,15	*)SEND A1- A0
	BL	@SEND	*)
	BL	@SEND	*)
	MOV	15,0	*)
	MOV	5,15	*)SEND B1- B0
	BL	@SEND	*)
	BL	@SEND	*)
	MOV	15,5	*)
	MOV	8,15	*)SEND 'DIV.F2'-'DIV.F2'
	BL	@SEND	*)
	BL	@SEND	*)
	MOV	15,8	*)
	DEC	14	* AB1 AND ABO ENTERED ?
	JEQ	NEXT	* IF YES, GET OUTPUT D
	B	@DIV	* IF NOT, ENTER (AB)0
NEXT	MOV	8,15	*)SEND 'LREMAIN'
	BL	@SEND	*)
	BL	@SEND	*)
	MOV	15,8	*)
	MOV	8,15	*)SEND 'QBIT3' 9 TIMES
LOOP	LI	14,9	*)R14 IS COUNTER FOR QBIT3
	BL	@SEND	*)
	BL	@SEND	*)
	DECT	15	*)
	DECT	7	*)
	DECT	7	*)
	DEC	14	*)
	JNE	LOOP	*)
	MOV	15,8	*)
	INCT	8	*)
	INCT	7	*)
	INCT	7	*)
	BL	@GET	* GET D1
	MOV	8,15	*)
	BL	@SEND	*)
	BL	@SEND	*)
	MOV	15,8	*)
	BL	@GET	* GET D0
	DEC	6	* IF NOT, DECREMENT B COUNTER
	JEQ	CHECK	* IF IT IS ZERO, CHECK A
	DECT	0	*)IF NOT ZERO, RETAIN FIRST
	DECT	0	*)VALUE OF A AND APPLY
	JMP	CLEAR	*)ANOTHER VALUE OF B
CHECK	DEC	1	* DECREMENT A COUNTER

```

JNE     START      * IF NOT ZERO, GO ALL OVER
CI      3,8        * IF BOTH ZERO,CHECK ERRORS
JNE     DSPB       * IF ERRORS,DISPLAY BAD
DSPG    LI         1,DSP1    * IF NOT,DISPLAY GOOD
        B         @DSPR     *
DSPB    LI         1,DSP2    *
        B         @DSPR     *
*****
SEND    MOV        *7+,10    * SUB ROUTINE FOR
        MOVB       *15+,10   * SENDING DATA TO TEST BOARD
        MOV        10,*13+   * AND ALSO TO STORE IN A
        LI         12,>20    * TABLE FOR VERIFICATION
        LDCR       10,0      *
        B         *11       *
*****
SEND1   MOV        *7+,10    *
        SWPB       10        *
        MOV        10,*13+   *
        LI         12,>20    *
        LDCR       10,8      *
        B         *11       *
*****
GET     LI         12,0      * SUB ROUTINE FOR SOFTWARE
        SBO        0         * RESET AND GETTING DATA
        SBZ        15        * FROM TEST BOARD.
        MOV        11,@STORE *
        BL         @SEND1    *
        LI         12,>30    *
        STCR       9,8       *
        SWPB       9         *
        BL         @SEND1    *
        MOV        @STORE,11 *
        LI         12,>30    *
        STCR       9,8       *
        C          9,*2+     * IS TEST O/P EQUAL DESIRED O/P ?
        JNE        ERR       * NO, DECREMENT ERROR COUNTER
        B         *11       * YES, GET NEXT O/P
ERR     DEC        3         * ERROR COUNTER
        B         *11       *
*****
DSPR    LI         12,>420    * SUB ROUTINE FOR
        LI         0,>20     * DISPLAYING
GO      MOV        1,5       * 'DIV..GOOD' OR 'DIV..BAD'
        LI         2,10      *
AGAIN   LDCR       *5+,13    *
        SBZ        12        *
        MOV        0,8       *
LOOP1   DEC        8         *
        JNE        LOOP1     *
        DEC        2         *
        JNE        AGAIN     *
        JMP        GO        *
*****
CLK     DATA >00C3,>00C2,>00D1,>00E1
        DATA >00C5,>00C9,>00C3,>00C2
        DATA >00D1,>00E1,>00C5,>00C9
        DATA >00C3,>00C2,>00C3,>00C2
        DATA >00C3,>00C2,>0081,>0041
        DATA >00C3,>00C2,>0081,>0041
FIN     DATA >0000,>1616,>1616,>1818
        DATA >1212,>1F1F,>1F1F
DSP1    DATA >1A10,>1F91,>1952,>17F3

```

VITA

Mrugesh Trivedi was born on January 15, 1952, at Ahmedabad, India. After outstanding school years, he joined St. Xavier's College in India for his first year of science. He then joined L. D. College of Engineering for his engineering studies. He graduated in 1974 with a Bachelor's degree in Instrumentation and Control Engineering, obtaining the third rank in the university.

Immediately after his graduation, he joined Indian Petrochemicals Corporation Limited, Baroda, India, as an Instrument Engineer. During his tenure of seven years with that company, he erected, commissioned, and maintained about 3000 instruments. He was also responsible for smooth running of the Instrument Department comprising 35 people.

He joined The University of Tennessee, Knoxville, in September, 1981, in a M.S. program in Computer Engineering.

He has been married to the former Miss Shaivalini Mody since July, 1983, and expecting the NEW ARRIVAL in May, 1984.