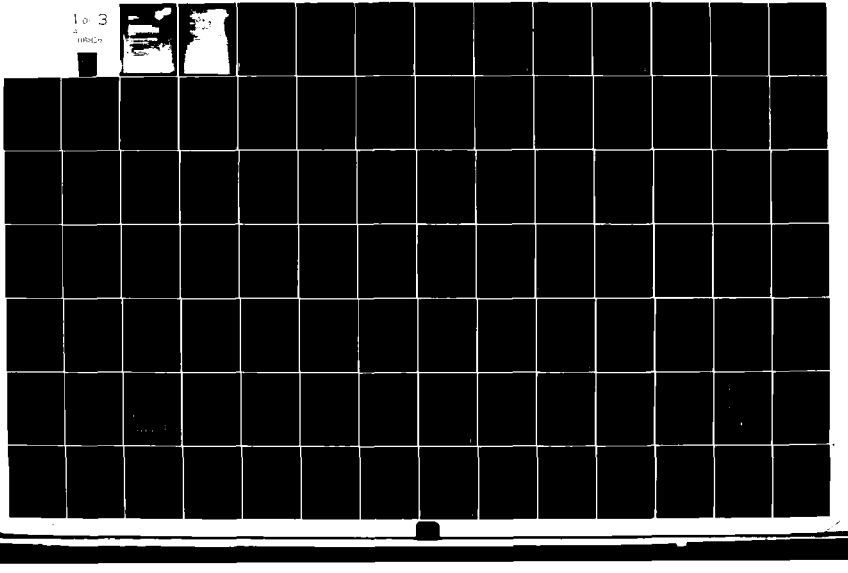
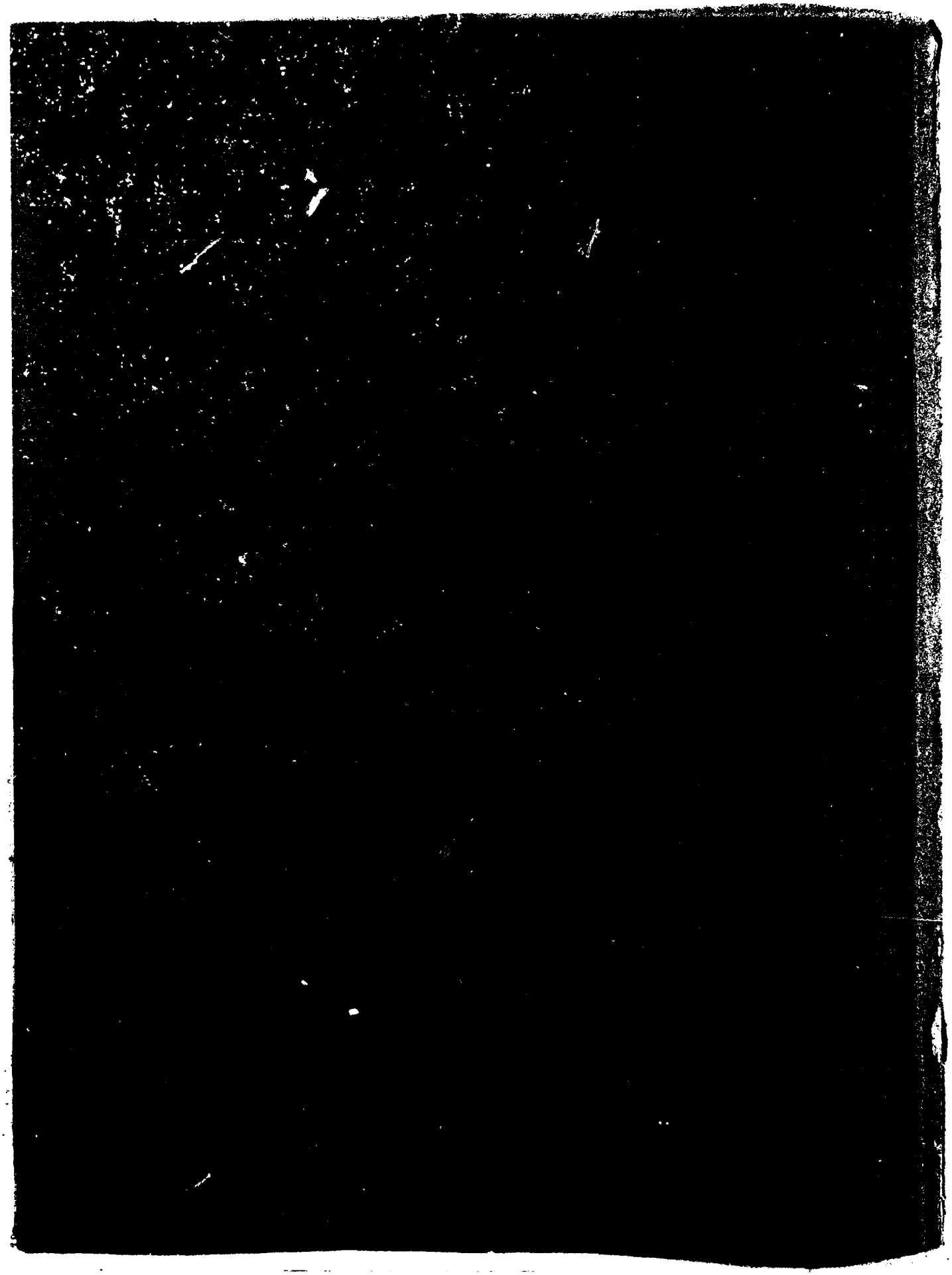
AD-A118 826 VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG D--ETC F/G 9/2
MICROPROCESSOR SELF-TEST: SOFTWARE SELF-TEST FOR AN 8080-BASED --ETC(U)
JUN 82 J R ARMSTRONG, F G GRAY F30602-80-C-0200
UNCLASSIFIED RADC-TR-82-80 NL





UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-82-80	2. GOVT ACCESSION NO.: AD-A118826
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): MICROPROCESSOR SELF-TEST SOFTWARE SELF-TEST FOR AN 8080-BASED SYSTEM USING A MINIMUM OF ADDITIONAL HARDWARE	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report 1 Sep 80 - 31 Aug 81
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): James R. Armstrong F. Gail Gray	8. CONTRACT OR GRANT NUMBER(s): F30602-80-C-0200
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Virginia Polytechnic Institute & State Univ. Dept of Electrical Engineering Blacksburg VA 24061	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F 55811722
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (COEA) Griffiss AFB NY 13441	12. REPORT DATE: June 1982 13. NUMBER OF PAGES: 243
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Lt Dean W. Gonzalez (COEA)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprocessor
Self-Test
Simulation

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

A self-test program for an 8080-based microprocessor system is developed and verified using both high-level simulation and actual hardware components. The goals were minimum execution time, minimum added hardware, and minimum impact on applications software. Within these constraints, maximum fault coverage was obtained.

A high-level simulation language (GSP) was used to verify the execution

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

of the self-test program and to determine the fault coverage. The self-test programs provided excellent diagnostic routines to test the simulation models and indeed were used to discover several faulty simulation models.

Finally, a complete self-testing hardware system was constructed to verify that the self-test program would run in the background of an applications program.

Report Summary

Introduction

This report describes the research efforts carried out under Air Force Contract F30602-80-C-0200. The purpose of the research was to develop techniques for the self-testing of microprocessors. These techniques were then implemented for a specific, 8080 based microcomputer system. The implementation took the form of a set of self-test routines and a small amount of added, self-test hardware. In order to assess the effectiveness of the self-test software, a "chip" level simulation model was developed and used to simulate faults in the systems and thus rate the effectiveness of the self-test software. Finally, a real 8080 system was built and the self-test software executed on it in order to demonstrate its compatibility with a real time computer system environment.

Self-Test Techniques

The report describes one program in a proposed library of self-test programs for microprocessor based systems. The library is to contain a set of programs with varying degrees of fault coverage and execution times. This report describes a self-test designed for minimum execution time, minimum use of added hardware, and minimum interference with the main system tasks. Within these constraints, maximum fault coverage is desired.

The basic approach was to partition the self-test program into segments that require from 2 to 4 milliseconds each to

execute. A timer is used to generate program interrupts at a frequency selected by the user (e.g., every two seconds). Each interrupt causes execution of the next self-test segment. The CPU test and parallel I/O port test both execute in the first segment. Memory tests posed the greatest demand upon execution time. Only 128 bytes of ROM or 32 bytes of RAM can be tested in the 2-4 ms window. Thus memory testing is carried out in a series of segments. Serial I/O port tests require 2 segments at 9600 Baud.

A second timer is used to insure that the interrupt request is acknowledged within a reasonable time. Once the interrupt is acknowledged, a timer is set for 2-4 ms, depending upon the program segment, to time execution of the self-test segment. If the self-test does not execute within the allotted time, an error condition is generated.

Two LED indicators are used to provide redundant error signals. One LED is normally ON. If the self-test software detects an error, if the interrupt acknowledge is not generated fast enough, or if a self-test program segment does not execute within the 2-4 ms window, then this LED is turned OFF to indicate an error. This provides a fail-safe indicator since the most prominent failure mode for an LED is to "burn-out" which indicates an error. The second LED provides a "heart-beat" status signal. This LED is toggled on and off at a fixed rate by the self-test program. This provides a redundant indication of failure. This system thus detects failure in both

the primary system hardware and in the self-test hardware.

A small amount of additional hardware is required to provide "wraparound" data paths for testing I/O ports. The added hardware required constitutes a small percentage of the total system hardware and all added hardware is covered by the self-test mechanism except the final isolation buffer that prevents external devices from corrupting the self-test data.

Fault Simulation

One of the difficulties in developing self-tests for LSI systems is trying to rate the effectiveness of the software. The reason for this is that, presently, the only known way of testing the effectiveness of self-test software is to conduct "fault injection experiments". One can either run these experiments with a real hardware system or through simulation. Using a hardware system is not feasible because obtaining LSI devices with known internal defects is much more difficult than obtaining good devices. Simulation does provide an answer, but there are problems here also. LSI devices contain thousands of gates, thus using traditional gate level simulation techniques can present great difficulties. The biggest problem is that accurate gate level models of LSI devices are usually known only by the manufacturer and in most cases they are unwilling to divulge this information. Secondly, even given a gate level model of an LSI system, the simulations require too much host CPU time, i.e. money, when validating self-test software. The only solution to this problem is to develop a simulation model

at a higher level.

On this contract a simulation language known as GSP (General Simulation Program) was used to develop a "chip" level model of the 8080 system under consideration. In modeling at the chip level, internal chip micro-operations and interface signal timing are modeled without resorting to detailed description of the internal gate structure. This allows accurate simulation of an LSI system in an efficient manner.

Once the simulation model was developed, it was used to conduct fault injection experiments. In these experiments; faults were injected into the simulation model, and the execution of the self-test software was simulated. The fault types simulated were (1) incorrect device micro-operations (2) stuck faults and (3) timing faults. Because of the high probability of interconnect failures between chips (vs internal defects), 43% of the defects simulated were interconnect faults.

The simulation experiments allowed us to calculate a "figure of merit" for the self-test routines, i.e. approximately 80% of faults injected were detected by self-test mechanisms.

Hardware System Checkout

In this effort an 8080 laboratory system was constructed and all self-test routines were executed on it. The purpose of this activity was to verify that the test routines would operate properly in a real system and that they would, when finished with their execution, leave the system in a state compatible with the operational program. Building of the hardware system

also allowed us to verify that the limited amount of added self-test hardware functioned as anticipated. Finally, experience with the hardware system provided the test program writer and simulation model developers with useful information about its characteristics.

Table of Contents

1. INTRODUCTION

The advent of LSI technology has presented computer system designers with a powerful design capability. However, along with this increased capability has come the attendant problem of trying to verify that the LSI devices in a system are operating correctly. The large number of logic gates on an LSI chip can make this testing process difficult. On the other hand, the LSI chips in a system tend to exhibit a degree of functional independence from each other and usually contain powerful logic capabilities. These features make possible the implementation of self-test mechanisms in LSI systems.

The purpose of the research carried out under Air Force Contract F30602-80-C-0200 was to develop self-test software for microprocessor systems and verify the effectiveness of this software through fault simulation. This document is the final report for this research.

The research efforts carried out under this contract can be divided into three major areas. The body of the report will treat each area in detail but we briefly summarize them here:

(1) Development of Self Test Software. Under this effort, a system self-test scheme was first developed which identified the major functions to be performed by software and a limited amount of self-test hardware to achieve the testing goals. Next, within the system scheme, self-test routines were designed and written to test the particular microprocessor chip set

chosen for the research: an 8080 microprocessor, semiconductor random access memory (RAM), read only memory (ROM) and 8228, 8251, and 8255 support chips. Details of this research area are given in section 2 of the report.

(2) Fault Simulation. In this part of the research a simulation model was developed for the microprocessor system. The modeling was done using the General Simulation Program (GSP) previously developed at VPI. Once developed and checked out, the simulation model was used for fault simulation. Functional faults were injected into the model and the execution of the self-test routines was simulated in order to test their effectiveness in detecting faults. Details of this research are given in section 3 of the report.

(3) Check out of the Self Test Scheme on a Hardware System. In this effort an 8080 laboratory system was constructed and all self-test routines were executed on it. The purpose of this activity was to verify that the test routines would operate properly in a real system and that they would, when finished with their execution, leave the system in a state compatible with the operational program. Building of the hardware system also allowed us to verify that the limited amount of added self-test hardware functioned as anticipated. Finally, experience with the hardware system provided the test program writer and simulation model developers with useful information about the characteristics of the chips. Details of this research are given in report section 4.

SELF-TEST METHODOLOGY

The motivation for this study is the development of a library of self-test software for microprocessor-based control systems. At one end of the scale would be a very fast executing self-test program that would provide as much fault coverage as possible using a minimal amount of extra hardware and a small amount of memory. The added cost for the self-test would be minimal. At the other end of the scale would be a comprehensive self-test that would provide the maximum possible fault coverage. It is anticipated that this test would require considerable execution time and possibly costly extra hardware. In between these two extremes would be a variety of self-test mechanisms that would provide a wide range of fault coverages with intermediate execution times, memory requirements, and hardware costs. If such a library existed, microprocessor system designers could select the library program that best matched their particular requirements.

This report describes one program in the library in detail. The primary goal of this project was to develop a short test using minimal extra hardware that would achieve the highest possible fault coverage. It was intended that successful completion of this project would establish credibility for the library concept in addition to being directly applicable in its own right.

2.1 Self-Test Environment

Of all the proposed library programs, this one would minimize the impact on system cost, on power requirements, on system programming and on system reliability. The extra hardware required can be classified into three categories.

Indicators of system status. Two LED's provide status information. One LED will be normally ON to indicate that the system is operational. This LED will be turned off by the self-test program if it detects an error condition or by a hardware time-out if the self-test program fails to respond within an appropriate time period. The normally ON condition makes the LED fail-safe since the most likely failure mode of an LED is to "burn out". However, it is possible for the electronic driving circuit to fail in such a way as to make the LED remain permanently ON. Therefore a second LED will act as a "heartbeat" for the system. It will be toggled off and on at a fixed visible rate by the self-test program. The "heartbeat" indicator will detect catastrophic type failures where the program is executing in an erratic manner that provides interrupt acknowledge and false pass signals to the fixed LED. This active redundant indicator will also detect stuck-at failures in the driving circuit of the first LED. The heartbeat then protects against failures in the self-test hardware and catastrophic failure of the CPU. For the system to be operating properly, the fixed LED must remain ON and the "heartbeat" must oscillate at a fixed visible rate.

Function Timers

Two timers are employed. One initiates the self-test program by periodically generating a program interrupt. The other times the response to the interrupt and the execution time of the self-test program. If a system failure prevents the microprocessor from responding to the interrupt request or prevents the microprocessor from executing the self-test program in the allotted time interval, the second timer will time out and indicate a system failure by turning off the fixed LED.

Wraparound and Isolation: Hardware for I/O Ports

Since testing 8255 and 8251 peripheral I/O chips was an important part of the self-test objective, a means for reading back the data written to the ports must be provided. In addition, we must isolate the peripheral device during testing to prevent distortion of the testing data by the external devices and to prevent test data from being transmitted to the external devices (when necessary). A device signal is provided to the external device during testing to indicate CPU busy status. The details of this logic can be found in Section 4.

This hardware represents the minimal amount necessary to implement complete system testing. The only burden placed on external hardware is to observe the busy signal and hold input data until the processor is ready to accept it. A complete description of the hardware is provided in section 4.

2.2 Alternative Approaches for a Short Periodic Test

A primary objective is to make the periodic test transparent to the users. This has two major ramifications: first, the user's registers and stack must be preserved; second, interrupts must be disabled during the test since execution of an interrupt service routine could lead to a hardware timeout (i.e., the test, once started, must run to completion uninterrupted); third, the test must execute in as short a time as possible so that its execution would not be noticed by the controller program. However, in general, a shorter test routine results in less fault coverage. In order to reduce execution time, one tries to design test algorithms with many operations between verifications; but too few verifications may allow a fault to escape detection. Thus a tradeoff between speed and fault coverage seems inevitable.

Three different approaches were considered for the short periodic test. The first is a simple, straightforward, quick test. A second approach uses a longer, more thorough (but slower) test and partitions it into a set of short segments that are executed one by one at consecutive test times. A third approach combines the first two method . Again a series of tests is employed, but now a common 'core' test is executed each time. This core attempts to verify enough operations so that the housekeeping and dispatch functions required to decide which segment is to execute next can be expected to function reliably.

The primary advantage of the single comprehensive test is its simplicity. No overhead is required to schedule test segments. The major disadvantage is that fault coverage may not be adequate for a test that would execute in the available time window. The single test would certainly be preferable if the coverage is adequate. Our research indicates that such a test is practical for the 8080 CPU. However, we found that the execution time required for even a short memory test was excessive for this application. Since our objective is to test the whole system, we were forced to reject the single pass approach.

Experience with the test routine showed that the additional fault coverage gained by approach three was not worth the additional execution time. Therefore, we adopted approach two and partitioned the self-test into disjoint segments.

2.3 Constraints Imposed on System Design

A major objective of the self-test project was to provide an add-on package with minimal impact on the system design. This section describes the interaction required with the application system.

In the hardware area, the system must react to the self-test busy signal by holding input data until the self-test is completed. The required display isolation buffers must be provided. Three output port numbers must be reserved for

reporting the status and controlling the timers. About 1K bytes of ROM must be reserved for the self-test program.

In the software area, two vectored interrupts must be reserved for the self-test program. One is used to initiate the self-test execution and the other as an error exit. Sufficient additional stack depth must be provided to service the self-test program. The 8080 implementation requires 16 bytes of stack space to execute.

The application program must call the self-test initialization subroutine (INIT) whenever the system executes a cold start or system reset. See Section 2.6.1 for details. The programmer must also reserve 8 or 9 bytes of system RAM for the use of the self-test program.

In addition, the application program must operate with interrupts enabled most of the time. An extensive period with interrupts disabled would cause a hardware time-out. If the user has ROM to be included in the self-test, he must provide one checksum byte somewhere in every 128 byte block. See Section 2.6.4 for details.

2.4 Partitioning the System

Since we are doing functional testing, the most logical method to use in partitioning is based on function. In general, the CPU test, if possible, should be done in one segment. Memory will generally require many segments. As many I/O

devices as possible should be included in the remaining segments. Each segment should be approximately the same length in execution time.

For the 8080 system, we were able to test the 8080 CPU and the 8255 I/O port in the first segment. The ROM test was partitioned into 128 byte segments and the RAM test into 32 byte segments. The 8251 test required two segments. Execution time of each segment was approximately 4 milliseconds. The 8251 test execution time is practically clock independent since it depends mostly on the baud rate.

2.5 General Statements about the Short Test Algorithms

The self-test algorithms are designed to provide systemwide functional GO/NO-GO tests; they do not provide diagnostic information about what fault occurred. As such, they employ the 'start big' approach; i.e., they jump right into testing various functional elements rather than slowly building up from a small core. The tests are systemwide in that failures cannot be isolated to a single device; for example, a ROM or RAM fault could well cause the CPU test to fail. The algorithms were developed to cover single functional faults, although most multiple faults will also be detected.

Since there is virtually no failure mode data for microprocessors and their support chips, we don't know what faults are most likely to occur and cannot concentrate on testing for specific faults. Therefore the basic goal of the

self-test is to exercise all functional elements and data paths of the microprocessor system (excluding user peripherals, but including their I/O ports). Of course, exercising a faulty element or function does not guarantee detecting the fault; therefore we have made use of fault simulation results to determine how effective the self-tests are. (See Section 3.)

2.5.1 The CPU

The most complex component to test is the CPU itself; in general it consists of an ALU (arithmetic & logic unit), a (user) register array, other assorted registers and latches (accumulator(s), instruction register, etc.), some flags, instruction decoding logic (probably an internal ROM), timing and control circuitry, and the data paths connecting these elements.

The self-test exercises all functions of the ALU, thus testing the ALU control logic. The full adders that perform addition and subtraction are exercised by applying all input combinations to each adder (i.e. each bit position); since a full adder is a 3-input device (2 source operands and a carry in), 8 input combinations per adder are required. This tests for all detectable stuck-at faults and some shorts/opens in the adders. Similarly, the logic that performs AND, OR, and XOR is exercised by applying all four possible input combinations to each bit position. (That is, each bit position of each logic function is tested with inputs of 00, 01, 10, and 11.) Other functions (such as rotates) are tested in a similar manner, by applying all input combinations to each bit position for thorough coverage. A decimal (BCD) adjust function, if present, should be tested for no adjustment, adjustment due to flags, and adjustment due to a digit greater than nine.

The register array contains RAM (register memory), register select logic and multiplexers, and possibly increment/decrement, rotate, clear, and/or complement logic. The RAM must be tested for stuck-ats and shorts between adjacent bits; just which bits are adjacent depends on the RAM layout (which, in general, is unknown). Therefore the test requires loading (and verifying) at least three patterns into each register; the patterns apply both a 0 and a 1 to each bit position (stuck-at test), and the same and complement values to each pair of adjacent bits (test for shorts). These same patterns will also be used in testing the processor's other registers and latches and the data paths. This will be done by executing instructions that move the register data through the desired data paths to the other registers. When the internal layout of register memory and data paths is unknown (as usual), the most logical assumption is that logically adjacent bits are physically adjacent (this is certainly true for at least some of the registers, latches, and/or data paths). Under this assumption, some suitable patterns are (in hexadecimal): 00, 55, & AA; 33, 66, & CC; D9, 6C, & 36. The patterns are distributed in the registers so that register select faults may also be detected, assuming either logical ORing or logical ANDing (as appropriate for the technology employed) of register contents for a multiple select fault on reading. Thus the tests employ different patterns in the different user registers so that R1 OR R2 (or R1 AND R2, as

appropriate) equals neither R1 nor R2. This allows both erroneous select and multiple select faults to be detected. A fault resulting in no register being selected will be easily detected when using these patterns. The contents of all user registers are verified near the end of the test, so that an erroneous write or multiple write fault can be detected.

Increment/decrement logic would probably be centralized in one unit within the register array (as in the 8080); however, the exact gate implementation is most likely unknown. For this reason, the self-test applies only a few basic test vectors, such as incrementing -1 (all ones) and decrementing zero, which tests carry/borrow propagation through every bit. Carry/borrows through no bits (incrementing an even number and decrementing an odd number) and through an intermediate number of bits are also tested. It should be noted that this logic may also be used to increment the program counter (PC) and/or stack pointer (SP) (as is the case for the 8080), which provides some additional increment testing. Other possible register functions, such as clear or complement, are most likely built into each register if they are present at all. These are easily tested, but testing and verifying each function of each register will be time consuming, so that a tradeoff may be necessary.

Any microprocessor will contain, in addition to the register array, some internal registers and latches and possibly special accumulator registers. These registers may be tested

for stuck-ats and shorts between adjacent bits by using the same patterns discussed above, assuming they can be loaded by software (either directly or indirectly). The instruction register (IR) is of interest in that it is loaded, not with data, but with instruction opcodes; this means that a fault will result in the execution of some erroneous instructions, possibly causing loss of program control. This cannot be avoided; however, as long as the GO signal is not generated, a hardware timeout will provide the needed NO-GO result, so that the fault will be detected. Accumulator registers are also of interest in that they inevitably have special functions, such as complement and/or increment. The self-test must exercise each function of each accumulator with sufficient patterns to ensure the detection of any (detectable) stuck bits. Two complementary patterns will suffice to test complement logic; but note that two successive complements (with no verification in between) results in a poor test, since the correct final result will be obtained if each complement does nothing at all. The program counter (PC) and address latches/buffers are of special interest as well, because, for meaningful results, only valid memory and I/O addresses can be applied. However, these registers/latches are used for all memory reads and writes, including instruction fetches, and sometimes for I/O, so that enough patterns will be applied during the course of the test to detect most stuck-ats/shorts. Increment and decrement functions are tested as described above.

The condition flags make up another CPU element to test. The self-test simply applies and verifies (by conditional jump, add with carry, ...) both one and zero (true and false) for each flag. Shorts between the flag flip flops are conceivable, but cannot be efficiently tested for without knowing the internal layout. The logic driving the flags is exercised throughout the test by numerous arithmetic and logical instructions; but obviously the test can only verify the flags at strategic points (optimally where another function is also verified) to minimize execution time.

The instruction decoding and machine cycle encoding unit of the CPU is the most difficult component to test. To simplify LSI implementation, the decoder most likely employs ROM, the exact nature of which is unknown. Thus a fault in the ROM could conceivably be manifested for only a single instruction. Also, note that faults in instruction decoding, like faults in the instruction register, may cause erratic behavior or even loss of program control (hopefully resulting in a hardware timeout). Since the self-test cannot execute and verify every instruction in the short time available, it simply covers as many classes of instructions and as many micro-operations as possible. For example, only a few register to register moves are tested, instead of trying to move each register to each other register. All types of addressing and all types of parameters (registers, immediate data of all lengths, immediate addresses) are employed

through the course of the test. However some instructions (such as halt) cannot be self-tested without special hardware. The most commonly used instructions (such as adds, compares, branches, etc.) are tested most thoroughly. Conditional transfers (jumps, calls, and returns), for example, are tested for both transfer and no-transfer; note that this overlaps with flag testing and almost everything else, since the conditional transfers are used for verifications of other functions. This overlap is typical of the 'start big' approach, several functions being tested together.

The self-test is designed to exercise all possible micro-operations, thus testing some decoding and most of the machine cycle encoding. The micro-operations are determined from the processor's User's Manual based on the data paths, CPU elements, and functions employed by each instruction. All registers of the register array are considered equivalent since they use identical data paths external to the array (only register selection differs, and this was considered previously). The other registers (IR, accumulator, ...) are treated independently since their data paths differ. Conditional type instructions, which employ different micro-operations under different conditions, are considered to cover only those micro-operations common to both conditions. This prevents the illusion of covering micro-ops that may in fact not be performed during instruction execution. Software has been developed to

determine the micro-operation coverage of a given test algorithm, and to find a minimal set of instructions that cover any set of micro-operations. Fault simulation will be required to determine actual fault coverage, since exercising a faulty micro-op does not guarantee detecting the fault.

Since the micro-operations are derived based in part on the data paths they use, exercising all micro-operations also serves to exercise all data paths. Most of the data paths are exercised with the register test patterns previously described, thus testing for stuck-ats and shorts between adjacent bits. As mentioned earlier, this is performed by executing instructions that move data from the registers over the desired data paths (thus providing more overlap). Some data paths, however, cannot be directly exercised with these patterns. For example, consider the data paths leading to the instruction register and those leading to an address buffer. For meaningful results, only valid opcodes and valid addresses, respectively, can be applied to these data paths. However, during the course of the test, enough opcodes will pass into the instruction register to effectively test for any stuck-ats/shorts in IR or its data paths. Also, the RAM, ROM, and I/O tests will access all valid addresses so that most stuck-ats/shorts in the address circuitry can be detected. Thus although complete stuck-at/short coverage is not in general possible for all data paths, the self-test can still verify that valid data will not be distorted. What happens to invalid addresses is not important anyway.

The final part of the CPU is its timing and control circuitry; this cannot be directly self-tested. However, by exercising all micro-operations, the self-test will also exercise much of this control circuitry. Further, the hardware timeout feature guards against some possible major control faults that are totally transparent to the software, such as generation of numerous unneeded hold or wait states. Also, interrupt control is verified since the test is initiated by interrupt. Still, some control signals cannot be self-tested without considerable extra hardware because of their nature (e.g., the HOLD/HOLD Acknowledge circuitry of the 8080 is used for DMA by external devices and thus is completely transparent to software). This is a limitation that cannot be avoided without the addition of considerable extra hardware.

2.5.2 ROM

ROMs are the easiest system component to test. A checksum is stored in the ROM itself to produce a known result when the contents of all (or a certain piece of) ROM are summed. Several different methods of forming this sum have been considered. The first uses a modulo 2 sum, in effect an exclusive OR; each column (bit position) is independent of the others (there are no carries). However, two faults in the same column would go undetected; hence this method was rejected. The second method uses a modulo 256 sum, namely the ADD instruction; carries out of the most significant bit (MSB, bit 7) are lost. But two

faults in the MSB column would again go undetected, so this method was enhanced to form the third approach: the carry out of the MSB (from the ADD) is added back to the least significant bit (LSB, bit 0) of the sum; thus nothing is lost. Now to escape detection, the two faults must not only be in the same column, but they must also be complementary. That is, one must be a 0 turned 1, and the other a 1 turned 0. But due to the physical nature of a ROM, PROM, or EPROM, this is extremely unlikely; faults will normally occur in only one direction. Thus 0's may turn to 1's or 1's to 0's, but not both. Under this assumption, the third test method will prove quite effective.

Another consideration for the ROM test is how many checksums to use. The test algorithm is passed the start address of the ROM to test to make it address independent, but this also allows the ROM to be tested in pieces of any size desired, so long as each piece contains a checksum byte (to give the required sum); the checksum may be anywhere in the block of ROM under test. The ROM test could thus be broken up into a series of tests, so that periodic tests can execute in the short time available.

The final consideration is what number to use as the final known result of the sum. A sum to -1 (FF hex) was considered until we noticed that a 'dead' ROM (permanently deselected) would pass the test. (Since the bus would float when the ROM

should be active, FF hex would be read as the contents of each byte, resulting in the net sum of FF hex.) A sum to zero was similarly rejected in favor of a sum to AA hex, since the latter is a 'checkerboard' pattern (10101010 binary).

The test algorithm thus tests the ability of the ROM to access each location which verifies the address decoders, select logic, output drivers, and the ROM contents.

2.5.3 RAM

Many techniques for RAM testing have been developed over the past decade, each with its own advantages and disadvantages. But all thorough RAM tests share a common drawback: they take forever. Faster, specialized tests could be developed were the internal cell layout of the RAM known, since then a cell's true neighbors would be known. This would permit minimal tests of the decoders (access each row and column of the RAM only once) and allow true nearest neighbor (disturb) tests. But cell layouts vary from manufacturer to manufacturer and are almost never made available to users.

The Moving Inversions (MOVI) test technique is one example of a thorough test, and it is much faster than Walking or Galloping tests (1,2). A memory location is first read to verify it contains the previous pattern; then the current pattern is written and immediately read back for verification (to try and detect write recovery faults). This continues until the memory is filled with the current pattern. The patterns

employed are (in hexadecimal): 00, 01, 03, 07, ..., 7F, FF, FE, FC, F8, ..., 80, and back to 00; thus one bit is inverted each time. MOVI sequences through the RAM first moving forward (up) and then backward (down). In addition, MOVI steps through memory using each address bit as the LSB; that is, MOVI first moves from location N to N+1 (N-1 on down cycle), then from N to N+2 (N-2) on the next pass, then N to N+4 (N-4), etc. Thus all fundamental address transitions are tested (i.e. a change by a power of two, both forwards and backwards); this provides some testing for cell, row, and column disturb faults (despite the unknown layout). In order to test all of these basic address transitions, the test program tests all of (contiguous) RAM at once (testing smaller pieces would not test all basic transitions). Refer to reference [1] for complete details on MOVI. Note that MOVI does not test refresh for dynamic RAMs.

MOVI is a fairly good test for address decoder switching speed, cell, row, and column disturb faults, data sensitivity, and write recovery faults [2]. It is a very good test of address uniqueness , and a good general test of both functional and dynamic behavior [1].

The MOVI test requires 12 x B x n x N memory cycles, where B = number of bits per word, n = number of address bits, and N = 2**n = number of RAM locations. Naturally, a self-test program is much slower due to all the overhead it must perform. The MOVI test implemented for the 8080 requires about 20 seconds per

1K of RAM. Also, this test is, of necessity, destructive; that is, original RAM contents are lost. Thus MOVI is not at all suitable for a short, periodic test.

Therefore, two nondestructive tests were developed for the short periodic test, but they are, of necessity, not as thorough. Both algorithms make use of 'random' patterns, generated using a feedback shift register technique. This employs an irreducible polynomial of degree 8 to generate a sequence of 255 test patterns. The programs generate the next pattern by shifting the current pattern left and exclusive-ORing bits 2, 3, and 4 with the carry out from the MSB (bit 7). This carry out is also shifted into the LSB of the new pattern. Starting with 55 hex, the sequence is: 55, AA, 49, 92, 39, 72, E4, etc. Thus the desired 'randomness' is achieved. All 8-bit patterns except 00 will be generated before the sequence repeats. Note that test algorithms using this technique require one byte of RAM in which to store the current pattern. Since these random patterns change from test run to test run, numerous different patterns will be applied (over time), which provides some likelihood of detecting pattern sensitive faults. The algorithms are passed the start location of the RAM to test, so that RAM may be tested in segments.

The first algorithm tests RAM one location at a time, thus not testing address decoding (uniqueness) at all. It works as follows:

(1) Read & save a RAM byte

(2) Write/verify complement of original contents

(3) Write/verify 'random' pattern

(4) Write/verify complement of 'random' pattern

(5) Restore/verify original contents

The second algorithm tests groups of two successive RAM locations, thus providing a minimal test for address uniqueness. It works as follows, where M and M+1 are the two locations under test:

(1) Read & save both M & M+1

(2) Write 'random' pattern to M+1; verify both M & M+1

(3) Write complement of 'random' pattern to M;
verify both M & M+1

(4) Restore & verify both M & M+1

Location M+1 then becomes the next M, and the test continues.

Both algorithms immediately follow each memory write with a read from the same location in an effort to detect write recovery faults. However, the algorithms test primarily for data errors in a RAM cell and the ability to read and write, with at best minimal testing of other RAM faults (such as cell, row, and column disturb faults, address uniqueness, and address decoder switching speed). Note that although the sequence of

'random' patterns does not include zero, a zero pattern will eventually be written since the complement of the 'random' pattern is also used.

2.5.4 I/O Ports

As mentioned in Section 2.1, self-testing I/O ports requires external wraparound hardware. Consider a serial I/O port chip (such as an 8251); three tri-state buffers can provide wraparound. Two are normally enabled to allow passage of user I/O data; the third, normally disabled, connects the serial output to the serial input. Then in test mode the first two buffers are disabled (tri-stated), isolating the user's external serial device, and the third buffer is enabled for wraparound. Thus serial data output will be received by the same chip's serial input, simultaneously exercising both transmit and receive logic if the serial chip is full duplex. Parallel I/O chips are tested similarly, using a single I/O chip if it has multiple I/O ports (as does the 8255) or using one output chip and one input chip if not (thus testing both at once).

Serial I/O chips (UARTs/USARTs) and some parallel I/O chips handshake with the CPU by means of status bits. In general, the processor must wait for proper status before performing input or output. This means that a fault in the I/O chip could leave the CPU in an infinite wait; this, however, will result in a hardware timeout so that the NO-GO test result will still be generated.

Some of the more recent LSI I/O chips (like the 8251 & 8255) are software programmable and can function in more than one operational mode. Unfortunately, the chip's current mode cannot, in general, be read back from the chip. This presents a severe problem to the self-test program. If it is to be nondestructive (as the periodic test must be), the chip's current mode must be restored before returning control to the application program. But, since the current mode cannot be read from hardware, the applications program would have to be constrained to store current mode data in RAM accessible to the self-test routine. Since most applications never change the mode of the I/O port, the best solution seems to be testing the chips thoroughly (in more than one mode) only after system RESET (or upon user request), and performing configuration dependent nondestructive tests periodically.

The serial port test algorithm consists of three parts: a transmit/receive test; a break send/receive test; and an overrun error detection test. Tests for framing or parity errors would require external hardware, and are therefore not performed. The transmit/receive test simply outputs test patterns and verifies that the same pattern is received (via the wraparound) with no errors. (Note that this also serves to test the wraparound.) The patterns used are (hexadecimal) 00, 55, and AA -- the same patterns that were used in the CPU register test. This tests

for stuck-ats or shorts between adjacent bits in the parallel data paths leading to the I/O port and in the chip's data registers, as well as testing the serial send/receive circuitry and portions of the status logic. The break test works in a similar fashion: a send break command is issued and then the CPU waits for the break detect (received) status bit to come active. In the overrun error test, the CPU outputs one character and waits until the UART has received it; then, without reading this character, the CPU outputs a second character and waits for it to be transmitted and received. Then the processor verifies that the overrun error status bit is set and that the second character can be read correctly (the first is lost). Thus most of the status and I/O circuitry has been tested. A more thorough destructive test would repeat the test for different baud rates, character lengths, and/or other programmable features.

A parallel I/O port test is much more chip dependent; simple I/O can be tested by applying the same patterns used in CPU register testing (00, 55, & AA hex will do). This tests the data paths, data registers (if any), output drivers, wraparound data paths, and input circuitry for stuck-at's or shorts between adjacent bits. (Note that strobe driven input may require some special hardware to generate the strobe signal.) Special functions (like the 8255's Port C single bit set/reset function)

are then tested for correct functional operation, provided they do not alter the chip's current operating mode (so that the test remains nondestructive).

2.6 Implementation of the Algorithm for the 8080 System

The preceding section considered test algorithms for microprocessor systems in general. That methodology has been applied to a system consisting of an 8080 CPU, RAM, ROM, 8251 serial I/O port, and 8255 parallel I/O port. This section of the report will discuss the self-tests developed for this 8080 system. It should be noted that the 8228 (system controller) and 8224 (clock generator) are considered part of the CPU element, along with the 8080 itself. This is reasonable since these chips are usually located on the same printed circuit board as the CPU.

2.6.1 Preliminary Considerations

This program is written in such a way as to be applicable to as wide a variety of user environments as possible. Also, the user responsibilities are reduced to a minimum. However, it is never possible to be completely transparent. The program must know certain parameters about the actual system. The user must provide these constants as shown in the configuration dependent assembly language equate statements on page A.1.

The first item that must be specified is the starting address of a 1.25K (1280) byte segment of ROM to be used for the self-test program. This is specified by defining the system parameter START as shown on page A.1.

Other items to be specified include the address ranges of RAM and ROM, the memory mapped addresses of the 8255 and 8251 data ports and the three memory mapped I/O addresses necessary to configure the wraparound logic and the self-test timers. (See Appendix A, page A.1.)

Also, 8 contiguous bytes of system RAM must be reserved for use by the self-test program. The user must specify the address of the first of these eight bytes by defining the value of TSTAD (page A.1).

The ROM and RAM test segment sizes may be changed by altering the ROMSS and RAMSS parameters. This segment size must always divide the ROM or RAM into an integral number of segments, and hence should usually be a power of 2. Note that changing the ROM segment size (ROMSS) will change the number of checksums required for the self-test program listed in Appendix A. Increasing ROMSS results in fewer checksums, while decreasing ROMSS increases the number of checksums and may require adding JMP's around the checksum byte (if the checksum must be placed within a block of program code). Naturally, increasing a segment size increases the time per test pass, while decreasing either segment size makes for faster test passes.

In addition, any user ROM to be included in the self-test must include a checksum byte somewhere within each 128 byte

block. The checksum is chosen so that the modulo 256 sum (with end around carry) of all 128 bytes in the block is AA(Hex). A jump around this checksum may also be needed, so that 4 bytes out of each 128 bytes of user ROM must be dedicated to the self-test function. Only one byte is required if there is an unconditional branch in the block, since the checksum may then be placed immediately after this transfer. The 128 byte block size was chosen so that each ROM test segment will execute in approximately the same amount of time as the CPU/8255 test segment. The block size is small because the CPU self-test was made as short as possible.

The initialization routine is shown on page A.23. This routine (INIT) must be called by the applications program when processing a reset. This routine initializes all of the self-test variables in the self-test portion of RAM and intializes the programmable timers.

The entry point (START) must be reached from one of the 8 vectored interrupt locations selected by the system designer. All registers and flags from the main program are saved on the main program stack. The entry routine also initializes the timeout counter and reads the address of the next scheduled test segment from TSTAD. At the end of each routine, TSTAD is loaded with the address of the next segment to follow. Successive interrupts will then execute successive test segments. Variable

values that need to be retained are stored in the 8 reserved RAM locations.

2.6.2 Reporting Status

If the self-test discovers an error, the ERR routine (shown on page A.2) is executed. In our implementation, the LED that normally remains on is turned OFF to indicate an error condition. The processor is then halted. This routine was placed at the beginning of the self-test program to increase the probability of the HLT being executed when the CPU is bad. Three successive HLT statements take care of possible byte skew due to software failure.

If the current test segment executes properly, the GOEXIT routine is executed. This routine initializes the main interrupt counter to the desired interval and restarts the timeout counter to insure the next interrupt is processed. If the self-test program did not execute properly in the allotted time, then the timeout counter would turn off the error LED. This would fail only if the GOEXIT routine were accidentally executed properly by a faulty processor. To minimize this probability three HLT instructions precede this routine. Also, the software error routine immediately precedes this routine. We therefore minimize the probability of a faulty GO signal as much as possible.

2.6.3 The CPU Test

The 8080 CPU test is shown in A.4--A.11. The 8080 CPU self-test exercises all ALU functions, which overlaps with exercising all possible micro-operations. These ALU functions are: ADD and SUBtract both with and without carry/borrow (and for carry/borrow of both 0 and 1); XOR, OR, AND; rotates left and right both through the carry flag (RAL, RAR) and without the carry (RLC, RRC); decimal adjust (DAA); and no-operation (after INR/DCR). Since arithmetic is very common, the full adders used for addition and subtraction are tested with all input combinations for each adder (8 combinations for each of the 8 adders) during the course of the test. Note that the compare instructions are subtracts as far as the ALU is concerned. A self-contained subtest applies all input combinations to each of the ALU's XOR, OR, and AND logic function gates (4 combinations for each of the 8 gates for each function). (Users not concerned with the logic functions could omit this subtest.) The ALU's rotates are tested for functionality by performing each of the four different rotates once; this seems like a minimal test, but since the exact implementation of these functions is unknown, what is optimal? Also, the rotates are not one of the most commonly used ALU functions, so a longer test seems unjustified. The (BCD) decimal adjust function is tested for four cases: no-adjustment, adjustment due to the

carry flag CY=1, adjustment due to the auxiliary carry flag AC=1, and, finally, for adjustment due to a digit greater than nine. This test also verifies that the AC flag can be both one and zero (providing overlap with flag testing).

The 8080 register array consists of the user registers B&C, D&E, and H&L, plus the stack pointer SP, program counter PC, the W&Z internal registers, a 16-bit increment/decrement circuit, and a 16-bit address latch (plus multiplexers and demultiplexers for register select). Registers B,C,D,E,H,L, and SP are tested with the patterns described in Section 2.7 of this report. It is important to note that the 8080 XCHG instruction (exchange D&E with H&L) operates by switching the internal addressing of the DE and HL register pairs, and does not actually move any data at all [3]. Thus to test these registers' RAM cells, one must be wary of XCHG's, or the test coverage will not be what it seems. The register test patterns used for SP (hex AAAA, 5555, and 0000/FFFF) are also (while testing SP) applied to the array's address latch and inc/decrement circuit. This should detect any stuck bits or shorts between adjacent bits in these elements. The address buffer (which drives the address bus from the address latch), however, cannot be tested with these patterns, since (for meaningful results) only valid memory (and I/O) addresses can be applied (without adding more test hardware). The same restriction applies to the program counter

PC. In addition, the W&Z internal registers are not tested with these patterns either, despite the fact that they can be loaded via XTHL (exchange top of stack with H&L). This is because W&Z are used for all branching (jumps, calls, returns, and RSTs) except PCHL, which is unconditional. Thus though a stuck-at/short in W&Z could be detected, the conditional branch that must be used for verification would probably jump to the wrong address. (Also, the test would require a fair amount of time as XTHL is the slowest 8080 instruction.) Note that W&Z are implicitly tested to some extent since the test includes an XTHL and numerous branches; PC is similarly tested as it steps through the self-test programs.

The increment/decrement circuit is tested for worst case transitions (decrementing 0 and incrementing -1), for carry/borrow to/from the high order register of the pair, and for no carry/borrow propagation. Also, some additional testing is performed while testing stack operations and by the incrementing of PC after each fetch. Finally, register select logic is tested by using different and logically distinct patterns in the various user registers, as explained in Section 2.3.

The 8080 also contains several other 8-bit registers: the accumulator A, accumulator latch ACT, a TMP register, instruction register IR, and a data buffer/latch (driving the

data bus). The first three, A, ACT, and TMP, are used for almost all ALU operations and are tested (for stuck-ats and shorts between adjacent bits) with the register test patterns (described in Section 2.3) during the course of various arithmetic instructions (overlapped with ALU testing). Sufficient opcodes are applied to IR to verify that it too is free of stuck-ats and shorts between bits. The data bus buffer/latch is also tested for these faults by the opcodes and data bytes read from memory (latch) and by the patterns output by the CPU for testing memory and stack writes (buffer). The 8228 bidirectional data paths are also tested by these data transfers. The 8-bit increment/decrement function is tested in the same manner as the 16-bit inc/decrement circuit, with worst case transitions (incrementing -1 and decrementing 0) and various others. Finally, the accumulator's complement function (CMA) is also tested for any stuck-ats or shorts (between adjacent bits) while generating patterns for other uses (overlap).

The flags (CY, AC, even Parity, Sign, & Zero) are tested by verifying each as true (1) and false (0). This is simultaneous with the testing of conditional jumps: JZ, JNZ, JM, JP, JC, and JNC are all verified for both branch and no-branch. The remaining two, JPE and JPO, are tested for no-branch only since the Parity flag is not commonly used and the time seemed better

spent elsewhere. The AC flag is tested while testing the decimal adjust function (DAA). The CY and Z flags are considered most important and as such are tested most thoroughly. CY is the easiest to test, since it is used by adds with carry, subtracts with borrow, certain rotates, and decimal adjust. Some additional testing of all flags occurs while testing the PUSH/POP PSW instructions.

The 8080 instruction decoding, implemented by ROM, is tested by exercising all possible classes of instructions and all possible micro-operations. Also, the test includes instructions with zero, one, and two bytes of immediate data, and instructions with an immediate address (such as LDA). Likewise, all forms of addressing, direct, register, register indirect, and immediate, are exercised. Despite the fact that a decoder ROM fault may show up for only a single instruction, the self-test does not try to execute all instructions. Rather the test exercises classes of instructions, where, for example, ADD r, ADD M, and ADI xxx form a class, DCX rp form another, and XCHG is a class of its own. All classes of instructions are exercised except for DI, HLT, IN, and OUT (since extra hardware would be required to test these), and conditional calls/returns. IN and OUT would be tested in I/O port tests and/or status reporting if memory mapped I/O is not employed. A test containing conditional calls and returns has been implemented,

with all but the parity conditions tested for both call (return) and no-call (no-return). However, the extra time required is considerable (about 300 clock cycles) so that user priority will determine whether or not to use this version of the self-test. More importantly, the self-test exercises all micro-operations except those covered only by DI, HLT, IN, and OUT; this exercises most of the decoding and control logic of the CPU. Refer to Appendix G for a list of the 8080 micro-operations.

The data paths connecting the various elements of the CPU are tested (for stuck-ats/shorts) "in the process of testing the elements themselves. Also, as was mentioned in Section 2.2, the micro-operations were developed with the data paths in mind; so all data paths are exercised to some degree. The weakest point is the path to the address buffer which, as stated before, is restricted to valid memory and I/O addresses (for meaningful results).

Tests were combined where possible. For example, on page A.4, the portion of the test between OK0 and OK1 executes the DCX, ADD, and ACI to set the carry and zero flags and EVEN parity. The JNZ, JNC, JPO instructions verify the correct operation of instructions, flags, and conditional jumps. Note that the JZ to OK1 is followed by a JMP error in case the JZ fails. The RST ERX is yet another branching mechanism that may work if the JMP fails. Finally, the HLT should hang up the

program if no transfer at all is executed. The RST ERX transfers control to one of 8 vectors in low memory, where a copy of the ERR routine is located.

Similarly, segments from OK1 to OK5 verify other combinations of operations. The segment OK5 verifies DAA, DAD, INX, and other register and ALU operations. As a final check, the contents of all registers are verified by adding them together modulo 256 with end around carri and checoing for the expected sum. This tests for multiple register selects.

The other instruction tests are documented in appendix A, pages A.2 to A.10.

2.6.4 ROM Test Program

A ROM test using checksums formed by modulo 256 addition with carries added back to the LSB (as described in Section 2.5.2) has been implemented for the 8080 processor. The algorithm is passed the start address of the block of ROM to test and forms a sum of AA hex for a GO pass. The test routine object code occupies only 64 bytes of memory and runs in about 45 N clock cycles, where N is the number of bytes of ROM to test. Thus 128 bytes of ROM can be tested in a single pass of just under 3 milliseconds (with a 2 MHz clock). See page A.16.

A sample program, written in Microsoft BASIC, for calculating the checksum needed for each ROM segment is listed in Appendix H. The program requests the segment size and

desired final sum for flexibility; these should be 128 and 170 decimal (AA Hex), respectively, to match the self-test program listed in Appendix A. The CHKSUM program accepts an INTEL ASCII format object file as input. This object file should have a zero byte where each checksum byte will be placed; the zeroes must then be changed to the appropriate checksums, and the file reassembled prior to burning the ROM.

2.6.5 RAM Test Program

A MOVI RAM test and both quick, nondestructive RAM tests described in Section 2.5.3 have been coded for the 8080. The thorough MOVI (Appendix F) test requires almost 20 seconds per 1K of RAM (for a 2 MHz clock) and, as stated earlier, tests all of contiguous RAM at once. The first quick test, which tests RAM location by location (A.17), requires about 150N clock cycles to test N locations, or about 75 milliseconds per 1K. The 32 byte segment used in the program (A.17) requires 2.3 milliseconds to execute. The second quick test, which tests two successive RAM bytes at a time, requires about 235N clock cycles for N locations, or about 120 milliseconds for 1K. Both of these tests are passed the start address of the RAM segment to be tested to allow segment testing. Each of the nondestructive test routines requires less than 100 bytes of object code, while MOVI requires about 200 bytes. The first nondestructive test is included in the self-test program listed in Appendix A; it was chosen for its high speed.

2.6.6 I/O Ports Test Program

The 8080 has two software programmable I/O port chips: the 8251 USART for serial I/O, and the 8255 for parallel I/O. Nondestructive tests have been coded for both chips using memory mapped I/O so that the same test routine may be used to test several different 8251's (or several 8255's). However, the

routines may be easily converted to discrete I/O (using IN and OUT).

The asynchronous mode of the 8251 is tested and has been modelled for simulation. As described in Section 2.5.4, the test consists of an I/O test, a break send/detect test, and an overrun error detect test. Unfortunately, not all versions of the 8251 have the break detect capability [4,5]; the break is transmitted and wrapped around to the serial input, but a framing error is detected instead of the break. For these chips the break test has become a break send/framing error detect test. The 8251 test routine requires about 128 bytes of object code; its execution time depends upon the character length and baud rate selected. With one stop bit, 8 bit characters, no parity, and at 9600 baud, the 8251 test requires just under 8 milliseconds. Therefore, it was divided into two parts requiring about 4 ms each. (See p A.19)

An 8255 test has been coded to test Mode 0 operation (basic I/O with no strobes/handshaking)(See page A.11). Although the 8255 remains in Mode 0 throughout, it changes the port configurations (i.e., changes which ports are inputs and which are outputs). However, the test restores the 8255 to its original configuration, which must be known in advance. Before the test commences, all wraparound and isolation buffers are tri-stated (isolating the external device); then a pattern is

written to each port. This is a no-op if the port is defined as an input, while if it is an output, the pattern is latched and can be read back by reading that port. Each port is then read back; if it was an input, FF hex (all ones) is read (since the lines driving the inputs are tri-stated). If the port was an output, the pattern is read back; note that if neither value is read back, there is a fault and a NO-GO result is generated.

The first part of the test routine tests the 8255's three I/O ports (A, B & C) and the data paths involved for stuck bits or shorts between bits as well as testing the 8255's basic functionality. The test defines one port as output and the other two as input, and then outputs test patterns and verifies that they have been read back correctly through the two input ports. Each port has a turn as output port, as shown below:

Output port	Input ports	Patterns used (hex)
A	B, C	55, AA, 00
B	A, C	CC, 66, 33
C	A, B	D9, 6C, 36

The second part of the test checks the Port C single bit set/reset function. Each bit is set (verified) and reset; the test then verifies that setting a set bit and resetting a zero bit have no effect. The patterns in Port C are read back for verification through Port B. The total test routine requires approximately 300 bytes of object code and takes between 1700

and 1800 clock cycles to execute (0.85 to 0.90 milliseconds for a 2 MHz clock), depending upon the original configuration. Therefore, this test was combined with the CPU test to make a 2 ms segment.

A self-test methodology has been described for microprocessor systems in general and specific algorithms for an 8080 system have been discussed. The methodology has been developed under the constraints of minimum additional hardware, minimum impact on system users, and has been tailored to quick, periodic, transparent tests. Some elements/functions are untestable under these constraints (as was Halt), or can be given only a partial test (as for RAM). The test procedures proposed follow the 'start-big' approach and so attempt to overlap testing one element/function with testing others. Overlap is quite important under the above constraints in order to maximize fault coverage while minimizing testing time. Unfortunately, this overlap makes automatic generation of tests most difficult, but research toward this end is desirable.

2.7 Organization of Self-Test Segments

The overall system self-test is organized as a sequence of short periodic test segments. With a 2 MHz processor clock and fast memory that does not require WAIT states, the execution time of each segment is as follows:

where m = (ROM memory size in bytes)/128
n = (RAM memory size in bytes)/32

Segment (1) Tests CPU and 8255 (2 ms)

```
Segment (2) Tests first 128 bytes of ROM (3 ms)
Segment (3) Tests next 128 bytes of ROM (3ms)
                    .
                    .
                    .
Segment (m+1) Tests last 128 bytes of ROM (3ms)
Segment (m+2) Tests first 32 bytes of RAM (2.5ms)
Segment (m+3) Tests next 32 bytes of RAM (2.5ms)
                    .
                    .
                    .
Segment (m+n+1) Tests last 32 bytes of RAM (2.5ms)
Segment (m+n+2) Tests 8251              (4ms at 9600 baud)
Segment (m+n+3) Tests 8251              (4ms at 9600 baud)
```

The total test time is 3m+2.5n+10 milliseconds. For example, for a system with 16K of RAM and 2K of ROM, the total test time would be 1338 ms. The CPU test uses only 2ms of the total time. The 8251 uses 4 ms, the ROM requires 48 ms and the RAM requires the rest (1280 ms). The test is executed in *m+n+3* = 531 sequential segments.

2.8 Hardcore Assumptions

The self-test, as mentioned previously, cannot test all CPU elements/functions completely without special extra hardware. Those elements not tested are: the HLT instruction; the DI (disable interrupts) instruction; the IN and OUT instructions (since memory mapped I/0 is employed); conditional calls and returns (an enhanced CPU test verifying these calls/returns has been coded, but was not simulated); the program counter (PC) and address buffer (since only valid addresses may be used for meaningful results); and the WZ register pair. Faults in the

latter three elements may be detected by a hardware timeout should the program lose control due to faulty addresses. Finally, the 8080's timing and control circuitry cannot be self-tested directly (without special hardware).

3. FAULT SIMULATION

One of the difficulties in developing self-tests for LSI systems is trying to rate the effectiveness of the software. When one reads articles on self test development for LSI systems, the authors are usually mute on this point or make vague statements such as "the test routines were shown to be effective." The reason for this is that, presently, the only known way of testing the effectiveness of self-test software is to conduct "fault injection experiments." One can either run these experiments with a real hardware system or through simulation. Using a hardware system is not feasible because obtaining LSI devices with known internal defects is much more difficult than obtaining good devices. Simulation does provide an answer, but there are problems here also. LSI devices contain thousands of gates, thus using traditional gate level simulation techniques can present great difficulties. The biggest problem is that accurate gate level models of LSI devices are usually known only by the manufacturer and in most cases they are unwilling to divulge this information. Secondly, even given a gate level model of an LSI system, the simulations require too much host CPU time, i.e. money, when validating self-test software. The only solution to this problem is to develop a simulation model at a higher level. During the past several years at VPI, we've developed an approach to higher

level simulation known as chip level simulation. In chip level simulation, the internal microperations of a device and the timing characteristics of the signals at its interface pins are simulated. This is done without modeling the detailed internal gate structure of the chip.

The simulation language that we employ is called GSP (General Simulation Program). It was developed under previous U. S. Navy research contracts. It has been used on this contract and we are also using it to do fault modeling for the NASA-Langley Research Center.

We have found GSP to be a very effective tool for the fault modeling required by this contract. As will be detailed, using GSP, we were successful in modeling the microprocessor system and conducting the required fault injection experiments.

3.1. A Description of the General Simulation Program (GSP)

The General Simulation Program (GSP) is a general purpose simulation program designed specifically to simulate LSI devices at the chip level, i.e. internal device micro-operations and detailed interface signal timing are simulated. The program is written in FORTRAN to insure portability. Presently the system runs in either a batch (MVS) or interactive (CMS) mode on an IBM 3032 Processor. An optimizing compiler (G1-HX(2)) is used to speed up the simulation.

When utilizing the GSP system one goes through the following three phases:

Phase 1 - Chip Description. The user models the device at the chip level (an example of this is given below) and codes its description using the GSP assembly language. This code description is then processed by the GSP assembler to produce an integer microcode file which can be used in any subsequent simulation requiring that device.

Phase 2 - Interconnect Description. The user edits an interconnect description file which is used to link the microcode descriptions of the individual modules into a total system microcode description. This description can be used for all subsequent simulations of the particular system.

Phase 3 - Simulation. External system inputs are specified and simulation is begun and repeated as necessary.

The prccess of modeling and coding a "sample" module is illustrated in Figures 3.1 through 3.4.

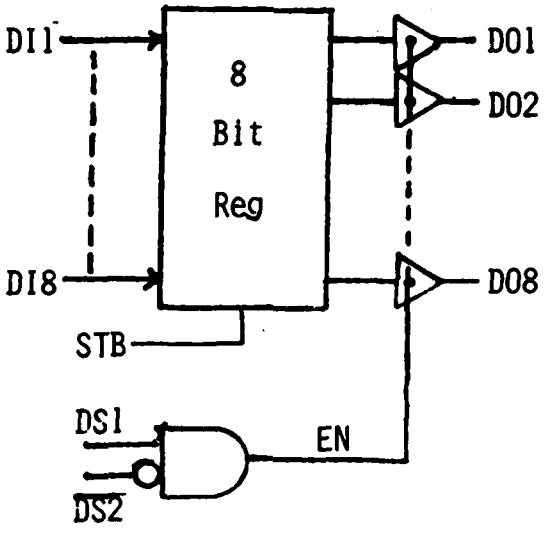
The sample module is an 8 bit register with buffered outputs. Data is clocked into the register on the fall of the strobe (STB). The ouput buffers are enabled (EN = 1) when the input select function (DS1 DS2) is true.

The first step in the modeling process is to examine the chip description and timing specifications to identify module events. As shown in Figure 3.1, the sample module has a 55 NS

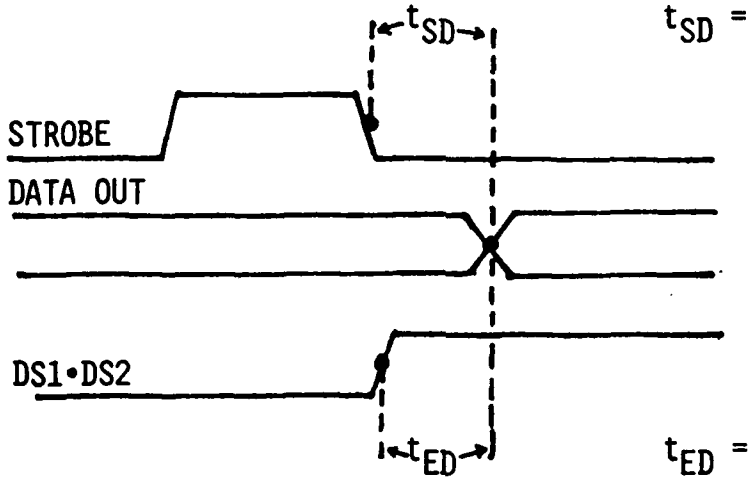
delay (t_{SD}) from the fall of the strobe until data appears on the ouputs (assuming the buffers are enabled). Also, there is a 45 NS delay (t_{ED}) specified from the presence of the enabling input logic condition (DS1 DS2) until ouput data appears.

If the buffer delay is 10 NS, then three delay events can be identified: (1) negative strobe transition to register self call (45NS), (2) positive enable transition to enable self call (35NS), and (3) self calls to data output. The term "self calls" refers to the fact that after the module routine identifies either of the two external events (1) or (2), it causes an event to be placed in the time queue which will call the module routine in a specified number of nanoseconds. Once called, the routine will schedule the register content to appear in the outputs after 10 NS, provided that the buffers are enabled.

Figure 3.2 illustrates the second step in the modeling process: generation of the module flow chart. We have found this step in the modeling process to be very important in assuring an accurate model. A good deal of time can be fruitfully spent in this phase before proceeding to the coding phase.



Timing Specifications:



Events:

1. Neg strobe transition - register self call (45 ns)
2. Pos enable transition - enable self call (35 ns)
3. Self calls to output (10 ns)

Figure 3.1 Sample Module Specifications

For the example under discussion, the flowchart illustrates the logical structure of the model. After the enable (EN) signal is computed, a check is made to see if either the enable or the strobe events have occurred. Note that the events are detected by comparing the present value of a signal with the value of the signal that was stored the previous time the module was called (e.g. STB vs STBO). If either of the events has occurred, an appropriate self call is scheduled. Also, the value of the data or signal involved is "carried along" with the self call event for later storage.

The next section of the flow chart checks for either an enable self call (ENSC) or a data register self call (DRSC). If DRSC = 1, the data register is updated using the previous value of the data input value that was "carried along" with the data register self call. Next, the value of the delayed enable (END) is checked. If END = 1, the data outputs are scheduled to take on the value of the data register in 10 NS. If END = 0, the data outputs will be forced to the all ones configuration simulating the high impedance state. The final activity in the flowchart is the updating of the "old values" of EN (ENO) and STB (STBO). After this, the module procedure is exited.

The final step is the coding of the module description using the GSP assembly language. This is illustrated in Figure 3.3.

Note that the description contains a declaration section and a section containing module micro-operations. The declaration section specifies registers (REG), pin connections (PIN) and events (EVW). Module micro-operations are specified using a rather normal looking assembly language except for instructions like: MOV(W10) D, DO. This instruction causes the contents of the D register to be moved to the ouput in 10 nanosJc.

Figure 3.4 shows the form of the integer microcode file for the sample module.

This section of the report has given only a cursory introduction to the General Simulation Program (GSP). For more information the reader is referred to reference 6.

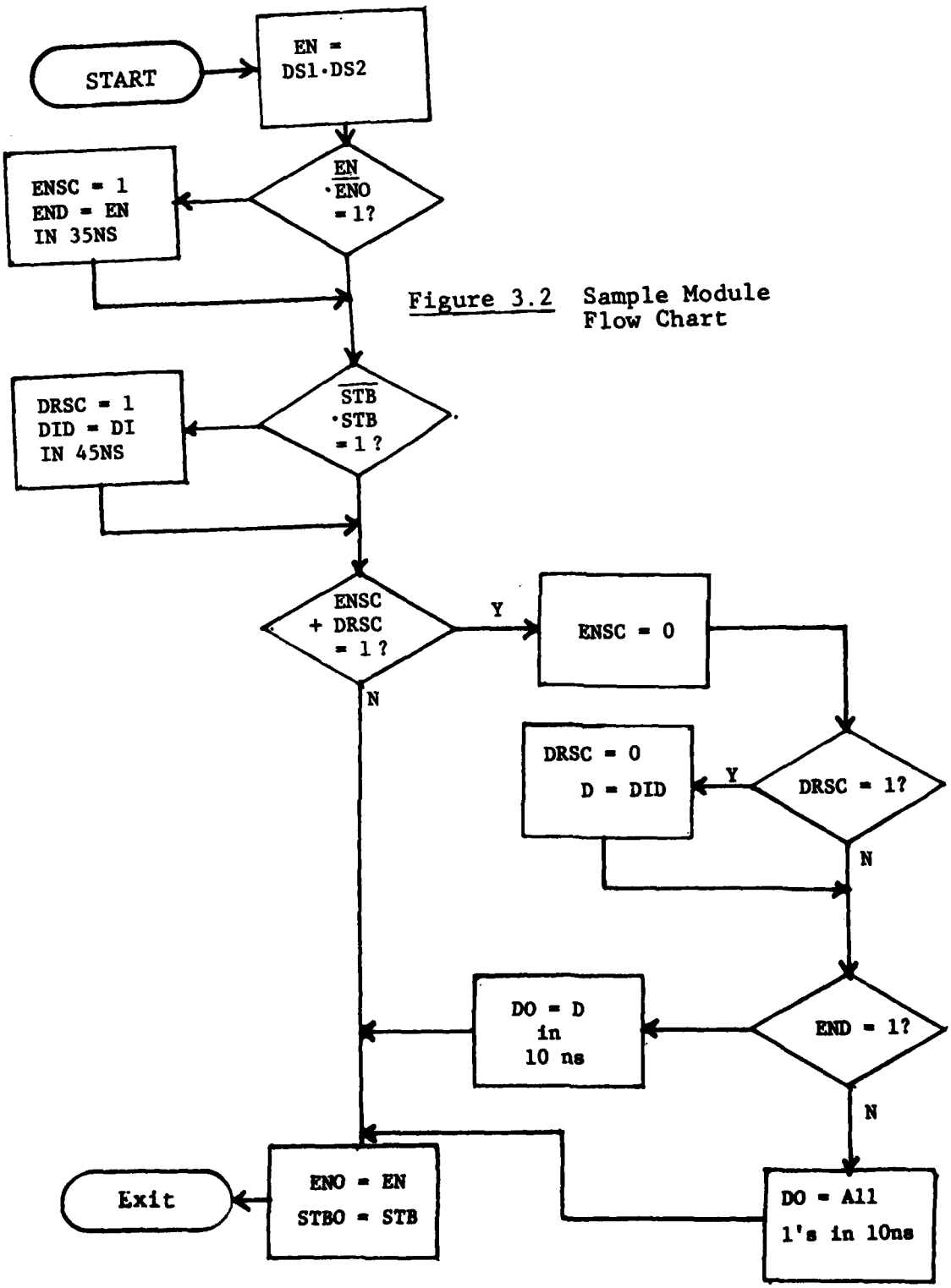


Figure 3.2 Sample Module Flow Chart

```
DECLARATION SECTION:

REG(8) D          ;SAMPLE MODULE, 8 BIT CLOCKED REGISTER
REG(1) TEMP1,EN,ENO,STBO   ;TRI STATE OUTPUT WITH ENABLE CONTROL
PIN DI(1,8),DO(9,16),DS1(17),NDS2(18),STB(19)  ;PIN SPECIFICATIONS
PIN DID(20,27),END(28),EX(61),ENSC(62),DRSC(63) ;PSEUDO PINS
EVW(500) W10(10),W35(35),W45(45)



     MODULE MICROPERATIONS:

     MOV NDS2,TEMP1       ;COMPUTE EN=DS1.NDS2'
     COM TEMP1
     AND DS1,TEMP1,EN
ENC: XOR EN,ENO, TEMP1    ;CHANGE IN OUTPUT ENABLE?
     BEQ TEMP1,STBC
     MOV(W35) #1,ENSC     ;SCHEDULE ENABLE SELF CALL IN 35NS
     MOV(W35) EN,END ;CARRY ALONG CURRENT ENABLE VALUE
STBC:MOV STB,TEMP1        ;COMPUTE STB'.STBO
     COM TEMP1
     AND TEMP1,STBO,TEMP1
     BEQ TEMP1,FLGCK      ;NEG STROBE TRANSITION?
     MOV(W45) #1,DRSC     ;SCHEDULE REG SELF CALL IN 45NS
     MOV(W45) DI,DID      ;CARRY ALONG THE CURRENT INPUT VALUE
FLGCK:OR ENSC,DRSC,TEMP1
     BEQ TEMP1,UPDATE       ;EITHER SELF CALL FLAG SET?
     MOV #0,ENSC          ;RESET FLAG
     BEQ DRSC,OUT         ;CHECK FOR REGISTER SELF CALL
     MOV #0,DRSC          ;RESET FLAG
     MOV DID,D            ;UPDATE D REGISTER
OUT: BEQ END,HIZ           ;CHECK OUTPUT ENABLE
     MOV(W10) D,DO        ;DO=D IN 10NS
     BRU UPDATE
HIZ: MOV(W10) #225,DO     ;DO=ALL 1'S IN 10NS
UPDATE:MOV EN,ENO         ;UPDATE VARIABLES
     MOV STB,STBO
     MOV #0,EX            ;EXIT MODULE
     END
```

FIGURE 3.3 MODULE DESCRIPTION

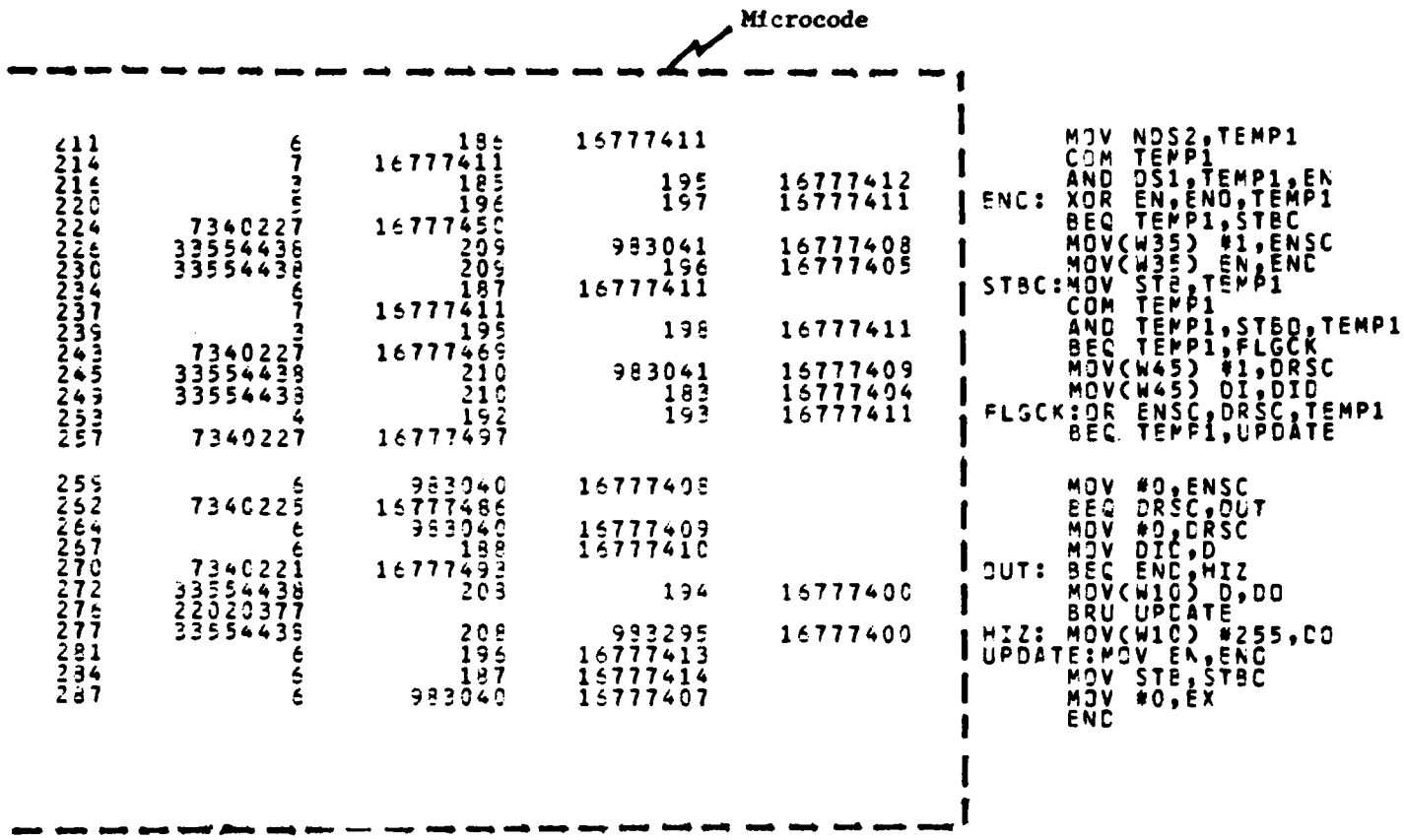


Figure 3.4

Module Microcode File

3.2. The Simulation Model for 8080 System

The system that was modeled consisted of the same chips that are in the hardware system: (1) 8080 microprocessor (2) 8228 System Controller and Bus Driver (3) Semiconductor Random Access Memory (RAM) (4) Read Only Memory (ROM) (5) 8251 Programmable Communicaton Interface (UART) and (6) 8255 Programmable Peripheral Interface (Parallel Port). The real system also contains an 8224 clock chip, however for simulation purposes this was considered to be part of the microprocessor. In addition to the above six models, the gating used to perform address selection was grouped together to form a Select Module. Finally the bus interconnect between the chips was also modeled as a module. Figure 3.5 below shows a diagram of the system model.

As pointed out in section 2, the system test scheme employs wrap around from I/O outputs to I/O inputs. It was not necessary to model this wrap around as separate modules in that we were able to use our regular method of interconnect specification.

3.3. The Modeling Process

The development of the simulation model for a computer system consists of steps which are analogous to hardware development. Models for the individual chips are first developed and checked out. This model development consists of

four steps: (1) examination of the manufacturer's specifications (2) development of a model flow chart (3) coding of the model (4) assembling the model code to produce a "micro code" file. This process was illustrated in section 3.1 for a sample module.

The development of the models for the test system followed the same four steps. Model development and model checkout for LSI chips is a sophisticated process, requiring that the modeler have a thorough understanding of the chip logic. We estimate that approximately 7 man months of effort were expended in model development. We do not discuss each model in detail in this report. However, in appendix B the model flow charts for the test system are given. Appendix C gives the assembly language description for each module. At the present time, a separate document discussing modeling techniques is in the writing process. We will forward this to RADC when completed.

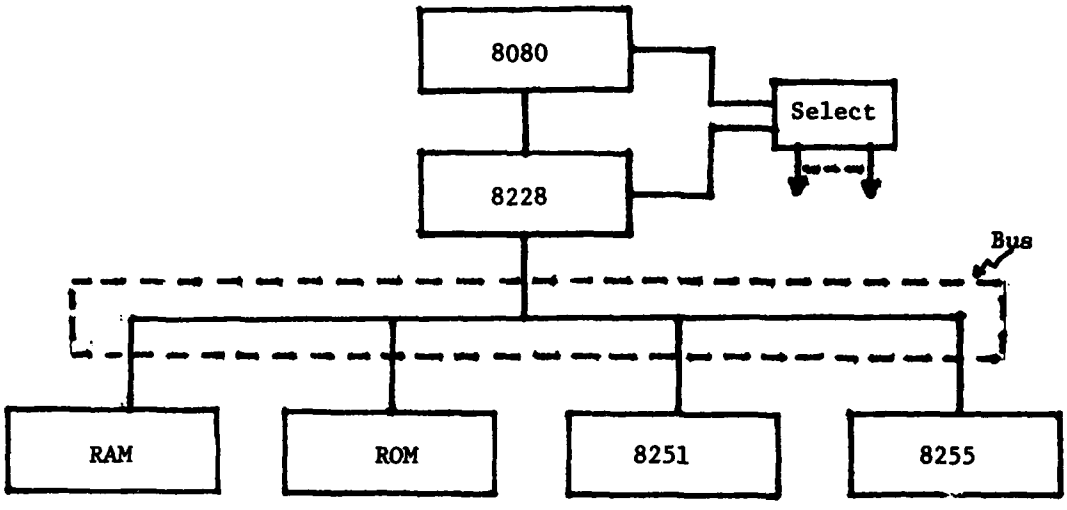


Figure 3.5

System Simulation Model

3.4. Development of the System Model

Once the individual models have been coded and checked out, they are merged to form a system microcode file. Figure 3.6 illustrates the process. It assumes that individual microcode files have been prepared. These files are then merged to form the LINK file. The LINK file holds the microcode for the entire system. In addition it also is used to maintain the state of all system signals initially and as simulation progresses. The merging of module descriptions to form the LINK file is performed automatically upon command.

Another file, the DATA file, contains necessary information on: (1) module interconnection (2) initial signal values (3) module input marking (4) input vector specification. The DATA file for the test system is given in Appendix D. Details of how to prepare the data file are given in reference 6.

Once the LINK file and the DATA file are in place the simulation can begin. As simulation progresses, the state of the system is maintained in the LINK file. Specified system outputs are routed to an output file for storage. The simulation output can also be simultaneously routed to the user's terminal and/or printer. The above discussion implies operation in an interactive computing system; however the simulator can be run in the batch mode as well. In fact, most of our longer simulation runs were made this way.

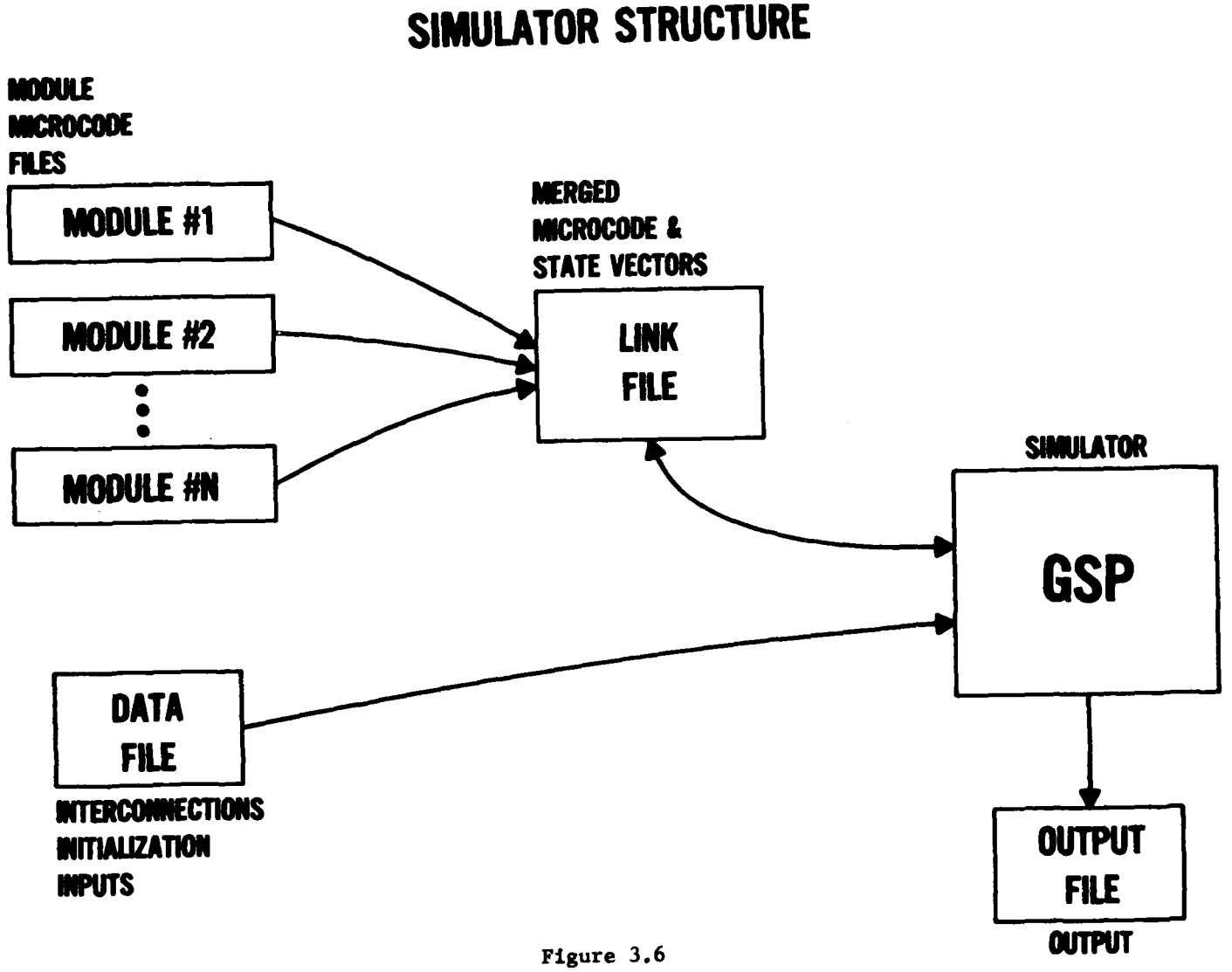

SIMULATOR STRUCTURE
MODULE
MICROCODE
FILES
MODULE #1
MODULE #2
MODULE #N
MERGED
MICROCODE &
STATE VECTORS
LINK
FILE
SIMULATOR
GSP
DATA
FILE
INTERCONNECTIONS
INITIALIZATION
INPUTS
OUTPUT
FILE
OUTPUT

Figure 3.6

3.5. Fault Injection Experiments

The purpose of constructing the simulation model was to conduct "fault injection experiments" in order to assess the effectiveness of the test software. The first step in the process is the injection of the fault into the module microcode. This is done by assessing the effect the fault has on the module response and then incorporating this effect into the model. This incorporation is accomplished by: (1) deleting module code or (2) modifying module code or (3) adding additional module code or (4) some combination of (1), (2) and (3).

This contract was the first time we had ever done this on a large scale. Our approach to doing this was therefore "ad hoc", but we will study the overall results and attempt to derive general principles for fault insertion.

A full list of the faults injected for each chip are given in appendix E. However, they can loosely be divided into three categories:

(1) Incorrect microperations - Examples: "Incorrect operation of carry flag for subtract instruction" (CPU), "multiple register select on read, selecting C also selects L" (CPU), "Bit set/reset command clears Port C output completely" (8255).

(2) stuck at faults

Examples: "Data bus line 2 from the 8228 stuck at zero", address line ADO stuck open (ROM)", and "status register stuck at all zeros" (8251).

(3) timing faults.

Examples: "Write pulse must be 500NS to write correctly instead of the specified 250 NS (8251)" or "Hold times for address and data had to be too large--300NS from the end of the write pulse" (8255).

A basic question that might be asked is how were the faults chosen. With the very high reliability of LSI devices [7], the most likely faults will be interconnect faults, e.g. cold solder joints and printed circuit board defects. Therefore, in our simulation of faults, a high priority was given to interface defects: 43 per cent of the faults injected were interface faults.

In selecting interior chip faults to simulate, the ideal approach would be to first compile a list of the most likely faults from literature data and information obtained directly from the manufacturer. The faults to inject could then be selected from this list. In the case of memory devices, failure modes are well documented [2] so that we were able to do this. For the other chips in the system, i.e. the 8080 processor and it's support chips, no such data is available. We contacted people involved in fault simulation at INTEL and their reply was

that they knew of no fault history for their chips. In light of this situation, we decided that the next best approach was to have the simulation modeler of each chip select the faults, e.g. the person who developed the model for the 8251 would compile a fault list for that chip. Also, to as great a degree as possible, the selector of the faults should not be aware of the characteristics of the test programs. We followed these guidelines as closely as possible given the limited number of people working on the project at one time (4-5) and believe we were able to select an unbiased set of faults to test the self test software.

In conducting the actual injection experiments we wanted to insure that the necessary fault information was collected. To insure this, a standard Fault Injection Experiment Record form was used for each experiment. Figure 3.7 gives an example of this. The Fault Description, as its name implies, describes the physical fault that is inserted, in this example: "data line DØ to 8251 shorted to ground." The System Configuration category lists the module files that were used for the run. The Initial Conditions and Input categories are self explanatory. This information is stored in a DATA file so the name of that file is specified here. The test routine that was being executed is recorded in the "Program Executed" category. A description of how the fault manifested itself is given in the Fault Syndrome category. Finally space is given for additional comments.

We did not include the Fault Experiment Record for each experiment in the report. (They are on file in the Department of Electrical Engineering at VPI&SU.) Instead we prepared for this report, as appendix E, a Fault Experiments Summary. The summary has an entry for each experiment. The entry contains a "description of the fault" and the test results. Test results are classified as detected, program control lost, or not detected. Under our system test concept, a fault can be detected in two ways: (1) the test routine detects the fault or (2) the test routine hangs up in an infinite loop due to lack of response from some section of the hardware. In this second case, a "watch dog timer" would time out indicating a system failure. Loss of program control means that the processor receives faulty instruction data from the ROM and therefore is no longer executing the test routine. It is highly probable, we believe, that a time out will occur in these cases also, but we list them separately since the probability of detection is not unity.

Fault Injection Experiment Record

Date: 8-25-1

Fault Description: Interconnect Fault #8251IB, Data line DØ to 8251 shorted to ground

System Configuration: SYS51

RAM32RB

A8255V6

B8228

A8080N

CSL

BUS

TEST8251 (ROM)

A8251V5

Initial Conditions	Input	Program Executed
	A8251IB DATA A1	TEST8251 SOR A1

Fault Syndrome: Mode word and command word to 8251 were modified to 4C and 14 respectively instead of 4D and 15.

Test incomplete. Processor hangs up. Hardware timer should detect the fault.

Comments:

Figure 3.7

3.6 Analysis of Fault Coverage

Table 3.7 below gives a numerical summary of the results of the fault injection experiments.

System Component	DET.	PCL	NOT DET.	Totals
CPU	33	2	1	36
8228	4		2	6
BUS		3		3
ROM	1	3	1	5
RAM	2	2	3	7
8255	31		7	38
8251	21		2	23
Totals	92	10	16	118
Percent	78	8	14	100

Table 3.7

The data shows that of the 118 faults injected, 92, or 78 percent would definitely be detected, 102, or 86 percent would probably be detected, while 16, or 14 percent, would definitely go undetected. The table also shows, in particular, the sensitivity that the system has to faults in the data path involving the ROM, BUS and 8228. Of the total of 15 faults injected in these three modules, only 3 (20 percent) are definitely detected, 8 (53 percent) are probably detected, while 4 (27 percent) went definitely undetected. This is because these faults effect the instructions that the processor reads and thus would alter the program flow. The coverage of internal CPU faults, on the other hand, was excellent, with 92% definitely detected and 97% probably detected. This compares well with the data given in reference 8.

4. SELF-TEST HARDWARE EXPERIMENTATION AND DOCUMENTATION

The hardware added to the microprocessor system for self-testing was kept to a minimum. The hardware consists of three basic parts: wraparound, isolation, and control logic for self-testing I/O ports; timers and control for initiating the self-tests; and a display for reporting the test results. In addition, some system ROM (1.5K) is required to store the self-test routines and 8 bytes of system RAM must be dedicated to the self-test. Figure 4.1 shows a block diagram of a self-testing 8080 system.

4.1 Experimental Verification of Self-Test System

The self-test described in Appendix A has been successfully verified on the self-test board. A "user" program was run in the foreground to ensure that the self-testing did not interfere with user programs. The program read characters from the console terminal (through the 8251), echoed them, and printed back the whole line of text when a carriage return was typed. It had no problems executing--even through the 8251 tests--despite the fact that self-test passes were made very 75 ms instead of about once a second as would probably be the case in practice.

4.2 Status Display

The self-test displays system status on two LEDs and, optionally, on a 7-segment display. The first LED represents

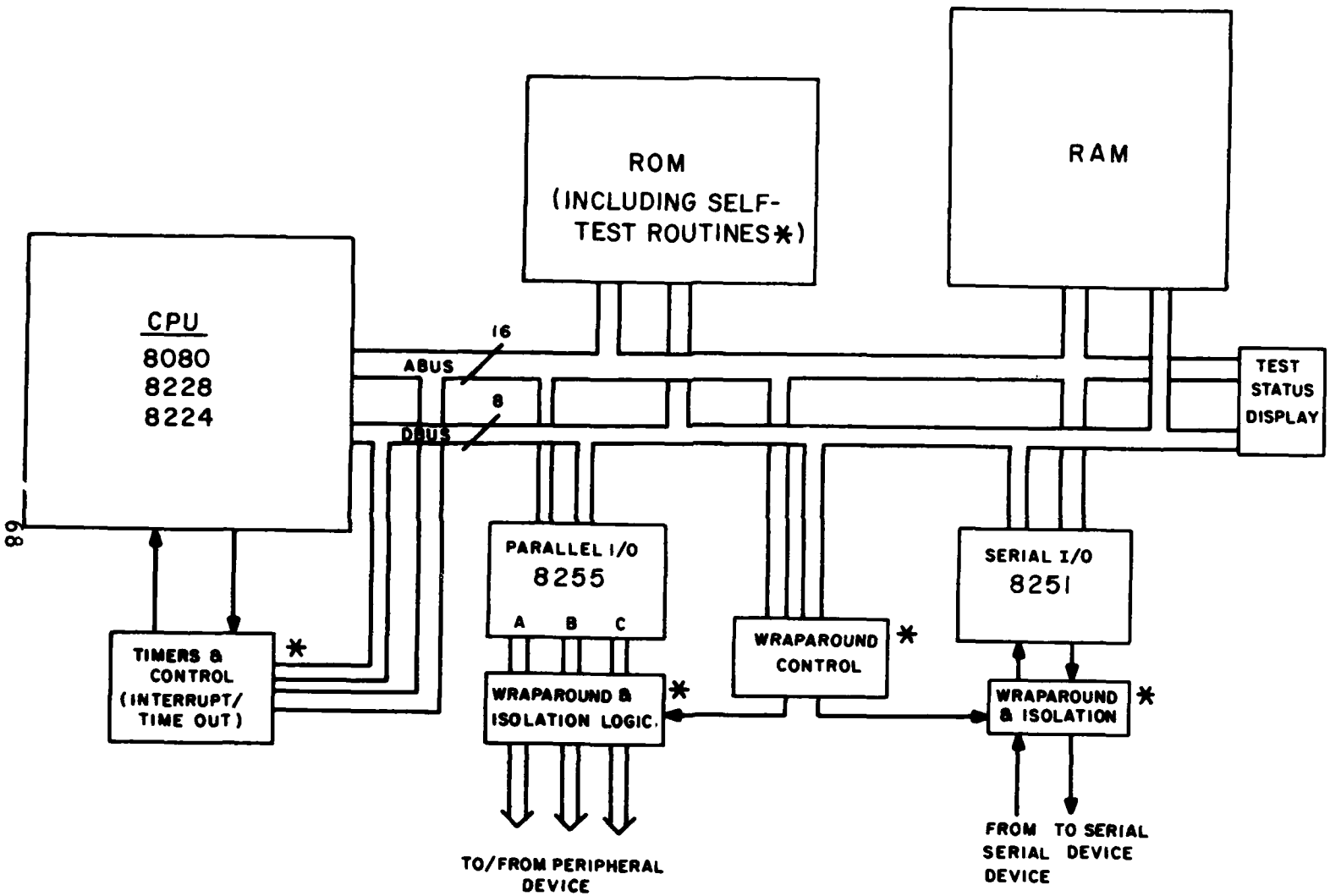


Figure 4.1: Functional Block Diagram of Self-Test System

* indicates logic added for self-testing.

NOT ERROR, it is normally on to indicate NO-ERROR and is extinguished to indicate ERROR. The second LED is a "heartbeat" indicator. It is toggled on and off by successive test passes (or by every nth test pass if n passes are made per second) so that the LED blinks on and off about once per second. This provides a visual indication that the system is successfully self-testing at (about) the proper rate. Note that should either LED burn out an error condition results. The LEDs do, of course, require monitoring. Figure 4.2, shows the LEDs and some of the driving circuitry. The 74373 8-bit latch is used as a memory mapped output port (at address ADDR1); a one or zero (alternately) is written to control bit B0 at the end of each successful test pass (or each n passes) to make the "heartbeat" blink. The ERROR signal driving the ERROR LED is generated by logic to be described in the next section of the report.

The 7-segment display, also shown in Figure 2, can be included to allow some fault location; that is, different codes are written to the display by different tests (CPU, 8255, ROM, RAM, 8251) so that a fault may be isolated to a specific element or card. Since our self-test was not designed for fault diagnosis, many faults in one module are detected by the self-test routine for a different module. For example, a memory fault could cause the CPU memory read/write test to fail, indicating a bad CPU when really the memory is at fault. The

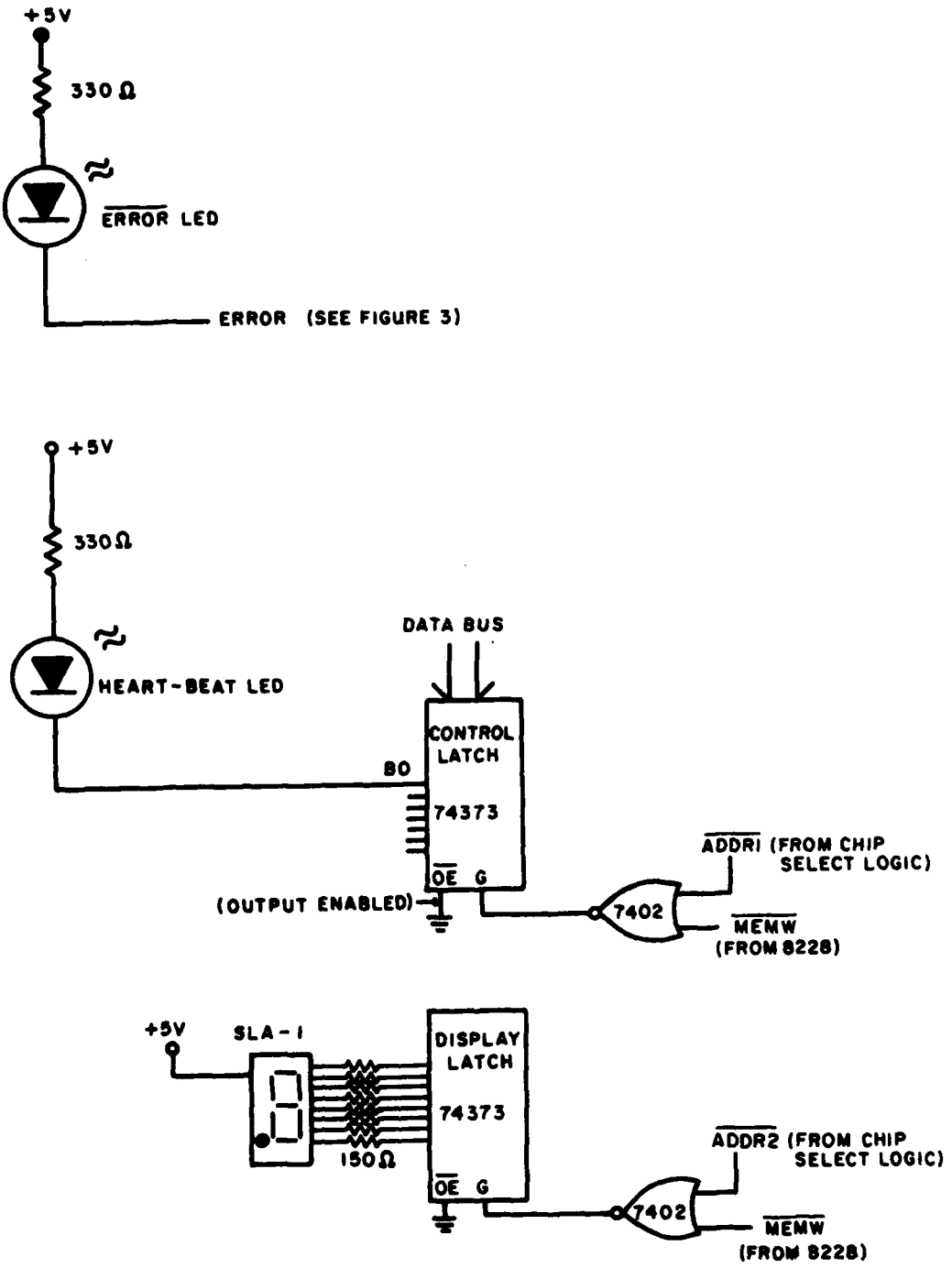


Figure 4.2: Test Status Displays.

display was included on the self-test prototype board mostly as a development aid, but can be incorporated into the system self-test if desired.

Of the three displays, the "heart-beat" provides the best indication that the system is up and running correctly because it must blink on and off (at approximately the right rate). However, certain failures in the self-test hardware could cause loss of the "heartbeat" while the ERROR LED would stay lit (indicating NO ERROR). The two indicators provide a measure of redundancy that allows detection of errors in the self-test hardware itself.

4.3 Timers: Interrupt and Timeout

Figure 4.3 shows the counters and control logic needed to generate the periodic self-test interrupt calls and the hardware timeout. Note that an 8253 Programmable Interval Timer provides the two counters (with one spare); the counters are software controlled. This has the advantage of allowing different test cycle times for different passes of the self-test (an 8251 test pass, for example, requires more time than a ROM test pass). The disadvantage is that the system must function correctly to initialize the counters; failure to do so, however, will be indicated by the loss of the "heartbeat". Both timers (0 & 1) are configured in Mode 0 so that the outputs, initially zero, will change to one and stay there upon the terminal count;

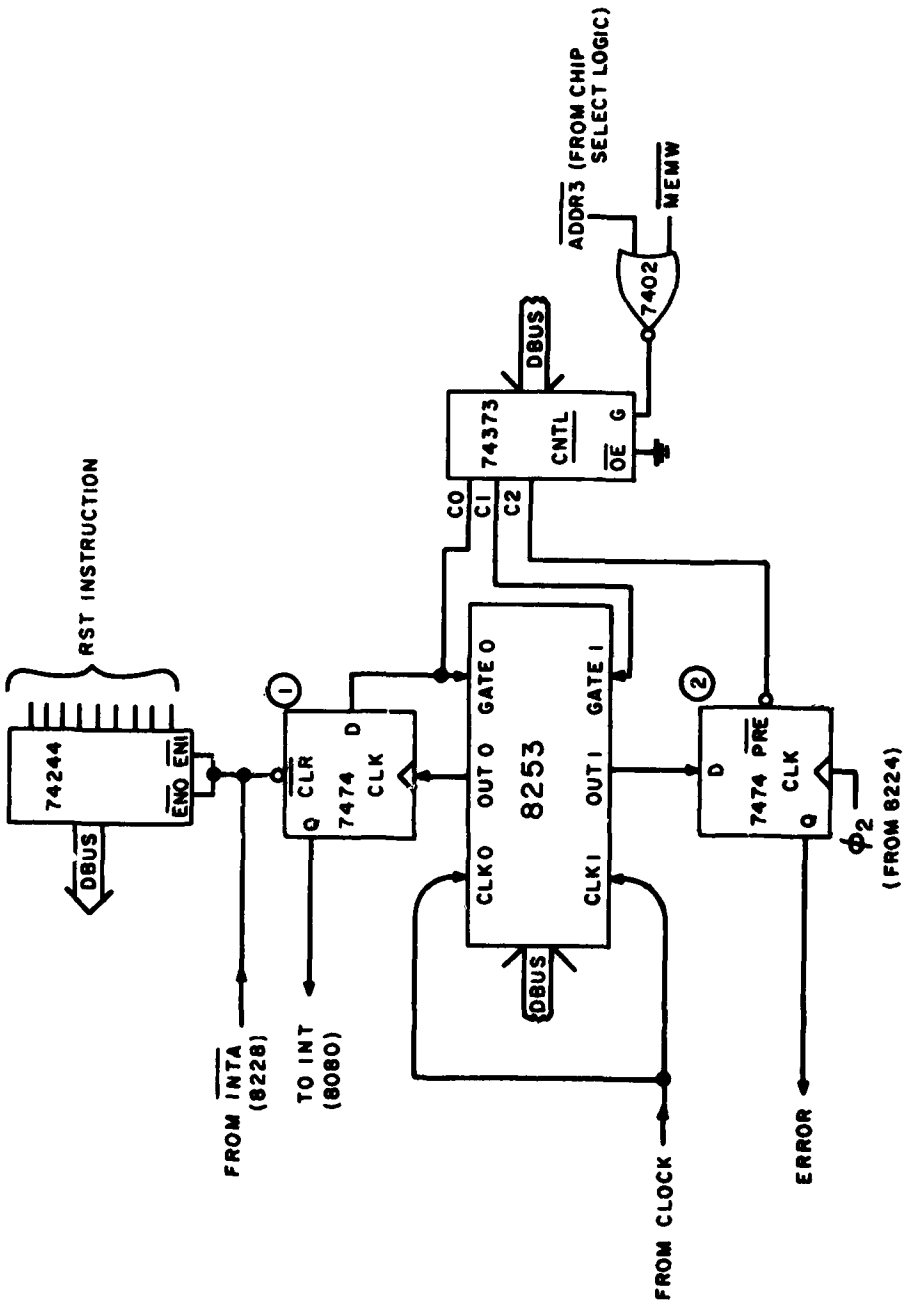


Figure 4.3: Interrupt & Timeout Control Logic.

counting will then be suspended until a new start value is loaded.

Timer 0 is responsible for generating the interrupt request while Timer 1 provides the hardware timeout feature. The 8080 loads Timer 0 with the count value for the desired time between self-test passes, T_0, and loads Timer 1 with a slightly larger value, $T_1 = T_0 + \delta$. The counters are started simultaneously by writing ones to control bits C0 and C1 of the CNTL latch (at memory mapped address ADDR3), thus enabling the Gate inputs on the 8253. When Timer 0 reaches T_0, OUT 0 rises from zero to one, clocking a one (C0) through F/F #1 to generate the interrupt request; Timer 1 will still be counting. If the self-test has not begun within time δ after T_0, Timer 1 will time out and generate the ERROR condition (latched by F/F #2). Time δ is the maximum time needed for the 8080 to process the interrupt, save user registers, and stop Timer 1. Thus Timer 1's first function is to ensure that the self-test is initiated within the allotted time.

Upon interrupt acknowledge, F/F #1 is cleared and an RST instruction is gated onto the Data Bus by the 74244. (The 74244 may be omitted if RST7 is used for the self-test, since that opcode is FF Hex.) The self-test has now been successfully initiated and zeroes are written to CNTL bits C0 and C1, disabling the timers. Timer 1 is now loaded with a new count

value corresponding to the length of the test to be performed on this pass (test times vary depending on what is being tested). A one is then written to CNTL bit C1 to start this timer. Upon successful completion of the test pass, Timer 1 is stopped (by writing a zero to CNTL bit C1), Timers 0 and 1 are reloaded with T_0 and T_1, respectively, and the timers are started (by writing ones to C0 and C1) to await the next test pass. If, for any reason, the self-test does not finish within the allotted time interval, Timer 1 will time out and generate the ERROR signal. Thus Timer 1's second function is to ensure the self-test pass reaches a timely completion (or an error is generated).

During all previous write operations to the CNTL latch (actually writes to address ADDR3), bit C2 was a logic one. If the self-test detects a fault, a zero is written to C2, presetting F/F #2 to generate the ERROR signal; the processor then halts (since successful return of control to the user program is not guaranteed). Thus ERROR may be generated by either a hardware timeout (Timer 1) or by software. The ERROR signal turns off the ERROR LED and is available to the user for any other desired action. Note that upon power-up the ERROR signal may be true (indicating ERROR) for a short while before the software initializes the 8253 and CNTL latch. A one-shot could be used to eliminate this possibility, if necessary.

The hardware shown in Figure 4.3 allows complete software control of the self-testing and hardware timeout. This allows a user program to suspend self-testing during a time-critical operation. The user program simply writes zeroes to CNTL bits C0 and C1 (and a one to C2) to suspend testing, and then writes ones to C0, C1, and C2 to resume self-testing. Note that the "heartbeat" LED will be affected by a long suspension, so that suspensions for over 100 ms or so are not recommended. Note also that if the user program fails to resume self-testing, then the "heartbeat" will stop altogether, indicating a fault.

The "heartbeat" also minimizes the effect of a fault in the hardware of Figure 4.3 or in the software control. If tests are not successfully completed at (about) the proper rate, the "heartbeat" will indicate a fault even if the ERROR LED is still lit.

4.4 Hardware Required to Self-Test I/O Ports

Figures 4.4a and 4.4b illustrate the logic needed for self-testing parallel and serial I/O for an 8080 system. The 8255 parallel I/O port is especially easy to test in Mode 0 since each of the three ports can be configured as either input or output; this allows data output to one port to be input through another port of the same chip. Similarly, the 8251 (being a full duplex USART) has both serial input and serial output so that another UART is not required to self-test it.

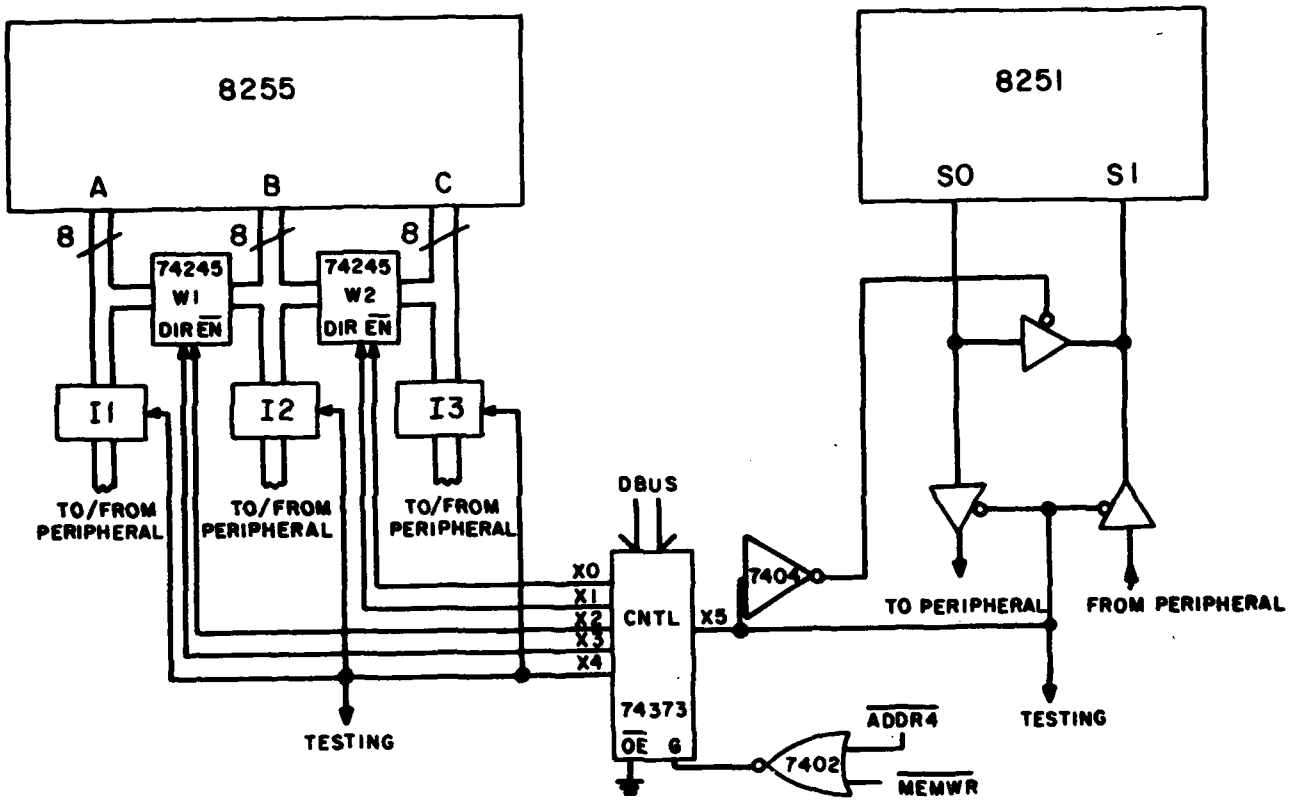


Figure 4.4: Self-Testing I/O

As shown in Figure 4.4a, the self-test logic for the 8255 consists of three (8-bit) isolation buffers (I1-I3), two bidirectional (8-bit) wraparound buffers, and another control latch, memory mapped at address ADDR4. Buffers I1, I2, and I3 isolate the 8255 from its peripheral devices during self-testing of the 8255. Figure 4.5 shows the three possible implementations for an isolation buffer. In Figure 4.5a, the 8255 port is configured as an output port; the 74373 (octal D-type latch) outputs follow the inputs (from the 8255 port) during normal operation. During testing, the TESTING signal goes low and the 74373 latches its current ouputs; thus the user signals are preserved during the test and the peripheral device never sees the test patterns. The 8255 output value is restored upon successful completion of the test, and TESTING is set high once again for normal operation. The TESTING signal is supplied by the CNTL latch bit X4 as shown in Figure 4.4.

In Figure 4.5b, the 8255 port is configured as an input port; now the 74373 outputs feed the peripheral data into the 8255 during normal operation. During testing, the TESTING signal goes high, tri-stating the 74373 outputs. Thus the peripheral input data is prevented from conflicting with test patterns. Figure 4.5c shows a special case; here the 8255 port is sometimes used as an input port and sometimes as an output port. The Y signal is controlled by the user (through a latch-

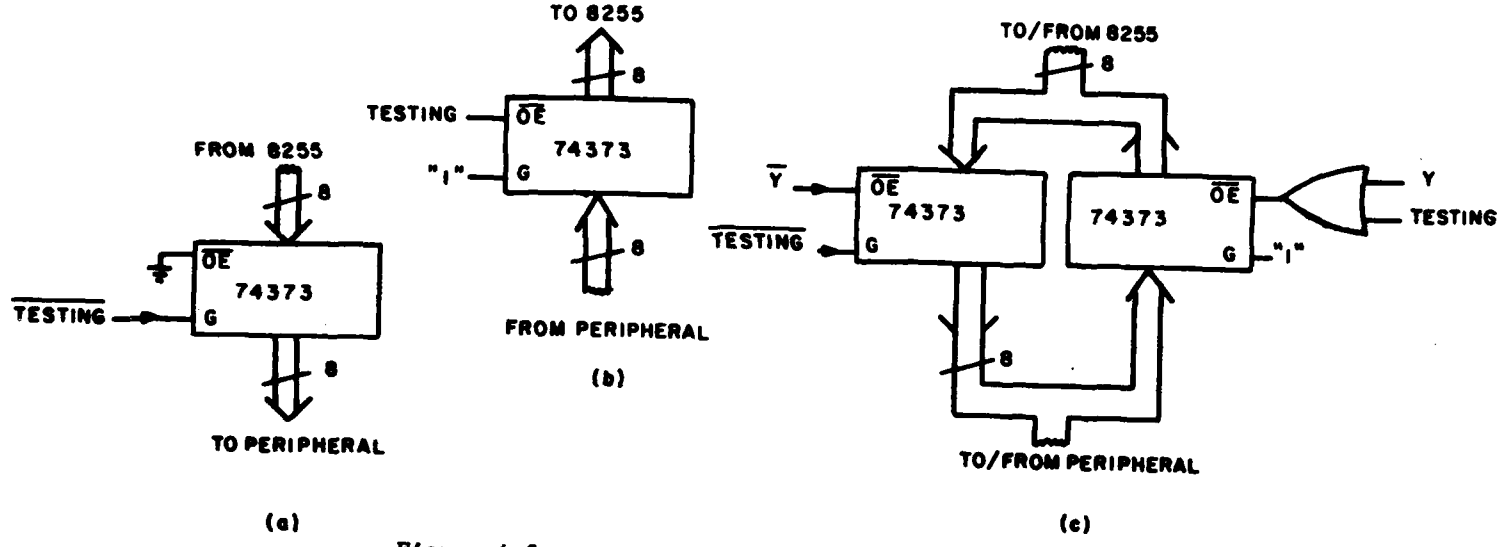


Figure 4.5: Details of the Parallel I/O Isolation.

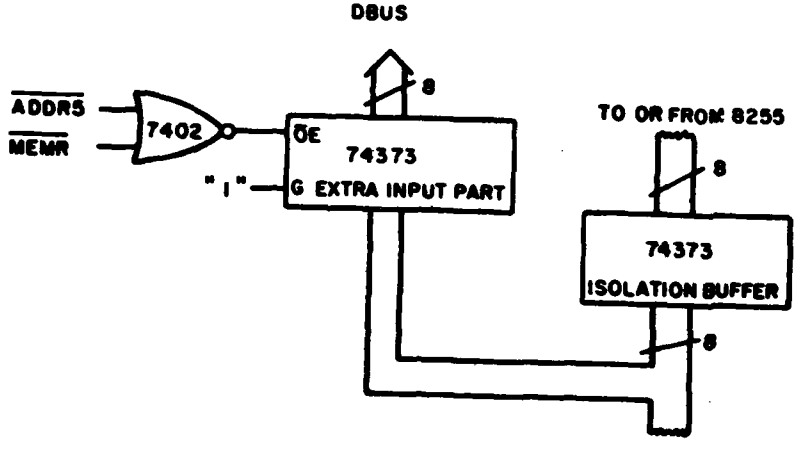


Figure 4.6: Testing Parallel Isolation Buffer.

like CNTL) to define the direction of data flow. This case is merely a combination of the first two cases. During testing, the lefthand 74373 latches its data (in case the port is an output, Y = 1) while the righthand 74373 is tri-stated (in case the port is an input, Y = 0).

The wraparound buffers, W1 and W2, are 74245's, octal bus transceivers with tri-state outputs. During normal operation, their EN inputs (X0 and X2) are high, and both directions have outputs tri-stated. During testing, CNTL bits X0 and X2 are set at zero and bits X1 and X3 are used to define the direction of data flow. Thus each 8255 port can be tested as both an input port and output port.

The serial I/0 self-test logic, as shown in Figure 4.4b, requires two tri-state buffers to isolate the serial peripheral device and one more tri-state buffer to provide wraparound (requiring one 74125). Only one control bit (X5) is required to define either normal operation or self-test mode (wraparound). During testing, the serial input is connected to the serial output so that the 8251 receives what it transmits; the peripheral device is isolated by the tri-stated buffers. During normal operation, the wraparound buffer is tri-stated and the other two buffers enabled for normal serial I/0.

One important consideration here is the effect of a fault in the isolation/ wraparound hardware. A fault in a wraparound

buffer would be detected by the I/O port's self-test. However, a fault in an isolation buffer would not be detected. The isolation can be tested, but an additional input port is required and, more importantly, the testing must be done while writing to or reading from the peripheral device. This requires knowledge of the peripheral device. Figure 4.6 shows an example of testing parallel I/O isolation buffers. If the isolation buffers passes data out to the peripheral, data must be written to the 8255, input through the extra port (ADDR5), and compared. If the isolation buffer passes data in from the peripheral, data must be read through both the 8255 and the extra port, and compared. Thus data from both sides of the isolation buffers is compared. For serial I/O, an extra UART would replace the extra 74373 parallel input port. Again, since the peripheral will either receive or provide the test data, this testing must be part of the applications program.

5. CONCLUSIONS

The main conclusions that can be drawn from this report are:

(1) It is possible to develop efficient self-test routines for detecting faults in the processor, memory, and support chip areas of a microprocessor system. These routines comprise the essential element of a total self-test strategy for microprocessor systems. We found that the CPU and parallel I/0 port tests required the least amount of execution time, in fact both of these tests are made in a single test pass in 2 ms, while RAM memory testing required by far the most time, approx. 80 ms for 1K. ROM testing also required a fair amount of time, approx. 24 ms for 1K. These figures are for a 2MHz clock rate. Serial I/0 testing depends almost entirely upon the baud rate employed as opposed to the other tests which depend on the processor clock rate.

Fault injection experiments indicate that the fault coverage of the self-test strategy is approximately 80%. Failures detected by self-test mechanisms include not only those detected by the self-test routines directly, but also those uncovered by "watch dog" timer mechanisms. This second category of faults is characteristic of situations in which the system becomes totally unresponsive and when the self-test hardware is itself faulty.

(2) A chip level simulation model is an effective tool for evaluating self-test software. This model was used to construct fault injection experiments in order to assess the effectiveness of the self-test software. The results of these simulation experiments were used to calculate the 80% fault coverage figure mentioned above.

The fact that the model was constructed and was used to evaluate self-test software constitutes an important contribution to the state of the art in system validation. To date, accurate modeling and simulation of LSI devices has been prohibitively expensive in many validation situations. The work carried out in this contract has demonstrated the effectiveness of the GSP simulation language in solving this problem.

(3) An effective self-testing system requires only a small amount of self-test hardware. An actual 8080 hardware system was constructed and put into full working order. All self-test routines were run on this system to verify that: (a) the self-test routines will actually run on real equipment (b) the self-test routines, when finished with their execution, leave the system in a state compatible with an operational program (c) that the small amount of self-test hardware that was added, functioned as expected. Our laboratory system operation verified that all three of these requirements were satisfied.

In summary, then, we feel that we have met the three goals of the research contract. In doing so, we have developed an approach to the self-test of microprocessor systems which has been demonstrated as being effective. In addition we have accumulated a large base of concepts and ideas for further important research in the areas of system self-test and system modeling and simulation. We will present some of these ideas in the next section.

6. RECOMMENDATIONS FOR FURTHER RESEARCH

As was emphasized in the conclusion section, a large body of important knowledge has been accumulated in completing Air Force Contract F30602-80-C-0200. In light of this we make the following recommendations for further research:

Future Study

(1) Develop additional self-test library programs that provide greater fault coverage. The logical next step would be to develop a self-test that achieved coverage as close to 100% as possible without imposing any time constraints. This would provide an indication of the time required for such a test. This report describes such a test for RAM memory that requries approximately 20 seconds to test 1K of memory. It would be useful to know the time and the amount of added hardware required to achieve maximum coverage for the CPU as well. Next it would be useful to develop a set of programs with intermediate execution times and fault coverage. The system designer could then select the self-test that best suits his needs. The tradeoffs would be execution time and added hardware cost for additional fault coverage.

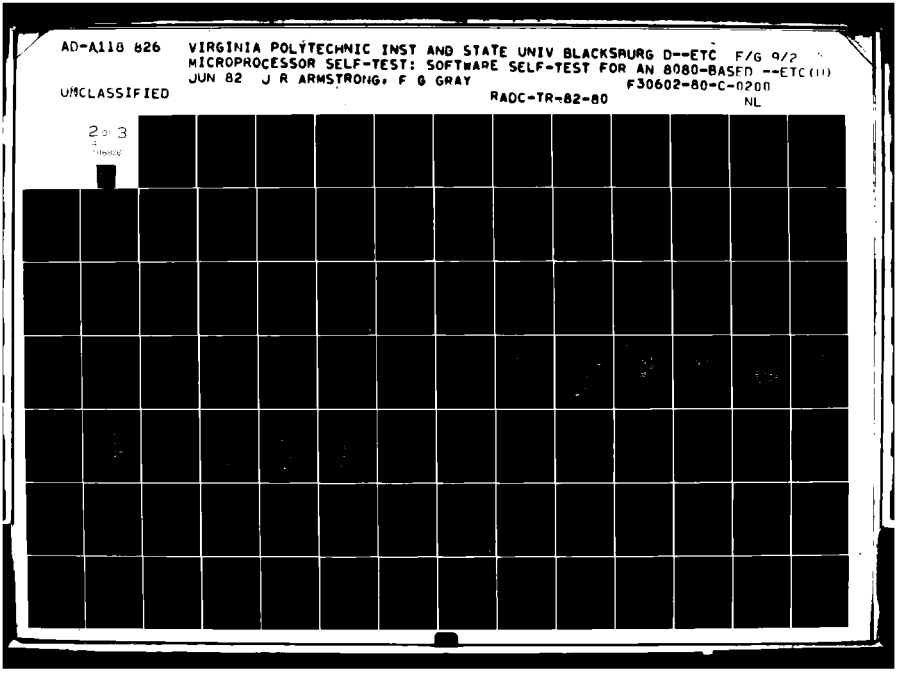
(2) Extension of the self-test techniques to other microprocessor technologies. The self-techniques that were developed were applied to an 8080 system. An important extension of this work could be made in two areas. First, self-

test techniques could be developed for microprocessor systems which possess a higher level of integration. A first step here might be the 8085 system which combines the functions of the 8080 processor and the 8228 and 8224 support chips into one chip. The 8085 has essentially the same instruction set as the 8080, so that this effort would constitute a rather straight forward extension of the present research results. Next, self-test techniques could be developed for a full microcomputer on a chip, such as the Motorola 6802. These chips extend the level of integration present in the 8085 by having both RAM and ROM memory in the chip. The obvious cost and reliability advantage of such chips will dictate their use in avionics systems and it is important that self-test techniques be developed for these chip types also. The second possible area of extension, could be to 16-bit microprocessors such as the 8086, MC68000, and the Z8000. The greater computational power and accuracy of these chips will no doubt result in their extensive use in avionics designs and self-test techniques should be developed for them.

(3) Abstract the features of the various library programs, possibly using a branch of formal mathematics, that would allow the tradeoff to be defined in general terms applicable to variety of microprocessor systems.

(4) Development of simulation models for other microprocessor technologies. This effort would support the work

AD-A118 826 VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG D--ETC F/G 9/2
MICROPROCESSOR SELF-TEST: SOFTWARE SELF-TEST FOR AN 8080-BASED --ETC(U)
JUN 82 J R ARMSTRONG, F G GRAY F30602-80-C-0200
UNCLASSIFIED RADC-TR-82-80 NL



in (2) by allowing fault simulations to be done for these other microprocessor technologies as well.

(5) Development of systematic approaches to modeling of both good and faulty LSI devices. We now have in place, a complete, chip level model of a microprocessor system. Such a resource offers great opportunity for exploring various approaches to the modeling of good and faulty LSI devices. Information gained in such a study, plus that gained in the current contract F30602-80-C-0200 should allow us to develop systematic approaches to this modeling.

References

[1] Jonge, J. Henk and A. J. Smulders, "Moving Inversions Test Pattern is Thorough, Yet Speedy," IEEE Tutorial: LSI Testing, IEEE Computer Society, Long Beach, CA 1978.

[2] Fee, Warren G., "Memory Testing," IEEE Tutorial: LSI Testing, IEEE Computer Society, Long Beach, CA 1978.

[3] Product Evaluation Group \RADC/RBRM] and Howard Dicken, MC8080A Microprocessor and Related Peripheral Devices--Product Evaluations, RADC, Griffiss AFB, NY 1978.

[4] Intel Corporation, MCS-80 User's Manual, 1977.

[5] Kane, Jerry with Adam Osborne, An Introduction to Microcomputers, Volume 3: Some Real Support Devices, Osborne & Associates, Inc., Berkeley, CA 1978.

[6] Devlin, Donald E., "A Chip Level, Multimodule Logic Simulator," Masters Thesis, EE Department VPI&SU, May, 1981.

[7] Queyssac, D., "Projecting VLSI's Impact on Microprocessors," IEEE Spectrum, Vol. 16, No. 5, 1979, pp 38-41.

[8] Baruso, S. J., McGough J. J., and Sworn, F. "Latent Fault Modeling and Measurement Methodology for Application to Digital Flight Controls," Proceedings of the Advanced Flight Control Symposium USAF Academy, Colorado Springs, Colorado, Aug. 5-7, 1981.

Appendix A

Self Test Program Listings

```
8080 MACRO ASSEMBLER, VER 2.4
     ERRORS = 0 PAGE 1

                        ;**********************************************************************
                        ;**                                                                  **
                        ;**              RADC MICROPROCESSOR SELF-TEST PROJECT               **
                        ;**                                                                  **
                        ;**********************************************************************
                        ;
0A00                            START   EQU     0A00H   ;INTERRUPT ENTRY POINT FOR SELF-TEST PASS
                        ;
                        ;       'INIT' IS DEFINED BY ITS POSITION IN THE FILE; CHECK THE
                        ;          SYMBOL TABLE AFTER COMPILATION FOR ITS VALUE.
                        ;
0006                            ERX     EQU     6       ;RST NUMBER FOR ERROR EXIT (RST 6)
                        ;
                        ;****** CONFIGURATION DEPENDENT SYMBOLS DEFINED:
                        ;
0F00                            RAMBEG  EQU     0F00H   ;START ADDRESS OF RAM
0018                            RAMTOP  EQU     18H     ;HIGH BYTE OF RAM END ADDR PLUS ONE!
                        ;
0A00                            ROMBEG  EQU     0A00H   ;START ADDRESS OF ROM
000F                            ROMTOP  EQU     0FH     ;HIGH BYTE OF ROM END ADDR PLUS ONE!
                        ;
3C3C                            PRT55   EQU     3C3CH   ;MEMORY MAPPED ADDR. OF 8255
3C2C                            PRT51   EQU     3C2CH   ;MEMORY MAPPED ADDR. OF 8251
                        ;
3C28                            CNTL1   EQU     3C28H   ;CONTROL FOR 8251 WRAPAROUND & BAUD RATES
3C30                            CNTL2   EQU     3C30H   ;8255 WRAPAROUND & 8253 TIMER/TIMEOUT CONTROL
3C38                            CNTR    EQU     3C38H   ;8253 TIMER MEM. MAPPED ADDR, COUNTER 0
                        ;
                        ;****** STORAGE LOCATIONS:
                        ;
1700                            TSTAD   EQU     1700H   ;2 BYTES FOR TEST ROUTINE ADDRESS
1702                            RBEG    EQU     TSTAD+2 ;2 BYTES FOR RAM/ROM TEST START ADDR
1704                            RANDOM  EQU     RBEG+2  ;1 BYTE FOR RAM TEST 'RANDOM' PATTERN
                        ;
1705                            RAM1    EQU     RANDOM+1 ;3 BYTES FOR MEMORY WRITE TESTING
1706                            RAM2    EQU     RAM1+1  ;   (IN CPU TEST)
1707                            RAM3    EQU     RAM2+1
                        ;
17BE                            BAUDS   EQU     17BEH   ;1 BYTE FOR BAUD RATE/8251 WRAPAROUND CNTL
                        ;
                        ;               BAUDS MUST BE INITIALIZED BY THE USER BEFORE CALLING
                        ;                INIT; BITS 1-3 & 5-7 ARE USER CONTROLLED BAUD RATE BITS;
                        ;                BIT 0 IS 1 FOR WRAPAROUND; BIT 4 IS THE HEARTBEAT.
                        ;
                        ;**********************************************************************
                        ;
                        ;       ROM AND RAM TEST SEGMENT SIZES DEFINED:
                        ;
0080                            ROMSS   EQU     128D    ;TEST ROM IN 128 BYTE SEGMENTS
0020                            RAMSS   EQU     32D     ; AND RAM IN 32 BYTE SEGMENTS
                        ;
```

```
                        ;**********************************************************************
                        ;**                                                                  **
                        ;**                        S E L F - T E S T                         **
                        ;**                                                                  **
                        ;**  PROGRAM:  SELFTEST              VERSION:  2.5                   **
                        ;**                                                                  **
                        ;**  AUTHOR:  DAVID HAISLETT         DATE:  8/28/81                  **
                        ;**                                                                  **
                        ;**  DESCRIPTION:  SERIES STYLE SELF-TEST.  PERFORMS                 **
                        ;**  CPU & 8255 TEST, THEN ROM TEST IN 128 BYTE SEGMENTS,            **
                        ;**  THEN RAM TEST IN 32 BYTE SEGMENTS, THEN 8251 TEST               **
                        ;**  IN 2 PARTS, THEN REPEATS.                                       **
                        ;**                                                                  **
                        ;**********************************************************************
                        ;
0A00                            ORG     START
                        ;**********************************************************************
                        ;**                                                                  **
                        ;**       ENTRY POINT FROM INTERRUPT:  START A SELF-TEST PASS         **
                        ;**                                                                  **
                        ;**********************************************************************
                        ;
0A00  F5                STM:    PUSH    PSW             ;SAVE USERS FLAGS AND REGISTERS
0A01  C5                        PUSH    B
0A02  D5                        PUSH    D
0A03  E5                        PUSH    H
                        ;
0A04  3E70                      MVI     A,      70H     ;SET COUNTER 1 TO MODE 0,
0A06  323B3C                    STA     CNTR+3          ;  LOAD LSB THEN MSB
                        ;                               ;(THESE LOADED IN APPROPRIATE TEST ROUTINE)
0A09  2A0017                    LHLD    TSTAD           ;GT  ADDR OF NEXT TEST OF SERIES
0A0C  E9                        PCHL                    ;AND START IT
                        ;
                        ;**********************************************************************
                        ;**                                                                  **
                        ;**                    E R R :    FAULT DETECTED                    **
                        ;**                                                                  **
                        ;**********************************************************************
                        ;
0A0D  3EF0              ERR:    MVI     A,      0F0H    ;FAULTY SYSTEM, TURN ERR-BAR LED OFF
0A0F  32303C                    STA     CNTL2
0A12  76                        HLT                     ; AND HALT PROCESSING
0A13  76                        HLT
0A14  76                        HLT
                        ;
                        ;
                        ;
                        ;
```

```
                        ;**********************************************************************
                        ;**                                                                  **
                        ;**                  EXIT POINT AFTER A 'GO' PASS                    **
                        ;**                                                                  **
                        ;**********************************************************************
                        ;
0A15  3E30      GOXIT:  MVI     A,      30H     ;SET COUNTER 0 TO MODE 0,
0A17  323B3C            STA     CNTR+3          ; AND LOAD LSB THEN MSB
0A1A  3E70              MVI     A,      70H     ;SET COUNTER 1 TO MODE 0, LOAD LSB, MSB
0A1C  323B3C            STA     CNTR+3
0A1F  21383C            LXI     H,      CNTR    ;POINT HL TO COUNTER 0
0A22  3600              MVI     M,      00H     ;LSB (OF WAIT TIME BETWEEN TEST CYCLES)
0A24  364B              MVI     M,      75D     ;MSB
0A26  23                INX     H               ;POINT HL TO COUNTER 1
0A27  3603              MVI     M,      03D     ;LSB (THIS COUNTER INSURES THAT THE
0A29  364B              MVI     M,      75D     ;MSB (...NEXT TEST CYCLE IS INITIATED)
0A2B  3ABE17            LDA     BAUDS           ;LOAD CNTL1 BYTE SO WE CAN TOGGLE HEARTBEAT
0A2E  EE10              XRI     10H             ; SO TOGGLE IT
0A30  32283C            STA     CNTL1           ;WRITE IT
0A33  32BE17            STA     BAUDS           ;SAVE NEW BAUDS BYTE FOR NEXT PASS
0A36  3EF7              MVI     A,      0F7H    ;START COUNTERS
0A38  32303C            STA     CNTL2
                        ;
0A3B  E1        RSEXIT: POP     H               ;RESTORE USER REGISTERS AND FLAGS
0A3C  D1                POP     D
0A3D  C1                POP     B
0A3E  F1                POP     PSW
0A3F  FB                EI                      ;RE-ENABLE INTERRUPTS
0A40  C9                RET                     ;RETURN TO USER
                        ;
                        ;#########################
                        ;
0A41  00                DB      000H            ;CHECKSUM FOR ROM SEGMENT 1
                        ;
                        ;#########################
```

```
                        ;*****************************************************************
                        ;**                                                             **
                        ;**                      C P U   T E S T                        **
                        ;**                                                             **
                        ;**          VERSION:  2.6.3              DATE:  7/28/81        **
                        ;**                                                             **
                        ;**          DESCRIPTION:  THIS IS VERSION 2 OF THE QUICK CPU   **
                        ;**          SELF-TEST PLUS THE MWRITE, PSW, & LOGIC TESTS.     **
                        ;**                                                             **
                        ;*****************************************************************
                        ;
0A42  3E9F      CPUTS:  MVI     A,      9FH     ;DISPLAY A '1' ON THE 7 SEGMENT DISPLAY
0A44  323D3C            STA     PRT55+1
0A47  21393C            LXI     H,      CNTR+1  ;POINT TO TIMEOUT COUNTER (CNTR 1)
0A4A  3659              MVI     M,      89D     ; LOAD LSB OF TEST TIME (IN CLOCK CYCLES)
0A4C  3600              MVI     M,      0       ; AND THEN MSB
0A4E  3EF6              MVI     A,      0F6H    ;START TIMEOUT COUNTER &
0A50  32303C            STA     CNTL2           ; ENABLE THE TIMEOUT
                        ;
0A53  3AB90B            LDA     INPAT           ;A<-C2H
0A56  0633              MVI     B,33H           ;B<-33H
0A58  1155CC            LXI     D, 0CC55H       ;D,E <- CC55H
0A5B  2AB80B            LHLD    ALHLD           ;H<-66H, L<-AAH
0A5E  EB                XCHG                    ;DE<-66AAH, HL<-CC55
0A5F  4B                MOV     C, E            ;C<-AAH
0A60  C3650A            JMP     OK0             ;TEST UNCOND. JMP
                        ;
0A63  F7                RST     ERX
0A64  76                HLT
                        ;
0A65  1B        OK0:    DCX     D               ;DE<-66A9H
0A66  82                ADD     D               ;A<-28H, CY<-1
0A67  CED7              ACI     0D7H            ;A<-00H
0A69  C20D0A            JNZ     ERR
0A6C  D20D0A            JNC     ERR
0A6F  E20D0A            JPO     ERR
0A72  CA7A0A            JZ      OK1
                        ;
0A75  C30D0A            JMP     ERR
0A78  F7                RST     ERX
0A79  76                HLT
                        ;
0A7A  B4        OK1:    ORA     H               ;A<-CCH
0A7B  2F                CMA                     ;A<-33H
0A7C  04                INR     B               ;B<-34H
0A7D  90                SUB     B               ;A<-FFH, CY<-1
0A7E  F20D0A            JP      ERR
0A81  CA0D0A            JZ      ERR
0A84  C28C0A            JNZ     OK2A
                        ;
0A87  C30D0A            JMP     ERR
```

```
0A8A  F7          RST   ERX
0A8B  76          HLT
                  ;
0A8C  99    OK2A: SBB   C             ;A<-54H
0A8D  FA0D0A      JM    ERR
0A90  2D          DCR   L             ;L<-54H
0A91  BD          CMP   L             ;TEST CMP INSTR.
0A92  C20D0A      JNZ   ERR
0A95  DA0D0A      JC    ERR
                  ;
0A98  42          MOV   B, D          ;B<-66
0A99  A1          ANA   C             ;A<-00 ; CY<-0
0A9A  4F          MOV   C, A          ;C<-0
0A9B  C5          PUSH  B             ;STACK<-6600
0A9C  23          INX   H             ;HL<-CC55
0A9D  E3          XTHL                ;HL<-6600; STACK<-CC55
0A9E  8C          ADC   H             ;A<-66; (CY WAS 0)
0A9F  AA          XRA   D             ;A<-00; CY<-0
0AA0  9D          SBB   L             ;A<-00 (STILL)
0AA1  C20D0A      JNZ   ERR           ;VERIFY
0AA4  D2AD0A      JNC   OK2B
                  ;
0AA7  C30D0A      JMP   ERR
0AAA  F7          RST   ERX
0AAB  76          HLT
                  ;
                  ;##########################
                  ;
0AAC  00          DB    00H           ;CHECKSUM FOR ROM SEGMENT 2
                  ;
                  ;##########################
                  ;
0AAD  0B    OK2B: DCX   B             ;BC<-65FF
0AAE  80          ADD   B             ;A<-65
0AAF  91          SUB   C             ;A<-66
0AB0  C1          POP   B             ;BC<-CC55
0AB1  82          ADD   D             ;A<-CC
0AB2  B8          CMP   B             ;VERIFY B=CC
0AB3  C20D0A      JNZ   ERR
0AB6  F2BE0A      JP    OK3
                  ;
0AB9  C30D0A      JMP   ERR
0ABC  F7          RST   ERX
0ABD  76          HLT
                  ;
0ABE  91    OK3:  SUB   C             ;A<-77
0ABF  A1          ANA   C             ;A<-55
0AC0  3F          CMC                 ;CY<-1
0AC1  99          SBB   C             ;A<-FF; CY<-1
0AC2  DAC A0A     JC    OK4           ;VERIFY
                  ;
```

```
0AC5  C3000A          JMP   ERR
0AC8  F7              RST   ERX
0AC9  76              HLT
                ;
0ACA  AB        OK4:  XRA   E             ;A<-56
0ACB  2F              CMA                 ;A<-A9
0ACC  93              SUB   E             ;A<-00; CY<-0
0ACD  C2000A          JNZ   ERR
0AD0  78              MOV   A,    B       ;A<-CC
0AD1  1F              RAR                 ;A<-66
0AD2  0F              RRC                 ;A<-33
0AD3  57              MOV   D,    A       ;D<-33
0AD4  78              MOV   A,    B       ;A<-CC
0AD5  B2              ORA   D             ;A<-FF
0AD6  FADE0A          JM    OK5           ;VERIFY
                ;
0AD9  C3000A          JMP   ERR
0ADC  F7              RST   ERX
0ADD  76              HLT
                ;
0ADE  3C        OK5:  INR   A             ;A<-00
0ADF  FA000A          JM    ERR           ;CHECK S FLAG RESET
0AE2  C2000A          JNZ   ERR           ;AND Z FLAG SET
0AE5  5F              MOV   E,    A       ;E<-00
0AE6  1D              DCR   E             ;E<-FF
0AE7  93              SUB   E             ;A<-01
0AE8  EA000A          JPE   ERR
0AEB  27              DAA                 ;A<-61
0AEC  C605            ADI   05H           ;A<-66
0AEE  27              DAA                 ;A=66 STILL
0AEF  BC              CMP   H             ;VERIFY
0AF0  C2000A          JNZ   ERR
0AF3  27              DAA                 ;A<-6C
0AF4  27              DAA                 ;A<-72
0AF5  FE72            CPI   72H           ;VERIFY
0AF7  C2000A          JNZ   ERR
0AFA  19              DAD   D             ;HL<-99FF
0AFB  19              DAD   D             ;HL<-CDFE
0AFC  23              INX   H             ;HL<-CDFF
0AFD  23              INX   H             ;HL<-CE00
0AFE  16CC            MVI   D,    0CCH    ;D<-CCH
                ;
                ;     VERIFY ALL REGISTER CONTENTS
                ;
0B00  80              ADD   B             ;A<-3E; CY<-1
0B01  8C              ADC   H             ;A<-0D; CY<-1
0B02  83              ADD   E             ;A<-0C; CY<-1
0B03  85              ADD   L             ;A=0C STILL; CY<-0
0B04  89              ADC   C             ;A<-61; CY<-0
0B05  82              ADD   D             ;A<-2D; CY<-1
0B06  D2000A          JNC   ERR           ;VERIFY CY=1
```

```
0809  EE2D          XRI   2DH           ;VERIFY RIGHT CONTENTS
080B  C20D0A        JNZ   ERR
                    ;
080E  11BA0B        LXI   D,    ADDR1   ;POINT DE TO MEM. LOC
0811  1A            LDAX  D             ;READ CONTENTS TO A: A<-74
0812  EB            XCHG                ;SWAP ADDR1 TO HL
0813  BE            CMP   M             ;VERIFY SAME ACCESS; CY<-0
0814  C20D0A        JNZ   ERR
0817  17            RAL                 ;A<-E8
0818  DEE8          SBI   0E8H          ;NOW VERIFY RIGHT CONTENTS
081A  C20D0A        JNZ   ERR
081D  DA0D0A        JC    ERR           ;CY SHOULD BE ZERO NOW
0820  CD8E0B        CALL  SPTEST        ;GO TEST SP; SET CY BEFORE RETURN
0823  D20D0A        JNC   ERR           ;VERIFY CALL WORKED
                                        ;A=00 UPON RETURN
0826  C637          ADI   37H           ;A<-37
0828  07            RLC                 ;A<-6E; CY<-0
0829  DE6E          SBI   6EH           ;VERIFY
082B  C20D0A        JNZ   ERR
                    ;
                    ;*************************************************************************
                    ;       PSW V1.1:   TEST OF PUSH/POP PSW INSTRUCTIONS
                    ;
082E  3EC6          MVI   A,    0C6H    ;A<-86
0830  C6CE          ADI   0CEH          ;A<-94H
                                        ;FLAGS:  AC,CY,S<-1; P,Z<-0
0832  F5            PUSH  PSW           ;WRITE A & FLAGS TO STACK
0833  AF            XRA   A             ;REVERSE ALL FLAGS & CLEAR A:
                                        ;A<-00; AC,CY,S<-0; P,Z<-1
0834  F1            POP   PSW           ;RESTORE A & FLAGS
0835  D20D0A        JNC   ERR           ;VERIFY FLAGS RESTORED
0838  CA0D0A        JZ    ERR
083B  F20D0A        JP    ERR
083E  EA0D0A        JPE   ERR
0841  27            DAA                 ;VERIFY AC=1; A<-FAH
0842  FEFA          CPI   0FAH          ;IS IT?
0844  CA4B0B        JZ    LOGIC         ;YES, O.K.
0847  C30D0A        JMP   ERR
                    ;
                    ;#########################
                    ;
084A  00            DB    00H           ;CHECKSUM FOR ROM SEGMENT 3
                    ;
                    ;#########################
                    ;
                    ;*************************************************************************
                    ;       LOGIC V1.1:  TESTS THE LOGIC FUNCTIONS (XOR/OR/AND) OF THE
                    ;       ALU FOR STUCK-AT'S AND FUNCTIONAL ERRORS.
                    ;        ALL GATE INPUT COMBINATIONS ARE APPLIED (00,01,10, & 11)
                    ;
084B  3E00      LOGIC:  MVI   A,    0       ;CLEAR A
```

```
0B4D  47          MOV   B,    A       ; AND B
0B4E  0EFF        MVI   C,    0FFH    ;PUT ALL 1'S IN C
                  ;
0B50  AF          XRA   A             ;APPLY 00 TO XOR GATES: A=00 STILL
0B51  B7          ORA   A             ;APPLY 00 TO OR GATES: A=00
0B52  A7          ANA   A             ;APPLY 00 TO AND GATES: A=00
0B53  C20D0A      JNZ   ERR           ;VERIFY
                  ;
0B56  A1          ANA   C             ;APPLY 01 TO AND GATES: A=00 STILL
0B57  C20D0A      JNZ   ERR           ;VERIFY
                  ;
0B5A  B1          ORA   C             ;APPLY 01 TO OR GATES: A<-FF
0B5B  A9          XRA   C             ;APPLY 11 TO XOR GATES: A<-00
0B5C  C20D0A      JNZ   ERR           ;VERIFY
                  ;
0B5F  A9          XRA   C             ;APPLY 01 TO XOR'S: A<-FF
0B60  A1          ANA   C             ;APPLY 11 TO AND'S: A=FF
0B61  B0          ORA   B             ;APPLY 10 TO OR'S: A=FF
0B62  B9          CMP   C             ;VERIFY
0B63  C20D0A      JNZ   ERR
                  ;
0B66  B1          ORA   C             ;APPLY 11 TO OR'S: A=FF
0B67  A8          XRA   B             ;APPLY 10 TO XOR'S: A=FF
0B68  B9          CMP   C             ;VERIFY
0B69  C20D0A      JNZ   ERR
                  ;
0B6C  A0          ANA   B             ;APPLY 10 TO AND'S: A<-00
0B6D  C20D0A      JNZ   ERR           ;VERIFY
                  ;*****************************************************************************
                  ;       MWRITE V1.3:  MEMORY WRITE TESTS (STA,STAX,SHLD,MOV M)
                  ;
0033              PATT1 EQU   33H
00CC              NPAT1 EQU   0FFH - PATT1
00C1              PATT2 EQU   0C1H
AA55              PAT16 EQU   0AA55H
                  ;
0B70  3E33        MVI   A,    PATT1       ;PUT TEST PATTERN IN A
0B72  210517      LXI   H,    RAM1        ;POINT HL TO A RAM BYTE
0B75  110617      LXI   D,    RAM2        ;AND DE TO NEXT BYTE
0B78  010717      LXI   B,    RAM3        ;AND BC TO A 3RD BYTE
0B7B  02          STAX  B                 ;WRITE A TO RAM3
0B7C  2F          CMA
0B7D  12          STAX  D                 ;AND COMPLEMENT PATTERN TO RAM2
0B7E  36C1        MVI   M,    PATT2       ;WRITE ANOTHER PATTERN TO RAM1
0B80  0A          LDAX  B                 ;READ PATT1 BACK FROM RAM3
0B81  EE33        XRI   PATT1             ;VERIFY
0B83  C20D0A      JNZ   ERR
0B86  3A0617      LDA   RAM2              ;READBACK PATT1-BAR FROM RAM2
0B89  FECC        CPI   NPAT1             ;VERIFY
0B8B  C20D0A      JNZ   ERR
```

```
                        ;
                        ;       AT THIS POINT STAX B & STAX D ARE VERIFIED
                        ;
0B8E   3EC1             MVI     A,      PATT2           ;PUT PATTERN 2 INTO A
0B90   4E               MOV     C,      M               ;READ FROM RAM1
0B91   B9               CMP     C                       ;VERIFY
0B92   C20D0A           JNZ     ERR
0B95   2F               CMA
0B96   320517           STA     RAM1                    ;WRITE PATT2-BAR INTO RAM1
0B99   BE               CMP     M                       ;VERIFY
0B9A   C20D0A           JNZ     ERR
0B9D   2155AA           LXI     H,      PAT16           ;PUT 16 BIT PATTERN INTO HL
0BA0   220617           SHLD    RAM2                    ;STORE IN RAM2 & RAM3
0BA3   EB               XCHG                            ;POINT HL TO RAM2
0BA4   7E               MOV     A,      M               ;READBACK RAM2
0BA5   BB               CMP     E                       ;VERIFY
0BA6   C20D0A           JNZ     ERR
0BA9   23               INX     H                       ;POINT TO RAM3
0BAA   7E               MOV     A,      M               ;READBACK RAM3
0BAB   BA               CMP     D                       ;VERIFY
0BAC   C20D0A           JNZ     ERR
                        ;
0BAF   C601             ADI     01H                     ;INCREMENT A WITH AN ADD
0BB1   34               INR     M                       ;THEN INCREMENT MEMORY
0BB2   BE               CMP     M                       ;VERIFY...
0BB3   C20D0A           JNZ     ERR
                        ;
                        ;*******>>>>>>>>>>>>>>>> CPU PASSED THE TEST <<<<<<<<<<<<<<<<<<<<********
                        ;
0BB6   C30B0C           JMP     PIOTS                   ;GO TEST PARALLEL I/O: 8255
                        ;
0BB9   C2               INPAT:  DB      0C2H
                        ;
0BBA   74               ADDR1:  DB      74H
                        ;
0BBB   AA66             ALHLD:  DB      0AAH, 66H
                        ;
                        ;#########################
                        ;
0BBD   00                       DB      00H             ;CHECKSUM FOR SEGMENT 4
                        ;
                        ;#########################
                        ;
                        ;*************************************************************************
                        ;       SPTEST V1.3:  TESTS SP, H, L, ADDRESS LATCH, THE 16 BIT
                        ;        INCREMENTER/DECREMENTER AND THE ACCUMULATOR A FOR STUCK-AT'S
                        ;        AND FOR SHORTS BETWEEN ADJACENT BITS.  THIS ASSUMES THAT
                        ;        THERE WILL BE NO TRIPLE FAULT -- I.E., THAT THE SAME
                        ;        FAULT WILL NOT APPEAR IN (A,H,L) OR (A,SPLOW,SPHIGH).
```

```
                        ;       * INX/DCX ARE ALSO TESTED FOR FULL CARRY/BORROW PROPAGATION.
                        ;
0BBE    210000  SPTEST: LXI     H,      0       ;HL=0000
0BC1    39              DAD     SP              ;READ SP
0BC2    EB              XCHG                    ;SAVE THE OLD SP IN DE
                        ;
0BC3    21AAAA          LXI     H,      0AAAAH  ;LOAD HL WITH 1010 CHECKERBOARD
0BC6    F9              SPHL                    ;WRITE TO SP VIA ADDR. LATCH & INC/DEC CKT.
0BC7    210000          LXI     H,      0       ;ZERO OUT HL
0BCA    3EAA            MVI     A,      0AAH    ;PUT SAME PATTERN IN A
0BCC    39              DAD     SP              ;READ BACK SP INTO HL
0BCD    BC              CMP     H               ;VERIFY CORRECT PATTERN FROM SPHIGH
0BCE    C20D0A          JNZ     ERR             ;--ERROR IF NOT
0BD1    BD              CMP     L               ;CHECK PATTERN FROM SPLOW
0BD2    C20D0A          JNZ     ERR
                        ;
0BD5    2F              CMA                     ;A=55H
0BD6    215555          LXI     H,      5555H   ;WRITE COMPLEMENT CHECKERBOARD TO HL
0BD9    F9              SPHL                    ;AND THEN TO SP
0BDA    210000          LXI     H,      0       ;HL=0000
0BDD    39              DAD     SP              ;READ BACK SP INTO HL
0BDE    BC              CMP     H               ;VERIFY...
0BDF    C20D0A          JNZ     ERR
0BE2    BD              CMP     L
0BE3    C20D0A          JNZ     ERR
                        ;
0BE6    210000          LXI     H,      0       ;HL=0000
0BE9    F9              SPHL                    ;SP=0000
0BEA    3B              DCX     SP              ;SP<-FFFF (TEST BORROW PROPAGATION)
0BEB    3EFF            MVI     A,      0FFH    ;A<-FF
0BED    39              DAD     SP              ;READ IT BACK TO VERIFY
0BEE    BC              CMP     H
0BEF    C20D0A          JNZ     ERR
0BF2    BD              CMP     L
0BF3    C20D0A          JNZ     ERR
                        ;
0BF6    23              INX     H               ;HL<-0000 (TEST CARRY PROPAGATE)
0BF7    2F              CMA                     ;A<-00
0BF8    BC              CMP     H               ;VERIFY
0BF9    C20D0A          JNZ     ERR
0BFC    BD              CMP     L
0BFD    C20D0A          JNZ     ERR
                        ;
                        ;       >>>>>> PASSED TEST
                        ;
0C00    EB              XCHG                    ;MOVE OLD SP TO HL
0C01    F9              SPHL                    ;RESTORE OLD SP
                        ;
0C02    C0              RNZ                     ;TEST RNZ FOR NO-RETURN
0C03    F8              RM                      ; AND RM TOO
0C04    37              STC                     ;SET CY=1 TO SAY 'GO' FOR SPTEST
```

```
0C05  C9              RET                     ;RETURN
                ;
0C06  C30D0A          JMP     ERR             ;RET DIDN'T WORK
0C09  F7              RST     ERX
0C0A  76              HLT
                ;
                ;
                ;*****************************************************************
                ;**                                                             **
                ;**            P A R A L L E L   I / O   T E S T                **
                ;**                                                             **
                ;**          VERSION:  2.1                 DATE:  7/20/81       **
                ;**                                                             **
                ;**          DESCRIPTION:   TEST FOR 8255 MODE 0 OPERATION -    **
                ;**           8 BIT READS/WRITES AND PORT C BIT SET/RESET        **
                ;**           (NO PORT C SPLIT MODE (4 BIT) OPERATION).         **
                ;**                                                             **
                ;*****************************************************************
                ;
                ;       **** ISOLATION BUFFERS WOULD BE TRI-STATED HERE ****
                ;            (EXCEPT THIS HARDWARE HAS NONE)
                ;
0C0B  3E25      PIOTS:  MVI     A,      25H     ;DISPLAY A '2' ON 7-SEGMENT DISPLAY
0C0D  323D3C            STA     PRT55+1
                ;
0C10  213C3C            LXI     H,      PRT55   ;POINT HL TO PORT A ADDRESS
                ;
                ;       ** IF THERE WERE MULTIPLE 8255'S, THEN THE REST OF THIS
                ;       ROUTINE WOULD BECOME A SUBROUTINE. **
                ;
0C13  0680              MVI     B,      80H     ;CMD WORD FOR MODE 0, ALL OUTPUTS
0C15  0E55              MVI     C,      55H     ;A PATTERN TO TEST FOR OUTPUT PORTS
0C17  56                MOV     D,      M       ;SAVE (POSSIBLE) OUTPUT PORT VALUE
0C18  71                MOV     M,      C       ;WRITE PATTERN TO PORT A
0C19  7E                MOV     A,      M       ;READ BACK FROM PORT A
0C1A  B9                CMP     C               ;IS THIS PORT AN OUTPUT?
0C1B  CA240C            JZ      N1              ;YES
0C1E  3C                INR     A               ;MAKE SURE WE READ FF HEX IF NOT 55
0C1F  C20D0A            JNZ     ERR             ;ERROR IF NOT
0C22  0690              MVI     B,      90H     ;PORT A IS AN INPUT
                ;
0C24  23        N1:     INX     H               ;POINT TO PORT B
0C25  5E                MOV     E,      M       ;SAVE (POSSIBLE) OUTPUT PORT VALUE
0C26  D5                PUSH    D               ;SAVE PORT A & B (POSSIBLE) OUTPUT VALUES
0C27  71                MOV     M,      C       ;WRITE PATTERN
0C28  7E                MOV     A,      M       ;READ BACK
0C29  B9                CMP     C               ;OUTPUT?
0C2A  CA350C            JZ      N2              ;YES
0C2D  3C                INR     A               ;NO, MAKE SURE WE READ FF
0C2E  C20D0A            JNZ     ERR
```

```
0C31  3E02            MVI     A,      02H     ;O.K., PORT B IS AN INPUT
0C33  B0              ORA     B               ;OR THIS INTO CMD WORD
0C34  47              MOV     B,      A       ;STORE IN B
                      ;
0C35  23      N2:     INX     H               ;POINT TO PORT C
0C36  56              MOV     D,      M       ;SAVE (POSSIBLE) PORT C OUTPUT VALUE
0C37  71              MOV     M,      C       ;WRITE PATTERN
0C38  7E              MOV     A,      M       ;READ BACK
0C39  B9              CMP     C               ;OUTPUT?
0C3A  CA490C          JZ      N3              ;YES
0C3D  3C              INR     A               ;NO, MAKE SURE WE READ BACK FF
0C3E  CA450C          JZ      N2A             ;O.K.
0C41  C30D0A          JMP     ERR             ; PORT C SPLIT MODE WILL 'FAIL' HERE
                      ;
                      ;##########################
                      ;
0C44  00              DB      000H            ;CHECKSUM FOR SEGMENT 5
                      ;
                      ;##########################
                      ;
0C45  3E09    N2A:    MVI     A,      09H     ;PORT C IS AN INPUT
0C47  B0              ORA     B               ;ADD REST OF CMD WORD
0C48  47              MOV     B,      A       ;STORE IN B
                      ;
0C49  4A      N3:     MOV     C,      D       ;MOVE PORT C VALUE TO REG. C
0C4A  C5              PUSH    B               ;SAVE CONFIG. WORD & (POSS) PORT C OUTPUT VAL
                      ;
0C4B  23              INX     H               ;POINT TO CMD PORT
                      ;
                      ;****** MODE 0 PARALLEL I/O PORT TEST (8255A) ******
                      ;
0C4C  368B    MOPIOT: MVI     M,      8BH     ;SET MODE 0; DEFINE A AS OUTPUT, B & C AS INPUT
0C4E  3E26            MVI     A,26H           ;WRAP A AROUND TO B & C...
0C50  32303C          STA     CNTL2
                      ;****
0C53  45              MOV     B,      L       ;SAVE CMD. ADDR. IN B
0C54  7D              MOV     A,      L       ;PUT ADDR IN A SO WE CAN...
0C55  D603            SUI     03H             ;ADJUST TO PORT A ADDR.
0C57  6F              MOV     L,      A       ;MOVE BACK TO L
0C58  4F              MOV     C,      A       ;SAVE THIS IN C
                      ;
0C59  3E55            MVI     A,      55H     ;PUT CHECKERBOARD IN A AS 1ST PATTERN
0C5B  CD2F0D          CALL    RVBC            ;WRITE IT, READ & VERIFY B & C
0C5E  3EAA            MVI     A,      0AAH    ;2ND PATTERN
0C60  CD2F0D          CALL    RVBC
0C63  3E00            MVI     A,      00H     ;3RD PATTERN
0C65  CD2F0D          CALL    RVBC
                      ;
0C68  3E06            MVI     A,      06H     ;WRAP B AROUND TO A & C
0C6A  32303C          STA     CNTL2
                      ;****
```

```
0C6D  68          MOV   L,    B       ;POINT TO CMD. ADDR.
0C6E  3699        MVI   M,    99H     ;MODE 0:A & C ARE INPUTS; B IS OUTPUT
0C70  2B          DCX   H
0C71  2B          DCX   H             ;POINT TO PORT B
0C72  3ECC        MVI   A,    0CCH    ;1ST PATTERN
0C74  CD3E0D      CALL  RVAC          ;WRITE IT; READ & VERIFY PORTS A & C
0C77  3E66        MVI   A,    66H     ;2ND PATTERN
0C79  CD3E0D      CALL  RVAC
0C7C  3E33        MVI   A,    33H     ;3RD PATTERN
0C7E  CD3E0D      CALL  RVAC
                  ;
0C81  3E86        MVI   A,    86H     ;WRAP C AROUND TO A & B
0C83  32303C      STA   CNTL2
                  ;****
0C86  68          MOV   L,    B       ;POINT TO CMD. ADDR
0C87  3692        MVI   M,    92H     ;SET MODE 0: A & B AS INPUTS; C AS OUTPUT
0C89  2B          DCX   H             ;POINT TO PORT C
0C8A  3ED9        MVI   A,    0D9H    ;1ST PATTERN
0C8C  CD4D0D      CALL  RVAB          ;WRITE IT; READ & VERIFY A & B
0C8F  3E6C        MVI   A,    6CH     ;2ND PATTERN
0C91  CD4D0D      CALL  RVAB
0C94  3E36        MVI   A,    36H     ;3RD PATTERN
0C96  CD4D0D      CALL  RVAB
                  ;
                  ;       PASSED FIRST PART OF TEST
                  ;
                  ; ***** BIT SET/RESET TEST (PORT C, MODE 0) *****
                  ;
                  ;    PORTS A AND B MUST BE INPUTS; PORT C MUST BE OUTPUT
                  ;    WRAPAROUND MUST BE ENABLED FROM C TO B
                  ;
0C99  3655    BTEST:  MVI   M,    55H     ;WRITE CHECKERBOARD TO C -> 55
0C9B  2B          DCX   H             ;POINT TO PORT B
0C9C  54          MOV   D,    H       ;COPY HL TO DE
0C9D  5D          MOV   E,    L
0C9E  23          INX   H             ;POINT TO C
0C9F  23          INX   H             ;THEN TO CMD PORT
0CA0  3600        MVI   M,    00H     ;RESET BIT 0 -> 54
0CA2  EB          XCHG                ;POINT TO PORT B
0CA3  7E          MOV   A,    M       ;READ C THRU B
0CA4  FE54        CPI   54H           ;VERIFY
0CA6  C2000A      JNZ   ERR
                  ;
0CA9  EB          XCHG                ;POINT TO CMD
0CAA  3603        MVI   M,    03H     ;SET BIT 1 -> 56
0CAC  360F        MVI   M,    0FH     ;SET BIT 7 -> D6
0CAE  EB          XCHG                ;PORT B
0CAF  7E          MOV   A,    M       ;READ BACK & VERIFY
0CB0  FED6        CPI   0D6H          ;RIGHT PATTERN?
0CB2  C2000A      JNZ   ERR
                  ;
```

```
0CB5  EB          XCHG                    ;CMD
0CB6  3604        MVI     M,      04H     ;RESET BIT 2 -> D2
0CB8  3607        MVI     M,      07H     ;SET BIT 3 -> DA
0CBA  EB          XCHG                    ;PORT B
0CBB  7E          MOV     A,      M       ;READ BACK & VERIFY
0CBC  FEDA        CPI     0DAH
0CBE  C20D0A      JNZ     ERR
                ;
0CC1  EB          XCHG                    ;CMD
0CC2  360C        MVI     M,      0CH     ;RESET BIT 6 -> 9A
0CC4  3608        MVI     M,      08H     ;RESET BIT 4 -> 8A
0CC6  EB          XCHG                    ;PORT B
0CC7  7E          MOV     A,      M       ;READ BACK & VERIFY
0CC8  FE8A        CPI     8AH
0CCA  C20D0A      JNZ     ERR
                ;
0CCD  EB          XCHG                    ;CMD
0CCE  3601        MVI     M,      01H     ;SET BIT 0 -> 8B
0CD0  3608        MVI     M,      0BH     ;SET BIT 5 -> AB
0CD2  3605        MVI     M,      05H     ;SET BIT 2 -> AF
0CD4  EB          XCHG                    ;PORT B
0CD5  7E          MOV     A,      M       ;VERIFY
0CD6  FEAF        CPI     0AFH            ;RIGHT PATTERN?
0CD8  C20D0A      JNZ     ERR
                ;
0CDB  EB          XCHG                    ;CMD
0CDC  3606        MVI     M,      06H     ;RESET 3 -> A7
0CDE  360E        MVI     M,      0EH     ;RESET BIT 7 -> 27
0CE0  360D        MVI     M,      0DH     ;SET BIT 6 -> 67
0CE2  EB          XCHG                    ;PORT B
0CE3  7E          MOV     A,      M       ;VERIFY
0CE4  FE67        CPI     67H
0CE6  CAED0C      JZ      BTC             ;O.K.
0CE9  C30D0A      JMP     ERR
                ;
                ;##########################
                ;
0CEC  00          DB      000H            ;CHECKSUM FOR SEGMENT 6
                ;
                ;##########################
                ;
0CED  EB        BTC:  XCHG                ;CMD
0CEE  3609        MVI     M,      09H     ;SET BIT 4 -> 77
0CF0  360A        MVI     M,      0AH     ;RESET BIT 5 -> 57
0CF2  3602        MVI     M,      02H     ;RESET BIT 1 -> 55
0CF4  EB          XCHG                    ;PORT B
0CF5  7E          MOV     A,      M       ;READ BACK & VERIFY
0CF6  FE55        CPI     55H             ;BACK TO WHAT WE STARTED WITH
0CF8  C20D0A      JNZ     ERR
                ;
                ;       *HAVE NOW SET & RESET EACH BIT*
```

```
                        ;
0CFB   EB               XCHG                    ;CMD
0CFC   3601             MVI     M,      01H     ;SET BIT 0 -> 55 STILL (1 ALREADY SET)
0CFE   360E             MVI     M,      0EH     ;RESET 7 -> 55 STILL
0D00   3603             MVI     M,      03H     ;SET 1 -> 57
0D02   EB               XCHG                    ;PORT B
0D03   7E               MOV     A,      M       ;READ BACK & VERIFY
0D04   FE57             CPI     57H
0D06   C20D0A           JNZ     ERR
0D09   EB               XCHG                    ;CMD
0D0A   3603             MVI     M,      03H     ;SET 1 AGAIN -> 57 STILL
0D0C   EB               XCHG                    ;PORT B
0D0D   BE               CMP     M               ;VERIFY
0D0E   C20D0A           JNZ     ERR
                        ;
                        ;****>>>>>>>>>> PASSED TEST <<<<<<<<<<****
                        ;
0D11   3EF6             MVI     A,      0F6H    ;DISABLE WRAPAROUND, LEAVE TIMEOUT ENABLED
0D13   32303C           STA     CNTL2
0D16   EB               XCHG                    ;POINT HL TO CMD PORT
0D17   C1               POP     B               ;B=CONFIG. WORD; C=PORT C (POSS) OUTPUT VAL
0D18   70               MOV     M,      B       ;RESTORE USER'S CONFIGURATION
0D19   2B               DCX     H               ;POINT TO PORT C
0D1A   71               MOV     M,      C       ;RESTORE OUTPUT (MAYBE)
0D1B   D1               POP     D               ;GET PORTS A & B (POSS) OUTPUT VALS
0D1C   2B               DCX     H               ;PORT B
0D1D   73               MOV     M,      E       ;RESTORE PORT B...
0D1E   2B               DCX     H
0D1F   72               MOV     M,      D       ;... AND PORT A
                        ;
                        ;       **** NOW ENABLE ISOLATION BUFFERS AGAIN TO ALLOW DATA
                        ;       THROUGH TO PERIPHERAL. ****
                        ;
0D20   215D0D           LXI     H,      ROMTS   ;NEXT TEST OF SERIES IS ROM TEST
0D23   220017           SHLD    TSTAD
0D26   21000A           LXI     H,      ROMBEG  ;STORE ROM START ADDR FOR ROMTS
0D29   220217           SHLD    RBEG
0D2C   C3150A           JMP     GOXIT           ;SAY 'GO' AND EXIT
                        ;
                        ;*************************************************************
                        ;
                        ;       READ & VERIFY PORTS B & C
                        ;
0D2F   77       RVBC:   MOV     M,      A       ;WRITE PATTERN TO PORT A
0D30   23               INX     H               ;POINT TO PORT B
0D31   56               MOV     D,      M       ;READ FROM PORT B
0D32   BA               CMP     D               ;VERIFY
0D33   C20D0A           JNZ     ERR
                        ;
0D36   23               INX     H               ;POINT TO PORT C
0D37   56               MOV     D,      M       ;READ FROM PORT C
```

```
0D38  BA          CMP   D              ;VERIFY
0D39  C20D0A      JNZ   ERR
0D3C  69          MOV   L,    C        ;POINT TO PORT A AGAIN
0D3D  C9          RET
                  ;
                  ;
                  ;       READ & VERIFY PORTS A & C
                  ;
0D3E  77    RVAC: MOV   M,    A        ;WRITE PATTERN TO PORT B
0D3F  23          INX   H              ;POINT TO PORT C
0D40  56          MOV   D,    M        ;READ FROM C
0D41  BA          CMP   D              ;VERIFY
0D42  C20D0A      JNZ   ERR
                  ;
0D45  69          MOV   L,    C        ;POINT TO PORT A
0D46  56          MOV   D,    M        ;READ FROM A
0D47  BA          CMP   D              ;VERIFY
0D48  C20D0A      JNZ   ERR
0D4B  23          INX   H              ;POINT TO PORT B AGAIN
0D4C  C9          RET
                  ;
                  ;
                  ;       READ & VERIFY PORTS A & B
                  ;
0D4D  77    RVAB: MOV   M,    A        ;WRITE PATTERN TO PORT C
0D4E  69          MOV   L,    C        ;POINT TO PORT A
0D4F  56          MOV   D,    M        ;READ FROM A
0D50  BA          CMP   D              ;VERIFY
0D51  C20D0A      JNZ   ERR
                  ;
0D54  23          INX   H              ;POINT TO PORT B
0D55  56          MOV   D,    M        ;READ FROM B
0D56  BA          CMP   D              ;VERIFY
0D57  C20D0A      JNZ   ERR
0D5A  23          INX   H              ;POINT TO C AGAIN
0D5B  C9          RET
                  ;
                  ;##########################
                  ;
0D5C  00          DB    000H           ;CHECKSUM FOR ROM SEGMENT 7
                  ;
                  ;##########################
                  ;
                  ;
                  ;
                  ;
                  ;
                  ;
                  ;
```

```
                        ;**************************************************************
                        ;**                                                          **
                        ;**                      R O M   T E S T                     **
                        ;**                                                          **
                        ;**          VERSION:  2.1              DATE:  5/18/81       **
                        ;**                                                          **
                        ;**          DESCRIPTION: THIS ROM TEST ADDS TOGETHER ALL    **
                        ;**          ROM BYTES, WRAPPING THE CARRYOUT BACK TO THE LSB.**
                        ;**          THE RESULTING SUM SHOULD BE AAH (10101010 BINARY).**
                        ;**                                                          **
                        ;**************************************************************
                        ;
0D5D  3E0D      ROMTS:  MVI     A,      0DH     ;DISPLAY A '3' ON 7-SEGMENT DISPLAY
0D5F  323D3C            STA     PRT55+1
0D62  21393C            LXI     H,      CNTR+1  ;POINT TO TIMEOUT COUNTER (CNTR 1)
0D65  3667              MVI     M,      103D    ; LOAD LSB OF TEST TIME (IN CLOCK CYCLES)
0D67  3600              MVI     M,      0       ; AND THEN MSB
0D69  3EF6              MVI     A,      0F6H    ;START TIMEOUT COUNTER &
0D6B  32303C            STA     CNTL2           ; ENABLE THE TIMEOUT
                        ;
0D6E  2A0217            LHLD    RBEG            ;GET START ADDR IN HL
0D71  1680              MVI     D,      ROMSS   ;GET # BYTES TO TEST PER PASS
0D73  97                SUB     A               ;CLEAR A
0D74  4F                MOV     C,      A       ;CLEAR C TOO
                        ;
0D75  86        XLOOP:  ADD     M               ;ADD IN A ROM LOCATION
0D76  89                ADC     C               ;WRAP CARRY AROUND TO LSB
0D77  23                INX     H               ;ADVANCE TO NEXT LOCATION
0D78  15                DCR     D               ;COUNT OFF THE BYTE JUST DONE
0D79  C2750D            JNZ     XLOOP           ;NO MATCH, CONTINUE THE ROM TEST
                        ;
                        ;       PASS COMPLETE:  THIS SEGMENT OF ROM EXAMINED.
                        ;
0D7C  FEAA              CPI     0AAH            ;IS CHECKSUM CORRECT?
0D7E  C20D0A            JNZ     ERR             ;NOPE, FAULT
0D81  7C                MOV     A,      H       ;YES, DONE ALL OF ROM YET?
0D82  FE0F              CPI     ROMTOP
0D84  C2900D            JNZ     XR1             ;NOPE, STILL MORE TO DO
                        ;
0D87  21960D            LXI     H,      RAMTS   ;YES, RAM TEST IS NEXT
0D8A  220017            SHLD    TSTAD
0D8D  21000F            LXI     H,      RAMBEG  ;STORE RAM STARTING ADDR. FOR RAMTS
                        ;
0D90  220217    XR1:    SHLD    RBEG            ; OR STORE NEXT ROM START ADDR.
0D93  C3150A            JMP     GOXIT           ;SAY 'GO' AND EXIT
                        ;
                        ;
                        ;
                        ;
                        ;
```

```
                        ;*****************************************************************
                        ;**                                                             **
                        ;**                      R A M   T E S T                        **
                        ;**                                                             **
                        ;**          VERSION:  2.1                 DATE:  7/25/81       **
                        ;**                                                             **
                        ;**          DESCRIPTION:  NONDESTRUCTIVE QUICKIE RAM TEST.     **
                        ;**          TESTS RAM ONE LOCATION AT A TIME USING 'RANDOM'     **
                        ;**          PATTERNS.  DESIGNED FOR SERIES TEST, 32 BYTES/PASS **
                        ;**                                                             **
                        ;*****************************************************************
                        ;
                        ;       THIS ROUTINE PERFORMS A QUICKIE RAM TEST BY PERFORMING
                        ;       A WRITE/READ-VERIFY ON EACH RAM LOCATION IN THE BLOCK
                        ;       WITH EACH OF THE FOLLOWING PATTERNS:
                        ;               (1)     COMPLEMENT OF ORIGINAL CONTENTS
                        ;               (2)     RANDOM PATTERN
                        ;               (3)     COMPLEMENT OF (2)
                        ;               (4)     ORIGINAL CONTENTS (RESTORATION/VERIFICATION)
                        ;
                        ;       ** THE TEST DOES NOTHING TO CHECK ADDRESS DECODING. **
                        ;
                        ;       NOTE THAT THIS IS A NONDESTRUCTIVE TEST (ORIGINAL RAM CONTENTS
                        ;       ARE RESTORED).
                        ;*****************************************************************
                        ;
0D96  3E99      RAMTS:  MVI     A,      99H     ;DISPLAY A '4' ON 7-SEGMENT DISPLAY
0D98  323D3C            STA     PRT55+1
0D9B  21393C            LXI     H,      CNTR+1  ;POINT TO TIMEOUT COUNTER (CNTR 1)
0D9E  367B              MVI     M,      123D    ; LOAD LSB OF TEST TIME (IN CLOCK CYCLES)
0DA0  3600              MVI     M,      0       ; AND THEN MSB
0DA2  3EF6              MVI     A,      0F6H    ;START TIMEOUT COUNTER &
0DA4  32303C            STA     CNTL2           ; ENABLE THE TIMEOUT
                        ;
0DA7  2A0217            LHLD    RBEG            ;PASS START ADDRESS IN HL
0DAA  1620              MVI     D,      RAMSS   ;TEST RAMSS BYTES OF RAM PER PASS
0DAC  3A0417            LDA     RANDOM          ;LOAD RANDOM PATTERN
0DAF  4F                MOV     C,      A       ; AND PASS IT IN C
                        ;
0DB0  46        READQ:  MOV     B,      M       ;READ A RAM LOCATION; SAVE IN B
0DB1  78                MOV     A,      B       ;MOVE IT INTO A TOO
                        ;
0DB2  2F                CMA                     ;COMPLEMENT THE PATTERN
0DB3  77                MOV     M,      A       ;WRITE THIS
0DB4  BE                CMP     M               ;VERIFY
0DB5  C20D0A            JNZ     ERR
                        ;
0DB8  79                MOV     A,      C       ;LOAD RANDOM PATTERN INTO A
0DB9  77                MOV     M,      A       ;WRITE IT
0DBA  BE                CMP     M               ;VERIFY
```

```
0DBB  C20D0A          JNZ     ERR
                ;
0DBE  2F              CMA                             ;COMPLEMENT THIS PATTERN
0DBF  77              MOV     M,      A               ;WRITE IT
0DC0  BE              CMP     M                       ;VERIFY
0DC1  C20D0A          JNZ     ERR
                ;
0DC4  70              MOV     M,      B               ;RESTORE ORIGINAL CONTENTS
0DC5  7E              MOV     A,      M               ;READ IT BACK
0DC6  B8              CMP     B                       ;VERIFY
0DC7  C20D0A          JNZ     ERR
                ;
0DCA  23              INX     H                       ;ADVANCE TO NEXT POSITION
0DCB  15              DCR     D                       ;COUNT OFF A BYTE TESTED
0DCC  C2B00D          JNZ     READQ                   ;MORE TO DO, SO DO IT
                ;
0DCF  7C              MOV     A,      H               ;DONE ALL OF RAM YET?
0DD0  FE18            CPI     RAMTOP
0DD2  C2EC0D          JNZ     RN1                     ;NOPE, STILL SOME LEFT
                ;
0DD5  79              MOV     A,      C               ;YES, DONE ALL OF RAM; GET RANDOM PATTERN
0DD6  A7              ANA     A                       ; AND TEST IT TO MAKE SURE
0DD7  CA0D0A          JZ      ERR                     ;THAT IT'S NOT 0 *(ERROR IF SO)*
0DDA  07              RLC                             ;SHIFT LEFT: MSB -> LSB, CY
0DDB  D2E00D          JNC     RN0                     ;MSB WAS 0, SO WE HAVE THE NEW PATTERN
0DDE  EE1C            XRI     1CH                     ;MSB WAS 1, COMPLEMENT BITS 2,3,4
0DE0  320417  RN0:    STA     RANDOM                  ;SAVE NEW RANDOM PATTERN FOR NEXT PASS
0DE3  21F30D          LXI     H,      SIOT1           ;LOAD ADDR. OF NEXT TEST OF SERIES
0DE6  220017          SHLD    TSTAD                   ;SO WE CAN STORE IT
0DE9  C3150A          JMP     GOXIT                   ;SAY 'GO' AND EXIT
                ;
0DEC  220217  RN1:    SHLD    RBEG                    ;STORE CURRENT END ADDR AS NEXT START ADDR
0DEF  C3150A          JMP     GOXIT                   ;SAY GO AND EXIT
                ;
                ;#########################
                ;
0DF2  00              DB      00H                     ;CHECKSUM FOR ROM SEGMENT 8
                ;
                ;#########################
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
```

```
                        ;**********************************************************************
                        ;**                                                                  **
                        ;**                    S E R I A L   I / O   T E S T                 **
                        ;**                                                                  **
                        ;**          VERSION: 2.2                    DATE:  8/07/81          **
                        ;**                                                                  **
                        ;**          DESCRIPTION:  NONDESTRUCTIVE TEST OF 8251 ASYNCHRONOUS  **
                        ;**          I/O.  TEST IS SPLIT INTO 2 PARTS, A SEND/RECEIVE TEST   **
                        ;**          AND A BREAK SEND/FRAMING ERROR/OVERRUN ERROR DETECT TEST.**
                        ;**                                                                  **
                        ;**********************************************************************
                        ;
                        ;       *** 8251 MUST BE CONFIGURED FOR 8 BIT CHARS ***
                        ;         (IF NOT, YOU MUST CHANGE THE PATTERNS IN PATTS
                        ;         TO THE APPROPRIATE LENGTHS -- MAKE UNUSED BITS 0'S)
                        ;
0002                            RXRDY   EQU  02H        ;RECEIVER INPUT BUFFER FULL (CHAR READY)
0004                            TXMTY   EQU  04H        ;TRANSMIT BUFFER EMPTY (READY FOR CHAR)
                        ;
0000                            LSTPAT  EQU  00H        ;LAST PATTERN IN PATTS
                        ;
0DF3  3E49      SIOT1:  MVI     A,      49H     ;DISPLAY A '5' ON THE
0DF5  323D3C            STA     PRT55+1         ;  7-SEGMENT DISPLAY
0DF8  21393C            LXI     H,      CNTR+1  ;POINT HL TO TIMEOUT COUNTER (CNTR 1)
0DFB  365C              MVI     M,      92D     ;LOAD LSB OF TEST TIME (IN CLOCK CYCLES)
0DFD  3600              MVI     M,      0       ;  AND THEN MSB
0DFF  3EF6              MVI     A,      0F6H    ;START TIMEOUT COUNTER AND
0E01  32303C            STA     CNTL2           ;  ENABLE THE TIMEOUT
                        ;
0E04  212D3C            LXI     H,      PRT51+1 ;POINT HL TO 8251 MEMORY MAPPED ADDR
0E07  3600              MVI     M,      00H     ;DISABLE XMIT & RCV
0E09  3ABE17            LDA     BAUDS           ;LOAD BAUD RATE CONTROL WORD...
0E0C  F601              ORI     01H             ;SET BIT 0 TO ENABLE 8251 WRAPAROUND
0E0E  32283C            STA     CNTL1
0E11  3615              MVI     M,      15H     ;ENABLE XMIT & RCV, CLEAR ERRORS
0E13  2B                DCX     H               ;*POINT TO 8251 DATA ADDR
                        ;
0E14  7E        ASIOT:  MOV     A,      M       ;CLEAR RECEIVE BUFFER
0E15  23                INX     H               ;*ADJUST HL FOR COMMAND
0E16  114C0E            LXI     D,      PATTS   ;POINT DE TO PATTERNS
                        ;
0E19  1A        L0:     LDAX    D               ;READ PATTERN INTO A
0E1A  47                MOV     B,      A       ;TRANSFER TO B
                        ;
0E1B  7E        L1:     MOV     A,      M       ;READ STATUS
0E1C  E604              ANI     TXMTY           ;READY FOR CHAR. TO SEND?
0E1E  CA1B0E            JZ      L1              ;LOOP UNTIL IT IS
0E21  2B                DCX     H               ;*YES, THIS WILL BE DATA
0E22  70                MOV     M,      B       ;WRITE THE PATTERN
0E23  23                INX     H               ;*COMMAND
                        ;
```

```
0E24  7E        L2:     MOV     A,      M       ;READ STATUS
0E25  E602              ANI     RXRDY           ;RECEIVED CHAR. YET?
0E27  CA240E            JZ      L2              ;LOOP TILL IT HAS
0E2A  7E                MOV     A,      M       ;READ STATUS AGAIN TO CHECK:
0E2B  E638              ANI     38H             ;ANY ERRORS?
0E2D  C20D0A            JNZ     ERR             ;YES, PROBLEMS => NO GO
0E30  2B                DCX     H               ;*NO, WANT TO READ DATA NOW
0E31  7E                MOV     A,      M       ;SO DO SO
0E32  B8                CMP     B               ;SAME AS WHAT WE SENT?
0E33  C20D0A            JNZ     ERR             ;ERROR IF NOT
0E36  23                INX     H               ;*ADJUST HL FOR CMD
0E37  13                INX     D               ;POINT DE TO NEXT PATTERN
0E38  FE00              CPI     LSTPAT          ;WAS THAT THE LAST PATTERN?
0E3A  C2190E            JNZ     LO              ;NOPE, CONTINUE
                ;
                ;       *** 8251 PASSED SEND/RECEIVE TEST ***
                ;
0E3D  3ABE17            LDA     BAUDS           ;LOAD BAUD RATE CONTROL WORD
0E40  32283C            STA     CNTL1           ; SO WE CAN TURN OFF WRAPAROUND
0E43  21500E            LXI     H,      SIOT2   ;NEXT TEST IN SERIES IS PART 2 OF 8251 TEST
0E46  220017            SHLD    TSTAD           ; STORE IT
0E49  C3150A            JMP     GOXIT           ; SAY 'GO' & EXIT
                ;
0E4C  55AA00    PATTS:  DB      55H, 0AAH, 00H
                ;
                ;##########################
                ;
0E4F  00                DB      00H             ;CHECKSUM FOR ROM SEGMENT 9
                ;
                ;##########################
                ;
                ;**************************************************************************
                ;
                ;       BREAK SEND/FRAMING ERROR DETECT & OVERRUN ERROR DETECT TESTS
                ;
                ;**************************************************************************
                ;
0E50  21393C    SIOT2:  LXI     H,      CNTR+1  ;POINT HL TO TIMEOUT COUNTER (CNTR 1)
0E53  365C              MVI     M,      92D     ;LOAD LSB OF TEST TIME (IN CLOCK CYCLES)
0E55  3600              MVI     M,      0       ;  AND THEN MSB
0E57  3EF6              MVI     A,      0F6H    ;START TIMEOUT COUNTER AND
0E59  32303C            STA     CNTL2           ;  ENABLE THE TIMEOUT
                ;
0E5C  212D3C            LXI     H,      PRT51+1 ;POINT HL TO 8251 MEMORY MAPPED ADDR
0E5F  3600              MVI     M,      00H     ;DISABLE XMIT & RCV
0E61  3ABE17            LDA     BAUDS           ;LOAD BAUD RATE CONTROL WORD...
0E64  F601              ORI     01H             ;SET BIT 0 TO ENABLE 8251 WRAPAROUND
0E66  32283C            STA     CNTL1
                ;
0E69  361D              MVI     M,      1DH     ;ENABLE XMIT/RCV, CLEAR ERRS, SEND BREAK
                ;
```

```
8080 MACRO ASSEMBLER, VER 2.4
    ERRORS = 0 PAGE 22

0E6B   7E       L3:   MOV   A,     M      ;READ STATUS
                ; * * * * * * * * * * * * * *
0E6C   E620           ANI   20H           ;DETECT *FRAMING ERROR* YET?
                ; * * * * * * * * * * * * * *
0E6E   CA6B0E         JZ    L3            ;LOOP IF NOT
0E71   3615           MVI   M,     15H    ;STOP BREAK XMIT IF SO; CLEAR ERRS
                ;
0E73   7E       L4:   MOV   A,     M      ;READ STATUS
0E74   E604           ANI   TXMTY         ;READY FOR CHAR. TO SEND?
0E76   CA730E         JZ    L4            ;NO, LOOP
0E79   2B             DCX   H             ;*DATA --
0E7A   3663           MVI   M,     63H    ;WRITE PATTERN
0E7C   23             INX   H             ;*COMMAND
                ;
0E7D   7E       L5:   MOV   A,     M      ;WAIT UNTIL CHAR. HAS BEEN SENT
0E7E   E604           ANI   TXMTY
0E80   CA7D0E         JZ    L5
0E83   2B             DCX   H             ;*DATA
0E84   36CC           MVI   M,     0CCH   ;SEND ANOTHER CHAR. WITHOUT READING LAST ONE
0E86   23             INX   H             ;*COMMAND
                ;
0E87   7E       L6:   MOV   A,     M      ;READ STATUS
0E88   E614           ANI   14H           ;WAIT UNTIL BOTH TXEMPTY & OVERRUN ERROR
0E8A   FE14           CPI   14H           ;    FLAGS ARE SET
0E8C   C2870E         JNZ   L6            ;    (OR UNTIL TIMEOUT)
                ;
0E8F   2B             DCX   H             ;*DATA
0E90   7E             MOV   A,     M      ;READ BACK CHAR.
0E91   FECC           CPI   0CCH          ;VERIFY IT'S THE 2ND CHAR
0E93   C20D0A         JNZ   ERR
                ;
                ;     *** 8251 PASSED WHOLE TEST ***
                ;
0E96   3ABE17         LDA   BAUDS         ;LOAD BAUD RATE CONTROL WORD
0E99   32283C         STA   CNTL1         ; SO WE CAN DISABLE WRAPAROUND
0E9C   21420A         LXI   H,     CPUTS  ;REPEAT SERIES OF TESTS STARTING WITH
0E9F   220017         SHLD  TSTAD         ; THE CPU/8255 TEST
0EA2   C3150A         JMP   GOXIT         ;SAY 'GO' AND EXIT
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
                ;
```

```
                        ;*****************************************************************
                        ;**                                                             **
                        ;**                  INITIALIZATION OF SELF-TEST                **
                        ;**                                                             **
                        ;*****************************************************************
                        ;
0EA5  21420A    INIT:   LXI     H,      CPUTS   ;FIRST TEST OF SERIES IS CPU/8255 TEST
0EA8  220017            SHLD    TSTAD           ;STORE IT
0EAB  3E55              MVI     A,      55H     ;INITIALIZE 'RANDOM' PATTERN FOR RAM TEST
0EAD  320417            STA     RANDOM          ;AND STORE THAT
0EB0  3EF8              MVI     A,      0F8H    ;DISABLE TIMERS AND WRAPAROUND
0EB2  32303C            STA     CNTL2
0EB5  21383C            LXI     H,      CNTR+3  ;POINT HL TO 8253 CONTROL
0EB8  3630              MVI     M,      30H     ;SET CNTR 0 (INT) MODE 0, LOAD LSB, MSB
0EBA  3670              MVI     M,      70H     ;SET CNTR 1 (TIMEOUT) MODE 0, LOAD LSB,MSB
                ;;;;;;; MVI     M,      XXXH    ;SET CNTR 2 IF DESIRED
0EBC  2B                DCX     H               ;POINT HL TO CNTR 2
                ;;;;;;; MVI     M,      XXXH    ;LSB    (IF COUNTER 2 TO BE USED)
                ;;;;;;; MVI     M,      XXXH    ;MSB    (IF USED)
0EBD  2B                DCX     H               ;POINT TO CNTR 1
0EBE  3603              MVI     M,      03D     ;LSB (OF TIME 'TILL TIMEOUT IF NO INTERRUPT)
0EC0  364B              MVI     M,      75D     ;MSB
0EC2  2B                DCX     H               ;POINT HL TO COUNTER 0
0EC3  3600              MVI     M,      0       ;LSB
0EC5  364B              MVI     M,      75D     ;MSB (OF WAIT TIME 'TILL INTERRUPT)
0EC7  3EF7              MVI     A,      0F7H    ;START COUNTERS 0 & 1
0EC9  32303C            STA     CNTL2
0ECC  3ABE17            LDA     BAUDS           ;LOAD 8251 BAUD RATE CNTL1 BYTE
0ECF  F610              ORI     10H             ;INITIALIZE THE HEARTBEAT TO 1 (OFF)
0ED1  32BE17            STA     BAUDS           ;STORE THIS
0ED4  32283C            STA     CNTL1           ;AND INIT THE LED
0ED7  FB                EI                      ;ALLOW INTERRUPTS
                        ;
0ED8  C9                RET                     ;RETURN TO USER'S INITIALIZATION ROUTINE...
                        ;
                        ;
                        ;#########################
                        ;
0ED9  00                DB      000H            ;CHECKSUM FOR ROM SEGMENT 10
                        ;
                        ;#########################
                        ;
0EDA  00000000          DW 0,0,0,0, 0,0,0,0
0EDE  00000000
0EE2  00000000
0EE6  00000000
0EEA  00000000          DW 0,0,0,0, 0,0,0,0
0EEE  00000000
0EF2  00000000
0EF6  00000000
0EFA  00000000          DW 0,0,0,0, 0,0
```

```
0EFE   00000000
0F02   00000000
                ;
                ;*****************************************************************
                ;**                                                             **
                ;**            D U M M Y   U S E R   P R O G R A M              **
                ;**                                                             **
                ;*****************************************************************
                ;
                ;       ** THIS PROGRAM IS LOADED INTO RAM TESTED BY THE SELFTEST **
                ;
                ;       READ FROM CONSOLE & PRINT BACK SAME
                ;
1000                    ORG     1000H
                ;
1000   CDA50E   USER:   CALL    INIT            ;INITIALIZE SELF-TESTING
1003   FB               EI                      ;ALLOW INTERRUPTS
1004   214E10   QRY:    LXI     H,      PROMPT  ;POINT HL TO PROMPT
1007   CD3C10           CALL     PRINT          ;PRINT IT
100A   215C10           LXI     H,      BUF     ;POINT HL TO STORAGE BUFFER
                ;
100D   3A2D3C   TREAD:  LDA     PRT51+1         ;READ 8251 STATUS
1010   E602             ANI     RXRDY           ;ANY INPUT THERE?
1012   CA0D10           JZ      TREAD           ;WAIT FOR SOME IF NOT
1015   3A2C3C           LDA     PRT51           ;YES, READ THE CHAR
1018   47               MOV     B,      A       ;STORE IN B
1019   3A2D3C   XECHO:  LDA     PRT51+1         ;READ STATUS AGAIN...
101C   E604             ANI     TXMTY           ;READY TO OUTPUT?
101E   CA1910           JZ      XECHO
1021   78               MOV     A,      B       ;YES, RESTORE CHAR TO A
1022   322C3C           STA     PRT51           ;AND ECHO IT
1025   70               MOV     M,      B       ;STORE IT IN BUFFER
1026   23               INX     H               ;ADVANCE POINTER
1027   E67F             ANI     7FH             ;STRIP OFF PARITY
1029   FE0D             CPI     0DH             ;<CR>?
102B   C20D10           JNZ     TREAD           ;NOPE, KEEP READING
102E   360A             MVI     M,      0AH     ;YES, STORE A <LF> TOO
1030   23               INX     H
1031   3600             MVI     M,      0H      ;STORE A ZERO TO MARK END
1033   215B10           LXI     H,      LFBUF   ;POINT TO <LF> BEFORE BUFFER
1036   CD3C10           CALL    PRINT           ;PRINT <LF> THEN THE BUFFER CONTENTS
1039   C30410           JMP     QRY
                ;
                ;       *** PRINT: PRINT STRING POINTED TO BY HL & TERMINATED BY 00 ***
                ;
103C   3A2D3C   PRINT:  LDA     PRT51+1         ;READ STATUS
103F   E604             ANI     TXMTY           ;READY TO OUTPUT?
1041   CA3C10           JZ      PRINT
1044   7E               MOV     A,      M       ;YES, READ CHAR FROM TEXT STRING
```

```
8080 MACRO ASSEMBLER, VER 2.4
     ERRORS = 0 PAGE 25


  1045   A7                ANA     A               ;IS THIS 00?
  1046   C8                RZ                      ;RETURN IF SO
  1047   322C3C            STA     PRT51           ;OUTPUT IT
  104A   23                INX     H               ;POINT TO NEXT CHAR
  104B   C33C10            JMP     PRINT
                   ;
  104E   454E5445  PROMPT:  DB      'ENTER TEXT: '
  1052   52205445
  1056   58543A20
  105A   00                DB 0
                   ;
  105B   0A        LFBUF:   DB      0AH
  105C   00000000  BUF:     DB 0,0,0,0,0
  1060   00
                   ;
                           END
NO PROGRAM ERRORS
```

SYMBOL TABLE

* 01

A	0007	ADDR1	0BBA	ALHLD	0BBB	ASIOT	0E14 *
B	0000	BAUDS	17BE	BTC	0CED	BTEST	0C99 *
BUF	105C	C	0001	CNTL1	3C28	CNTL2	3C30
CNTR	3C38	CPUTS	0A42	D	0002	E	0003
ERR	0A0D	ERX	0006	GOXIT	0A15	H	0004
INIT	0EA5	INPAT	0B89	L	0005	L0	0E19
L1	0E18	L2	0E24	L3	0E68	L4	0E73
L5	0E7D	L6	0E87	LFBUF	105B	LOGIC	0B4B
LSTPA	0000	M	0006	MOPIO	0C4C *	N1	0C24
N2	0C35	N2A	0C45	N3	0C49	NPAT1	00CC
OK0	0A65	OK1	0A7A	OK2A	0A8C	OK2B	0AAD
OK3	0ABE	OK4	0ACA	OK5	0ADE	PAT16	AA55
PATT1	0033	PATT2	00C1	PATTS	0E4C	PIOTS	0C0B
PRINT	103C	PROMP	104E	PRT51	3C2C	PRT55	3C3C
PSW	0006	QRY	1004	RAM1	1705	RAM2	1706
RAM3	1707	RAMBE	0F00	RAMSS	0020	RAMTO	0018
RAMTS	0D96	RANDO	1704	RBEG	1702	READQ	0DB0
RN0	0DE0	RN1	0DEC	ROMBE	0A00	ROMSS	0080
ROMTO	000F	ROMTS	0D5D	RSEXI	0A3B *	RVAB	0D4D
RVAC	0D3E	RVBC	0D2F	RXRDY	0002	SIOT1	0DF3
SIOT2	0E50	SP	0006	SPTES	0BBE	START	0A00
STM	0A00 *	TREAD	100D	TSTAD	1700	TXMTY	0004
USER	1000 *	XECHO	1019	XLOOP	0D75	XR1	0D90

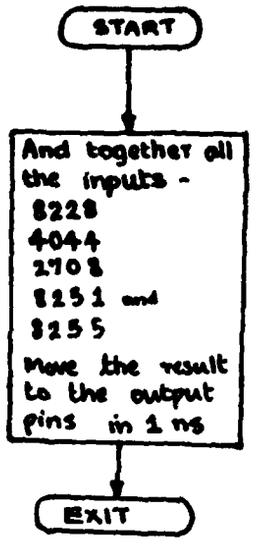
Interested qualified requesters may obtain copies of Appendix B from RADC (COEA), Griffiss AFB NY 13441.

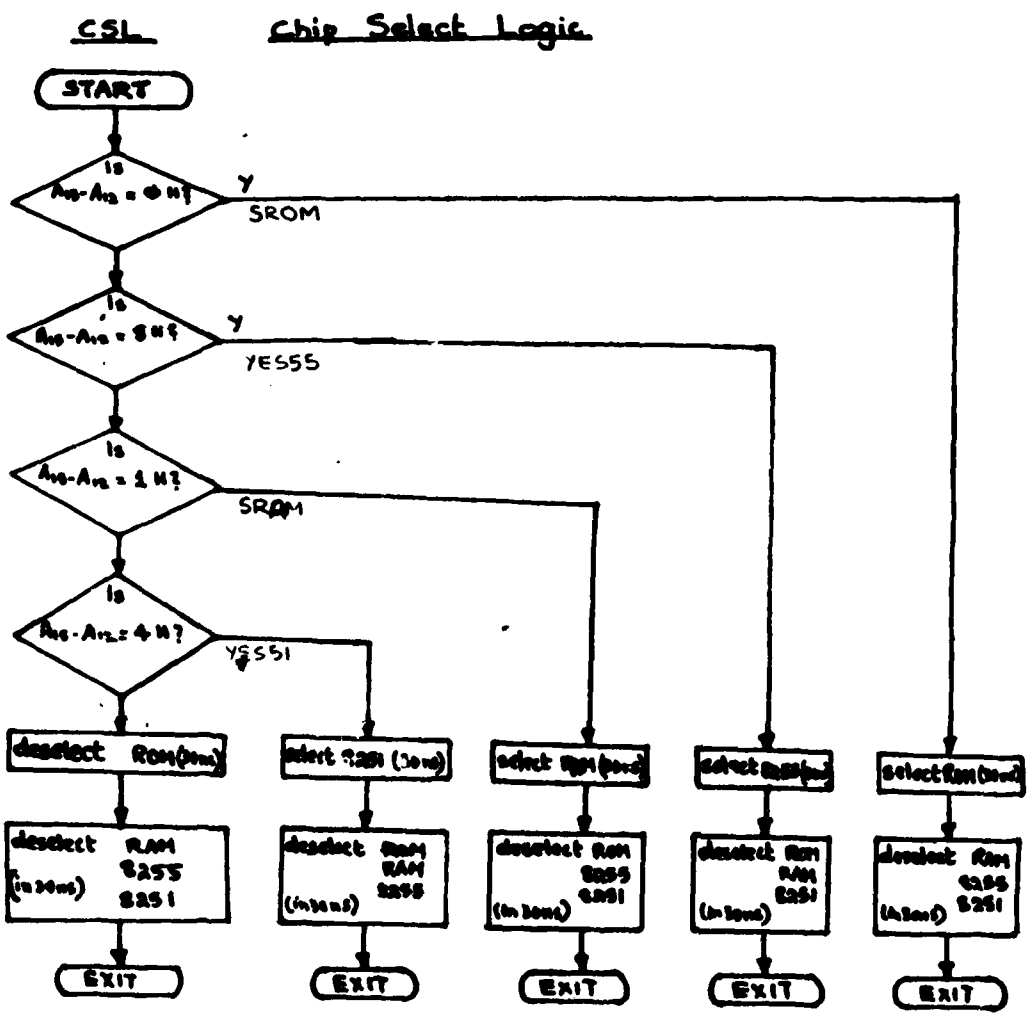
Appendix B

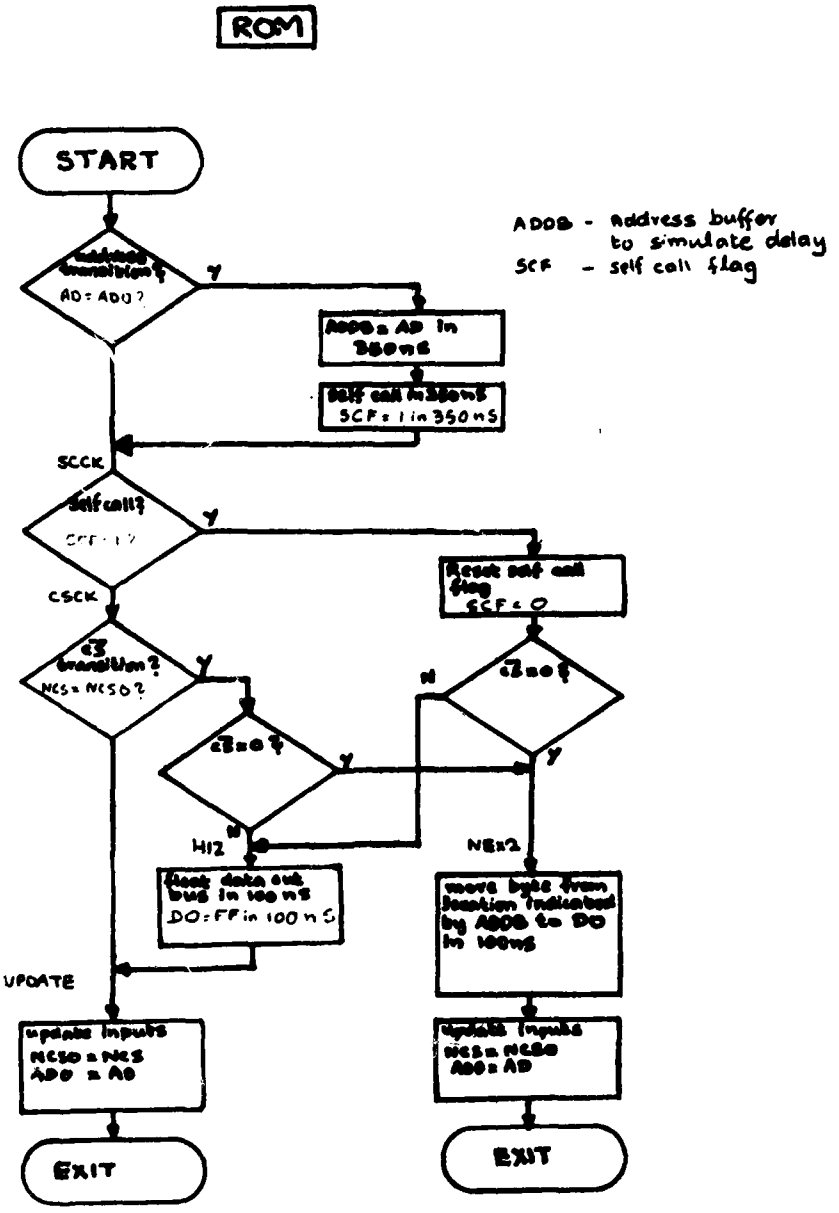
Simulation Model

Flowcharts

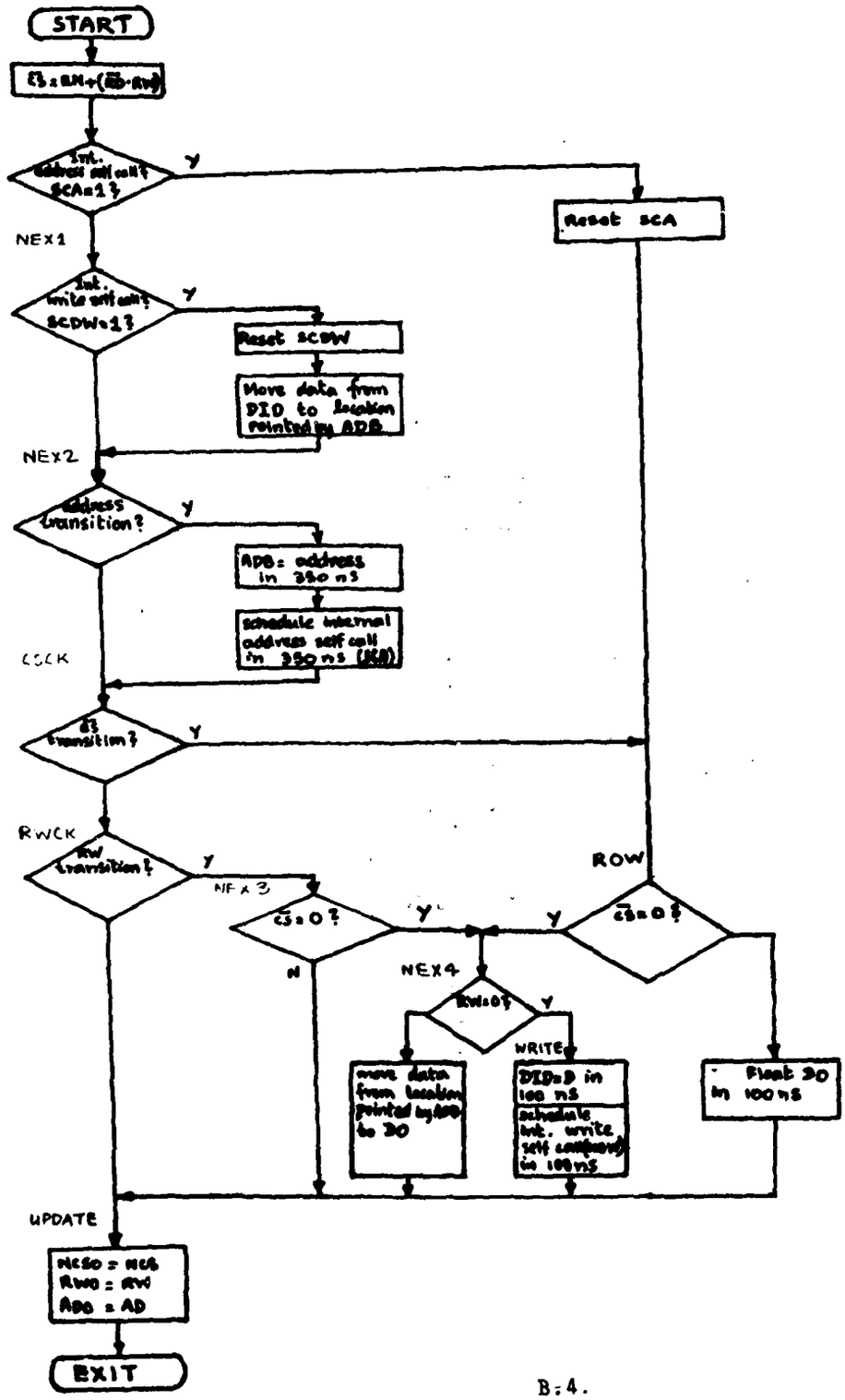
BUS


START
And together all the inputs -
8228
4044
2708
8251 and
8255
Move the result to the output pins in 1 ms
EXIT

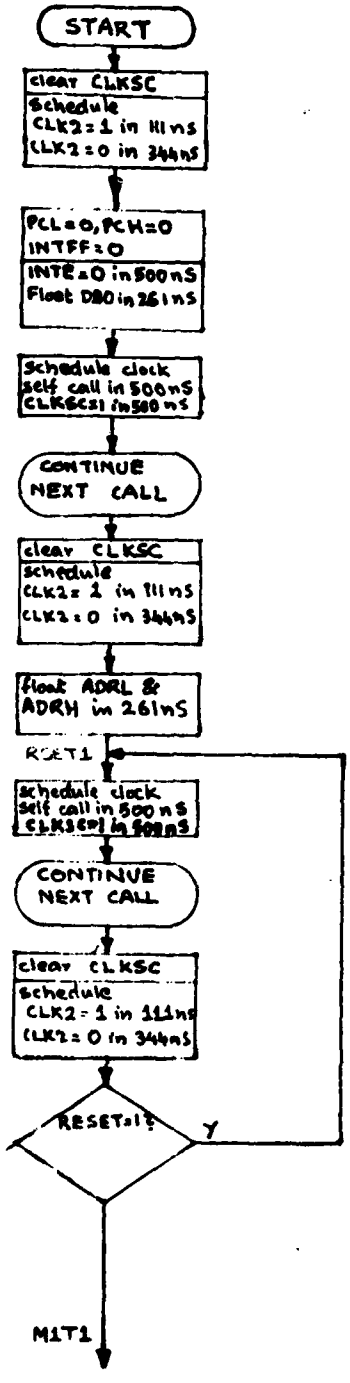

CSL Chip Select Logic
START
SROM
YESSS
SRAM
YSSSI
EXIT
EXIT
EXIT
EXIT
EXIT


ROM
START
ADDB - address buffer to simulate delay
SCF - self call flag
AD=ADD?
y
ADDB = AD in 350nS
Self call in 350nS
SCF = 1 in 350 nS
SCCK
Self call?
y
Reset self call flag
SCF = 0
CSCK
CS transition?
NCS = NCS0?
y
CS=0?
y
CS=0?
n
y
HIZ
NEXT
float data out bus in 100 nS
DO = FF in 100 nS
move byte from location indicated by ADDB to DO in 100nS
UPDATE
update inputs
NCS0 = NCS
ADD = AD
update inputs
NCS = NCS0
ADD = AD
EXIT
EXIT

RAM


START
Int. address self call? SCA=1?
Y
Reset SCA
NEX1
Int. write self call? SCDW=1?
Y
Reset SCDW
Move data from DID to location pointed by ADB
NEX2
Address transition?
Y
ADB = address in 350 nS
Schedule internal address self call in 350 nS (ACA)
CSCK
CS transition?
Y
RWCK
RW transition?
Y
NEX3
ROW
CS=0?
Y
N
CS=0?
Y
NEX4
RW=0?
Y
WRITE
move data from location pointed by ADB to DO
DID=D in 100 nS
Schedule int. write self command in 100 nS
Float DO in 100 nS
UPDATE
NCSO = NCS
RWO = RW
ADO = AD
EXIT

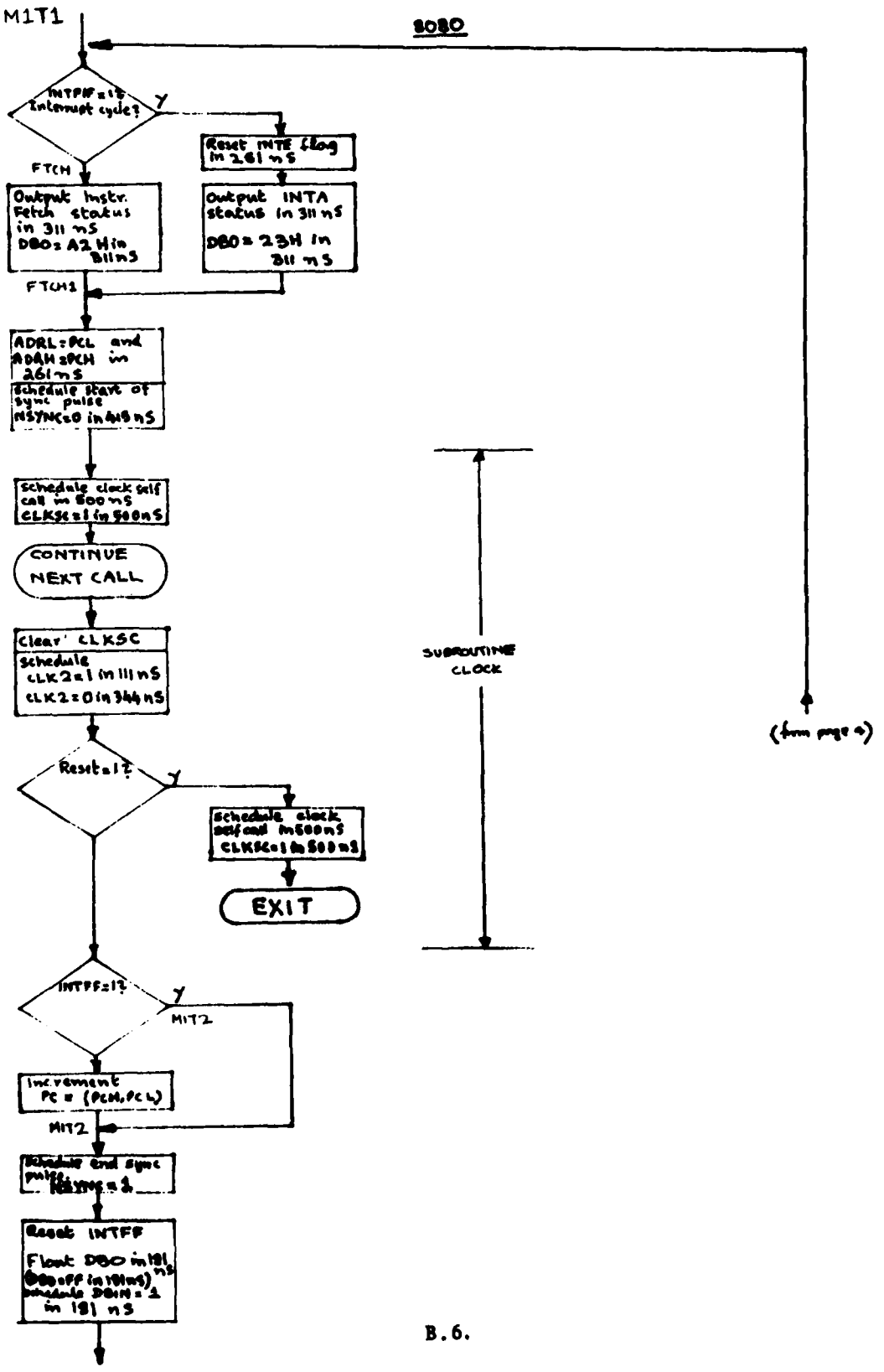
8080

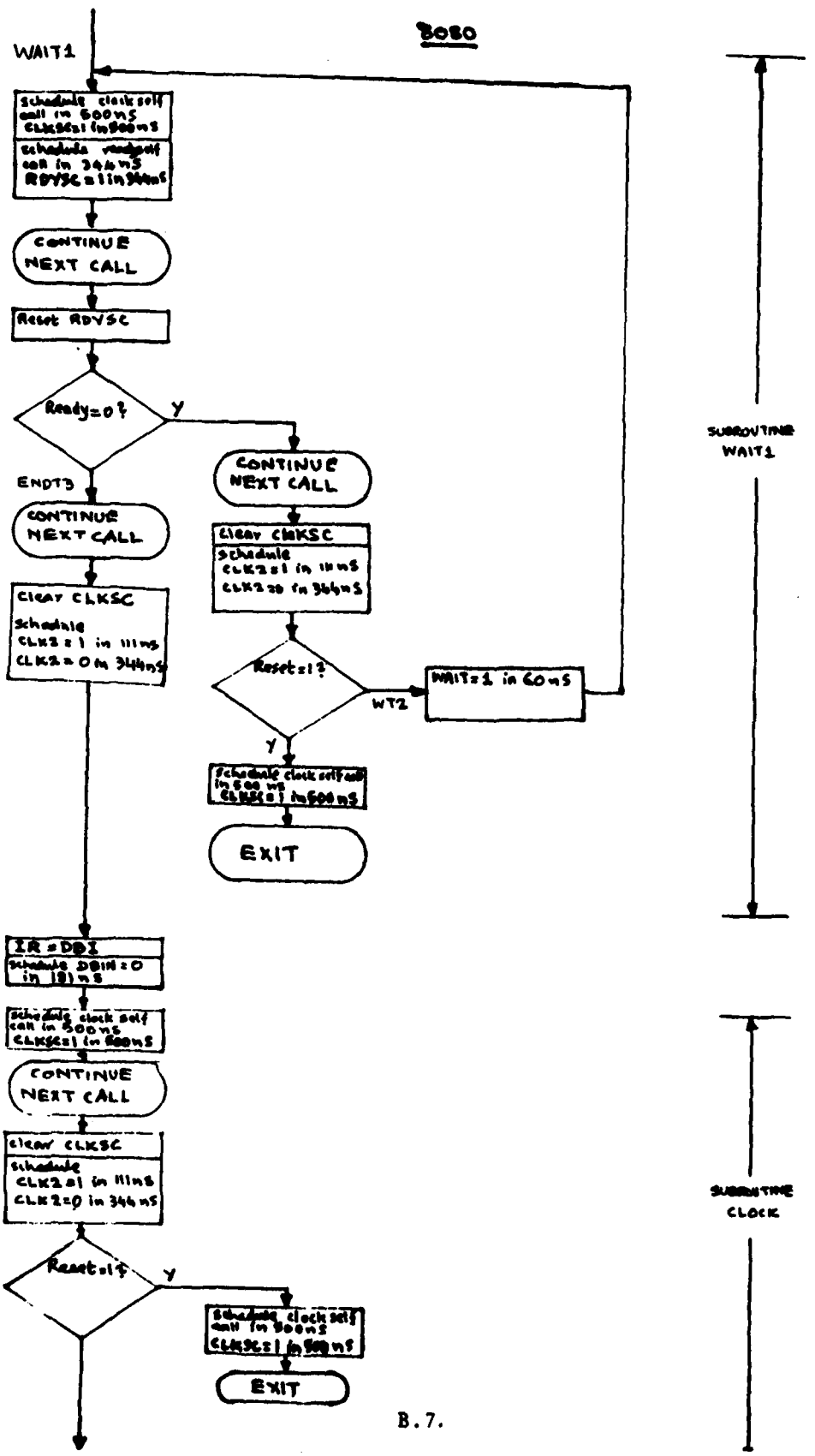


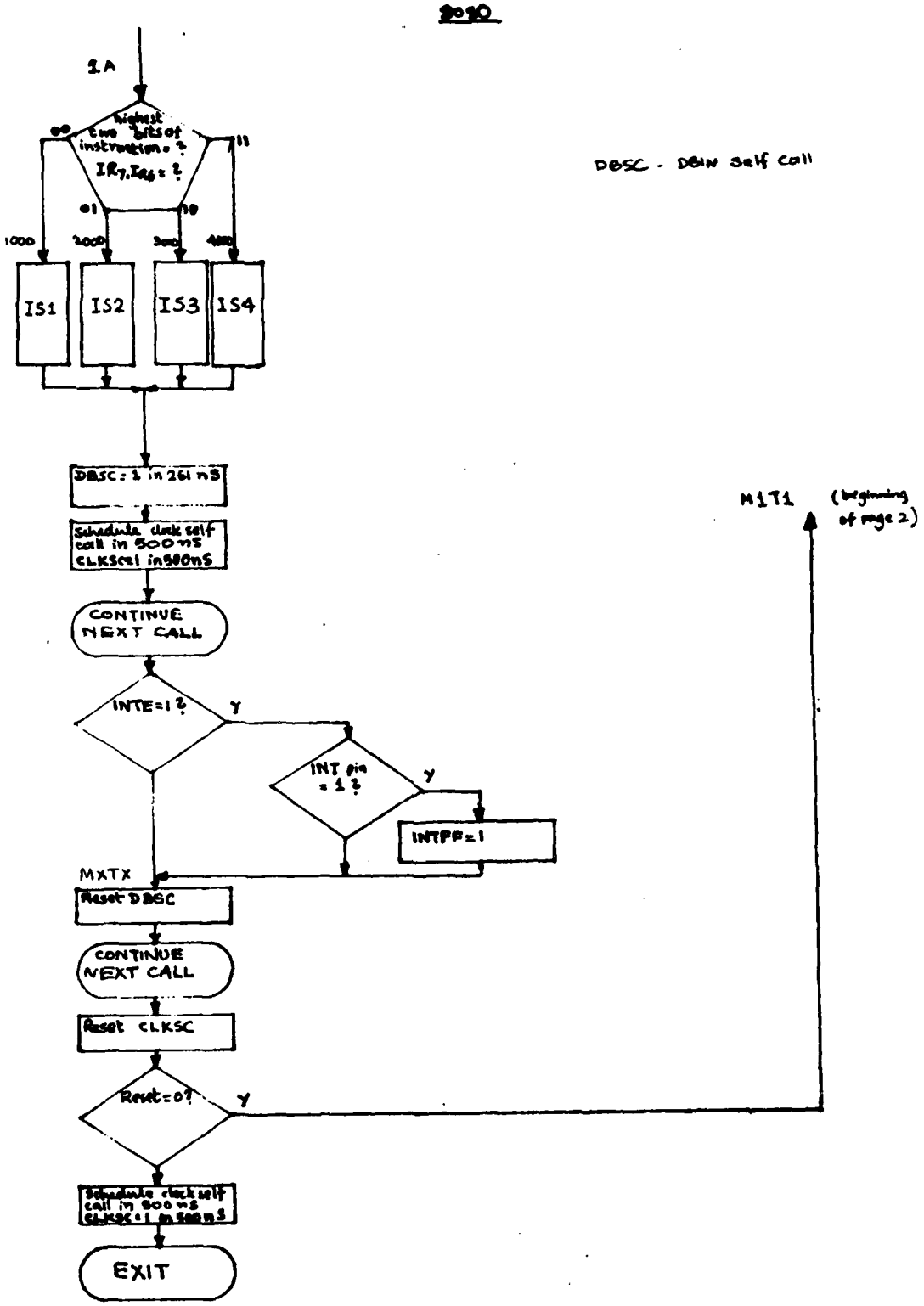
CLKSC - clock Self call
PCL - Progamm Counter low byte
PCH - " high byte

INTFF - Interrupt Flip Flop
INTE - Interrupt Enable flag

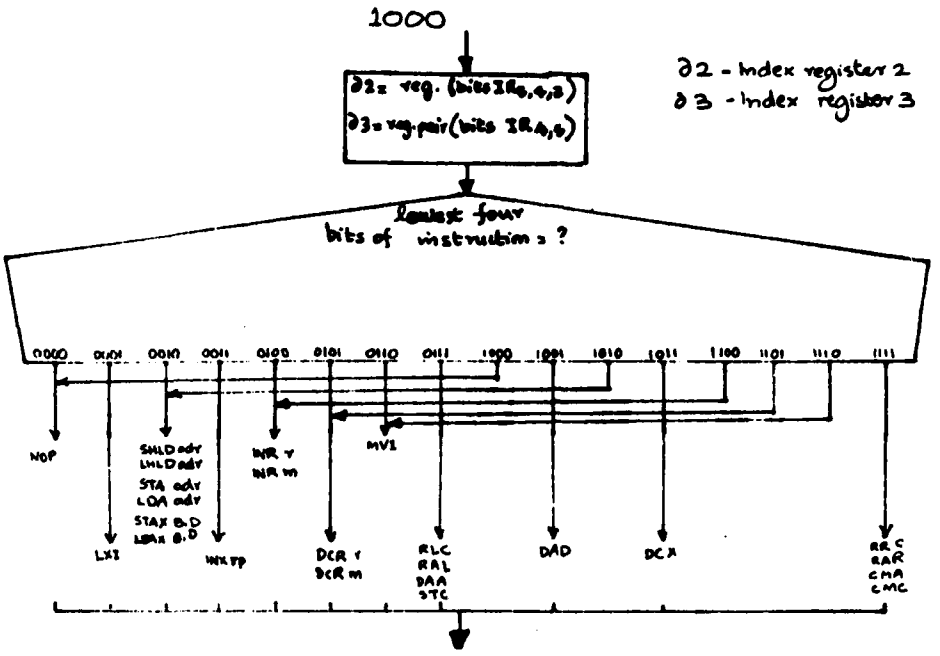
ADRL - Address low byte
ADRH - Address high byte

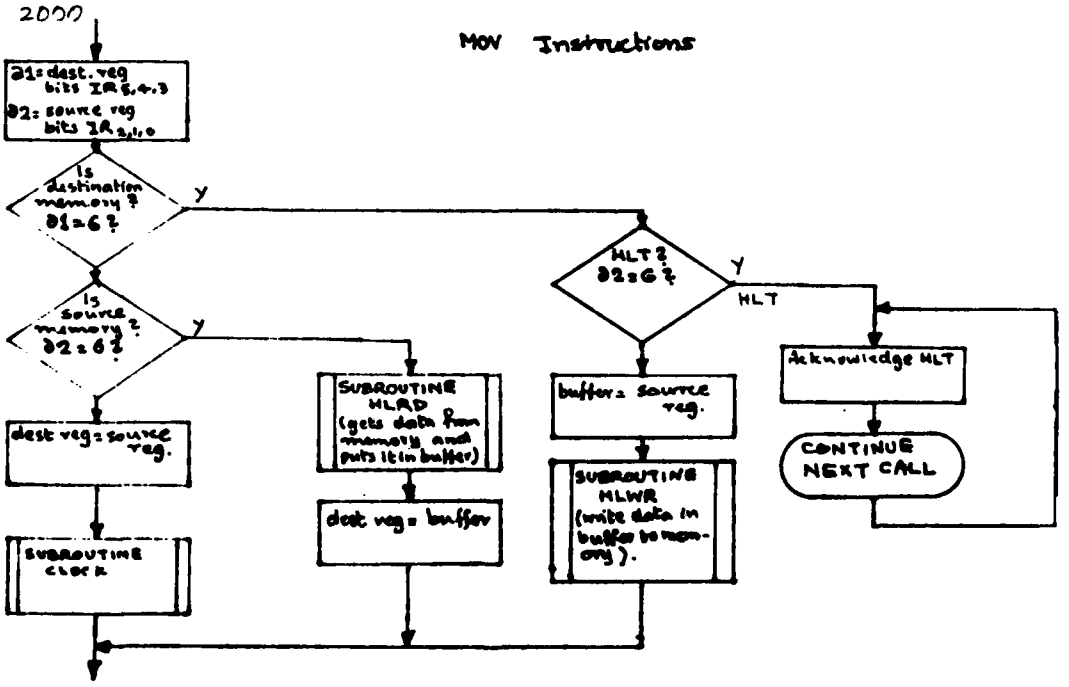

M1T1
8080
INTPF=1? Interrupt cycle?
Y
FTCH
Reset INTE flag in 261 nS
Output Instr. Fetch status in 311 nS DBO= A2 Hin 311nS
Output INTA status in 311 nS DBO= 23H in 311 nS
FTCH1
ADRL=PCL and ADRH=PCH in 261 nS
schedule start of sync pulse MSYNC=0 in 416 nS
schedule clock self call in 500 nS CLKSC=1 in 500nS
CONTINUE NEXT CALL
Clear CLKSC
schedule CLK2=1 in 111 nS CLK2=0 in 344 nS
Reset=1?
schedule clock self call in 500nS CLKSC=1 in 500 nS
EXIT
INTPF=1?
M1T2
Increment PC = (PCH, PCL)
M1T2
Schedule end sync pulse MSYNC=1
Reset INTFF
Float DBO in 181 nS DBO off in 181nS schedule DBIN=1 in 181 nS
SUBROUTINE CLOCK
(from page 4)


BOBO
WAIT1
Schedule clock self call in 500ns CLKSC=1 in 500ns
schedule vsync self call in 344ns RDYSC=1 in 344ns
CONTINUE NEXT CALL
Reset RDYSC
Ready=0?
Y
ENDT3
CONTINUE NEXT CALL
CONTINUE NEXT CALL
Clear CLKSC
Schedule CLK2=1 in 111ns CLK2=0 in 344ns
Clear CLKSC
Schedule CLK2=1 in 1ns CLK2=0 in 344ns
Reset=1?
WT2
WAIT=1 in 60ns
Y
Schedule clock self call in 500 ns CLKSC=1 in 500 ns
EXIT
IR=DBI
Schedule DBIN=0 in 191ns
Schedule clock self call in 500ns CLKSC=1 in 500ns
CONTINUE NEXT CALL
clear CLKSC
Schedule CLK2=1 in 111ns CLK2=0 in 344 ns
Reset=1?
Y
Schedule clock self call in 500ns CLKSC=1 in 500 ns
EXIT
SUBROUTINE WAIT1
SUBROUTINE CLOCK

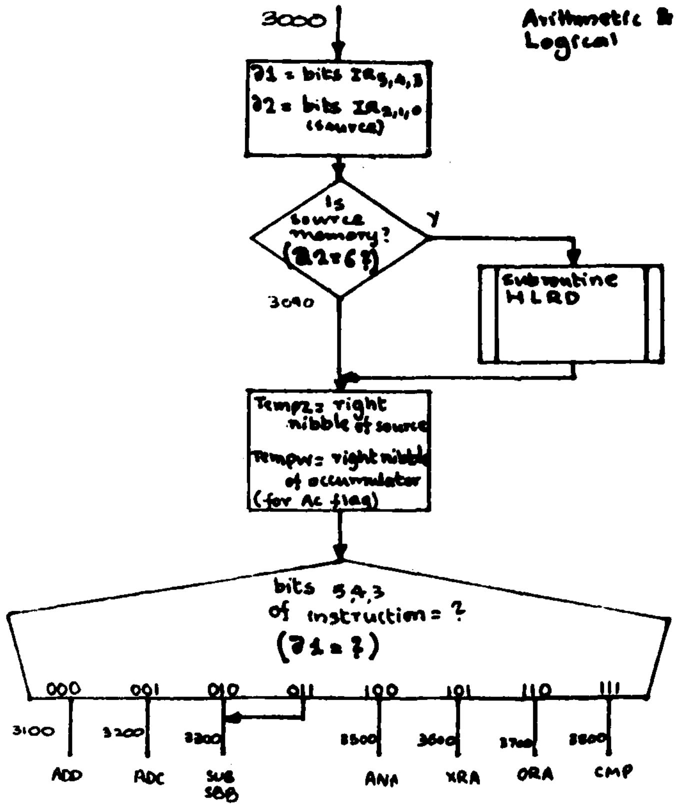


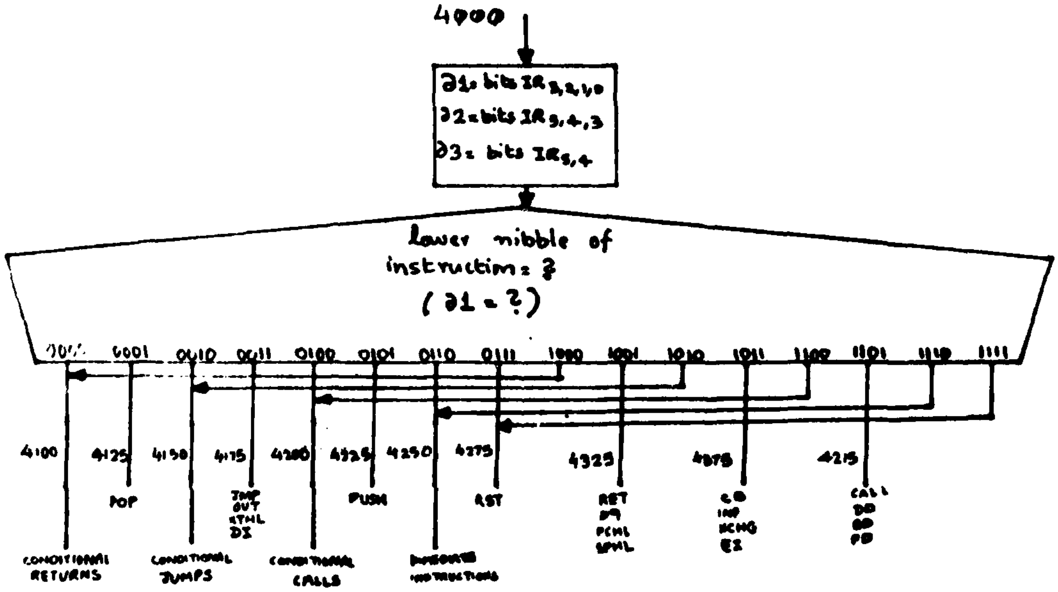
DBSC - DBIN Self Call


1000
∂2= reg. (bits IR5,4,3)
∂3= reg.pair(bits IR5,4)
∂2 - Index register 2
∂3 - Index register 3
Last four bits of instruction = ?
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
NOP
LXI
SHLD adr
LHLD adr
STA adr
LDA adr
STAX B/D
LDAX B/D
INX rp
INR r
INR m
DCR r
DCR m
MVI
RLC
RAL
DAA
STC
DAD
DCX
RRC
RAR
CMA
CMC

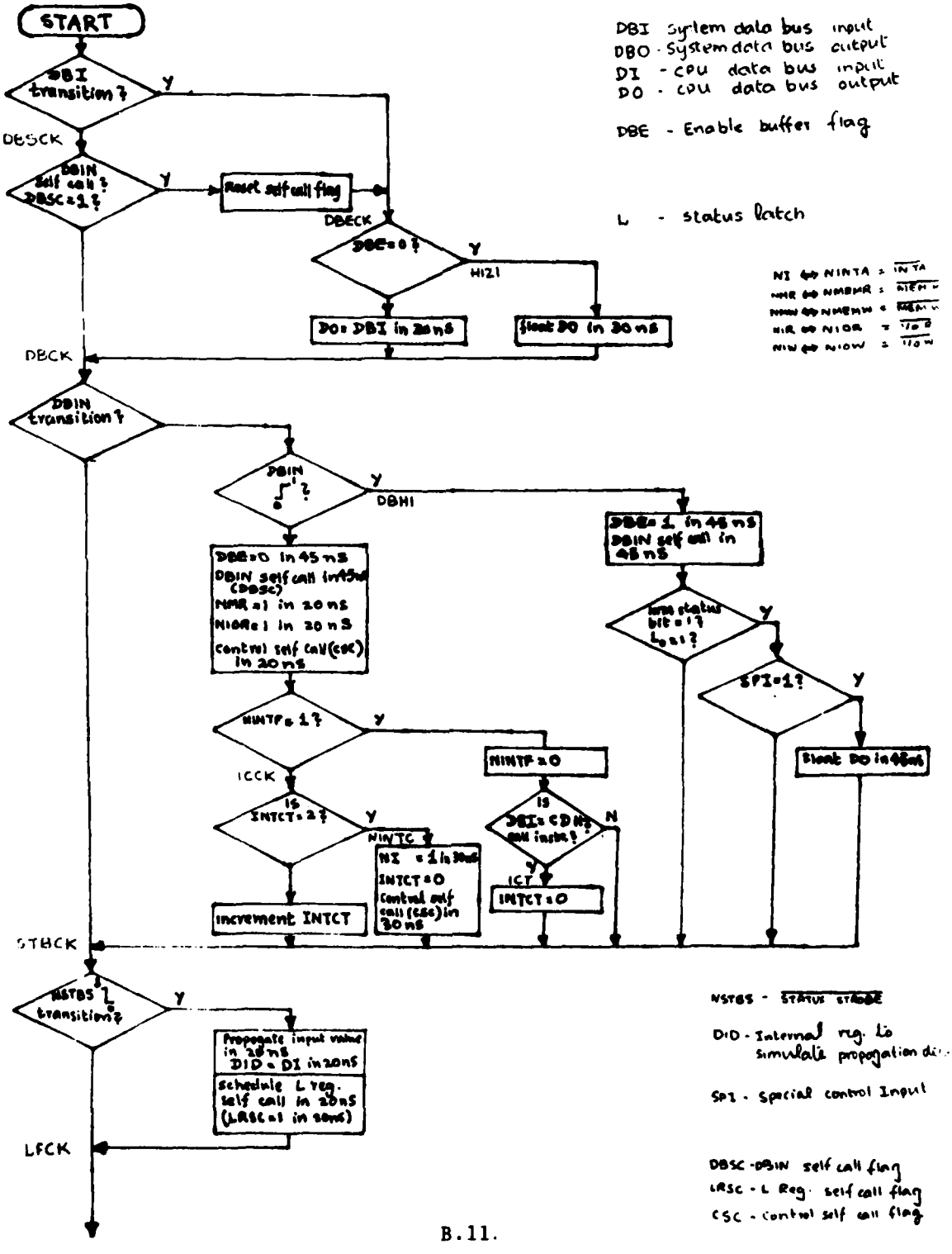

2000
MOV Instructions
∂1= dest. reg bits IR5,4,3
∂2= source reg bits IR2,1,0
Is destination memory? ∂1=6?
Y
Is source memory? ∂2=6?
Y
dest reg= source reg.
SUBROUTINE CLOCK
SUBROUTINE HLRD (gets data from memory and puts it in buffer)
dest reg= buffer
HLT? ∂2=6?
Y
HLT
buffer= source reg.
SUBROUTINE HLWR (write data in buffer to memory).
Acknowledge HLT
CONTINUE NEXT CALL

8080


3000
Arithmetic &
Logical
∂1 = bits IR5,4,3
∂2 = bits IR2,1,0 (source)
Is source memory? (∂2=6?)
Y
Subroutine HLRD
3040
Temp2 = right nibble of source
Tempw = right nibble of accumulator (for Ac flag)
bits 5,4,3 of instruction = ?
(∂1 = ?)
000
001
010
011
100
101
110
111
3100
3200
3300
3500
3600
3700
3800
ADD
ADC
SUB
SBB
ANA
XRA
ORA
CMP


4000
∂1 = bits IR3,2,1,0
∂2 = bits IR5,4,3
∂3 = bits IR5,4
Lower nibble of instruction = ?
(∂1 = ?)
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
4100
4125
4150
4175
4200
4225
4250
4275
4325
4375
4215
POP
PUSH
RST
RET
CONDITIONAL RETURNS
CONDITIONAL JUMPS
CONDITIONAL CALLS
IMMEDIATE INSTRUCTIONS

8228



DBI System data bus input
DBO - System data bus output
DI - CPU data bus input
DO - CPU data bus output

DBE - Enable buffer flag

L - status latch

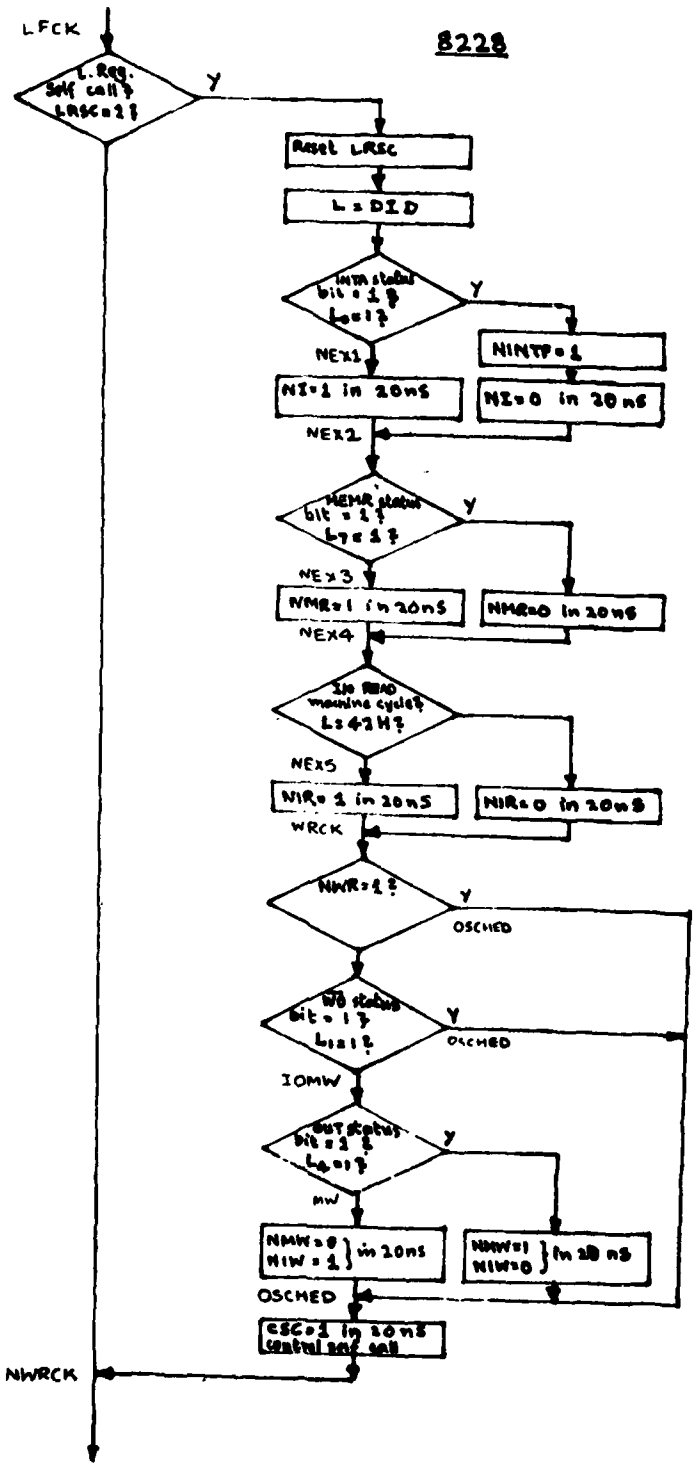
NI ⇔ NINTA = $\overline{\text{INTA}}$
NMR ⇔ NMEMR = $\overline{\text{MEMR}}$
NMW ⇔ NMEMW = $\overline{\text{MEMW}}$
NIR ⇔ NIOR = $\overline{\text{I/O R}}$
NIW ⇔ NIOW = $\overline{\text{I/O W}}$

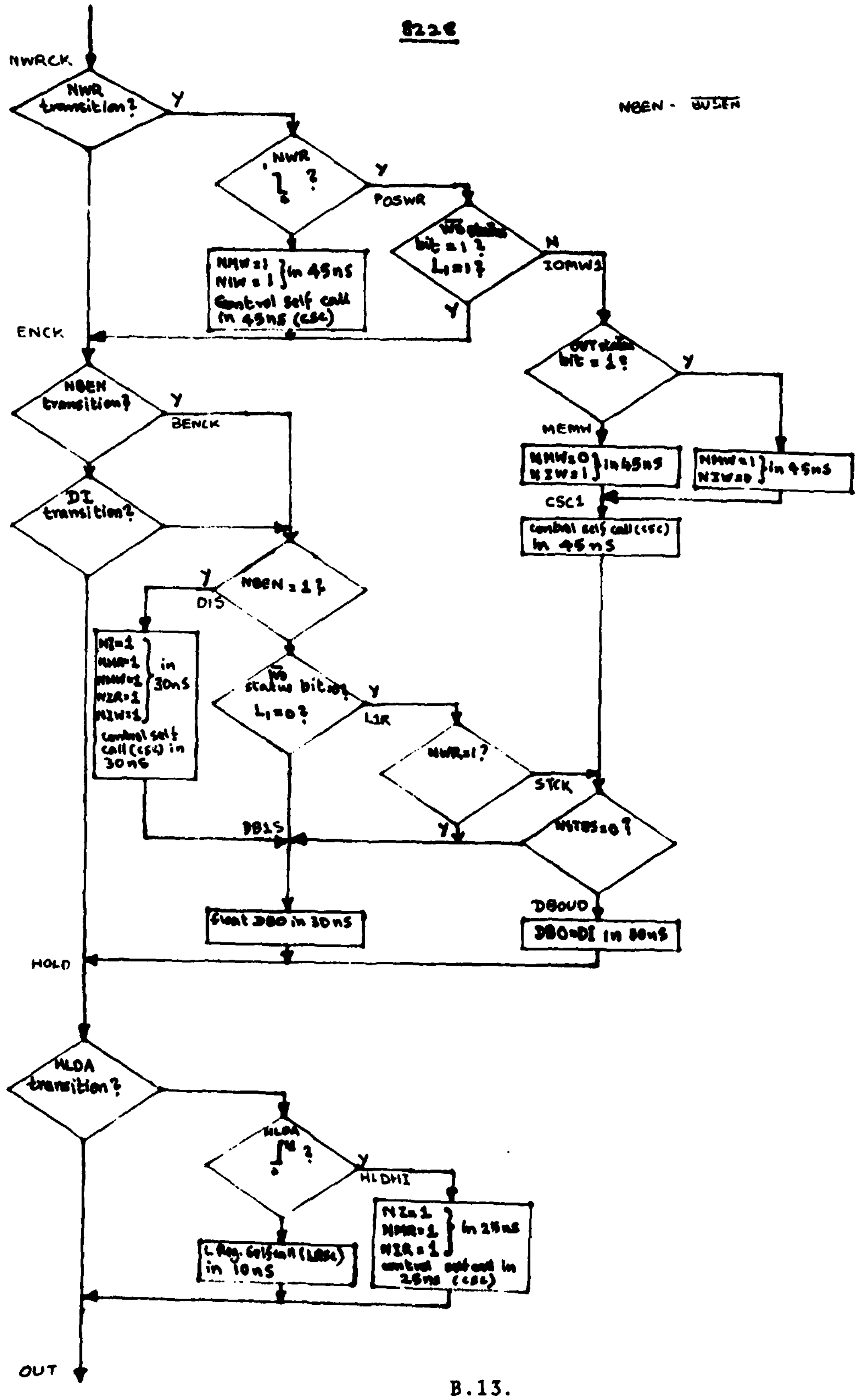
NSTBS - $\overline{\text{STATUS STROBE}}$

DID - Internal reg. to simulate propagation dir.

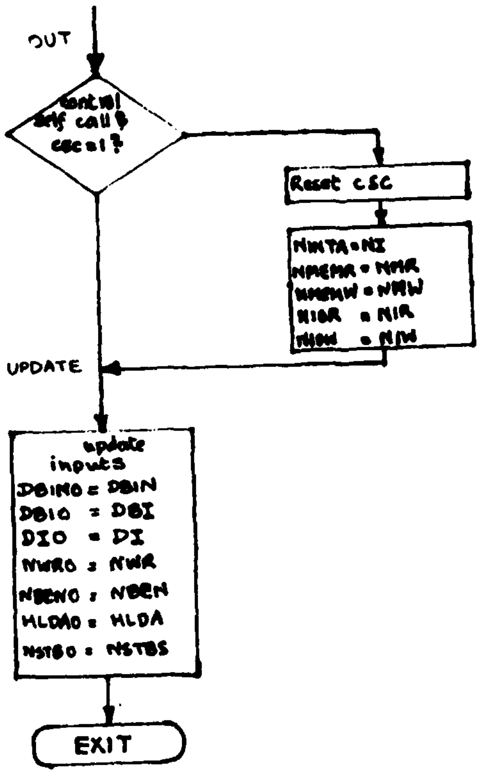
SPI - special control Input

DBSC - DBIN self call flag
LRSC - L Reg. self call flag
CSC - control self call flag

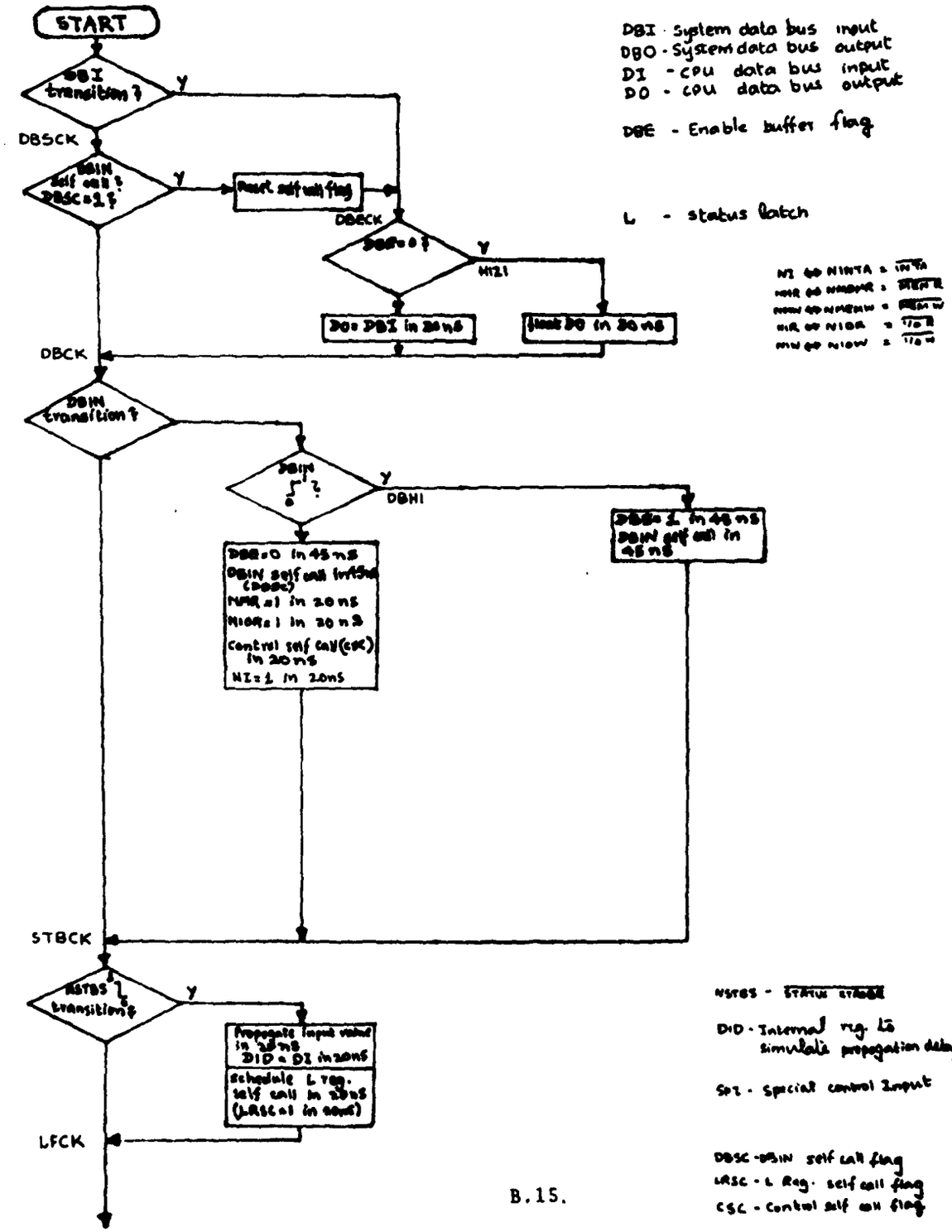

8228
LFCK
L. Req. Self call? LRSC=1?
Y
Reset LRSC
L=DID
INTA status bit=1? L=1?
Y
NINTP=1
NEX1
NI=1 in 20nS
NI=0 in 20nS
NEX2
MEMR status bit=1? L7=1?
Y
NEX3
NMR=1 in 20nS
NMR=0 in 20nS
NEX4
IO READ machine cycle? L=42H?
NEX5
NIR=1 in 20nS
NIR=0 in 20nS
WRCK
NWR=1?
Y
OSCHED
MB status bit=1? L=1?
Y
OSCHED
IOMW
OUT status bit=1? L6=1?
Y
MW
NMW=0 NIW=1 in 20nS
NMW=1 NIW=0 in 20nS
OSCHED
CSC=1 in 20nS (control self call)
NWRCK


8228
NBEN - BUSEN
NWRCK
NWR transition?
Y
POSWR
N
IOMW1
ENCK
NBEN transition?
BENCK
DI transition?
NBEN = 1?
DIS
MEMW
CSC1
control self call (CSC) in 45 nS
L1R
NWR=1?
STCK
DB1S
DBOUD
float DBO in 30 nS
DBO=DI in 30nS
HOLD
HLDA transition?
HLDHI
OUT

8228


OUT
control
self call &
csc=1?
Reset csc
NINTA = NI
NMEMR = NMR
NMEMW = NMW
NIOR = NIR
NIOW = NIW
UPDATE
update
inputs
DBINO = DBIN
DBIO = DBI
DIO = DI
NWRO = NWR
NBENO = NBEN
HLDAO = HLDA
NSTBO = NSTBS
EXIT

8228 SHORT VERSION



DBI - System data bus input
DBO - System data bus output
DI - CPU data bus input
DO - CPU data bus output

DBE - Enable buffer flag

L - status latch

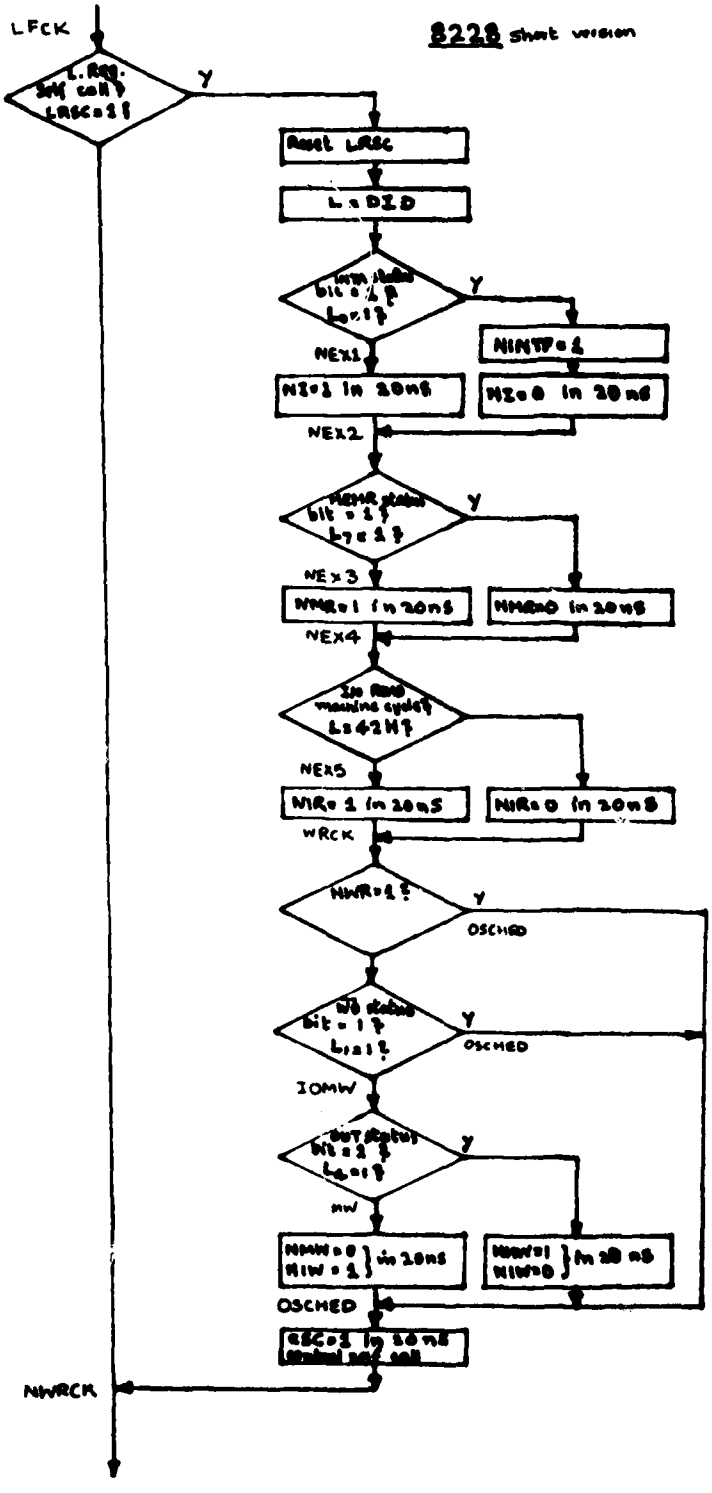
MI ⇔ MINTA = INTA
MRR ⇔ MMEMR = MEMR
MMW ⇔ MMEMW = MEMW
MIR ⇔ MIOR = I/OR
MIW ⇔ MIOW = I/OW

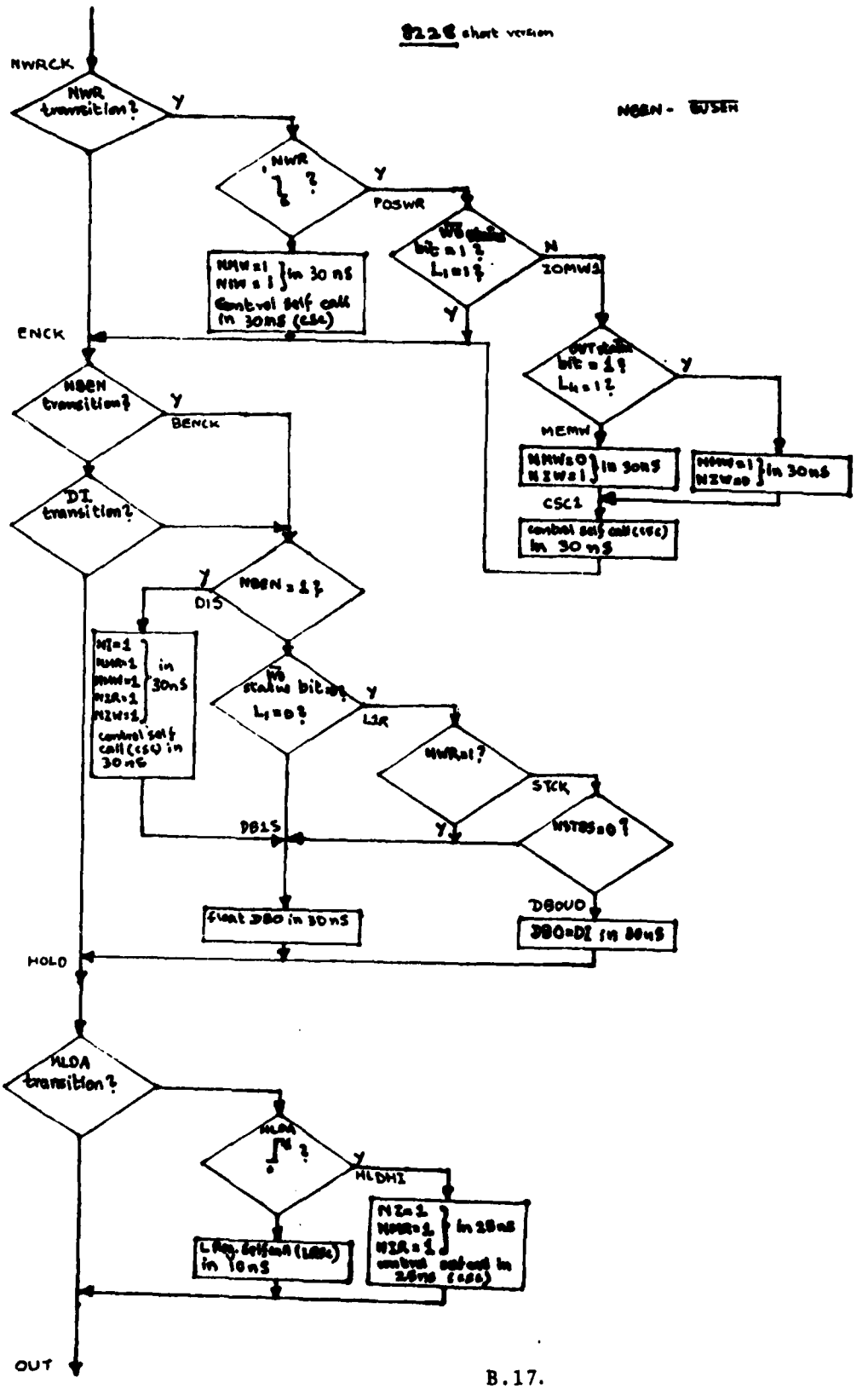
NSTBS - STATUS STROBE

DID - Internal reg. to simulate propagation delay

SPI - special control input

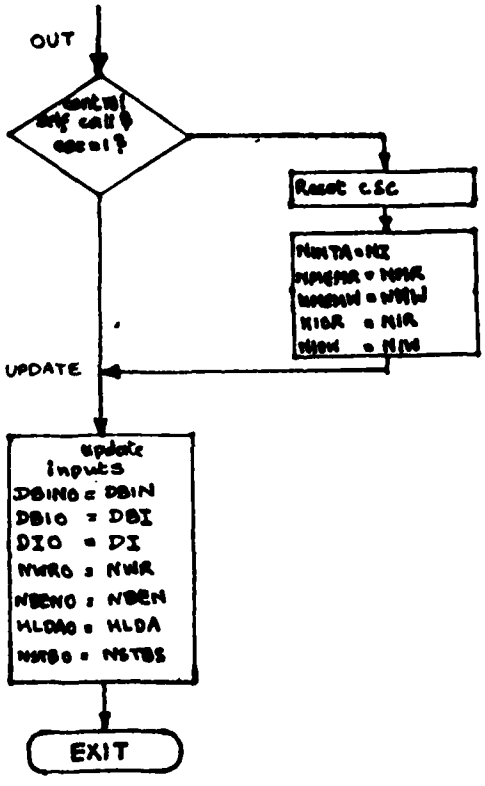
DBSC - DBIN self call flag
LRSC - L Reg. self call flag
CSC - Control self call flag

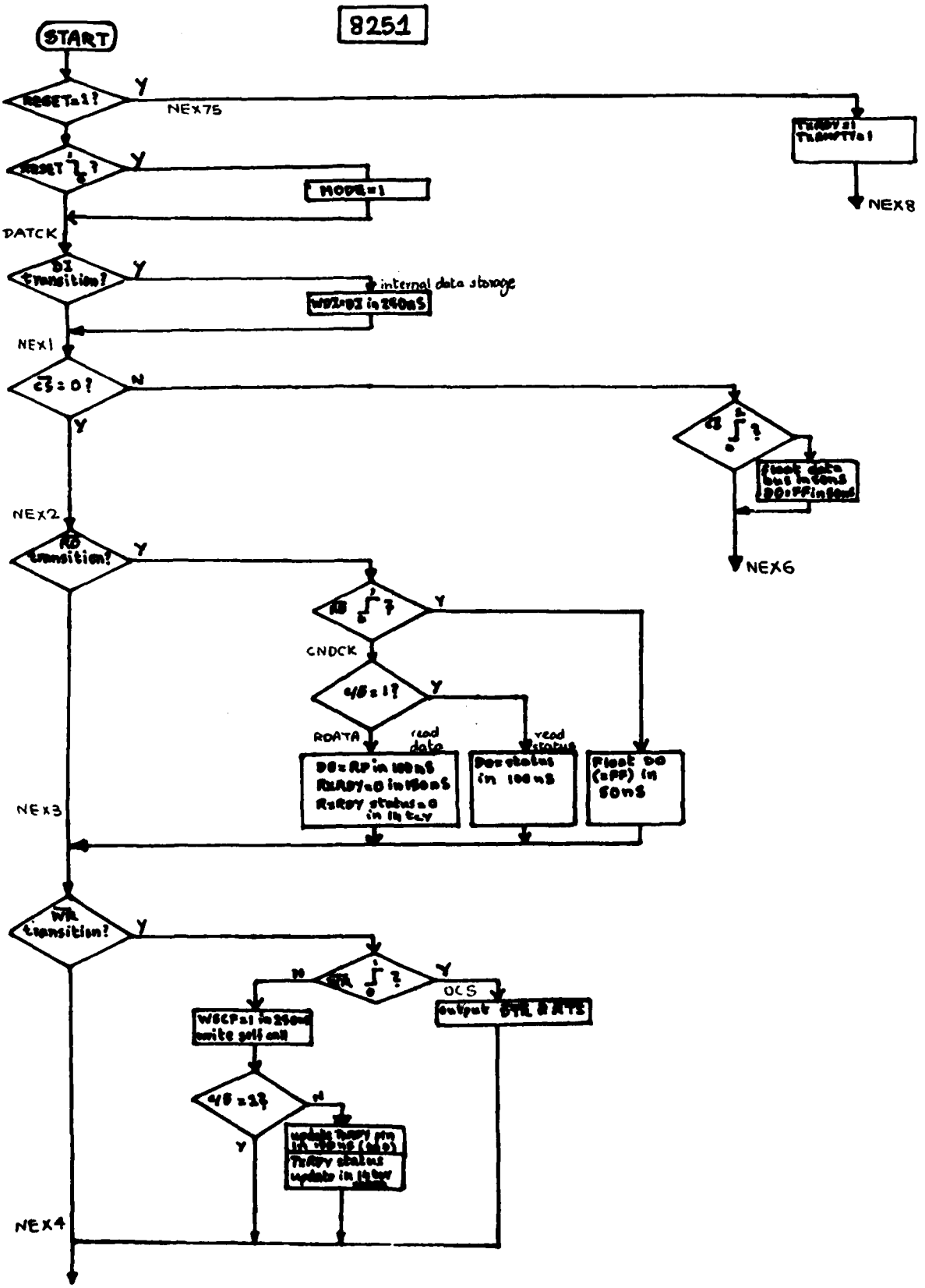

LFCK
8228 short version
L.Reg. Sel call ? LRSC=1 ?
Y
Reset LRSC
L = DID
bit = 1 ?
Y
NEX1
NINTP=1
NI=1 in 20nS
NI=0 in 20nS
NEX2
bit = 1 ?
Y
NEX3
NMR=1 in 20nS
NMR=0 in 20nS
NEX4
NEX5
NIR=1 in 20nS
NIR=0 in 20nS
WRCK
NWR=1 ?
Y
OSCHED
bit = 1 ?
Y
OSCHED
IOMW
bit = 1 ?
Y
NW
NWW=0 NIW=1 in 20ns
NWW=1 NIW=0 in 20 nS
OSCHED
NWRCK

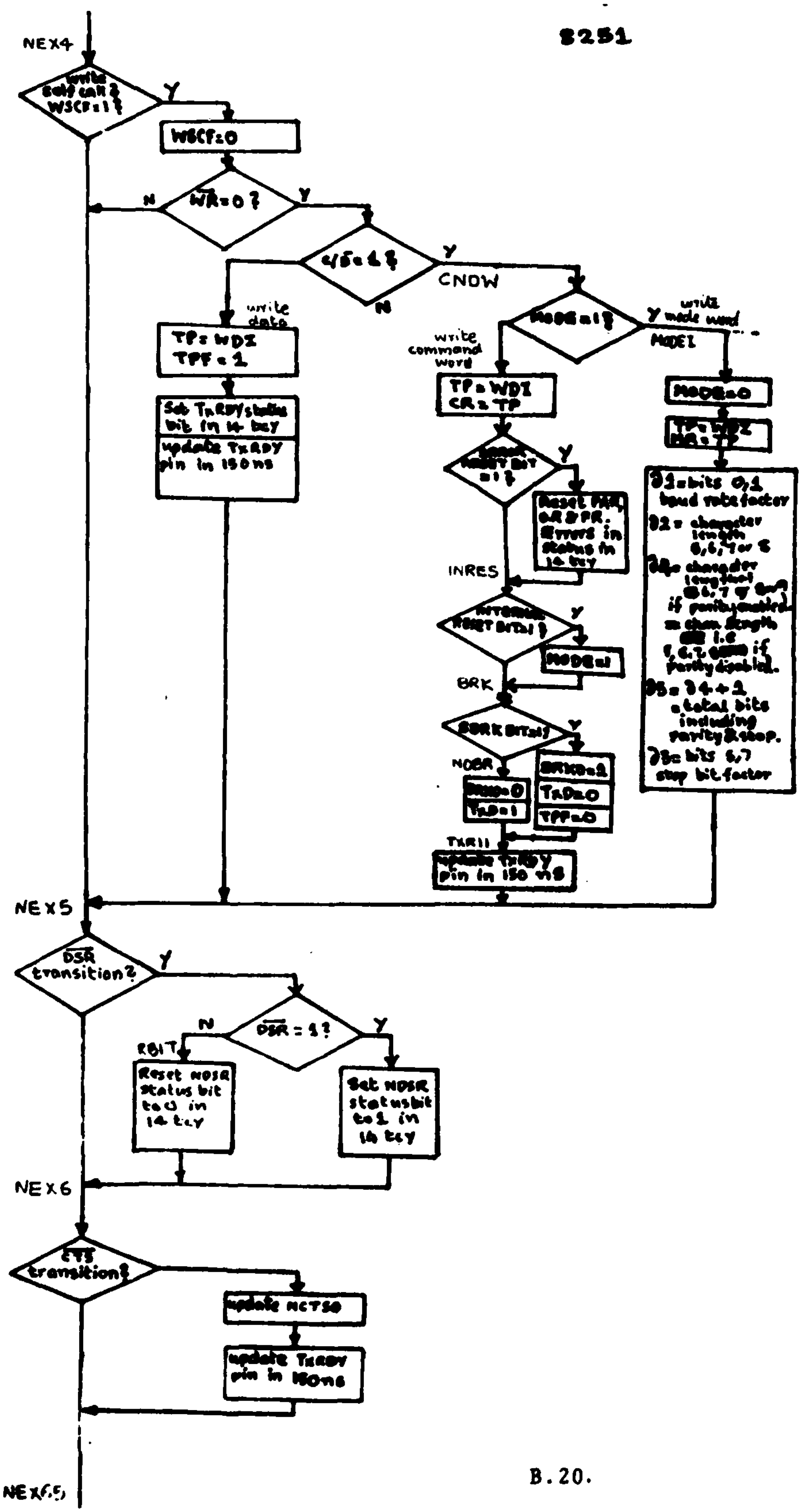


NSEN - $\overline{SUSEN}$

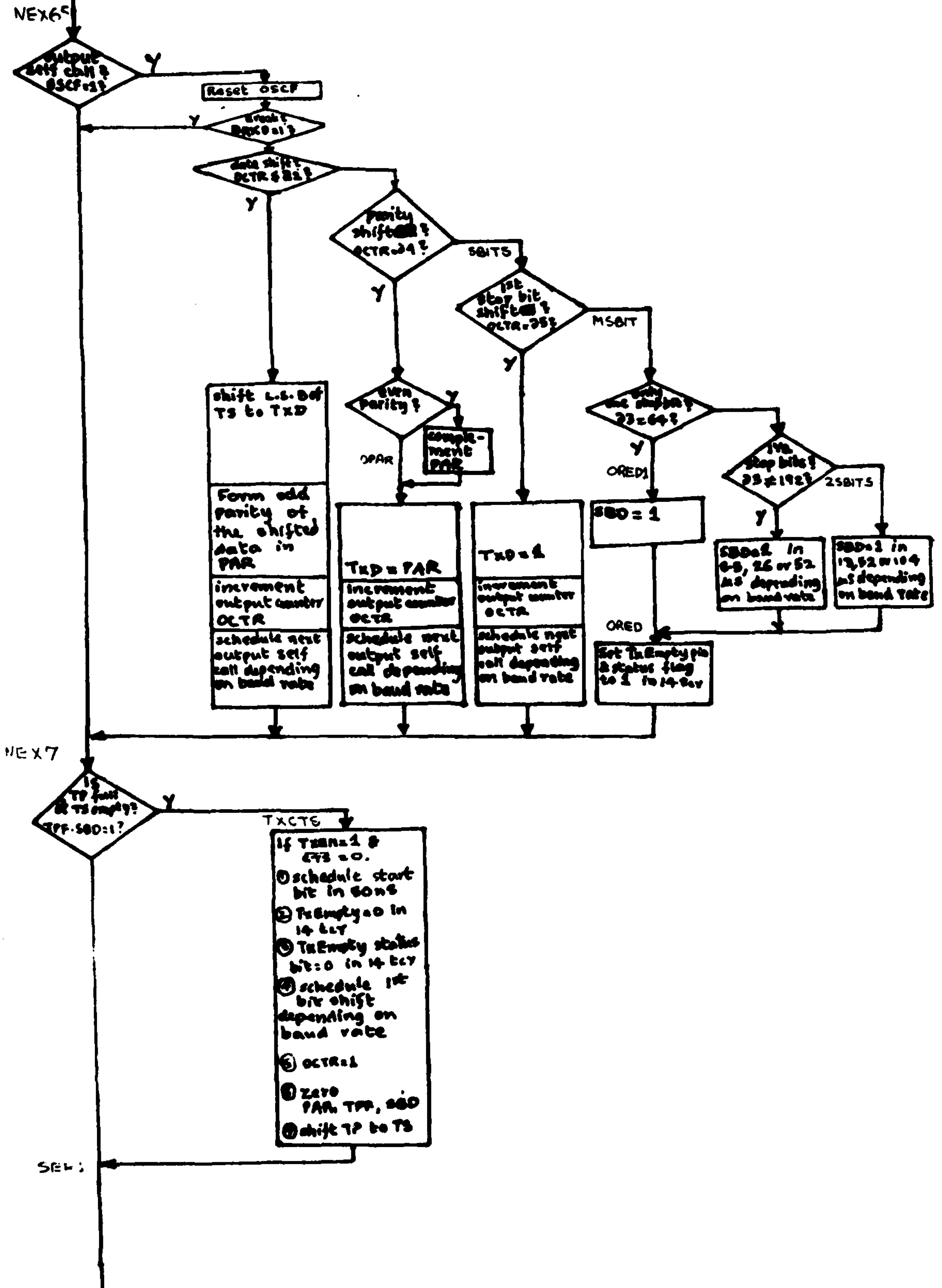
8228 short version

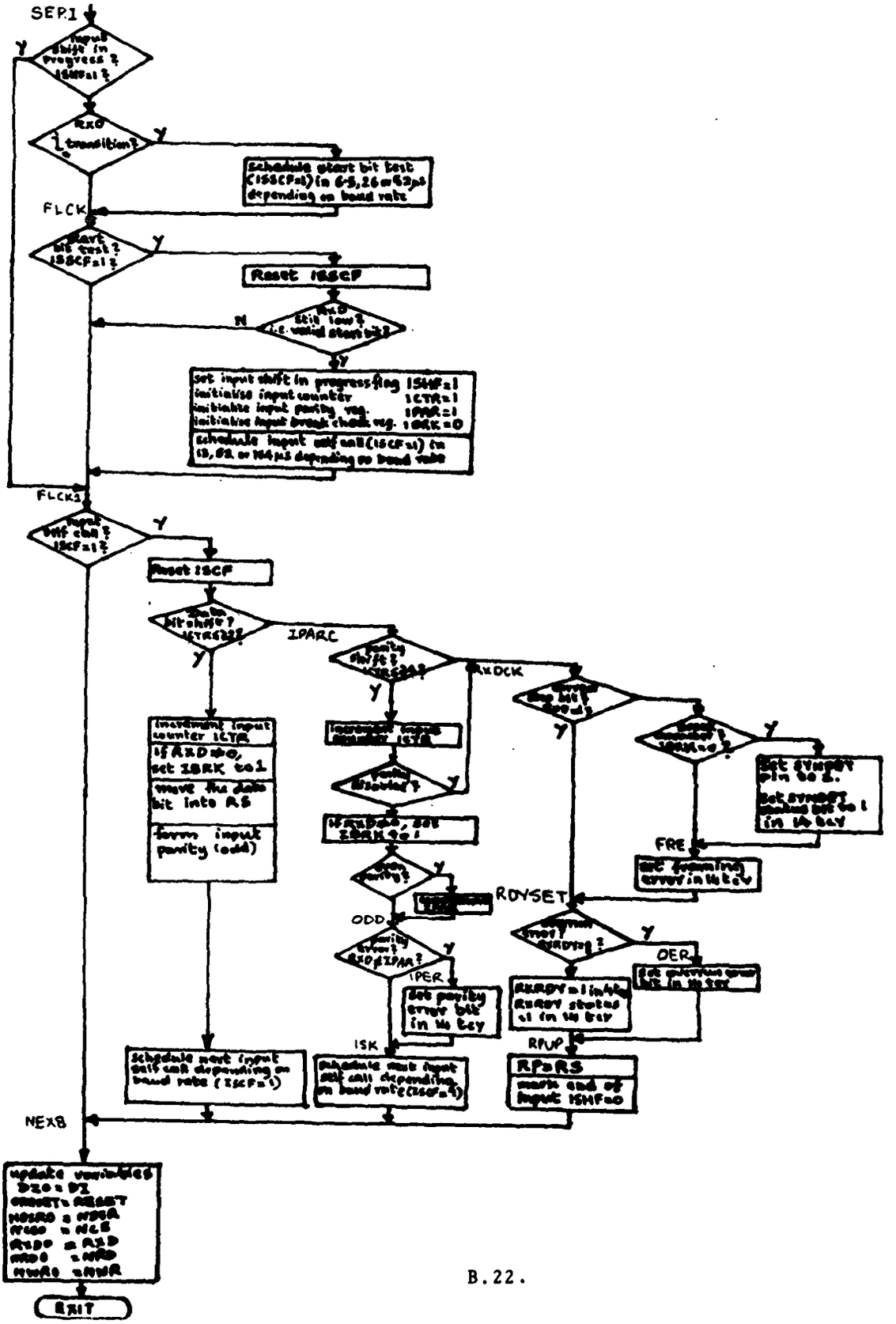


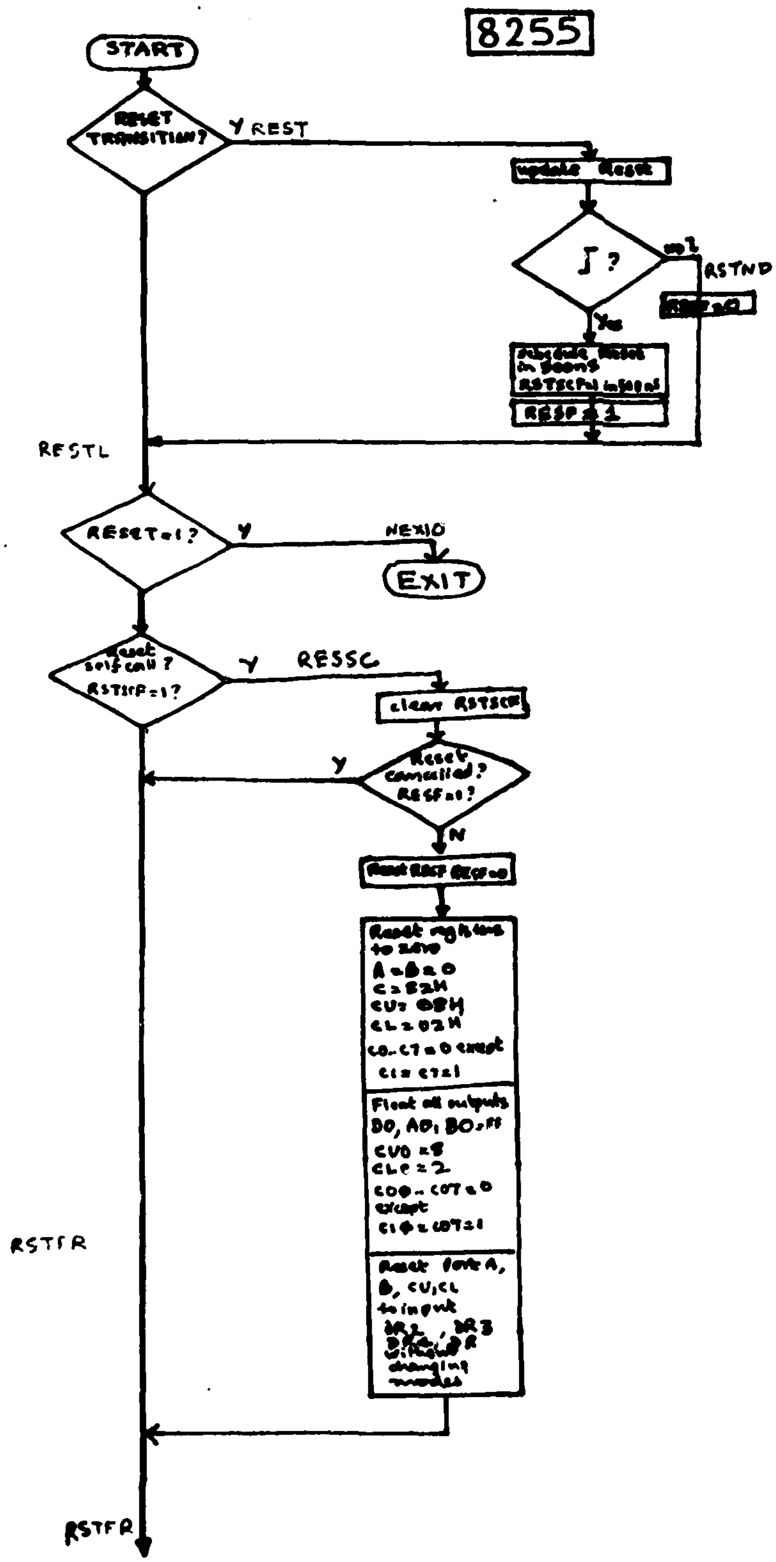

8251
START
RESET=1?
Y
NEX75
NEX8
MODE=1
DATCK
internal data storage
NEX1
N
NEX6
NEX2
NEX3
CNDCK
RDATA
read data
read status
NEX4
OCS

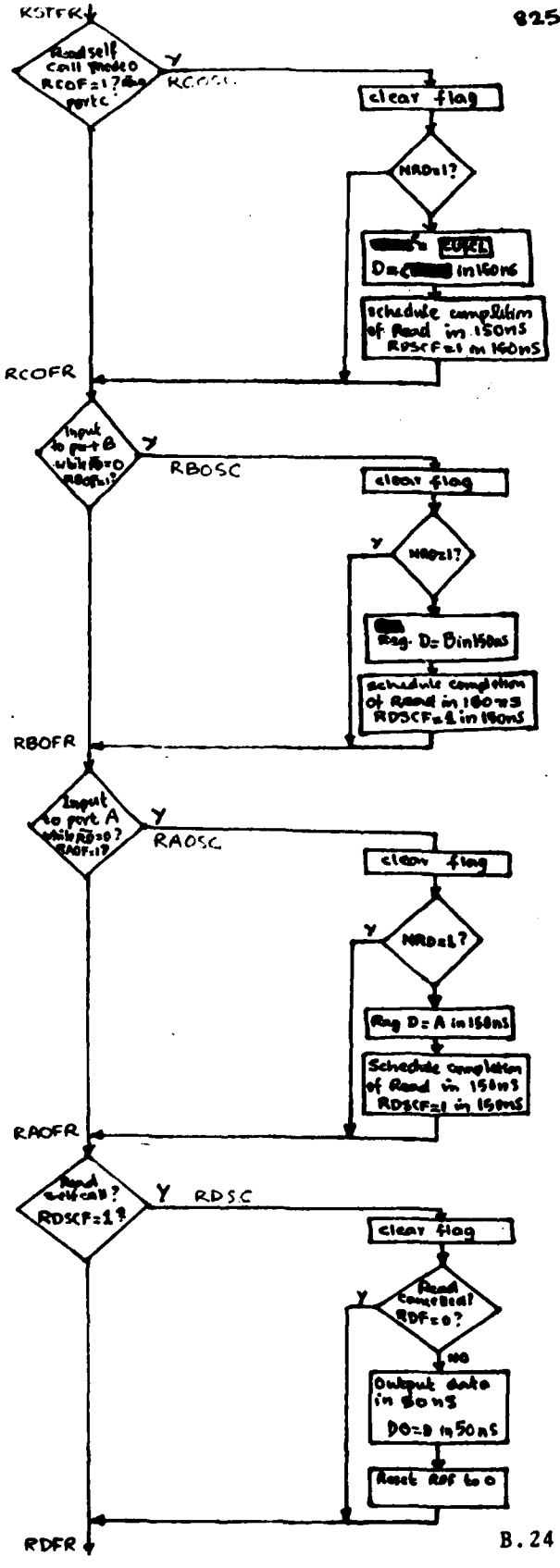

NEX4
WSCF=0
Y
N
CNOW
write data
TP= WDI
TPF = 1
write command word
TP= WDI
CR= TP
MODE=1?
write mode word
MODEI
MODE=0
update TxRDY pin in 150 ns
INRES
BRK
NOBR
TXRII
MODE=1
BRKD=1
TxD=0
TPF=0
TxD=1
NEX5
DSR transition?
RBIT
Reset NDSR status bit to 0 in 14 tcy
Set NDSR status bit to 1 in 14 tcy
NEX6
CTS transition?
update NCTSO
update TxRDY pin in 150ns

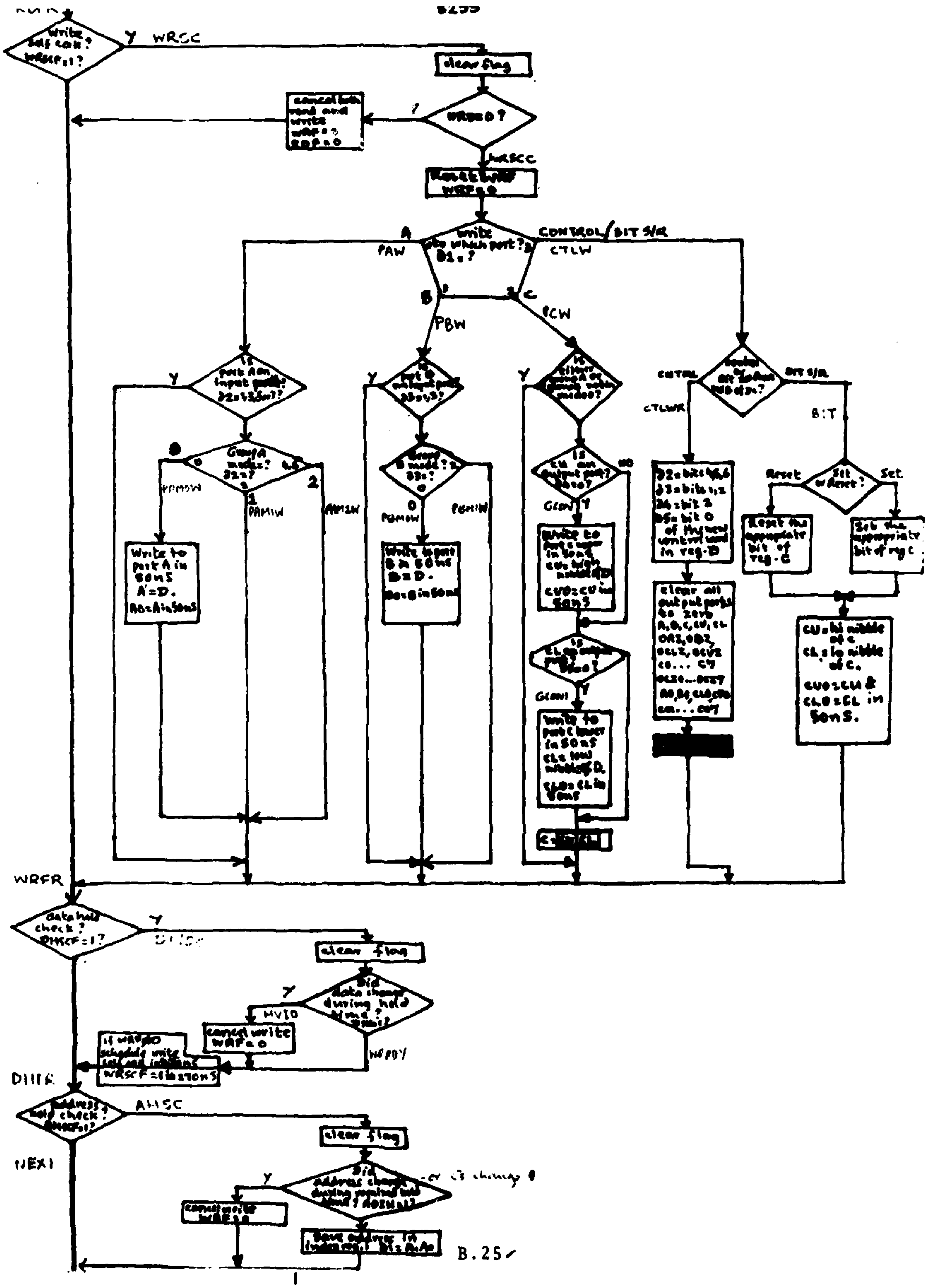
8251

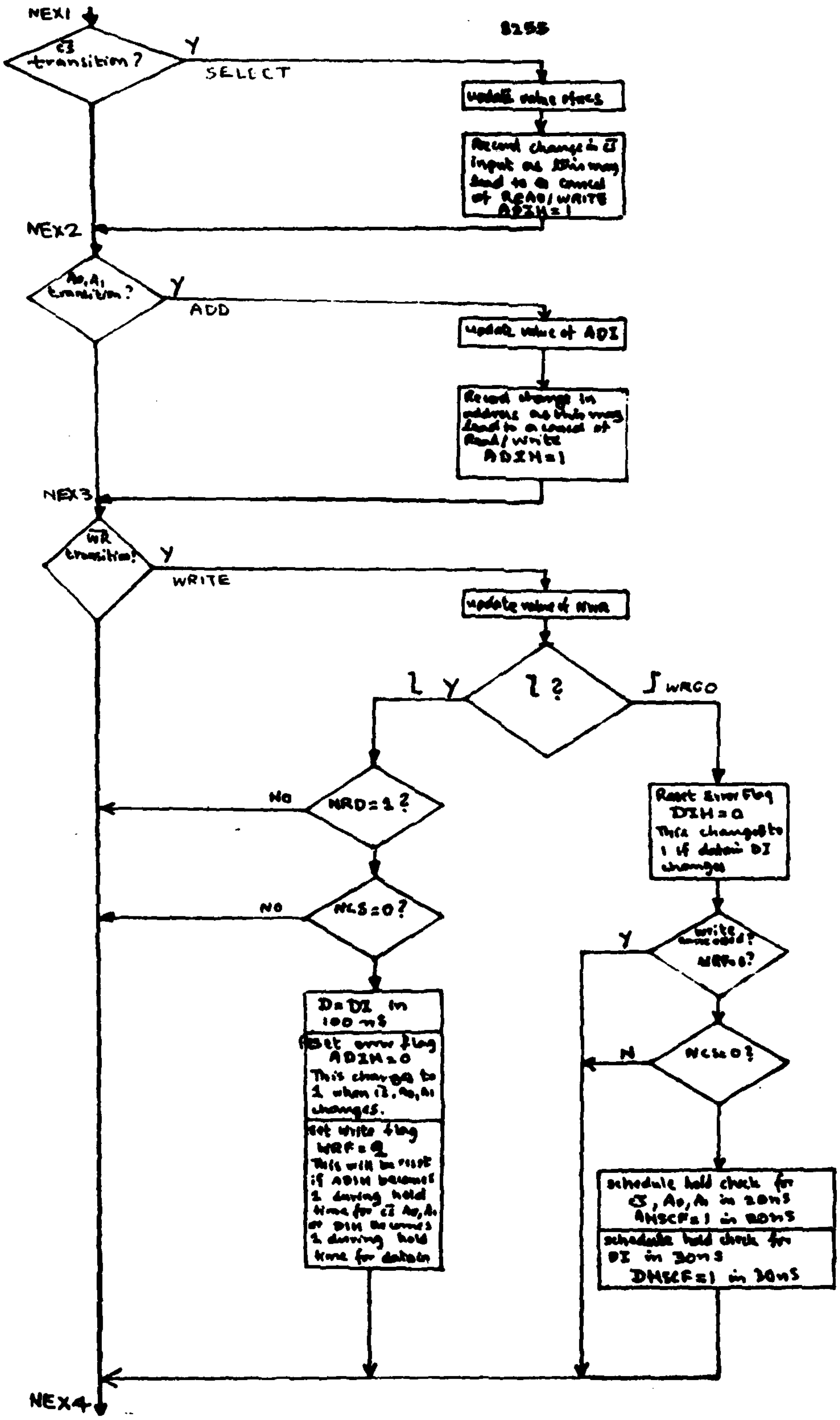

Y
Reset OSCF
Y
Y
SBITS
Y
MSBIT
Y
Y
OPAR
ORED1
Y
2SBITS
Form odd parity of the shifted data in PAR
TxD = PAR
SBD = 1
increment output counter OCTR
schedule next output self call depending on baud rate
ORED
NEX7
Y
TXCTS
schedule start bit in 60nS
schedule 1st bit shift depending on baud rate
zero PAR, TPF, SBD
shift TP to TS

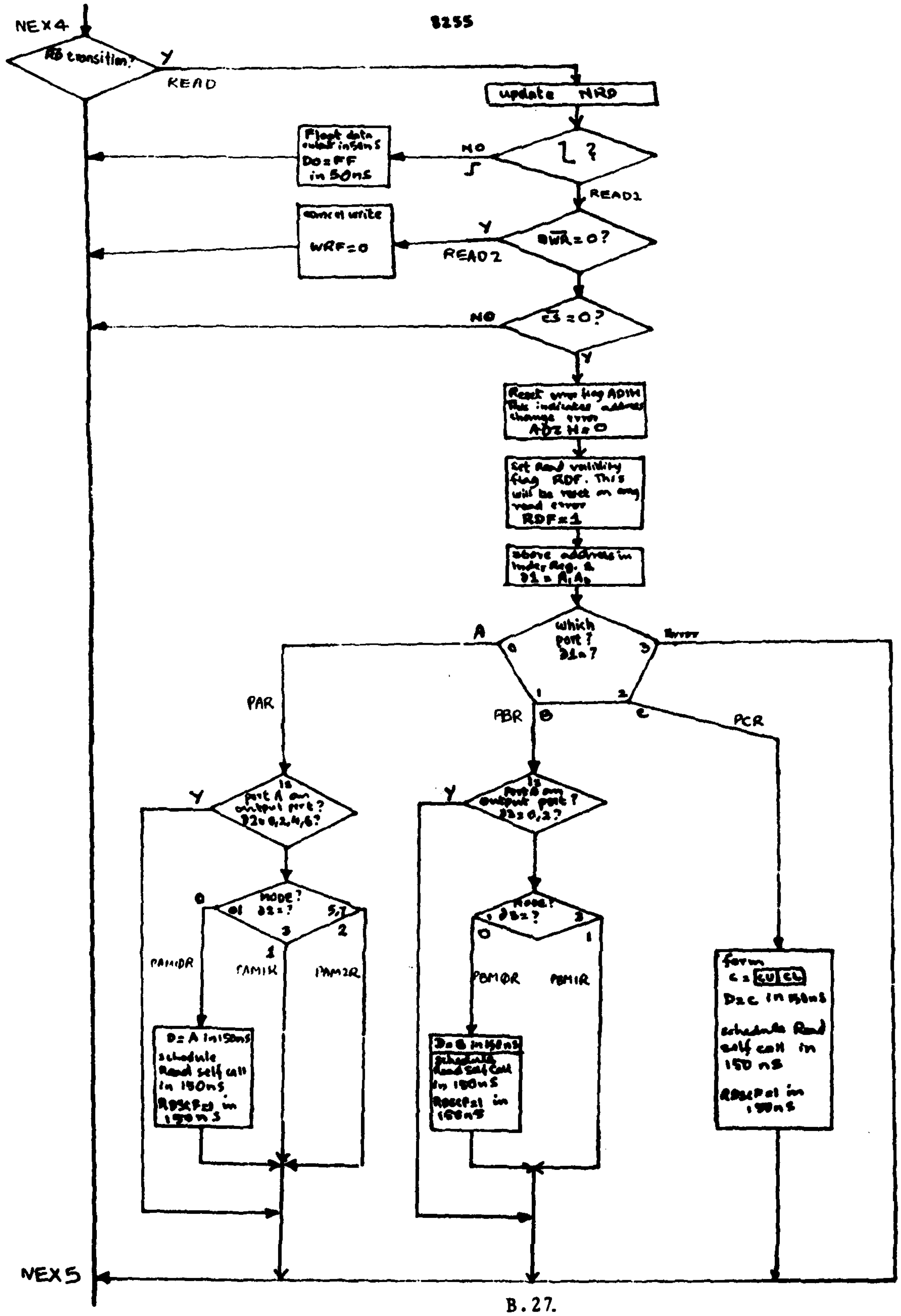

SERI
RxD transition?
schedule start bit test (ISSCF=1) in 6·5, 26 or 52 μs depending on baud rate
FLCK
Reset ISSCF
RxD still low? i.e. valid start bit?
N
Y
set input shift in progress flag ISHF=1
initialise input counter ICTR=1
initialise input parity reg. IPAR=1
initialise input break check reg. IBRK=0
schedule input bit call (ISCF=1) in 13, 52 or 104 μs depending on baud rate
FLCK1
Input bit call? ISCF=1?
Reset ISCF
Data bit? ICTR=2-9?
IPARC
Parity bit? ICTR=10?
RxDCK
Stop bit? ICTR=11?
increment input counter ICTR
if RXD=0, set IBRK to 1
move the data bit into A5
form input parity (add)
increment input parity register ISPR
Parity discarded?
if RXD=0, set IBRK to 1
Stop bit?
Error? ICTR=12?
Set synchr pin to 1.
Set synchr pending bit to 1 in IA key
FRE
set framing error in IA key
RDYSET
ODD
Parity error? RXD≠IPAR?
IPER
set parity error bit in IA key
Character read? RXRDY=1?
OER
set overrun error bit in IA key
RXRDY=1 in IA key
RXRDY status =1 in IA key
ISK
RFUP
schedule next input bit call depending on baud rate (ISCF=1)
schedule next input stop bit call depending on baud rate (ISCF=1)
RP=RS
mark end of input ISHF=0
NEXB
update variables
DZO = D1
CHSET = CHSET
HOSRO = HOSA
HCSO = HCS
RXDO = RXD
HDO = HDO
HWRO = HWR
EXIT


8255
START
RESET TRANSITION?
Y REST
RSTND
RESTL
RESET=1?
Y
NEXIO
EXIT
Y
RESSC
Y
N
Float all outputs
RSTFR
RSTFR

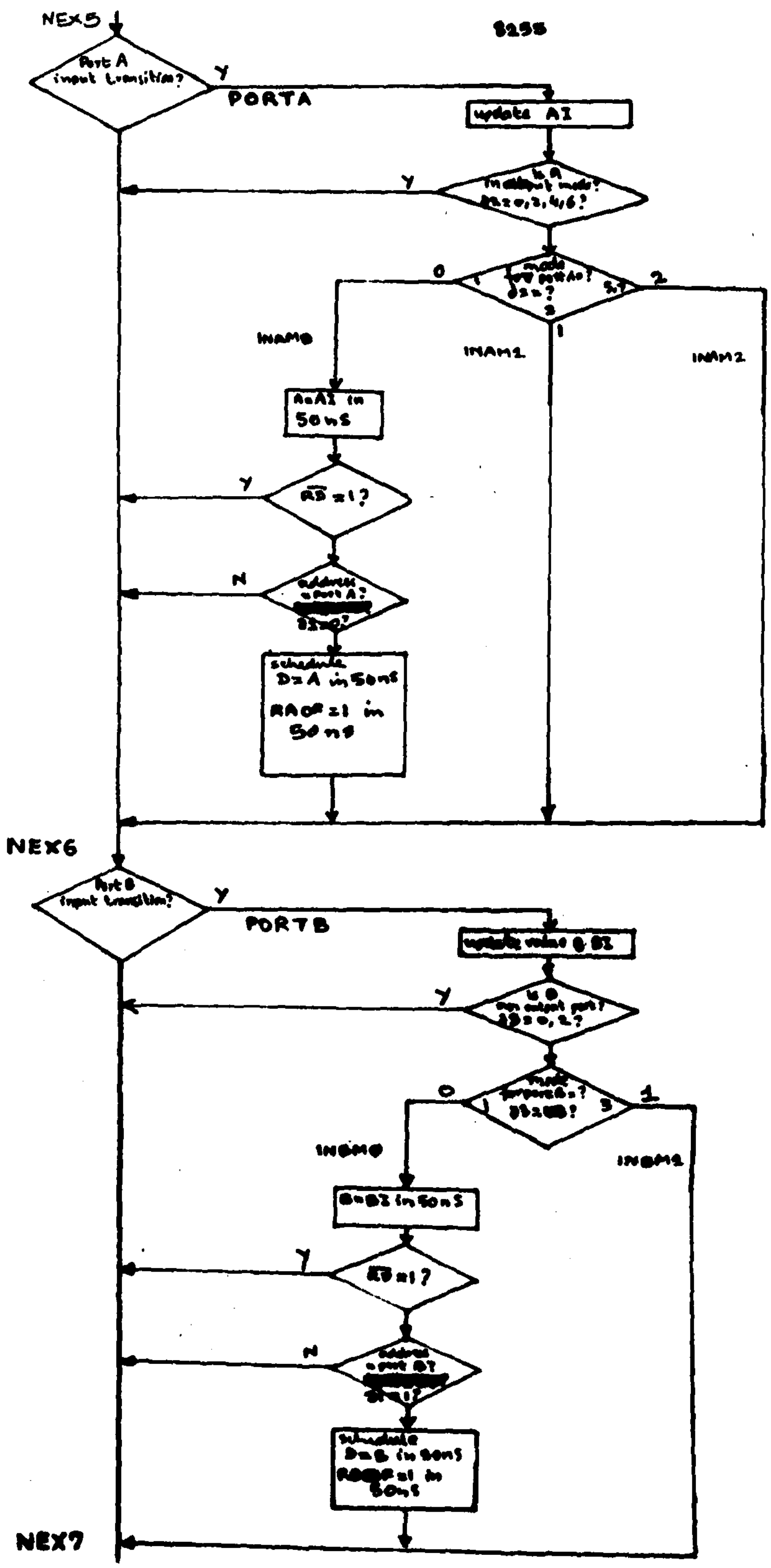

KSTFR
Read self call mode 0 RCOF=1? in port C
Y
RCOSC
clear flag
NRD=1?
D= in 150ns
schedule completion of Read in 150ns RDSCF=1 in 150ns
RCOFR
Input to port B while IBF=0 RBOF=1?
Y
RBOSC
clear flag
Y
NRD=1?
Reg. D= Bin 150ns
schedule completion of Read in 150ns RDSCF=1 in 150ns
RBOFR
Input to port A while IBF=0? RAOF=1?
Y
RAOSC
clear flag
Y
NRD=1?
Reg D= A in 150ns
Schedule completion of Read in 150ns RDSCF=1 in 150ns
RAOFR
Read self call? RDSCF=1?
Y
RDSC
clear flag
Y
Read completed? RDF=0?
no
Output data in 50ns DO=D in 50ns
Reset RDF to 0
RDFR

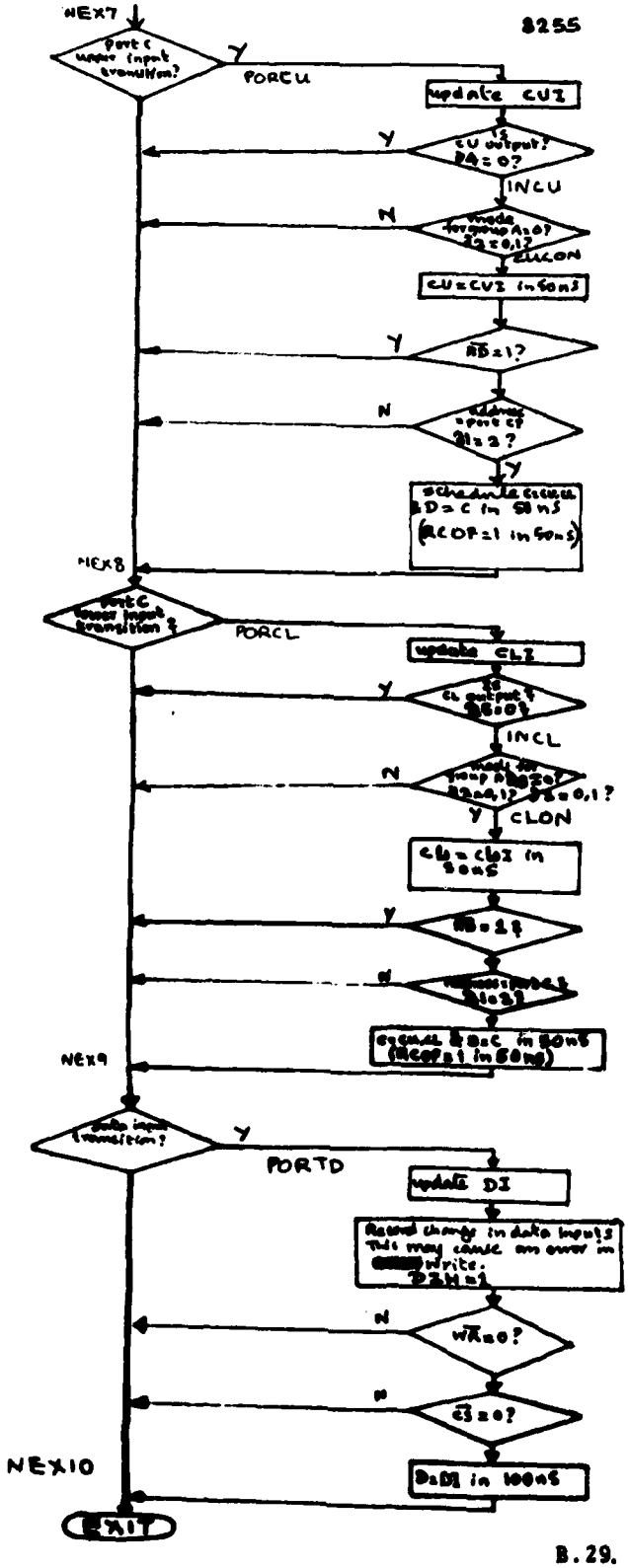

WRSC
clear flag
WRSCC
Write to which port ?
CONTROL/BIT S/R
PAW
CTLW
PBW
PCW
CTLWR
BIT
Reset
Set
WRFR
clear flag
HVIO
cancel write
WRF=0
WRPDY
AHSC
clear flag
NEXT


NEX1
8255
transition?
Y
SELECT
NEX2
ADD
Update value of ADI
ADIH=1
NEX3
WR
transition?
WRITE
Y
NO
NO
N
Y
NRD=1?
NCS=0?
D=DI in
100 ns
ADIH=0
Reset Error Flag
DIH=0
NCS=0?
DHSCF=1 in 30nS
NEX4


8255
NEX4
RD transition?
Y
READ
update NRD
NO
Float data output in 50nS
D0=FF in 50nS
READ1
Y
cancel write
WRF=0
READ2
NO
Y
ADZH=0
RDF=1
which port?
A
PAR
B
PBR
C
PCR
Error
Y
mode?
PAMIR
PAM2R
D=A in 150nS
schedule Read self call in 150nS
Y
PBM0R
PBM1R
form
schedule Read self call in 150 nS
NEX5

B.27.


8255
NEX5
Port A input transition?
Y
PORTA
update AI
Y
0
2
1
INAM0
INAM1
INAM2
Y
N
NEX6
Port B input transition?
Y
PORTB
Y
0
1
INBM0
INBM1
Y
N
NEX7


8255
NEX7
Y
PORCU
update CUI
Y
INCU
N
CLKCON
CU=CUI in 50ns
Y
N
Y
NEX8
PORCL
update CLI
Y
INCL
N
Y
CLON
CL=CLI in 50ns
Y
N
NEX9
Y
PORTD
update DI
N
Y
NEX10
EXIT

B.29.

Appendix C

Module Assembly Language Descriptions

```
        REG(8)    A1,B1,C1,D1,E1,H1,L1,IR
        REG(8)    PCH,PCL,SPH,SPL,TEMPW,TEMPZ,TEMP1,BUF
        REG(1)    ZERO,CARRY,SIGN,PAR,AC,INTFF,TEMP
;
        PIN       DBI(1,8),DBO(9,16),ADRL(17,24),ADRH(25,32)
        PIN       RESET(33),READY(34),HOLD(35),INT(36)
        PIN       NSYNC(37),WAIT(38) ,HLDA(39),INTE(40)
        PIN       NWR(41),DBIN(42),CLK1(43),CLK2(44),MONON(45)
        PIN       EX(150),CLKSC(151),DBSC(152),RDYSC(153)
;
        EVW       CP(500),RISE2(111),FALL2(344),FSYNC(415),RSYNC(0)
        EVW     RRDY(344),RDBIN(181),FDBIN(181), RWAIT(60),FWAIT(60)
        EVW    FINTE(261), RADR(261),RSYDB(311),RDBWR(311)
        EVW     RWR(600), FWR(600)
;
                  ; THIS IS THE BEGINNING OF THE FETCH/EXECUTE LOOP
                  ; FOR AN 8080 MICROP. THIS MAIN LOOP TAKES CARE OF
                  ; NSYNC, ADRESS, DBUS, WAIT STATES AND MORE.
                  ; INPUTS THAT ARE SAMPLED ARE RESET AND READY.
                  ; INTERUPTS ARE ALSO SIMULATED IN THIS MODEL.
;
        MOV       #0,CLKSC          ; CLEAR CLOCK SELF CALL FLAG
        MOV(RISE2) #1,CLK2          ; CLOCK PULSE
        MOV(FALL2) #0,CLK2          ;
        MOV       #0,PCL            ; RESET LOW BYTE OF PROGRAM COUNTER
        MOV       #0,PCH            ; RESET HIGH BYTE OF PROGRAM COUNTER
        MOV(CP)   #0,INTE           ; RESET INTERUPT ENABLE FLAG
        MOV       #0,INTFF          ; RESET INTERNAL INTERUPT FLIP/FLOP
        MOV(RADR) #255,DBO          ; FLOAT OUTPUT BUS
        MOV(CP)   #1,CLKSC          ; SET UP SELF CALL
        MOV       #1,EX             ; MOVE AHEAD ONE CLOCK PULSE
        MOV(RISE2) #1,CLK2          ; CLOCK PULSE
        MOV(FALL2) #0,CLK2          ;
        MOV       #0,CLKSC          ; RESET CLOCK SELF CALL
        MOV(RADR) #255,ADRL         ; FLOAT ADDRESS LOW LINES
        MOV(RADR) #255,ADRH         ; FLOAT ADDRESS HIGH LINES
RSET1:  MOV(CP)   #1,CLKSC          ; SET UP CLOCK SELF CALL
        MOV       #1,EX             ; EXIT AND MOVE AHEAD A CYCL
        MOV       #0,CLKSC          ; RESET CLOCK SELF CALL FLA
        MOV(RISE2) #1,CLK2          ; RISEING EDGE OF THETA 2
        MOV(FALL2) #0,CLK2          ; FALLING EDGE OF THETA 2
        BNE       RESET,RSET1       ; CONTINUE CLOCK PULSES UNTIL NOT RESET
                                    ;
M1T1:   BEQ       INTFF,FTCH        ; TEST IF THIS IS FETCH CYCLE
        MOV(RSYDB) #35,DBO          ; STATUS WORD FOR INTA
        MOV(FINTE) #0,INTE          ; RESET INTE FLAG
        BRU       FTCH1             ;
FTCH:   MOV(RSYDB) #162,DBO         ; STATUS WORD FOR FETCH
FTCH1:  MOV(RADR) PCL,ADRL          ; MOVE OUT LOWER BYTE OF PC
        MOV(RADR) PCH,ADRH          ; MOVE OUT UPPER BYTE OF PC
        MOV(FSYNC) #0,NSYNC         ; SYNC PULSE
        JSR       CLOCK             ; GO AHEAD ONE CLOCK PULSE
        BNE       INTFF,M1T2        ; CHECK FLAG TO SEE IF PC IS INC.
        INC       PCL               ; INCREMENT LOW BYTE OF PC
        BEQ       Z,M1T2            ; CHECK IF PCL IS ZERO
```

```
        INC         PCH             ; IF SO THEN INC PCH
M1T2:   MOV(RSYNC)  #1,NSYNC        ; TRAILLING EDGE OF SYNC PULSE
        MOV(RDBIN)  #255,DBO        ; TRISTATE DBO SO BUS MODULE WORKS
        MOV         #0,INTFF        ; RESET INTERNAL INTERUPT FLAG
        MOV(RDBIN)  #1,DBIN         ; SET DBIN SO THAT INSTRUCTION CAN BE
        JSR         WAIT1           ; FETCHED, CHECK FOR WAIT STATES
        MOV         DBI,IR          ; INPUT THE INSRUCTION INTO THE REG.
        MOV(FDBIN)  #0,DBIN         ; FALLING EDGE OF DBIN
        JSR         CLOCK           ; CLOCK PULSE
        BEQ         MONON,1A        ; EXECUTE ROUTINE.
        MON                         ; THIS MON CALL IS TOGGLED ON AND
1A:     IDX         IR(6),2,1       ; OFF BY THE MONON PIN.
        JSR         TBL1@1          ; THIS DOES THE EXECUTE PART.
                                    ;
                    ; AT THIS POINT WE SHOULD BE IN THE MACHINE CYCLE PRIOR
                    ; TO M1T1.  THIS IS NECCESSARY SO I CAN CHECK IF AN
                    ; INTERUPT HAS OCCURRED.
                                    ;
        MOV(FINTE)  #1,DBSC         ; PREPARE SELFCALL FOR INT TEST
        MOV(CP)     #1,CLKSC        ; SELFCALL FOR CLOCK
        MOV         #1,EX           ; DO EXIT
        BEQ         INTE,MXTX       ; IS INTE SET?
        BEQ         INT,MXTX        ; IS INT  SET?
        MOV         #1,INTFF        ; YES, THEN SET INTERNAL INT FF
MXTX:   MOV         #0,DBSC         ; RESET FLAG
        MOV         #1,EX           ; FINISH CLOCK PULSE
        MOV         #0,CLKSC        ; RESET FLAG
        BEQ         RESET,M1T1      ; GO BACK AND FETCH NEW INSTR.
        MOV(CP)     #1,CLKSC        ; SET UP FOR RESET CLOCK
        MOV         #0,EX           ; GOTO RESET
                                    ;
TBL1:   BYT         1000,2000,3000,4000
                                    ; WAIT STATE (TW) GENERATOR SUBROUTINE
WAIT1:  MOV(CP)     #1,CLKSC        ; SETUP FOR NEXT CLOCK PULSE
        MOV(RRDY)   #1,RDYSC        ; DELAY SO THAT READY CAN BE CHECKED
        MOV         #1,EX           ; EXIT AND WAIT FOR READY SIGNAL
        MOV         #0,RDYSC        ; RESET RDYSC AND CHECK READY SIG.
        BNE         READY,ENDT3     ; THIS CHECKS IF WAIT STATES ARE NEEDED
        MOV         #1,EX           ; COMPLETE CLOCK CYCLE
        MOV         #0,CLKSC        ; RESET CLOCK SELF CALL
        MOV(RISE2)  #1,CLK2         ; RISEING EDGE OF CLOCK
        MOV(FALL2)  #0,CLK2         ; FALLING EDGE OF CLOCK
        BEQ         RESET,WT2       ; CHECK FOR RESET
        MOV(CP)     #1,CLKSC        ; SET UP FOR RESET SELF CAL'.
        MOV         #0,EX           ; EXIT TO RESET
WT2:    MOV(RWAIT)  #1,WAIT         ; ENTER WAIT STATE
        BRU         WAIT1           ; GO BACK AND CHECK FOR MORE STATES
ENDT3:  MOV         #1,EX           ; FINISH CLOCK
        MOV         #0,CLKSC        ; RESET THE CLOCK SELF CALL
        MOV(RISE2)  #1,CLK2         ; CLOCK PULSE
        MOV(FALL2)  #0,CLK2         ;
        RTS                         ; RETURN TO CALLING PROGRAM
                                    ;
                    ;ASSUMPTIONS:   PC => ADDRESS
                    ;               MEM READ STATUS => DBO
```

```
                        ;                       PC IS INCREMENTED
PCRD:   JSR             CLOCK                   ; COMPLETE CURRENT CLOCK CYCLE
        MOV(RSYDB)      #130,DBO                ; STATUS BYTE IS PUT OUT ON THE DBUS
        MOV(RADR)       PCL,ADRL                ; PUT LOW BYTE OF PC ON ADDRESS BUS
        MOV(RADR)       PCH,ADRH                ; MOV THE HIGH BYTE OF PC ONTO BUS
        INC             PCL                     ; INCREMENT PROGRAM COUNTER
        BEQ             C,MEMRD                 ; INC UPPER 8 BITS IF THERE IS A CARRY
        INC             PCH                     ; INC UPPER 8 BITS
                        ;
                        ; ASSUMPTIONS:          ADDRESS AND SYNC BYTE ALREADY SET
                        ;                       BUF WILL RECIEVE DATA BYTE FROM DBI
MEMRD:  MOV(FSYNC)      #0,NSYNC                ; START SYNC PULSE
        JSR             CLOCK                   ; CLOCK PULSE
        MOV(RSYNC)      #1,NSYNC                ; TRAILLING EDGE OF SYNC PULSE
        MOV(RDBIN)      #255,DBO                ; TRISTATE DBO SO THAT BUS MODULE WORKS
        MOV(RDBIN)      #1,DBIN                 ; START DBIN PULSE
        JSR             WAIT1                   ; CHECK FOR WAIT STATES
        MOV             DBI,BUF                 ; INPUT DATA BYTE
        MOV(FDBIN)      #0,DBIN                 ; TRAILLING EDGE OF DBIN PULSE
        RTS
                        ;
                        ; ASSUMPTIONS:          ADDRESS IS ALREADY SET
                        ;                       DBO HAS SYNC BYTE
                        ;                       BUF HAS BYTE TO OUTPUT
MEMWR:  MOV(FSYNC)      #0,NSYNC                ; SYNC PULSE
        JSR             CLOCK                   ; CLOCK PULSE
        MOV(RSYNC)      #1,NSYNC                ; END SYNC PULSE
        MOV(RDBWR)      BUF,DBO                 ; OUTPUT DATA BYTE
        MOV(FWR)        #0,MWR                  ; SET WRITE PULSE
        JSR             WAIT1                   ; CHECK FOR WAIT STATES
        MOV(RWR)        #1,MWR                  ; REMOVE WRITE PULSE
        RTS
                        ;
CLOCK:  MOV(CP)         #1,CLKSC                ; CLOCK SUBROUTINE, SET SELFCALL
        MOV             #1,EX                   ; STOP EXECUTION UNTIL TIME IS REACHED
        MOV             #0,CLKSC                ; RESET CLKSC FLAG
        MOV(RISE2)      #1,CLK2                 ; OUTPUT CLOCK PULSE
        MOV(FALL2)      #0,CLK2                 ;
        BEQ             RESET,ESCHL             ; BRANCH IF NOT RESET
        MOV(CP)         #1,CLKSC                ; SET UP FOR RESET SELF CALL
        MOV             #0,EX                   ; GOTO RESET
ESCHL:  RTS                                     ; RETURN FOR CLOCK AND ESCAPE FOR HLRD
                        ;
                        ; THIS PART GETS THE SSS BYTE FROM A REGISTER (A1-L1) OR
                        ; FROM MEMORY USING THE HL PAIR. TO IMPLIMENT USE TABLE 2
                        ; EXAMPLE:  JSR  HLRD
                        ;           MOVE TBL2@5,A1
HLRD:   JSR             CLOCK                   ; CLOCK PULSE BEG. M1T5 OR M2T1
        BNE             #6,@2,ESCHL             ; CHECK IF REGISTER OR HL PAIR USED
        MOV(RSYDB)      #130,DBO                ; STATUS BYTE => MEMORY READ
        MOV(RADR)       L1,ADRL                 ; OUTPUT LOW ADDRESS ( L REGISTER )
        MOV(RADR)       H1,ADRH                 ; OUTPUT HIGH ADDRESS ( H REGISTER )
        BRU             MEMRD                   ; USE MEMRD TO COMPLETE CYCLE
                        ;
                        ; THIS SETS UP THE ADDRESS FOR A MEMORY READ USING THE
```

```
                        ; HL PAIR.
HLWR:   JSR             CLOCK
        MOV(RSYDB)      #0,DBO              ; STATUS BYTE => MEMORY WRITE
        MOV(RADR)       L1,ADRL             ; OUTPUT ADDRESS
        MOV(RADR)       H1,ADRH
        BRU             MEMWR               ; LET MEMWR COMPLETE CYCLE
                                            ;
DECSP:  XOR             TEMP1,TEMP1         ; THIS SUBROUTINE IS USED ANYTIME
        COM             TEMP1               ; SOMETHING IS PUSHED ONTO THE STACK
        ADD             TEMP1,SPL           ;   -1 IS ADD TO THE STACK POINTER
        BNE             C,5                 ; PAIR
        ADD             TEMP1,SPH
5:      RTS
                                            ;
FLAGS:  MOV             C,CARRY             ; SET 8080 CARRY FLAG TO NEW VALUE
FLAG1:  MOV             Z,ZERO              ; CHANGE ZERO FLAG
        MOV             N,SIGN              ; CHANGE SIGN FLAG
        ADD             TEMPZ,TEMPW         ; ADD LOW ORDER BITS
        IDX             TEMPW(4),1,6        ; GET AUX. CARRY BIT
        MOV             @6,AC               ; STORE FLAG
FLGPA:  IDX             TEMP1(0),4,7        ; CONVERT 4 LOW ORDER BIT TO @7
        MOV             PARIT@7,PAR         ; FIND PARITY OF LOWER BITS
        IDX             TEMP1(4),4,7        ; 4 UPPER BITS TO @7
        MOV             PARIT@7,TEMP        ; STORE PARITY OF UPPER BITS
        BEQ             PAR,TEMP,FLAG2      ; BRANCH ON EVEN PARITY
        MOV             #0,PAR              ; THERFORE WE HAVE ODD PARITY
        RTS                                 ; DONE
FLAG2:  MOV             #1,PAR              ; EVEN PARIY
        RTS                                 ; DONE
                                            ;
ACFLG:  BYT             #0,#0,#0,#0,#0,#0,#0,#0,#0,#0,#0,#1,#1,#1,#1,#1,#1
PARIT:  BYT             #0,#1,#1,#0,#1,#0,#0,#1,#1,#0,#0,#1,#0,#1,#1,#0
                                            ;
                                            ; 0  =>   BIT IS 0
                                            ; *  =>   BIT IS 1
                                            ; X  =>   BIT IS NOT TESTED YET
                                            ; 1-8 =>  INDEX REG USED FOR BITS
                                            ;
                                            ; MSB   LSB
                                            ; 00XX XXXX     ALL INST. WITH 00 IN 6,7
1000:   IDX             IR(0) ,4,1          ;      1111     DECODE LOWER 4 BITS
        IDX             IR(3) ,3,2          ;    22 2       DECODE REGISTER
        IDX             IR(4) ,2,3          ;    33         DECODE REGISTER PAIR
        BRU             TBL4@1
                                            ;
TBL4:   BYT             1100,1125,1150,1175
        BYT             1200,1225,1250,1275
        BYT             1100,1325,1150,1375
        BYT             1200,1225,1250,1475
                                            ;
1100:   RTS                                 ; 00XX X000     00H IS NOP, ALL OTHERS
                                            ;               ARE NOT DEFINED
                                            ;
1125:   JSR             PCRD                ; 00XX 000*     LXI INSTRUCTIONS
        MOV             BUF,TBL6@3          ;   33          BYTE IS PUT IN PROP. REG
```

```
        JSR         PCRD                ; GET SECOND BYTE
        MOV         BUF,TBL5@3          ; PUT SECOND BYTE IN PROPER REGISTER
        RTS
                                        ; 00XX X0*0     <= INSTRUCTION TYPE
1150:   BEQ         #2,@3,1165          ; 00*0 X0*0     SHLD ADR  OR LHLD ADR
        BEQ         #3,@3,1165          ; 00** X0*0     STA  ADR  OR LDA  ADR
        JSR         CLOCK               ; 000X X0*0     STAX B,D  OR LDAX B,D
        MOV(RADR)   TBL6@3,ADRL         ; PUT PROPER ADDRESS ON ADR LINES
        MOV(RADR)   TBL5@3,ADRH         ;
        BEQ         #2,@1,1151          ; 000X 00*0     SEPERATE STORES FROM LOADS
        MOV(RSYDB)  #130,DBO            ; PUT OUT PROPER STATUS
        JSR         MEMRD               ; 000X *0*0     LDAX B,D
        MOV         BUF,A1              ; PUT BYTE INTO A1 REGISTER
        RTS
                                        ;
1151:   MOV         A1,BUF              ; 000X 00*0     STAX B,D
        MOV(RSYDB)  #0,DBO              ; MEMORY WRITE STATUS
        JSR         MEMWR               ; STORE BYTE INTO MEMORY
        RTS
                                        ;
                                        ; 00*X X0*0     STA,LDA,SHLD,LHLD
1165:   JSR         PCRD                ; GET BYTE FROM MEMORY
        MOV         BUF,TEMPZ           ; STORE BYTE INTO TEMP REGISTER
        JSR         PCRD                ; GET SECOND BYTE FROM MEM.
        MOV         BUF,TEMPW           ; STORE THAT BYTE INTO REGISTER
        JSR         CLOCK               ; CLOCK PULSE
        MOV(RADR)   TEMPZ,ADRL          ; OUTPUT THE ADDRESS IN PREP. FOR
        MOV(RADR)   TEMPW,ADRH          ; MEMORY READ OR WRITE
        BNE         #2,@1,1170          ; 00*X 00*0     LDA OR LHLD
        MOV(RSYDB)  #0,DBO              ; STATUS = MEMORY WRITE
        BEQ         #2,@3,1166          ; 00*0 *0*0     SHLD
        MOV         A1,BUF              ; 00** *0*0     STA
        BRU         1167
                                        ;
1166:   MOV         L1,BUF              ;
1167:   JSR         MEMWR               ; WRITE BYTE INTO THE MEMORY
        BEQ         #3,@3,1176          ; STA IS FINISHED SO BAIL OUT !
1172:   INC         TEMPZ               ; MOVE TO NEXT LOCATION FOR
        BEQ         C,1168              ; SHLD OR LHLD
        INC         TEMPW
1168:   JSR         CLOCK               ; ADVANCE TO NEXT CLOCK CYCLE
        MOV(RADR)   TEMPZ,ADRL          ; OUTPUT ADDRESS
        MOV(RADR)   TEMPW,ADRH
        BNE         #2,@1,1173          ; THIS SEPERATES SHLD FROM LHLD
        MOV         H1,BUF              ; PUT REGISTER INTO BUF SO IT'S STORED
        MOV(RSYDB)  #0,DBO              ; STATUS BYTE => MEMORY WRITE
        BRU         MEMWR               ; LET MEMWR STORE BYTE
                                        ;
1170:   MOV(RSYDB)  #130,DBO            ; STATUS BYTE => MEMORY READ
        BEQ         #2,@3,1171          ; SEPERATE LHLD FROM LDA
        JSR         MEMRD               ; LDA INSTRUCTION
        MOV         BUF,A1              ; STORE
        RTS
                                        ;
1171:   JSR         MEMRD               ; READ FIRST BYTE FROM MEMORY
```

```
        MOV     BUF,L1          ; STORE IT IN L1
        BRU     1172            ; GO BACK AND INCREMENT TEMPW-Z PAIR
                                ;
1173:   JSR     MEMRD           ; READ SECOND BYTE FROM MEMORY
        MOV     BUF,H1          ; AND PUT IT IN H REGISTER
        RTS
                                ;
1175:   INC     TBL6@3          ; 00XX 00**    INX RP  (INDEX 3 )
        BEQ     C,1176          ; CHECK IF CARRY
        INC     TBL5@3          ; INC UPPER BYTE
1176:   RTS
                                ;
1200:   JSR     HLRD            ; 00XX X*00    INR DDD
1201:   INC     TBL2@2          ; INC THAT BYTE
        MOV     TBL2@2,TEMP1    ; SET UP FOR FLAGS
        MOV     #1, TEMPZ       ;*** SET UP AC FLAG STUFF
        AND     #15, TEMP1, TEMPW ;***
        JSR     FLAG1           ; SET CERTAIN FLAGS
        BNE     #6,@2,1176      ; MEMORY?
        BRU     HLWR            ; YES, STORE BYTE
                                ;
1225:   JSR     HLRD            ; 00XX X*0*    DCR
        XOR     TEMPZ,TEMPZ     ; ZERO TEMPZ
        COM     TEMPZ           ; SET TO -1
        ADD     TEMPZ,TBL2@2    ; ADD -1 TO REGISTER
        MOV     TBL2@2,TEMP1    ; SET UP FOR FLAGS
        JSR     FLAG1           ; GOTO FLAGS ROUTINE
        BNE     #6,@2,1252      ; WAS IT A MEMORY TYPE INSTR.?
        BRU     HLWR            ; STORE BYTE INTO MEMORY
                                ;
1250:   JSR     PCRD            ; 00XX X**0    MVI
        BNE     #6,@2,1251      ; MEMORY?
        BRU     HLWR            ; YES, PUT BYTE INTO MEMORY
                                ;
1251:   MOV     BUF,TBL2@2      ; PUT BYTE INTO REGISTER
1252:   RTS
                                ;
1275:   BRU     TBL7@3          ; 00XX 0***    RLC, RAL, DAA, STC
                                ;
TBL7:   BYT     1276,1280,1285,1290
                                ;
                                ; 0000 0***    RLC
1276:   ADD     A1,A1
        MOV     C,CARRY         ; SHIFT BIT INTO CARRY FLAG
        BEQ     C,1176          ; IS CARRY SET?
         OR     #1,A1           ; YES SO SET LSB OF REGISTER
        RTS
                                ;
                                ; 000* 0***    RAL
1280:   MOV     CARRY,TEMP      ; STORE CARRY
        ADD     A1,A1           ; SHIFT
        MOV     C,CARRY         ; STORE CARRY
        BEQ     TEMP,1282       ; TEMP SET  ******************
         OR     #1,A1           ; MOVE BIT INTO LSB
1282:   RTS
```

```
                                        ;
1285:  BNE      AC,1286                 ; 00*0 0***    DAA
       IDX      A1(0),4,7               ; SELECT 4 LOWER BITS
       MOV      ACFLG@7,TEMP            ; GET AC FLAG
       BEQ      TEMP,1287               ; IS IT SET?
1286:  ADD      #6,A1                   ; LOWER NIBLE NOW CORRECTED
1287:  BNE      CARRY,1288              ; IS CARRY FLAG SET ?
       IDX      A1(4),4,7               ; SELECT TOP 4 BITS
       MOV      ACFLG@7,TEMP            ; GET AC FLAG FOR UPPER BITS
       BEQ      TEMP,1289               ; FLAG SET ?
1288:  ADD      #96,A1                  ; CORRECT UPPER BITS
1289:  MOV      A1,TEMP1                ; SET UP FOR FLAGS
       BRU      FLAGS                   ; GO SET FLAGS
                                        ;
1290:  MOV      #1,CARRY                ; 00** 0***    STC  SET CARRY BIT
       RTS                              ;
                                        ;
1325:  ADD      TBL6@3,L1               ; 00XX *00*    DAD  DOUBLE ADD
       BEQ      C,1326                  ; CHECK IF HIGH BYTE GETS INC
       INC      H1                      ; INC HIGH BYTE
1326:  ADD      TBL5@3,H1               ; ADD HIGH BYTE
       MOV      C,CARRY                 ; STORE CARRY
       JSR      CLOCK                   ; I NEED TO WASTE 6 CLOCK PULSES
       JSR      CLOCK
       JSR      CLOCK
       JSR      CLOCK
       JSR      CLOCK
       JSR      CLOCK
       RTS
                                        ;
1375:  JSR      CLOCK                   ; THIS IS A M1T5
       XOR      TEMPZ,TEMPZ             ; 00XX *0**    DCX  DEC REG PAIR
       COM      TEMPZ                   ; SET TEMPZ TO -1
       ADD      TEMPZ,TBL6@3            ; ADD -1 TO REG
       BNE      C,1176                  ; CARRY?
       ADD      TEMPZ,TBL5@3            ; ADD -1 TO UPPER BYTE
       RTS
                                        ;
1475:  BRU      TBL8@3                  ; 00XX ****    RRC, RAR, CMA, CMC
                                        ;
TBL8:  BYT      1400,1425,1485,1490
                                        ;
                                        ; 0000 ****    RRC
1400:  ROR      A1
       BRU      1440
                                        ;
                                        ; 000* ****    RAR
1425:  MOV      CARRY,@7                ; STORE CARRY IN 7
       SHR      A1                      ; SHIFT REST OF BYTE
1440:  MOV      C,CARRY                 ; STORE NEW CARRY
       BNE      #6,@2,1176              ;
       BRU      HLWR
                                        ;
                                        ;
1485:  COM      A1                      ; 00*0 ****    CMA
```

```
        RTS
                                        ;
1490:   XOR     #1,CARRY                ; 00** ****    CMC
        RTS
                                        ; 0*XX XXXX    MOVE INSTRUCTIONS
2000:   IDX     IR(3) ,3,1              ;   11 1       DESTINATION
        IDX     IR(0) ,3,2              ;        222   SOURCE
        BEQ     #6,@1,2100              ; DEST TO MEMORY
        BEQ     #6,@2,2200              ; SOURCE FROM MEMORY
        MOV     TBL2@2,TBL2@1           ; MOV R,R
        BRU     CLOCK                   ; ADD CLOCK CYCLE AND RETURN
                                        ;
2100:   BEQ     #6,@2,HLT               ; MOV M,R      CHECK IF HALT IS FOUND
        MOV     TBL2@2,BUF              ; DO MOVE
        BRU     HLWR
                                        ;
2200:   JSR     HLRD                    ; MOV R,M
        MOV     BUF,TBL2@1
        RTS
                                        ; 0*** 0**0    HALT INSTRUCTION
HLT:    MOV     #1,HLDA                 ; SIGNAL HALT ACKNOWLAGE
        MOV     #1,EX                   ; THIS IS TEMPORARY, IT MUST BE EXPANDED
        BRU     HLT                     ; SO THAT THE TIMING IS CORRECT AND
        ; INTERUPTS CAN OCCUR.
                                        ;
                                        ; *0XX XXXX    ARITH, AND LOGICAL
3000:   IDX     IR(3) ,3,1              ;   11 1       TYPE OF INSTRUCTION
        IDX     IR(0) ,3,2              ;        222   SOURCE
        BNE     #6,@2,3090
        JSR     HLRD                    ; GET BYTE FROM MEMORY
3090:   AND     #15,TBL2@2,TEMPZ        ; THIS IS FOR FIGURING AC FLAG
        AND     #15,A1,TEMPW
        BRU     TBL3@1                  ; DO FUNCTION
                                        ;
TBL3:   BYT     3100,3200,3300,3300
        BYT     3500,3600,3700,3800
                                        ;
3200:   BEQ     CARRY,3100              ; *000 *XXX    ADC
        INC     TEMPW                   ; KEEP TRACK OF ALL ADDITIONS
        INC     A1                      ; ADD CARRY
3100:   ADD     TBL2@2,A1               ; *000 0XXX    ADD
        MOV     A1,TEMP1                ; SET UP TEMP1 FOR FLAGS
        BRU     FLAGS                   ; GO AND SET FLAGS
                                        ;
3300:   MOV     TBL2@2,TEMP1            ; *00* XXXX    SUB,SBB
        COM     TEMPZ                   ;***AC FLAG
        AND     #15, TEMPZ              ;***
        BEQ     #2,@1,3350
3340:   BEQ     CARRY,3350              ; *00* *XXX    SBB
        INC     TEMP1
        BRU     3351                    ;***
3350:   INC     TEMPZ                   ; *00* 0XXX    SUB
3351:   SUB     A1,TEMP1,A1             ;***
        MOV     A1,TEMP1
        BRU     FLAGS                   ;***
```

```
                                        ;
3500:  AND      TBL2@2,A1               ; *0*0 0XXX    ANA
3511:  MOV      #0,AC                   ; CLEAR AC FLAG
       MOV      #0,CARRY                ; CLEAR CARRY
       MOV      N,SIGN                  ; SET SIGN FLAG
       MOV      Z,ZERO                  ; SET ZERO FLAG
       MOV      A1,TEMP1
       BRU      FLGPA
                                        ;
3600:  XOR      TBL2@2,A1               ; *0*0 *XXX    XRA
       BRU      3511
                                        ;
3700:  OR       TBL2@2,A1               ; *0** 0XXX    ORA
       BRU      3511
                                        ;
3800:  COM      TEMPZ                   ;***
       AND      #15, TEMPZ              ;***
       INC      TEMPZ                   ;***
       SUB      A1,TBL2@2,TEMP1         ; *0** *XXX    CMP
       BRU      FLAGS
                                        ; **XX XXXX    INSTRUCTIONS BEGIN WITH **
4000:  IDX      IR(0) ,4,1              ;      1111    DECODE INSTRUCTION TYPE
       IDX      IR(3) ,3,2              ;    22 2      CONDITIONS
       IDX      IR(4) ,2,3              ;    33        ADDITIONAL DECODING
       BRU      TBL9@1
                                        ;
TBL9:  BYT      4100,4125,4150,4175
       BYT      4200,4225,4250,4275
       BYT      4100,4325,4150,4375
       BYT      4200,4215,4250,4275
                                        ;
4100:  IDX      IR(3) ,1,4              ; **XX X000    CONDITIONAL RETURNS
       JSR      CLOCK                   ; ALL HAVE 5 CYCLES IN M1
       BRU      TBL17@3
                                        ;
TBL17: BYT      4101,4102,4103,4104
                                        ;
4101:  BEQ      @4,ZERO,4105            ; **00 4000    RZ,RNZ
       RTS
                                        ;
4102:  BEQ      @4,CARRY,4105           ; **0* 4000    RC,RNC
       RTS
                                        ;
4103:  BEQ      @4,PAR,4105             ; ***0 4000    RPO,RPE
       RTS
                                        ;
4104:  BEQ      @4,SIGN,4105            ; **** 4000    RP,RM
       RTS
                                        ;
4105:  MOV      #0,IR                   ; **00 *00*    RET
4106:  JSR      CLOCK
       MOV(RADR) SPL,ADRL               ; THIS SECTION IS USED TWICE (IR COUNTS)
       MOV(RADR) SPH,ADRH               ; OUTPUT STACK POINTER
       MOV(RSYDB) #134,DBO              ; STATUS BYTE => STACK READ
       JSR      MEMRD                   ; GET BYTE
```

```
        INC     SPL             ; GOTO NEXT STACK POSITION
        BEQ     C,4107          ; DETERMINE IF INC HIGH BYTE
        INC     SPH             ; INC HIGH BYTE OF STACK POINTER
4107:   BNE     IR,4108         ; SEPERATE FIRST PASS FROM SECOND
        MOV     #1,IR           ; SET UP FOR SECOND PASS
        MOV     BUF,TEMPZ       ; STORE BYTE IN BUFFER
        BRU     4106            ; START SECOND PASS
                                ;
4108:   MOV     TEMPZ,PCL       ; THIS DOES BRANCH BACK TO MAIN ROUTINE
        MOV     BUF,PCH         ; FOR THE RET INSTR
4109:   RTS
                                ;
4125:   MOV     #0,TEMPZ        ; **XX 000*     POP       B,D,H,PSW
4126:   JSR     CLOCK
        MOV(RADR) SPL,ADRL      ; OUTPUT STACK POINTER
        MOV(RADR) SPH,ADRH
        MOV(RSYDB) #134,DBO     ; STATUS BYTE => STACK READ
        JSR     MEMRD
        INC     SPL             ; INC STACK REGISTER PAIR
        BEQ     C,4127
        INC     SPH
4127:   BNE     TEMPZ,4128      ;
        MOV     #1,TEMPZ
        MOV     BUF,TBL11@3
        BRU     4126
                                ;
4128:   MOV     BUF,TBL10@3     ; STORE BYTE
        BNE     #3,@3,4109      ; SIFT OUT ALL BUT PSW POP
        SHR     TEMP1           ; MOVE LOW ORDER BIT OUT
        MOV     C,CARRY         ; SAVE FLAG
        SHR     TEMP1           ; MOVE OUT NEXT BIT
        SHR     TEMP1           ; MOVE OUT NEXT BIT
        MOV     C,PAR           ; SAVE PARITY
        SHR     TEMP1           ; MOVE OUT NEXT BIT
        SHR     TEMP1           ; MOVE OUT NEXT BIT
        MOV     C,AC            ; SAVE AUX. CARRY
        SHR     TEMP1           ; NEXT BIT
        SHR     TEMP1           ; MOVE NEXT BIT OUT
        MOV     C,ZERO          ; SAVE ZERO FLAG
        SHR     TEMP1           ; NEXT BIT
        MOV     C,SIGN          ; SAVE SIGN FLAG
        RTS                     ; WE ARE DONE SHIFTING
                                ;
4150:   IDX     IR(3) ,1,4      ; **XX 40*0     CONDITIONAL JUMPS
        JSR     CLOCK           ; M1 HAS 5 CYCLES
        JSR     PCRD            ; GET LOW ADDRESS
        MOV     BUF,TEMPZ       ; STORE IN TEMP REGISTER
        JSR     PCRD            ; GET HIGH ADDRESS
        MOV     BUF,TEMPW       ; STORE IN TEMP REG
        BRU     TBL13@3         ; DO CONDITION
                                ;
TBL13:  BYT     4151,4152,4153,4154
                                ;
4151:   BEQ     @4,ZERO,4155    ; JZ,JNZ
        BRU     4158
```

```
                                        ;
4152:  BEQ          @4,CARRY,4155       ; JC,JNC
       BRU          4158
                                        ;
4153:  BEQ          @4,PAR,4155         ; JPO,JPE
       BRU          4158
                                        ;
4154:  BEQ          @4,SIGN,4155        ; JP,JM
4158:  RTS                              ; CONDITION NOT MET
                                        ;
4155:  MOV          TEMPZ,PCL           ; DO BRANCH BY REPLACING PROGRAM COUNTER
       MOV          TEMPW,PCH           ;
       RTS
                                        ;
4175:  BRU          TBL14@3             ; **XX 00**    JMP, OUT, XTHL, DI
                                        ;
TBL14: BYT          4156,4176,4180,4185
                                        ;
4156:  JSR          PCRD                ; **00 00**    JMP UNCOND
       MOV          BUF,TEMPZ           ; SAVE LOW ADDRESS
       JSR          PCRD                ; GET HIGH BYTE
       MOV          BUF,TEMPW           ; SAVE HIGH BYTE
       BRU          4155                ; GOTO TRUE PART OF CONDITIONAL
                                        ;
4176:  JSR          PCRD                ; **0* 00**    OUT
       JSR          CLOCK
       MOV(RADR)    BUF,ADRL
       MOV(RADR)    BUF,ADRH            ; OUTPUT AND INPUT INST USE BOTH UP&LOW
       MOV(RSYDB)   #16,DBO             ; STATUS WORD => OUTPUT
       MOV          A1,BUF              ; SET UP FOR WRITE
       BRU          MEMWR               ; DO IT
                                        ;
                                        ; ***0 00**    XTHL
4180:  JSR          CLOCK
       MOV(RADR)    SPL,ADRL            ; OUTPUT STACK POINTER TO READ TOP
       MOV(RADR)    SPH,ADRH
       MOV(RSYDB)   #134,DBO            ; STATUS => STACK READ
       JSR          MEMRD
       MOV          BUF,TEMPZ           ; STORE BYTE
       INC          SPL                 ; INC STACK POINTER PAIR
       BEQ          C,4186              ;
       INC          SPH                 ;
4186:  JSR          CLOCK
       MOV(RADR)    SPL,ADRL            ; OUTPUT STACK POINTER
       MOV(RADR)    SPH,ADRH
       MOV(RSYDB)   #134,DBO            ; STATUS => STACK READ
       JSR          MEMRD               ; GET NEXT BYTE
       MOV          BUF,TEMPW           ; SAVE BYTE
       JSR          CLOCK
       MOV(RSYDB)   #4,DBO              ; STATUS => STACK WRITE
       MOV          H1,BUF              ; SET UP TO STORE H REG ON TOP
       JSR          MEMWR               ; H STORED
       JSR          DECSP               ; DEC STACK POINTER
       JSR          CLOCK
       MOV(RADR)    SPL,ADRL            ; OUTPUT ADDRESS
```

```
        MOV(RADR)   SPH,ADRH
        MOV(RSYDB)  #4,DBO              ;
        MOV         L1,BUF              ; STATUS => STACK WRITE
        JSR         MEMWR               ; GET READY TO STORE L1 REGISTER
        MOV         TEMPW,H1            ; STORE IT
        MOV         TEMPZ,L1            ; SAVE HL VALUES
        RTS

4185:   MOV         #0,INTE             ;
        RTS                             ; **** 00**    DI        DISABLE INT

                                        ; **XX X*00    CONDITIONAL CALL
4200:   IDX         IR(3) ,1,4          ;      4
        JSR         CLOCK
        JSR         PCRD                ; GET LOW BYTE OF ADDRESS
        MOV         BUF,TEMPZ           ; STORE IT IN TEMP REGISTER
        JSR         PCRD                ; GET HIGH BYTE OF ADDRESS
        MOV         BUF,TEMPW           ; STORE IN TEMP REGISTER
        BRU         TBL16@3
                                        ;
TBL16:  BYT         4201,4202,4203,4204
                                        ;
4201:   BEQ         @4,ZERO,4206        ; CZ,CNZ
        RTS
                                        ;
4202:   BEQ         @4,CARRY,4206       ; CC,CNC
        RTS
                                        ;
4203:   BEQ         @4,PAR,4206         ; CPO,CPE
        RTS
                                        ;
4204:   BEQ         @4,SIGN,4206        ; CP,CM
        RTS
                                        ;
4205:   JSR         CLOCK               ; **00 **0*    CALL
        JSR         PCRD                ; GET LOW BYTE
        MOV         BUF,TEMPZ           ; STORE IT
        JSR         PCRD                ; GET HIGH BYTE
        MOV         BUF,TEMPW           ; STORE IT
4206:   MOV         #0,IR               ; DO CALL INSTRUCTION
        MOV         PCH,BUF             ; SET UP TO STORE PC
4208:   JSR         DECSP               ; TWO PASS, DEC STACK POINTER
        JSR         CLOCK               ; ADVANCE TO NEXT CLOCK PULSE
        MOV(RADR)   SPL,ADRL            ; OUTPUT STACK POINTER
        MOV(RADR)   SPH,ADRH
        MOV(RSYDB)  #4,DBO              ; STATUS => STACK WRITE
        JSR         MEMWR               ; STORE IT
        BNE         IR,4210             ; CHECK WHICH PASS WE ARE ON
        MOV         #1,IR               ; SET UP FOR NEXT PASS
        MOV         PCL,BUF             ; SET UP PC TO BE PUT ON STACK
        BRU         4208
                                        ;
4210:   MOV         TEMPZ,PCL           ; THIS DOES THE BRANCH
        MOV         TEMPW,PCH
        RTS
                                        ;
```

```
4215:  BRU          TBL22@3              ; **XX **0*    CALL AND ILLEGAL INSTR
                                         ;
TBL22: BYT          4205,1100,1100,1100  ; CALL, DD, ED, FD
                                         ;
4225:  JSR          CLOCK                ; MIT5
       MOV          #0,IR                ; **XX 0*0*    PUSH
       MOV          TBL10@3,BUF          ; GET REGISTER
4226:  JSR          DECSP
       JSR          CLOCK
       MOV(RADR)    SPL,ADRL             ; OUTPUT STACK POINTER
       MOV(RADR)    SPH,ADRH
       MOV(RSYDB)   #4,DBO               ; STATUS => STACK WRITE
       JSR          MEMWR
       BNE          IR,1176              ; IF DONE THEN RETURN
       MOV          #1,IR
       BNE          #3,@3,4207           ; IS IT PUSH PSW?
       XOR          TEMP1,TEMP1          ; YES, PUT FLAGS INTO TEMP1
       MOV          CARRY,@7             ; STORE CARRY FLAG
       SHR          TEMP1                ; SHIFT BYTE
       SHR          TEMP1                ; SHIFT BYTE TWICE
       MOV          PAR,@7               ; STORE PARITY
       SHR          TEMP1                ;
       SHR          TEMP1                ;
       MOV          AC,@7                ; STORE AUX. CARRY
       SHR          TEMP1
       SHR          TEMP1                ; DUMMY BIT
       MOV          ZERO,@7              ; STORE ZERO FLAG
       SHR          TEMP1                ;
       MOV          SIGN,@7              ; STORE SIGN FLAG
       SHR          TEMP1
4207:  MOV          TBL11@3,BUF          ; SET UP FOR SECOND PASS
       BRU          4226
                                         ;THIS PART DOES IMMEDIATE TYPE 8080
4250:  JSR          PCRD                 ;INSTRUCTIONS.  DATA IS READ AND STORED
       AND          #15,BUF,TEMPZ        ; IN BUF. TABLE 12 THEN DECODES THE
       AND          #15,A1,TEMPW         ; PROPER INSTRUCTION.
       BRU          TBL12@2              ;
                                         ;
TBL12: BYT          4251,4252,4253,4254
       BYT          4255,4256,4257,4258
                                         ;
4251:  ADD          BUF,A1               ; ADI
       MOV          A1,TEMP1
       BRU          FLAGS                ; SET ALL OF THE FLAGS
                                         ;
4252:  BEQ          CARRY,4251           ; ACI
       INC          TEMPW
       INC          A1
       BRU          4251
                                         ;
4253:  COM          TEMPZ                ;***
       AND          #15, TEMPZ           ;***
       INC          TEMPZ                ;***
4299:  SUB          A1,BUF,A1            ; SUI
       MOV          A1,TEMP1
```

```
        BRU         FLAGS               ; SET FLAGS
                                        ;
4254:   BEQ         CARRY,4253          ; SBI
        ADD         #255,A1
        COM         TEMPZ               ;***
        AND         #15, TEMPZ          ;***
        BRU         4299
                                        ;
4255:   AND         BUF,A1              ; ANI
4298:   MOV         #0,CARRY
        MOV         #0,AC
        MOV         Z,ZERO
        MOV         N,SIGN
        MOV         A1,TEMP1
        BRU         FLGPA               ; SET PARITY
                                        ;
4256:   XOR         BUF,A1              ; XRI
        BRU         4298                ; SET FLAGS
                                        ;
4257:    OR         BUF,A1              ; ORI
        BRU         4298                ; SET FLAGS
                                        ;
4258:   COM         TEMPZ               ;***
        AND         #15, TEMPZ          ;***
        INC         TEMPZ               ;***
        SUB         A1,BUF,TEMP1        ; CPI
        BRU         FLAGS               ; SET FLAGS
                                        ; **XX X***    RST       RESTART
4275:   MOV         IR,TEMPZ            ; INSTRUCTION IS THE LOCATION FOR CALL
        AND         #56,TEMPZ           ; STRIP OFF WRONG BITS   ****************
        XOR         TEMPW,TEMPW         ; CLEAR UPPER BYTE
        BRU         4206                ; GOTO THE CALL ROUTINE
                                        ;
4325:   BRU         TBL1903             ; **XX *00*
                                        ;
TBL19:  BYT         4105,1100,4326,4330 ; RET, D9, PCHL, SPHL
                                        ;
4326:   MOV         L1,PCL              ; PCHL
        MOV         H1,PCH
        RTS
                                        ;
4330:   MOV         L1,SPL              ; SPHL
        MOV         H1,SPH
4335:   RTS
                                        ;
4375:   BRU         TBL1503             ; **XX *0**    SEE TBL15
                                        ;
TBL15:  BYT         1100,4380,4390,4395 ; CB, INPUT, XCHG, EI
                                        ;
4380:   JSR         PCRD                ;IN
        JSR         CLOCK
        MOV(RADR)   BUF,ADRL            ; OUTPUT LACATION
        MOV(RADR)   BUF,ADRH            ;
        MOV(RSYDB)  #66,DBO             ; STATUS => INPUT
```

```
        JSR         MEMRD               ; GET BYTE
        MOV         BUF,A1
        RTS
                                        ;
4390:   XOR         L1,E1               ;XCHG
        XOR         E1,L1
        XOR         L1,E1
        XOR         H1,D1
        XOR         D1,H1
        XOR         H1,D1
        RTS
                                        ;
4395:   MOV(CP)     #1,INTE             ;EI
        RTS
                                        ;
TBL2:   BYT         B1,C1,D1,E1
        BYT         H1,L1,BUF,A1
TBL5:   BYT         B1,D1,H1,SPH
TBL6:   BYT         C1,E1,L1,SPL
TBL10:  BYT         B1,D1,H1,A1
TBL11:  BYT         C1,E1,L1,TEMP1
        END
```

```
        REG(8)   L,TEMP8,DBIO,DIO  ;8228 SIMULATOR
        REG(3)   INTCT
        REG(1)   NSTBO,DBINO,NWRO,NBENO,NINTF,TEMP1,HLDAO
        PIN      DI(1,8),DO(9,16),DBI(17,24),DBO(25,32)
        PIN      NSTBS(33),DBIN(34),NWR(35),HLDA(36),NBEN(37),SPI(38)
        PIN      NIOR(39),NIOW(40),NMEMW(41),NMEMR(42),NINTA(43),DBE(44)
        PIN      DID(46,53),NI(54),NMR(55),NMW(56),NIR(57),NIW(58)
        PIN      EX(150),DBSC(151),CSC(152),LRSC(153)
        EVW      W30(30),W25(25),W10(10),W20(20),W45(45),WZ(0)
          BEQ        DBI,DBIO,DBSCK
          BRU        DBECK
;
DBSCK:  BEQ          DBSC,DBCK
        MOV          #0,DBSC              ;RESET SELFCALL FLAG
DBECK:  BEQ          DBE,HIZI             ;CHECK DBE
        MOV(W30)     DBI,DO               ;DO =DBI IN 30NS
        BRU          DBCK
;
HIZI:   MOV(W30)     #255,DO              ;DO = ALL 1'S IN 30NS
DBCK:   BEQ          DBIN,DBINO,STBCK     ;CHECK DBIN FOR CHANGE
        BEQ          #1,DBIN,DBHI
        MOV(W45)     #1,DBSC              ;DBIN SELFCALL IN 45NS
        MOV(W45)     #0,DBE               ;DBE =0 THEN
        MOV(W20)     #1,NMR               ; RESET NOT MEM. READ
        MOV(W20)     #1,NIOR              ; RESET NOT IO READ
        MOV(W20)     #1,CSC               ; SET CONTROL SELF CALL
        BEQ          NINTF,ICCK           ;CHECK NINTF
        MOV          #0,NINTF
        XOR          #205,DBI,TEMP8       ;SELF CALL INST
        BEQ          TEMP8,ICT
        BRU          STBCK
;
ICT:    MOV          #0,INTCT
        BRU          STBCK
;
ICCK:   CMP          #2,INTCT
        BEQ          Z,NINTC              ;COUNT =2?
        INC          INTCT
        BRU          STBCK
;
NINTC:  MOV(W30)     #1,CSC               ;CALL CONT SIGNAL ROUTINE
        MOV(W30)     #1,NI                ;NINTA =1 IN 30NS
        MOV          #0,INTCT
        BRU          STBCK
;
DBHI:   MOV(W45)     #1,DBSC              ;DBIN SELFCALL IN 45NS
        MOV(W45)     #1,DBE               ;DBE =1 THEN
        XOR          LOM,L,TEMP8          ;LO=1?
        BEQ          TEMP8,STBCK
        BEQ          SPI,STBCK            ;CHECK THE SPECIAL CONTROL INPUT
        MOV(W45)     #255,DO              ;DO = ALL 1'S IN 45NS
STBCK:  MOV          NSTBS,TEMP1          ;CHECK FOR STROBE CHANGE
        COM          TEMP1
        AND          TEMP1,NSTBO,TEMP1
        BEQ          TEMP1,LFCK
```

```
        MOV(W20)    #1,LRSC             ;SCHEDULE L REG SELF CALL
        MOV(W20)    DI,DID              ;PROPAGATED INPUT VALUE
LFCK:   BEQ         LRSC,NWRCK          ;CHECK L REG SELFCALL FLAG
        MOV         #0,LRSC             ;RESET SELFCALL FLAG
        MOV         DID,L               ; UPDATE L
        AND         LOM,L,TEMP8         ;INT ACK?
        BEQ         TEMP8,NEX1
        MOV(W20)    #0,NI
        MOV         #1,NINTF
        BRU         NEX2
;
NEX1:   MOV(W20)    #1,NI
NEX2:   AND         L7M,L,TEMP8         ;MEMR?
        BEQ         TEMP8,NEX3
        MOV(W20)    #0,NMR
        BRU         NEX4
;
NEX3:   MOV(W20)    #1,NMR
NEX4:   XOR         L16M,L,TEMP8        ;IO READ?
        BNE         TEMP8,NEX5
        MOV(W20)    #0,NIR
        BRU         WRCK
;
NEX5:   MOV(W20)    #1,NIR
        BRU         WRCK
;
WRCK:   BEQ         #1,NWR,OSCHED       ;WRITE CONTROL OUTPUTS?
        AND         L1M,L,TEMP8         ;CHECK L1
        BEQ         TEMP8,IOMW
        BRU         OSCHED
;
IOMW:   AND         L4M,L,TEMP8         ;IO OR MEM WRITE?
        BEQ         TEMP8,MW
        MOV(W20)    #1,NMW
        MOV(W20)    #0,NIW
        BRU         OSCHED
;
MW:     MOV(W20)    #1,NIW
        MOV(W20)    #0,NMW
OSCHED:MOV(W20)     #1,CSC
NWRCK:  BEQ         NWR,NWRO,ENCK       ;CHANGE IN WRITE?
        BEQ         NWR,POSWR           ;POS. TRANS. IN WR (NEG IN NWR)
        MOV(W45)    #1,NMW
        MOV(W45)    #1,NIW
        MOV(W45)    #1,CSC
        BRU         ENCK
;
POSWR:  AND         L1M,L,TEMP8         ;WRITE?
        BEQ         TEMP8,IOMW1
        BRU         ENCK
;
IOMW1:  AND         L4M,L,TEMP8         ;IO OR MEM?
        BEQ         TEMP8,MEMW
        MOV(W45)    #0,NIW
        MOV(W45)    #1,NMW
```

```
        BRU         CSC1
;
MEMW:   MOV(W45)    #1,NIW
        MOV(W45)    #0,NMW
CSC1:   MOV(W45)    #1,CSC
        BRU         STCK
;
ENCK:   BNE         NBEN,NBENO,BENCK    ;CHANGE IN BUS ENABLE?
        BEQ         DI,DIO,HOLD         ;CHANGE IN PROCESSOR DATA?
BENCK:  BNE         NBEN,DIS
        AND         L1M,L,TEMP8         ;OUT?
        BEQ         TEMP8,L1R
        BRU         DB1S
;
L1R:    BEQ         #1,NMR,DB1S
STCK:   BEQ         NSTBS,DB1S
DBOUD:  MOV(W30)    DI,DBO              ;DBO=DI IN 30 NS
        BRU         HOLD
;
DIS:    MOV(W30)    #1,NI               ;CONTROL SIGNALS OFF
        MOV(W30)    #1,NMR
        MOV(W30)    #1,NMW
        MOV(W30)    #1,NIR
        MOV(W30)    #1,NIW
        MOV(W30)    #1,CSC
DB1S:   MOV(W30)    #255,DBO            ;DBO = ALL 1'S
HOLD:   BEQ         HLDA,HLDAO,OUT
        BNE         HLDA,HLDHI
        MOV(W10)    #1,LRSC             ;REESTABLISH CONTROL OUTPUTS
        BRU         OUT
;
HLDHI:  MOV(W25)    #1,NI
        MOV(W25)    #1,NMR
        MOV(W25)    #1,NIR
        MOV(W25)    #1,CSC
OUT:    BEQ         CSC,UPDATE
        MOV         #0,CSC              ;RESET SELFCALL FLAG
        MOV(WZ)     NI,NINTA            ; UPDATE CONTROL OUTPUTS
        MOV(WZ)     NMR,NMEMR
        MOV(WZ)     NMW,NMEMW
        MOV(WZ)     NIR,NIOR
        MOV(WZ)     NIW,NIOW
UPDATE: MOV         DBIN,DBINO
        MOV         DBI,DBIO
        MOV         DI,DIO
        MOV         NMR,NMRO
        MOV         NBEN,NBENO
        MOV         HLDA,HLDAO
        MOV         NSTBS,NSTBO
        MOV         #0,EX
L0M:    BYT         #1
L1M:    BYT         #2
L4M:    BYT         #16
L6M:    BYT         #64
L7M:    BYT         #128
```

```
L16M:  BYT
       END          #66
```

```
          REG(8)    ODI,OAI,OBI,D,A,B,C,TEM8
          REG(4)    OCUI,OCLI,TEM4,CU,CL
          REG(3)    PAMTM
          REG(2)    OADI,PBMTM,AD
          REG(1)    C0,C1,C2,C3,C4,C5,C6,C7
          REG(1)    OCI0,OCI1,OCI2,OCI3,OCI4,OCI5,OCI6,OCI7
          REG(1)    ORESET,ONRD,ONWR,ONCS
          REG(1)    TEM1,TEM2,TEM3,WWF,ADF,WRF,ADIH,DIH,RDF,RESF
          PIN  RESET(1),NRD(2),NWR(3),NCS(4),ADI(5,6),DI(7,14),DO(15,22)
          PIN  AI(23,30),AO(31,38),BI(39,46),BO(47,54),CI0(55),CI1(56)
          PIN  CI2(57),CI3(58),CI4(59),CI5(60),CI6(61),CI7(62),CO0(63)
          PIN  CO1(64),CO2(65),CO3(66),CO4(67),CO5(68),CO6(69),CO7(70)
          PIN  CUI(71,74),CLI(75,78),CUO(79,82),CLO(83,86)
          PIN  EX(150),RDSCF(151),RSTSCF(152),WRSCF(153),DHSCF(154)
          PIN  AHSCF(155),RAOF(156),RBOF(157),RCOF(158)
          EVW       W0(0),W1(500),W2(50),W3(250),W5(30),W6(300),W7(400)
          EVW       W8(20),W9(100),W10(150),W11(270)
                              ;
                              ;
          BNE  RESET,ORESET,REST;RESET TRANSITION ?
RESTL:    BNE  RESET,NEX10    ;IF RESET = 1 ,DO NOTHING.
          BNE  RSTSCF,RESSC   ;IF RESET SELF CALL,RESET THE DEVICE.
RSTFR:    BNE  RCOF,RCOSC     ;IF A READ SELF CALL FOR MODE 0 WHILE RD=0,GO
RCOFR:    BNE  RBOF,RBOSC     ;IF AN INPUT TO PORT B OCCURS WHILE RD=0,GO.
RBOFR:    BNE  RAOF,RAOSC     ;IF AN INPUT TO PORT A OCCURS WHILE RD=0, GO.
RAOFR:    BNE  RDSCF,RDSC     ;IF READ SELF CALL, THEN PROCESS A READ.
RDFR:     BNE  WRSCF,WRSC     ;IF WRITE SELF CALL, THEN PROCESS A WRITE.
WRFR:     BNE  DHSCF,DHSC     ;PROCESS A HOLD CHECK ON THE DATA INPUTS (THR).
DHFR:     BNE  AHSCF,AHSC     ;PROCESS A HOLD CHECK ON THE ADDRESS INPUTS (TWA).
NEX1:     BNE  NCS,ONCS,SELECT ;WAS THERE A CHANGE ON CHIP SELECT INPUT?
                              ;IF CHIP SELECT CHANGED,PROCESS IT.
NEX2:     BNE  ADI,OADI,ADD   ;WAS THERE A CHANGE ON ADDRESS INPUTS?
                              ;IF THE ADDRESS INPUTS CHANGED,PROCESS IT.
NEX3:     BNE  NWR,ONWR,WRITE ;WAS THERE A CHANGE ON THE WRITE INPUT?
                              ;IF THE WRITE INPUT CHANGED, PROCESS IT.
NEX4:     BNE  NRD,ONRD,READ  ;WAS THERE A CHANGE ON THE READ INPUT?
                              ;IF THERE WAS A CHANGE ON READ INPUT,PROCESS IT.
NEX5:     BNE  AI,OAI,PORTA   ;WAS THERE A CHANGE ON PORT A INPUT?
                              ;IF PORT A INPUTS CHANGED,PROCESS IT.
NEX6:     BNE  BI,OBI,PORTB   ;WAS THERE A CHANGE IN PORT B INPUTS?
                              ;IF PORT B INPUTS CHANGED,PROCESS IT.
NEX7:     BNE  CUI,OCUI,PORCU ;WAS THERE A CHANGE IN PORT C UPPER INPUTS?
                              ;IF PORT C UPPER INPUTS CHANGED,PROCESS IT.
NEX8:     BNE  CLI,OCLI,PORCL ;WAS THERE A CHANGE ON PORT C LOWER INPUTS?
                              ;IF PORT C LOWER INPUTS CHANGED, PROCESS IT.
NEX9:     BNE  DI,ODI,PORTD   ;WAS THERE A CHANGE ON DATA BUS INPUTS?
                              ;IF THE DATA BUS INPUTS CHANGED, PROCESS IT.
NEX10:    MOV  #0,EX          ;EXIT THE MODULE.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A RESET FUNCTION.                                 *
;*                                                                           *
;*****************************************************************************
REST:     MOV  RESET,ORESET   ;UPDATE RESET REGISTER.
          BEQ  RESET,RSTND    ;RESET CHANGED TO A ONE OR ZERO ?
```

```
     MOV(W1)  #1,RSTSCF      ;IF RESET CHANGED TO 1, SCHEDULE RESET IN 500 NS.
         MOV  #1,RESF        ;SET RESET VALIDITY FLAG.
                             ;RESET RESET VALIDITY FLAG.
         BRU  RESTL          ;PROCEED.
RSTND:   MOV  #0,RESF       ;RESET RESET VALIDITY FLAG.
         BRU  RESTL
RESSC:   MOV  #0,RSTSCF      ;CLEAR SELF CALL FLAG.
         BEQ  RESF,RSTFR     ;IF RESET IS CANCELLED,SKIP TO MAIN PROGRAM
         MOV  #0,RESF        ;AS A RESET WILL NOW BE PROCESSED,CLEAR THE
                             ;RESET VALIDITY FLAG.
         MOV  #0,A           ;RESET REGISTER A TO ZERO.
         MOV  #0,B           ;RESET REGISTER B TO ZERO.
         MOV  #8,CU          ;RESET REGISTER C UPPER TO 8H.
         MOV  #2,CL          ;RESET REGISTER C LOWER TO 2H.
         MOV  #130,C         ;RESET REGISTER C TO 82H.
         MOV  #0,C0          ;RESET REGISTER C BITS TO ZERO.
         MOV  #1,C1          ;EXCEPT C1
         MOV  #0,C2          ;
         MOV  #0,C3          ;
         MOV  #0,C4          ;
         MOV  #0,C5          ;
         MOV  #0,C6          ;
         MOV  #1,C7          ;EXCEPT C7
         MOV(W0)    #255,DO  ;FLOAT ALL OUTPUTS.
         MOV(W0)    #255,AO  ;
         MOV(W0)    #255,BO  ;
         MOV(W0)    #8,CUO   ;
         MOV(W0)    #2,CLO   ;
         MOV(W0)    #0,CO0   ;RESET C PORT OUTPUTS AS IN OSBORNE?
         MOV(W0)    #1,CO1   ;
         MOV(W0)    #0,CO2   ;
         MOV(W0)    #0,CO3   ;
         MOV(W0)    #0,CO4    ;
         MOV(W0)    #0,CO5   ;
         MOV(W0)    #0,CO6   ;
         MOV(W0)    #1,CO7   ;
         MOV  @2,PAMTM       ;RESET PORT A TO INPUT WITHOUT CHANGING MODE.
         OR   #1,PAMTM,@2    ;
         MOV  @3,PBMTM       ;RESET PORT B TO INPUT WITHOUT CHANGING MODE.
         OR   #1,PBMTM,@3    ;
         MOV  #1,@4          ;RESET PORT C UPPER TO INPUT.
         MOV  #1,PBMTM,@4    ;RESET PORT C LOWER TO INPUT.
         BRU  RSTFR          ;RETURN TO MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION OF CODE PROCESSES A CHANGE ON THE CHIP SELECT INPUT PIN.     *
;*                                                                           *
;*****************************************************************************
SELECT:  MOV  NCS,ONCS       ;UPDATE VALUE OF ONCS.
         MOV  #1,ADIH        ;RECORD CHANGE IN ADDRESS INPUTS.
         BRU  NEX2           ;
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION OF CODE PROCESSES A CHANGE ON THE ADDRESS INPUT PINS.        *
;*                                                                           *
```

```
;*****************************************************************************
ADD:    MOV  ADI,OADI       ;TEMPORARY EXIT.
        MOV  #1,ADIH        ;RECORD CHANGE IN ADDRESS INPUTS.
        BRU  NEX3           ;
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION OF CODE PROCESSES A CHANGE ON THE WRITE INPUT PIN (NWR).    *
;*                                                                           *
;*****************************************************************************
WRITE:  MOV  NWR,ONWR       ;UPDATE VALUE OF NWR.
        BNE  NWR,WRGO       ;IF NWR HAS CHANGED TO 1, SET UP THE WRITE PROCESS.
        BEQ  NRD,NEX4       ;IF READ IS ALSO SELECTED, DO NOTHING.
        BNE  NCS,NEX4       ;IF CHIP IS NOT SELECTED, DO NOTHING.
                            ;IF NWR HAS CHANGED TO 0, PREPARE FOR CHECKS ON
                            ;CONSTANT ADDRESS AND CHIP SELECT INPUTS.
        MOV(W9)  DI,D       ;SCHEDULE DATA INPUT TRANSFER IN 100 NS.
        MOV  #0,ADIH        ;CLEAR A FLAG. ANY CHANGE OF ADDRESS INPUTS WILL
                            ;SET THE FLAG. A TEST FOR 1 WILL THEN DETERMINE
                            ;IF THE ADDRESS INPUTS CHANGED DURING THE WRITE
                            ;OPERATION.
        MOV  #1,WRF         ;SET THE WRITE FLAG.  THIS FLAG WILL BE CLEARED
                            ;IF ANY ILLEGAL ACTION OCCURS DURING THE WRITE
                            ;OPERATION. BEFORE PERFORMING THE FINAL WRITE
                            ;THIS FLAG WILL BE CHECKED. THE WRITE WILL OCCUR
                            ;ONLY IF THIS FLAG STAYS EQUAL TO 1 DURING THE
                            ;ENTIRE WRITE OPERATION.
        BRU  NEX4           ;RETURN TO THE MAIN PROGRAM.
WRGO:   MOV  #0,DIH         ;CLEAR A FLAG.  THIS FLAG WILL BE SET IF ANY
                            ;CHANGE OCCURS ON THE DATA INPUTS DURING THE
                            ;REQUIRED HOLD TIME.
        BEQ  WRF,NEX4       ;IF WRITE HAS BEEN CANCELLED, DO NOTHING.
        BNE  NCS,NEX4       ;IF CHIP IS NOT SELECTED, THEN DO NOTHING.
        MOV(W5)  #1,DHSCF   ;SCHEDULE SELF CALL IN 30 NS. TO CHECK FOR
                            ;REQUIRED HOLD TIME ON DATA INPUTS.
        MOV(W8)  #1,AHSCF   ;SCHEDULE SELF CALL IN 20 NS. TO CHECK FOR
                            ;REQUIRED HOLD TIME ON ADDRESS INPUTS.
        BRU  NEX4           ;RETURN TO THE MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES THE SELF CALLS ASSOCIATED WITH THE WRITE OPERATION. *
;*                                                                           *
;*****************************************************************************
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION CHECKS FOR THE REQUIRED HOLD TIME ON THE DATA INPUTS        *
;*  FOLLOWING THE RISING EDGE OF NWR.                                        *
;*                                                                           *
;*****************************************************************************
DHSC:   MOV  #0,DHSCF       ;CLEAR THE SELF CALL FLAG.
        BNE  DIH,HVIO       ;IF THE DATA INPUTS CHANGED DURING THE REQUIRED
                            ;HOLD TIME, CANCEL THE WRITE OPERATION.
        BRU  WRRDY          ;OTHERWISE, PROCEED.
HVIO:   MOV  #0,WRF         ;CLEAR FLAG TO INDICATE CANCELLATION OF WRITE.
WRRDY:  BEQ  WRF,DHFR       ;IF WRF=0,CANCEL THE WRITE COMPLETELY.
        MOV(W11)  #1,WRSCF  ;WRF=1.WRITE IS ALL RIGHT. SCHEDULE WRITE
```

```
                        ;SELF CALL IN 270 NS.
        BRU  DHFR       ;RETURN TO THE MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION CHECKS FOR CONSTANT ADDRESS INPUTS AND CHIP SELECT INPUTS   *
;*  DURING THE WRITE OPERATION.                                              *
;*                                                                           *
;*****************************************************************************
AHSC:   MOV  #0,AHSCF   ;CLEAR THE SELF CALL FLAG.
        BNE  ADIH,AVIO  ;IF THE ADDRESS INPUTS CHANGED DURING THE WRITE
                        ;OPERATION,CANCEL THE WRITE OPERATION.
        MOV  ADI,@1     ;SAVE THE ADDRESS INPUT  CODE IN INDEX REGISTER.
        BRU  NEX1       ;OTHERWISE, PROCEED.
AVIO:   MOV  #0,WRF     ;CLEAR A FLAG TO INDICATE CANCELLATION OF THE
                        ;WRITE OPERATION.
        BRU  NEX1       ;RETURN TO THE MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES THE WRITE OPERATION IF ALL REQUIRED CONDITIONS ARE *
;*  MET.                                                                     *
;*                                                                           *
;*****************************************************************************
WRSC:   MOV  #0,WRSCF   ;CLEAR SELF CALL FLAG.
        BNE  NRD,WRSCC  ;IF NRD IS 1, THEN CONTINUE.
        MOV  #0,WRF     ;OTHERWISE, CANCEL BOTH THE READ AND THE WRITE.
        MOV  #0,RDF     ;
        BRU  WRFR       ;RETURN TO MAIN PROGRAM.
WRSCC:  MOV  #0,WRF     ;TURN OFF WRITE FLAG SINCE THE WRITE WILL NOW GO
                        ;TO COMPLETION.
        BRU  ATAB@1     ;INDEXED BRANCH.  SEE ATAB DECLARATION BELOW.
                        ;
                        ;
                        ;BRANCH TABLE TO SELECT THE PORT TO E WRITTEN INTO.
                        ;
ATAB:   BYT  PAW        ;WRITE TO PORT A.
        BYT  PBW        ;WRITE TO PORT B.
        BYT  PCW        ;WRITE TO PORT C.
        BYT  CTLW       ;WRITE IN CONTROL MODE OR BIT RESET MODE.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION SELECTS THE MODE FOR THE PORT A WRITE OPERATION.            *
;*                                                                           *
;*****************************************************************************
PAW:    BRU  WATAB@2    ;USE INDEX REG TO SELECT THE MODE OF PORT A.
                        ;
                        ;
                        ;BRANCH TABLE FOR PORT A MODE SELECTION.
                        ;
WATAB:  BYT  PAM0W      ;WRITE TO PORT A IN MODE 0.
        BYT  WRFR       ;ATTEMPT TO WRITE TO PORT A WHEN IN INPUT MODE.
        BYT  PAM1W      ;WRITE TO PORT A IN MODE 1.
        BYT  WRFR       ;ATTEMPT TO WRITE TO PORT A WHEN IN INPUT MODE.
        BYT  PAM2W      ;WRITE TO PORT A IN MODE 2.
```

```
        BYT  WRFR           ;ATTEMPT TO WRITE TO PORT A WHEN IN INPUT MODE.
        BYT  PAM2W          ;WRITE TO PORT A IN MODE 2.
        BYT  WRFR           ;ATTEMPT TO WRITE TO PORT A WHEN IN INPUT MODE.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION WRITES TO PORT A IN MODE 0.                                 *
;*                                                                           *
;*****************************************************************************
PAM0W:  MOV  D,A            ;WRITE TO PORT A.
        MOV(W2)  A,AO       ;SCHEDULE THE OUTPUT CHANGE IN 50 NS.
        BRU  WRFR           ;RETURN TO MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION  WRITES TO PORT A IN MODE 1.                                *
;*                                                                           *
;*****************************************************************************
PAM1W:  BRU  WRFR           ;TEMPORARY EXIT.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION WRITES TO PORT A IN MODE 2.                                 *
;*                                                                           *
;*****************************************************************************
PAM2W:  BRU  WRFR           ;TEMPORARY EXIT.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A WRITE TO PORT B.                                *
;*                                                                           *
;*****************************************************************************
PBW:    BRU  WBTAB@3        ;BRANCH TO PORT B WRITE ROUTINE.
                            ;
                            ;BRANCH TABLE FOR PORT B WRITE.
                            ;
WBTAB:  BYT  PBM0W          ;PROCESS A WRITE TO PORT B IN MODE 0.
        BYT  WRFR           ;ERROR CONDITION.
        BYT  PBM1W          ;WRITE TO PORT B IN MODE 1.
        BYT  WRFR           ;ERROR CONDITION.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A WRITE OPERATION TO PORT B IN MODE 0.            *
;*                                                                           *
;*****************************************************************************
;
PBM0W:  MOV  D,B            ;WRITE TO PORT B.
        MOV(W2)  B,BO       ;SCHEDULE THE OUTPUT CHANGE IN 50 NS.
        BRU  WRFR           ;RETURN TO MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A WRITE OPERATION TO PORT B IN MODE 1.            *
;*                                                                           *
;*****************************************************************************
```

```
;
PBM1W:   BRU  WRFR          ;TEMPORARY EXIT.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A WRITE OPERATION TO PORT C.                      *
;*                                                                           *
;*****************************************************************************
PCW:     MOV  @2,PAMTM      ;GET INDEX VALUE FOR COMPUTATION.
         AND  #6,PAMTM,PAMTM ;STRIP DIRECTION BIT FOR PORT A.
         BNE  PAMTM,WRFR    ;IF PORT A IS NOT MODE 0, QUIT.
         MOV  @3,PBMTM      ;GET INDEX VALUE FOR COMPUTATION.
         AND  #2,PBMTM,PBMTM ;STRIP OFF DIRECTION BIT FOR PORT B.
         BNE  PBMTM,WRFR    ;IF PORT B IS NOT MODE 0, THEN QUIT.
         BRU  CON@4         ;IF PORT C UPPER IS INPUT, SKIP THE WRITE.
CON:     BYT  GCON          ;SKIP TO LOWER.
         BYT  LOW           ;LOAD UPPER.
GCON:    IDX  D(4),4,7      ;READ  OFF UPPER FOUR BITS.
         MOV  @7,CU         ;OUTPUT UPPER C BITS.
         MOV(W2)  CU,CUO    ;SCHEDULE THE OUTPUT CHANGE IN 50 NS.
LOW:     BRU  CON1@5        ;IF PORT C LOWER IS INPUT, SKIP THE WRITE.
CON1:    BYT  GCON1         ;SKIP TO EXIT.
         BYT  PCWEX         ;LOAD LOWER.
GCON1:   IDX  D(0),4,7      ;READ OFF THE LOWER C BITS.
         MOV  @7,CL         ;OUTPUT THE LOWER C PORT BITS.
         MOV(W2)  CL,CLO    ;SCHEDULE THE PIN OUTPUTS IN 50 NS.
PCWEX:   IDX  CU(0),4,7     ; FORM C = CU,CL
         MOV  @7,C          ;
         ROR  C
         ROR  C             ;
         ROR  C             ;
         ROR  C             ;
         IDX  CL(0),4,7     ;
         MOV  @7,TEM8       ;
         OR   C,TEM8,C      ;
         BRU  WRFR          ;RETURN TO MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A WRITE TO THE CONTROL PORT.                      *
;*  THIS IS ACCOMPLISHED BY LOADING INDEX REGISTERS WITH THE STATUS OF EACH   *
;*  PORT.                                                                    *
;*                                                                           *
;*            INDEX REGISTER PORT STATUS                                     *
;*                                                                           *
;*            2              A                                               *
;*            3              B                                               *
;*            4              C UPPER                                         *
;*            5              C LOWER                                         *
;*                                                                           *
;*****************************************************************************
CTLW:    IDX  D(7),1,6      ;INDEX REG 6 HOLDS BIT 7 OF CONTROL WORD.
                            ;BIT 7 = 0 IMPLIES A BIT OPERATION TO PORT C.
                            ;BIT 7 = 1 IMPLIES A CHANGE IN CONTROL STATUS.
         BRU  CTAB@6        ;SELECT THE BRANCH ADDRESS FOR PROCESSING.
```

```
                                ;
                                ;
                                ;BRANCH TABLE FOR CONTROL WORD PROCESSING.
                                ;
                                ;
CTAB:     BYT  BIT              ;PROCESS A BIT TRANSFER TO PORT C.
          BYT  CTLWR            ;UPDATE STATUS WORD.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A BIT TRANSFER TO PORT C.                         *
;*                                                                           *
;*****************************************************************************
BIT:      IDX  D(0),1,8         ;@8=0 MEANS RESET.  @8=1 MEANS SET.
          IDX  D(1),3,7         ;CODE FOR BIT TO BE ALTERED.
          BRU  BITAB@8          ;DECIDE WHETHER TO SET OR RESET.
BITAB:    BYT  RES              ;RESET.
          BYT  SET              ;SET THE BIT.
RES:      BIR  @7,C             ;RESET THE APPROPRIATE BIT OF REGISTER C.
          BRU  BSR              ;SCHEDULE THE OUTPUT CHANGE.
SET:      BIS  @7,C             ;SET THE BIT
BSR:      IDX  C(0),4,8         ;COMPUTE THE LOWER 4 BITS.
          MOV  @8,CL            ;CL=LO NIBBLE OF C.
          MOV(W2)  @8,CLO       ;SCHEDULE THE OUTPUT CHANGE IN 50 NS.
          IDX  C(4),4,8         ;COMPUTE THE UPPER 4 BITS
          MOV  @8,CU            ;CU= HI NIBBLE OF C.
          MOV(W2)  @8,CUO       ;SCHEDULE THE OUTPUTS IN 50 NS.
          BRU  WRFR             ;RETURN TO MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A CONTROL WORD WRITE TO CHANGE THE STATUS OF PORTS. *
;*                                                                           *
;*****************************************************************************
CTLWR:    IDX  D(4),3,2         ;PUT PORT A STATUS IN INDEX REG 2.
          IDX  D(1),2,3         ;PUT PORT B STATUS IN INDEX REG 3.
          IDX  D(3),1,4         ;PUT PORT C UPPER STATUS IN INDEX REG 4.
          IDX  D(0),1,5         ;PUT PORT C LOWER STATUS IN INDEX REG 5.
          MOV  #0,A             ;A CHANGE IN PORT MODE WILL CAUSE ALL OUTPUT PORTS
          MOV  #0,B             ;TO BE CLEARED.
          MOV  #0,CL            ;
          MOV  #0,CU            ;
          MOV  #0,C             ;
          MOV  #0,OAI           ;
          MOV  #0,OBI           ;
          MOV  #0,OCLI          ;
          MOV  #0,OCUI          ;
          MOV  #0,C0            ;
          MOV  #0,C1            ;
          MOV  #0,C2            ;
          MOV  #0,C3            ;
          MOV  #0,C4            ;
          MOV  #0,C5            ;
          MOV  #0,C6            ;
          MOV  #0,C7            ;
          MOV  #0,OCIO          ;
```

```
          MOV  #0,OCI1          ;
          MOV  #0,OCI2          ;
          MOV  #0,OCI3          ;
          MOV  #0,OCI4          ;
          MOV  #0,OCI5          ;
          MOV  #0,OCI6
          MOV  #0,OCI7          ;
          MOV(W2)  #0,AO          ;
          MOV(W2)  #0,BO
          MOV(W2)  #0,CLO         ;
          MOV(W2)  #0,CUO         ;
          MOV(W2)  #0,CO0
          MOV(W2)  #0,CO1
          MOV(W2)  #0,CO2         ;
          MOV(W2)  #0,CO3         ;
          MOV(W2)  #0,CO4         ;
          MOV(W2)  #0,CO5         ;
          MOV(W2)  #0,CO6         ;
          MOV(W2)  #0,CO7         ;
          BRU  WRFR             ;RETURN TO MAIN PROGRAM.
;
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A READ OPERATION.                                 *
;*                                                                           *
;*****************************************************************************
READ:     MOV  NRD,ONRD         ;SAVE NEW VALUE OF NRD.
          BEQ  NRD,READ1        ;IF NRD HAS CHANGED TO 0,SET UP THE READ.
          MOV(W2)  #255,DO      ;FLOAT THE DATA BUS IF NRD CHANGED TO 1.
          BRU  NEX5             ;RETURN TO MAIN PROGRAM.
READ1:    BEQ  NWR,READ2        ;IF A WRITE IS ALSO SCHEDULED, CANCEL BOTH.
          BNE  NCS,NEX5         ;IF CHIP IS NOT SELECTED, THEN DO NOTHING.
          MOV  #1,RDF           ;SET FLAG TO BEGIN READ SEQUENCE. IF ANYTHING
                                ;GOES WRONG DURING THE READ, THIS FLAG WILL BE
                                ;BE CLEARED.  THE READ SELF CALL WILL NOT COMPLETE
                                ;THE READ OPERATION UNLESS THIS FLAG REMAINS SET
                                ;DURING THE ENTIRE READ SEQUENCE.
          MOV  #0,ADIH          ;CLEAR A FLAG TO CHECK FOR ADDRESS CHANGES.
          IDX  ADI(0),2,1       ;LOAD PORT CODE INTO IDR 1.
          BRU  RDTAB@1          ;BRANCH TO APPROPRIATE READ SUBROUTINE.
RDTAB:    BYT  PAR              ;READ FROM PORT A.
          BYT  PBR              ;READ FROM PORT B.
          BYT  PCR              ;READ FROM PORT C.
          BYT  NEX5             ;READ ERROR, DO NOTHING.
READ2:    MOV  #0,WRF           ;CANCEL THE WRITE OPERATION.
          BRU  NEX5             ;RETURN TO MAIN PROGRAM WITHOUT READING.
;
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A READ FROM PORT A.                               *
;*                                                                           *
;*****************************************************************************
;
PAR:      BRU  RATAB@2          ;BRANCH TO APPROPRIATE PORT A READ ROUTINE.
RATAB:    BYT  NEX5             ;READ ERROR. PORT A IS DEFINED AS OUTPUT.
```

```
          BYT   PAMOR           ;READ FROM PORT A IN MODE 0.
          BYT   NEX5            ;READ ERROR. PORT A IS DEFINED AS OUTPUT.
          BYT   PAM1R           ;READ FROM PORT A IN MODE 1.
          BYT   NEX5            ;READ ERROR. PORT A IS DEFINED AS OUTPUT.
          BYT   PAM2R           ;READ FROM PORT A IN MODE 2.
          BYT   PAM2R           ;
          BYT   PAM2R           ;
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A READ FROM PORT A IN MODE 0.                     *
;*                                                                           *
;*****************************************************************************
;
PAMOR:    MOV(W10)  A,D         ;READ PORT A TO DATA BUS.
          MOV(W10)  #1,RDSCF    ;COMPLETE THE READ IN 150 NS.
          BRU   NEX5            ;RETURN TO MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A READ FROM PORT A IN MODE 1.                     *
;*                                                                           *
;*****************************************************************************
;
PAM1R:    BRU   NEX6            ;TEMPORARY EXIT.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A READ FROM PORT A IN MODE 2.                     *
;*                                                                           *
;*****************************************************************************
;
PAM2R:    BRU   NEX6            ;TEMPORARY EXIT.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A READ FROM PORT B.                               *
;*                                                                           *
;*****************************************************************************
;
PBR:      BRU   RBTAB@3         ;BRANCH TO THE APPROPRIATE ROUTINE.
RBTAB:    BYT   NEX5            ;READ ERROR.  PORT B IS DEFINED AS OUTPUT.
          BYT   PBMOR           ;READ FROM PORT B IN MODE 0.
          BYT   NEX5            ;READ ERROR.  PORT B IS DEFINED AS OUTPUT.
          BYT   PBM1R           ;READ FROM PORT B IN MODE 1.
PBMOR:    MOV(W10)  B,D         ;SCHEDULE TRANSFER FROM B TO D IN 150 NS.
          MOV(W10)  #1,RDSCF    ;SCHEDULE THE READ SELF CALL IN 150 NS.
          BRU   NEX5            ;RETURN TO MAIN PROGRAM.
PBM1R:    BRU NEX5              ;TEMPORARY EXIT.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A READ FROM PORT C.                               *
;*                                                                           *
;*****************************************************************************
```

```
;
PCR:    IDX CU(0),4,7       ;LOAD CU INTO I.R. 7
        MOV @7,C            ;MOVE PORT C UPPER TO C
        ROR  C              ;ROTATE LOWER NIBBLE TO UPPER 4 BIT POSITIONS.
        ROR  C              ;
        ROR  C              ;
        ROR  C              ;
        IDX CL(0),4,7       ;GET LOWER NIBBLE FROM CL.
        MOV @7,TEMB         ;
        OR TEMB,C,C         ;
        MOV(W10)  C,D       ;SCHEDULE THE TRANSFER TO PORT D IN 150 NS.
        MOV(W10)  #1,RDSCF  ;SCHEDULE THE READ SELF CALL IN 150 NS.
        BRU  NEX5           ;
;*****************************************************************************
;*                                                                           *
;*   THIS SECTION PROCESSES A READ SELF CALL.                                *
;*                                                                           *
;*****************************************************************************
;
RDSC:   MOV  #0,RDSCF       ;CLEAR FLAG
        BEQ  RDF,RDFR       ;IF READ HAS BEEN CANCELLED, DO NOTHING.
        MOV(W2)   D,DO      ;SCHEDULE A TRANSFER TO PORT D OUTPUTS IN 50 NS.
        MOV  #0,RDF         ;TERMINATE READ OPERATION.
        BRU  RDFR           ;RETURN TO MAIN PROGRAM.
;*****************************************************************************
;*                                                                           *
;*  THIS SECTION PROCESSES A CHANGE IN PORT A INPUTS.                        *
;*                                                                           *
;*****************************************************************************
PORTA:  MOV  AI,OAI         ;UPDATE PORT A VALUE.
        BRU  INATB@2        ;BRANCH TO THE APPROPRITATE ROUTINE.
INATB:  BYT  NEX6           ;INPUT ERROR.  PORT A IS OUTPUT.
        BYT  INAM0          ;INPUT TO PORT A IN MODE ZERO.
        BYT  NEX6           ;INPUT ERROR.  PORT A IS OUTPUT.
        BYT  INAM1          ;INPUT TO PORT A IN MODE 1.
        BYT  NEX6           ;INPUT ERROR.  PORT A IS OUTPUT.
        BYT  INAM2          ;INPUT TO PORT A IN MODE 2.
        BYT  NEX6           ;INPUT ERROR.  PORT A IS OUTPUT.
        BYT  INAM2          ;INPUT TO PORT A IN MODE 2.
;
;  INPUT TO PORT A IN MODE 0.
;
INAM0:  MOV(W2)   AI,A      ;SCHEDULE A TRANSFER TO PORT A IN 50 NS.
        BNE  NRD,NEX6       ;TERMINATE TRANSFER IF NRD=1.
        MOV  @1,PBMTM       ;MOVE ADDRESS SELECT CODE TO TEMPORARY REGISTER.
        BNE  PBMTM,NEX6     ;IF ADDRESS CODE IS NOT PORT A, TERMINATE PROCESS.
        MOV  #1,RAOF        ;SCHEDULE A TRANSFER FROM A TO D IN 50 NS.
        BRU NEX6            ;TERMINATE THE INPUT PROCESS.
;
;
;  INPUT TO PORT A WHILE RD=0.
;
RAOSC:  MOV  #0,RAOF        ;CLEAR FLAG.
```

```
          BNE  NRD,RAOFR       ;IF NRD=1, DO NOTHING.
          MOV(W10)  A,D        ;SCHEDULE TRANSFER FROM A TO D IN 150 NS.
          MOV(W10)  #1,RDSCF   ;SCHEDULE READ SEFL CALL IN 150 NS.
          BRU  RAOFR           ;RETURN TO MAIN PROGRAM.
INAM1:    BRU  NEX6            ;TEMPORARY EXIT.
INAM2:    BRU  NEX6            ;TEMPORARY EXIT.
;*************************************************************************
;*                                                                       *
;*  THIS SECTION PROCESSES A CHANGE ON PORT B INPUTS.                    *
;*                                                                       *
;*************************************************************************
PORTB:    MOV  BI,OBI          ;UPDATE VALUE OF PORT B.
          BRU  INBTB@3         ;BRANCH TO THE APPROPRIATE ROUTINE.
INBTB:    BYT  NEX7            ;INPUT ERROR.  PORT B IS OUTPUT.
          BYT  INBM0           ;INPUT TRANSFER TO PORT B IN MODE 0.
          BYT  NEX7            ;INPUT ERROR.  PORT B IS OUTPUT.
          BYT  INBM1           ;INPUT TRANSFER TO PORT B IN MODE 1.
;
;   INPUT TRANSFER TO PORT B IN MODE 0.
;
INBM0:    MOV(W2)   BI,B       ;SCHEDULE THE TRANSFER TO PORT B IN 50 NS.
          BNE  NRD,NEX7        ;IF NRD=1, TERMINATE THE TRANSFER.
          BNE  #1,@1,NEX7      ;IF PORT B IS NOT SELECTED, TERMINATE PROCESS.
          MOV(W2)   #1,RBOF    ;SCHEDULE A TRANSFER FROM B TO D IN 50 NS.
          BRU  NEX7            ;TEMPORARY EXIT.
;
;   INPUT TRANSFER TO PORT B IN MODE 1.
;
INBM1:    BRU  NEX7            ;TEMPORARY EXIT.
;
;   TRANSFER TO PORT B IN MODE 0 WHILE NRD=0.
;
RBOSC:    MOV  #0,RBOF         ;CLEAR FLAG.
          BNE  NRD,RBOFR       ;IF NRD=1, DO NOTHING.
          MOV(W10)  B,D        ;TRANSFER PORT B TO PORT D IN 150 NS.
          MOV(W10)  #1,RDSCF   ;SCHEDULE THE OUTPUT TO DATA BUS IN 150 NS.
          BRU  RBOFR           ;RETURN TO MAIN PROGRAM.
;
;*************************************************************************
;*                                                                       *
;*  THIS SECTION PROCESSES A CHANGE IN PORT C UPPER INPUTS.              *
;*                                                                       *
;*************************************************************************
PORCU:    MOV  CUI,OCUI        ;UPDATE PORT C UPPER.
          BRU  INCUT@4         ;BRANCH TO THE APPROPRIATE ROUTINE.
INCUT:    BYT  NEX8            ;INPUT ERROR. PORT C UPPER IS OUTPUT.
          BYT  INCU            ;INPUT PORT C UPPER.
                               ;
                               ;INPUT FORM PORT C CONTINUES.
                               ;
INCU:     BEQ  #0,@2,CUCON     ;IF PORT A IS MODE 0 , CONTINUE PROCESSING.
          BEQ  #1,@2,CUCON     ;
          BRU  NEX8            ;OTHERWISE, TERMINATE TRANSFER TO PORT C UPPER.
CUCON:    MOV(W2)   CUI,CU     ;SCHEDULE A TRANSFER TO PORT C UPPER IN 50 NS.
          BNE  NRD,NEX8        ;IF RD=1, TERMINATE THE DATA TRANSFER.
```

```
        BNE  #2,@1,NEX8      ;IF REGISTER C IS NOT SELECTED, TERMINATE PROCESS.
        MOV(W2)  #1,RCOF     ;BEGIN A TRANSFER FROM PORT C TO D IN 50 NS.
        BRU  NEX8            ;RETURN TO MAIN PROGRAM.
;
;*************************************************************************
;*                                                                       *
;*  THIS SECTION PROCESSES A CHANGE ON PORT C LOWER INPUTS.              *
;*                                                                       *
;*************************************************************************
PORCL:  MOV  CLI,OCLI        ;UPDATE VALUE OF PORT C LOWER.
        BRU  INCLT@5         ;BRANCH TO THE APPROPRIATE SUBROUTINE.
INCLT:  BYT  NEX9            ;INPUT ERROR.  PORT C LOWER IS OUTPUT.
        BYT  INCL            ;INPUT PORT C LOWER.
INCL:   BEQ  #0,@2,CLCON     ;CONTINUE ONLY IF PORT A IS MODE 0.
        BEQ  #1,@2,CLCON     ;
        BRU  NEX9            ;OTHERWISE, TERMINATE THE PORT C INPUT ROUTINE.
CLCON:  BEQ  #0,@3,CLON      ;CONTINUE ONLY IF PORT B IS MODE 0.
        BEQ  #1,@3,CLON      ;
        BRU  NEX9            ;OTHERWISE, TERMINATE THE PORT C INPUT ROUTINE.
CLON:   MOV(W2)  CLI,CL      ;TRANSFER DATA TO PORT C LOWER IN MODE 0.
        BNE  NRD,NEX9        ;IF NRD=1, TERMINATE THE PROCESS.
        BNE  #2,@1,NEX9      ;
        MOV(W2)  #1,RCOF     ;BEGIN TRANSFER OF C TO D IN 50 NS.
        BRU  NEX9            ;RETURN TO MAIN PROGRAM.
                             ;
                             ;SELF CALL TO TRANSFER REGISTER C TO D.
                             ;
RCOSC:  MOV  #0,RCOF         ;CLEAR FLAG.
        BNE  NRD,RCOFR       ;IF NRD=1, TERMINATE THE TRANSFER.
        IDX CU(0),4,7        ;MOV CU TO I.R.7
        MOV @7,C             ;MOVE PORT C UPPER TO C
        ROR  C               ;ROTATE 4 POSITIONS.
        ROR  C               ;
        ROR  C               ;
        ROR  C               ;
        IDX CL(0),4,7        ;MOVE C LOWER TO LOW NIBBLE OF C
        MOV @7,TEMB
        OR   C,TEMB,C        ;
        MOV(W10)  C,D        ;SCHEDULE TRANSFER FROM CU,CL TO D IN 150 NS.
        MOV(W10)  #1,RDSCF   ;SCHEDULE COMPLETION OF READ OPERATION.
        BRU  RCOFR           ;RETURN TO MAIN PROGRAM.
;
;*************************************************************************
;*                                                                       *
;*  THIS SECTION PROCESSES A CHANGE ON PORT D INPUTS.                    *
;*                                                                       *
;*************************************************************************
PORTD:  MOV  DI,ODI          ;UPDATE VALUE OF PORT D.
        MOV  #1,DIH          ;SET FLAG TO INDICATE THAT DATA INPUTS HAVE
                             ;CHANGED.  THIS FLAG IS CHECKED LATER TO TEST
                             ;FOR REQUIRED DATA HOLD TIMES.
        BNE  NWR,NEX10       ;IF NWR IS NOT = 0, DO NOT INPUT DATA.
        BNE  NCS,NEX10       ;IF CHIP IS NOT SELECTED, DO NOTHING.
        MOV(W9)   DI,D       ;SCHEDULE THE DATA INPUTS IN 100 NS.
        BRU  NEX10           ;RETURN TO MAIN PROGRAM.
```

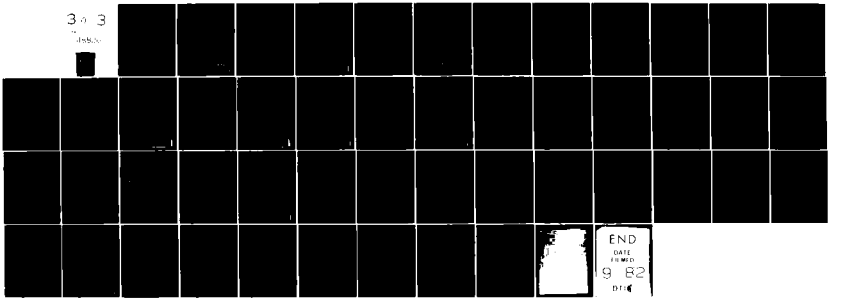
END

```
          REG(8) RP,RS,TP,TS,MR,CR,STATINP,STATUS,WDI,DIO,TEMP8,TEMPA
          REG(4) OCTR,ICTR,TEMP4
          REG(1) MODE,ORESET,TEMP1,NDSRO,PAR,RXDO,TPF,ISHF,IPAR
          REG(1) IBRK,BRKO,NCTSO,NRDO,NCSO,NWRO,SYNBDO,SBD
          PIN DI(1,8),DO(9,16),RESET(17),CLK(18),NWR(19),NRD(20)
          PIN NCS(21),CND(22),NTXC(23),NRXC(24),RXD(25),TXD(26)
          PIN RXRDY(27),TXRDY(28),TXEMPTY(29),NDTR(30),NRTS(31)
          PIN NDSR(32),NCTS(33)
          PIN EX(150),WSCF(151),ISSCF(152),ISCF(153),OSCF(154)
          EVW W0(0),W1(150),W2(50),W3(100),W5(250),W6(2000),W7(7000)
          EVR R0(0)
          EVW W8(13020),W9(6510),W10(26040),W11(52080),W12(104160)
     BNE RESET,NEX75          ;CHECK RESET
     BEQ ORESET,DATCK         ;RESET HI TO LO?
     MOV #1,MODE              ;NEXT CONTROL WORD IS A MODE WORD
DATCK:BEQ DI,DIO,NEX1         ;CHANGE INPUT DATA?
     MOV(W5) DI,WDI           ;SCHEDULE INTERNAL DATA STORAGE
NEX1:BEQ NCS,NEX2             ;CHIP SELECTED?
     BEQ NCS,NCSO,NEX6        ;CHANGE IN CHIP SELECT?
     MOV(W2) #255,DO          ;NO,SET DO TO HI Z IN 50 NS
     BRU NEX6
NEX2:BEQ NRD,NRDO,NEX3        ;CHANGE IN NRD?
     BEQ NRD,CNDCK            ;CHECK NRD
     MOV(W2) #255,DO          ;DO = HIZ IN 50 NS
     BRU NEX3
CNDCK:BEQ CND,RDATA           ;CHECK CND
     MOV(W3) STATUS,DO        ;OUTPUT STATUS IN 100N
     BRU NEX3
RDATA:MOV(W3) RP,DO           ;OUTPUT DATA IN 100NS
     MOV(W1) #0,RXRDY         ;RXRDY =0 IN 150NS
     BIR #1,STATINP           ;RXRDY STATUS =0
     MOV(W7) STATINP,STATUS   ;UPDATE STATUS IN 14TCY
NEX3:BEQ NWR,NWRO,NEX4        ;CHANGE IN NWR?
     BNE NWR,OCS              ;CHECK NWR
     MOV(W5) #1,WSCF          ;SCHEDULE WRITE SELFCALL
     BNE  CND,NEX4            ;COMMAND OR DATA?
     BIR #0,STATINP           ;TXRDY STATUS=0
     MOV(W7) STATINP,STATUS   ;UPDATE STATUS IN 14TCY
     JSR CTXRD               ;UPDATE TXRDY PIN.
     BRU NEX4
OCS: AND R1M,CR,TEMP8         ;CHECK DTR
     BEQ TEMP8,DTRR
     MOV(W6) #0,NDTR          ;NDTR=0 IN 2US
     BRU CRTS
DTRR:MOV(W6) #1,NDTR          ;NDTR=1 IN 2US
CRTS:AND R5M,CR,TEMP8         ;CHECK RTS
     BEQ TEMP8,RTSR
     MOV(W6) #0,NRTS          ;NRTS=0 IN 2US
     BRU NEX4
RTSR:MOV(W6) #1,NRTS          ;NRTS=1 IN 2US
NEX4:BEQ WSCF,NEX5            ;CHECK WRITE SELF CALL FLAG
     MOV #0,WSCF
     BNE NWR,NEX5             ;NWR STILL LOW?
     BNE CND,CNDW             ;CHECK CND
     MOV WDI,TP               ;MOVE DATA TO TRANSMIT REG
```

```
      MOV #1,TPF              ;SET TP FLAG TO 1
      BIS #0,STATINP          ;SET TXRDY STATUS BIT.
      MOV(W7) STATINP,STATUS  ;
      JSR CTXRD               ;UPDATE TXRDY.
      BRU NEX5
CNDW: BNE MODE,MODE1          ;CHECK MODE FLAG
      MOV WD1,TP              ;TP IS EFFECTED TOO
      MOV TP,CR               ;SET UP CONTROL REG
      AND R4M,CR,TEMP8        ;CHECK ERROR RESET FLAG
      BEQ TEMP8,INRES
      BIR #3,STATINP          ;RESET PARITY,OVERRUN AND FRAMING ERRORS
      BIR #4,STATINP
      BIR #5,STATINP
      MOV(W7) STATINP,STATUS
INRES:AND R6M,CR,TEMP8        ;CHECK INTERNAL RESET FLAG
      BEQ TEMP8,BRK
      MOV #1,MODE             ;NEXT COMMAND WILL BE A MODE COMMAND
BRK:  AND R3M,CR,TEMP8        ;BREAK CHAR?
      BEQ TEMP8,NOBR
      MOV #1,BRKO             ;SET UP BREAK
      MOV(W0) #0,TXD          ;START OUTPUTTING BREAK.
      MOV #0,TPF              ;KILL PREVIOUSLY WRITTEN DATA
      BRU TXR11
NOBR: MOV #0,BRKO             ;SET UP NO BREAK
      MOV(W0) #1,TXD          ;
TXR11:JSR CTXRD               ;UPDATE TXRDY.
      BRU NEX5
MODE1:MOV #0,MODE             ;RESET MODE FLAG
      MOV WD1,MR              ;STORE MODE WORD
      MOV WD1,TP              ;TP EFFECTED TOO
      IDX MR(0),2,1           ;INDEX REG 1 GETS BAUD RATE FACTOR
      AND R23M,MR,TEMP8       ;INDEX REG 2 GETS CHAR LENGTH
      MOV #0,@7
      SHR TEMP8               ;FORM CHAR LENGTH COUNT
      SHR TEMP8
      ADD #5,TEMP8
      MOV TEMP8,@2
      AND R4M,MR,TEMP8        ;PARITY ENABLED?
      BEQ TEMP8,PARNO         ;SKIP NEXT INST FOR NO PAR
      MOV @2,TEMP8
      ADD #1,TEMP8,TEMP8      ;IR4 =S CHAR LENGTH +PAR
      MOV TEMP8,@4
      BRU STOPB
PARNO:MOV @2,@4
      MOV @4,TEMP8
STOPB:ADD #1,TEMP8,TEMP8      ;IR5 =S CHAR LENGTH +PAR +1 STOP BIT
      MOV TEMP8,@5
      AND R67M,MR,TEMP8       ;INDEX REG 3 GETS # OF STOP BITS
      MOV TEMP8,@3
NEX5: BEQ NDSR,NDSRO,NEX6     ;CHANGE IN NDSR?
      BEQ NDSR,RBIT
      BIS #7,STATINP          ;RESET NDSR STATUS
      BRU STATUP
RBIT: BIR #7,STATINP          ;RESET NDSR STATUS
STATUP:MOV(W7) STATINP,STATUS;UPDATE STATUS IN 14TCY
```

```
NEX6:BEQ NCTS,NCTSO,NEX65     ;NCTS TRANSITION?
     MOV NCTS,NCTSO           ;UPDATE NCTS.
     JSR CTXRD                ;UPDATE TXRDY PIN.
NEX65:BEQ OSCF,NEX7           ;CHECK OUTPUT SELFCALL FLAG
     MOV #0,OSCF              ;RESET SELFCALL FLAG
     BNE BRKO,NEX7            ;IF BREAK IS SET,QUIT OUTPUTTING.
     MOV @2,TEMP4             ;
     CMP TEMP4,OCTR           ;END OF DATA SHIFT?
     BNE C,PARTY
     SHR TS                   ;SHIFT TRANSMITT REG
     MOV (W0) C,TXD           ;OUTPUT DATA
     XOR PAR,C,PAR            ;FORM ODD PARITY
     INC OCTR
     JSR SHFTSKED             ;SCHEDULE SHIFT
     BRU NEX7
PARTY:MOV @4,TEMP4            ;
     CMP TEMP4,OCTR           ;END OF PARITY SHIFT?
     BEQ Z,SBITS
     AND R5M,MR,TEMP8         ;EVEN OR ODD PAR?
     BEQ TEMP8,OPAR
     COM PAR                  ;OUTPUT EVEN PARITY OR BREAK CHAR
OPAR:MOV (W0) PAR,TXD         ;OUTPUT PARITY
OCINC:INC OCTR
     JSR SHFTSKED
     BRU NEX7
SBITS:MOV @5,TEMP4            ;1ST STOP BIT SHIFTED?
     CMP TEMP4,OCTR           ;
     BEQ Z,MSBIT
     MOV (W0) #1,TXD          ;OUTPUT 1ST STOPBIT OR BREAK CHAR
     INC OCTR
     JSR SHFTSKED             ;SCHEDULE SHIFT
     BRU NEX7
MSBIT:MOV @3,TEMP8            ;ONLY ONE STOP BIT?
     CMP #64,TEMP8
     BNE Z,ORED1
     CMP #192,TEMP8           ;1 AND 1 HALF OR 2 STOP BITS?
     BNE Z,2SBITS
     BRU TAB1@1
100: MOV #0,EX                ;ILLEGAL ,EXIT
101: MOV(W9) #1,SBD           ;TXD LOW IN 6.51USEC
     BRU ORED
102: MOV(W10) #1,SBD          ;TXD LOW IN 26.04 USEC
     BRU ORED
103: MOV(W11) #1,SBD          ;TXD LOW IN 52.08 USEC
     BRU ORED
2SBITS:BRU TAB2@1
200: MOV #0,EX                ;ILLEGAL, EXIT
201: MOV(W8) #1,SBD           ;TXD LOW IN 13.02 USEC
     BRU ORED
202: MOV(W11) #1,SBD          ;TXD LOW IN 52.08 USEC
     BRU ORED
203: MOV(W12) #1,SBD          ;TXD LOW IN 104.16 USEC
     BRU ORED
ORED1:MOV #1,SBD
ORED:BIS #2,STATINP           ;SET TXEMPTY STATUS IN 14 TCY
```

AD-A118 826 VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG D--ETC F/G 9/2
MICROPROCESSOR SELF-TEST: SOFTWARE SELF-TEST FOR AN 8080-BASED --ETC(U)
JUN 82 J R ARMSTRONG, F G GRAY F30602-80-C-0200
UNCLASSIFIED RADC-TR-82-80 NL



```
     MOV(W7) STATINP,STATUS
     MOV(W7) #1,TXEMPTY        ;TXEMPTY=1 IN 14TCY
NEX7:AND TPF,SBD,TEMP1         ;TPF.SBD=1?
     BEQ TEMP1,SER1
     JSR TXCTS
SER1:BNE ISHF,FLCK1            ;IS A INP SHIFT IN PROCESS?
     COM RXD,TEMP1
     AND TEMP1,RXDO,TEMP1      ;RXD HI-LO TRANSITION?
     BEQ TEMP1,FLCK
     BRU TAB3@1                ;SCHEDULE START BIT TEST
300: MOV #0,EX
301: MOV(W9) #1,ISSCF          ;SELFCALL IN 6.51USEC
     BRU FLCK
302: MOV(W10) #1,ISSCF         ;SELFCALL IN 26.04USEC
     BRU FLCK
303: MOV(W11) #1,ISSCF         ;SELFCALL IN 52.08 USEC
FLCK:BEQ ISSCF,FLCK1           ;CHECK INP START SELFCALL FLAG
     MOV #0,ISSCF              ;RESET FLAG
     BNE RXD,FLCK1             ;RXD STILL LOW?
     MOV #1,ISHF               ;SET INPUT SHIFT FLAG
     MOV #1,ICTR               ;INIT INPUT COUNTER
     MOV #1,IPAR               ;INIT INP PARITY
     MOV #0,IBRK               ;INIT INP BREAK CHECK
     JSR ISKED
     BRU FLCK1
FLCK1:BEQ ISCF,NEX8            ;CHECK INPUT SELFCALL FLAG
     MOV #0,ISCF               ;RESET FLAG
     MOV @2,TEMP4
     CMP TEMP4,ICTR            ;END OF DATA BITS?
     BNE C,IPARC
     MOV RXD,@7                ;GET SERIAL INPUT
     MOV RXD,IBRK              ;FORM BREAK CHECK
     SHR RS                    ;SHIFT SER INP REG
     XOR RXD,IPAR,IPAR         ;FORM INPUT PARITY
     INC ICTR                  ;INC INPUT COUNTER
     JSR ISKED                 ;SCHEDULE INPUT SHIFT
     BRU NEX8
IPARC:MOV @4,TEMP4
     CMP TEMP4,ICTR            ;END OF PARITY SHIFT?
     BNE C,RXDOK
     INC ICTR                  ;INC INPUT CTR
     AND R4H,MR,TEMP8          ;PARITY?
     BEQ RXDCK
     MOV RXD,IBRK              ;FORM BREAK CHECK
     AND R5H,MR,TEMP8          ;EVEN OR ODD PARITY?
     BEQ TEMP8,ODD
     COM IPAR,IPAR
ODD:BNE IPAR,RXD,IPER          ;CHECK PARITY
     BRU ISK
IPER:BIS #3,STATINP            ;SET PARITY ERROR STATUS
     MOV(W7) STATINP,STATUS    ;UPDATE STATUS IN 14TCY
ISK: JSR ISKED                 ;INPUT SHIFT SELFCALL SCHEDULE
     BRU NEX8
RXDCK:BNE RXD,RDYSET           ;CHECK FOR STOP BIT
     MOV RXD,IBRK              ;FORM BREAK CHECK
```

```
      BNE IBRK,FRE              ;BREAK CHAR?
      MOV #1,SYNBDO             ;SET BREAK DETECT PIN.
      BIS #6,STATINP            ;UPDATE STATUS
FRE:  BIS #5,STATINP            ;SET FRAMING ERROR STATUS
RDYSET:BNE RXRDY,OER            ;CHECK FOR OVERRUN ERROR
      MOV(W6) #1,RXRDY          ;RXRDY = 1 IN 4TCY
      BIS #1,STATINP            ;SET RXRDY STATUS
      BRU RPUP
OER:BIS #4,STATINP              ;SET OVERRUN STATUS BIT
RPUP:MOV RS,RP
      MOV #0,ISHF               ;RESET ISHF
      MOV(W7) STATINP,STATUS    ;UPDATE STATUS IN 14 TCY
      BRU NEX8
NEX75:BIS #0,STATINP            ;SET TXRDY & TXEMPTY STATUS BITS.
       BIS #2,STATINP           ;
       MOV STATINP,STATUS
       JSR CTXRD                ;UPDATE TXRDY.
      MOV #1,TXEMPTY            ;TXEMPTY GET SET TO 1.
      MOV (W0) #1,TXD                ;
      MOV #1,SBD
NEX8:MOV DI,DIO                 ;UPDATE VARIABLES
      MOV RESET,ORESET
      MOV NDSR,NDSRO
      MOV NCS,NCSO
      MOV RXD,RXDO
      MOV NRD,NRDO
      MOV NWR,NWRO
      MOV #0,EX                 ;EXIT MODULE
TXCTS:AND ROM,CR,TEMP8          ;CHECK TXEN
      BEQ TEMP8,RET
      BNE NCTS,RET              ;CHECK NCTS
      MOV(W2) #0,TXD            ;SCHEDULE START BIT IN 50NS(ASSUMED)
      BIR #2,STATINP            ;RESET TXEMPTY STATUS IN 14TCY
      MOV(W7) STATINP,STATUS
      MOV(W7) #0,TXEMPTY        ;RESET TXEMPTY PIN ALSO
      JSR SHFTSKED              ;SCHEDULE FIRST BIT SHIFT
      MOV #1,OCTR               ;INITIALISE OUTPUT COUNTER
      MOV #0,PAR                ;ZERO PARITY
      MOV #0,TPF                ;ZERO TP FULL FLAG
      MOV #0,SBD
      MOV TP,TS                 ;PARA T REG TO SER TREG
RET:  RTS
SHFTSKED:BRU TAB401             ;WHICH BAUDRATE?
400:  MOV #0,EX
401:  MOV(W8) #1,OSCF           ;SELFCALL IN 13.02 USEC
      RTS
402:MOV(W11) #1,OSCF            ;SELFCALL IN 52.08 USEC
      RTS
403:MOV(W12) #1,OSCF            ;SELFCALL IN 104.16 USEC
      RTS
ISKED:BRU TAB501                ;WHICH BAUDRATE?
500:  MOV #0,EX                 ;ILLEGAL ,EXIT
501:  MOV(W8) #1,ISCF
      RTS
502:MOV(W11) #1,ISCF
```

```
      RTS
503:MOV(W12) #1,ISCF
      RTS
                              ;
CTXRD:BNE NCTS,RTXRD          ;IF NCTS=1,SCHEDULE TXRDY=0.
      AND ROM,CR,TEMP8        ;IF TXEN=0,SCHEDULE TXRDY=0.
      BEQ TEMP8,RTXRD
      AND ROM,STATINP,TEMP8   ;IF D.B. BUFFER IS NOT EMPTY,
      BEQ TEMP8,RTXRD         ;SCHEDULE TXRDY=0.
      MOV (W1) #1,TXRDY       ;
      RTS
RTXRD:MOV (W1) #0,TXRDY
      RTS
TAB1:BYT 100,101,102,103
TAB2:BYT 200,201,202,203
TAB3:BYT 300,301,302,303
TAB4:BYT 400,401,402,403
TAB5:BYT 500,501,502,503
ROM:  BYT #1
R1M:  BYT #2
R2M:  BYT #4
R3M:  BYT #8
R4M:  BYT #16
R5M:  BYT #32
R6M:  BYT #64
R7M:  BYT #128
R01M:  BYT #3
R23M:  BYT #12
R67M:  BYT #192
      END
```

```
      REG(8)   TEMP1,TEMP2
      PIN      8228(1,8),4044(9,16),2708(17,24),8251(25,32)
      PIN      8255(33,40),BUSOUT(41,48),EX(150)
      EVW      W10(1)
STR:  AND      8228,4044,TEMP1
      AND      2708,8251,TEMP2
      AND      8255,TEMP1,TEMP1
      AND      TEMP1,TEMP2,TEMP2
      MOV(W10)      TEMP2,BUSOUT
      MOV      #0,EX
      BRU      STR
      END
```

```
;       THIS CHIP SIMULATES CHIP SELECT LOGIC.
;       SELECTION LOGIC IS AS FOLLOWS.
;       IF  ADRH = 00H              SELECT  ROM
;       IF  ADRH = 10H              SELECT  RAM32RB
;       IF  ADRH = 80H              SELECT  A8255
;       IF  ADRH = 40H              SELECT  A8251
;       IF  NONE OF THE ABOVE , DESELECT ALL FOUR CHIPS.
;       ------------------------------------------------
        REG(8) TEMP8,CAT
        PIN ADRL(1,8),ADRH(9,16)
        PIN ROM(17),RAM(18),A8251(19),A8255(20)
        PIN EX(150)
        EVW W30(30)
;
        AND #240,ADRH,TEMP8     ;TEMP8 = A15,A14,A13,A12,0,0,0,0
        BEQ TEMP8,SROM          ;IF A15-A12 = 0 H ,SELECT ROM
        MOV #128,CAT            ;
        BEQ CAT,TEMP8,YES55     ;IF A15-A12 = 8 H , SELECT A8255.
        MOV #16,CAT             ;
        BEQ CAT,TEMP8,SRAM      ;IF A15-A12 = 1 H , SELECT RAM.
        MOV #64,CAT             ;
        BEQ CAT,TEMP8,YES51     ;IF A15-A12 = 4 H ,SELECT 8251
        MOV(W30) #1,ROM         ;DESELECT ALL CHIPS.
        MOV(W30) #1,RAM         ;
        MOV(W30) #1,A8255       ;
        MOV(W30) #1,A8251       ;
        MOV #0,EX               ;RETURN
YES55:  MOV(W30) #0,A8255       ;SELECT  8255
        MOV(W30) #1,ROM         ;
        MOV(W30) #1,RAM         ;
        MOV(W30) #1,A8251       ;
        MOV #0,EX               ;RETURN
YES51:  MOV(W30) #0,A8251       ;SELECT 8251
        MOV(W30) #1,ROM         ;
        MOV(W30) #1,RAM         ;
        MOV(W30) #1,A8255       ;
        MOV #0,EX               ;RETURN
SROM:   MOV(W30) #0,ROM         ;SELECT ROM
        MOV(W30) #1,RAM         ;
        MOV(W30) #1,A8255       ;
        MOV(W30) #1,A8251       ;
        MOV #0,EX               ;RETURN
SRAM:   MOV(W30) #0,RAM         ;SELECT RAM.
        MOV(W30) #1,ROM         ;
        MOV(W30) #1,A8255       ;
        MOV(W30) #1,A8251       ;
        MOV #0,EX               ;RETURN
        END
```

```
          REG(5) ADO,TEMP5     ;4044 RAM TIMING,32 LOC
          REG(1) NCSO,TEMP1,TEMP2,RWO                ;SIMULATES ADDRESS RIPPLE
          PIN DI(1,8),DO(9,16),NCS(17),RW(18)
          PIN AD(19,23),ADB(24,28),DID(29,36),EN(37),NRD(38),NWR(39)
          PIN EX(150),SCA(151),SCDW(152)
          EVW  W35(350),W10(100)
     MOV EN,TEMP1          ;SIM EXT SEL LOGIC
     COM TEMP1
     AND NRD,RW,TEMP2          ;NCS=(EN.(NRD'+WR')'=EN'+NRD.WR
     OR TEMP1,TEMP2,NCS
     BEQ SCA,NEX1          ;INTERNAL ADDR SELF CALL?
     MOV #0,SCA
     BRU ROW          ;ADDR CHANGE,READ OR WRITE?
NEX1:BEQ SCDW,NEX2          ;INTERNAL WRITE SELF CALL?
     IDX ADB(0),5,1
     MOV DID,MEM@1           ;WRITE CURRENT DATA AT OLD ADDR
     MOV #0,SCDW
NEX2:BEQ AD,ADO,CSCK           ;ADDR CHANGE?
     MOV(W35) AD,ADB
     MOV(W35) #1,SCA          ;SCHED INT ADDR CHANGE
CSCK:BEQ NCS,NCSO,RWCK    ;CHANGE IN NCS?
ROW: BEQ NCS,NEX4
     MOV(W10) #255,DO  ;DO =ALL 1'S
     BRU UPDATE
RWCK:BNE RW,RWO,NEX3      ;CHANGE IN WR?
UPDATE:MOV NCS,NCSO
     MOV RW,RWO
     MOV AD,ADO
     MOV #0,EX       ;EXIT
NEX3:BEQ #1,NCS,UPDATE
NEX4:BEQ RW,WRITE       ;READ OR WRITE?
     IDX ADB(0),5,1
     MOV(W10) MEM@1,DO  ;OUTPUT DATA IN 100NS
     BRU UPDATE
WRITE:MOV(W10) #1,SCDW
     MOV(W10) DI,DID    ;PROAGATE INPUT DATA
     BRU UPDATE
MEM: BYT #0,#0,#0,#0,#0,#0,#0,#0
     BYT #0,#0,#0,#0,#0,#0,#0,#0
     BYT #0,#0,#0,#0,#0,#0,#0,#0
     BYT #0,#0,#0,#0,#0,#0,#0,#0
      END
```

```
          REG(9) TEMP5,ADO
          REG(1) NCSO,TEMP1                    ;SIMULATES ADDRESS RIPPLE
          PIN DO(1,8),NCS(9) ;INITIALIZED TO QCPU2LS V2.6.2
          PIN AD(10,18),ADDB(19,27),EX(150),SCF(151)
          EVW W350(350),W100(100)
     XOR AD,ADO,TEMP5    ; ADDR CHANGE?
     BEQ TEMP5,SCCK
     MOV(W350) AD,ADDB   ;SCHEDULE INT ADDR CHANGE
     MOV(W350) #1,SCF
SCCK:BEQ SCF,CSCK
     MOV #0,SCF
     BEQ NCS,NEX2
     BRU HIZ
CSCK:XOR NCS,NCSO,TEMP1  ;CHANGE IN NCS?
     BEQ TEMP1,UPDATE
     BEQ NCS,NEX2   ;CHECK NCS
HIZ: MOV(W100) #255,DO          ;DO = ALL 1'S
UPDATE:MOV NCS,NCSO          ;UPDATE
     MOV AD,ADO
     MOV #0,EX              ;EXIT
NEX2:IDX ADDB(0),9,1
     MOV(W100) MEMB1,DO  ;OUTPUT DATA IN 100NS
     MOV NCS,NCSO        ;UPDATE
     MOV AD,ADO
     MOV #0,EX           ;EXIT
MEM: BYT #195, #56, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #175, #211, #2, #118, #0, #0, #0, #0
     BYT #49, #31, #16, #58, #152, #1, #6, #51
     BYT #17, #85, #204, #42, #154, #1, #235, #75
     BYT #195, #77, #0, #247, #118, #27, #130, #206
     BYT #215, #194, #48, #0, #210, #48, #0, #226
     BYT #48, #0, #202, #97, #0, #195, #48, #0
     BYT #247, #180, #47, #4, #144, #242, #48, #0
     BYT #202, #48, #0, #194, #114, #0, #195, #48
     BYT #0, #247, #153, #250, #48, #0, #45, #189
     BYT #194, #48, #0, #218, #48, #0, #66, #161
     BYT #79, #197, #35, #227, #140, #170, #157, #194
     BYT #48, #0, #210, #145, #0, #195, #48, #0
     BYT #247, #11, #128, #145, #193, #130, #184, #194
     BYT #48, #0, #242, #161, #0, #195, #48, #0
     BYT #247, #145, #161, #63, #153, #218, #172, #0
     BYT #195, #48, #0, #247, #171, #47, #147, #194
     BYT #48, #0, #120, #31, #15, #87, #120, #178
     BYT #250, #191, #0, #195, #48, #0, #247, #60
     BYT #250, #48, #0, #194, #48, #0, #95, #29
     BYT #147, #234, #48, #0, #39, #198, #5, #39
     BYT #188, #194, #48, #0, #39, #39, #254, #114
     BYT #194, #48, #0, #25, #25, #35, #35, #22
     BYT #204, #128, #140, #131, #133, #137, #130, #210
     BYT #48, #0, #238, #45, #194, #48, #0, #17
```

```
BYT #153, #1, #26, #235, #190, #194, #48, #0
BYT #23, #222, #232, #194, #48, #0, #218, #48
BYT #0, #205, #156, #1, #210, #48, #0, #198
BYT #55, #7, #222, #110, #194, #48, #0, #62
BYT #198, #198, #206, #245, #175, #241, #210, #48
BYT #0, #202, #48, #0, #242, #48, #0, #234
BYT #48, #0, #39, #254, #250, #194, #48, #0
BYT #62, #0, #71, #14, #255, #175, #183, #167
BYT #194, #48, #0, #161, #194, #48, #0, #177
BYT #169, #194, #48, #0, #169, #161, #176, #185
BYT #194, #48, #0, #177, #168, #185, #194, #48
BYT #0, #160, #194, #48, #0, #62, #51, #33
BYT #0, #16, #17, #1, #16, #1, #2, #16
BYT #2, #47, #18, #54, #193, #10, #238, #51
BYT #194, #48, #0, #58, #1, #16, #254, #204
BYT #194, #48, #0, #62, #193, #78, #185, #194
BYT #48, #0, #47, #50, #0, #16, #190, #194
BYT #48, #0, #33, #85, #170, #34, #1, #16
BYT #235, #126, #187, #194, #48, #0, #35, #126
BYT #186, #194, #48, #0, #198, #1, #52, #190
BYT #194, #48, #0, #62, #255, #211, #2, #118
BYT #194, #116, #170, #102, #33, #0, #0, #57
BYT #235, #33, #170, #170, #249, #33, #0, #0
BYT #62, #170, #57, #188, #194, #48, #0, #189
BYT #194, #48, #0, #47, #33, #85, #85, #249
BYT #33, #0, #0, #57, #188, #194, #48, #0
BYT #189, #194, #48, #0, #33, #0, #0, #249
BYT #59, #62, #255, #57, #188, #194, #48, #0
BYT #189, #194, #48, #0, #35, #47, #188, #194
BYT #48, #0, #189, #194, #48, #0, #235, #249
BYT #192, #248, #55, #201, #195, #48, #0, #247
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
 END
```

```
          REG(8) TEMP5,ADO
          REG(1) NCSO,TEMP1                  ;SIMULATES ADDRESS RIPPLE
          PIN DO(1,8),NCS(9) ;INITIALIZED TO ROMADDS V1.4
          PIN AD(10,17),ADDB(18,25),EX(150),SCF(151)
          EVW W350(350),W100(100)
     XOR AD,ADO,TEMP5   ; ADDR CHANGE?
     BEQ TEMP5,SCCK
     MOV(W350) AD,ADDB   ;SCHEDULE INT ADDR CHANGE
     MOV(W350) #1,SCF
SCCK:BEQ SCF,CSCK
     MOV #0,SCF
     BEQ NCS,NEX2
     BRU HIZ
CSCK:XOR NCS,NCSO,TEMP1  ;CHANGE IN NCS?
     BEQ TEMP1,UPDATE
     BEQ NCS,NEX2   ;CHECK NCS
HIZ: MOV(W100) #255,DO         ;DO = ALL 1'S
UPDATE:MOV NCS,NCSO         ;UPDATE
     MOV AD,ADO
     MOV #0,EX              ;EXIT
NEX2:IDX ADDB(0),8,1
     MOV(W100) MEM@1,DO  ;OUTPUT DATA IN 100NS
     MOV NCS,NCSO        ;UPDATE
     MOV AD,ADO
     MOV #0,EX           ;EXIT
MEM: BYT #33, #0, #0, #17, #32, #0, #151, #79
     BYT #134, #137, #35, #71, #125, #187, #120, #194
     BYT #8, #0, #124, #186, #120, #194, #8, #0
     BYT #211, #2, #118, #114, #170, #0, #54, #96
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
```

```
      BYT #0, #0, #0, #0, #0, #0, #0, #0
      BYT #0, #0, #0, #0, #0, #0, #0, #0
        END
```

```
          REG(9) TEMP5,ADO
          REG(1) NCSO,TEMP1                    ;SIMULATES ADDRESS RIPPLE
          PIN DO(1,8),NCS(9) ;INITIALIZED TO RAMTEST
          PIN AD(10,18),ADDB(19,27),EX(150),SCF(151)
          EVW W350(350),W100(100)
     XOR AD,ADO,TEMP5    ; ADDR CHANGE?
     BEQ TEMP5,SCCK
     MOV(W350) AD,ADDB   ;SCHEDULE INT ADDR CHANGE
     MOV(W350) #1,SCF
SCCK:BEQ SCF,CSCK
     MOV #0,SCF
     BEQ NCS,NEX2
     BRU HIZ
CSCK:XOR NCS,NCSO,TEMP1  ;CHANGE IN NCS?
     BEQ TEMP1,UPDATE
     BEQ NCS,NEX2    ;CHECK NCS
HIZ: MOV(W100) #255,DO          ;DO = ALL 1'S
UPDATE:MOV NCS,NCSO         ;UPDATE
     MOV AD,ADO
     MOV #0,EX           ;EXIT
NEX2:IDX ADDB(0),9,1
     MOV(W100) MEM@1,DO  ;OUTPUT DATA IN 100NS
     MOV NCS,NCSO        ;UPDATE
     MOV AD,ADO
     MOV #0,EX           ;EXIT
MEM: BYT #33, #0, #16, #22, #32, #62, #85, #79
     BYT #70, #120, #47, #113, #190, #194, #42, #0
     BYT #121, #119, #190, #194, #42, #0, #47, #119
     BYT #190, #194, #42, #0, #112, #126, #184, #194
     BYT #42, #0, #35, #21, #194, #8, #0, #62
     BYT #255, #118, #175, #118
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
```

```
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0, #0, #0, #0, #0, #0, #0, #0
BYT #0
BYT #0
BYT #0
BYT #0
 END
```

```
          REG(9) TEMP5,ADO
          REG(1) NCSO,TEMP1                    ;SIMULATES ADDRESS RIPPLE
          PIN DO(1,8),NCS(9) ;INITIALIZED TO QCPU3S V3.3
          PIN AD(10,18),ADDB(19,27),EX(150),SCF(151)
          EVW W350(350),W100(100)
     XOR AD,ADO,TEMP5    ; ADDR CHANGE?
     BEQ TEMP5,SCCK
     MOV(W350) AD,ADDB   ;SCHEDULE INT ADDR CHANGE
     MOV(W350) #1,SCF
SCCK:BEQ SCF,CSCK
     MOV #0,SCF
     BEQ NCS,NEX2
     BRU HIZ
CSCK:XOR NCS,NCSO,TEMP1  ;CHANGE IN NCS?
     BEQ TEMP1,UPDATE
     BEQ NCS,NEX2    ;CHECK NCS
HIZ: MOV(W100) #255,DO          ;DO = ALL 1'S
UPDATE:MOV NCS,NCSO          ;UPDATE
     MOV AD,ADO
     MOV #0,EX              ;EXIT
NEX2:IDX ADDB(0),9,1
     MOV(W100) MEM01,DO  ;OUTPUT DATA IN 100NS
     MOV NCS,NCSO        ;UPDATE
     MOV AD,ADO
     MOV #0,EX           ;EXIT
MEM: BYT #195, #56, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #175, #211, #2, #118, #0, #0, #0, #0
     BYT #49, #31, #16, #58, #23, #1, #6, #51
     BYT #17, #85, #204, #42, #25, #1, #235, #75
     BYT #195, #77, #0, #247, #118, #27, #130, #206
     BYT #215, #194, #48, #0, #210, #48, #0, #226
     BYT #48, #0, #202, #97, #0, #195, #48, #0
     BYT #247, #220, #89, #1, #144, #242, #48, #0
     BYT #202, #48, #0, #194, #114, #0, #195, #48
     BYT #0, #247, #153, #250, #48, #0, #196, #98
     BYT #1, #194, #48, #0, #218, #48, #0, #66
     BYT #161, #220, #48, #0, #79, #197, #35, #227
     BYT #204, #104, #1, #170, #157, #196, #48, #0
     BYT #210, #151, #0, #195, #48, #0, #247, #11
     BYT #128, #204, #48, #0, #145, #193, #130, #184
     BYT #194, #48, #0, #242, #170, #0, #195, #48
     BYT #0, #247, #145, #161, #63, #212, #48, #0
     BYT #153, #218, #184, #0, #195, #48, #0, #247
     BYT #171, #212, #109, #1, #178, #244, #48, #0
     BYT #250, #199, #0, #195, #48, #0, #247, #60
     BYT #250, #48, #0, #194, #48, #0, #95, #29
     BYT #252, #124, #1, #198, #5, #39, #188, #194
     BYT #48, #0, #252, #48, #0, #39, #39, #244
     BYT #135, #1, #22, #204, #128, #140, #131, #133
     BYT #137, #130, #210, #48, #0, #238, #45, #194
```

```
BYT #48, #0, #17, #24, #1, #26, #235, #190
BYT #194, #48, #0, #23, #222, #232, #194, #48
BYT #0, #216, #48, #0, #205, #27, #1, #210
BYT #48, #0, #198, #55, #7, #222, #110, #194
BYT #48, #0, #62, #255, #211, #2, #118, #194
BYT #116, #170, #102, #33, #0, #0, #17, #0
BYT #0, #57, #235, #49, #170, #170, #62, #170
BYT #57, #188, #194, #48, #0, #189, #194, #48
BYT #0, #47, #49, #85, #85, #33, #0, #0
BYT #57, #188, #194, #48, #0, #189, #194, #48
BYT #0, #33, #0, #0, #249, #175, #57, #188
BYT #194, #48, #0, #189, #194, #48, #0, #235
BYT #249, #192, #248, #55, #201, #195, #48, #0
BYT #247, #180, #216, #200, #47, #4, #192, #195
BYT #48, #0, #45, #189, #200, #195, #48, #0
BYT #140, #208, #195, #48, #0, #47, #147, #194
BYT #48, #0, #224, #120, #31, #15, #87, #120
BYT #240, #195, #48, #0, #240, #147, #208, #234
BYT #48, #0, #39, #216, #195, #48, #0, #254
BYT #114, #194, #48, #0, #25, #25, #35, #35
BYT #184, #248, #195, #48, #0
 END
```

```
          REG(9) TEMP5,ADO
          REG(1) NCSO,TEMP1                    ;SIMULATES ADDRESS RIPPLE
          PIN DO(1,8),NCS(9) ;INITIALIZED TO QCPU4S V4.3
          PIN AD(10,18),ADDB(19,27),EX(150),SCF(151)
          EVW W350(350),W100(100)
     XOR AD,ADO,TEMP5    ; ADDR CHANGE?
     BEQ TEMP5,SCCK
     MOV(W350) AD,ADDB   ;SCHEDULE INT ADDR CHANGE
     MOV(W350) #1,SCF
SCCK:BEQ SCF,CSCK
     MOV #0,SCF
     BEQ NCS,NEX2
     BRU HIZ
CSCK:XOR NCS,NCSO,TEMP1  ;CHANGE IN NCS?
     BEQ TEMP1,UPDATE
     BEQ NCS,NEX2    ;CHECK NCS
HIZ: MOV(W100) #255,DO          ;DO = ALL 1'S
UPDATE:MOV NCS,NCSO          ;UPDATE
     MOV AD,ADO
     MOV #0,EX              ;EXIT
NEX2:IDX ADDB(0),9,1
     MOV(W100) MEM@1,DO  ;OUTPUT DATA IN 100NS
     MOV NCS,NCSO           ;UPDATE
     MOV AD,ADO
     MOV #0,EX              ;EXIT
MEM: BYT #255, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #0, #0, #0, #0, #0, #0, #0, #0
     BYT #175, #211, #2, #118, #0, #0, #0, #0
     BYT #49, #31, #16, #58, #27, #1, #6, #51
     BYT #17, #85, #204, #42, #29, #1, #235, #75
     BYT #195, #77, #0, #247, #118, #27, #147, #156
     BYT #218, #48, #0, #238, #33, #61, #194, #48
     BYT #0, #226, #48, #0, #202, #99, #0, #195
     BYT #48, #0, #247, #140, #161, #144, #189, #194
     BYT #48, #0, #250, #48, #0, #47, #141, #242
     BYT #48, #0, #202, #48, #0, #194, #124, #0
     BYT #195, #48, #0, #247, #129, #171, #194, #48
     BYT #0, #178, #66, #23, #148, #194, #48, #0
     BYT #242, #143, #0, #195, #48, #0, #247, #230
     BYT #255, #79, #197, #227, #170, #11, #137, #210
     BYT #48, #0, #132, #234, #48, #0, #210, #165
     BYT #0, #195, #48, #0, #247, #152, #193, #7
     BYT #184, #194, #48, #0, #173, #250, #180, #0
     BYT #195, #48, #0, #247, #63, #31, #218, #48
     BYT #0, #214, #179, #87, #120, #178, #242, #48
     BYT #0, #60, #194, #48, #0, #250, #48, #0
     BYT #95, #29, #147, #218, #210, #0, #195, #48
     BYT #0, #247, #39, #198, #5, #39, #188, #194
     BYT #48, #0, #39, #39, #254, #114, #194, #48
     BYT #0, #25, #25, #35, #35, #22, #204, #20
     BYT #128, #140, #139, #141, #137, #138, #210, #48
```

```
BYT #0, #238, #48, #194, #48, #0, #17, #28
BYT #1, #26, #235, #190, #194, #48, #0, #23
BYT #222, #232, #194, #48, #0, #218, #48, #0
BYT #205, #31, #1, #210, #48, #0, #198, #55
BYT #7, #222, #110, #194, #48, #0, #62, #255
BYT #211, #2, #118, #150, #116, #170, #102, #33
BYT #0, #0, #17, #0, #0, #57, #235, #49
BYT #170, #170, #62, #170, #57, #188, #194, #48
BYT #0, #189, #194, #48, #0, #47, #49, #85
BYT #85, #33, #0, #0, #57, #188, #194, #48
BYT #0, #189, #194, #48, #0, #33, #0, #0
BYT #249, #175, #57, #188, #194, #48, #0, #189
BYT #194, #48, #0, #235, #249, #192, #248, #55
BYT #201, #195, #48, #0, #247
 END
```

Appendix D

Test System

Data File

```
;MICMA   ADDRESS   DATA     MREAD     MWR      ROM     RAM & 51     8255
Y
; 1-7, 11-18      21-28      30       33       35       37-38       40
T 7000000
;#1 =>RAM32RB, #2 =>CSL, #3 =>BUS, #4 =>B8228
;#5 =>A8255V6, #6 =>A8251V5, #7 =>TESTCPU*, #8 =>A8080N
;   ================================================================
;  BUS MODULE
C   3,41    6,1     4,17     1,1     5,7     0,21      ;BUSOUT  (LSB - M
C   3,42    6,2     4,18     1,2     5,8     0,22      ;FRONT PANEL DATA
C   3,43    6,3     4,19     1,3     5,9     0,23
C   3,44    6,4     4,20     1,4     5,10    0,24
C   3,45    6,5     4,21     1,5     5,11    0,25
C   3,46    6,6     4,22     1,6     5,12    0,26
C   3,47    6,7     4,23     1,7     5,13    0,27
C   3,48    6,8     4,24     1,8     5,14    0,28
N 3 1 40
N 3 49 64
A C 3 41 255
A C 3 25 255
A C 3 33 255
;   ================================================================
;   ROMQCPU*
C   7,1         3,17      ; DO      => DATA OUT CONNECTED TO BUS MODULE
C   7,2         3,18
C   7,3         3,19
C   7,4         3,20
C   7,5         3,21
C   7,6         3,22
C   7,7         3,23
C   7,8         3,24
N 7 9 17
N 7 150 151
A C 7 1 255
A C 7 168 1             ; PRESET INTERNAL FLAG (NCSO)
;   ================================================================
;   RAM32RA
C   1,9         3,9       ; DATA OUT CONNECTED TO BUS MODULE
C   1,10        3,10
C   1,11        3,11
C   1,12        3,12
C   1,13        3,13
C   1,14        3,14
C   1,15        3,15
C   1,16        3,16
N 1 1 8
N 1 18  23
N 1 37 39
N 1 150 152
A C 1 9 255
;   ================================================================
;   A8228
C   4,9         8,1       ; DATA OUT TO 8080
C   4,10        8,2
C   4,11        8,3
```

D.1.

```
C    4,12        8,4
C    4,13        8,5
C    4,14        8,6
C    4,15        8,7
C    4,16        8,8
C    4,42        6,20        1,38        5,2        0,30    ; MREAD
C    4,41        6,19        1,16        5,3        0,33    ; MWRITE
C    4,25        3,1                                        ; DATA OUT TO BUS MODULE
C    4,26        3,2
C    4,27        3,3
C    4,28        3,4
C    4,29        3,5
C    4,30        3,6
C    4,31        3,7
C    4,32        3,8
N 4 1 8
N 4 17 24
N 4 33 38
N 4 150 152
A C 4 39 1
A C 4 40 1
A C 4 41 1
A C 4 42 1
A C 4 43 1
A C 4 54 1
A C 4 55 1
A C 4 56 1
A C 4 57 1
A C 4 58 1
A C 4 9 255
A C 4 25 255
A C 4 46 255
;    ================================================================
;    ABOBON
C    8,9         4,1                                        ; DATA BUS OUT TO 8228
C    8,10        4,2
C    8,11        4,3
C    8,12        4,4
C    8,13        4,5
C    8,14        4,6
C    8,15        4,7
C    8,16        4,8
C    8,37        4,33
C    8,41        4,35
C    8,42        4,34
C    8,39        4,36
;  CONNECTION FROM ADDR BUS.
C    8,17        7,10        1,19        0,11        2,1        5,5        6,22
C    8,18        7,11        1,20        0,12        2,2        5,6
C    8,19        7,12        1,21        0,13        2,3
C    8,20        7,13        1,22        0,14        2,4
C    8,21        7,14        1,23        0,15        2,5
C    8,22        7,15        1,24        0,16        2,6
C    8,23        7,16                    0,17        2,7
C    8,24        7,17                    0,18        2,8
```

```
C   8,25        7,18                0,1       2,9
C   8,26                            0,2       2,10
C   8,27                            0,3       2,11
C   8,28                            0,4       2,12
C   8,29                            0,5       2,13
C   8,30                            0,6       2,14
C   8,31                            0,7       2,15
C   8,32                            0,8       2,16
N 8 150 153
A C 8 34 1
A C 8 41 1
A C 8 9 255
;   ================================================================
;;  CSL     ( CHIP SELECT LOGIC)
C    2,17     7,9       0,35
C    2,18     1,37      0,37
C    2,19     6,21      0,38
C    2,20     5,4       0,40
N 2   1   8
N 2   9   16
N 2   150 150
;  ================================================================
;;  8251
C    6,9      3,25
C    6,10     3,26
C    6,11     3,27
C    6,12     3,28
C    6,13     3,29
C    6,14     3,30
C    6,15     3,31
C    6,16     3,32
C    6,26     6,25       ;CONNECTION FOR 'TEST51'
N 6   1   8
N 6   17  21
N 6   23  25
N 6   33  33
N 6   151 154
A C 6 9 255
A C 6 29  1
A C 6 35  4
A C 6 202 4
A C 6 19  1
A C 6 20  1
A C 6 21  1
A C 6 30  1
A C 6 31  1
A C 6 32  1
A C 6 43  1
A C 6 44  1
A C 6 45  1
A  6  17  10  1
A  6  17  40  0
;   ================================================================
;;   A8255
C    5,15     3,33
```

D.3.

```
C    5,16      3,34
C    5,17      3,35
C    5,18      3,36
C    5,19      3,37
C    5,20      3,38
C    5,21      3,39
C    5,22      3,40
N 5 1 14
N 5 150 155
A C 5 23 255
A C 5 39 255
A C 5 55 1
A C 5 56 1
A C 5 57 1
A C 5 58 1
A C 5 59 1
A C 5 60 1
A C 5 61 1
A C 5 62 1
A C 5 71 15
A C 5 75 15
;  ================================================================
L 1 0
L 2 0
L 3 0
L 4 0
L 5 0
L 6 0
L 7 0
L 8 0
A 8 150 0 1   ; FIRST CALL TO START UP
B 900000
```

D.4.

APPENDIX E

Fault Experiments Summary

FAULT LIST

CHIP: 8080 CPU

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Push PSW results in A & PSW pushed onto stack in reverse order. PSW value on stack wrong	*			Detected by CPIØ4H at ØØ75H
WR appeared before data	*			detected by CPI Ø4H at ØØ75H
XTHL writes H twice and L not at all	*			
DCR C decremented 80Hz got ØØH	*			detected by CMPH/JNZ ERR at ØØ8CH
ADD did not set AC flag correctly	*			
Incorrect CY flag after subtracting zero CY = 1 instead of Ø	*			
Improper RAL execution LSB wrong	*			
SBI subtracted CY instead of CY	*			Detected by CMPH/JNZ ERR at ØØ8CH
Incorrect CY flag after DAA operation	*			
Incorrect parity flag	*			detected by JPO at ØØ52H
Incorrect CY flag after subtract and compare instructions	*			detected by SBB C and JH at ØØ62H
Multiple register select on READ. Select B also selects D	*			
Multiple register select on READ. Select C also selects L.	*			
Multiple register select on READ. Select H also selects B.	*			

FAULT LIST

CHIP: 8080 CPU

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Multiple register select on READ. Select D also select E	*			
Register C bit 3 stuck-at-0	*			
Register C bit 5 stuck-at-1	*			
CMA always sets bit 2 of of Accum.	*			
WR pulse duration too short (about half of nom. value)	*			
Parity flag determination ignores bit 3. Treats it as stuck-at-1	*			
Parity flag determination ignores bit 3. Treats it as stuck-at-0	*			
SYNC pulse is delayed till about the falling edge of $\phi 2$ clock			*	Fault had no effect on the system performance.
Data Line D3 stuck-at-1		*		Hardware timer would Probably detect the fault
Data line stuck-at-0	*			
Reset pin open (stuck-at-1)	*			8080 CPU does not function hardware timer should detect the fault
Interrupt pin open (stuck-at-1)	*			Results in RST7 which causes a call to address 0038H. The test program incidentally, starts at the same location. Timer detected
Incorrect register select on READ & WRITE (select E selects B)	*			
Incorrect register select on READ & WRITE (select E selects C)	*			
Multiple register select on write (Write E also writes to C)	*			

FAULT LIST

CHIP: 8080 CPU

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Multiple register select on WRITE (Write C also writes to B)	*			
Multiple register select on WRITE (Write C also writes to E)	*			
Multiple register select on WRITE (Write B also writes to D)	*			
Judge condition microperation says CY=0 when CY=1	*			
Ready line shorted to ground at time=10,000 NS	*			CPU wait state. Hardware timer should detect the fault.
DAA did not add 6H to low nibble	*			detected by CMP/JNZ ERR at ØØ8C H
SBI did not move result to accumulator	*			"

FAULT LIST

CHIP: 8228

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Blocks data out from 8080 by floating bus	*			Detected by CPI ØØ4H at 75H
BUSEN input shorted to ground	*			Bus is never disabled
BUSEN open (stuck-at-1)			*	Bus disabled
DBIN input (stuck-at-0)	*			Hardware timer should detect the fault.
DBIN input open (stuck-at-1)			*	
Data line 2 from 8080 open	*			

FAULT LIST

CHIP: BUS

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Data bus line 4 from open (stuck-at-1)		*		opcodes altered
Data bus line 2 from 8228 stuck-at-Ø		*		"
Short between data bus lines 4 & 5 from 8228		*		"

FAULT LIST

CHIP: ROM

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Faulty decoder. If address is within the range for ROM, location 1FH is read all the time		*		
address line AD$\emptyset$ stuck-at $\emptyset$		*		halt reached early
address line AD$\emptyset$ stuck-at-$\emptyset$		*		
ROM is always selected			*	
CS stuck-at-1				Test incomplete Hardware timer should detect fault
V_{CC} line to ROM open	"	"	"	"
Data bus from ROM at Hi-Z				

FAULT LIST

CHIP: RAM

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
address line AD2 stuck-at-Ø			*	
address line AD2 stuck-at-1			*	
Decoder in RAM ignores most significant address bit. Only lower half of memory can be accessed			*	Test incomplete Hardware timer should detect the fault.
Data line D1 stuck-at-Ø		*		
Data line D1 stuck-at-1		*		
Data bit 3 stuck-at-Ø	*			Accumulator contains Ø1H instead of ØAAH
Data line 3 stuck-at-1	*			Test incomplete Hardware timer should detect the fault

FAULT LIST

CHIP: 8251

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
TxEMPTY & TxRDY status bits do not get set to '1' at the end of Reset mode word & command word	*			Test program gets stuck in a status check loop. Detected by hardware timer
Data line D$\emptyset$ open (s-a-1)	*			
Data line D$\emptyset$ stuck-at-$\emptyset$	*			hardware timer should detect the fault
WR shorted to ground	*			hardware timer should detect the fault
Reset line open (stuck-at-1)	*			
No stop bit transmitted	*			
Status register s-a-$\emptyset$	*			Test program gets stuck in a status check loop. Detected by hardware timer
Output parity always even			*	Chip was configured for NO parity-hence no parity checking
short between C/D & RD inputs	*			
short between WR & CS inputs			*	
short between CS & C/D inputs	*			hardware timer should detect the fault
RD stuck-at-$\emptyset$	*			
RD input open (s-a-1)	*			
C/D input stuck-at-$\emptyset$	*			hardware timer should detect the fault
C/D input open (stuck-at-1)	*			hardware timer should detect the fault

FAULT LIST

CHIP: 8251

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
CS stuck-at-Ø	*			
WR input open (stuck-at-1)	*			
CS input open (stuck-at-1)	*			
8251 does not respond to read or write commands. Date bus to Hi-Z state	*			
On a data read, data remains stable for only 100NS after it is gated onto the bus	*			
Write pulse requires to be 500NS long instead of 250NS	*			
Both mode word and command word are loaded into the command register.	*			hardware timer should detect the fault
Output counter OCTR stuck-at-all 1's	*			

FAULT LIST

CHIP: 8255

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Short between adjacent data input lines (D_2 and D_3)	*			Data read back is different from data output to the 8255
data line shorted to ground (D_7)	*			*
data line open ($D_\emptyset$)	*			
delay for data appearing at an output port excessive (delay increase of 100NS)			*	(chip not meeting time specifications)
wrong bit is SET/RESET stuck-at-$\emptyset$ on bit select line #$\emptyset$	*			
wrong bit is SET/RESET stuck-at-1 on bit select line #2	*			
Bit SET/RESET operation is reversed	*			
BIT reset does not work set works ok	*			
Bit set does not work Reset works ok	*			
Address pin open (A_1)				
Address pin open ($A_\emptyset$)	*			
Port C does not work in split mode			*	Test routine does not test port C in split mode.
Output drivers in the the 8255 fail (bit 5, port stuck at $\emptyset$)	*			
incorrect register selection (writing to port A results in writing to port B too)			*	not detected since data is same at both ports due to wraparound
incorrect register selection (read from port port A results in data from port A & B			*	*

FAULT LIST

CHIP: 8255

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
output driver in the 8255 fails. Bit stuck-at-1	*			
incorrect register selection (write to port A results in writing to port B)	*			
Port C lower (4 bits) stuck at Ø	*			
Hold times for address and data had to be too (300NS from end of write pulse)	*			chip not meeting timing specifications
Bit set/reset command clears port C output completely	*			
Reset fails to clear A & B			*	
Port A (input) bit 1 open	*			
Port B (input) bit 7 stuck-at-Ø	*			
Read port A operation fails (Data buffer does not get loaded from port A)			*	Data buffer contains data which was previously written to it.
A read operation always fails (Data bus tri-stated all the time)	*			
Pattern ØFFH fails to be read correctly (pattern sensitive fault)			*	test program does not test using the pattern FF H
A write operation fails completely	*			
short between RD and CS of 8255	*			
short between RESET & WR of 8255	*			
Port A (input) bit 1 stuck-at-Ø				

FAULT LIST

CHIP: 8255

FAULT DESCRIPTION	TEST RESULT			COMMENTS
	Detected	Program Control Lost	Not Detected	
Port B (input) bits 4 & 5 shorted	*			
Address pin stuck-at-ø (Aø)	*			
Address pin stuck-at-ø (A_1)	*			
short between address pins $A_ø$ & A_1	*			
$\overline{WR}$ stuck-at-ø	*			
$\overline{WR}$ open	*			
$\overline{CS}$ open	*			
$\overline{RD}$ open	*			

Appendix F

MOVI RAM Test Program Listing

```
;*****************************************************************
;***                                                           ***
;***       RAMMOVI ASM80              VERSION 1.2              ***
;***                                                           ***
;***       MOVI RAM TEST FOR 8080/8085                         ***
;***                                                           ***
;***       MOVING INVERSIONS TEST PATTERN                      ***
;***                                                           ***
;***        DAVE HAISLETT         7/7/81                       ***
;***                                                           ***
;*****************************************************************
;***                                                           ***
;***    SYMBOLS:      RAMBEG = START ADDR OF RAM (XX00H)       ***
;***                  RAMBH  = HIGH BYTE OF RAMBEG             ***
;***                  RAMEND = (END ADDR OF RAM)+1 (XX00H)     ***
;***                  RAMEH  = HIGH BYTE OF RAMEND             ***
;***                  RAMSIZ = RAMEND - RAMBEG (= #BYTES OF RAM) ***
;***                  RSIZH  = HIGH BYTE OF RAMSIZ             ***
;***                                                           ***
;***        ** NOTE THAT RAMBEG, RAMEND, AND RAMSIZ MUST       ***
;***           ALL HAVE A LOW BYTE OF ZERO (IE., THEY MUST     ***
;***           BE OF THE FORM XX00 HEX.)                       ***
;***                                                           ***
;*****************************************************************
;***                                                           ***
;***     REGISTERS:     A     COMPARISONS                      ***
;***                    B     NEW PATTERN                      ***
;***                    C     OLD PATTERN                      ***
;***                    D,E   ADDRESS INCREMENT/DECREMENT      ***
;***                    H,L   MEMORY POINTER                   ***
;***                    SP    WRAP AROUND OFFSET               ***
;***                                                           ***
;*****************************************************************
;***                                                           ***
;***        NOTE:  SINCE THE STACK POINTER IS USED AS A        ***
;***               16 BIT REGISTER, INTERRUPTS MUST BE         ***
;***               DISABLED.  (IF STACK IS USED DURING         ***
;***               THE RAM TEST, PATTERN DATA WILL BE          ***
;***               DESTROYED AND BOGUS ERRORS 'DETECTED.')     ***
;***                                                           ***
;*****************************************************************
;
1000                    ORG     1000H
;
1400                    RAMBEG  EQU     1400H
0014                    RAMBH   EQU     RAMBEG/100H
1800                    RAMEND  EQU     1800H
0018                    RAMEH   EQU     RAMEND/100H
0400                    RAMSIZ  EQU     RAMEND - RAMBEG
0004                    RSIZH   EQU     RAMSIZ/100H
;
```

F.1.

```
                        ;
1000  210014  RAMTST: LXI     H,      RAMBEG          ;POINT TO START OF RAM
1003  3E18           MVI     A,      RAMEH           ;PUT END MARKER IN A FOR CMP'S
1005  110100         LXI     D,      0001H           ;CLEAR D & MAKE E=1
                        ;
                        ; CLEAR RAM CONTENTS (WRITE ALL 0'S)
                        ;
1008  72      CLOOP:  MOV     M,      D               ;CLEAR A RAM BYTE
1009  23             INX     H                       ;ADVANCE PTR
100A  BC             CMP     H                       ;REACH END YET?
100B  C20810         JNZ     CLOOP                   ;NOPE
                        ;
                        ;*************************************************************
                        ;***                                                       ***
                        ;***            FORWARD SEQUENCE OF RAM TEST               ***
                        ;***                                                       ***
                        ;*************************************************************
                        ;
100E  010001  FTLOOP: LXI     B,      0100H           ;TEST 1ST BIT POS (B=1)
                                                      ;OLD PATTERN (C) = 0
1011  210014  FTL1:   LXI     H,      RAMBEG          ;POINT TO START OF RAM AGAIN
1014  310000         LXI     SP,     0000H           ;CLEAR THE WRAP AROUND OFFSET
1017  7E      RLOOP:  MOV     A,      M               ;READ A LOCATION INTO A
1018  B9             CMP     C                       ;OLD PATTERN THERE?
1019  C2C210         JNZ     RAMEC                   ;NO, ERROR
101C  70             MOV     M,      B               ;YES, WRITE NEW PATTERN
101D  7E             MOV     A,      M               ;READ IT BACK
101E  B8             CMP     B                       ;IS IT THERE?
101F  C2CC10         JNZ     RAMEB                   ;NOPE, ERROR
1022  3EFF           MVI     A,      0FFH            ;IS THE ADDRESS XXFFH?
1024  BD             CMP     L                       ; ...THIS COULD BE END OF RAM
1025  C22E10         JNZ     CNTF                    ;NOPE, PROCEED
1028  3E17           MVI     A,      RAMEH-1         ;YES, IS XXFFH, TEST HIGH BYTE.
102A  BC             CMP     H                       ; REACHED END OF RAM?
102B  CA4010         JZ      LASTF                   ;YES, DONE THIS PASS
                        ;
102E  19      CNTF:   DAD     D                       ;ADD ADDRESS INCREMENT TO H,L
102F  DA3810         JC      OVR                     ;CARRY MEANS OVERFLOW
1032  7C             MOV     A,      H               ;CHECK:
1033  FE18           CPI     RAMEH                   ;ADDRESS STILL VALID?
1035  FA1710         JM      RLOOP                   ;YES, PROCEED
1038  210014  OVR:    LXI     H,      RAMBEG          ;NO, WRAP AROUND
1038  33             INX     SP                      ;INCREMENT THE WRAP AROUND OFFSET
103C  39             DAD     SP                      ;AND ADD IT IN
103D  C31710         JMP     RLOOP                   ;CONTINUE THE PASS
                        ;
                        ; ***** DONE A PASS (A BIT POSITION) *****
                        ;
1040  48      LASTF:  MOV     C,      B               ;NEW 'OLD PATTERN' IS CURRENT PATTERN
1041  78             MOV     A,      B               ;MOVE INTO A FOR MANIPULATION
1042  FEFF           CPI     0FFH                    ;LAST BIT PATT. FOR WRITING 1'S?
```

```
1044  CA5410          JZ      DATO                    ;YES, NOW DO COMPLEMENT PATTERNS
1047  A7              ANA     A                       ;TEST A: IS IT ZERO?
1048  CA5910          JZ      RVRS                    ;YES, TIME FOR REVERSE SEQUENCE
104B  87              ADD     A                       ;NO, MAKE NEXT PATTERN
104C  DA5010          JC      RX1                     ;CY=1 MEANS WRITING 0'S; PATT. OK
104F  3C              INR     A                       ;CY=0 MEANS WRITING 1'S; HAVE TO INC
1050  47      RX1:    MOV     B,      A               ;PUT NEW PATTERN IN B
1051  C31110          JMP     FTL1                    ; & DO NEXT PASS
1054  06FE    DATO:   MVI     B,      0FEH            ;COMP. PATTERN STARTS WITH BIT 0 = 0
1056  C31110          JMP     FTL1                    ; NOW DO NEXT SERIES OF PASSES
                ;
                ;*****************************************************************
                ;***                                                           ***
                ;***                     REVERSE SEQUENCE                      ***
                ;***                                                           ***
                ;*****************************************************************
                ;
                ;               MEMORY CONTAINS ALL ZEROES UPON ENTRY HERE
                ;
1059  7A      RVRS:   MOV     A,      D               ;NEGATE THE ADDR. INCREMENT
105A  2F              CMA
105B  57              MOV     D,      A
105C  7B              MOV     A,      E
105D  2F              CMA
105E  5F              MOV     E,      A
105F  13              INX     D                       ;THIS COMPLETES THE 16 BIT NEGATION
1060  0601            MVI     B,      01H             ;START WITH BIT 0 = 1
                ;
1062  21FF17    RTL1:   LXI     H,      RAMEND-1        ;POINT TO END OF RAM
1065  310000            LXI     SP,     0000H           ;CLEAR WRAP AROUND OFFSET
                ;
1068  7E        RL2:    MOV     A,      M               ;READ MEMORY
1069  B9              CMP     C                       ;OLD PATTERN THERE?
106A  C2C210            JNZ     RAMEC                   ;NO -> RAM ERROR
106D  70                MOV     M,      B               ;YES, WRITE NEW PATTERN
106E  7E                MOV     A,      M               ;READ IT BACK
106F  B8                CMP     B                       ;IS IT RIGHT?
1070  C2CC10            JNZ     RAMEB                   ;NO
1073  AF                XRA     A                       ;CLEAR A SO WE CAN TEST...
1074  BD                CMP     L                       ;IS ADDR XX00H?
1075  C27E10            JNZ     CNTR                    ;NOPE, PROCEED
1078  3E14              MVI     A,      RAMBH           ;MAYBE, CHECK HIGH BYTE
107A  BC                CMP     H                       ;FINISHED?
107B  CA9010            JZ      LASTR                   ;YES, DONE THIS PASS
                ;
107E  19        CNTR:   DAD     D                       ;ADD THE ADDRESS DECREMENT
107F  D28810            JNC     UNDR                    ;UNDERFLOW IF CY=0
1082  7C                MOV     A,      H               ;CHECK:
1083  FE14              CPI     RAMBH                   ;ADDRESS STILL VALID?
1085  F26810            JP      RL2                     ;YES, PROCEED
1088  21FF17    UNDR:   LXI     H,      RAMEND-1        ;NO, WRAP AROUND
```

```
108B  3B          DCX   SP                ;DEC. WRAP AROUND OFFSET
108C  39          DAD   SP                ;AND ADD THIS TO H,L FOR NEW ADDR
108D  C36810      JMP   RL2               ; & CONTINUE
                  ;
                  ; ***** DONE A PASS (BIT PATTERN) *****
                  ;
1090  48    LASTR: MOV   C,    B          ;NEW 'OLD PATTERN' IS LAST 'NEW PATT'
1091  78          MOV   A,    B          ;MOVE INTO A FOR MANIPULATION
1092  FEFF        CPI   OFFH             ;LAST PATTERN FOR WRITING ONES?
1094  CAA410      JZ    DATRO            ;YES, NOW HAVE TO WRITE ZEROES
1097  A7          ANA   A                ;NO, IS A ZERO?
1098  CAA910      JZ    NXTAD            ;YES, DONE ALL BIT PATTERNS, THIS SEQ
109B  87          ADD   A                ;NO, MAKE NEW PATTERN
109C  DAA010      JC    RX2              ;CY=1 MEANS NEW PATERN IS OK NOW
109F  3C          INR   A                ;CY=0 MEANS HAVE TO ADD 1
10A0  47    RX2:  MOV   B,    A          ;PUT NEW PATTERN IN B
10A1  C36210      JMP   RTL1             ;DO NEXT PASS
                  ;
10A4  06FE  DATRO: MVI  B,    OFEH       ;COMP. PATT., BIT 0 = 0
10A6  C36210      JMP   RTL1             ;DO NEXT 8 PASSES
                  ;*****************************************************************
                  ;***                                                           ***
                  ;***              ADVANCE TO NEXT ADDRESS INCREMENT            ***
                  ;***                                                           ***
                  ;*****************************************************************
                  ;
10A9  7A    NXTAD: MOV   A,    D          ;NEGATE THE ADDR. INCREMENT
10AA  2F          CMA
10AB  67          MOV   H,    A
10AC  7B          MOV   A,    E
10AD  2F          CMA
10AE  6F          MOV   L,    A
10AF  23          INX   H                ;NOW (H,L)= -(D,E)
10B0  29          DAD   H                ;SHIFT LEFT FOR NEXT BIT
10B1  EB          XCHG                   ;MOVE ADDR. INC. BACK TO D,E
10B2  7A          MOV   A,    D          ;NOW CHECK:
10B3  FE04        CPI   RSIZH            ;IS ADDR. INC. TOO BIG?
10B5  FA0E10      JM    FTLOOP           ;NO, O.K.
                                         ;YES, WE ARE DONE!!!!!!!!!!!!
                  ;*************************** RAM PASSED ********************************
                  ;
10B8  21FD10      LXI   H,    MEMOK      ;POINT TO GOOD MEMORY MSG
10BB  C3D910      JMP   PRINT            ;PRINT IT
                  ;
10BE  310018 EXIT: LXI  SP,   1800H      ;FIX STACK POINTER
10C1  76          HLT                    ;STOP
                  ;
                  ;
                  ;
```

F.4.

```
                        ;
                        ;       RAMERR:  ERROR DETECTED
                        ;
10C2    320F11  RAMEC:  STA     MERR                    ;STORE BAD MEMORY CONTENTS
10C5    79              MOV     A,      C               ;AND NOW...
10C6    321011          STA     EXPCT                   ;STORE THE EXPECTED CONTENTS
10C9    C3D310          JMP     RAMERR
10CC    320F11  RAMEB:  STA     MERR                    ;STORE BAD MEMORY CONTENTS
10CF    78              MOV     A,      B               ;AND THEN...
10D0    321011          STA     EXPCT                   ;STORE EXPECTED CONTENTS
10D3    220D11  RAMERR: SHLD    HLERR                   ;SAVE H & L
10D6    21EE10          LXI     H,      EMSGR           ;POINT TO ERROR MESSAGE
                        ;
3C2C                     PORT EQU 3C2CH         ;CONSOLE TERMINAL PORT MEMORY MAPPED ADDR.
                        ;
                        ;       PRINT SUBRT:  PRINT STRING POINTED TO BY HL ON CONSOLE
                        ;
10D9    7E      PRINT:  MOV A, M                ;READ A CHAR FROM MEMORY
10DA    B7              ORA A                   ;TEST IT
10DB    CABE10          JZ EXIT                 ;WE'RE DONE IF ZERO
10DE    3A2D3C  TEST:   LDA PORT+1              ;READ STATUS
10E1    E604            ANI 04H                 ;TRANSMIT BUFFER EMPTY?
10E3    CADE10          JZ TEST                 ;NO, WAIT
10E6    7E              MOV A, M                ;GET CHAR AGAIN
10E7    23              INX H                   ;POINT TO NEXT CHAR
10E8    322C3C          STA PORT                ;WRITE THIS CHAR
10EB    C3D910          JMP PRINT               ;BACK FOR NEXT ONE
                        ;
10EE    0D0A3F52 EMSGR:  DB ODH,OAH,'?RAM ERROR',ODH,OAH,OOH ;ERROR MESSAGE
10F2    414D2045
10F6    52524F52
10FA    0D0A00
10FD    0D0A5241 MEMOK:  DB ODH,OAH,'RAM PASSED!',ODH,OAH,OOH
1101    4D205041
1105    53534544
1109    210D0A00
                        ;
110D    0000    HLERR:  DB 0, 0                 ;STORAGE FOR H & L ON ERROR
110F    00      MERR:   DB 0                    ;STORAGE FOR BAD MEMORY CONTENTS
1110    00      EXPCT:  DB 0                    ;STORAGE FOR EXPECTED MEMORY CONTENTS
                        ;
                        END
NO PROGRAM ERRORS
```

8080 MACRO ASSEMBLER, VER 2.4
ERRORS = 0 PAGE 6

SYMBOL TABLE

* 01

A	0007	B	0000	C	0001	CLOOP	1008
CNTF	102E	CNTR	107E	D	0002	DATO	1054
DATRO	10A4	E	0003	EMSGR	10EE	EXIT	10BE
EXPCT	1110	FTL1	1011	FTLOO	100E	H	0004
HLERR	110D	L	0005	LASTF	1040	LASTR	1090
M	0006	MEMOK	10FD	MERR	110F	NXTAD	10A9
OVR	1038	PORT	3C2C	PRINT	10D9	PSW	0006
RAMBE	1400	RAMBH	0014	RAMEB	10CC	RAMEC	10C2
RAMEH	0018	RAMEN	1800	RAMER	10D3	RAMSI	0400
RAMTS	1000 *	RL2	1068	RLOOP	1017	RSIZH	0004
RTL1	1062	RVRS	1059	RX1	1050	RX2	10A0
SP	0006	TEST	10DE	UNDR	1088		

F.6.

APPENDIX G

The following is a list of the micro-operations determined for the 8080 CPU based on the clock cycle by clock cycle breakdown of each instruction <4 , pp. 2-16 to 2-20) and the CPU functional block diagram <4 , p. 2-2).

Key:

r8	Any 8-bit register of the register array
r16	Any 16-bit register pair of the register array
ACT	Accumulator Latch
A	Accumulator
CY	Carry flag
TMP	Temp. register
Dbus	8-bit data bus
Abus	16-bit address bus

8080 Micro-operations:

1) r16 = Abus
2) r16a + 1 = r16b
3) Dbus = TMP and IR
4) r8 = TMP
5) A = TMP
6) TMP = r8
7) TMP = A
8) Dbus = r8
9) Dbus = A
10) Dbus = TMP
11) TMP = Dbus
12) A = Dbus
13) r8 = Dbus
14) (HL) = (DE) [XCHG]
15) r16a = r16b
16) A = ACT
17) r8 = ACT
18) ACT + TMP = ALU [i.e. ALU outputs]
19) ACT + TMP + CY = ALU
20) ACT - TMP = ALU
21) ACT - TMP - CY = ALU
22) TMP + 1 = ALU
23) TMP - 1 = ALU
24) ALU = r8
25) ALU = A
26) ALU = Dbus
27) r16a - 1 = r16b
28) DAA = A, flags [decimal adjust]
29) ACT AND TMP = ALU
30) ACT OR TMP = ALU
31) ACT XOR TMP = ALU
32) ALU = flags S, Z, P & AC
33) ALU = CY
34) CY = ALU
35) A = ALU

36) Rotate Right
37) Rotate Right through CY
38) Rotate Left
39) Rotate Left through CY
40) NOT A = A
41) NOT CY = CY
42) 1 = CY
43) Judge Condition
44) 00 = W reg.
45) Flags = Dbus
46) Dbus = Flags
47) Set INTE F/F
48) Reset INTE F/F
49) Halt Mode
50) Status: Instruction Fetch
51) Status: Memory Read
52) Status: Memory Write
53) Status: Stack Read
54) Status: Stack Write
55) Status: IN Read
56) Status: OUT write
57) Status: Interrupt Acknowledge
58) Status: Halt Acknowledge
59) Status: Interrupt Ack. while Halt
60) Dbus = r16high and r16low

Appendix H

Checksum Calculation Program

APPENDIX H

```
100       /     CHKSUM -- CALCULATE CHECKSUMS FOR AN OBJECT FILE
          /               IN SEGMENTS OF A DESIRED LENGTH.
          /     THE SUM IS FORMED USING A MODULO 256 ADD WITH CARRY
          /     OUTS ADDED BACK INTO LSB.
          /
110       /     THE SEGMENT LENGTHS AND DESIRED FINAL CHECKSUM
          /     ARE SELECTED BY THE USER.
          /
120       /     DAVID HAISLETT     MICROPROCESSOR SELF-TEST PROJECT
          /
400       HEX0$ = "0123456789ABCDEF"
500       DEF FNH%(D$) = (INSTR(1,HEX0$,LEFT$(D$,1))-1)*16 +
              INSTR(1,HEX0$,MID$(D$,2,1))-1
          /  RETURN INTEGER VALUE OF 2 DIGIT HEX STRING
          /
1000      PRINT CHR$(10);"CHKSUM - V01.01";CHR$(10)
1010      PRINT "INPUT FILE"; : INPUT FIL$ :
          IF INSTR(1,FIL$,".")=0 THEN FIL$=FIL$+".HEX"
1020      OPEN "I", #1, FIL$
1030      PRINT "ROM SEGMENT SIZE"; : INPUT ROMSS% :
          IF ROMSS%<3 THEN PRINT "?INVALID SIZE:  ";ROMSS% : GOTO 1030
1040      PRINT "DESIRED FINAL SUM"; : INPUT DSUM% :
          IF DSUM%<0 OR DSUM%>255 THEN PRINT "?OUT OF RANGE, ENTER 0-255"
          GOTO 1040
1050      HLINE$ = "" : CSUM% = 0
2000      FOR B% = 1 TO ROMSS% : GOSUB 10000 :
          IF OP% = -1 THEN IF B% = 1 THEN 2100 ELSE 2030
2005      ADDRL%=ADDR% : IF B%=1 THEN ADDR1%=ADDR%
2010      CSUM%=CSUM% + OP% :
          Z%=CSUM% AND (NOT 255) : CSUM%=CSUM% AND 255:
          IF Z% THEN CSUM%=CSUM%+1
2020      NEXT B%
2030      RSUM%=DSUM%-CSUM% :
          IF RSUM%<0 THEN RSUM%=RSUM%-1
2040      RSUM%=RSUM% AND 255
2050      PRINT "CHECKSUM FOR SEGMENT ";RIGHT$("000"+HEX$(ADDR1%),4);
          "H TO "; RIGHT$("000"+HEX$(ADDRL%),4);"H IS ";
          RIGHT$("0"+HEX$(RSUM%),2);"H"
2060      CSUM%=0
2070      IF OP%<>-1 THEN 2000
2100      CLOSE 1 : GOTO 32767
9999      /     ****************************************
          /     READ AN OBJECT CODE BYTE FROM HEX FILE #
          /     RETURN IT IN OP%
10000     IF HLINE$="" THEN LINE INPUT #1, H$ : H$=MID$(H$,2) :
          IF MID$(H$,1,1)<>":"
          THEN 10000 ELSE HBYTE%=FNH%(MID$(H$,2,2)) :
          ADDR%=FNH%(MID$(H$,4,2))*256 + FNH%(MID$(H$,6,2)) :
          FO%=-1 : HLINE$=MID$(H$,10,2*HBYTE%) : IF HBYTE%=0 THEN OP%=-1: RETURN
10010     OP%=FNH%(HLINE$) : OP$=LEFT$(HLINE$,2) :
          HLINE$=MID$(HLINE$,3)
10020     IF FO% THEN FO%=0 ELSE ADDR%=ADDR%+1
```

Appendix I

RADC Microprocessor Self-Test Project

Hardware System Documentation

The self-test system consists of an 8080 CPU, 8228 system controller, 2 8251 serial I/O ports, an 8255 parallel I/O port, 8253 programmable interval timer (with 3 independent timers), 2 8-bit data latches for various control functions, 2K of ROM containing the system monitor, and 4K of RAM. The system employs memory mapped I/O and thus does not use IN or OUT instructions. The system memory map is shown below:

Address (Hex)	Assignment
0000 - 07FF	ROM (monitor)
0800 - 17FF	System RAM
3C24	Down-line load 8251, data port
3C25	Down-line load 8251, command port
3C28	CNTL1: 8-bit latch; see below
3C2C	Console 8251, data port
3C2D	Console 8251, command port
3C30	CNTL2: 8-bit latch; see below
3C38	8253 Timer 0 (interrupt)
3C39	8253 Timer 1 (timeout)
3C3A	8253 Timer 2 (unused)
3C3B	8253 Control
3C3C	8255 Port A
3C3D	8255 Port B
3C3E	8255 Port C
3C3F	8255 Control

The 8253 Timer 0 is used to generate the interrupt that initiates the periodic self-test. This timer should be configured for Mode 0 since the interrupt is generated on the rising edge of Output 0. A RST 3 is executed upon interrupt acknowledge, and control is passed to RAM location 0A00H. Timer 1 is used to generate a hardware timeout; this occurs if the self-test is not initiated or is not completed in its allotted time. Timer 1 should be configured for Mode 0, since its output must go high and stay high for the timeout. This causes the ERR-bar LED to go out (indicating error). Timer 2 is unused and thus available to the user. The Timer 0 and 1 Gate inputs are controlled by the CNTL2 latch (3C30H); this latch is also responsible for enabling or disabling the hardware timeout (see below). Both Timers 0 and 1 are driven by the processor clock (.89 MHz).

The CNTL1 8-bit latch (IC 22) is responsible for controlling the baud rates of both 8251's; it also controls the console 8251 wraparound (self-test) logic and drives the 'heartbeat' LED. Its bits have the following meanings:

Bit(s)	Function
0	Low connects console 8251 to the console terminal; High wraps serial output to serial input.
1-3	Console 8251 baud rate control (see below).
4	Controls the 'heartbeat' LED: low = ON; high = OFF.
5-7	Down-line load 8251 baud rate control (see below).

Assuming the 8251's are programmed for 16x operation, the baud rate control is defined as follows:

Bit: 3 2 1 -- console 7 6 5 -- down-line load	
0 0 0	Do NOT use
0 0 1	19200 baud
0 1 0	9600 baud
0 1 1	4800 baud
1 0 0	2400 baud
1 0 1	1200 baud
1 1 0	600 baud
1 1 1	300 baud

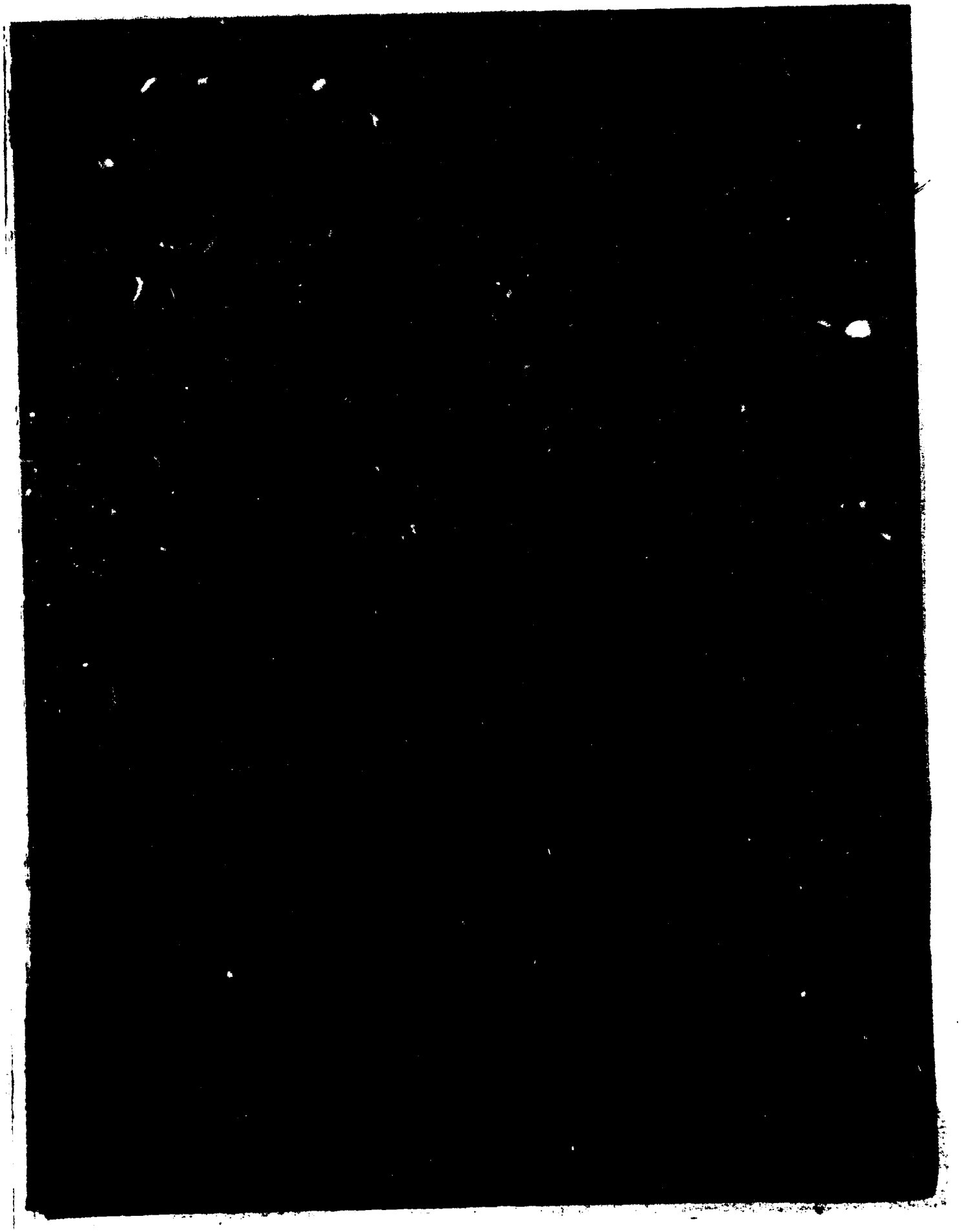
The CNTL2 8-bit latch (IC 17) is responsible for controlling the 8255 wraparound self-test logic, the 8253 Counters 0 & 1 gate inputs, and for enabling/disabling the hardware timeout. Its bits have the following functions:

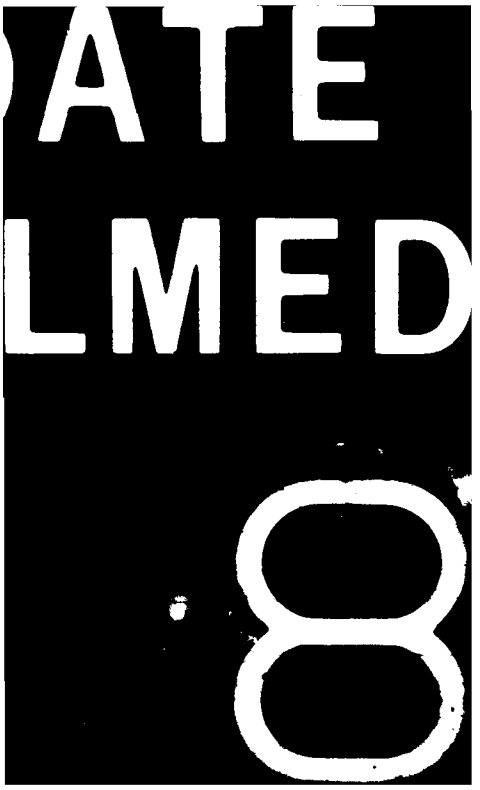
Bit	Function
0	8253 Gate 0 (Timer 0)
1	8253 Gate 1 (Timer 1)
2	High allows hardware timeout; Low forces the timeout.
3	High disables hardware timeout; Low enables it.
4	Low enables 8255 Port A wraparound logic; High disables it (& tri-states Port A).
5	Low sets 8255 Port A wraparound INTO Port A; High sets wraparound OUT OF Port A.
6	Low enables 8255 Port C wraparound logic; High disables it (& tri-states Port C).
7	Low sets 8255 Port C wraparound INTO Port C; High sets wraparound OUT OF Port C.

Note that Bit 5 is meaningless if Bit 4 is 1, and Bit 7 is meaningless if Bit 6 is 1.

To enable a hardware timeout (Timer 1), Bits 2 and 3 should be 1 and 0, respectively. To indicate an error (fault detected), Bits 2 and 3 should both be set to 0; this forces a 'timeout' and turns off the ERR-bar LED.

The system's 7-segment display is connected to the 8255's Port B; if Port B is configured as an output, then Port B is used to drive the display. When Port B is configured as an input, Port A or Port C may be used to drive the display via the wraparound logic. Note that low bits light segments, while high bits turn segments off.




ATE
LMED
8