

This document is for internal Xerox use only.

# D0 Diagnostic User's Guide

by Mike Spaur

April 10, 1980

Revision 1.0

This document describes the use of the D0 diagnostic microcode developed by the Electronics Division to aid in the functional testing of the D0 subsystem.

[ISIS]<EDMOD>Rev-1>DiagGuide.dm

**XEROX**

**REPROGRAPHIC TECHNOLOGY GROUP**  
**Electronics Division**  
Los Angeles

This document is for internal Xerox use only.

**TABLE OF CONTENTS**

1.0 SCOPE	2
2.0 APPLICABLE DOCUMENTS	2
3.0 THE ENHANCED DIAGNOSTICS	3
4.0 ORGANIZATION OF DOCUMENTS	6
4.1 Run Files	6
4.2 Documentation Files	7
5.0 THE SOURCE CODE	7
6.0 FLOW CHART CONVENTIONS	9
7.0 THE MACHINE LISTING	11
8.0 USING THE MIDAS LOGGER	12
9.0 THE NEW MIDAS DISPLAY	13
APPENDIX	15

## 1.0 Scope

This guide is intended to show how to use the Diagnostics that were enhanced by the Electronics Division (ED) to test the TOR D0 processor configuration. This document will cover the following areas:

1. What the enhanced diagnostics are.
2. Where they are and how to get them.
3. How to read the related documentation.
4. How to run the programs.
5. How to use the Midas logger.
6. How the new Midas display is organized.

In order for this document to be useful, the user must have some experience with running microprograms under Midas control, but does not need to be able to write or understand microcode statements. *The D0 Hardware Manual of May 16, 1979 is a required reference to this guide.*

I would like to extend my thanks to the ED diagnostics team for their help in preparing this document, in particular I would like to thank Ray Matsuda and Carter Tseng who have been especially helpful. Any questions or comments regarding this guide can be directed to: Mike Spaur, DPP/ED, El Segundo, extension 1507.

## 2.0 Applicable Documents

1. D0 Hardware Manual  
RTG, May 16, 1979
2. D0 Micro Assembler Manual  
B. Rosen, December 30, 1977
3. Micro: Machine Independent Micro Assembler  
E. Fiala et. al., August 29, 1978
4. MicroD Manual  
P. Deutsch, October 20, 1978
5. D0 Micro Programmer's Guide  
C. Hankins, August 22, 1978
6. Midas Manual  
E. Fiala, December 29, 1977
7. D0 Midas Manual  
B. Rosen, December 30, 1977
8. D0/TOR ESS Manufacturing Test Strategy  
H. Kakita, December 4, 1979

### 3.0 The Enhanced Diagnostics

The ED microdiagnostics are enhanced versions of the functional microcode that have been standardized for the following purposes:

1. To reduce familiarization time through better readability of the documentation.
2. To install certain features that are common to the Midas display for all tests, thereby making them easier to use.
3. To install looping capabilities to make the programs more effective for diagnosing hardware errors.
4. To install a Midas logger to enable Midas to control the test and log the breakpoints that are encountered.

Note that in most cases the coverage of these diagnostics have not been increased. Therefore, if the original functional microcode has left certain areas of the processor untested, the enhanced diagnostics also leave these same areas untested.

The diagnostic programs are tests written in D0 microcode to test various portions of the D0 processor under Midas control. The tests are loaded into the D0 instruction memory using Midas. Midas is a loader/debugger that runs on an Alto and controls the D0 remotely. Some of the files required to support Midas are:

Midas.run	(the Midas run file)
Midas.programs	(contains the name of command files required by Midas)
Midas.syms	(a symbols file for Midas)
Kernel.mb	(microcode that runs in the D0 and talks to Midas)
Midas.midas	(defines the initial screen)

All of the files needed to support Midas (there are 10) can be retrieved by loading from the dump file [ISIS]<EDMOD>Rev-1>Midas.dm. Note that these files require that your disk executive be at Operating System 17. The following is a list of the diagnostics that have been revised, with an indication of the hardware it is intended to test. *Note that the name of the revised diagnostics are the same as the original functional microcode except that the name is preceded by the letters ED.*

EDALU:	This test exercises some basic functions of the ALU module. Configuration: The standard four CPU boards.
EDBitBlt:	This test exercises the Bitblt (bit boundary block transfer) hardware. It moves information from one region of main storage to another, and modifies the information at the destination. Configuration: The standard four CPU boards.
EDBootD0:	This is a BCPL program that runs in the Alto and loads the Kernel.mb file into the D0 Control Store. Because this program is not written in D0 microcode, it's documentation is not the same as the other tests. Configuration: The standard four CPU boards.

- EDCSEx:** This test exercises the Control Store module as a 4K by 36 bit memory. Note that this test does not exercise the first 256 or the last 512 Control Store locations to protect the test program and the kernel.  
Configuration: The standard four CPU boards.
- EDCyM:** This test exercises all functions of the Cyclor/Masker except the FixVA functions which are tested by EDTNF.  
Configuration: The standard four CPU boards.
- EDField:** This test simulates the RF, WFA and WFB field operations without using the cyclor masker and then compares the results with the actual executions of the corresponding operations.  
Configuration: The standard four CPU boards.
- EDMemAbort:** This diagnostic tests the memory reference abort conditions.  
Configuration: The standard four CPU boards and one 96K storage module.
- EDMemEx:** This program tests the memory mapping logic and 96K storage modules.  
Configuration: The Standard four CPU boards and the 96K modules to be tested (up to eight 96K boards may be tested).
- EDProc:** This is a midas command file that links all of the other EDXXX tests together and runs them one at a time, beginning with CPU tests and progressing to the memory tests. Note that this program stops at the EDMemEx breakpoint "Found X 96K" where X is the number of 96K storage boards found by the EDMemEx program. The user should verify at this point that the number of boards found by the EDMemEx is correct and then bug continue.  
Configuration: The standard four CPU boards and at least one 96K storage module.
- EDLProc:** This test is the same as EDProc except that it has added looping capabilities for overnight testing. Thus it doesnot stop at any breakpoints.  
Configuration: The standard four CPU boards and at least one 96K storage module.
- EDRDCCD:** This test exercises the SA4000 Rigid Disk Controller.  
Configuration: The standard four CPU boards, one 96K storage module, an RDC module and an SA4000 disk drive.
- EDRMEx:** This program exercises all of the R-registers except those used in this program or by the kernel.  
Configuration: The standard four CPU boards.

- EDSmallMem:** This test exhaustively exercises the control store as a 4K by 36 bit memory excluding the first 256 and the last 512 locations. Also, this tests exercises all of the T-registers except T[16] and T[17].  
Configuration: The standard four CPU boards.
- EDTask:** This program exercises the task switching hardware.  
Configuration: The standard four CPU boards.
- EDTimEx:** This test exercises timers 0 to 15. It replaces the page 16 portion of the standard kernel.  
Configuration: The standard four CPU boards.
- EDTMP:** This test exercises various features of the maintenance panel display. Note that this test involves interaction with the user. The user is required to observe that the proper numbers appear on the the maintenance panel display. For this reason no Midas logger was implemented for this program.  
Configuration: The standard four CPU boards and a four digit maintenance pannel (featuring the TMS1000 circuit).
- EDTNF:** TNF stands for Test New Functions. This program tests the NewInst feature, the Byte Code register and the new field descriptors.  
Configuration: The standard four CPU boards.
- EDUTVFC1:** This diagnostic exercises the User Terminal Variable Format Controller.  
Configuration: The standard four CPU boards, one 96K storage module, a UTVFC module and the Large Format (LF) display, LF Keyboard and associated power supply/interface unit.

As an overview, the following table gives a list of the D0 modules and the diagnostics that deal with each one. Note that all of the diagnostics rely on the ALU, the Control Store, and the Miscellaneous Processor modules to be in working order, but the EDALU test, for example, is specifically designed to test certain features of the ALU module.

<u>Processor Hardware</u>	<u>Diagnostics Involved</u>
1. ALU	EDALU EDBitBlt EDCyM EDField EDTNF

2. Control Store	EDBootD0 EDCSEx EDMemAbort EDRMEx EDSmallMem EDTask
3. Ethernet Controller	EDPackets
4. Maintenance Panel	EDTMP
5. Memory Controller	EDMemAbort EDMemEx
6. Miscellaneous Processor Module	EDBitBlt EDMemAbort EDTask
7. Rigid Disk Controller (SA4000)	EDRDCD
8. User Terminal Full-page Controller	EDUTVFC1
9. 96K Storage Board	EDMemEx EDMemAbort

#### 4.0 Organization of Documents

All of the enhanced diagnostics are kept on [ISIS]KEDMOD>Rev-1>. The files on this account can be divided into two main categories: run files and documentation files. Strict naming conventions have been used in creating these files. All of the documentation for the EDXXX test for example, would be kept in files that begin with EDXXX and end with an extension that indicates what the contents of the file are. Sections 4.1 and 4.2 give a list of file extensions with an explanation of the corresponding contents.

##### 4.1 Run Files

All of the files necessary to run the EDXXX test are obtained by loading from the EDXXX.dmttest file on [ISIS]KEDMOD>Rev-1>. This dump file contains all of the necessary files needed to run the EDXXX test assuming that you already have a Midas system on your disk. The following files are contained in EDXXX.dmttest.

- EDXXX.mb This is a microbinary file which is loaded into the D0 by Midas, and executed by the D0 under Midas control.
- EDXXX.midas This file defines the Midas display for the EDXXX test.
- EDXXXLog.midas This is a command file that allows Midas to run the EDXXX test and log the results in a bravo file called EDXXX.report. The EDXXXLog.midas file will be referred to in this guide as the Midas logger and its use is discussed in section 8.0.

## 4.2 Documentation Files

You can obtain a hard copy of the documentation files by loading from EDXXX.dmdoc at [ISIS]<EDMOD>Rev-1> assuming that you have Empress.run on your disk. This dump file contains all of the following documentation for the EDXXX test.

- EDXXX.mc: This is the source code that was compiled to produce the EDXXX.mb file. The organization of the EDXXX.mc file is described in detail in section five.
- EDXXX.silpress: This is a set of flow charts that describe the workings of the EDXXX test. The symbols used in these charts are defined in section six.
- EDXXX.dls: This document is a list of all of the instructions that are loaded into the D0 and their corresponding locations in the D0 Control Store. How to read this file is discussed in section seven.

## 5.0 The Source Code

The source code for D0 diagnostics is contained in a file that will be named EDXXX.mc. The ~.mc file contains the D0 microcode statements that were compiled to produce the microbinary file that is loaded into the D0 Control Store by Midas. The ~.mc file has some very useful information on the front pages. The organization of this information is described in this section. All of the descriptions in this section refer to pages A1 to A7 of the appendix. These pages contain sections of the EDMemEx.mc file, and are listed for illustrative purposes only.

The first pages of the ~.mc file are organized into blocks of information. At the top of the first page is the title block (see item 1 page A1). This block contains the name of the test, the purpose, the required hardware configuration, the author, and a history of the modifications, along with name of the person who altered the test.

Item 2 on page A1 contains a list of the subtests in the EDMemEx test, with a short description of what each subtest does.



Item 3 on page A1 contains a list of breakpoints that may be encountered during the EDMemEx test, and explains the meaning of each one.

Item 4 on page A2 contains a table of the breakpoints with a list of corresponding Control Store addresses. A logic analyzer can be synchronized to these addresses and events that occur near this location in the program can be observed. (see section 7.0 for more details).

Item 5 on page A2 contains the names of special user definable R-registers, with an explanation of the meaning of each one. These registers will be shown on the Midas display. They can be changed by the user and their value will have an effect on the parameters of the program as it runs in the D0. All ~.mc file numbers and all numbers displayed by Midas are in octal.

Item 6 on page A3 contains a list of the subroutines used by the EDMemEx test, with a comment on what each one does.

The actual source code begins on page A5. The initialization statements occur at the beginning of the source code (see item 7 page A5). After the initialization statements, the initial values of certain constants and R-registers are given (items 8 and 9 on page A5). Here are some examples:

SET[Mumble,X]: This statement initializes a 16 bit constant named "Mumble" and sets it equal to the octal value of X.

MC[Foo,Y]: This statement initializes an 8 bit constant named "Foo" and sets it equal to the octal value of Y. Either the least significant or most significant half of Foo may be specified by the octal value of Y. The other 8 bits must be zero.

RV[Rexample,A,B]: This statement gives the Ath R-register the name "Rexample", and sets it equal to B, where A and B are octal numbers. The statement of A is optional.

Along side of these initialization statements may be some comments that are preceded by an asterisk (\*). These comments will explain how the constant or register is used in the program.

After the initialization statements are made, the main routine is listed (see item 10 on page A7). each line of this main routine equates to an actual instruction in the D0 Control Store. Each of the microcode statements ends with a colon (:). These statements may be followed by a comment which will be preceded by an asterisk (\*). These comments will make some statement about what the program is doing.

After the main routine, the source code for the subroutines is listed. It is not necessary that you be able to understand all of these microcode statements to be able to use the test, but certainly it could be of assistance to you. See reference numbers 2 through 5 in section 2.0 for further information regarding D0 microcode.

## 6.0 Flow Chart Conventions

This section gives a description of the conventions used in flow charting the new diagnostics. The basic symbols used are shown on page A8 of the appendix. All of the comments in this section refer to that page.

Figure 1 on page A8 is the symbol used to indicate the beginning of execution of the program.

Figure 2 is a statement box. The program executes these boxes in sequential order. All statements in any box will be executed before the program moves on to the next item.

Figure 3 shows an instruction box. This box symbolizes a special instruction that has been implemented to avoid repetitious subroutines. Figures 7, 8, 10, 11 and 12 are examples of these instructions, and will be discussed in more detail below.

Figure 4 is a decision box. The statement it contains will be mathematical or logical in nature. If the statement contained in the box is true then the program will proceed to the next item that occurs in the "yes" branch, otherwise program flow will follow the "no" branch.

Figure 5 illustrates a label. All of the names on the flow chart that appear in bold letters in front of the boxes and are followed by a colon are labels. These labels can be found in the source code file (~.mc) and are associated with actual locations in the Control Store. The EDXXX.dls file will be referenced to these labels, and Midas is aware of them. It is possible to begin executing or stop executing the program at any location using Midas and referencing these labels.

Figure 6 is a jump box. Following this item, execution will begin at label "mumble" which appears on page X *of the flow charts*.

Figure 7 is a symbol that represents a special function internal to the processor which clears the maintenance panel. If execution of the program halts immediately after this instruction then the display should read 0000.

Figure 8 is a symbol that represents another internal function which increments the maintenance panel by one. These maintenance panel functions are used in the diagnostics to display where execution is occurring in the program.

Figure 9 is an example of a call statement. If this symbol occurs to the right side or underneath a statement or decision box, then after the program is done with that box it will begin execution of the items that follow the label "Foo" on page y. The program will continue to execute the statements that follow the "Foo" label in a normal way until it encounters a return statement.

Figure 10 is the symbol that represents the return statement. When it is encountered, the program will jump back to the previous call statement and start execution of the items directly beneath that call. However, this may not happen immediately. If there is a higher priority task requesting service (such as memory refresh), then that task may get control of the processor after the return statement is encountered. If this happens, then control of the processor will be returned to the program when the higher priority tasks have finished. The program will then begin execution of the items that follow the previous call.

Figure 11 is the task symbol. This symbol represents a special microcode statement that allows another higher priority task to have control of the processor. It is equivalent to a call followed by a return statement. When the higher priority tasks are finished, execution will begin at the item which is directly beneath the task statement. Branches may be connected to the path underneath a task symbol, especially if the diagnostic is designed for input output hardware. This is to let you know that other tasks will possibly get control of the processor if certain conditions occur. A note should appear by each of these branches to let you know what these conditions are. For instance a note may appear by one of the branches that reads: "Task 2 if wakeup occurs".

Figure 12 is the notify symbol. This statement causes task switching like the task statement. But it also forces the processor to return to a specific task and location. This task and location is specified by the contents of the APC and APCTask registers. For a more detailed description of the notify statement, see page 54 of reference 5 (section 2.0). Again branches may appear under the notify symbol to indicate the various tasks that may run next.

Figure 13 is an illustration of the loop path symbol. When the decision box is reached, the processor looks at the register "ShortLoop" and sees if it is equal to one. If it is, the program execution follows the heavy black line, and will continue to do so until ShortLoop is no longer equal to one, or until Midas encounters a breakpoint or a mouse halt. Whenever you see heavy black lines on the flow chart it means that this is a path that can be looped on. Loop paths have been installed so that some special part of the program can be looped on for trouble shooting purposes.

Figure 14 is the breakpoint symbol. When the program encounters this item, execution will halt and control will be given to user via Midas. If the "continue"

item on the Midas menu is bugged, the program will resume execution with the item that is directly beneath this breakpoint. Breakpoints are usually inserted in a program to halt execution when the hardware being tested has failed to pass some subtest of the program.

Figure 15 is the dispatch symbol. A dispatch is a multiple branch to up to sixteen different locations. First a variable (X in this example) is set up. The branch takes place based on the value of this variable. The numbers on the legs of the branches represent the possible values that X can assume. If all of the possible values are not indicated, some default condition should be stated.

Figure 16 is the end symbol. It means that if the program encounters the breakpoint preceding this symbol, then it has finished one complete iteration of the EDXXX test. Usually the program will start again from the beginning for another iteration if the continue item is bugged from this breakpoint.

## 7.0 The Machine Listing

The machine listing file is produced by a compiler from the source file. The machine listing file will be named EDXXX.dls. The letters dls stand for D-machine listing. An example of the ~.dls file for the EDCyM program is given on page A9 of the appendix. The following is a list of the information given in column form in the EDXXX.dls file.

Imag	(The Control Store Address in imaginary format)
Real	(The Control Store Address as it appears on the CIA lines)
W0	(The least significant 15 bits of the micro instruction)
W1	(The next significant 16 bits of the micro instruction)
W2	(The most significant 4 bits of the micro instruction)
Symbol	(A reference to the closest preceding label)

Note that next to imaginary address 26 in the EDCyM.dls file there is a "b" between the Imag and Real columns. This is to let you know that there is a breakpoint inserted in this location. You may see an "@" sign between the Imag and Real columns. This is to let you know that the instruction that appears next to it has a fix location which is specified in the ~.mc file by the At[N] instruction.

This information is given for every instruction in the EDXXX.mc file. The most interesting information for the purpose of debugging is the real address column. The real address column lists the actual address that will appear on the 12 bit Control Store address lines CIA[00:11] during that instruction's cycle.

In the real address column you may see some number such as 1463 for a real address. The numbers that appear in the ~.dls file are all in octal. The binary equivalent 1463 octal is 001 100 110 011. In this case, the least significant bit

(CIA.11) would be a one, and the most significant bit (CIA.00) would be a zero.

Using the appropriate sil drawings for a particular revision of the Control Store module, a good place to connect test probes can be found. Then a logic analyzer can be triggered on the Control Store address lines. Any event in the processor can be observed by synchronizing on the address of an instruction which occurs just before the event. *Be careful, the CIA bus exists in complemented form in several places on the Control Store module.*

The real Control Store addresses of each breakpoint in the program have been put into table form and listed at the beginning of the ~.mc file (see page A2 item 4). These are the Real addresses to which the program will proceed from the given breakpoint.

## 8.0 Using The Midas Logger

A Midas logger has been created for each microdiagnostic. The logger command file for the EDXXX test will be named "EDXXXLog.midas". This logger is a Midas command file that runs Midas and logs certain information in a file called EDXXX.report. An example of the print out for the EDCyM program is given on page A10 of the appendix.

If the hardware should fail the test at some given breakpoint in the program, the logger will print pertinent information regarding that breakpoint. In the EDCyM program for instance, if the logger encounters the breakpoint "ResultBad", it enters the text "Result does not equal myResult.". It then logs the value of the following registers: (see page A10)

```
Result
myResult
SubTest
PassCount
```

If the processor should fail to run the program properly and fail at some location other than a program breakpoint, then the logger will record the state of the following column A registers:

```
Parity
CIA
CTask
APCTask
APC
TPC
```

Once the Midas system is started, the logger can be activated by typing "EDXXXLOG" on the Midas input text line and then bugging "Read-Cmds". The logger command file will then restart the EDXXX test and begin logging the results.

To stop the Midas logger, bug the "Abort" item while the logger is running the test. Control will then return to the Alto executive and a hard copy of EDXXX.report can be obtained using empres.

Note that when the test fails not at a standard breakpoint, the logger records the status in EDXXX.report and then attempts to restart the EDXXX test. In this instance the logger cannot loop on the failure. If ShortLoop is set equal to one when the logger encounters this failure, it will log the results and restart the test, which will reinitialize the ShortLoop register to zero. The logger should be helpful though for determining reliability and for long term exercising.

## 9.0 The New Midas Display

Page A11 of the appendix shows an example of the standardized Midas display (this one is for the EDCyM program). Column A has the same processor registers that are displayed for all tests as defined by the Midas.midas file.

Note that some of the registers shown on the display on page A11 have an asterisk (\*) next to their name. Any register on the midas display whose value has changed since the last time displayed (since the last breakpoint) will have an asterisk next to its name.

Locations B0 to B4 have been standardized, and are now the same for all EDXXX tests. Here is what they contain:

<u>Location</u>	<u>Register Name</u>	<u>Meaning</u>
B0	Revision	The revision level of this version of the EDXXX test
B1	Run-Time	The octal number of seconds it takes to run the EDXXX test once.
B2	Passcount	The number of passes of the entire EDXXX test that have been completed.
B3	MaxPass	The number of passes of the entire EDXXX test after which the program will breakpoint at "Passed-EDXXX-Test". Feel free to change this number by using Midas.
B4	SubTest	The current subtest of the EDXXX program that you are in.

Locations C0 to C3 are the communications registers Comm-Err0, Comm-Err1, Comm-Err2, and Boot-Err. These registers are counters that keep track of how many times certain boot and communications events occurred with errors. They should always contain zero values.

Location C4 is the Boot Reason register. This is a 6 bit register that indicates the cause of the last processor boot. Only one bit should be set in this register. The bits have the following meaning:

<u>BootReason Displayed</u>	<u>Corresponding bit</u>	<u>Reason for boot</u>
40C	Bit 10 = 1	Tester boot
20C	Bit 11 = 1	Maintenance panel boot
10C	Bit 12 = 1	Watchdog timer boot
4C	Bit 13 = 1	Group B function boot
2C	Bit 14 = 1	Power boot
1C	Bit 15 = 1	Parity error during task 15

While running Midas, the BootReason register will normally contain a 40 to indicate that the processor was boot strapped by tester hardware.

Location C5 contains the MemSyndrome register. The contents of this register is a memory syndrome that is used to correct single errors and detect double errors in memory data. This register will probably never be of use to you, but donot be alarmed by a non-zero value here.

In the lower right hand corner of the register fields are the user definable registers. There may be several registers in this field but there will always be one register called ShortLoop or LoopOn there. The ShortLoop or LoopOn register is set equal to one by the user if he wishes to loop on the current subtest. Always check the flow charts to see how this register is used in the test. Not all subtests in every program have this capability.

If there are any other registers in this user definable area, then their meaning and how to use them will be discussed in the special register section of the ~.mc file (see page A2 item 5). All of the user definable registers will always be the bottom group of registers in the C column.

The rest of the area on the Midas display is devoted to registers that are peculiar to the EDXX program. The purpose of these registers will be explained in the flow charts and in the front pages of the ~.mc file. These registers can be altered by the user, but for the results to be meaningful you must have a clear idea how the register is being used in the program. In this area, knowledge of microcoding is necessary. However, the diagnostics should be fairly thorough as written, and you should not need to alter these special registers to test the hardware extensively. These registers were placed on the display to show you what state the program is in.

**Appendix**





figure 1



figure 2

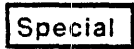


figure 3

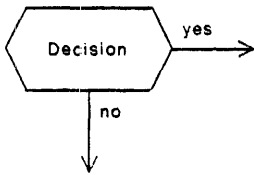


figure 4

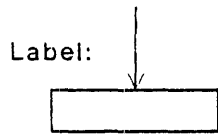


figure 5

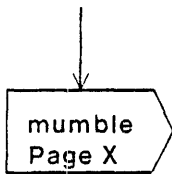


figure 6



figure 7



figure 8

CALL: Foo  
Page y

figure 9

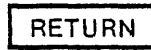


figure 10

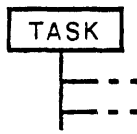


figure 11



figure 12

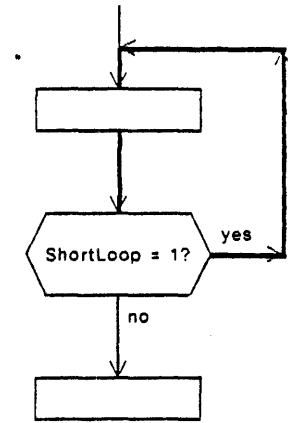


figure 13



figure 14

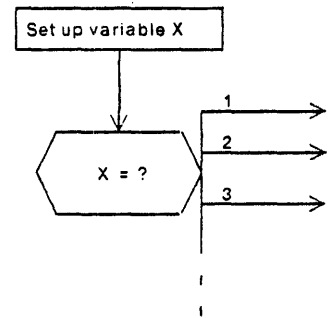


figure 15



figure 16

### Standard Flow Chart Symbols

MicroD 8.6 (OS 16) of April 27, 1979  
at 20-Feb-80 8:22:11

microd.run EDCym

EDCym.DIB 561b instructions written 20-Feb-80 8:21:02

Total of 561b instructions

Checking for errors...

Linking...

Building allocation lists...

Assigning locations...

561b instructions in rings involving ONPAGE or AT

Reloading binaries...

Checking assignment...

Writing .MB file...

Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
EDCym.DIB:					
0	1404	22005	107060	15	GO START
1	1530	22320	101057	15	(+1)
2	1527	22001	123055	11	(+2)
3	1526	22323	115053	11	(+3)
4	1525	47	7051	1	(+4)
5	1524	20020	101046	11	(+5)
6	1523	20020	101044	1	(+6)
7	1522	10020	101043	11	(+7)
10	1521	10020	101041	15	(+10)
11	1520	12020	101037	1	(+11)
12	1517	12020	101035	5	(+12)
13	1516	12020	101033	11	(+13)
14	1515	12020	101031	15	(+14)
15	1514	10004	101026	5	(+15)
16	1513	10320	105024	5	(+16)
17	1512	50	25023	1	(+17)
20	1401	47	5023	0	BIGLOOP
21	1411	21050	125020	0	(+1)
22	1410	20150	65017	4	(+2)
23	1407	21450	25015	0	(+3)
24	1406	50	24205	0	(+4)
25	1403	50	25023	1	(+5)
26	b 1402	50	25012	0	PASSED-EDCYM-TEST
27	1405	20020	101022	1	(+1)
30	1511	22020	101020	1	MAINLOOP
31	1510	22020	101017	5	(+1)
32	1507	21050	125015	11	(+2)
33	1506	50	24002	0	(+3)
34	1400	22150	65013	15	(+4)
35	1505	23174	45010	15	(+5)
36	1504	23174	67007	15	(+6)
37	1503	23150	65004	11	(+7)
40	1502	22050	125003	15	(+10)
41	1501	22000	103001	1	SETVARS
42	1500	22165	67177	14	(+1)
43	1477	24050	125174	0	(+2)
44	1476	20150	65172	0	(+3)
45	1475	23150	65171	14	(+4)
46	1474	24050	125167	4	(+5)
47	1473	10150	65164	4	(+6)
50	1472	26050	125162	14	(+7)

\*\*\*\*\* START EDCym Test :31-Dec-00 16:59:18 PST\*\*\*\*\*

----- Passed EDCym Test :31-Dec-00 16:59:32 PST-----

----- Passed EDCym Test :31-Dec-00 16:59:38 PST-----

----- Passed EDCym Test :31-Dec-00 16:59:45 PST-----

\*\*\* FAILED: at my Breakpoint  
\*       Result does not equal to myResult  
' Result =1037  
' myResult =0  
' SUBTEST =4  
' PASSCOUNT =0

\*\*\* FAILED: Not at my breakpoint  
' Parity =0  
' CIA =7777  
' CTASK =16  
' APCTASK =0  
' APC =1037  
' TPC =7777

----- Passed EDCym Test :31-Dec-00 17:00:04 PST-----

Standard A Column

Standard ED Midas Registers

Communications Registers

	A	B	C
0	PARITY 0	REVISION 1	COMM-ER0 0
	CYCLECONTROL 43	RUN-TIME 7	COMM-ER1 0
2	PCXREG 0	*PASSCOUNT 2	COMM-ER2 0
	PCFREG 0	MAXPASS 2	BOOT-ERR 0
4	DBREG 17	SUBTEST 0	BOOTREASON 40
	SBREG 17		MEMSYNDROME 177757
6	MNBR 1037		
	SSTKP 377	*XA 162505	*MYRESULT 16
8	STKP 0	*FNUM 345	*RESULT 16
	*ALURESULT 6	*OPERAND 162506	*APGRESULT 1037
10	SALUF 0		
	*T 20 2	INNERLOOPCOUN 0	*CS0 14176
12	AATOVA 0		*CS1 113002
	TPC 20 7777		*CS2 5
14	CALLER ILC0+7217	*LOFTTEST 6777	*CSP 70405
	PAGE 3	*DISPATCHTEST 72000	
16	APC 7011	*LSHTTEST 17000	SHORTLOOP 0
	APCTASK 16	*LCYTEST 16777	
18	*PASSED-EDCYM-TEST	*LHMASKTEST 1000	User Definable Registers
	CTASK 0	*ZEROTEST 1000	

Loaded: EDCYM

Time: 05.89

Resume 0:60+1, 8P at 0: PASSED-EDCYM-TEST

EDCyM Program Registers

Exit Boot Run-Prog Read-Cmds Break UnBreak C1rAddedBPs C1rAllBPs ShowBPs Go  
 SS Continue Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

%

```

*** **>Edalu.mc Rev 1 1/8/80
*****
*** EDALU.mc : Central-data-paths-exerciser microcode
*** Purpose : This test exercises some basic functions of the ALU module.
*** Hardware Configuration : Standard 4 CPU boards.
*** Written by : (unknown)
*** Modified by : C. Thacker, June 16, 1979
*** Modified by : M. Thomson & J. Kellman, Jan. 8, 1979
*** Standardize title page and code format.
*****

```

```

*****
* SubTest Description:
* Subtests are indicated in the flow charts and .mc file by:
* 1) Mcount (in octal)
* 2) Mpanel (in decimal)
*****

```

```

*****
* Subroutine Description:
* compare0: Compares contents of s0 (R file) with T
* poptst: Checks the contents of the stack after it has been loaded by push operations
* TchKR: Takes pattern in T, break points if discrepancy is found
* TfillR: Write pattern in T from Rmax through Rmin in R file
*****

```

```

*****
* Breakpoints:
* AchkRer: Value in R did not match R address.
* badnotify: Task failed to switch.
* fail: Value in s0 (R file) did not match value in T.
* TchkRer0: Value in s1 (R file) did not match value in T.
* TchkRer1: Value in s0 (R file) did not match value in T.
* TchkRer2: Value in stack did not match stack address.
* TfillRer: Value in s1 (R file) did not match value in T.
* Passed-EDALU-Test: the system passed thru all the passes of EDALU.
*****

```

```

*****
* Breakpoint Logic Analyzer Sync Points:
* AchkRer: Control Store address 415
* badnotify: Control Store address 2555
* fail: Control Store address 3
* TchkRer0: Control Store address 407
* TchkRer1: Control Store address 405
* TchkRer2: Control Store address 403
* TfillRer: Control Store address 525
* Passed-EDALU-Test: Control Store address 1000
*****

```

%

\*\*\*\*\*  
 \* INITIALIZATION:

BUILTIN[INSERT,24];  
 INSERT[d0lang];  
 TITLE[Central-data-paths-exerciser];

\*\*\*\*\* Set constants: \*\*\*\*\*

\*\*\* Code to page allocation: \*\*\*

\*\*\* Page \*\*\*            \*\*\* principal entry points \*\*\*

SET[Rpage, 1];            \* stkst, flone, flzero, TfillR, TchKR, AfillR, AchKR  
 SET[pushpage, 2];        \* push1, pushm1, push2, pushm2, push3, pushm3  
 SET[popxpage, 3];        \* popst extension: pop3, popm3  
 SET[poppage, 4];         \* popst: pop1, popm1, pop2e, pop2o, popm2e, popm2o  
 SET[alupage, 5];         \* go, contst, acontst, alutst

\*\*\*\*\* Macro definitions: \*\*\*\*\*

M@[ones, (ZERO)-1];

\* compares s0 with T, break point results if the two are not equal

M@[compare,  
 IFG@[10,FVAL@[PGE@],  
 SELECT@[FVAL@[PGE@], call[compare0], call[compare1], call[compare2],  
 call[compare3], call[compare4], call[compare5], call[compare6],  
 call[compare7]],  
 SELECT@[SUB[FVAL@[PGE@], 10], call[compare10], call[compare11],  
 call[compare12], call[compare13], call[compare14], call[compare15],  
 call[compare16], call[compare17]]];

\*\*\*\*\* Macro constants: \*\*\*\*\*

MC[Rmin, 20];            \* First R location tested  
 MC[Rmax, 317];          \* Last R location tested  
  
 MC[faultTrapLoc, 0];    \* special control store locations used by hardware  
 MC[bootTrapLoc, 1];

\*\*\*\*\* R-registers: \*\*\*\*\*

RV[s0, 0];                \* R store temporaries  
 RV[s1, 1];  
 RV[rt, 2];                \* stack test iterations count  
 RV[oldapc, 3];            \* holds return address during popst  
 RV[revision,4,1];        \* revision number is 1 for this program  
 RV[Mcount,5];            \* octal equivalent of maintenance panel display  
 RV[Run-time,6,11];      \* run time for this test is 9 seconds  
 RV[PassCount,7,0];      \* pass count for this program  
 RV[MaxPass,10,1000];    \* maximum number of passes for this run

\*\*\*\*\*  
 \*\*\* PRELIMINARY routine:

SET[sTestBase, lshift[alupage, 10]];  
 MC[sTestTask, add[sTestBase, 150000]];  
 MC[sTestBC, sTestBase];

ON PAGE[alupage];

start:

go:    clearMpanel; Mcount ← ZERO;    \* initialize Mpanel to 0000, Mcount to 0000

    s0 ← sTestBC;  
     APCtask&APC ← s0;  
     return;                          \* generate task switch

badnotify: breakpoint, goto[go];    \* should never reach here

    \* set tasks 1-15 to breakpoint if awakened

T0:    s0 ← sTestTask, at[sTestBC1];    \* task 15  
       s0 ← (s0) or (20C);

```

bwnotify: apc&apctask ← s0;
         return;

         s0 ← (s0) - (10000C), at[add[sTestBase,31]]; *notify returns to here
         LU ← ldf[s0,0,4];
         dblgoto[bwnotify,TtoR,alu#0];

```

```

bwnot:  s1 ← sTestBC.call[xret], at[add[sTestBase,20]];
badwake: usectask;
         T ← apc&apctask;
         breakpoint, goto[go];
         return;

```

```

xret:   s1 ← (s1) or (31C);
         apc&apctask ← s1;
         return;

```

\* guaranteed to be in task 0 now

```

*****
*** MAIN routine:

```

\* try the T pass around path

```

TtoR:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0001, Mcount to 0001

         s0 ← T + 0C;
         T ← 377C;
         s0 ← T;
         compare;

```

\* try the R pass around path

```

RtoT:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0002, Mcount to 0002

         s0 ← T + 0C;
         s0 ← 377C;
         T ← s0;
         compare;

```

\* test constants from micro instruction

```

const: incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0003, Mcount to 0003

         s0 ← (ZERO);      T ← 0C;      compare;
         s0 ← (ZERO) + 1;  T ← 1C;      compare;
         s0 ← (s0) + (T);  T ← 2C;      compare;
         s0 ← (s0) + (T);  T ← 4C;      compare;
         s0 ← (s0) + (T);  T ← 10C;     compare;
         s0 ← (s0) + (T);  T ← 20C;     compare;
         s0 ← (s0) + (T);  T ← 40C;     compare;
         s0 ← (s0) + (T);  T ← 100C;    compare;
         s0 ← (s0) + (T);  T ← 200C;    compare;
         s0 ← (s0) + (T);  T ← 400C;    compare;
         s0 ← (s0) + (T);  T ← 1000C;   compare;
         s0 ← (s0) + (T);  T ← 2000C;   compare;
         s0 ← (s0) + (T);  T ← 4000C;   compare;
         s0 ← (s0) + (T);  T ← 10000C;  compare;
         s0 ← (s0) + (T);  T ← 20000C;  compare;
         s0 ← (s0) + (T);  T ← 40000C;  compare;
         s0 ← (s0) + (T);  T ← 100000C; compare;

```

\* test the basic ALU operations

```

alutst: incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0004, Mcount to 0004

         s0 ← ZERO;      * test cycler/masker function ZERO
         T ← 0C;
         compare;

```

```

         s0 ← 0C;      * test (ZERO) - 1
         s1 ← T + ones; * set s1 to all ones for subsequent tests
         T ← (ZERO) + (T) + 1;
         compare;

```

```

alu0:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0005, Mcount to 0005

```

```

T ← ones;          * 1 from T
s0 ← T;
compare;

T ← 0C;            * 0 from T
s0 ← T;
compare;

alu1:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0006, Mcount to 0006

s0 ← 0C;
T ← s0;           * 0 from R
compare;

s0 ← ones;       * 1 from R
T ← s0;
compare;

alu2:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0007, Mcount to 0007

T ← 0C;          T ← (ZERO) and (T);   * 0 and 0
s0 ← 0C;          compare;

T ← ones;        T ← (ZERO) and (T);   * 0 and 1
s0 ← 0C;          compare;

T ← 0C;          T ← (s1) and (T);     * 1 and 0
s0 ← 0C;          compare;

T ← ones;        T ← (s1) and (T);     * 1 and 1
s0 ← ones;       compare;

alu3:  :incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0008, Mcount to 0010

T ← 0C;          T ← (ZERO) or (T);    * 0 or 0
s0 ← 0C;          compare;

T ← ones;        T ← (ZERO) or (T);    * 0 or 1
s0 ← ones;       compare;

T ← 0C;          T ← (s1) or (T);     * 1 or 0
s0 ← ones;       compare;

T ← ones;        T ← (s1) or (T);     * 1 or 1
s0 ← ones;       compare;

alu4:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0009, Mcount to 0011

T ← 0C;          T ← (ZERO) xor (T);   * 0 xor 0
s0 ← 0C;          compare;

T ← ones;        T ← (ZERO) xor (T);   * 0 xor 1
s0 ← ones;       compare;

T ← 0C;          T ← (s1) xor (T);    * 1 xor 0
s0 ← ones;       compare;

T ← ones;        T ← (s1) xor (T);    * 1 xor 1
s0 ← 0C;         compare;

alu5:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0010, Mcount to 0012

T ← 0C;          T ← (ZERO) andnot (T); * 0 andnot 0
s0 ← 0C;          compare;

T ← ones;        T ← (ZERO) andnot (T); * 0 andnot 1
s0 ← 0C;          compare;

T ← 0C;          T ← (s1) andnot (T);  * 1 andnot 0
s0 ← ones;       compare;

T ← ones;        T ← (s1) andnot (T);  * 1 andnot 1
s0 ← 0C;         compare;

alu6:  incMpanel, Mcount ← (Mcount) + 1; * increment Mpanel to 0011, Mcount to 0013

```



```

T ← 0C;          T ← (ZERO) ornot (T); * 0 ornot 0
s0 ← ones;      compare;

T ← ones;        T ← (ZERO) ornot (T); * 0 ornot 1
s0 ← 0C;        compare;

T ← 0C;          T ← (s1) ornot (T); * 1 ornot 0
s0 ← ones;      compare;

T ← ones;        T ← (s1) ornot (T); * 1 ornot 1
s0 ← ones;      compare;

```

alu7: incMpanel, Mcount ← (Mcount) + 1; \* increment Mpanel to 0012, Mcount to 0014

```

T ← 0C;          T ← (ZERO) xnor (T); * 0 xnor 0
s0 ← ones;      compare;

T ← ones;        T ← (ZERO) xnor (T); * 0 xnor 1
s0 ← 0C;        compare;

T ← 0C;          T ← (s1) xnor (T); * 1 xnor 0
s0 ← 0C;        compare;

T ← ones;        T ← (s1) xnor (T); * 1 xnor 1
s0 ← ones;      compare;

```

```

LOAD PAGE[Rpage];
gotop[stktst];

```

\*\*\*\*\*

ON PAGE[Rpage];

\* try the stack pointer

stktst: incMpanel, Mcount ← (Mcount) + 1; \* increment Mpanel to 0013, Mcount to 0015

```

T ← s0 ← Rmax;
stkp ← s0;
T ← stkp;
s1 ← 377C;
T ← (s1) and (T);
T ← (s1) xor (T); * stkp comes back complemented
compare;

```

\* exercise the entire R store access through the stkp

```

flone: T ← 0C;      call[TfillR]; * increment Mpanel to 0014, Mcount to 0016
        T ← 0C;      call[TchkR];

        T ← 1C;      call[TfillR]; * increment Mpanel to 0015, Mcount to 0017
        T ← 1C;      call[TchkR];

        T ← 2C;      call[TfillR]; * increment Mpanel to 0016, Mcount to 0020
        T ← 2C;      call[TchkR];

        T ← 4C;      call[TfillR]; * increment Mpanel to 0017, Mcount to 0021
        T ← 4C;      call[TchkR];

        T ← 10C;     call[TfillR]; * increment Mpanel to 0018, Mcount to 0022
        T ← 10C;     call[TchkR];

        T ← 20C;     call[TfillR]; * increment Mpanel to 0019, Mcount to 0023
        T ← 20C;     call[TchkR];

        T ← 40C;     call[TfillR]; * increment Mpanel to 0020, Mcount to 0024
        T ← 40C;     call[TchkR];

        T ← 100C;    call[TfillR]; * increment Mpanel to 0021, Mcount to 0025
        T ← 100C;    call[TchkR];

        T ← 200C;    call[TfillR]; * increment Mpanel to 0022, Mcount to 0026
        T ← 200C;    call[TchkR];

        T ← 400C;    call[TfillR]; * increment Mpanel to 0023, Mcount to 0027

```

```

T ← 400C;      call[TchkR];
T ← 1000C;     call[TfillR]; * increment Mpanel to 0024, Mcount to 0030
T ← 1000C;     call[TchkR];
T ← 2000C;     call[TfillR]; * increment Mpanel to 0025, Mcount to 0031
T ← 2000C;     call[TchkR];
T ← 4000C;     call[TfillR]; * increment Mpanel to 0026, Mcount to 0032
T ← 4000C;     call[TchkR];
T ← 10000C;    call[TfillR]; * increment Mpanel to 0027, Mcount to 0033
T ← 10000C;    call[TchkR];
T ← 20000C;    call[TfillR]; * increment Mpanel to 0028, Mcount to 0034
T ← 20000C;    call[TchkR];
T ← 40000C;    call[TfillR]; * increment Mpanel to 0029, Mcount to 0035
T ← 40000C;    call[TchkR];
T ← 100000C;   call[TfillR]; * increment Mpanel to 0030, Mcount to 0036
T ← 100000C;   call[TchkR];

flzero: T ← 0C;      T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0031, Mcount to 0037
T ← 0C;      T ← (ZERO) ornot (T); call[TchkR];
T ← 1C;      T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0032, Mcount to 0040
T ← 1C;      T ← (ZERO) ornot (T); call[TchkR];
T ← 2C;      T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0033, Mcount to 0041
T ← 2C;      T ← (ZERO) ornot (T); call[TchkR];
T ← 4C;      T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0034, Mcount to 0042
T ← 4C;      T ← (ZERO) ornot (T); call[TchkR];
T ← 10C;     T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0035, Mcount to 0043
T ← 10C;     T ← (ZERO) ornot (T); call[TchkR];
T ← 20C;     T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0036, Mcount to 0044
T ← 20C;     T ← (ZERO) ornot (T); call[TchkR];
T ← 40C;     T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0037, Mcount to 0045
T ← 40C;     T ← (ZERO) ornot (T); call[TchkR];
T ← 100C;    T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0038, Mcount to 0046
T ← 100C;    T ← (ZERO) ornot (T); call[TchkR];
T ← 200C;    T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0039, Mcount to 0047
T ← 200C;    T ← (ZERO) ornot (T); call[TchkR];
T ← 400C;    T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0040, Mcount to 0050
T ← 400C;    T ← (ZERO) ornot (T); call[TchkR];
T ← 1000C;   T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0041, Mcount to 0051
T ← 1000C;   T ← (ZERO) ornot (T); call[TchkR];
T ← 2000C;   T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0042, Mcount to 0052
T ← 2000C;   T ← (ZERO) ornot (T); call[TchkR];
T ← 4000C;   T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0043, Mcount to 0053
T ← 4000C;   T ← (ZERO) ornot (T); call[TchkR];
T ← 10000C;  T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0044, Mcount to 0054
T ← 10000C;  T ← (ZERO) ornot (T); call[TchkR];
T ← 20000C;  T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0045, Mcount to 0055
T ← 20000C;  T ← (ZERO) ornot (T); call[TchkR];
T ← 40000C;  T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0046, Mcount to 0056
T ← 40000C;  T ← (ZERO) ornot (T); call[TchkR];
T ← 100000C; T ← (ZERO) ornot (T); call[TfillR]; * incr. Mpanel to 0047, Mcount to 0057
T ← 100000C; T ← (ZERO) ornot (T); call[TchkR];

```

\* write address in contents all through R

AfillR: incMpanel, Mcount ← (Mcount) + 1; \* increment Mpanel to 0048, Mcount to 0060

```

s0 ← T + Rmax;
AfillR1: stkp ← s0;
stack ← T;
s0 ← T + (s0)-1;
LU ← (s0) - (Rmin);
goto[AfillR1, ALU>=0];

```

\* check that address is in contents all through R

```

AchKR: s0 ← T + Rmax;
AchKR1: stkp ← s0;
LU ← (stack) # T;
goto[.+2, ALU=0];
AchKRer: breakpoint; * contents do not match address
s0 ← T + (s0)-1;
LU ← (s0) - (Rmin);
goto[AchKR1, ALU>=0];

LOAD PAGE[pushpage];
gotop[push1];

```

\*\*\*\*\* start SUBROUTINE: TfillR \*\*\*\*\*

\* write pattern in T from Rmax through Rmin in R store, increment Mpanel and Mcount  
 \* s0 holds address  
 \* s1 holds pattern

```

TfillR: incMpanel, Mcount ← (Mcount) + 1;
s1 ← T;
LU ← (s1) # T;
goto[.+2, ALU=0];
TfillRer: breakpoint, goto[.-3]; * pattern and T disagree
s0 ← Rmax;
TfillR1: stkp ← s0;
stack ← T;
s0 ← (s0) - 1;
LU ← (s0) - (Rmin);
goto[TfillR1, ALU>=0];
T ← s1;
s0 ← T, return;

```

\*\*\*\*\* start SUBROUTINE: TchKR \*\*\*\*\*

\* takes pattern in T, break points if discrepancy is found

```

TchKR: LU ← (s1) # T; * special check on pattern
goto[.+2, ALU=0];
TchKRer0: breakpoint; * pattern and T disagree
LU ← (s0) # T; * TfillR put the pattern in S0 as part of a RETURN (checks
* that R writes are done properly across TASK switches)
goto[.+2, ALU=0];
TchKRer1: breakpoint;
s0 ← Rmax;
TchKR1: stkp ← s0; * stkp loaded from ALUA, sigh
LU ← (stack) # T;
goto[.+2, ALU=0];
TchKRer2: breakpoint; * pattern does not match R
s0 ← (s0) - 1;
LU ← (s0) - (Rmin);
goto[TchKR1, ALU>=0];
return;

```

\*\*\*\*\*  
 ON PAGE[pushpage];

\* test stack push and pop operations

```

push1: T ← ones;
LOAD PAGE[Rpage];

```

```

callp[TfillR];          * initialize area to -1, increment Mpanel to 0049 and Mcount to 0061
s0 ← 37C; stkp ← s0;    * use regs. 20-37 for this test (init. to 37 for increment to 20)
s1 ← T + 20C;          * starting data (to be stored in address 20)
rt ← 17C;              * iterations count for 20 iterations

push1l: (stack&+1) ← T;  * store data in incremented stack (will increment mod. 20)
s1 ← T + (s1) + 1;     * increment data to be stored
rt ← (rt) - 1;        * decrement iterations count
goto[push1l, ALU>=0];

LOAD PAGE[poppage];
callp[poptst];         * check proper contents

pushm1: T ← ones;
LOAD PAGE[Rpage];
callp[TfillR];        * initialize area to -1, increment Mpanel to 0050 and Mcount to 0062
s0 ← 20C; stkp ← s0;   * use regs. 37-20 for this test (init. to 20 for decrement to 37)
s1 ← T + 37C;         * starting data (to be stored in address 37)
rt ← 17C;             * iterations count for 20 iterations

pushm1l: (stack&-1) ← T; * store data in decremented stack (will decrement mod. 20)
s1 ← T + (s1) - 1;    * decrement data to be stored
rt ← (rt) - 1;       * decrement iterations count
goto[pushm1l, ALU>=0];

LOAD PAGE[poppage];
callp[poptst];       * check proper contents

push2: T ← ones;
LOAD PAGE[Rpage];
callp[TfillR];        * initialize area to -1, increment Mpanel to 0051 and Mcount to 0063
s0 ← 36C; stkp ← s0;   * use even regs. 20-36 for this test (init. to 36 for increment to 20)
s1 ← T + 20C;         * starting data (to be stored in address 20)
rt ← 7C;              * iterations count for 10 iterations

push2le: (stack&+2) ← T; * store data in incremented stack (will increment mod. 20)
s1 ← T + (s1) + (2C); * increment data to be stored
rt ← (rt) - 1;       * decrement iterations count
goto[push2le, ALU>=0];

s0 ← 37C; stkp ← s0;   * use odd regs. 21-37 for this test (init. to 37 for increment to 21)
s1 ← T + 21C;         * starting data (to be stored in address 21)
rt ← 7C;             * iterations count for 10 iterations

push2lo: (stack&+2) ← T; * store data in incremented stack (will increment mod. 20)
s1 ← T + (s1) + (2C); * increment data to be stored
rt ← (rt) - 1;       * decrement iterations count
goto[push2lo, ALU>=0];

LOAD PAGE[poppage];
callp[poptst];       * check proper contents

pushm2: T ← ones;
LOAD PAGE[Rpage];
callp[TfillR];        * initialize area to -1, increment Mpanel to 0052 and Mcount to 0064
s0 ← 20C; stkp ← s0;   * use even regs. 36-20 for this test (init. to 20 for decrement to 36)
s1 ← T + 36C;         * starting data (to be stored in address 36)
rt ← 7C;             * iterations count for 10 iterations

pushm2le: (stack&-2) ← T; * store data in decremented stack (will decrement mod. 20)
s1 ← T + (s1) - (2C); * decrement data to be stored
rt ← (rt) - 1;       * decrement iterations count
goto[pushm2le, ALU>=0];

s0 ← 21C; stkp ← s0;   * use odd regs. 37-21 for this test (init. to 21 for decrement to 37)
s1 ← T + 37C;         * starting data (to be stored in address 37)

```

```

    rt ← 7C;          * iterations count for 10 iterations

pushm21o: (stack&-2) ← T;  * store data in decremented stack (will decrement mod. 20)
s1 ← T ← (s1) - (2C);  * decrement data to be stored
rt ← (rt) - 1;        * decrement iterations count
goto[pushm21o, ALU>=0];

LOAD PAGE[poppage];
callp[poptst];      * check proper contents

push3: T ← ones;
LOAD PAGE[Rpage];
callp[TfillR];      * initialize area to -1, increment Mpanel to 0053 and Mcount to 0065

s0 ← 35C; stkp ← s0;  * use regs. 20-37 for this test (init. to 35 for increment to 20)
s1 ← T ← 20C;        * starting data (to be stored in address 20)
rt ← 17C;           * iterations count for 20 iterations

push31: (stack&+3) ← T;  * store data in incremented stack (will increment mod. 20)

s1 ← (s1) + (3C);
s1 ← (s1) and (17C);
s1 ← T ← (s1) + (20C); * data incremented by 3's (mod. 20)

rt ← (rt) - 1;      * decrement iterations count
goto[push31, ALU>=0];

LOAD PAGE[poppage];
callp[poptst];      * check proper contents

pushm3: T ← ones;
LOAD PAGE[Rpage];
callp[TfillR];      * initialize area to -1, increment Mpanel to 0054 and Mcount to 0066

s0 ← 20C; stkp ← s0;  * use regs. 37-20 for this test (init. to 20 for decrement to 35)
s1 ← T ← 36C;        * starting data (to be stored in address 35)
rt ← 17C;           * iterations count for 20 iterations

pushm31: (stack&-3) ← T;  * store data in decremented stack (will decrement mod. 20)

s1 ← (s1) - (3C);
s1 ← (s1) and (17C);
s1 ← T ← (s1) + (20C); * data decremented by 3's (mod. 20)

rt ← (rt) - 1;
goto[pushm31, ALU>=0];

LOAD PAGE[poppage];
callp[poptst];      * check proper contents

PassCount ← (PassCount) + 1; * increment pass count
T ← MaxPass;
lu ← (PassCount) - (T);
goto[.+3,ALU>=0];    * finished all passes?
LOAD PAGE[alupage];
gotop[go];

Passed-EDALU-Test: PassCount ← 0C, breakpoint;

LOAD PAGE[alupage];
gotop[go];

*****

ON PAGE[poppage];

***** start SUBROUTINE: poptst *****

poptst: usectask;
T ← apc&apctask;    * save the return address
oldapc ← T;

```

```

pop1:  s0 ← 20C;
        stkp ← s0;
        T ← (stack&+1);      * retrieves and then increments
        compare;

```

```

s0 ← 21C; T ← (stack&+1); compare;
s0 ← 22C; T ← (stack&+1); compare;
s0 ← 23C; T ← (stack&+1); compare;
s0 ← 24C; T ← (stack&+1); compare;
s0 ← 25C; T ← (stack&+1); compare;
s0 ← 26C; T ← (stack&+1); compare;
s0 ← 27C; T ← (stack&+1); compare;
s0 ← 30C; T ← (stack&+1); compare;
s0 ← 31C; T ← (stack&+1); compare;
s0 ← 32C; T ← (stack&+1); compare;
s0 ← 33C; T ← (stack&+1); compare;
s0 ← 34C; T ← (stack&+1); compare;
s0 ← 35C; T ← (stack&+1); compare;
s0 ← 36C; T ← (stack&+1); compare;
s0 ← 37C; T ← (stack&+1); compare;
s0 ← 20C; T ← (stack&+1); compare;

```

```

popm1: s0 ← 20C;
        stkp ← s0;
        T ← (stack&-1);
        compare;

```

```

s0 ← 37C; T ← (stack&-1); compare;
s0 ← 36C; T ← (stack&-1); compare;
s0 ← 35C; T ← (stack&-1); compare;
s0 ← 34C; T ← (stack&-1); compare;
s0 ← 33C; T ← (stack&-1); compare;
s0 ← 32C; T ← (stack&-1); compare;
s0 ← 31C; T ← (stack&-1); compare;
s0 ← 30C; T ← (stack&-1); compare;
s0 ← 27C; T ← (stack&-1); compare;
s0 ← 26C; T ← (stack&-1); compare;
s0 ← 25C; T ← (stack&-1); compare;
s0 ← 24C; T ← (stack&-1); compare;
s0 ← 23C; T ← (stack&-1); compare;
s0 ← 22C; T ← (stack&-1); compare;
s0 ← 21C; T ← (stack&-1); compare;
s0 ← 20C; T ← (stack&-1); compare;

```

```

pop2e: s0 ← 20C;
        stkp ← s0;
        T ← (stack&+2);
        compare;

```

```

s0 ← 22C; T ← (stack&+2); compare;
s0 ← 24C; T ← (stack&+2); compare;
s0 ← 26C; T ← (stack&+2); compare;
s0 ← 30C; T ← (stack&+2); compare;
s0 ← 32C; T ← (stack&+2); compare;
s0 ← 34C; T ← (stack&+2); compare;
s0 ← 36C; T ← (stack&+2); compare;
s0 ← 20C; T ← (stack&+2); compare;

```

```

pop2o: s0 ← 21C;
        stkp ← s0;
        T ← (stack&+2);
        compare;

```

```

s0 ← 23C; T ← (stack&+2); compare;
s0 ← 25C; T ← (stack&+2); compare;
s0 ← 27C; T ← (stack&+2); compare;
s0 ← 31C; T ← (stack&+2); compare;
s0 ← 33C; T ← (stack&+2); compare;
s0 ← 35C; T ← (stack&+2); compare;
s0 ← 37C; T ← (stack&+2); compare;
s0 ← 21C; T ← (stack&+2); compare;

```

```

popm2e: s0 ← 20C;
         stkp ← s0;
         T ← (stack&-2);

```

compare;

```
s0 ← 36C; T ← (stack&-2); compare;
s0 ← 34C; T ← (stack&-2); compare;
s0 ← 32C; T ← (stack&-2); compare;
s0 ← 30C; T ← (stack&-2); compare;
s0 ← 26C; T ← (stack&-2); compare;
s0 ← 24C; T ← (stack&-2); compare;
s0 ← 22C; T ← (stack&-2); compare;
s0 ← 20C; T ← (stack&-2); compare;
```

```
popm2o: s0 ← 21C;
        stkp ← s0;
        T ← (stack&-2);
        compare;
```

```
s0 ← 37C; T ← (stack&-2); compare;
s0 ← 35C; T ← (stack&-2); compare;
s0 ← 33C; T ← (stack&-2); compare;
s0 ← 31C; T ← (stack&-2); compare;
s0 ← 27C; T ← (stack&-2); compare;
s0 ← 25C; T ← (stack&-2); compare;
s0 ← 23C; T ← (stack&-2); compare;
s0 ← 21C; T ← (stack&-2); compare;
```

```
LOAD PAGE[popxpage];
gotop[pop3];
```

\*\*\*\*\*

```
ON PAGE[popxpage];
```

```
pop3:  s0 ← 20C;
        stkp ← s0;
        T ← (stack&+3);
        compare;
```

```
s0 ← 23C; T ← (stack&+3); compare;
s0 ← 26C; T ← (stack&+3); compare;
s0 ← 31C; T ← (stack&+3); compare;
s0 ← 34C; T ← (stack&+3); compare;
s0 ← 37C; T ← (stack&+3); compare;
s0 ← 22C; T ← (stack&+3); compare;
s0 ← 25C; T ← (stack&+3); compare;
s0 ← 30C; T ← (stack&+3); compare;
s0 ← 33C; T ← (stack&+3); compare;
s0 ← 36C; T ← (stack&+3); compare;
s0 ← 21C; T ← (stack&+3); compare;
s0 ← 24C; T ← (stack&+3); compare;
s0 ← 27C; T ← (stack&+3); compare;
s0 ← 32C; T ← (stack&+3); compare;
s0 ← 35C; T ← (stack&+3); compare;
s0 ← 20C; T ← (stack&+3); compare;
```

```
popm3: s0 ← 20C;
        stkp ← s0;
        T ← (stack&-3);
        compare;
```

```
s0 ← 35C; T ← (stack&-3); compare;
s0 ← 32C; T ← (stack&-3); compare;
s0 ← 27C; T ← (stack&-3); compare;
s0 ← 24C; T ← (stack&-3); compare;
s0 ← 21C; T ← (stack&-3); compare;
s0 ← 36C; T ← (stack&-3); compare;
s0 ← 33C; T ← (stack&-3); compare;
s0 ← 30C; T ← (stack&-3); compare;
s0 ← 25C; T ← (stack&-3); compare;
s0 ← 22C; T ← (stack&-3); compare;
s0 ← 37C; T ← (stack&-3); compare;
s0 ← 34C; T ← (stack&-3); compare;
s0 ← 31C; T ← (stack&-3); compare;
s0 ← 26C; T ← (stack&-3); compare;
s0 ← 23C; T ← (stack&-3); compare;
s0 ← 20C; T ← (stack&-3); compare;
```

```
        apc&apctask + oldapc;
        return;

***** end SUBROUTINE: poptst *****

*****

ONPAGE[0];

***** start SUBROUTINE: compare0 *****

compare0: LU ← (s0) # (T);
          goto[success, ALU=0];
fail:    breakpoint;
success: return;

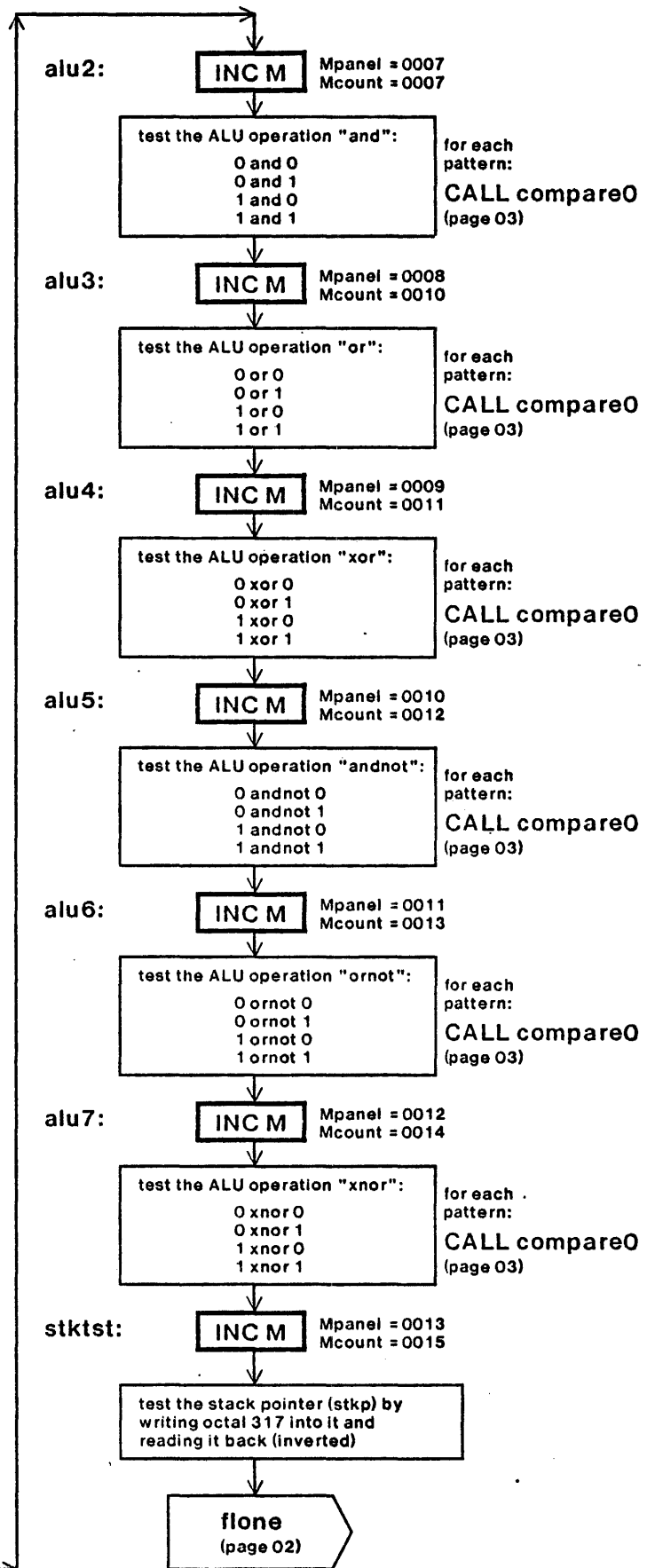
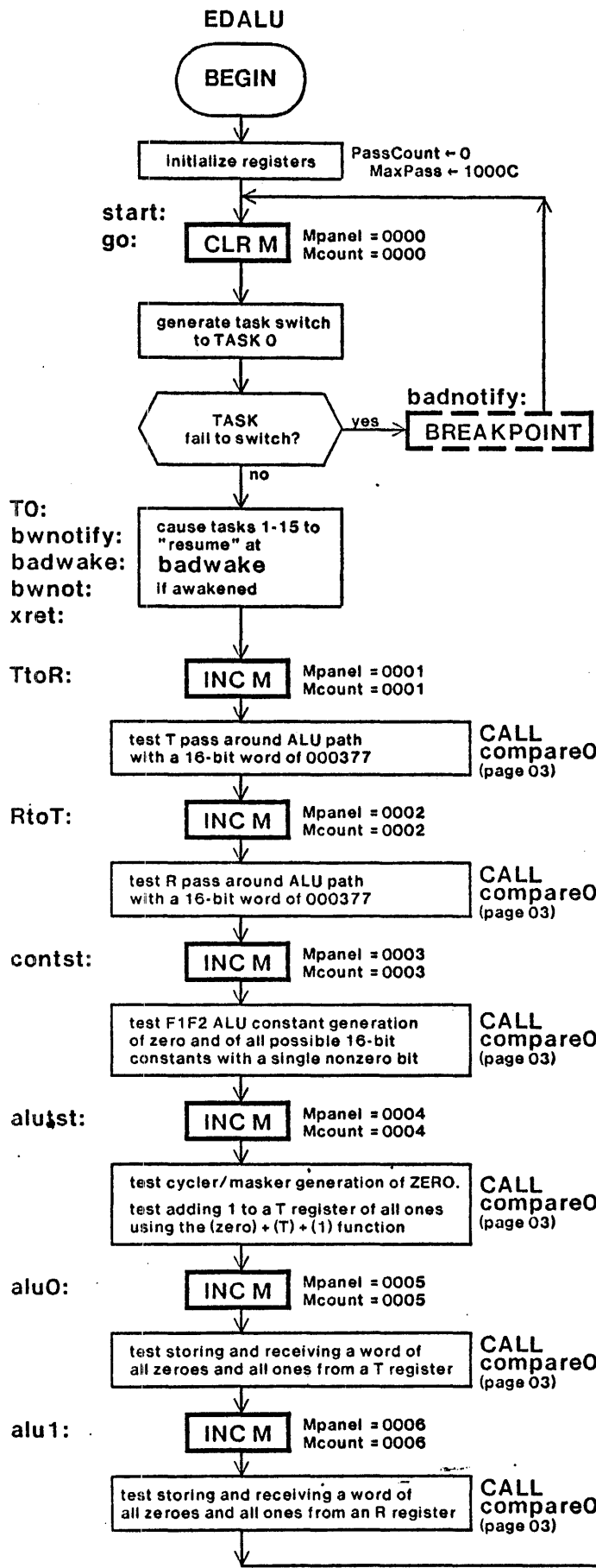
***** end SUBROUTINE: compare0 *****

*** similarly: ***

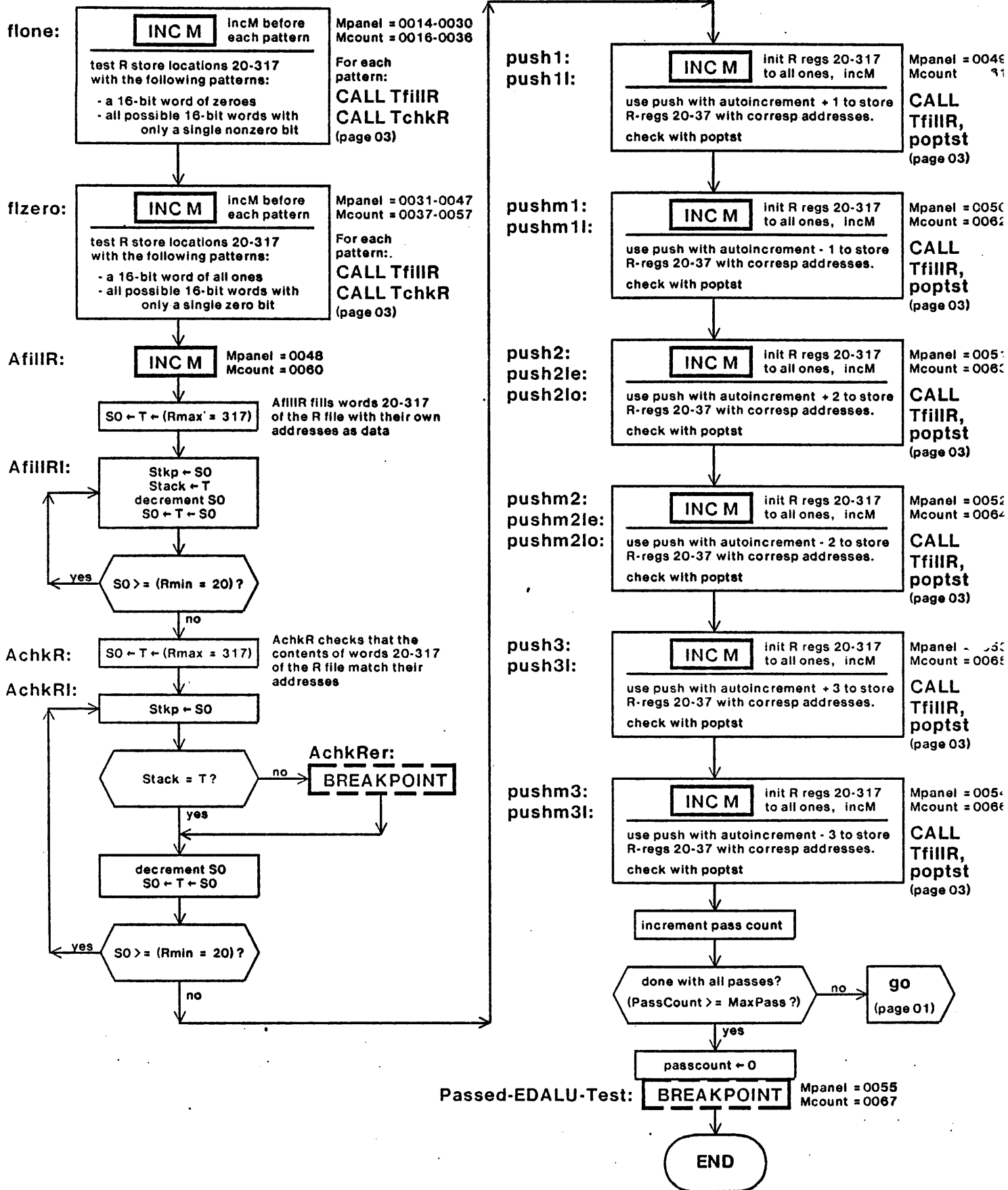
ONPAGE[1];
compare1: LOADPAGE[0];
          gotop[compare0];
ONPAGE[2];
compare2: LOADPAGE[0];
          gotop[compare0];
ONPAGE[3];
compare3: LOADPAGE[0];
          gotop[compare0];
ONPAGE[4];
compare4: LOADPAGE[0];
          gotop[compare0];
ONPAGE[5];
compare5: LOADPAGE[0];
          gotop[compare0];

        end;
```





XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDALU	EDALU-01.sil	Thomson/Kellman	1	1/11/80	01

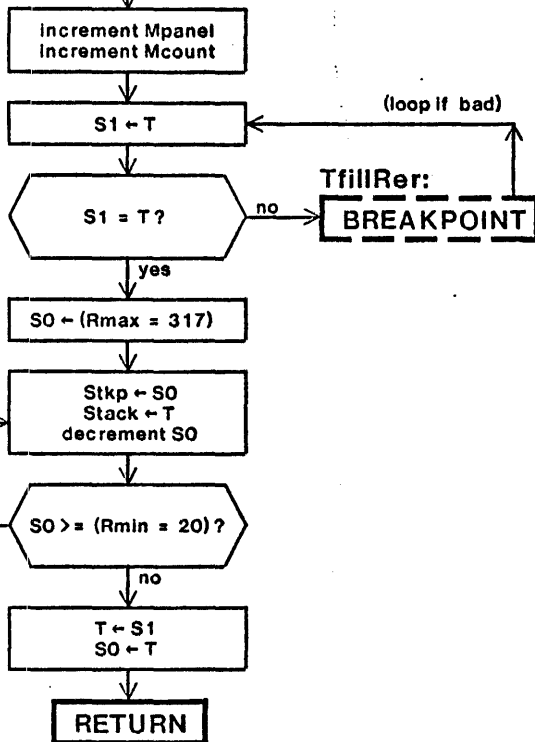


**SUBROUTINE**



TfillR fills words 20-317 of the R file with a constant provided by T. Increments Mpanel and Mcount

TfillRI:

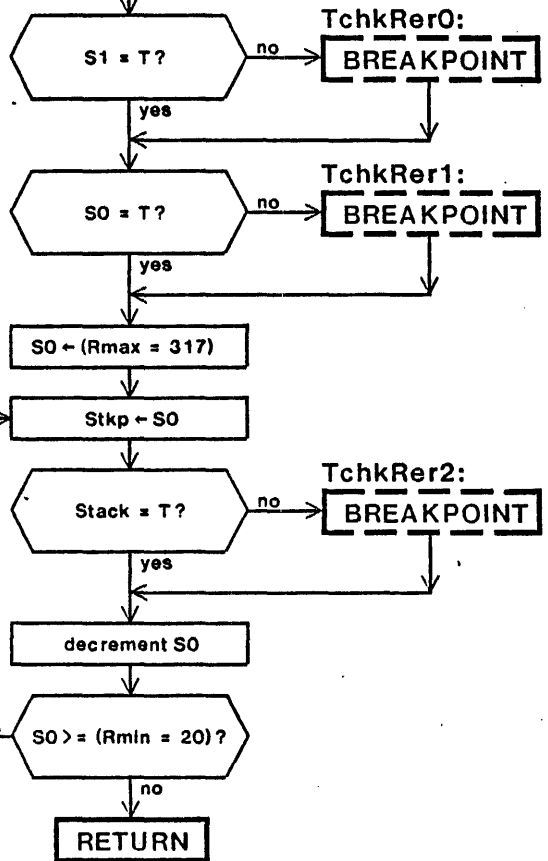


**SUBROUTINE**



TchkR checks that the contents of words 20-317 of the R file are all equal to the constant in T

TchkRI:



**SUBROUTINE**



poptst checks that the contents of words 20-37 of the R file match their addresses and that the stack pointer counts properly

NOTE: compare0 is reached by Midas routing of compare

**SUBROUTINE**

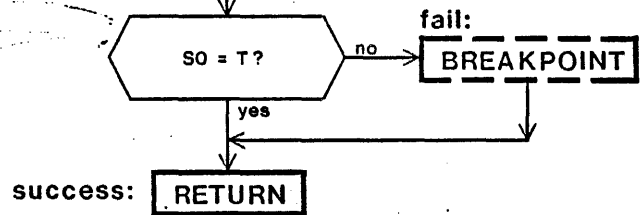


compare0 checks that the contents of SO (R file) match the contents of T

- pop1: pop stack with autoincrement of +1  
addr: 20-37, 20
- popm1: pop stack with autoincrement of -1  
addr: 20, 37-20
- pop2e: pop stack with autoincrement of +2  
addr: 20-36, 20 (even only)
- pop2o: pop stack with autoincrement of +2  
addr: 21-37, 21 (odd only)
- popm2e: pop stack with autoincrement of -2  
addr: 20, 36-20 (even only)
- popm2o: pop stack with autoincrement of -2  
addr: 21, 37-21 (odd only)
- pop3: pop stack with autoincrement of +3  
addr: 20-37, 20 (interlaced)
- popm3: popm3: pop stack with autoincrement of -3  
addr: 20, 37-20 (interlaced)

for each of the 104 stack pop operations

CALL compare0 (page 03)



PARITY	0	REVISION	1	COMM-ER0	0
CYCLECONTROL	63	RUN-TIME	11	COMM-ER1	0
PCXREG	2	PASSCOUNT	0	COMM-ER2	0
PCFREG	2	MAXPASS	1000	BOOT-ERR	0
DBREG	77			*BOOTREASON	40
SBREG	77			MEMSYNDROME	164247
MNBR	161007				
SSTKP	0				
STKP	0	MCOUNT	0		
*ALURESULT	3				
SALUF	0				
T 20	7000				
AATOVA	0				
TPC 20	7777				
CALLER	ILC@+6217				
*PAGE	5	T 0	7000		
*APC	7011	S0	176405		
*APCTASK	16	S1	0		
*CIA	60+1				
CTASK	0				

Loaded: EDALU

Time: 09.88

Step at 0:60, BP at 0:60+1

Exit Boot Run-Prog Read-Cmds Break UnBreak C1rAddedBPs C1rAllBPs ShowBPs Go  
 SS Continue Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

MicroD 8.6 (OS 16) of April 27, 1979  
at 10-Jan-80 12:52:59

microd.run EDALU.d1b

EDALU.d1b 1560b instructions written 10-Jan-80 12:51:04

Total of 1560b instructions

Checking for errors...

Linking...

Building allocation lists...

Assigning locations...

1560b instructions in rings involving ONPAGE or AT

Reloading binaries...

Checking assignment...

Writing .MB file...

Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
EDALU.d1b:					
0	2401	47	7130	1	GO START
1	2554	32176	101126	5	(+1)
2	2553	30020	113124	1	(+2)
3	2552	30147	21060	0	(+3)
4	2430	50	25401	0	(+4)
5 b	2555	50	25003	0	BADNOTIFY
6	@2400	30035	113043	3	TO
7	2721	30301	101006	0	(+1)
10	2403	30147	21147	1	BWNOTIFY
11	2563	50	25401	0	(+1)
12	@2431	31421	101152	1	(+2)
13	2565	30162	33151	1	(+3)
14	2564	50	24004	0	(+4)
15	@2420	30020	113344	5	BWNOT
16	2421	47	27137	1	BADWAKE
17	2557	50150	65135	15	(+1)
20 b	2556	50	25003	0	(+2)
21	2722	50	25401	0	(+3)
22	2562	30301	123143	5	XRET
23	2561	30147	21140	5	(+1)
24	2560	50	25401	0	(+2)
25	2402	33047	105040	7	TTOR
26	2720	30020	141037	3	(+1)
27	2717	17	77034	3	(+2)
30	2716	30050	125120	1	(+3)
31	2550	50	25232	3	(+4)
32	2551	33047	105161	5	RTOT
33	2570	30020	141157	1	(+1)
34	2567	30017	137154	1	(+2)
35	2566	30150	65114	1	(+3)
36	2546	50	25232	3	(+4)
37	2547	33047	105164	5	CONST
40	2572	30176	101162	1	(+1)
41	2571	20	41111	1	(+2)
42	2544	50	25232	3	(+3)
43	2545	31076	101167	1	(+4)
44	2573	0	43105	1	(+5)
45	2542	50	25232	3	(+6)
46	2543	31150	125170	1	(+7)
47	2574	0	45100	1	(+10)
50	2540	50	25232	3	(+11)
51	2541	31150	125173	1	(+12)
52	2575	0	51075	1	(+13)
53	2536	50	25232	3	(+14)
54	2537	31150	125175	1	(+15)
55	2576	0	61070	1	(+16)
56	2534	50	25232	3	(+17)
57	2535	31150	125176	1	(+20)
60	2577	1	41064	1	(+21)
61	2532	50	25232	3	(+22)

62	2533	31150	125000	2	(+23)
63	2600	2	41061	1	(+24)
64	2530	50	25232	3	(+25)
65	2531	31150	125003	2	(+26)
66	2601	4	41054	1	(+27)
67	2526	50	25232	3	(+30)
70	2527	31150	125006	2	(+31)
71	2602	10	41051	1	(+32)
72	2524	50	25232	3	(+33)
73	2525	31150	125006	2	(+34)
74	2603	20	43044	1	(+35)
75	2522	50	25232	3	(+36)
76	2523	31150	125011	2	(+37)
77	2604	20	45041	1	(+40)
100	2520	50	25232	3	(+41)
101	2521	31150	125012	2	(+42)
102	2605	20	51035	1	(+43)
103	2516	50	25232	3	(+44)
104	2517	31150	125014	2	(+46)
105	2606	20	61030	1	(+48)
106	2514	50	25232	3	(+47)
107	2515	31150	125017	2	(+50)
110	2607	21	41024	1	(+51)
111	2512	50	25232	3	(+52)
112	2513	31150	125021	2	(+53)
113	2610	22	41021	1	(+54)
114	2510	50	25232	3	(+55)
115	2511	31150	125022	2	(+56)
116	2611	24	41014	1	(+57)
117	2506	50	25232	3	(+60)
120	2507	31150	125024	2	(+61)
121	2612	30	41011	1	(+62)
122	2504	50	25232	3	(+63)
123	2505	33047	105030	6	ALUTST
124	2614	30176	101027	2	(+1)
125	2613	20	41004	1	(+2)
126	2502	50	25232	3	(+3)
127	2503	30020	101034	2	(+4)
130	2616	31376	141033	6	(+5)
131	2615	1276	41001	1	(+6)
132	2500	50	25232	3	(+7)
133	2501	33047	105041	6	ALU0
134	2620	1376	41036	2	(+1)
135	2617	30050	125174	0	(+2)
136	2476	50	25232	3	(+3)
137	2477	20	41043	2	(+4)
140	2621	30050	125171	0	(+5)
141	2474	50	25232	3	(+6)
142	2475	33047	105047	6	ALU1
143	2623	30020	101045	2	(+1)
144	2622	30150	65164	0	(+2)
145	2472	50	25232	3	(+3)
146	2473	31376	101050	2	(+4)
147	2624	30150	65161	0	(+5)
150	2470	50	25232	3	(+6)
151	2471	33047	105056	6	ALU2
152	2627	20	41054	2	(+1)
153	2626	276	41053	2	(+2)
154	2625	30020	101154	0	(+3)
155	2466	50	25232	3	(+4)
156	2467	1376	41063	2	(+5)
157	2631	276	41060	2	(+6)
160	2630	30020	101151	0	(+7)
161	2464	50	25232	3	(+10)
162	2465	20	41067	2	(+11)
163	2633	30250	65065	6	(+12)
164	2632	30020	101145	0	(+13)
165	2462	50	25232	3	(+14)
166	2463	1376	41072	2	(+15)
167	2635	30250	65071	6	(+16)
170	2634	31376	101141	0	(+17)
171	2460	50	25232	3	(+20)
172	2461	33047	105101	6	ALU3
173	2640	20	41076	2	(+1)
174	2637	376	41075	2	(+2)
175	2636	30020	101134	0	(+3)

176	2456	50	25232	3	(+4)
177	2457	1376	41104	2	(+5)
200	2642	376	41103	2	(+6)
201	2641	31376	101130	0	(+7)
202	2454	50	25232	3	(+10)
203	2455	20	41111	2	(+11)
204	2644	30350	65106	6	(+12)
205	2643	31376	101124	0	(+13)
206	2452	50	25232	3	(+14)
207	2453	1376	41115	2	(+15)
210	2646	30350	65112	6	(+16)
211	2645	31376	101121	0	(+17)
212	2450	50	25232	3	(+20)
213	2451	33047	105123	6	ALU4
214	2651	20	41121	2	(+1)
215	2650	476	41116	2	(+2)
216	2647	30020	101115	0	(+3)
217	2446	50	25232	3	(+4)
220	2447	1376	41126	2	(+6)
221	2653	476	41125	2	(+6)
222	2652	31376	101111	0	(+7)
223	2444	50	25232	3	(+10)
224	2445	20	41133	2	(+11)
225	2655	30450	65131	6	(+12)
226	2654	31376	101105	0	(+13)
227	2442	50	25232	3	(+14)
230	2443	1376	41137	2	(+15)
231	2657	30450	65134	6	(+16)
232	2656	30020	101101	0	(+17)
233	2440	50	25232	3	(+20)
234	2441	33047	105145	6	ALU5
235	2662	20	41142	2	(+1)
236	2661	576	41141	2	(+2)
237	2660	30020	101074	0	(+3)
240	2436	50	25232	3	(+4)
241	2437	1376	41151	2	(+5)
242	2664	576	41147	2	(+6)
243	2663	30020	101071	0	(+7)
244	2434	50	25232	3	(+10)
245	2435	20	41155	2	(+11)
246	2666	30550	65153	6	(+12)
247	2665	31376	101064	0	(+13)
250	2432	50	25232	3	(+14)
251	2433	1376	41161	2	(+15)
252	2670	30550	65156	6	(+16)
253	2667	30020	101055	0	(+17)
254	2426	50	25232	3	(+20)
255	2427	33047	105167	6	ALU6
256	2673	20	41165	2	(+1)
257	2672	676	41163	2	(+2)
260	2671	31376	101051	0	(+3)
261	2424	50	25232	3	(+4)
262	2425	1376	41173	2	(+5)
263	2675	676	41171	2	(+6)
264	2674	30020	101044	0	(+7)
265	2422	50	25232	3	(+10)
266	2423	20	41177	2	(+11)
267	2677	30650	65174	6	(+12)
270	2676	31376	101034	0	(+13)
271	2416	50	25232	3	(+14)
272	2417	1376	41002	3	(+15)
273	2701	30650	65000	7	(+16)
274	2700	31376	101031	0	(+17)
275	2414	50	25232	3	(+20)
276	2415	33047	105010	7	ALU7
277	2704	20	41006	3	(+1)
300	2703	776	41004	3	(+2)
301	2702	31376	101025	0	(+3)
302	2412	50	25232	3	(+4)
303	2413	1376	41015	3	(+5)
304	2706	776	41013	3	(+6)
305	2705	30020	101021	0	(+7)
306	2410	50	25232	3	(+10)
307	2411	20	41021	3	(+11)
310	2710	30750	65016	7	(+12)
311	2707	30020	101011	0	(+13)

312	2404	50	25232	3	(+14)
313	2406	1376	41028	3	(+15)
314	2713	30750	65025	7	(+16)
315	2712	31376	101015	0	(+17)
316	2406	50	25232	3	(+20)
317	2407	45	3022	3	(+21)
320	2711	50	25070	2	(+22)
321	634	33047	106100	7	STKTST
322	740	30014	177077	3	(+1)
323	737	30150	3074	3	(+2)
324	736	70166	47073	17	(+3)
325	735	30017	137071	7	(+4)
326	734	30250	65067	7	(+5)
327	733	30450	65065	6	(+6)
330	632	50	25274	2	(+7)
331	633	20	41045	1	FLONE
332	522	50	25361	2	(+1)
333	523	20	41061	2	(+2)
334	630	50	25264	3	(+3)
335	631	0	43040	1	(+4)
336	520	50	25361	2	(+5)
337	521	0	43054	2	(+6)
340	626	50	25264	3	(+7)
341	627	0	45034	1	(+10)
342	516	50	25361	2	(+11)
343	517	0	45051	2	(+12)
344	624	50	25264	3	(+13)
345	625	0	51031	1	(+14)
346	514	50	25361	2	(+15)
347	515	0	51045	2	(+16)
350	622	50	25264	3	(+17)
351	623	0	61025	1	(+20)
352	512	50	25361	2	(+21)
353	513	0	61040	2	(+22)
354	620	50	25264	3	(+23)
355	621	1	41020	1	(+24)
356	510	50	25361	2	(+25)
357	511	1	41034	2	(+26)
360	616	50	25264	3	(+27)
361	617	2	41015	1	(+30)
362	506	50	25361	2	(+31)
363	507	2	41031	2	(+32)
364	614	50	25264	3	(+33)
365	615	4	41010	1	(+34)
366	504	50	25361	2	(+35)
367	505	4	41025	2	(+36)
370	612	50	25264	3	(+37)
371	613	10	41004	1	(+40)
372	502	50	25361	2	(+41)
373	503	10	41020	2	(+42)
374	610	50	25264	3	(+43)
375	611	20	43000	1	(+44)
376	500	50	25361	2	(+45)
377	501	20	43014	2	(+46)
400	606	50	25264	3	(+47)
401	607	20	45174	0	(+50)
402	476	50	25361	2	(+51)
403	477	20	45011	2	(+52)
404	604	50	25264	3	(+53)
405	605	20	51171	0	(+54)
406	474	50	25361	2	(+55)
407	475	20	51005	2	(+56)
410	602	50	25264	3	(+57)
411	603	20	61165	0	(+60)
412	472	50	25361	2	(+61)
413	473	20	61000	2	(+62)
414	600	50	25264	3	(+63)
415	601	21	41160	0	(+64)
416	470	50	25361	2	(+65)
417	471	21	41175	1	(+66)
420	576	50	25264	3	(+67)
421	577	22	41155	0	(+70)
422	466	50	25361	2	(+71)
423	467	22	41170	1	(+72)
424	574	50	25264	3	(+73)
425	575	24	41150	0	(+74)



426	464	50	25361	2	(+75)
427	465	24	41164	1	(+76)
430	572	50	25264	3	(+77)
431	573	30	41144	0	(+100)
432	462	50	25361	2	(+101)
433	463	30	41161	1	(+102)
434	570	50	25264	3	(+103)
436	571	20	41076	2	FLZERO
436	637	676	41140	0	(+1)
437	460	50	25361	2	(+2)
440	461	20	41183	2	(+3)
441	671	676	41155	1	(+4)
442	566	50	25264	3	(+5)
443	567	0	43100	2	(+6)
444	640	676	41134	0	(+7)
445	456	50	25361	2	(+10)
446	457	0	43165	2	(+11)
447	672	676	41150	1	(+12)
450	564	50	25264	3	(+13)
451	565	0	45103	2	(+14)
452	641	676	41131	0	(+15)
453	454	50	25361	2	(+16)
454	455	0	45166	2	(+17)
455	673	676	41144	1	(+20)
456	562	50	25264	3	(+21)
457	563	0	51105	2	(+22)
460	642	676	41125	0	(+23)
461	452	50	25361	2	(+24)
462	453	0	51171	2	(+25)
463	674	676	41141	1	(+26)
464	560	50	25264	3	(+27)
465	561	0	61106	2	(+30)
466	643	676	41120	0	(+31)
467	450	50	25361	2	(+32)
470	451	0	61172	2	(+33)
471	675	676	41135	1	(+34)
472	556	50	25264	3	(+35)
473	557	1	41111	2	(+36)
474	644	676	41115	0	(+37)
475	446	50	25361	2	(+40)
476	447	1	41174	2	(+41)
477	676	676	41130	1	(+42)
500	554	50	25264	3	(+43)
501	555	2	41112	2	(+44)
502	645	676	41110	0	(+45)
503	444	50	25361	2	(+46)
504	445	2	41177	2	(+47)
505	677	676	41124	1	(+50)
506	552	50	25264	3	(+51)
507	553	4	41114	2	(+52)
510	646	676	41104	0	(+53)
511	442	50	25361	2	(+54)
512	443	4	41000	3	(+55)
513	700	676	41121	1	(+56)
514	550	50	25264	3	(+57)
515	551	10	41117	2	(+60)
516	647	676	41101	0	(+61)
517	440	50	25361	2	(+62)
520	441	10	41003	3	(+63)
521	701	676	41114	1	(+64)
522	546	50	25264	3	(+65)
523	547	20	43120	2	(+66)
524	650	676	41074	0	(+67)
525	436	50	25361	2	(+70)
526	437	20	43004	3	(+71)
527	702	676	41111	1	(+72)
530	544	50	25264	3	(+73)
531	545	20	45123	2	(+74)
532	651	676	41071	0	(+75)
533	434	50	25361	2	(+76)
534	435	20	45007	3	(+77)
535	703	676	41105	1	(+100)
536	542	50	25264	3	(+101)
537	543	20	51125	2	(+102)
540	652	676	41065	0	(+103)
541	432	50	25361	2	(+104)

542	433	20	51010	3	(+105)
543	704	676	41100	1	(+106)
544	540	50	25264	3	(+107)
545	541	20	61126	2	(+110)
546	653	676	41060	0	(+111)
547	430	50	25361	2	(+112)
550	431	20	61013	3	(+113)
551	705	676	41075	1	(+114)
552	536	50	25264	3	(+116)
553	537	21	41131	2	(+116)
554	664	676	41055	0	(+117)
555	426	50	25361	2	(+120)
556	427	21	41015	3	(+121)
557	706	676	41070	1	(+122)
560	534	50	25264	3	(+123)
561	535	22	41132	2	(+124)
562	655	676	41050	0	(+125)
563	424	50	25361	2	(+126)
564	425	22	41016	3	(+127)
565	707	676	41064	1	(+130)
566	532	50	25264	3	(+131)
567	533	24	41134	2	(+132)
570	656	676	41044	0	(+133)
571	422	50	25361	2	(+134)
572	423	24	41020	3	(+136)
573	710	676	41061	1	(+136)
574	530	50	25264	3	(+137)
575	531	30	41137	2	(+140)
576	657	676	41021	0	(+141)
577	410	50	25361	2	(+142)
600	411	30	41046	3	(+143)
601	723	676	41041	0	(+144)
602	420	50	25264	3	(+146)
603	421	33047	105045	7	AFILLR
604	722	30014	177035	0	(+1)
605	416	30150	3030	3	AFILLRL
606	714	40050	125027	17	(+1)
607	713	31350	165025	3	(+2)
610	712	31401	1023	3	(+3)
611	711	50	24235	0	(+4)
612	417	30014	177024	0	ACHKR
613	412	30150	3041	3	ACHKRL
614	720	40450	25036	17	(+1)
615	717	50	24031	0	(+2)
616 b	415	50	25030	0	ACHKRER
617	414	31350	165034	3	(+1)
620	716	31401	1032	3	(+2)
621	715	50	24224	0	(+3)
622	413	45	5042	3	(+4)
623	721	50	25125	0	(+5)
624	670	33047	105157	6	TFILLR
625	667	30050	125155	6	(+1)
626	666	30450	25153	6	(+2)
627	665	50	24050	1	(+3)
630 b	525	50	25156	2	TFILLRER
631	524	30014	137055	1	(+1)
632	526	30150	3147	2	TFILLRL
633	663	40050	125144	16	(+1)
634	662	31350	125142	2	(+2)
635	661	31401	1140	2	(+3)
636	660	50	24254	1	(+4)
637	527	30150	65151	6	(+5)
640	664	30050	125400	0	(+6)
641	732	30450	25063	7	TCHKR
642	731	50	24015	0	(+1)
643 b	407	50	25014	0	TCHKRERO
644	406	30450	25061	3	(+1)
645	730	50	24010	0	(+2)
646 b	405	50	25011	0	TCHKRER1
647	404	30014	137001	0	(+1)
650	400	30150	3056	3	TCHKRL
651	727	40450	25055	17	(+1)
652	726	50	24004	0	(+2)
653 b	403	50	25005	0	TCHKRER2
654	402	31350	125053	3	(+1)
655	725	31401	1051	3	(+2)

656	724	50	24200	0	(+3)
657	401	50	25401	0	(+4)
660	1052	1376	41125	1	PUSH1
661	1152	45	3120	0	(+1)
662	1050	50	25361	2	(+2)
663	1051	30001	137122	1	(+3)
664	1151	30150	3121	1	(+4)
665	1150	30001	141117	5	(+5)
666	1147	30000	137115	10	(+6)
667	1046	42050	125133	14	PUSH1L
670	1055	31050	165131	4	(+1)
671	1054	31350	125127	10	(+2)
672	1053	50	24315	0	(+3)
673	1047	45	11110	0	(+4)
674	1044	50	25230	2	(+5)
675	1045	1376	41115	1	PUSHM1
676	1146	45	3104	0	(+1)
677	1042	50	25361	2	(+2)
700	1043	30001	101112	1	(+3)
701	1145	30150	3111	1	(+4)
702	1144	30001	177106	5	(+5)
703	1143	30000	137101	10	(+6)
704	1040	44050	125141	14	PUSHM1L
705	1060	31350	165137	4	(+1)
706	1057	31350	125135	10	(+2)
707	1056	50	24301	0	(+3)
710	1041	45	11074	0	(+4)
711	1036	50	25230	2	(+5)
712	1037	1376	41104	1	PUSH2
713	1142	45	3071	0	(+1)
714	1034	50	25361	2	(+2)
715	1035	30001	135102	1	(+3)
716	1141	30150	3100	1	(+4)
717	1140	30001	141076	5	(+5)
720	1137	30000	117064	10	(+6)
721	1032	40050	107146	14	PUSH2LE
722	1063	31100	145145	4	(+1)
723	1062	31350	125142	10	(+2)
724	1061	50	24265	0	(+3)
725	1033	30001	137075	1	(+4)
726	1136	30150	3073	1	(+5)
727	1135	30001	143071	5	(+6)
730	1134	30000	117061	10	(+7)
731	1030	40050	107154	14	PUSH2LO
732	1066	31100	145152	4	(+1)
733	1065	31350	125150	10	(+2)
734	1064	50	24260	0	(+3)
735	1031	45	11055	0	(+4)
736	1026	50	25230	2	(+5)
737	1027	1376	41066	1	PUSHM2
740	1133	45	3050	0	(+1)
741	1024	50	25361	2	(+2)
742	1025	30001	101064	1	(+3)
743	1132	30150	3062	1	(+4)
744	1131	30001	175060	5	(+5)
745	1130	30000	117045	10	(+6)
746	1022	46050	125162	14	PUSHM2LE
747	1071	31400	145161	4	(+1)
750	1070	31350	125156	10	(+2)
751	1067	50	24244	0	(+3)
752	1023	30001	103056	1	(+4)
753	1127	30150	3054	1	(+5)
754	1126	30001	177052	5	(+6)
755	1125	30000	117040	10	(+7)
756	1020	46050	125170	14	PUSHM2LO
757	1074	31400	145167	4	(+1)
760	1073	31350	125165	10	(+2)
761	1072	50	24241	0	(+3)
762	1021	45	11035	0	(+4)
763	1016	50	25230	2	(+5)
764	1017	1376	41050	1	PUSH3
765	1124	45	3030	0	(+1)
766	1014	50	25361	2	(+2)
767	1015	30001	133047	1	(+3)
770	1123	30150	3045	1	(+4)
771	1122	30001	141043	5	(+5)

772	1121	30000	137024	10	(+6)
773	1012	42050	107003	15	PUSH3L
774	1101	31100	107001	5	(+1)
775	1100	30200	137177	4	(+2)
776	1077	31101	141175	4	(+3)
777	1076	31350	125172	10	(+4)
1000	1075	50	24224	0	(+5)
1001	1013	45	11021	0	(+6)
1002	1010	50	25230	2	(+7)
1003	1011	1376	41041	1	PUSHM3
1004	1120	45	3014	0	(+1)
1005	1006	50	25361	2	(+2)
1006	1007	30001	101037	1	(+3)
1007	1117	30150	3034	1	(+4)
1010	1116	30001	173033	5	(+5)
1011	1116	30000	137011	10	(+6)
1012	1004	46050	107015	15	PUSHM3L
1013	1106	31400	107013	5	(+1)
1014	1105	30200	137011	5	(+2)
1015	1104	31101	141007	5	(+3)
1016	1103	31350	125005	11	(+4)
1017	1102	50	24211	0	(+5)
1020	1005	45	11005	0	(+6)
1021	1002	50	25230	2	(+7)
1022	1003	33050	125030	15	(+10)
1023	1114	34150	65027	1	(+11)
1024	1113	33450	25024	15	(+12)
1025	1112	50	24200	0	(+13)
1026	1001	45	13022	1	(+14)
1027	1111	50	25003	0	(+15)
1030	b 1000	32020	101021	15	PASSED-EDALU-TEST
1031	1110	45	13017	1	(+1)
1032	1107	50	25003	0	(+2)
1033	2214	47	27075	3	POPTST
1034	2336	50150	65072	17	(+1)
1035	2335	30050	125070	17	(+2)
1036	2334	30001	101066	3	POP1
1037	2333	30150	3065	3	(+1)
1040	2332	42150	65025	16	(+2)
1041	2212	50	25262	3	(+3)
1042	2213	30001	103033	2	(+4)
1043	2215	42150	65020	16	(+5)
1044	2210	50	25262	3	(+6)
1045	2211	30001	105035	2	(+7)
1046	2216	42150	65015	16	(+10)
1047	2206	50	25262	3	(+11)
1050	2207	30001	107037	2	(+12)
1051	2217	42150	65010	16	(+13)
1052	2204	50	25262	3	(+14)
1053	2205	30001	111041	2	(+15)
1054	2220	42150	65004	16	(+16)
1055	2202	50	25262	3	(+17)
1056	2203	30001	113043	2	(+20)
1057	2221	42150	65001	16	(+21)
1060	2200	50	25262	3	(+22)
1061	2201	30001	115045	2	(+23)
1062	2222	42150	65174	15	(+24)
1063	2176	50	25262	3	(+25)
1064	2177	30001	117047	2	(+26)
1065	2223	42150	65171	15	(+27)
1066	2174	50	25262	3	(+30)
1067	2175	30001	121050	2	(+31)
1070	2224	42150	65165	15	(+32)
1071	2172	50	25262	3	(+33)
1072	2173	30001	123052	2	(+34)
1073	2225	42150	65160	15	(+35)
1074	2170	50	25262	3	(+36)
1075	2171	30001	125054	2	(+37)
1076	2226	42150	65155	15	(+40)
1077	2166	50	25262	3	(+41)
1100	2167	30001	127056	2	(+42)
1101	2227	42150	65150	15	(+43)
1102	2164	50	25262	3	(+44)
1103	2165	30001	131061	2	(+45)
1104	2230	42150	65144	15	(+46)
1105	2162	50	25262	3	(+47)

1106	2163	30001	133063	2	(+50)
1107	2231	42150	65141	15	(+51)
1110	2160	50	25262	3	(+62)
1111	2161	30001	135065	2	(+53)
1112	2232	42150	65135	15	(+54)
1113	2166	50	25262	3	(+55)
1114	2157	30001	137067	2	(+56)
1115	2233	42150	65130	15	(+57)
1116	2154	50	25262	3	(+60)
1117	2155	30001	101070	2	(+61)
1120	2234	42150	65124	15	(+62)
1121	2152	50	25262	3	(+63)
1122	2153	30001	101075	2	POP1
1123	2236	30150	3073	2	(+1)
1124	2235	44150	65121	15	(+2)
1125	2150	50	25262	3	(+3)
1126	2151	30001	137076	2	(+4)
1127	2237	44150	65114	15	(+5)
1130	2146	50	25262	3	(+6)
1131	2147	30001	135101	2	(+7)
1132	2240	44150	65111	15	(+10)
1133	2144	50	25262	3	(+11)
1134	2145	30001	133102	2	(+12)
1135	2241	44150	65105	15	(+13)
1136	2142	50	25262	3	(+14)
1137	2143	30001	131105	2	(+15)
1140	2242	44150	65100	15	(+16)
1141	2140	50	25262	3	(+17)
1142	2141	30001	127107	2	(+20)
1143	2243	44150	65075	15	(+21)
1144	2136	50	25262	3	(+22)
1145	2137	30001	125111	2	(+23)
1146	2244	44150	65070	15	(+24)
1147	2134	50	25262	3	(+25)
1150	2135	30001	123112	2	(+26)
1151	2245	44150	65064	15	(+27)
1152	2132	50	25262	3	(+30)
1153	2133	30001	121115	2	(+31)
1154	2246	44150	65061	15	(+32)
1155	2130	50	25262	3	(+33)
1156	2131	30001	117116	2	(+34)
1157	2247	44150	65054	15	(+35)
1160	2126	50	25262	3	(+36)
1161	2127	30001	115121	2	(+37)
1162	2250	44150	65051	15	(+40)
1163	2124	50	25262	3	(+41)
1164	2125	30001	113122	2	(+42)
1165	2251	44150	65045	15	(+43)
1166	2122	50	25262	3	(+44)
1167	2123	30001	111125	2	(+45)
1170	2252	44150	65040	15	(+46)
1171	2120	50	25262	3	(+47)
1172	2121	30001	107127	2	(+50)
1173	2253	44150	65034	15	(+51)
1174	2116	50	25262	3	(+52)
1175	2117	30001	105131	2	(+53)
1176	2254	44150	65031	15	(+54)
1177	2114	50	25262	3	(+55)
1200	2115	30001	103132	2	(+56)
1201	2255	44150	65025	15	(+57)
1202	2112	50	25262	3	(+60)
1203	2113	30001	101135	2	(+61)
1204	2256	44150	65020	15	(+62)
1205	2110	50	25262	3	(+63)
1206	2111	30001	101141	2	POP2E
1207	2260	30150	3136	2	(+1)
1210	2257	40150	47014	15	(+2)
1211	2106	50	25262	3	(+3)
1212	2107	30001	105143	2	(+4)
1213	2261	40150	47011	15	(+5)
1214	2104	50	25262	3	(+6)
1215	2105	30001	111145	2	(+7)
1216	2262	40150	47005	15	(+10)
1217	2102	50	25262	3	(+11)
1220	2103	30001	115147	2	(+12)
1221	2263	40150	47000	15	(+13)

1222	2100	50	25262	3	(+14)
1223	2101	30001	121151	2	(+15)
1224	2264	40150	47174	14	(+16)
1225	2076	50	25262	3	(+17)
1226	2077	30001	125153	2	(+20)
1227	2265	40150	47171	14	(+21)
1230	2074	50	25262	3	(+22)
1231	2075	30001	131155	2	(+23)
1232	2266	40150	47165	14	(+24)
1233	2072	50	25262	3	(+25)
1234	2073	30001	135157	2	(+26)
1235	2267	40150	47160	14	(+27)
1236	2070	50	25262	3	(+30)
1237	2071	30001	101160	2	(+31)
1240	2270	40150	47155	14	(+32)
1241	2066	50	25262	3	(+33)
1242	2067	30001	103164	2	POP20
1243	2272	30150	3163	2	(+1)
1244	2271	40150	47150	14	(+2)
1245	2064	50	25262	3	(+3)
1246	2065	30001	107166	2	(+4)
1247	2273	40150	47144	14	(+5)
1250	2062	50	25262	3	(+6)
1251	2063	30001	113171	2	(+7)
1252	2274	40150	47141	14	(+10)
1253	2060	50	25262	3	(+11)
1254	2061	30001	117173	2	(+12)
1255	2275	40150	47135	14	(+13)
1256	2056	50	25262	3	(+14)
1257	2057	30001	123174	2	(+15)
1260	2276	40150	47130	14	(+16)
1261	2054	50	25262	3	(+17)
1262	2055	30001	127176	2	(+20)
1263	2277	40150	47124	14	(+21)
1264	2052	50	25262	3	(+22)
1265	2053	30001	133001	3	(+23)
1266	2300	40150	47121	14	(+24)
1267	2050	50	25262	3	(+25)
1270	2051	30001	137003	3	(+26)
1271	2301	40150	47114	14	(+27)
1272	2046	50	25262	3	(+30)
1273	2047	30001	103004	3	(+31)
1274	2302	40150	47111	14	(+32)
1275	2044	50	25262	3	(+33)
1276	2045	30001	101011	3	POP2ME
1277	2304	30150	3006	3	(+1)
1300	2303	46150	65105	14	(+2)
1301	2042	50	25262	3	(+3)
1302	2043	30001	135013	3	(+4)
1303	2305	46150	65100	14	(+5)
1304	2040	50	25262	3	(+6)
1305	2041	30001	131014	3	(+7)
1306	2306	46150	65075	14	(+10)
1307	2036	50	25262	3	(+11)
1310	2037	30001	125017	3	(+12)
1311	2307	46150	65070	14	(+13)
1312	2034	50	25262	3	(+14)
1313	2035	30001	121020	3	(+15)
1314	2310	46150	65064	14	(+16)
1315	2032	50	25262	3	(+17)
1316	2033	30001	115022	3	(+20)
1317	2311	46150	65061	14	(+21)
1320	2030	50	25262	3	(+22)
1321	2031	30001	111025	3	(+23)
1322	2312	46150	65054	14	(+24)
1323	2026	50	25262	3	(+25)
1324	2027	30001	105026	3	(+26)
1325	2313	46150	65051	14	(+27)
1326	2024	50	25262	3	(+30)
1327	2025	30001	101030	3	(+31)
1330	2314	46150	65045	14	(+32)
1331	2022	50	25262	3	(+33)
1332	2023	30001	103034	3	POP20
1333	2316	30150	3033	3	(+1)
1334	2315	46150	65040	14	(+2)
1335	2020	50	25262	3	(+3)

1336	2021	30001	137036	3	(+4)
1337	2317	46150	65034	14	(+5)
1340	2016	50	25262	3	(+6)
1341	2017	30001	133040	3	(+7)
1342	2320	46150	65031	14	(+10)
1343	2014	50	25262	3	(+11)
1344	2016	30001	127043	3	(+12)
1345	2321	46150	65025	14	(+13)
1346	2012	50	25262	3	(+14)
1347	2013	30001	123044	3	(+15)
1350	2322	46150	65020	14	(+16)
1351	2010	50	25262	3	(+17)
1352	2011	30001	117046	3	(+20)
1353	2323	46150	65015	14	(+21)
1354	2006	50	25262	3	(+22)
1355	2007	30001	113050	3	(+23)
1356	2324	46150	65010	14	(+24)
1357	2004	50	25262	3	(+25)
1360	2005	30001	107053	3	(+26)
1361	2325	46150	65001	14	(+27)
1362	2000	50	25262	3	(+30)
1363	2001	30001	103057	3	(+31)
1364	2327	46150	65004	14	(+32)
1365	2002	50	25262	3	(+33)
1366	2003	45	7054	3	(+34)
1367	2326	50	25010	1	(+35)
1370	1504	30001	101127	1	POP3
1371	1553	30150	3124	1	(+1)
1372	1552	42150	47004	15	(+2)
1373	1502	50	25323	1	(+3)
1374	1503	30001	107013	1	(+4)
1375	1505	42150	47001	15	(+5)
1376	1500	50	25323	1	(+6)
1377	1501	30001	115015	1	(+7)
1400	1506	42150	47175	14	(+10)
1401	1476	50	25323	1	(+11)
1402	1477	30001	123016	1	(+12)
1403	1507	42150	47170	14	(+13)
1404	1474	50	25323	1	(+14)
1405	1475	30001	131020	1	(+15)
1406	1510	42150	47164	14	(+16)
1407	1472	50	25323	1	(+17)
1410	1473	30001	137023	1	(+20)
1411	1511	42150	47161	14	(+21)
1412	1470	50	25323	1	(+22)
1413	1471	30001	105024	1	(+23)
1414	1512	42150	47154	14	(+24)
1415	1466	50	25323	1	(+25)
1416	1467	30001	113026	1	(+26)
1417	1513	42150	47151	14	(+27)
1420	1464	50	25323	1	(+30)
1421	1465	30001	121030	1	(+31)
1422	1514	42150	47145	14	(+32)
1423	1462	50	25323	1	(+33)
1424	1463	30001	127033	1	(+34)
1425	1515	42150	47140	14	(+35)
1426	1460	50	25323	1	(+36)
1427	1461	30001	135035	1	(+37)
1430	1516	42150	47134	14	(+40)
1431	1466	50	25323	1	(+41)
1432	1457	30001	103036	1	(+42)
1433	1517	42150	47131	14	(+43)
1434	1454	50	25323	1	(+44)
1435	1455	30001	111041	1	(+45)
1436	1520	42150	47125	14	(+46)
1437	1452	50	25323	1	(+47)
1440	1453	30001	117042	1	(+50)
1441	1521	42150	47120	14	(+51)
1442	1450	50	25323	1	(+52)
1443	1451	30001	125045	1	(+53)
1444	1522	42150	47115	14	(+54)
1445	1446	50	25323	1	(+55)
1446	1447	30001	133047	1	(+56)
1447	1523	42150	47110	14	(+57)
1450	1444	50	25323	1	(+60)
1451	1445	30001	101051	1	(+61)

1452	1524	42150	47104	14	(+62)
1453	1442	50	25323	1	(+63)
1454	1443	30001	101054	1	POPM3
1455	1526	30150	3052	1	(+1)
1456	1525	46150	47100	14	(+2)
1457	1440	50	25323	1	(+3)
1460	1441	30001	133056	1	(+4)
1461	1527	46150	47075	14	(+5)
1462	1436	50	25323	1	(+6)
1463	1437	30001	125061	1	(+7)
1464	1530	46150	47070	14	(+10)
1465	1434	50	25323	1	(+11)
1466	1435	30001	117063	1	(+12)
1467	1531	46150	47064	14	(+13)
1470	1432	50	25323	1	(+14)
1471	1433	30001	111065	1	(+16)
1472	1532	46150	47061	14	(+16)
1473	1430	50	25323	1	(+17)
1474	1431	30001	103066	1	(+20)
1475	1533	46150	47054	14	(+21)
1476	1426	50	25323	1	(+22)
1477	1427	30001	135071	1	(+23)
1500	1534	46150	47051	14	(+24)
1501	1424	50	25323	1	(+25)
1502	1425	30001	127072	1	(+26)
1503	1535	46150	47045	14	(+27)
1504	1422	50	25323	1	(+30)
1505	1423	30001	121074	1	(+31)
1506	1536	46150	47040	14	(+32)
1507	1420	50	25323	1	(+33)
1510	1421	30001	113076	1	(+34)
1511	1537	46150	47034	14	(+35)
1512	1416	50	25323	1	(+36)
1513	1417	30001	106101	1	(+37)
1514	1540	46150	47031	14	(+40)
1515	1414	50	25323	1	(+41)
1516	1415	30001	137103	1	(+42)
1517	1541	46150	47025	14	(+43)
1520	1412	50	25323	1	(+44)
1521	1413	30001	131105	1	(+45)
1522	1542	46150	47020	14	(+46)
1523	1410	50	25323	1	(+47)
1524	1411	30001	123106	1	(+50)
1525	1543	46150	47015	14	(+51)
1526	1406	50	25323	1	(+52)
1527	1407	30001	115111	1	(+53)
1530	1544	46150	47010	14	(+54)
1531	1404	50	25323	1	(+55)
1532	1405	30001	107112	1	(+56)
1533	1545	46150	47001	14	(+57)
1534	1400	50	25323	1	(+60)
1535	1401	30001	101117	1	(+61)
1536	1547	46150	47004	14	(+62)
1537	1402	50	25323	1	(+63)
1540	1403	30147	21114	15	(+64)
1541	1546	50	25401	0	(+65)
1542	4	30450	25013	0	COMPARE0
1543	5	50	24004	0	(+1)
1544	b 3	50	25005	0	FAIL
1545	2	50	25401	0	SUCCESS
1546	636	45	1072	2	COMPARE1
1547	635	50	25011	0	(+1)
1550	1153	45	1131	1	COMPARE2
1551	1154	50	25011	0	(+1)
1552	1551	45	1120	1	COMPARE3
1553	1550	50	25011	0	(+1)
1554	2331	45	1061	3	COMPARE4
1555	2330	50	25011	0	(+1)
1556	2715	45	1031	3	COMPARE5
1557	2714	50	25011	0	(+1)

Page 0: 4 locations used, 374 free  
Page 400: 341 locations used, 37 free  
Page 1000: 155 locations used, 223 free  
Page 1400: 154 locations used, 224 free  
Page 2000: 337 locations used, 41 free



Page 2400: 323 locations used, 55 free

RM:

0		S0
1		S1
2		RT
3		OLDAPC
4	1	REVISION
5		MCOUNT
6	11	RUN-TIME
7	0	PASSCOUNT
10	1000	MAXPASS
11		RLC@

Time: 24 seconds; 0 error(s), 0 warning(s), 11798 words free

```
.....  
:EDALULog.MIDAS : Logger for EDALU program  
: By: J. Kellman  
: Dec. 10 1979  
.....
```

```
.start L X AppendOutput EDALU.report;  
L X WriteMessage ~***** START EDALU Test : ;  
L X WriteDT;  
L X WriteMessage *****~ ;  
L X Skip .continue;  
  
.breakpoint L X AppendOutput EDALU.report;  
  
L A18 SkipNE BADNOTIFY;  
L X Skip .badnotify;  
L A18 SkipNE BADWAKE+2;  
L X Skip .badwake;  
L A18 SkipNE TFILLRER;  
L X Skip .tfillrer;  
L A18 SkipNE TCHKRER0;  
L X Skip .tchkrer0;  
L A18 SkipNE TCHKRER1;  
L X Skip .tchkrer1;  
L A18 SkipNE TCHKRER2;  
L X Skip .tchkrer2;  
L A18 SkipNE ACHKRER;  
L X Skip .achkrer;  
L A18 SkipNE FAIL;  
L X Skip .fail;  
L A18 SkipNE PASSED-EDALU-TEST;  
L X Skip .passtest;  
  
.notmybreak L X AppendOutput EDALU.report;  
L X WriteMessage *** FAILED: Not at my breakpoint ~;  
  
L X WriteMessage ' Parity = ;  
R A0 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' CIA = ;  
R A18 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' CTASK = ;  
R A19 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' APCTASK = ;  
R A17 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' APC = ;  
R A16 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' TPC = ;  
R A13 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X CloseOutput;  
L X Exit;  
  
.badnotify L X WriteMessage *** FAILED: at my Breakpoint Bad NOTIFY ~;  
bad L X WriteMessage ' MCOUNT = ;  
R B8 Val;  
L X WriteMessage;  
! X WriteMessage ~;
```

```
L X WriteMessage ' T 0 =  ;
R B15 Val;
L X WriteMessage;
L X WriteMessage ~;

L X WriteMessage ' S0 =  ;
R B16 Val;
L X WriteMessage;
L X WriteMessage ~;

L X WriteMessage ' S1 =  ;
R B17 Val;
L X WriteMessage;
L X WriteMessage ~;

L X Skip .continue;

.badwake L X WriteMessage *** FAILED: at my Breakpoint BAD WAKE ~;
L X BackSkip .bad;

.tfillrer L X WriteMessage *** FAILED: at my Breakpoint T FILL R Error ~;
L X BackSkip .bad;

.tchkrer0 L X WriteMessage *** FAILED: at my Breakpoint T CHECK R Error0 ~;
L X BackSkip .bad;

.tchkrer1 L X WriteMessage *** FAILED: at my Breakpoint T CHECK R Error1 ~;
L X BackSkip .bad;

.tchkrer2 L X WriteMessage *** FAILED: at my Breakpoint T CHECK R Error2 ~;
L X BackSkip .bad;

.achkrer L X WriteMessage *** FAILED: at my Breakpoint Addr Check R Error ~;
L X BackSkip .bad;

.fail L X WriteMessage *** FAILED: at my Breakpoint compare FAIL~;
L X BackSkip .bad;

.passtest L X WriteMessage ~----- PASSEd EDALU Test :  ;
L X WriteDT;
L X WriteMessage ~-----~ ;
L X Skip .continue;

.continue L X WriteMessage ~;
L X CloseOutput;
L X DisplayOn;
L X Confirm;
L X TimeOut 10000000;
L X Continue;
L X Skip 2;
L X ShowError Program failed to CONTINUE.;
L X BackSkip .notmybreak;
L X DisplayOff;
L X BackSkip .breakpoint;
```

L A19 Val 0  
L X Confirm  
L X Load EDALU  
L B0 Addr REVISION  
L B1 Addr RUN-TIME  
L B2 Addr PASSCOUNT;  
L B3 Addr MAXPASS;  
L B8 Addr MCOUNT  
L B15 Addr T 0  
L B16 Addr S0  
L B17 Addr S1  
L X DisplayOn;  
L X TimeOut 10000  
L X SS GO  
L X Skip 1  
L X ShowError Single-step at GO hung

\*\*\* \*\* \* <D0Diag>Rev-1>EDBitBlt.mc Revision 1 Dec. 6, 1979 \*\* \* \*\* \* \*\*

EDBitBlt.mc : Bit Boundary Block Transfer microcode
\*\*\* Purpose : This test moves information from one region of main storage to another,
modifying the information at the destination as the transfer is done.
\*\*\* Hardware Configuration : Standard 4 CPU boards.
\*\*\* Written by : Tom Horsley, Nov. 15, 1977
\*\*\* Modified by : Bill Kennedy, Mar. 10, 1978
Took code off of Page 0.
\*\*\* Modified by : Bill Kennedy, Apr. 7, 1978
Added MPanel use.
\*\*\* Modified by : Chuck Thacker, Aug. 22, 1978
Looped on errors.
\*\*\* Modified by : Camellia Chan, Oct. 30, 1979
Standardize title page, code format and labels, modified looping.

\*\*\*\*\*

\*SubTest Description:
\* SubTest 1: the number of times the hardware goes around the loop to complete an item
(LoopCount) is compared with the simulators loop count (simLoopCount).
\* SubTest 2: confirms that the reason the hardware exited the loop (Result) is the same
as the simulators (simResult). The reasons are encoded as follows:
1 = item done
2 = fill source and destination
3 = fill source
4 = fill destination
\* SubTest 3: confirm that the first word of the hardware destination quadword (Dest0)
matches the simulated equivalent (simDest0).
\* SubTest 4: confirm that the second word of the hardware destination quadword (Dest1)
matches the simulated equivalent (simDest1).
\* SubTest 5: confirm that the third word of the hardware destination quadword (Dest2)
matches the simulated equivalent (simDest2).
\* SubTest 6: confirm that the fourth word of the hardware destination quadword (Dest3)
matches the simulated equivalent (simDest3).
\* SubTest 7: confirm that the hardware register MNBR contains the expected value (simNBR).
\* SubTest 10: confirm that the hardware register SB contains the expected value (simSB).
\* SubTest 11: confirm that the hardware register DB contains the expected value (simDB).

\*\*\*\*\*

\*BreakPoints:
\* Error1: hardware loop count does not equal to simulator's loop count.
\* Error2: reason for hardware existed the loop is not the same as the simulator.
\* Error3: the first word of the hardware destination quadword (Dest0) does not match
the simulated equivalent (simDest0).
\* Error4: the second word of the hardware destination quadword (Dest1) does not match
the simulated equivalent (simDest1).
\* Error5: the third word of the hardware destination quadword (Dest2) does not match
the simulated equivalent (simDest2).
\* Error6: the fourth word of the hardware destination quadword (Dest3) does not match
the simulated equivalent (simDest3).
\* Error7: the hardware register MNBR does not contain the expected value (simNBR).
\* Error10: the hardware register SB does not contain the expected value (simSB).
\* Error11: the hardware register DB does not contain the expected value (simDB).
Passed-EDBitBlt-Test: Passed all tests, and all passes.

```
*****
* ShortLoop Logic Analyzer Sync Points at Control Store address:
* Error1: Control Store address 600 at Repeat.
* Error2: Control Store address 600 at Repeat.
* Error3: Control Store address 600 at Repeat.
* Error4: Control Store address 600 at Repeat.
* Error5: Control Store address 600 at Repeat.
* Error6: Control Store address 600 at Repeat.
* Error7: Control Store address 600 at Repeat.
* Error10: Control Store address 600 at Repeat.
* Error11: Control Store address 600 at Repeat.
```

```
*****
*Subroutine Description:
* BitBlt:      invoke hardware bitblt.
* SimBitBlt:  simulate a bitblt given simDB, simSB, simNBR, simOP, simMaskFillSrc,
               simBBF1.
```

```
*****
*Special Reg. Definition:
* LoopOn:     At any breakpoint, the user has the choice of setting LoopOn to 1, 2, 3, 4, 5, 6,
               7, 10, or 11 to loop from Subtest0 to that subtest repeatedly for trouble shooting.

               0. no looping on any subtest.
               N, loop on subtest N, for N=1,2,3,4,5,6,7,10,11.
```

```
* XA and XB:  The two random numbers held in these registers XA and XB are used to choose
               the bitblt starting values as follows:
```

- SB - XA[0. 6]
- DB - XA[6. 6]
- MaskDestination - XB[0. 1]
- Mask and Fill Source - XB[1. 1]
- SALUF function - XB[2. 6]
- MNBR - -(XB[10. 7] - 1)
- Src0 - XA
- Src1 - XB
- Src2 - NOT XA
- Src3 - NOT XB
- Dest0 - XB
- Dest1 - XA
- Dest2 - 0
- Dest3 - 177777

Note that the random number generator has been constructed so that it produces each number in the range [0. 64K) once and only once before repeating any number. Thus it is guaranteed to exhaust all possible combinations of the fields derived from it each time the inner loop is exhausted.

```
InnerLoopCounter: 16 bits inner loop counter.
PassCount:       Outer loop pass counter,
                 incremented each time when InnerLoopCounter reached the limit
MaxPass:         number of times outer loop is to repeat before breakpointing.
```

```
*****
```

\*\*\*\*\*  
 \*INITIALIZATION:

```
BUILTIN[INSERT,24];
INSERT[d0lang];
TITLE[EDBitBlt]; * Bitblt test program, ED revision
SET[BBO, 1200]; * base for BBFB dispatch
SET[BB1, 1300]; * base for BBFBX dispatch
SET[MainPage, 1]; * tag for Main Program page number
SET[SubPage1, 2]; * tag for Subroutine page number
```

\*\*\*\*\* Macro constants: \*\*\*\*\*

```
MC[fillSource, 3]; * bitblt dispatch type indicating source ran out
MC[fillDest, 4]; * bitblt dispatch type indicating destination ran out
MC[fillBoth, 2]; * bitblt dispatch type indicating both source and destination ran out
MC[itemDone, 1]; * bitblt dispatch type indicating item done
```

\*\*\*\*\* R-registers: \*\*\*\*\*

```
RV[CA,1]; * used in random number generation, A*XA + CA
RV[XA,2]; * random number generated via A*XA + CA
RV[CB,3]; * used in the random number calculation (a*XB + CB)
RV[XB,4]; * second random number
RV[InnerLoopCounter,5,0]; * 16 bits inner loop counter
RV[PassCount,6]; * Outer loop pass counter incremented each time when InnerLoopCounter
* reached the limit
RV[MaxPass,7,2]; * number of times outer loop is to repeat before breakpointing
RV[LoopOn,13,0]; * loop on subtest
RV[SubTest,14]; * current location of test

RV[allOnes,15]; * holds 177777 during mask creations
RV[Dest0,20]; * hardware bitblt destination quadword
RV[Dest1,21];
RV[Dest2,22];
RV[Dest3,23];
RV[destBitsToGo,24]; * number of bits between current DB and next word boundary
RV[destFieldDescriptor,25]; * describes the destination bit field
RV[destMask,26]; * mask used to set background in source field and/or clear destination field
RV[destStart,27]; * lower 4 bits of DB (word offset)
RV[destWord,30]; * working register for current destination word
RV[LoopCount,31]; * number of times through loop of hardware bitblt
RV[MaskDest,32]; * flag indicating whether to clear the destination field or leave it
RV[nbits,33]; * distance to next word boundary or the end of the item
RV[Result,34]; * indicates which dispatch was taken out of hardware bitblt
* (see fillSource etc. below)

RV[simDB,35]; * simulated bitblt's DB
RV[simDest0,40]; * simulated bitblt destination quadword
RV[simDest1,41];
RV[simDest2,42];
RV[simDest3,43];
RV[simLoopCount,44]; * simulated bitblt loop count
RV[simMaskFillSrc,45]; * flag indicating whether background of source field should be 0 or 1
RV[simNBR,46]; * simulated bitblt MNR
RV[simOp,47]; * simulated bitblt SALUF operator
RV[simResult,52]; * indicates which dispatch was indicated by simulated bitblt
* (see fillSource etc. below)

RV[simSB,53]; * simulated bitblt SB
RV[Src0,54]; * source quadword for both bitblt's
RV[Src1,55];
RV[Src2,56];
RV[Src3,57];
RV[srcBitsToGo,60]; * number of bits between current SB and next word boundary
RV[srcFieldDescriptor,61]; * describes the source bit field
RV[srcStart,62]; * lower 4 bits of SB (word offset)
RV[srcWord,63]; * working register for current source word
RV[Tmp,64]; * temporary register
RV[tmpWord,65]; * temporary register
RV[Revision,66,1]; * REVISION 1
RV[Run-Time,67,14]; * Run-Time is 14b or 12 seconds
```

```

*****
*** MAIN routine:

ONPAGE[MainPage];

go:
start:      PassCount ← 0C;
            CLEARMPANEL;

            *RandomInit (Initialize random generator registers: XA ← 123, CA ← 33031)
            XA ← AND@[0377, 123]C;          *Load16Bits (XA ← 123)
            XA ← (XA) OR (AND@[177400, 123]C);
            CA ← AND@[0377, 33031]C;       *Load16Bits (CA ← 33031)
            CA ← (CA) OR (AND@[177400, 33031]C);

            *RandomInit (Initialize random generator registers: XB ← 012300, CB ← 33037)
            XB ← AND@[0377, 012300]C;     *Load16Bits (XB ← 012300)
            XB ← (XB) OR (AND@[177400, 012300]C);
            CB ← AND@[0377, 33037]C;     *Load16Bits (CB ← 33037)
            CB ← (CB) OR (AND@[177400, 33037]C);

bigLoop:    INCMMPANEL;
            t ← PassCount + (PassCount) + 1;
            LU ← (MaxPass) - (t);          * check for maximum pass counter reached
            GOTO[Then1A, ALU >= 0];

Passed-EDBitBlt-Test: BREAKPOINT, goto[go];

Then1A:     XB ← (XB) + 1;

mainloop:   InnerLoopCounter ← (InnerLoopCounter) + 1;
            GOTO[bigLoop, CARRY];

            * Random (4005*XA + CA mod 2**16)
            t ← XA;
            t ← (LSH[XA, 2]) + t;
            t ← (LSH[XA, 13]) + t;
            t ← (CA) + t;
            XA ← t;

            * Random (4005*XB + CB mod 2**16)
            t ← XB;
            t ← (LSH[XB, 2]) + t;
            t ← (LSH[XB, 13]) + t;
            t ← (CB) + t;
            XB ← t;

            * SUBTEST 0:
Repeat:     Subtest ← 0C;
            TASK;                               * allow mouse halt

            * load source and destination registers
            t ← XA;
            src0 ← t;
            src2 ← (ZERO) OR NOT (t);
            dest1 ← t;
            simDest1 ← t;
            t ← XB;
            src1 ← t;
            src3 ← (ZERO) OR NOT (t);
            dest0 ← t;
            simDest0 ← t;
            dest2 ← 0C;
            simDest2 ← 0C;
            dest3 ← (ZERO) - 1;
            simDest3 ← (ZERO) - 1;

```



```

* load DB and SB
  t ← LDF[XA, 6, 6];
  simDB ← t;
  DB ← (simDB);
  t ← LDF[XA, 0, 6];
  simSB ← t;
  SB ← (simSB);

* load MNBR
  t ← (LDF[XB, 10, 7]) + 1;
  simMNB ← (ZERO) - t;
  MNBR ← (simMNB);

* load SALUF and related bits
  t ← LDF[XB, 0, 10];
  SALUF ← t;
  t ← LDF[XB, 2, 6];
  simOP ← t;
  t ← LDF[XB, 0, 1];
  simMaskFillSrc ← t;
  t ← LDF[XB, 1, 1];
  MaskDest ← t;

* do it
  LOADPAGE[SubPage1];
  CALLP[SimBitb1t];
  LOADPAGE[SubPage1];
  CALLP[B1tb1t];

CompareResult:
SUBTEST1:  subTest ← 1C;
           t ← subTest;
           lu ← (LoopOn)-(t);
           goto[. +2, ALU # 0];          * check for looping on this subtest
           goto[Repeat];
           t ← simLoopCount;           * hardware loop count is the same as simulator's loop count?
           LU ← (LoopCount) - (t);
           GOTO[SUBTEST2, ALU = 0];
Error1:   BREAKPOINT, goto[Repeat];

SUBTEST2:  subTest ← 2C;
           t ← subTest;
           lu ← (LoopOn)-(t);
           goto[. +2, ALU # 0];          * check for looping on this subtest
           goto[Repeat];
           t ← simResult;              * reason for hardware existed the loop is the same as simulator?
           LU ← (Result) - (t);
           GOTO[SUBTEST3, ALU = 0];
Error2:   BREAKPOINT, goto[Repeat];

SUBTEST3:  subTest ← 3C;
           t ← subTest;
           lu ← (LoopOn)-(t);
           goto[. +2, ALU # 0];          * check for looping on this subtest
           goto[Repeat];
           t ← simDest0;               * 1st word of the hardware dest. quadword matches the
                                           * simulated equivalent?
           LU ← (Dest0) - (t);
           GOTO[SUBTEST4, ALU = 0];
Error3:   BREAKPOINT, goto[Repeat];

SUBTEST4:  subTest ← 4C;
           t ← subTest;
           lu ← (LoopOn)-(t);
           goto[. +2, ALU # 0];          * check for looping on this subtest
           goto[Repeat];
           t ← simDest1;               * 2nd word of the hardware dest. quadword matches the
                                           * simulated equivalent?
           LU ← (Dest1) - (t);
           GOTO[SUBTEST5, ALU = 0];
Error4:   BREAKPOINT, goto[Repeat];

SUBTEST5:  subTest ← 5C;
           t ← subTest;
           lu ← (LoopOn)-(t);

```

```

goto[.+2,ALU # 0];      * check for looping on this subtest
goto[Repeat];
t ← simDest2;          * 3rd word of the hardware dest. quadword matches the
                      * simulated equivalent?

LU ← (Dest2) - (t);
GOTO[SUBTEST6, ALU = 0];
Error5: BREAKPOINT,goto[Repeat];

SUBTEST6: subTest ← 6C;
          t ← subTest;
          lu ← (LoopOn)-(t);
          goto[.+2,ALU # 0];      * check for looping on this subtest
          goto[Repeat];          * 4th word of the hardware dest. quadword matches the
          t ← simDest3;          * simulated equivalent?

LU ← (Dest3) - (t);
GOTO[SUBTEST7, ALU = 0];
Error6: BREAKPOINT,goto[Repeat];

SUBTEST7: subTest ← 7C;
          t ← subTest;
          lu ← (LoopOn)-(t);
          goto[.+2,ALU # 0];      * check for looping on this subtest
          goto[Repeat];          * the hardware register MNBR contains the
          t ← MNBR;              * expected value (simMNB)?

LU ← (simMNB) - (t);
GOTO[SUBTEST10, ALU = 0];
Error7: BREAKPOINT,goto[Repeat];

SUBTEST10: subTest ← 10C;
          t ← subTest;
          lu ← (LoopOn)-(t);
          goto[.+2,ALU # 0];      * check for looping on this subtest
          goto[Repeat];          * the hardware register SB contains the
          t ← DBSB;              * expected value (simSB)?

          Tmp ← t;
          t ← LDF[Tmp, 12, 6];
          LU ← (simSB) - (t);
          GOTO[SUBTEST11, ALU = 0];
Error10: BREAKPOINT,goto[Repeat];

SUBTEST11: subTest ← 11C;
          t ← subTest;
          lu ← (LoopOn)-(t);
          goto[.+2,ALU # 0];      * check for looping on this subtest
          goto[Repeat];          * the hardware register DB contains the
          t ← DBSB;              * expected value (simDB)?

          Tmp ← t;
          t ← LDF[Tmp, 4, 6];
          LU ← (simDB) - (t);
          GOTO[TheEnd, ALU = 0];
Error11: BREAKPOINT,goto[Repeat];

TheEnd:  LOADPAGE[MainPage];
          GOTOP[mainloop];

```

```
*****
*SUBROUTINE
ONPAGE[SubPage1];
***** SUBROUTINE: Bitblt *****
*
*   invoke hardware bitblt

Bitblt:      LoopCount ← 0C;
             GOTO[maskTheDestination, MB];

dontMaskDestination:
BBFB;      * initialize internal bitblt registers
BBLoop0:   LoopCount ← (LoopCount) + 1, AT[BB0, 07];
           t ← BBFA[SB[src0]] OR (t);
           DB[dest0] ← (BBFBX[DB[dest0]]) SALUFOP (t), DISP[BBLoop0];

BBFillSource0: Result ← fillSource, RETURN, AT[BB0, 06];
BBFillDest0:   Result ← fillDest, RETURN, AT[BB0, 06];
BBFillBoth0:   Result ← fillBoth, RETURN, AT[BB0, 04];
BBItemDone0:   Result ← itemDone, RETURN, AT[BB0, 03];

maskTheDestination:
BBFB;      * initialize internal bitblt registers
BBLoop1:   LoopCount ← (LoopCount) + 1, AT[BB1, 07];
           t ← BBFA[SB[src0]] OR (t);
           DB[dest0] ← (BBFB[DB[dest0]]) SALUFOP (t), DISP[BBLoop1];

BBFillSource1: Result ← fillSource, RETURN, AT[BB1, 06];
BBFillDest1:   Result ← fillDest, RETURN, AT[BB1, 06];
BBFillBoth1:   Result ← fillBoth, RETURN, AT[BB1, 04];
BBItemDone1:   Result ← itemDone, RETURN, AT[BB1, 03];

RETURN;
```

\*\*\*\*\* SUBROUTINE: SimBitblt \*\*\*\*\*

\* simulate a bitblt given simDB, simSB, simNBR, simOp, simMaskFillSrc, simBBF1

SimBitblt: simLoopCount ← 0C;

SimBitStart: simLoopCount ← (simLoopCount) + 1;

\* get correct source word, based on simSB

t ← LDF[simSB, 12, 2];

tmpWord ← t;

LU ← (tmpWord);

GOTO[TrySrc1, ALU # 0]; \* go to TrySrc1 if the correct source word is not Src0

t ← (Src0); \* Copy (srcWord ← Src0)

srcWord ← t;

GOTO[CalSrcBit];

TrySrc1: LU ← (tmpWord) - (1C);

GOTO[TrySrc2, ALU # 0]; \* go to TrySrc2 if the correct source word is not Src1

t ← (Src1); \* Copy (srcWord ← Src1)

srcWord ← t;

GOTO[CalSrcBit];

TrySrc2: LU ← (tmpWord) - (2C);

GOTO[IsSrc3, ALU # 0]; \* go to IsSrc3 if the correct source word is not Src2

t ← (Src2); \* Copy (srcWord ← Src2)

srcWord ← t;

GOTO[CalSrcBit];

IsSrc3: t ← (Src3);

\* Copy (srcWord ← Src3)

srcWord ← t;

\* calculate actual source bit related values

CalSrcBit: t ← LDF[simSB, 14, 4];

srcStart ← t;

srcBitsToGo ← (ZERO) - t;

srcBitsToGo ← (srcBitsToGo) + (20C);

\* get correct destination word, based on simDB

t ← LDF[simDB, 12, 2];

tmpWord ← t;

LU ← (tmpWord);

GOTO[TryDest1, ALU # 0];

\* go to TryDest1 if the correct destination word is not simDest0

t ← (simDest0);

\* Copy (destWord ← simDest0)

destWord ← t;

GOTO[CalDstBit];

TryDest1: LU ← (tmpWord) - (1C);

GOTO[TryDest2, ALU # 0];

\* go to TryDest2 if the correct destination word is not simDest1

t ← (simDest1);

\* Copy (destWord ← simDest1)

destWord ← t;

GOTO[CalDstBit];

TryDest2: LU ← (tmpWord) - (2C);

GOTO[IsDest3, ALU # 0];

\* go to IsDest3 if the correct destination word is not simDest2

t ← (simDest2);

\* Copy (destWord ← simDest2)

destWord ← t;

GOTO[CalDstBit];

IsDest3: t ← (simDest3);

\* Copy (destWord ← simDest3)

destWord ← t;

\* calculate actual destination bit related values

CalDstBit: t ← LDF[simDB, 14, 4];

destStart ← t;

destBitsToGo ← (ZERO) - t;

destBitsToGo ← (destBitsToGo) + (20C);

\* calculate number of bits to next word boundary (or end)

t ← destBitsToGo;

LU ← (srcBitsToGo) - (t);

GOTO[GetSrcBit, ALU < 0];

\* go to GetSrcBit if srcBitsToGo is less than destBitsToGo

```

t ← (destBitsToGo);          * Copy (nbits + destBitsToGo)
nbits ← t;
GOTO[ThenNBR];

GetSrcBit: t ← (srcBitsToGo);  * Copy (nbits + srcBitsToGo)
nbits ← t;

ThenNBR: t ← nbits;
LU ← (simNBR) + (t);
GOTO[SrcField, ALU < 0];
t ← (simNBR);                * Copy (nbits + -simNBR)
nbits ← (ZERO) - t;

* create source field descriptor
SrcField: t ← LSH[srcStart, 4];
srcFieldDescriptor ← t;
t ← (nbits) - 1;
srcFieldDescriptor ← (srcFieldDescriptor) OR t;

* load the source field
CYCLECONTROL ← srcFieldDescriptor;
t ← RF[srcWord];
srcWord ← t;

* create destination field descriptor
t ← LSH[destStart, 4];
destFieldDescriptor ← t;
t ← (nbits) - 1;
destFieldDescriptor ← (destFieldDescriptor) OR t;

* set up mask
allOnes ← (ZERO) - 1;
destMask ← 0C;
CYCLECONTROL ← destFieldDescriptor;
t ← WFA[allOnes];
destMask ← WFB[(destMask) OR (t)];

* align source with destination
tmpWord ← 0C;
CYCLECONTROL ← destFieldDescriptor;
t ← WFA[srcWord];
tmpWord ← WFB[(tmpWord) OR (t)];
t ← (tmpWord);                * Copy (srcWord + tmpWord)
srcWord ← t;

* set source background bits if required
LU ← (simMaskFillSrc);
GOTO[ClearDBit, ALU # 0];    * go to ClearDBit if source background bits do not
                             * need to be set
t ← (destMask);              * set source background bits
srcWord ← (srcWord) OR NOT t;

* clear destination bits if required
ClearDBit: LU ← (MaskDest);
GOTO[Perform, ALU = 0];     * go to Perform if destination bits do not need
                             * to be cleared
t ← (destMask);              * clear destination bits
destWord ← (destWord) AND NOT t;

Perform: t ← (srcWord);      * perform the operation
destWord ← (destWord) SALUFOP t;

* stuff result into correct destination register
t ← LDF[simDB, 12, 2];
tmpWord ← t;
LU ← (tmpWord);
GOTO[TrySimD1, ALU # 0];    * go to TrySimD1 if the correct dest. reg. is not
                             * simDest0
t ← (destWord);              * Copy (simDest0 + destWord)
simDest0 ← t;
GOTO[IncBitCnt];

TrySimD1: LU ← (tmpWord) - (1C);
GOTO[TrySimD2, ALU # 0];    * go to TrySimD2 if the correct dest. reg. is not
                             * simDest1
t ← (destWord);              * Copy (simDest1 + destWord)

```

```

simDest1 ← t;
GOTO[IncBitCnt];

TrySimD2:    LU ← (tmpWord) - (2C);
             GOTO[IsSimD3, ALU # 0]; * go to IsSimD3 if the correct dest. reg. is not simDest2
             t ← (destWord); * Copy (simDest2 ← destWord)
             simDest2 ← t;
             GOTO[IncBitCnt];

IsSimD3:    t ← (destWord); * Copy (simDest3 ← destWord)
            simDest3 ← t;

* increment various bit counters
IncBitCnt:  t ← (nbits);
            simSB ← (simSB) + t;
            simSB ← LDF[simSB, 12, 6];
            simDB ← (simDB) + t;
            simDB ← LDF[simDB, 12, 6];
            simNBR ← (simNBR) + t;

* decide if finished or not
            LU ← (simNBR);
            GOTO[CheckSB, ALU # 0]; * go to CheckSB if simulated bitblt MNBR did not finish
            simResult ← 1C, RETURN; * item done, return to main program

CheckSB:    LU ← (simSB);
            GOTO[CheckDB, ALU # 0]; * go to CheckDB if simulated bitblt SB did not finish
            LU ← (simDB);
            GOTO[SrcOut, ALU # 0]; * go to SrcOut if simulated bitblt DB did not finish
            simResult ← 2C, RETURN; * source and destination ran out, return to main program

SrcOut:     simResult ← 3C, RETURN; * source ran out, return to main program

CheckDB:    LU ← (simDB);
            GOTO[NotFinish, ALU # 0]; * go to NotFinish if simulated bitblt DB did not finish
            simResult ← 4C, RETURN; * destination ran out, return to main program

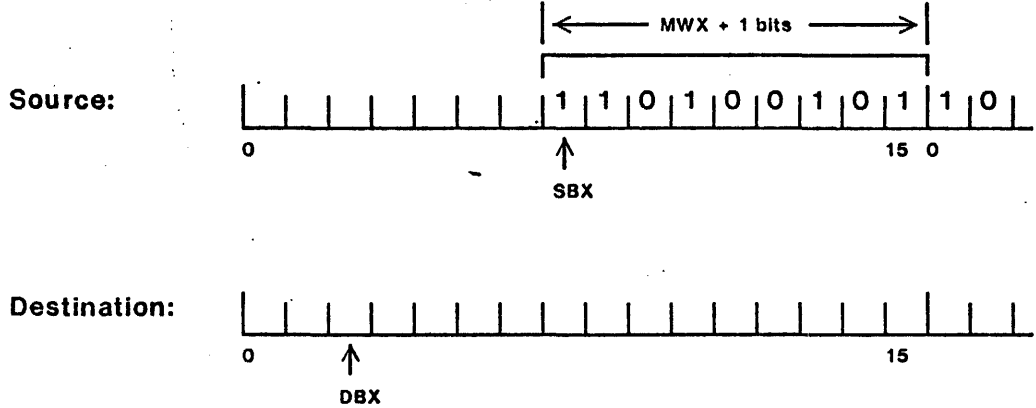
NotFinish:  GOTO[SimBltStart]; * not finished
            RETURN;

end; * to end the MAIN routine

```

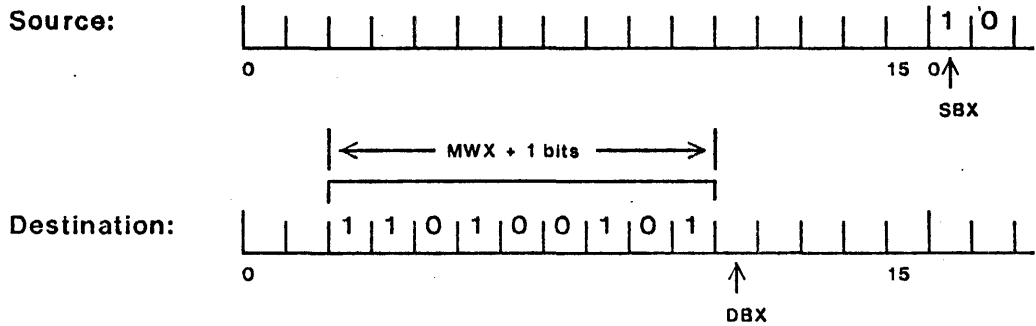
**Before BitBLT Loop:**

SBX = 7  
 DBX = 2  
 MWX = 8



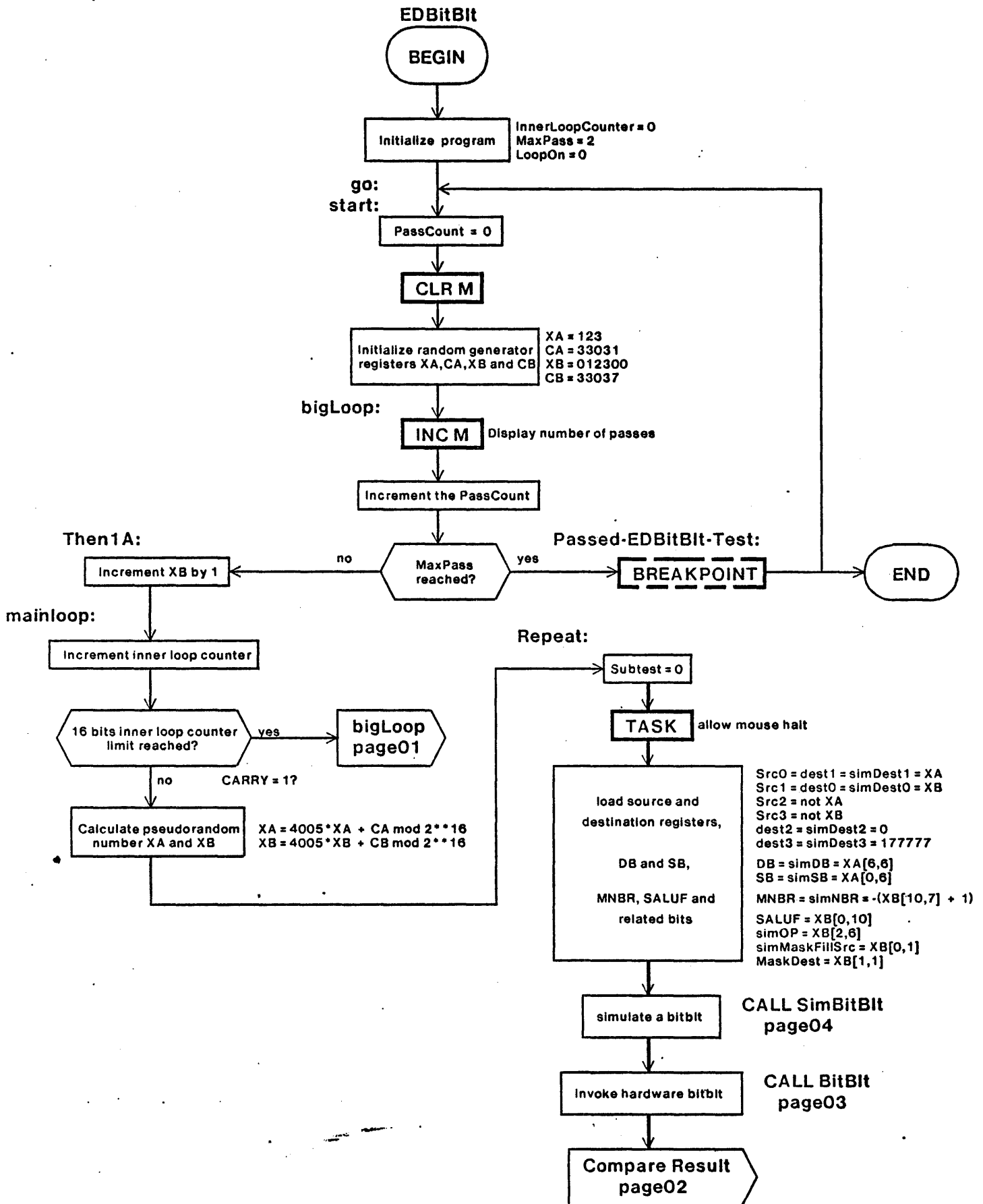
**After BitBLT Loop:**

SBX = 0  
 DBX = 11



The BitBLT inner loop transfers as much of one word as possible between the source and destination buffers. This number is the minimum of (1) the number of bits required to reach the next source word boundary, (2) the number of bits required to reach the next destination word boundary, and (3) the number of bits remaining in the current item (-MNBR). This quantity is calculated by PROMS from the registers SB, DB, and MNBR, and is loaded into the register MWX (precisely,  $MWX \leftarrow \min(\dots) - 1$ ) when BBFB or BBFBX is executed. The BBFA function ( $F1 = 0$ ), left-cycles and masks the source data (from R) in the cyclor masker. The cycle count is  $SBX - DBX$ , and the source mask extends from bits DBX to bit  $DBX + MWX$ . The diagram above illustrates the source and destination words and the values of SBX and DBX before and after a single iteration of the BitBLT loop.

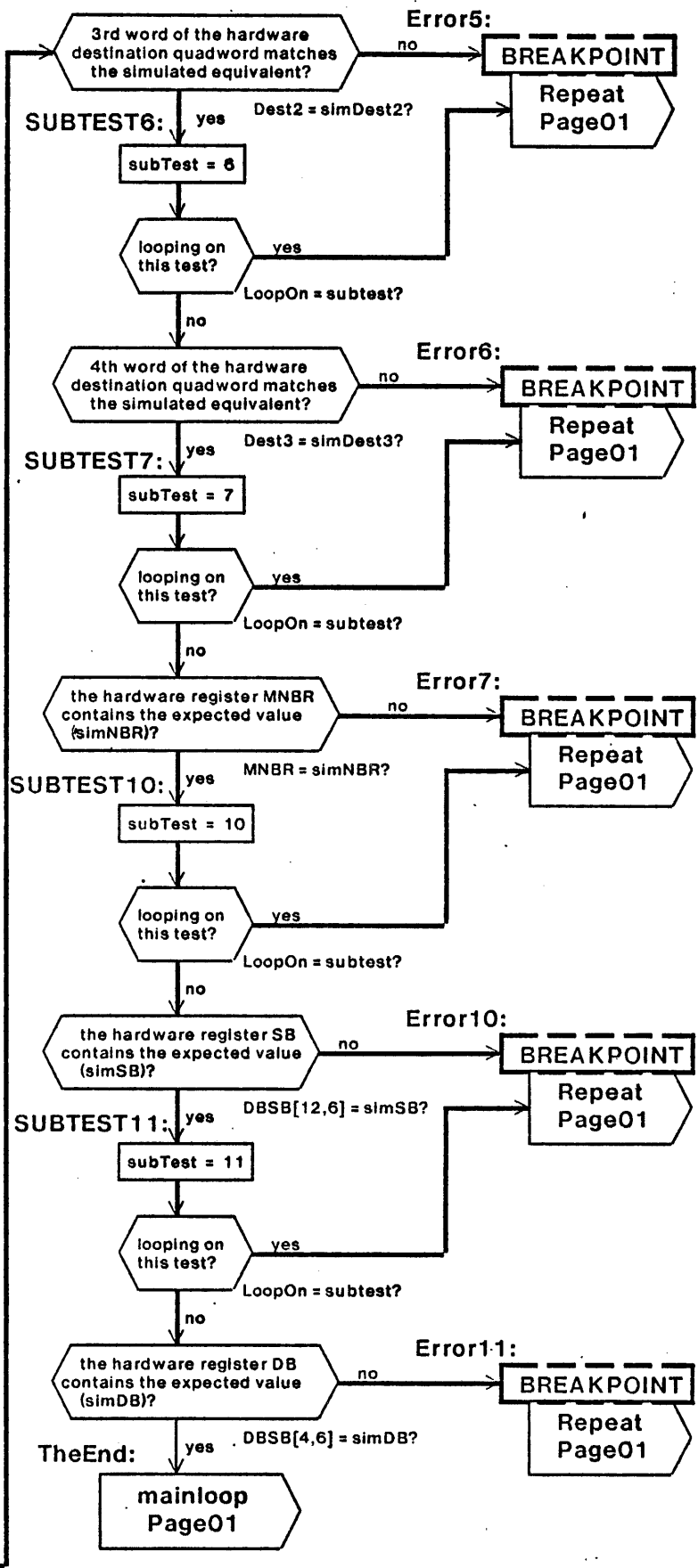
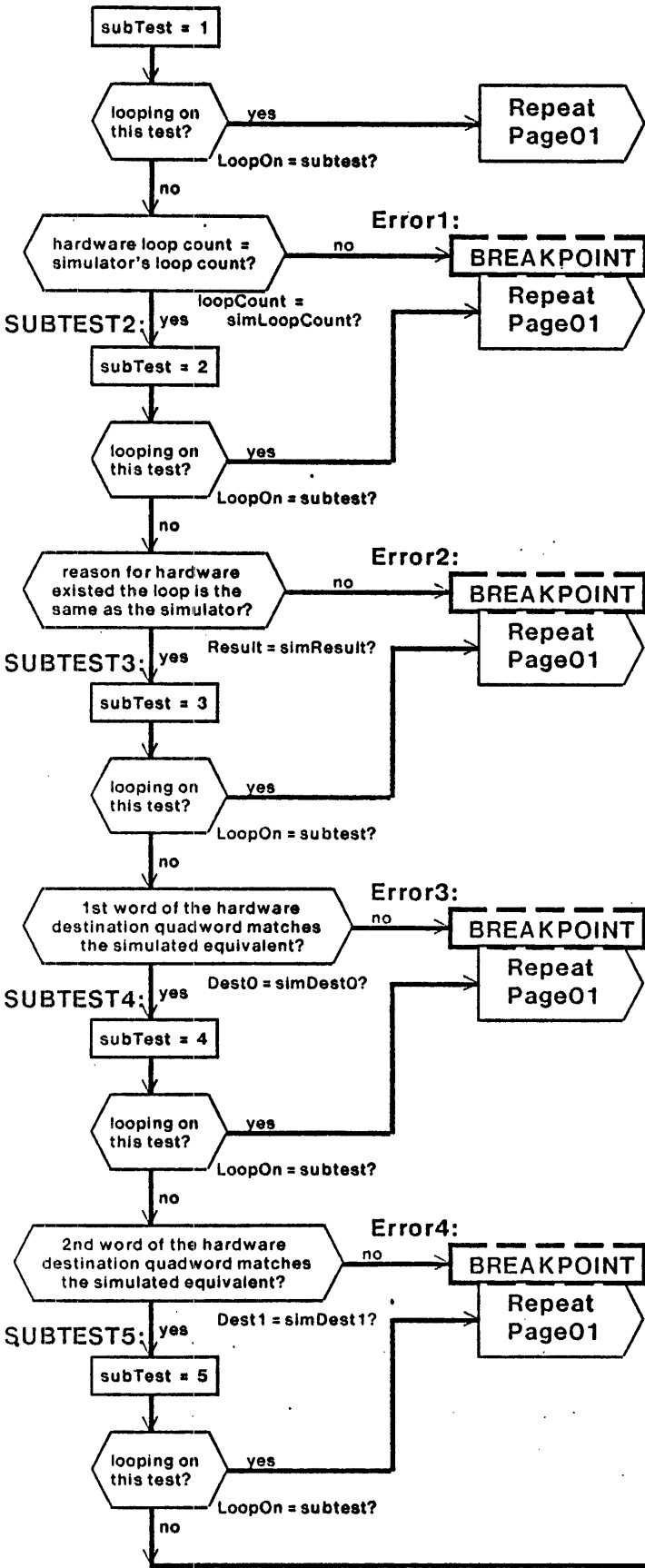
XEROX ED	D(0) Diagnostic	PROGRAM NAME EDBitBit	DOCUMENTATION FILE EDBitBit00.sil	DESIGNER Camellia Chan	REV 1	DATE 12/05/79	PAGE 00
-------------	--------------------	--------------------------	--------------------------------------	---------------------------	----------	------------------	------------





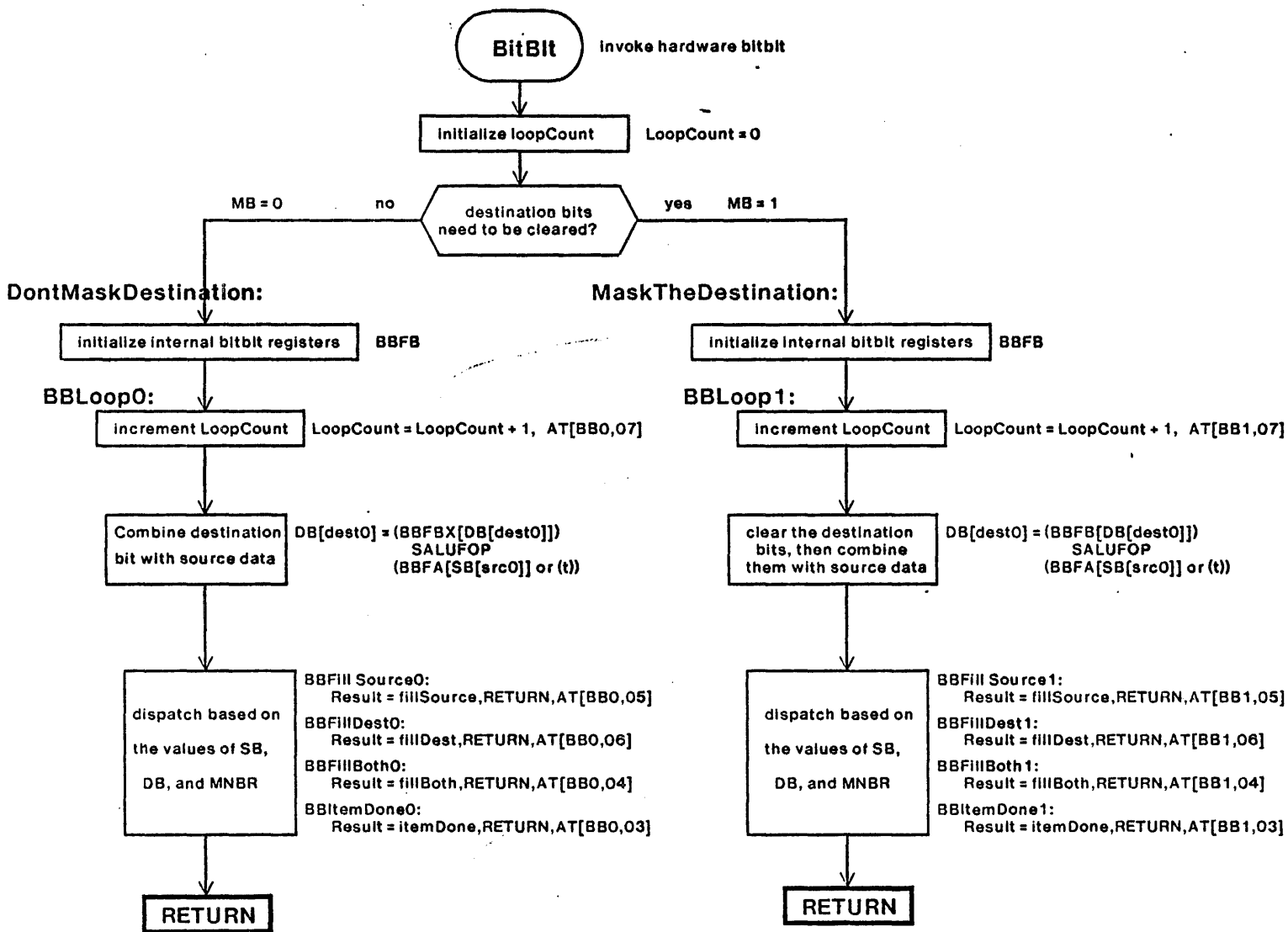
CompareResult:

SUBTEST1:

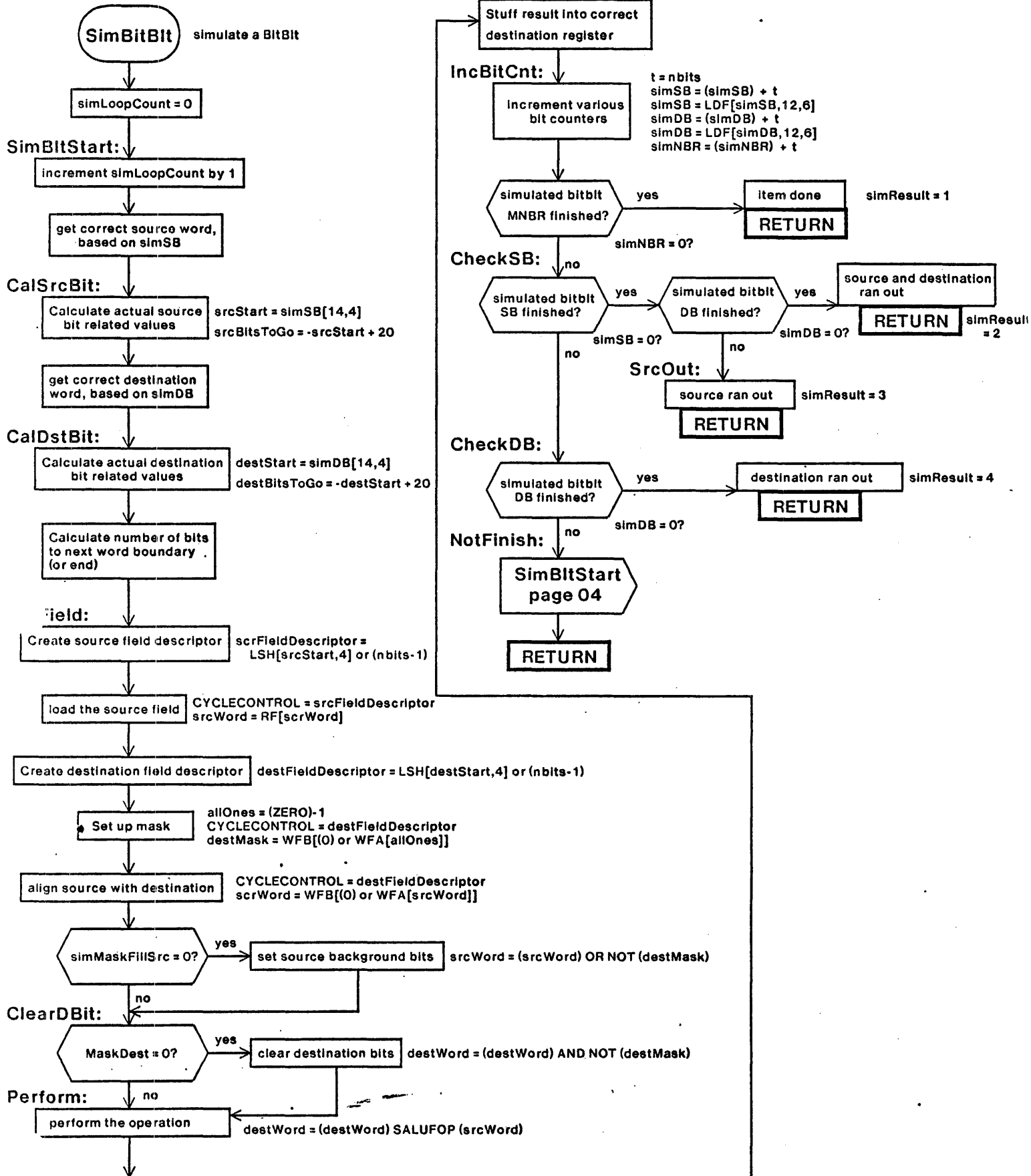


XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDBitBit	EDBitBit02.sil	Camellia Chan	1	12/05/79	02

**Subroutine**



**Subroutine**



XEROX ED	D(0) Diagnostic	PROGRAM NAME EDBitBit	DOCUMENTATION FILE EDBitBit04.sil	DESIGNER Camellia Chan	REV 1	DATE 12/05/79	PAGE 04
-------------	--------------------	--------------------------	--------------------------------------	---------------------------	----------	------------------	------------

PARITY	0	REVISION	1	COMM-ERR0	0
CYCLECONTROL	0	RUN-TIME	14	COMM-ERR1	0
PCXREG	4	PASSCOUNT	0	COMM-ERR2	0
PCFREG	4	MAXPASS	2	BOOT-ERR	0
DBREG	20	SUBTEST	0	*BOOTREASON	40
SBREG	77			MEMSYNOROME	7777
MNBR	3423	DEST0	0	-SBREG	77
*SSTKP	377	SIMDEST0	0	SIMSB	0
STKP	0	DEST1	0	DBREG	20
*ALURESULT	1	SIMDEST1	0	SIMDB	0
*SALUF	377	DEST2	0		
T 20	7000	SIMDEST2	0		
AATOVA	0	DEST3	0	SRC0	0
TPC 20	7777	SIMDEST3	0	SRC1	0
CALLER	ILC0+7320	LOOPCOUNT	0	SRC2	0
*PAGE	1	SIMLOOPCOUNT	0	SRC3	0
*APC	7011	RESULT	0		
*APCTASK	16	SIMRESULT	0		
*CIA	60+1	MNBR	3423	LOOPON	0
CTASK	0	SIMNBR	0		

Loaded: EDBITBLT

Time: 10.54

Step at 0:60, BP at 0:60+1

Exit Boot Run-Prog Read-Cmds Break UnBreak CtrAddedBPs CtrAllBPs ShowBPs Go  
 SS Continue Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

MicroD 8.6 (OS 16) of April 27, 1979  
at 12-Dec-79 14:20:40

microd.run edbitb1t

edbitb1t.DIB 457b instructions written 12-Dec-79 14:19:21

Total of 457b instructions

Checking for errors...

Linking...

Building allocation lists...

Assigning locations...

457b instructions in rings involving ONPAGE or AT

Reloading binaries...

Checking assignment...

Writing .MB file...

Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
edbitb1t.DIB:					
0	456	32020	101057	12	GO START
1	627	47	7054	2	(+1)
2	626	30005	107052	12	(+2)
3	625	30320	101050	12	(+3)
4	624	30001	123047	6	(+4)
5	623	30323	115044	6	(+5)
6	622	32014	101043	2	(+6)
7	621	32321	111041	2	(+7)
10	620	30001	137037	16	(+10)
11	617	30323	115006	14	(+11)
12	403	47	5035	2	BIGLOOP
13	616	33050	165033	12	(+1)
14	615	33450	25030	16	(+2)
15	614	50	24200	0	(+3)
16	b 401	50	25134	0	PASSED-EDBITBLT-TEST
17	400	33050	125027	2	THEN1A
20	613	33050	125025	6	MAINLOOP
21	612	50	24007	0	(+1)
22	402	30150	65022	12	(+2)
23	611	31174	45021	12	(+3)
24	610	31174	67017	12	(+4)
25	607	31150	65015	6	(+5)
26	606	30050	125013	12	(+6)
27	605	32150	65011	2	(+7)
30	604	33174	45006	2	(+10)
31	603	33174	67005	2	(+11)
32	602	31150	65003	16	(+12)
33	601	32050	125001	2	(+13)
34	600	36020	101131	0	REPEAT
35	454	50	25336	0	(+1)
36	457	30150	65400	10	(+2)
37	455	16050	125032	1	(+3)
40	515	16676	101031	11	(+4)
41	514	20050	125027	5	(+5)
42	513	10050	125024	5	(+6)
43	512	32150	65022	1	(+7)
44	511	16050	125021	5	(+10)
45	510	16676	101017	15	(+11)
46	507	20050	125015	1	(+12)
47	506	10050	125013	1	(+13)
50	505	20020	101010	11	(+14)
51	504	10020	101007	11	(+15)
52	503	21376	101004	15	(+16)
53	502	11376	101002	15	(+17)
54	501	30164	71001	11	(+20)
55	500	26050	125177	4	(+21)
56	477	26150	15174	4	(+22)
57	476	30164	55173	10	(+23)
60	475	14050	125171	14	(+24)
61	474	14150	13166	14	(+25)

62	473	33065	63165	0	(+26)
63	472	13476	101163	10	(+27)
64	471	12150	27160	10	(+30)
65	470	32166	67157	0	(+31)
66	467	50	23154	0	(+32)
67	466	32164	61153	0	(+33)
70	466	12060	125150	14	(+34)
71	464	32160	41147	0	(+35)
72	463	12050	125145	4	(+36)
73	462	32160	43143	0	(+37)
74	461	24050	125140	10	(+40)
75	460	45	5125	0	(+41)
76	452	50	25327	0	(+42)
77	453	45	5120	0	(+43)
100	450	50	25311	0	(+44)
101	451	36000	103040	1	COMPARERESULT SUBTEST1
102	520	36150	65037	1	(+1)
103	517	35450	25035	15	(+2)
104	516	50	24111	0	(+3)
106	444	50	25001	2	(+4)
108	445	12150	65052	1	(+5)
107	525	25450	25050	5	(+8)
110	524	50	24114	0	(+7)
111	b 447	50	25001	2	ERROR1
112	446	36000	105046	1	SUBTEST2
113	523	36150	65045	1	(+1)
114	522	35450	25042	15	(+2)
115	521	50	24100	0	(+3)
116	440	50	25001	2	(+4)
117	441	14150	65066	11	(+5)
120	532	27450	25063	1	(+6)
121	531	50	24105	0	(+7)
122	b 443	50	25001	2	ERROR2
123	442	36000	107060	1	SUBTEST3
124	530	36150	65057	1	(+1)
125	527	35450	25055	15	(+2)
126	526	50	24070	0	(+3)
127	434	50	25001	2	(+4)
130	435	10150	65077	1	(+5)
131	537	21450	25074	1	(+6)
132	536	50	24075	0	(+7)
133	b 437	50	25001	2	ERROR3
134	436	36000	111073	1	SUBTEST4
135	535	36150	65070	1	(+1)
136	534	35450	25066	15	(+2)
137	533	50	24061	0	(+3)
140	430	50	25001	2	(+4)
141	431	10150	65111	5	(+5)
142	544	21450	25106	5	(+6)
143	543	50	24064	0	(+7)
144	b 433	50	25001	2	ERROR4
145	432	36000	113104	1	SUBTEST5
146	542	36150	65103	1	(+1)
147	541	35450	25101	15	(+2)
150	540	50	24051	0	(+3)
151	424	50	25001	2	(+4)
152	425	10150	65122	11	(+5)
153	551	21450	25121	11	(+6)
154	550	50	24054	0	(+7)
155	b 427	50	25001	2	ERROR5
156	426	36000	115116	1	SUBTEST6
157	547	36150	65114	1	(+1)
160	546	35450	25113	15	(+2)
161	545	50	24040	0	(+3)
162	420	50	25001	2	(+4)
163	421	10150	65134	15	(+5)
164	556	21450	25132	15	(+6)
165	555	50	24045	0	(+7)
166	b 423	50	25001	2	ERROR6
167	422	36000	117130	1	SUBTEST7
170	554	36150	65127	1	(+1)
171	553	35450	25125	15	(+2)
172	552	50	24031	0	(+3)
173	414	50	25001	2	(+4)
174	415	66150	41147	15	(+5)
175	563	13450	25146	11	(+6)

176	562	50	24034	0	(+7)
177	b 417	50	25001	2	ERROR7
200	416	36000	121142	1	SUBTEST10
201	561	36150	65141	1	(+1)
202	560	35460	25137	15	(+2)
203	557	50	24016	0	(+3)
204	406	50	25001	2	(+4)
205	407	64150	41176	15	(+5)
206	577	2050	125174	1	(+6)
207	576	2165	41173	1	(+7)
210	575	15450	25171	15	(+10)
211	574	50	24025	0	(+11)
212	b 413	50	25001	2	ERROR10
213	412	36000	123167	1	SUBTEST11
214	573	36150	65165	1	(+1)
215	572	35460	25162	15	(+2)
216	571	50	24020	0	(+3)
217	410	50	25001	2	(+4)
220	411	64150	41161	15	(+5)
221	570	2050	125156	1	(+6)
222	567	2164	65155	1	(+7)
223	566	27450	25153	5	(+10)
224	565	50	24010	0	(+11)
225	b 405	50	25001	2	ERROR11
226	404	45	3150	1	THEEND
227	564	50	25026	2	(+1)
230	1044	24020	101122	4	BITBLT
231	1051	50	24706	0	(+1)
232	1042	52	25017	2	DONTMASKDESTINATION
233	@1207	25050	125114	4	BBLOOP0
234	1046	56340	41112	4	(+1)
235	1045	61755	125616	12	(+2)
236	@1205	26000	107401	0	BBFILLSOURCE0
237	@1206	26000	111400	0	BBFILLDEST0
240	@1204	26000	105400	0	BBFILLBOTH0
241	@1203	26000	103400	0	BBITEMDONE0
242	1043	52	25016	3	MASKTHEDESTINATION
243	@1307	25050	125121	4	BBLOOP1
244	1050	56340	41117	4	(+1)
245	1047	61752	125616	13	(+2)
246	@1305	26000	107401	0	BBFILLSOURCE1
247	@1306	26000	111400	0	BBFILLDEST1
250	@1304	26000	105400	0	BBFILLBOTH1
251	@1303	26000	103400	0	BBITEMDONE1
252	1052	50	25401	0	(+1)
253	1053	12020	101040	2	SIMBITBLT
254	1220	13050	125037	2	SIMBLTSTART
255	1217	14161	65035	16	(+1)
256	1216	2050	125033	6	(+2)
257	1215	2150	25030	6	(+3)
260	1214	50	24100	0	(+4)
261	1040	16150	65027	2	(+5)
262	1213	50	125024	16	(+6)
263	1212	50	25023	2	(+7)
264	1041	3400	3135	4	TRYSRC1
265	1056	50	24075	0	(+1)
266	1036	16150	65132	4	(+2)
267	1055	50	125130	14	(+3)
270	1054	50	25023	2	(+4)
271	1037	3400	5142	4	TRYSRC2
272	1061	50	24070	0	(+1)
273	1034	16150	65140	10	(+2)
274	1060	50	125136	14	(+3)
275	1057	50	25023	2	(+4)
276	1035	16150	65144	14	ISSRC3
277	1062	50	125022	16	(+1)
300	1211	14163	63020	16	CALSRCBIT
301	1210	50	125004	12	(+1)
302	1202	1476	101002	2	(+2)
303	1201	1101	101001	2	(+3)
304	1200	26161	65176	5	(+4)
305	1177	2050	125175	5	(+5)
306	1176	2150	25173	5	(+6)
307	1175	50	24064	0	(+7)
310	1032	10150	65170	1	(+10)
311	1174	24050	125167	1	(+11)

312	1173	50	25165	1	(+12)
313	1033	3400	3153	4	TRYDEST1
314	1065	50	24061	0	(+1)
315	1030	10150	65151	4	(+2)
316	1064	24050	125147	0	(+3)
317	1063	50	25165	1	(+4)
320	1031	3400	5160	4	TRYDEST2
321	1070	50	24054	0	(+1)
322	1026	10150	65157	10	(+2)
323	1067	24050	125155	0	(+3)
324	1066	50	25165	1	(+4)
325	1027	10150	65163	14	ISDEST3
326	1071	24050	125164	1	(+1)
327	1172	26163	63163	5	CALDSTBIT
330	1171	22050	125161	15	(+1)
331	1170	23476	101156	1	(+2)
332	1167	23101	101155	1	(+3)
333	1166	22150	65153	1	(+4)
334	1165	1450	25150	1	(+5)
335	1164	50	24250	0	(+6)
336	1024	22150	65147	1	(+7)
337	1163	24050	125145	15	(+10)
340	1162	50	25142	1	(+11)
341	1025	150	65164	0	GETSRCBIT
342	1072	24050	125143	15	(+1)
343	1161	24150	65141	15	THENNBR
344	1160	13150	25137	11	(+1)
345	1157	50	24205	0	(+2)
346	1002	12150	65134	11	(+3)
347	1156	25476	101006	14	(+4)
350	1003	174	51133	11	SRCFIELD
351	1155	50	125130	5	(+1)
352	1154	25350	65127	15	(+2)
353	1153	350	125124	5	(+3)
354	1152	150	11123	5	(+4)
355	1151	154	65120	15	(+5)
356	1150	50	125116	15	(+6)
357	1147	22174	51115	15	(+7)
360	1146	22050	125112	5	(+10)
361	1145	25350	65111	15	(+11)
362	1144	22350	125106	5	(+12)
363	1143	37376	101105	5	(+13)
364	1142	22020	101102	11	(+14)
365	1141	22150	11101	5	(+15)
366	1140	36151	65076	5	(+16)
367	1137	22353	125075	11	(+17)
370	1136	2020	101073	5	(+20)
371	1135	22150	11071	5	(+21)
372	1134	151	65066	15	(+22)
373	1133	2353	125065	5	(+23)
374	1132	2150	65062	5	(+24)
375	1131	50	125060	15	(+25)
376	1130	12150	25057	5	(+26)
377	1127	50	24010	0	(+27)
400	1004	22150	65055	11	(+30)
401	1126	650	125013	14	(+31)
402	1005	24150	25052	11	CLEARDBIT
403	1125	50	24045	0	(+1)
404	1023	22150	65166	10	(+2)
405	1073	24550	125045	0	(+3)
406	1022	150	65051	15	PERFORM
407	1124	25750	125047	1	(+1)
410	1123	26161	65044	5	(+2)
411	1122	2050	125042	5	(+3)
412	1121	2150	25041	5	(+4)
413	1120	50	24040	0	(+5)
414	1020	24150	65037	1	(+6)
415	1117	10050	125034	1	(+7)
416	1116	50	25032	1	(+10)
417	1021	3400	3174	4	TRYSIMD1
420	1076	50	24034	0	(+1)
421	1016	24150	65173	0	(+2)
422	1075	10050	125171	4	(+3)
423	1074	50	25032	1	(+4)
424	1017	3400	5003	5	TRYSIMD2
425	1101	50	24031	0	(+1)



426	1014	24150	65001	1	(+2)
427	1100	10050	125177	10	(+3)
430	1077	50	25032	1	(+4)
431	1015	24150	65004	1	ISSIMD3
432	1102	10050	125032	15	(+1)
433	1115	24150	65031	15	INCBITCNT
434	1114	15150	125027	15	(+1)
435	1113	14166	101025	15	(+2)
436	1112	27150	125022	5	(+3)
437	1111	26166	101020	5	(+4)
440	1110	13150	125016	11	(+5)
441	1107	12150	25015	11	(+6)
442	1106	50	24026	0	(+7)
443	1012	14000	103400	10	(+10)
444	1013	14150	25012	15	CHECKSB
445	1105	50	24015	0	(+1)
446	1006	26150	25006	5	(+2)
447	1103	50	24020	0	(+3)
450	1010	14000	105400	10	(+4)
451	1011	14000	107401	10	SRCOUT
452	1007	26150	25011	5	CHECKDB
453	1104	50	24001	0	(+1)
454	1000	14000	111400	10	(+2)
455	1001	50	25040	2	NOTFINISH
456	1221	50	25401	0	(+1)

Page 400: 230 locations used, 150 free

Page 1000: 227 locations used, 151 free

RM:

1	CA
2	XA
3	CB
4	XB
5	0 INNERLOOPCOUNTER
6	PASSCOUNT
7	2 MAXPASS
13	0 LOOPON
14	SUBTEST
15	ALLONES
20	DEST0
21	DEST1
22	DEST2
23	DEST3
24	DESTBITSTOGO
25	DESTFIELDDESRIPTOR
26	DESTMASK
27	DESTSTART
30	DESTWORD
31	LOOPCOUNT
32	MASKDEST
33	NBITS
34	RESULT
35	SIMDB
40	SIMDEST0
41	SIMDEST1
42	SIMDEST2
43	SIMDEST3
44	SIMLOOPCOUNT
45	SIMMASKFILLSRC
46	SIMNBR
47	SIMOP
52	SIMRESULT
53	SIMSB
54	SRC0
55	SRC1
56	SRC2
57	SRC3
60	SRCBITSTOGO
61	SRCFIELDDESRIPTOR
62	SRCSTART
63	SRCWORD
64	TMP
65	TMPWORD
66	1 REVISION

67 14 RUN-TIME  
70 RLC@

Time: 11 seconds; 0 error(s), 0 warning(s), 11504 words free

```
.....  
;;:EDBitBlTLog.MIDAS : Logger for EDBitBlT program  
;;:                               By: C. Iseng                               Dec. 6, 1979  
.....
```

```
.start      L X AppendOutput EDBitBlT.report;  
            L X WriteMessage ~***** START EDBitBlT Test : ;  
            L X WriteDT;  
            L X WriteMessage *****~ ;  
            L X Skip .continue;  
  
.breakpoint L X AppendOutput EDBitBlT.report;  
            L A18 SkipNE ERROR1;  
            L X Skip .error1;  
            L A18 SkipNE ERROR2;  
            L X Skip .error2;  
            L A18 SkipNE ERROR3;  
            L X Skip .error3;  
            L A18 SkipNE ERROR4;  
            L X Skip .error4;  
            L A18 SkipNE ERROR5;  
            L X Skip .error5;  
            L A18 SkipNE ERROR6;  
            L X Skip .error6;  
            L A18 SkipNE ERROR7;  
            L X Skip .error7;  
            L A18 SkipNE ERROR10;  
            L X Skip .error10;  
            L A18 SkipNE ERROR11;  
            L X Skip .error11;  
            L A18 SkipNE PASSED-EDBITBLT-TEST;  
            L X Skip .passtest;  
  
.notmybreak L X AppendOutput EDBitBlT.report;  
            L X WriteMessage *** FAILED: Not at my breakpoint ~;  
  
            L X WriteMessage ' Parity = ;  
            R A0 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CIA = ;  
            R A18 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CTASK = ;  
            R A19 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APCTASK = ;  
            R A17 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APC = ;  
            R A16 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' TPC = ;  
            R A13 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X CloseOutput;  
            L X Exit;  
  
.error1     L X WriteMessage *** FAILED: at my Breakpoint ~;  
            L X WriteMessage * LoopCount does not equal to SimLoopCount ~;  
            L X WriteMessage ' LoopCount = ;  
            R B14 Val;  
            L X WriteMessage;
```

```
L X WriteMessage ~;
L X WriteMessage ' SimLoopCount =      ;
R B15 Val;
L X WriteMessage;
L X WriteMessage ~;

.bad      L X WriteMessage ' SUBTEST =      ;
          R B4 Val;
          L X WriteMessage;
          L X WriteMessage ~;

          L X WriteMessage ' PASSCOUNT =    ;
          R B2 Val;
          L X WriteMessage;
          L X WriteMessage ~;

          L X Skip .continue;

.error2   L X WriteMessage *** FAILED: at my Breakpoint ~;
          L X WriteMessage *          Result does not equal to SimResult ~;
          L X WriteMessage ' Result =      ;
          R B16 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X WriteMessage ' SimResult =    ;
          R B17 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X BackSkip .bad;

.error3   L X WriteMessage *** FAILED: at my Breakpoint ~;
          L X WriteMessage *          Dest0 does not equal to SimDest0 ~;
          L X WriteMessage ' Dest0 =      ;
          R B6 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X WriteMessage ' SimDest0 =    ;
          R B7 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X BackSkip .bad;

.error4   L X WriteMessage *** FAILED: at my Breakpoint ~;
          L X WriteMessage *          Dest1 does not equal to SimDest1 ~;
          L X WriteMessage ' Dest1 =      ;
          R B8 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X WriteMessage ' SimDest1 =    ;
          R B9 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X BackSkip .bad;

.error5   L X WriteMessage *** FAILED: at my Breakpoint ~;
          L X WriteMessage *          Dest2 does not equal to SimDest2 ~;
          L X WriteMessage ' Dest2 =      ;
          R B10 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X WriteMessage ' SimDest2 =    ;
          R B11 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X BackSkip .bad;

.error6   L X WriteMessage *** FAILED: at my Breakpoint ~;
          L X WriteMessage *          Dest3 does not equal to SimDest3 ~;
          L X WriteMessage ' Dest3 =      ;
          R B12 Val;
          L X WriteMessage;
          L X WriteMessage ~;
          L X WriteMessage ' SimDest3 =    ;
          R B13 Val;
          L X WriteMessage;
          L X WriteMessage ~;
```

```
L X BackSkip .bad;

.error7 L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage * MNBR does not equal to SimNBR ~;
L X WriteMessage ' MNBR = ;
R B18 Val;
L X WriteMessage;
L X WriteMessage ~;
L X WriteMessage ' SimNBR = ;
R B19 Val;
L X WriteMessage;
L X WriteMessage ~;
L X BackSkip .bad;

.error10 L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage * SB does not equal to SimSB ~;
L X WriteMessage ' SB = ;
R C6 Val;
L X WriteMessage;
L X WriteMessage ~;
L X WriteMessage ' SimSB = ;
R C7 Val;
L X WriteMessage;
L X WriteMessage ~;
L X BackSkip .bad;

.error11 L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage * DB does not equal to SimDB ~;
L X WriteMessage ' DB = ;
R C8 Val;
L X WriteMessage;
L X WriteMessage ~;
L X WriteMessage ' SimDB = ;
R C9 Val;
L X WriteMessage;
L X WriteMessage ~;
L X BackSkip .bad;

.passtest L X WriteMessage ~----- PASSEd EDBitBlT Test : ;
L X WriteDT;
L X WriteMessage ----- ;
L X Skip .continue;

.continue L X WriteMessage ~;
L X CloseOutput;
L X DisplayOn;
L X Confirm;
L X TimeOut 10000000;
L X Continue;
L X Skip 2;
L X ShowError Program failed to CONTINUE.;
L X BackSkip .notmybreak;
L X DisplayOff;
L X BackSkip .breakpoint;
```

```
I A19 Val 0
I X Confirm
L X Load EDBITBLT;

L B0 Addr REVISION;
L B1 Addr RUN-TIME;
L B2 Addr PASSCOUNT;
L B3 Addr MAXPASS;
L B4 Addr SUBTEST;
L B6 Addr DEST0;
L B7 Addr SIMDEST0;
L B8 Addr DEST1;
L B9 Addr SIMDEST1;
L B10 Addr DEST2;
L B11 Addr SIMDEST2;
L B12 Addr DEST3;
L B13 Addr SIMDEST3;
L B14 Addr LOOPCOUNT;
L B15 Addr SIMLOOPCOUNT;
L B16 Addr RESULT;
L B17 Addr SIMRESULT;
L B18 Addr MNBR;
L B19 Addr SIMNBR;

L C6 Addr SBREG;
L C7 Addr SIMSB;
L C8 Addr DBREG;
L C9 Addr SIMDB;
L C12 Addr SRC0;
L C13 Addr SRC1;
L C14 Addr SRC2;
L C15 Addr SRC3;
L C18 Addr LOOPON;
L X DisplayOn;
L X TimeOut 10000
L X SS GO
L X Skip 1
L X ShowError Single-step at GO hung
```

%

\*\*\* \*\*>Rev-1>EDCSEX.mc Revision 1 Nov 13,1979 \*\*\* \*\*>

```

*****
*** EDCSEX.mc : Control Store Exerciser microcode
*** Purpose : This test exercises the control store as a 4K x 36 bit memory.
               Only locations not occupied by the program or the kernel are exercised.
*** Minimum Hardware : Standard 4 CPU boards.
*** Approximate Run Time : 30 seconds.
*** Written by : C. Thacker, Dec. 12, 1978
               Added link saving to writeCS and readCSX to save link
               smashed by control store operations.
*** Modified by : T. Henning, Oct. 5, 1979
               Standardize title page and code format.
*** Modified by : T. Henning, Oct. 15, 1979
               Implement looping and standard labels.
*** Modified by : T. Henning, Oct. 24, 1979
               Added capability for all zeros, all ones, and checker patterns.
*****

```

```

*****
*SubTest Description:
* SubTest 0: writes a data pattern into each location, reads the contents back
              and compares them with what was written.
* SubTest 1: reads each location and compares them with what was written.
*****

```

```

*****
*BreakPoints:
* WORD0BAD: Word 0 of the Control Store read did not match word 0 written in SubTest0.
* WORD1BAD: Word 1 of the Control Store read did not match word 1 written in SubTest0.
* WORD2BAD: Word 2 of the Control Store read did not match word 2 written in SubTest0.
* WORD0BADREAD: Word 0 of the Control Store read did not match word 0 in SubTest1.
* WORD1BADREAD: Word 1 of the Control Store read did not match word 1 in SubTest1.
* WORD2BADREAD: Word 2 of the Control Store read did not match word 2 in SubTest1.
* Passed-EDCSEX-Test: Passed all tests, and all passes.
*****

```

```

*****
* ShortLoop Logic Analyzer Sync Points at Control Store address:
* WORD0BAD: Control Store address 424 at TESTLOOP0.
* WORD1BAD: Control Store address 424 at TESTLOOP0.
* WORD2BAD: Control Store address 424 at TESTLOOP0.
* WORD0BADREAD: Control Store address 464 at TESTLOOP1.
* WORD1BADREAD: Control Store address 464 at TESTLOOP1.
* WORD2BADREAD: Control Store address 464 at TESTLOOP1.
*****

```

\*\*\*\*\*

\*Special Reg. Definition:

- \* ShortLoop: At any breakpoint, the user has the choice of setting ShortLoop to a 1 to loop on the current test. During the short loop, the user can modify the address and data to the Control Store at will by changing CurrentLoc and Wd0, Wd1, and Wd2.

- 1, the current test will loop repeatedly for trouble shooting
  - 0, no looping in current test

\* PatternChoice:

- Bit 15 - all zeros pattern, enable by 1, disable by 0
- Bit 14 - all ones pattern, enable by 1, disable by 0
- Bit 13 - checker pattern, enable by 1, disable by 0
- Bit 12 - random pattern, enable by 1, disable by 0

Example: PatternChoice=1 enables the all zeros pattern only  
 PatternChoice=2 enables the all ones pattern only  
 PatternChoice=4 enables the checker pattern only  
 PatternChoice=10 enables the random pattern only  
 PatternChoice=17 enables all four of the patterns  
 PatternChoice=11 enables the random and all zeros patterns

\* LoopControl: bit 0 & 15 are used to control the type of looping

bit 0	bit 15	
0	0	write/read VARYing data at ONE address
0	1	write/read VARYing data at ALL address
1	0	write & check CONSTANT data at ONE address
1	1	write/read CONSTANT data at ALL address

Note that if the address is not allowed to increment, the program will stay in the write loop forever. Also, during the read loop, it is not sensible to have LoopControl.0 =1, since a different value was written into each cell during the write pass.

\*\*\*\*\*

\*Subroutine Description:

- \* FillBuf: places 2 16-bit patterns in Wd0-1, and a 4 bit pattern in Wd2. The pattern may be all zeros, all ones, checker, or random, depending on the value of CurrentPattern.
- \* Rand: produces a pseudo-random number in t & xa registers by  $Xa \leftarrow 4005 * Xa + Ca \text{ mod } 2^{**16}$ .
- \* ReadCSX: reads control store location CurrentLoc into Rd0-2.
- \* WriteCS: writes 3 words from Wd0-2 to the control store at CurrentLoc.

\*\*\*\*\*

%



\*\*\*\*\*  
\*INITIALIZATION:

```
BUILTIN[INSERT,24];
INSERT[d0lang];
TITLE[Control Store Exerciser];
SET[ProgPage,1];      *program is one page long
ONPAGE[ProgPage];
```

## \*\*\*\*\* R-Registers: \*\*\*\*\*

```
RV[LowLoc,20,ADD[LSHIFT[ProgPage,10],400]]; *first location tested - one beyond program page
RV[CurrentLoc,21];      *the current Control Store location involved
RV[HighLoc,22,6777];   *last location tested - below kernel

RV[Rd0,23];            *3 word read buffer
RV[Rd1,24];
RV[Rd2,25];

RV[Wd0,26,177777];     *3 word write buffer
RV[Wd1,27,177777];
RV[Wd2,30,17];

RV[Md0,31,177777];     *3 word mask - 1's mean compare the bit
RV[Md1,32,177777];
RV[Md2,33,17];

RV[Rlink0,34];        *subroutine return link

RV[Xa,35,123];        *Random Number Generator (RNG) registers
RV[Ca,36,33031];
RV[SavedXa,37];

RV[PassCount,40,0];
RV[MaxPass,41,100];
RV[SubTest,42];

RV[ShortLoop,43,0];   *Disable looping at program initialization
RV[LoopControl,44,1];

RV[PatternChoice,45,17]; *Enable all four pattern at program initialization
RV[PatternTry,46,1];   *Initialize to all zeros pattern
RV[CurrentPattern,47,1]; *Initialize to all zeros pattern
RV[Ones,50,177777];   *define ones to be 177777
RV[Checker1,51,125252]; *checker pattern register
RV[Checker0,52,052525]; *checker pattern register
RV[Toggle,53,0];     *checker toggle register

RV[Revision,54,1];    *REVISION 1
RV[Run-Time,55,36];  *Run-Time is 36b or 30D seconds
```

\*\*\*\*\*

\*\*\* MAIN routine:

```

start:
go:    t←Xa;          *save RNG
      SavedXa←t;
      t←LowLoc;     *set address
      CurrentLoc←t;

Again: t ← (PatternTry) AND (177760C); *what pattern to use?
      goto[WhatPattern,alu=0];        *exhausted all four pattern types?
      PatternTry ← 1C;                *yes, select the zero pattern again
      call[Rand];                     *process the (not very good) RNG
      Toggle ← 0C;                    *reset checker pattern toggle
      PassCount← t ←(PassCount)+1;    *increment pass count
      lu ← (MaxPass) - (t);
      goto[. +2, alu>=0];             *finished all passes?
Passed-EDCSEX-Test: PassCount ← 0C, goto[start], breakpoint;
      nop;
WhatPattern: t ← PatternChoice;      *determine what pattern to use
            t ← (PatternTry) AND (t);
            goto[NextPattern,alu=0]; *do we want to use this pattern?
ThisPattern: CurrentPattern ← t, goto[. +2]; *yes, use this pattern
NextPattern: PatternTry ← LSH[PatternTry,1], goto[Again]; *no, try the next pattern

*SUBTEST 0
      SubTest ← 0C;

writeloo0: Call[FillBuf];
writeloo1:  nop;

TestLoop0: call[WriteCS];            *write data into Control Store
          call[ReadCSX];             *read Control Store and store it in Rd0-2
          t←Wd0, TASK;               *allow mouse halt
          t←(Rd0) xor (t);           *compare Rd0 with Wd0 under mask Md0
          ShortLoop ← ShortLoop, goto[. +2, R EVEN]; *test for ShortLoop option
          goto[TestLoop0];          *short loop selected
          t←(Md0) and (t);
          goto[. +2, alu=0];

word0bad: breakpoint;

          t←Wd1;                     *compare Rd1 with Wd1 under mask Md1
          t←(Rd1) xor (t);
          ShortLoop ← ShortLoop, goto[. +2, R EVEN]; *test for ShortLoop option
          goto[TestLoop0];          *short loop selected
          t←(Md1) and (t);
          goto[. +2, alu=0];

word1bad: breakpoint;

          t←Wd2; *compare Rd2 with Wd2 under mask Md2
          t←(Rd2) xor (t);
          ShortLoop ← ShortLoop, goto[. +2, R EVEN]; *test for ShortLoop option
          goto[TestLoop0];          *short loop selected
          t←(Md2) and (t);
          goto[. +2, alu=0];

word2bad: breakpoint;
          ShortLoop ← ShortLoop, goto[. +2, R EVEN]; *test for ShortLoop option
          goto[TestLoop0];          *short loop selected

      t ← (LoopControl) and (1C);
      CurrentLoc ← t ← (CurrentLoc)+(t); *done?
      lu←(HighLoc)-t;
      goto[. +2, alu<0];
      lu ← LoopControl, dblgoto[writeloo1,writeloo0,R<0];
                                          *do not change data if LoopControl.0=1

      t←LowLoc; *reset address
      CurrentLoc ← t;
      t←SavedXa; *reset RNG
      Xa←t;
      Toggle ← 0C; *reset checker pattern toggle

```

## \*SUBTEST 1

```

SubTest ← 1C;

readloop0: call[FillBuf];          *fill buffer
readloop1:  nop;                  *odd-even placement constraints

TestLoop1:  call[ReadCSX];        *read Control Store and store it in Rd0-2
           t←Wd0, TASK;          *allow mouse halt
           t←(Rd0) xor (t);      *compare Rd0 with Wd0 under mask Md0
           ShortLoop ← ShortLoop, goto[.+2,R EVEN]; *test for ShortLoop option
           goto[TestLoop1];      *short loop selected
           t←(Md0) and (t);
           goto[.+2,alu=0];
word0badRead: breakpoint;

           t←Wd1;                *compare Rd1 with Wd1 under mask Md1
           t←(Rd1) xor (t);
           ShortLoop ← ShortLoop, goto[.+2,R EVEN]; *test for ShortLoop option
           goto[TestLoop1];      *short loop selected
           t←(Md1) and (t);
           goto[.+2,alu=0];
word1badRead: breakpoint;

           t←Wd2;                *compare Rd2 with Wd2 under mask Md2
           t←(Rd2) xor (t);
           ShortLoop ← ShortLoop, goto[.+2,R EVEN]; *test for ShortLoop option
           goto[TestLoop1];      *short loop selected
           t←(Md2) and (t);
           goto[.+2,alu=0];
word2badRead: breakpoint;
           ShortLoop ← ShortLoop, goto[.+2,R EVEN]; *test for ShortLoop option
           goto[TestLoop1]; *short loop selected

           t ← (LoopControl) and (1C);
           CurrentLoc ← t ← (CurrentLoc)+(t); *done?
           lu←(HighLoc)-t;
           goto[.+2, alu<0];
           lu ← LoopControl, dblgoto[readloop1,readloop0,R<0];
           *do not change data if LoopControl.0=1

PatternTry ← LSH[PatternTry,1], goto[start]; *use the next pattern

```

```

***** SUBROUTINE: FillBuf *****
*
* puts pattern into Wd0, Wd1, Wd2. The pattern depends on the value
* of CurrentPattern:
*
* CurrentPattern      Wd0, Wd1      Wd2
* 1                   000000      00      All Zeros pattern
* 2                   177777      17      All Ones pattern
* 4                   125252      12      Checker pattern
*                    or 1252525    or 05
*                    random      ~random Random pattern
*                    10
*
FillBuf: usectask;
        t←apc&apctask;
        Rlink0←t;

        t ← (CurrentPattern) AND (1C);
        goto[Try1,alu=0];      *want the zeros pattern?
        Wd0 ← 0C;              *yes, fill Wd0-2 with zeros pattern
        Wd1 ← 0C;
        Wd2 ← 0C, goto[Bottom];
Try1:   t ← (CurrentPattern) AND (2C); *no, try the ones pattern
        goto[Try2,alu=0];      *want the ones pattern?
        t ← Ones;              *yes, fill Wd0-2 with ones pattern
        Wd0 ← t;
        Wd1 ← t;
        t ← (Ones) AND (17C);
        Wd2 ← t, goto[Bottom];
Try2:   t ← (CurrentPattern) AND (4C); *no, try the checker pattern
        goto[Try3,alu=0];      *want the checker pattern?
        Toggle ← Toggle, goto[Checker01,R ODD]; *yes, fill Wd0-2 with checker pattern
        t ← Checker1;          *10101010101010 pattern
        Wd0 ← t;
        Wd1 ← t;
        t ← (Checker1) AND (17C);
        Wd2 ← t;
        Toggle ← (Toggle) + 1, goto[Bottom]; *toggle checker pattern
Checker01: t ← Checker0; *01010101010101 pattern
        Wd0 ← t;
        Wd1 ← t;
        t ← (Checker0) AND (17C);
        Wd2 ← t;
        Toggle ← (Toggle) + 1, goto[Bottom]; *toggle checker pattern
Try3:   t ← (CurrentPattern) AND (10C); *no, try the random pattern
        goto[Bottom,alu=0];      *want the random pattern?
        call[Rand];              *yes, fill Wd0-2 with random pattern
        Wd0←t;
        call[Rand];
        Wd1←t;
        call[Rand];
        Wd2←t;
        Wd2←(Wd2)and (17C);
Bottom: apc&apctask←Rlink0;
        return;

```

```

***** SUBROUTINE: Rand *****
*
*   to produce a pseudo-random number in t & Xa reg.
*   formula: Xa ← 4005*Xa + ca mod 2**16

Rand: t←Xa;
      t←(lsh[Xa,2])+(t);      *t← 5*Xa
      t←(lsh[Xa,13])+(t);    *t←4005*Xa
      t←(Ca)+t;
      Xa←t,return;

***** SUBROUTINE: ReadCSX *****
*
*   to read control store location CurrentLoc into RD0-2

ReadCSX: usectask;
         t←apc&apctask;
         Rlink0←t;
         t←ZERO;
         apc&apctask←CurrentLoc;
         readCS;
         t←csdata;
         Rd0←t;

         t←1C;
         apc&apctask ← CurrentLoc;
         readCS;
         t←csdata;
         Rd1←t;

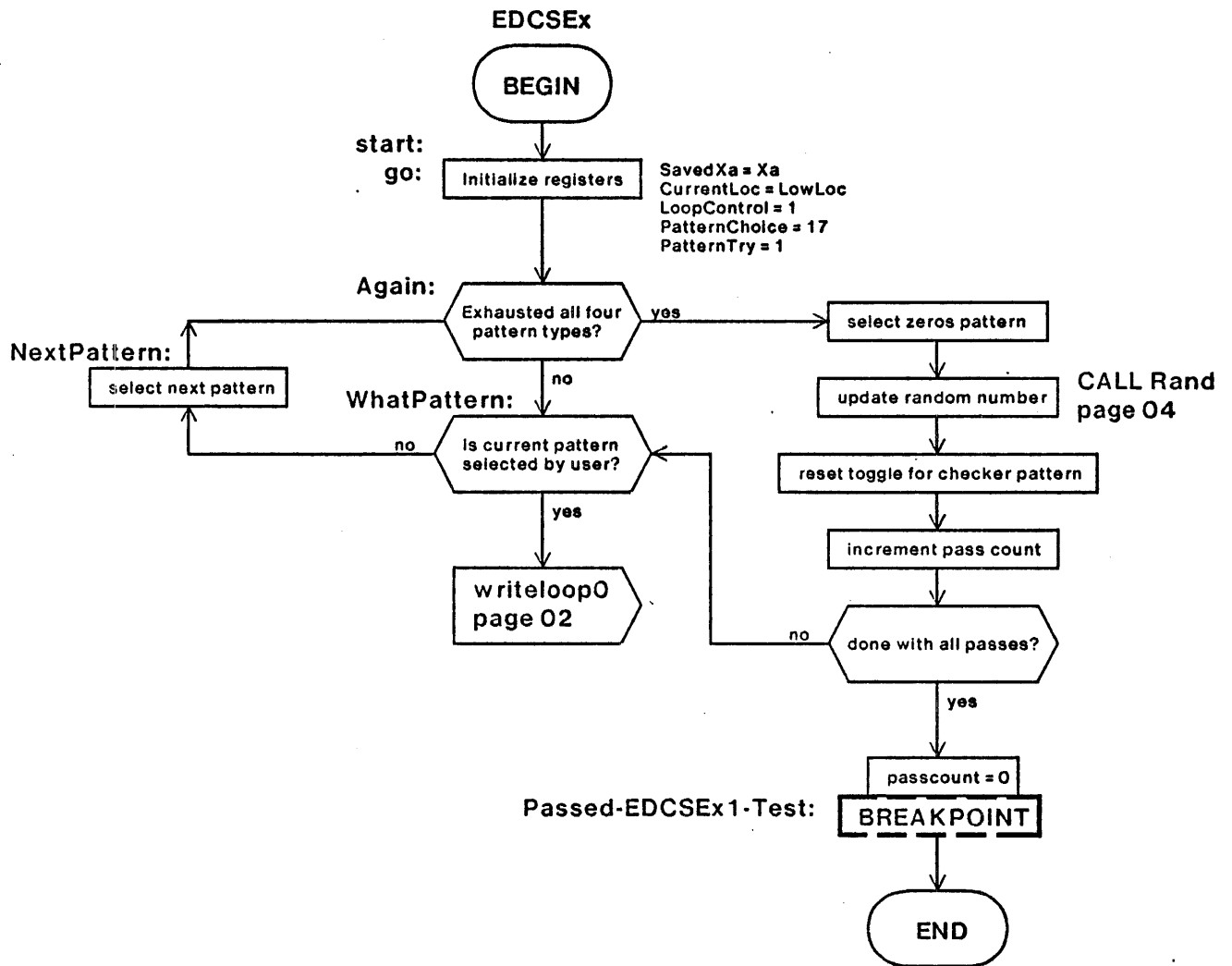
         t←3C;
         apc&apctask ← CurrentLoc;
         readCS;
         t←csdata;
         Rd2←t;
         Rd2←(LDF[Rd2,0,4]);      *control store bits come back in bits 0-3
         apc&apctask←Rlink0;
         return;

***** SUBROUTINE: WriteCS *****
*
*   to write 3 words from WD0-2 to the control store at CurrentLoc.

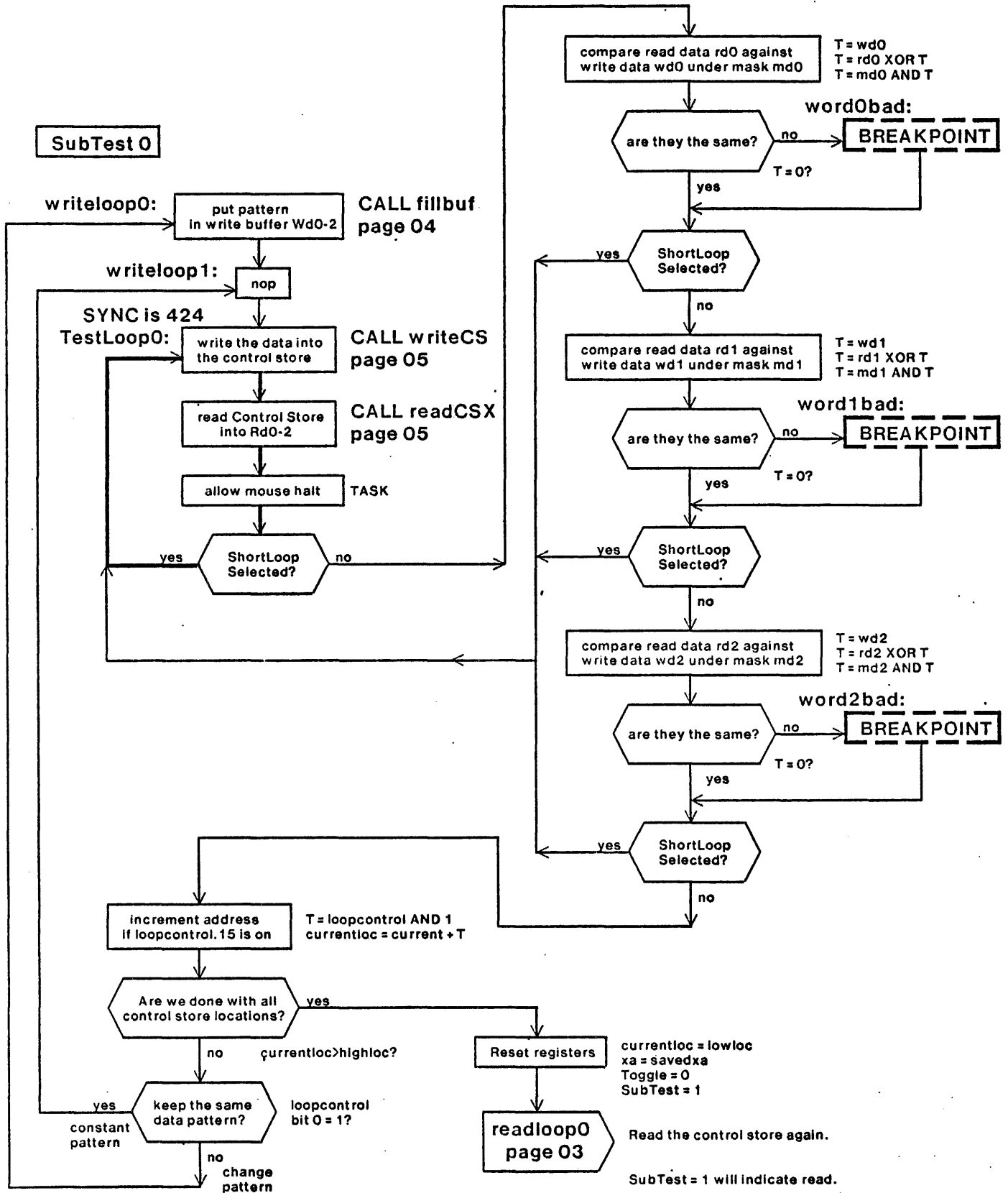
WriteCS: usectask;
         t←apc&apctask;
         Rlink0←t;
         t←Wd2;
         lu←Wd0;
         apc&apctask←CurrentLoc;
         writeCS0&2;
         lu←Wd1;
         apc&apctask←CurrentLoc;
         writeCS1;
         apc&apctask←Rlink0;
         return;

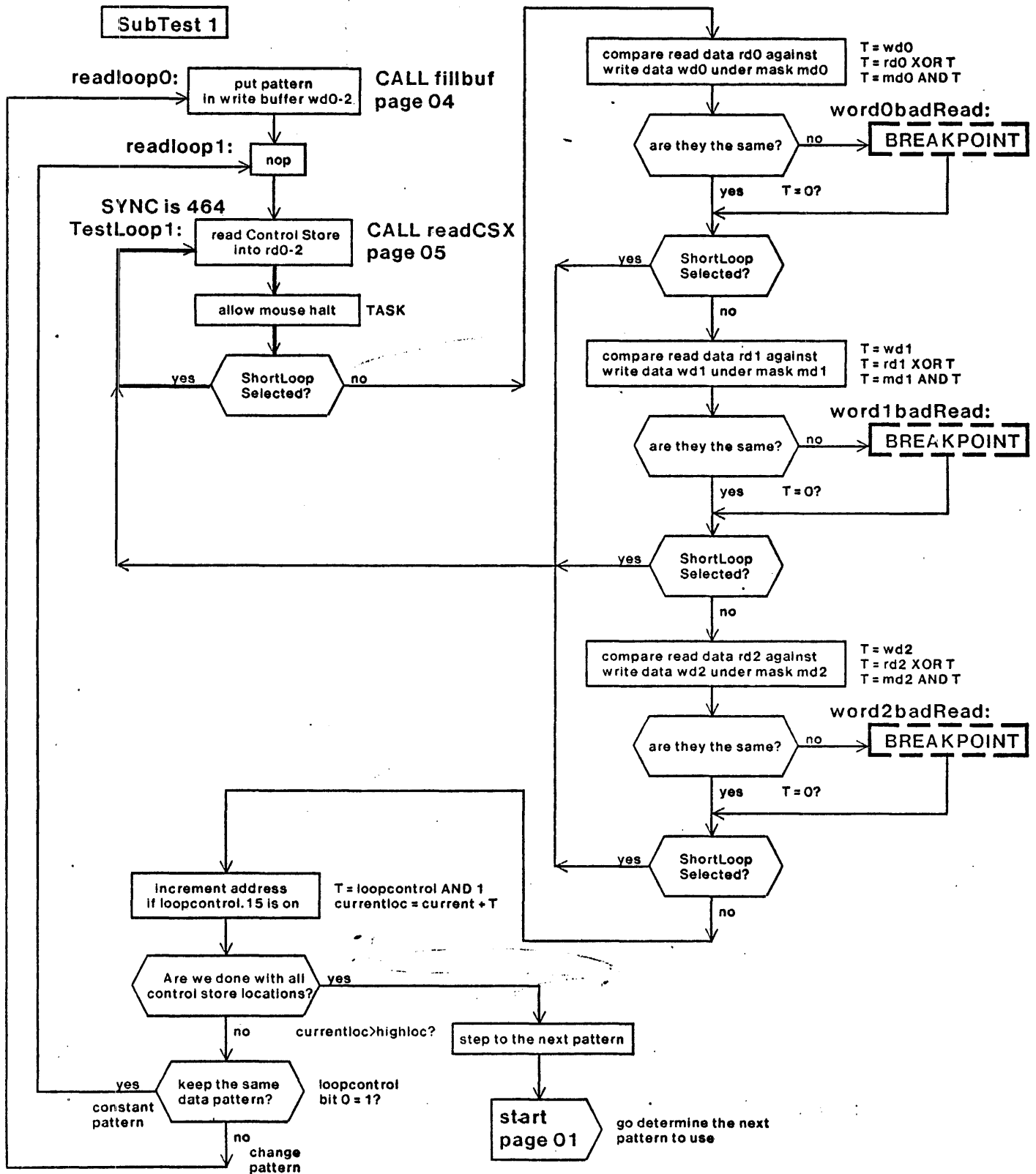
end;          *to end the MAIN routine

```



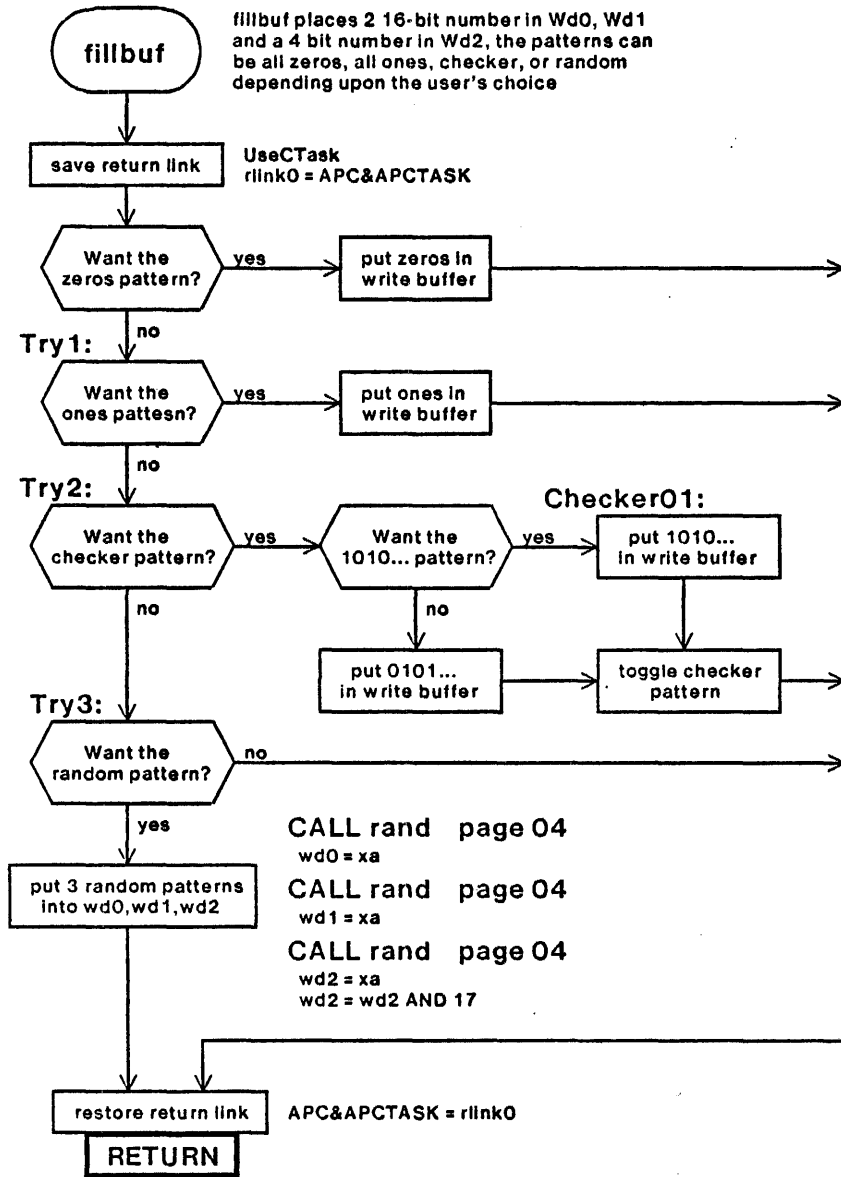
XEROX	D(0)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDCSEx.mc	EDCSEx01.sil	Tom Henning	1	10/24/79	01



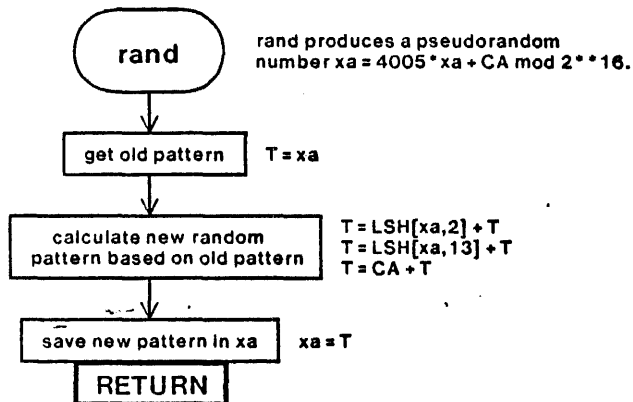




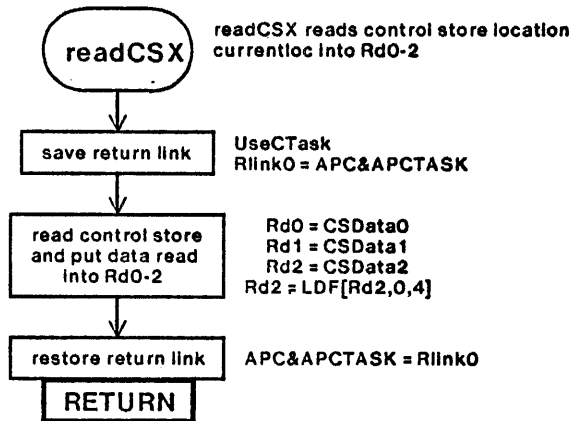
**SUBROUTINE**



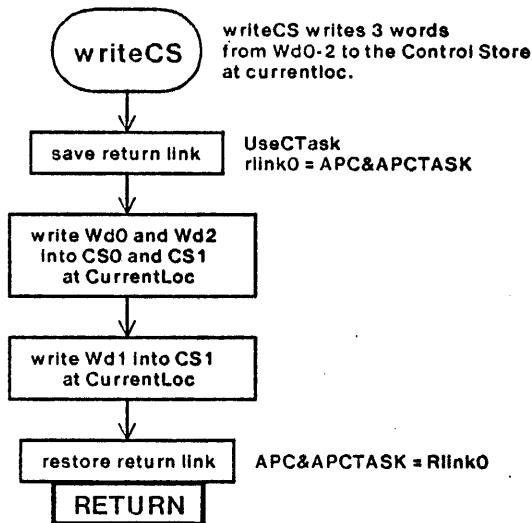
**SUBROUTINE**



**SUBROUTINE**



**SUBROUTINE**



XEROX ED	D(0) Diagnostic	PROGRAM NAME EDCSEx.mc	DOCUMENTATION FILE EDCSEx05.sil	DESIGNER Tom Henning	REV 1	DATE 10/24/79	PAGE 05
-------------	--------------------	---------------------------	------------------------------------	-------------------------	----------	------------------	------------

PARITY	0	REVISION	1	COMM-ER0	0
CYCLECONTROL	377	RUN-TIME	36	COMM-ER1	0
PCXREG	7	PASSCOUNT	0	COMM-ER2	0
PCFREG	7	MAXPASS	100	BOOT-ERR	0
DBREG	15	SUBTEST	0	BOOTREASON	40
SBREG	17			MEMSYNDROME	170167
MNBR	64				
SSTKP	377				
STKP	0				
ALURESULT	7	MD0	177777	LOWLOC	1000
SALUF	0	WD0	177777	HIGHLOC	6777
T 20	123	RD0	0	CURRENTLOC	0
AATOVA	0				
TPC 20	7777	MD1	177777	LOOPCONTROL	1
CALLER	ILC0+7526	WD1	177777	PATTERNCHOICE	17
PAGE	1	RD1	0	SHORTLOOP	0
APC	7011				
APCTASK	16	MD2	17		
CIA	60+1	WD2	17		
CTASK	0	RD2	0		

Loaded: EDCSEx

Time: 00.10

Exit Boot Run-Prog Read-Cmds Break UnBreak CtrAddedBPs CtrAllBPs ShowBPs Go  
 SS Continue Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

MicroD 8.6 (OS 16) of April 27, 1979  
at 27-Nov-79 11:14:00

microd.run EDCSEx

EDCSEx.DIB 251b instructions written 27-Nov-79 11:13:19

Total of 251b instructions

Checking for errors...

Linking...

Building allocation lists...

Assigning locations...

251b instructions in rings involving ONPAGE or AT

Reloading binaries...

Checking assignment...

Writing .MB file...

Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
EDCSEx.DIB:					
0	467	26150	65121	6	GO START
1	650	26050	125117	16	(+1)
2	647	20150	65115	2	(+2)
3	646	20050	125113	6	(+3)
4	645	12217	41110	12	AGAIN
5	644	50	24174	0	(+1)
6	477	12000	103165	10	(+2)
7	472	50	25224	1	(+3)
10	473	14020	101013	15	(+4)
11	505	11050	165010	1	(+5)
12	504	11450	25007	5	(+6)
13	503	50	24205	0	(+7)
14 b	403	10020	101156	0	PASSED-EDCSEX-TEST
15	402	50	25175	0	(+1)
16	476	12150	65107	6	WHATPATTERN
17	643	12250	65104	12	(+1)
20	642	50	24160	0	(+2)
21	471	12050	125102	16	THISPATTERN
22	470	12174	103112	12	NEXTPATTERN
23	641	10020	101030	10	(+1)
24	414	50	25351	1	WRITELOOP0
25	415	50	25050	0	WRITELOOP1
26	424	50	25301	2	TESTLOOP0
27	425	50	25250	2	(+1)
30	426	22150	65377	11	(+2)
31	577	20450	65400	14	(+3)
32	427	10150	124426	14	(+4)
33	413	50	25050	0	(+5)
34	412	24250	65175	5	(+6)
35	576	50	24045	0	(+7)
36 b	423	50	25044	0	WORD0BAD
37	422	22150	65172	15	(+1)
40	575	22450	65171	1	(+2)
41	574	10150	124423	14	(+3)
42	411	50	25050	0	(+4)
43	410	24250	65167	11	(+5)
44	573	50	24040	0	(+6)
45 b	421	50	25041	0	WORD1BAD
46	420	24150	65165	1	(+1)
47	572	22450	65162	5	(+2)
50	571	10150	124416	14	(+3)
51	407	50	25050	0	(+4)
52	406	24250	65160	15	(+5)
53	570	50	24034	0	(+6)
54 b	417	50	25035	0	WORD2BAD
55	416	10150	124413	14	(+1)
56	405	50	25050	0	(+2)
57	404	12200	43157	1	(+3)
60	567	21150	165155	5	(+4)
61	566	21450	25153	11	(+5)

62	565	50	24331	0	(+6)
63	454	12150	24431	0	(+7)
64	455	20150	65065	1	(+10)
65	532	20050	125063	5	(+11)
66	531	26150	65060	15	(+12)
67	530	26050	125066	5	(+13)
70	527	14020	101054	15	(+14)
71	526	10000	103060	10	(+15)
72	430	50	25351	1	READLOOP0
73	431	50	25151	0	READLOOP1
74	464	50	25250	2	TESTLOOP1
75	465	22150	65252	11	(+1)
76	525	20450	65400	14	(+2)
77	466	10150	124503	14	(+3)
100	441	50	25151	0	(+4)
101	440	24250	65050	5	(+5)
102	524	50	24144	0	(+6)
103	b 463	50	25145	0	WORD0BADREAD
104	462	22150	65046	15	(+1)
105	523	22450	65045	1	(+2)
106	522	10150	124476	14	(+3)
107	437	50	25151	0	(+4)
110	436	24250	65042	11	(+5)
111	521	50	24141	0	(+6)
112	b 461	50	25140	0	WORD1BADREAD
113	460	24150	65040	1	(+1)
114	520	22450	65036	5	(+2)
115	517	10150	124473	14	(+3)
116	435	50	25151	0	(+4)
117	434	24250	65034	15	(+5)
120	516	50	24135	0	(+6)
121	b 457	50	25134	0	WORD2BADREAD
122	456	10150	124467	14	(+1)
123	433	50	25151	0	(+2)
124	432	12200	43033	1	(+3)
125	515	21150	165031	5	(+4)
126	514	21450	25026	11	(+5)
127	513	50	24200	0	(+6)
130	400	12150	24461	0	(+7)
131	401	12174	103157	10	(+10)
132	564	47	27146	1	FILLBUF
133	563	50150	65144	15	(+1)
134	562	26050	125142	1	(+2)
135	561	12200	43140	15	(+3)
136	560	50	24121	0	(+4)
137	451	22020	101074	11	(+5)
140	536	22020	101073	15	(+6)
141	535	24020	101001	1	(+7)
142	450	12200	45137	15	TRY1
143	557	50	24114	0	(+1)
144	447	14150	65105	1	(+2)
145	542	22050	125103	11	(+3)
146	541	22050	125101	15	(+4)
147	540	14200	77076	1	(+5)
150	537	24050	125000	1	(+6)
151	446	12200	51134	15	TRY2
152	556	50	24111	0	(+1)
153	445	14150	124507	14	(+2)
154	442	14150	65116	5	(+3)
155	547	22050	125114	11	(+4)
156	546	22050	125113	15	(+5)
157	545	14200	77110	5	(+6)
160	544	24050	125107	1	(+7)
161	543	15050	125001	15	(+10)
162	443	14150	65131	11	CHECKER01
163	554	22050	125127	11	(+1)
164	553	22050	125125	15	(+2)
165	552	14200	77123	11	(+3)
166	551	24050	125120	1	(+4)
167	550	15050	125001	15	(+5)
170	444	12200	61132	15	TRY3
171	555	50	24000	1	(+1)
172	501	50	25224	1	(+2)
173	502	22050	125170	10	(+3)
174	474	50	25224	1	(+4)
175	475	22050	125124	14	(+5)

176	452	50	25224	1	(+6)
177	453	24050	125066	1	(+7)
200	533	24200	137001	1	(+10)
201	500	26147	21071	1	BOTTOM
202	534	50	25401	0	(+1)
203	512	26150	65023	5	RAND
204	511	27174	45020	5	(+1)
205	510	27174	67016	5	(+2)
206	507	27150	65014	11	(+3)
207	506	26050	125400	4	(+4)
210	624	47	27047	2	READCSX
211	623	50150	65045	16	(+1)
212	622	26050	125043	2	(+2)
213	621	176	41041	2	(+3)
214	620	20147	21037	6	(+4)
215	617	47	35434	2	(+5)
216	616	54150	65033	16	(+6)
217	615	20050	125031	16	(+7)
220	614	0	43026	2	(+10)
221	613	20147	21025	6	(+11)
222	612	47	35423	2	(+12)
223	611	54150	65021	16	(+13)
224	610	22050	125017	2	(+14)
225	607	0	47014	2	(+15)
226	606	20147	21013	6	(+16)
227	605	47	35410	2	(+17)
230	604	54150	65006	16	(+20)
231	603	22050	125004	6	(+21)
232	602	22162	133003	6	(+22)
233	601	26147	21000	2	(+23)
234	600	50	25401	0	(+24)
235	640	47	27077	2	WRITECS
236	637	50150	65075	16	(+1)
237	636	26050	125073	2	(+2)
240	635	24150	65070	2	(+3)
241	634	22150	25067	12	(+4)
242	633	20147	21064	6	(+5)
243	632	47	31463	2	(+6)
244	631	22150	25060	16	(+7)
245	630	20147	21057	6	(+10)
246	627	47	33454	2	(+11)
247	626	26147	21053	2	(+12)
250	625	50	25401	0	(+13)

Page 400: 251 locations used, 127 free

RM:

20	1000	LOWLOC
21		CURRENTLOC
22	6777	HIGHLOC
23		RDO
24		RD1
25		RD2
26	177777	WDO
27	177777	WD1
30	17	WD2
31	177777	MDO
32	177777	MD1
33	17	MD2
34		RLINKO
35	123	XA
36	33031	CA
37		SAVEDXA
40	0	PASSCOUNT
41	100	MAXPASS
42		SUBTEST
43	0	SHORTLOOP
44	1	LOOPCONTROL
45	17	PATTERNCHOICE
46	1	PATTERNRY
47	1	CURRENTPATTERN
50	177777	ONES
51	125252	CHECKER1
52	52525	CHECKER0
53	0	TOGGLE

54 1 REVISION  
55 36 RUN-TIME  
56 RLCØ

Time: 8 seconds; 0 error(s), 0 warning(s), 11850 words free





```

L X WriteMessage ' PASSCOUNT =      ;
R B2 Val;
L X WriteMessage;
L X WriteMessage ~;

L X Skip .continue;

.word1bad L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage *          WORD 1 Bad ~;
L X BackSkip .bad;

.word2bad L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage *          WORD 2 Bad ~;
L X BackSkip .bad;

.word0badread L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage *          WORD 0 READ Bad ~;
L X BackSkip .bad;

.word1badread L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage *          WORD 1 READ Bad ~;
L X BackSkip .bad;

.word2badread L X WriteMessage *** FAILED: at my Breakpoint ~;
L X WriteMessage *          WORD 2 READ Bad ~;
L X BackSkip .bad;

.passtest L X WriteMessage ----- PASSEd EDCSEx Test : ;
L X WriteDT;
L X WriteMessage ----- ;
L X Skip .continue;

.continue L X WriteMessage ~;
L X CloseOutput;
L X DisplayOn;
L X Confirm;
L X TimeOut 10000000;
L X Continue;
L X Skip 2;
L X ShowError Program failed to CONTINUE.;
L X BackSkip .notmybreak;
L X DisplayOff;
L X BackSkip .breakpoint;
```

L A19 Val 0  
L X Confirm  
L X load EDCSEx;

L B0 Addr REVISION  
L B1 Addr RUN-TIME  
L B2 Addr PASSCOUNT  
L B3 Addr MAXPASS  
L B4 Addr SUBTEST

L B9 Addr MD0;  
L B10 Addr WD0;  
L B11 Addr RD0;  
L B13 Addr MD1;  
L B14 Addr WD1;  
L B15 Addr RD1;  
L B17 Addr MD2;  
L B18 Addr WD2;  
L B19 Addr RD2;

L C9 Addr LOWLOC;  
L C10 Addr HIGHLOC;  
L C11 Addr CURRENTLOC;

L C13 Addr LOOPCONTROL  
L C14 Addr PATTERNCHOICE  
L C15 Addr SHORTLOOP

L X DisplayOn;  
L X TimeOut 10000  
L X SS GO  
L X Skip 1  
L X ShowError Single-step stuck at GO

```

%
*** **> <D0Diag>Rev-1>EDCym.mc      Revision 1   Jan. 2, 1980   *** **>
*****
*** EDCym.mc:  Cyclor Masker Test microcode
*** Purpose:   To Test all of the Cyclor Masker functions except FixVA which is tested by EDTNF.mc
*** Minimum Hardware:  Standard 4 CPU boards.
*** Approximate run time: 7 seconds.
*** Written by: Tom Horsley, December 30, 1977
*** Modified by: Bill Kennedy, February 22, 1978
                 To add control store parity.
*** Modified by: Bill Kennedy, March 2, 1978
                 To move code off of Page 0.
*** Modified by: Bill Kennedy, April 24, 1978
                 To add panel increments.
*** Modified by: Chuck Thacker, May 29, 1978
                 To improve readability.
                 Note: if the FIELD test is run before EDCym, then CYM2 is useless,
                       since the only difference between it and EDCym is that CYM2 does not
                       use WFA and WFB to load the control store. Accordingly, CYM2 should be
                       eliminated from the set of tests.
*** Modified by: Mike Spaur, January 14, 1980
                 To standardize documentation and install looping capabilities.
*****

```

```

*****
* SubTest Descriptions:

```

- \* SubTest0: Generate a random number that determines the Cyclor Masker mode of operation.
- \* SubTest1: Read the control store micro instruction at location 1100 and alter the F1 and F2 fields so that they contain an 8 bit random number. Then calculate the new parity bit and insert the new instruction back into control store location 1100.
- \* SubTest2: Calculate what the result of the Cyclor Masker operation should be and save the answer in the "myResult" register.
- \* SubTest3: Allow the cyclor masker to execute the control store instruction at location 1100 and save the result in the "Result" register.
- \* SubTest4: Compare the result of the hardware operation (SubTest3) to the calculated result (SubTest2).
- \* SubTest5: Examine the contents of the APC to see if it is correct. SubTest5 is only executed if the random F fields specify a dipatch type operation.

```

*****
* BreakPoints:

```

- \* ResultBad: The hardware result of the Cyclor Masker operation (Result) did not equal the calculated result (myResult).
- \* APCBad: The contents of the APC (APCResult) did not equal the calculated result (myResult).
- \* Passed-EDCym-Test: Passed all tests, and all passes.

```

*****
* ShortLoop Logic Analyzer Sync Points at Control Store address:

```

- \* ResultBad: Control Store address 1036
- \* APCBad: Control Store address 1036

- \*\*\*\*\*
- \* Special Register Definitions:
  - \* ShortLoop: At any breakpoint the user has the option of changing the value of ShortLoop to 1, which will cause Subtest3 to loop endlessly. If ShortLoop is a zero then the program will proceed to the next test.
  - \* InnerLoopCounter: This register contains the number of passes through the mainloop. The mainloop is simply the sum of all of the sub tests. When InnerLoopCounter = 2\*\*16 all of the functions that are tested by EDCym have been tested exactly once.
  - \* PassCount: This register displays the number of completed passes of the entire EDCym test (2\*\*16 iterations of the mainloop). Also, the contents of this register is the octal equivalent of the Maintenance Pannel display.
  - \* MaxPass: This register can be set by the user. It determines how many repetitions of the entire EDCym test will be made.
  - \* XA: This test is driven by a sequence of 16-bit random numbers. In each iteration of the mainloop, the random number is generated. The random number generator has been constructed so that it produces each number in the range of [0-64K] once and only once before repeating any number. Thus it is guaranteed to have exhausted all possible combinations of the fields derived from it when InnerLoopCounter reaches 2\*\*16. XA[0:7] is used to define the Cycler Masker function through use of the F1 and F2 fields of the microinstruction. These fields are built up from XA as follows:
    - F1 = XA[0:3]
    - F2 = XA[4:7]
  - \* Fnum: This register is used to Load the F1 & F2 fields. Fnum[0:7] = 0, and Fnum[10:17] = XA[0:7] This register is then used to decide which Cycler Masker function has been defined. There are six possible functions that are tested by the mainloop:
 

Fnum = 0 to 206	The Load Field function (LDFTest)
Fnum = 207 to 300	The Dispatch function (DispatchTest)
Fnum = 301 to 317	The LeftShift function (LSHTest)
Fnum = 320 to 336	The Left Cycle function (LCYTest)
Fnum = 337	The Left Hand Mask function (LHMaskTest)
Fnum = 340	The Zero function (ZeroTest)
  - \* Operand: This is a random 16 bit word that is used to simulate the operation of the Cycler Masker using adds and carries. Operand = XA + PassCount.
- \*\*\*\*\*

%

\*\*\*\*\*

\* INITIALIZATION:

BUILTIN[INSERT, 24];

INSERT[DOLANG];

TITLE[CyclerMaskerTester];

SET[MainPage, 3];

\* Set page number for main part of program

SET[SubPage1, 1];

\* Set page number for first sub-part of program

SET[SubPage2, 2];

\* Set page number for first sub-part of program

\*\*\*\*\* R-registers \*\*\*\*\*

RV[PassCount,20,0];

\* number of passes of the EDCym test executed

RV[MaxPass,21,2];

\* number of times EDCym is to repeat before breakpointing

RV[InnerLoopCounter,22];

\* number of passes of the mainloop (subtests 0 to 5)

RV[ShortLoop,23,0];

\* 0 = continue on next test, 1 = loop on Subtest 3

RV[SubTest,24];

\* current location of test

RV[CheckAPC,25];

\* flag indicating whether the APC result is to be checked

RV[CA,26];

\* used in random number generation, A\*XA + CA

RV[XA,27];

\* random number generated via A\*XA + CA

RV[Fnum,30];

\* current cycle mask function being tested

RV[Operand,31];

\* word to be cycle masked

RV[NBits,32];

\* number of bits to be cycled or shifted

RV[LeftBit,33];

\* first bit of field to be operated on

RV[bitNum,34];

\* loop counter in simulated shifts

RV[CS0,35];

\* temporary storage for first word of a control store location

RV[CS1,36];

\* temporary storage for second word of a control store location

RV[CS2,37];

\* temporary storage for third word of a control store location

RV[CSP,40];

\* temporary register for control store parity calculations

RV[Address,41];

\* location of CS instruction to be stuffed

RV[LDFTest,42];

\* number of times LDF test has been executed

RV[DispatchTest,43];

\* number of times Dispatch test has been executed

RV[LSHTest,44];

\* number of times LSH test has been executed

RV[LCYTest,45];

\* number of times LCY test has been executed

RV[LHMaskTest,46];

\* number of times LHMask test has been executed

RV[ZeroTest,47];

\* number of times Zero test has been executed

RV[myResult,50];

\* result of simulated function

RV[Result,51];

\* result of hardware function

RV[APCResult,52];

\* contents of APC&APCTASK after test

RV[StuffTmp,53];

\* used in the stuff operations

RV[tmp,54];

\* temporary register

RV[tmp2,55];

\* temporary register

RV[tmp3,56];

\* temporary register

RV[Revision,57,1];

\* current revision of EDCym

RV[Run-Time,60,7];

\* the octal number of seconds it takes

\* to complete two passes of the EDCym test

\*\*\*\*\*

\*\*\* MAIN routine

ONPAGE[MainPage];

go:

```
start:  XA ← AND@[0377, 123]C;
        XA ← (XA) OR (AND@[177400, 123]C);

        CA ← AND@[0377, 33031]C;
        CA ← (CA) OR (AND@[177400, 33031]C);
```

```
CLEARMPANEL;
InnerLoopCounter ← 0C;
PassCount ← 0C;
LDFTest ← 0C;
DispatchTest ← 0C;
LSHTest ← 0C;
LCYTest ← 0C;
LHMaskTest ← 0C;
ZeroTest ← 0C;
Address ← AND@[0377, 1100]C;
Address ← (Address) OR (AND@[177400, 1100]C);
GOTO[mainLoop];
```

```
bigLoop: INCM PANEL;
        PassCount ← (PassCount) + 1;
        t ← (MaxPass);
        LU ← (PassCount) - (t);
        GOTO[Passed-EDCym-Test, ALU >= 0];
        GOTO[mainLoop];
```

```
Passed-EDCym-Test: BREAKPOINT;
        PassCount ← 0C;
```

\*\*\* SUBTEST 0

```
mainLoop:
        SubTest ← 0C;
        CheckAPC ← 0C;

        InnerLoopCounter ← (InnerLoopCounter) + 1;
        GOTO[bigLoop, CARRY];
```

```
t ← XA;
t ← (LSH[XA, 2]) + t;
t ← (LSH[XA, 13]) + t;
t ← (CA) + t;
XA ← t;

* This is the psuedo random number algorithm
* XA (new) = [ [ 4005 * XA(old) ] + CA ] modulo 2**16
```

\*\*\* SUBTEST 1

```
SetVars: SubTest ← 1C;
        t ← LDF[XA, 0, 10];
        Fnum ← t;
```

```
t ← PassCount;
t ← (XA) + t;
Operand ← t;
```

```
t ← (Address);
CS2 ← t;
* use CS2 for a temporary register until end
```

```
GetCS:  t ← 0C;
        APCTASK&APC ← (CS2);
        READCS;
        t ← CSData;
        CS0 ← t;
        t ← 1C;
        APCTASK&APC ← (CS2);
        READCS;
        t ← CSData;
        CS1 ← t;
        t ← 3C;
        APCTASK&APC ← (CS2);

* These lines of code read the control store data
* at adress 1100 into the variables CS0,CS1 & CS2.
```

```

READCS;
t ← CSData;
CS2 ← t;
CS2 ← RSH[CS2, 14];

```

## ChangeFields:

```

t ← LDF[Fnum, 10, 4];      * Extract the F1 field from Fnum
Tmp ← t;

```

```

StuffTmp ← OR@[LSHIFT[14, 4], SUB[4, 1]]C;
CYCLECONTROL ← StuffTmp;
t ← WFA[Tmp];              * These four lines of code change the F1 field in CS0
CS0 ← WFB[(CS0) OR t];    * so that it is identical to Fnum[10:13]

```

```

t ← LDF[Fnum, 14, 4];      * Extract the F2 field from Fnum
Tmp ← t;

```

```

StuffTmp ← OR@[LSHIFT[2, 4], SUB[4, 1]]C;
CYCLECONTROL ← StuffTmp;
t ← WFA[Tmp];              * These four lines of code change the F2 field in CS1
CS1 ← WFB[(CS1) OR t];    * so that it is identical to Fnum[14:17]

```

## CalcPar: t ← CS0;

```

CSP ← t;                    * Calculate the new control store parity
t ← CS1;                    * put CS0 in the temp. reg.
CSP ← t + (CSP) XOR (t);    * get CS1
t ← (LDF[CS2,14,4]) XOR (t); * exclusive or the first two CS words
CSP ← t + (LDF[CSP,0,10]) XOR (t); * now exclusive or the third CS word with the result
CSP ← t + (LDF[CSP,10,4]) XOR (t); * now start halving process to get parity
CSP ← t + (LDF[CSP,14,2]) XOR (t);
CSP ← t + (LDF[CSP,16,1]) XNOR (t); * Do last part and complement it
t ← (LDF[CSP,17,1]);        * put parity bit in the t-register
CS1 ← (CS1) XOR (t);        * exclusive or the parity bit into bit 31 of CS (15 of CS1)

```

## WriteCS: t ← (CS2);

```

LU ← (CS0);                  * Put the new instruction back into the control Store
APCTASK&APC ← (Address);    * writeCS -- write control store location
WriteCS0&2;
LU ← (CS1);
APCTASK&APC ← (Address);
WriteCS1;

```

```

LOADPAGE[SubPage1];        * Page (change to page SubPage1)
GOTOP[.+1];
ONPAGE[SubPage1];

```

## \*\*\* SUBTEST2

## SubTest2:

```

SubTest ← 2C;
t ← 207C;                    * last LDF + 1
LU ← (Fnum) - (t);
GOTO[FnumGT206, ALU >= 0];

```

```

LU ← (Fnum) - (20C);        * LDF Tests
GOTO[FnumGT017, ALU >= 0];

```

```

NBits ← 1C;                  * 20 1-bit fields
t ← Fnum;
LeftBit ← t;
GOTO[myLDF];

```

## FnumGT017:

```

LU ← (Fnum) - (37C);
GOTO[FnumGT036, ALU >= 0];

```

```

NBits ← 2C;                  * 17 2-bit fields
t ← (Fnum) - (20C);
LeftBit ← t;
GOTO[myLDF];

```

## FnumGT036:

```

LU ← (Fnum) - (55C);
GOTO[FnumGT054, ALU >= 0];

```

```

NBits ← 3C;                  * 16 3-bit fields
t ← (Fnum) - (37C);

```

```
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT054:

```
LU ← (Fnum) - (72C);  
GOTO[FnumGT071, ALU >= 0];
```

```
NBits ← 4C; * 15 4-bit fields  
t ← (Fnum) - (55C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT071:

```
LU ← (Fnum) - (108C);  
GOTO[FnumGT105, ALU >= 0];
```

```
NBits ← 5C; * 14 5-bit fields  
t ← (Fnum) - (72C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT105:

```
LU ← (Fnum) - (121C);  
GOTO[FnumGT120, ALU >= 0];
```

```
NBits ← 6C; * 13 6-bit fields  
t ← (Fnum) - (106C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT120:

```
LU ← (Fnum) - (133C);  
GOTO[FnumGT132, ALU >= 0];
```

```
NBits ← 7C; * 12 7-bit fields  
t ← (Fnum) - (121C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT132:

```
LU ← (Fnum) - (144C);  
GOTO[FnumGT143, ALU >= 0];
```

```
NBits ← 10C; * 11 10-bit fields  
t ← (Fnum) - (133C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT143:

```
LU ← (Fnum) - (154C);  
GOTO[FnumGT153, ALU >= 0];
```

```
NBits ← 11C; * 10 11-bit fields  
t ← (Fnum) - (144C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT153:

```
LU ← (Fnum) - (163C);  
GOTO[FnumGT162, ALU >= 0];
```

```
NBits ← 12C; * 7 12-bit fields  
t ← (Fnum) - (154C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT162:

```
LU ← (Fnum) - (171C);  
GOTO[FnumGT170, ALU >= 0];
```

```
NBits ← 13C; * 6 13-bit fields  
t ← (Fnum) - (163C);  
LeftBit ← t;  
GOTO[myLDF];
```

FnumGT170:



```

    LU ← (Fnum) - (176C);
    GOTO[FnumGT175, ALU >= 0];

    NBits ← 14C;
    t ← (Fnum) - (171C);
    LeftBit ← t;
    GOTO[myLDF];
    * 5 14-bit fields

FnumGT175:
    t ← 202C;
    LU ← (Fnum) - (t);
    GOTO[FnumGT201, ALU >= 0];
    * done this way due to sign extend on arithmetic

    NBits ← 15C;
    t ← (Fnum) - (176C);
    LeftBit ← t;
    GOTO[myLDF];
    * 4 15-bit fields

FnumGT201:
    t ← 205C;
    LU ← (Fnum) - (t);
    GOTO[FnumGT204, ALU >= 0];
    * done this way due to sign extend on arithmetic

    NBits ← 16C;
    t ← 202C;
    t ← (Fnum) - (t);
    LeftBit ← t;
    GOTO[myLDF];
    * 3 16-bit fields

FnumGT204:
    NBits ← 17C;
    t ← 205C;
    t ← (Fnum) - (t);
    LeftBit ← t;
    * 2 17-bit fields

myLDF:  t ← (Operand);
        tmp3 ← t;

        t ← (1C);
        bitNum ← t;

LDFupperLoop:
    t ← (LeftBit);
    LU ← (bitNum) - (t) - 1;
    GOTO[LDFReset, ALU >= 0];

    t ← (tmp3);
    tmp3 ← (tmp3) + t;

    bitNum ← (bitNum) + 1;
    GOTO[LDFupperLoop];

LDFReset:
    t ← (NBits);
    tmp2 ← t;
    myResult ← 0C;
    t ← (tmp3);
    tmp ← t;
    t ← (1C);
    bitNum ← t;

LDFlowLoop:
    t ← (tmp2);
    LU ← (bitNum) - (t) - 1;
    GOTO[DoneWithFakeLDF, ALU >= 0];

    t ← (myResult);
    myResult ← (myResult) + t;
    t ← (tmp);
    tmp ← (tmp) + t;
    GOTO[LDFincBitNum, NOCARRY];
    myResult ← (myResult) + 1;

LDFincBitNum:
    bitNum ← (bitNum) + 1;
    GOTO[LDFlowLoop];

```

## DoneWithFakeLDF:

```
LDFTest ← (LDFTest) + 1;
LOADPAGE[SubPage2];
GOTOP[hardwareDoIt];
```

## FnumGT206:

```
LOADPAGE[SubPage2];          * Page (change to page SubPage2)
GOTOP[. +1];
ONPAGE[SubPage2];

t ← 301C;                    * last DISPATCH + 1
LU ← (Fnum) - (t);
GOTO[FnumGT300, ALU >= 0];

t ← 227C;                    * Dispatch Tests
LU ← (Fnum) - (t);
GOTO[FnumGT226, ALU >= 0];

NBits ← 1C;                  * 20 1-bit fields
t ← 207C;
t ← (Fnum) - (t);
LeftBit ← t;
GOTO[myDispatch];
```

## FnumGT226:

```
t ← 246C;
LU ← (Fnum) - (t);
GOTO[FnumGT245, ALU >= 0];

NBits ← 2C;                  * 17 2-bit fields
t ← 227C;
t ← (Fnum) - (t);
LeftBit ← t;
GOTO[myDispatch];
```

## FnumGT245:

```
t ← 264C;
LU ← (Fnum) - (t);
GOTO[FnumGT263, ALU >= 0];

NBits ← 3C;                  * 16 3-bit fields
t ← 246C;
t ← (Fnum) - (t);
LeftBit ← t;
GOTO[myDispatch];
```

## FnumGT263:

```
NBits ← 4C;                  * 15 4-bit fields
t ← 264C;
t ← (Fnum) - (t);
LeftBit ← t;
```

## myDispatch:

```
t ← (Operand);
tmp3 ← t;

t ← (1C);
bitNum ← t;
```

## DispUpperLoop:

```
t ← (LeftBit);
LU ← (bitNum) - (t) - 1;
GOTO[DispReset, ALU >= 0];

t ← (tmp3);
tmp3 ← (tmp3) + t;

bitNum ← (bitNum) + 1;
GOTO[DispUpperLoop];
```

## DispReset:

```
t ← (NBits);
tmp2 ← t;
myResult ← 0C;
t ← (tmp3);
```

```

    tmp ← t;
    t ← (1C);
    bitNum ← t;

DispLowLoop:
    t ← (tmp2);
    LU ← (bitNum) - (t) - 1;
    GOTO[DoneWithFakeDisp, ALU >= 0];

    t ← (myResult);
    myResult ← (myResult) + t;
    t ← (tmp);
    tmp ← (tmp) + t;
    GOTO[DispIncBitNum, NOCARRY];
    myResult ← (myResult) + 1;

DispIncBitNum:
    bitNum ← (bitNum) + 1;
    GOTO[DispLowLoop];

DoneWithFakeDisp:
    CheckAPC ← 1C;
    DispatchTest ← (DispatchTest) + 1;
    GOTO[hardwareDoIt];

FnumGT300:
    t ← 320C;
    LU ← (Fnum) - (t);
    GOTO[FnumGT317, ALU >= 0];
    * last LSH + 1

    t ← 300C;
    t ← (Fnum) - (t);
    NBits ← t;
    * LSH Tests

    t ← (Operand);
    myResult ← t;

    t ← (1C);
    bitNum ← t;

LSHloop: t ← (NBits);
    LU ← (bitNum) - (t) - 1;
    GOTO[DoneWithFakeLSH, ALU >= 0];

    t ← (myResult);
    myResult ← (myResult) + t;

    bitNum ← (bitNum) + 1;
    GOTO[LSHloop];

DoneWithFakeLSH:
    LSHTest ← (LSHTest) + 1;
    GOTO[hardwareDoIt];

FnumGT317:
    t ← 317C;
    LU ← (Fnum) - (t);
    GOTO[FnumGT336, ALU >= 0];
    * last LCY + 1

    t ← 317C;
    t ← (Fnum) - (t);
    NBits ← t;
    * LCY Tests

    t ← (Operand);
    myResult ← t;

    t ← (1C);
    bitNum ← t;

LCYloop: t ← (NBits);
    LU ← (bitNum) - (t) - 1;
    GOTO[DoneWithFakeLCY, ALU >= 0];

    t ← (myResult);
    myResult ← (myResult) + t;
    GOTO[LCYincBitNum, NOCARRY];

```

```

    myResult ← (myResult) + 1;

LCYincBitNum:
    bitNum ← (bitNum) + 1;
    GOTO[LCYloop];

DoneWithFakeLCY:
    LCYTest ← (LCYTest) + 1;
    GOTO[hardwareDoIt];

FnumGT336:
    t ← 337C;
    LU ← (Fnum) - (t);
    GOTO[FnumNeq337, ALU # 0];
    * LHMASK function

FnumEq337:
    t ← (operand) AND (177400C);
    myResult ← t;
    LHMaskTest ← (LHMaskTest) + 1;
    GOTO[hardwareDoIt];
    * LHMASK Test

FnumNeq337:
    t ← 340C;
    LU ← (Fnum) - (t);
    GOTO[FnumNotUseful, ALU # 0];
    * ZERO fuhction

FnumEq340:
    myResult ← 0C;
    ZeroTest ← (ZeroTest) + 1;
    GOTO[hardwareDoIt];
    * ZERO Test

FnumNotUseful:
    LOADPAGE[MainPage];
    GOTOP[mainLoop];
    * untested functions 341 - 377

H1:    NOP;
H2:    NOP;
    * resolves a branching conflict
    * resolves a branching conflict

*** SUBTEST 3
hardwareDoIt:

    TASK;

    SubTest ← 3C;
    t ← (Operand);
    Result ← t;

    Result ← ZERO[Result], AT[1100];
    * Does not do the ZERO function because F1
    * and F2 fields of location 1100 have been altered

    t ← APC&APCTASK;
    APCResult ← t;
    * save this for dispatch test

    ShortLoop ← ShortLoop, GOTO[.+2,R EVEN];
    GOTO[hardwareDoIt];
    * Test for ShortLoop option
    * ShortLoop selected

    NOP;
    * resolves a branching conflict

*** SUBTEST 4
SubTest4:
    SubTest ← 4C;
    t ← Result;
    LU ← (myResult) - (t);
    GOTO[SubTest5, ALU = 0];

ResultBad: BREAKPOINT;

    ShortLoop ← ShortLoop, GOTO[.+2,R EVEN];
    GOTO[H1];
    * Test for ShortLoop option
    * ShortLoop selected

    NOP;
    * resolves a branching conflict

*** SUBTEST 5
SubTest5:
    SubTest ← 5C;
    LU ← (CheckAPC);

```

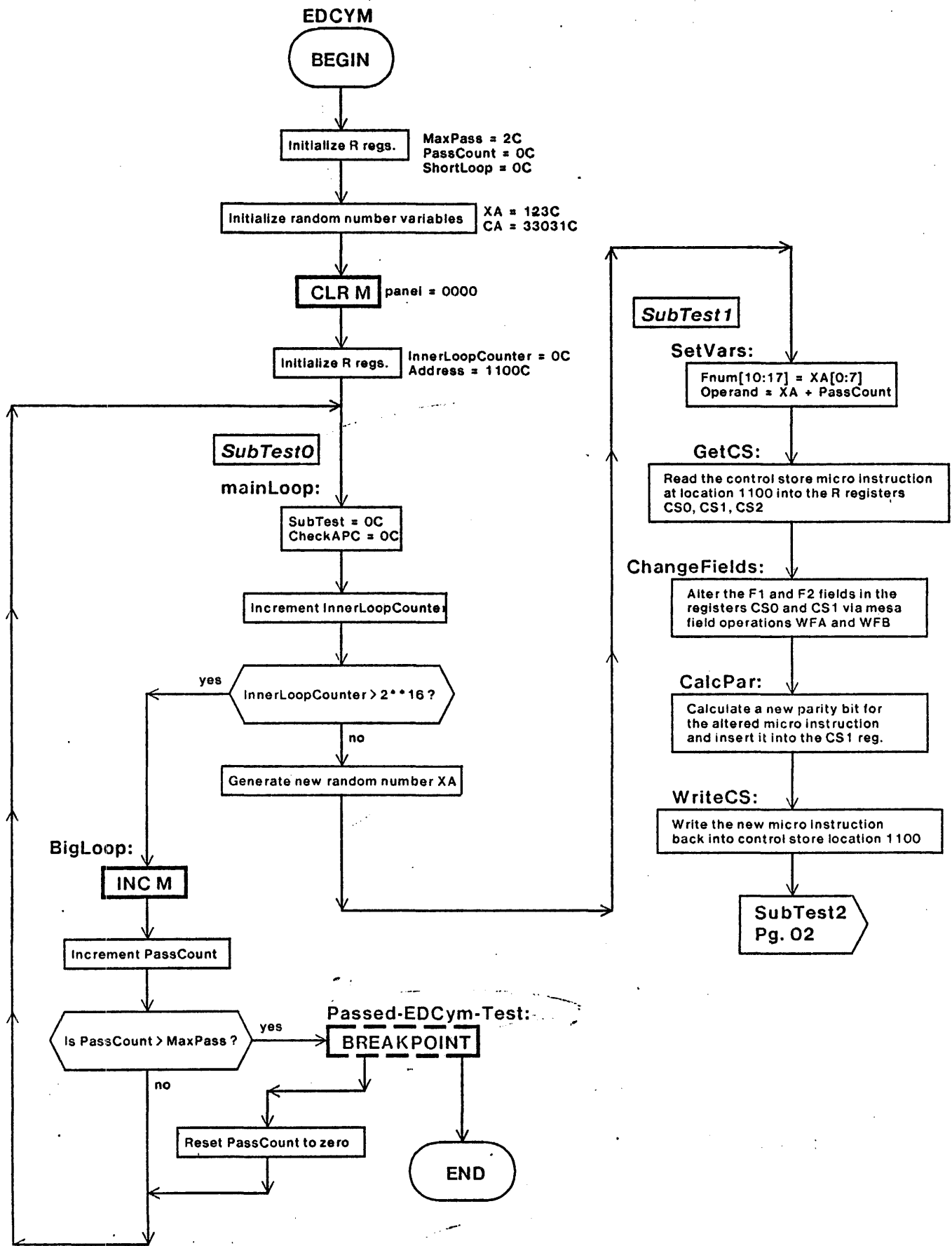
```
GOTO[AllDone, ALU = 0];  
t ← APCResult;  
LU ← (myResult) - (t);  
GOTO[APCOK, ALU = 0];
```

```
APCBad: BREAKPOINT;  
ShortLoop ← ShortLoop, GOTO[.+2,R EVEN]; * Test for ShortLoop option  
GOTO[H2]; * ShortLoop selected
```

```
APCOK: NOP; * resolves a branching conflict  
NOP; * resolves a branching conflict
```

```
AllDone: LOADPAGE[MainPage];  
GOTOP[mainLoop];
```

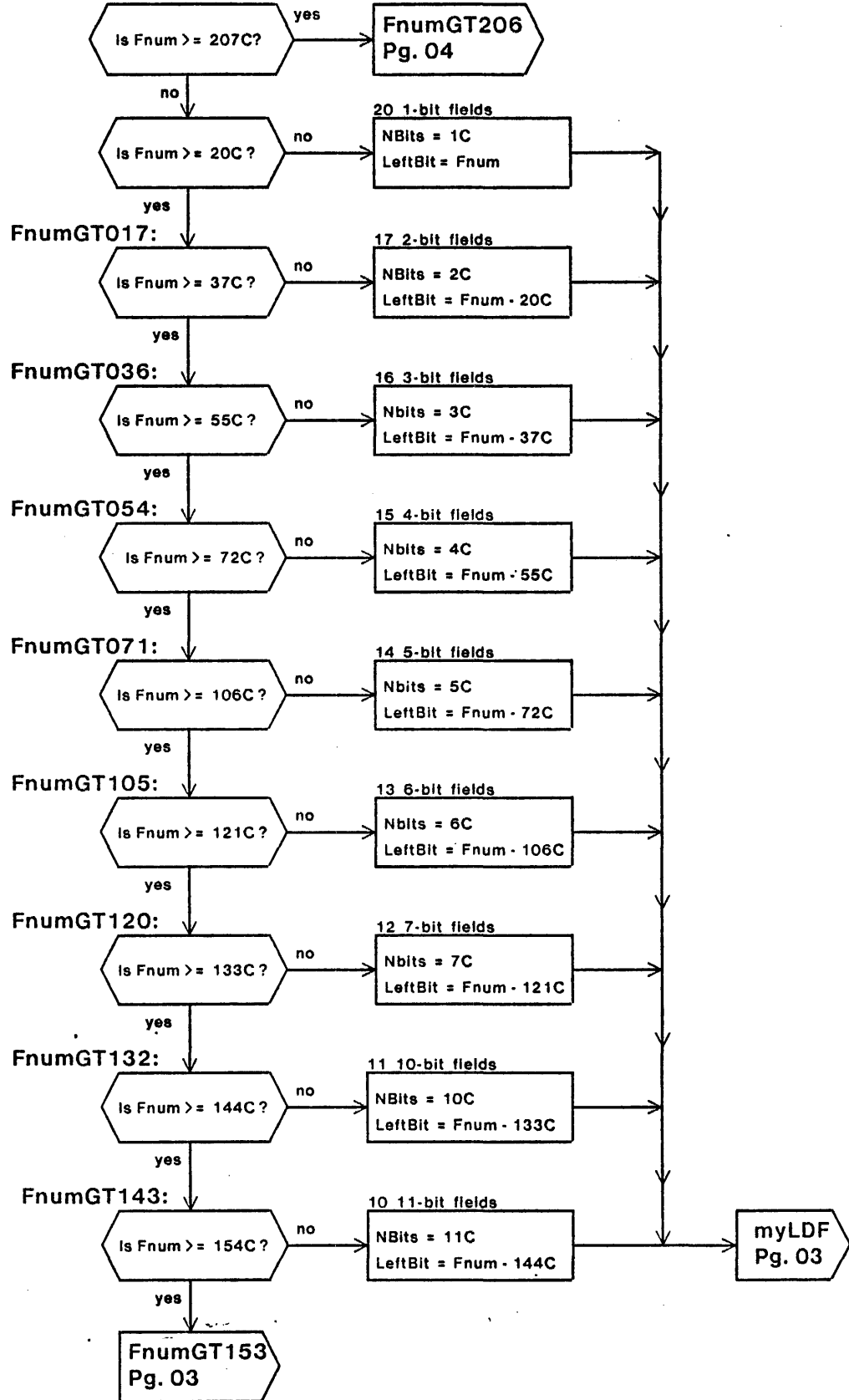
```
END;
```



XEROX ED	D(0) Diagnostic	PROGRAM NAME EDCYM	DOCUMENTATION FILE EDCym01.sil	DESIGNER Spaur	REV 1	DATE 1/8/80	PAGE 01
-------------	--------------------	-----------------------	-----------------------------------	-------------------	----------	----------------	------------

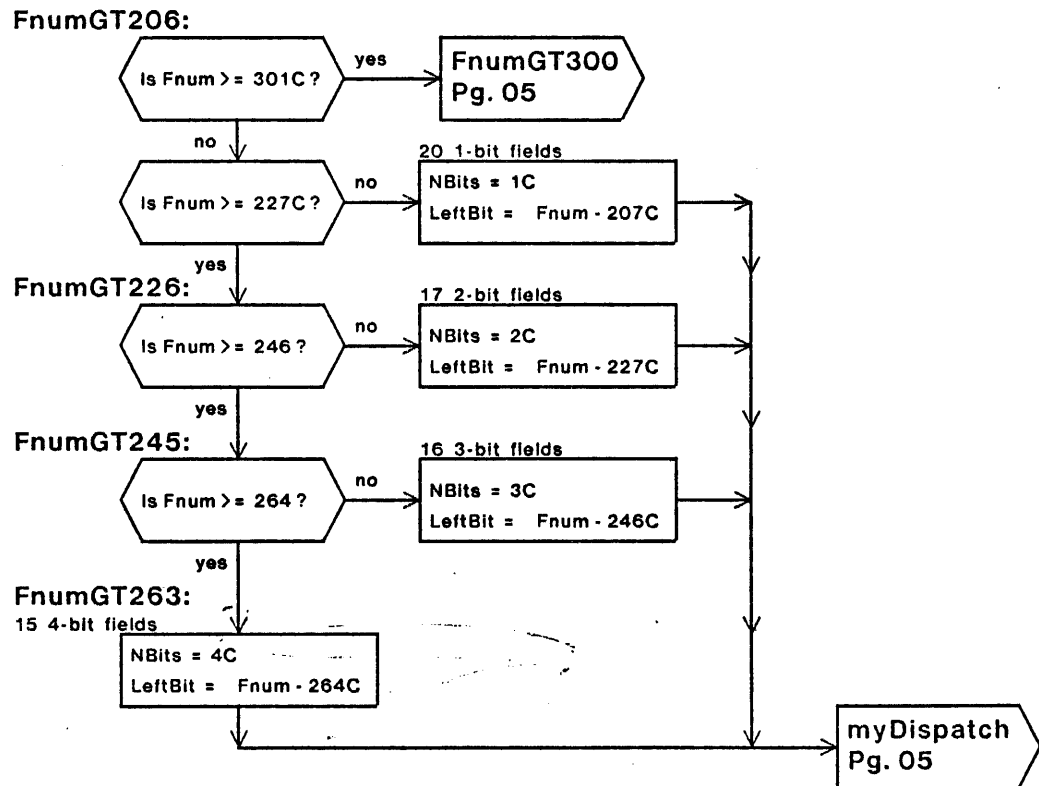
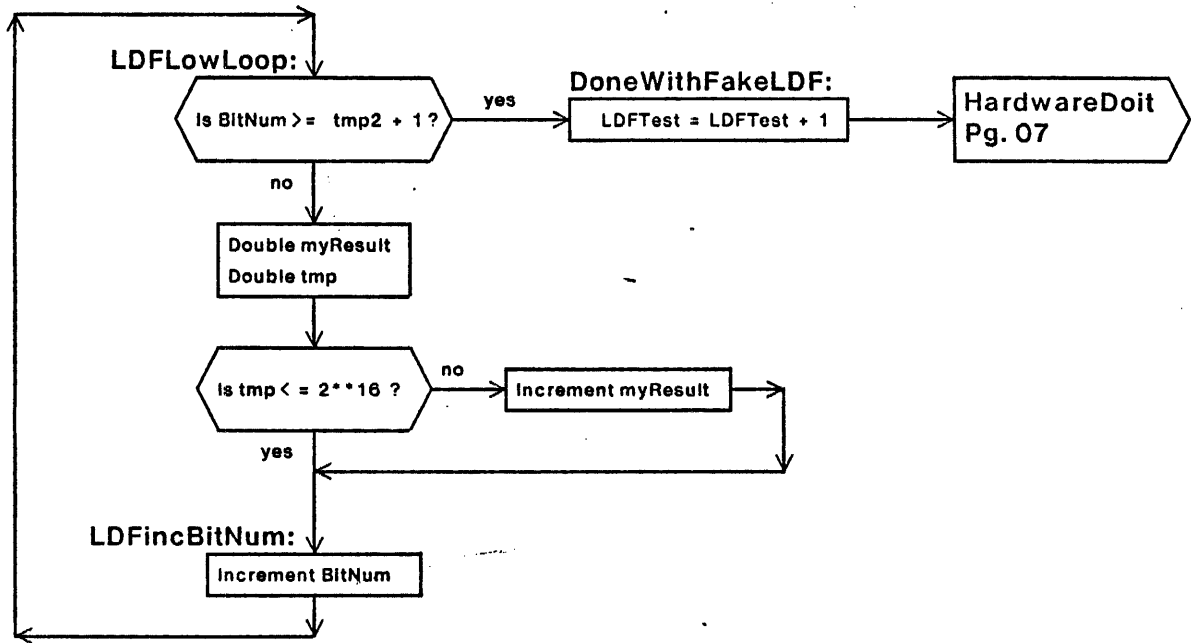
**SubTest2**

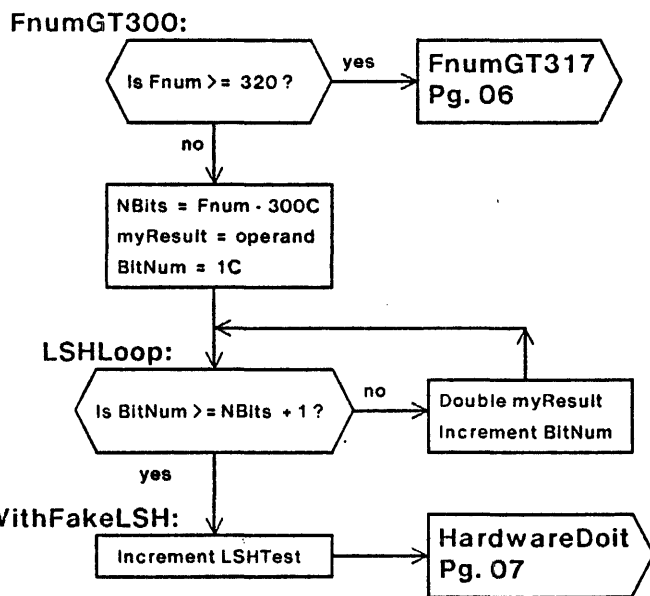
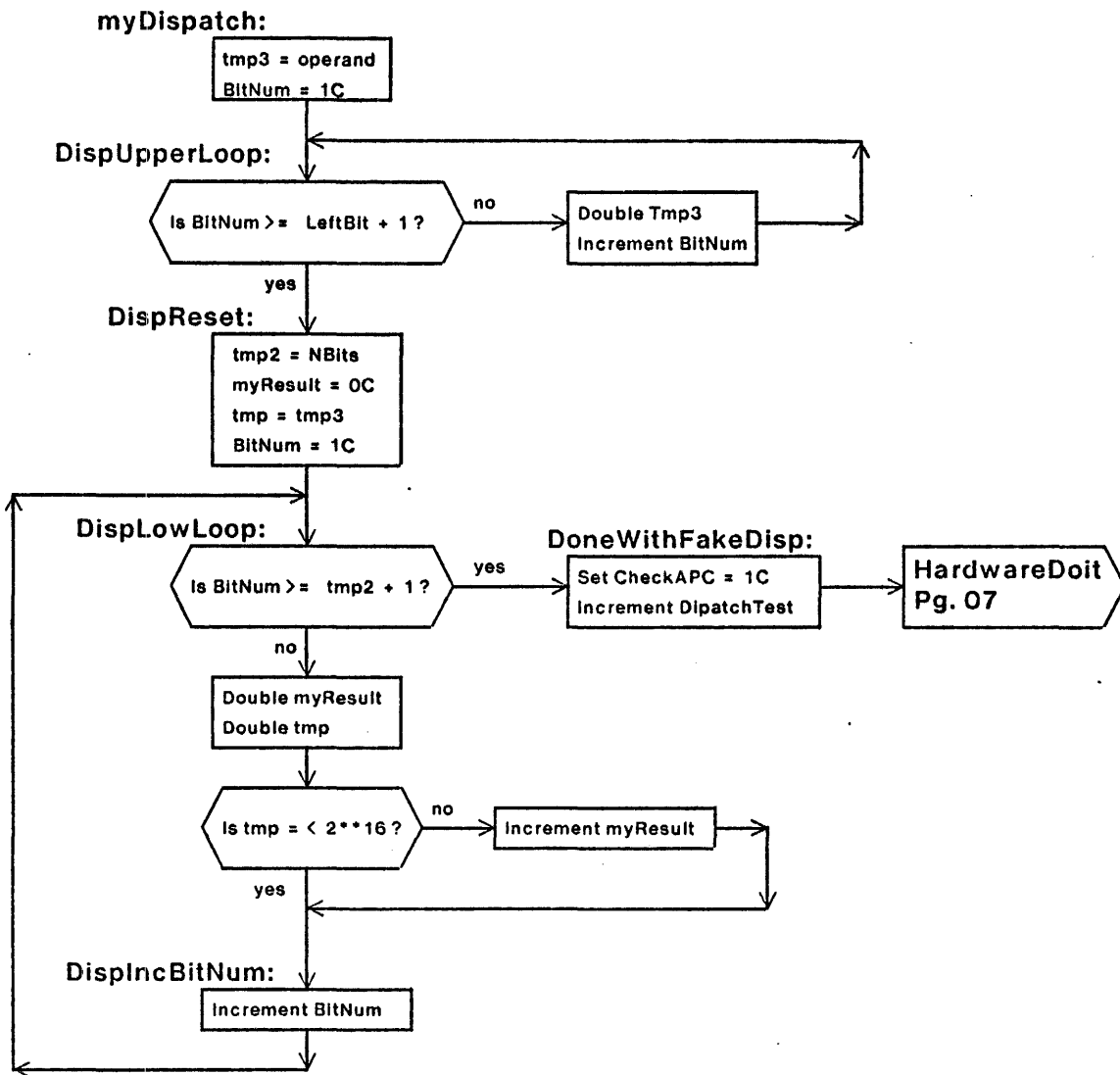
**SubTest2:**

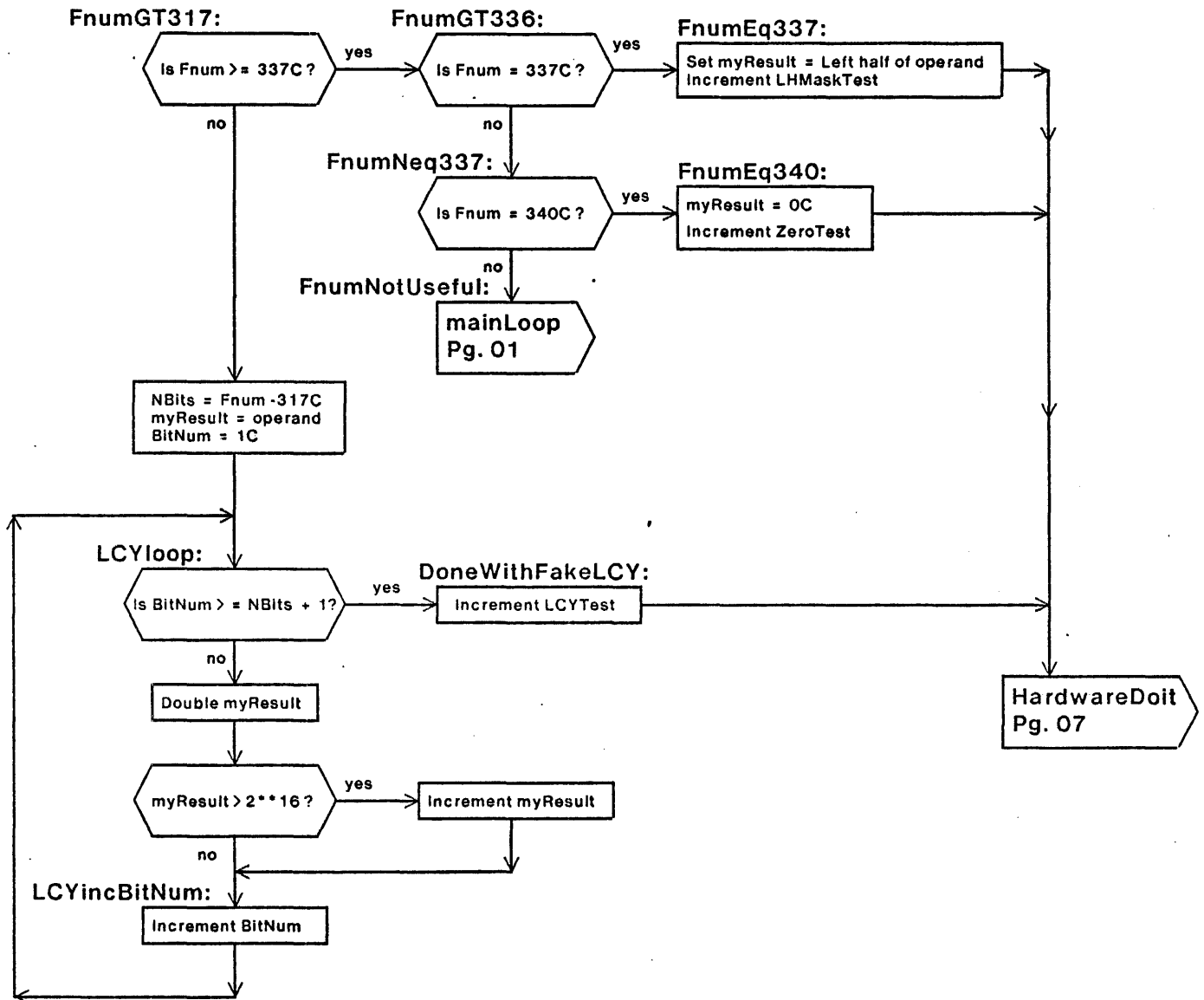


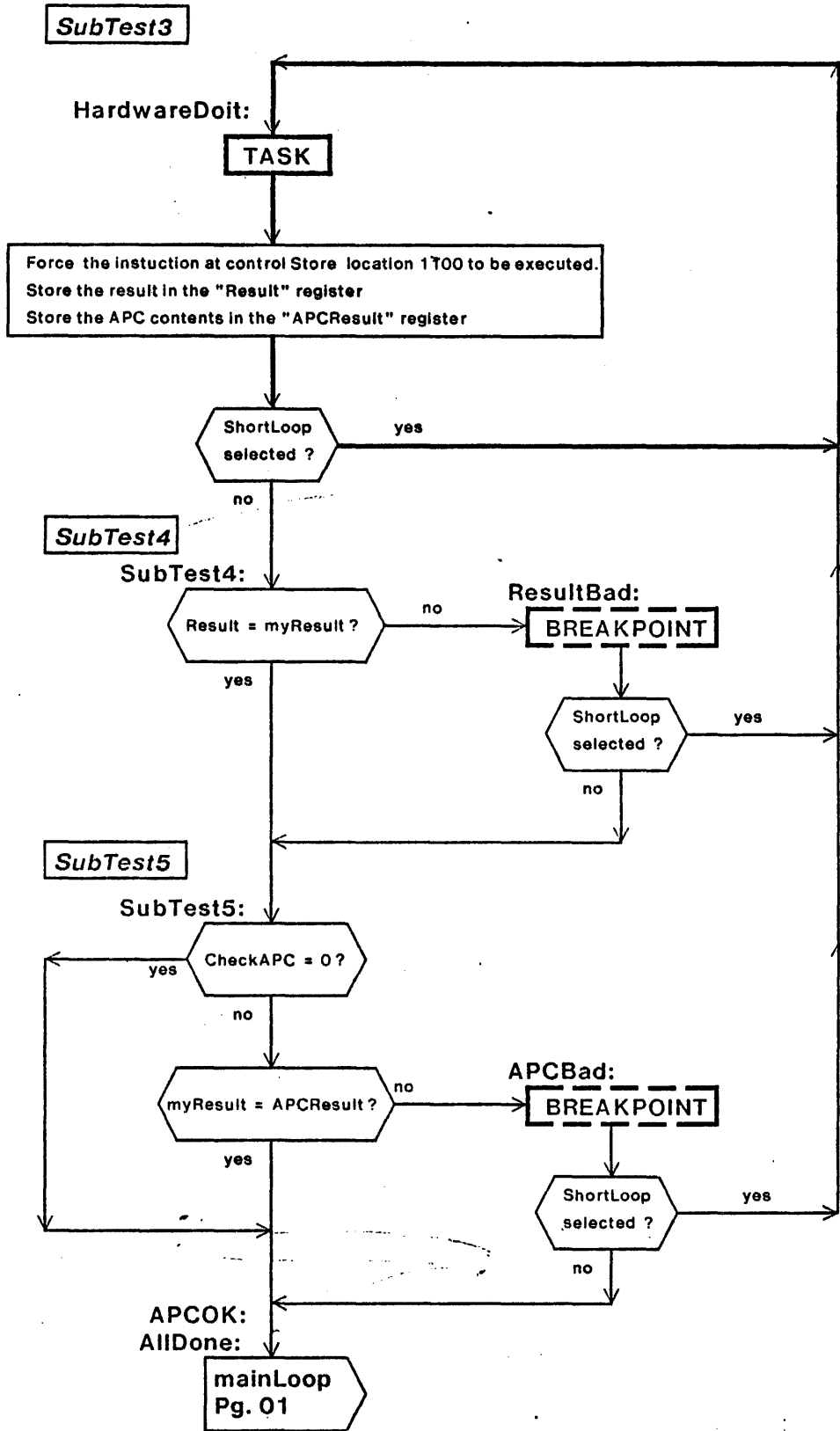












PARITY	0	REVISION	1	COMM-ER0	0
CYCLECONTROL	377	RUN-TIME	7	COMM-ER1	0
PCXREG	16	PASSCOUNT	0	COMM-ER2	0
*PCFREG	6	MAXPASS	2	BOOT-ERR	0
DBREG	15	SUBTEST	0	*BOOTREASON	40
SBREG	17			MEMSYNDROME	170167
MNBR	5003				
*SSTKP	377	*XA	123	MYRESULT	0
STKP	0	FNUM	0	RESULT	0
*ALURESULT	3	OPERAND	0	APCRESULT	0
SALUF	0				
T 20	7000	INNERLOOPCOUN	0	CS0	0
AATOVA	0			CS1	0
TPC 20	7777			CS2	0
CALLER	ILC0+7216	LDFTEST	0	CSP	0
*PAGE	3	DISPATCHTEST	0		
*APC	7011	LSHTEST	0	SHORTLOOP	0
*APCTASK	16	LCYTEST	0		
*CIA	60+1	LHMASKTEST	0		
CTASK	0	ZEROTEST	0		

Loaded: EDCYM

Time: 09.31

Step at 0:60, BP at 0:60+1

Exit Boot Run-Prog Read-Cmds Break UnBreak C1rAddedBPs C1rA11BPs ShowBPs Go  
 88 **Continue** Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

MicroD 8.6 (OS 16) of April 27, 1979  
at 20-Feb-80 8:22:11

microd.run EDCym

EDCym.DIB 561b instructions written 20-Feb-80 8:21:02

Total of 561b instructions

Checking for errors...  
Linking...  
Building allocation lists...  
Assigning locations...  
561b instructions in rings involving ONPAGE or AT  
Reloading binaries...  
Checking assignment...  
Writing .MB file...  
Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
EDCym.DIB:					
0	1404	22005	107060	15	GO START
1	1530	22320	101057	15	(+1)
2	1527	22001	123055	11	(+2)
3	1526	22323	115053	11	(+3)
4	1525	47	7051	1	(+4)
5	1524	20020	101046	11	(+5)
6	1523	20020	101044	1	(+6)
7	1522	10020	101043	11	(+7)
10	1521	10020	101041	15	(+10)
11	1520	12020	101037	1	(+11)
12	1517	12020	101035	5	(+12)
13	1516	12020	101033	11	(+13)
14	1515	12020	101031	15	(+14)
15	1514	10004	101026	5	(+15)
16	1513	10320	105024	5	(+16)
17	1512	50	25023	1	(+17)
20	1401	47	5023	0	BIGLOOP
21	1411	21050	125020	0	(+1)
22	1410	20150	65017	4	(+2)
23	1407	21450	25015	0	(+3)
24	1406	50	24205	0	(+4)
25	1403	50	25023	1	(+5)
26	b 1402	50	25012	0	PASSED-EDCYM-TEST
27	1405	20020	101022	1	(+1)
30	1511	22020	101020	1	MAINLOOP
31	1510	22020	101017	5	(+1)
32	1507	21050	125015	11	(+2)
33	1506	50	24002	0	(+3)
34	1400	22150	65013	15	(+4)
35	1505	23174	45010	15	(+5)
36	1504	23174	67007	15	(+6)
37	1503	23150	65004	11	(+7)
40	1502	22050	125003	15	(+10)
41	1501	22000	103001	1	SETVARS
42	1500	22165	67177	14	(+1)
43	1477	24050	125174	0	(+2)
44	1476	20150	65172	0	(+3)
45	1475	23150	65171	14	(+4)
46	1474	24050	125167	4	(+5)
47	1473	10150	65164	4	(+6)
50	1472	26050	125162	14	(+7)
51	1471	20	41161	0	GETCS
52	1470	26147	21156	14	(+1)
53	1467	47	35554	0	(+2)
54	1466	54150	65153	14	(+3)
55	1465	26050	125150	4	(+4)
56	1464	0	43146	0	(+5)
57	1463	26147	21144	14	(+6)
60	1462	47	35543	0	(+7)
61	1461	54150	65141	14	(+10)

62	1460	26050	125136	10	(+11)
63	1457	0	47135	0	(+12)
64	1456	26147	21133	14	(+13)
65	1455	47	35531	0	(+14)
66	1454	54150	65127	14	(+15)
67	1453	26050	125125	14	(+16)
70	1452	26162	133122	14	(+17)
71	1451	24163	53120	0	CHANGEFIELDS
72	1450	16050	125116	0	(+1)
73	1447	14014	107114	14	(+2)
74	1446	14150	11113	14	(+3)
75	1445	16151	65110	0	(+4)
76	1444	26353	125106	4	(+5)
77	1443	24163	63104	0	(+6)
100	1442	16050	125102	0	(+7)
101	1441	14002	107101	14	(+10)
102	1440	14150	11077	14	(+11)
103	1437	16151	65074	0	(+12)
104	1436	26353	125073	10	(+13)
105	1435	26150	65071	4	CALCPAR
106	1434	10050	125066	0	(+1)
107	1433	26150	65065	10	(+2)
110	1432	10450	165063	0	(+3)
111	1431	26463	63061	14	(+4)
112	1430	10465	167057	0	(+5)
113	1427	10463	153055	0	(+6)
114	1426	10461	171052	0	(+7)
115	1425	10760	175051	0	(+10)
116	1424	10160	77046	0	(+11)
117	1423	26450	125044	10	(+12)
120	1422	26150	65043	14	WRITECS
121	1421	26150	25040	4	(+1)
122	1420	10147	21036	4	(+2)
123	1417	47	31434	0	(+3)
124	1416	26150	25032	10	(+4)
125	1415	10147	21030	4	(+5)
126	1414	47	33427	0	(+6)
127	1413	45	3024	0	(+7)
130	1412	50	25110	0	(+10)
131	444	22000	105001	2	SUBTEST2
132	600	10	57176	1	(+1)
133	577	25450	25174	1	(+2)
134	576	50	24304	0	(+3)
135	443	25401	1173	1	(+4)
136	575	50	24265	0	(+5)
137	433	24000	103005	11	(+6)
140	502	24150	65002	1	(+7)
141	501	24050	125000	15	(+10)
142	500	50	25176	0	(+11)
143	432	25401	37170	1	FNUMGT017
144	574	50	24260	0	(+1)
145	431	24000	105012	11	(+2)
146	505	25401	41010	1	(+3)
147	504	24050	125006	15	(+4)
150	503	50	25176	0	(+5)
151	430	25402	33166	1	FNUMGT036
152	573	50	24255	0	(+1)
153	427	24000	107020	11	(+2)
154	510	25401	77016	1	(+3)
155	507	24050	125014	15	(+4)
156	506	50	25176	0	(+5)
157	426	25403	25165	1	FNUMGT054
160	572	50	24250	0	(+1)
161	425	24000	111027	11	(+2)
162	513	25402	73024	1	(+3)
163	512	24050	125022	15	(+4)
164	511	50	25176	0	(+5)
165	424	25404	15162	1	FNUMGT071
166	571	50	24244	0	(+1)
167	423	24000	113034	11	(+2)
170	516	25403	65033	1	(+3)
171	515	24050	125030	15	(+4)
172	514	50	25176	0	(+5)
173	422	25405	3161	1	FNUMGT105
174	570	50	24241	0	(+1)
175	421	24000	115043	11	(+2)

176	521	25404	55040	1	(+3)
177	520	24050	125036	15	(+4)
200	517	50	25176	0	(+5)
201	420	25405	27157	1	FNUMGT120
202	567	50	24235	0	(+1)
203	417	24000	117050	11	(+2)
204	524	25405	43046	1	(+3)
205	523	24050	125044	15	(+4)
206	522	50	25176	0	(+5)
207	416	25406	11154	1	FNUMGT132
210	566	50	24230	0	(+1)
211	415	24000	121056	11	(+2)
212	527	25405	67054	1	(+3)
213	526	24050	125053	15	(+4)
214	525	50	25176	0	(+5)
215	414	25406	31153	1	FNUMGT143
216	565	50	24224	0	(+1)
217	413	24000	123064	11	(+2)
220	532	25406	51062	1	(+3)
221	531	24050	125060	15	(+4)
222	530	50	25176	0	(+5)
223	412	25407	7151	1	FNUMGT153
224	564	50	24221	0	(+1)
225	411	24000	125073	11	(+2)
226	535	25406	71071	1	(+3)
227	534	24050	125066	15	(+4)
230	533	50	25176	0	(+5)
231	410	25407	23146	1	FNUMGT162
232	563	50	24214	0	(+1)
233	407	24000	127101	11	(+2)
234	540	25407	47076	1	(+3)
235	537	24050	125074	15	(+4)
236	536	50	25176	0	(+5)
237	406	25407	35144	1	FNUMGT170
240	562	50	24211	0	(+1)
241	405	24000	131106	11	(+2)
242	543	25407	63105	1	(+3)
243	542	24050	125102	15	(+4)
244	541	50	25176	0	(+5)
245	404	10	45143	1	FNUMGT175
246	561	25450	25141	1	(+1)
247	560	50	24205	0	(+2)
250	403	24000	133115	11	(+3)
251	546	25407	75113	1	(+4)
252	545	24050	125110	15	(+5)
253	544	50	25176	0	(+6)
254	402	10	53136	1	FNUMGT201
255	557	25450	25135	1	(+1)
256	556	50	24200	0	(+2)
257	401	24000	135125	11	(+3)
260	552	10	45123	1	(+4)
261	551	25450	65120	1	(+5)
262	550	24050	125116	15	(+6)
263	547	50	25176	0	(+7)
264	400	24000	137133	11	FNUMGT204
265	555	10	53130	1	(+1)
266	554	25450	65126	1	(+2)
267	553	24050	125177	14	(+3)
270	477	24150	65174	4	MYLDF
271	476	16050	125172	10	(+1)
272	475	0	43170	0	(+2)
273	474	26050	125167	0	(+3)
274	473	24150	65157	14	LDUPPERLOOP
275	467	27550	25154	0	(+1)
276	466	50	24271	0	(+2)
277	435	16150	65164	10	(+3)
300	472	17150	125163	10	(+4)
301	471	27050	125160	0	(+5)
302	470	50	25167	0	(+6)
303	434	24150	65153	10	LDRESET
304	465	16050	125150	4	(+1)
305	464	14020	101146	0	(+2)
306	463	16150	65145	10	(+3)
307	462	16050	125143	0	(+4)
310	461	0	43140	0	(+5)
311	460	26050	125137	0	(+6)



312	457	16150	65123	4	LDFLOWLOOP
313	451	27550	25120	0	(+1)
314	450	50	24301	0	(+2)
315	441	14150	65134	0	(+3)
316	456	15150	125133	0	(+4)
317	455	16150	65130	0	(+5)
320	454	17150	125126	0	(+6)
321	453	50	24076	0	(+7)
322	437	15050	125074	0	(+10)
323	436	27050	125124	0	LDFINCBITNUM
324	452	50	25137	0	(+1)
325	440	11050	125116	10	DONEWITHFAKELDF
326	447	45	5115	0	(+1)
327	446	50	25074	0	(+2)
330	442	45	5113	0	FNUMGT206
331	445	50	25125	0	(+1)
332	1052	14	43054	2	(+2)
333	1226	25450	25052	2	(+3)
334	1225	50	24224	0	(+4)
335	1013	11	57142	1	(+5)
336	1161	25450	25141	1	(+6)
337	1160	50	24241	0	(+7)
340	1021	24000	103101	11	(+10)
341	1140	10	57077	1	(+11)
342	1137	25450	65074	1	(+12)
343	1136	24050	125072	15	(+13)
344	1135	50	25070	1	(+14)
345	1020	12	55137	1	FNUMGT226
346	1157	25450	25135	1	(+1)
347	1156	50	24235	0	(+2)
350	1017	24000	105110	11	(+3)
351	1144	11	57106	1	(+4)
352	1143	25450	65104	1	(+5)
353	1142	24050	125102	15	(+6)
354	1141	50	25070	1	(+7)
355	1016	13	51132	1	FNUMGT246
356	1155	25450	25130	1	(+1)
357	1154	50	24230	0	(+2)
360	1015	24000	107121	11	(+3)
361	1150	12	55116	1	(+4)
362	1147	25450	65115	1	(+5)
363	1146	24050	125113	15	(+6)
364	1145	50	25070	1	(+7)
365	1014	24000	111126	11	FNUMGT263
366	1153	13	51125	1	(+1)
367	1152	25450	65123	1	(+2)
370	1151	24050	125071	15	(+3)
371	1134	24150	65066	5	MYDISPATCH
372	1133	16050	125065	11	(+1)
373	1132	0	43062	1	(+2)
374	1131	26050	125061	1	(+3)
375	1130	24150	65051	15	DISPUPPERLOOP
376	1124	27550	25046	1	(+1)
377	1123	50	24244	0	(+2)
400	1023	16150	65057	11	(+3)
401	1127	17150	125055	11	(+4)
402	1126	27050	125053	1	(+5)
403	1125	50	25061	1	(+6)
404	1022	24150	65044	11	DISPRESET
405	1122	16050	125042	5	(+1)
406	1121	14020	101040	1	(+2)
407	1120	16150	65037	11	(+3)
410	1117	16050	125034	1	(+4)
411	1116	0	43032	1	(+5)
412	1115	26050	125031	1	(+6)
413	1114	16150	65015	5	DISPLOWLOOP
414	1106	27550	25013	1	(+1)
415	1105	50	24255	0	(+2)
416	1027	14150	65026	1	(+3)
417	1113	15150	125024	1	(+4)
420	1112	16150	65022	1	(+5)
421	1111	17150	125020	1	(+6)
422	1110	50	24052	0	(+7)
423	1025	15050	125050	0	(+10)
424	1024	27050	125017	1	DISPINCBITNUM
425	1107	50	25031	1	(+1)

426	1026	22000	103011	5	DONEWITHFAKEDISP
427	1104	11050	125006	15	(+1)
430	1103	50	25074	0	(+2)
431	1012	15	41051	2	FNUMGT300
432	1224	25450	25046	2	(+1)
433	1223	50	24214	0	(+2)
434	1007	14	41175	1	(+3)
435	1176	25450	65173	1	(+4)
436	1175	24050	125171	11	(+5)
437	1174	24150	65167	5	(+6)
440	1173	14050	125164	1	(+7)
441	1172	0	43163	1	(+10)
442	1171	26050	125160	1	(+11)
443	1170	24150	65151	11	LSHLOOP
444	1164	27550	25147	1	(+1)
445	1163	50	24221	0	(+2)
446	1011	14150	65156	1	(+3)
447	1167	15150	125154	1	(+4)
450	1166	27050	125152	1	(+5)
451	1165	50	25160	1	(+6)
452	1010	13050	125144	1	DONEWITHFAKELSH
453	1162	50	25074	0	(+1)
454	1006	15	77045	2	FNUMGT317
455	1222	25450	25043	2	(+1)
456	1221	50	24200	0	(+2)
457	1001	14	77027	2	(+3)
460	1213	25450	65024	2	(+4)
461	1212	24050	125023	12	(+5)
462	1211	24150	65021	6	(+6)
463	1210	14050	125017	2	(+7)
464	1207	0	43015	2	(+10)
465	1206	26050	125013	2	(+11)
466	1205	24150	65003	12	LCYLOOP
467	1201	27550	25001	2	(+1)
470	1200	50	24211	0	(+2)
471	1005	14150	65010	2	(+3)
472	1204	15150	125006	2	(+4)
473	1203	50	24007	0	(+5)
474	1003	15050	125005	0	(+6)
475	1002	27050	125005	2	LCYINCBITNUM
476	1202	50	25013	2	(+1)
477	1004	13050	125176	5	DONEWITHFAKELCY
500	1177	50	25074	0	(+1)
501	1000	15	77040	2	FNUMGT336
502	1220	25450	25037	2	(+1)
503	1217	50	24114	0	(+2)
504	1046	24237	77034	6	FNUMEQ337
505	1216	14050	125033	2	(+1)
506	1215	13050	125030	12	(+2)
507	1214	50	25074	0	(+3)
510	1047	16	41137	0	FNUMNEQ337
511	1057	25450	25134	0	(+1)
512	1056	50	24121	0	(+2)
513	1050	14020	101131	0	FNUMEQ340
514	1054	13050	125126	14	(+1)
515	1053	50	25074	0	(+2)
516	1051	45	7133	0	FNUMNOTUSEFUL
517	1055	50	25023	1	(+1)
520	1061	50	25140	0	H1
521	1060	50	25074	0	H2
522	1036	50	25344	0	HARDWAREDOIT
523	1062	22000	107400	0	(+1)
524	1037	24150	65005	5	(+2)
525	1102	14050	125001	5	(+3)
526	@1100	14176	101002	5	(+4)
527	1101	50150	65176	14	(+5)
530	1077	14050	125175	10	(+6)
531	1076	20150	124503	14	(+7)
532	1041	50	25074	0	(+10)
533	1040	50	25173	0	(+11)
534	1075	22000	111170	0	SUBTEST4
535	1074	14150	65166	4	(+1)
536	1073	15450	25164	0	(+2)
537	1072	50	24070	0	(+3)
540	b 1035	50	25146	0	RESULTBAD
541	1063	20150	124506	14	(+1)

542	1043	50	25143	0	(+2)
543	1042	50	25071	0	(+3)
544	1034	22000	113163	0	SUBTEST5
545	1071	22150	25161	4	(+1)
546	1070	50	24064	0	(+2)
547	1033	14150	65153	10	(+3)
550	1065	15450	25151	0	(+4)
551	1064	50	24061	0	(+5)
552 b	1031	50	25154	0	APCBAD
553	1066	20150	124512	14	(+1)
554	1045	50	25140	0	(+2)
555	1044	50	25060	0	(+3)
556	1030	50	25065	0	APCOK
557	1032	45	7156	0	ALLDONE
560	1067	50	25023	1	(+1)

Page 400: 201 locations used, 177 free

Page 1000: 227 locations used, 151 free

Page 1400: 131 locations used, 247 free

RM:

20	0	PASSCOUNT
21	2	MAXPASS
22		INNERLOOPCOUNTER
23	0	SHORTLOOP
24		SUBTEST
25		CHECKAPC
26		CA
27		XA
30		FNUM
31		OPERAND
32		NBITS
33		LEFTBIT
34		BITNUM
35		CS0
36		CS1
37		CS2
40		CSP
41		ADDRESS
42		LDFTEST
43		DISPATCHTEST
44		LSHTEST
45		LCYTEST
46		LHMASKTEST
47		ZEROTEST
50		MYRESULT
51		RESULT
52		APCRESULT
53		STUFFTMP
54		TMP
55		TMP2
56		TMP3
57	1	REVISION
60	7	RUN-TIME
61		RLC@

Time: 12 seconds; 0 error(s), 0 warning(s), 11447 words free

```
.....  
:::EDCymLog.MIDAS : Logger for EDCym program  
:::                               By: M. Spaur                               Jan. 8, 1980  
.....
```

```
.start      L X AppendOutput EDCym.report;  
            L X WriteMessage ~***** START EDCym Test : ;  
            L X WriteDT;  
            L X WriteMessage *****~ ;  
            L X Skip .continue;  
  
.breakpoint L X AppendOutput EDCym.report;  
            L A18 SkipNE RESULTBAD;  
            L X Skip .ResultBad;  
            L A18 SkipNE APCBAD;  
            L X Skip .APCBad;  
            L A18 SkipNE PASSED-EDCYM-TEST;  
            L X Skip .passtest;  
  
.notmybreak L X AppendOutput EDCym.report;  
            L X WriteMessage *** FAILED: Not at my breakpoint ~;  
  
            L X WriteMessage ' Parity = ;  
            R A0 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CIA = ;  
            R A18 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CTASK = ;  
            R A19 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APCTASK = ;  
            R A17 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APC = ;  
            R A16 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' TPC = ;  
            R A13 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X CloseOutput;  
            L X Exit;  
  
.ResultBad L X WriteMessage *** FAILED: at my Breakpoint ~;  
            L X WriteMessage * Result does not equal to myResult ~;  
            L X WriteMessage ' Result = ;  
            R C9 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
            L X WriteMessage ' myResult = ;  
            R C8 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
.bad       L X WriteMessage ' SUBTEST = ;  
            R B4 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' PASSCOUNT = ;  
            R B2 Val;  
            L X WriteMessage;
```

```
L X WriteMessage ~:
L X Skip .continue;
.APCBad L X WriteMessage *** FAILED: at my Breakpoint ~:
L X WriteMessage * APCResult does not equal to myResult ~:
L X WriteMessage ' APCResult = ;
R C10 Val;
L X WriteMessage;
L X WriteMessage ~;
L X WriteMessage ' myResult = ;
R C8 Val;
L X WriteMessage;
L X WriteMessage ~;
L X BackSkip .bad;
.passtest L X WriteMessage ~----- Passed EDCym Test : ;
L X WriteDT;
L X WriteMessage ----- ;
L X Skip .continue;
.continue L X WriteMessage ~;
L X CloseOutput;
L X DisplayOn;
L X Confirm;
L X TimeOut 1000000000;
L X Continue;
L X Skip 2;
L X ShowError Program failed to CONTINUE.;
L X BackSkip .notmybreak;
L X DisplayOff;
L X BackSkip .breakpoint;
```

L A19 Val 0  
L X Confirm  
L X Load EDCYM;

L B0 Addr REVISION;  
L B1 Addr RUN-TIME;  
L B2 Addr PASSCOUNT;  
L B3 Addr MAXPASS;  
L B4 Addr SUBTEST;  
L B7 Addr XA;  
L B8 Addr FNUM;  
L B9 Addr OPERAND;  
L B11 Addr INNERLOOPCOUNTER;  
L B14 Addr LDFTEST;  
L B15 Addr DISPATCHTEST;  
L B16 Addr LSHTEST;  
L B17 Addr LCYTEST;  
L B18 Addr LHMASKTEST;  
L B19 Addr ZEROTEST;  
L C7 Addr MYRESULT;  
L C8 Addr RESULT;  
L C9 Addr APCRESULT;  
L C11 Addr CS0;  
L C12 Addr CS1;  
L C13 Addr CS2;  
L C14 Addr CSP;  
L C16 Addr SHORTLOOP;

L X DisplayOn;  
L X TimeOut 10000  
L X SS GO  
L X Skip 1  
L X ShowError Single-step at GO hung

%

\*\*\* Revision 1 \*\*\*

```

*****
*** EDField.mc : microcode to test the Mesa field selection operations RF, WFA, WFB
*** Purpose : This test simulates the RF, WFA, and WFB operations without
              using the cyler-masker and then compares the results with
              the actual executions of the corresponding operations.
*** Hardware Configuration : Standard 4 CPU boards.
*** IMPORTANT NOTE : ON VIRGIN SYSTEMS RUN EDFIELD BEFORE EDCYM.
              This EDField test checks hardware that must work properly
              for EDCym to be useful.
*** Approximate run time : twenty seconds.
*** Written by : Tom Horsley, Dec. 13, 1977
*** Modified by : Bill Kennedy, March 10, 1978
              Took code off of Page 0.
*** Modified by : Bill Kennedy, April 24, 1978
              Now uses Maintenance Panel.
*** Modified by : C. Thacker, (date unknown)
              Improved readability.
*** Modified by : C. Thacker, August 22, 1978
              Made XB = -1 always.
*** Rewritten by : J. Kellman, March 6, 1980
              Eliminated all cyler-masker functions from both the
              simulated field selection operations and random number
              generators, as well as standardizing the program format.
*****

```

\*\*\*\*\*  
\* SubTest Description:

- \* SubTest 0: (not really a test) generates and decodes a Mesa Field descriptor.
- \* SubTest 1: simulates and then executes the RF operation (Mesa Read-Field) and compares the results.
- \* SubTest 2: simulates and then executes the WFA operation (Mesa Write-Field A) and compares the results.
- \* SubTest 3: simulates and then executes the WFB operation (Mesa Write-Field B) and compares the results.

\*\*\*\*\*  
\* Subroutine Description:

- \* LeftCycle: uses ALU additions to simulate the LCycle function without using the cyler-masker.
- \* LeftShift: uses ALU additions to simulate the LShift function without using the cyler-masker.

\*\*\*\*\*  
\* Breakpoints:

- \* RFBAD: actual RF result failed to agree with simulated RF result
- \* WFABAD: actual WFA result failed to agree with simulated WFA result
- \* WFBAD: actual WFB result failed to agree with simulated WFB result
- \* Passed-EDField-Test: the system passed thru all the passes of EDField

\*\*\*\*\*  
\* Breakpoint Logic Analyzer Sync Points:

- \* RFBAD: Control Store address 457
- \* WFABAD: Control Store address 451
- \* WFBAD: Control Store address 441
- \* Passed-EDField-Test: Control Store address 401

\*\*\*\*\*

```
*****
* Special Register Definitions:
*
* LoopWithin: At any breakpoint the user may change the value of LoopWithin.
               Setting LoopWithin to the nonzero value N will cause EDField to loop
               endlessly within the subtest N (when it gets to this subtest).
               If LoopWithin is zero (the default) then the program goes straight
               through all the tests as normal.
*
* RandFlag: At any breakpoint the user may change the value of RandFlag
            to 1, which will cause CurrentPattern to take on random values.
            If RandFlag is zero (the default) then CurrentPattern will be set to
            a 16-bit word of all ones during each execution of subtest 0.
*
* XA and XB: The two random numbers held in these registers XA and XB are used to
            choose field descriptor (FDescr) and CurrentPattern values
            as follows:
               FDescr ← XA[0:7]
               CurrentPattern ← XB (only if RandFlag = 1)
*
* InnerLoopCounter: This register contains the number of passes through the mainloop.
                   The mainloop is simply the set of all of the subtests. When
                   InnerLoopCounter = (2**16) - 1, all of the possibilities that are tested
                   by EDField have been tested exactly once.
*
* PassCount: This register displays the number of the current pass of the entire
             EDField test (2**16 iterations of the mainloop). The maintenance
             panel shows the number of the currently running pass.
*
* MaxPass: This register determines how many times the entire EDField test will repeat.
*****
%
```



```

*****
* INITIALIZATION:

BUILTIN[INSERT,24]:
INSERT[d01ang];
TITLE[MesaFieldTest]:

***** R-registers: *****

RV[Revision,1,1];      * revision number is 1 for this program
RV[Run-Time,2,24];    * run time for this test is twenty seconds
RV[PassCount,3];      * pass count for this program
RV[MaxPass,4,2];      * maximum number of passes for this run
RV[SubTest,5];        * current location of test
RV[InnerLoopCounter,6]; * inner loop counter

RV[LoopWithin,10,0];  * set to N to loop on subtest n
RV[RandFlag,11,0];    * 0 => CurrentPattern always 177777, 1 => random pattern

RV[CurrentPattern,12]; * holds the argument for the RF, WFA, and WFB operations
RV[CA,13];            * used in random number generation, A*XA + CA
RV[XA,14];            * random number generated via A*XA + CA
RV[CB,15];            * used in random number generation, A*XB + CB
RV[XB,16];            * random number generated via A*XB + CB
RV[FDescr,17];        * Mesa field descriptor
RV[StartBit,20];      * start field of mesa field descriptor (first 4 bits)
RV[EndBit,21];        * last bit of mesa field
RV[Length,22];        * actual length of mesa field from descriptor (last 4 bits plus 1)
RV[ShiftIndex,23];    * index register in LeftCycle and LeftShift
RV[ShiftSize,24];     * number of bits to be shifted in simulated shift (or cycle)
RV[ShiftValue,25];    * holds word to be shifted in simulated shift (or cycle)
RV[Mask,26];          * the mask used in simulating the RF, WFA, and WFB operations
RV[SimResult,27];     * result of simulated operations
RV[Result,30];        * holds the hardware result of current operation

*****
*** MAIN routine:

      SET[MainPage,1];      * set label for Main Program page
      ONPAGE[MainPage]:

go:
start:
* Initialize random generator registers: XA ← 123, CA ← 33031
  XA ← AND@[0377, 123]C;
  XA ← (XA) OR (AND@[177400, 123]C);

  CA ← AND@[0377, 33031]C;
  CA ← (CA) OR (AND@[177400, 33031]C);

* Initialize random generator registers: XB ← 456, CB ← 33035
  XB ← AND@[0377, 456]C;
  XB ← (XB) OR (AND@[177400, 456]C);

  CB ← AND@[0377, 33035]C;
  CB ← (CB) OR (AND@[177400, 33035]C);

  CLEARMPANEL;
  InnerLoopCounter ← 0C: * Initialize inner loop counter
  PassCount ← 0C:      * Initialize outer loop counter

bigLoop: t ← PassCount ← (PassCount) + 1;
  LU ← (MaxPass) - (t);
  GOTO [.+2. ALU >= 0];

Passed-EDField-Test: BREAKPOINT. goto[go]:

      INCMMPANEL;

*** SUBTEST 0 ***
mainloop: SubTest ← 0C;

      LU ← (LoopWithin);

```

```

GOTO [ExtractFDescr, ALU # 0]: * Do next pattern value?

InnerLoopCounter ← (InnerLoopCounter) + 1:
GOTO [bigLoop, CARRY]:

* Random (4005*XA + CA mod 2**16)
t ← XA:
ShiftValue ← t:
ShiftSize ← 2C: * left shift old XA by 2 bits
CALL [LeftShift]:
t ← ShiftValue:
XA ← (XA) + t: * add shifted value to XA
ShiftSize ← 11C: * left shift original old XA by a total of 13C bits
CALL [LeftShift]:
t ← ShiftValue:
t ← (XA) + t: * add shifted value to subtotal
t ← (CA) + t: * add CA
XA ← t: * result is new XA

* Random (4005*XB + CB mod 2**16)
t ← XB:
ShiftValue ← t:
ShiftSize ← 2C: * left shift old XB by 2 bits
CALL [LeftShift]:
t ← ShiftValue:
XB ← (XB) + t: * add shifted value to XB
ShiftSize ← 11C: * left shift original old XB by a total of 13C bits
CALL [LeftShift]:
t ← ShiftValue:
t ← (XB) + t: * add shifted value to subtotal
t ← (CB) + t: * add CB
XB ← t: * result is new XB

LU ← RandFlag:
DBLGOTO [PatternAllOnes, PatternRandom, ALU = 0]:
PatternAllOnes: CurrentPattern ← (ZERO) - 1, goto[ExtractFDescr]:
PatternRandom: CurrentPattern ← t, goto[ExtractFDescr]:

ExtractFDescr: t ← XA: * XA will yield the field descriptor for this test
ShiftValue ← t: * we extract all our descriptor values by
* ALU-simulated cycling and masking
* (to avoid using the cyclor-masker)

ShiftSize ← 10C: * simulate an eight-bit left cycle
CALL [LeftCycle]:

t ← ShiftValue:
FDescr ← t:
FDescr ← (FDescr) and (377C): * mask to get entire field descriptor

DecodeFDescr: t ← FDescr: * now decode this FDescr
ShiftValue ← t:

ShiftSize ← 14C: * simulate a twelve-bit left cycle
CALL [LeftCycle]:

t ← ShiftValue:
StartBit ← t:
StartBit ← (StartBit) and (17C): * mask to get field start bit index

ShiftSize ← 4C: * simulate a further four-bit left cycle
CALL [LeftCycle]:

t ← ShiftValue:
Length ← t:
Length ← (Length) and (17C): * mask to get length field
Length ← (Length) + 1: * adjust length value to actual length

t ← (Length) - 1:
t ← (StartBit) + t:
EndBit ← t: * proper end bit index calculated

LU ← (EndBit) - (20C): * check if end bit makes sense
GOTO [EndBitPastEnd, ALU = 0]:
GOTO [RFTTest]:
EndBitPastEnd: GOTO [maintloop]:

```

## \*\*\* SUBTEST 1 \*\*\*

```

RFTest: SubTest ← 1C:          * first simulate RF

    ShiftValue ← (ZERO) - 1: * form Mask

    t ← Length;
    ShiftSize ← t;
    CALL [LeftShift];

    t ← (ShiftValue) xnor (0C);
    Mask ← t;

    t ← CurrentPattern:      * left cycle current pattern by (Startbit + Length) bits
    ShiftValue ← t;
    t ← Length;
    t ← (StartBit) + t;
    ShiftSize ← t;
    CALL [LeftCycle];

    t ← Mask;                * apply Mask to left-cycled CurrentPattern
    t ← (ShiftValue) and t;
    SimResult ← t;

    CYCLECONTROL ← FDescr;   * now do real RF
    t ← RF[CurrentPattern];
    Result ← t;

    lu ← (LoopWithin) - (1C); * loop within this test?
    GOTO [.,+2, ALU # 0];
    GOTO [RFTest];

    LU ← (SimResult) - (t);  * compare RF results
    GOTO[WFAtest, ALU = 0];
RFBAD:  BREAKPOINT, goto[RFTest];

```

## \*\*\* SUBTEST 2 \*\*\*

```

WFAtest: SubTest ← 2C:          * first simulate WFA

    ShiftValue ← (ZERO) - 1: * form Mask

    t ← Length;
    ShiftSize ← t;
    CALL [LeftShift];

    ShiftValue ← (ShiftValue) xnor (0C);

    t ← (StartBit) - (20C);
    t ← (Length) + t;
    ShiftSize ← (ZERO) - t;
    CALL [LeftShift];

    t ← ShiftValue;
    Mask ← t;

    t ← CurrentPattern:      * left cycle CurrentPattern by 20C - (StartBit + Length) bits
    ShiftValue ← t;
    CALL [LeftCycle];

    t ← Mask;                * apply Mask to left-cycled CurrentPattern
    t ← (ShiftValue) and t;
    SimResult ← t;

    CYCLECONTROL ← FDescr;   * now do real WFA
    t ← WFA[CurrentPattern];
    Result ← t;

    lu ← (LoopWithin) - (2C); * loop within this test?
    GOTO [.,+2, ALU # 0];
    GOTO [WFAtest];

    LU ← (SimResult) - (t);  * compare WFA results
    GOTO[WFBtest, ALU = 0];
WFBAD:  BREAKPOINT, goto[WFAtest];

```

## \*\*\* SUBTEST 3 \*\*\*

```

WFBTest: SubTest ← 3C;          * first simulate WFB
      ShiftValue ← (ZERO) - 1;  * form Mask
      t ← Length;
      ShiftSize ← t;
      CALL [LeftShift];

      ShiftValue ← (ShiftValue) xnor (0C);

      t ← (StartBit) - (20C);
      t ← (Length) + t;
      ShiftSize ← (ZERO) - t;
      CALL [LeftShift];

      t ← ShiftValue;
      Mask ← t;

      t ← CurrentPattern;      * apply inverse Mask to CurrentPattern
      SimResult ← t;

      t ← Mask;
      SimResult ← (SimResult) andnot t;

      t ← Mask;                * insert masked CurrentPattern into XB's pattern
      t ← (XB) and t;
      SimResult ← (SimResult) or t;

      CYCLECONTROL ← FDescr;   * now do real WFB
      t ← (WFB[CurrentPattern]) OR t; * register t still holds XB and Mask
      Result ← t;

      lu ← (LoopWithin) - (3C); * loop within this test?
      GOTO [. +2, ALU # 0];
      GOTO [WFBtest];

      LU ← (SimResult) - (t);   * compare WFB results
      GOTO [. +2, ALU = 0];
WFBBAD: BREAKPOINT.goto[wfbtest];

      GOTO[mainLoop];

```

## \*\*\*\*\* SUBROUTINE: LeftCycle \*\*\*\*\*

```

*
* (to simulate LCycle without using the cyclor-masker)
* This subroutine left cycles ShiftValue by ShiftSize bits.

```

```

LeftCycle: ShiftIndex ← 1C;
CycleLoop: t ← (ShiftSize);
      LU ← (ShiftIndex) - (t) - 1;
      GOTO[CycleEnd, ALU >= 0]; * done with cycles yet?

      t ← ShiftValue;
      ShiftValue ← (ShiftValue) + t; * ShiftValue holds the value to be left cycled
      * simulate a left shift by adding to self

      GOTO[. +2, NOCARRY]; * bit shifted out left comes back into right side
      ShiftValue ← (ShiftValue) + 1;

      ShiftIndex ← (ShiftIndex) + 1;
      GOTO[CycleLoop];

```

```

CycleEnd: RETURN;

```

## \*\*\*\*\* SUBROUTINE: LeftShift \*\*\*\*\*

```

*
* (to simulate LShift without using the cyclor-masker)
* This subroutine left shifts ShiftValue by ShiftSize bits.

```

```

LeftShift: ShiftIndex ← 1C;
Shiftloop: t ← (ShiftSize);

```

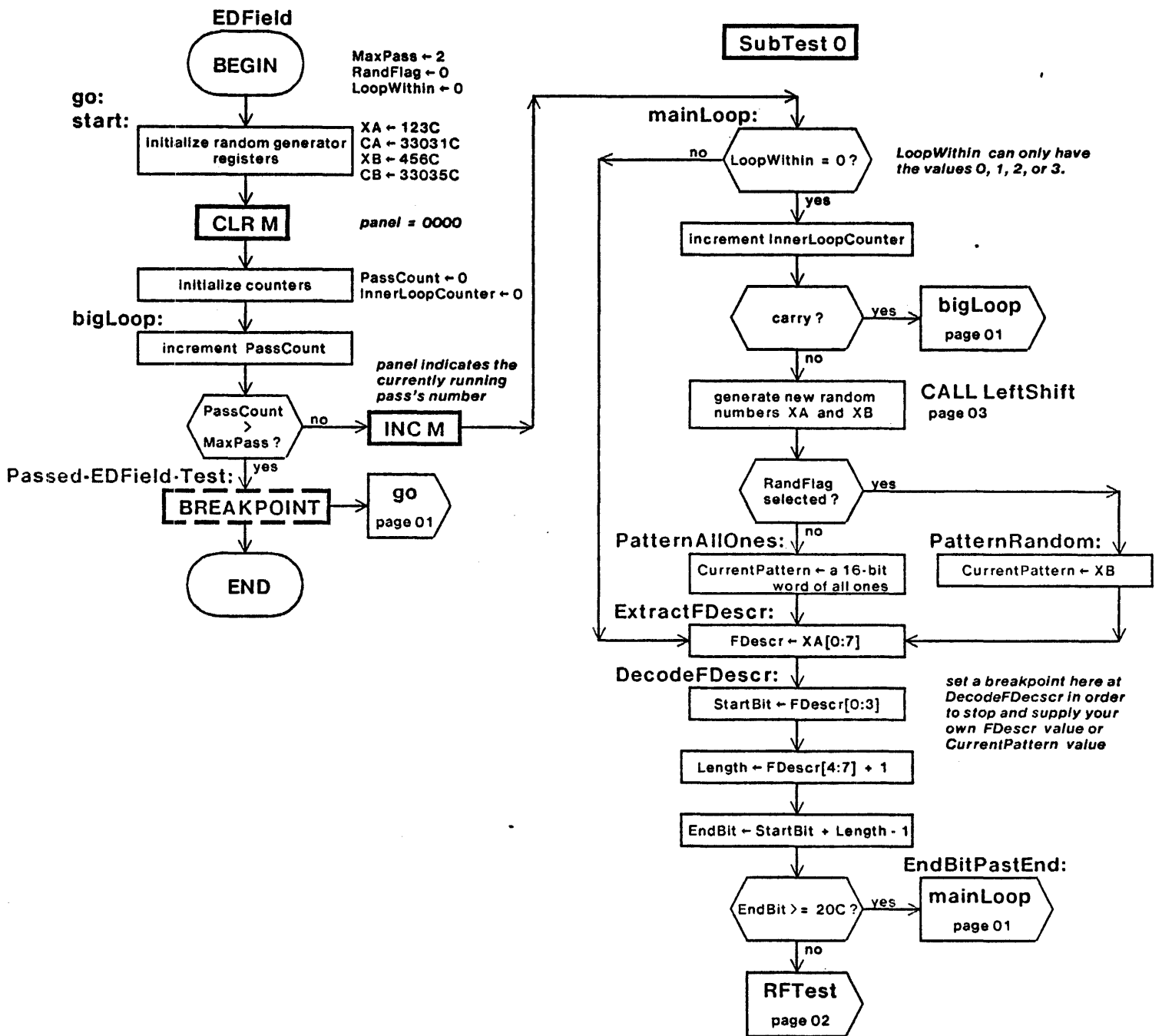
```
LU ← (ShiftIndex) - (t) - 1;
GOTO[ShiftEnd. ALU >= 0];      * done with shifts yet?

t ← ShiftValue;                * ShiftValue holds the value to be left cycled
ShiftValue ← (ShiftValue) + t; * simulate a left shift by adding to self

ShiftIndex ← (ShiftIndex) + 1;
GOTO[ShiftLoop];

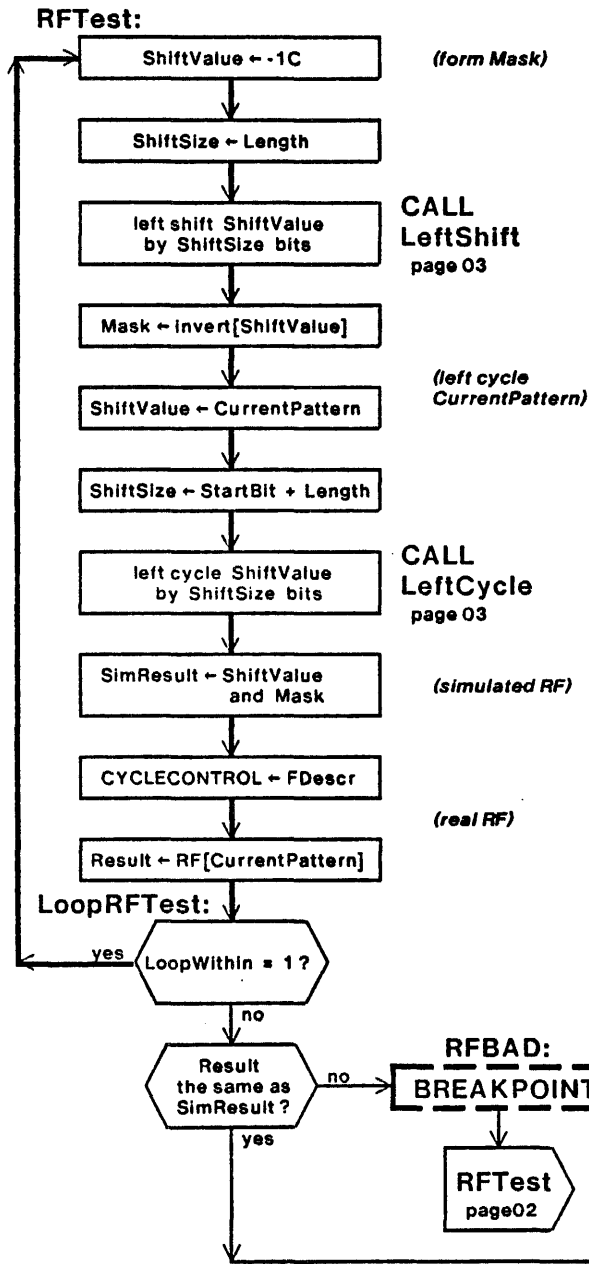
ShiftEnd: RETURN;

END;
```

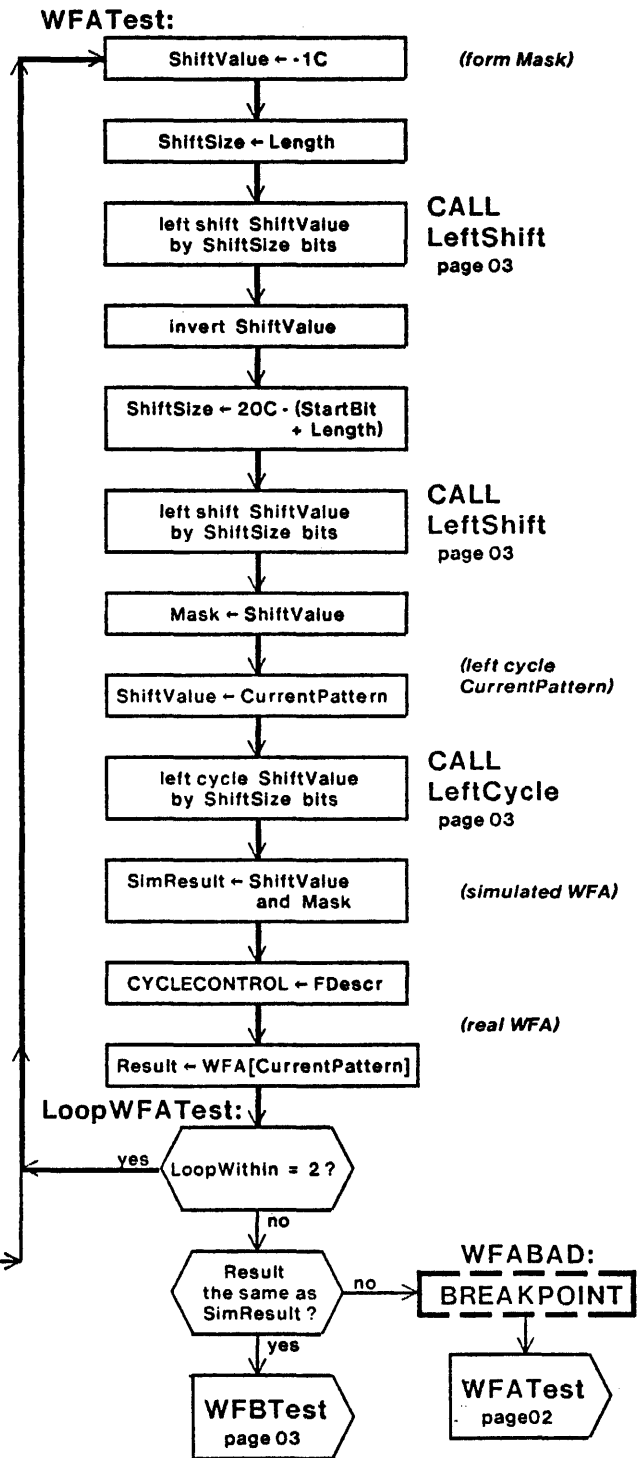


XEROX ED	D(0) Diagnostic	PROGRAM NAME EDField	DOCUMENTATION FILE EDField -01.sil	DESIGNER Kellman	REV 1	DATE 3/6/80	PAGE 01
-------------	--------------------	-------------------------	---------------------------------------	---------------------	----------	----------------	------------

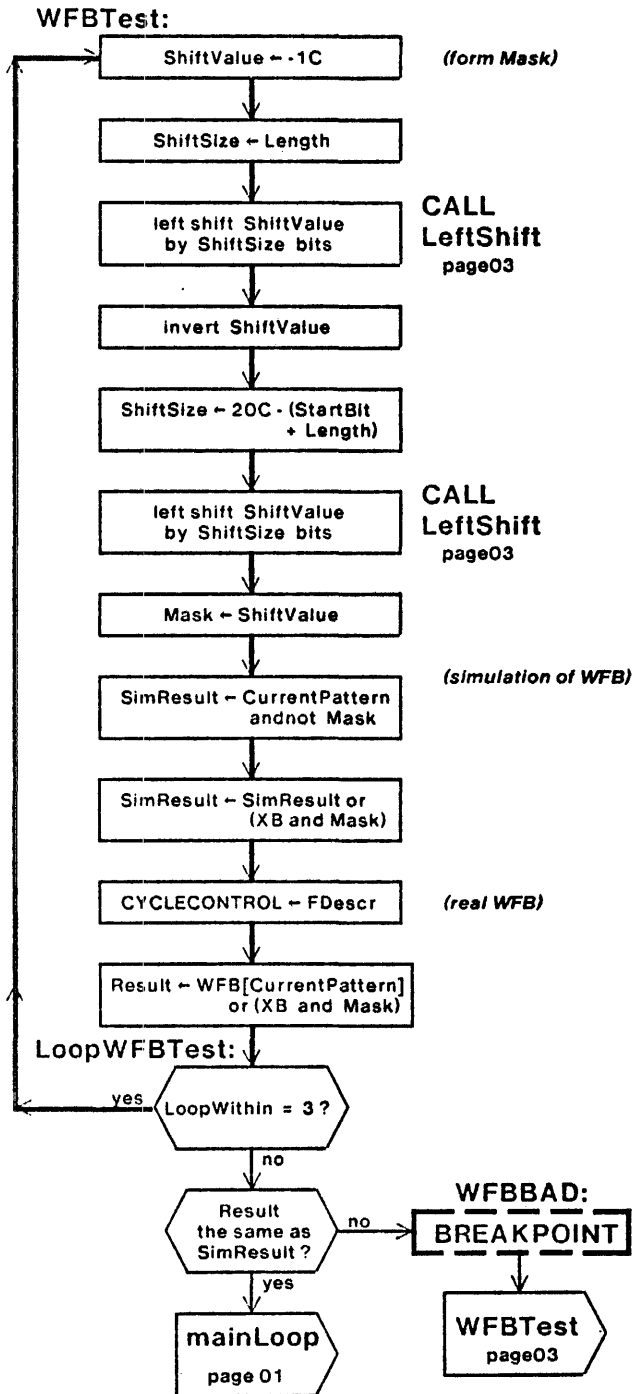
**SubTest 1**



**SubTest 2**



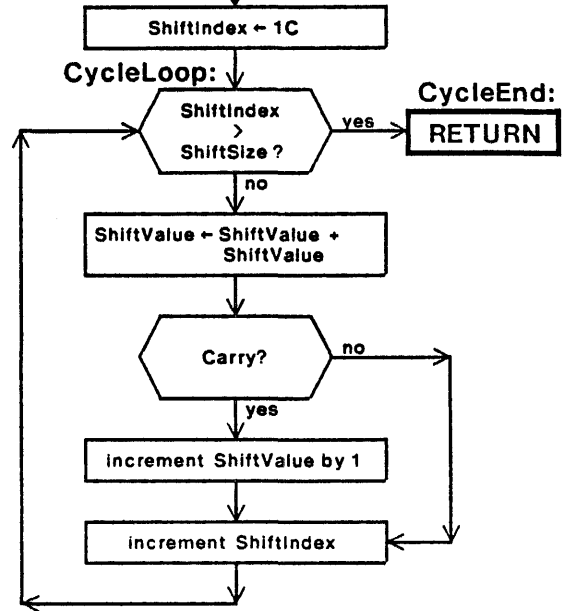
**SubTest 3**



**SUBROUTINE**



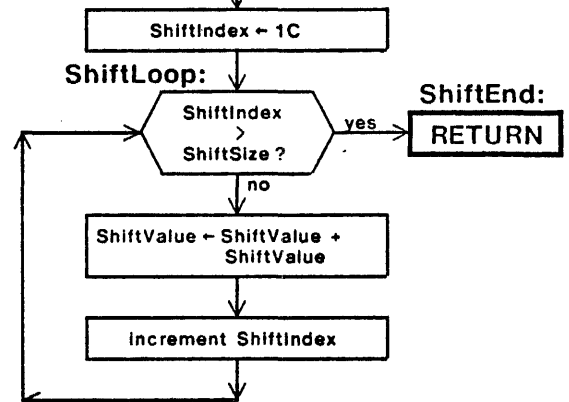
*This subroutine simulates left cycling of a 16-bit word without using the cyclor-masker hardware. It left cycles ShiftValue by ShiftSize bits.*



**SUBROUTINE**



*This subroutine simulates left shifting of a 16-bit word without using the cyclor-masker hardware. It left shifts ShiftValue by ShiftSize bits.*





PARITY	0	REVISION	1	COMM-ER0	0
CYCLECONTROL	63	RUN-TIME	24	COMM-ER1	0
PCXREG	0	PASSCOUNT	0	COMM-ER2	0
PCFREG	0	MAXPASS	2	BOOT-ERR	0
DBREG	20	SUBTEST	0	*BOOTREASON	40
SBREG	40	INNERLOOPCOUN	0	MEMSYNDROME	177777
MNBR	4200				
*SSTKP	377				
STKP	0	FDESCR	0		
*ALURESULT	3	STARTBIT	0		
*SALUF	377	ENDBIT	0		
T 20	7000	LENGTH	0		
AATOVA	0				
TPC 20	7777			RANDFLAG	0
CALLER	ILC@+7516	CURRENTPATTER	0		
*PAGE	1	XB	0	LOOPWITHIN	0
*APC	7011	SIMRESULT	0		
*APCTASK	16	RESULT	0		
*CIA	60+1				
CTASK	0				

Loaded: EDFIELD

Time: 10.33

Step at 0:60, BP at 0:60+1

Exit Boot Run-Prog Read-Cmds Break UnBreak C1rAddedBPs C1rA11BPs ShowBPs Go  
 SS Continue Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

MicroD 8.6 (OS 16) of April 27, 1979  
at 6-Mar-80 12:10:51

microd.run EDfield.dib

EDfield.dib 261b instructions written 6-Mar-80 12:10:08

Total of 261b instructions

Checking for errors...

Linking...

Building allocation lists...

Assigning locations...

261b instructions in rings involving ONPAGE or AT

Reloading binaries...

Checking assignment...

Writing .MB file...

Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
EDfield.dib:					
0	470	36005	107140	2	GO START
1	660	36320	101136	2	(+1)
2	657	34001	123134	16	(+2)
3	656	34323	115132	16	(+3)
4	655	36002	135130	12	(+4)
5	654	36320	103127	12	(+5)
6	653	36001	133124	6	(+6)
7	652	36323	115123	6	(+7)
10	651	47	7121	2	(+10)
11	650	32020	101117	12	(+11)
12	647	30020	101006	14	(+12)
13	403	31050	165115	16	BIGLOOP
14	646	33450	25113	2	(+1)
15	645	50	24200	0	(+2)
16	b 401	50	25161	0	PASSED-EDFIELD-TEST
17	400	47	5110	2	(+1)
20	644	32020	101064	6	MAINLOOP
21	632	34150	25062	2	(+1)
22	631	50	24061	0	(+2)
23	430	33050	125163	11	(+3)
24	571	50	24007	0	(+4)
25	402	36150	65160	1	(+5)
26	570	22050	125156	5	(+6)
27	567	22000	105154	0	(+7)
30	466	50	25354	1	(+10)
31	467	22150	65165	4	(+11)
32	472	37150	125163	0	(+12)
33	471	22000	123150	0	(+13)
34	464	50	25354	1	(+14)
35	465	22150	65000	5	(+15)
36	500	37150	65177	0	(+16)
37	477	35150	65175	14	(+17)
40	476	36050	125172	0	(+20)
41	475	36150	65171	10	(+21)
42	474	22050	125167	4	(+22)
43	473	22000	105145	0	(+23)
44	462	50	25354	1	(+24)
45	463	22150	65005	5	(+25)
46	502	37150	125002	11	(+26)
47	501	22000	123054	0	(+27)
50	426	50	25354	1	(+30)
51	427	22150	65175	5	(+31)
52	576	37150	65172	11	(+32)
53	575	37150	65171	5	(+33)
54	574	36050	125166	11	(+34)
55	573	34150	25164	5	(+35)
56	572	50	24051	0	(+36)
57	424	35376	101062	10	PATTERNALLONES
58	425	34050	125062	10	PATTERNRANDOM
59	431	36150	65061	2	EXTRACTFDESCR

62	630	22050	125057	6	(+1)
63	627	22000	121044	0	(+2)
64	422	50	25255	2	(+3)
65	423	22150	65006	6	(+4)
66	603	36050	125005	16	(+5)
67	602	36217	137002	16	(+6)
70	601	36150	65001	16	DECODEFDESCR
71	600	22050	125177	5	(+1)
72	577	22000	131040	0	(+2)
73	420	50	25255	2	(+3)
74	421	22150	65014	6	(+4)
75	606	20050	125013	2	(+5)
76	605	20200	137011	2	(+6)
77	604	22000	111014	0	(+7)
100	406	50	25255	2	(+10)
101	407	22150	65104	6	(+11)
102	642	20050	125102	12	(+12)
103	641	20200	137100	12	(+13)
104	640	21050	125076	12	(+14)
106	637	21350	65075	12	(+15)
106	636	21150	65073	2	(+16)
107	635	20050	125071	6	(+17)
110	634	21401	1066	6	(+20)
111	633	50	24331	0	(+21)
112	455	50	25032	1	(+22)
113	454	50	25111	2	ENDBITPASTEND
114	515	32000	103013	5	RFTEST
115	505	23376	101010	5	(+1)
116	504	20150	65007	11	(+2)
117	503	22050	125034	0	(+3)
120	416	50	25354	1	(+4)
121	417	22720	41031	6	(+5)
122	614	22050	125026	12	(+6)
123	613	34150	65025	12	(+7)
124	612	22050	125023	6	(+10)
125	611	20150	65020	12	(+11)
126	610	21150	65016	2	(+12)
127	607	22050	125141	0	(+13)
130	460	50	25255	2	(+14)
131	461	22150	65030	11	(+15)
132	514	22250	65027	5	(+16)
133	513	22050	125024	15	(+17)
134	512	36150	11023	15	(+20)
135	511	34154	65021	11	(+21)
136	510	24050	125017	1	(+22)
137	507	35400	3015	1	(+23)
140	506	50	24114	0	(+24)
141	446	50	25032	1	(+25)
142	447	23450	25061	15	(+26)
143	530	50	24135	0	(+27)
144	b 457	50	25032	1	RFBAD
145	456	32000	105040	5	WFATEST
146	520	23376	101037	5	(+1)
147	517	20150	65034	11	(+2)
150	516	22050	125111	0	(+3)
151	444	50	25354	1	(+4)
152	445	22720	101067	5	(+5)
153	533	21401	41065	1	(+6)
154	532	21150	65063	11	(+7)
155	531	23476	101030	0	(+10)
156	414	50	25354	1	(+11)
157	415	22150	65036	6	(+12)
160	617	22050	125034	12	(+13)
161	616	34150	65032	12	(+14)
162	615	22050	125125	4	(+15)
163	452	50	25255	2	(+16)
164	453	22150	65056	11	(+17)
165	527	22250	65055	5	(+20)
166	526	22050	125053	15	(+21)
167	525	36150	11051	15	(+22)
170	524	34151	65047	11	(+23)
171	523	24050	125044	1	(+24)
172	522	35400	5043	1	(+25)
173	521	50	24075	0	(+26)
174	436	50	25134	0	(+27)
175	437	22450	25130	15	(+30)

176	554	50	24121	0	(+31)
177	b 451	50	25134	0	WFABAD
200	450	32000	107074	5	WFBTEST
201	536	23376	101073	5	(+1)
202	535	20150	65070	11	(+2)
203	534	22050	125070	0	(+3)
204	434	50	25354	1	(+4)
205	435	22720	101136	5	(+5)
206	557	21401	41134	1	(+6)
207	556	21150	65132	11	(+7)
210	565	23476	101104	0	(+10)
211	442	50	25354	1	(+11)
212	443	22150	65126	5	(+12)
213	553	22050	125124	11	(+13)
214	552	34150	65122	11	(+14)
215	551	22050	125120	15	(+15)
216	550	22150	65116	11	(+16)
217	547	22550	125115	15	(+17)
220	546	22150	65113	11	(+20)
221	545	36250	65110	11	(+21)
222	544	22350	125107	15	(+22)
223	543	36150	11105	15	(+23)
224	542	34353	65102	11	(+24)
225	541	24050	125101	1	(+25)
226	540	35400	7077	1	(+26)
227	537	50	24010	0	(+27)
230	404	50	25120	0	(+30)
231	405	23450	25106	16	(+31)
232	643	50	24100	0	(+32)
233	b 441	50	25120	0	WFBBAD
234	440	50	25111	2	(+1)
235	626	20000	103053	16	LEFTCYCLE
236	625	22150	65043	2	CYCLELOOP
237	621	21550	25040	16	(+1)
240	620	50	24224	0	(+2)
241	413	22150	65050	6	(+3)
242	624	23150	125046	6	(+4)
243	623	50	24023	0	(+5)
244	411	23050	125020	4	(+6)
245	410	21050	125044	16	(+7)
246	622	50	25052	2	(+10)
247	412	50	25401	0	CYCLEEND
250	566	20000	103152	15	LEFTSHIFT
251	565	22150	65142	1	SHIFTLOOP
252	561	21550	25141	15	(+1)
253	560	50	24265	0	(+2)
254	433	22150	65151	5	(+3)
255	564	23150	125147	5	(+4)
256	563	21050	125145	15	(+5)
257	562	50	25153	1	(+6)
260	432	50	25401	0	SHIFTEND

Page 400: 261 locations used, 117 free

RM:

1	1	REVISION
2	24	RUN-TIME
3		PASSCOUNT
4	2	MAXPASS
5		SUBTEST
6		INNERLOOPCOUNTER
10	0	LOOPWITHIN
11	0	RANDFLAG
12		CURRENTPATTERN
13		CA
14		XA
15		CB
16		XB
17		FDESCR
20		STARTBIT
21		ENDBIT
22		LENGTH
23		SHIFTINDEX
24		SHIFTSIZE
25		SHIFTVALUE

26 MASK  
27 SIMRESULT  
30 RESULT RLC@

Time: 8 seconds: 0 error(s). 0 warning(s). 11927 words free

```
.....  
:EDFieldLog.MIDAS : Logger for EDField program  
: By: J. Keilman  
: Feb. 22, 1980  
:.....
```

```
.start L X AppendOutput EDField.report;  
L X WriteMessage ~***** START EDField Test : ;  
L X WriteDT;  
L X WriteMessage *****~ ;  
L X Skip .continue;
```

```
.breakpoint L X AppendOutput EDField.report;  
  
L A18 SkipNE RFBAD;  
L X Skip .rfbad;  
L A18 SkipNE WFABAD;  
L X Skip .wfabad;  
L A18 SkipNE WFBBAD;  
L X Skip .wfbbad;  
L A18 SkipNE PASSED-EDFIELD-TEST;  
L X Skip .passtest;
```

```
.notmybreak L X AppendOutput EDField.report;  
L X WriteMessage *** FAILED: Not at my breakpoint ~;  
  
L X WriteMessage ' Parity = ;  
R A0 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' CIA = ;  
R A18 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' CTASK = ;  
R A19 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' APCTASK = ;  
R A17 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' APC = ;  
R A16 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' TPC = ;  
R A13 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X CloseOutput;  
L X Exit;
```

```
.rfbad L X WriteMessage *** FAILED: at my Breakpoint RF BAD ~;  
.bad L X WriteMessage ' PASSCOUNT = ;  
R B2 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' RESULT = ;  
R B17 Val;  
L X WriteMessage;  
L X WriteMessage ~;  
  
L X WriteMessage ' SIMRESULT = ;  
R B16 Val;  
L X WriteMessage;  
L X WriteMessage ~;
```

```
L X WriteMessage ' FDESCR =    ;
R B8 Val;
L X WriteMessage;
L X WriteMessage ~;

L X WriteMessage ' STARTBIT =    ;
R B9 Val;
L X WriteMessage;
L X WriteMessage ~;

L X WriteMessage ' ENDBIT =    ;
R B10 Val;
L X WriteMessage;
L X WriteMessage ~;

L X WriteMessage ' CURRENTPATTERN =    ;
R B14 Val;
L X WriteMessage;
L X WriteMessage ~;

L X WriteMessage ' XB =    ;
R B15 Val;
L X WriteMessage;
L X WriteMessage ~;

L X Skip .continue;

.wfabad L X WriteMessage *** FAILED: at my Breakpoint WFA BAD ~;
L X BackSkip .bad;

.wfbbad L X WriteMessage *** FAILED: at my Breakpoint WFB BAD ~;
L X BackSkip .bad;

.passtest L X WriteMessage ~----- PASSEd EDField Test :    ;
L X WriteDT;
L X WriteMessage ----- ;
L X Skip .continue;

.continue L X WriteMessage ~;
L X CloseOutput;
L X DisplayOn;
L X Confirm;
L X TimeOut 10000000;
L X Continue;
L X Skip 2;
L X ShowError Program failed to CONTINUE.;
L X BackSkip .notmybreak;
L X DisplayOff;
L X BackSkip .breakpoint;
```

L A19 Val 0  
L X Confirm  
L X Load EDFIELD:

L B0 Addr REVISION  
L B1 Addr RUN-TIME  
L B2 Addr PASSCOUNT  
L B3 Addr MAXPASS  
L B4 Addr SUBTEST  
L B5 Addr INNERLOOPCOUNTER  
L B8 Addr FDESCR  
L B9 Addr STARTBIT  
L B10 Addr ENDBIT  
L B11 Addr LENGTH  
L B14 Addr CURRENTPATTERN  
L B15 Addr XB  
L B16 Addr SIMRESULT  
L B17 Addr RESULT

L C13 Addr RANDFLAG  
L C15 Addr LOOPWITHIN

L X DisplayOn:  
L X TimeOut 10000  
L X SS GO  
L X Skip 1  
L X ShowError Single-step at GO hung



%

\*\*\* <DDIag>Rev-1>EDSmallMem.mc Revision 1 Nov 15,1979 \*\*\*

```

*****
*** EDSmallMem.mc : Small Memory Exerciser microcode
*** Purpose : This test exhaustively exercises the control store as a 4K x 36 bit memory.
              except for locations occupied by the program or the kernel:
              and the T registers, except for T[15] and T[17].
*** Minimum Hardware : Standard 4 CPU boards.
*** Approximate Run Time : 30 seconds.
*** Written by : Tom Horsley, January 3, 1978
              note: since this test tests all of control store except Page 0, the main program has
              been left on Page 0. March 10, 1978 1:39 PM Bill Kennedy
*** Modified by : Bill Kennedy, April 20, 1978
              to re-initialize Control Store
*** Modified by : Chuck Thacker, December 12, 1978
              to force CS reads and writes in RCSLoop to be on even locations.
*** Modified by : Chuck Thacker, June 14, 1979
              to avoid R0, R11-17, CS pages 16 and 17, tasks 16 and 17.
*** Modified by : T. Henning, November 5, 1979
              to standardize title page and code format, add looping and additional patterns.
*****

```

\*SubTest Description:

- \* SubTest 0: The test has stopped at an unexpected place, something else is interfering with this test.
- \* SubTest 1: Confirm that the value written (Pattern) into Control Store bits 0 to 15 is the one read out (Result).
- \* SubTest 2: Confirm that the value written (Pattern) into Control Store bits 16 to 31 is the one read out (Result).
- \* SubTest 3: Confirm that the value written (Pattern) into Control Store bits 32 to 35 is the one read out (Result).
- \* SubTest 4: Before writing a t-register confirm that it contains its index. If it doesn't then possibly the wrong register has been accessed. Also, confirm that the value written (Pattern) into the T register is the one read out (Result).

\*BreakPoints:

- \* PATTERNERROR: Result read did not match Pattern written.
- \* BADT: Index read from the current T register was not correct in SubTest 4.
- \* BAD-MEMADDR: MemAddr is beyond allowed values, legal ranges are:
  - SubTest 1: 400 to 6777 (StartWord to EndWord)
  - SubTest 1: 400 to 6777 (StartWord to EndWord)
  - SubTest 1: 400 to 6777 (StartWord to EndWord)
  - SubTest 1: 0 to 15 (tasks 16, 17 disallowed for Timer and Kernel)
- \* PASSED-EDSMALLMEM-TEST: Passed all tests, and all passes.

\* ShortLoop Logic Analyzer Sync Points at Control Store address:

- \* PATTERNERROR: Control Store address 145 at MAINLOOP.
- \* BADT: Control Store address 145 at MAINLOOP.

```
*****
*Special Reg. Definition:
* ShortLoop: At any breakpoint, the user has the choice of setting ShortLoop to a 1 to
  loop on the current test. During the short loop, the user can modify the address
  and data to the Control Store at will by changing MemAddr and Pattern. For the
  T register test in subtest 4, the T register and the test pattern can also be
  changed at will.
      1. the current test will loop repeatedly for trouble shooting
      0. no looping in current test
* PatternChoice:
  Bit 15 - all zeros pattern, enable by 1, disable by 0
  Bit 14 - all ones pattern, enable by 1, disable by 0
  Bit 13 - checker pattern, enable by 1, disable by 0
  Bit 12 - random pattern, enable by 1, disable by 0
  Example: PatternChoice=1 enables the all zeros pattern only
           PatternChoice=2 enables the all ones pattern only
           PatternChoice=4 enables the checker pattern only
           PatternChoice=10 enables the random pattern only
           PatternChoice=17 enables all four of the patterns
           PatternChoice=11 enables the random and all zeros patterns
*****
*Subroutine Description:
* ReInitCS: zeros out Control Store and puts in correct parity.
*****
```

\*\*\*\*\*

\*INITIALIZATION:

```
BUILTIN[INSERT.24]:
INSERT[DOLANG]:
TITLE[SmallMemoriesTester]: * Exhaustively exercises various small memories
MIDASINIT:
SET[MainPage.0]: * set tag for Main Program page
ONPAGE[MainPage]:
```

\*\*\*\*\* R-Registers: \*\*\*\*\*

```
RV[PassCount.20]: *outer loop counter
RV[MaxPass.21.10]: *number of times big loop is to repeat before breakpointing
RV[SubTest.22]: * current location of test
RV[TestCounter.23]: * inner loop counter

RV[CA.24]: *used in random number generation. A*XA - CA
RV[XA.25]: *random number generated via A*XA + CA
RV[CurrentXA.26]: *value of XA to be used (usually XA. sometimes OldXA)
RV[OldXA.27]: *last value of XA

RV[CS0Test.30]: *number of test iterations
RV[CS1Test.31]: *number of test iterations
RV[CS2Test.32]: *number of test iterations

RV[StartWord.33.400]: * beginning of control store to be tested
RV[Endword.34.6777]: *end of control store to be tested

RV[RepeatCounter.35]: *number of test repeats
RV[TmemTest.36]: *number of test iterations
RV[CS0.37]: * temporary register used in re-initializing Control Store
RV[CS1.40]: * temporary register used in re-initializing Control Store
RV[CS2.41]: * temporary register used in re-initializing Control Store

RV[NewTask.42]: *used in task switching
RV[Tmp.43]: * temporary register

SET[wordLoc.44]: *address of register 'MemAddr'
RV[MemAddr.wordLoc]: *address of memory cell to be tested
MC[wordAddress.wordLoc]: *address of register 'MemAddr'
RV[Pattern.ADD[wordLoc.1]]: *pattern to be stuffed into word
RV[Result.ADD[wordLoc.2]]: *result of memory read

RV[PatternChoice.47.17]: *enable all patterns at program start
RV[CurrentPattern.50.1]: *initialize to all zeros pattern
RV[PatternTry.51.1]: *initialize to all zeros pattern
RV[Ones.52.177777]: *define ones to be 177777
RV[Checker1.53.125252]: *checker pattern register
RV[Checker0.54.052525]: *checker pattern register
RV[Toggle.55.0]: *checker toggle register

RV[ShortLoop.56.0]: * 1 => loop on current test. 0 => continue on next test

RV[Revision.57.1]: *REVISION 1
RV[Run-Time.60.35]: *Run-Time is 36b or 30D seconds
```

\*\*\*\*\* Task Entry Points: \*\*\*\*\*

```
SET[higherTaskLoc.40]: *entry point to higher task
MC[higherTaskEntry.higherTaskLoc]: *entry point to higher task

SET[lowerTaskLoc.50]: *entry point to task 0
MC[lowerTaskEntry.lowerTaskLoc]: *entry point to task 0

SET[higherTaskLoc1.50]: *entry point to higher task
MC[higherTaskEntry1.higherTaskLoc1]: *entry point to higher task

SET[lowerTaskLoc1.70]: *entry point to task 0
MC[lowerTaskEntry1.lowerTaskLoc1]: *entry point to task 0

SET[testSwitch.20]: *location of main switch
SET[MainPageBase.LSHIFT[MainPage.10]]:
```

```

*****
*** MAIN routine:

go:
start:
  XA ← AND@[0377, 123]C;          *Load 16 Bits (XA ← 123)
  XA ← (XA) OR (AND@[177400, 123]C);

  CA ← AND@[0377, 33031]C;       *Load 16 Bits (CA ← 33031)
  CA ← (CA) OR (AND@[177400, 33031]C);

  CLEARMPANEL;
  TestCounter ← 0C;
  PassCount ← 0C;
  CS0Test ← 0C;
  CS1Test ← 0C;
  CS2Test ← 0C;
  TmemTest ← 0C;
  RepeatCounter ← 0C;
  t ← 20000C;                    *set up CurrentXA so that it contains valid address
  t ← (LSH[StartWord, 1]) OR (t);
  CurrentXA ← t;

  t ← (1C);                      *Initialize task registers to their task index
  MemAddr ← t;
IndexT: t ← (17C);
  LU ← (MemAddr) - (t) - 1;
  GOTO[IndexTDone, ALU >= 0];

  t ← LSH[MemAddr, 14];
  NewTask ← t;

  NewTask ← (NewTask) OR (higherTaskEntry1);
  APCTASK&APC ← (NewTask);
  RETURN;

  Tmp ← wordAddress, AT[higherTaskLoc1]; *write value into t register
  STKP ← Tmp;
  t ← STACK;

  Tmp ← lowerTaskEntry1;         *return to task 0
  APCTASK&APC ← (Tmp);
  RETURN;

  NOP, AT[lowerTaskLoc1];

  MemAddr ← (MemAddr) + 1;      *Increment FOR loop counter
  GOTO[IndexT];

IndexTDone:
  nop;

bigLoop: t ← (PatternTry) AND (177760C); *what pattern to use?
  goto[WhatPattern, alu=0];      *exhausted all four pattern types?
  PatternTry ← 1C;              *yes, select the zero pattern again
  Toggle ← 0C;                  *reset checker pattern toggle
  INCM PANEL;
  PassCount ← t - (PassCount) - 1; *increment pass count
  lu ← (MaxPass) - (t);
  goto[EndTest, alu<0];        *finished all passes?
  nop;

WhatPattern: t ← PatternChoice; *determine what pattern to use
  t ← (PatternTry) AND (t);
  goto[NextPattern, alu=0];     *do we want to use this pattern?
ThisPattern: CurrentPattern ← t, goto[mainLoop]; *yes, use this pattern
NextPattern: PatternTry ← LSH[PatternTry, 1], goto[bigLoop]; *no, try the next pattern

EndTest:
  CALL[ReInitCS];              * go re-initialize control store
Passed-EDSmallMem-Test: BREAKPOINT, goto[go];

```

\* SUBTEST 0

```

mainLoop:
  SubTest = 0C, AT[145]:          *nail down scope trigger point
  ShortLoop = ShortLoop, GOTO[decipherXA, R ODD]:      *ShortLoop selected?

  TestCounter = (TestCounter) + 1:
  GOTO[.+2, NOCARRY]:
  PatternTry = LSH[PatternTry,1], goto[bigLoop]:      *use next pattern

  t = (CurrentXA):
  OldXA = t:

  t = XA, TASK:                  *task so that Midas can mouse halt
  t = (LSH[XA, 2]) + t:          *Random (4005*XA - CA mod 2**16)
  t = (LSH[XA, 13]) + t:
  t = (CA) - t:
  XA = t:
  t = (XA):
  CurrentXA = t:

  t = (CurrentPattern) AND (1C):
  goto[Try1,alu=0]:             *want the zeros pattern?
  Pattern = 0C, goto[decipherXA]: *yes
Try1:  t = (CurrentPattern) AND (2C): *no, try the ones pattern
  goto[Try2,alu=0]:             *want the ones pattern?
  t = Ones: *yes
  Pattern = t, goto[decipherXA]:
Try2:  t = (CurrentPattern) AND (4C): *no, try the checker pattern
  goto[Try3,alu=0]:             *want the checker pattern?
  Toggle = Toggle, goto[Checker01,R ODD]: *yes
  t = Checker1: *1010101010101010 pattern
  Pattern = t:
  Toggle = (Toggle) - 1, goto[decipherXA]: *toggle checker pattern
Checker01: t = Checker0: *0101010101010101 pattern
  Pattern = t:
  Toggle = (Toggle) + 1, goto[decipherXA]: *toggle checker pattern
Try3:  t = (CurrentPattern) AND (10C): *no, try the random pattern
  goto[bigLoop,alu=0]:          *want the random pattern?
  t = (CurrentXA): *yes
  Pattern = t:
  t = PassCount:
  Pattern = (Pattern) + (t):

decipherXA: SET[Switch0, TestSwitch]: *pick memory to be tested
  DISPATCH[CurrentXA, 0, 3]:
  DISP[SwitchTab0]:
SwitchTab0:
  GOTO[Case0], AT[Switch0, 0]:
  GOTO[Case1], AT[Switch0, 1]:
  GOTO[Case2], AT[Switch0, 2]:
  GOTO[Case3], AT[Switch0, 3]:
  GOTO[Case4], AT[Switch0, 4]:
  GOTO[Case5], AT[Switch0, 5]:
  GOTO[Case6], AT[Switch0, 6]:
  GOTO[Case7], AT[Switch0, 7]:

```

## \* SUBTEST 1

```

Case0:      SubTest - 1C:                                *CS0 Memory

            t = LDF[CurrentXA, 3, 14];
            ShortLoop = ShortLoop, GOTO[.-2, R ODD];    *ShortLoop selected?
            MemAddr = t;

            t = MemAddr;
            LU = (StartWord) - (t) - 1;
            GOTO[Range1, ALU < 0];                      *Check Range
            GOTO[OutOfRange];

Range1:     LU = (EndWord) - (t);
            GOTO[Range2, ALU >= 0];
            GOTO[OutOfRange];

Range2:     LU = (Pattern);                              *write the pattern
            APCTASK&APC = (MemAddr);
            WriteCS0&2;

            t = 0C; *read the word
            APCTASK&APC = (MemAddr);
            READCS;
            t = CSData;
            Result = t;

            CS0Test = (CS0Test) - 1;
            GOTO[Endswitch0];

```

## \* SUBTEST 2

```

Case1:      SubTest - 2C:                                *CS1 Memory

            t = LDF[CurrentXA, 3, 14];
            ShortLoop = ShortLoop, GOTO[.-2, R ODD];    *ShortLoop selected?
            MemAddr = t;

            t = MemAddr;
            LU = (StartWord) - (t) - 1;
            GOTO[Range3, ALU < 0];                      *Check Range
            GOTO[OutOfRange];

Range3:     LU = (EndWord) - (t);
            GOTO[Range4, ALU >= 0];
            GOTO[OutOfRange];

Range4:     LU = (Pattern);                              *write the pattern
            APCTASK&APC = (MemAddr);
            WriteCS1;

            t = 1C; *read the word
            APCTASK&APC = (MemAddr);
            READCS;
            t = CSData;
            Result = t;

            CS1Test = (CS1Test) - 1;
            GOTO[Endswitch0];

```

## \* SUBTEST 3

```

Case2:      SubTest - 3C:                                *CS2 Memory

            t = LDF[CurrentXA, 3, 14];
            ShortLoop = ShortLoop, GOTO[.-2, R ODD];    *ShortLoop selected?
            MemAddr = t;

            t = MemAddr;
            LU = (StartWord) - (t) - 1;
            GOTO[Range5, ALU < 0];                      *Check Range

```

```

        GOTO[OutRange]:
Range5:
    LU ← (EndWord) - (t);
    GOTO[Range6, ALU >= 0];
    GOTO[OutRange]:
Range6:
    t ← (Pattern);
    LU ← 0C;
    APCTASK&APC ← (MemAddr);
    WriteCS0&2:
    t ← 3C;
    APCTASK&APC ← (MemAddr);
    READCS;
    t ← CSDData;
    Result ← t;
    Result ← LDF[Result, 0, 4];
    Pattern ← LDF[Pattern, 14, 4];
    CS2Test ← (CS2Test) + 1;
    GOTO[Endswitch0]:
    *write the pattern
    *This shouldn't be necessary?!!!!!
    *read the word
    *abbreviate expected result
* SUBTEST 4
Case3:
    Subtest ← 4C;
    t ← LDF[CurrentXA, 3, 4];
    ShortLoop ← ShortLoop, GOTO[.+2, R ODD];
    MemAddr ← t;
    MemAddr ← (MemAddr) AND (17C);
    T ← (MemAddr) and (16C);
    Tmp ← T;
    lu ← (Tmp) xor (16C);
    goto[.+2, alu#0];
    goto[OutRange];
    t ← LSH[MemAddr, 14];
    NewTask ← t;
    NewTask ← (NewTask) OR (higherTaskEntry);
    t ← 0C;
    APCTASK&APC ← (NewTask);
    RETURN;
    Tmp ← wordAddress, AT[higherTaskLoc];
    STKP ← Tmp;
    Tmp ← t;
    LU ← (STACK&+1) - (t);
    GOTO[EndT, ALU = 0];
    ShortLoop ← ShortLoop, GOTO[BADT, R EVEN];
    goto[EndT];
BADT: BREAKPOINT;
EndT:
    t ← STACK&-1;
    STACK ← t;
    t ← Tmp;
    Tmp ← lowerTaskEntry;
    APCTASK&APC ← (Tmp);
    RETURN;
    TmemTest ← (TmemTest) + 1, AT[lowerTaskLoc];
    GOTO[Endswitch0];
    *t Memory
    *ShortLoop selected?
    *use the LSB 4 bits for T(task) to be tested
    *don't do tasks 16 or 17
    *enter higher task
    *check to ascertain correct t-register
    *ShortLoop for troubleshooting?
    *t ← Pattern
    *Result ← t
    *restore task number in t
    *return to task 0
Case4:
    GOTO[OutRange]: *Repeat the last test, i.e., hit the last memory location again
Case5:
    GOTO[OutRange]: *Repeat the last test, i.e., hit the last memory location again
Case6:

```

```
GOTO[OutOfRange]: *Repeat the last test. i.e., hit the last memory location again
Case7: GOTO[OutOfRange]: *Repeat the last test. i.e., hit the last memory location again
Endswitch0:
tests: TASK: *enable mouse halt
      t = Result;
      LU = (Pattern) - (t);
      GOTO[Endif0, ALU = 0];

      ShortLoop = ShortLoop, GOTO[PATTERNERROR, R EVEN]; *ShortLoop for troubleshooting?
      goto[Endif0];
PATTERNERROR:
BREAKPOINT:

Endif0: GOTO[mainLoop];

OutOfRange: ShortLoop = ShortLoop, GOTO[.+2, R EVEN]; *ShortLoop for troubleshooting?
Bad-MemAddr: breakpoint;
             goto[mainloop];
*           t = (OldXA);
*           CurrentXA = t;

*           RepeatCounter = (RepeatCounter) + 1;
*           GOTO[decipherXA];
```



\*\*\*\*\* SUBROUTINE: ReInitCS \*\*\*\*\*

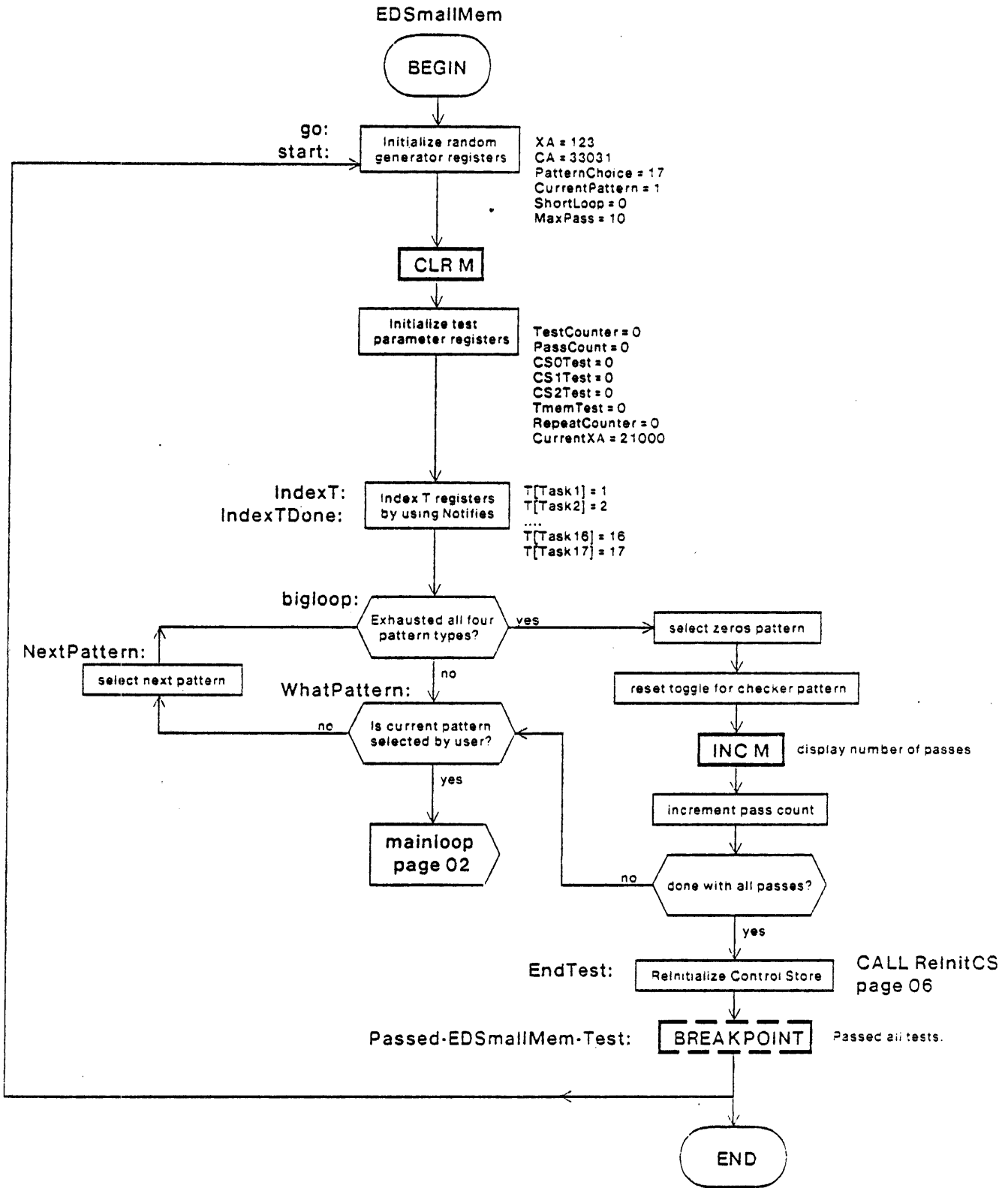
```

>
*      Puts zeros into the Control Store from StartWord
*      to EndWord and also puts in the correct parity.

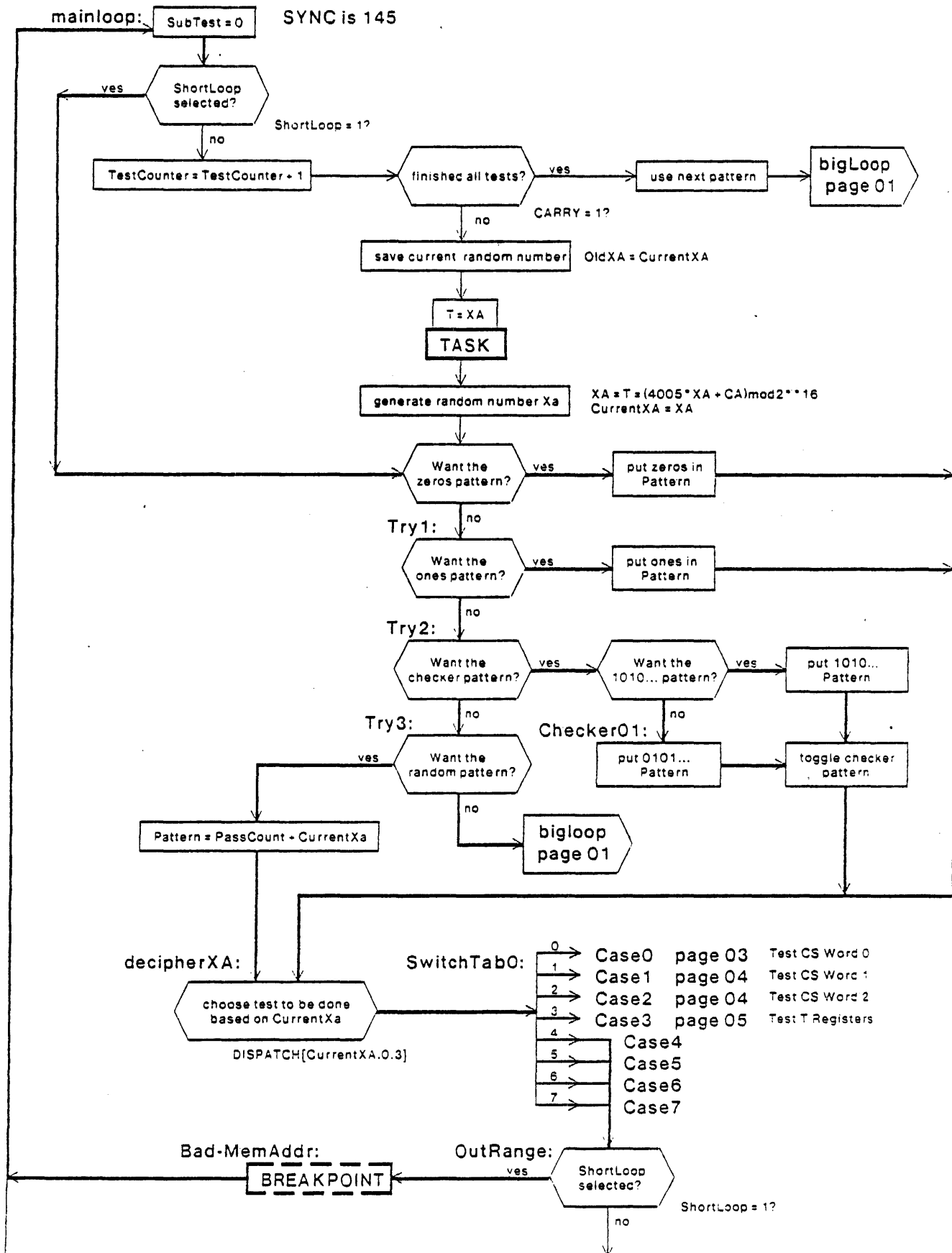
      ONPAGE[MainPage]:
ReInitCS:
  CSC = ZERO:          * zero what's to be written into CS
  CS1 = ZERO:          * zero what's to be written into CS
  CS2 = ZERO:          * zero what's to be written into CS
  t = StartWord:      * Write control store from 'StartWord' to 'EndWord'
  NewTask = t:
  t = CS0:             *WriteCS (write control store location 'NewTask')
  Tmp = t:             *put CS0 in the temp. reg.
  t = CS1:             *get CS1
  Tmp = t = (Tmp) XOR (t): *xor first two CS words
  t = (LDF[CS2.14.4]) XOR (t): *xor third CS word with the result
  Tmp = t = (LDF[Tmp.0.10]) XOR (t): *start halving process to get parity
  Tmp = t = (LDF[Tmp.10.4]) XOR (t):
  Tmp = t = (LDF[Tmp.14.2]) XOR (t):
  Tmp = t = (LDF[Tmp.16.1]) XNOR (t): *Do last part and complement it
  t = (LDF[Tmp.17.1]): *put parity bit in the t-register
  CS1 = (CS1) XOR (t): *exclusive or parity bit into bit 31 of CS (15 of CS1)
RCSLoop:
  t = (CS2):
  LU = (CS0):
  APCTASK&APC = (NewTask):
  WriteCS0&2:
  LU = (CS1), at[MainPageBase,340]: *force WriteCS to have JA.7=1
  APCTASK&APC = (NewTask):
  WriteCS1:
  t = NewTask = (NewTask) + 1, at[MainPageBase,350]:
  LU = (EndWord) - (t) - 1: *increment address - Force WriteCS1's JA.7=0
  GOTO[RCSLoop, CARRY]: * see if done yet
  RETURN:

END:

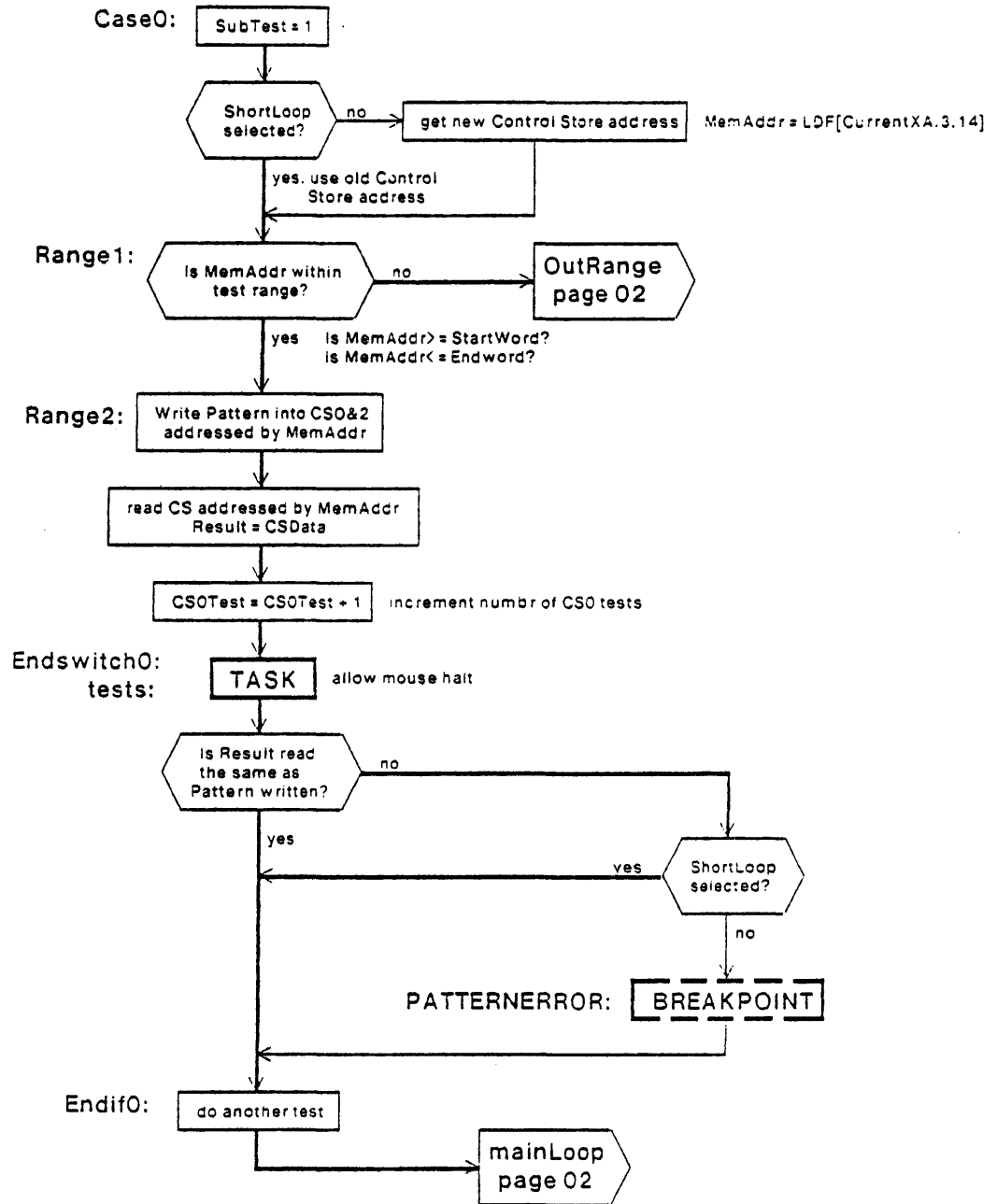
```



XEROX	D(O)	MICROCODE SOURCE	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDSmallMem.mc	EDSmallMem01.sil	Tom Henning	1	11/02/79	01

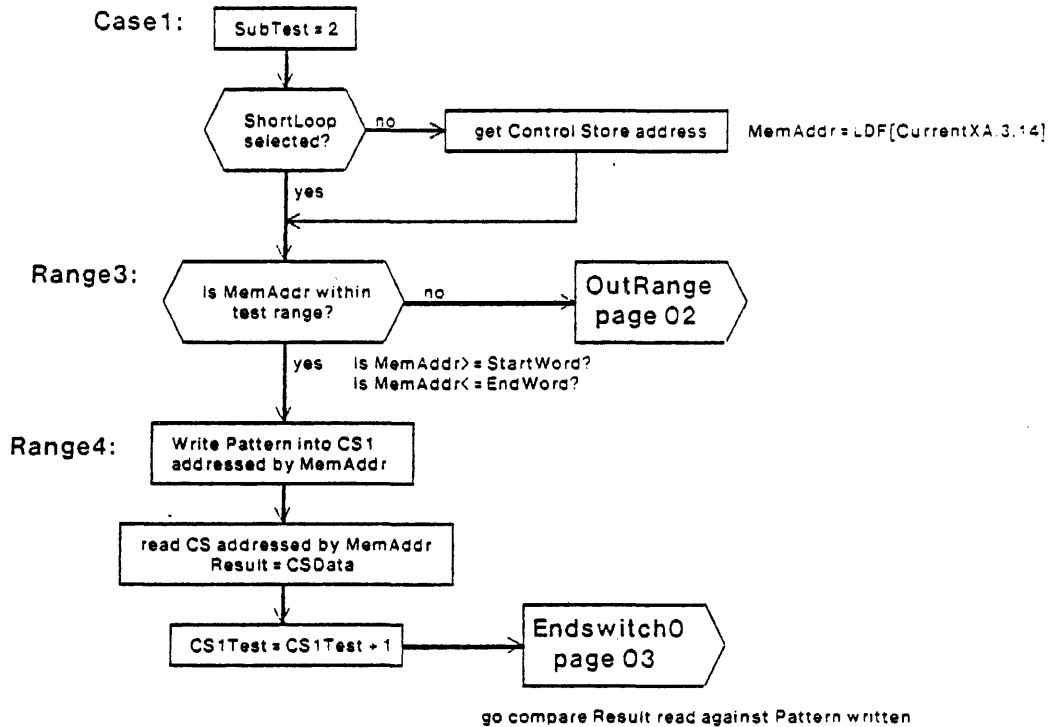


**SubTest 1**      Test for CS0 (Control Store word 0)

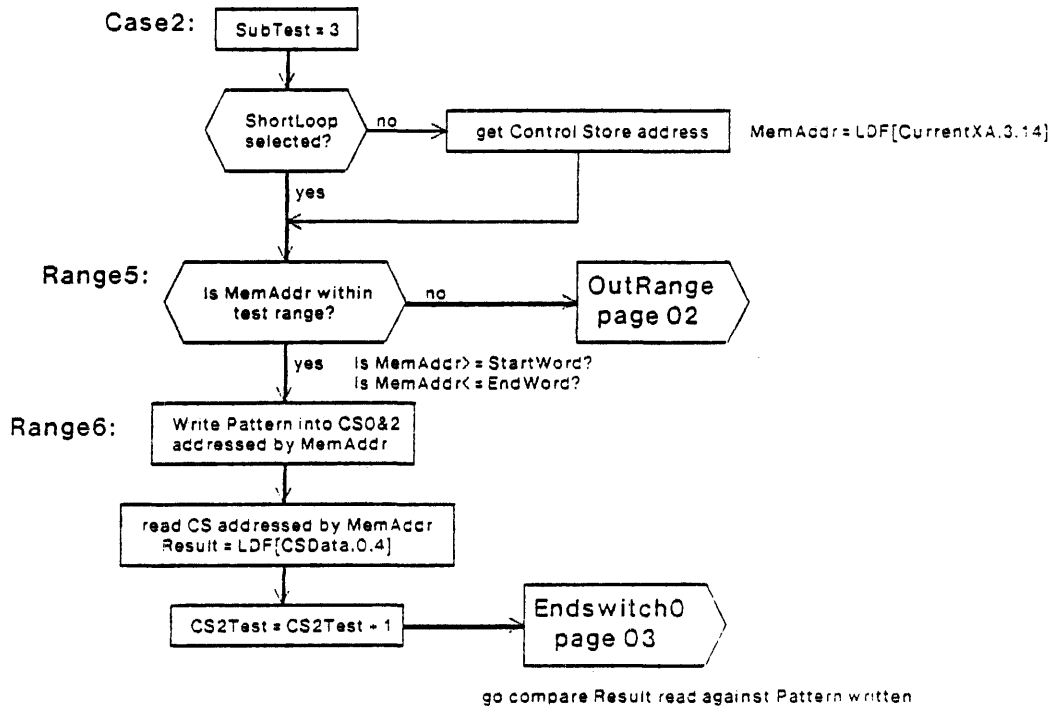


XEROX	D(0)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDSmallMem.mc	EDSmallMem03.sil	Tom Henning	1	11/02/79	03

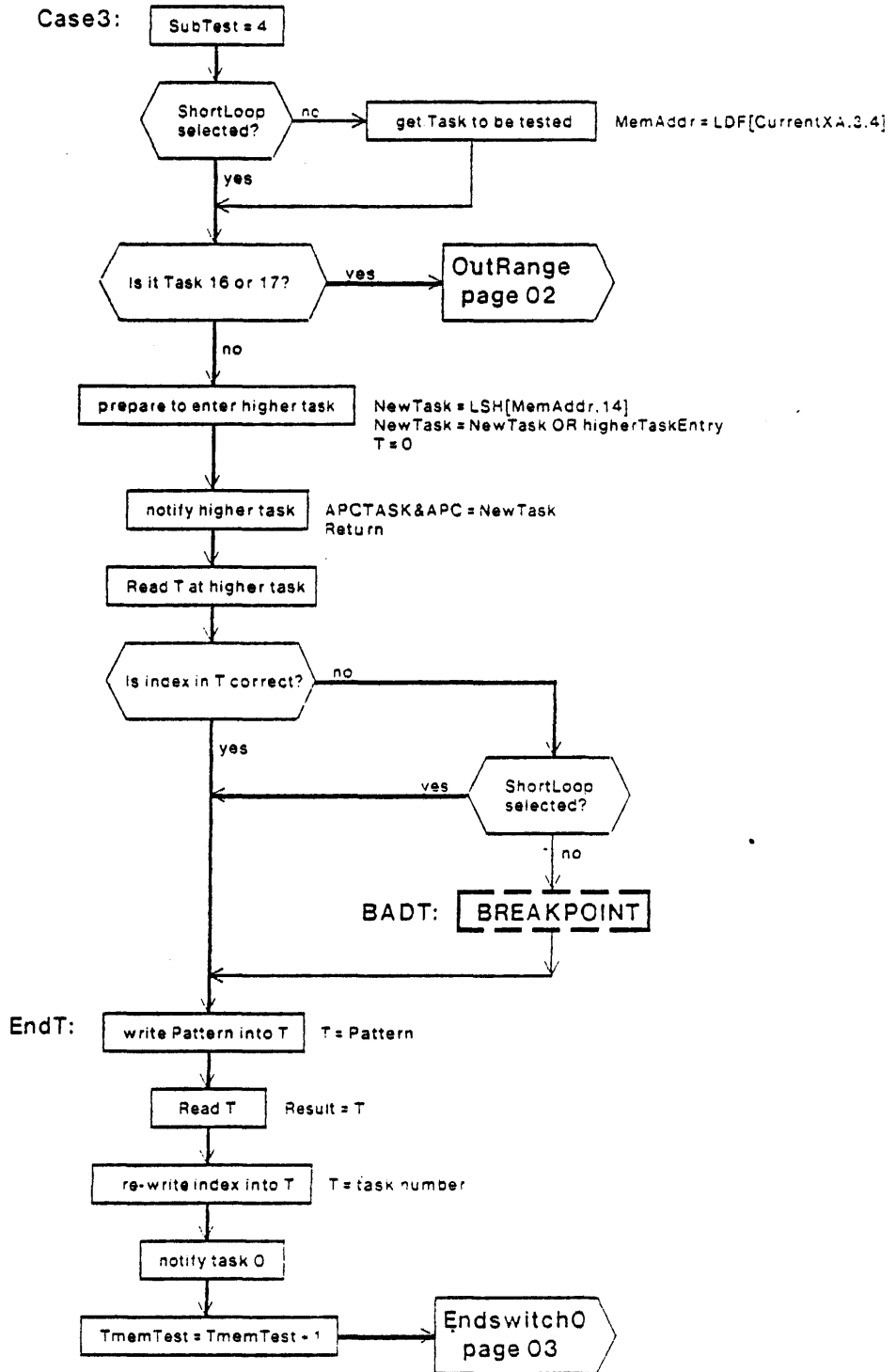
**SubTest 2** Test for CS1 (Control Store word 1)



**SubTest 3** Test for CS2 (Control Store word 2)



**SubTest 4** Test for T Memory



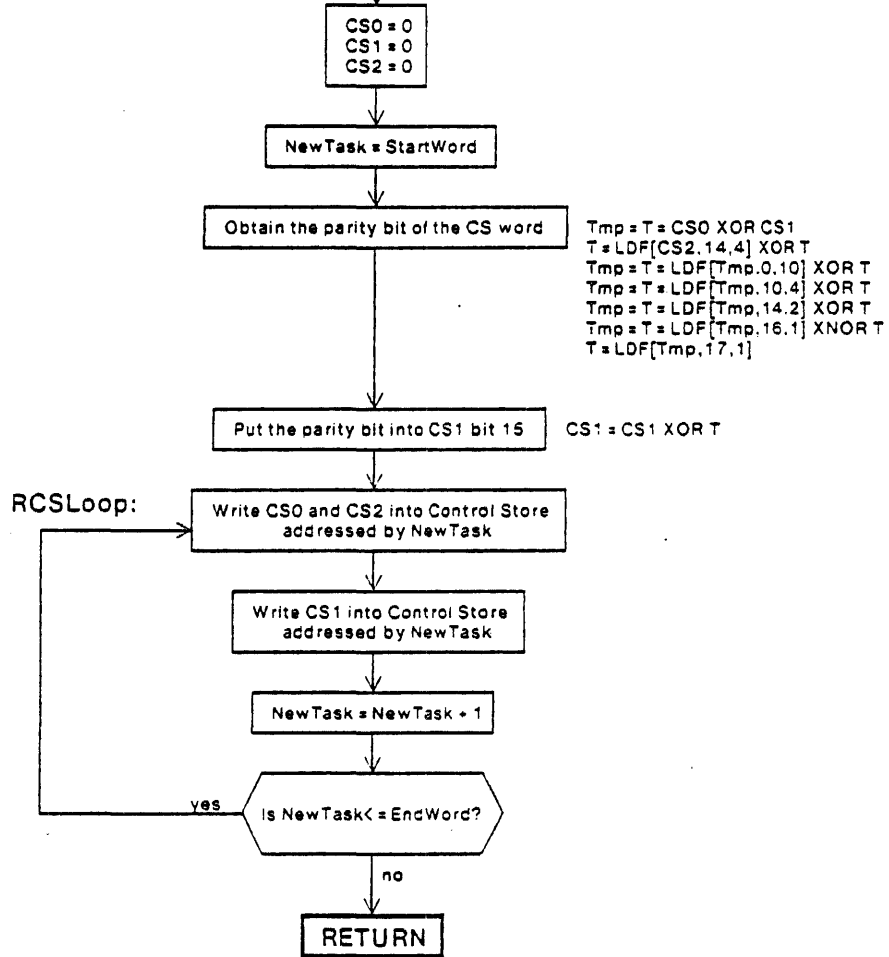
go compare Result read against Pattern written

XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDSmallMem.mc	EDSmallMem05.sil	Tom Henning	1	11/02/79	05

Subroutine

ReinitCS

Subroutine to zero out Control Store and to remove any parity errors.



XEROX ED	D(0) Diagnostic	PROGRAM NAME EDSmallMem.mc	DOCUMENTATION FILE EDSmallMem06.sil	DESIGNER Tom Henning	REV 1	DATE 11/02/79	PAGE 06
-------------	--------------------	-------------------------------	--	-------------------------	----------	------------------	------------

PARITY	0	REVISION	1	COMM-ERR0	0
CYCLECONTROL	377	RUN-TIME	36	COMM-ERR1	0
POXREG	4	PASSCOUNT	0	COMM-ERR2	0
POFREG	4	MAXPASS	10	BOOT-ERR	0
DBREG	17	SUBTEST	0	*BOOTREASON	40
SBREG	77			MEMSYNDROME	0
MNBR	3423				
*SSTKP	377				
STKP	0				
*ALURESULT	3	*XA	123	PATTERN	0
*SALUF	377	TESTCOUNTER	0	RESULT	0
T 20	7000	CURRENTPATTER	1	MEMADDR	0
AATOVA	0				
TPC 20	7777			STARTWORD	400
CALLER	ILC@+7422			ENDWORD	6777
PAGE	0				
*APC	7011	CS0TEST	0	PATTERNCHOICE	17
*APCTASK	16	CS1TEST	0	SHORTLOOP	0
*DIA	00+1	CS2TEST	0		
CTASK	0	TMENTEST	0		

Loaded: EDSmallMem

Time: 08.79

Step at 0:00, BP at 0:00+1

Exit Boot Run-Prog Read-Cmds Break UnBreak CtrAddedBPs CtrAllBPs ShowBPs Go  
 SS Continue Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual



MicroD 3.6 (OS 16) of April 27, 1979  
at 5-Dec-79 10:50:58

microd.run EDSmallMem

EDSmallMem.DIB 355b instructions written 5-Dec-79 10:50:05

Total of 355b instructions

Checking for errors...  
Linking...  
Building allocation lists...  
Assigning locations...  
355b instructions in rings involving ONPAGE or AT  
Reloading binaries...  
Checking assignment...  
Writing .MB file...  
Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
EDSmallMem.DIB:					
0	11	22005	107050	7	GO START
1	330	22320	101057	7	(+1)
2	327	22001	123055	3	(+2)
3	326	22323	115053	3	(+3)
4	325	47	7050	3	(+4)
5	324	20020	101046	17	(+5)
6	323	20020	101045	3	(+6)
7	322	24020	101042	3	(+7)
10	321	24020	101040	7	(+10)
11	320	24020	101037	13	(+11)
12	317	26020	101035	13	(+12)
13	316	26020	101033	7	(+13)
14	315	22	41031	3	(+14)
15	314	24374	43027	17	(+15)
16	313	22050	125024	13	(+16)
17	312	0	43022	3	(+17)
20	311	12050	125020	3	(+20)
21	310	0	77013	3	INDEXT
22	305	13550	25010	3	(+1)
23	304	50	24230	0	(+2)
24	15	12174	71005	3	(+3)
25	303	10050	125004	13	(+4)
26	302	10303	101002	13	(+5)
27	301	10147	21000	13	(+6)
30	300	50	25401	0	(+7)
31	@ 60	10002	111154	17	(+10)
32	366	10150	3153	17	(+11)
33	365	40150	65150	17	(+12)
34	364	10003	121147	17	(+13)
35	363	10147	21144	17	(+14)
36	362	50	25401	0	(+15)
37	@ 70	50	25017	3	(+16)
40	307	13050	125014	3	(+17)
41	306	50	25021	3	(+20)
42	14	50	25064	1	INDEXTDONE
43	132	14217	41110	5	_BIGLOOP
44	144	50	24101	1	(-1)
45	141	14000	103042	5	(-2)
46	121	15020	101162	4	(-3)
47	71	47	5147	0	(-4)
50	63	21050	165123	0	(-5)
51	51	21450	25102	-	(-6)
52	41	50	24214	0	(-7)
53	5	50	25100	1	(-10)
54	140	12150	65106	15	WHATPATTERN
55	143	14250	65104	5	(+1)
56	142	50	24150	0	(+2)
57	55	14050	125113	1	THISPATTERN
60	54	14174	103064	5	NEXTPATTERN
61	7	50	25342	3	ENDTEST

52	b	10	50	25022	0	PASSED-EDSMALLMEM-TEST
53	@	145	20020	101031	12	MAINLOOP
64		214	16150	124477	10	(+1)
65		35	21050	125027	16	(-2)
66		213	50	24077	1	(+3)
67		137	14174	103054	5	(-4)
70		136	22150	65024	12	(-5)
71		212	22050	125071	15	(+6)
72		134	22150	65223	6	(-7)
73		211	23174	45400	4	(-10)
74		135	23174	67134	5	(-11)
75		155	23150	65132	1	(+12)
76		155	22050	125130	5	(+13)
77		154	22150	65126	5	(+14)
100		153	22050	125124	11	(+15)
101		152	14200	43123	1	(-16)
102		151	50	24070	0	(-17)
103		35	12020	101076	4	(+20)
104		34	14200	45120	1	TRY1
105		150	50	24064	0	(+1)
106		33	14150	65161	12	(+2)
107		270	12050	125077	4	(+3)
110		32	14200	51116	1	TRY2
111		147	50	24061	0	(+1)
112		31	16150	124436	4	(+2)
113		16	14150	65165	16	(+3)
114		272	12050	125163	6	(+4)
115		271	17050	125077	4	(+5)
116		17	16150	65170	2	CHECKER01
117		274	12050	125166	6	(+1)
120		273	17050	125077	4	(+2)
121		30	14200	61115	1	TRY3
122		146	50	24065	1	(+1)
123		133	22150	65176	12	(+2)
124		277	12050	125174	6	(+3)
125		276	20150	65173	2	(+4)
126		275	13150	125077	4	(+5)
127		37	22172	15156	12	DECIPHERXA
130		267	50	25641	0	(-1)
131	@	20	50	25155	2	SWITCHTABO
132	@	21	50	25147	1	(+1)
133	@	22	50	25160	1	(+2)
134	@	23	50	25010	2	(+3)
135	@	24	50	25013	2	(+4)
136	@	25	50	25015	2	(+5)
137	@	26	50	25016	2	(+6)
140	@	27	50	25020	2	(+7)
141		256	20000	103153	12	CASE0
142		255	22167	71150	12	(+1)
143		254	16150	124507	10	(+2)
144		42	12050	125106	0	(+3)
145		43	12150	65147	2	(+4)
146		253	25550	25145	16	(+5)
147		252	50	24325	0	(+6)
150		52	50	25034	2	(+7)
151		53	27450	25143	2	RANGE1
152		261	50	24251	1	(-1)
153		131	50	25034	2	(+2)
154		130	12150	25141	6	RANGE2
155		260	12147	21136	2	(+1)
156		257	47	31534	2	(+2)
157		256	20	41133	2	(+3)
158		255	12147	21130	2	(+4)
159		254	47	35527	2	(+5)
162		253	54150	65126	18	(+6)
163		252	12050	125122	12	(+7)
164		251	25050	125121	2	(+10)
165		250	50	25143	0	(+11)
166		163	20000	103144	11	CASE1
167		162	22167	71142	11	(+1)
170		161	16150	124457	11	(+2)
171		125	12050	125056	1	(+3)
172		127	12150	65141	1	(+4)
173		160	25550	25137	15	(+5)
174		157	50	24315	0	(+6)
175		156	50	25034	2	(+7)

176	47	27450	25050	2	RANGE3
177	230	50	24251	1	(+1)
200	125	50	25034	2	(-2)
201	124	12150	25057	6	RANGE4
202	227	12147	21054	2	(+1)
203	226	47	33452	2	(+2)
204	225	0	43051	2	(+3)
205	224	12147	21046	2	(+4)
206	223	47	35445	2	(-5)
207	222	54150	65042	16	(+6)
210	221	12050	125040	12	(+7)
211	220	25050	125036	6	(+10)
212	217	50	25143	0	(+11)
213	170	20000	107157	11	CASE2
214	167	22167	71156	11	(-1)
215	166	15150	124446	11	(+2)
216	122	12050	125047	1	(+3)
217	123	12150	65153	1	(-4)
220	165	25550	25151	15	(+5)
221	164	50	24310	0	(+6)
222	44	50	25034	2	(+7)
223	45	27450	25113	2	RANGE5
224	245	50	24375	0	(+1)
225	77	50	25034	2	(-2)
226	76	12150	65110	6	RANGE6
227	244	20	1107	2	(+1)
230	243	12147	21105	2	(-2)
231	242	47	31502	2	(+3)
232	241	0	47101	2	(+4)
233	240	12147	21076	2	(+5)
234	237	47	35475	2	(+6)
235	236	54150	65072	16	(+7)
236	235	12050	125070	12	(+10)
237	234	12162	133066	12	(+11)
240	233	12163	123065	6	(-12)
241	232	25050	125063	12	(+13)
242	231	50	25143	0	(+14)
243	204	20000	111007	12	CASE3
244	203	22163	41005	12	(+1)
245	202	15150	124557	10	(+2)
246	72	12050	125166	0	(-3)
247	73	12200	137002	2	(+4)
250	201	12200	75000	2	(+5)
251	200	10050	125177	15	(-6)
252	177	10400	35175	15	(-7)
253	176	50	24171	0	(+10)
254	74	50	25034	2	(+11)
255	75	12174	71172	1	(-12)
256	175	10050	125170	11	(-13)
257	174	10302	101166	11	(+14)
260	173	20	41165	1	(+15)
261	172	10147	21163	11	(+16)
262	171	50	25401	0	(+17)
263	40	10002	111176	17	(+20)
264	377	10150	3174	17	(+21)
265	376	10050	125173	17	(+22)
266	375	43450	25170	17	(-23)
267	374	50	24004	0	(+24)
270	3	16150	124412	10	(-25)
271	5	50	25005	0	(-25)
272	4	50	25005	0	BADT
273	2	42150	65157	17	ENDT
274	373	40050	125154	17	(+1)
275	372	10150	65163	17	(+2)
276	371	10002	121161	17	(-3)
277	370	10147	21155	17	(-4)
300	367	50	25401	0	(-5)
301	50	27050	125114	12	(+6)
302	246	50	25143	0	(+7)
303	205	50	25034	2	CASE4
304	206	50	25034	2	CASE5
305	207	50	25034	2	CASE6
306	210	50	25034	2	CASE7
307	61	50	25316	2	ENDSWITCHO TESTS
310	247	12150	65400	10	(-1)
311	62	13450	25033	6	(-2)

312	215	50	24130	0	(+3)
313	55	15150	124537	10	(+4)
314	57	50	25131	0	(+5)
315	b 56	50	25131	0	PATTERNERROR
316	54	50	25112	1	ENDIFO
317	216	15150	124557	10	OUTRANGE
320	b 67	50	25154	0	BAD-MEMADDR
321	56	50	25112	1	(+1)
322	361	26176	101140	17	REINITCS
323	360	10176	101137	3	(+1)
324	357	10176	101135	7	(+2)
325	356	24150	65132	17	(+3)
326	355	10050	125130	13	(+4)
327	354	26150	65127	17	(+5)
330	353	10050	125125	17	(+6)
331	352	10150	65122	3	(+7)
332	351	10450	165116	17	(+10)
333	347	10463	63115	7	(+11)
334	346	10465	167112	17	(+12)
335	345	10463	153110	17	(+13)
336	344	10461	171106	17	(+14)
337	343	10760	175105	17	(+15)
340	342	10160	77103	17	(+16)
341	341	10450	125026	0	(+17)
342	13	10150	65077	7	RCSLOOP
343	337	26150	25074	17	(+1)
344	336	10147	21072	13	(+2)
345	335	47	31500	3	(+3)
346	@ 340	10150	25071	3	(+4)
347	334	10147	21066	13	(+5)
350	333	47	33520	3	(+6)
351	@ 350	11050	165064	13	(+7)
352	332	27550	25063	3	(+10)
353	331	50	24026	0	(+11)
354	12	50	25401	0	(+12)

Page 0: 355 locations used. 23 free

RM:

20		PASSCOUNT
21	10	MAXPASS
22		SUBTEST
23		TESTCOUNTER
24		CA
25		XA
26		CURRENTXA
27		OLDXA
30		CSOTEST
31		CS1TEST
32		CS2TEST
33	400	STARTWORD
34	6777	ENDWORD
35		REPEATCOUNTER
36		TMEMTEST
37		CS0
40		CS1
41		CS2
42		NEWTASK
43		TMP
44		MEMADDR
45		PATTERN
46		RESULT
47	17	PATTERNCHOICE
50	1	CURRENTPATTERN
51	1	PATTERNTRY
52	177777	ONES
53	125252	CHECKER1
54	52525	CHECKER0
55	0	TOGGLE
56	0	SHORTLOOP
57	1	REVISION
60	36	RUN-TIME
61		RCS

Time: 11 seconds: 0 error(s): 0 warning(s): 11747 words free

```
.....  
:::EDSmallMemLog.MIDAS : Logger for EDSmallMem program  
:::                               By: T. Henning                               Nov. 20 1979  
.....
```

```
.start      L X AppendOutput EDSmallMem.report;  
            L X WriteMessage -***** START EDSmallMem Test : ;  
            L X WriteDT;  
            L X WriteMessage *****~ ;  
            L X Skip .continue;  
  
.breakpoint L X AppendOutput EDSmallMem.report;  
            L A18 SkipNE BADT;  
            L X Skip .badt;  
            L A18 SkipNE PATTERNERROR;  
            L X Skip .patternerror;  
            L A18 SkipNE PASSED-EDSMALLMEM-TEST;  
            L X Skip .passtest;  
  
.notmybreak L X AppendOutput EDSmallMem.report;  
            L X WriteMessage *** FAILED: Not at my breakpoint ~;  
  
            L X WriteMessage ' Parity = ;  
            R A0 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CIA = ;  
            R A18 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CTASK = ;  
            R A19 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APCTASK = ;  
            R A17 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APC = ;  
            R A16 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' TPC = ;  
            R A13 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X CloseOutput;  
            L X Exit;  
  
.badt      L X WriteMessage *** FAILED: at my Breakpoint ~;  
            L X WriteMessage * T register index miscompared ~;  
.bad      L X WriteMessage ' SUBTEST = ;  
            R B4 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' MEMADDR = ;  
            R C11 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' PASSCOUNT = ;  
            P B2 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X Skip .continue;
```

```
.patternerror L X WriteMessage *** FAILED: at my Breakpoint ~:
L X WriteMessage *          RESULT read did not match PATTERN written ~:
L X BackSkip .bad:

.passtest L X WriteMessage ----- PASSEd EDSmallMem Test : :
L X WriteDT:
L X WriteMessage -----
L X Skip .continue:

.continue L X writeMessage ~:
L X CloseOutput:
L X DisplayOn:
L X Confirm:
L X TimeOut 10000000:
L X Continue:
L X Skip 2:
L X ShowError Program failed to CONTINUE.:
L X BackSkip .notmybreak:
L X DisplayOff:
L X BackSkip .breakpoint:
```

L A19 Val 0  
L X Confirm  
L X Load EDSmallMem:

L B0 Addr REVISION  
L B1 Addr RUN-TIME  
L B2 Addr PASSCOUNT  
L B3 Addr MAXPASS  
L B4 Addr SUBTEST;  
L B9 Addr XA;  
L B10 Addr TESTCOUNTER;  
L B11 Addr CURRENTPATTERN;  
L B16 Addr CSOTEST;  
L B17 Addr CSITEST  
L B18 Addr CS2TEST  
L B19 Addr TMEMTEST

L C9 Addr PATTERN  
L C10 Addr RESULT  
L C11 Addr MEMADDR  
L C13 Addr STARTWORD  
L C14 Addr ENDWORD  
L C16 Addr PATTERNCHOICE;  
L C17 Addr SHORTLOOP;  
L X DisplayOn;  
L X TimeOut 10000  
L X SS GO  
L X Skip 1  
L X ShowError Single-step stuck at GO

%

\*\*\* \*\*

Revision 1

\*\*\* \*\*

\*\*\*\*\*

\*\*\* EDTask.mc : The Task Switching and Register Addressing Test Program

\*\*\* Purpose : To test task mechanism and register addressing modes.

\*\*\* Hardware Configuration : Standard 4 CPU boards

\*\*\* Written by : Tom Horsley, Dec. 12, 1977

\*\*\* Modified by : Bill Kennedy, Feb. 23, 1978

Added control store parity.

Took code off page 0.

\*\*\* Modified by : Bill Kennedy, Apr. 6, 1978

Stayed from a KERNEL register and added meaningful tags to Breakpoints.

\*\*\* Modified by : Chuck Thacker, Jan. 25, 1979

Changed for 8G configuration.

\*\*\* Modified by : Chuck Thacker, June 16, 1979

Changed to avoid stack overflow.

\*\*\* Modified by : Camellia Chan, Mar. 4, 1980

Standardize title page, code format, labels and looping.

\*\*\*\*\*

\*\*\*\*\*

## \*SubTest Description:

\* SubTest 1: In the READ test, confirm that the contents (Result) of the register read (TargetRegister) match those expected (TargetValue).

In this test, the absolute address of the register to be tested is calculated using RMOD, RSEL, TASK, STKSHFT, and REGSHIFT from XA. Then control store instruction 760 is stuffed with an appropriate read instruction. Finally a random value from XB is written into the absolute register location via the stack. a switch is made to the relevant task and the instruction at location 760 is executed.

\* SubTest 2: In the StackCompare test, confirm that the expected stack pointer (ExpectedStkp) matches the value of the STKP found immediately after the read or write instruction (SaveStkp).

\* SubTest 3: In the WRITE test, confirm that the contents (t) of the register written (TargetRegister) match those expected (TargetValue).

The write test is similar to the read test except that location 762 is stuffed and the test is not performed on some of the special hardware registers.

\* Note: Registers outside the range R46 to R360 are not read or written in any test.

\*\*\*\*\*

## \*BreakPoints:

\* ReadFail: the contents (Result) of the register read (TargetRegister) do not match those expected (TargetValue).

\* StackFail: the content (ExpectedStkp) does not match the value of the STKP found immediately after the read or write instruction (SaveStkp).

\* WriteFail: the contents (t) of the register written (TargetRegister) do not match those expected (TargetValue).

\* Passed-EDTask-Test: Passed all tests, and all passes.

\*\*\*\*\*

\* ShortLoop Logic Analyzer Sync Points at Control Store address:

\* ReadFail: Control Store address 770 at SubTest1.

\* StackFail: Control Store address 1546 at StackCompare.

\* WriteFail: Control Store address 772 at SubTest3.



```
*****
*Subroutine Description:
* FirstSixBits:   deciphers the first six bits of the absolute R-register from Target Task.
* LastTwoBits:   inserts the last two bits of the absolute R-register from t.
* NewInstx:      NewInst is now done when control is at 2001b+4*n, n=0-377b.
* PrimeTarget:   prime the target register with a value.
* ReadTest:     executes the read test.
* SetUpRead:    create environment for read test.
* StackCompare: make sure the stack pointer is correct.
* WriteTest:    sets up and executes the write test.

```

```
*****
*Special Reg. Definition:
* XA: The arguments of the test are built up from the random 16-bit word (XA) as follows:

```

```

RMOD      =   XA[0]
RSEL      =   XA[1-6]
TASK      =   XA[7-12]
PCF       =   XA[13-15]
DB        =   XA[12-17]
SB        =   XA[12-17]
STKSHFT   =   XA[7]
REGSHIFT  =   XA[10]
STKP      =   XA[10-17]
Rd/Wr     =   XA[13]

```

XA is used to choose a task and a register addressing mode (RMOD and RSEL).  
It is also used to decide whether to test reading or writing the register and to  
provide starting values for such registers as DB, SB, PCF and STKP.

- \* InnerLoopCounter: 16 bits inner loop counter.
- \* Note that the random number generator has been constructed so that it produces each  
number in the range [0, 64K) once and only once before repeating any number. Thus it  
is guaranteed to exhaust all possible combinations of the fields derived from it each  
time the inner loop is exhausted.
- \* PassCount: Outer loop pass counter,  
incremented each time when InnerLoopCounter reached the limit.
- \* MaxPass: number of times outer loop is to repeat before breakpointing.
- \* NewRand: 1 = change the pseudorandom number.  
0 = keep the current pseudorandom number.
- \* ShortLoop: At any breakpoint the user has the option of changing the value of ShortLoop  
to 1, which will cause the current subtest to loop endlessly. If ShortLoop is a zero  
then the program will proceed to the next subtest.

```
*****
%
```

\*\*\*\*\*

\*INITIALIZATION:

```

INSERT[DOLANG];
TITLE[EDTask];
SET[MainPage, 2];
SET[SubPage1, 1];
SET[sp1b,lshift[SubPage1,10]];
SET[SubPage2, 4];
SET[sp2b,lshift[SubPage2,10]];
SET[SubPage3, 3];

```

\*Task and register test program, ED revision  
\*definition of Main Program page  
\*definition of Subroutine page 1  
\*definition of Subroutine page 2  
\*definition of Subroutine page 3

\*\*\*\*\* Switch Base: \*\*\*\*\*

```

SET[SpecialBigSwitch, 1120];
SET[SpecialSmallSwitch, 1140];
SET[outerSwitch, 1240];
SET[StkSwitch0, 1460];
SET[StkSwitch1, 1500];
SET[RegBase, 40];

```

\*switch base for special register selection  
\*switch base for special register  
\*base of switch on RSEL[4:5]  
\*stack switch base  
\*stack switch base

\*\*\*\*\* Offsets: \*\*\*\*\*

```

SET[InitialStkpOffset, 0];
SET[TargetValueOffset, 1];
SET[TmpOffset, 2];
SET[SaveStkpOffset, 3];
SET[ResultOffset, 4];
SET[ReentryOffset, 5];

```

\*\*\*\*\* R-registers: \*\*\*\*\*

```

RV[Revision,0,1]
RV[Run-Time,1,5]

RV[InnerLoopCounter,2,0];
RV[PassCount,3];

RV[MaxPass,4,2];
RV[NewRand,5,1];

RV[ShortLoop,6,0];

RV[SubTest,7];

RV[DBTest,11];
RV[MiscTest,12];
RV[PCFTest,13];
RV[RRTest,14];
RV[SBTest,15];
RV[StkTest,16];

RV[CS0,17];
RV[CS1,20];
RV[CS2,21];
RV[DeltaStack,22];
RV[DummyRegister,23];
RV[ExpectedStkp,24];
RV[FieldMask,25];
RV[InitialCycCtl,26];

RV[InstructionAddress,27];
RV[NewTask,30];
RV[RegShiftFlag,31];
RV[StackShiftFlag,32];
RV[StkpTest,33];
RV[StuffTmp,34];
RV[TargetRegister,35];
RV[TargetTask,36];

```

\*REVISION 1  
\*Run-Time is 5 seconds

\*16 bits inner loop counter  
\*outer loop pass counter incremented each time when  
\* InnerLoopCounter reached the limit  
\*number of times outer loop is to repeat before breakpointing  
\*1 = change the pseudorandom number  
\*0 = keep the current pseudorandom number  
\*1 = loop endlessly on current subtest  
\*0 = proceed to the next subtest  
\*current location of test

\*number of times the DB register test has executed  
\*number of times the Misc registers test has executed  
\*number of times the PCF register test has executed  
\*number of times the straight register test has executed  
\*number of times the SB register test has executed  
\*number of times the stack register test has executed

\*temporary storage for first word of a control store location  
\*temporary storage for second word of a control store location  
\*temporary storage for third word of a control store location  
\*amount by which stack is to be incremented or decremented  
\*used as a place holder in instruction to be stuffed  
\*value of STKP after stack operation  
\*indicates part of result to be tested (usually all 1's)  
\*value to be stuffed into cyclecontrol immediately preceding  
\* read test  
\*location of CS instruction to be stuffed  
\*task switching contents for APC&APCTASK  
\*indicates that the REGSHIFT function is to be used  
\*indicates that the STACKSHIFT function is to be used  
\*indicates that the STKP is to be checked after the test  
\*used in the stuff operations  
\*the absolute address of the register to be tested  
\*the task that will be briefly entered for test purposes

```
RV[InitialStkp,40,ADD[RegBase, InitialStkpOffset]]: *value of STKP before stack operation
RV[TargetValue,41,ADD[RegBase, TargetValueOffset]]: *value to be stuffed in register
RV[Tmp,42,ADD[RegBase, TmpOffset]]: *used to load APC in remote task
* (different for each task)
RV[SaveStkp,43,ADD[RegBase, SaveStkpOffset]]: *contents of STKP placed here by upper task
RV[Result,44,ADD[RegBase, ResultOffset]]: *contents of target register placed here
* by upper task
RV[WriteTestReentryLoc,45,ADD[RegBase, ReentryOffset]]: *location in task 0 to return to

RV[CA,52]: *used in random number generation, A*XA + CA
RV[CB,53]: *used in random number generation, A*XB + CB
RV[XA,54]: *random number generated via A*XA + CA
RV[XB,55]: *used in random number generation, A*XB + CB
```

```

*****
****  MAIN routine:
ONPAGE[MainPage];

go:
start: *RandomInit (Initialize random generator registers: XA ← 123, CA ← 33031)
      XA ← AND@[0377, 123]C;          *Load16Bits (XA ← 123)
      XA ← (XA) OR (AND@[177400, 123]C);

      CA ← AND@[0377, 33031]C;        *Load16Bits (CA ← 33031)
      CA ← (CA) OR (AND@[177400, 33031]C);

      *RandomInit (Initialize random generator registers: XB ← 456, CB ← 33035)
      XB ← AND@[0377, 456]C;          *Load16Bits (XB ← 456)
      XB ← (XB) OR (AND@[177400, 456]C);

      CB ← AND@[0377, 33035]C;        *Load16Bits (CB ← 33035)
      CB ← (CB) OR (AND@[177400, 33035]C);

      CLEARMPANEL;
      PassCount ← 0C;
      RRTest ← 0C;
      PCFTest ← 0C;
      SBTest ← 0C;
      DBTest ← 0C;
      StkTest ← 0C;
      MiscTest ← 0C;

      WriteTestReentryLoc ← AND@[0377, 772]C; *Load16Bits (WriteTestReentryLoc ← 772)
      WriteTestReentryLoc ← (WriteTestReentryLoc) OR (AND@[177400, 772]C);

bigLoop:  INCMPANEL;
          t ← PassCount ← (PassCount) + 1;
          LU ← (MaxPass) - (t);          * check for maximum pass counter reached
          GOTO[mainLoop, ALU >= 0];

Passed-EDTask-Test:  BREAKPOINT, goto[go];

****  SUBTEST0  ****
mainLoop:  SubTest ← 0C;
          RegShiftFlag ← 0C;
          StackShiftFlag ← 0C;
          StkpTest ← 0C;
          InitialStkp ← 20c;
          FieldMask ← (ZERO) - 1;

          LU ← (NewRand);
          GOTO[SetTTask, ALU = 0];

          InnerLoopCounter ← (InnerLoopCounter) + 1;
          GOTO[bigLoop, CARRY];

      *Random (4005*XA + CA mod 2**16)
      t ← XA;
      t ← (LSH[XA, 2]) + t;
      t ← (LSH[XA, 13]) + t;
      t ← (CA) + t;
      XA ← t;

SetTTask:  t ← LDF[XA, 7, 4];
          TargetTask ← t;
          lu ← (TargetTask) - (16C);
          GOTO[ChooseTest, ALU < 0];
          GOTO[mainLoop];          *don't touch tasks 16 or 17 (kernel)

*choose test
ChooseTest:  LU ← (XA) AND (100000C);
          GOTO[OutSwitch, ALU # 0];

*xxxxxxRR
          LOADPAGE[SubPage3];
          CALLP[FirstSixBits];
          RRTest ← (RRTest) + 1;
          GOTO[RunSubTest];

```

\* Switch4 (Use DISPATCH[XA, 5, 2] to select cases. Locate switch table at outerSwitch.)

```
OutSwitch:   SET[Switch10, outerSwitch];
             DISPATCH[XA, 5, 2];
             DISP[SwitchTab10];
```

```
SwitchTab10: GOTO[Case10], AT[Switch10, 0];
              GOTO[Case11], AT[Switch10, 1];
              GOTO[Case12], AT[Switch10, 2];
              GOTO[Case13], AT[Switch10, 3];
```

```
*xxxxxxPP - PCF
Case10:     LOADPAGE[SubPage3];
             CALLP[FirstSixBits];
             t ← LDF[XA, 13, 2];
             LOADPAGE[SubPage3];
             CALLP[LastTwoBits];
             t ← LDF[XA, 13, 3];
             Tmp ← t;
             PCF ← Tmp;
             PCFTest ← (PCFTest) + 1;
             GOTO[RunSubTest];
```

```
*xxxxxxSS - SB
Case11:     LOADPAGE[SubPage3];
             CALLP[FirstSixBits];
             t ← LDF[XA, 12, 2];
             LOADPAGE[SubPage3];
             CALLP[LastTwoBits];
             t ← LDF[XA, 12, 6];
             Tmp ← t;
             SB ← Tmp;
             BBFB;
             SBTest ← (SBTest) + 1;
             GOTO[RunSubTest];
             *to advance SB to SBX
```

```
*xxxxxxDD - DB
Case12:     LOADPAGE[SubPage3];
             CALLP[FirstSixBits];
             t ← LDF[XA, 12, 2];
             LOADPAGE[SubPage3];
             CALLP[LastTwoBits];
             t ← LDF[XA, 12, 6];
             Tmp ← t;
             DB ← Tmp;
             BBFB;
             DBTest ← (DBTest) + 1;
             GOTO[RunSubTest];
             *to advance DB to DBX
```

```
Case13:     t ← 3C;
             LU ← (LDF[XA, 1, 2]) - (t);
             GOTO[SpecialReg. ALU # 0];
```

```
* SSSSSSSS - STKP
             LOADPAGE[SubPage3];
             GOTOP[.+1];
             ONPAGE[SubPage3];

             t ← LDF[XA, 10, 10];
             TargetRegister ← t;
             t ← LDF[XA, 7, 1];
             StackShiftFlag ← t;
             t ← LDF[XA, 10, 10];
             ExpectedStkp ← t;
             t ← (ExpectedStkp);
             InitialStkp ← t;
             lu ← (InitialStkp) and not (37c);
             goto[.+3,alu#0];
```

```
AbandonTest: loadpage[mainPage];
              gotop[mainLoop];
              *don't load stkp with <40b

              LU ← (StackShiftFlag);
              GOTO[StkSwitchB. ALU # 0];
```

```

* Switch4 (Use DISPATCH[XA, 3, 2] to select cases. Locate switch table at StkSwitch0.)
StkSwitchA:  SET[Switch20, StkSwitch0];
             DISPATCH[XA, 3, 2];
             DISP[SwitchTab20];

SwitchTab20: GOTO[Case20], AT[Switch20, 0];
             GOTO[Case21], AT[Switch20, 1];
             GOTO[Case22], AT[Switch20, 2];
             GOTO[Case23], AT[Switch20, 3];

Case20:     DeltaStack ← 0C;
             GOTO[DltStack];

Case21:     DeltaStack ← 1C;
             GOTO[DltStack];

Case22:     DeltaStack ← (ZERO) - 1;
             GOTO[DltStack];

Case23:     t ← 2C;
             DeltaStack ← (ZERO) - (t);
             GOTO[DltStack];

* Switch4 (Use DISPATCH[XA, 3, 2] to select cases. Locate switch table at StkSwitch1.)
StkSwitchB:  SET[Switch30, StkSwitch1];
             DISPATCH[XA, 3, 2];
             DISP[SwitchTab30];

SwitchTab30: GOTO[Case30], AT[Switch30, 0];
             GOTO[Case31], AT[Switch30, 1];
             GOTO[Case32], AT[Switch30, 2];
             GOTO[Case33], AT[Switch30, 3];

Case30:     DeltaStack ← 2C;
             GOTO[DltStack];

Case31:     DeltaStack ← 3C;
             GOTO[DltStack];

Case32:     LOADPAGE[MainPage];
             GOTOP[mainLoop];

Case33:     t ← 3C;
             DeltaStack ← (ZERO) - (t);

DltStack:   LU ← (XA) AND (000020C);
             GOTO[SubtractStk, ALU # 0];

*add DeltaStack to lower 4 bits of ExpectedStkp with no carry into upper bits
t ← (ExpectedStkp) AND (17C);
ExpectedStkp ← (ExpectedStkp) AND NOT (17C);
t ← (DeltaStack) + (t);
Tmp ← t;
t ← (Tmp) AND (17C);
ExpectedStkp ← (ExpectedStkp) OR (t);
GOTO[SetStkp];

*subtract DeltaStack from lower 4 bits of InitialStkp with no carry into upper bits
SubtractStk: t ← (InitialStkp) AND (17C);
             InitialStkp ← (InitialStkp) AND NOT (17C);
             Tmp ← t;
             Tmp ← (Tmp) + (20C);
             t ← DeltaStack;
             Tmp ← (Tmp) - (t);
             t ← (Tmp) AND (17C);
             InitialStkp ← (InitialStkp) OR (t);

SetStkp:    StkpTest ← 1C;
             StkTest ← (StkTest) + 1;
             LOADPAGE[2];
             GOTOP[.+1];
             ONPAGE[2];

             GOTO[RunSubTest];

```

\*Special Registers

```

SpecialReg:    t ← LDF[XA, 10, 1];
               RegShiftFlag ← t;

               LU ← (NewRand);
               GOTO[SetTarValue, ALU = 0];

*Random (4005*XB + CB mod 2**16)
t ← XB;
t ← (LSH[XB, 2]) + t;
t ← (LSH[XB, 13]) + t;
t ← (CB) + t;
XB ← t;

SetTarValue:  t ← (XB);
               TargetValue ← t;

               LU ← (XA) AND (040000C);
               GOTO[SmallSwitch, ALU # 0];

* Switch8 (Use DISPATCH[XA, 2, 3] to select cases. Locate switch table at SpecialBigSwitch.)
BigSwitch:    SET[Switch40, SpecialBigSwitch];
               DISPATCH[XA, 2, 3];
               DISP[SwitchTab40];

SwitchTab40:  GOTO[Case40], AT[Switch40, 0];
               GOTO[Case41], AT[Switch40, 1];
               GOTO[Case42], AT[Switch40, 2];
               GOTO[Case43], AT[Switch40, 3];
               GOTO[Case44], AT[Switch40, 4];
               GOTO[Case45], AT[Switch40, 5];
               GOTO[Case46], AT[Switch40, 6];
               GOTO[Case47], AT[Switch40, 7];

* RSEL = 3, SSTKP, STKP
*Load SSTKP
Case40:       t ← LDF[XB, 0, 10];
               Tmp ← t;
               lu ← (Tmp) and not (37c);
               goto[.+3,alu#0];

AbandonTest1: loadpage[mainPage];
               gotop[mainLoop];           *don't load stkp with <40

               STKP ← Tmp;
               loadpage[4];
               callp[NewInstx],           *NewInst is now done when control is at 2001b+4*n, n=0-377b

*Load STKP
t ← LDF[XB, 10, 10];
InitialStkp ← t;
InitialStkp ← (InitialStkp) xor (377c);
lu ← (InitialStkp) and not (37c);
goto[IncMiscTest,ALU#0];
goto[AbandonTest1];

* RSEL = 7, (ALURESULT), SALUF
Case41:       FieldMask ← 377C;
               t ← TargetValue;
               t ← (ZERO) OR NOT (t);
               SALUF ← t;
               GOTO[IncMiscTest];

* RSEL = 13, MEMSYNDROME
Case42:       GOTO[mainLoop];

* RSEL = 17, MEMERROR
Case43:       GOTO[mainLoop];

* RSEL = 23, UNUSED
Case44:       GOTO[mainLoop];

* RSEL = 27
Case45:       LU ← (RegShiftFlag);
               GOTO[GoBack, ALU # 0];

* CYCLECONTROL, PCXREG, PCFREG

```

```

FieldMask ← AND@[0377, 177567]C;          *Load16Bits (FieldMask ← 177567)
FieldMask ← (FieldMask) OR (AND@[177400, 177567]C);

* Load PCXREG
t ← LDF[XB, 10, 4];
Tmp ← t;
PCF ← Tmp;
loadpage[4];
callp[NewInstx];          *NewInst is now done when control is at 2001b+4*n, n=0-377b

* NEWINST:
* Load PCXREG
t ← LDF[XB, 14, 4];
Tmp ← t;
PCF ← Tmp;

* Load CycleControl (DBX and MWX)
t ← LDF[XB, 0, 10];
InitialCycCtl ← t;
GOTO[IncMiscTest];

* PRINTER
GoBack:      GOTO[mainLoop];

* RSEL = 33
Case46:      LU ← (RegShiftFlag);
             GOTO[SetDB&SB, ALU # 0];

* TIMER
             GOTO[mainLoop];

* DBREG, SBREG
SetDB&SB:    FieldMask ← AND@[0377, 7777]C;          *Load16Bits (FieldMask ← 7777)
             FieldMask ← (FieldMask) OR (AND@[177400, 7777]C);
             t ← LDF[TargetValue, 4, 6];
             Tmp ← t;
             DB ← Tmp;
             t ← LDF[TargetValue, 12, 6];
             Tmp ← t;
             SB ← Tmp;
             BBFB;          *to advance DB to DBX and SB to SBX
             GOTO[IncMiscTest];

* RSEL = 37
Case47:      LU ← (RegShiftFlag);
             GOTO[SetMNBR, ALU # 0];

* RS232
             GOTO[mainLoop];

* MNBR
SetMNBR:     MNBR ← TargetValue;
             GOTO[IncMiscTest];

* Switch4 (Use DISPATCH[XA, 3, 2] to select cases. Locate switch table at SpecialSmallSwitch.)
SmallSwitch: SET[Switch50, SpecialSmallSwitch];
             DISPATCH[XA, 3, 2];
             DISP[SwitchTab50];

SwitchTab50: GOTO[Case50], AT[Switch50, 0];
             GOTO[Case51], AT[Switch50, 1];
             GOTO[Case52], AT[Switch50, 2];
             GOTO[Case53], AT[Switch50, 3];

* RSEL = 43, APCTASK, APC
Case50:     GOTO[mainLoop];

* RSEL = 47, CTASK, NCIA
Case51:     GOTO[mainLoop];

* RSEL = 53, CSDATA
* CSDATA will be loaded just prior to switching to higher task
Case52:     GOTO[IncMiscTest];

* RSEL = 57, PAGE, PARITY, BOOTREASON
Case53:     GOTO[mainLoop];

```



```
IncMiscTest:  MiscTest ← (MiscTest) + 1;
              LOADPAGE[SubPage1];
              CALLP[SetUpRead];
              LOADPAGE[SubPage1];
              CALLP[ReadTest];
              GOTO[mainLoop];

RunSubTest:   LOADPAGE[SubPage3];
              GOTOP[.+1];
              ONPAGE[SubPage3];

      *actual test
      LU ← (XA) AND (000020C);
      GOTO[Write, ALU #.0];

      *Read Test
      LOADPAGE[SubPage1];
      CALLP[SetUpRead];
      LOADPAGE[SubPage2];
      CALLP[PrimeTarget];
      LOADPAGE[SubPage1];
      CALLP[ReadTest];
      GOTO[CheckStkp];

      *Write Test
Write:        LOADPAGE[SubPage2];
              CALLP[WriteTest];

CheckStkp:   LU ← (StkpTest);
              GOTOP[Repeat, ALU = 0];
              LOADPAGE[SubPage3];
              CALLP[StackCompare];
              nop;

Repeat:      LOADPAGE[MainPage];
              GOTOP[mainLoop];
```

```
*****
*SUBROUTINE
***** SUBROUTINE: FirstSixBits *****
*
*   deciphers the first six bits of the absolute R-register from Target Task
ONPAGE[SubPage3];
FirstSixBits:  LU ← (XA) AND (060000C);
               GOTO[ShiftTarget, ALU # 0];

               *TTTTRRxx
               t ← LSH[TargetTask, 4];
               t ← (LDF[XA, 3, 4]) OR t;
               GOTO[SetTarget];

               *TTRRRRxx
ShiftTarget:  t ← RSH[TargetTask, 2];
               TargetRegister ← t;
               t ← LSH[TargetRegister, 6];
               t ← (LDF[XA, 1, 6]) OR t;

SetTarget:   TargetRegister ← t;
               RETURN;

***** SUBROUTINE: LastTwoBits *****
*
*   inserts the last two bits of the absolute R-reg from t
ONPAGE[SubPage3];
LastTwoBits:  TargetRegister ← RSH[TargetRegister, 2];
               TargetRegister ← (LSH[TargetRegister, 2]) OR t;
               RETURN;

***** SUBROUTINE: NewInstx *****
*
*   NewInst is now done when control is at 2001b+4*n, n=0-377b
OnPage[SubPage2];
NewInstx:    Return, at[2001];
```

```
***** SUBROUTINE: PrimeTarget *****
*
*   prime the target register with a value
ONPAGE[SubPage2]:
PrimeTarget:   LU ← (TargetRegister) - (46C);
               GOTO[SkipRead, ALU < 0];
               t ← 360C;
               LU ← (TargetRegister) - (t);
               GOTO[SetStkp2, ALU < 0];
               NOP;
               *done this way (with t) due to assembler bug

SkipRead:     LoadPage[mainPage];
               gotop[mainLoop];
               *don't do read test

SetStkp2:     stkp ← TargetRegister;
               LU ← (NewRand);
               GOTO[SetTarValue2, ALU = 0];

               *Random (4005*XB + CB mod 2**16)
               t ← XB;
               t ← (LSH[XB, 2]) + t;
               t ← (LSH[XB, 13]) + t;
               t ← (CB) + t;
               XB ← t;

SetTarValue2: t ← XB;
               TargetValue ← t;
               STACK ← t;
               RETURN;
```

\*\*\*\*\* SUBROUTINE: ReadTest \*\*\*\*\*

\*  
\* executes the read test.

ONPAGE[SubPage1]:

```

*switch to higher task
*set up CSDATA
ReadTest:  LU ← TargetValue;
           APC&APCTASK ← InstructionAddress;
           WRITECS1;
           t ← 1C,at[sp1b,14];
           APC&APCTASK ← InstructionAddress;
           READCS;
           CYCLECONTROL ← InitialCycCtl,at[sp1b,16];
           STKP ← InitialStkp;

           APC&APCTASK ← NewTask;
           RETURN;

```

HighTask760:

```

*IN HIGHER TASK
  t ← DummyRegister, REGSHIFT, AT[760];

```

```

*Save test results
  Result ← t;
  t ← (GETRSPEC[103]) OR (177400C);

```

\*Result will be different for each task  
\*get the STKP in a form ready to be inverted

```

*Put test results in low register memory
  Tmp ← ADD[RegBase, SaveStkpOffset]C;
  STKP ← Tmp;
  STACK ← (ZERO) OR NOT (t);
  t ← Result;
  STACK&+1 ← t;

```

\*write SaveStkp( inverting STKP first)  
\*write Result

```

*Load16Bits (Tmp ← 770)
  Tmp ← AND@[0377, 770]C;
  Tmp ← (Tmp) OR (AND@[177400, 770]C);

```

```

*Tmp will be different for each task
  APC&APCTASK ← Tmp;
  RETURN;

```

\*\*\*\*\* SUBTEST1 \*\*\*\*\*

```

SubTest1:  SubTest ← 1C, AT[770];
           t ← FieldMask;
           TargetValue ← (TargetValue) AND (t);
           t ← Result ← (Result) AND (t);
           LU ← (TargetValue) - (t);
           GOTO[ReadSuccess, ALU = 0];

           ShortLoop ← ShortLoop, GOTO[.+2, R EVEN]; *Test for shortLoop option
           GOTO[SubTest1]; *ShortLoop selected

```

```

ReadFail:  BREAKPOINT;

           ShortLoop ← ShortLoop, GOTO[.+2, R EVEN]; *Test for shortLoop option
           GOTO[SubTest1]; *ShortLoop selected
           NOP; *resolves a branching conflict

```

```

ReadSuccess: RETURN;

```

\*\*\*\*\* SUBROUTINE: SetUpRead \*\*\*\*\*

\*  
\* create environment for read test

ONPAGE[SubPage1]:  
NOP, AT[777]:

SetUpRead:

\*modify the register reference instruction  
InstructionAddress ← AND@[0377, 760]C; \*Load16Bits (InstructionAddress + 760)  
InstructionAddress ← (InstructionAddress) OR (AND@[177400, 760]C);

\*readCS (read control store location 'InstructionAddress')

\*use CS2 for a temporary register until end

t ← (InstructionAddress);

CS2 ← t;

t ← 0C;

APCTASK&APC ← (CS2);

READCS:

t ← CSData, AT[sp1b,2];

CS0 ← t;

t ← 1C;

APCTASK&APC ← (CS2);

READCS:

t ← CSData, AT[sp1b,4];

CS1 ← t;

t ← 3C;

APCTASK&APC ← (CS2);

READCS:

t ← CSData, AT[sp1b,6];

CS2 ← t;

CS2 ← RSH[CS2, 14];

\*stuffRsel (set the RMOD and RSEL fields in CS0, CS1, CS2 to contents of XA)

\*CopyField (CS0[01, 5] ← XA[0, 5], temp is stuffTmp)

t ← LDF[XA, 0, 5];

stuffTmp ← 5C;

stuffTmp ← (stuffTmp) - 1;

stuffTmp ← (stuffTmp) OR (LSHIFT[01, 4]C);

CYCLECONTROL ← stuffTmp;

stuffTmp ← t;

t ← WFA[stuffTmp];

CS0 ← WFB[(CS0) OR t];

CS0 ← (CS0) XOR (030000C);

\*invert bits in microinstruction

\*CopyField (CS2[14, 2] ← XA[5, 2], temp is stuffTmp)

t ← LDF[XA, 5, 2];

stuffTmp ← 2C;

stuffTmp ← (stuffTmp) - 1;

stuffTmp ← (stuffTmp) OR (LSHIFT[14, 4]C);

CYCLECONTROL ← stuffTmp;

stuffTmp ← t;

t ← WFA[stuffTmp];

CS2 ← WFB[(CS2) OR t];

\*stuffF2 (set the F2 field in CS0, CS1, CS2 to 12)

StuffTmp ← OR@[LSHIFT[2, 4], SUB[4, 1]]C;

CYCLECONTROL ← StuffTmp;

StuffTmp ← 12C;

t ← WFA[StuffTmp];

CS1 ← WFB[(CS1) OR t];

LU ← (StackShiftFlag);

GOTO[CheckFlag, ALU = 0];

\*stuffF2 (set the F2 field in CS0, CS1, CS2 to 03)

StuffTmp ← OR@[LSHIFT[2, 4], SUB[4, 1]]C;

CYCLECONTROL ← StuffTmp;

StuffTmp ← 03C;

t ← WFA[StuffTmp];

CS1 ← WFB[(CS1) OR t];

CheckFlag: LU ← (RegShiftFlag);  
GOTO[WriteCS, ALU = 0];

```

*stuffF2 (set the F2 field in CS0, CS1, CS2 to 00)
  StuffTmp ← OR@[LSHIFT[2, 4], SUB[4, 1]]C;
  CYCLECONTROL ← StuffTmp;
  StuffTmp ← 00C;
  t ← WFA[StuffTmp];
  CS1 ← WFB[(CS1) OR t];

*writeCS (write control store location 'InstructionAddress')
WriteCS:
  t ← (CS0);
  Tmp ← t;
  t ← (CS1);
  Tmp ← t ← (Tmp) XOR (t);
  t ← (LDF[CS2,14,4]) XOR (t);
  Tmp ← t ← (LDF[Tmp,0,10]) XOR (t);
  Tmp ← t ← (LDF[Tmp,10,4]) XOR (t);
  Tmp ← t ← (LDF[Tmp,14,2]) XOR (t);
  Tmp ← t ← (LDF[Tmp,16,1]) XNOR (t);
  t ← (LDF[Tmp,17,1]);
  CS1 ← (CS1) XOR (t);
  t ← (CS2);
  LU ← (CS0);
  APCTASK&APC ← (InstructionAddress);
  WriteCS0&2;
  LU ← (CS1), at[sp1b,10];
  APCTASK&APC ← (InstructionAddress);
  WriteCS1;

*set up higher task
  t ← LSH[TargetTask, 14], at[sp1b,12];
  NewTask ← t;

  Tmp ← AND@[0377, 760]C;
  Tmp ← (Tmp) OR (AND@[177400, 760]C);
  t ← Tmp;
  NewTask ← (NewTask) OR (t);

*set up for writing into control store the value that will be read into CSDATA
  InstructionAddress ← AND@[0377, 777]C;
  InstructionAddress ← (InstructionAddress) OR (AND@[177400, 777]C);

  RETURN;

```

```

***** SUBROUTINE: StackCompare *****

```

```

*
*   make sure the stack pointer is correct

```

```

ONPAGE[SubPage3];

```

```

***** SUBTEST2 *****

```

```

StackCompare:  SubTest ← 2C;
               t ← (SaveStkp) AND (0377C);
               LU ← (ExpectedStkp) - (t);
               GOTO[StackSuccess, ALU = 0];

```

```

               ShortLoop ← ShortLoop, GOTO[.+2, R EVEN];
               GOTO[StackCompare];
               *Test for shortLoop option
               *ShortLoop selected

```

```

StackFail:    BREAKPOINT;

```

```

               ShortLoop ← ShortLoop, GOTO[.+2, R EVEN];
               GOTO[StackCompare];
               NOP;
               *Test for shortLoop option
               *ShortLoop selected
               *resolves a branching conflict

```

```

StackSuccess: RETURN;

```

```

***** SUBROUTINE: WriteTest *****
*
*   sets up and executes the write test

ONPAGE[SubPage2];

WriteTest:
  *modify the register reference instruction
  InstructionAddress ← AND@[0377, 762]C; *Load16Bits (InstructionAddress ← 762)
  InstructionAddress ← (InstructionAddress) OR (AND@[177400, 762]C);

  *readCS (read control store location 'InstructionAddress')
  *use CS2 for a temporary register until end
  t ← (InstructionAddress);
  CS2 ← t;
  t ← 0C;
  APCTASK&APC ← (CS2);
  READCS;
  t ← CSData.at[sp2b,20];
  CS0 ← t;
  t ← 1C;
  APCTASK&APC ← (CS2);
  READCS;
  t ← CSData.at[sp2b,22];
  CS1 ← t;
  t ← 3C;
  APCTASK&APC ← (CS2);
  READCS;
  t ← CSData.at[sp2b,24];
  CS2 ← t;
  CS2 ← RSH[CS2, 14];

  *stuffRsel (set the RMOD and RSEL fields in CS0, CS1, CS2 to contents of XA)
  *CopyField (CS0[01, 5] ← XA[0, 5], temp is stuffTmp)
  t ← LDF[XA, 0, 5];
  stuffTmp ← 5C;
  stuffTmp ← (stuffTmp) - 1;
  stuffTmp ← (stuffTmp) OR (LSHIFT[01, 4]C);
  CYCLECONTROL ← stuffTmp;
  stuffTmp ← t;
  t ← WFA[stuffTmp];
  CS0 ← WFB[(CS0) OR t];
  CS0 ← (CS0) XOR (030000C);
  *invert bits in microinstruction

  *CopyField (CS2[14, 2] ← XA[5, 2], temp is stuffTmp)
  t ← LDF[XA, 5, 2];
  stuffTmp ← 2C;
  stuffTmp ← (stuffTmp) - 1;
  stuffTmp ← (stuffTmp) OR (LSHIFT[14, 4]C);
  CYCLECONTROL ← stuffTmp;
  stuffTmp ← t;
  t ← WFA[stuffTmp];
  CS2 ← WFB[(CS2) OR t];

  *stuffF2 (set the F2 field in CS0, CS1, CS2 to 12)
  StuffTmp ← OR@[LSHIFT[2, 4], SUB[4, 1]]C;
  CYCLECONTROL ← StuffTmp;
  StuffTmp ← 12C;
  t ← WFA[StuffTmp];
  CS1 ← WFB[(CS1) OR t];
  LU ← (StackShiftFlag);
  GOTO[CheckFlag2, ALU = 0];

  *stuffF2 (set the F2 field in CS0, CS1, CS2 to 03)
  StuffTmp ← OR@[LSHIFT[2, 4], SUB[4, 1]]C;
  CYCLECONTROL ← StuffTmp;
  StuffTmp ← 03C;
  t ← WFA[StuffTmp];
  CS1 ← WFB[(CS1) OR t];

CheckFlag2:  LU ← (RegShiftFlag);
             GOTO[WriteCS2, ALU = 0];

  *stuffF2 (set the F2 field in CS0, CS1, CS2 to 00)

```

```

StuffTmp ← OR@[LSHIFT[2, 4], SUB[4, 1]]C;
CYCLECONTROL ← StuffTmp;
StuffTmp ← OOC;
t ← WFA[StuffTmp];
CS1 ← WFB[(CS1) OR t];

```

WriteCS2:

```

*writeCS (write control store location 'InstructionAddress')
t ← (CS0);
Tmp ← t;
t ← (CS1);
Tmp ← t ← (Tmp) XOR (t);
t ← (LDF[CS2,14,4]) XOR (t);
Tmp ← t ← (LDF[Tmp,0,10]) XOR (t);
Tmp ← t ← (LDF[Tmp,10,4]) XOR (t);
Tmp ← t ← (LDF[Tmp,14,2]) XOR (t);
Tmp ← t ← (LDF[Tmp,16,1]) XNOR (t);
t ← (LDF[Tmp,17,1]);
CS1 ← (CS1) XOR (t);
t ← (CS2);
LU ← (CS0);
APCTASK&APC ← (InstructionAddress);
WriteCS0&2;
LU ← (CS1),at[sp2b,26];
APCTASK&APC ← (InstructionAddress);
WriteCS1;

*set up higher task
t ← LSH[TargetTask, 14],at[sp2b,30];
NewTask ← t;

Tmp ← AND@[0377, 761]C;
Tmp ← (Tmp) OR (AND@[177400, 761]C);
t ← Tmp;
NewTask ←(NewTask) QR (t);

*check range of target register and establish target value
LU ← (TargetRegister) - (46C);
GOTO[BackToMain, ALU < 0];

t ← 360C;
LU ← (TargetRegister) + (t);
GOTO[ChangeXB, ALU < 0];
NOP;

BackToMain:  LOADPAGE[MainPage];
             GOTOP[mainLoop];

ChangeXB:   LU ← (NewRand);
             GOTO[SetTargetVal, ALU = 0];

*Random (4005*XB + CB mod 2**16)
t ← XB;
t ← (LSH[XB, 2]) + t;
t ← (LSH[XB, 13]) + t;
t ← (CB) + t;
XB ← t;

SetTargetVal: t ← XB;
              TargetValue ← t;

*switch to higher task
APC&APCTASK ← NewTask;
RETURN;

*IN HIGHER TASK
HighTask761: Tmp ← ADD[RegBase, TargetValueOffset]C, AT[761];
             STKP ← Tmp;
             t ← STACK&-1;
             STKP ← STACK;
             DummyRegister ← t, REGSHIFT, AT[762];

*save STKP
t ← ADD[RegBase, SaveStkpOffset]C;
t ← (ZERO) OR NOT (t);
SALUF ← t;

```



```
t ← (GETRSPEC[103]) OR (177400C); *get the STKP in a form ready to be inverted
STKP ← GETRSPEC[107]; *use SALUF as general purpose register
STACK ← (ZERO) OR NOT (t); *write SaveStkp (invert STKP)
LU ← STACK&+2; *increment STKP
APC&APCTASK ← STACK; *get return register off of stack
RETURN;
```

## \*\*\*\*\* SUBTEST3 \*\*\*\*\*

```
SubTest3: SubTest ← 3C, AT[772];
          STKP ← TargetRegister;
          t ← STACK;
          LU ← (TargetValue) - (t);
          GOTO[WriteSuccess, ALU = 0];

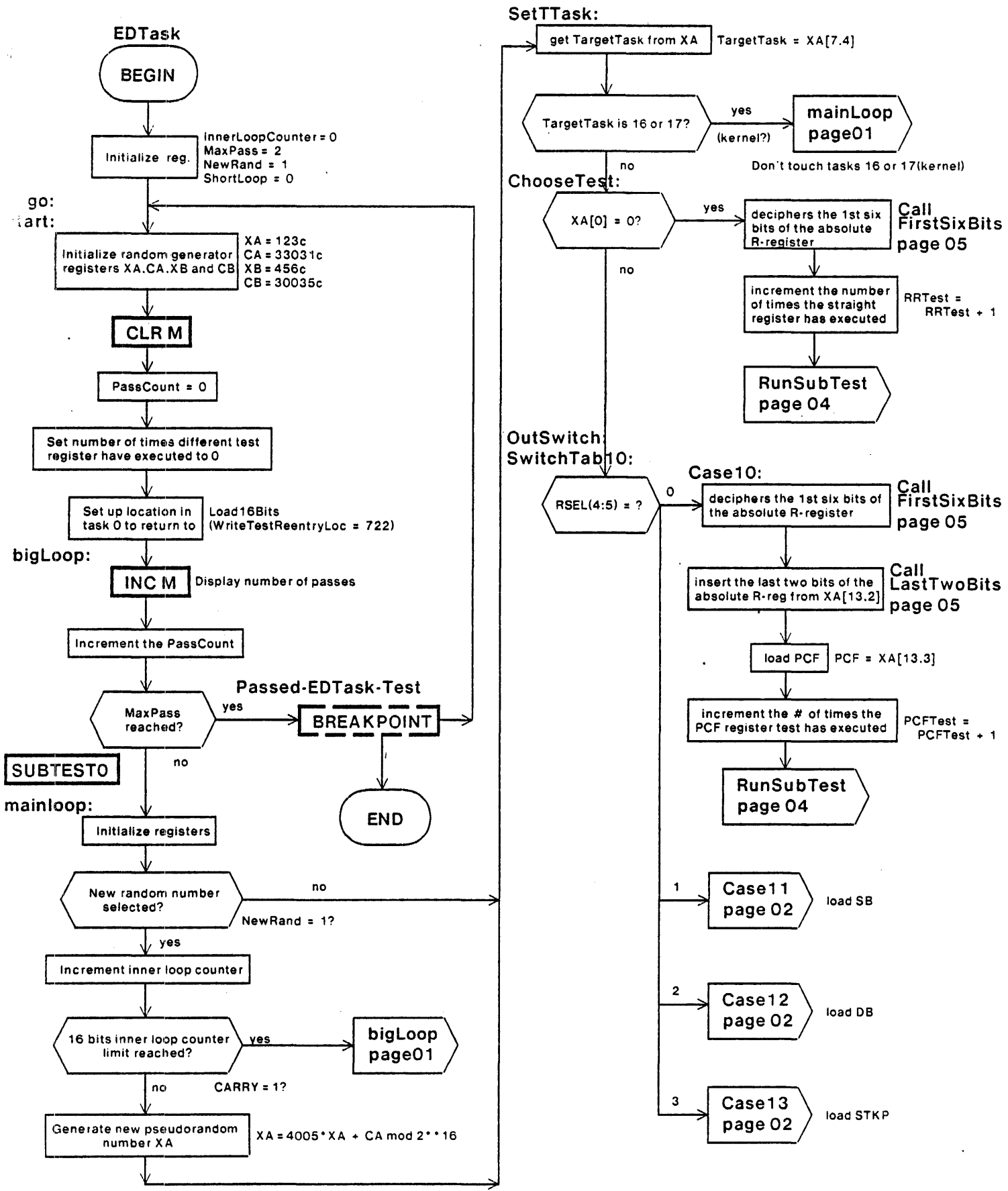
          ShortLoop ← ShortLoop, GOTO[.+2, R EVEN]; *Test for shortLoop option
          GOTO[SubTest3]; *ShortLoop selected

WriteFail: BREAKPOINT;

          ShortLoop ← ShortLoop, GOTO[.+2, R EVEN]; *Test for shortLoop option
          GOTO[SubTest3]; *ShortLoop selected
          NOP; *resolves a branching conflict

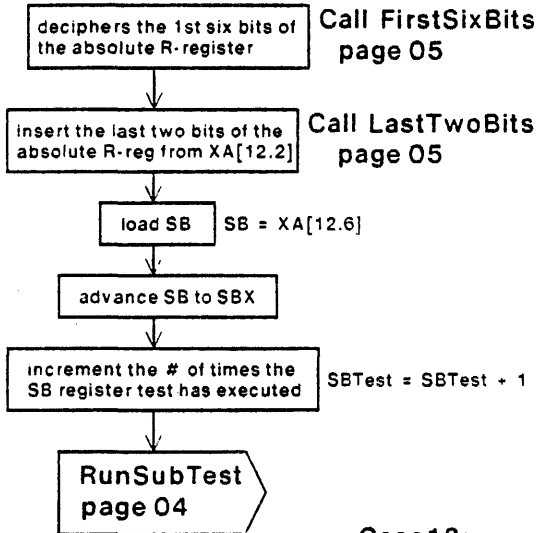
WriteSuccess: RETURN;

          END; * to end the main routine
```

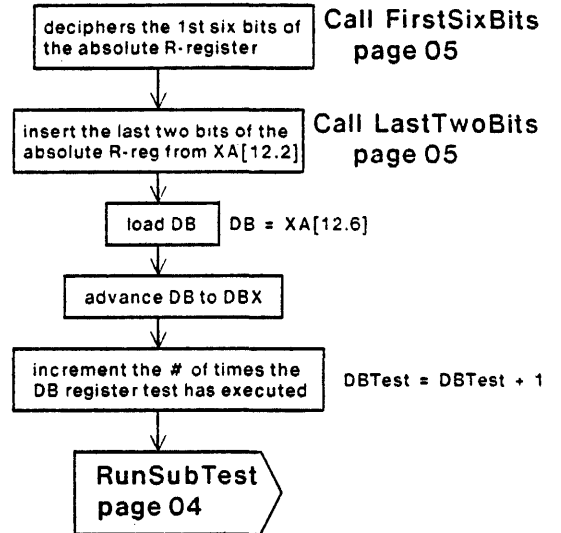


XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDTask	EDTask-01.sil	Camellia Chan	1	03/20/80	01

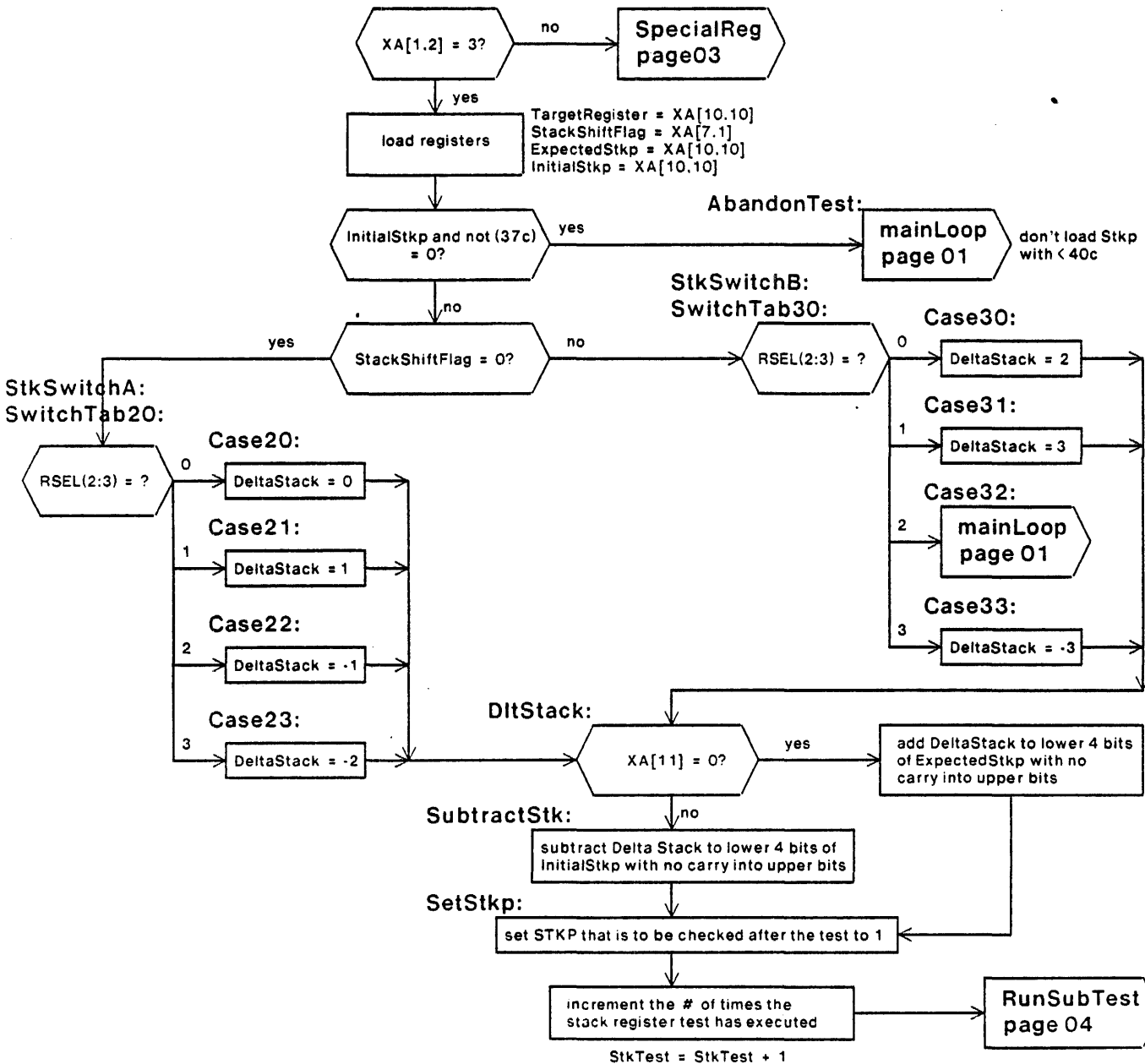
Case11:



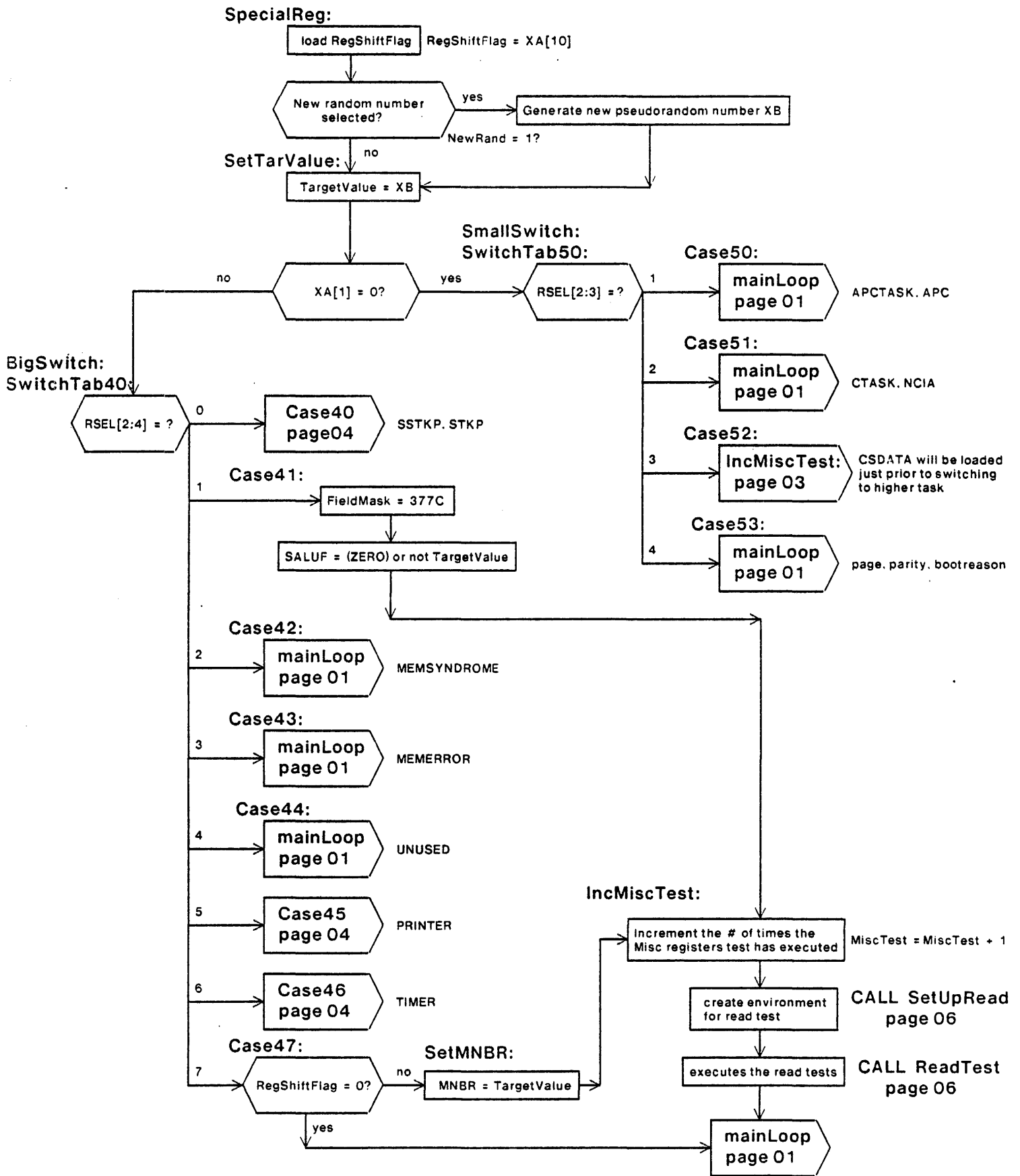
Case12:



Case13:

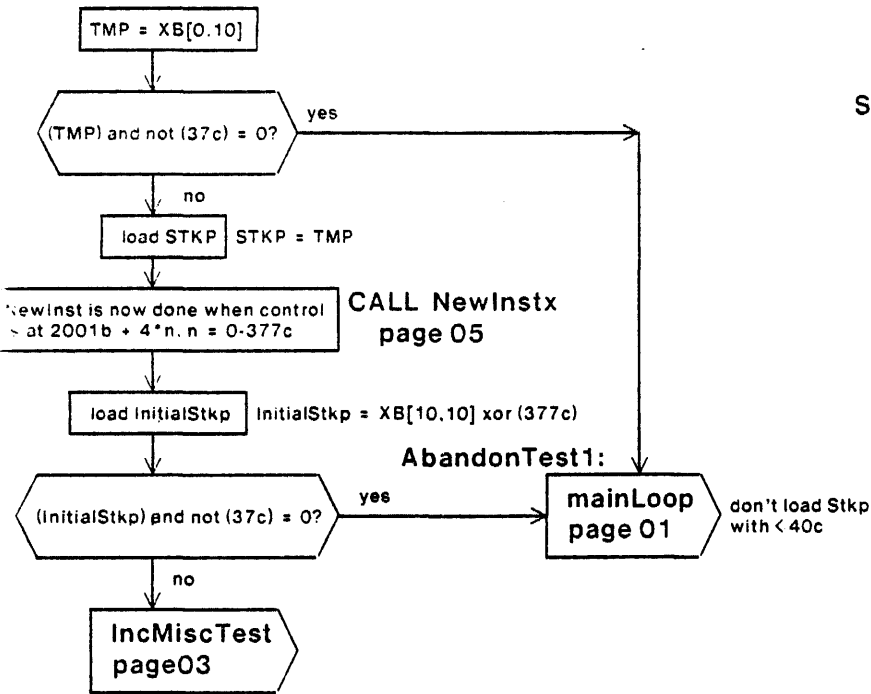


XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDTask	EDTask-02.sil	Camellia Chan	1	03/20/80	02

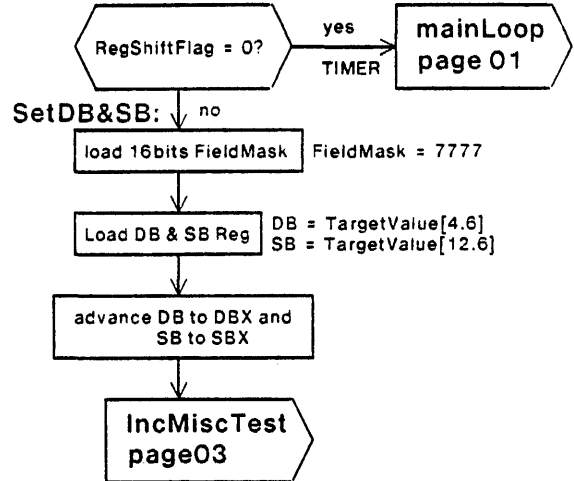


XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostics	EDTask	EDTask-03.sil	Camellia Chan	1	03/20/80	03

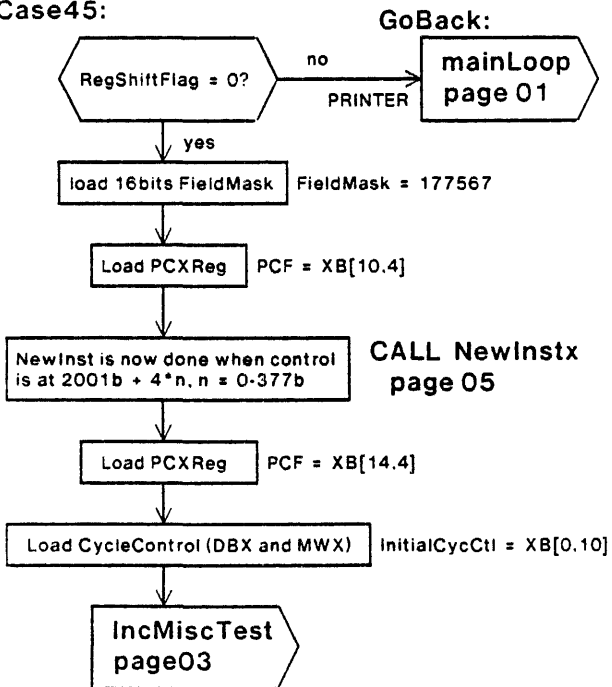
**Case40:**



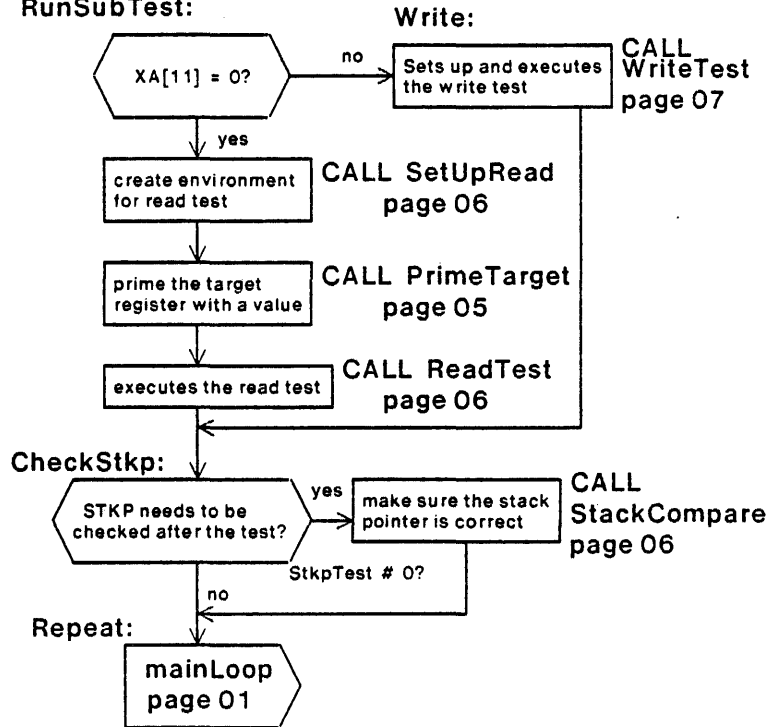
**Case46:**



**Case45:**

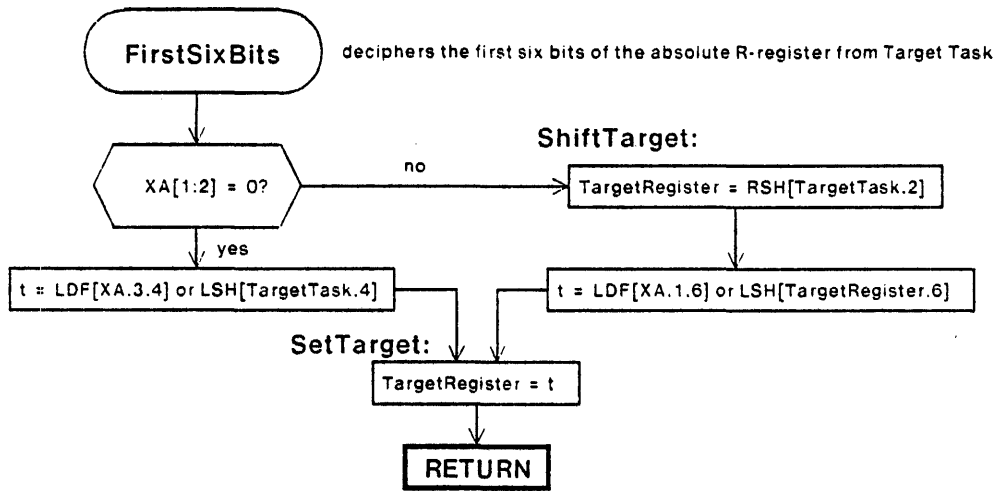


**RunSubTest:**

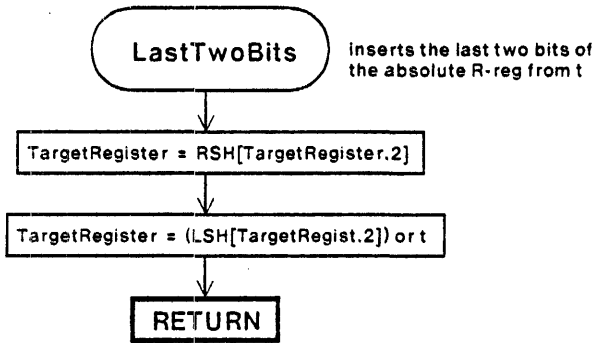


XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDTask	EDTask-04.sil	Camellia Chan	1	03/20/80	04

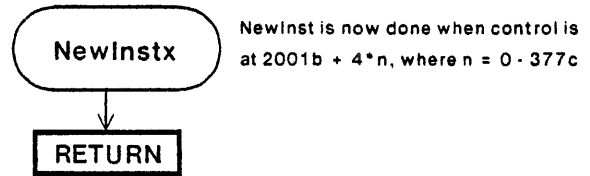
**Subroutine:**



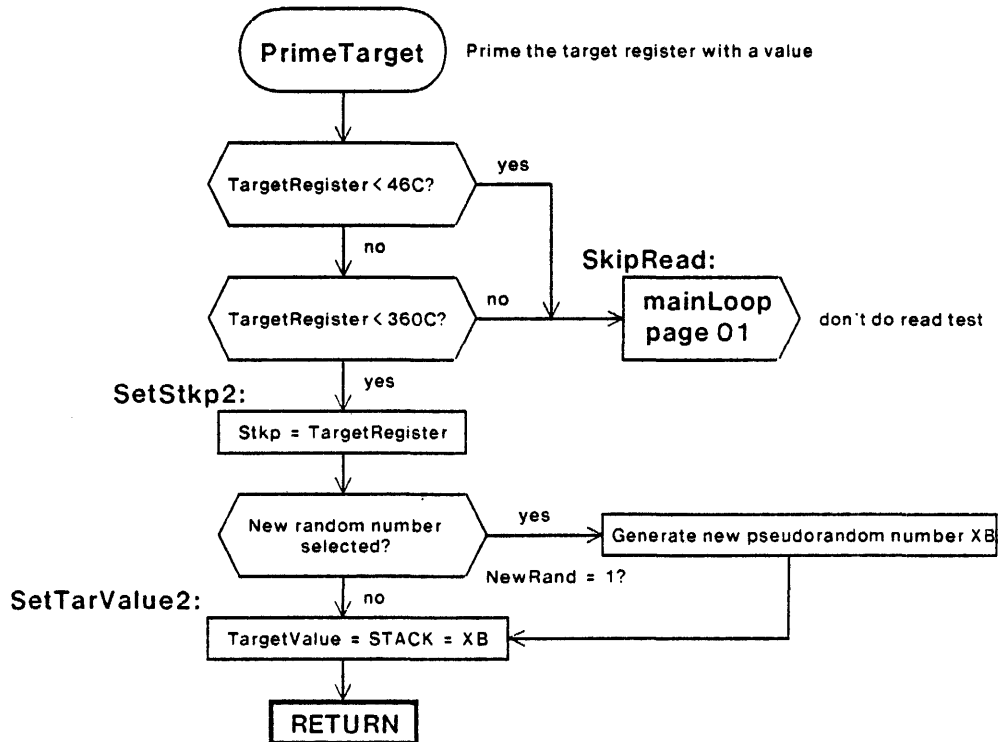
**Subroutine:**



**Subroutine:**

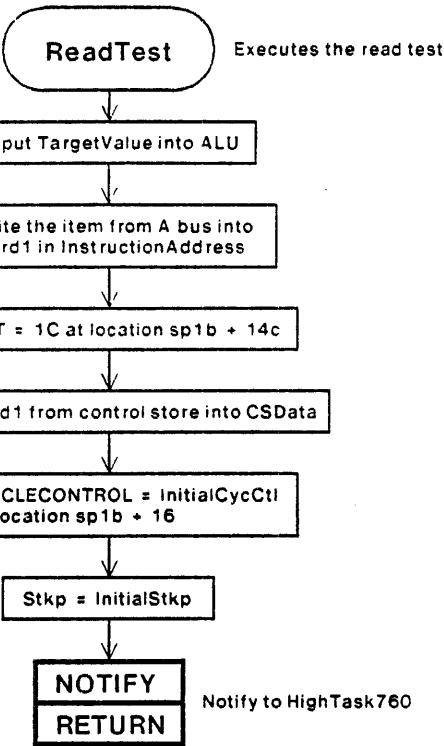


**Subroutine:**

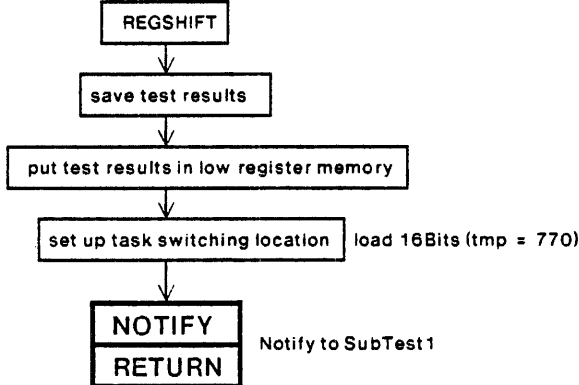


XEROX	D(0)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDTask	EDTask-05.sil	Camellia Chan	1	03/20/80	05

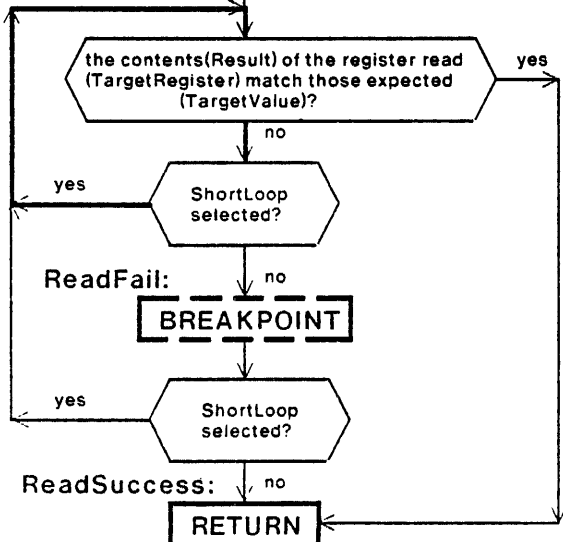
Subroutine:



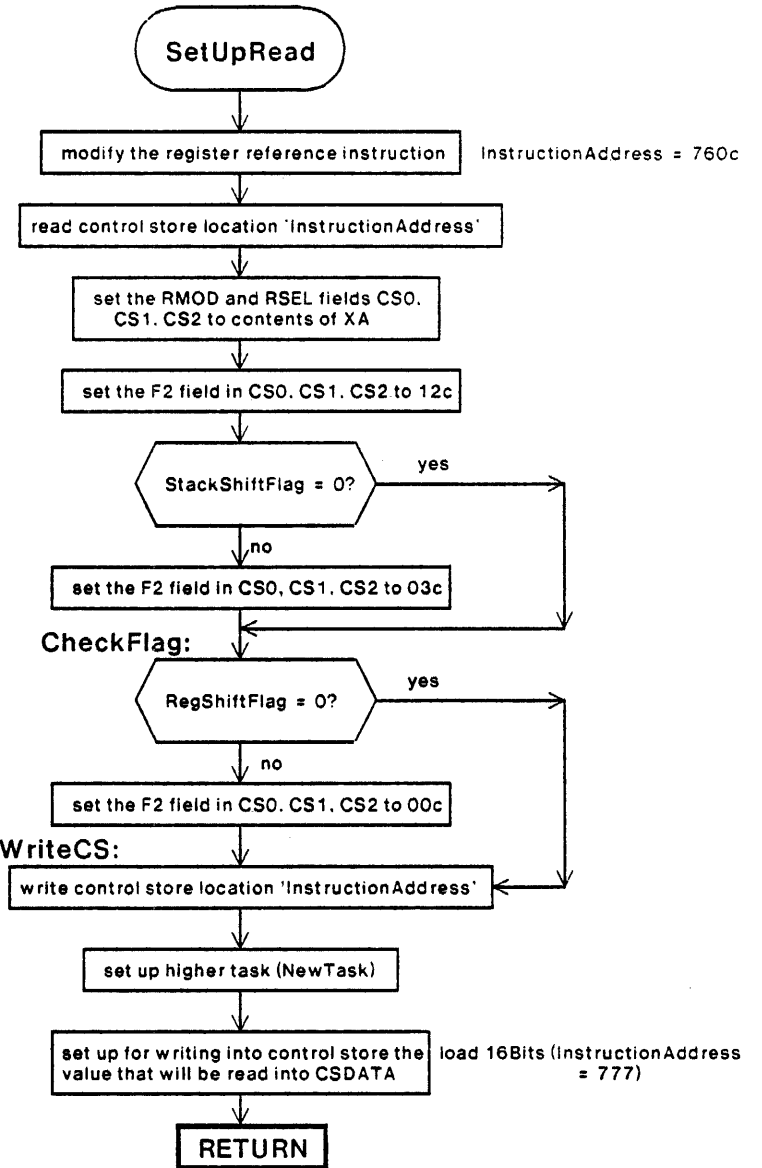
HighTask760:



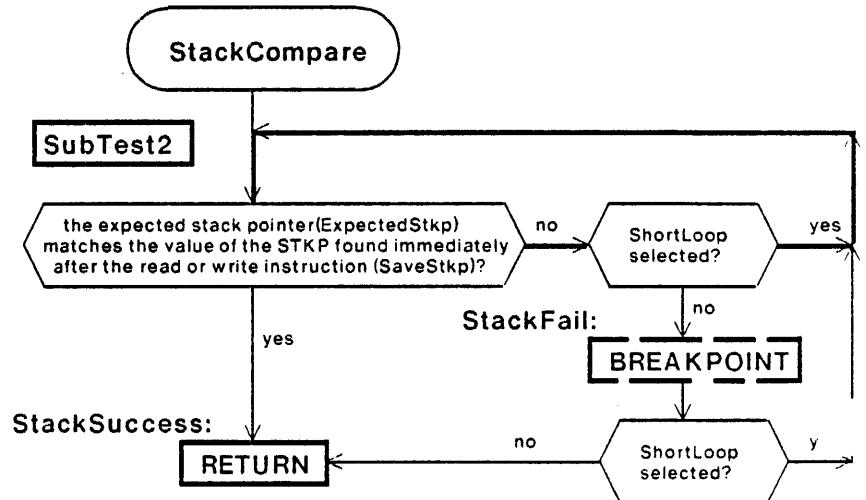
SubTest1



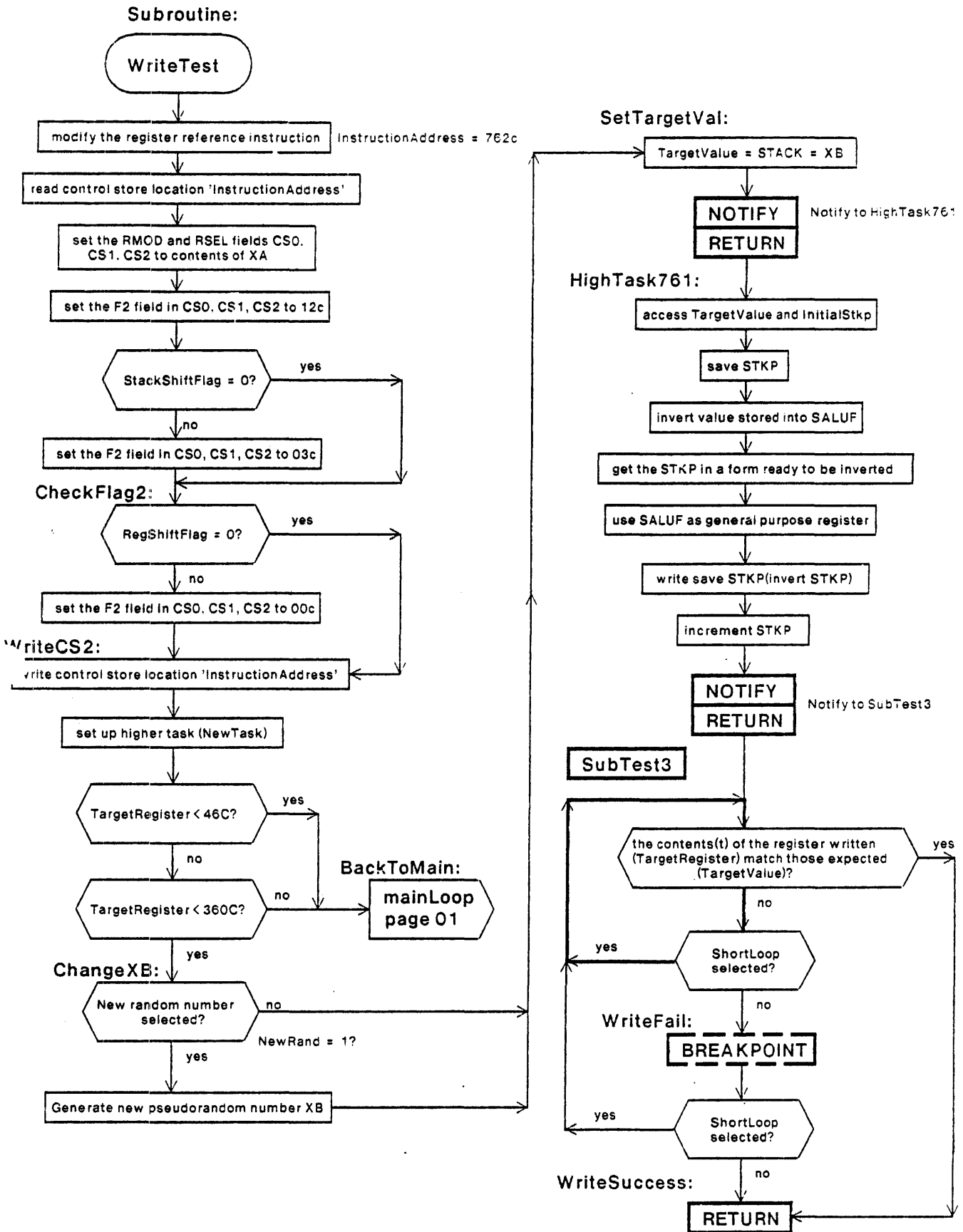
Subroutine:



Subroutine:



XEROX	D(0)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	D(0)	EDTask	EDTask-06.sil	Camellia Chan	1	03/20/80	06



XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diagnostic	EDTask	EDTask-07.sil	Camellia Chan	1	03/20/80	07



PARITY	0	REVISION	1	COMM-ER0	0
CYCLECONTROL	16	RUN-TIME	5	COMM-ER1	0
PCXREG	5	PASSCOUNT	0	COMM-ER2	0
PCFREG	5	MAXPASS	2	BOOT-ERR	0
DBREG	15	SUBTEST	0	*BOOTREASON	40
SBREG	17			MEMSYNDROME	170167
MNBR	47550	DBTEST	0		
*SSTKP	377	MISCTEST	0		
STKP	0	PCFTEST	0		
*ALURESULT	3	RRTEST	0		
SALUF	0	SBTEST	0		
T 20	7000	STKTEST	0		
AATOVA	0				
TPC 20	7777	FIELDMASK	0		
CALLER ILC@+6714		INITIALSTKP	40		
*PAGE	2				
*APC	7011	EXPECTEDSTKP	0	SHORTLOOP	0
*APCTASK	16	RESULT	44	NEWRAND	1
*CIA	60+1	TARGETVALUE	41		
CTASK	0				

Loaded: EDTASK

Time: 09.31

Step at 0:60, BP at 0:60+1

Exit Boot Run-Prog Read-Cmds Break UnBreak ClnAddedBPs ClnAllBPs ShowBPs Go  
 SS **Generate** Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

MicroD 8.6 (OS 16) of April 27, 1979

at 6-Mar-80 15:42:33

microd.run edtask

edtask.DIB 1063b instructions written 6-Mar-80 15:41:03

Total of 1063b instructions

Checking for errors...

Linking...

Building allocation lists...

Assigning locations...

1063b instructions in rings involving ONPAGE or AT

Reloading binaries...

Checking assignment...

Writing .MB file...

Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
edtask.DIB:					
0	1060	16005	107174	2	GO START
1	1276	16320	101173	2	(+1)
2	1275	14001	123170	12	(+2)
3	1274	14323	115167	12	(+3)
4	1273	16002	135164	6	(+4)
5	1272	16320	103162	6	(+5)
6	1271	14001	133161	16	(+6)
7	1270	14323	115156	16	(+7)
10	1267	47	7155	2	(+10)
11	1266	30020	101153	16	(+11)
12	1265	36020	101150	2	(+12)
13	1264	34020	101146	16	(+13)
14	1263	36020	101145	6	(+14)
15	1262	34020	101142	6	(+15)
16	1261	36020	101140	12	(+16)
17	1260	34020	101136	12	(+17)
20	1257	12017	125135	6	(+20)
21	1256	12320	103006	4	(+21)
22	1003	47	5132	2	BIGLOOP
23	1255	31050	165131	16	(+1)
24	1254	33450	25126	2	(+2)
25	1253	50	24200	0	(+3)
26	b 1001	50	25140	0	PASSED-EDTASK-TEST
27	1000	32020	101113	15	MAINLOOP
30	1145	24020	101110	5	(+1)
31	1144	24020	101077	11	(+2)
32	1137	24020	101075	15	(+3)
33	1136	10001	101072	1	(+4)
34	1135	23376	101070	5	(+5)
35	1134	32150	25066	5	(+6)
36	1133	50	24135	0	(+7)
37	1057	31050	125153	10	(+10)
40	1065	50	24007	0	(+11)
41	1002	16150	65150	0	(+12)
42	1064	17174	45147	0	(+13)
43	1063	17174	67144	0	(+14)
44	1062	15150	65143	10	(+15)
45	1061	16050	125134	0	(+16)
46	1056	16163	51064	1	SETTTASK
47	1132	26050	125063	11	(+1)
50	1131	27400	35060	11	(+2)
51	1130	50	24331	0	(+3)
52	1054	50	25000	0	(+4)
53	1055	16230	1037	1	CHOOSETEST
54	1117	50	24075	0	(+1)
55	1036	45	7124	0	(+2)
56	1052	50	25265	1	(+3)
57	1053	37050	125026	1	(+4)
58	1113	50	25025	1	(+5)
51	1037	16171	31034	1	OUTSWITCH

62	1116	50	25700	2	(+1)
63	@1240	50	25032	1	SWITCHTAB10
64	@1241	50	25001	1	(+1)
65	@1242	50	25016	1	(+2)
66	@1243	50	25174	1	(+3)
67	1115	45	7070	0	CASE10
70	1034	50	25265	1	(+1)
71	1035	16161	67030	1	(+2)
72	1114	45	7121	0	(+3)
73	1050	50	25307	1	(+4)
74	1051	16162	65162	0	(+5)
75	1071	10050	125160	10	(+6)
76	1070	10150	31156	10	(+7)
77	1067	35050	125155	14	(+10)
100	1066	50	25025	1	(+11)
101	1100	45	7114	0	CASE11
102	1046	50	25265	1	(+1)
103	1047	16161	65176	0	(+2)
104	1077	45	7111	0	(+3)
105	1044	50	25307	1	(+4)
106	1045	16165	41174	0	(+5)
107	1076	10050	125172	10	(+6)
110	1075	10150	13171	10	(+7)
111	1074	52	25166	0	(-10)
112	1073	37050	125165	4	(+11)
113	1072	50	25025	1	(+12)
114	1107	45	7105	0	CASE12
115	1042	50	25265	1	(+1)
116	1043	16161	65015	1	(+2)
117	1106	45	7100	0	(+3)
120	1040	50	25307	1	(+4)
121	1041	16165	41012	1	(+5)
122	1105	10050	125011	11	(+6)
123	1104	10150	15006	11	(+7)
124	1103	52	25005	1	(+10)
125	1102	35050	125002	5	(+11)
126	1101	50	25025	1	(+12)
127	1176	0	47173	1	CASE13
130	1175	17461	3170	1	(+1)
131	1174	50	24061	0	(+2)
132	1030	45	7176	1	(+3)
133	1177	50	25071	0	(+4)
134	1434	16166	47047	1	(+5)
135	1523	26050	125044	5	(+6)
136	1522	16160	57043	1	(+7)
137	1521	24050	125040	11	(+10)
140	1520	16166	47036	1	(+11)
141	1517	22050	125035	1	(+12)
142	1516	22150	65032	1	(+13)
143	1515	10050	125031	1	(+14)
144	1514	10501	37026	1	(+15)
145	1513	50	24064	0	(+16)
146	1432	45	5072	0	ABANDONTEST
147	1435	50	25000	0	(+1)
150	1433	24150	25025	11	(+2)
151	1512	50	24061	0	(+3)
152	1430	16171	25023	1	STKSWITCHA
153	1511	50	25740	0	(+1)
154	@1460	50	25020	1	SWITCHTAB20
155	@1461	50	25077	0	(+1)
156	@1462	50	25102	0	(+2)
157	@1463	50	25110	0	(+3)
160	1510	20020	101016	11	CASE20
161	1507	50	25015	1	(+1)
162	1437	20000	103074	10	CASE21
163	1436	50	25015	1	(+1)
164	1441	21376	101100	10	CASE22
165	1440	50	25015	1	(+1)
166	1444	0	45107	0	CASE23
167	1443	21476	101104	10	(+1)
170	1442	50	25015	1	(+2)
171	1431	16171	25116	0	STKSWITCHB
172	1447	50	25601	1	(+1)
173	@1500	50	25115	0	SWITCHTAB30
174	@1501	50	25123	0	(+1)
175	@1502	50	25052	1	(+2)

176	@1503	50	25126	0	(+3)
177	1446	20000	105113	10	CASE30
200	1445	50	25015	1	(+1)
201	1451	20000	107121	10	CASE31
202	1450	50	25015	1	(+1)
203	1525	45	5051	1	CASE32
204	1524	50	25000	0	(+1)
205	1453	0	47124	0	CASE33
206	1452	21476	101015	11	(+1)
207	1506	16201	1012	-1	DLTSTACK
210	1505	50	24054	0	(+1)
211	1426	22200	77010	1	(+2)
212	1504	22500	137177	0	(+3)
213	1477	21150	65174	10	(+4)
214	1476	10050	125172	10	(+5)
215	1475	10200	77170	10	(+6)
216	1474	22350	125155	0	(+7)
217	1473	50	25154	0	(+10)
220	1427	10200	77155	0	SUBTRACTSTK
221	1466	10500	137152	0	(+1)
222	1465	10050	125150	10	(+2)
223	1464	11101	101137	10	(+3)
224	1457	20150	65134	10	(+4)
225	1456	11450	125133	10	(+5)
226	1455	10200	77131	10	(+6)
227	1454	10350	125164	0	(+7)
230	1472	24000	103162	14	SETSTKP
231	1471	37050	125160	10	(+1)
232	1470	45	5157	0	(+2)
233	1467	50	25020	1	(+3)
234	1110	50	25025	1	(+4)
235	1031	16160	61166	1	SPECIALREG
236	1173	24050	125165	5	(+1)
237	1172	32150	25162	5	(+2)
240	1171	50	24054	0	(+3)
241	1027	16150	65007	6	(+4)
242	1203	17174	45004	6	(+5)
243	1202	17174	67002	6	(+6)
244	1201	15150	65000	16	(+7)
245	1200	16050	125054	4	(+10)
246	1026	16150	65160	5	SETTARVALUE
247	1170	10050	125157	5	(+1)
250	1167	16224	1155	1	(+2)
251	1166	50	24015	0	(+3)
252	1006	16172	21120	2	BIGSWITCH
253	1250	50	25640	1	(+1)
254	@1120	50	25117	2	SWITCHTAB40
255	@1121	50	25023	2	(+1)
256	@1122	50	25114	1	(+2)
257	@1123	50	25117	1	(+3)
260	@1124	50	25121	1	(+4)
261	@1125	50	25124	1	(+5)
262	@1126	50	25142	1	(+6)
263	@1127	50	25147	1	(+7)
264	1247	16165	67115	6	CASE40
265	1246	10050	125113	12	(+1)
266	1245	10501	37110	12	(+2)
267	1244	50	24020	0	(+3)
270	1010	45	5010	2	ABANDONTEST1
271	1204	50	25000	0	(+1)
272	1011	10150	3076	12	(+2)
273	1237	45	11024	0	(+3)
274	1012	50	25202	0	(+4)
275	1013	16166	47075	6	(+5)
276	1236	10050	125073	2	(+6)
277	1235	10417	137071	2	(+7)
300	1234	10501	37067	2	(+10)
301	1233	50	24051	0	(+11)
302	1024	50	25021	0	(+12)
303	1211	22017	137020	6	CASE41
304	1210	10150	65016	6	(+1)
305	1207	676	41015	2	(+2)
306	1206	50	23013	2	(+3)
307	1205	50	25053	0	(+4)
310	1146	50	25000	0	CASE42
311	1147	50	25000	0	CASE43

312	1150	50	25000	0	CASE44
313	1152	24150	25122	5	CASE45
314	1151	50	24064	0	(+1)
315	1032	22007	117136	5	(+2)
316	1157	22337	137134	5	(+3)
317	1156	16163	53133	5	(+4)
320	1155	10050	125131	11	(+5)
321	1154	10150	31126	11	(+6)
322	1153	45	11044	0	(+7)
323	1022	50	25202	0	(+10)
324	1023	16163	63034	6	(+11)
325	1216	10050	125033	12	(+12)
326	1215	10150	31030	12	(+13)
327	1214	16165	67027	6	(+14)
330	1213	22050	125025	12	(+15)
331	1212	50	25053	0	(+16)
332	1033	50	25000	0	GOBACK
333	1161	24150	25141	5	CASE46
334	1160	50	24040	0	(+1)
335	1020	50	25000	0	(+2)
336	1021	22017	137057	6	SETDB&SB
337	1227	22320	137055	6	(+1)
340	1226	10164	65053	6	(+2)
341	1225	10050	125050	12	(+3)
342	1224	10150	15047	12	(+4)
343	1223	10165	41045	6	(+5)
344	1222	10050	125042	12	(+6)
345	1221	10150	13041	12	(+7)
346	1220	52	25036	2	(+10)
347	1217	50	25053	0	(+11)
350	1163	24150	25144	5	CASE47
351	1162	50	24034	0	(+1)
352	1016	50	25000	0	(+2)
353	1017	10150	27061	6	SETMNR
354	1230	50	25053	0	(+1)
355	1007	16171	25153	1	SMALLSWITCH
356	1165	50	25700	1	(+1)
357	@1140	50	25150	1	SWITCHTAB50
360	@1141	50	25122	2	(+1)
361	@1142	50	25062	2	(+2)
362	@1143	50	25124	2	(+3)
363	1164	50	25000	0	CASE50
364	1251	50	25000	0	CASE51
365	1231	50	25053	0	CASE52
366	1252	50	25000	0	CASE53
367	1025	35050	125064	12	INCMISCTEST
370	1232	45	3030	0	(+1)
371	1014	50	25362	1	(+2)
372	1015	45	3011	0	(+3)
373	1004	50	25207	0	(+4)
374	1005	50	25000	0	(+5)
375	1112	45	7022	1	RUNSUBTEST
376	1111	50	25054	1	(+1)
377	1526	16201	1063	1	(+2)
400	1531	50	24020	0	(+3)
401	1410	45	3050	0	(+4)
402	1424	50	25362	1	(+5)
403	1425	45	11044	0	(+6)
404	1422	50	25201	0	(+7)
405	1423	45	3041	0	(+10)
406	1420	50	25207	0	(+11)
407	1421	50	25027	0	(+12)
410	1411	45	11024	0	WRITE
411	1412	50	25311	0	(+1)
412	1413	24150	25060	15	CHECKSTKP
413	1530	50	24034	0	(+1)
414	1417	45	7031	0	(+2)
415	1414	50	25315	1	(+3)
416	1415	50	25035	0	(+4)
417	1416	45	5057	1	REPEAT
420	1527	50	25000	0	(+1)
421	1532	16226	1104	1	FIRSTSIXBITS
422	1542	50	24015	0	(+1)
423	1406	26174	51102	11	(+2)
424	1541	16363	41100	1	(+3)
425	1540	50	25076	1	(+4)

426	1407	26170	45073	11	SHIFTTARGET
427	1535	26050	125071	5	(+1)
430	1534	26174	55067	5	(+2)
431	1533	16364	57076	1	(+3)
432	1537	26050	125074	5	SETTARGET
433	1536	50	25401	0	(+1)
434	1543	26170	105112	5	LASTTWOBITS
435	1545	26374	105111	5	(+1)
436	1544	50	25401	0	(+2)
437	@2001	50	25401	0	NEWINSTX
440	2000	27402	15106	4	PRIMETARGET
441	2043	50	24265	0	(+1)
442	2032	17	41105	0	(+2)
443	2042	27450	25102	4	(+3)
444	2041	50	24235	0	(+4)
445	2016	50	25066	0	(+5)
446	2033	45	5042	0	SKIPREAD
447	2021	50	25000	0	(+1)
450	2017	26150	3101	4	SETSTKP2
451	2040	32150	25076	4	(+1)
452	2037	50	24031	0	(+2)
453	2015	16150	65063	4	(+3)
454	2031	17174	45056	4	(+4)
455	2027	17174	67053	4	(+5)
456	2025	15150	65046	14	(+6)
457	2023	16050	125031	4	(+7)
460	2014	16150	65074	4	SETTARVALUE2
461	2036	10050	125073	4	(+1)
462	2035	40050	125071	14	(+2)
463	2034	50	25401	0	(+3)
464	403	10150	25075	4	READTEST
465	436	22147	21036	14	(+1)
466	417	47	33430	0	(+2)
467	@ 414	0	43033	0	(+3)
470	415	22147	21027	14	(+4)
471	413	47	35435	0	(+5)
472	@ 416	22150	11023	10	(+6)
473	411	10150	3016	0	(+7)
474	407	24147	21012	0	(+10)
475	405	50	25401	0	(+11)
476	@ 760	20150	41122	14	HIGHTASK760
477	451	12050	125121	0	(+1)
500	450	70337	77117	14	(+2)
501	447	10002	107115	10	(+3)
502	446	10150	3113	10	(+4)
503	445	40676	101111	14	(+5)
504	444	12150	65107	0	(+6)
505	443	42050	125105	14	(+7)
506	442	10017	121102	10	(+10)
507	441	10320	103100	10	(+11)
510	440	10147	21077	10	(+12)
511	437	50	25401	0	(+13)
512	@ 770	32000	103136	14	SUBTEST1
513	457	22150	65135	4	(+1)
514	456	10250	125132	4	(+2)
515	455	12250	165130	0	(+3)
516	454	11450	25126	4	(+4)
517	453	50	24070	0	(+5)
520	435	32150	124466	10	(+6)
521	433	50	25161	3	(+7)
522	b 432	50	25125	0	READFAIL
523	452	32150	124463	10	(+1)
524	431	50	25161	3	(+2)
525	430	50	25071	0	(+3)
526	434	50	25401	0	READSUCCESS
527	@ 777	50	25163	1	(+1)
530	571	22017	101161	15	SETUPREAD
531	570	22320	103157	15	(+1)
532	567	22150	65155	15	(+2)
533	566	20050	125152	5	(+3)
534	565	20	41150	1	(+4)
535	564	20147	21147	5	(+5)
536	563	47	35405	0	(+6)
537	@ 402	54150	65145	15	(+7)
540	562	36050	125143	15	(+10)
541	561	0	43141	1	(+11)

542	560	20147	21136	5	(+12)
543	557	47	35411	0	(+13)
544	@ 404	54150	65134	15	(+14)
545	556	20050	125133	1	(+15)
546	555	0	47131	1	(+16)
547	554	20147	21127	5	(+17)
550	553	47	35414	0	(+20)
551	@ 406	54150	65125	15	(+21)
552	552	20050	125123	5	(+22)
553	551	20162	133121	5	(+23)
554	550	16163	65116	1	(+24)
555	547	26000	113114	1	(+25)
556	546	27350	125113	1	(+26)
557	545	26301	101110	1	(+27)
560	544	26150	11107	1	(+30)
561	543	26050	125105	1	(+31)
562	542	26151	65103	1	(+32)
563	541	36353	125101	15	(+33)
564	540	36423	101077	15	(+34)
565	537	16161	53075	1	(+35)
566	536	26000	105072	1	(+36)
567	535	27350	125071	1	(+37)
570	534	26314	101067	1	(+40)
571	533	26150	11065	1	(+41)
572	532	26050	125062	1	(+42)
573	531	26151	65061	1	(+43)
574	530	20353	125056	5	(+44)
575	527	26002	107056	1	(+45)
576	526	26150	11053	1	(+46)
577	525	26000	125051	1	(+47)
600	524	26151	65046	1	(+50)
601	523	20353	125045	1	(+51)
602	522	24150	25043	11	(+52)
603	521	50	24054	0	(+53)
604	427	26002	107147	0	(+54)
605	463	26150	11144	0	(+55)
606	462	26000	107143	0	(+56)
607	461	26151	65140	0	(+57)
610	460	20353	125055	0	(+60)
611	426	24150	25040	5	CHECKFLAG
612	520	50	24051	0	(+1)
613	425	26002	107156	0	(+2)
614	467	26150	11155	0	(+3)
615	466	26020	101153	0	(+4)
616	465	26151	65151	0	(+5)
617	464	20353	125050	0	(+6)
620	424	36150	65037	15	WRITECS
621	517	10050	125035	11	(+1)
622	516	20150	65033	1	(+2)
623	515	10450	165030	11	(+3)
624	514	20463	63026	5	(+4)
625	513	10465	167025	11	(+5)
626	512	10463	163022	11	(+6)
627	511	10461	171020	11	(+7)
630	510	10760	175016	11	(+10)
631	507	10160	77015	11	(+11)
632	506	20450	125012	1	(+12)
633	505	20150	65010	5	(+13)
634	504	36150	25006	15	(+14)
635	503	22147	21004	15	(+15)
636	502	47	31420	0	(+16)
637	@ 410	20150	25002	1	(+17)
640	501	22147	21001	15	(+20)
641	500	47	33424	0	(+21)
642	@ 412	26174	71176	10	(+22)
643	477	24050	125174	0	(+23)
644	476	10017	101172	10	(+24)
645	475	10320	103171	10	(+25)
646	474	10150	65167	10	(+26)
647	473	24350	125165	0	(+27)
650	472	22017	137163	14	(+30)
651	471	22320	103160	14	(+31)
652	470	50	25401	0	(+32)
653	1546	32000	105125	15	STACKCOMPARE
654	1552	10217	77123	15	(+1)
655	1551	23450	25121	1	(+2)

656	1550	50	24010	0	(+3)
657	1405	32150	124406	10	(+4)
660	1403	50	25114	1	(+5)
661	b 1402	50	25117	1	STACKFAIL
662	1547	32150	124403	10	(+1)
663	1401	50	25114	1	(+2)
664	1400	50	25011	0	(+3)
665	1404	50	25401	0	STACKSUCCESS
666	2044	22017	105156	15	WRITETEST
667	2167	22320	103154	15	(+1)
670	2166	22150	65153	15	(+2)
671	2165	20050	125151	5	(+3)
672	2164	20	41147	1	(+4)
673	2163	20147	21144	5	(+5)
674	2162	47	35441	0	(+6)
675	@2020	54150	65143	15	(+7)
676	2161	36050	125140	15	(+10)
677	2160	0	43136	1	(+11)
700	2157	20147	21135	5	(+12)
701	2156	47	35444	0	(+13)
702	@2022	54150	65132	15	(+14)
703	2155	20050	125130	1	(+15)
704	2154	0	47126	1	(+16)
705	2153	20147	21124	5	(+17)
706	2152	47	35450	0	(+20)
707	@2024	54150	65123	15	(+21)
710	2151	20050	125120	5	(+22)
711	2150	20162	133117	5	(+23)
712	2147	16163	65115	1	(+24)
713	2146	26000	113112	1	(+25)
714	2145	27350	125110	1	(+26)
715	2144	26301	101107	1	(+27)
716	2143	26150	11104	1	(+30)
717	2142	26050	125103	1	(+31)
720	2141	26151	65100	1	(+32)
721	2140	36353	125077	15	(+33)
722	2137	36423	101074	15	(+34)
723	2136	16161	53073	1	(+35)
724	2135	26000	105071	1	(+36)
725	2134	27350	125066	1	(+37)
726	2133	26314	101064	1	(+40)
727	2132	26150	11063	1	(+41)
730	2131	26050	125061	1	(+42)
731	2130	26151	65057	1	(+43)
732	2127	20353	125055	5	(+44)
733	2126	26002	107053	1	(+45)
734	2125	26150	11050	1	(+46)
735	2124	26000	125046	1	(+47)
736	2123	26151	65045	1	(+50)
737	2122	20353	125043	1	(+51)
740	2121	24150	25040	11	(+52)
741	2120	50	24025	0	(+53)
742	2013	26002	107121	0	(+54)
743	2050	26150	11117	0	(+55)
744	2047	26000	107115	0	(+56)
745	2046	26151	65113	0	(+57)
746	2045	20353	125024	0	(+60)
747	2012	24150	25037	5	CHECKFLAG2
750	2117	50	24020	0	(+1)
751	2011	26002	107130	0	(+2)
752	2054	26150	11127	0	(+3)
753	2053	26020	101124	0	(+4)
754	2052	26151	65123	0	(+5)
755	2051	20353	125021	0	(+6)
756	2010	36150	65034	15	WRITECS2
757	2116	10050	125033	11	(+1)
760	2115	20150	65030	1	(+2)
761	2114	10450	165027	11	(+3)
762	2113	20463	63025	5	(+4)
763	2112	10465	167023	11	(+5)
764	2111	10463	153021	11	(+6)
765	2110	10461	171016	11	(+7)
766	2107	10760	175015	11	(+10)
767	2106	10160	77013	11	(+11)
770	2105	20450	125011	1	(+12)
771	2104	20150	65007	5	(+13)



772	2103	36150	25005	15	(+14)
773	2102	22147	21002	15	(+15)
774	2101	47	31454	0	(+16)
775	@2026	20150	25001	1	(+17)
776	2100	22147	21176	14	(+20)
777	2077	47	33460	0	(+21)
1000	@2030	26174	71175	10	(+22)
1001	2076	24050	125172	0	(+23)
1002	2075	10017	103170	10	(+24)
1003	2074	10320	103166	10	(+25)
1004	2073	10150	65164	10	(+26)
1005	2072	24350	125163	0	(+27)
1006	2071	27402	15160	4	(+30)
1007	2070	50	24214	0	(+31)
1010	2006	17	41156	0	(+32)
1011	2067	27450	25154	4	(+33)
1012	2066	50	24211	0	(+34)
1013	2004	50	25017	0	(+35)
1014	2007	45	5132	0	BACKTOMAIN
1015	2055	50	25000	0	(+1)
1016	2005	32150	25153	4	CHANGEXB
1017	2065	50	24004	0	(+1)
1020	2003	16150	65143	4	(+2)
1021	2061	17174	45140	4	(+3)
1022	2060	17174	67137	4	(+4)
1023	2057	15150	65135	14	(+5)
1024	2056	16050	125004	4	(+6)
1025	2002	16150	65151	4	SETTARGETVAL
1026	2064	10050	125147	4	(+1)
1027	2063	24147	21145	0	(+2)
1030	2062	50	25401	0	(+3)
1031	@ 761	10002	103012	12	HIGHTASK761
1032	605	10150	3010	12	(+1)
1033	604	44150	65007	16	(+2)
1034	603	40150	3144	17	(+3)
1035	@ 762	20050	101004	16	(+4)
1036	602	2	47002	2	(+5)
1037	601	676	41001	2	(+6)
1040	600	50	23177	1	(+7)
1041	577	70337	77175	15	(+10)
1042	576	72150	3172	15	(+11)
1043	575	40676	101170	15	(+12)
1044	574	40150	7166	15	(+13)
1045	573	40147	21164	15	(+14)
1046	572	50	25401	0	(+15)
1047	@ 772	32000	107025	16	SUBTEST3
1050	612	26150	3023	6	(+1)
1051	611	40150	65021	16	(+2)
1052	610	11450	25016	6	(+3)
1053	607	50	24045	0	(+4)
1054	423	32150	124442	10	(+5)
1055	421	50	25164	3	(+6)
1056	b 420	50	25015	2	WRITEFAIL
1057	606	32150	124403	10	(+1)
1060	401	50	25164	3	(+2)
1061	400	50	25044	0	(+3)
1062	422	50	25401	0	WRITESUCCESS

Page 400: 221 locations used, 157 free  
Page 1000: 277 locations used, 101 free  
Page 1400: 153 locations used, 225 free  
Page 2000: 170 locations used, 210 free

RM:

0	1	REVISION
1	5	RUN-TIME
2	0	INNERLOOPCOUNTER
3		PASSCOUNT
4	2	MAXPASS
5	1	NEWRAND
6	0	SHORTLOOP
7		SUBTEST
11		DBTEST
12		MISCTEST
13		PCFTTEST

14		RRTEST
15		SBTEST
16		STKTEST
17		CS0
20		CS1
21		CS2
22		DELTASTACK
23		DUMMYREGISTER
24		EXPECTEDSTKP
25		FIELDMASK
26		INITIALCYCTL
27		INSTRUCTIONADDRESS
30		NEWTASK
31		REGSHIFTFLAG
32		STACKSHIFTFLAG
33		STKPTST
34		STUFFTMP
35		TARGETREGISTER
36		TARGETTASK
40	40	INITIALSTKP
41	41	TARGETVALUE
42	42	TMP
43	43	SAVESTKP
44	44	RESULT
45	45	WRITETESTREENTRYLOC
52		CA
53		CB
54		XA
55		RLC0 XB

Time: 13 seconds; 0 error(s), 0 warning(s), 11403 words free

```

.....
:::EDTaskLog.MIDAS : Logger for EDTask program
:::                      By: C. Chan                      Mar. 20. 1980
.....

```

```

.start      L X AppendOutput EDTask.report:
            L X WriteMessage ~***** START EDTask Test : :
            L X WriteDT:
            L X WriteMessage *****~ ;
            L X Skip .continue:

.breakpoint L X AppendOutput EDTask.report:
            L A18 SkipNE READFAIL:
            L X Skip .ReadFail:
            L A18 SkipNE STACKFAIL:
            L X Skip .StackFail:
            L A18 SkipNE WRITEFAIL:
            L X Skip .WriteFail:
            L A18 SkipNE PASSED-EDTASK-TEST:
            L X Skip .passtest:

.notmybreak L X AppendOutput EDTask.report:
            L X WriteMessage *** FAILED: Not at my breakpoint ~:

            L X WriteMessage ' Parity = ;
            R A0 Val;
            L X WriteMessage:
            L X WriteMessage ~:

            L X WriteMessage ' CIA = ;
            R A18 Val;
            L X WriteMessage:
            L X WriteMessage ~:

            L X WriteMessage ' CTASK = ;
            R A19 Val;
            L X WriteMessage:
            L X WriteMessage ~:

            L X WriteMessage ' APCTASK = ;
            R A17 Val;
            L X WriteMessage:
            L X WriteMessage ~:

            L X WriteMessage ' APC = ;
            R A16 Val;
            L X WriteMessage:
            L X WriteMessage ~:

            L X WriteMessage ' TPC = ;
            R A13 Val;
            L X WriteMessage:
            L X WriteMessage ~:

            L X CloseOutput:
            L X Exit:

.ReadFail  L X WriteMessage *** FAILED: at my Breakpoint ~:
            L X WriteMessage *      Result of the register read does not match the TargetValue ~:
            L X WriteMessage ' Result = ;
            R B17 Val;
            L X WriteMessage:
            L X WriteMessage ~:
            L X WriteMessage ' TargetValue = ;
            R B18 Val;
            L X WriteMessage:
            L X WriteMessage ~:

.bad       L X WriteMessage ' SUBTEST = ;
            R B4 Val;
            L X WriteMessage:
            L X WriteMessage ~:

            L X WriteMessage ' PASSCOUNT = ;

```

```
R B2 Val:
L X WriteMessage:
L X WriteMessage ~:

L X Skip .continue:

.StackFail L X WriteMessage *** FAILED: at my Breakpoint ~:
L X WriteMessage * ExpectedStkp does not match the value of the STKP found ~ :
L X WriteMessage * immediately after the read or write instruction (SaveStkp) ~:
L X BackSkip .bad:

.writeFail L X WriteMessage *** FAILED: at my Breakpoint ~:
L X WriteMessage * The contents (t) of the register written (TargetRegister) ~ :
L X WriteMessage * do not match the TargetValue ~:
L X BackSkip .bad:

.passtest L X WriteMessage ----- PASSEd EDTask Test : :
L X WriteDT:
L X WriteMessage ----- ;
L X Skip .continue:

.continue L X WriteMessage ~:
L X CloseOutput;
L X DisplayOn;
L X Confirm;
L X TimeOut 10000000;
L X Continue;
L X Skip 2;
L X ShowError Program failed to CONTINUE.;
L X BackSkip .notmybreak;
L X DisplayOff;
L X BackSkip .breakpoint;
```

L A19 Val 0  
L X Confirm  
L X Load EDTASK:  
L B0 Addr REVISION:  
L B1 Addr RUN-TIME:  
L B2 Addr PASSCOUNT:  
L B3 Addr MAXPASS:  
L B4 Addr SUBTEST:  
L B6 Addr DBTEST:  
L B7 Addr MISCTEST:  
L B8 Addr PCFTTEST:  
L B9 Addr RRTEST:  
L B10 Addr SBTEST:  
L B11 Addr STKTEST:  
L B13 Addr FIELDMASK:  
L B14 Addr INITIALSTKP:  
L B16 Addr EXPECTEDSTKP:  
L B17 Addr RESULT:  
L B18 Addr TARGETVALUE:  
L C16 Addr SHORTLOOP:  
L C17 Addr NEWRAND:  
L X DisplayOn:  
L X TimeOut 10000  
L X SS GO  
L X Skip 1  
L X ShowError Single-step at GO hung

%

```
*****
*** EDtimex.mc : Timer Exerciser microcode          *** Revision 1 ***
*** Purpose : This test exercises timers 0 - 15d.
               It replaces the page 16d portion of standard KERNEL.
*** Minimum Hardware : Standard 4 CPU boards.
*** Approximate Run Time : 6d seconds.
*** Written by : C. Thacker, Feb. 25, 1979
*** Modified by : K. Mayekawa, Feb. 28, 1980
*** Standardize title page and code format.
*****
```

```
*****
*SubTest Description:
```

- ```
* SubTest 0: Timer Initalization (Task 14)
    Clears the timers and sets up the refresh timer.

* SubTest 1: Timer Wakeup routine (Task 14)
    Timer wakeups come here.
    If the timer 15d expires, memory is refreshed and a new value
    is added to the timer, then Mouse halt is checked.
    If any of the timers 0-14d expires, it is restarted and timeout
    counter for that timer is set to TimeOut.

* SubTest 2: Check Timeout routine (Task 0)
    Loads the timers 0-14d, sets a timeout value for these timers
    to TimeOut, then checks for timeouts.
```

```
*****
*BreakPoints:
```

- ```
* Fail : The timer was not restarted (timeout register was not set
        to TimeOut) when it expired.
* Passed-EDtimex-Test: Passed all tests, and all passes.
* WrongSlotExpired : Wrong timer slot expired. Timer slots should expire from
        14d to 0 sequentially.
```

```
*Note: The program also breaks at MouseHalt.
```

```
*****
*BreakPoint Logic Analyzer Sync Points:
```

- ```
* There is no short looping capability for this exerciser.
```

```
*****
*Loading and Reading Timers:
```

- ```
* Timers are loaded from the ALUA bus by functions LOADTIMER and ADDTOTIMER.
* Also they can be read as an external R source(TIMER).
* When loading or adding to timers, bits are divided as follows:
* Bits 0 - 3 : new state (loaded by both LOADTIMER and ADDTOTIMER)
* Bits 4 - 11 : new data to be loaded (by LOADTIMER) or new data to be added
                to the current data (by ADDTOTIMER)
* Bits 12 - 15 : slot number to be loaded
```

\*\*\*\*\*  
 \*Special Reg. Definition:

- \* RM[20b] - RM[36b]: contains timeout values for each slot.  
 set to TimeOut(=40b) when timers are started initially.

Each time through the BigLoop, these timeouts are decremented.  
 If one of the timeouts becomes zero, a breakpoint occurs.  
 The maintenance panel is also incremented, so if you kill the  
 breakpoint at FAIL, you will see if any errors occur by noting  
 a nonzero panel.

Whenever a timer expires normally, it is restarted and its associated  
 timeout register is set to TimeOut. Thus, we should never see a zero  
 value in any of the timeout registers.

- \* InnerCount: incremented each time through the BigLoop, i.e. each time  
 TimeOuts for all slots 0 through 14d are decremented by one.
- \* MaxInnerCount: contains the maximum value for InnerCount.  
 When InnerCount=MaxInnerCount, the program completes one  
 pass, resets InnerCount, and goes back to the start of the  
 program if PassCount<MaxPass.

\*\*\*\*\*  
 \*Subroutine Description:

- \* StartTimer: loads the timer whose slot is in Tslot0 and sets RM[Tslot0 + 20b]  
 to TimeOut(=40b).  
 It sets the state of the timer to 5 (Simple Timer).  
 Value loaded is (112d - (slot# x 4)), loading timer 14d  
 with the smallest value and timer 0 with the largest value.  
 Thus the timers should expire from slot 14d through 0 sequentially.

Slot number (decimal)	State loaded	Value loaded (decimal)	TimeOut stored at
14	5	56	RM[36b]
13	5	60	RM[35b]
12	5	64	RM[34b]
11	5	68	RM[33b]
10	5	72	RM[32b]
9	5	76	RM[31b]
8	5	80	RM[30b]
7	5	84	RM[27b]
6	5	88	RM[26b]
5	5	92	RM[25b]
4	5	96	RM[24b]
3	5	100	RM[23b]
2	5	104	RM[22b]
1	5	108	RM[21b]
0	5	112	RM[20b]

\*\*\*\*\*  
 %

```

*****
*INITIALIZATION:

BUILTIN[INSERT,24];
INSERT[d0lang];
NOMIDASINIT;
TITLE[Timer Exerciser];

Set[TTask,16];
Set[TimerPage,2];

Set[TimerInitLoc,add[lshift[TimerPage,10],20]];
MC[TimerInitLoc,add@[TimerInitLoc,377]];
MC[TimerInitLocH,add[160000,add@[TimerInitLoc,177400]]]; *Notify to task 16, location TimerInitLoc
Set[TimerTable,add[lshift[TimerPage,10],100]];

MC[TimerValue,53400]; *State 5, Value 112d
MC[TimeOut,40];

***** R-Registers: *****
* Task 0 Registers
SETTASK[0];

RV[SubTest, 60];
RV[Rlink0, 61]; *subroutine return link
RV[Revision, 62, 1]; *Revision 1
RV[Run-Time, 63, 6]; *Run-Time is 6d seconds
RV[PassCount, 64, 0];
RV[MaxPass, 65, 5000];
RV[InnerCount, 66];
RV[MaxInnerCount, 67, 100];
RV[Tslot0, 70]; *slot number for timers
RV[Temp, 71];
RV[Temp1, 72];

* Task 14 Registers
SETTASK[TTask];

RV[Tslot,40];
RV[ExpectedSlot,41]; *slot number which should expire next
RV[TimerTemp,42]; *temporary register
RV[RTimer,43]; *constant for memory refresh timer
RV[REFR,44]; *refresh address

```



```
*****
*** MAIN routine:

*SubTest 0 (Timer Initialization)

    SETTASK[TTask];
    ONPAGE[TimerPage];

InitTimers: TimerTemp ← (100000C), AT[TimerInitLoc];

ClrTimers:  LOADTIMER[TimerTemp];           *clear out all Timers
            NOP;                             *Timers can be loaded only once
            NOP;                             *every 7 microinstructions
            NOP;
            TimerTemp ← (TimerTemp) + 1, ResetMemErrs; *clear any pending memory errors
            LU ← (TimerTemp) AND (17C);        *there are 16d timers
            REFR ← (0C), DBLGOTO[InitDone, ClrTimers, ALU=0];

InitDone:  LU ← TIMER;                       *clear all wakeups
            RTimer ← (50000C);                *set up the Refresh timer
            RTimer ← (RTimer) OR (257C);      *simple timer, value 10d, slot 15d
            LOADTIMER[RTimer];
            ExpectedSlot ← 16c;               *insist that the timers expire in order
            CALL[TimerRet];                  *returns to task 0
```

## \*SubTest 1 (Timer Wakeup routine)

TimerWakeup: DISPATCH[TIMER,14,4];  
DISP[Timers];

\*Timer wakeups come here  
\*initialize base register

\*Timer dispatch table for task 14 (Timers are read in complemented form.)

Timers: REFRESH[REFR], GOTO[RefreshNext], AT[TimerTable,00]; \*slot 15d(used as a refresh timer)  
Tslot ← T + 16c, GOTO[RestartTimer], AT[TimerTable,01]; \*slot 14d  
Tslot ← T + 15c, GOTO[RestartTimer], AT[TimerTable,02]; \*slot 13d  
Tslot ← T + 14c, GOTO[RestartTimer], AT[TimerTable,03]; \*slot 12d  
Tslot ← T + 13c, GOTO[RestartTimer], AT[TimerTable,04]; \*slot 11d  
Tslot ← T + 12c, GOTO[RestartTimer], AT[TimerTable,05]; \*slot 10d  
Tslot ← T + 11c, GOTO[RestartTimer], AT[TimerTable,06]; \*slot 9d  
Tslot ← T + 10c, GOTO[RestartTimer], AT[TimerTable,07]; \*slot 8d  
Tslot ← T + 7c, GOTO[RestartTimer], AT[TimerTable,10]; \*slot 7d  
Tslot ← T + 6c, GOTO[RestartTimer], AT[TimerTable,11]; \*slot 6d  
Tslot ← T + 5c, GOTO[RestartTimer], AT[TimerTable,12]; \*slot 5d  
Tslot ← T + 4c, GOTO[RestartTimer], AT[TimerTable,13]; \*slot 4d  
Tslot ← T + 3c, GOTO[RestartTimer], AT[TimerTable,14]; \*slot 3d  
Tslot ← T + 2c, GOTO[RestartTimer], AT[TimerTable,15]; \*slot 2d  
Tslot ← T + 1c, GOTO[RestartTimer], AT[TimerTable,16]; \*slot 1d  
Tslot ← T + 0c, GOTO[RestartTimer], AT[TimerTable,17]; \*slot 0d

\*Refresh has been started.

RefreshNext: ADTOTIMER[RTimer];

\*load the refresh timer

\*Check for Mouse halt

CheckMouse: T ← 10000C;  
LU ← (PRINTER) AND (T);  
DBLGOTO[MouseHalt,TimerRet,ALU#0];  
TimerRet: RETURN;

\*check for mouse halt

\*for task switching

\*Mouse halt, Midas breakpoint

MouseHalt: GOTO[.], SETFAULT;

\*timers cannot be restarted.

RestartTimer: LU ← (ExpectedSlot) - (T);

\*insist that the slots expire in order 14..0

WrongSlotExpired: BREAKPOINT;  
ExpectedSlot ← (ExpectedSlot)-1;  
SKIP[ALU>=0];  
ExpectedSlot ← 16c;  
T ← (Tslot) OR (TimerValue);  
TimerTemp ← T;  
ADDTOTIMER[TimerTemp];

\*reset expectedslot if negative

\*Set RM[20b + Tslot] to TimeOut - if this register ever becomes zero.

\*the main program will restart the timer and cause an error.

Tslot ← (Tslot) + (20c);  
T ← STKP;  
Tslot ← T, STKP ← Tslot;  
Tslot ← (Tslot) XOR (377C);  
STACK ← TimeOut;  
STKP ← Tslot. RETURN;

\*Array of count words is in RM[20b]-RM[36b]

\*restore stackpointer

\*SubTest 2 (Check Timeout routine)

```
SETTASK[0];
ONPAGE[1];
```

start:

```
go:      SubTest ← 2C;
        InnerCount ← 0C;
        CLEARMPANEL, CALL[SwitchTo14];
```

\*reset InnerCount  
\*Notify timer initialization stuff

\*come back here after timer initialization

```
SetTimers:  SubTest ← 2C;
            Tslot0 ← T ← (16C), GOTO[StartTLoop];
```

\*switch to Task 14, InitTimers

```
SwitchTo14: SubTest ← 0C;
            Tslot0 ← TimerInitlocl;
            Tslot0 ← (Tslot0) OR (TimerInitlocl);
            APC&APCTask ← Tslot0;
            RETURN;
```

```
StartTLoop: CALL[StartTimer];
            Tslot0 ← T ← (Tslot0)-1;
            GOTO[BigLoop, ALU<0];
            GOTO[StartTLoop];
```

\*start timer, set count to TimeOut

```
BigLoop:   Tslot0 ← T ← (16C);
MainLoop:  Temp ← T;
```

```
NOP;
Temp ← (Temp) + (20c);
SubTest ← 1C, TASK;
```

\*make the loop longer

```
CheckTimeout: SubTest ← 2C;
              STKP ← Temp;
              STACK ← (STACK) - (1C), GOTO[NoFail,R>=0];
```

\*slot Tslot0 got restarted in time

```
Fail:      BREAKPOINT, INCOMPANEL;
            GOTO[go];
```

```
NoFail:    Tslot0 ← T ← (Tslot0) - (1C);
            GOTO[MainLoop, ALU>=0];
            InnerCount ← T ← (InnerCount) + (1C);
            LU ← (MaxInnerCount) - (T);
            GOTO[OnePassFinished, ALU=0];
            GOTO[BigLoop];
```

\*increment InnerCount

\*finished one pass ?

```
OnePassFinished: PassCount ← T ← (PassCount) + (1C);
                 LU ← (MaxPass) - (T);
                 GOTO[Passed-EDtimex-Test, ALU=0];
                 GOTO[go];
```

\*increment PassCount

\*finished all passes ?

```
Passed-EDtimex-Test: PassCount ← 0C, GOTO[go], BREAKPOINT;
```

```
***** SUBROUTINE: StartTimer *****
*
*   to load the timer whose slot is in Tslot0 and
*   to set RM[Tslot0 + 20b] to TimeOut

StartTimer: USECTASK;
            T ← APC&APCTASK;
            Rlink0 ← T;

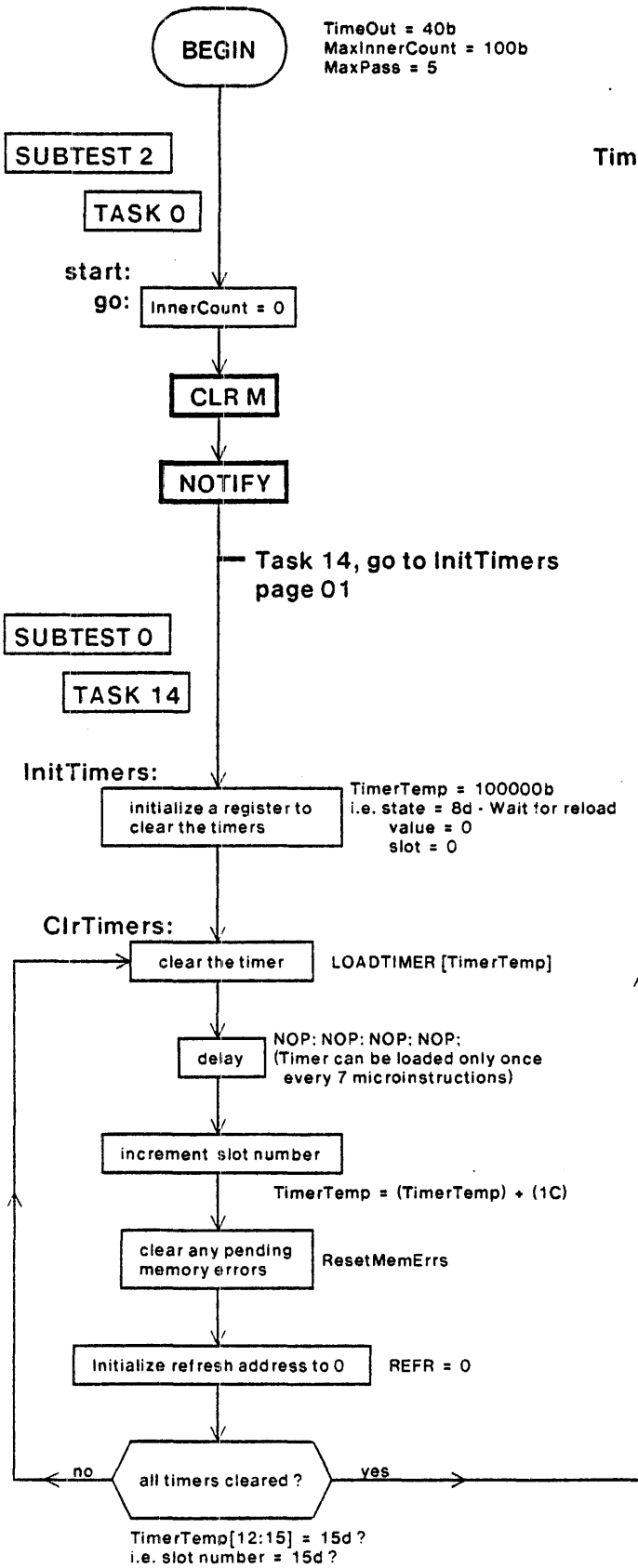
            T ← Tslot0;                    *Tslot0 has the slot number
            Temp ← T;
            Temp ← (Temp) OR (TimerValue);
            T ← 1sh[Tslot0,6];
            Temp ← (Temp) - (T);          *subtract (slot number x 4) from TimerValue
            LoadTimer[Temp];
            T ← Tslot0;
            Temp ← T;
            Temp ← (Temp) + (20c);
            STKP ← Temp;                  *set count to TimeOut
            STACK ← TimeOut;
            Temp ← T;                    *restore the slot number

            APC&APCTASK ← Rlink0;
            RETURN;

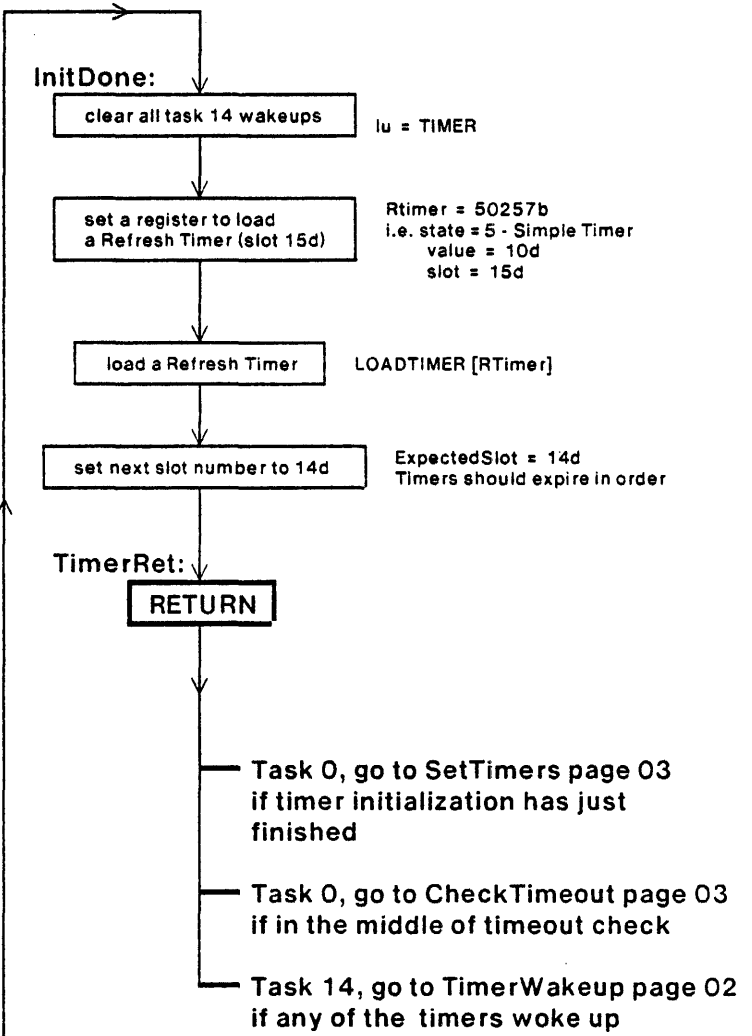
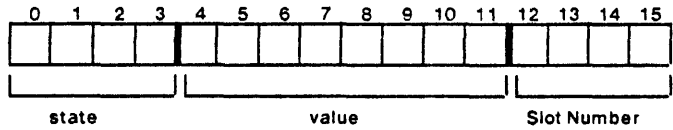
END;
```

**EDtimex**

TimeOut = 40b  
 MaxInnerCount = 100b  
 MaxPass = 5



**Timer Configuration:**

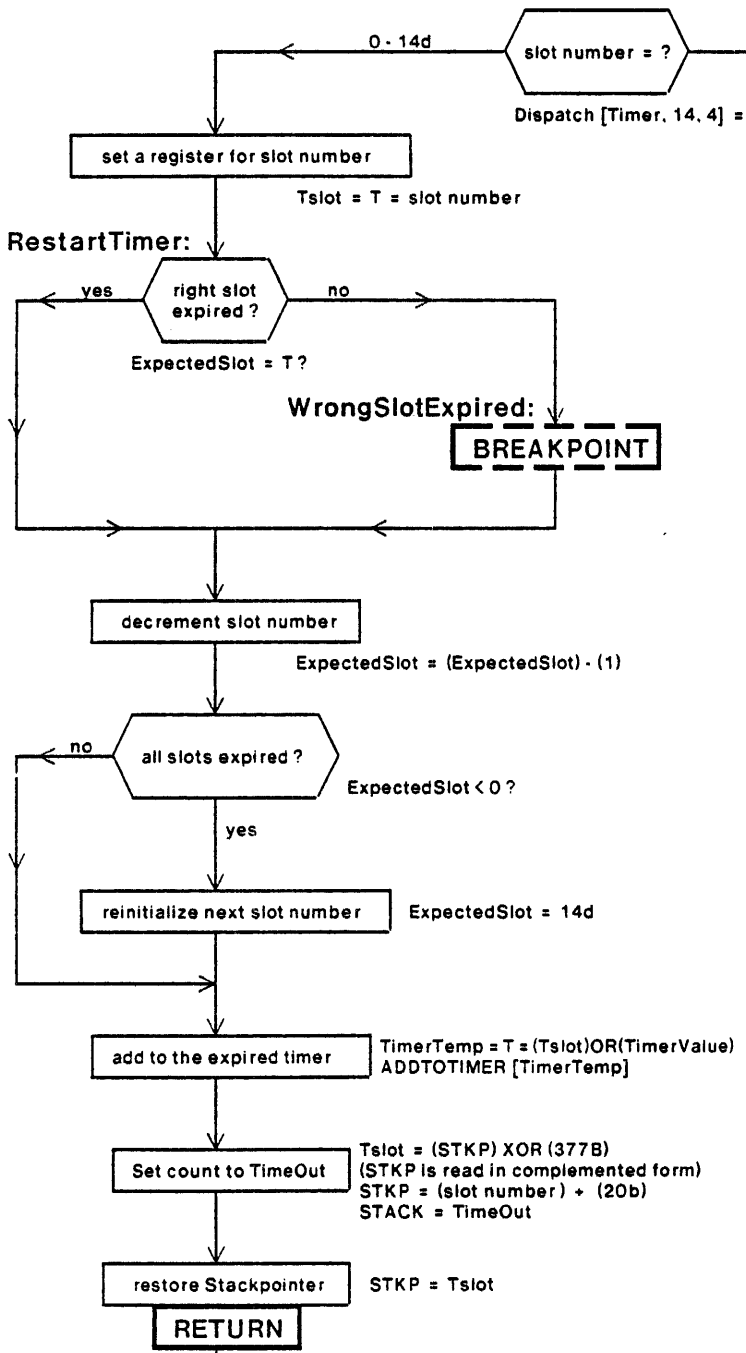


XEROX ED	D(0) Diagnostic	PROGRAM NAME EDtimex.mc	DOCUMENTATION FILE EDtimex01.sil	DESIGNER K. Mayekawa	REV 1	DATE 2/26/80	PAGE 01
-------------	--------------------	----------------------------	-------------------------------------	-------------------------	----------	-----------------	------------

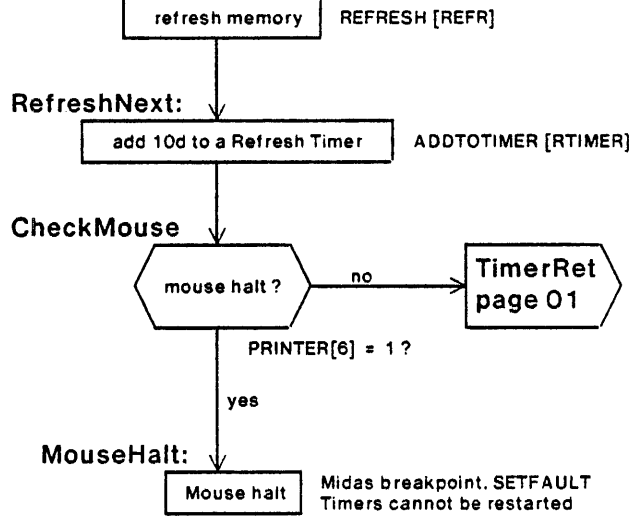
**SUBTEST 1**

**TASK 14**

**TimerWakeup:**



**Timers:**

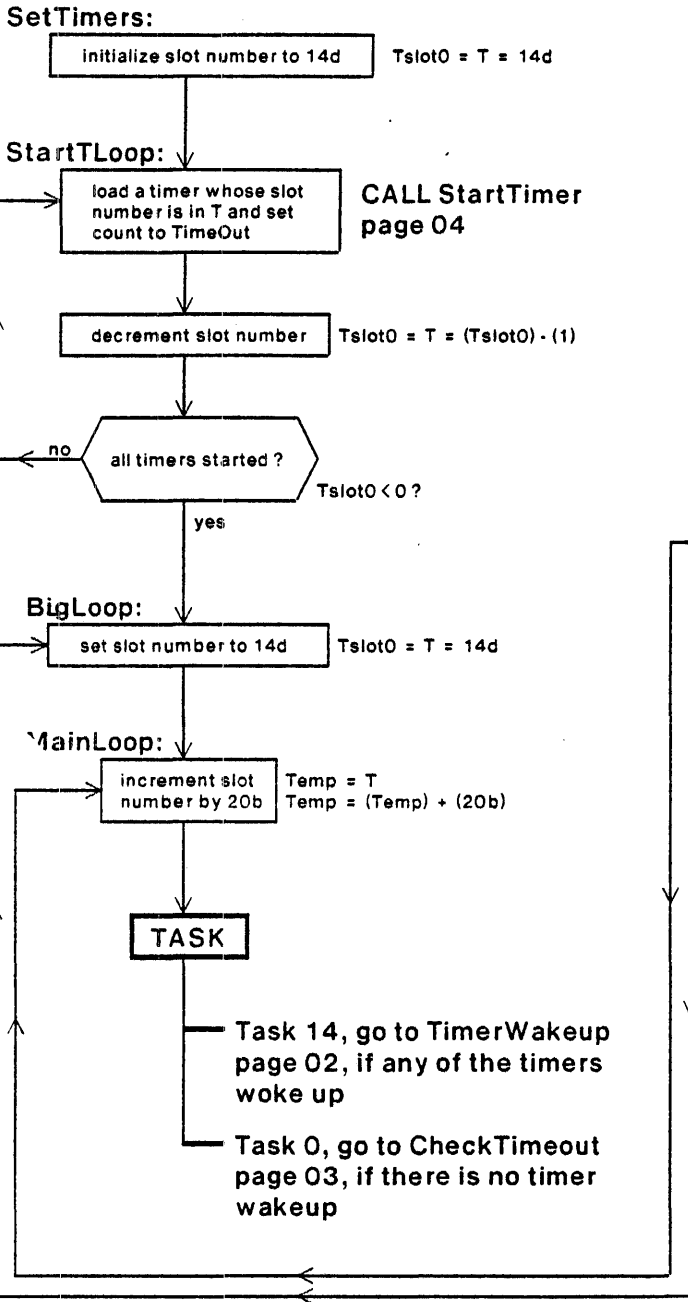


Task 0, go to CheckTimeout page 03 if there is no timer wakeup  
 Task 14, go to TimerWakeup page 02 if any of the timers woke up

XEROX ED	D(O) Diagnostic	PROGRAM NAME EDtimex.mc	DOCUMENTATION FILE EDtimex02.sil	DESIGNER K. Mayekawa	REV 1	DATE 2/22/80	PAGE 02
-------------	--------------------	----------------------------	-------------------------------------	-------------------------	----------	-----------------	------------

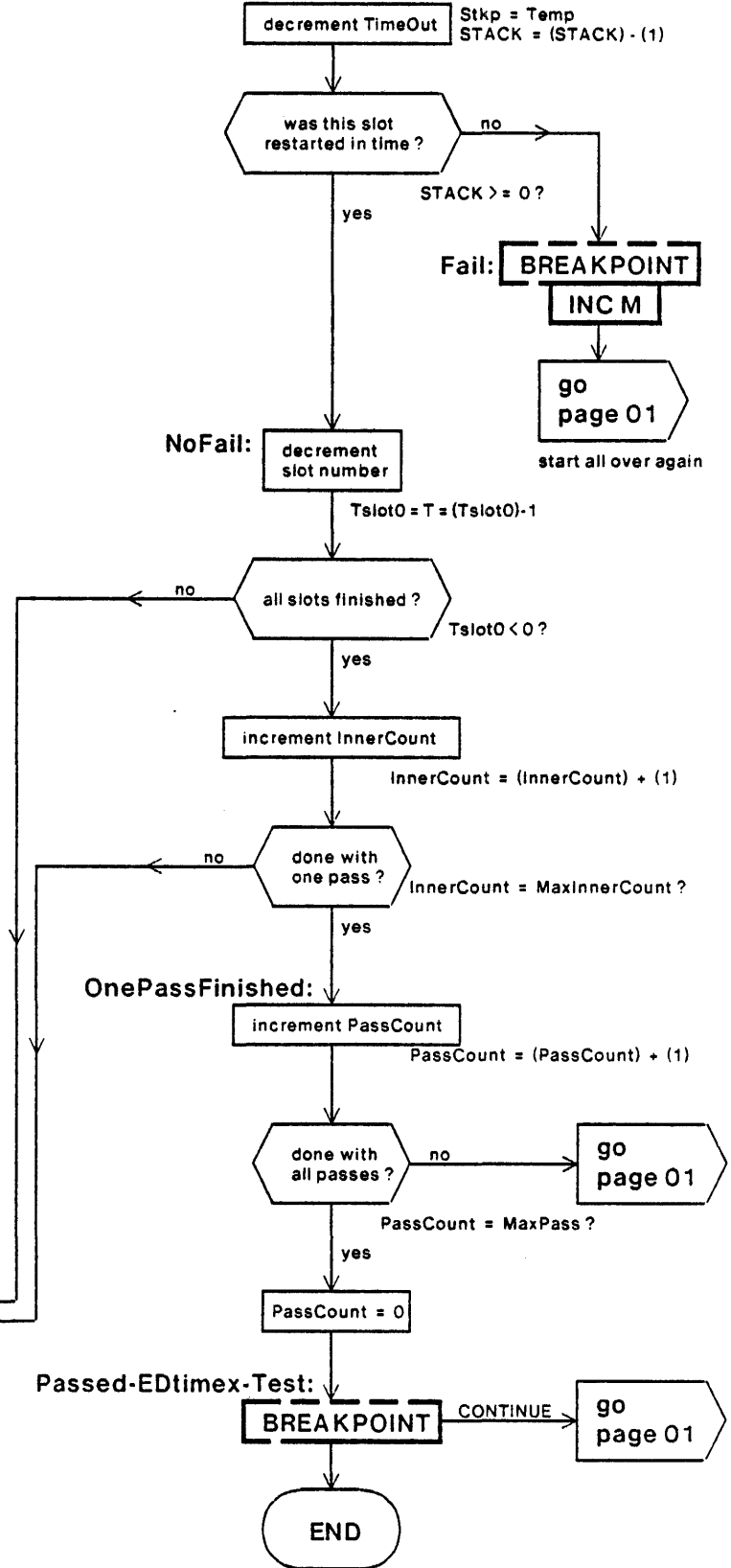
**SUBTEST 2**

**SK 0**



**TASK 0**

**CheckTimeout:**



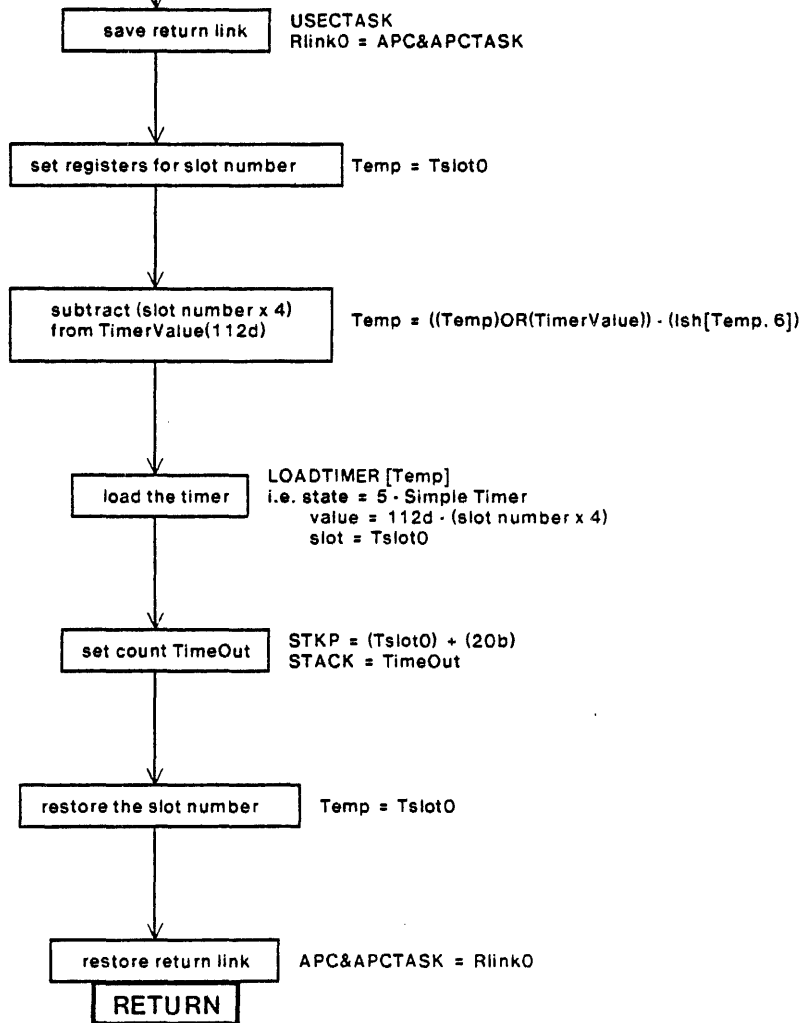
XEROX	D(O)	PROGRAM NAME	DOCUMENTATION FILE	DESIGNER	REV	DATE	PAGE
ED	Diaagnostic	EDtimex.mc	EDtimex03.sil	K. Mayekawa	1	2/26/80	03

**SUBROUTINE**

**StartTimer**

StartTimer loads the timer whose slot is in Tslot0 and sets RM[Tslot0 + 20b] to TimeOut (= 40b). Value loaded is (112d - (slot number x 4)) so that the timers expire from slot 14d through slot 0 sequentially.

slot number (decimal)	value loaded (decimal)
14	56
13	60
12	64
11	68
10	72
9	76
8	80
7	84
6	88
5	92
4	96
3	100
2	104
1	108
0	112





PARITY	0	REVISION	1	COMM-ER0	0
CYCLECONTROL	63	RUN-TIME	12	COMM-ER1	0
PCXREG	7	PASSCOUNT	0	COMM-ER2	0
PCFREG	7	MAXPASS	5	BOOT-ERR	0
DBREG	77	*SUBTEST	2	*BOOTREASON	40
SBREG	77			MEMSYNDROME	131000
MNBR	3423	INNERCOUNT	0		
*SSTKP	377	MAXINNERCOUNT	100		
STKP	0				
*ALURESULT	3	TSLOT0	0		
*SALUF	377	TSLOT	0		
T 20	7000	EXPECTEDSLOT	0		
AATOVA	0	RTIMER	0		
TPC 20	7777				
CALLER	ILC@+7630				
*PAGE	1				
*APC	7011				
*APCTASK	16				
*CIA	00+1				
CTASK	0				

Loaded: EDTIMEX

Time: 08.47

Step at 0:00, BP at 0:00+1

Exit Boot Run-Prog Read-Cmds Break UnBreak C1rAddedBPs C1rAllBPs ShowBPs Go  
 SS **Continue** Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds  
 Virtual

MicroD 8.6 (OS 16) of April 27, 1979  
at 29-Feb-80 9:28:41

microd.run Edtimex

Edtimex.DIB 152b instructions written 29-Feb-80 9:27:57

Total of 152b instructions

Checking for errors...  
Linking...  
Building allocation lists...  
Assigning locations...  
152b instructions in rings involving ONPAGE or AT  
Reloading binaries...  
Checking assignment...  
Writing .MB file...  
Writing listing...

IM:

Imag	Real	W0	W1	W2	Symbol
Edtimex.DIB:					
0	@1020	10030	101013	10	INITTIMERS
1	1005	10142	25064	10	CLRTIMERS
2	1032	50	25063	0	(+1)
3	1031	50	25060	0	(+2)
4	1030	50	25056	0	(+3)
5	1027	50	25055	0	(+4)
6	1026	11050	133052	10	(+5)
7	1025	10200	37051	10	(+6)
10	1024	12020	100010	0	(+7)
11	1004	64150	25047	14	INITDONE
12	1023	10025	101045	14	(+1)
13	1022	10312	137042	14	(+2)
14	1021	10142	25036	14	(+3)
15	1017	10000	135021	4	(+4)
16	1010	50	25215	0	(+5)
17	1011	64174	1035	14	TIMERWAKEUP
20	1016	50	25601	1	(+1)
21	@1100	152350	1033	0	TIMERS
22	@1101	10000	175113	0	(+1)
23	@1102	10000	173113	0	(+2)
24	@1103	10000	171112	0	(+3)
25	@1104	10000	167113	0	(+4)
26	@1105	10000	165112	0	(+5)
27	@1106	10000	163112	0	(+6)
30	@1107	10000	161113	0	(+7)
31	@1110	10000	157113	0	(+10)
32	@1111	10000	155112	0	(+11)
33	@1112	10000	153112	0	(+12)
34	@1113	10000	151113	0	(+13)
35	@1114	10000	147112	0	(+14)
36	@1115	10000	145113	0	(+15)
37	@1116	10000	143113	0	(+16)
40	@1117	10020	141113	0	(+17)
41	1015	10143	25031	14	REFRESHNEXT
42	1014	21	41026	0	CHECKMOUSE
43	1013	62250	1024	14	(+1)
44	1012	50	24015	0	(+2)
45	1006	50	25401	0	TIMERRET
46	1007	47	17016	0	MOUSEHALT
47	1045	11450	25110	4	RESTARTTIMER
50	1044	50	24004	0	(+1)
51	b 1003	50	25005	0	WRONGSLOTEXPIRED
52	1002	11350	125107	4	(+1)
53	1043	50	24200	0	(+2)
54	1001	10000	135000	4	(+3)
55	1000	10325	57104	0	(+4)
56	1042	10050	125103	10	(+5)
57	1041	10143	25101	10	(+6)
60	1040	11101	101076	0	(+7)
61	1037	70166	47075	14	(+10)

62	1036	10050	103073	0	(+11)
63	1035	10417	137070	0	(+12)
64	1034	40002	101067	14	(+13)
65	1033	10150	3400	0	(+14)
66	420	0	105134	0	GO START
67	456	2020	101035	10	(+1)
70	416	47	7252	0	(+2)
71	417	0	105132	0	SETTIMERS
72	455	4000	175014	0	(+1)
73	425	20	101050	0	SWITCHTO14
74	424	4001	101046	0	(+1)
75	423	4336	105045	0	(+2)
76	422	4147	21043	0	(+3)
77	421	50	25401	0	(+4)
100	406	50	25314	0	STARTTLOOP
101	407	5350	165131	0	(+1)
102	454	50	24230	0	(+2)
103	414	50	25014	0	(+3)
104	415	4000	175000	0	BIGLOOP
105	400	4050	125122	4	MAINLOOP
106	451	50	25120	0	(+1)
107	450	5101	101024	4	(+2)
110	412	0	103317	0	(+3)
111	447	0	105401	0	CHECKTIMEOUT
112	413	4150	3126	4	(+1)
113	453	41400	102410	14	(+2)
114	b 405	47	5136	0	FAIL
115	457	50	25041	0	(+1)
116	404	5400	143125	0	NOFAIL
117	452	50	24200	0	(+1)
120	401	3100	143147	10	(+2)
121	463	3450	25144	14	(+3)
122	462	50	24020	0	(+4)
123	411	50	25033	0	(+5)
124	410	3100	143143	0	ONEPASSFINISHED
125	461	3450	25140	4	(+1)
126	460	50	24004	0	(+2)
127	403	50	25041	0	(+3)
130	b 402	2020	101040	0	PASSED-EDTIMEX-TEST
131	446	47	27112	0	STARTTIMER
132	445	50150	65110	14	(+1)
133	444	50	125107	4	(+2)
134	443	4150	65105	0	(+3)
135	442	4050	125103	4	(+4)
136	441	4325	117100	4	(+5)
137	440	4174	55076	0	(+6)
140	437	5450	125075	4	(+7)
141	436	4142	25073	4	(+10)
142	435	4150	65070	0	(+11)
143	434	4050	125067	4	(+12)
144	433	5101	101065	4	(+13)
145	432	4150	3063	4	(+14)
146	431	40002	101061	14	(+15)
147	430	4050	125057	4	(+16)
150	427	147	21054	4	(+17)
151	426	50	25401	0	(+20)

Page 400: 64 locations used. 314 free

Page 1000: 66 locations used. 312 free

RM:

60	SUBTEST
61	RLINKO
62	1 REVISION
63	12 RUN-TIME
64	0 PASSCOUNT
65	5 MAXPASS
66	INNERCOUNT
67	100 MAXINNERCOUNT
70	TSLOT0
71	TEMP
72	TEMP1
340	TSLOT
341	EXPECTEDSLOT
342	TIMERTEMP

343

RTIMER

344

REFR RLC@

Time: 7 seconds; 0 error(s), 0 warning(s). 11947 words free

```
.....  
:EDtimexLog.MIDAS : Logger for EDtimex program  
: By: K. Mayekawa  
: Feb. 25, 1980  
:.....
```

```
.start      L X AppendOutput EDtimex.report;  
            L X WriteMessage ~***** START EDtimex Test : ;  
            L X WriteDT;  
            L X WriteMessage *****~ ;  
            L X Skip .continue;
```

```
.breakpoint L X AppendOutput EDtimex.report;  
            L A18 SkipNE WRONGSLOTEXPIRED;  
            L X Skip .wrongslotexpired;  
            L A18 SkipNE FAIL;  
            L X Skip .fail;  
            L A18 SkipNE PASSED-EDTIMEX-TEST;  
            L X Skip .passtest;
```

```
.notmybreak L X AppendOutput EDtimex.report;  
            L X WriteMessage *** FAILED: Not at my breakpoint ~;  
  
            L X WriteMessage ' Parity = ;  
            R A0 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CIA = ;  
            R A18 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' CTASK = ;  
            R A19 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APCTASK = ;  
            R A17 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' APC = ;  
            R A16 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X WriteMessage ' TPC = ;  
            R A13 Val;  
            L X WriteMessage;  
            L X WriteMessage ~;  
  
            L X CloseOutput;  
            L X Exit;
```

```
.wrongslotexpired L X WriteMessage *** FAILED: at my Breakpoint ~;  
                  L X WriteMessage * Wrong Slot Expired ~;
```

```
.bad            L X WriteMessage ' SUBTEST = ;  
                R B4 Val;  
                L X WriteMessage;  
                L X WriteMessage ~;  
  
                L X WriteMessage ' PASSCOUNT = ;  
                R B2 Val;  
                L X WriteMessage;  
                L X WriteMessage ~;  
  
                L X WriteMessage ' INNERCOUNT = ;  
                R B6 Val;  
                L X WriteMessage;  
                L X WriteMessage ~;  
  
                L X Skip .continue;
```

```
.fail      L X WriteMessage *** FAILED: at my Breakpoint ~:
           L X WriteMessage *           Timer Not Restarted ~:
           L X BackSkip .bad;

.passtest  L X WriteMessage ~----- PASSEd EDtimex Test : ;
           L X WriteDT;
           L X WriteMessage ~----- ;
           L X Skip .continue;

.continue  L X WriteMessage ~:
           L X CloseOutput;
           L X DisplayOn;
           L X Confirm;
           L X TimeOut 10000000;
           L X Continue;
           L X Skip 2;
           L X ShowError Program failed to CONTINUE.:
           L X BackSkip .notmybreak;
           L X DisplayOff;
           L X BackSkip .breakpoint;
```

L A19 Val 0  
L X Confirm  
L X Load EDTIMEX:  
L B0 Addr REVISION  
L B1 Addr RUN-TIME  
L B2 Addr PASSCOUNT  
L B3 Addr MAXPASS  
L B4 Addr SUBTEST  
L B6 Addr INNERCOUNT  
L B7 Addr MAXINNERCOUNT  
L B9 Addr TSLOT0  
L B10 Addr TSLOT  
L B11 Addr EXPECTEDSLOT  
L B12 Addr RTIMER  
L X DisplayOn:  
L X TimeOut 10000  
L X SS GO  
L X Skip 1  
L X ShowError Single-step at GO hung