To: Distribution

From: James A. Bush

Date: August 2, 1978

Subject: Isolated Online Testing of Multics Processors

Note: This document incorporates changes that were agreed upon during the design review held on March 16, 1978, and attended by representatives from CISL, FED, MSS and Maintainability Design. Changes necessitated during the design, coding and debug phases of this project are also reflected in this document.

INTRODUCTION

This memorandom describes in detail, the requirements and implementation plans for a new subsystem that will provide the capability to test and diagnose Multics processors, in an isolated online environment. This new subsystem, which runs as a sub-executive under the TOLTS test and diagnostic executive, will be known as the ISolated OnLine Test System and will be referred to throughout this document by the acronym, ISOLTS. Comments on the material provided in this document are invited and may be mailed, via U. S. mail to:

James A. Bush
Honeywell Information Systems
Mail Station B-84
P. O. Box 6000
Phoenix, Arizona 85005

or by Multics mail on System M to:

Bush.MTD

PURPOSE

The purpose of the ISOLTS subsystem is to provide Multics sites that have redundant processors, a high degree of system availability and reduce the mean time to repair (MTTR) for a failed Multics processor. The ISOLTS subsystem will allow Field Engineering to:

- o run tests on a failing or suspect processor
- o diagnose and repair the failing processor
- o rerun the tests to verify successful repair
- o return the repaired processor to computer operations to be added to the system

All this can be done online, in an isolated environment without fear that the suspect processor will damage the Multics storage system or impair other users of the system.

DESIGN CRITERIA

Redundant processors are prevalent at most current Multics sites and with the recent announcement of the Level 68 DPS, all future Multics sites will have at least 2 processors. Multiple processor systems make a site much more flexible and less subject to long system interruptions, due to processor failure. This is important because most Multics sites operate in a time sharing service bureau environment and long system interruptions could have costly and other adverse effects on the user community of such a system. However, a service bureau environment normally requires that a system remain in service for most of the day and night. This is where the problem lies. After a system interruption has occurred and it is determined that a processor has failed, the failing processor is removed from the configuration and the system is brought back up for service in a degraded mode. How is the failing processor diagnosed and fixed, so it may be returned to service? There are several alternatives listed below:

a. Field Engineering personnel could keep the system down, run the offline T & Ds on the failing processor, fix the problem and then return the failing processor and the system to service.
b. If a redundant IOM, console, tape subsystem and printer are available, Field Engineering personnel could diagnose and fix the failing processor in parallel with degraded service.
c. The failing processor could be added to the system and online tests could be run against it, to determine the failure.
d. The failing processor and an area of memory could be dedicated to a diagnostic user. Isolated tests could be run on the failing processor from the active system. In this case the power of the online system would be available to provide test sequencing, control and diagnostics.

Alternatives (a) and (b) are too costly to be considered practical. Alternative (c) is risky at best. A means exists to guarantee that a user will execute on a particular processor, but there is no means available to keep other users from using the same processor. If the failure only occurs in the user ring, this type of testing might prove sufficient, but if the failure occurs in the hardcore supervisor, file system damage may result from continued use of the failing processor.

Alternative (d) however, offers all the advantages of alternative (b), but without requiring the redundant hardware. Therefore, alternative (d) has been chosen as the basis for the design of ISOLTS.

OVERVIEW

The ISOLTS subsystem, may be thought of as two isolated and independent pieces of procedure code that communicate with each other for service. The ISOLTS driver operates from an active Multics process and controls the testing and test sequencing of a processor that is under test.

Test programs, loaded into an appropriate area of memory by the ISOLTS driver and executed by the processor under test, are used to do the actual testing. Communication between the ISOLTS driver and the test programs is accomplished through the use of a communication region, located at a predefined and unpaged area in memory.

Communication from the ISOLTS driver to the test programs, might include such requests as: execute test program; halt operation immediately; execute program options; etc. Communication requests from the test programs to the ISOLTS driver might include: test program complete; load next test program; display error message; etc.

Isolation can be said to occur by the fact that the memory area used for the test programs is dedicated and is not subject to demand paging as most other system memory is. The base 64k words of memory on a non-bootload system controller is used for testing (until processor addressing is verified sufficiently to assure that an erroneous address above 64k will not be generated, the entire system controller and its memory is usurped from the system). The operator is requested to reconfigure the processor under test such that the selected system controller and memory will become zero based memory for that processor. The operator is also asked to disable all other processor ports and insert a 64k address plug in place of the current address plug for the selected system controller port.

Because operator intervention is required to reconfigure the processor, and very few functions on that processor can be assumed to be operating correctly, many precautions are taken to insure that the processor under test will not degrade system integrity.

The active Multics process, in which the ISOLTS driver is executing, has access to the dedicated memory area by means of an unpaged Segment Descriptor Word (SDW). The unpaged SDW that is built has the absolute system address of the dedicated memory area, and a bounds of 64k. This SDW is swapped for the SDW of a segment in the hardcore address space reserved for that purpose. The ISOLTS driver process may now reference the dedicated memory as contiguous storage by merely using a pointer to the reserved hardcore segment.

The ISOLTS driver process can now load test programs into the dedicated memory area and cause the processor under test to execute these programs by simulating an interrupt with a Set Memory Controller Interrupt Cell (SMIC) instruction. The communication region is checked periodically for test completion and or an error condition. This sequence is repeated for the next test and continues until no tests remain to be executed, an error occurs or the ISOLTS user requests program options. If an error occurs, the resultant error message is written to an error message file, which can be selectively displayed on the ISOLTS user terminal or oprinted by user option.

When ISOLTS testing is complete or if the ISOLTS process is wrapped up, either by error or by the user requesting termination, the reconfiguration process is reversed. The dedicated memory is returned to the system, the processor under test is made available for dynamic reconfiguration, and the SDW for the reserved hardcore segment is set to "0"b.

THE ISOLTS DRIVER

The ISOLTS driver is a pl1 procedure that provides service and interface functions for the entire ISOLTS subsystem. Control is passed to the ISOLTS driver from the TOLTS executive, when the keyword "isolts" is entered by the user in response to the TOLTS query:

```
***enter "polts", "molts", "isolts", "quit", or "msg1"
???
```

Although the functions of the ISOLTS driver are described in more detail in later sections of this document, it might be helpful to summarize these functions here. The ISOLTS driver will:

- o Provide the user with configuration information concerning the CPUs that are available for testing. (e.g. test pcd).
- o Provide an interface to the ISOLTS hardcore reconfiguration software, to set up or destroy the testing environment.
- o Provide an interface with the operator query facility to communicate manual reconfiguration requests to the system operator.
- o Provide a hardcore interface to execute privileged instructions (SMIC, SSCR, etc.).
- o Provide a user interface and options control for the active test programs.
- o Provide service functions like loading tests, displaying error messages, executing test options, etc.
- o Provide error message file processing and display control.

ISOLTS HARDCORE RECONFIGURATION

In order to provide isolation from the active Multics system and file system, the dynamic reconfiguration of a processor to be tested as well as a non-bootload system controller, will be necessary. To effect this reconfiguration, entries will be added to several hardcore reconfiguration modules. These new entries will be described in later paragraphs. First however, new data structures in the system configuration data base (scs), must be defined. These new data structures will facilitate communication between the new reconfiguration modules. These new data structures are defined below:

```
declare 1 scs$processor_test_data aligned ext,
        (2 active bit (1),
         2 scu_state bit (2),
         2 pad bit (15),
         2 cpu_tag fixed bin (5),
         2 scu_tag fixed bin (5),
         2 mask_cpu fixed bin (5)) unaligned;
declare scs$cfg_data_save fixed bin (71) aligned ext;
declare scs$cpu_test_mask bit (72) aligned ext;
declare scs$cpu_test_pattern bit (36) aligned ext;
```

1. processor_test_data.active
   This is a flag to indicate that the contents of the processor_test_data structure is valid and isolated testing of the defined processor is in progress.

2. processor_test_data.scu_state
   is a value representing the current state of the SCU, defined by processor_test_data.scu_tag, for reconfiguration and deconfiguration purposes. The scu_state values are defined below:

   "00"b - SCU defined by processor_test_data.scu_tag not yet effected.
   "01"b - All memory has been removed from the paging pool, port mask still unchanged.
   "10"b - All memory has been removed from the paging pool, port masks have been changed to enable only the cpu under test and the cpu designated by processor_test_data.mask_cpu.
   "11"b - All memory above 64k has been returned to the system paging pool, original port mask restored with the addition of the port that the cpu under test is on.

3. processor_test_data.cpu_tag
   This is the tag of the processor under test.

4. processor_test_data.scu_tag
   This is the tag of the system controller that is being used for isolated testing.

5. processor_test_data.mask_cpu
   Is the tag of the active processor that has an interrupt mask assigned to the system controller being used for testing.

6. cfg_data_save
   is an area to store the original SCU config data, via an RSCR-CFG instruction, to be restored after ISOLTS testing is complete.

7. cpu_test_mask
   is the SCU interrupt mask used to enable interrupt cells 0 and 12 for ISOLTS testing.

8. cpu_test_pattern
   is a SMIC pattern used to send either an interrupt cell 0 or 12 interrupt to the CPU under test.

The following entry descriptions describe the new reconfiguration entries and there functions:

Entry: reconfig$check_resource

This is the initial entry to the ISOLTS reconfiguration software. It is called by the ISOLTS driver, through a hardcore gate, to check the processor and system controller resources required for isolated testing of a processor.

Usage

```
     declare reconfig$check_resource entry (fixed bin (5),
             fixed bin (5), fixed bin (5), fixed bin (35));
     call reconfig$check_resource (cpu_tag, scu_tag,
             scu_port, code);
```

where:

1.	cpu_tag	is the tag of the processor to be tested. (input)
2.	scu_tag	is the tag of the SCU to be used for isolated testing. (input/output)
3.	scu_port	is the SCU port on which the CPU defined by cpu_tag is configured (output).
4.	code	is a standard system status code. (output)

Functions_Performed:

o The reconfig lock is set and it will remain set until processor testing has been terminated. Note that this will effectively disallow any system dynamic reconfiguration to take place while processor testing is in progress.

o The processor_data array in the scs data base is tested to determine if the processor to be tested is configured and offline. (i.e. cpu card is present and reflects the "off" state).

o A test is made to determine if at least 2 system controllers are configured and online. If this is true, and if the scu_tag on input has a value of -1, the non-bootload system controllers are tested for the following criteria to determine the system controller to use for ISOLTS testing:

  a. The system controller must have at least 64k of memory.
  b. The system controller cannot be externally interlaced.
  c. The system controller cannot have any abs_wired pages within its memory.

  Of the system controllers found meeting the above criteria, the one with the smallest amount of memory will be picked as the system controller to be used for ISOLTS testing.

  If the scu_tag on input has a value other than -1, the user wishes to specify which system controller to use. Tests are made to ensure that the requested system controller is online and is not the bootload SCU.

o Control is returned to the ISOLTS driver.

Entry: reconfig$create_cpu_test_env

This entry is called by the ISOLTS driver (through a hardcore gate) to set up the isolated environment for testing a processor. It is called after the operator has performed the necessary manual reconfiguration.

Usage

```
     declare reconfig$create_cpu_test_env entry (fixed bin (5),
             fixed bin (5), (4) bit (36), fixed bin (35));
     call reconfig$create_cpu_test_env (cpu_tag, scu_tag,
             switches, code);
```

where:

1.	cpu_tag	is the tag of the processor to be tested. (input)
2.	scu_tag	is the tag of the SCU to be used for isolated testing. (input)
3.	switches	is an array containing discrepancies between the expected and actual data read by the RSW 1 through RSW 4 instructions. (output)
4.	code	is a standard system status code. (output)

Functions Performed:

- o The processor_test_data structure, in scs, is initialized. The active flag is set and the cpu_tag and scu_tag are set with their respective values.
- o The entry start_cpu$configure_test_cpu (described in a later paragraph) is called to reconfigure the desired processor for isolated testing.
- o An unpaged SDW is constructed and its absolute address is set to the base of the selected system controller. The SDW bounds are set to 64k, the user's access is set to read and write, and the ring brackets set to the user's current ring validation level. This SDW is swapped for the SDW of the reserved hardcore segment "isolts_abs_seg". This will allow the user to store test programs in this memory area to be executed by the processor under test. A pointer to isolts_abs_seg can be obtained by the ring 4 user by calling the ring0_get_$segptr subroutine.
- o The config card image of the processor under test is changed from the "off" state to the "test" state.
- o Control is returned to the ISOLTS driver.

Entry: reconfig$destroy_cpu_test_env

This entry is called (through a hardcore gate) to effect a reversal of the reconfiguration that was done by the create_cpu_test_env entry. It may also be called by the hardcore answering service entry deact_proc, in the event of abnormal ISOLTS process termination.

Usage

```
declare reconfig$destroy_cpu_test_env entry;
call reconfig$destroy_cpu_test_env;
```

Functions Performed:

o The active flag in scs$processor_test_data is tested.

o If the active flag is found on, the scu_state flag in scs$processor_test_data is tested.

o If the scu_state flag indicates that the entire system controller and memory is being used, the system controller ports that were enabled before testing was initiated are re-enabled. If the scu_state indicates that only 64k is being used, this step is skipped.

o The mask assigned to the processor under test is destroyed.

o All of the memory in the selected system controller is given back to the system by calling freecore repeatedly.

o The SDW for isolts_abs_seg is set to "0"b.

o The scs$processor_test_data structure is initialized.

o The reconfig lock is reset. At this time, system dynamic reconfiguration of other mainframe system components, may be resumed.

o The config card image of the processor under test is changed from the "test" state back to the "off" state.

o Control is returned to the caller.

Entry: start_cpu$configure_test_cpu

This entry is called by create_cpu_test_env to do the physical work of reconfiguration of the processor to be tested and the selected system controller.

Usage

```
declare start_cpu$configure_test_cpu entry (fixed bin (35));
call start_cpu$configure_test_cpu (code);
```

where:

1. code	is a standard system status code. (output)

Functions Performed:

- The active flag in scs$processor_test_data is checked. To continue, it must be on.

- A template of the expected switch settings and a switch mask are built, based on information known about the processor to be tested (cpu tag, scu tag, fault base, etc). The switch template and mask will be used later to compare the results of read switch instructions.

- A call is made to init_scu$isolate_test_cpu (described in a later paragraph) to reconfigure the selected system controller.

- A primitive test of the processor will verify that the manual reconfiguration was done correctly and that the processor to be tested will not pose a threat to system integrity by running test programs.

- Because there is no way to determine if the operator set the port enable switches for the system controller correctly, several precautions are taken here.

- The lower 256k of memory on the selected system controller is filled with a two instruction sequence of STA *, DIS (* is equal to the current address).

- The first two pages of all other non-bootload system controllers are "borrowed" from the system and the STA *, DIS sequence is stored in each pair of locations of these two pages.

- The connect lock is set and all other active processors are forced to cease operation by sending them a connect.

- Selected fault and interrupt vectors in the system zero based memory are overlayed with the STA *, DIS sequence after saving their original contents in the process stack. The fault and interrupt vectors overlayed are:
  processor_start interrupt
  op_not_complete fault
  start_up fault
  lock_up fault
  trouble fault

- A SMIC instruction is now sent through the selected system controller to the processor to be tested with a processor_start_pattern (interrupt cell 0).

- A wait loop is entered, waiting for location 0 in the selected system controller's memory to change as a result of the STA 0 instruction previously stored there. If this location changes values, this means that the manual reconfiguration was done correctly.

- o If the wait loop times out, this might mean that either the processor to be tested was not manually reconfigured correctly or that the processor is not responding to the SMIC instruction correctly. Tests are made by searching the other non-bootload system controller's memory and looking at the fault and interrupt cells that were overlayed in the bootload system controller's memory.

- o If location 0 of one of the other system controllers has changed, the operator has enabled the wrong port. In this case, the operator will be informed to recheck the config panel on the processor to be tested.

- o In any case, the fault and interrupt vectors in the system zero based memory are restored to their original contents and the connect lock is reset, allowing other processors to resume operation.

- o If the test of the other non-bootload system controllers base 2k of memory did not find any of the cells changed, if must be assumed that the processor did not answer the interrupt correctly. In any case, the base 2k of memory on each of these system controllers is given back to the system.

- o Each location of the selected system controller's memory is now tested to see if any location has changed. If some location has changed other than location 0, this could mean that the processor to be tested has an addressing problem.

- o In any case, an error code is returned to the caller, indicating the nature of the failure. All reconfiguration that has been done up to this point will be undone before returning.

- o Assuming that the interrupt was answered correctly by the processor to be tested, we now must execute a read switch program to ensure that other switches have been set correctly.

- o This is accomplished as follows:

  a. A RSW "n", DIS sequence is set up in locations 0 and 1 of the selected system controller's memory.
  b. A SMIC instruction is issued to the processor to be tested.
  c. After an appropriate time period has elapsed, a STA 0 is written over the RSW "n" instruction.
  d. A SMIC instruction is issued to the processor to be tested.

e. Location 0 should now contain information that was stored in the A register, as a result of executing the RSW "n" instruction.

f. This information is saved and this RSW, STA sequence is repeated for each of 4 tags (1 through 4) of the RSW instruction.

o The read switch data is now exclusive ored with the switch template and "anded" with the switch mask. The result of this boolean operation should yield a value of zero if all the processor config switches were set correctly. If discrepancies are found, the resultant switch data is returned to the caller along with an appropriate error code.

o Assuming that the read switch test found no discrepancies, a test is now made to determine if a "LDA 2" instruction operates correctly. This is accomplished as follows:

a. Location 2 of our dedicated memory is set with a value of 0.

b. A LDA 2, DIS instruction pair is set in locations 0 an 1 of our dedicated memory.

c. A SMIC instruction is issued to the processor to be tested.

d. After an appropriate time period has elapsed, a STA 0 is written over the LDA 0 instruction.

e. A SMIC instruction is issued to the processor to be tested.

f. Location 0 (and the A register) should now contain a value of 0. If this is not true, an appropriate error code is returned to the caller.

o Since we want to return all memory in the selected system controller above 64k to the system paging pool, we must verify that the processor to be tested cannot address above 64k. (Because of the 64k size plug.) This is accomplished as follows:

a. The "store fault" fault vector address in our dedicated memory is overlayed with a STA *, DIS pair.

b. Locations 0 and 1 of our dedicated memory are overlayed with a LDA 65536 (64k), DIS pair.

c. A SMIC instruction is issued to the processor to be tested.

d. After an appropriate delay period, the store fault vector is checked, it should contain a value of 0. If it does not, an appropriate error code is returned to the caller.

o Since our active test programs will be using a simulated IOM 0 terminate interrupt (interrupt cell 12) for testing, we must check to see if the processor to be tested is capable of answering this interrupt. This is done as follows:

a. scs$cpu_test_pattern is set with bit 13 which is analogous to interrupt cell 12.
b. The IOM 0 terminate vector location in our dedicated memory is overlayed with a STA *, DIS pair.
c. A SMIC instruction is issued to the processor to be tested with the SMIC pattern in scs$cpu_test_pattern.
d. After an appropriate delay period, the IOM 0 terminate interrupt vector location is checked and should be equal to zero. If this is not true an appropriate error code is returned to the caller.

o The original port masks for the selected system controller, in addition to the port on which the processor to be tested is on, is restored by an SSCR-CFG instruction.

o The processor required state is now reset.

o All memory above 64k in the selected system controller is given back to the system by calling freecore repeatedly.

o The scu_state indicator is changed to a value of "11"b.

o If all the above tests worked correctly, the error code is set to zero and control is returned to the caller.

Entry: start_cpu$int_test_cpu

This entry is called by the ISOLTS driver (through a hardcore gate) to send an interrupt to the processor under test, via a SMIC instruction.

Usage

```
declare start_cpu$int_test_cpu entry (fixed bin (35));
call start_cpu$int_test_cpu (code);
```

where:

1. code is a standard system status code. (output)

Functions_Performed:

o The active flag in scs$processor_test_data is tested. It must be on to continue.

- The mask_cpu tag in scs$processor_test_data is picked up and the ISOLTS process is forced to run on that processor by calling pxss$set_proc_required.

- The cpu_tag of the processor under test is picked up from scs$processor_test_data and used in the call to privileged_mode_util$smic_port (described in a later section) along with the smic_pattern from scs$cpu_test_pattern.

- The processor_required state is reset if required. (If the scu_state state indicates we have the entire system controller usurped, the processor_required state is not reset).

- Control is returned to the caller.

<u>Entry</u>: init_scu$isolate_test_cpu

This entry is called by start_cpu$configure_test_cpu to reconfigure the selected system controller in order to effect isolation of the processor to be tested.

<u>Usage</u>

```
     declare init_scu$isolate_test_cpu entry (fixed bin (35));
     call init_scu$isolate_test_cpu (code);
```

where:

1. code   is a standard system status code. (output)

<u>Functions Performed</u>:

- The active flag in scs$processor_test_data is tested. It must be set to continue.

- A search is made to find a processor that has a mask assigned to the selected system controller. This processor is then designated as our active processor and the cpu tag is saved in scs$processor_test_data.mask_cpu.

- The config data for the selected system controller is saved in scs$cfg_data_save to be restored after testing has been terminated.

- All of the memory configured on the selected system controller is removed from the paging pool by calling pc_abs$remove_core.

- The scu_state flag is changed to a value of "01"b.

- Our active process is forced to run on our designated active processor by calling pxss$set_proc_required with our active cpu as an argument.

- scs$cpu_test_mask is initialized with the port mask set for only the ports of the processor under test and the mask_cpu. The interrupt mask field is set to allow a cell 0 and cell 12 interrupt only.

- An SSCR-CFG instruction is issued to the selected system controller with the port enable bits set on for the processor under test and our active cpu. An interrupt mask register is also assigned to each of these ports.

- The scu_state flag is changed to a value of "10"b.

- Control is returned to the caller. Note that control is returned without releasing the processor_required state. We are therefore forced to run on the mask_cpu.

ADDITIONAL HARDCORE ENTRIES REQUIRED TO SUPPORT ISOLTS

New Hardcore Gate Segment

In order to control access to the new entries in the reconfiguration software, a new hardcore gate segment (tandd_) will be necessary. Currently access to system reconfiguration software is controlled on the hphcs_ gate segment. It would seem an unacceptable security risk to give the TOLTS project access to this highly privileged gate. In addition, the TOLTS project currently must have access to the phcs_ gate segment. By making the phcs_ entries required by the TOLTS project also available on the tandd_ gate, the test and diagnostic software could be changed to use this new gate. This would require that the TOLTS project have access to only one hardcore gate segment, tandd_, and system administrators would be able to control this access much more effectively.

Operator Query Facility

In order to allow operations to have complete control of the system configuration, a generalized operator query facility will be necessary. Isolated testing of a processor might well be attempted by a Field Engineering specialist at a remote location. In such a situation, the Field Engineer would have no knowledge of what was happening at any particular instant at the computer site. It would be unwise to allow this Field Engineer to effect system reconfiguration without knowledge and approval of the computer operations personnel. Because of this, the ISOLTS hardcore reconfiguration software will be designed to not allow any reconfiguration to take place without permission asked and granted from the operator. Also, since the ISOLTS reconfiguration software must rely on the operator to change configuration switch settings, a means must be available to ask the operator to change the switches and wait for his response. This is somewhat like the pre-MR5.0 dynamic reconfiguration software.

A New Privileged Entry

A new entry in the privileged_mode_util subroutine will be necessary to allow execution of a SMIC instruction through any system controller port. The current "smic" entry only allows sending a SMIC through the bootload system controller. The new entry, smic_port, will require the port number through which to execute the SMIC instruction as an argument, in addition to the smic pattern argument.

TEST PROGRAMS USED FOR ISOLATED TESTING

Ideally, new test routines could be written to execute under ISOLTS. The fact that an active system is available to inspect the results of a particular test execution would be a powerful diagnostic aide. However manpower, budget, and time constraints dictate that existing test routines will have to be used.

The offline Processor And Store version 2 (PAS2) processor test programs, in conjunction with the PAS2 test and diagnostic executive and the Primitive Function Tests (PFTs) will be used as the active test routines for ISOLTS.

The PFTs are a group of go-nogo tests which run, in the offline mode, as a result of booting the offline T & D tape. Their function is to verify the basic 20 instruction subset of the series 6000 instruction repertoire that will be used by the PAS2 executive. If the first PFT executes correctly, the second PFT is read in and so on until finally the PAS2 executive is read in by the last PFT. If any of the PFTs fail, the processor halts at a DIS instruction and by obtaining the instruction counter value of the DIS instruction from the processor maintenance panel and looking at a listing for the failing PFT, a field engineer can determine the functional area of failure.

Together, the PAS2 executive and processor test programs provide functional test and limited diagnostics for all functions of a processor not tested by the PFTs.

The PAS2 executive provides control sequencing and utility service functions for the PAS2 test programs. In the offline mode, all requests to perform I/O to a tape unit, printer or console, are either communicated to the PAS2 executive from the test programs via Master Mode Entry faults (MMEs), or initiated by the PAS2 executive directly. Program option control and operator interruption and termination of a test are handled by the PAS2 executive as well.

The PAS2 processor tests provide functional testing for all logic in a processor. If run in a fixed sequence, as they are designed to be, they can be thought of as building blocks. Each test checks out a particular functional area of the processor, and this functional area is used in the next test in the sequence to check out a functional area of greater complexity. This sequence continues through all of the tests, until the last test is encountered. This last test, the instruction sequence test, checks out processor instructions in random sequences to ensure that no instruction interaction or crosstalk occurs. If the PAS2 processor tests are run out of their designed sequence, through operator option, a warning is printed that the test called in uses unverified instructions and then a list of these unverified instructions is displayed for the operator's inspection.

In addition to the above test routines, a very basic and primitive test is performed by the ISOLTS reconfiguration software in hardcore to ensure that the CPU under test is configured correctly and will not jeopardize system integrity when running the aforementioned test programs. Because this basic configuration test is performed before the primitive function tests are executed, only four software instructions are used. Either a STA *, DIS, a RSW n, DIS or a LDA n, DIS sequence is overlayed in selected parts of the dedicated memory. These instruction pairs are executed by the processor under test, upon receipt of an interrupt (forcing a hardware XED instruction to execute the 2 instructions in the interrupt cell address). If a failure occurs during this primitive test, further testing of the processor is not allowed to continue and diagnostics about the nature of the failure are communicated to the ISOLTS user.

ISOLTS DRIVER/TEST EXECUTIVE COMMUNICATION REGION

Communication between the ISOLTS driver and the active test executives (the PFTs and the PAS2 executive), will be accomplished using a four-word communication region or "mailbox" area. This mailbox area will be set in all the PFTs and the PAS2 exec in a predetermined and standard location and all of the locations within the mailbox area will be assembled to contain a value of zero. The format of this mailbox area is described below:

Word 0 - ISOLTS/offline control.
If word 0 has a value of zero, then the PFT or PAS2 exec is operating in the offline mode. If word 0 has a non zero value, the ISOLTS subsystem is controlling the execution of a particular test.

Word 1 - ISOLTS service requested.
If word 1 has a value of zero, then no service was requested. If word 1 has a non zero value, service is required by the ISOLTS driver. The particular type of service is indicated by the action code in word 2 of the mailbox area.

Word 2 - Service action codes.
The least significant 15 bits of mailbox word 2 contain the action code for a service request for either the test executives or the ISOLTS driver depending on the particular action code. Bit 21 is the only action code that will be acted on by the test executive (PAS2). Bits 27 through 35 contain the action codes initiated by the test executives to be acted on by the ISOLTS driver. The definition of these action codes is described below:

bit 21 - Halt operation immediate. The result of a quit request or the invocation of the ISOLTS cleanup condition handler. An output options message request (bit 29) followed by an input request (bit 30) will be returned by PAS2.

bits 22-26 - reserved for future use.

bit 27 - Test complete/load next program. Load indicated test program into slave program area. The BCD name of the program to load is located in mailbox word 3. This is the only action code that is set by the PFTs.
bit 28 - Test completion/output options message. Used if only one test program has been requested to be executed, or after the end

of the test sequence. This request will be followed by an input request (bit 30).

bit 29 - Output options message. Used after an error message, a halt immediate request or PAS2 initialization. This request will be followed by an inout request (bit 30).

bit 30 - Inout request. Request for program options to be entered or a question to answered. A pointer to an area to store the requested input string and word length is located in mailbox word 3.

bit 31 - Type message. Output message to the *ISOLTS* driver terminal. A pointer to the message and the word length is located in mailbox word 3.

bit 32 - Print message. Outout message to the ISOLTS users terminal or route message to the isolts_err_log depending on the state of the error flag (bit 33).

bit 33 - Error flaq. This flag is used in conjunction with the print message request and when set indicates that the print message request if for the first line of an error message. In this case the message to be printed will be routed to the isolts_err_log instead of the ISOLTS users terminal.

bits 34-35 - Reserved for future use.

Word 3 - Program name/message pointer and length. Contains either a BCD character string, or a relative pointer in the upper half and word length in the lower half, depending upon the particular action code in word 2.

METHOD OF TESTING

The method of testing and communication between the ISOLTS driver and the test executives can best be described by the following sequential example, starting with the PFTs and continuing into the PAS2 test programs.

o An ISOLTS test request is issued (i.e. test cpu a).

o After determination of resources and appropriate hardcore reconfiguration (described in an earlier section) is accomplished, control is returned to the ISOLTS driver.

o At this time the ISOLTS driver has a pointer to an unpaged segment that bounds our dedicated memory area and a mask register in the selected SCU has been set up enabling the active process to communicate with the processor under test.

o The name of the first PFT is picked up from an internal static location within the ISOLTS driver and a library search is initiated to find this PFT within the Multics storage system.

o Assuming that the requested test is found, the driver goes through a loading process and copies the core image of the appropriate PFT into the dedicated memory area.

o Word 0 of the previously mentioned mailbox area is set to a non-zero value, indicating that ISOLTS is in control.

o An IOM 0 terminate interrupt (interrupt cell 12) is simulated in the CPU under test, by issuing a SMIC instruction from the active process through the selected SCU.

o At this time, the active process sets a timer, to go off at 2-second intervals and goes to sleep.

o The CPU under test gets the interrupt because it was left in a DIS state by the minimal hardcore test that was performed earlier (described in an earlier section).

o Since this is the same way that the PFT gets control in the offline world (assuming that the tape unit from which the boot was issued was attached to IOM 0), the PFT continues as normal with no knowledge that ISOLTS is in control.

o If the first PFT finds no problems, IOM mailboxes are set up to read the next record off the boot tape as if the PFT was executing in the offline mode.

o Before issuing the CIOC instruction to read tape, the PFT tests the ISOLTS control flag and finds it set. He will now transfer to a subroutine to set up the ISOLTS mailbox instead of issuing a CIOC instruction. Within this subroutine, the load next program action code (bit 27) is set in mailbox word 2 and the program name of the next PFT is stored in mailbox word 3.

o The PFT now sets word 1 of the mailbox area, to a non-zero value and falls into a DIS state, awaiting action by the ISOLTS driver.

o While the PFT was executing, the ISOLTS driver was sleeping, waking up every 2 seconds and checking to see if word 1 of the mailbox area had changed from a zero value to a non-zero value. If word 1 had not changed states, the driver goes back to sleep. If mailbox word 1 had not changed states in approximately 1 minute, it is assumed that the test has failed and the field engineer is instructed to physically inspect the processor under test to determine the failure.

o If the PFT just executed did not fail and mailbox word 1 has been set to a non-zero value, the ISOLTS driver picks up the next PFT test from mailbox word 3, finds this PFT in the appropriate library, loads the core image of the new PFT in the dedicated memory area, sets mailbox word 0 to a non-zero value, sets the connect operand word port number to the port number of the CPU under test and issues a simulated IOM 0 terminate interrupt.

o This sequence continues until all of the PFTs have been executed or until an ISOLTS driver timeout occurs.

o After the last PFT has completed successfully, a load next program action code is again set up in mailbox word 2 and the BCD name of the PAS2 exec is stored in mailbox word 3.

o The PAS2 executive program is now found in the appropriate library and the core image is loaded into the dedicated memory area. Mailbox word 0 is set to a non-zero value. A system survey table is built in the appropriate area as if the last PFT had performed this function. (This part of the last PFT was bypassed as the result of the ISOLTS control flag being non zero.) This survey table will contain simulated channels and device numbers for a tape subsystem, a printer and a console, as well as a simulated IOM port number, that will again be set to the port number of the CPU under test. To bypass unnecessary questions from the PAS2 exec, the simulated tape subsystem is set to a MTS400 type (ASA) and the simulated printer is set to a PRT201 type.

o A simulated IOM 0 terminate interrupt is issued via a SMIC instruction, and the active process goes into a sleep, check mailbox service request flag, sleep loop.

o The last PFT gets the interrupt and transfers to the canned address value of the PAS2 initialization routine.

o During initialization, the PAS2 exec tests the ISOLTS control flag and finds it set. He then overlays the CIOC instruction in the IOM connect routine to a transfer to a subroutine to set up the ISOLTS mailbox.

o After the PAS2 exec initializes itself, an I/O is set up to output the PAS2 greeting message on the console. However instead of issuing a CIOC instruction, control is transferred to the ISOLTS service routine. The PAS2 exec uses the simulated console channel as a key to determine what service code to set. In this case the service code for type message (bit 31) is set in mailbox word 2 and pointer and length information, about the PAS2 greeting message is stored in mailbox word 3. The ISOLTS service request flag is now set and the PAS2 exec falls into a DIS.

o The next interval that the ISOLTS driver wakes up, the ISOLTS service flag is checked and found to be set. The ISOLTS driver checks the service code and determines that it must display a message on the ISOLTS terminal. The message pointer and word count are used to pick up the message. The message is converted to ASCII, any ignore characters are stripped off, and the message is output on the ISOLTS terminal.

o An IOM 0 terminate interrupt is simulated and the ISOLTS driver goes back to sleep.

o When the PAS2 exec receives the terminate interrupt, it proceeds just as if the terminate had come from the write console request to output the greeting message.

o The PAS2 exec now prepares a formatted message that contains the processor configuration. In the offline mode, this information is normally output to the printer, so PAS2 sets up the first line to be output to the simulated printer channel. A print message service code (bit 32) is set in mailbox word 2. The ISOLTS service request flag is set and the PAS2 exec falls into a DIS.

o The next interval that the ISOLTS driver wakes up, the ISOLTS service flag is checked and is found to be set. The service code is checked and it is determined that a print request is required. The message pointer and length information in mailbox word 3, are used to pick up the requested message. The message is converted to ascii, and since the error service request flag (bit 33) was not set, the message is output on the ISOLTS terminal.

- o An IOM 0 terminate interrupt is simulated and the ISOLTS driver goes back to its sleep loop.

- o This sequence of interactions between the ISOLTS driver and the PAS2 exec is continued until all lines of the configuration message have been processed. At this time, control is returned to the PAS2 exec. The PAS2 exec responds by setting up a output options message (bit 29) service code.

- o After the ISOLTS driver displays the "options?" message on the ISOLTS terminal, a terminate is again sent to the PAS2 exec. The PAS2 exec responds with an input request (bit 30). The ISOLTS driver waits for the ISOLTS operator to respond with some set of valid PAS2 options. A program might be called in with a "pgmxxx" response, or the operator might depress a line feed character or a carriage return character (simulating the offline response of depressing the EOM button on the system console). This is a default request to sequence through all of the PAS2 processor tests.

- o Assuming that the ISOLTS operator wants to sequence through all of the processor tests, and depresses the return key on the terminal, control is returned to PAS2 by issuing a IOM 0 terminate interrupt.

- o The PAS2 exec responds by searching its tables of legal tests and finds the first test in the sequence. The BCD name of the test is loaded into mailbox word 3 and a load test program service code (bit 27) is set in mailbox word 2.

- o The ISOLTS driver searches the appropriate library for the requested test program. Assuming it is found, the core image of the requested test is loaded in the dedicated memory area and a IOM 0 terminate interrupt is simulated.

- o The PAS2 exec responds by transferring control to the test program loaded at the slave program base location.

- o If the test program found no errors, control is returned to the PAS2 exec and the next program in the sequence is requested to be loaded by the ISOLTS driver.

- o This sequence continues until all test programs have been executed or until an error is detected.

- o If an error is detected, an error message is written to the error message file, a line at a time, and then a request for program options service code is returned by the PAS2 exec At this time any valid PAS2 options may be entered.

CHANGES TO CURRENT OFFLINE TEST PROGRAMS

Changes to the current offline Primitive Function Tests, PAS2 exec and PAS2 processor programs, should be kept to a minimum. A design goal for the ISOLTS subsystem is to make all necessary changes in such a fashion as to have one common version of the affected test programs that will run offline as well as online. Also, changes will be made as general as possible so that other operating systems might take advantage of the ISOLTS technique.

Changes to the Primitive Function Tests

Changes to the PFTs involve assembling in the four word ISOLTS mailbox area, transferring to a routine to set the ld_program service code and set the service request flag instead of issuing a CIOC instruction, and branching around Port and IOM channel surveys, depending on the condition of the ISOLTS control flag.

Changes to the PAS2 Executive

The PAS2 exec will require the assembling in of the four word mailbox area (set at 1170 octal), and transferring to a routine that can decode or set the appropriate ISOLTS action codes in the mailbox area instead of issuing a CIOC instruction. These changes are fairly minor. However, since the PAS2 exec is currently running out of space (currently there is less than one k of space left between the end of the exec and the slave base), and since anyone making changes to PAS2 is constrained to use only the twenty instruction subset of the 6000 repertoire, the addition of coding of this nature may run over into the slave program area. If this happens, then the slave program base will have to be moved from the current 60000 (octal) address to 100000 (octal) as has recently been done for the NSA PAS2 exec.

Changes to the PAS2 Processor Test Programs

Unless the PAS2 slave base is moved up to 100000 (octal), then no changes will have to be made to the test programs themselves. However if the slave base is moved, it will require changes to all the PAS2 processor tests. Most of the tests could be changed by merely changing the "org" pseudo-op to a value of 100000 (octal) and resembling. However, in some tests this may prove more difficult, as some are address dependent.

Besides the changes listed above, it would be highly desirable if the PFTs, PAS2 exec, and the PAS2 test programs could be assembled in relocatable form, with symdefs defining any address that the ISOLTS driver has to know This would make the ISOLTS driver completely independent of any future changes to these programs, as required addresses could be determined at program load time.

THE ISOLTS ERROR MESSAGE FILE

Error messages that are produced by the PAS2 executive and test programs are rather lengthy and, in the offline world, are output to a printer. In the interest of making as few changes as possible to the PAS2 scheme for ISOLTS, a printer could be required to be attached to the ISOLTS driver process. However this would be a gross misuse of system resources and could inhibit some sites from allowing ISOLTS to be run until normal printer traffic had died down. In addition, one of the features of ISOLTS is that a Field Engineering specialist might be running ISOLTS from a remote location. It would be of little use to this specialist to have error messages go to a printer at the computer site. Because of the above reasons, the PAS2 error messages will be written to a Multics storage system file.

The ISOLTS error message file will be maintained by the ISOLTS driver and will be permanently located in the ISOLTS users default working directory (home directory). It will be organized much like the syserr_log file. It will have a maximum length of 64k, the message entries will be time stamped and will be structured in a threaded list. When the 64k maximum has been reached, the least recently used message entries will be overlayed. A header at the base of the error file will be maintained and will contain a pointer to the last message entry. This pointer will be updated as each new message entry is added.

ISOLTS driver requests will be available (when the driver is at an "OPTIONS?" request point) to:

a. display the last message entry on the users terminal.
b. display the nth message entry on the users terminal.
c. queue a dprint request for the last message entry.
d. queue a dprint request for the last n entries.
e. queue a dprint request for the entire error file.

During ISOLTS initialization, the ISOLTS driver searches for the error message file in the users default working directory (it will have a name of "isolts_err_log"). If it is not found, it is created and initialized. If it is found, error messages will be written to the next available entry location. In this way, a date/time-stamped continuous log of ISOLTS error messages will be maintained.

A WORD TO MARKETING

In recent years, MOS memory technology has undergone many evolutionary changes. Currently a four mega word system controller cabinet may contain up to 512k words of MOS memory and I am sure that density will increase to one mega word per system controller cabinet, before another year has passed. This certainly makes it tempting for marketing to sell a system with only one system controller to significantly reduce the cost. However I would like to point out here that if a system is delivered with only one system controller, not only does it disallow the online testing of processors, but also the presence of only one system controller on a system makes it impossible to maintain system availability in the event that the system controller fails. Another point is that although only two system controllers are required to allow ISOLTS to be run, it is highly desirable to have at least three system controllers. Consider a system with two system controllers each of which has 512k words of memory configured. When ISOLTS is started and until basic processor addressing is verified, an entire system controller and its memory is usurped from the system, in this case half of the systems memory. It is important to note that although ISOLTS requires the exclusive use of a system controller and all of its memory for a period of time, it requires a relatively small amount of memory (as little as 64k words) to be configured on that system controller. It is therefore highly desirable to have a system controller with a small amount of memory configured as a non-bootload system controller. This function could prove to be a market area for older technology memories and system controllers.