MCR10092
Fix test_cpu Test 51
Revision 1.1

Eric Swenson

July 22, 2021

1 Problem Description

When running `test_cpu` test 51 fails with a `fault_tag_1` exception. This occurs due to a bug in the test program, `sreg_no_write`, which attempts to load a pointer into a pointer register, but the data loaded is not guaranteed to contain a valid ITS pointer.

The messages emitted by `test_cpu` are:

```
Test 51  <sreg_no_write>
test_cpu: Test sreg_no_write encountered an unexpected fault_tag_1.
                         *** HARDWARE FAILING ***
         The function under test does not normally exhibit this failure.

MACHINE CONDITIONS AT 234|6500:

fault_tag_1 by sreg_no_write$|1774
(>system_library_tools>bound_cpu_tests_|105004)
referencing stack_4|40040 (in process dir)
Ascii data where pointer expected.
  Cannot get line number in sreg_no_write
105004  600024351520   epp1   pr6|24,*

Machine registers at time of fault

pr0 (ap) 234|4376          stack_4|4376
pr1 (ab) 234|6620          stack_4|6620
pr2 (bp) 234|6300          stack_4|6300
pr3 (bb) 234|6340          stack_4|6340 ( -> "tempseg_1            ")
pr4 (lp) 247|25050         !BBBKQGFJKBhFLp.area.linker|25050 (internal static|0 for
    ci_mod_case_2)
pr5 (lb) 247|25050         !BBBKQGFJKBhFLp.area.linker|25050 (internal static|0 for
    ci_mod_case_2)
pr6 (sp) 234|6220          stack_4|6220
pr7 (sb) 234|0             stack_4|0
```

```
x0  37324   x1   4376   x2   3320   x3   3164
x4      0   x5   3357   x6     10   x7      1
a 000376000004 q 107006000000 e 0
Timer reg - 117701, Ring alarm reg - 0

SCU Data:

  6530   400376250011 000000000007 400234000000 000000000000
         105004102200 040040000000 040040351440 200036251500

Fault Tag 1 Fault ( 7)
By: 376|105004 sreg_no_write|1774 (bound_cpu_tests_|105004)
Referencing: 234|40040 stack_4|40040
On: cpu a (#0)
Indicators: cary, tro, ^bar
APU Status: xsf, sd-on, pt-on, fabs
Instructions:
  6536   040040 3514 40    epp1     40040,f
  6537  2 00036 2515 00    spri1    pr2|36

Time stored: 07/22/21 1451.8 pdt Thu (154101741012027323)
Ring:         4

EIS Pointers and Lengths:

  6550   000400000101 000000000000 000000000000 000000000000
         000000000000 000000000000 000000000000 000000000000

CPU HISTORY REGISTERS AT TIME OF FAULT
DPS8 History Register Analysis

HR          IC or          c   Memory
id   Seg# [tpr.ca] opcode tag y Address mc flags

CU                  scpr  n* ?   242204 0 inh its
CU                        n* ?   242204 0 inh its
CU                        n* ?   242260 0 inh its
CU                 cmpx3  du d       0  0 inh
CU                   tze     *       0  0 inh
CU                  eax3  x3 ?       3  0 inh
CU                   xec     ?   127155 0 inh its
CU                  scpr  n* ?   242204 0 inh its
CU                        n* ?   242204 0 inh its
CU                        n* ?   242204 0 inh its
CU                        n* ?   242260 0 inh its
CU                 cmpx3  du d       0  0 inh
CU                   tze     *       0  0 inh
CU                  eax3  x3 ?       3  0 inh
CU                   xec     ?   127155 0 inh its
CU                  scpr  n* ?   242204 0 inh its
```

```
Test 52  <tnz>
r 14:55 1.263 83
```

The following ticket describes this issue:

https://multics-trac.swenson.org/ticket/229

2 Analysis

The offending instructions in `sreg_no_write.alm` are:

```
        epp1    pr6|20,* return ptr
        spri1   pr2|30   save return ptr in new frame
```

The first `epp1` instruction, which loads an ITS pointer into PR1, references an effective address of pr6|20. PR6 points to the current stack frame, and offset 20 refers to the `return_ptr` – the address to be returned to when the current call returns. In order to not get a fault during an `epp1` instruction, the 72-bit value referenced by the effective address must be a valid ITS pointer. If it isn't, a `fault_tag_1` fault will be raised by the hardware.

In a new stack frame, the `return_ptr` this pointer will be undefined. It will remain undefined until the procedure makes an external call. In the case of `sreg_no_write`, at this point in the instruction stream, no external call has been made, and therefore this value will be undefined. Since stack frames are allocated and deallocated based on the call history of the process, the values that exist in a new stack frame will be arbitrary, and quite likely not a valid ITS pointer.

The beginning of `sreg_no_write.alm|` looks like this;

```
        entry   sreg_no_write
        name    sreg_no_write

sreg_no_write:
        push

        tra     START
        org     1014

START:
        epp2    pr6|18,* next frame ptr
        epbp7   pr6|0    stack base
        epp3    pr2|32   next frame area
        nop     0,du
        inhibit on
        spri3   pr6|18   NEW next frame ptr
        spri3   pr7|20   NEW stack end ptr
        inhibit off
        epp1    pr6|20,* return ptr
        spri1   pr2|30   save return ptr in new frame
```

The `push` instruction directly after the entrypoint, pushes a new stack frame, and doesn't initialize the `return_ptr` field. As a result of the `push` instruction, PR6 is set to point to the new frame. Since there are no additional writes to pr6|20, the value there will be arbitrary.

None of the code that follows the above references pr6|20, and with the exception of the last instruction in the above code fragment, no code would use this value or the value stored by the `spri1` instruction. Consequently, the lsat two instructions in the above block are not actually needed. It seems that the code for `sreg_no_write` was created by disassembling other code which original exhibited the problem being tested by this test, and then extracting a fragment. Some of the lines of code have been commented out, suggesting that after the code was created, it was shortened. However, the two bad instructions were left in.

Simply removing these two instructions, while preventing the `fault_tag_1` fault, could well invalid the test because the following comment appears in the code:

```
" The following instruction will fail with a access violation
" no_write_permission, because it asks for PR2's segment number
" and gets THIS segment number instead.
" The placement of this instruction is also very important. It must
" be two locations prior to the page boundary in the inst. segment.
         sreg     pr2|16   save regs in new frame
         sti      pr2|26   save indicators in new frame
         spri5    pr2|28   save pr5 in new frame
```

If the two instructions are removed, two new instructions must be inserted in order to ensure that the actual instruction that the test is attempting to validate remains at an offset 2 locations prior to the page boundary of the segment.

The instruction that is being tested is:

```
sreg     pr2|16   save regs in new frame
```

3 Proposed Changes

Although an alternative proposal was explored, it was rejected by the MCRB. See the section "Alternative Proposed Change" for details.

The proposed change is to simply comment out the invocation of the `cpu_tests_$sreg_no_write` call (and the loop that calls it) so that Test 51 appears to run and pass. The reason for choosing this alternative is that the test has several coding issues and it was felt that the simple fix proposed in "Alternative Proposed Change" would still leave bad code that doesn't follow Multics coding guidelines. Furthermore, it is very unlikely that this test would ever again be useful since no Multics hardware is likely to be found and placed in service, and in the unlikely event that hardware is resurrected, it is unlikely to not have had the FCO installed that corrected the issue with the `sreg` instruction.

The DPS8M simulator does not have the same issue, which only occurred when the `sreg` instruction was 2 locations from the end of a page boundary.

As the test is no longer useful and applicable, and fixing the bad code would perhaps render the test

incable of detecting the original failure, and with no way to test that the rewrite would still detect a failing `sreg` instruction, the MCRB concluded that the test should not be fixed.

The proposed changes is to the `test_cpu` program. A comparison of the old and new version is shown below:

```
 ---------- test_cpu.pl1

compare_ascii >ldd>tools>source>bound_system_test_.s.archive::test_cpu.pl1 ==

Inserted in B:
B9
B10
B11      /****^ HISTORY COMMENTS:
B12        1) change(2021-07-22,Swenson), approve(2021-07-22,MCR10092),
B13           audit(2021-07-22,GDixon):
B14           Comment out test 51, which fails when run under test_cpu when preceeded by
B15           test 49.
B16                                                    END HISTORY COMMENTS */
B17
B18
Preceding:
A9       /* format: style4,delnl,insnl,ifthenstmt,indnoniterend,ifthendo,ifthen,^thendo */


Inserted in B:
B2124    /*
B2125       This test is commented out. See MCR10092 for details. The test makes invalid
    references to the return_ptr
B2126       field in the stack frame, such that it sometimes causes a fault_tag_1 fault
    during execution. Whether it
B2127       fails in this way or not is dependent on stack garbage left from earlier calls.
B2128
B2129       The MCRB decided to not fix sreg_no_write.alm because there were several
    things wrong with the test and there
B2130       is no way to really test whether the test would properly diagnose the issue it
    was intended to diagnose -- a
B2131       failure of the SREG instruction to properly perform its effective address
    calculation. The original hardware
B2132       was fixed with an FCO. That FCO was presumably applied to all Multics
    hardware. We have no working hardware
B2133       today to fix this. The conditions under which the SREG instructions failed
    were constrained -- the instruction
B2134       had to be within 2 words of a page boundary. This is unlikely to be an issue
    in the DPS8M simulator today.
B2135
B2136       Rather than let test 51 fail, causing confusion, it was felt better to simply
    comment out the test invocation.
B2137    */
B2138    /*
Preceding:
A2114             do CYCL = 1 to num_to_cycle;
```

```
Inserted in B:
B2142    */
Preceding:
A2117
A2118    exclude51:

Comparison finished: 3 differences, 26 lines.
```

4 Alternative Proposed Change

This proposed change was rejected by the MCRB. For documentation purposes, it is described below.

The fix involves commenting out these two instructions:

```
        epp1     pr6|20,* return ptr
        spri1    pr2|30   save return ptr in new frame
```

and two `NOP` instructions will be inserted, preserving the offset of the `sreg` instruction.

A comparison of the old and new versions, excluding history comments, follows:

```
 mbuild: cmp

---------- sreg_no_write.alm

compare_ascii >ldd>tools>source>bound_cpu_tests_.s.archive::sreg_no_write.alm ==

A24              epp1     pr6|20,* return ptr
A25              spri1    pr2|30   save return ptr in new frame
Changed by B to:
B24      "       epp1     pr6|20,* return ptr
B25      "       spri1    pr2|30   save return ptr in new frame
B26              nop
B27              nop

Comparison finished: 1 difference, 6 lines.
```

The `mbuild` description of the change looks like this:

```
Segments found by read request:

  Build_script:          cpu_tests.mb;

  bound_cpu_tests_.s.archive:
```

```
   source:              sreg_no_write.alm              IN: tools.source REPLACE compiler:
       alm;

mbuild:
```

---

5 Documentation

There are no documentation changes for this change.

6 Testing

Testing involves running the `test_cpu` program repeatedly, with different `-fm` values that ensure that test 51 is executed, but with different prior tests (to generate different stack garbage).

7 Version History

2021-07-22	1.0	Initial version of the MCR.
2021-07-22	1.1	Update to change proposed fix.