RISC-V Simulator Test Plan Record

Version: 0.00 | 2/12/2022

Editors: Hector Soto
Created By: Hector Soto (sotohec@pdx.edu)
December 2/12/2022

Preface

This document describes test plans and their results. The purpose of this document is to track tests and their results until tested code is functional with no/minimal errors. Errors/bugs encountered during tests will be recorded until fixed. If a bug is fixed, it will be removed from this document.

Table of Contents

Section 1 - SimMemory

This section goes over the testing of simulated memory in our simulation. It is defined as the class SimMemory with its definition + implementation in files “SimMemory.h”, “SimMemory.cpp”. SimMemory uses 4-byte aligned addressing (2 LSBs = 0). So simulated load/store instructions may have to make 2 calls instead of 1 when requesting 4-byte words or 2-byte half-words.

1.1 – Basic Load/Store Test (test_mm.cpp)

“test_mm.cpp” is a test designed for SimMemory’s ability to read/write to memory consistently if given a large amount of random memory requests with random amounts of contiguous memory being requested.

Runs by default have 65,536 memory request tests. A memory request test looks like the following.

```
Address = rand() % 0xFFFFFFFF; //Addresses from 0 -> 0xFFFFFFFE.
Amount = rand() % 512 + 1; //Will write 1 to 512 words.
RanData = rand() % 0xFFFFFFFF;
ExtraData += Amount - 1;
for(uint32_t j = 0; j < Amount; ++j)
{
        Data = Memory.GetDataPointer(Address + j*4); //j*4 to ge
        *Data = RanData;
        Data = Memory.GetDataPointer(Address + j*4);
        if(*Data == RanData)
        {
                ++Success;
        }
        else
        {
                printf("Error: Mismatch at address 0x%08x | 0x%0
        }
}
```

By the end of a run, around 16,800,000 attempts at reading and writing to a memory location are performed. It is possible for some attempts to be on the same address though as addresses are generated semi-randomly.

Biggest Flaw: Since this test immediately checks a memory location after writing it. There’s no guarantee the data is stored in that location upon the request of data at a different address. The code for SimMemory indicates that is not the case, but this test doesn’t check that.

There was only one bug encountered during testing, and it was from GetDataPointer() not correctly calculating where in the memory hierarchy a page was due to simply not shifting masked addresses. **Test now has a 100% success rate.**

```
hector@hector-VirtualBox:~/Desktop/RISC-V_Simulator/test_mm$ ./executable
Success/Failure: 16839061/16839061 | Success Rate: 100.00%
```

1.2 – Consistent Memory Test (test_cm.cpp)

“test_mm.cpp” is a test designed for SimMemory’s advanced load/store functions which can load/store a variable amount of data. More specifically, unlike GetDataPointer(), Store() and Load() can access 1-byte, 2-bytes, or 4-bytes of contiguous memory at any address. GetDataPointer() was limited to accessing 4-bytes on 4-byte aligned addresses.

This test generates one random address and one random 4-byte word. The first byte is stored at the address. The first 2-bytes is stored at the address + 1 (account for the byte written), and the entire 4-byte word is stored at the address + 3 (account for the 3-bytes written). This process then repeats on subsequent addresses from the original. So all memory locations written to should be contiguous. Then all memory locations written to are tested to see if they hold the correct data.

Biggest Flaw: Since only one random 4-byte word is generated and checked for, it is possible this test could fail to catch a consistently occurring bug in the Load/Store functions.

There was a minimal amount of bugs during testing.

Test now has a 100% success rate.

```
hector@hector-VirtualBox:~/Desktop/RISC-V_Simulator/test_cm$ ./executable
Success/Failure: 196605/196605 | Success Rate: 100.00%
```

Runs by default have 65,536 memory request tests. A memory request test looks like the following.

```
Address = rand() % 0xFFFFFFFF; //Addresses from 0 -> 0xFFFFFFFE.
RanData = rand() % 0xFFFFFFFF;
for(uint32_t i = 0; i < TEST_COUNT; ++i)
{
        Memory.Store(Address + 7*i, RanData, LS_BYTE); //Store LSByte.
        Memory.Store(Address + 1 + 7*i, RanData, LS_HALF); //Store first h
        Memory.Store(Address + 3 + 7*i, RanData, LS_WORD);
        ExtraData += 2;
}
for(uint32_t i = 0; i < TEST_COUNT; ++i)
{
        if(Memory.Load(Address + 7*i, LS_BYTE) == (RanData & 0xFF))
        {
                ++Success;
        }
        else
        {
                printf("Address: 0x%08X Data: 0x%08X | BYTE: 0x%08lX\n", A
        }
        if(Memory.Load(Address + 1 + 7*i, LS_HALF) == (RanData & 0xFFFF))
        {
                ++Success;
        }
        else
        {
                printf("Address: 0x%08X Data: 0x%08X | HALF: 0x%08lX\n", A

        }
        if(Memory.Load(Address + 3 + 7*i, LS_WORD) == RanData)
        {
                ++Success;
        }
        else
        {
                printf("Address: 0x%08X Data: 0x%08X | WORD: 0x%08lX\n", A

        }
```