CPE 233 Hardware Assignment 1

Reverse Engineering and Testing ROM

Report by:
Ethan Vosburg (evosburg@calpoly.edu)

January 12, 2024

Table of Contents

1 Project Description **3**

2 Reverse Engineering **3**

2.1 Reverse Engineering Table 3

2.2 Reversed Engineered Program. 3

3 Testing rom **4**

4 Conclusion. **5**

1 Project Description

Reverse engineer the excerpts from the `otter_memory.mem` file is shown below to determine the assembly instructions implemented by this file. This can be done by first creating a table similar to Table 1 from the Example Program. Any other lines from this file can be assumed to be all zeros and disregarded.

2 Reverse Engineering

2.1 Reverse Engineering Table

Below is a table of the machine code and assembly instructions from the `otter_memory.mem` file. The machine code was converted to binary and then to assembly instructions. By converting to binary the structure of instruction could be more easily seen and then specific instructions could be identified and then converted to assembly instructions. The assembly instructions were then written in the table. To verify the assembly instructions, once the assembly was written, it was recompiled and the compared to the original memory file. The assembly instructions were correct and the program was able to be run.

Table 1: Machine Code to Instruction Table

Address	Machine Code (Hex)	Machine Code (Binary)	Assembly Instruction
0000	11000537	00010001000000000000010100110111	lui x10, 69632
0004	00a00f13	00000000101000000000111100010011	addi x30, x0, 10
0008	00051783	00000000000001010001011110000011	jump:lh x15, 0(x10)
000C	41e7da33	01000001111001111101101000110011	sra x20, x15, x30
0010	00fa4633	00000000111110100100011000110011	xor x12, x20, x15
0014	04c52023	00000100110001010010000000100011	sw x12, 64(x10)
0018	ff1ff06f	11111111000111111111000001101111	jal x0, jump

2.2 Reversed Engineered Program

Below is the reverse-engineered program from the `otter_memory.mem` file. The program is a simple loop that shifts the value in `x15` right by the value in `x30`, exclusive ors the result with the original value in `x15`, and then stores the result in `x12` and loops.

Listing 1: Assembly instructions from reverse engineered file

```
        lui     x10, 69632      // load upper immediate
        addi    x30, x0, 10     // add immediate
jump:
        lh      x15, 0(x10)     // load halfword
        sra     x20, x15, x30   // shift right arithmetic
        xor     x12, x20, x15   // exclusive or
        sw      x12, 64(x10)    // store word
        jal     x0,  jump       // jump
```

3 Testing rom

To create a more robust testbench, the `ProgRom_TB` was created. This testbench reads in the `otter_memory.mem` file and then compares the machine code instructions from the file to the machine code instructions from the `ProgRom` module. Once iterating through all of the instructions an overall pass/fail is displayed. The testbench was able to be run and passed. The testbench is shown below.

Listing 2: Assembly instructions from reverse engineered file

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 01/11/2024 10:40:40 PM
// Design Name: ProgRom Testbench
// Module Name: ProgRom_TB
// Project Name: HW1-ProgROM
// Target Devices: Basys 3
// Description: This is a testbench for the ProgRom module that takes in a mem file and checks that the program memory is correct
//
// Revision: 1.0
// Revision 0.01 - File Created
// Additional Comments: Still need to work on dynamic array assignment
//
//////////////////////////////////////////////////////////////////////////////////


module ProgRom_TB();
    // Inputs
    logic PROG_CLK_TB; // 1-bit clock
    logic [31:0] PROG_ADDR_TB; // 32-bit address

    // Outputs
    logic [31:0] INSTRUCT_TB; // 32-bit machine code instruction

    // Logic
    logic pass; // 1 = pass, 0 = fail

    // rom_TB type set up
    (* rom_style="{distributed | block}" *)
   (* ram_decomp = "power" *) logic [31:0] rom_TB [0:16383];

    // Instantiate the Unit Under Test (UUT)
    ProgRom uut (
        .PROG_CLK(PROG_CLK_TB),
        .PROG_ADDR(PROG_ADDR_TB),
        .INSTRUCT(INSTRUCT_TB)
    );

    // Begin simulation code
    initial begin
        // Initialize Logic
        PROG_CLK_TB = 0; // Initialize PROG_CLK_TB
        pass = 1; // Initialize pass to 1

        $readmemh("otter_memory.mem", rom_TB , 0, 16383); // Read in otter_memory.mem file

        // Iterate through rom_TB and compare to INSTRUCT_TB
        // If there is a mismatch, set pass to 0
        // Note: the comparison is hardcoded to stop for this case
        for (int i = 0; i < 7; i++) begin
            PROG_ADDR_TB = 32’h0000_0000 + (i*4); // Request machine code instruction from address
            #20 // Wait for INSTRUCT_TB to be updated

            // Display values for debugging and verification
            $display("rom[%0d]: %h", i, INSTRUCT_TB);
            $display("rom_tb[%0d]: %h", i, rom_TB[i]);

            // Compare INSTRUCT_TB to rom_TB[i]
            if (INSTRUCT_TB == (32'h0000_0000 + rom_TB[i])) begin
                $display("Match rom_TB[%0d] = %h", i, rom_TB[i]);
            end else begin
                $display("No Match rom_TB[%0d] = %h", i, rom_TB[i]);
                pass = 0; // Do not allow the test to pass
            end
            #20;
        end

        // Display overall pass/fail
        if (pass == 1) begin
            $display("OVERALL PASS");
        end else begin
            $display("OVERALL FAIL");
        end
    end

    // Toggle the clock
    always #5 PROG_CLK_TB = ~PROG_CLK_TB;
endmodule
```

Listing 3: TCl Output from ProgRom_TB

```
rom[0]: 11000537
rom_tb[0]: 11000537
Match rom_TB[0] = 11000537
rom[1]: 00a00f13
rom_tb[1]: 00a00f13
Match rom_TB[1] = 00a00f13
rom[2]: 00051783
rom_tb[2]: 00051783
Match rom_TB[2] = 00051783
rom[3]: 41e7da33
rom_tb[3]: 41e7da33
Match rom_TB[3] = 41e7da33
rom[4]: 00fa4633
rom_tb[4]: 00fa4633
Match rom_TB[4] = 00fa4633
rom[5]: 04c52023
rom_tb[5]: 04c52023
Match rom_TB[5] = 04c52023
rom[6]: ff1ff06f
rom_tb[6]: ff1ff06f
Match rom_TB[6] = ff1ff06f
OVERALL PASS
```

4 Conclusion

The ProgRom module was able to be tested and verified to be working correctly. The assembly instructions from the `otter_memory.mem` file were able to be reverse-engineered and then verified by recompiling the assembly instructions and comparing the machine code instructions. All code for this assignment can be found here.