Lab 03: Don't be RISC-Averse!
Jonathan Gusler
2/2/2019

Overview
The purpose of this lab was to generate familiarity with MIPS assembly code and the processes that occur in the hardware of a basic computer. This was done by analyzing code that performed I-type and R-type instructions, and then creating our own instructions based off pseudo code that involved I-type, R-type, and, possibly, jump instructions.

Background
This lab requires knowledge of MIPS assembly language, machine language, familiarity with QUARTUS hardware simulator, VHDL, how a computer fetches, decodes, and executes instructions, and how the data control path works.

Discussion
The instructions to be executed are kept in a file show in Figure 2 in Appendix A. The pseudo code shown in Figure 1 is what needed to be translated into assembly language and then machine code. Both the assembly language and corresponding machine code translation are shown in Figure 4 in Appendix A in the file that is read by QUARTUS. An important component of this QUARTUS hardware simulation is that each register originally contains a value equivalent to the register number.

```
-- Algorithm to add 5+5+5 and store result in memory location 1
START:    counter = 0
          result = 0
ADD:      result = result + 5
          counter = counter + 1
          if counter = 3 branch to DONE
          branch to ADD
DONE:     memory(1) = result
```

Figure 1: Pseudo code to be implemented in machine code.

In the waveform in Figure 3, the clock edge that begins execution of the first "load word" instruction and the clock edge just before register $2 changes to the value 00000002 are the same clock edge and is marked 1 and 2. The clock period during which the beq $1,$2, -4 instruction executes is circled in red. Figure 5 is marked in blue where Result is incremented by 5 and in purple where the final memory write operation takes place.
Question 1: What is the program counter value when the ADD instruction first executes?
Answer: 0x004

Question 2: Once the program begins executing, during which clock cycle is the result of the ADD

operation available for writing into register $1?
Answer: The same clock cycle

Question 3: In the clock cycle following the writing of the result of the ADD operation to register $1, what is the value of the ALU output? Why is it not still FFFFFFFF?
Answer: 00000003, it is calculating and outputting what the memory location to store it is

Question 4: What file contains the MIPS program to be executed? How does the simulator know to use this file?
Answer: Program.mif. On line 33 of IFECTH, which handles instructions, it says init_file Program.mif, which I assume means initialize usage of the file. When implementing our own code, the file is Lab03Program.mif and the code is changed to reflect this correspondingly.

Question 5: What file is used to initialize the memory? Where in the code is this file specified?
Answer: DMEMORY, line 28 init_file dmemory.mif

Question 6: What is the meaning of the following line of VHDL?

```
Val <= X"10" WHEN Reset = '1'
```

Answer: Val gets the value of the 10^{th} bit of X when Reset is HIGH

Question 7: Which of the 32 bits in a MIPS instruction contain the opcode? Looking at Figure 14.1, what are the opcode bits used for?
Answer: The first 6. They peel off and go to the CONTROL which orchestrates the different components.

Question 8: Was it necessary to recompile the project when only the MIPS program file had changed?
Answer: Yes

Question 9: Explain beq offsets. Discuss the offset value that appears in the mnemonic portion of the instruction, as well as the value encoded in the 32-bit MIPS instruction word itself.
Answer: The offset changes the PC relative to the value of the PC for the instruction after the branch. In the mnemonic, the value is in bytes, but in the instruction, it is in units of words.

Analysis and Results
Figure 5 in Appendix A shows the waveform of the execution of the new program. As can be seen in the waveform the program, as intended, branches back to ADD until Result is 15. Once it does equal 15, the program proceeds to execute the next branch instruction to DONE where it stores Result in data memory and the program ends.

Appendix A

```
1   -- MIPS Instruction Memory Initialization File
2   Depth = 256;
3   Width = 32;
4   Address_radix = HEX;
5   Data_radix = HEX;
6   Content
7   Begin
8   -- Use NOPS for default instruction memory values
9       [00..FF]: 00000000; -- nop (sll r0,r0,0)
10  -- Place MIPS Instructions here
11  -- Note: memory addresses are in words and not bytes
12  -- i.e. next location is +1 and not +4
13      00: 8C020000;    -- lw $2,0 ;memory(00)=55
14      01: 8C030001;    -- lw $3,1 ;memory(01)=AA
15      02: 00430820;    -- add $1,$2,$3
16      03: AC010003;    -- sw $1,3 ;memory(03)=FF
17      04: 1022FFFF;    -- beq $1,$2,-4
18      05: 1021FFFA;    -- beq $1,$1,-24
19  End;
20
```

Figure 2: Code for the original program.

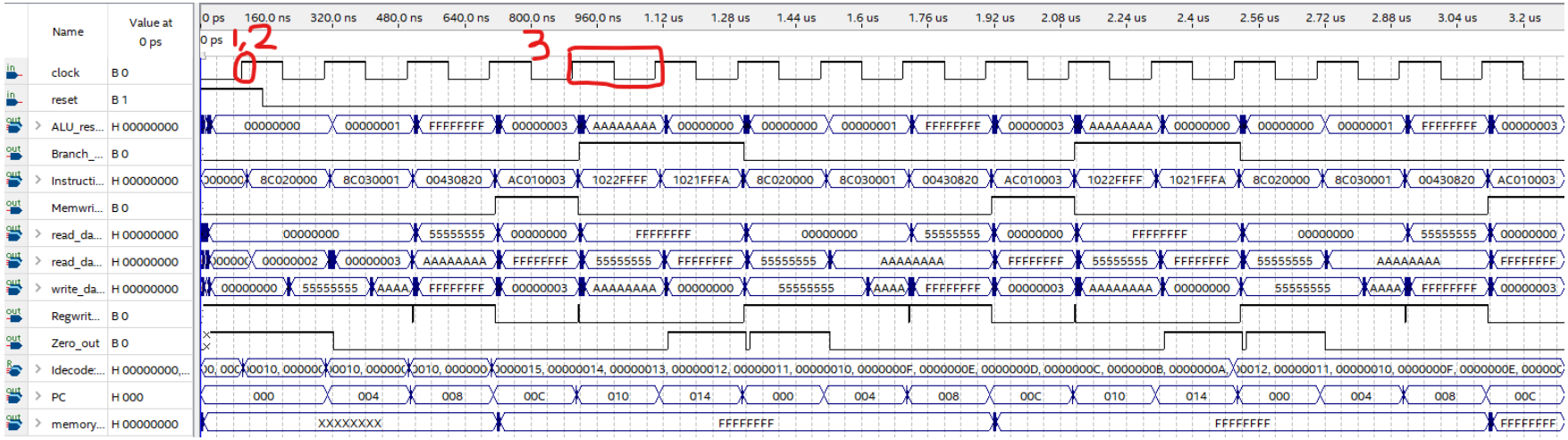

Figure 3: Waveform of original program.

```
1   -- MIPS Instruction Memory Initialization File
2   Depth = 256;
3   Width = 32;
4   Address_radix = HEX;
5   Data_radix = HEX;
6   Content
7   Begin
8   -- Use NOPS for default instruction memory values
9       [00..FF]: 00000000; -- nop (sll r0,r0,0)
10  -- Place MIPS Instructions here
11  -- Note: memory addresses are in words and not bytes
12  -- i.e. next location is +1 and not +4
13      00: 00001020;    -- add $2,$0,$0      ; set counter to 0
14      01: 00002020;    -- add $4,$0,$0      ; set result to 0
15      02: 00852020;    -- add $4,$4,$5      ; add 5 to result
16      03: 00411020;    -- add $2,$2,$1      ; add 1 to counter
17      04: 10430001;    -- beq $2,$3,1       ; go to DONE if result is 15
18      05: 10C6FFFC;    -- beq $6,$6,-4      ; go back to ADD if result isn't 15
19      06: AC040000;    -- sw $4, 0($0)      ; store result in memory(0)
20  End;
```

Figure 4: Code for the new program.

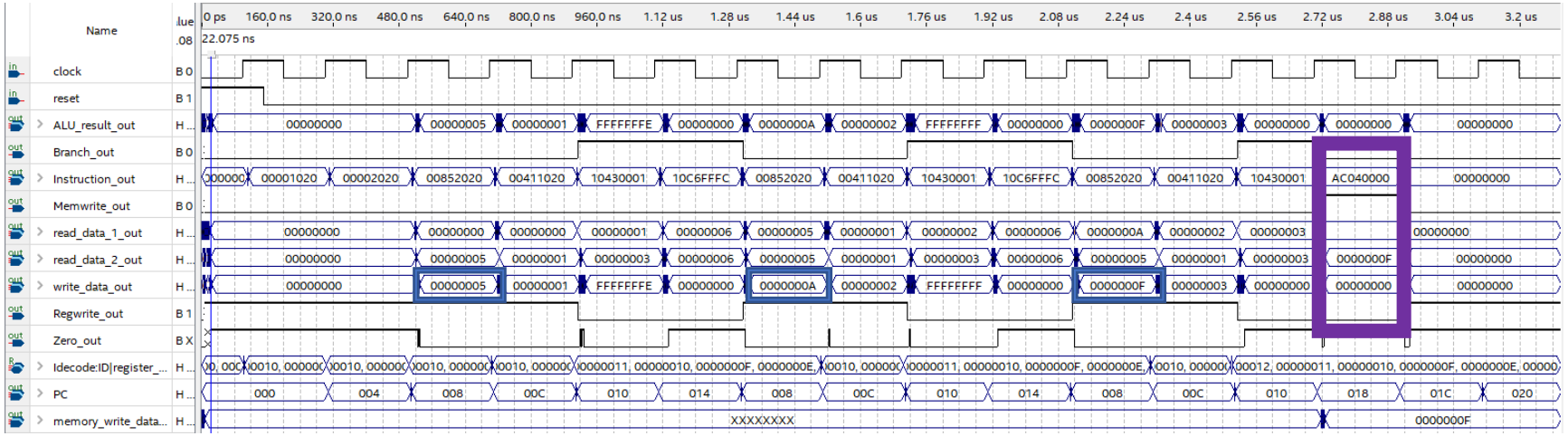

Figure 5: Waveform for the new program.