CprE 381 – Computer Organization and Assembly Level Programming

Lab-04

[Note from Joe: this is going to be the final individual lab of the semester. My hope is that after finishing this lab you will be sufficiently proficient in VHDL to be able to design the rest of the MIPS processor components more independently with your project teams. If you are not yet in a project team let your TA or me know immediately.]

0) Prelab. In the attached *Lab-04.zip*, there is a directory called *prelab/*. Open up the included *dmem_waveforms.html* document (if you can't see the three waveform images make sure you have unzipped the folder first). Based on the waveforms, provide a description in your own words of how this component operates. This can be in the form of a textual description, flow chart, or state machine. Note that these waveforms do not exactly correspond to the *mem.vhd* component we will be using in this lab.

1) Several instructions in the MIPS ISA implicitly require **sign** or **zero extension** of 8-bit or 16-bit values such that these values can be operated on using a 32-bit ALU and/or stored in a 32-bit register file. Provide your solution to this problem (VHDL code, simulation waveforms) in a folder called 'P1/'. *[This is "more of the same" from what you've been doing both in terms of VHDL, Modelsim simulation, and should not take you more than 30 minutes.]*

- **(a)** What are the MIPS instructions that require some value to be sign extended? What are the MIPS instructions that require some value to be zero extended? *[We've covered some of this in class, but if you are unsure, the answers can be found in P&H MIPS "Green Sheet". Don't just rely on where it says 'SignExtImm' or 'ZeroExtImm'.]*

- **(b)** Given your solution to part a), what are the different "extender" components that would be required by a MIPS processor implementation? *[Hint – considering the Core Instruction Set on the "Green Sheet", there are two different components required. Verify with your TA before proceeding.]*

- **(c)** Implement the two 16-bit to 32-bit MIPS extender components using any VHDL style you prefer. These can be implemented as two separate entities with no control signals, or as a single entity with a control bit or a generic specifying "sign" versus "zero" extension. *[This is a relatively simple task using either "for/generate" or the "others" VHDL construct from Lab-03. There are many different ways to do this; if your code looks complex for this part you are probably overthinking the problem.]*

- **(d)** Use Modelsim to test your extender components to make sure they are working as expected, and include waveform screenshots in your report PDF.

2) The way **Memory Structures** are designed is typically very technology-dependent, and consequently while there are simple methods for implementing generic (behavioral) memories in VHDL, for a specific memory design most computer architects either follow technology-

dependent coding styles so that synthesis tools can infer the underlying structure, or they select and configure a pre-designed soft Intellectual Property (IP) core. For our MIPS instruction and data memories we will follow the latter approach. Provide your solution to this problem (VHDL code, simulation waveforms) in a folder called 'P2/'.

(a) In your Lab-04 folder should be a behavioral implementation of a generic memory module (*mem.vhd*) and associated initialization file (*dmem.hex*). The *dmem.hex* file provided as part of this lab contains only a few simple memory initialization values. Before continuing, modify this file to contain the following 32-bit values first starting at address 0x0 in memory: 1, 2, -3, 4, -5, 6, -7, 8, -9, 10. *[The included* dmem.hex *file contains some hints, but be aware that the* mem.vhd *RAM module we have is word-addressable and not byte-addressable as MIPS expects.]*

(b) As you will find out, even though the majority of the code is provided for you, working with IP is not always a trivial task. The first issue is that this module comes with no documentation (unfortunate, but the price was too good to pass up). Read through the *mem.vhd* file, and based on your understanding of the VHDL implementation, provide a 2-3 sentence description of each of the individual ports (both generic and regular). For example, what is port "q" for and when does it become valid? Confirm your understanding with the TA before proceeding.

(c) Create a VHDL testbench (*tb_dmem.vhd*) that i) instantiates the *mem* module as data memory (labeled as *dmem*), ii) reads the initial 10 values stored in memory, iii) writes those same values back to consecutive locations in memory starting at 0x100, and then iv) reads those new values back to ensure they were written properly. As always, include waveform screenshots in your report PDF. *[Note that in order to load the dmem.hex file, you will need to start a simulation in Modelsim and use the following command from a .do file or from the command window:*

```
mem load -infile dmem.hex -format hex /tb_dmem/dmem/ram
```

This command assumes 1) you have disabled optimization when you started simulation, 2) are currently in the same directory as dmem.hex, and 3) you've labeled your memory dmem and your testbench tb_dmem.]

(d) In your writeup, briefly describe how the waveforms for this *mem.vhd* module differ from those that you analyzed as part of the pre-lab.

3) Running the following **Second MIPS Application** is very much possible given the components you have already created in this and the previous week's lab. Provide your solution to this problem (VHDL code, simulation waveforms) in a folder called 'P3/'.

(a) As discussed in class, incorporating a data memory module into the MIPS processor requires additional control signals associated with store and load instructions. In order to support the two 32-bit integer load and store instructions as listed in P&H chapter 2, what control signals will need to be added to the simple processor from Lab-03? How do these control signals correspond to the ports on the *mem.vhd* component analyzed in problem 2)?

(b) Draw a schematic of a simplified MIPS processor consisting only of the base components used in Lab-03, the extender component described in problem 1), and the data memory from problem 2). This design should only require inputs for a clock, the control signals, the three register address ports, and a 16-bit immediate value.

(c) Implement this simplified MIPS processor using structural VHDL. Create a VHDL testbench to demonstrate that your design can generate the correct value when "running" the following code. *[Similar to Lab-03, you do not have to assemble these instructions into their proper MIPS machine language equivalents. Instead, determine what values the inputs to your design would correspond to for these instructions.*]

```
addi  $25, $0, 0          # Load &A into $25
addi  $26, $0, 256        # Load &B into $26
lw    $1,  0($25)         # Load A[0] into $1
lw    $2,  4($25)         # Load A[1] into $2
add   $1,  $1, $2         # $1 = $1 + $2
sw    $1,  0($26)         # Store $1 into B[0]
lw    $2,  8($25)         # Load A[2] into $2
add   $1,  $1, $2         # $1 = $1 + $2
sw    $1,  4($26)         # Store $1 into B[1]
lw    $2,  12($25)        # Load A[3] into $2
add   $1,  $1, $2         # $1 = $1 + $2
sw    $1,  8($26)         # Store $1 into B[2]
lw    $2,  16($25)        # Load A[4] into $2
add   $1,  $1, $2         # $1 = $1 + $2
sw    $1,  12($26)        # Store $1 into B[3]
lw    $2,  20($25)        # Load A[5] into $2
add   $1,  $1, $2         # $1 = $1 + $2
sw    $1,  16($26)        # Store $1 into B[4]
lw    $2,  24($25)        # Load A[6] into $2
add   $1,  $1, $2         # $1 = $1 + $2
addi  $27, $26, 512       # Load &B[128] into $27
sw    $1,  -4($27)        # Store $1 into B[255]
```