Computer Architecture II

Computer Engineering Department

An-Najah National University

Dr. Suleiman Abu Kharmeh

Assignment #2: Understanding MIPS Pipeline CPI (Parts 2 and 3)

Assignment date: Wednesday 03/November/2021

Due date: Thursday 11/Novermber/2021

Submission:

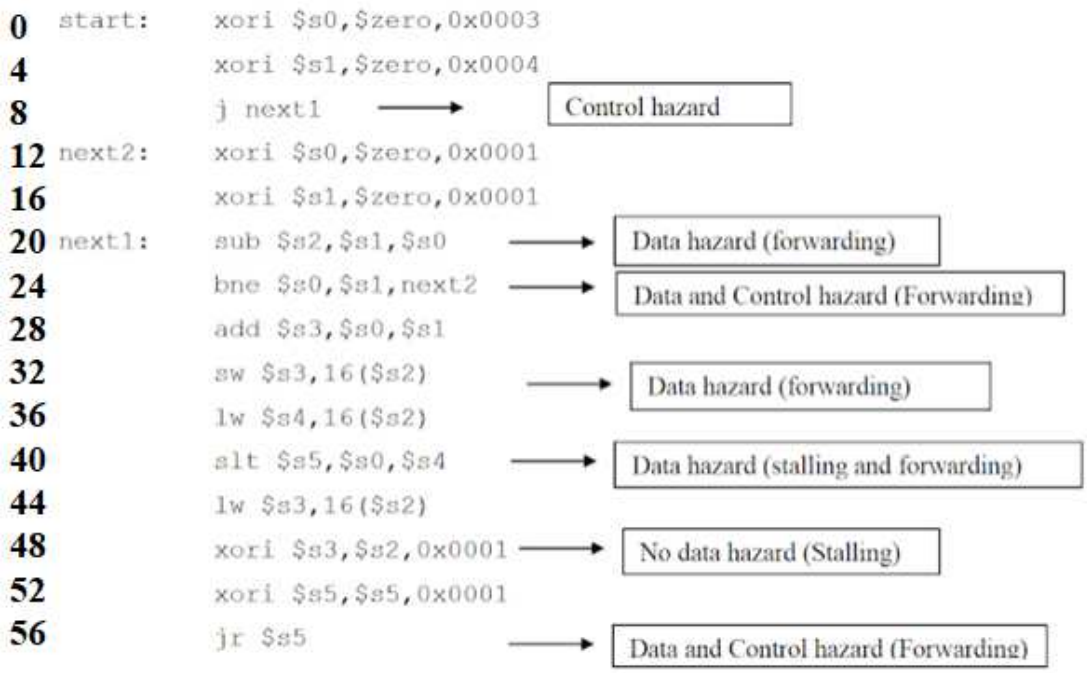
- **Word/PDF report of all the details:**
  - **Explanation of the results.**
  - **Screenshot(s) of the waveform.**
- **Complete project directory (compressed file) including:**
  - **All Verilog files.**
  - **All memory initialization files.**
  - **All ModelSim project file.**
  - **All assembly files used (.asm).**
- **Work individually on this assignment please.**
- **Late submissions would incure 5% penalty per day (maximum of 7 days late).**

Part 1 (for reference):
For the PIPS pipeline verilog implementation available at:

https://www.fpga4student.com/2017/06/32-bit-pipelined-mips-processor-in-verilog-1.html

Perform the following tasks:

1. Save the above MIPS pipeline verilog modules a reasonable set of verilog files.
    a. Ideally (but not necessarily) each module in a seperate verilog file. The file name would match the module name. You should have 37 design files in total.
    b. Alternatively save each group of modules in a seperate file as they are grouped on FPGA4Student webside. You could end up in about 20 files.
    c. DO NOT save all modules in one big verilog file. That is bad practice.
2. Create a new ModelSim project and add all the files you have just created in step 1 to that project.
3. Save the following instruction sequence (also available on the website above) in a text file called ”instr.txt” as a sequence of zeros and ones in ASCII:



This sequence should translate to the following text in the ”instr.txt” file:

```
0   00111000000100000000000000000011
4   00111000000100010000000000000100
8   00001000000000000000000000000101
12  00111000000100000000000000000001
16  00111000000100010000000000000001
20  00000010001100001001000000100010
24  00010110000100011111111111111100
28  00000010000100011001100000100000
32  10101110010100110000000000010000
36  10001110010101000000000000010000
40  00000010000101001010100000101010
44  10001110010100110000000000010000
48  00111010010100110000000000000001
52  00111010101101010000000000000001
56  00000010101000000000000000001000
```

This instr.txt file will be used for initializing the instruction memory from which the processor will start executing.

4. Explain how the above sequence should work. Run it using the SPIM or MARS simulators. Take a snapshot of the simulator state at the last instruction (before it executes JR $S5) and put it in the report.

   Note that standard MIPS simulators would not start data and code from address 0, they also do not assume the instruction and data memories are seperate memories. For the example above to work in SPIM/MARS, you need to modify two instructions slightly as highlighted below (and also add the section highlighted at the top). As a result, you will get slightly different values in $s2 and $s5 at the end of the iteration compared to what you would get from ModelSim. Try to understand/explain the difference.

```
.data                #data section
test: .word 256
.text                #code section
start:          xori $s0, $zero, 0x3
                xori $s1, $zero, 0x4
                j next1
next2:          xori $s0, $zero, 0x1
                xori $s1, $zero, 0x1
next1:          la  $s2, test                   # modelsim: sub $s2, $s1, $s0
                bne $s0, $s1, next2
                add $s3, $s0, $s1
                sw $s3, 16($s2)
                lw $s4, 16($s2)
                slt $s5, $s0, $s4
                lw $s3, 16($s2)
                xori $s3, $s2, 0x1
                la $s5, start                   # modelsim: xori $s5, $s5, 0x1
                jr $s5
```

5. Back to ModelSim: Start the simulation by loading the MIPSStimulus module into the simulator.

   Hint: the MIPS pipelined/CPU/Design is instanciated in a TestBench module called **MIPSStimulus.** Use this module as top-level module for simulation.
6. Simulate your design for about 120 NS.
7. Observe the changes to the registers (MIPSStimulus/myMIPS/Register_File/reg[0:31]) during this time and relate the changes to the instruction sequence above and your observations from the instruction set simulator (SPIM or MARS).
8. Explain your observation using a waveform snapshot and show how each instructon changes the relevant register.

Part 2 (50%):

You should have realized from analyzing the code above that it is an infinte loop that executes 17 instructions in each iteration. Convert the code to run a specific number of iterations. Use the rightmost two digits of your student ID for the number of iterations. You are free to update existing instructions or add new ones as you see appropriate. You just need to keep the original code as it is. Use $S6 as your counter. Replace ”JR $S5” with a conditional branch instruction which with ”start” as the target. Then add an infinite loop after your main loop that does nothing. E.g.:

```
halt:    j halt
```

Demonstrate your simulation, specially the changes to the loop counter ($S6) and the changes to the PC near the end of the main loop and start of the infinite loop.

Note about address changes:

Be careful when porting the binary representation of the instructions from Mars to ModelSim, you need to carefully check all addresses used in any absolute jump instruction (J, JR) and also for load and store instructions. Those you must double check that they reference the correct data/code address.

Part 3 (50%):

Analyze the performance of your final code during the execution of the main loop iteration:

- How many cycles does it take to execute one iteration?
- How many instructions are excuted per iteration?
- What is the average CPI per iteration?
- Does this match your expectations? Why?

Hint: ignore any cycles needed to fill the pipeline at the start. Just perform the analysis after the first iteration.

Hint2: first observe the following delays in the original sequence:

- J and LW consumed 2 cycles
- JR and BNE both consumed 3 cycles
- All other instructions consumed 1 cycle
- There were 17 instructions executed per iteration (static analysis of the code)
- There were 23 cycles consumed per iteration (observed in simulation between restarts of the Program Counter (PC) register)
- On average the CPI was 1.35 (1.35 cycles consumed per instruction)