Amit Nijjar

A11489111

CSE 141L

Due 5/4/16

Lab 2

Q1:

Make sure that you have read and understood the code before answering these questions.

* The opcode space in the Vanilla instruction format is fixed. How many more instructions beyond the basic instruction set can be supported?
* Record the reported Fmax, number of registers used, number of combinational functions used, and the number of memory bits used before making any changes to the design.

Add a left bitwise rotate instruction (ROL) to alu.sv and add the mapping to definitions.sv (HINT: refer to the [manual](https://drive.google.com/file/d/0B4jzMMK0wZ_9cDhRRndfY1pxeVU/view) and understand why the bitwise shift instructions are defined the way they are). You will want to think about what other parts of the Vanilla core need to be updated in order for the ROL instruction to work properly.

32 instructions can be supported

Fmax = 51.4 MHz

Registers used = 2069

Combinational functions = 4347

Memory bits = 16384

Q2:

Make sure that you have read and understood the code before answering these questions.

* The Vanilla core has a separate memory for data and instructions. What are the advantages of this type of architecture?
* What is the maximum size of an assembly program in terms of # of instructions?
* Is reading from the instruction memory synchronous?

Separate data and instructions make the processor more efficient. Fewer instructions means that it is faster

1024 instructions

Yes, it is synchronous

Q3:

* How many cycles will an operation that accesses data memory take in the best case?

2

Q4:

Make sure that you have read and understood the code before answering these questions (a few of the details of how the network interfaces to the core may be unclear at this point).

* Draw a state machine for the Vanilla core, including the signals that cause a state transition.
* Under what conditions does the execution of the core stall?
* Draw a module hierarchy for the Vanilla core.
* In MIPS the program counter advances by four (i.e. PC + 4) if a branch does not occur. What is the program counter step size for the Vanilla core?

Read data

Read instructions

start

pch

Net adder

Write data

There is a stall when the previous computation is not done yet. When two r type instructions follow each other

Step size = 1

\_\_\_\_\_\_\_\_\_\_\_\_\_

Core

Instruction memory

Decode

Register

Alu

Data memory

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q5:

* The assembler will need to recognize your new ROL instruction before you can use it in your \*.asm files. Add a mapping for the left bitwise rotate instruction (ROL) to the assembler.
* Write a unit test for the new instruction in the style of the other tests in tester.asm, add it to the tester.asm file, and assemble it (follow instructions from Lab 1 if you forget how).
* Run the tester.asm program on the Modelsim simulator and verify that functional (aka behavioural) simulation is successful
* List the reported Fmax of your design before and after adding ROL.
* List the cycle time of your design before and after adding ROL. Show your work.
* List the number of registers used, the number of combinational functions used, and the number of memory bits used before and after adding ROL.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | fmax | c-time | reg | functions | Bits |
| Before | 54.1 | 18.4 | 2069 | 3920 | 16384 |
| After | 55.7 | 17.9 | 2069 | 4086 | 16384 |

i/fmax = 18 ns; 17.9 ns

Q6:

Up to this point we have given you the scripts to compile and simulate your designs in ModelSim. Use these as a reference to fill in the compile.tcl and sim.tcl scripts to compile and start the simulations for simple\_core\_tb. Note that this must work for our auto-grader to run your code! Your sim.tcl script must include at least one set of waveforms. You may pick any radix that you want.

Changed code

Q7:

Add both pipecuts to your design as described above and once your code is in a working state (passes simple\_core\_tb), answer the following questions:

* What signals must you pass through across the first pipecut? The second?
* List the number of registers used, the number of combinational functions used, and the number of memory bits used before and after adding your pipecuts.
* List the reported Fmax of your design before and after adding your pipecuts.
* List the cycle time of your design before and after adding your pipecuts. Show your work.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | fmax | C time | reg | funct | Bits |
| Before | 260MHz | 3.8ns | 35 | 25 | 0 |
| After | 307MHZ | 3.2ns | 50 | 35 | 0 |

Q8:

* Describe all the new instructions that your team has implemented.

XOR and ROR (right rotate)

Q9:

* List the reported Fmax of your design before and after optimization.
* List the cycle time of your design before and after optimization. Show your work.
* List the number of registers used, the number of combinational functions used, and the number of memory bits used before and after optimization.

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Fmax | 55.7 MHz | 54.1 MHZ |
| Cycle time | 17.9ns | 18.5ns |
| registers | 2069 | 2069 |
| Functions | 4086 | 4086 |
| bits | 16384 | 16384 |

Q10:

* How many cycles did it take to find a bitcoin before and after optimization?
* What was your hash (nonce) rate in hashes per cycle before and after optimization?

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Cycles | 361890 | 168534 |
| nonce | 24 micro noches | 53 micro nonces |

Q11:

* How many instructions did it take to find a bitcoin before and after optimization?
* What was your hash rate in hashes per instruction before and after optimization?

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Instructions | 350960 | 157425 |
| Hash rate | 25 micro hashes | 57 micro hashes |

Q12:

* What is the execution time (# of cycles x cycle time) to find a bitcoin before and after optimization?
* What was your hash (nonce) rate in hashes per second before and after optimization?
* What is your speedup (baseline execution time/optimized execution time)?

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Time | 361ps | 168ps |
| hashes | About 24670 | About 53458 |

361/168 = 2.14 faster