Project Checkpoint 3

Processor

Logistics

This is the third Project Checkpoint for our processor.

- Due: **Thursday, October 31, 2019** by **11:59 PM EST**
- Late policy can be found on the course webpage/syllabus. **Be sure to start this assignment early.**

Introduction

Design and simulate a single cycle 32-bit processor, as described in class, using Verilog. A skeleton has been built for you, including many of the essential components that make up the CPU. This skeleton module includes the top-level entity (“skeleton”), processor (“processor”), data memory (“dmem”), instruction memory (“imem”), and regfile (‘regfile”).

Your task is to generate the processor module. Please make sure that your design:

- Integrates your register file and ALU units
- Properly generates the dmem and imem files by generating Quartus syncram components

We will post clarifications, updates, etc. on Piazza so please monitor the threads there.

Module Interface

Designs that do not adhere to the following specification will incur significant penalties.

Please follow the base code **and read through it** in your Github repository. The processor includes a skeleton file that serves as a wrapper around your code. **The skeleton is the top-level module** and it allows for integrating all of your required components together.

Permitted and Banned Verilog

Designs that do not adhere to the following specifications will incur significant penalties.

No "megafunctions" or built-in modules.

Use only structural Verilog such as:

- `and and_gate(output_1, input_1, input_2 ... );`

In general, **do not use** syntactic sugar (behavioral Verilog) like:

- `if (some_condition)`
- `switch`

except in constructing your DFFE or clock dividers (i.e. you can use whatever verilog you need to construct a DFFE or clock divider).

However, **feel free to use** the following syntactic sugar and primitives:

- `assign ternary_output = condition ? if_true : if_false;`
- `for` loops and `genvars`
- `assign` statements that simply assign one wire to another
    - `assign x = y;`
    - `assign x[15:14] = y[9:8];`
- `&, &&, |, ||, <<, <<<, >>, >>>`

Other Specifications

Designs that do not adhere to the following specifications will incur significant penalties.

Your design must operate correctly with a **50 MHz clock**. You may use **clock dividers (**see here for background**)** as needed for your processor to function correctly. Also, please remember that we are ultimately deploying these modules on our FPGAs. Therefore, when setting up your project in Quartus, be sure to pick the correct device (check the tutorial if you are unsure).

1. Memory rules:
    a. Memory is word-addressed (32-bits per read/write)
    b. Instruction (imem) and data memory (dmem) are separate
    c. Static data begins at data memory address 0
    d. Stack data begins at data memory address 2^16-1 and grows downward
2. After a reset, all register values should be 0 and program execution begins from instruction memory address 0. Instruction and data memory is not reset.

Register Naming

We use two conventions for naming registers:

- `$i` or `$ri`, e.g. `$r23` or `$23`; this refers to the same register, i.e. register 23

Special Registers

- `$r0` should always be zero
    - Protip: make sure your bypass logic handles this
- `$r30` is the status register, also called `$rstatus`
    - It may be set and overwritten like a normal register; however, as indicated in the ISA, it can also be set when certain exceptions occur
    - **Exceptions take precedent** when writing to `$r30`
- `$r31` or `$ra`; is the return address register, used during a `jal` instruction
    - It may also be set and overwritten like normal register

Submission Instructions

Designs which do not adhere to the following specifications cannot receive a score.

Use the autogenerated Github repository and commit/push into it. **The last commit on master before the deadline** on your Github repository will be considered your submission. Be sure to include all of your code in this submission. This project checkpoint also **requires a design report**, explained below.

Submission Requirements

- A repository containing your code

- A design report

Submitting for Grading

- Push to Github in your repository
- Complete design report via Google Form

Grading

Grading will be different from what it has been for the other project checkpoints. Your grade will consist of:

- A design report
- The correctness of your processor

Design Report

- The design report will take the form of a Google Form found here
- The report will serve as a way to explain your processor design

Correctness

- Your code will be run through AG350 as it has in previous project checkpoints
- A grading skeleton file, imem, and dmem will be swapped in for yours

ISA

Instruction	Opcode (ALU op)	Type	Operation
add $rd, $rs, $rt	00000 (00000)	R	$rd = $rs + $rt $rstatus = 1 if overflow
addi $rd, $rs, N	00101	I	$rd = $rs + N $rstatus = 2 if overflow
sub $rd, $rs, $rt	00000 (00001)	R	$rd = $rs - $rt $rstatus = 3 if overflow
and $rd, $rs, $rt	00000 (00010)	R	$rd = $rs & $rt
or $rd, $rs, $rt	00000 (00011)	R	$rd = $rs | $rt
sll $rd, $rs, shamt	00000 (00100)	R	$rd = $rs << shamt
sra $rd, $rs, shamt	00000 (00101)	R	$rd = $rs >>> shamt
sw $rd, N($rs)	00111	I	MEM[$rs + N] = $rd
lw $rd, N($rs)	01000	I	$rd = MEM[$rs + N]
j T	00001	JI	PC = T
bne $rd, $rs, N	00010	I	if ($rd != $rs) PC = PC + 1 + N

jal T	00011	JI	$r31 = PC + 1, PC = T
jr $rd	00100	JII	PC = $rd
blt $rd, $rs, N	00110	I	if ($rd < $rs) PC = PC + 1 + N
bex T	10110	JI	if ($rstatus != 0) PC = T
setx T	10101	JI	$rstatus = T
custom_r $rd, $rs, $rt	00000 (01000 - 11111)	R	$rd = custom_r($rs, $rt) (For use on Final Project - **not needed for this checkpoint**)
custom ...	xxxxx+	X	Whatever custom instructions you need for your Final Project - **not needed for this checkpoint**

Instruction Machine Code Format

Instruction Type	Instruction Format
R	Opcode [31:27] | $rd [26:22] | $rs [21:17] | $rt [16:12] | shamt [11:7] | ALU op [6:2] | Zeroes [1:0]
I	Opcode [31:27] | $rd [26:22] | $rs [21:17] | Immediate (N) [16:0]
JI	Opcode [31:27] | Target (T) [26:0]
JII	Opcode [31:27] | $rd [26:22] | Zeroes [21:0]

ISA Clarifications

1. I-type immediate field [16:0] (N) is signed and is sign-extended to a signed 32-bit integer
2. JII-type target field [26:0] (T) is unsigned. PC and STATUS registers' upper bits [31:27] are guaranteed to never be used