Project Checkpoint 5

Full Processor -- Full Processor

Logistics

This is the second checkpoint for our processor. We will post clarifications, updates, etc. on Sakai and Ed.

- Due: **Friday, November 25, 2022, by 11:59 PM** (Duke Time).
    - Late policy can be found on the course webpage/syllabus
- Collaboration: you have to form a group of two or three. It's recommended to keep the same group until the last project.

Introduction

Finish designing and simulating your single-cycle 32-bit processor based on the last checkpoint, as described in class, using Verilog.

For this checkpoint, you are required to implement ***J-type*** instructions in addition to your submission in Checkpoint 4.

Module Interface

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

Please follow the base code in the cpuone-base-master folder. It 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. Please double-check your code compiles with the skeleton set as the top-level entity before submission.

Grading

Grading will be different from previous project grading methods. Your grade will be determined by a design report and the correctness of your processor.

Design Report

- Everyone should complete **a design report** via Google Form (click here). You should submit it individually without discussing it with others. The report will explain the implementation method of your processor design.

Processor Design

- Your code will be run and graded based on correctness. Please make sure you have instantiated all the modules giving proper names or your code may not work in our environment.
- A grading skeleton file, imem, and dmem will be swapped with yours
- Your skeleton module will take in a 50 MHz clock (and reset). Your skeleton module needs the four clocks (imem_clock, dmem_clock, processor_clock, and regfile_clock).

Please submit your regrading request on Gradescope within one week after the grade is published.

Permitted and Banned Verilog

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

No "megafunctions."

- *Tip: think about whether your codes can specify only one design!*

Use structural Verilog like:

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

Not allowed to use SystemVerilog or syntactic sugar like:

- `+, -, *, /, %, **, ==, >=, &&, ||, !, <<, <<<, etc`
- ***if*, *else*,** and ***case*** statements, ***for*** loop, etc

except in provided modules. Please do not change the name of provided modules to allow the style checker to bypass the behavioral codes in the modules. You can choose between your own designs and provided modules for the implementation of the corresponding components.

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

- **`Bitwise not(~)`**
- **`assign ternary_output`** `= cond? High : Low;`
  - The ternary operator is a simple construction that passes on the “High” wire if the cond wire is asserted and “Low” wire if the cond wire is not asserted
- **`generate if, generate for,`** and/or **`genvar`**
  - It could reduce the repeated lines but maintain the structural design

Any expression to specify the range, e.g., a[(i+24)%7]

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, when setting up your project in Quartus, make sure to pick the correct device (mentioned in Recitation 1).

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.

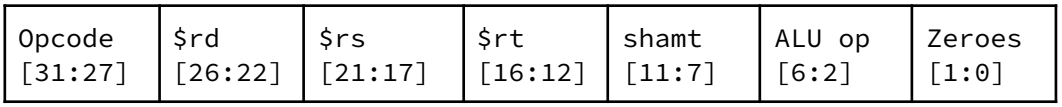
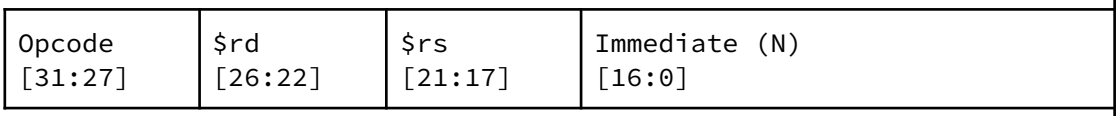
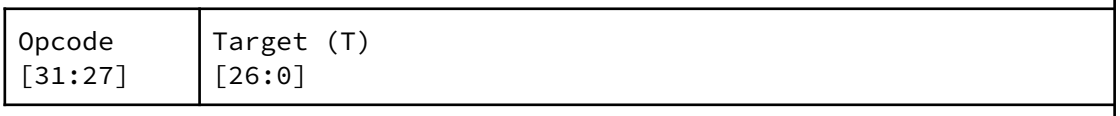
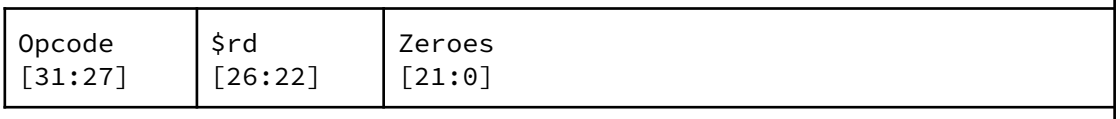
When using **Gradescope**, please submit **one .zip file** and the file should include your code and a README.md file. For **Github** submission, click ‘connect GitHub’, link your account, and select the correct repo and branch you want to submit.

- **One group should submit only one work.** Select ‘group member’ at the bottom right of the submission page on Gradescope. Make sure you add all group members each time you resubmit. Group members should be able to see the same submission.
- Submitted files should include a README.md file and **all necessary *.v modules** to execute your processor. The autograder will read and examine all .v files in the .zip file; therefore, you may be able to include subdirectories but you should be aware that if you submit unnecessary .v files it could cause compile errors.
- **Make sure the name of all testbench files ends with ‘_tb.v’**; otherwise, it will be involved in the style check and negatively affect your submission grade.
- A README.md (written in markdown, Github flavor) should include
    - **Your and your partner’s name and netID**
    - A simple text description of your design implementation and, if there are bugs or issues, descriptions of what they are and what you think caused them.

ISA and Clarifications

Instruction	ALU Opcode	Type	Operation
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 ...	xxxxx+	X	Custom instructions if needed - **not required for this checkpoint**

Instruction Machine Code Format

Instruction Type	Instruction Format
R	
I	
JI	
JII	

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