Project Checkpoint 4

Simple Processor -- R-type and I-type

Logistics

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

- Due: **Tuesday, October 29, 2024, 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. It's recommended to keep the same group until the last project.

Introduction

In this and the next checkpoints, you will design and simulate a single-cycle 32-bit processor, 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

For this checkpoint, you are required to implement some basic functionalities of a processor. Specifically, you will implement the following ***R-type and I-type*** instructions: ***`add`, `addi`, `sub`, `and`, `or`, `sll`, `sra`, `sw`, `and` `lw`.*** **DO NOT** implement J-type (and other I-type) instructions in this checkpoint. In the next checkpoint, you will add other instructions to be supported by your processor.

Module Interface

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

Please follow the provided basecode in the cpuone-base directory. Do not make any modifications to the interface of processor or regfile. The basecode 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 make sure your code compiles with the skeleton set as the top-level entity before submission.

Permitted and Banned Verilog

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

You can use

1. Ternary assign: `assign out = cond ? high : low;`(cond, high, low must be wire(s) (or input/output ports), you should not write an expression in cond)
    1. For example, assign data_result = (ctrl_ALUopcode == 2'b00000) ? Add_result : Sub_result; You cannot use '==' here, because you should not write an expression in cond.
2. Primitive instantiation: `and (out, in1, in2)`
3. Bit-slice, Bit-repeat, and Bit-assemble: `assign c = {5{1'b0}, a[2:0]}`
4. ***Bitwise logic operation:*** `~, &, ^, |`
5. Generate Blocks: generate if, generate for, genvar (This is a tutorial if you do not know them: https://fpgatutorial.com/verilog-generate/)
6. Any expression you like **inside the range specifier**: `a[i*15+36/2-13%2]`
7. Parameters: `parameter a=0; localparam b = a*2;`

You cannot use

1. Behavioral Description Structures: `if ... else ... for...` (This is a loop, not the generate for, which is allowed.) `case ...`
2. Megafunctions outside the range, generate control or parameter expressions: `+, -, *, /, %, **, ==, >=, <=, &&, ||, !, <<, <<<, >>, >>>, etc`
3. SystemVerilog

except in constructing your DFFE and clock dividers (i.e., you can use whatever you need to construct the designs). Please name the DFFE module file ‘dffe.v’ to allow the style checker to bypass your DFFE implementation, or ignore the violation message when it points to your DFFE or clock divider lines. If you decide to use the reference alu/regfile we provided, do not change the file name.

Grading

Grading will be different from previous project grading methods:

- Your code will be run as previous project checkpoints have been and grade is 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 must output the four clocks in order to receive credit (imem_clock, dmem_clock, processor_clock, and regfile_clock).

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

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 this link for background**)** as needed for your processor to function correctly. Also, in the setup of your project in Quartus, make sure to pick the correct device (designated 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 memories are 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.
- **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 before submission. 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.
- **Please do not include testbench files in your submission.**
- A README.md (written in markdown, Github flavor) should include
    - Your name and netID,
    - A text description of your design implementation (e.g., "I used X,Y,Z to ..."),
    - A brief one-sentence functionality description of each self-designed module.
    - If there are bugs or issues, descriptions of what they are and what you think caused them.

- Reminder: Basically, it is a general description of your design and does not exceed 1 page. However, descriptions that are too simple (e.g., contain only a few keywords) will receive a grade deduction.

ISA

Instruction	ALU Opcode	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 (**not required for this checkpoint**)
bne $rd, $rs, N	00010	I	if ($rd != $rs) PC = PC + 1 + N (**not required for this checkpoint**)
jal T	00011	JI	$r31 = PC + 1, PC = T (**not required for this checkpoint**)
jr $rd	00100	JII	PC = $rd (**not required for this checkpoint**)
blt $rd, $rs, N	00110	I	if ($rd < $rs) PC = PC + 1 + N (**not required for this checkpoint**)
bex T	10110	JI	if ($rstatus != 0) PC = T (**not required for this checkpoint**)
setx T	10101	JI	$rstatus = T (**not required for this checkpoint**)
custom_r $rd, $rs, $rt	00000 (01000 - 11111)	R	$rd = custom_r($rs, $rt) (For use on Final Project - **not required for this checkpoint**)

custom ...	xxxxx+	X	Whatever custom instructions you need for your Final Project - **not required 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. JI-type target field [26:0] (T) is unsigned. PC and STATUS registers' upper bits [31:27] are guaranteed to never be used.

Resources

We have provided base codes in the cpuone-base directory. Sample alu.v and regfile.v files can be found in the reference directory for you to use if you are not confident in your own. Feel free to modify the given reference files to suit your own design. Here are some hints for testing your processor. Generally, you would want to write instructions in assembly (e.g., the basic_test.s and halfTestCases.s files in the reference directory), and convert it to a .mif file using the assembler. An assembler is provided to help you convert the instruction into machine code and generate the memory initialization file (.mif) for your design. You can get the assembler here. Then you can change the init_file for your imem to the location of the newly generated mif file. After you have set up your imem .mif file, you can either run a

testbench or waveform to observe the behavior of your processor. Please remember to revert the change when declaring internal wires as outputs temporarily for testing purposes.

In some cases the macOS version of the 550-assembler we provided won't open on the machine with the pop-up window shown in the attachment. This is because this program is not signed and notarized (which requires purchasing an Apple developer ID). You can try to follow this step to try to run it: https://support.apple.com/en-us/HT202491 (Section: "*If you want to open an app that hasn't been notarized or is from an unidentified developer*")

If you still cannot open the assembler or don't trust this program because of this warning, unfortunately, we do not have a better solution. You may use the Windows version in the virtual machine which you used to run Quartus.