Project Checkpoint 4

Simple Processor -- R-type and I-type

Logistics

This is the first checkpoint for our processors.

- Due: **Tuesday, October 26, 2021 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 as the last project, but not strictly required.

Introduction

In this checkpoint and the next one, you will 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

For this checkpoint, you are required to implement the basic functions of a processor. Specifically, you will implement ***R-type and I-type*** instructions: ***`add, addi, sub, and, or, sll, sra, sw, lw.`*** **DO NOT** implement J-type instructions in this checkpoint. In the next checkpoint, you will add the J-type instructions to your processor based on your submission of this checkpoint.

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

Module Interface

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

Please follow the provided base code in 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.

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. When setting up your project in Quartus, be sure to pick the correct device (check the recitation 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.

As usual, you can choose to submit a .zip file or a Github link on Sakai.

- **One group should only submit one work** (Another person should not submit on Sakai). For attachment submission you should upload one **.**zip file named by you and your partner's name; for Github submission, give the link of the final version commit.
- Submitted files should include your code and a README.md file.
- The code should only contains **all necessary *.v modules** for your project. You **should not submit the folders, the test files or the backup files**.
- A README.md (written in markdown, Github flavor) should include
    - You and your partner's name and netID
    - A text description of your design implementation. Please clarify your processor structure (i.e. the function and explanation of each module or each .v file in your solution).
    - If there are bugs or issues, descriptions of what they are and what you think caused them

Grading

Grading will be different from what it has been for the other project checkpoints:

- Your code will be run as it is in previous project checkpoints 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 in for 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).

For regrading, please check this link to find and contact your corresponding TA within one week after grade release.

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, $\overset{rt}{\boxed{N}}$($rs)	00111	I	MEM[$rs + N] = $rd
lw $rd, $\boxed{N}$($rs)	01000	I	$rd = MEM[$rs + N]
j T	00001	JI	PC = T (**not needed for this checkpoint**)
bne $rd, $rs, N	00010	I	if ($rd != $rs) PC = PC + 1 + N (**not needed for this checkpoint**)
jal T	00011	JI	$r31 = PC + 1, PC = T (**not needed for this checkpoint**)
jr $rd	00100	JII	PC = $rd (**not needed for this checkpoint**)
blt $rd, $rs, N	00110	I	if ($rd < $rs) PC = PC + 1 + N (**not needed for this checkpoint**)
bex T	10110	JI	if ($rstatus != 0) PC = T (**not needed for this checkpoint**)
setx T	10101	JI	$rstatus = T (**not needed for this checkpoint**)
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 5 [31:27]	$rd 5 [26:22]	$rs 5 [21:17]	Immediate (N) 17 [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

Resources

We have provided you with the base code in cpuone-base-master folder. Sample alu.v and regfile.v files can be found in the reference folder 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.
On testing your processor, here are some hints. Generally, you would want to write instructions in assembly (i.e the basic_test.s and halfTestCases.s files in the reference folder), and convert it to a .mif file using the assembler given in the 350assembler folder. Please read "README" carefully before launching the jar executable file. Also note you will need Java 10 to run this program. 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 testbench or waveform to observe the behavior of your processor. Please remember to revert the change back if you declare internal wires as outputs temporarily for testing purposes.