# Project Checkpoint #4

This is the fourth Project Checkpoint for our processor.

- Assignment Link: https://classroom.github.com/a/tmpCsN56
- Due: **Thursday, March 1, 2018** at **7:59 PM** with **4 hour grace**
- Late Penalty: No penalty until end of grace. No credit will be given after. **Be sure to start this assignment early. Completion of the processor is required to receive a passing grade in the class.**

Design and simulate a five-stage single-issue 32-bit processor, as described in class, using Verilog. Please make sure that your design:

- Integrates your register file, ALU, and multdiv units
- Uses pipeline "latches" as described in class (and review from ECE/CS 250)
- Implements bypassing to maximize efficiency (i.e. avoid stalls)
- Handles hazards

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 (except for FSMs):

- `if (some_condition)`
- `switch`

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

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. 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 video tutorial if you are unsure).

1. Instructions that change control flow (bne, blt, j, jal, jr) do not have a delay slot
2. Memory rules:
    a. Memory is word-addressed (32-bits per read/write)
    b. Instruction and data memory 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
3. 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.

### Stalls, Hazards, Bypasses, Oh My!

In order of priority, we care about:

1. Correctness: you must detect and avoid hazards
2. Efficiency: use bypass logic to make your processor as efficient as possible

### 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

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

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
- A technical report, as a .tex file, and any necessary assets (e.g. .png files, etc.). Be sure that this compiles correctly with latexmk and pdflatex.

## Grading

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

- A presentation defending your processor design
- A technical report
- The correctness of your processor
- Peer reviewing another processor (pass/fail)

As the deadlines approach, emails will be sent about scheduling.

### Presentation

- The week following the submission deadline, you will meet with TA's to defend your processor design and accuracy
- Prepare a presentation (not necessarily a powerpoint, but there should be a visual element) to explain the function of your processor to the TAs
- Focus on your processor design and you should be able to justify your design choices
- Your presentation should be 5–10 minutes (**hard max** at 10 minutes) and you should budget 20–25 minutes for questions

### Technical Report

- Your report should cover the details of your processor design, including regfile, ALU, and multdiv
- You should include details on:
    - How you implemented each instruction
    - Why you chose your implementation
    - How you tested your implementation
    - The challenges you faced and how you overcame them
- If your processor has errors or issues, you should include extensive details as to what these issues are, why they occur, how you attempted to fix them, how you tested your processor to find these issues, and how these issues can be fixed
- More details will be provided as the deadline approaches

### Correctness

- You will spend 60 minutes with TAs (in addition to the presentation) to verify the correctness of your processor
- A grading skeleton file, imem, and dmem will be swapped in for yours
- We will checkout the last submission **before the deadline** on Github for you; you **cannot** bring in additional code.

### Peer Review

- Additionally, you will be responsible for sitting in on the presentation element of one of your peers and providing feedback, as well as learning about their design.

## ISA

| Instruction | Opcode (ALU op) | Type | Operation |
|---|---|---|---|
| add $rd, $rs, $rt | 00000 (00000) | R | $rd = $rs + $rt<br>$rstatus = 1 if overflow |
| addi $rd, $rs, N | 00101 | I | $rd = $rs + N<br>$rstatus = 2 if overflow |
| sub $rd, $rs, $rt | 00000 (00001) | R | $rd = $rs - $rt<br>$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 |
| mul $rd, $rd, $rt | 00000 (00110) | R | $rd = $rs * $rt<br>$rstatus = 4 if overflow |
| div $rd, $rs, $rt | 00000 (00111) | R | $rd = $rs / $rt<br>$rstatus = 5 if exception |
| 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<br>(01000 - 11111) | R | $rd = custom_r($rs, $rt) (n.b. For use on Final Project) |
| custom ... | xxxxx+ | X | Whatever custom instructions you need for your Final Project |

## Instruction Machine Code Format

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

## 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