# Computer Organization (2022-Fall)



# Final Project - Simple Processor

## Design Descr

A processo specified in the form of instructions. In this project, a string e given. Your job is to decode these instructions and execute.

➢ The followi red instruction format: (similar to MIPS)

| Type | Fields (32 bits) | | | | | |
|---|---|---|---|---|---|---|
| R-type | opcode (6) | rs (5) | rt (5) | rd (5) | shamt (5) | funct (6) |
| I-type | opcode (6) | rs (5) | rt (5) | immediate (16) | | |

Register s(rs), Register t(rt) and Register d(rd) represent the address of registers. Since the instruction takes 5 bits to store the address, it means we have 32 registers, from r[0] to r[31]. Each register reserve 32 bits to store data, e.g. rs = 5'b00000 means one of operands is "r[0]". rt = 5'b00101 means one of operands is "r[5]", and so on. The register file [31:0] r [0:31] have been declared by TA in SP.v. **DO NOT** edit the name of register file, or TA's pattern would catch wrong results.

➢ The following is the required instruction set of this design:

| Type | Function | Description | opcode | funct |
|---|---|---|---|---|
| R-type | and | `R[$rd] ← R[$rs] & R[$rt]` | 0x00 | 0x00 |
| | or | `R[$rd] ← R[$rs] \| R[$rt]` | 0x00 | 0x01 |
| | add | `R[$rd] ← R[$rs] + R[$rt]` | 0x00 | 0x02 |
| | sub | `R[$rd] ← R[$rs] - R[$rt]` | 0x00 | 0x03 |
| | slt | `if(R[$rs] < R[$rt])`<br>`  R[$rd] ← 1`<br>`else`<br>`  R[$rd] ← 0` | 0x00 | 0x04 |
| | sll | `R[$rd] ← R[$rs] << shamt` | 0x00 | 0x05 |
| | nor | `R[$rd] ← ~(R[$rs] \| R[$rt])` | 0x00 | 0x06 |
| I-type | andi | `R[$rt] ← R[$rs] & ZE(imm)` | 0x01 | |
| | ori | `R[$rt] ← R[$rs] \| ZE(imm)` | 0x02 | |
| | addi | `R[$rt] ← R[$rs] + SE(imm)` | 0x03 | |
| | subi | `R[$rt] ← R[$rs] - SE(imm)` | 0x04 | |
| | lw | `R[$rt] ← Mem(R[$rs] + SE(imm))` | 0x05 | |
| | sw | `Mem(R[$rs] + SE(imm)) ← R[$rt]` | 0x06 | |
| | beq | `if(R[$rs] == R[$rt])`<br>`  PC ← PC + 4 + SE({imm, 00})` | 0x07 | |
| | bne | `if(R[$rs] != R[$rt])`<br>`  PC ← PC + 4 + SE({imm, 00})` | 0x08 | |
| | lui | `R[$rt] ← {imm, 0x0000}` | 0x09 | |

In the above table, SE or ZE mean doing sign extension or zero extension on the number. There will be a 4096 * 32 data memory in this project. You can access this memory according to the instruction.

The pattern will give instruction memory in this project. You should access pattern with inst_addr to get next instruction. Be careful of the 4 bytes offset.

## Design Input

■ The following are the definitions of input signals.

| Input Signal | Connection | Bit Width | Description |
|---|---|---|---|
| clk | PATTERN | 1 | Positive edge trigger clock. |
| rst_n | PATTERN | 1 | Asynchronous active-low reset. |
| in_valid | PATTERN | 1 | High when **inst** is valid. |
| inst | PATTERN | 32 | Instruction given be pattern. |
| mem_dout | MEM | 32 | Data output from memory according to **mem_addr**. |

■ The following are the definitions of output signals.

| Output Signal | Connection | Bit Width | Description |
|---|---|---|---|
| out_valid | PATTERN | 1 | Pattern will give next Instruction according to **inst_addr** if **out_valid** is high. |
| inst_addr | PATTERN | 32 | Next instruction address. (PC) |
| mem_wen | MEM | 1 | When **WEN** is high, you can load data from memory.<br>When **WEN** is low, you can write data into memory. |
| mem_addr | MEM | 12 | Data memory address input. |
| mem_din | MEM | 32 | Data memory data input |

## Specification



1. Top module name: SP (Design file name: SP.v)
2. It is **asynchronous** and **active-low reset**. If you use synchronous reset in your design, you may fail to reset signals.
3. The reset sig[illegible] given only once at the beginning of simulation.
4. **inst_addr**, o[illegible]**r file r** should be reset after the reset signal is asserted.
5. The **out_vali**[illegible]gh for only **1 cycle** in non pipeline design.
6. **out_valid** sh[illegible]hen **in_valid** is high for non pipeline design.
7. The **out_vali**[illegible]within 10 cycles after **in_valid** raised.
8. Next instruct[illegible]he next cycle after **out_valid** is raised.
9. Pattern will check the **register file r result** and **inst_addr** for each instruction at clock negative edge when **out_valid** is high.
10. Pattern will check the final result of mem.v.
11. Released and demo pattern will not cause overflow.



## Additional specification for pipelined design



12. Use the same **instruction.txt** and **mem.txt** file as non-pipeline design.
13. Use **TESTBED.v, PATTERN.v** to test **non-pipeline** design.
14. Use **TESTBED_p.v, PATTERN_p.v** to test **pipeline** design.



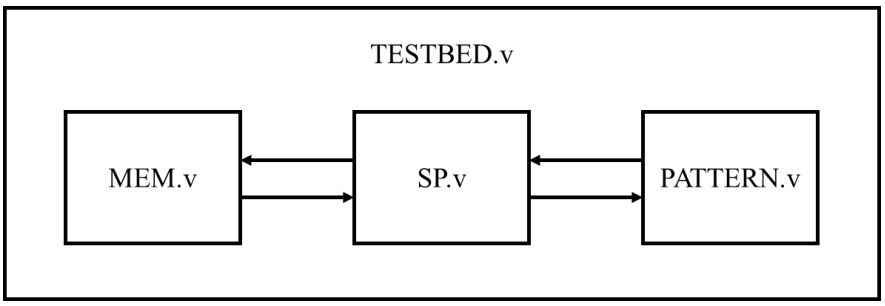
15. No need to consider branch prediction. The correct **inst_addr** should be given by design after instructions fetched.
16. No need to consider data hazards, data dependency or load-use will be separated by at least two instructions to avoid data hazards in pattern.



17. Once **out_valid** raised, it should be kept high until simulation end.
18. Pattern will check the sequence **inst_addr**.



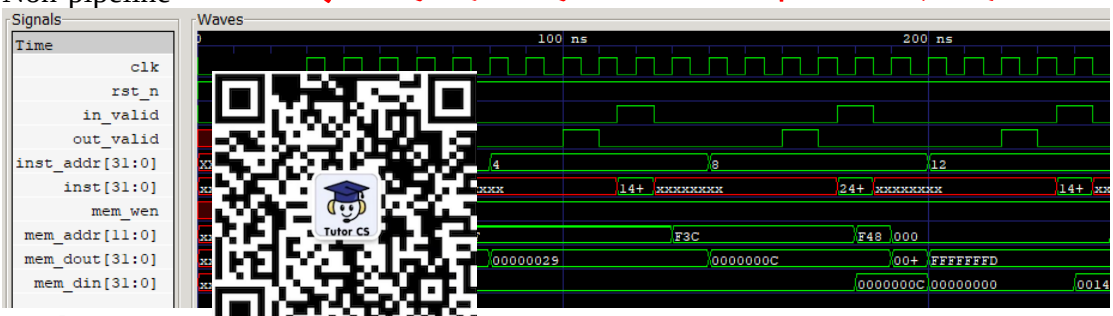
## Block Diagram



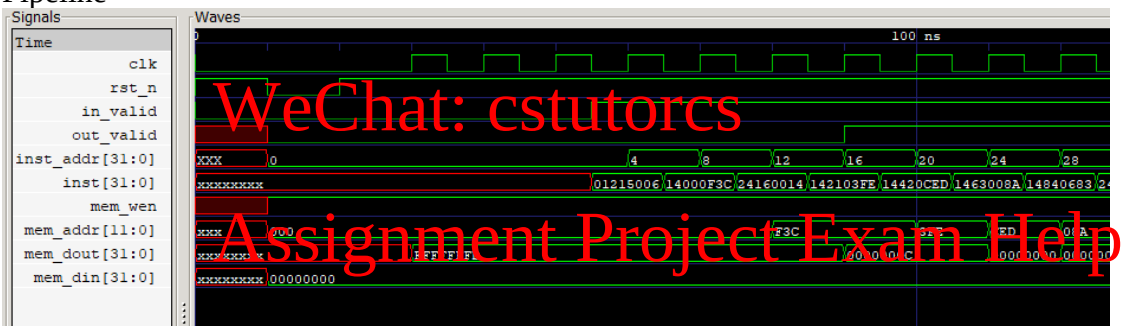
## Example Waveform



Non-pipeline




Pipeline






## Grading Policy

1. Pipelined design : 10%
2. Function validity (released pattern): 60%
3. Function validity (demo pattern): 40%
   - Only 20% for 2nd demo.

## Note

- Please upload “SP_studentID.v” or “SP_studentID_pipeline.v” (for pipelined deisgn) on New e3.
- You can submit both pipeline and non pipeline design, TA will choose the higher grade you submitted.
- 1st demo Due Date: **23:59, Jan 15, 2023**.
  (If you have a late submission by 1 to 7 days, you will only get 80% of the score. We DO NOT accept any late submission after 7 days after the deadline.)
- If the uploaded file violates the naming rule, you will get 5 deduct points on this project.