Computer Organization (2025-Fall)
Final Project - Simple Processor

Design Description

A processor executes operations specified in the form of instructions. In this project, a string of instructions will be given. Your job is to decode these instructions and execute.

➢ The following is the basic required 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] (signed value) have been declared by TA in *SP_pipeline.v* and *SP_non_pipeline.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])`
		`  R[$rd] ← 1`
		`else`
		`  R[$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[$rt]`	0x06	
	lui	`R[$rt] ← {imm, 0x0000}`	0x09	

In the above table, SE or ZE means applying sign extension or zero extension to the number. There will be a 4096 * 32 data memory provided in *DATA_MEM.v* in this project. You can access this memory with **mem_addr** according to the instruction.

You should access the instruction memory provided in *INST_MEM.v* with **inst_addr** to get instruction. Both the instruction address and the memory address use word addressing, so the next instruction should be fetched using **PC + 1**, not **PC + 4**.

Note that DATA_MEM and INST_MEM are designed to produce their outputs one cycle after the clock's rising edge, you must ensure that inst_addr and mem_addr are controlled appropriately.

Design Inputs and Outputs

➢ The following are the definitions of input signals.

Input Signal	Connection	Bit Width	Description
clk	PATTERN	1	Positive-edge-triggered clock.
rst_n	PATTERN	1	Asynchronous active-low reset.
in_valid	PATTERN	1	You can read instructions ONLY when in_valid is **HIGH**.
inst	INST_MEM	32	Instructions from INST_MEM according to **inst_addr**.
mem_dout	DATA_MEM	32	Data output from DATA_MEM according to **mem_addr**.

➢ The following are the definitions of output signals.

Output Signal	Connection	Bit Width	Description
out_valid	PATTERN	1	out_valid is **HIGH** when each instruction is finished.
inst_addr	INST_MEM	12	Next instruction address.
mem_wen	DATA_MEM	1	When mem_wen is **HIGH**, you can load data from DATA_MEM. When mem_wen is **LOW**, you can write data into DATA_MEM.
mem_addr	DATA_MEM	12	Data memory address input.
mem_din	DATA_MEM	32	Data input to DATA_MEM according to **mem_addr**.

Specification

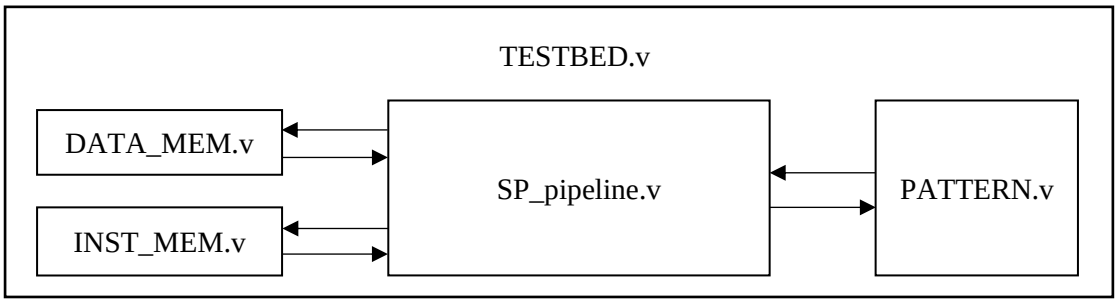
1. Top module name: SP_pipeline, file name: *SP_pipeline.v*
   SP_non_pipeline, file name: *SP_non_pipeline.v*
2. It is **asynchronous** and **active-low reset**. If you use synchronous reset in your design, you may fail to reset signals.
3. The reset signal (**rst_n**) would be given only once at the beginning of simulation.
4. **out_valid** and **register file r** should be reset after the reset signal is asserted.
5. The next **out_valid** should be raised within 10 cycles after the previous one falls.
6. Pattern will check the result of **register file r** for each instruction at clock negative edge when **out_valid** is high.
7. Pattern will check the result of **DATA_MEM** when the last instruction is finished.
8. Released and hidden patterns will not cause overflow.
9. **PLAGIARISM is strictly prohibited**! TAs will check very carefully.

Additional specification for pipelined design

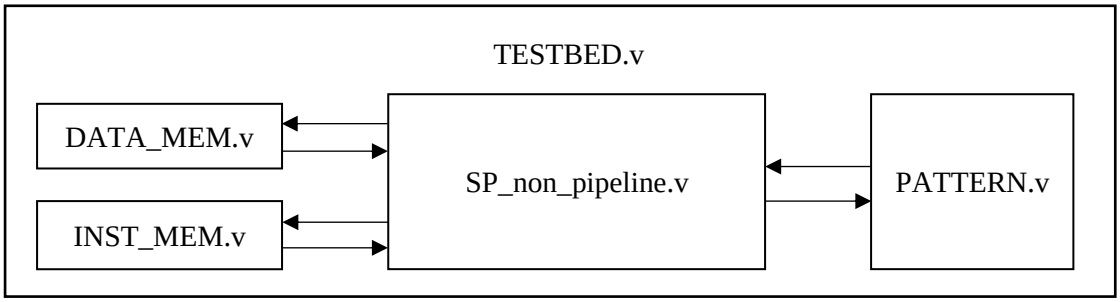
10. Use the same **instruction.txt** and **mem.txt** file as non-pipelined design.
11. Use FP_25/TESTBED.v, FP_25/PATTERN.v to test **pipelined** design.
12. Use FP_25_nonpipe/TESTBED.v, FP_25_nonpipe/PATTERN.v to test **non-pipelined** design.
13. No need to consider data hazards. Data dependency or load-use will be separated by at least two instructions to avoid data hazards in pattern.
14. Once **out_valid** raised, it should be kept high until all instructions are finished.

Block Diagram

Pipelined design

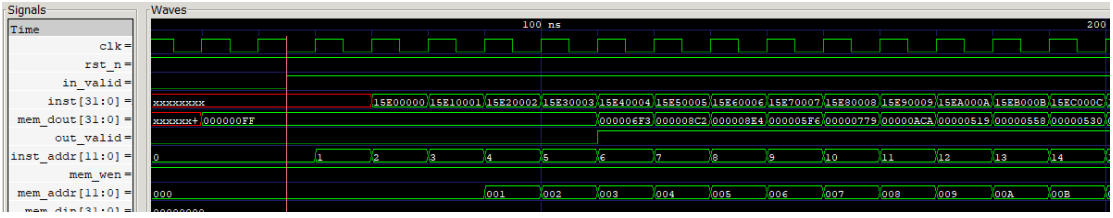


Non-pipelined design

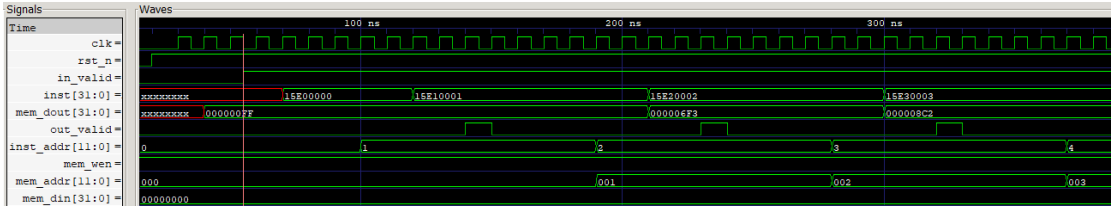


Example Waveform

Pipelined design



Non-pipelined design



Grading Policy

Functionally

Your design will be demonstrated using the released testbench with both the published patterns and the hidden patterns. You are required to pass both to obtain the scores.

However, you must ensure that your design can compile and run with no errors on its own, without relying on the testbench. If it cannot do so, the demo will be deemed a failure.

Grading

1) Function validity (published pattern) with pipelined design: 60 pts
2) Function validity (hidden pattern) with pipelined design: 40 pts

You may choose to submit only the non-pipelined design for partial credit,

1) Function validity (published pattern) with non-pipelined design: 30 pts
2) Function validity (hidden pattern) with non-pipelined design: 20 pts

Only 50% score in 2nd demo!

Note

1. Please upload "**SP_studentID_pipeline.v**" or "**SP_studentID_non_pipeline.v**" on New e3.
2. You can submit **BOTH** pipelined and non-pipelined design, TA will choose the **HIGHER** grade you submitted.
3. 1st demo Due Date: **23:59, Dec 29, 2025**
   2nd demo Due Date: **23:59, Dec 31, 2025**
4. If the uploaded file violates the naming rule, you will get 5 deduct points on this project.