Computer Organization (2024-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)		
Jump	opcode (6)	address(26)				

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])`
		` R[$rd] ← 1`
		`else`
		`R[$rd] ← 0`
			0x00	0x04

	sll	`R[$rd] ← R[$rs] << shamt`	0x00	0x05
	nor	`R[$rd] ← ~(R[$rs] | R[$rt])`	0x00	0x06
	jr	`PC ← R[$rs]`	0x00	0x07
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])` `PC ← PC + 4 + SE({imm, 00})`	0x07	
	bne	`if(R[$rs] != R[$rt])` `PC ← PC + 4 + SE({imm, 00})`	0x08	
	lui	`R[$rt] ← {imm, 0x0000}`	0x09	
Jump	j	`PC←{PC[31:28], address[25:0]<<2}`	0x0a	
	jal	`R[31] ← PC + 4` `PC←{PC[31:28], address[25:0]<<2}`	0x0b	

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 also be used as instruction memory in this project. You should access pattern with inst_addr (PC) to get next instruction. Be careful of the 4 bytes offset.

Design Inputs and Outputs

- 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. 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

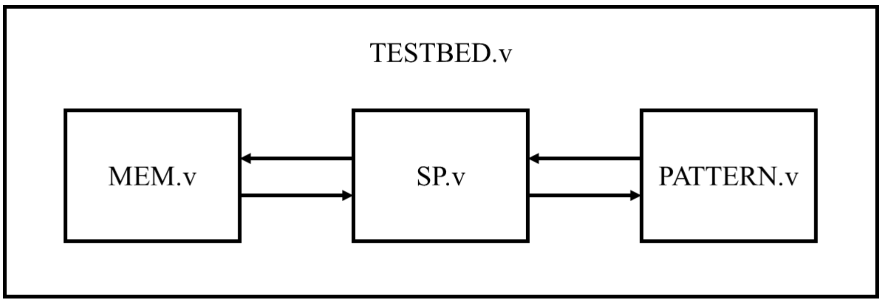
Specification

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 signal (**rst_n**) would be given only once at the beginning of simulation.
4. **inst_addr**, **out_valid** and **register file r** should be reset after the reset signal is asserted.
5. The **out_valid** is limited to be high for only **1 cycle** in non-pipeline design.
6. **out_valid** should not be raised when **in_valid** is high for non-pipeline design.
7. The **out_valid** should be raised within 10 cycles after **in_valid** raised.
8. Next instruction will be given in the 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 **beq/bne** 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

- Generate your PATTERN.v/PATTERN_p.v & Pass TA's design: 30%
- Non-pipelined design (SP.v) pass TA's pattern: 30%
- Pipelined design (SP_pipeline.v) pass TA's pattern: 40%
- Bonus: Show your project design methods in report. (1) Methods of generate pattern (2) Design architecture (3) Course feedback: 15%

Note

- Please upload "PATTERN_studentID.v" / "PATTERN_p_studentID.v" / "SP_studentID.v" / "SP_pipeline_studentID.v" / "Report_studentID.pdf" on New e3.
- Due Date: 23:59, Jan 6, 2025 (1st Demo)
- (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.