

Programming (90%)

In this homework, we will use verilog to implement simple ALU, FPU and CPU in this homework. We use **Icarus Verilog** to run the simulation, and we use **gtkwave** to check waveform. We will score your implementations under these settings.

Folder structure for this homework:

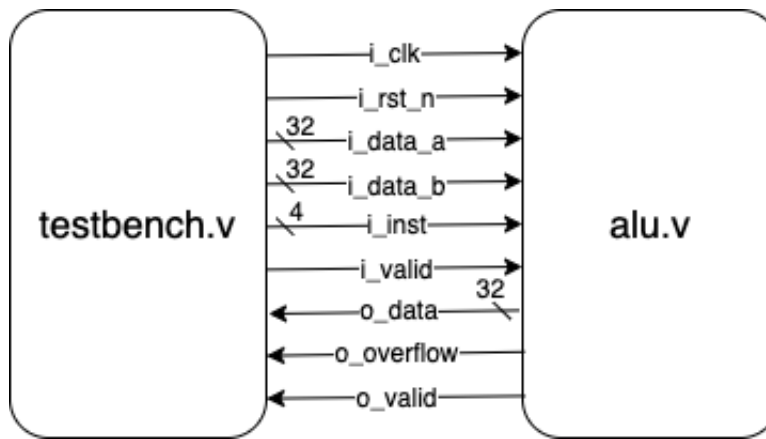
```
HW3
|-- ALU                                // Part 1
|   |-- alu.f                         <-- specify the files you use
|   '-- codes                         <-- put all the *.v here
|       |-- alu.v
|   |-- test_alu                      <-- run the test
|   |-- testbench.v                  <-- test for correttness
|   '-- testcases/                   <-- testcases
|       '-- generate.cpp              <-- used to generate testcases
|-- FPU                                // Part 2
|   |-- fpu.f
|   '-- codes
|       |-- fpu.v
|   |-- test_fpu
|   |-- testbench.v
|   '-- testcases/
|       '-- generate.cpp
'-- CPU                                // Part 3
    |-- cpu.f
    '-- codes
        |-- instruction_memory.v    <-- instruction memory with access latency
        |-- data_memory.v           <-- data memory with access latency
        '-- cpu.v
    |-- test_cpu
    |-- testbench.v
    '-- testcases/
        |-- generate.s
        '-- generate.cpp
```

ALU (30%)

The ALU spec is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_data_a	Input	32	Input data A may be signed or unsigned depending on the i_inst signal
i_data_b	Input	32	Input data B may be signed or unsigned depending on the i_inst signal
i_inst	Input	4	Instruction signal representing functions to be performed
i_valid	input	1	One clock signal when input data a and b are valid
o_data	Output	32	Calculation result
o_overflow	Output	1	Overflow signal
o_valid	Output	1	Should be one cycle signal when your results are valid

The test environment is as follows:



You are asked to implement the following functions in ALU:

i_inst	Function	Description
4'd0	Signed Add	i_data_a + i_data_b (signed)
4'd1	Signed Sub	i_data_a - i_data_b (signed)
4'd2	Signed Mul	i_data_a * i_data_b (signed)
4'd3	Signed Max	max(i_data_a, i_data_b) (signed)
4'd4	Signed Min	min(i_data_a, i_data_b) (signed)
4'd5	Unsigned Add	i_data_a + i_data_b (unsigned)
4'd6	Unsigned Sub	i_data_a - i_data_b (unsigned)
4'd7	Unsigned Mul	i_data_a * i_data_b (unsigned)
4'd8	Unsigned Max	max(i_data_a, i_data_b) (unsigned)
4'd9	Unsigned Min	min(i_data_a, i_data_b) (unsigned)
4'd10	And	i_data_a & i_data_b
4'd11	Or	i_data_a i_data_b
4'd12	Xor	i_data_a ^ i_data_b
4'd13	BitFlip	~ i_data_a
4'd14	BitReverse	Bit reverse i_data_a

More details:

- We will compare the output data and overflow signal with the provided answers
- For signed 32-bit integer Add, Sub, Mul, Max, Min
 - Two-input signal functions
 - Overflow signal only needs to be considered when Add, Sub or Mul is performed. For Max and Min, set the output overflow signal to **0**.
 - We will **not** compare the return data with the answer provided when overflow happens
- For unsigned 32-bit integer Add, Sub, Mul, Max, Min
 - Same criteria as signed operations'
- Xor, And, Or, BitFlip, and BitReverse
 - Set output overflow signal to **0** when the above functions are performed.
 - Xor, And and Or are two-input signal functions.
 - BitFlip and BitReverse are one-input signal functions, therefore, i_data_b can be ignored.

Grading:

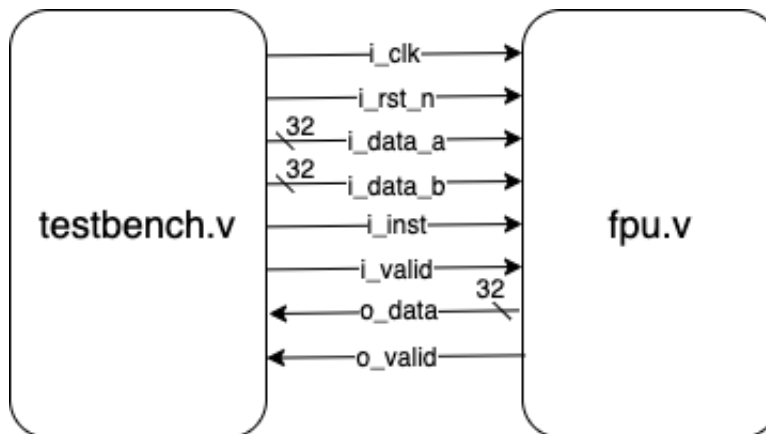
- There are four test cases for each function. Overall, there are 60 test cases.
- 0.5% for each test case

FPU (20%)

The FPU spec is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_data_a	Input	32	Single precision floating point a
i_data_b	Input	32	Single precision floating point b
i_inst	Input	1	Instruction signal representing functions to be performed
i_valid	input	1	One clock signal when input data a and b are valid
o_data	Output	32	Calculation result
o_valid	Output	1	Should be one cycle signal when your results are valid

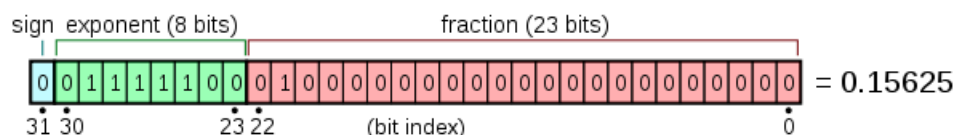
The test environment is as follows:



You are asked to implement the following functions in ALU:

i_inst	Function	Description
1'd0	Add	i_data_a + i_data_b (single precision floating point)
1'd1	Mul	i_data_a * i_data_b (single precision floating point)

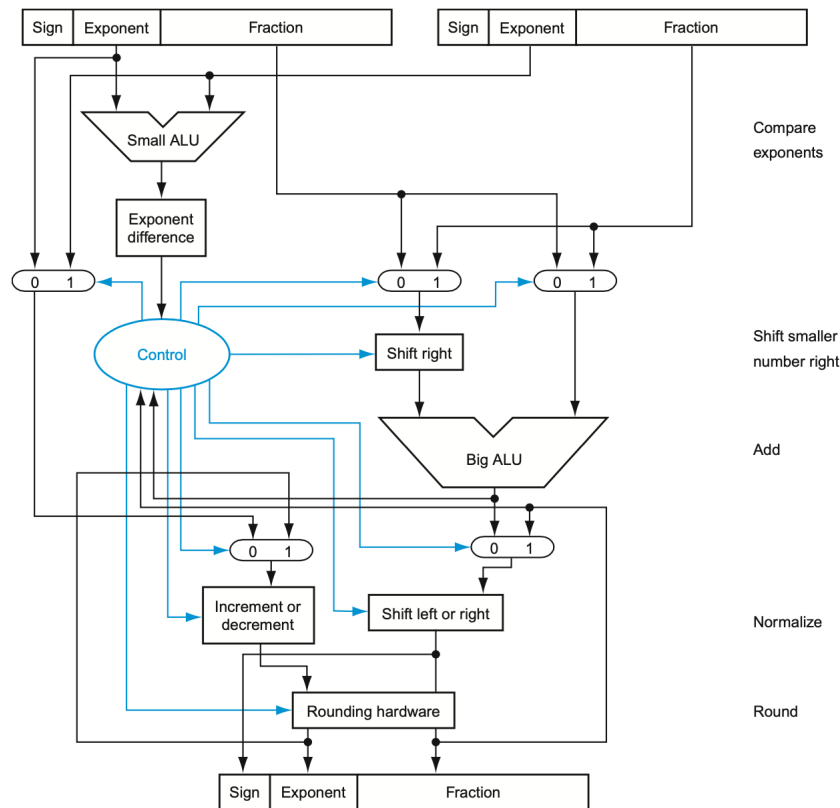
Floating point:



More details:

- We will compare the output data with provided answers.
- Follow [IEEE-754](#) single precision floating point format
- The inputs will not be denormal numbers, infinities, and NaNs, nor will the calculated result.
- Simple testcases
- During the computation, the one with smaller exponent will be shifted, you should keep the precision until rounding. As for rounding mode, we use default **rounding to nearest even**.
 - I find this [pdf](#) useful to explain the rounding and the GRS bits.
 - The testcases may be too easy to worry about the rounding.

You may want to reference the diagram described in class to have better idea implementing FPU.



Grading:

- There are 10 test cases for add and mul. Overall, there are 20 test cases.
- 1.0% for each test case

CPU (40%)

In this section, you are asked to implement a CPU that supports basic RV64I (not all of them). The CPU spec is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_valid_inst	Input	1	One cycle signal when the instruction from instruction memory is ready
i_inst	Input	32	32-bits instruction from instruction memory
i_d_valid_data	Input	1	One cycle signal when the data from data memory is ready (used when ld happens)
i_d_data	Input	64	64-bits data from data memory (used when ld happens)
o_i_valid_addr	Output	1	One cycle signal when the pc-address is ready to be sent to instruction memory (fetch the instruction)
o_i_addr	Output	64	64-bits address to instruction memory (fetch the instruction)
o_d_data	Output	64	64-bits data to data memory (used when sd happens)
o_d_addr	Output	64	64-bits address to data memory (used when ld or sd happens)
o_d_MemRead	Output	1	One cycle signal telling data memory that the current mode is reading (used when ld happens)
o_d_MemWrite	Output	1	One cycle signal telling data memory that the current mode is writing (used when sd happens)
o_finish	Output	1	Stop signal when EOF happens

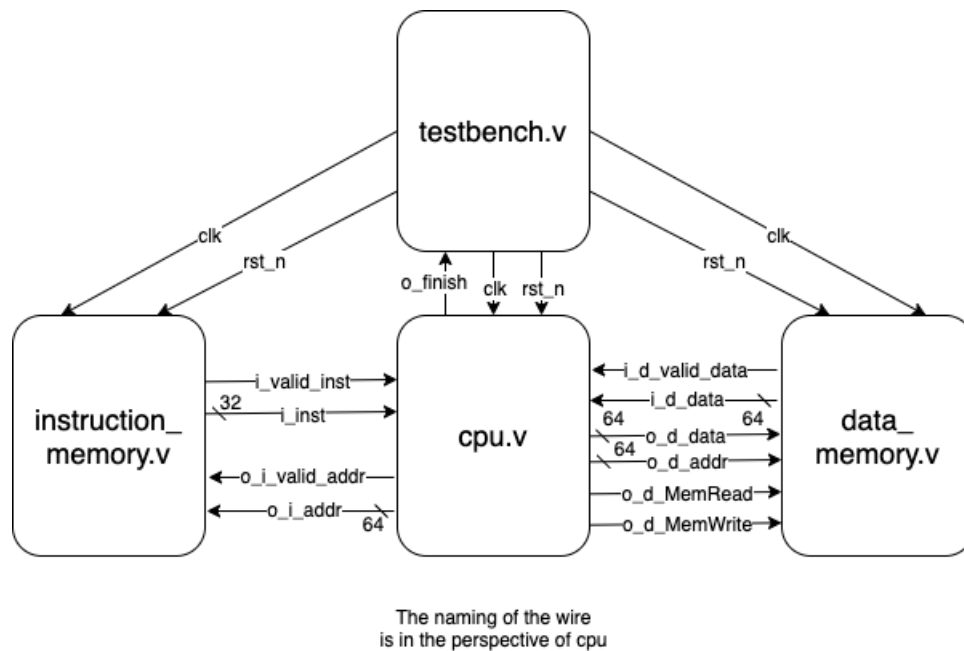
The provided instruction memory is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_valid	Input	1	Signal that tells pc-address from cpu is ready
i_addr	Input	64	64-bits address from cpu
o_valid	Output	1	Valid when instruction is ready
o_inst	Output	32	32-bits instruction to cpu

And the provided data memory is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_data	Input	64	64-bits data that will be stored (used when sd happens)
i_addr	Input	64	Write to or read from target 64-bits address (used when ld or sd happens)
i_MemRead	Input	1	One cycle signal and set current mode to reading
i_MemWrite	Input	1	One cycle signal and set current mode to writing
o_valid	Output	1	One cycle signal telling data is ready (used when ld happens)
o_data	Output	64	64-bits data from data memory (used when ld happens)

The test environment is as follows:



We will only test the instructions highlighted in the red box, as the figures below

imm[11:0]		rs1	110	rd	0000011	LWU
imm[11:0]		rs1	011	rd	0000011	LD
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	SD
000000	shamt	rs1	001	rd	0010011	SLLI
000000	shamt	rs1	101	rd	0010011	SRLI
010000	shamt	rs1	101	rd	0010011	SRAI
imm[11:0]		rs1	000	rd	0011011	ADDIW
0000000	shamt	rs1	001	rd	0011011	SLLIW
0000000	shamt	rs1	101	rd	0011011	SRLIW
0100000	shamt	rs1	101	rd	0011011	SRAIW
0000000	rs2	rs1	000	rd	0111011	ADDW
0100000	rs2	rs1	000	rd	0111011	SUBW
0000000	rs2	rs1	001	rd	0111011	SLLW
0000000	rs2	rs1	101	rd	0111011	SRLW
0100000	rs2	rs1	101	rd	0111011	SRAW

imm[31:12]				rd		0110111	LUI			
imm[31:12]				rd		0010111	AUIPC			
imm[20:10:11:19:12]				rd		1101111	JAL			
imm[11:0]		rs1		000		rd	1100111	JALR		
imm[12:10:5]		rs2		000		imm[4:1:11]	1100011	BEQ		
imm[12:10:5]		rs2		001		imm[4:1:11]	1100011	BNE		
imm[12:10:5]		rs2		100		imm[4:1:11]	1100011	BLT		
imm[12:10:5]		rs2		101		imm[4:1:11]	1100011	BGE		
imm[12:10:5]		rs2		110		imm[4:1:11]	1100011	BLTU		
imm[12:10:5]		rs2		111		imm[4:1:11]	1100011	BGEU		
imm[11:0]		rs1		000		rd	0000011	LB		
imm[11:0]		rs1		001		rd	0000011	LH		
imm[11:0]		rs1		010		rd	0000011	LW		
imm[11:0]		rs1		100		rd	0000011	LBU		
imm[11:0]		rs1		101		rd	0000011	LHU		
imm[11:5]		rs2		000		imm[4:0]	0100011	SB		
imm[11:5]		rs2		001		imm[4:0]	0100011	SH		
imm[11:5]		rs2		010		imm[4:0]	0100011	SW		
imm[11:0]		rs1		000		rd	0010011	ADDI		
imm[11:0]		rs1		010		rd	0010011	SLTI		
imm[11:0]		rs1		011		rd	0010011	SLTIU		
imm[11:0]		rs1		100		rd	0010011	XORI		
imm[11:0]		rs1		110		rd	0010011	ORI		
imm[11:0]		rs1		111		rd	0010011	ANDI		
0000000		shamt		rs1		001	rd	0010011	SLLI	
0000000		shamt		rs1		101	rd	0010011	SRLI	
0100000		shamt		rs1		101	rd	0010011	SRAI	
0000000		rs2		rs1		000	rd	0110011	ADD	
0100000		rs2		rs1		000	rd	0110011	SUB	
0000000		rs2		rs1		001	rd	0110011	SLL	
0000000		rs2		rs1		010	rd	0110011	SLT	
0000000		rs2		rs1		011	rd	0110011	SLTU	
0000000		rs2		rs1		100	rd	0110011	XOR	
0000000		rs2		rs1		101	rd	0110011	SRL	
0100000		rs2		rs1		101	rd	0110011	SRA	
0000000		rs2		rs1		110	rd	0110011	OR	
0000000		rs2		rs1		111	rd	0110011	AND	
fm		pred		succ		rs1	000	rd	0001111	FENCE
000000000000				00000		000	00000	1110011	ECALL	
000000000001				00000		000	00000	1110011	EBREAK	

And one more instruction to be implemented is

i_inst	Function	Description
32'b11111111111111111111111111111111	Stop	Stop and set o_finish to 1

More details:

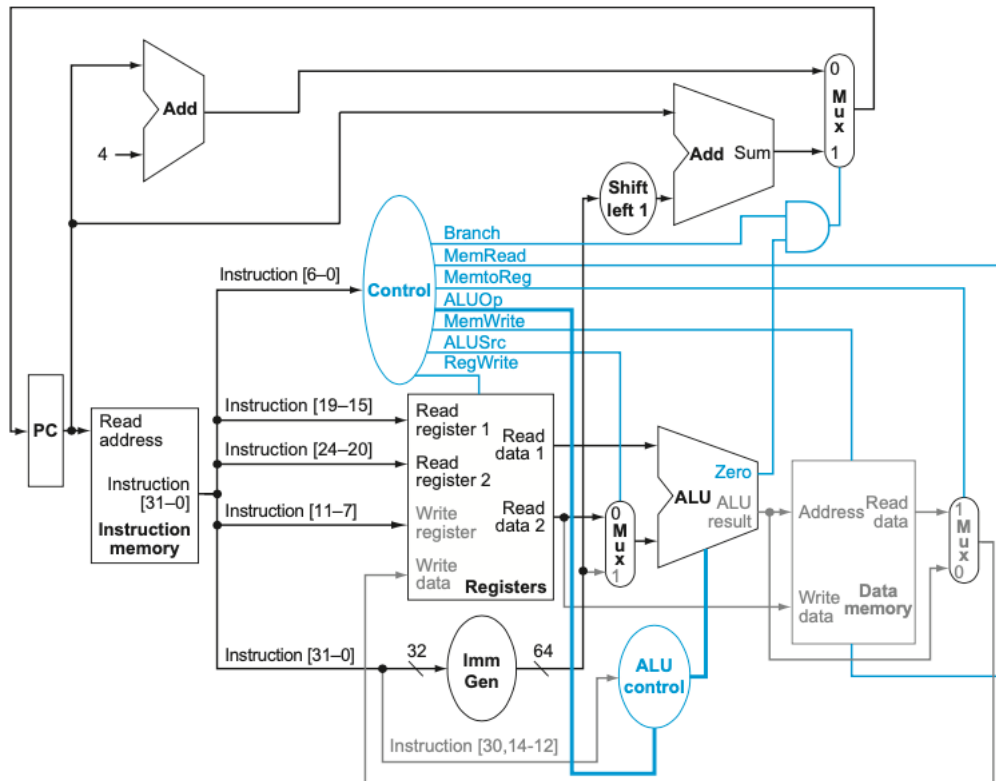
- The instruction_memory.v, data_memory.v and testbench.v files **should not be modified**
- We will compare the mem[1024] in data_memory.v result with provided answers to check for correctness
- There are 1024 bytes memory in data_memory module and 16x32 bits memory for instruction memory. No invalid access to instruction or data memory will be involved in the testcases. Hence, there is no need to handle these issues.
- All the arithmetic operations here are unsigned, including $A + B$ and $A + \text{imm}$. And there is no need to deal with overflow here.
- You may notice that there's latency when we want to access the memory
 - For instruction memory, when i_valid is set, the instruction memory will stall for 5 cycles, and then return the instruction to cpu
 - For data memory, when i_MemRead or i_MemWrite is set, the data memory will stall for 7 cycles in both cases, and then return the data to cpu or write the data to memory
 - The latency comes from freezing the module for certain amount of cycles, as shows below

```

always @(*) begin
  case (cs)
    0: ns = (i_valid) ? 1 : 0;
    1: ns = 2;
    2: ns = 3;
    3: ns = 4;
    4: ns = 5;
    5: ns = 6;
    6: ns = 7;
    7: ns = 0;
  endcase
end

```

You may want to reference the block diagram of cpu from slides or textbook to have better idea implementing cpu. Notice that the diagram provided here is single cycle cpu, while in this homework, there's additional latency accessing memory that needs to be considered.



Grading:

- There are 8 testcases. 5% each. (eof, store, load, add, sub, and, or, xor, andi, ori, xori, slli, srli, bne, beq)

Report (20%)

Write a report about how you implement ALU, FPU, and CPU.

You can draw some block diagrams to show your execution method more clearly.

"draw.io" is suggested to help you to draw the block diagrams easily.

Submission

- Zip and upload your file to COOL in the following format:

```
Rxxxxxxx      <-- zip this folder
|-- ALU
|  |-- codes/
|  |    '-- *.v  <-- files you used
|  '-- alu.f     <-- list all the files needed
|-- FPU
|  |-- codes/
|  |    '-- *.v  <-- files you used
|  '-- fpu.f     <-- list all the files needed
|-- CPU
|  |-- codes/
|  |    '-- *.v  <-- files you used
|  '-- cpu.f     <-- list all the files needed
'-- report.pdf
```

- Grading
 - Run code: TA. XR Wang
 - Report: TA. YH Chung
 - After grading, if you have any problem, you can consult the responsible TA to regrade.
 - Of course, all the TA can help you about your homework3.
- Late submission within one-week: $(\text{Total score}) \times 0.8$
- Late submission within two-week: $(\text{Total score}) \times 0.6$
- Late submission over two-week: $(\text{Total score}) \times 0$
- If there's any question, please send email to 110fall.ca@gmail.com.
- TA hour for this homework:
 - Thur. 2:00 5:00 p.m
 - [Discord group](#) is constructed, you can join and ask question on the platform.
 - If needed, you can send a email to us and ask another time for discussion.
 - If there is anything in the homework that makes you feel unclear and didn't receive the reply over 24 hrs after sending emails to the email of TA, you can email to TA. Chung r09944072@csie.ntu.edu.tw, remember this email will only reply for the errata to Homework.
 - NOTE (some murmur):
 - * Please **don't send the private message to TA (such as Facebook messenger (meta?), discord's private message)** causing the unfairness to other students.
 - All the private messages will be ignored. Please send a direct message to the public platform.

Important Supplementary

- [**HW3 introduction-video \(2020\)](#)
- If you are not familiar with Verilog, you are suggested to read the following links before you email TAs QQ.
 - [20201118 verilog introduction](#)
 - [Verilog Tutorial](#)
 - [Icarus Verilog and GTKWave](#)
 - [The RISC-V Instruction Set Manual](#)

Deadline

- Deadline: 2021/11/30 23:59 (from 2021/11/02 12:00)
- NOTE:
 - Due to there may be some students are not be familiar with "Verilog", and the midterm exam week is coming, the deadline for homework3 is after a month.
 - However, the deadline will never be postponed.
 - **Start writing your homework as early as possible.**