Computer-Aided VLSI System Design
Homework 2: Simple RISC-V CPU

TA: 顏子茗 r13943119@ntu.edu.tw
TA: 陳柏任 d13943013@ntu.edu.tw

Due Tuesday, Oct. 14, 13:59

Data Preparation

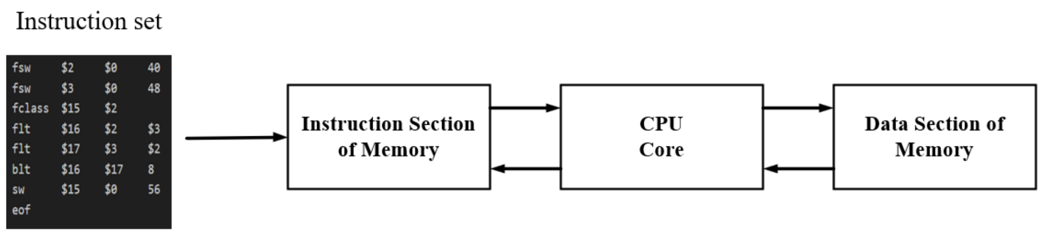
1. Decompress 1141_hw2.tar with following command

```
tar -xvf 1141_hw2.tar
```

Folder	File	Description
00_TB	data_mem.vp	Module of memory (protected)
	define.v	File of definition
	testbed_temp.v	Testbench template
00_ TB / PATTERN/p*	inst.dat	Pattern of instruction in binary format
	inst_assembly.dat	Corresponding assembly code of the instruction pattern
	data.dat	Pattern of final data in memory
	status.dat	Pattern of corresponding status
01_RTL	core.v	Your design
	rtl.f	File list
	01_run	VCS command for simulation
	99_clean_up	Command to clean temporary data

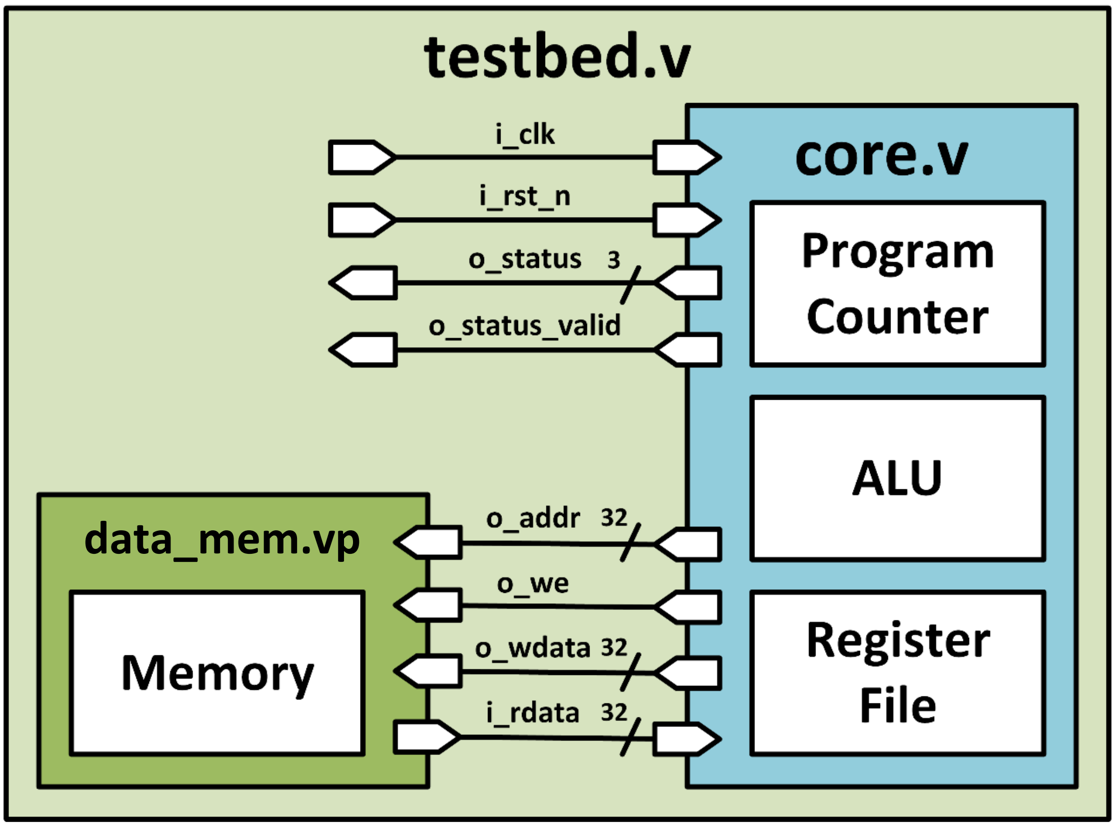
Introduction

Central Processing Unit (CPU) is the important core in the computer system. In this homework, you are asked to design a simple RISC-V CPU, which contains the basic module of program counter, ALU and register files. The instruction set of the simple CPU is similar to RISC-V structure. Since the files of testbench (testbed.v) are either protected or not provided, you also need to design the testbench to test your design.



```
fsw     $2    $0    40
fsw     $3    $0    48
fclass  $15   $2
flt     $16   $2    $3
flt     $17   $3    $2
blt     $16   $17   8
sw      $15   $0    56
eof
```

Block Diagram



Specifications

1. Top module name: core
2. Input/output description:

Signal Name	I/O	Width	Simple Description
i_clk	I	1	Clock signal in the system.
i_rst_n	I	1	Active **low** asynchronous reset.
o_we	O	1	Write enable of memory Set **low** for reading mode, and **high** for writing mode
o_addr	O	32	Address for memory
o_wdata	O	32	Data input to memory
i_rdata	I	32	Data or instruction output from memory
o_status	O	3	Status of core processing to each instruction
o_status_valid	O	1	Set **high** if ready to output status

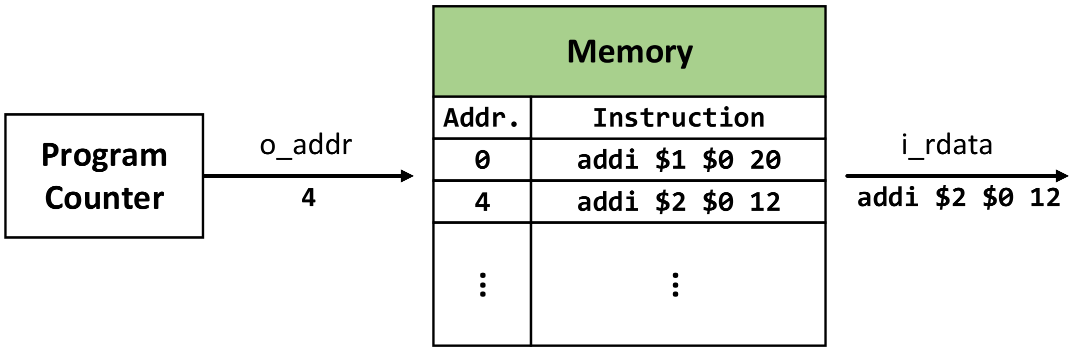
3. All outputs should be synchronized at clock **rising** edge.
4. You should set all your **outputs** and **register file** to be **zero** when i_rst_n is **low**. Active low asynchronous reset is used.

5. Memory is provided. All values in memory are reset to be zero.
6. You should create **32 signed 32-bit registers** and **32 single-precision floating -point registers** in register file.
7. To load data or instruction from the memory, set o_we to **0** and o_addr to relative address value. i_rdata can be received at the next rising edge of the clock.
8. To save data to the memory, set o_we to **1**, o_addr to relative address value, and o_wdata to the written data.
9. Your o_status_valid should be turned to **high** for only **one cycle** for every o_status.
10. The testbench will get your output at negative clock edge to check the o_status if your o_status_valid is **high**.
11. When you set o_status_valid to **high** and o_status to **6**, stop processing. The testbench will check your memory value with golden data.
12. If invalid operation happened, stop processing and raise o_status_valid to **high** and set o_status to **5**. The testbench will check your memory value with golden data.
13. **Less than 1024** instructions are provided for each pattern.
14. The whole processing time can't exceed **120000** cycles for each pattern.

Design Description

1. Program counter is used to control the address of memory for instruction.

   `$pc = $pc + 4` for every instruction (except **`beq`**, **`blt`**, **`jalr`**)



2. Register file contains 32 signed 32-bit registers and 32 single-precision floating -point registers for operation.

3. Instruction mapping

   a. R-type

[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
funct7	r2/f2	r1/f1	funct3	rd/fd	opcode

b. **I-type**

[31:20]	[19:15]	[14:12]	[11:7]	[6:0]
imm[11:0]	r1/f1	funct3	rd/fd	opcode

c. **S-type**

[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
imm[11:5]	r2/f2	r1/f1	funct3	imm[4:0]	opcode

d. **B-type**

[31]	[30:25]	[24:20]	[19:15]	[14:12]	[11:8]	[7]	[6:0]
imm[12]	imm[10:5]	r2/f2	r1/f1	funct3	imm[4:1]	imm[11]	opcode

e. **U-type**

[31:12]	[11:7]	[6:0]
imm[31:12]	rd/fd	opcode

f. **EOF**

[31:7]	[6:0]
Not used	opcode

4. The followings are the instructions you need to design for this homework:

Operation	Assemble	Type	Meaning	Note
Subtract	`sub`	R	`$rd = $r1 - $r2`	Signed Operation
Add immediate	`addi`	I	`$rd = $r1 + im`	Signed Operation
Load word	`lw`	I	`$rd = Mem[$r1 + im]`	Signed Operation
Store word	`sw`	S	`Mem[$r1 + im] = $r2`	Signed Operation

Branch on equal	`beq`	B	`if($r1==$r2), $pc = $pc + im;`
			`else, $pc = $pc + 4`
				PC-relative
				Signed Operation

Branch less than	`blt`	B	`if($r1<$r2), $pc = $pc + im;`
			`else, $pc = $pc + 4`
				PC-relative
				Signed Operation

Jump and link register	`jalr`	I	`$rd = $pc + 4;`
			`$pc = ($r1 + im) & (~0x1)`
				PC-relative
				Signed Operation

Add upper immediate to PC	`auipc`	U	`$rd = $pc + (im << 12)`	PC-relative Signed Operation

Set on less than	`slt`	R	`if($r1<$r2), $rd = 1;`
			`else, $rd = 0`
				Signed Operation

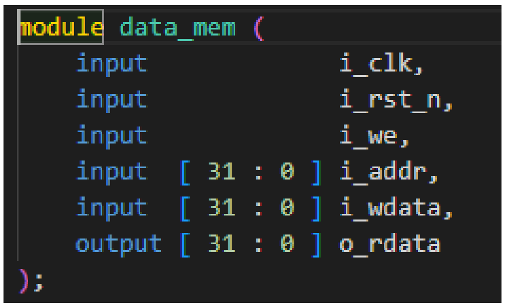
Shift right logical	`srl`	R	`$rd = $r1 >> $r2`	Unsigned Operation

Floating-point substract	`fsub`	R	`$f1 = $f2 - $f3`	Floating-point Operation
Floating-point multiply	`fmul`	R	`$f1 = $f2 * $f3`	Floating-point Operation
Floating-point to signed integer conversion	`fcvt.w.s`	R	`$rd = s32f32($f1)`	Floating-point Operation
Load floating-point	`flw`	I	`$fd = Mem[$r1 + im]`	Signed Operation
Store floating-point	`fsw`	S	`Mem[$r1 + im] = $f2`	Signed Operation
Floating-point classify	`fclass`	R	`$rd = fclass($f1)`	Classify floating-point format
End of File	`eof`	EOF	`Stop processing`	Last instruction in the pattern

Note: The notation of `im` in instruction is **2's complement**.
Note: The $r notes that the data is read/written to integer register file; the $f notes that the data is read/written to floating-point register file.
Note: Set the result of fsub and fmul to +0 if the arithmetic result is 0

5. Interface of memory (size: 2048×32 bit)
   - i_addr[12:2] for address mapping in memory
   - Instructions are stored in address 0 - address 4095
   - Data are should be write to address 4096 - address 8191
   - To fetch data of memory in your testbench, use following instance name

```
u_data_mem.mem_r[i]
```



```
module data_mem (
    input                i_clk,
    input                i_rst_n,
    input                i_we,
    input   [ 31 : 0 ]   i_addr,
    input   [ 31 : 0 ]   i_wdata,
    output  [ 31 : 0 ]   o_rdata
);
```

6. Invalid operation may be happened.
   - **<u>Situation1</u>**: Overflow happened at integer arithmetic instructions (**sub, addi**)

- **<u>Situation2</u>**: Infinite, NaN happened at floating-point arithmetic instructions (**fsub**, **fmul, fcvt.w.s**)
    - For **fcvt.w.s** instruction, additional consideration is required when the floating-point value being converted to a 32-bit signed integer exceeds the representable range of a 32-bit signed integer
    - Do not consider when loading/storing infinite or NaN numbers from memory
    - Do not consider when executing **fclass** on infinite or NaN numbers

- **<u>Situation3</u>**: Overflow and underflow result happened at floating-point arithmetic instructions (**fsub**, **fmul**)
    - Consider the overflow and underflow before rounding arithmetic result
    - Underflow does not include zero

- **<u>Situation4</u>**: If output address are mapped to unknown address in memory.
    - Consider the case when an instruction is trying to load/store the address of memory for instruction
    - Consider the case when program counter is fetching instruction from the address of memory for data
    - Do not consider the case if instruction address is beyond eof, but the address mapping is in the size of memory for instruction

7. 7 statuses of o_status

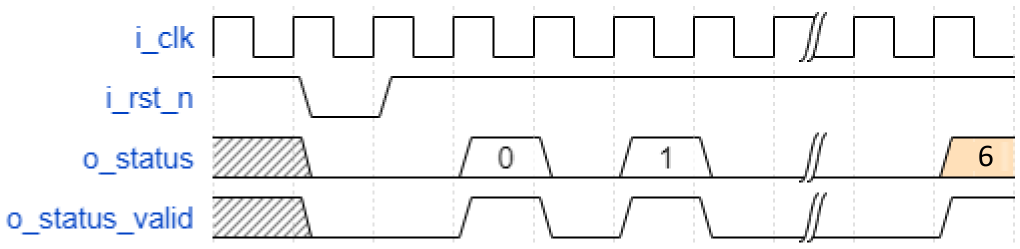
o_status	**Definition**
3’d0	R_TYPE_SUCCESS
3’d1	I_TYPE_SUCCESS
3’d2	S_TYPE_SUCCESS
3’d3	B_TYPE_SUCCESS
3’d4	U_TYPE_SUCCESS
3’d5	INVALID_TYPE
3’d6	EOF_TYPE

8. Last instruction would be `eof` for every pattern.

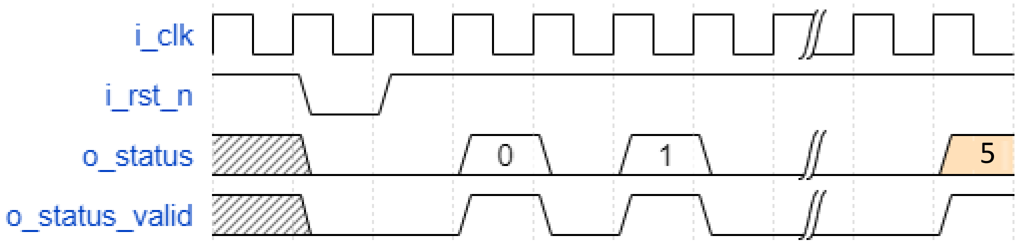
9. There is no unknown opcode in the pattern.

Sample Waveform

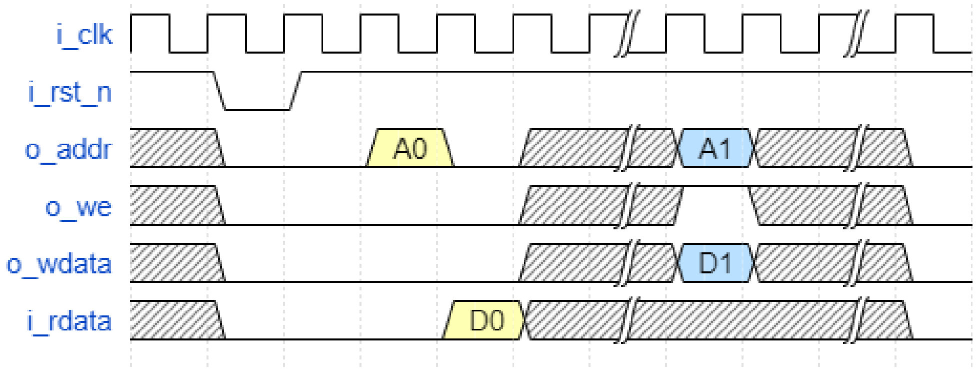
1. Status check
    a. End of file (eof)



b. Invalid operation



2. Memory interface

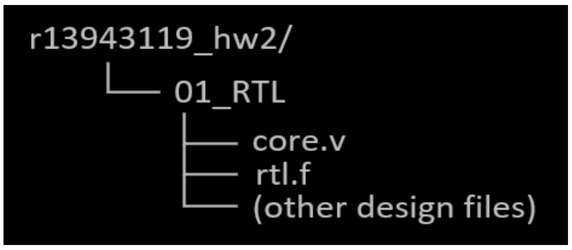


Testbed

1. Things to add in your testbench
   - Clock
   - Reset
   - Waveform file (.fsdb)
   - Function test
   - …

Submission

2. Create a folder named **studentID_hw2**, and put all below files into the folder



```
r13943119_hw2/
    └── 01_RTL
          ├── core.v
          ├── rtl.f
          └── (other design files)
```

Note: Use **lower case** for the letter in your student ID. (Ex. r13943119_hw2)

3. Compress the folder **studentID_hw2** in a **tar file** named **studentID_hw2_v*k*.tar (*k* is the number of version, *k* =1,2,…)**

```
tar -cvf studentID_hw2_vk.tar studentID_hw2
```

   TA will only check the last version of your homework.
   Note: Use **lower case** for the letter in your student ID. (Ex. r13943119_hw2_v1)
   Note: Pack the folder on IC Design LAB server to avoid OS related problems.

4. Submit to NTU Cool

Grading Policy

1. TA will run your code with following format of command. Make sure to run this command with no error message. [p0–p3 (p0 shown as example below)]

```
vcs -f rtl.f -full64 -R -sverilog -debug_access+all +define+p0 -v2k
```

2. Pass the patterns to get full score.
   - Provided pattern: **70%** (patterns: 4)
     - **15%** for each pattern
     - **10%** for spyglass check (lint_rtl and lint_rtl_enhanced)
     - **Don't implement the answers in your design directly!**
   - Hidden pattern: **30%**
     - **20 patterns** in total
3. Delay submission
   - **No delay submission is allowed**
   - Lose **5 point** for any wrong naming rule. Don't compress all homework folder.

Hint

1. Design your FSM with following states
   - Idle
   - Instruction Fetching
   - Instruction decoding
   - ALU computing/ Load data
   - Data write-back
   - Next PC generation
   - Process end

Reference

[1] RISC-V User Manual
- https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[2] IEEE 754 Single Precision Format
- https://zh.wikipedia.org/zh-tw/IEEE_754

[3] Round to Nearest Even

- https://www.cs.cmu.edu/afs/cs/academic/class/15213-s16/www/lectures/04-float.pdf

[4] F Standard Extension for Single-Precision Floating-Point

- https://five-embeddev.com/riscv-user-isa-manual/Priv-v1.12/f.html