HOMEWORK 5 RISC-V CPU

Due date:

Overview

The goal of this homework is to help you understand how a RISC-V work and how to use Verilog hardware description language (Verilog HDL) to model electronic systems. In this homework, you need to implement ALU and decoder module and make your codes be able to execute *17 RISC-V* instructions. You need to follow the instruction table in this homework and satisfy all the homework requirements. In addition, you need to verify your CPU by using Modelsim.

General rules for deliverables

- You need to complete this homework INDIVIDUALLY. You can discuss the homework with other students, but you need to do the homework by yourself. You should not copy anything from someone else, and you should not distribute your homework to someone else. If you violate any of these rules, you will get NEGATIVE scores, or even fail this course directly
- When submitting your homework, compress all files into a single **zip** file, and upload the compressed file to Moodle.
  - Please follow the file hierarchy shown in Figure 1.

F740XXXXX (your id) (folder)

src (folder) * Store your source code

report.docx (project report. The report template is already included. Follow the template to complete the report.)

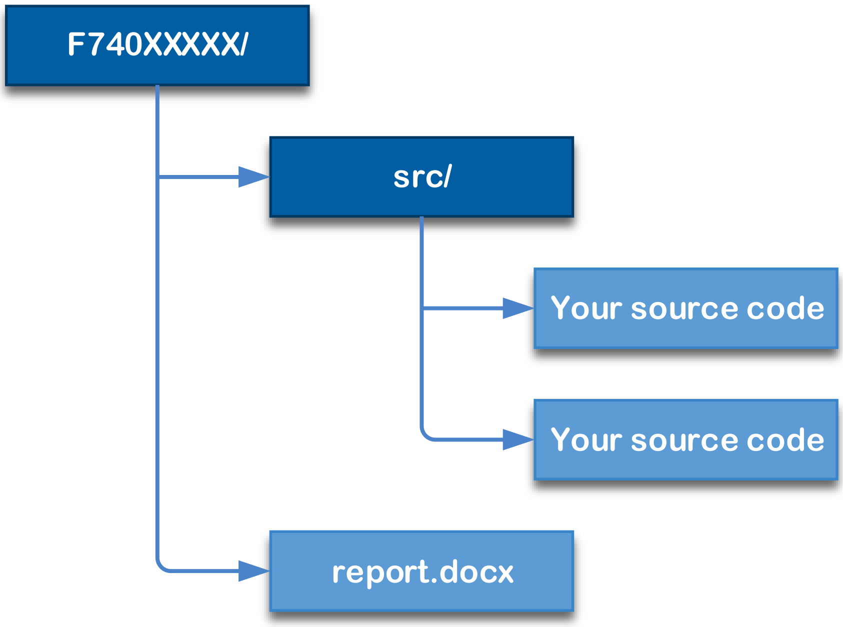


Figure 1. File hierarchy for homework submission

- **Important!** DO NOT submit your homework in the last minute. Late submission is not accepted.
- You should finish all the requirements (shown below) in this homework and Project report.
- If your code can not be recompiled by TA successfully using modelsim, you will receive NO credit.
- Verilog and SystemVerilog generators aren't allowed in this course.

Instruction format:

The highlighted instructions that have been completed in the previous job.

- **R-type**

31 25	24 20	19 15	14 12	11 7	6 0		
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**	**Mnemonic**	**Description**
0000000	rs2	rs1	000	rd	0110011	ADD	rd = rs1 + rs2
0100000	rs2	rs1	000	rd	0110011	SUB	rd = rs1 - rs2
0000000	rs2	rs1	100	rd	0110011	XOR	rd = rs1 ^ rs2
0000000	rs2	rs1	110	rd	0110011	OR	rd = rs1 | rs2
0000000	rs2	rs1	111	rd	0110011	AND	rd = rs1 & rs2

- **I-type**

31 20	19 15	14 12	11 7	6 0		
imm[11:0]	**rs1**	**funct3**	**rd**	**opcode**	**Mnemonic**	**Description**
imm[11:0]	rs1	010	rd	0000011	LW	rd = M[rs1 + imm]
imm[11:0]	rs1	000	rd	0010011	ADDI	rd = rs1 + imm
imm[11:0]	rs1	100	rd	0010011	XORI	rd = rs1 ^ imm
imm[11:0]	rs1	110	rd	0010011	ORI	rd = rs1 | imm
imm[11:0]	rs1	111	rd	0010011	ANDI	rd = rs1 & imm
imm[11:0]	rs1	000	rd	1100111	JALR	rd = PC + 4 PC = imm + rs1 (Set LSB of PC to 0)

- **S-type**

31 25	24 20	19 15	14 12	11 7	6 0		
imm[11:5]	**rs2**	**rs1**	**funct3**	**imm[4:0]**	**opcode**	**Mnemonic**	**Description**
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW	M[rs1 + imm] = rs2

- **B-type**

31 25	24 20	19 15	14 12	11 7	6 0		
imm[12|10:5]	**rs2**	**rs1**	**funct3**	**imm[4:1|11]**	**opcode**	**Mnemonic**	**Description**
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ	PC = (rs1 == rs2) ? PC + imm : PC + 4
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE	PC = (rs1 != rs2) ? PC + imm : PC + 4

- **U-type**

31	12	11 7	6 0		
imm[31:12]		**rd**	**opcode**	**Mnemonic**	**Description**
imm[31:12]		rd	0010111	AUIPC	rd = PC + imm
imm[31:12]		rd	0110111	LUI	rd = imm

- **J-type**

31	12	11 7	6 0		
imm[20|10:1|11|19:12]		**rd**	**opcode**	**Mnemonic**	**Description**
imm[20|10:1|11|19:12]		rd	1101111	JAL	rd = PC + 4 PC = PC + imm

Homework Description

- **Module**

  a. top_tb module

  b. "top_tb" is not a part of CPU, it is a file that controls all the program and verify the correctness of our CPU. The main features are as follows: send periodical signal CLK to CPU, set the initial value of IM, print the value of DM, end the program.
  ※You do not need to modify this module.

  c. top module

  "top" is the outmost module. It is responsible for connecting wires between CPU, IM and DM.
  Here are the wires:
  - *instr_read* represents the signal whether the instruction should be read in IM.
  - *instr_addr* represents the instruction address in IM.
  - *instr_out* represents the instruction send from IM .
  - *data_read* represents the signal whether the data should be read in DM.
  - *data_write* has four signal , and every signal represents the byte of the data whether should be wrote in DM.
    Mem[0] =
    {Mem[0][31:24],Mem[0][23:16],Mem[0][15:8],Mem[0][7:0]}
    data_write[3] => control Mem[0][31:24]
    data_write[2] => control Mem[0][23:16]
    data_write[1] => control Mem[0][15:8]
    data_write[0] => control Mem[0][7:0]

- *data_addr* represents the data address in DM.
- *data_in* represents the data which will be wrote into DM .
- *data_out* represents the data send from DM .

※You do not need to modify this module.

d. <u>SRAM module</u>

"SRAM" is the abbreviation of "Instruction Memory" (or "Data Memory"). This module saves all the instructions (or data) and send instruction (or data) to CPU according to request.

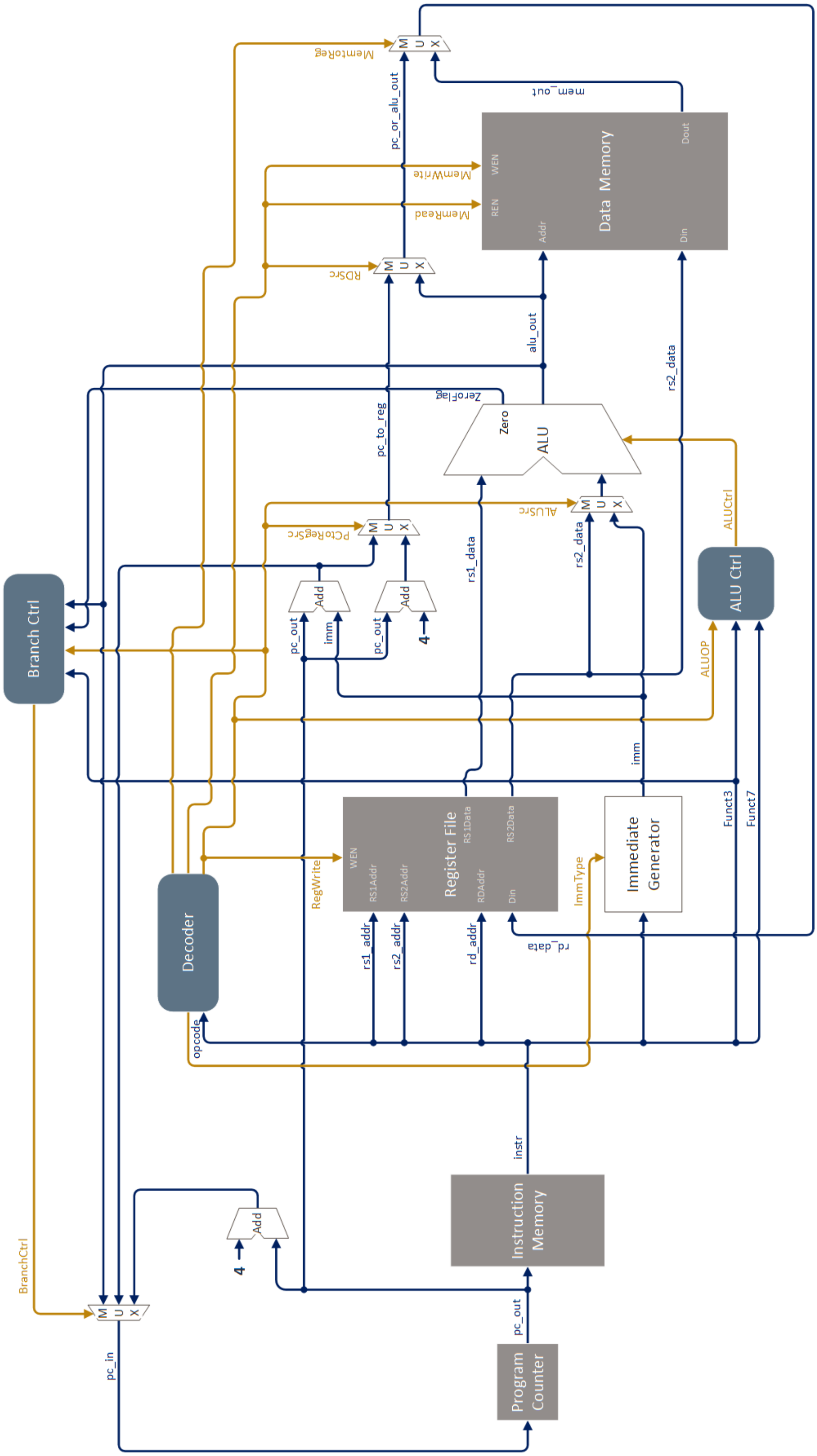
※You do not need to modify this module

e. <u>CPU module</u>

"CPU" is responsible for connecting wires between modules, please add your code to complete this module

※You should modify this module.

- **Reference Block Diagram**



- **Register File**

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	---
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	---
x4	tp	Thread pointer	---
x5	t0	Temporary / alternate link register	Caller
x6 – 7	t1 - 2	Temporaries	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10 - 11	a0 - 1	Function arguments / return values	Caller
x12 - 17	a2 - 7	Function arguments	Caller
x18 - 27	s2 - 11	Saved registers	Callee
x28 - 31	t3 - 6	Temporaties	Caller

- **Test Instruction**

a. <u>Memory layout</u>

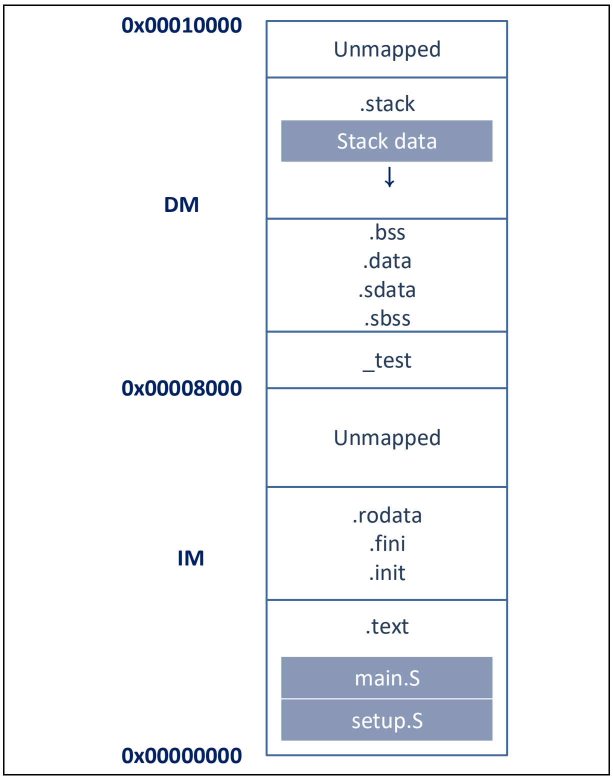


Figure 2. Memory layout

- .text: Store instruction code.
- .init & .fini: Store instruction code for entering & leaving the process.
- .rodata: Store constant global variable.
- .bss & .sbss: Store uninitiated global variable or global variable initiated as zero.
- .data & .sdata: Store global variable initiated as non-zero
- .stack: Store local variables

b. <u>main.S</u>

This will verify RISC-V instructions above (17 instructions).

c. <u>main0.hex & main1.hex & main2.hex & main3.hex</u>

Using the cross compiler of RISC-V to compile test program, and write result in verilog format. So you do not need to compile above program again.

Homework Requirements

1. Complete the CPU that can execute 17 instructions from the *RISC-V ISA* section.

2. Verify your CPU with the benchmark and take a snapshot (e.g. Figure 3)

```
# MIPS CPU
#          ****************************
#          **                        **        /|__/|
#          **   Congratulations !!   **       / O,O  |
#          **                        **     /_____   |
#          **   Simulation PASS!!    **    /^ ^ ^ \  |
#          **                        **   |^ ^ ^ ^ |w|
#          ***************  ************  \m___m__|_|
#
```

Figure 3. **S**napshot of correct simulation

   a. You can verify the execution results by checking waveforms.

3. Finish the Project Report.
   a. Complete the project report. The report template is provided "report.docx".

Important

When you upload your file, please make sure you have satisfied all the homework requirements, including the **File hierarchy**, **Requirement file** and **Report format**.

If you have any questions, please contact us.

Score

Your score is divided into two parts:

a. Functional Simulation (75%): TA will give you score based on the number of correct results. There are 15 test data, if you pass one of them, you will get 5 points.

b. Report (25%): You should take a screenshot of your result, and write your report in "report.docx".