Computer Organization 2017

HOMEWORK III

Overview

The goal of this homework is to help you understand how a pipelined CPU works and how to use Verilog hardware description language (Verilog HDL) to model electronic systems. In this homework, you need to implement a pipelined MIPS CPU that can execute all the instructions shown in the MIPS ISA section. 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. TAs will provide test fixtures that will run a MIPS program for the CPU. We will use hidden test fixtures to test your CPU. Please implement all the modules and use the test fixtures provided by TAs to verify the CPU. You also need to take a snapshot and explain it, including the wires, signals, etc. in your report.

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. If you copy your codes from someone else, you will not get any scores.
- 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

F740XXXXX_Report.docx (Project Report. The report template is already included. Follow the template to complete the report.)

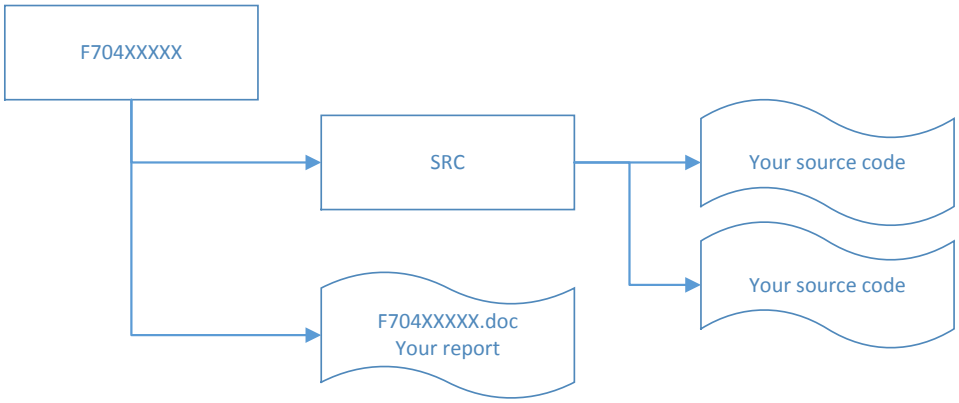

Figure 1. File hierarchy for homework submission

- **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.

Homework Description

1. The pipelined CPU in Figure 5 has five stages: instruction fetch (IF), instruction decode and register file read (ID), execution and address calculation (EX), memory access (MEM), and write back (WB).
2. There are four pipeline registers (*IF_ID, ID_EX, EX_M and M_WB*) between these five stages. These pipeline registers can store the data & control signals of the instructions and pass the information from one stage to next stage. These modules are implemented in IF_ID.v, ID_EX.v, EX_M.v and M_WB.v files. These modules are sequential circuits and they are triggered by negative clock edges.
3. The program counter is stored in the PC module. The file PC.v implements the program counter. It is also triggered by negative clock edges.
4. The instruction memory is used to store instructions and it is implemented in IM.v files. The data memory is used to store data and it is implemented in DM.v file. Both modules are triggered by positive clock edges. Their *write* operations are also triggered by positive clock edges.
5. You need to resolve the data hazard and control hazard problems in the pipelined CPU. These problems are handled by the Hazard detection unit (HDU) and Forwarding unit (FU).
6. The MIPS ISA is shown in the following. Figure 2 shows the R-type instructions in the MIPS ISA. Figure 3 shows the I-type instructions in the MIPS ISA. Figure 4 shows the J-type instructions in the MIPS ISA.

R-type Instruction Machine Code Format

opcode	Mnemonics	SRC1	SRC2	DST	funct	Description
000000	nop	00000	00000	00000	000000	No operation
000000	add	$Rs	$Rt	$Rd	100000	Rd = Rs + Rt
000000	sub	$Rs	$Rt	$Rd	100010	Rd = Rs – Rt
000000	and	$Rs	$Rt	$Rd	100100	Rd = Rs & Rt
000000	or	$Rs	$Rt	$Rd	100101	Rd = Rs | Rt
000000	xor	$Rs	$Rt	$Rd	100110	Rd = Rs ^ Rt
000000	nor	$Rs	$Rt	$Rd	100111	Rd = ~(Rs | Rt)
000000	slt	$Rs	$Rt	$Rd	101010	Rd = (Rs < Rt)?1:0
000000	sll		$Rt	$Rd	000000	Rd = Rt << shamt
000000	srl		$Rt	$Rd	000010	Rd = Rt >> shamt
000000	jr	$Rs			001000	PC=Rs
000000	jalr	$Rs			001001	R[31] = PC + 8 ; PC=Rs

Figure 2. R-type MIPS instructions

I-type Instruction Machine Code Format

opcode	Mnemonics	SRC1	DST	SRC2	Description
001000	addi	$Rs	$Rt	imm	Rt = Rs + imm
001100	andi	$Rs	$Rt	imm	Rt = Rs & imm
001010	slti	$Rs	$Rt	imm	Rt = (Rs < imm) ? 1 : 0
000100	beq	$Rs	$Rt	imm	If(Rs == Rt) PC=PC+4+imm
000101	bne	$Rs	$Rt	imm	If(Rs != Rt) PC=PC+4+imm
100011	lw	$Rs	$Rt	imm	Rt = Mem[Rs + imm]
100001	lh	$Rs	$Rt	imm	data = Mem[Rs + imm] Rt = data[15:0] <- Sign-extend 16bits
101011	sw	$Rs	$Rt	imm	Mem[Rs + imm] = Rt
101001	sh	$Rs	$Rt	imm	data <- Rt[15:0] Sign-extend 16bits Mem[Rs + imm] = Rt

Figure 3. **I**-type MIPS instructions

J-type Instruction Machine Code Format

opcode	Mnemonics	Address	Description
000010	j	jumpAddr	PC = jumpAddr
000011	jal	jumpAddr	R[31] = PC + 8 ; PC = jumpAddr

Figure 4. **J**-type MIPS instructions

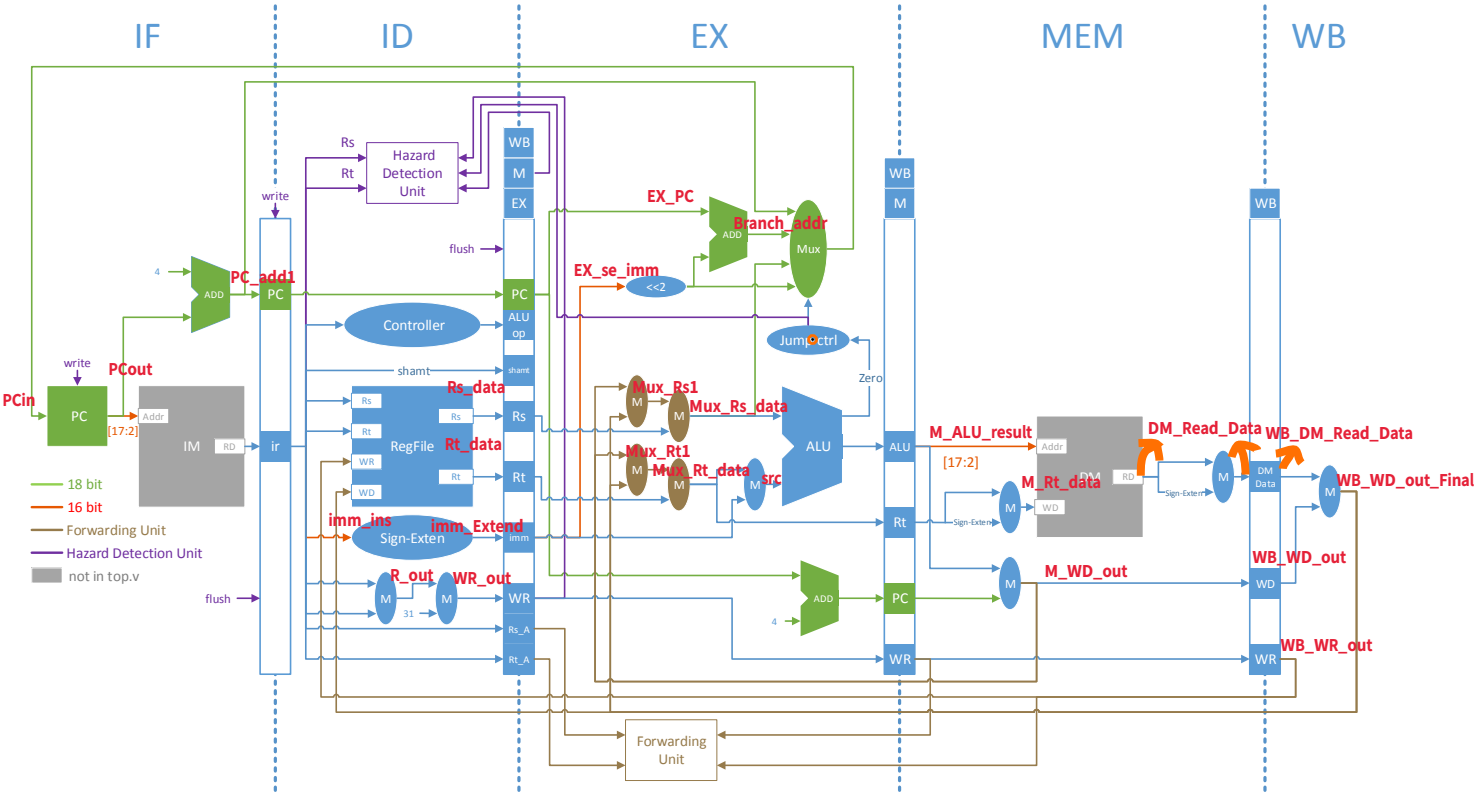

Figure 5. Pipelined datapaths

Homework Requirements

1. Complete the Pipeline CPU in Figure 5 that can execute all the instructions from the *MIPS ISA* section. You need to modify the following files.
   a. top.v (your Pipeline CPU)
   b. FU.v: The forwarding unit solve data hazards. You can find more details in Section 4.7 Data Hazard.
   c. HDU.v: Forwarding cannot solve all the data hazards. One kind of the data hazards that cannot be solved is load word data hazard. The hazard detection unit solves load word data hazard by stalling. The hazard detection unit also solve branch control hazards by stalling. You can find more details in Section 4.8 Control Hazard.
   d. IF_ID.v: Pipeline register between IF and ID.
   e. ID_EX.v: Pipeline register between ID and EX.
   f. EX_M.v: Pipeline register between EX and MEM.
   g. M_WB.v: Pipeline register between MEM and WB.

   The following files are from Homework 2-Single-cycle CPU. You may need to modify in this homework.
   a. Controller.v
   b. Regfile.v
   c. ALU.v
   d. PC.v: The program counter is triggered by negative edges, not positive edges.
   e. Jump_Ctrl.v

2. Verify your CPU with the provided test fixture. If the simulation result is correct, take a snapshot of the results (e.g. Figure 6)

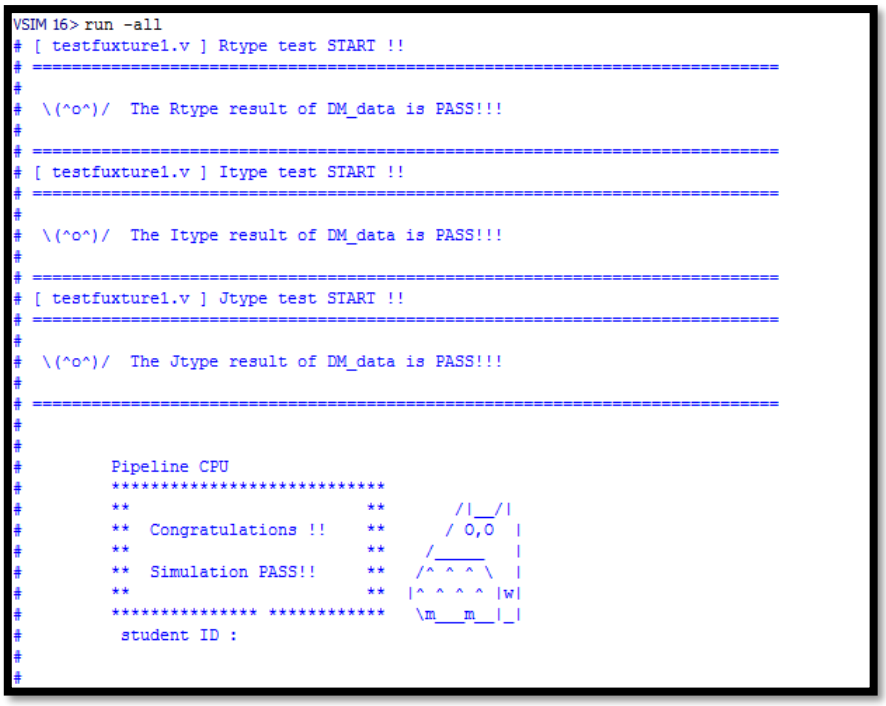

Figure 6. **S**napshot of correct simulation

3. Waveform snapshots and explanation of the waveforms
   a. Using waveforms to verify the execute results.
   b. Please annotate the waveform (as shown in Figure 7)

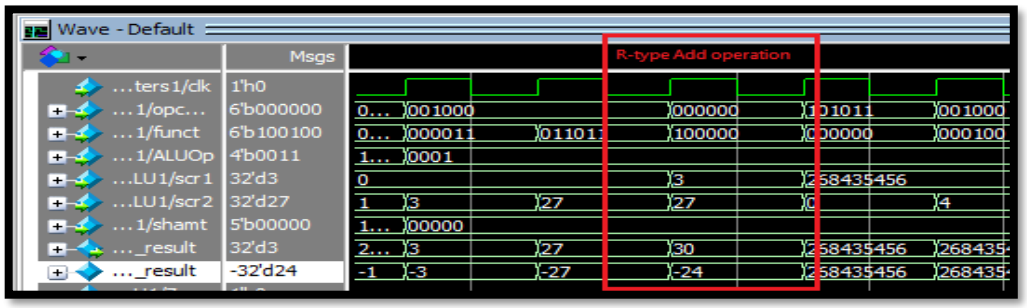

Figure 7. Waveform snapshots

   c. Show the following situations in the waveform snapshot and explain them as detailed as possible.
      i. Instructions with data forwarding: You need to find a situation with data forwarding. You can try R-type and one I-type instructions at least.
      ii. Load stall: describe why the load word instruction should stall and where the stall occurs (at waveforms).
      iii. Branch Delay (& Flush): describe why branch instruction should delay and where the Flush occurs (at waveforms).

4. Finish the Project Report.
    a. Complete the project report. The report template is provided.
    b. If your CPU datapath is different from Figure 5, submit the Verilog files of your CPU design and explain how it works.

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.