Computer Organization 2016

HOMEWORK III

Due date: 2016/6/9 23:59

The goal of this homework is to let the students be familiar with the MIPS instruction set and Verilog, a hardware description language (HDL) used to model electronic systems. In this homework, you need to implement a Pipeline MIPS CPU. Follow the instruction table in this homework and satisfy all the homework requirements. Please use the benchmark provided by TA to verify your CPU correctly. The verification tool is Modelsim.

General rules for deliverables

- This homework needs to be completed by INDIVIDUAL student. If your code is copied, you will not get any scores.
- Compress all files into a single zip file, and upload the compressed file to Moodle.
  - The file hierarchy

    F740XXXXX(your id)(folder)

    SRC(folder) * Store your source code

    F740XXXXX.docx(your Project Report)

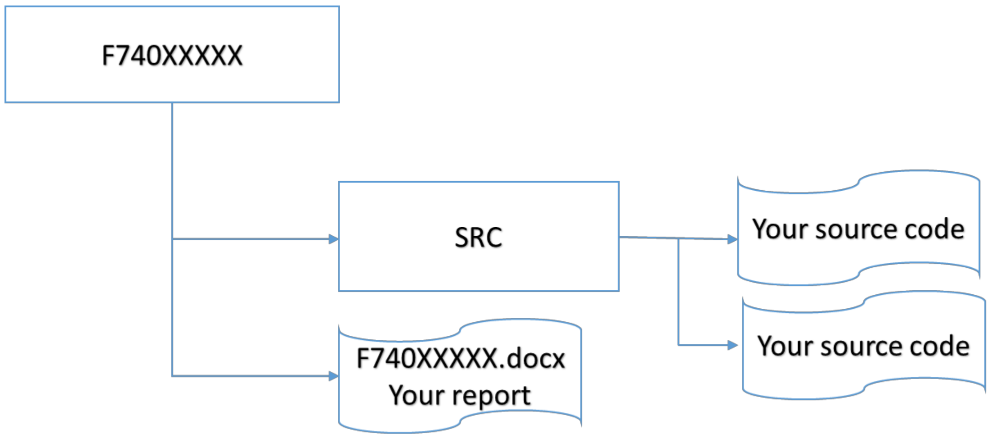


Fig.1 File hierarchy for homework submission

- **Important!** AVOID submitting your homework in the last minute. Late submission is not accepted.
- You should finish all the requirements (shown below) in this HW and Project report.
- Please use Project Report template on Archive to finish your Project report. HW3 Archive on Course website:
  [Computer Organization 2016 course website]()

Exercise

You need to implement a pipeline CPU that can execute all the instructions shown in the *MIPS ISA* section. In addition, you need to verify your CPU by using Modelsim. Note that, TAs will provide a simple architecture for the CPU and you just need to implement some incomplete/missing modules, e.g.top, controller, hdu ,etc.

Please finish all the modules, use the benchmark provided by TAs to verify the CPU, take a snapshot, and finally explain the snapshot, including the wires, signals …… in your report.

MIPS ISA

R Type

Assembler Syntax

instruction	rd	rs	rt

Machine code Format

opcode	rs	rt	rd	shamt	funct
31 26	25 21	20 16	15 11	10 6	5 0

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

I Type

Assembler Syntax

instruction	rt	rs	imm

Machine code Format

opcode	rs	rt	immediate
31 26	25 21	20 16	15 0

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]
101011	sw	$Rs	$Rt	imm	Mem[Rs + imm] = Rt

J Type

Assembler Syntax

instruction	Target(label)

Machine code Format

opcode	address
31 26	25 0

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

Pipeline CPU Datapath

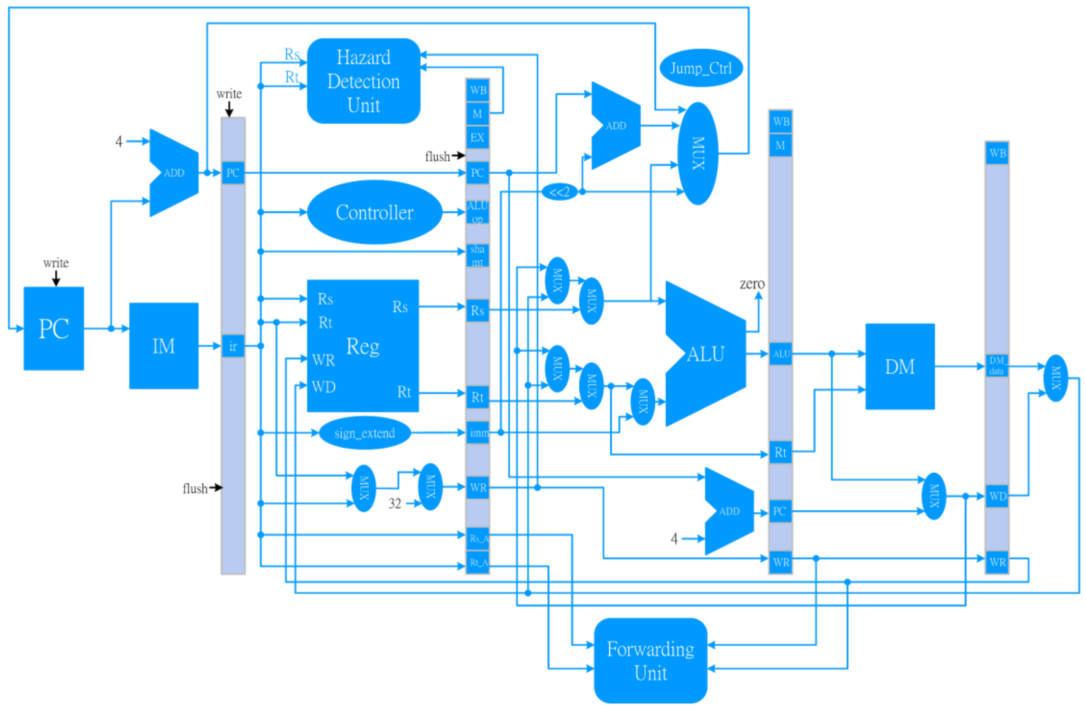


Fig.2 Pipeline CPU DataPath reference

Description

1. The Pipeline CPU has 5 stages : *IF, ID, EX, MEM, WB*. There are registers between these 5 stages in order to register the data & control signals of the instruction in each stage.
2. These modules with these registers are called: **IF_ID, ID_EX, EX_M, M_WB**. These modules are sequential circuit.
   They are triggered by **negedge clk.** So is **PC.v.**
3. The write of IM, DM, Regfile is triggered by **posedge clk.**
4. You need to solve the hazard problems in the pipeline cpu
   I. Data Hazards
   II. Branch Hazards
   These problems are handled by the Hazard detection unit(HDU) and Forwarding unit(FU). Now you're asked to complete these modules.
5. Port description:

HDU.v		
port	I/O	Description
ID_Rs	input	Rs input source from **IF_ID pipe**
ID_Rt	input	Rt input source from **IF_ID pipe**
EX_WR_out	input	The Write Register from **ID_EX pipe**
EX_MemtoReg	input	MemtoReg control signal from **ID_EX pipe**
EX_JumpOP	input	The Jump operation control signal (used to judge branch or not)
PCWrite	output	PC write control (used for stall)
IF_IDWrite	output	IF_ID pipe write signal (used for stall)
IF_Flush	output	**Flush signal** of IF_ID pipe you can refer to IF_ID.v to see how to use this signal
ID_Flush	output	**Flush signal** of ID_EX pipe you can refer to ID_EX.v to see how to use this signal
Branch_Flush	output	Flush signal of Branch (redundant)*1
Load_wait	output	Stall signal of Load (redundant)*1

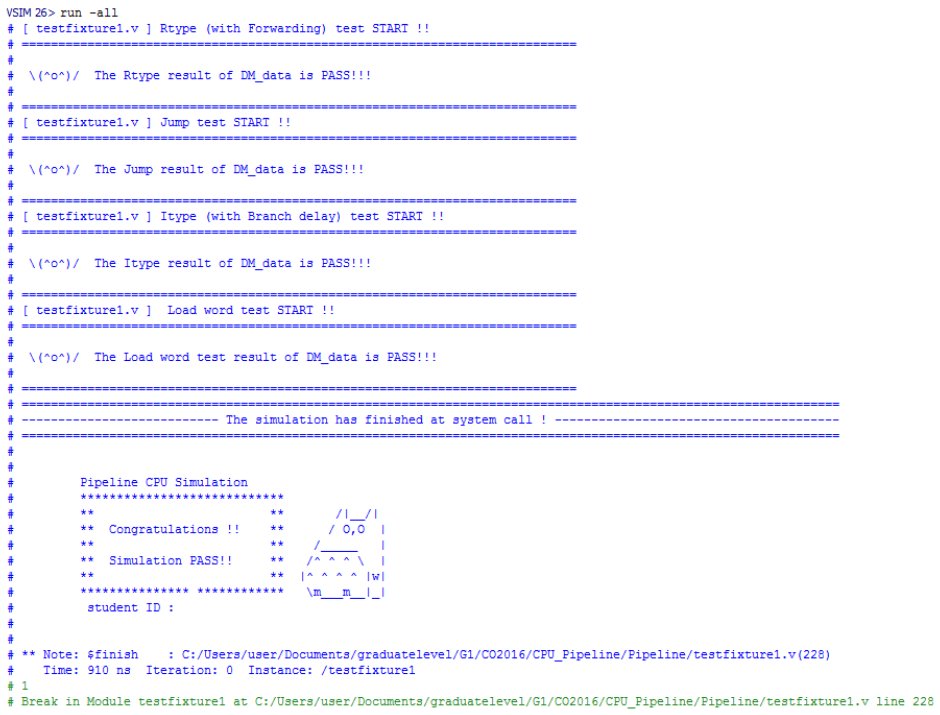
*1The Branch_Flush and Load_wait are not used in this project, so you can ignore these two signals.

FU.v		
port	I/O	Description
EX_Rs	input	Rs input source from **ID_EX pipe**
EX_Rt	input	Rt input source from **ID_EX pipe**
M_RegWrite	input	The RegWrite signal from **EX_M pipe**
M_WR_out	input	The Write Register from **EX_M pipe**

WB_RegWrite	input	The RegWrite signal from **M_WB pipe**
WB_WR_out	input	The Write Register from **M_WB pipe**
enF1	output	Enable Forward signal - select origin Rs or the forward data
enF2	output	Enable Forward signal - select origin Rt or the forward data
sF1	output	Select Forward signal - select the source of Rs from M or WB
sF2	output	Select Forward signal - select the source of Rt from M or WB

Homework Requirements

1. Please implement these modules:
   I. **top.v** (wire declarations and connections)
   II. Controller - **Controller.v**
   III. Hazard detection unit **- HDU.v**
   IV. Forwarding unit **- FU.v**
2. Verify your CPU with the benchmark and take a snapshot (e.g. Fig.3)



```
VSIM 26> run -all
# [ testfixture1.v ] Rtype (with Forwarding) test START !!
# ========================================================================
#
#  \(^o^)/  The Rtype result of DM_data is PASS!!!
#
# ========================================================================
# [ testfixture1.v ] Jump test START !!
# ========================================================================
#
#  \(^o^)/  The Jump result of DM_data is PASS!!!
#
# ========================================================================
# [ testfixture1.v ] Itype (with Branch delay) test START !!
# ========================================================================
#
#  \(^o^)/  The Itype result of DM_data is PASS!!!
#
# ========================================================================
# [ testfixture1.v ]  Load word test START !!
# ========================================================================
#
#  \(^o^)/  The Load word test result of DM_data is PASS!!!
#
# ========================================================================
# ==========================================================================================================
# -------------------------- The simulation has finished at system call ! --------------------------------------
# ==========================================================================================================
#
#
#         Pipeline CPU Simulation
#         ****************************
#         **                        **       /|__/|
#         **  Congratulations !!    **      / 0,0  |
#         **                        **    /_____   |
#         **  Simulation PASS!!     **   /^ ^ ^ \  |
#         **                        **  |^ ^ ^ ^ |w|
#         *************** ************   \m___m__|_|
#          student ID :
#
#
# ** Note: $finish    : C:/Users/user/Documents/graduatelevel/G1/CO2016/CPU_Pipeline/Pipeline/testfixture1.v(228)
#    Time: 910 ns  Iteration: 0  Instance: /testfixture1
# 1
# Break in Module testfixture1 at C:/Users/user/Documents/graduatelevel/G1/CO2016/CPU_Pipeline/Pipeline/testfixture1.v line 228
```

Fig.3 Simulation Successful snapshot

3. Take snapshot
   I. Using waveform to verify the execute results.
   II. Please annotate the waveform (as shown in Fig. 4)

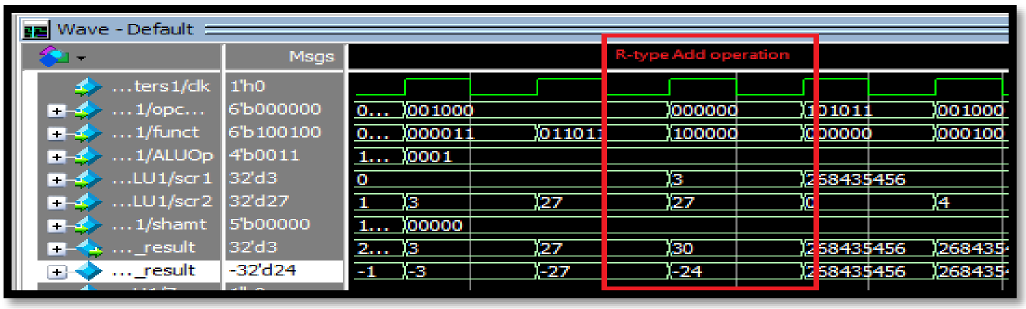


Fig.4 Instruction waveform snapshot

III. At least list the following situations in the waveform snapshot. Then explain them as detailed as possible.

   i. Instructions with forwarding

      1. one Rtype and one Itype at least

   ii. Load stall

      1. Describe why load word should do stalling and where the stall occurs(at waveform)

   iii. Branch Delay (& Flush)

      1. Describe why branch instruction should delay and where the Flush occurs(at waveform)

4. Finish the Project Report.
5. Bonus

   I. Use Qtspim to compile your HW1 assembly program, and run the compiled machine code on the CPU you have developed in HW3.

   II. The compiled machine code should be formatted as the origin IM_data.dat. That means the beginning of every line should be the machine code rather than the PC.

   III. Take a snapshot of reg $t0 waveform and explain your CPU works correctly.

   IV. The IM_data you submit should be the original one, DO NOT submit the IM_data of HW1 you produced! Just write the result in your report.

TIPS

- **Please refer to the lab2 tutorial and build your project.**

Important

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

If you have any questions, please contact us.