Computer Organization Project 3

Part1 :Implement a 5-stage pipelined processor with R-format instructions.

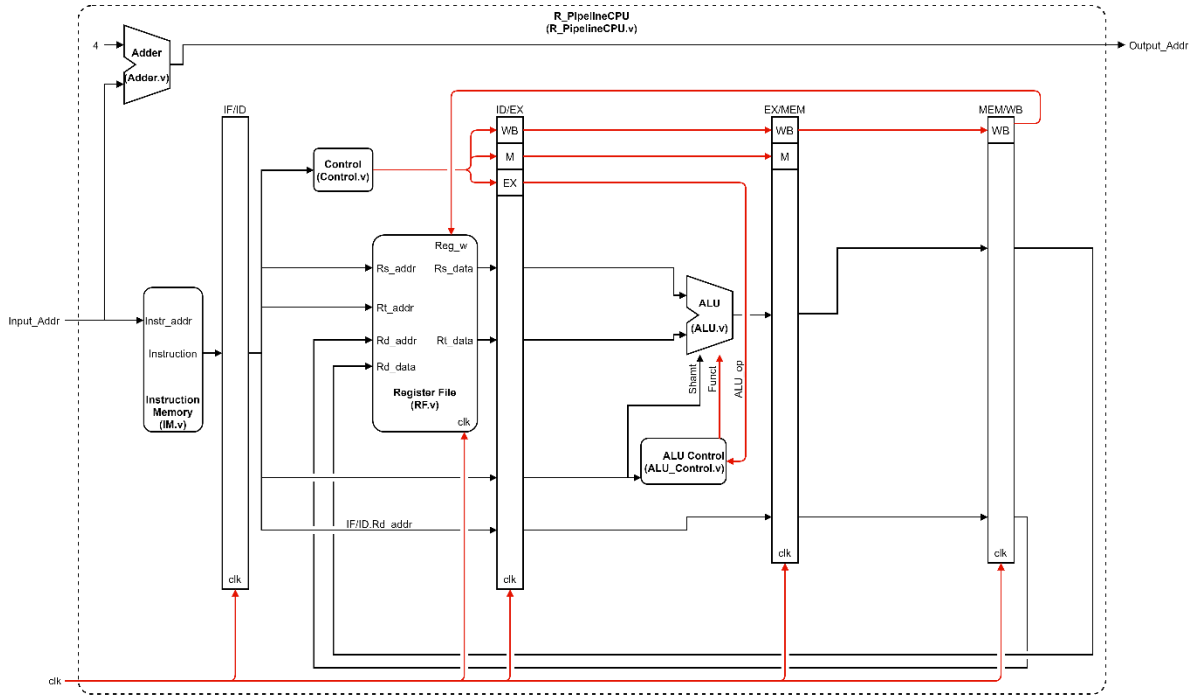


Figure 1：Architecture of a 5-stage pipelined processor with R-format instructions

Implements a 32-bits processor and supports the following R-format instructions.

Instruction	Example	Meaning	OpCode	Funct_ctrl	Funct
Add unsigned	Addu $Rd, $Rs, $Rt	$Rd = $Rs + $Rt	000000	001011	001001
Sub unsigned	Subu $Rd, $Rs, $Rt	$Rd = $Rs − $Rt	000000	001101	001010
Shift left logical	Sll $Rd, $Rs, Shamt	$Rd = $Rs << Shamt	000000	100110	100001
Shift left logical variable	Sllv $Rd, $Rs, $Rt	$Rd = $Rs ≪ $Rt	000000	110110	110101

Note: Please refer to HW1 for the method of converting text instructions into 32-bits execution codes.

Note: When executing the R-format instruction, ALU_op is set as "10". Then, the ALU Control recognizes the "Funct_ctrl" and converts the corresponding ALU function code "Funct".

I/O Interface

```
module R_PipelineCPU (
        output  wire    [31:0]  Output_Addr,
        input   wire    [31:0]  Input_Addr,
        input   wire            clk
);
```

Part2 :Implement a 5-stage pipelined processor with I-format instructions.

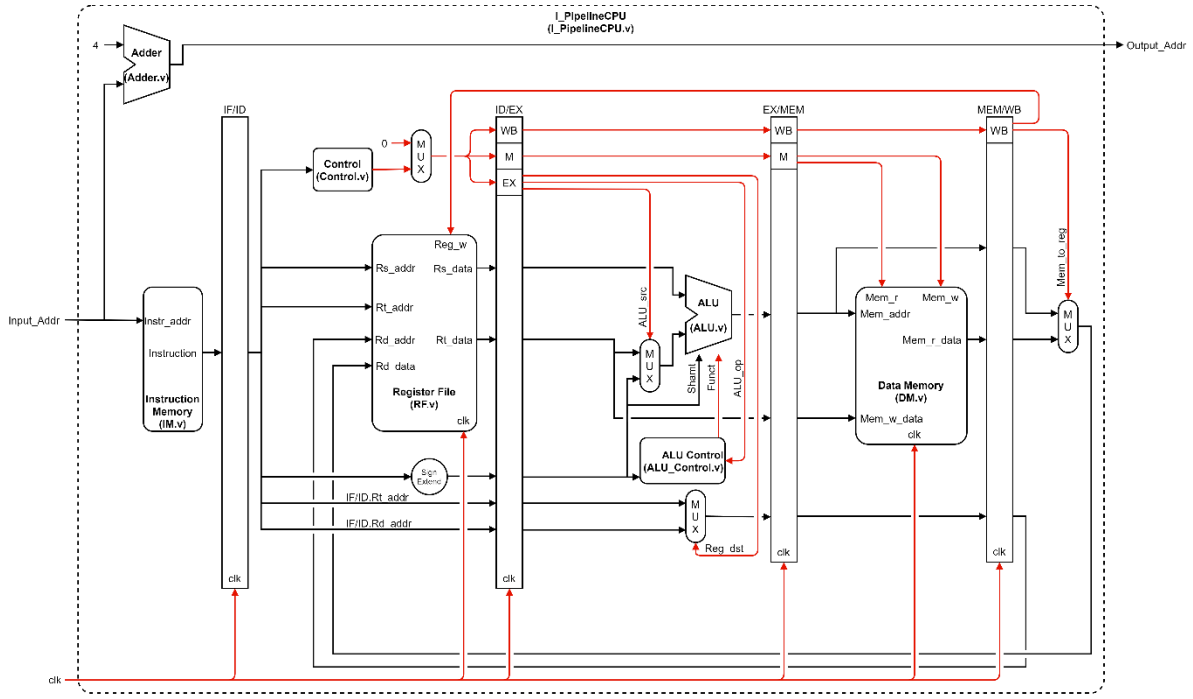


Figure 2：Architecture of a 5-stage pipelined processor with R-format and I-format instructions

Implement a 32-bits processor that supports the R-format of the previous part and supports the following I-format instructions.

Instruction	Example	Meaning	OpCode	Funct
Sub.imm.unsigned	Subiu $Rt, $Rs, Imm.	$Rt = $Rs – Imm.	001101	001010
Store word	Sw $Rt, Imm. ($Rs)	Mem.[$Rs+Imm.] =$Rt	010000	001001
Load word	Lw $Rt, Imm. ($Rs)	$Rt =Mem.[$Rs+Imm.]	010001	001001
Set Less Than Imm	Slti $Rt, $Rs, Imm.	If $Rs < Imm. $Rt=1 Else $Rt=0	101010	101010

Note: When executing the I-format instruction, ALU_op is set as "00", "01", "11". Then, ALU Control ignores the "Funct_ctrl", and triggers the ALU to perform "addition", "subtraction" or "set less than immediate value" and outputs the corresponding "Funct".

I/O Interface

```
module I_PipelineCPU (
        output  wire    [31:0]  Output_Addr,
        input   wire    [31:0]  Input_Addr,
        input   wire            clk
);
```

Part3 :Implement a 5-stage pipelined processor with forwarding and hazard detection.

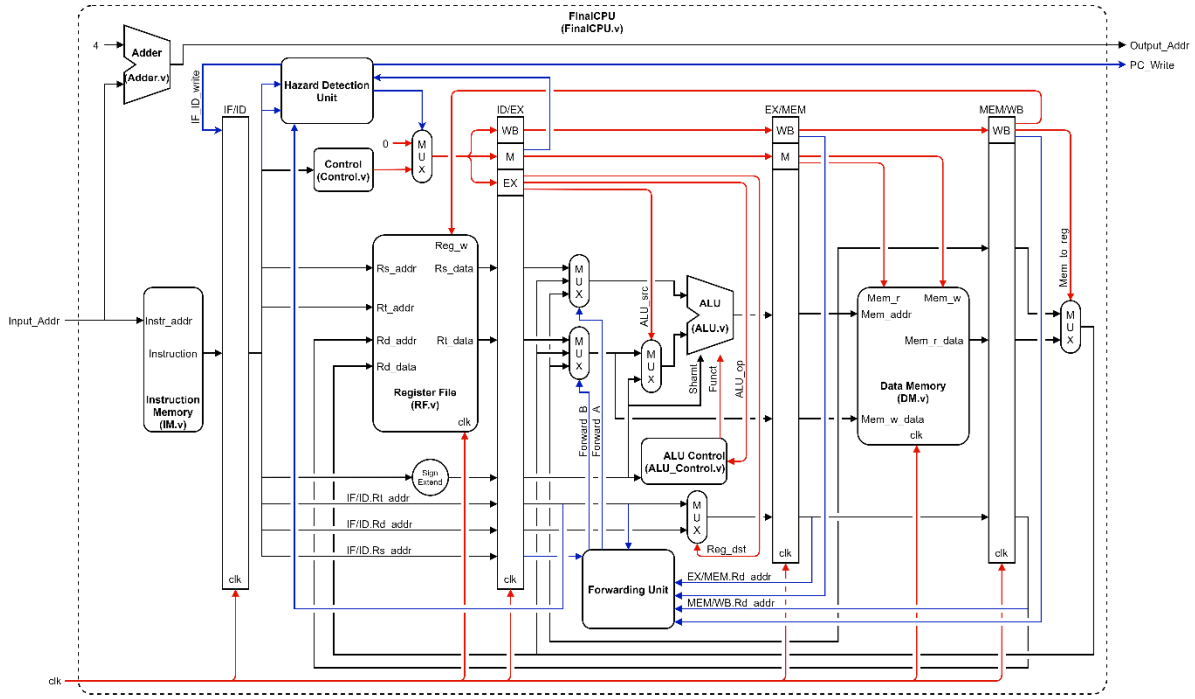


Figure 3：Architecture of a 5-stage pipelined processor supporting forwarding and hazard detection

Implement a 32-bit processor to support R-format and I-format of the first two parts, and support forwarding and hazard detection.

I/O Interface

```
module FinalCPU (
        output  wire            PC_Write,
        output  wire    [31:0]  Output_Addr,
        input   wire    [31:0]  Input_Addr,
        input   wire            clk
);
```

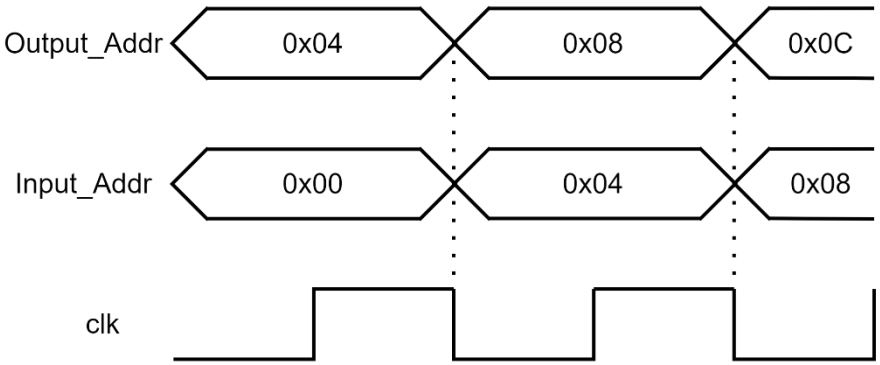
Analyze the FinalCPU, and ensure to include your CPU clock period and screenshots of the timing, utilization, and power results in the report.

Testbench Description

a. **Initialize**
Execute Testbench ("tb_R_ PipelineCPU.v", "tb_I_ PipelineCPU.v", "tb_ FinalCPU.v") to initializes Instruction Memory, Register File, and Data Memory, respectively, according to "/testbench/IM.v", "/testbench/RF.v", "/testbench/ DM.v".

b. **Clock**
Generate a periodic clock (clk) to drive the CPU module in the testbench.

c. **Addressing and Termination**
For the Input_Addr signal, the Testbench is initialized to 0 and Output_Addr is assigned to Input_Addr before each negative edge of the clk.（Part 3 will use PC_Write as the control signal, and Input_Addr will be updated when it is "1"）. The Testbench will execute until Input_Addr is greater than or equal to the maximum addressing space, at which point it will output the current contents of the registers and memory ("/testbench/RF.out", "/testbench/DM.out") for analysis program correctness. The following figure shows the basic waveform of Testbench action:



Note: If the system Output_Addr fails or the program enters an infinite loop, please terminate the simulation and manually identify the problem.

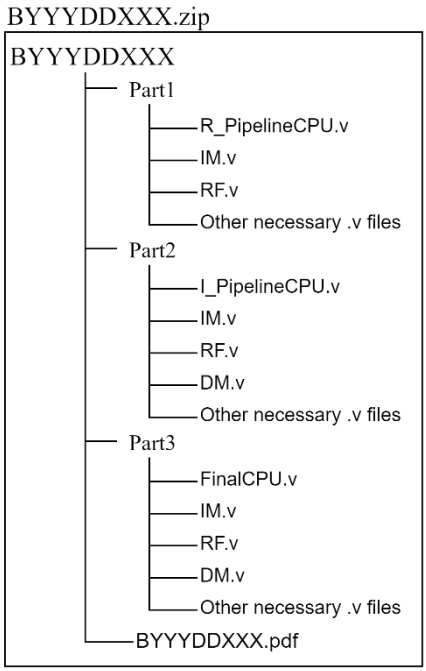
Submission

Report (BYYYDDXXX.pdf) : (no more than 30 pages)

a. Cover.
b. Screenshots and descriptions of each module code.
c. Describes the custom test program and analyzes its results ("/testbench/RF.out", "/testbench/DM.out") in each part.
d. Analysis of FinalCPU in Part 3.
e. Export it as a PDF format and name the file by the student ID - "BYYYDDXXX.pdf ".

Compressed files (BYYYDDXXX.zip):

- Report (BYYYDDXXX.pdf)
- Part1
    a. R_PipelineCPU.v
    b. IM.v
    c. RF.v
    d. Other necessary .v files
- Part2
    a. I_PipelineCPU.v
    b. IM.v
    c. RF.v
    d. DM.v
    e. Other necessary .v files
- Part3
    a. FinalCPU.v
    b. IM.v
    c. RF.v
    d. DM.v
    e. Other necessary .v files



- Note: Please ensure that all your program files and PDF files are directly placed in the zipped file, rather than being wrapped in a single folder.

Score :

- Part1 (30%)
    1) Each testbench gets 5 points. (20%)
    2) Screenshots of each program(.v), and describe how you implement it. (5%)
    3) The execution results of the custom programs in each part, screenshots, and explanations(5%)
- Part2 (30%)
    1) Each testbench gets 5 points. (20%)
    2) Screenshots of each program(.v), and describe how you implement it. (5%)
    3) The execution results of the custom programs in each part, screenshots, and explanations(5%)
- Part3 (40%)
    1) Each testbench gets 5 points. (20%)
    2) Screenshots of each program(.v), and describe how you implement it. (5%)
    3) The execution results of the custom programs in each part, screenshots, and explanations(5%)
    4) Your CPU clock period and Screenshots of the timing, utilization, and power results(10%)
- Penalty for any executable error.
- No plagiarism.

Submission time: Upload to Moodle before 13:00 on 2024/05/30