Computer Organization Project 3

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

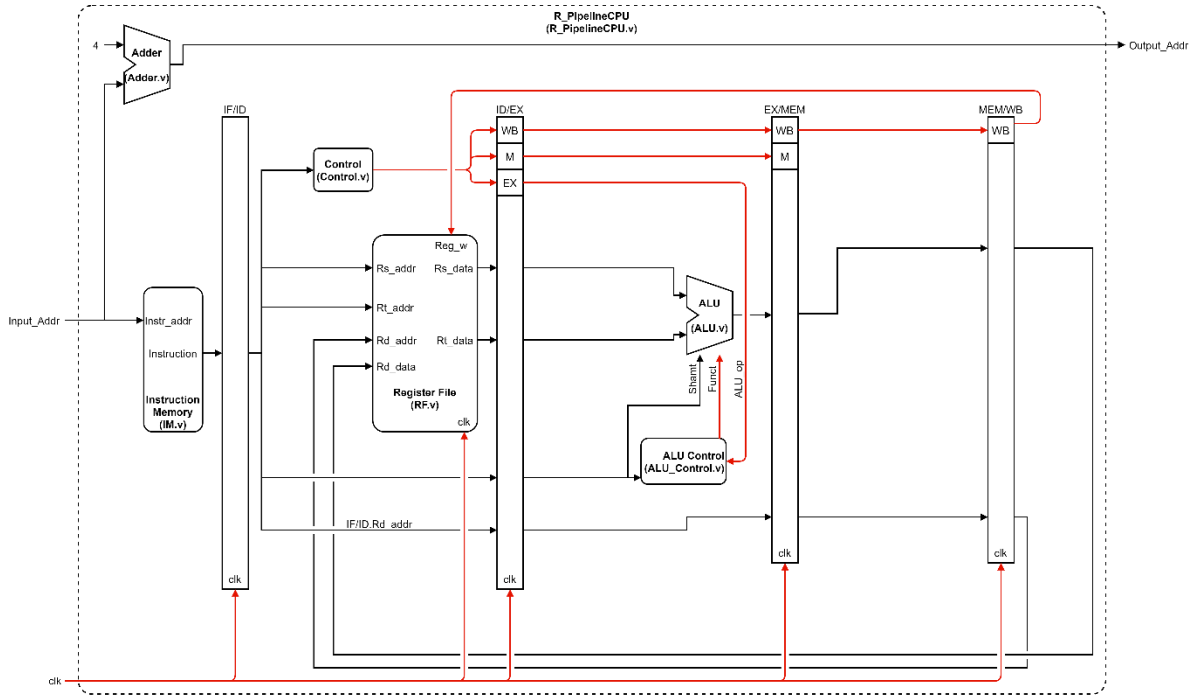


Figure 1：Architecture of a 5-stage pipelined processor with R-format instructions

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	100001	001001
Sub unsigned	Subu $Rd, $Rs, $Rt	$Rd = $Rs − $Rt	000000	100011	001010
Shift left logical	Sll $Rd, $Rs, Shamt	$Rd = $Rs << Shamt	000000	000000	100001
OR	Or $Rd, $Rs, $Rt	$Rd = $Rs | $Rt	000000	100101	100101

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.

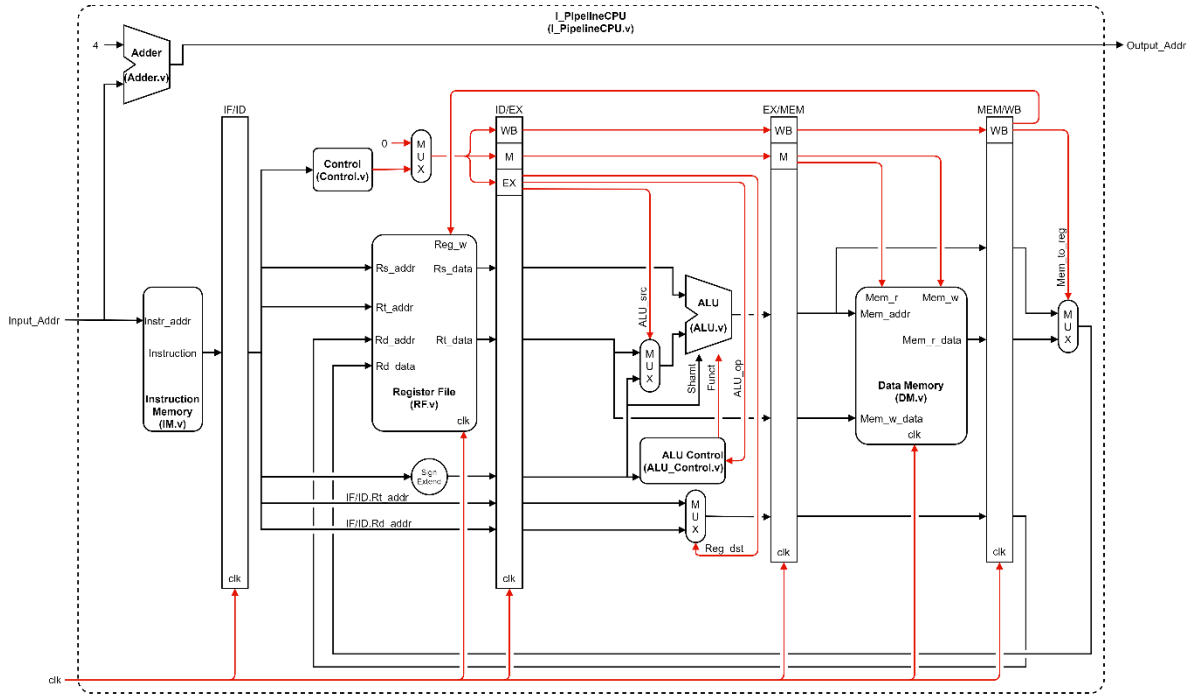


Figure 2：Architecture of a 5-stage pipelined processor with R-format and I-format instructions

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
Add imm unsigned	addiu $Rt, $Rs, Imm.	$Rt = $Rs + Imm.	001001	001001
Store word	Sw $Rt, Imm. ($Rs)	Mem.[$Rs+Imm.] =$Rt	101011	001001
Load word	Lw $Rt, Imm. ($Rs)	$Rt =Mem.[$Rs+Imm.]	100011	001001
Or Immediate	Ori $Rt, $Rs, Imm.	$Rt = $Rs | Imm.	001101	100101

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 “or” 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.

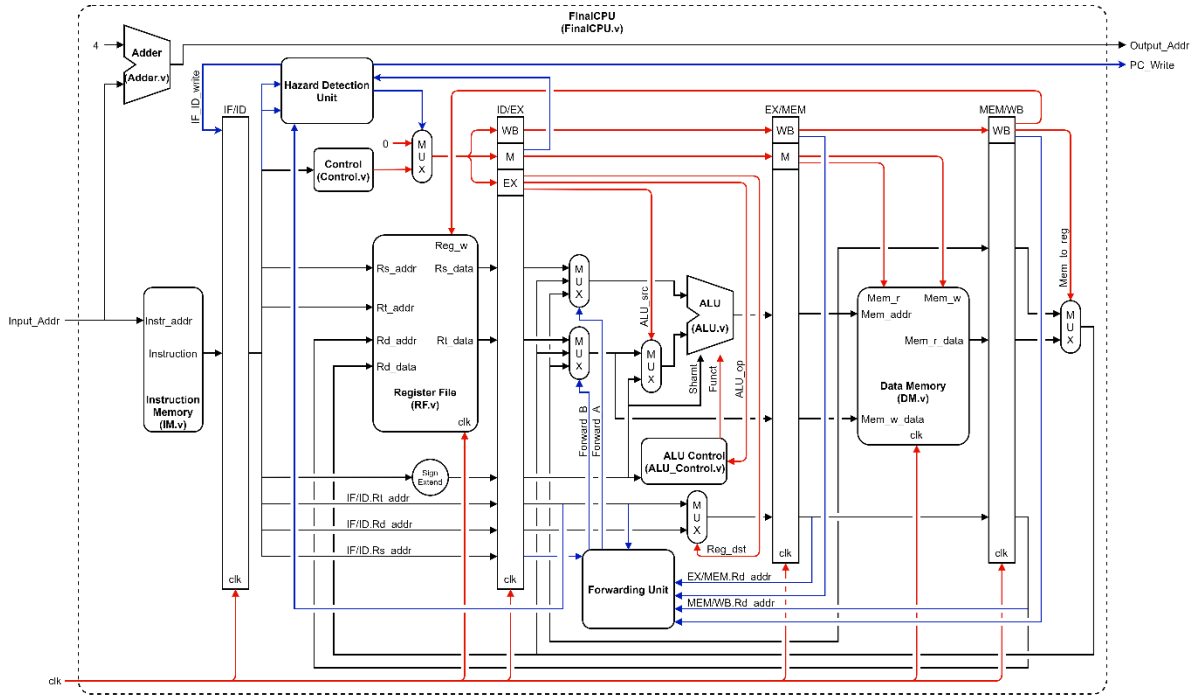


Figure 3：Architecture of a 5-stage pipelined processor supporting forwarding and hazard detection

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);
```

```
module RF (
    output  wire    [31:0]  RsData,
    output  wire    [31:0]  RtData,
    input   wire    [4:0]   RsAddr,
    input   wire    [4:0]   RtAddr,
    input   wire    [4:0]   RdAddr,
    input   wire    [31:0]  RdData,
    input   wire            RegWrite,
    input   wire            clk );
```

```
module IM (
    output  wire    [31:0]  Instr,
    input   wire    [31:0]  InstrAddr );
```

```
module DM (
    output  reg     [31:0]  MemReadData,
    input   wire    [31:0]  MemAddr,
    input   wire    [31:0]  MemWriteData,
    input   wire            MemWrite,
    input   wire            clk );
```

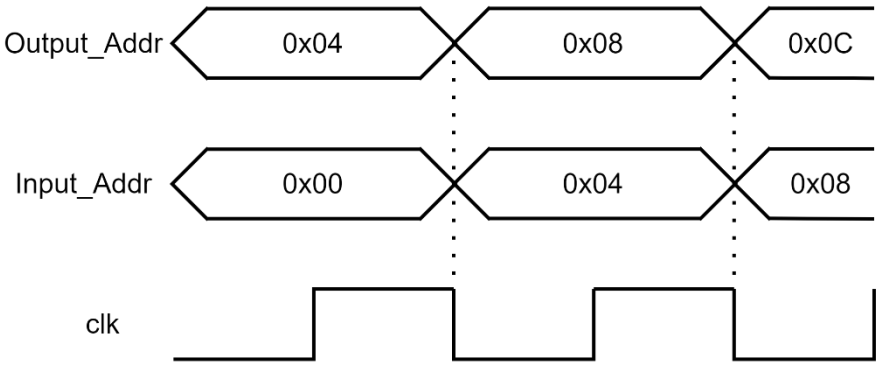
Analyze the FinalCPU and, in your report, include its clock period along with screenshots comparing timing, area, and power results against those of the SimpleCPU (PA2).

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.

OpenROAD Description of Part 3

For the area optimization section, the areas of RF.v, IM.v, and DM.v will be replaced. Please focus on the synthesis of the other modules.

a. ~/OpenROAD-flow-scripts/flow

```
|---------- designs
|         |------------ nangate45
|                   |------------ FinalCPU
|                             |------------ FinalCPU.v
|                             |------------ RF.v
|                             |------------ IM.v
|                             |------------ DM.v
|                             |------------ Control.v
|                             |------------ Other necessary .v files
|                             |------------ config.mk
|                             |------------ constraint.sdc
|---------- reports
|---------- Makefile
          (DESIGN_CONFIG ?= ./designs/nangate45/FinalCPU/config.mk)
```

Submission

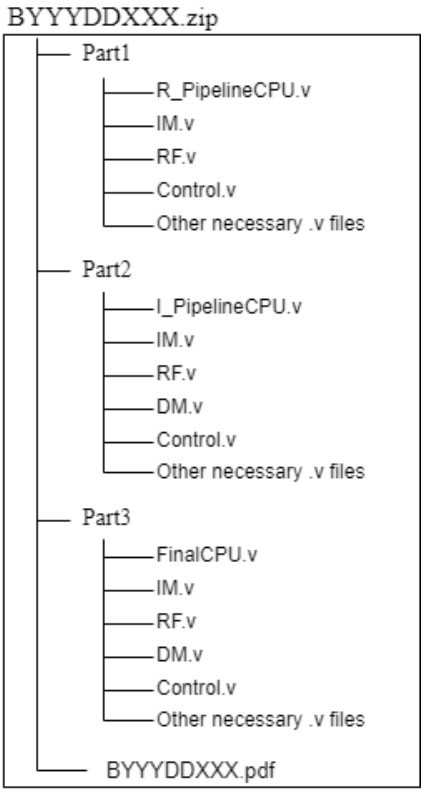
Report (BYYYDDXXX.pdf) : (no more than 30 pages)

a. Cover (Project Name, Student ID, Name, Area, Slack).
b. Descriptions of how you implement each module.
c. Describes the custom test and analyzes its results in each part ("/testbench/RF.out", "/testbench/DM.out"), then compare the timing, area and power against both the SimpleCPU (PA2) and the FinalCPU (PA3).
d. Memory Rethinking: If you were required to implement a multi-level cache in a pipelined CPU, how would you revise the datapath or design? Only a brief description or idea is needed.
e. Conclusion and insights.

※ Convert the report to PDF and name it the student ID - "BYYYDDXXX.pdf".

Compressed files (BYYYDDXXX.zip):

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



- Please do not include the testbench.
- 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 (20%): Each pattern gets 5 points.
- Part2 (24%): Each pattern gets 3 points.
- Part3 (16%): Each pattern gets 2 points.
- Area and speed optimization of Part3 (15%).
- Report (25%).
- Follow naming rules and file formats.
- No plagiarism.

Submission time: Upload to Moodle before 12:00 on 2024/05/29