Computer Organization Project 2

Part I ：Implement a single cycle processor with R-format instructions

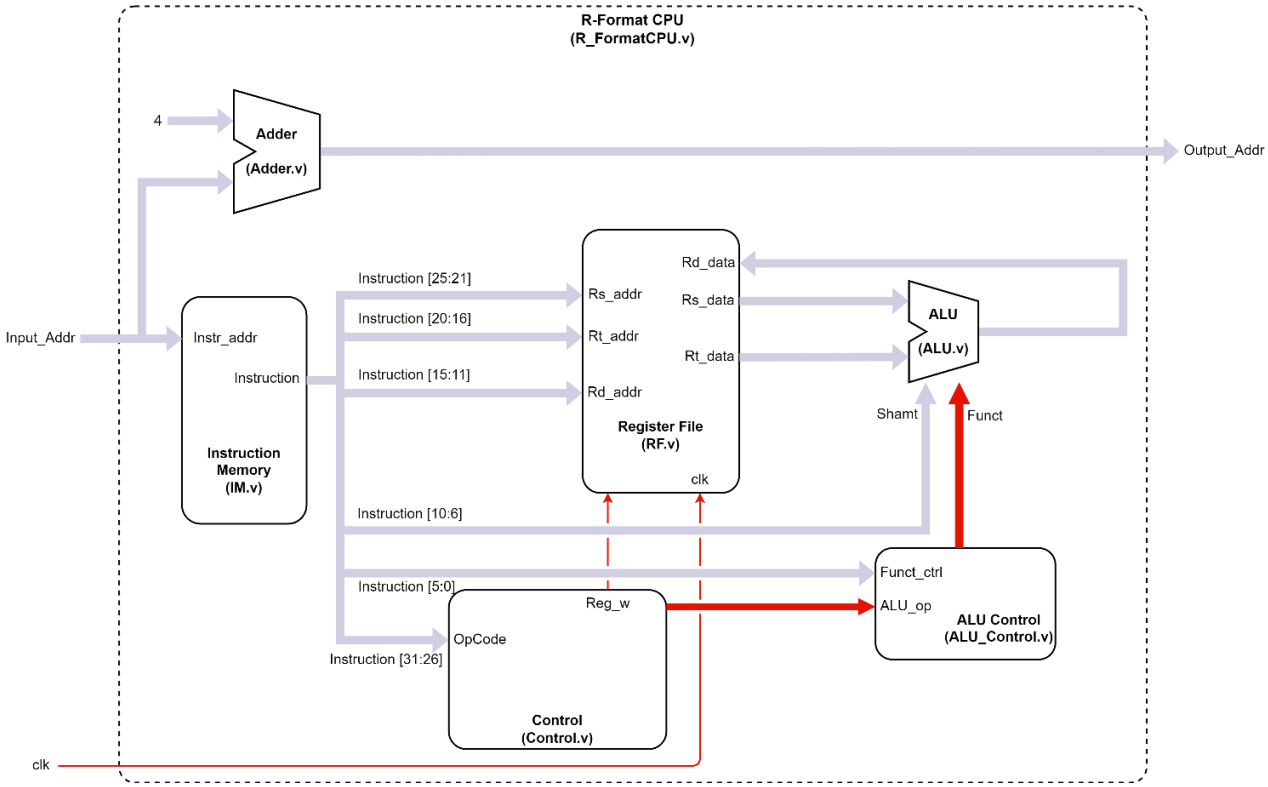


Figure 1：Architecture of a single cycle processor with R-format instructions

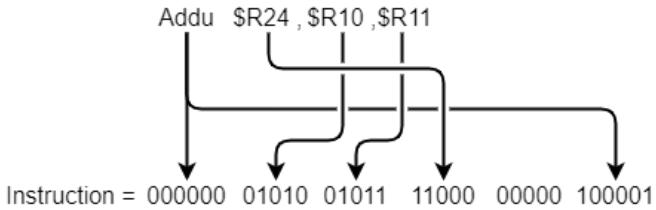
Implements a 32-bits processor and supports the following R-format instructions.

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".

Convert Instruction to Binary



I/O Interface

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

Part II ：Implement a single cycle processor with R-format and I-format instructions

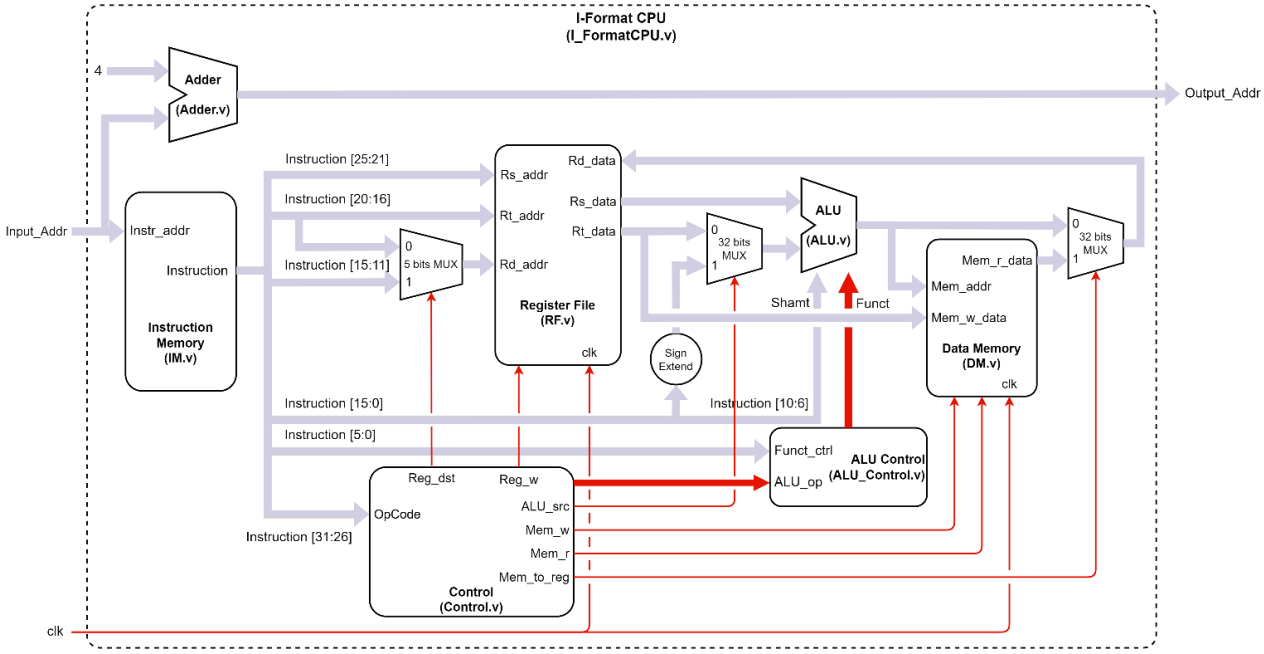


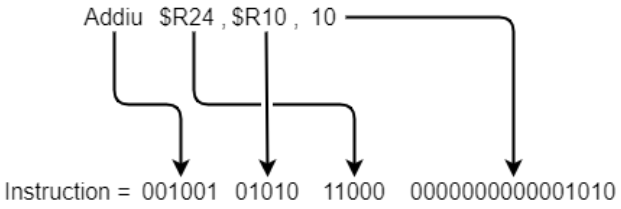
Figure 2：Architecture of a single cycle processor with R-format and I-format instructions

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".

Convert Instruction to Binary



I/O Interface

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

Part III ： Implement a single cycle processor with branch and jump instructions

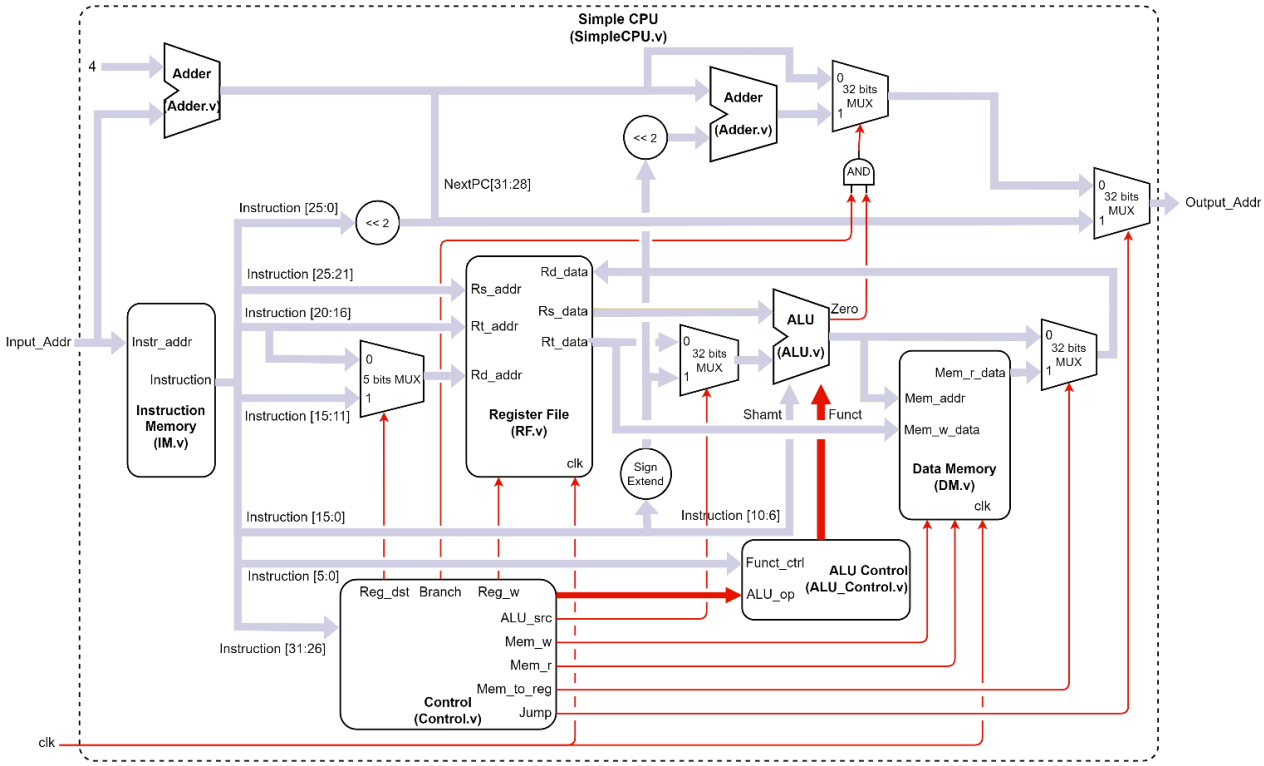


Figure 3：Architecture of a single cycle processor with branch and jump instructions

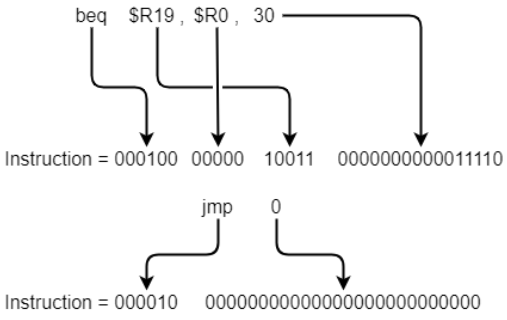
Implement a 32-bits processor that supports the first two parts of R-format and I-format, and supports the following branch and jump instructions.

Instruction	Example	Meaning	OpCode	Funct
Branch on equal	Beq $Rs, $Rt, Imm.	if ($Rs ≡ $Rt) Output_Addr = Input_Addr + 4 + Imm.∗ 4	000100	001010
Jump	J Imm.	Output_Addr = NextPC[31: 28] | Imm.∗ 4	000010	001010

Note: When executing the branch instruction, ALU_op is set as "01". Then, ALU Control ignores the "Funct_ctrl", triggers the ALU to perform "subtraction" and outputs the corresponding "Funct".

Note: NextPC = Input_addr + 4; " | " is OR operation.

Convert Instruction to Binary



I/O Interface

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

I/O Interface

```
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            MemRead,
   input   wire            clk
);
```

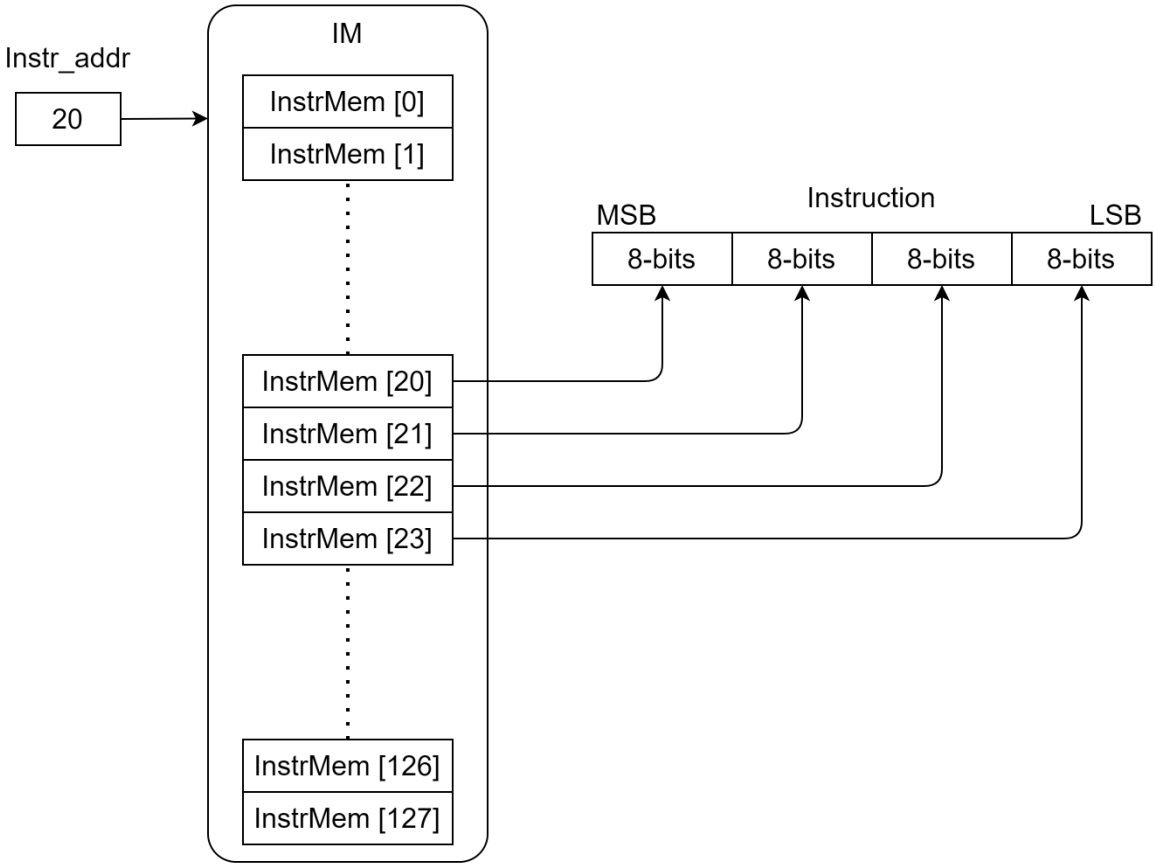
Register File

Different from HW1, the RF in this PA support dual-bus parallel read, and one write bus. The bit width and number of registers are the same as that in HW1, with 32 registers having a width of 32- bits. Its initial value is set by "/testbench/RF.dat".

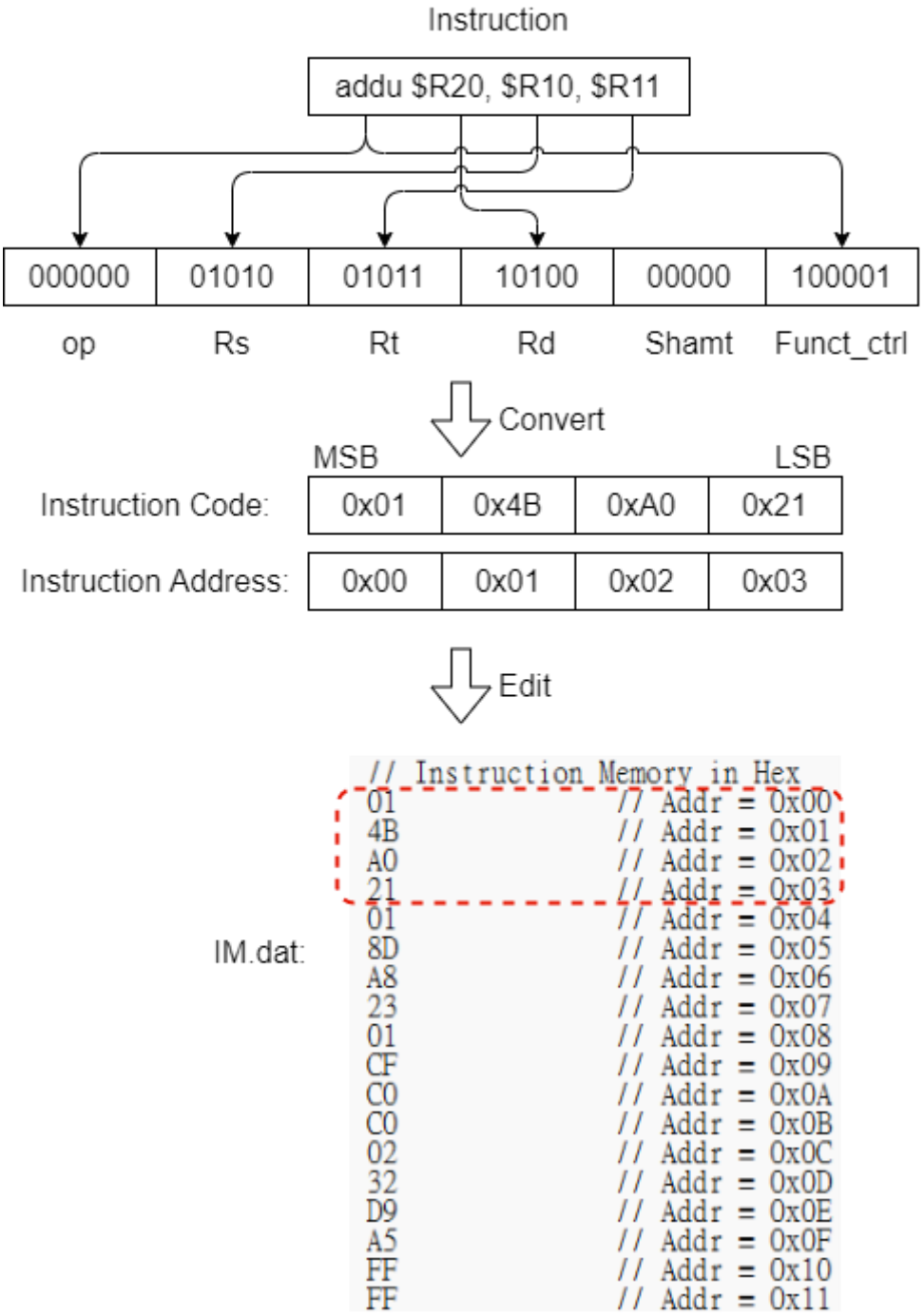
Instruction Memory

It consists of 128x8 bits memory with Big-endian. Its initial value is set by "/testbench/IM.dat".

a. Instruction reading



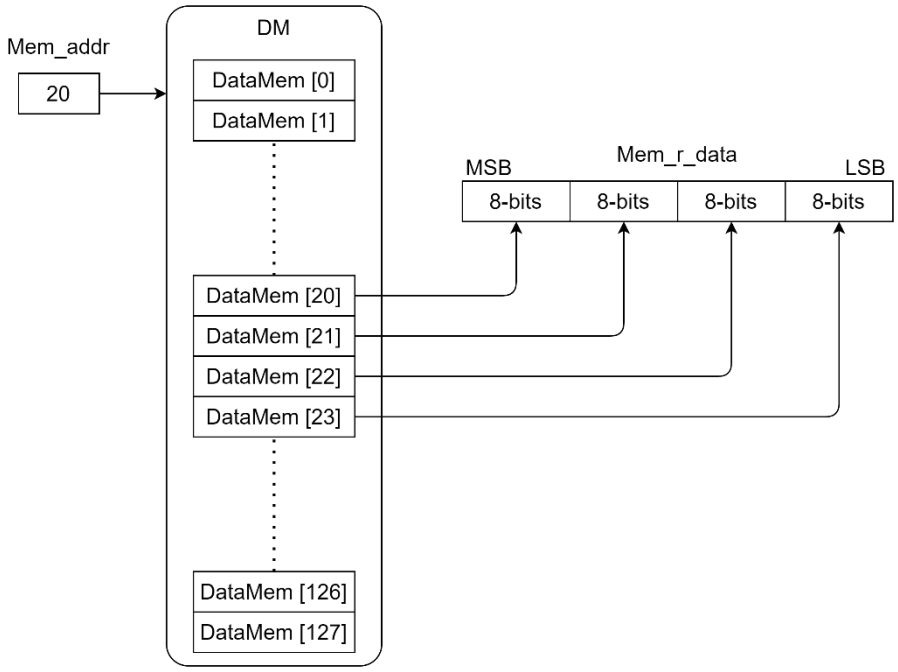
b. Command setting



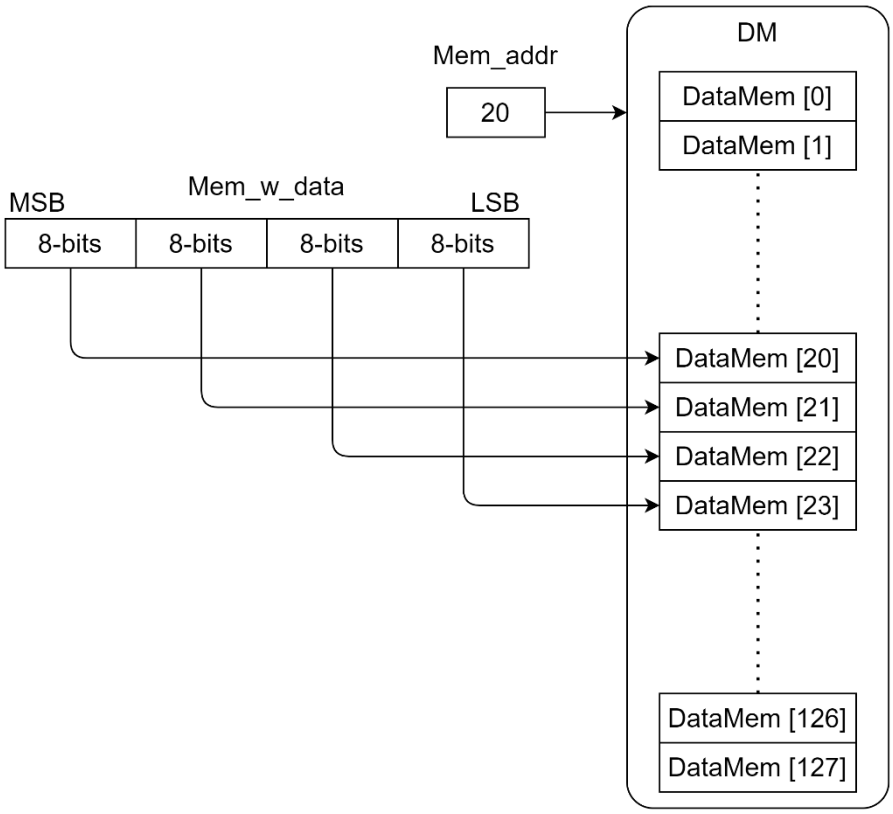
Data Memory

It consists of 128x8 bits memory with Big-endian. Its initial value is set by "/testbench/DM.dat".

a. Data reading



b. Data writing



Testbench Description

a. Initialize

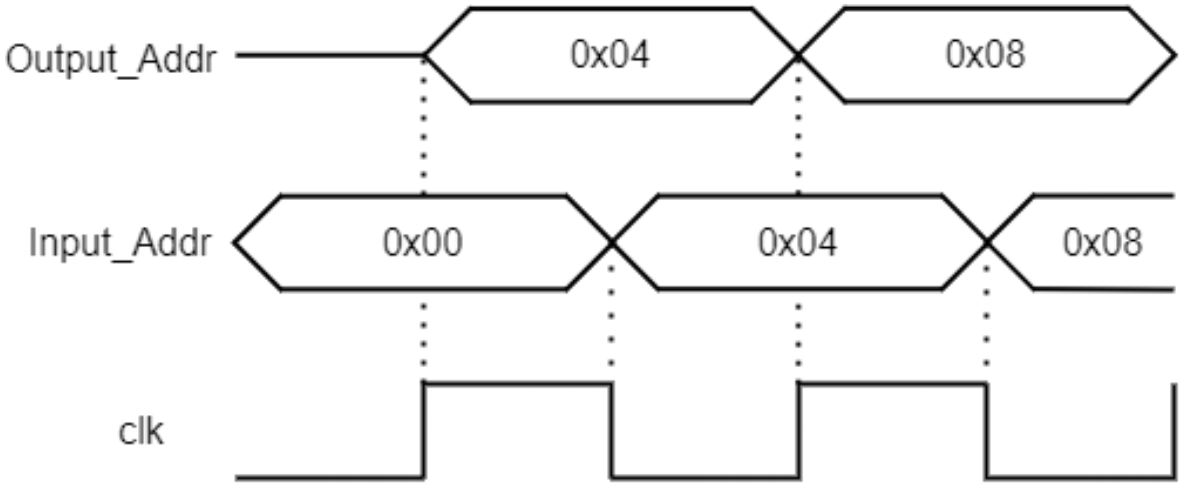
Execute Testbench ("tb_R_FormatCPU.v", "tb_I_FormatCPU.v", "tb_SimpleCPU.v") to initializes Instruction Memory, Register File, and Data Memory, respectively, according to "/testbench/IM.v", "/testbench/RF.v", "/testbench/ DM.v" (except Part I).

b. Clock

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

c. Addressing and Termination

Testbench initializes Input_Addr signal to 0, and assigns Output_Addr to Input_Addr before each positive edge of clk. When Input_Addr is greater than or equal to the maximum address space of the current job instruction, Testbench ends the execution and outputs the current register and memory content ("/testbench/RF.out", "/testbench/DM.out"). The following figure shows the basic waveform of Testbench action:



Note: When the system Output_Addr fails or the program is in an infinite loop, please terminate the simulation and determine the problem manually.

OpenROAD Description of Part III

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
|                   |------------ SimpleCPU
|                             |------------ SimpleCPU.v
|                             |------------ RF.v
|                             |------------ IM.v
|                             |------------ DM.v
|                             |------------ Control.v
|                             |------------ Other necessary .v files
|                             |------------ config.mk
|                             |------------ constraint.sdc
|---------- reports
|---------- Makefile
```

(DESIGN_CONFIG ?= ./designs/nangate45/SimpleCPU/config.mk)

Submission

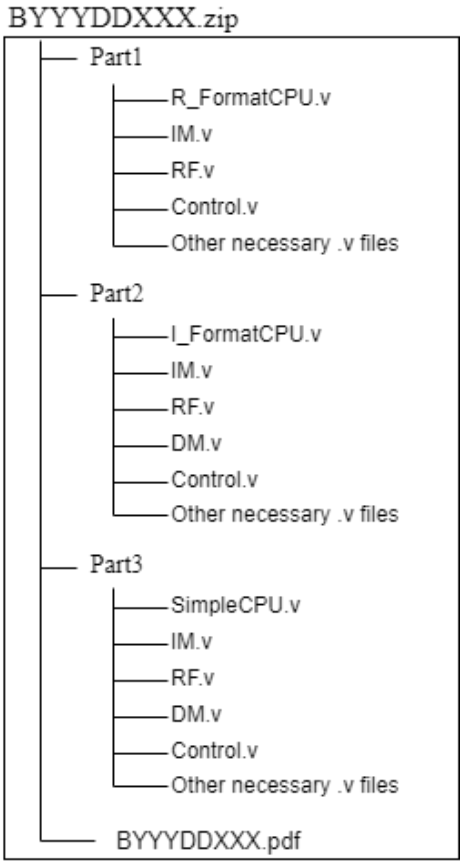
Report (BYYYDDXXX.pdf) :

a. Cover (Project Name, Student ID, Name, Area, Slack).
b. Descriptions of how you implement each module.
c. Descriptions of how you test your modules and the execution results ("/testbench/RF.out" , "/testbench/DM.out") of the sample programs ("/testbench/IM.dat") in each part.
d. Datapath rethinking: If you were required to implement the multiplier and divider (from PA1) in a single-cycle CPU, how would you revise the datapath? 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)
- Part I
  a. R_FormatCPU.v
  b. IM.v
  c. RF.v
  d. Control.v
  e. Other necessary .v files
- Part II
  a. I_FormatCPU.v
  b. IM.v
  c. RF.v
  d. DM.v
  e. Control.v
  f. Other necessary .v files
- Part III
  a. SimpleCPU.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.**



Grading :

1. Main program: Part I (28%), Part II (20%), Part III (12%). All programs are tested by an external testbench.
2. Area and speed optimization of Part III (15%).
3. Report (25%).
4. Follow naming rules and file formats.
5. No plagiarism.

Submission time: Upload to Moodle before 12:00 on 114/5/8