

# PROJECT3

**Student name: Yuxuan Sun 121090502**

To run my project, you should first get into my folder:

For alu: you may first cd src, then cd alu then type make test, the output will be in the console.

For cpu: you may first cd src, then cd cpu, then you can see a bin file "CPU\_instruction.bin", before test you should copy the machine\_code into the file "CPU\_instruction.bin" ,then you can make test! The output will be a file named "data.bin"(in folder"cpu")

Note: I solved all hazards I think.

## Introduction:

In this project, I used Verilog language to implement a CPU with pipeline structure(containing an ALU). According to the working principle of CPU, I separated the whole CPU function into 5 parts:(IF) instructions fetch from instruction memory, (ID)instructions decode and register read, (EXE)execute operation or calculate address, (MEM)access data memory operand and write the result back to register(WB). At the same time, my CPU can also handle some simple hazard such as data hazard, structural hazard and control hazard. Some results of given tests are shown in the appendix part of this report.

## 1. Accomplish:

### 1) Basic idea of the project and main method:

**ALU:** The ALU get 3 inputs: instruction, regA, regB, then output the result and 3

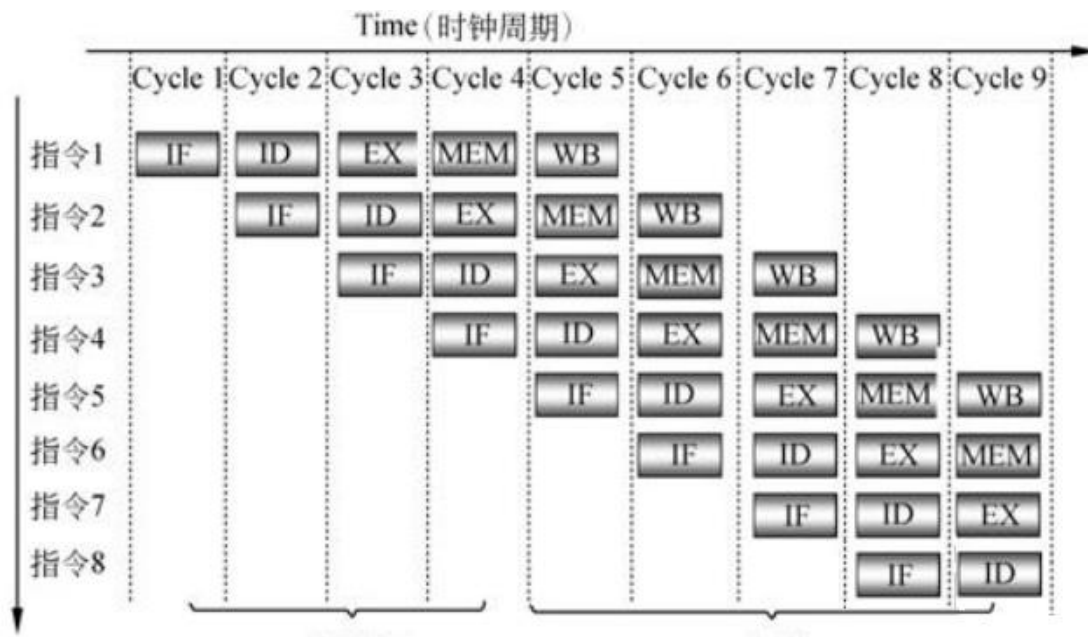
flags that is zero flag and negative flag and overflow flag. So, I identify which operation the code is and do the operation as the instruction. At the end, I assign the result and flags for output. The functions I implemented are as follows:

```

case(opcode)
//R type(func)// I type(opcode)
//add100000 addu100001 //addi001000 addiu 001001
//sub100010 subu100011
//and100100 nor100111 or100101 xor100110 //andi001100 ori001101 xori001110
//slt101010 sltu101011 //beq000100 bne000101 slti001010 sltiu001011
// //lw100011 sw101011
//sll000000 sllv000100 srl000010 srlv000110 sra000011 srav000111
6'b000000://R type
begin
case(func)

```

**CPU:** According to the working principle of CPU, I separated the whole CPU into 5 parts according to the 5 stages of CPU. However, in order to make them working separately without errors caused by different latency of signals, I used clock and buffers. Buffers can be triggered by positive or negative edge of clock. It can make sure all signals can be changed at the same time rather than changed separately. Therefore, all buffers are used to separate different stages and control the change of signals. Overall, the basic structure of this project is using buffers and clock to separate 5 main stages.



## 2) Details of the implementation:

In this part, some detailed explanation and methods of main module are shown as follows: (The section about alu is also included in the later part of the report. )

### 1. Instruction\_part:

#### 1) PC\_checker:

This module is used to get next PC of the program. There are four possible PC: current PC, the current PC plus 4, branch PC or jump PC. How to select the next PC is determined by three control signals: PC\_write, PC\_branch and jump. **PC\_write** is equal 1 when the whole system doesn't need to stall. Otherwise, the whole system needs to

stall may be one cycle, which need to give the current PC to be the next PC. **PC\_branch** is 1 if the program need to branch to other part of instruction. **Jump** is 1 if the program is executing the jump operation.

## 2) **Instruction\_memory:**

This module has two main functions. First, it can load all instructions in the instruction file to the instruction memory at the beginning of the program. Second, fetch the corresponding instructions according to the given PC. Before fetching the instruction from the instruction memory the PC need to divide 4 because each instruction just one element of the instruction memory array. However, the PC is design for general instruction memory. In that situation, each instruction occupied 4 address.

## 2. **Register\_part:**

### 1) **Signal.v:**

2) This module is designed to give different signals for the whole system depending on the different operation codes and function codes. IF\_flush is a signal that is used to flush these signals when the program needs to flush an instruction. If all signals are set to 0, the CPU will do nothing to the memory and register sections. It protects from errors due to the failure of flush instructions.

### 3) **Register:**

This module has three main functions. **First**, it can write the data into the register array when meeting the positive edges and get the data from the data from the register array when meeting the negative edges. This design is used to accomplished the internal forwarding,(data hazard). **Second**, calculate the branch PC. **Third**, calculate the jump PC. Both of last two function are helpful for reducing the flush instructions caused by branch or jump instructions. The earlier the program get the branch or jump PC, the less instructions should be flushed.

### 4) **hazard\_unit:**

This module is used to solve the hazard caused by LW or SW instructions. When the program load data into a register which will be used in the next instruction, the next instruction should stall one cycle to make sure the load instruction get to the memory stage. Although it also has data hazard in this situation, it be solved in the forwarding unit module.

## 3. **ALU:**

There are some differences in two ALU file in my project because the ALU in cpu need to implement three jump instructions. This module can give the write destination according to the given control signals of register destination. If RegDstE is equal to 0, the write destination register is rt which can be get from the decode module. Otherwise, the write back register is rd.

### 1) **forwarding\_unit:**

This module is designed to solve data hazard. There are two types of data hazard can be solved in this module. First, the register used in present instruction is the write destination of last instruction. In this situation, the input data of ALU should be the ALU result of last instruction. Additionally, I used rs or rt source equal to 2'b10 to control signal to control the input source of ALU. Second, the register used in present instruction is the write destination of two instruction before. In this situation, the input

data of ALU should be the write back data of that instruction. At the same time, the rs or rt source will be set to 2'b01. If all the inputs do not satisfy these two conditions, it means there is no data hazard in present instruction. The ALU just fetch data from register array is okay.

#### 4. memory\_part:

##### 1) Mainmemory:

This module is the memory of the whole program. It has three main functions. First, initialize the RAM at the beginning of the program. Make sure all the data in the RAM is 0 before the program writing the data into it. Second, load data from memory when the memory read control signal is equal to 1 (LW instruction). Third, write data into the RAM when the memory write is equal to 1 (SW instruction).

##### 2) Check\_write\_back-data:

This module is used to find the write back data of the instructions. The resource of the write back data can be the ALU result or the data stored from RAM. This can be determined by the control signal of memory to register.

#### 5. CPU:

This module is the combination of all previous modules. In this module, it uses clock and buffer connect all the components of CPU together. Finally, when the instruction is equal to 8'hffffff, which means the whole program comes to the end. However, in order to make sure all the instructions can be done including the write back stage. The CPU needs to finish the program after 4 clock cycle.

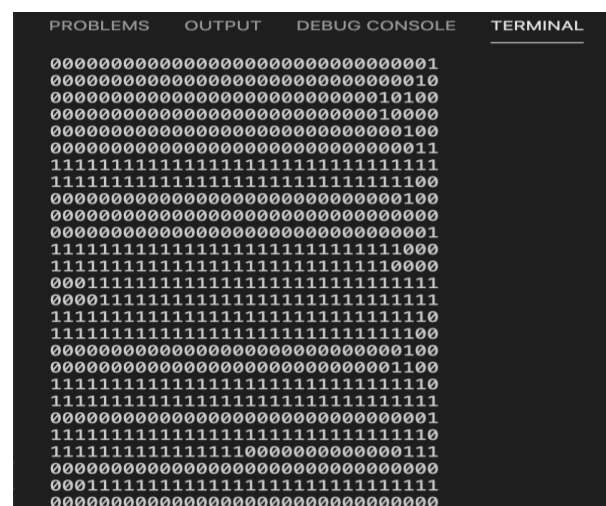
## 2. Data flow chart:(cpu)

The data flow chart is in the folder of project 3. And the file name is data flow chart

## 3. Sample output: (cpu)

The following 6 screen shoot are the 6 sample output of my program according to the given sample input. The test 1, 5 and 6 are the simple test of all simple functions including the branch and jump instructions. Test 2 is the test related to 2 kinds of data hazard. Test 3 is the test of LW instruction. Test 4 is the test of the internal forwarding in the register part.

### Sample result of test 1



## Sample result of test 2

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
<pre>11111111111111111111111111111111000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000</pre>			

## Sample result of test 3

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
<pre>000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000</pre>			

## Sample result of test 4

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
<pre>000000000000000000000000000000000 000000000000000000000000000000000 111111111111111111111111111111111 000000000000000000000000000000000 000000000000000000000000000000000</pre>			

## Sample result of test 5

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
<pre>000000000000000000000000000000000 111111111111111111111111111111111 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000</pre>			

## Sample result of test 6

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
<pre>111111111111111111111111111111110 1111111111111111111111111111111100 11111111111111111111111111111111000 111111111111111111111111111111110000 111111111111111111111111111111110000 1111111111111111111111111111111100000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000001 000000000000000000000000000000001 000000000000000000000000000000001 000000000000000000000000000000001 000000000000000000000000000000001 000000000000000000000000000000000 000000000000000000000000000000000 000000000000000000000000000000000</pre>			