# Report for Project 4

Name: Wang Sen

Student ID: 120090234

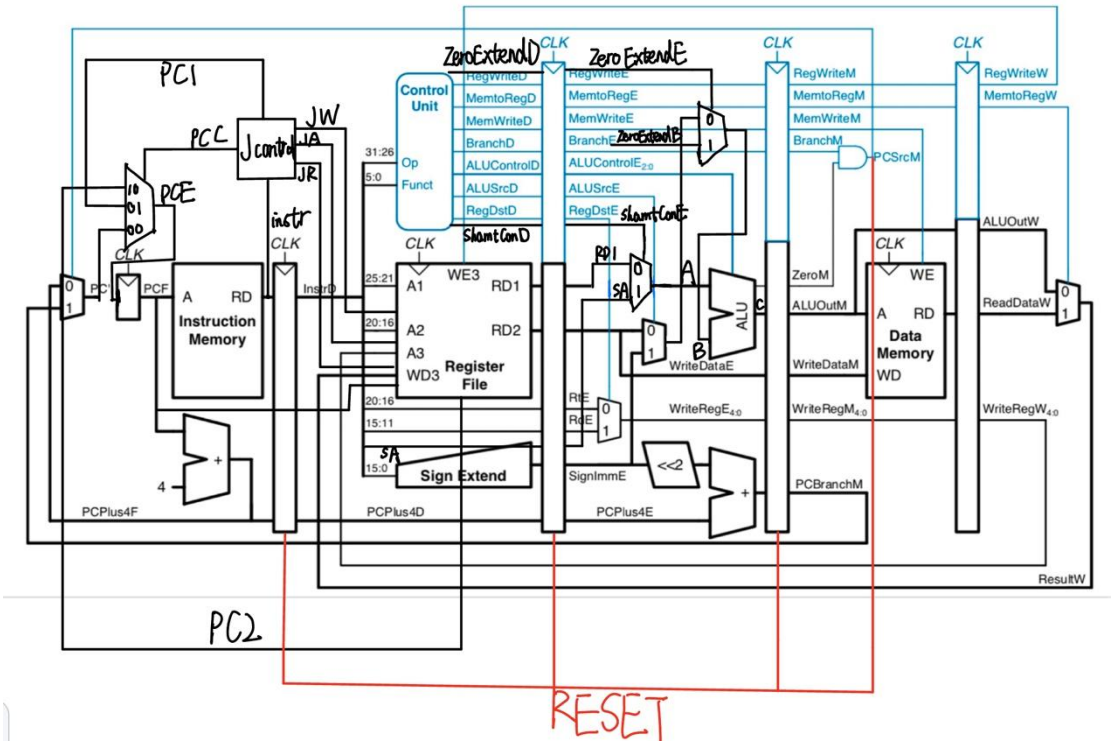
Date: May 1, 2022

## 1. Big Picture:

This project is aiming to achieve a pipelined 5-stage CPU. My idea is to decompose the CPU into small modules, complete the functions of small modules and clarify the input and output of small modules, and connect each small module together to form the CPU.

The focus of piplined 5- stage CPU is on five pipelines. Through these five pipelines, we can realize the synchronization of multiple instructions, rather than waiting for the end of one instruction to execute the next one.

## 2. Data Flow Chart




Extend details:

PCE: the Final PC translate to PCF

PCC: Choose which PC to be PCE

PC, PC1, PC2: Normal PC; PC from j and jal; PC from jr

J control Part: Handle the Hazard caused by j, jal and jr

JA,JW,JR: jr instruction's rs or jal instruction's 5'b11111; Whether write PCF to Reg[JA];Whether read Data in Reg[JA]

RESET: Equal to PCSrcM, use to handle the branch hazards

ZeroExtendD, ZeroExtendE: using in andi, xori, ori zero extend

ShamtConD, ShamtConE: using in sll, srl, sra to use the number from shamt

**3. Implement:**

I decompose the CPU into several small modules. For example, five pipelines and several selectors, register file, ALU, etc. First complete each small module. Focus on completing the functions that small modules should be able to achieve and determining what the input and output of each small module are. Then on the CPU.v file I connect these small modules we completed together, and the final operation has been completed.

In the following details, I will focus on how I extend the initial graph you give. And how to solve the hazards of branch and jump.

**4. Details**

1. How to achieve andi ,xori, and ori: First, we check the input instructions at the control unit. If we find these three instructions, we will change the ZeroExtend output of the control unit to 1. If the ZeroExtend is one, B(SrcBE) will be replaced with zero-extend immediate when entering B.

2. How to achieve shift instruction (using shamt): First, we identify the instructions entering the control unit. If we recognize that these instructions are adjacent, ShamtCon becomes 1. At the beginning, we record the value of shamt in the pipeline, and then judge the value of ShamtCon when entering A(SrcAE). if it is 1, enter the value of shamt into A.
3. How to avoid branch Hazards: The key to deal with this problem is PCSrcM. We set a RESTE for IF/ID, ID/EX, EX/MEM these three pipelines. When PCSrcM is 1, it means that we need to branch. Then set the RESET of the three pipelines to 1 and clear them all directly. Then start to receive the instruction that the branch arrives.
4. How to avoid Jump Hazards: I have introduced a new module, Jcontrol, which is specifically used to solve jump hazards. First, we input the instruction into Jcontrol before the instruction enters the IF / ID pipeline. If it is j, the address to which program jump is directly assigned to PCE. If it is jal, import PCF + 1 into ra in the register file, and then assign the jump address to PCE. If it is jr, read the corresponding address in the register file and send the data we read directly to PCE. (It is worth noting that Jcontrol also sends PCC to control which address PCE chooses as final one)