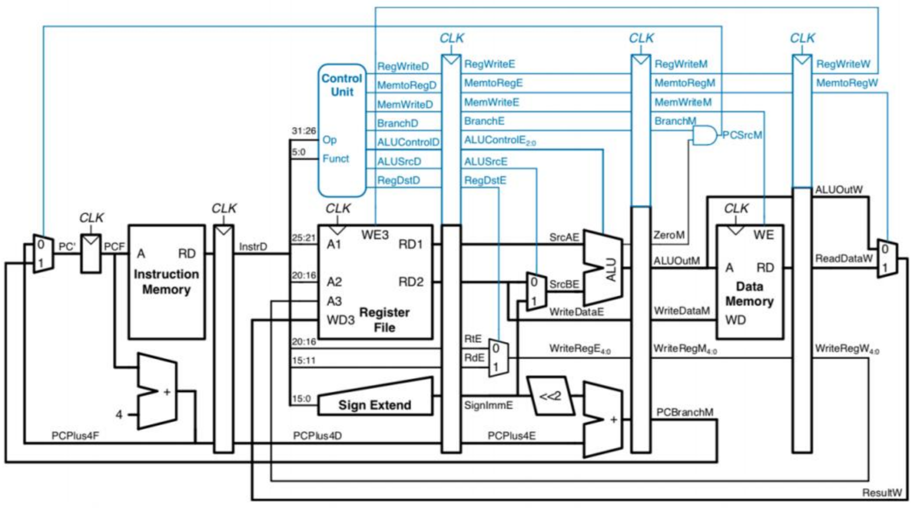
CSC3050 Project4 Report

Liang Mingrui 120090723

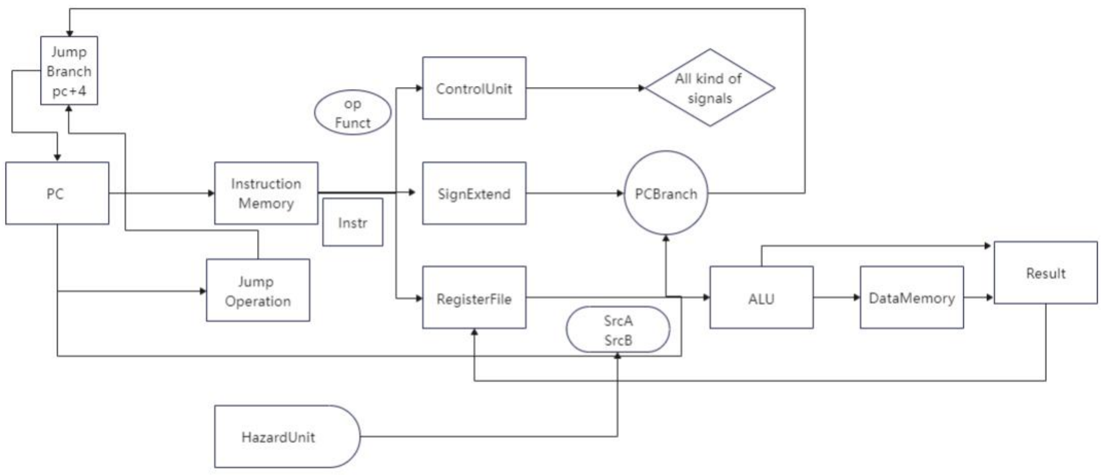
2022.4.23

Big Picture Thoughts

In this project, we are required to design a five stages pipeline to implement some instructions. These five stages include Instructionfetch, Instructiondecode, Excution, Memory and Writeback. In the Instructionfetch stage, I need to prepare a register PC to ensure the Fecth_Address. Also, I need to design a multiplexer for the PC to help it support the instructions such as branch and jump. As for the Instruction Memory has been provided, I can directly connect the PC to it to fetch the instruction and pass it to the IF/ID register. In the Instructiondecode stage, the instruction fetched in the last clock can be decomposed to different data to the controlunit and registerfile. After that, some control signals and registers information will be passed to the ID/EX register. In the Excution stage, some control signals will be used by ALU to decide what kind of excutions it need to implement and used to decide what data need to be passed in ALU. After excution, the outcome will be passed to the EX/MEM register. In the Memory stage, the outcome will be used as the Fetch_Address to get data from the Data Memory. The MEM/WB register will store the data and outcome directly. In the Writeback stage, control signals will decide write aluoutcome or readdata back to the registerfile.



Data Flow Chart



The High Level Implementation Ideas

The whole project is too big to solve directly. In order to make the task clear enough, I divide them into different modules. Totally, I design 12 modules to complete the whole pipeline. First, I create some modules foe each stages. Then, I design a module, CPU, to connect all the modules designed befor. Lastrly, I design a module, test_CPU, to give a clock, rest and enable signals to CPU and dump all the memory content when instruction is 32’hffffffff.

Followings are the main step of the project:

Step 1: Create modules for each stages.

Step 2: Create CPU to connect all these modules.

Step 3: Create test_CPU to dump the memory.

Testing Situation

Test 1,4,5,6: All fully succeed.

Test 2: Although I design a HazardUnit to do forwarding, it seems not work very well.

The rest of tests: Can not pass.

The implementation of the details

In this part, I introduce details in all my modules.

Module IF:

In this module, I make PC to be a register to update with every clock, and it changes depends on a multiplexer, jump, branch or directly plus 4. Then I connect it to the InstructionRAM to get the instruction.

Module Registers:

In this module, I firstly flash all registers to 32'b0. In order to avoid the 4th hazard mentioned in the test, I make this module write the content back first, and then allow it to read data.

Module ControlUnit:

In this module, I take the opcode and functioncode as input to get the control and alucontrol. The module will firstly get control, jumpcontrol and alucontrol from different op and funct. After that it connect control and jumpcontrol together to form the control output.

Module ID:

In this module, I decompose the instruction to different parts to do differents things. These parts are connected to ControlUnit, Registers and do the sign extend. And the jumpaddress will form in this part. After doing this, all result are connected to the output.

Module ALU:

In this module, the alucontrol is the most important thing. Alucontrol will decide what exactly the operation is needed now. The module will do operations to SrcA and SrcB depends on alucontrol. After that, it will generate a zero and result.

Module HazardUnit:

In this module, I set FA and FB to 2'b0. Accroding to the situation, the module will set FA and FB to different value and connect it to the output.

Module EX:

In this module, choosing SrcA and SrcB is the primal task. The finnal content in these two is decided by two multiplexer. That is to avoid hazard. After that, the alu will complete the result and pass it to the output. Also, the signextend will plus 4 to get the branchaddress.

Module MEM:

In this module, aluoutput will be used as fetchaddress to get the data in the mainmemory. After that, aluoutput and data will both be regarded as output.

Module WB:

This module will decide to write aluoutcome or data back to the registers accroding to the control signal.

Module IFID, Module IDEX, Module EXMEM, Module MEMWB:

These module are places that store all kinds of registers, and they can flash the whole module when the branch or jump occured.

Module CPU:

This module connects all the modules I mentioned before, and to keep them can perform correctly.

Module test_CPU:

This module mainly creates the clock, rest and enable signal. Finally, it dumps the memory and stops the whole program.

Moudle IFWire, Module IDWire, Module EXWire, Module MEMWire, Module WBWire:

These module are used to soft those wires in the previous modules, helping those modules connect wires correctly and beautifully.

Module InstructionRAM, Module MainMemory:

I directly use the module given by TA.