# Computer Architecture Project 1

## Members:

B01902010 李紹詮

B01902082 李穆竺

B01902107 張証傑

*All members contributed equally on the project.

In this project, our main goal is to implement a pipelined CPU. We added several components to achieve this goal. First, we added four components, “stage”. These module receive input from the previous stage, and passes the information to the next stage. Therefore we have four modules, IF/ID, ID/EX, EX/MEM and MEM/WB. Second, to solve hazards, we need two modules, the hazard detection unit and the forwarding unit. Finally, we then added more multiplexors in the system to make sure the system will work as we expected.

In the IF/ID module, the main action is to fetch instruction from the instruction memory, then send the instruction to the next stage. However, to make pipelining work, there are several more condition we need to deal on. First, if hazard exists, we will need to stall the pipeline, and this is detected by the input from hazard detection unit. Second, we will need to flush the instruction if jump or branch that we did not correctly predict occurred.

In the ID/EX module, it passes three information to the execution stage, which are the control signal, the data from registers, and the instruction that was passed from the IF/ID module. WB and M are then passed down to EX/MEM while EX are the signals that controls how the execution stage should behave.

In the EX/MEM module, M was received from the ID/EX module and WB will be sent to MEM/WB module. It keeps the result of ALU, the data that will be written into the data memory and part of the instruction from ID/EX.

Finally, the MEM/WB stage is rather simple. It only controls whether the register should be written and keeps some of the information to the forwarding unit.

The forwarding unit looks complicated, but the concept is easy. There are six inputs, two for each module (IF/ID excluded) explained above. The concept is that it reads the input and compares whether there exist some data that should be “written to some address” which is the same as the data for ALU that needs to be “read from some address”. If yes, we can just forward that data to the execution stage then write it into the memory.

The hazard detection unit controls the hazard situation and gives the stall signal. We can examine the hazard detection by the following procedure. First, we check if MemRead of ID/EX is true, if not, we are sure that stall is not needed. Then we now check if the rt-address of ID/EX is the same with the rs-address of IF/ID, if yes, we will then need a stall. Otherwise, we’ll need to check the rt-address of ID/EX and the rt-address of IF/ID. If they are the same, a stall will then be needed.

Flushing is also something that’s worth mentioned. We’ll need to flush the data under the two situations we discussed in the previous page. However, why would we need to flush the data rather than just stalling the pipeline? The main reason is that the instruction in the pipeline is incorrect. If we don’t flush the instruction, we might get some data from some stage which isn’t the result we want, since the instruction is still valid and this isn’t what we want to see.

There are some problem we met during implementing the system. First of all, when IF/ID received a flush signal, we only stalled the pipeline rather than flushing the instruction, which then resulted in some unpredicted behavior. To solve this problem, we then implemented a “no-op” for our system. Second, we added quite a few multiplexors, however, we weren’t careful enough therefore we have mistakenly connected some wire into the wrong port, which lead to incorrect results. Finally, the system is composed from many modules and need to be connected after each module is completed. However, linking the modules isn’t a easy work since we didn’t defined and discussed how the ports should be named, therefore, when we are linking the module together, we met some difficulties.