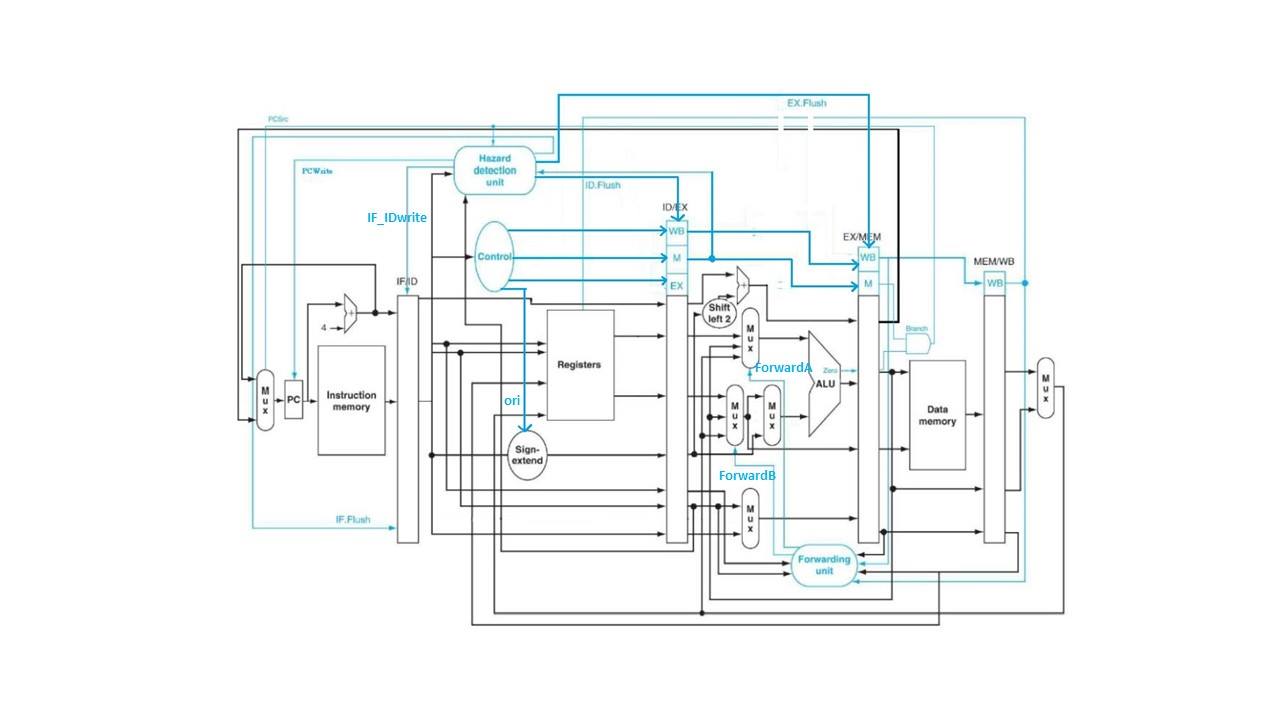
**Computer Organization**

**0416021曾香耘、0416246王彥茹**

**Architecture diagram:**



**Detailed description of the implementation:**

We built our design upon Lab2 because we felt like it would be a lot easier to implement lw, sw, and mul instead of getting rid of the jump family.

Next we built a form to organize the input and outputs of the pipeline registers, we used a huge bus as the output for each register to avoid renaming wires.

Here is the form we built, including the output array index, the original name for the input wire, and the actual input we led in:

IF/ID

63~32 adder1\_out

31~0 instruction

ID/EX

152~148 instruction[25:21] // IF\_ID\_out[25:21] // Rs

147 regwrite

146 memtoreg

145 branch

144 memread

143 memwrite

142 regdst

141~139 aluop

138 alusrc

137~106 adder1\_out // IF\_ID\_out[63:32]

105~74 read\_data1

73~42 read\_data2

41~10 se\_out

9~5 instruction[20:16] //IF\_ID\_out[20:16] // Rt

4~0 instruction[15:11] //IF\_ID\_out[15:11] // Rd

EX/MEM

106 regwrite // ID\_EX\_out[147]

105 memtoreg // ID\_EX\_out[146]

104 branch // ID\_EX\_out[145]

103 memread // ID\_EX\_out[144]

102 memwrite // ID\_EX\_out[143]

101~70 adder2\_out

69 zero

68~37 alu\_result

36~5 read\_data2 // ID\_EX\_out[73:42]

4~0 mux\_write\_reg\_out // Rd

MEM/WB

70 regwrite // EX\_MEM\_out[105]

69 memtoreg // EX\_MEM\_out[104]

68~37 readdata

36~5 alu\_result // EX\_MEM\_out[67:36]

4~0 mux\_write\_reg\_out // EX\_MEM\_out[4:0] // Rd

Then we dealt with the data hazards by forwarding and inserting bubbles (stalling).

For forwarding, we implemented the forwarding unit as stated in the lecture slides such that the cpu would forward correct data when the condition was met.

For load-use data hazard, we implemented the hazard detection unit as stated in the lecture slides, so we stalled ID stage and pause IF stage for 1 cycle.

For branch, we inserted 3 bubbles such that IF, ID, and EX stage would be stalled when the branch condition was met, otherwise it kept on going and nothing happens. We implemented branch by adding additional wires to the hazard detection unit.

And the flush in our cpu was implemented in a different way, instead of choosing 0 from the control signals using a mux, we just simply initiated the pipeline register itself, so all outputs of the pipeline register were 0.

We tested out cpu with the 3 test cases and all were passed.

**Problems encountered and solutions:**

Write back to register file was originally delayed by 1 cycle, we figured out it was because the positive edge clock triggering on both pipeline register and the register file itself, so we cancelled the clock triggering on the register file such that it always update its data when the input has changed. This would be fine because the input only changes when the pipeline register has updated its output, which is on the positive edge of the clock.

The diagram on the spec is different from that on the lecture slides, so when we were implementing the forwarding unit, assignments to the output (2’b10 and 2’b01) should be swapped.

**Lesson learnt (if any):**

Software development practice is important, such as variable names and comments.

It is horrifying to procrastinate until the deadline.