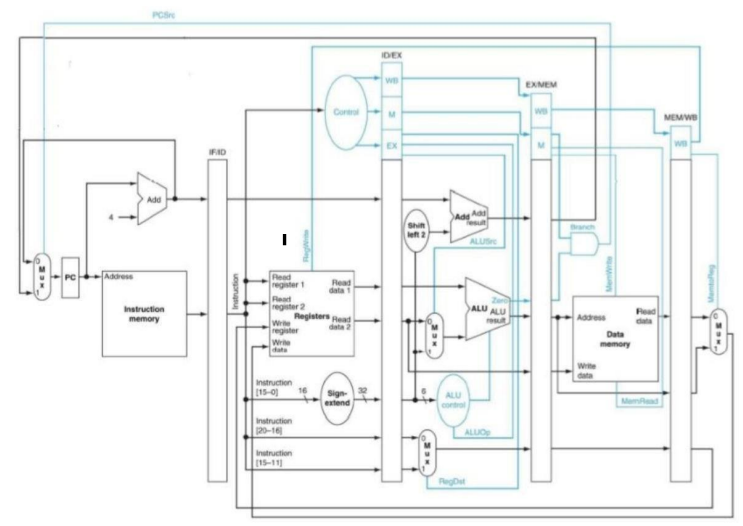
**Computer Organization Lab4**

**Name:林秉**

**ID:109550112**

**Architecture diagrams:**



**Hardware module analysis:**

**Comparing lab3, we don’t need to deal with jump, jal, etc. So we can omit some control bits, and can realize without 3to1 mux. Since most module is the same as lab3, I only introduce the changed part.**

**ALU\_Ctrl:**

一張含有 文字 的圖片

自動產生的描述

**The module is the same except for the jump signal output.**

**Decoder:**

一張含有 桌 的圖片

自動產生的描述

**Again, the module deletes the jump , jal signal, so we san simplify the output signal, and save output signal bits like RegDst.**

一張含有 文字 的圖片

自動產生的描述

**I used to write the output code as above, so I can avoid using a lot of unnecessary always@(\*), this works in Lab2 and 3, but fail when I testing, so I cover every signal with always@(\*) this time.**

一張含有 文字 的圖片

自動產生的描述

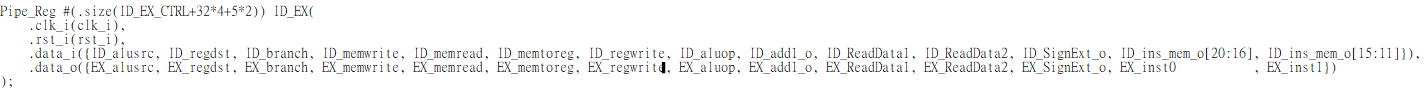
**Pipe\_reg:**

**I only introduce the stage register part since rest module is basically the same as lab3.**

一張含有 文字 的圖片

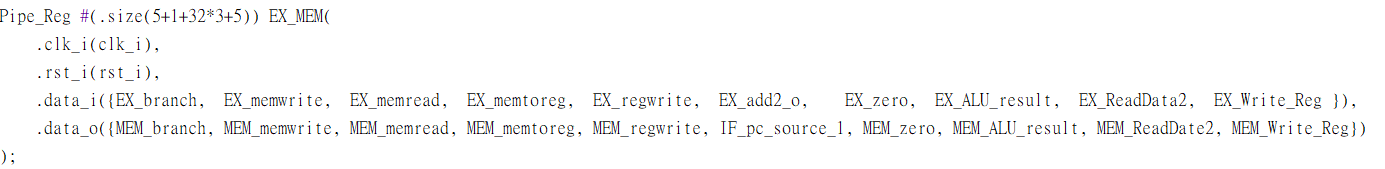
自動產生的描述

**IF/ID packs the next instruction sequence and the instruction. So, we need 32\*2 bits.**



**ID/EX packs all the control signal, the instruction sequence of IF/ID, the data read from register, sign\_extend address and the rs rt register number. So we need Control bytes and 4\*32 bits for register data(2) the instruction sequence data(1) and sign extend address(1) and 10 bits for rs rt, 5 bits each.**

**EX/MEM:**



**This register packs branch for beq, and memory control bits for memory module, and regwrite for mux total 5 bits, and 32 bits each for beq jump address, alu\_result for arithmetic calculation or calculating address, and data2 for Registers. Finally we have a 5 bits Write\_reg for arithmetic calculation instruction.**

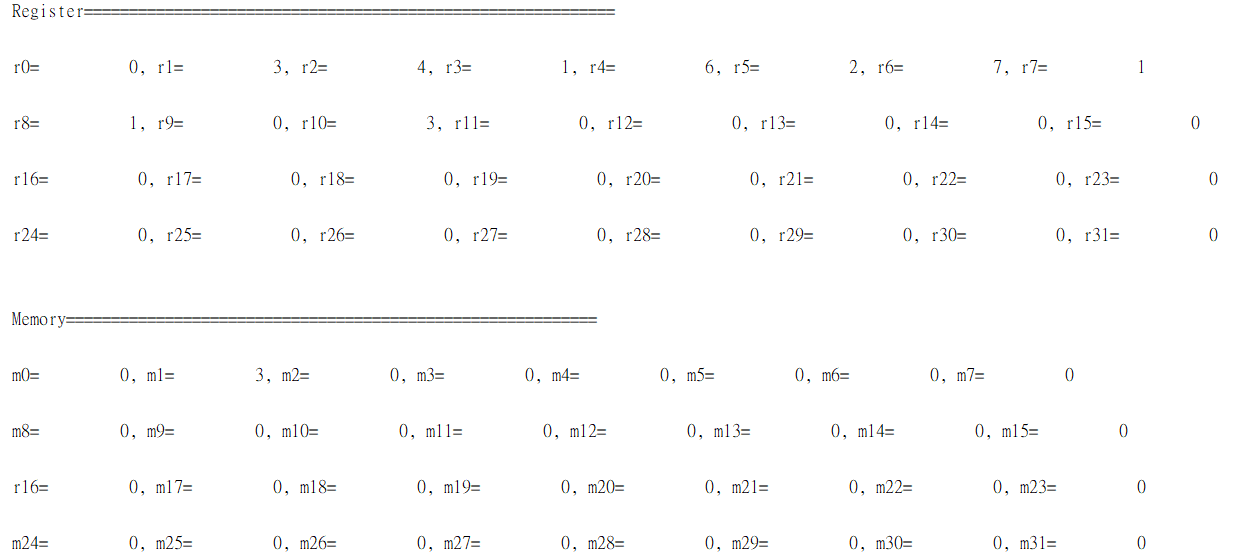
**MEM/WB:**

一張含有 文字 的圖片

自動產生的描述

**We store 2 control bits for memtoreg and regwrite . And store 32 bits each for ALU\_output and memory data output, finally store 5 bits for Write\_reg in EX/MEM section.**

**Finished part:**



**Problems you met and solutions:**

**As I mentioned in hardware module analysis, I find it weird to have the same coding style but with different results, I can’t figure the solution for the simplified version, so I use always for every control signal output. And I find it difficult to apply values in the pipe registers, so when I applying those values, I double care those values in case I make a mistake.**

**Bonus (optional):**

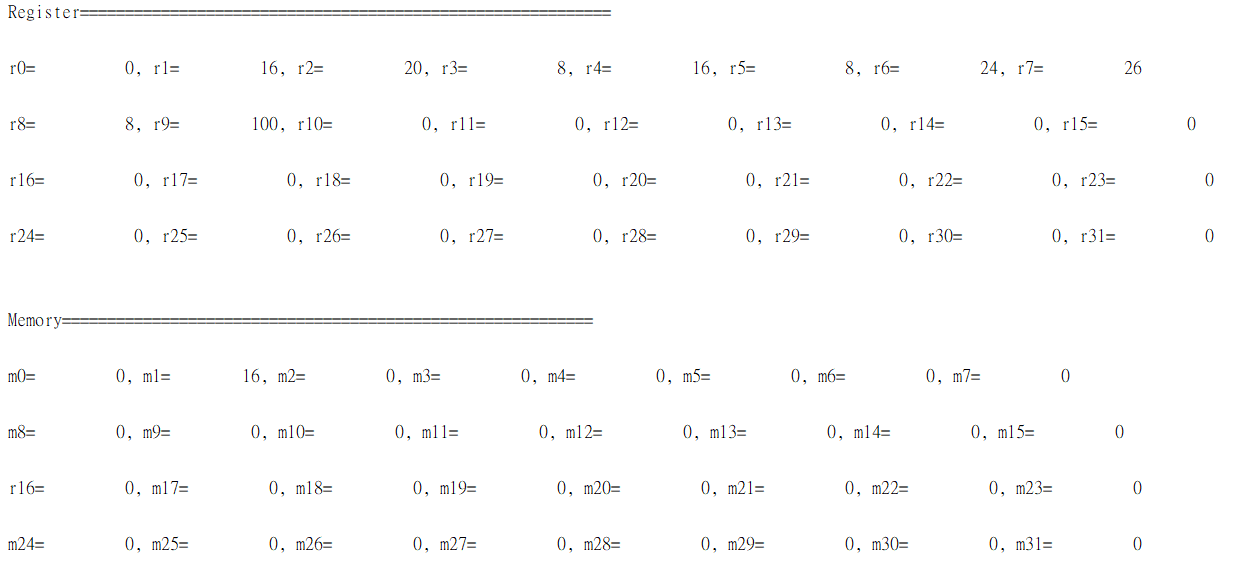
一張含有 文字 的圖片

自動產生的描述

**Sequence works as follows, the comment at end stands for the sequence order of the mips operation in pdf.**

**Since this cpu can’t do forwarding, for the data hazard area, we need to add 2 instructions not using the former data which works for separating the hazard instructions .**

**Execution result:**



**Summary:**

**This is the most difficult lab, it requires lots of concepts and patience since the variables are a lot more than ever, so naming every variable is critical, after a lot of complicated variable checking, I managed to complete the lab. And I find the bonus is pretty inspiring, by solving the question, I get used to the data hazard more, so I think I understand the structure better now.**