Report for Project1

Bzhe's workout is none of your business

1. **Members and Work divisions**
2. B06902067 (許育銘): 45 %

wire connection in *CPU.v*, implementation of pipeline registers and hazard detection, main coding and **DEBUGING**.

1. B06902053 (張維哲): 40 %

Datapath drawing, implementation of forwarding, main coding and **DEBUGING**.

1. B06201035 (高暐竣): 15 %

Implementation of *sign\_extend.v*, datapath drawing, final inspection, **DEBUGING** and report writing.

1. **Implementation of *CPU.v***
2. Direct connect input wire to output wire, so no more wire declaration.
3. The modules are declared in the following order:

IF stage -> IF/ID -> ID stage -> ID/EX -> EX stage -> EX/MEM -> MEM stage -> MEM/WB -> WB stage.

Hazard detection is in ID stage, while forwarding is in EX stage.

1. **Some notes for implementation of modules**
2. Except pipeline registers, we also implement *Control.v* and *Branch.v* with blocking assignment.
3. *IF\_ID.v*: Store previous address and instruction as *addr\_pre* and *instr\_pre*. Pass them as output if *stall*=1; pass 0 as output if *flush*=1.
4. *Control.v*:
5. Control signals are the same as P7 in slide, excepting that *ALUSrc\_o* of beq is 1, but which is in fact no-care.
6. The signals of addi are the same as ld, excepting that *RegWrite\_o*=0.
7. *ALU\_Control.v*: For addi, ld, sd and beq, we let *ALU\_Ctrl\_o* to be 000, which corresponds to add. It’s ok because *MemWrite\_o* and *RegWrite\_o* in *Control.v* are both 0 for beq instruction, so there is no write to memory and register.
8. *MUX32\_2/3.v*: Select from 2/3 32-bit input.
9. *Sign\_Extend.v*: Generate immediate and perform sign extension as P8 in slide.
10. **Forwarding**

We implement forwarding for *EX\_MEM* and *MEM\_WB* as the same as P13 in slide in *Forwarding.v*, excepting that there is only one RS input and one signal output. Therefore, there is two forwarding modules in *CPU.v*.

1. **Hazard Detection**
2. Flush: If *branch*=1 then *flush*=1 in *HazardDetection.v*; *IF\_ID.addr\_o* and *IF\_ID.instr\_o* of next cycle will be set to 0. There is no special case for flush in *Control.v*; its ok because *Branch.Branch\_o*, *Control.MemWrite\_o* and *ID\_EX.RD\_o* will be 0, so there is no branch, write to memory and write to register, respectively.
3. Stall: In *HazardDetection.v*, *stall*=1 when *ID\_EX* stage need to write back, also *RS1*=*RD* or [*RS2*=*RD* and *instr*[5]=1 (i.e. the instruction is not addi nor ld, which doesn’t use *RS2*)]. At the same time, *IF\_ID.addr\_o* and *IF\_ID.instr\_o* are passed to *HazardDetection*, and then passed back to *IF\_ID* as “previous state”. They will be used as output if *stall*=1.
4. ***testbench.v***We store *stall* and *flush* as integer, count them as the same way with P15 in slide; print them with format “%0d” to meet the reference output (no additional empty space).
5. **Difficulties encountered and solutions**
6. Since we use blocking assignment for all pipeline registers and signals in control, the initial assignment may conflict with the first posedge of *clk*. Therefore we initialize the registers after 1 sec.
7. There is the **LAST BUG** found in final inspection: In *IF\_ID.v*, we accidentally pass *instr\_pre* to *addr\_o*. Since *IF\_ID.addr\_o* is only passed to *Add\_branch*, the bug only affect the following case:

* A data hazard occurs at a beq instruction which is right after a ld instruction
* The result of beq instruction is to branch