Report for Project1

Bzhe's workout is none of your business

1. **Members and Work divisions**
   (1) B06902067 (許育銘): 45 %
   wire connection in *CPU.v*, implementation of pipeline registers and hazard detection, main coding and **DEBUGING**.
   (2) B06902053 (張維哲): 40 %
   Datapath drawing, implementation of forwarding, main coding and **DEBUGING**.
   (3) B06201035 (高暐竣): 15 %
   Implementation of *sign_extend.v*, datapath drawing, final inspection, **DEBUGING** and report writing.

2. **Implementation of *CPU.v***
   (1) Direct connect input wire to output wire, so no more wire declaration.
   (2) 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.

3. **Some notes for implementation of modules**
   (1) Except pipeline registers, we also implement *Control.v* and *Branch.v* with blocking assignment.
   (2) *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.
   (3) *Control.v*:
      (a) Control signals are the same as P7 in slide, excepting that *ALUSrc_o* of beq is 1, but which is in fact no-care.
      (b) The signals of addi are the same as ld, excepting that *RegWrite_o*=0.
   (4) *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.
   (5) *MUX32_2/3.v*: Select from 2/3 32-bit input.
   (6) *Sign_Extend.v*: Generate immediate and perform sign extension as P8 in slide.

4. **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*.

5. **Hazard Detection**
   (1) 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.
   (2) 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.

6. ***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).

7. **Difficulties encountered and solutions**
   (1) 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.
   (2) 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