VE370 Homework5

```
        lw R2,0(R1)
Label1: beq R2,R0,Label2 ; Not taken once, then taken
        lw R3,0(R2)
        beq R3,R0,Label1 ; Taken
        add R1,R3,R1
Label2: sw R1,0(R2)
```

Q1

1. The control hazard detection unit need to identify whether the current `branch` instruction depends on the previous result. As it is in the ID Stage, the previous one instruction just enters EX (affect R-type and `lw`) and second-previous just enters MEM (affect `lw`).
   - As the first branch instruction `beq R2,R0,Label2` depends on `R2`, which is the previous one instruction, the result of `Mem[0+R1]` will not be produced until two clock cycles later (when `beq` enters). So need to add two NOPS.
   - If the `lw` is the second-previous instruction of `beq`, only one NOP needed.
   - If depends on the previous R-type instruction, need to add one NOP.
2.
   - **IN EX STAGE:** if we assume taken at first. Only need to wait when beq with previous lw instruction and two beq together. (can't decide to branch together?)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw R2,0(R1)	IF	ID	EX	MEM	WB									
beq R2,R0,Label2		IF	ID	xx	EX	MEM	WB							
lw R3,0(R2)					(Flush three)	IF	ID	EX	MEM	WB				
beq R3,R0,Label1							IF	ID	xx	EX	MEM	WB		
beq R2,R0,Label2								IF	xx	ID	EX	MEM	WB	
sw R1,0(R2)										IF	ID	EX	MEM	WB

   - **IN ID STAGE:** If we add NOP as the previous question states, if we assume taken at first, for this particular program will achieve more speed up. The execution will be like (*beq* represents not taken, **beq** means taken)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw R2,0(R1)	IF	ID	EX	MEM	WB										
beq R2,R0,Label2		IF	xx	xx	ID	EX	MEM	WB							
lw R3,0(R2)					(Flush one)	IF	ID	EX	MEM	WB					
beq R3,R0,Label1							IF	xx	xx	ID	EX	MEM	WB		
beq R2,R0,Label2										IF	ID	EX	MEM	WB	
sw R1,0(R2)											IF	ID	EX	MEM	WB

So moving branch execution from **EX stage (14 clock cycles)** to **ID stage (15 clock cycles)** do not cause any speedup. This might due to the extra waiting cycles due to `lw`. Or represent the speed up as $14/15 = 0.93$.

3. Register `R2` between `lw R2,0(R1)` and `beq R2,R0,Label2`.

   Register `R3` between `lw R3,0(R2)` and `beq R3,R0,Label1`.

4. Branch in **ID**, then if there is previous dependent instruction and will give the expected result, need to add stall, then the forwarding path

   - *depends on previous R-type:* in MEM / WB stage, need to be forward.
   - *depends on previous load*: in WB stage, need to be forward.

   Notice that the forwarding unit actually take **the same input** (*MEM/WB.RegWrite, MEM/WB.RegisterRd, EX/MEM.RegWrite, EX/MEM.RegisterRd, ID/EX.RegisterRs, ID/EX.RegisterRt*). Then the **output** should be same as two 2 bit control signal that control two MUXes, as the input of the comparator.

   two MUX: selects among value from *register file*, *ALUResult from EX/MEM*, *data from MEM/WB*.

   beq **R2,R0,Label2** will add two stall then the forwarding unit detect the *ID/EX.RegisterRs(*`R2`*) == MEM/WB.RegisterRd(*`R2`*), MEM/WB.RegisterRd=1*, no forward from `EX/MEM`. Then the forward A will choose from **data from MEM/WB** (same as before, will be 01). Forward B still 00.

Q2

1. Because the branch outcome is determined in the `EX` stage, for every time the mis-predicted branch will need flush previous three instructions. $30\% \times 55\% = 16.5\%$ will add extra three clock cycles. Then the extra CPI is

$30\% \times (1 - 45\%) \times 3 = 0.495.$

2. Assume we will not jump. the jump outcome is determined in the `ID` stage, so one extra clock cycle needed. $30\% \times (1 - 55\%) \times 3 + 5\% \times 1 = 0.455$

Q3

Strong not taken: SNT

	T	**NT**	**NT**	**T**	**T**	**T**	**T**	**NT**	**(2)T**	**...**
State	SNT	SNT	SNT	NT	T	ST	ST	T	SNT	
Predict	True	True	False	False	True	True	False	True	...	

Find that after finish the eight instructions, the first instruction returns to *SNT* state, so it wil repeat these prediction. $accuracy = \frac{5}{8} = 0.625$