HW4

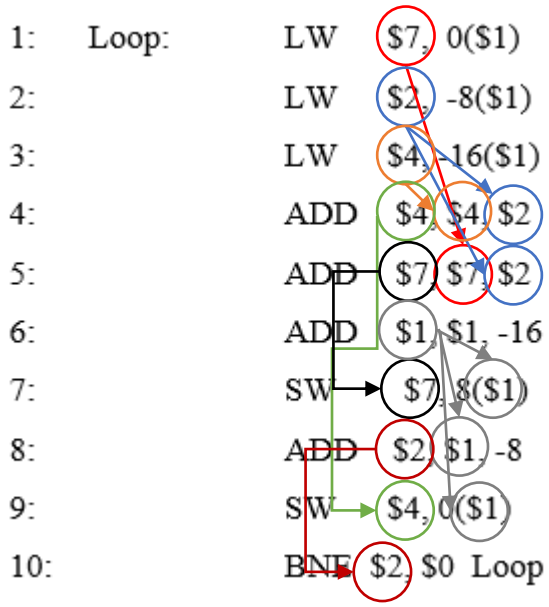
---

1. Pipelined processor [20 points]

Consider the instruction block given below:

```
1:    Loop:    LW    $7,  0($1)
2:             LW    $2,  -8($1)
3:             LW    $4, -16($1)
4:             ADD   $4, $4, $2
5:             ADD   $7, $7, $2
6:             ADD   $1, $1, -16
7:             SW    $7, 8($1)
8:             ADD   $2, $1, -8
9:             SW    $4, 0($1)
10:            BNE   $2, $0  Loop
```

(a) **[5 points]** Identify all RAW dependencies in the code above. You should draw a circle around both registers that exhibit the dependency and connect them with a line.



(b) **[5 points]** Assume no forwarding. Insert stall cycles between instructions to avoid data hazard. Insert one "stall" for each stall cycle. (Ignore control hazard in this problem.)

```
1:   Loop:   LW    $7,  0($1)
2:           LW    $2,  -8($1)
3:           LW    $4, -16($1)
                   Stall
                   Stall
4:           ADD   $4, $4, $2
5:           ADD   $7, $7, $2
6:           ADD   $1, $1, -16
                   Stall
                   Stall
7:           SW    $7, 8($1)
8:           ADD   $2, $1, -8
9:           SW    $4, 0($1)
                   Stall
10:          BNE   $2, $0  Loop
```

(c) **[5 points]** Assume FULL forwarding. Insert stall cycles.
(Ignore control hazard in this problem. Branch result and target address are computed in ID stage.)

```
1:   Loop:   LW    $7,  0($1)
2:           LW    $2,  -8($1)
3:           LW    $4, -16($1)
                   Stall
4:           ADD   $4, $4, $2
5:           ADD   $7, $7, $2
6:           ADD   $1, $1, -16
7:           SW    $7, 8($1)
8:           ADD   $2, $1, -8
9:           SW    $4, 0($1)
10:          BNE   $2, $0  Loop
```

(d) **[5 points]** Branch result and target address are computed in ID stage. Assume "predict taken" policy for branch prediction. How many stall cycles inserted when branch is taken? How many stall cycles inserted when branch is not taken?

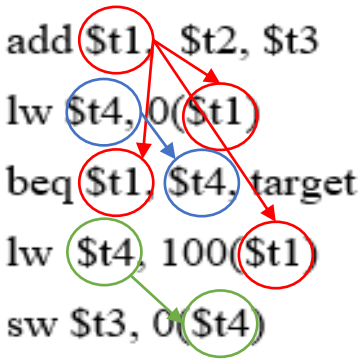
(sol) 1 stall cycle in both cases

2. **Pipelined processor [25 points]**

Please consider following codes

```
add $t1,  $t2, $t3
lw $t4, 0($t1)
beq $t1, $t4, target
lw  $t4, 100($t1)
sw $t3, 0($t4)
```

(a) [5 points] Identify all dependencies in the code above assuming the code is executed on a pipelined processor. You should draw a circle around both registers that exhibit the dependency and connect them with a line.



(d) **[10 points]** Assuming the pipelined processor provides full forwarding, show the execution of the code in part (a) in the table below (multi-cycle pipeline diagram). Show all required forwarding by drawing an arrow from the source to the destination. Indicate pipeline stalls by a dashed line (--). The first one has been done for you.

Hint: a stall cycle is inserted after an instruction is decoded in ID/RF stage and dependency is detected as a result of instruction decoding. All instruction following a stalled instruction are also stalled. E.g. if an instruction A is stalled by one clock cycle, the pipelined diagram is shown as:

Inst A IF | ID/RF| -- | EX | MEM | WB |

Inst B | IF | -- |ID/RF| EX | MEM | WB

In addition, assume that the branch target calculation and branch outcome resolution are performed in ID stage and no branch prediction scheme is used (assume no control hazard for this problem).

Instruction	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th
add $t1, $t2, $t3	IF	ID/RF	EX	MEM	WB									
lw $t4, 0($t1)		IF	ID/RF	EX	MEM	WB								
beq $t1, $t4, target			IF	--	--	ID/RF	EX	MEM	WB					
lw $t4, 100($t1)				--	--	IF	ID/RF	EX	MEM	WB				
sw $t3, 0($t4)							IF	ID/RF	--	EX	MEM	WB		

(c) [10 points] Branch instruction (BEQ) is 15 % of total instructions. BEQ has dependency with its preceding instructions with following statistics (showing only percentage of dependent instructions).

Distance between two instructions	> 3	3	2	1
Load instruction And BEQ	5 %	5 %	5 %	5 %
ALU instruction And BEQ	10 %	5 %	5 %	10 %

For instance, in problem (b), the frequency of two instructions (lw $t4, 0($t1) and beq $t1, $t4, target) is 5 % as shown in grey box.

Please compute the CPI stall cycles due to the dependency in above cases.

(sol) 각각의 경우 얼마의 install cycle 이 insert 되는지 알아야 함.

Dependent Load and beq

Distance 1: 15% * 5% * 2 cycles

Distance 2: 15% * 5% * 1 cycles

Distance 3 and beyond: 0

Dependent ALU and beq

Distance 1: 15% * 10% * 1 cycles

Distance 2 and beyond : 0

Adding all above number = 0.0375

3. **Cache [20 points]**

You have a simple computer system with the following description for the cache and memory system.

- A processor is running at 1 GHz.
- Memory access instructions are 30 % of total instructions.
- There are only one instruction cache and one data cache in the system.
- Miss rates for the instruction and data cache are 2% and 5 % respectively.
- Data cache is a writeback cache.
- 40 % of cache blocks in the data cache is dirty.
- Block size of the last-level cache is 64 bytes.
- Miss penalty is 100 cycles.
- No write buffer is used in data cache.
- A block is allocated on a write miss.
- Instruction and data cache size are 32 Kbytes each.
- Set-associativity for instruction and data cache are 2 and 4.

(a) [5 points] Calculate the CPI stall cycles due to instruction cache misses and required memory bandwidth to support the instruction cache misses.

(sol) CPIstall = 0.02*100 = 2 cycles.
Bandwidth = 10^9*0.02*64 bytes/sec = 1.28GB/s

(b) [5 points] Calculate the CPI stall cycles due to data cache misses and required memory bandwidth to support the data cache misses.

(sol) CPIstall = 0.3*0.05*(1+0.4)*100 = 2.1 cycles.
Bandwidth = 10^9*0.3*0.05*(1+0.4)*64 bytes/sec = 1.344GB/s

(c) [5 points] Calculate the number of tag bits, index bits, block offset bits for the data cache.

(sol) tag=19bits index=7bits block offset=6bits

(d) [5 points] Assume that we have a write buffer and the buffer is not full always. How does adding a write buffer change the answer in (b)? Please compute the CPI stall cycles due to data cache misses and required memory bandwidth to support the data cache misses.

(sol) Dirty block replacement traffic go to write buffer and does not affect performance.
So exclude 40 % from CPIstall.
CPIstall = 0.3*0.05*(1)*100 = 1.5 cycles.
Bandwidth is the same because all misses traffic including replacement ultimately goes to memory.