4 Two-Cycle Pipelined Integer ALU and Multiplier

4.a Part 4.A Control and Data Hazard Latencies

Jump Resolution Latency = 2
Branch Resolution Latency = 4
ALU-use Delay Latency = 2
Load-use Delay Latency = 3

4.b Resolving Data Hazards with Software Scheduling

```
1          bne r1, r0, done
2          lw r5, 0(r2)
3          lw r6, 0(r3)
4          addiu r8, r4, 4
5          nop
6          addu r7, r5, r6
7          nop
8          sw r7, 0(r8)
9          ...
10 done:   addiu r10, r9, 1
```

By swapping the addu and addiu instructions, we can avoid 1 nop for the dependence between the addu and the lw instructions. Regardless of how the addiu and addu instructions are scheduled, there will always be a nop right before the sw instruction due to the dependence.

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W							
2	`lw`	`r5, 0(r2)`		F	D	X0	X1	M	W						
3	`lw`	`r6, 0(r3)`			F	D	X0	X1	M	W					
4	`addiu`	`r8, r4, 4`				F	D	X0	X1	M	W				
5	`nop`						F	D	X0	X1	M	W			
6	`addu`	`r7, r5, r6`						F	D	X0	X1	M	W		
7	`nop`								F	D	X0	X1	M	W	
8	`sw`	`r7, 0(r8)`								F	D	X0	X1	M	W

Figure 11: Software Scheduling Pipeline

Instruction 6 (addu)'s D stage depends on instruction 2 (lw)'s M stage and instruction 3 (lw)'s M stage. Instruction 7 (sw)'s D stage depends on instruction 4 (addiu)'s X1 stage and instruction 6 (addu)'s X1 stage.

4.c Resolving Data Hazards with Stalling

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W								
2	`lw`	`r5, 0(r2)`		F	D	X0	X1	M	W							
3	`lw`	`r6, 0(r3)`			F	D	X0	X1	M	W						
4	`addu`	`r7, r5, r6`				F	D	D	D	X0	X1	M	W			
5	`addiu`	`r8, r4, 4`					F	F	F	D	X0	X1	M	W		
6	`sw`	`r7, 0(r8)`								F	F	D	X0	X1	M	W

Figure 12: Hardware Stalling Pipeline

Instruction 6 (addu)'s D stage depends on instruction 2 (lw)'s M stage and instruction 3 (lw)'s M stage.
Instruction 7 (sw)'s D stage depends on instruction 4 (addiu)'s X1 stage and instruction 6 (addu)'s X1 stage.

4.d New Stall Signal

```
ostall_load_use_X_rs_D
  =    val_D && rs_en_D && (val_X0 || val_X1) && (rf_wen_X0 || rf_wen_X1)
    && ((inst_rs_D == rf_waddr_X0) || (inst_rs_D == rf_waddr_X1))
    && ((rf_waddr_X0 != 0) || (rf_waddr_X1 != 0))
    && ((op_X0 == LW) || (op_X1 == LW))
ostall_load_use_X_rt_D
  =    val_D && rt_en_D && (val_X0 || val_X1) && (rf_wen_X0 || rf_wen_X1)
    && ((inst_rt_D == rf_waddr_X0) || (inst_rt_D == rf_waddr_X1))
    && ((rf_waddr_X0 != 0) || (rf_waddr_X1 != 0))
    && ((op_X0 == LW) || (op_X1 == LW))
ostall_alu_use_X_rs_D
  =    val_D && rs_en_D && val_X0 && rf_wen_X0
    && (inst_rs_D == rf_waddr_X0)
    && (rf_waddr_X0 != 0)
ostall_alu_use_X_rt_D
  =    val_D && rt_en_D && val_X0 && rf_wen_X0
    && (inst_rt_D == rf_waddr_X0)
    && (rf_waddr_X0 != 0)
stall_D
  =    val_D && !squash_D
    && ((ostall_load_use_X_rs_D || ostall_load_use_X_rt_D) ||
       (ostall_alu_use_X_rs_D || ostall_alu_use_X_rt_D))
```

4.e Resolving Control Hazards with Scheduling and Speculation

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W				
2	`lw`	`r5, 0(r2)`		F	D	X0	-	-	-			
3	`lw`	`r6, 0(r3)`			F	D	-	-	-	-		
4	`addu`	`r7, r5, r6`				F	-	-	-	-	-	
5	`addiu`	`r10, r9, 1`					F	D	X0	X1	M	W

Figure 13: Squashing Pipeline

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W				
2	`lw`	`r5, 0(r2)`		F	D	X0	X1	M	W			
3	`lw`	`r6, 0(r3)`			F	D	-	-	-	-		
4	`addu`	`r7, r5, r6`				F	-	-	-	-	-	
5	`addiu`	`r10, r9, 1`					F	D	X0	X1	M	W

Figure 14: Branch Delay Slot Pipeline