Homework 2 solutions

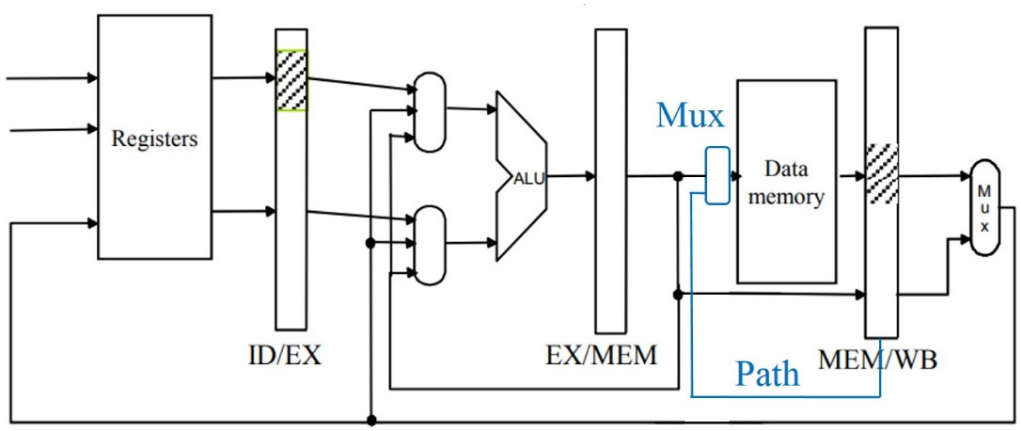
Question 1 (15 points):

As discussed in class, even with forwarding hardware, the pipeline has to be stalled when a "lw" instruction which loads data into a register "Rt" is followed by an instruction which reads from register "Rt". In this question, you will explore an exception to this rule in the case where the instruction following "lw $Rt, I1($R1)" is "sw $Rt, I2($R2)" that saves the content of "Rt" (just loaded) into memory.

a. Argue that in this case, it is possible to avoid stalling by providing a forwarding path between the MEM/WB buffer to the input of the data memory.

When the sw instruction reaches the MEM stage to store the value in memory, the lw instruction will be in the WB stage and the value loaded by the lw instruction will be stored in the MEM/WB buffer. If this value is not accessible (i.e. cannot be forwarded) to the input of the data memory for the sw instruction, hazard will occur unless the pipeline is stalled to separate the two instructions. However, if the value can be forwarded and made accessible, the sw instruction can read this loaded value and proceed through the MEM stage directly without stalling.

b. Draw a simple diagram for the part of the data path between the EX/MEM and MEM/WB buffers showing the forwarding path described in part (a).



Question 2 (20 points): In this question, we will consider a 10-stage pipeline in which target addresses are evaluated in the second stage and branch conditions are evaluated in the fifth stage (no delayed branching is assumed). In this pipeline, predicting that a branch is always taken means that if the branch instruction (at address PC) enters the pipeline at time t, then the instruction following the branch (the instruction at PC+4) will enter the pipeline at time t+1 and the instruction at the branch target will enter the pipeline at time t+2. When the branch instruction is in the fifth stage, the branch condition will be known and some of the instructions

in the pipeline will be changed into no-ops. In the following, assume that 20% of the instructions are branch instructions and that 60% of the branches are "taken".

a) What is the extra CPI that results from control hazards when the architecture assumes that branches are always "not taken"?

If the architecture follows branch is always not taken prediction. Then, we will have 4 bubbles when the branch is taken.

Hence, $IPC_{Extra} = 0.2 * (0.6*4 + 0.4 * 0) = 0.48$

b) What is the extra CPI that results from control hazards when the architecture assumes that branches are always "taken"?

The branch instruction is at PC. The first instruction after branch is fetched from PC+4 and the second is from PC+8 which is the branch target. We will discover the branch condition when the instruction is in the 5^{th} stage. Now the branch predictor always predicts the branch is taken. So, when the branch is actually taken, i.e., 60% of time, we need to kill instruction at PC+4. Whereas, when the branch is not taken, i.e., 40% of time, we need to kill 3 instructions.

Hence, $IPC_{Extra} = 0.2 * (0.6 * 1 + 0.4 * 3) = 0.39$

c) Assuming that 20% of the instructions are branches and denote by X the probability that the branches are "taken". What is the maximum X for which the "branches always not taken" scheme is more efficient than the "branches always taken" scheme?

Cost of "predict branch not taken" scheme = 4X

Cost of "predict branch taken" scheme = x + 3 (1-X)

Hence, 4 X <= X + 3 (1-X) if X <= 0.5

Question 3 (30 points):

(a) EX/MEM → ID/EX, MEM/WB → ID/EX, FPmult → ID/EX, FPadd → ID/EX, div → ID/EX, MEM/WB → EX/MEM, FPmult → EX/MEM, FPadd → EX/MEM, div → EX/MEM.
Where FPmult is the last stage in the multiply unit, and FPadd is the last stage in the add unit.

(b) 12 cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
fld f2, 0(x1)	IF	ID	EX	M	WB																	
fmult.d f4, f2, f0		IF	ID	ID	M1	M2	M3	M4	WB													
fld f6, 800(x1)			IF	IF	ID	EX	M			WB												
fadd.d f6, f4, f6					IF	ID	ID	ID	A1	A2	WB											
fsd 1600(x1), f6						IF	IF	IF	ID	EX	M											
addi x1, x1, 8									IF	ID	EX	M	WB									
bne x1, x2, L										IF	ID	EX	M	WB								
add x3, x4, x5											IF	ID										
add x6, x7, x8												IF										
fld f2, 0(x1)													IF	ID	EX	M	WB					
fmult.d f4, f2, f0														IF	ID	ID	M1	M2	M3	M4	WB	
fld f6, 800(x1)															IF	IF	ID	EX	M			WB
fadd.d f6, f4, f6																						

(c) 1200 cycles

(d) Will not change the number of cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
fld f2, 0(x1)	IF	ID	EX	M	WB																	
fmult.d f4, f2, f0		IF	ID	ID	M1	M2	M3	M4	WB													
fld f6, 800(x1)			IF	IF	ID	EX	M			WB												
fadd.d f6, f4, f6					IF	ID	ID	ID	A1	A2	WB											
addi x1, x1, 8						IF	IF	IF	ID	EX	M	WB										
fsd 1592(x1), f6									IF	ID	EX	M										
bne x1, x2, L										IF	ID	EX	M	WB								
add x3, x4, x5											IF	ID										
add x6, x7, x8												IF										
fld f2, 0(x1)													IF	ID	EX	M	WB					
fmult.d f4, f2, f0														IF	ID	ID	M1	M2	M3	M4	WB	
fld f6, 800(x1)															IF	IF	ID	EX	M			WB
fadd.d f6, f4, f6																						

Question 4 (35 points):

(a)

```
L: fld     f2, 0(x1)
   fld     f3, 8(x1)
   fmult.d f4, f2, f0
   fmult.d f5, f3, f0
   fld     f6, 800(x1)
   fld     f7, 808(x1)
   fadd.d  f6, f4, f6
   fadd.d  f7, f5, f7
   fsd     1600(x1), f6
   fsd     1608(x1), f7
   addi    x1, x1, 16
   bne     x1, x2, L
```

(b)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
fld f2, 0(x1)	IF	ID	EX	M	WB																	
fld f3, 8(x1)		IF	ID	EX	M	WB																
fmult.d f4, f2, f0			IF	ID	M1	M2	M3	M4	WB													
fmult.d f5, f3, f0				IF	ID	M1	M2	M3	M4	WB												
fld f6, 800(x1)					IF	ID	EX	M	M	M	WB											
fld f7, 808(x1)						IF	ID	EX	EX	EX	M	WB										
fadd.d f6, f4, f6							IF	ID	A1	A2	A2	A2	WB									
fadd.d f7, f5, f7								IF	ID	ID	ID	A1	A2	WB								
fsd 1600(x1), f6									IF	IF	IF	ID	EX	M								
fsd 1608(x1), f7												IF	ID	EX	M							
addi x1, x1, 8													IF	ID	EX	M	WB					
bne x1, x2, L														IF	ID	EX	M					
															IF	ID	EX	M	WB			
																IF	ID	EX	M	WB		
fld f2, 0(x1)																	IF	ID	EX	M	WB	

It takes 16 cycles to complete one iteration of the unrolled loop (assuming branch is resolved in EX).

Notes: no two instructions can be in the same stage during the same cycle

fsd and bne do not use WB

the second fadd.d cannot go into A1 before the data to be written in f5 and f7 is available

(c)

The 50 iterations of the unrolled loop takes 16 * 50 = 800 cycles

Speedup = 1200/800 = 1.5

(d)

The table in part (b) shows that the 12 instructions in each iteration of the unfolded loop completes in 16 cycles. Two wasted cycles are due to control hazard and two are due to data and structural hazards. Hence, there is some room (2 wasted cycles) for reducing data/structural hazards by unrolling the loop further. Moreover, further unrolling reduces the loop control overhead (reduces the number of addi and bne instructions) as well as reducing control hazards by reducing the number of times the "mis predicted branch" executes.

In my view, however, the complexity of further unfolding does not justify the expected improvement in execution time.

Note that if we completely unroll the loop, we still have to execute 5 * 100 = 500 instructions that will take at least 500 cycles to complete (cannot fetch more than one instruction per cycle – that is minimum CPI = 1).