CS2100 Computer Organisation

Tutorial #10: Pipelining

(Week 12: 7 – 11 April 2025)

Answers to Selected Questions

Tutorial Questions

1. [AY2014/5 Semester 2 Exam]
   Refer to the following MIPS program:

```
      # register $s0 contains a 32-bit value
      # register $s1 contains a non-zero 8-bit value
      #       at the right most (least significant) byte
      add  $t0, $s0, $zero      #inst A
      add  $s2, $zero, $zero    #inst B
lp:   bne  $s2, $zero, done     #inst C
      beq  $t0, $zero, done     #inst D
      andi $t1, $t0, 0xFF       #inst E
      bne  $s1, $t1, nt         #inst F
      addi $s2, $s2, 1          #inst G
nt:   srl  $t0, $t0, 8          #inst H
      j    lp                   #inst J
done:
```

We assume that the register **$s0** contains **0xAFAFFAFA** and **$s1** contains **0xFF**.

Given a 5-stage MIPS pipeline processor, for each of the parts below, give the total number of cycles needed for the first iteration of the execution from instructions **A** to **H** (i.e. excluding the "**j lp**" instruction). Remember to include the cycles needed for instruction **H** to finish the WB stage. Note that the questions are independent from each other.

(a) With only data forwarding mechanisms and no control hazard mechanism.

(b) With data forwarding and "assume not taken" branch prediction. Note that there is no early branching.

[Recall that early branching means branch decision is made at stage 2 (Decode stage); no early branch means branch decision is made at stage 4 (Memory stage).]

(c) By swapping two instructions (from Instructions **A** to **H**), we can improve the performance of **early branching (with all additional forwarding paths)**. Give the two instructions that can be swapped. You only need to indicate the instruction letters in your answer.

Give the total number of cycles needed for the execution of the whole code in the worst case for each of the following assumptions. You may assume that the jump instruction (**j**) computes the address of the instruction to jump to in the MEM stage.

(d) With only data forwarding mechanisms and no control hazard mechanism.

(e) With data forwarding and "assume not taken" branch prediction. Note that there is no early branching.

Answers:

(a) 20 cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
add	F	D	E	M	W															
add		F	D	E	M	W														
bne			F	D	E	M	W													
beq							F	D	E	M	W									
andi											F	D	E	M	W					
bne												F	D	E	M	W				
~~addi~~		The addi instruction is not executed.																		
srl																F	D	E	M	W

(b) 14 cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
add	F	D	E	M	W									
add		F	D	E	M	W								
bne			F	D	E	M	W							
beq				F	D	E	M	W						
andi					F	D	E	M	W					
bne						F	D	E	M	W				
addi							F	D	E	*	*			
srl								F	D	*	*	*		
j									F	*	*	*	*	
srl										F	D	E	M	W

Cost of wrong prediction

(c) Swap instructions A and B to reduce the delay between instructions B and C.

```
     add  $s2, $zero, $zero   #inst B
     add  $t0, $s0, $zero     #inst A
lp:  bne  $s2, $zero, done    #inst C
     beq  $t0, $zero, done    #inst D
     andi $t1, $t0, 0xFF      #inst E
     bne  $s1, $t1, nt        #inst F
     addi $s2, $s2, 1         #inst G
nt:  srl  $t0, $t0, 8         #inst H
     j    lp                  #inst J
done:
```

(d)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
add	F	D	E	M	W																
add		F	D	E	M	W															
bne			F	D	E	M	W														
beq							F	D	E	M	W										
andi											F	D	E	M	W						
bne												F	D	E	M	W					
~~addi~~																					
srl																F	D	E	M	W	
j																	F	D	E	M	W
bne																					F

In the worst case, 4 bytes of the data are examined → 4 iterations. The loop from Instructions C to J (one iteration) takes 18 cycles (not counting the WB stage of the j instruction which overlaps with the bne instruction). There are 2 cycles before the first iteration, and 9 cycles for Instructions C and D in the fifth iteration.
Therefore, total = 2 + (4×18) + 9 = **83 cycles**.

(e)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
add	F	D	E	M	W										
add		F	D	E	M	W									
bne			F	D	E	M	W								
beq				F	D	E	M	W							
andi					F	D	E	M	W						
bne						F	D	E	M	W					
addi							F	D	E	*	*				
srl								F	D	*	*	*			
j									F	*	*	*	*		
srl										F	D	E	M	W	
j											F	D	E	M	W
bne															F

In the worst case, 4 bytes of the data are examined → 4 iterations. The loop from Instructions C to J (one iteration) takes 12 cycles (not counting the WB stage of the j instruction which overlaps with the bne instruction). There are 2 cycles before the first iteration, and 6 cycles for Instructions C and D in the fifth iteration.
Therefore, total = 2 + (4×12) + 6 = **56 cycles**.

2. [AY2017/8 Semester 2 Exam]

   Refer to the MIPS code below. *A* and *B* are integer arrays whose base addresses are in **$s0** and **$s1** respectively. The arrays are of the same size *n* (number of elements). **$s2** contains the value *n*. For this question, we will focus on the code from Instruction 1 onwards.

```
 .data
A: .word 11, 9, 31, 2, 9, 1, 6, 10
B: .word 3, 7, 2, 12, 11, 41, 19, 35
n: .word 8
.text
main: la   $s0, A     # $s0 is the base address of array A
      la   $s1, B     # $s1 is the base address of array B
      la   $t0, n     # $t0 is the addr of n (size of array)
                      # $s2 is the content of n
      beq  $s2, $zero, End   # Inst1
      addi $t8, $s2, -1      # Inst2
      sll  $t8, $t8, 2       # Inst3
Loop: add  $t0, $s0, $t8     # Inst4
      add  $t1, $s1, $t8     # Inst5
      lw   $t2, 0($t0)       # Inst6
      lw   $t3, 0($t1)       # Inst7
      andi $t4, $t3, 3       # Inst8
      addi $t4, $t4, -3      # Inst9
      beq  $t4, $zero, A1    # Inst10
      add  $t2, $t2, $t3     # Inst11
      j    A2                # Inst12
A1:   addi $t2, $t2, 1       # Inst13
A2:   sw   $t2, 0($t0)       # Inst14
      addi $t8, $t8, -8      # Inst15
      slt  $t7, $t8, $zero   # Inst16
      beq  $t7, $zero, Loop  # Inst17
End:
```

   Assuming a 5-stage MIPS pipeline system with forwarding and early branching, that is, the branch decision is made at the ID stage. No branch prediction is made and no delayed branching is used. For the jump (**j**) instruction, the computation of the target address to jump to is done at the ID stage as well.

   Assume also that the first **beq** instruction begins at cycle 1.

(a) Suppose arrays *A* and *B* now each contains 200 positive integers. What is the minimum number and maximum number of instructions executed? (Consider only the above code segment from Inst1 to Inst17.)

(b) List out the instructions where some stall cycle(s) are inserted in executing that instruction in the pipeline. These include delay caused by data dependency and control hazard. You may write the instruction number InstX instead of writing out the instruction in full.

(c) How many cycles does one iteration of the loop (from Inst1 to Inst17) take if the **beq** instruction at Inst10 branches to *A1*? You have to count until the WB stage of Inst17.

(d) How many cycles does one iteration of the loop (from Inst1 to Inst17) take if the **beq** instruction at Inst10 does not branch to *A1*? You have to count until the WB stage of Inst17.

Answers:

The code does this:

```
int i;
for (i=n-1; i>=0; i-=2) {
   if (B[i]%4 == 3)
      A[i] = A[i] + 1;
   else
      A[i] = A[i] + B[i];
}
```

(a) **Minimum = 3 + 100 × 12 = 1203**

Maximum = 3 + 100 × 13 = 1303

In the loop (Inst4 to Inst17), there are two paths after Inst10: one that skips Inst11 and Inst12, and the other skips Inst13.

(b) **Due to control: Inst2, Inst4, Inst11, Inst13**

Due to data: Inst8, Inst10, Inst17

(c) **24 cycles**

(d) **26 cycles**

3. [AY2020/21 Semester 2 Exam]
Study the following MIPS code on integer arrays *A* and *B* which contain the same number of elements. $s0 and $s1 contain the base addresses of *A* and *B* respectively; $s2 is the number of elements in array *A*; $s5 is *count*.

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
end:
```

Assuming a 5-stage MIPS pipeline and all elements in array *A* are positive odd integers, answer the following questions. You need to count until the last stage of instruction I16.

(a) How many cycles does this code segment take to complete its execution in the first iteration (I1 to I16) in an ideal pipeline, that is, one with no delays?

For parts (b) to (d) below, given the assumption for each part, how many additional cycles does this code segment (I1 to I16) take to complete its execution in the first iteration as compared to an ideal pipeline? (For example, if part (a) takes 12 cycles and part (b) takes 20 cycles, you are to answer part (b) with the value 8 and not 20.)

(b) Assuming without forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made and no delayed branching is used.

(c) Assuming without forwarding and branch decision is made at ID stage (stage 2). No branch prediction is made and no delayed branching is used.

(d) Assuming with forwarding and branch decision is made at ID stage (stage 2). Branch prediction is made where the branch is predicted not taken, and no delayed branching is used.

(e) Assuming the setting in part (d) above and you are not allowed to modify any of the instructions, is it possible to reduce the additional delay cycles in part (d) by rearranging some instructions, and if possible, by how many cycles? Explain your answer. (Answer with no explanation will not be awarded any mark.)

Answers:

(a) 16 + 5 – 1 = **20** cycles.

Delays are highlighted under the columns (b), (c), (d) for parts (b),(c),(d) below respectively.

(b) **+24** cycles.

(c) **+20** cycles.

(d) **+4** cycles.

```
                                     (b)   (c)   (d)
      add  $s5, $0, $0     # I1
      add  $t0, $0, $0     # I2
loop: slt  $t8, $t0, $s2   # I3      +2    +2
      beq  $t8, $0, end    # I4      +2    +2    +1
      sll  $t1, $t0, 2     # I5      +3    +1
      add  $t3, $t1, $s0   # I6      +2    +2
      lw   $s3, 0($t3)     # I7      +2    +2
      andi $t9, $s3, 1     # I8      +2    +2    +1
      beq  $t9, $0, skip   # I9      +2    +2    +1
      add  $t4, $t1, $s1   # I10     +3    +1
      lw   $s4, 0($t4)     # I11     +2    +2
      sub  $s3, $s3, $s4   # I12     +2    +2    +1
      sw   $s3, 0($t3)     # I13     +2    +2
      addi $s5, $s5, 1     # I14
skip: addi $t0, $t0, 1     # I15
      j    loop            # I16
end:
                          Total:     +24   +20   +4
```

(e) Other answers possible. Example:

Move I14 (addi $s5, $s5, 1) to between I11 (lw $s4, 0($t4)) and I12 (sub $s3, $s3, $s4) to remove the 1 cycle of delay at I12.