INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ORGANIZAÇÃO DE COMPUTADORES

LEIC

Conjunto de Exercícios VIII
Pipelining

Versão 1.1

2022/2023

Exercício 1*

In this exercise, we examine how pipelining affects the clock cycle time of the processor. Assume that the individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

(a) What is the clock cycle time in a pipelined and non-pipelined processor?

(b) What is the total latency of an `lw` instruction in a pipelined and non-pipelined processor?

(c) What is the throughput of the pipelined and non-pipelined processor?

(d) If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

Exercício 2†

2.1. In this exercise, we examine how data dependencies affect execution in the basic 5-stage pipeline described in Section 4.5 of the textbook [1]. Consider the following sequence of instructions:

```
sw     $t1, -100($t1)
lw     $t2, 8($t1)
add    $t3, $t2, $t2
```

(a) Indicate the data dependencies.

(b) Assume there is no forwarding in this pipelined processor. Indicate the existing hazards and add nop instructions to eliminate them.

(c) Assume there is full forwarding. Indicate the existing hazards and add nop instructions to eliminate them.

2.2. In addition to the instruction sequence shown above, consider the values shown in the table below. These values represent the latencies for the slowest pipeline stage for three different setups: without forwarding, with full forwarding, and with ALU-ALU forwarding only.

Without forwarding	With full forwarding	W/ ALU-ALU forwarding only
250ps	300ps	290ps

(a) What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

(b) Add `nop` instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

(c) What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

*Exercícios 4.12.1—4.12.3 de [1].
†Exercícios 4.13.1—4.13.6 de [1].

Exercício 3[‡]

Consider the following loop. Assume (i) that perfect branch prediction is used (no stalls due to control hazards), (ii) that there are no delay slots, and (iii) that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

```
Loop:  add    $t1, $t2, $t1
       lw     $t2, 0($t1)
       lw     $t9, 16($t2)
       slt    $t1, $t2, $t4
       beq    $t1, $t9, Loop
```

(a) Identify the existing data dependencies in the code above.

(b) Assume that no forwarding has been implemented yet. Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration). What is the resulting CPI for this loop iteration?

(c) Consider now that the pipeline has full forwarding support. Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration). What is the resulting Speed Up for this loop given the full forwarding support?

(d) How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

(e) At the start of the cycle in which we fetch the first instruction of the third iteration of this loop, what is stored in the IF/ID register?

(f) Consider now a VLIW version of this processor identical to the one above, but supporting the issue and execution of two instruction simultaneously, namely ALU & L
S or ALU & Beq. Rewrite the code accordingly. What is the obtained Speed Up.

(g) Perform a two fold loop unrolling of the code assuming that the loop is always performed a even number of times.

(h) Rewrite the code targeting the above described VLIW version of the processor considering the unfolded code performed above.

Exercício 4[§]

This exercise is intended to help understand the relationship between delay slots, control hazards, and branch execution in a pipelined processor. Assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```
Label: lw   $t2, 0($t2)
       beq  $t2, $t4, Label
       or   $t2, $t2, $t3
       sw   $t2, 0($t5)
```

[‡]Exercícios 4.16.4—4.16.6 de [1].
[§]Exercícios 4.22.1—4.22.3 de [1].

(a) Draw the pipeline execution diagram for this code, assuming that: the branch is taken once, and then not taken; there are no delay slots; and that branches are resolved in the EX stage.

(b) Repeat 4.a, but assume that delay slots are used. In the given code, the instruction that follows the branch is now the delay slot instruction for that branch.

(c) One way to move the branch resolution one state earlier is to not need an ALU operation in conditional branches. The branch instructions would be "`bez Rd, Label`" and "`bnez Rd, Label`", and it would branch if the register has and does not have a zero value, respectively. Change this code to use these branch instructions instead of `beq`. Assume that register `$t8` is available for being used as a temporary register, and that an `seq` (set if equal) R-type instruction can be used.

Referências

[1] David Patterson and John Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 4$^{\text{th}}$ edition, 2011.

Conjunto de Exercícios VIII
Pipelining

Soluções

Exercício 1

(a) $T_{non\text{-}pipelined}$ = 1250 ps

$T_{pipelined}$ = 350 ps

(b) $T_{non\text{-}pipelined}$(`lw`) = 1.25 ns

$T_{pipelined}$(`lw`) = 1.75 ns

(c) Non-pipelined: 800 MIPS.
Pipelined: 2857 MIPS.

(d) Choose the ID stage.

$T'_{pipelined}$ = 300 ps.

Exercício 2

2.1.

(a) `$t2` from I3 depends on `$t2` from I2: read after write (RAW).

(b) With no forwarding, data is available only on stage W. In the basic five-stage pipeline WAR and WAW dependences do not cause any hazards. Without forwarding, any RAW dependence between an instruction and the next two instructions (if register read happens in the second half of the clock cycle and the register write happens in the first half) causes hazards.

1. Suppose we feed one instruction per clock cycle without worries about hazards.

`sw $t1, -100($t1)`	F	D	X	M	W				
`lw $t2, 8($t1)`		F	D	X	M	**W**			
`add $t3, $t2, $t2`			F	**D**	X	M	W		

There is one data hazard on $t2 in I2.

To eliminate hazards, insert two NOPs in order to align I3-D with I2-M, which is where the value for $t2 gets read from memory:

```
sw      $t1, -100($t1)
lw      $t2, 8($t1)
nop
nop
add     $t3, $t2, $t2
```

(c) With full forwarding any stage can forward results to any other stage. In this case I3 has to wait until M yields the $t2 value from memory.

`sw $t1, -100($t1)`	F	D	X	M	W				
`lw $t2, 8($t1)`		F	D	X	**M**	W			
`add $t3, $t2, $t2`			–	F	**D**	X	M	W	

The code that eliminates these hazards by inserting NOP instructions is:

```
sw      $t1, -100($t1)
lw      $t2, 8($t1)
nop
add     $t3, $t2, $t2
```

Delay I3 to avoid RAW hazard on R4 from I2. Value for R4 is forwarded from I2 now.

2.2.

(a)

Without forwarding	With forwarding	Speedup Due to Forwarding
(7 + 2) x 250ps = 2250ps	(7 + 1) x 300ps = 2400ps	$T_{w/ofw}$ / $T_{w/fw}$ = 0.94

Since Speedup < 1, this is a slowdown.

(b) ALU-ALU forwarding cannot eliminate the hazard in this case.

Thus, we have:

```
sw      $t1, -100($t1)
lw      $t2, 8($t1)
nop
nop
add     $t3, $t2, $t2
```

That is, this is similar to the no-forwarding case.

(c)

No forwarding	With ALU-ALU Forwarding Only	Speedup with ALU-ALU Forwarding
(7 + 2) x 250ps	(7 + 2) x 290ps	$T_{w/ofw}$ / $T_{w/fw}$ = 0.86

Because Speedup < 1, there's still a slowdown.

Exercício 3

(a) RAW (read after write) dependencies:

- $t1 from I2 depends on $t1 from I1
- $t2 from I3 depends on $t2 from I2

- $t2 from I4 depends on $t2 from I2
- $t1 from I5 depends on $t1 from I4
- $t9 from I5 depends on $t9 from I3

Technically, given the loop nature of the code, there are some data hazards related to the previous iteration of the loop, so it should also include:

- $t2 from I1 depends on $t2 from I2 (from the previous loop iteration)
- $t1 from I1 depends on $t2 from I4 (from the previous loop iteration)

(b)

`lw $t9, 16($t2)`	M	W															
`slt $t1, $t2, $t4`	X	M	W														
`beq $t1, $t9, loop`	D	–	–	X	M	W											
`add $t1, $t2, $t1`	F	–	–	D	X	M	W										
`lw $t2, 0($t1)`				F	D	–	–	X	M	W							
`lw $t9, 16($t2)`					F	–	–	D	–	–	X	M	W				
`slt $t1, $t2, $t4`								F	–	–	D	X	M	W			
`beq $t1, $t9, loop`											F	D	–	–	X	M	W
`add $t1, $t2, $t1`												F	–	–	D	X	M

$$CPI = 2.2\,CPI$$

(c)

`lw $t9, 16($t2)`	M	W									
`slt $t1, $t2, $t4`	X	M	W								
`beq $t1, $t9, loop`	D	X	M	W							
`add $t1, $t2, $t1`	F	D	X	M	W						
`lw $t2, 0($t1)`		F	D	X	M	W					
`lw $t9, 16($t2)`			F	D	X	–	M	W			
`slt $t1, $t2, $t4`				F	D	–	X	M	W		
`beq $t1, $t9, loop`					F	–	D	X	M	W	

$$CPI = 1.2\,CPI$$

$$SpeedUp = 1.83$$

(d) There is only 1 cycle that makes full use of the pipeline, yielding 1 / 6 = 16.7%.

(e) IF/ID register has: (i) PC + 4 for `beq`, and (ii) instruction for `beq`.

(f)
```
add     $t1, $t2, $t1 ; nop
lw      $t2, 0($t1) ; nop
lw      $t9, 16($t2) ; slt     $t1, $t2, $t4
beq     $t1, $t9, Loop ; nop
```

(g)
```
Loop:     add     $t1, $t2, $t1
          lw      $t2, 0($t1)
          lw      $t9, 16($t2)
```

```
            slt     $t1, $t2, $t4
            add     $t1, $t2, $t1
            lw      $t2, 0($t1)
            lw      $t9, 16($t2)
            slt     $t1, $t2, $t4
            beq     $t1, $t9, Loop
```

(h)
```
add     $t1, $t2, $t1 ; nop
lw      $t2, 0($t1) ; nop
lw      $t9, 16($t2) ; slt     $t1, $t2, $t4
add     $t1, $t2, $t1 ; nop
lw      $t2, 0($t1) ; nop
lw      $t9, 16($t2) ; slt     $t1, $t2, $t4
beq     $t1, $t9, Loop ; nop
```

Exercício 4

(a)

`lw $t2, 0($t2)`	F	D	X	M	W									
`beq $t2, $t4, Label`		F	D	–	X	M	W							
`lw $t2, 0($t2)`			F	–	D	X	M	W						
`beq $t2, $t4, Label`					F	D	–	X	M	W				
`lw $t2, 0($t2)`						F	–	D	~~X~~	~~M~~	~~W~~			
`beq $t2, $t4, Label`								F	~~D~~	~~X~~	~~M~~	~~W~~		
`or $t2, $t2, $t3`									F	D	X	M	W	
`sw $t2, 0($t5)`										F	D	X	M	W

(b)

`lw $t2, 0($t2)`	F	D	X	M	W									
`beq $t2, $t4, Label`		F	D	–	X	M	W							
`or $t2, $t2, $t3`			F	–	D	X	M	W						
`lw $t2, 0($t2)`					F	D	X	M	W					
`beq $t2, $t4, Label`						F	D	–	X	M	W			
`or $t2, $t2, $t3`							F	–	D	X	M	W		
`lw $t2, 0($t2)`									F	~~D~~	~~X~~	~~M~~	~~W~~	
`sw $t2, 0($t5)`										F	D	X	M	W

(c)
```
        Label: lw   $t2, 0($t2)
               seq  $t8, $t2, $t4
               bnez $t8, Label
               or   $t2, $t2, $t3
               sw   $t2, 0($t5)
```