CS3350B Computer Organization
Chapter 4: Instruction-Level Parallelism
Hazard Examples

Iqra Batool

Department of Computer Science
University of Western Ontario, Canada

Monday March 4, 2024

Introduction

- In pipelining examples, assume we always start with the “basic” datapath; the one as of the end of Lecture 11.
  - ↳ This datapath implicitly already solves the two structural hazards in memory and register file.
  - ↳ That is, we do not consider structural hazards.
- Each optimization should be explicitly added in the question or in your answer for a possible resolution.
  - ↳ Each type of forwarding (ALU-ALU, MEM-ALU, MEM-MEM).
  - ↳ Filling the load delay slot with something other than `nop`.
  - ↳ Branch comparator in ID stage.
  - ↳ Delayed branching and branch delay slot.

Example 1

```
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
subu  $t4, $t0, $t3
addi  $s1, $s1, -4
add   $t1, $t1, $t2
```

- If any dependencies exist where are they and what type are they?

Example 1

```
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
subu  $t4, $t0, $t3
addi  $s1, $s1, -4
add   $t1, $t1, $t2
```

- If any dependencies exist where are they and what type are they?
  - ↳ Load-use (RAW) between `lw` and `addu`.
  - ↳ WAW between `lw` and `addu`.
  - ↳ RAW between `addu` and `sub`.

Example 1

```
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
subu  $t4, $t0, $t3
addi  $s1, $s1, −4
add   $t1, $t1, $t2
```

- On the basic datapath, how many cycles does it take to execute the code fragment (including stalls)?

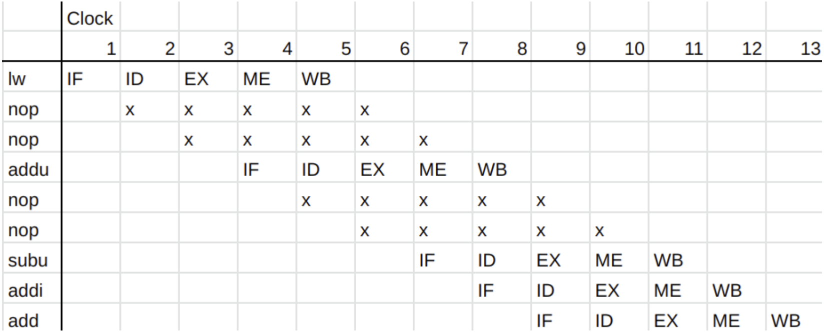
Example 1

```
lw   $t0, 0($s1)
addu $t0, $t0, $s2
subu $t4, $t0, $t3
addi $s1, $s1, −4
add  $t1, $t1, $t2
```

- On the basic datapath, how many cycles does it take to execute the code fragment (including stalls)?
  - 2 `nop` between `lw` and `addu`. MEM of `lw` and IF of `addu` can overlap.
  - 2 `nop` between `addu` and `subu`. MEM of `addu` and IF of `subu` can overlap.
  - On 5th cycle `lw` completes and then one cycle per instruction after that.
  - Including `nop` we get: $5 + 2$ `nop` $+ 1 + 2$ `nop` $+ 2 + 1 = 13$.

Example 1

```
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
subu  $t4, $t0, $t3
addi  $s1, $s1, —4
add   $t1, $t1, $t2
```



	Clock												
	1	2	3	4	5	6	7	8	9	10	11	12	13
lw	IF	ID	EX	ME	WB								
nop		x	x	x	x	x							
nop			x	x	x	x	x						
addu				IF	ID	EX	ME	WB					
nop					x	x	x	x	x				
nop						x	x	x	x	x			
subu							IF	ID	EX	ME	WB		
addi								IF	ID	EX	ME	WB	
add									IF	ID	EX	ME	WB

Example 1

```
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
subu  $t4, $t0, $t3
addi  $s1, $s1, −4
add   $t1, $t1, $t2
```

- What optimizations can be added to the datapath to reduce the number of cycles? How many cycles are needed to execute the code fragment after optimizations are added?

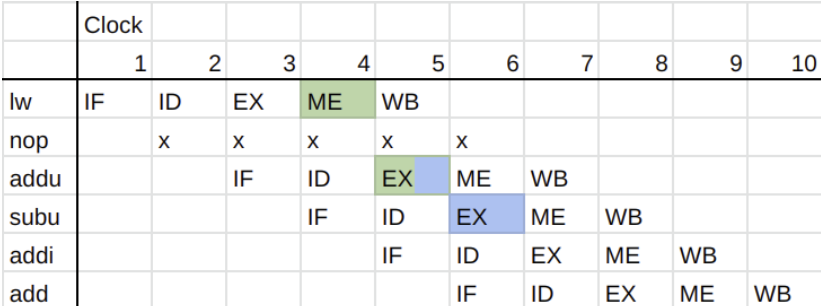
Example 1

```
lw   $t0, 0($s1)
addu $t0, $t0, $s2
subu $t4, $t0, $t3
addi $s1, $s1, -4
add  $t1, $t1, $t2
```

- What optimizations can be added to the datapath to reduce the number of cycles? How many cycles are needed to execute the code fragment after optimizations are added?
  - ↳ MEM-ALU forwarding for load-use. Reduces `nop` count to 1.
  - ↳ ALU-ALU forwarding removes both `nop` between `addu` and `sub`
  - ↳ Clock cycles: 5 + 1 nop + 4 = 10.

Example 1

```
lw    $t0,  0($s1)
addu  $t0,  $t0,  $s2
subu  $t4,  $t0,  $t3
addi  $s1,  $s1,  −4
add   $t1,  $t1,  $t2
```



	Clock									
	1	2	3	4	5	6	7	8	9	10
lw	IF	ID	EX	ME	WB					
nop		x	x	x	x	x				
addu			IF	ID	EX	ME	WB			
subu				IF	ID	EX	ME	WB		
addi					IF	ID	EX	ME	WB	
add						IF	ID	EX	ME	WB

Example 1

```
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
subu  $t4, $t0, $t3
addi  $s1, $s1, −4
add   $t1, $t1, $t2
```

- Can code re-organization along with datapath optimizations be used to further improve the number of clock cycles needed to execute the code? If so, re-order the code and declare any additional optimizations; what is the number of cycles needed to execute the re-ordered code?

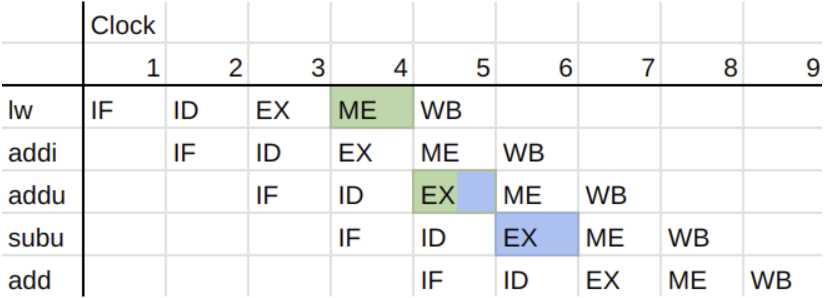
Example 1

```
lw   $t0, 0($s1)
addu $t0, $t0, $s2
subu $t4, $t0, $t3
addi $s1, $s1, −4
add  $t1, $t1, $t2
```

- Can code re-organization along with datapath optimizations be used to further improve the number of clock cycles needed to execute the code? If so, re-order the code and declare any additional optimizations; what is the number of cycles needed to execute the re-ordered code?
  - ↳ Yes.
  - ↳ Move `addi` or `add` into **load-delay slot**.
  - ↳ 9, since we remove the nop.

Example 1

```
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
subu  $t4, $t0, $t3
addi  $s1, $s1, −4
add   $t1, $t1, $t2
```



	Clock								
	1	2	3	4	5	6	7	8	9
lw	IF	ID	EX	ME	WB				
addi		IF	ID	EX	ME	WB			
addu			IF	ID	EX	ME	WB		
subu				IF	ID	EX	ME	WB	
add					IF	ID	EX	ME	WB

Example 2

```
sub   $t2, $t1, $t3
and   $t7, $t2, $t5
or    $t8, $t6, $t2
add   $t9, $t2, $t2
sw    $t5, 12($t2)
```

- If any dependencies exist where are they and what type are they?

Example 2

```
sub   $t2 , $t1 , $t3
and   $t7 , $t2 , $t5
or    $t8 , $t6 , $t2
add   $t9 , $t2 , $t2
sw    $t5 , 12($t2)
```

- If any dependencies exist where are they and what type are they?
  - ↳ RAW between `sub` and `and`.
  - ↳ RAW between `sub` and `or`.
  - ↳ RAW between `sub` and `add`.
  - ↳ RAW between `sub` and `sw`.

Example 2

```
sub   $t2, $t1, $t3
and   $t7, $t2, $t5
or    $t8, $t6, $t2
add   $t9, $t2, $t2
sw    $t5, 12($t2)
```

- Consider the basic datapath with ALU-ALU and MEM-ALU forwarding added. In this code fragment where do forwards occur? How many cycles does it take to execute the code fragment?

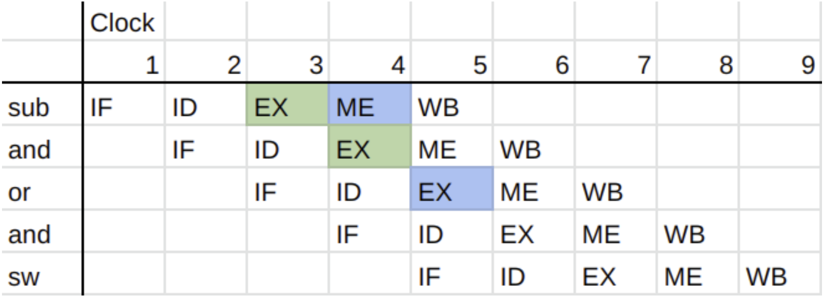
Example 2

```
sub   $t2 , $t1 , $t3
and   $t7 , $t2 , $t5
or    $t8 , $t6 , $t2
add   $t9 , $t2 , $t2
sw    $t5 , 12($t2)
```

- Consider the basic datapath with ALU-ALU and MEM-ALU forwarding added. In this code fragment where do forwards occur? How many cycles does it take to execute the code fragment?
  - ↳ ALU-ALU from `sub` to `and`.
  - ↳ MEM-ALU from `sub` to `or`.
  - ↳ `sub` to `and` RAW solved by register file design.
  - ↳ $5 + 1 + 1 + 1 + 1 = 9$

Example 2

```
sub   $t2, $t1, $t3
and   $t7, $t2, $t5
or    $t8, $t6, $t2
add   $t9, $t2, $t2
sw    $t5, 12($t2)
```



	Clock								
	1	2	3	4	5	6	7	8	9
sub	IF	ID	EX	ME	WB				
and		IF	ID	EX	ME	WB			
or			IF	ID	EX	ME	WB		
and				IF	ID	EX	ME	WB	
sw					IF	ID	EX	ME	WB

Example 3

```
for:  beq   $t6, $t7, end
      add   $t0, $t0, $t1
      addi  $t6, $t6, 1
      j for
end:  sub   $t1, $t6, $0
```

- Assuming the basic data path how many cycles does it take to execute two loops within the code fragment (therefore, excluding the sub)?

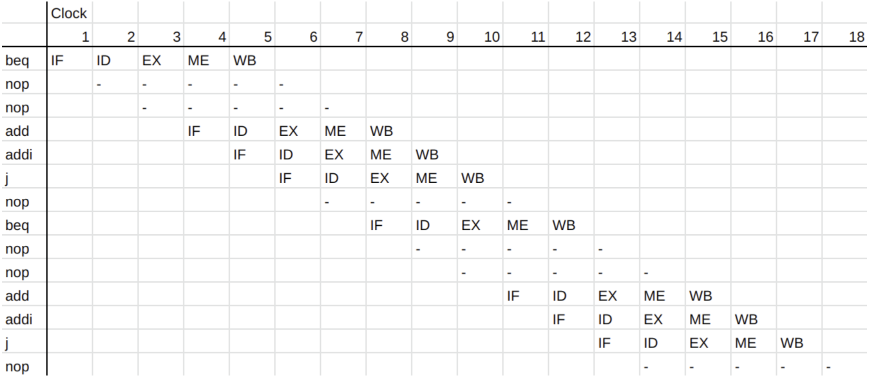
Example 3

```
for:  beq   $t6, $t7, end
      add   $t0, $t0, $t1
      addi  $t6, $t6, 1
      j  for
end:  sub   $t1, $t6, $0
```

- Assuming the basic data path how many cycles does it take to execute two loops within the code fragment (therefore, excluding the `sub`)?
  - ↳ Careful! Since a loop, RAW dependency between `andi` and `beq`.
  - ↳ Two `nop` follows `beq` for control hazard.
  - ↳ One `nop` follows `j` for control hazard.
  - ↳ First loop: 5 + 2 `nop` + 3 + 1 `nop`.
  - ↳ In the second loop `beq` overlaps with previous instructions.
  - ↳ Second loop: 1 + 2 `nop` + 3 + 1 `nop`.
  - ↳ Total: 18.

Example 3

```
for:  beq   $t6, $t7, end
      add   $t0, $t0, $t1
      addi  $t6, $t6, 1
      j  for
end:  sub   $t1, $t6, $0
```



	Clock																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
beq	IF	ID	EX	ME	WB													
nop		-	-	-	-	-												
nop			-	-	-	-	-											
add				IF	ID	EX	ME	WB										
addi					IF	ID	EX	ME	WB									
j						IF	ID	EX	ME	WB								
nop							-	-	-	-	-							
beq								IF	ID	EX	ME	WB						
nop									-	-	-	-	-					
nop										-	-	-	-	-				
add											IF	ID	EX	ME	WB			
addi												IF	ID	EX	ME	WB		
j													IF	ID	EX	ME	WB	
nop														-	-	-	-	-

Example 3

```
for:  beq   $t6, $t7, end
      add   $t0, $t0, $t1
      addi  $t6, $t6, 1
      j  for
end:  sub   $t1, $t6, $0
```

- Using any datapath optimizations and code re-ordering, minimize the clock cycles required to execute the loop two times. Name the optimizations used. How many cycles does it take to execute this optimized version?

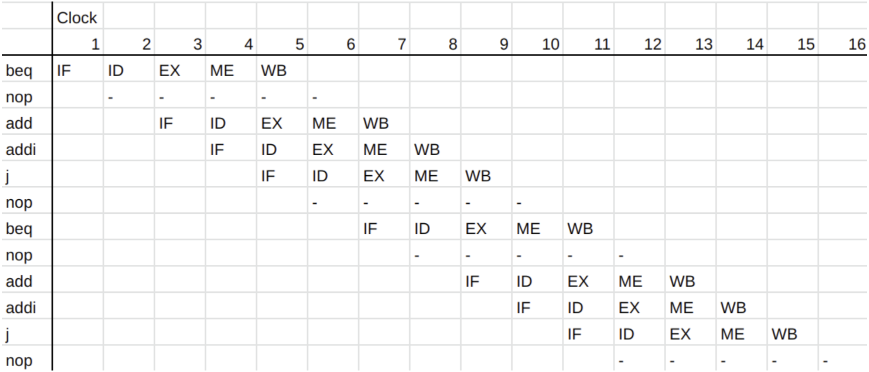
Example 3

```
for:  beq   $t6, $t7, end
      add   $t0, $t0, $t1
      addi  $t6, $t6, 1
      j  for
end:  sub   $t1, $t6, $0
```

- Using any datapath optimizations and code re-ordering, minimize the clock cycles required to execute the loop two times. Name the optimizations used. How many cycles does it take to execute this optimized version?
  - ↳ Special branch comparator in ID stage.
  - ↳ Careful! Cannot fill branch delay slot.
  - ↳ Using `add` would change code meaning.
  - ↳ Value of $t6 used again after loop so cannot use `addi`.
  - ↳ Cannot use jump for obvious control-flow reasons.
  - ↳ Total savings: 1 `nop` per branch ⇒ 16 cycles now.
  - ↳ (If using branch prediction, all `nop`s are removed after beq).

Example 3

```
for:  beq   $t6, $t7, end
      add   $t0, $t0, $t1
      addi  $t6, $t6, 1
      j  for
end:  sub   $t1, $t6, $0
```



	Clock															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
beq	IF	ID	EX	ME	WB											
nop		-	-	-	-	-										
add			IF	ID	EX	ME	WB									
addi				IF	ID	EX	ME	WB								
j					IF	ID	EX	ME	WB							
nop						-	-	-	-	-						
beq							IF	ID	EX	ME	WB					
nop								-	-	-	-	-				
add									IF	ID	EX	ME	WB			
addi										IF	ID	EX	ME	WB		
j											IF	ID	EX	ME	WB	
nop												-	-	-	-	-