Pipeline recap

Emil Viggo Dalsgaard

KØBENHAVNS UNIVERSITET

Agenda

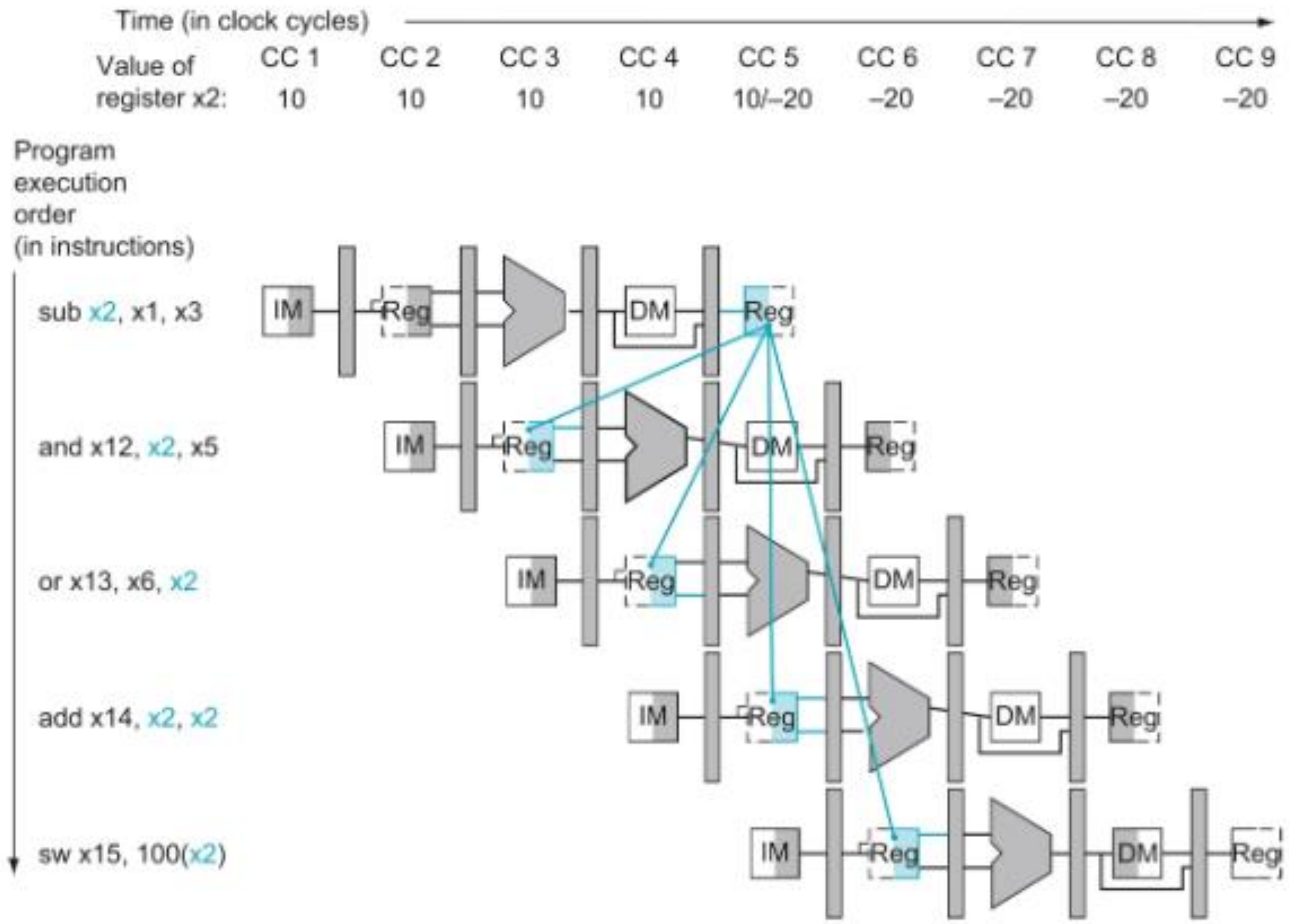
- Some theory
  - Data hazards
  - Full forwarding
  - Branch prediction

- Exam questions:
  - Simple 5-stage pipeline from reexam 2022/2023
  - 2-way in-order superscalar from exam 2022/2023
  - 4-way out-of-order from exam 2023/2024

Data hazards

When an instruction depends on the result from a previous instruction, which hasn't been computed yet

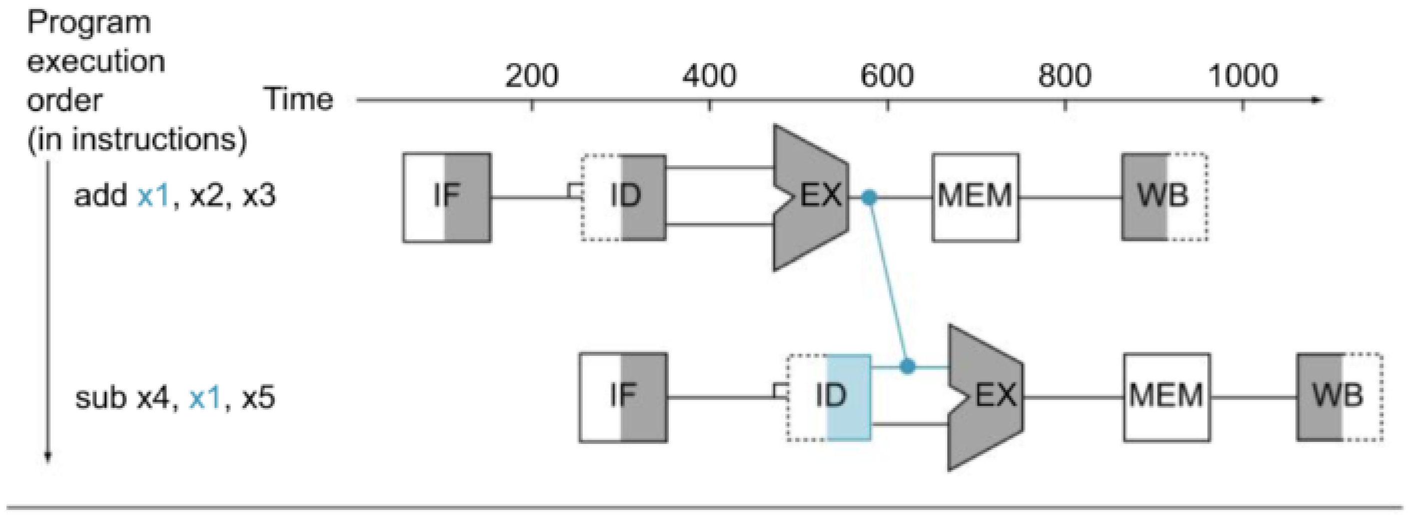
Fixed by stalling (expensive) and/or forwarding



Forwarding

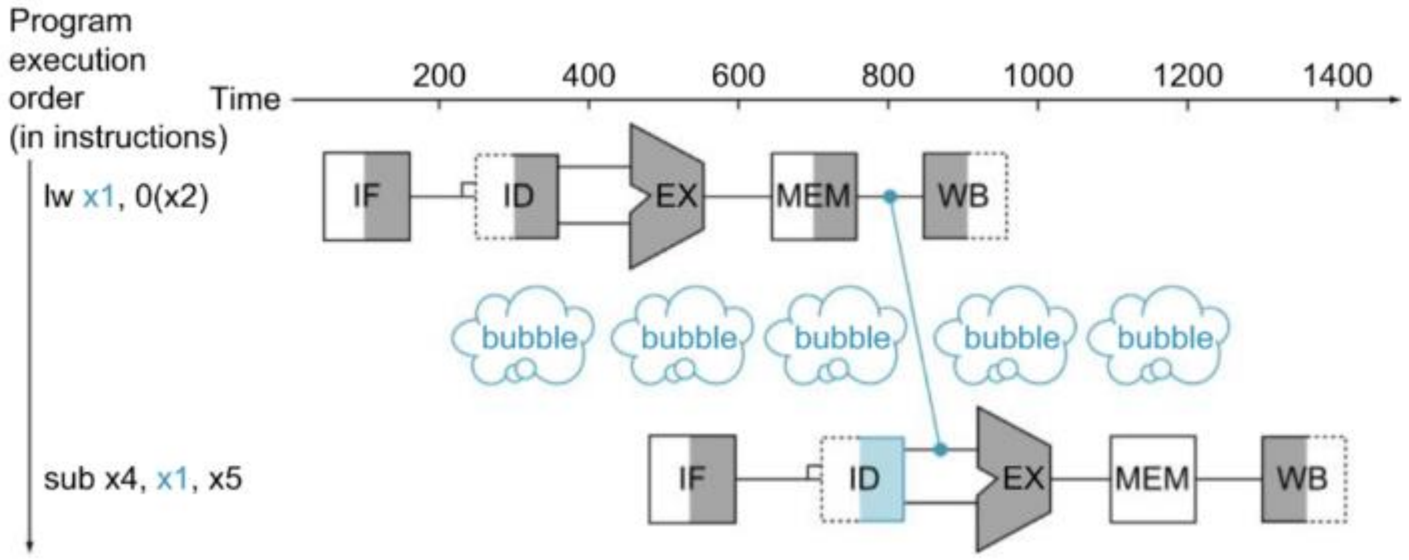
Alu operations:

- Result gets forwarded from the Ex stage



Load operations:

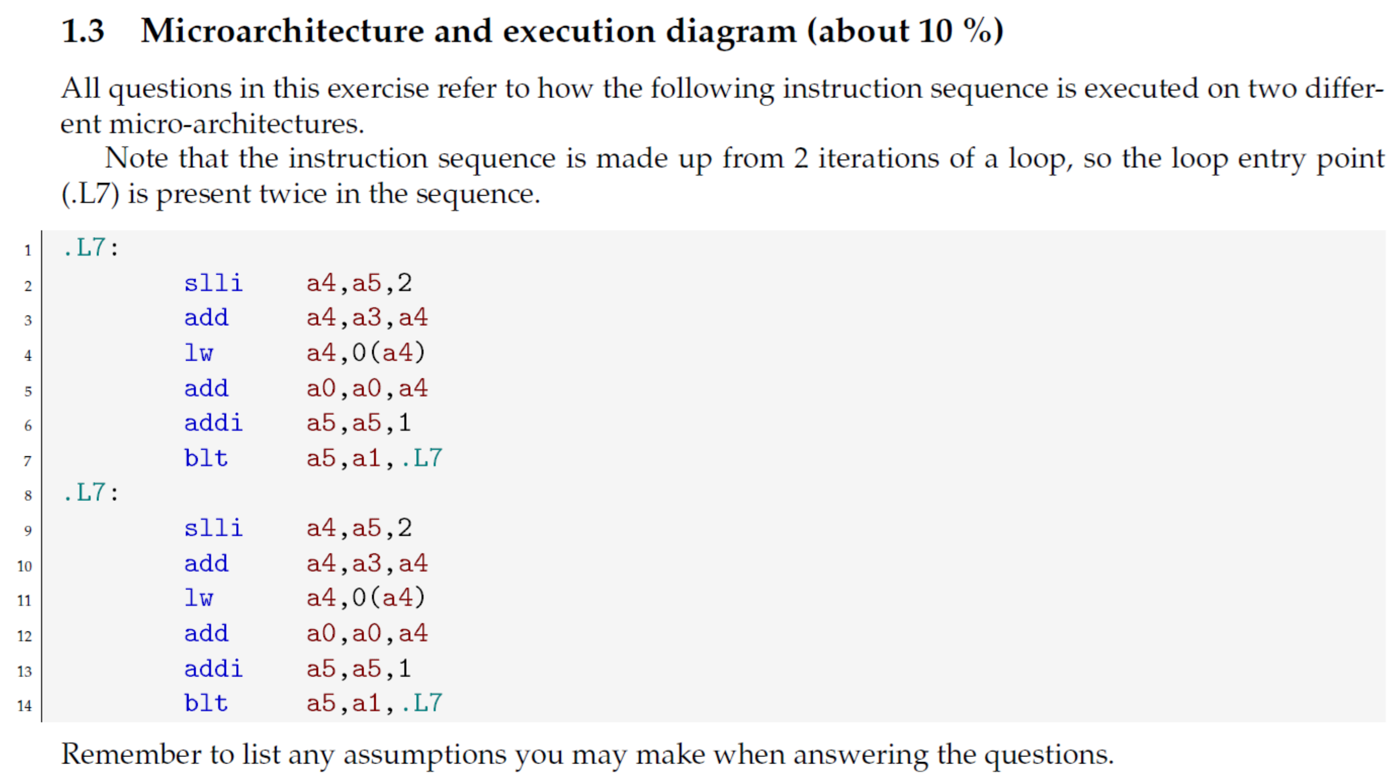
- Result gets forwarded from the Me stage



Branch prediction

- Backward branch taken => Predicted at De stage
  - PC gets written after the decode stage
- Backward branch not taken => Predicted at Ex stage
  - PC gets written after the execute stage
- Forward branch taken => Predicted at Ex stage
  - PC gets written after the execute stage

5-stage simpel pipeline from reexam 2022/2023



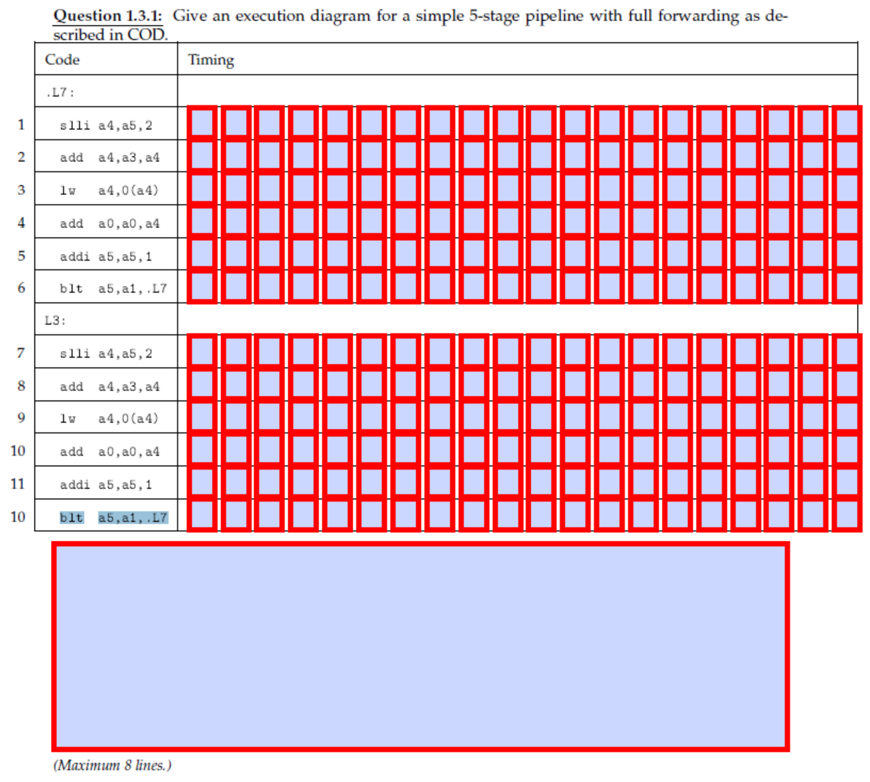
1.3 Microarchitecture and execution diagram (about 10 %)

All questions in this exercise refer to how the following instruction sequence is executed on two different micro-architectures.

Note that the instruction sequence is made up from 2 iterations of a loop, so the loop entry point (.L7) is present twice in the sequence.

```
.L7:
        slli    a4,a5,2
        add     a4,a3,a4
        lw      a4,0(a4)
        add     a0,a0,a4
        addi    a5,a5,1
        blt     a5,a1,.L7
.L7:
        slli    a4,a5,2
        add     a4,a3,a4
        lw      a4,0(a4)
        add     a0,a0,a4
        addi    a5,a5,1
        blt     a5,a1,.L7
```

Remember to list any assumptions you may make when answering the questions.



Question 1.3.1: Give an execution diagram for a simple 5-stage pipeline with full forwarding as described in COD.

	Code	Timing																			
	.L7:																				
1	slli a4,a5,2																				
2	add a4,a3,a4																				
3	lw a4,0(a4)																				
4	add a0,a0,a4																				
5	addi a5,a5,1																				
6	blt a5,a1,.L7																				
	L3:																				
7	slli a4,a5,2																				
8	add a4,a3,a4																				
9	lw a4,0(a4)																				
10	add a0,a0,a4																				
11	addi a5,a5,1																				
10	blt a5,a1,.L7																				

(Maximum 8 lines.)

5-stage simpel pipeline

- Stages:
  - “Fe” - fetch (Fetch instruction)
  - “De” - decode (Decode instruction)
  - “Ex” - execute (Execute arithmetic)
  - “Me” - memory (Memory access)
  - “Wb” - writeback’ (Write to register)

Fe: 1, De: 1, Ex: 1, Me: 1, Wb: 1

We assume backward branches are predicted taken in the De stage

The result from slli gets forwarded

.L7:													
`slli a4,a5,2`	Fe	De	Ex	Me	Wb								
`add a4,a3,a4`		Fe	De	Ex	Me	Wb							
`lw a4,0(a4)`													
`add a0,a0,a4`													
`addi a5,a5,1`													
`blt a5,a1,.L7`													
`L7:`													
`slli a4,a5,2`													
`add a4,a3,a4`													
`lw a4,0(a4)`													
`add a0,a0,a4`													
`addi a5,a5,1`													
`blt a5,a1,.L7`													

.L7:													
`slli a4,a5,2`	Fe	De	Ex	Me	Wb								
`add a4,a3,a4`		Fe	De	Ex	Me	Wb							
`lw a4,0(a4)`			Fe	De	Ex	Me	Wb						
`add a0,a0,a4`				Fe	De	Ex	Ex	Me	Wb				
`addi a5,a5,1`													
`blt a5,a1,.L7`													
`L7:`													
`slli a4,a5,2`													
`add a4,a3,a4`													
`lw a4,0(a4)`													
`add a0,a0,a4`													
`addi a5,a5,1`													
`blt a5,a1,.L7`													

Stall until a4 is forwarded from lw's me stage

.L7:																			
`slli a4,a5,2`	Fe	De	Ex	Me	Wb														
`add a4,a3,a4`		Fe	De	Ex	Me	Wb													
`lw a4,0(a4)`			Fe	De	Ex	Me	Wb												
`add a0,a0,a4`				Fe	De	Ex	Ex	Me	Wb										
`addi a5,a5,1`					Fe	De	De	Ex	Me	Wb									
`blt a5,a1,.L7`						Fe	Fe	De	Ex	Me	Wb								
`L7:`																			
`slli a4,a5,2`																			
`add a4,a3,a4`																			
`lw a4,0(a4)`																			
`add a0,a0,a4`																			
`addi a5,a5,1`																			
`blt a5,a1,.L7`																			

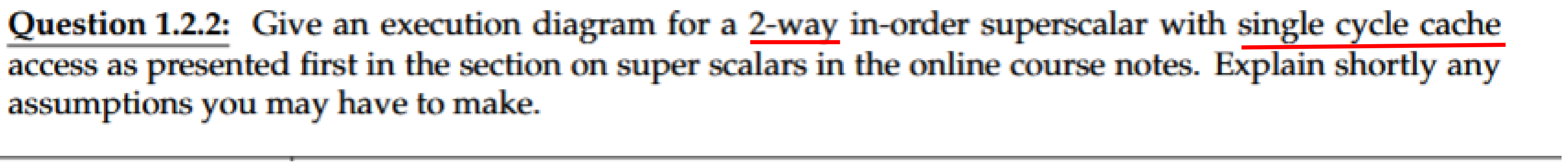
Stall until Ex is available

.L7:																			
`slli a4,a5,2`	Fe	De	Ex	Me	Wb														
`add a4,a3,a4`		Fe	De	Ex	Me	Wb													
`lw a4,0(a4)`			Fe	De	Ex	Me	Wb												
`add a0,a0,a4`				Fe	De	Ex	Ex	Me	Wb										
`addi a5,a5,1`					Fe	De	De	Ex	Me	Wb									
`blt a5,a1,.L7`						Fe	Fe	De	Ex	Me	Wb								
`L7:`																			
`slli a4,a5,2`									Fe	De	Ex	Me	Wb						
`add a4,a3,a4`										Fe	De	Ex	Me	Wb					
`lw a4,0(a4)`											Fe	De	Ex	Me	Wb				
`add a0,a0,a4`												Fe	De	Ex	Ex	Me	Wb		
`addi a5,a5,1`													Fe	De	De	Ex	Me	Wb	
`blt a5,a1,.L7`														Fe	Fe	De	Ex	Me	Wb

Correctly predicted backward branch, and PC gets fetched after De stage

.L7:																			
`slli a4,a5,2`	Fe	De	Ex	Me	Wb														
`add a4,a3,a4`		Fe	De	Ex	Me	Wb													
`lw a4,0(a4)`			Fe	De	Ex	Me	Wb												
`add a0,a0,a4`				Fe	De	Ex	Ex	Me	Wb										
`addi a5,a5,1`					Fe	De	De	Ex	Me	Wb									
`blt a5,a1,.L7`						Fe	Fe	De	Ex	Me	Wb								
`L7:`																			
`slli a4,a5,2`									Fe	De	Ex	Me	Wb						
`add a4,a3,a4`										Fe	De	Ex	Me	Wb					
`lw a4,0(a4)`											Fe	De	Ex	Me	Wb				
`add a0,a0,a4`												Fe	De	Ex	Ex	Me	Wb		
`addi a5,a5,1`													Fe	De	De	Ex	Me	Wb	
`blt a5,a1,.L7`														Fe	Fe	De	Ex	Me	Wb

2-way in-order superscalar with single cycle cache access from exam 2022/2023


Question 1.2.2: Give an execution diagram for a 2-way in-order superscalar with single cycle cache access as presented first in the section on super scalars in the online course notes. Explain shortly any assumptions you may have to make.

Stages for the instructions:

- Load:

Fe	De	Ag	Me	Wb

- Store:

Fe	De	Ag	Me

- Other:

Fe	De	Ex	Wb

- branch:

Fe	De	Ex

Fe: 2, De: 2, Ag: 1, Me: 1, Wb: 2

We assume backward branches are predicted taken in the De stage

L3:																			
`sb a5,0(a1)`	Fe	De	Ag	Me															
`lbu a5,1(a0)`	Fe	De	De	Ag	Me	Wb													
`addi a0,a0,1`																			
`addi a1,a1,1`																			
`bne a5,zero,.L3`																			
`L3:`																			
`sb a5,0(a1)`																			
`lbu a5,1(a0)`																			
`addi a0,a0,1`																			
`addi a1,a1,1`																			
`bne a5,zero,.L3`																			

Stall until Ag is available

L3:																			
`sb a5,0(a1)`	Fe	De	Ag	Me															
`lbu a5,1(a0)`	Fe	De	De	Ag	Me	Wb													
`addi a0,a0,1`		Fe	De	Ex	Wb														
`addi a1,a1,1`		Fe	Fe	De	Ex	Wb													
`bne a5,zero,.L3`		Stall until De is available																	
`L3:`																			
`sb a5,0(a1)`																			
`lbu a5,1(a0)`																			
`addi a0,a0,1`																			
`addi a1,a1,1`																			
`bne a5,zero,.L3`																			

L3:																			
sb a5,0(a1)	Fe	De	Ag	Me															
lbu a5,1(a0)	Fe	De	De	Ag	Me	Wb													
addi a0,a0,1		Fe	De	Ex	Wb														
addi a1,a1,1		Fe	Fe	De	Ex	Wb													
bne a5,zero,.L3			Fe	De	Ex	Ex													
L3 :																			
sb a5,0(a1)																			
lbu a5,1(a0)																			
addi a0,a0,1																			
addi a1,a1,1																			
bne a5,zero,.L3																			

Stall until a5 is forwarded from lbu

L3:																			
`sb a5,0(a1)`	Fe	De	Ag	Me															
`lbu a5,1(a0)`	Fe	De	De	Ag	Me	Wb													
`addi a0,a0,1`		Fe	De	Ex	Wb														
`addi a1,a1,1`		Fe	Fe	De	Ex	Wb													
`bne a5,zero,.L3`			Fe	De	Ex	Ex													
`L3:`																			
`sb a5,0(a1)`					Fe	De	Ag	Me											
`lbu a5,1(a0)`					Fe	De	De	Ag	Me	Wb									
`addi a0,a0,1`						Fe	De	Ex	Wb										
`addi a1,a1,1`						Fe	Fe	De	Ex	Wb									
`bne a5,zero,.L3`							Fe	De	Ex	Ex									

backward taken branch is predicted during De

L3:																			
sb a5,0(a1)	Fe	De	Ag	Me															
lbu a5,1(a0)	Fe	>>	De	Ag	Me	Wb													
addi a0,a0,1		Fe	De	Ex	Wb														
addi a1,a1,1		>>	Fe	De	Ex	Wb													
bne a5,zero,.L3			Fe	De	>>	Ex													
L3:																			
sb a5,0(a1)					Fe	De	Ag	Me											
lbu a5,1(a0)					Fe	>>	De	Ag	Me	Wb									
addi a0,a0,1						Fe	De	Ex	Wb										
addi a1,a1,1						>>	Fe	De	Ex	Wb									
bne a5,zero,.L3							Fe	De	>>	Ex									

4-way out-of-order with realistic (3-stage pipelined) cache access

- (Fa, Fb, Fc, De, Fu, Al, Rn): 4
- (Qu, Ca): 64
- ALU-op: 1 clock cycle latency
- Load-op: 4 clock cycles latency

Stages for the instructions (In-order/out-of-order):

- Load:

Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ag	ma	mb	mc	wb	Ca	Cb

- Store:

Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ag	ma	mb	mc	Ca	Cb

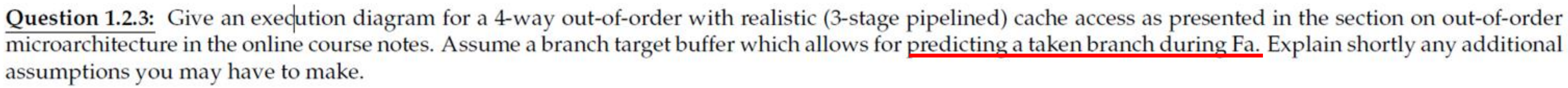
- Other:

Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb

- branch:

Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	Ca	Cb

4-way out-of-order with realistic (3-stage pipelined) cache access from exam 2023/2024



Question 1.2.3: Give an execution diagram for a 4-way out-of-order with realistic (3-stage pipelined) cache access as presented in the section on out-of-order microarchitecture in the online course notes. Assume a branch target buffer which allows for predicting a taken branch during Fa. Explain shortly any additional assumptions you may have to make.

L3:																				
addi a0,a0,1	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
add a5,a4,a0																				
lbu a5,0(a5)																				
bne a5,zero,.L3																				
L3:																				
addi a0,a0,1																				
add a5,a4,a0																				
lbu a5,0(a5)																				
bne a5,zero,.L3																				

L3:																				
`addi a0,a0,1`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
`add a5,a4,a0`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	Ca	Cb					
`lbu a5,0(a5)`																				
`bne a5,zero,.L3`																				
`L3:`																				
`addi a0,a0,1`																				
`add a5,a4,a0`																				
`lbu a5,0(a5)`																				
`bne a5,zero,.L3`																				

stall 1 cycle (ALU-op = 1 cycle latency),
because of add depends on a0 from addi

L3:																				
`addi a0,a0,1`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
`add a5,a4,a0`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	Ca	Cb					
`lbu a5,0(a5)`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb	
`bne a5,zero,.L3`																				
L3:																				
`addi a0,a0,1`																				
`add a5,a4,a0`																				
`lbu a5,0(a5)`																				
`bne a5,zero,.L3`																				

stall 1 cycle, because lbu needs a5 from add

L3:																				
`addi a0,a0,1`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
`add a5,a4,a0`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	Ca	Cb					
`lbu a5,0(a5)`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb	
`bne a5,zero,.L3`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	--	--	--	--	pk	rd	ex	Ca	Cb	
`L3:`																				
`addi a0,a0,1`																				
`add a5,a4,a0`																				
`lbu a5,0(a5)`																				
`bne a5,zero,.L3`																				

Stall 4 cycles (Load-op = 4 cycles latency), until a5 is ready from lbu

L3:																				
`addi a0,a0,1`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
`add a5,a4,a0`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	Ca	Cb					
`lbu a5,0(a5)`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb	
`bne a5,zero,.L3`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	--	--	--	--	pk	rd	ex	Ca	Cb	
`L3:`	backward taken branch is predicted during Fa																			
`addi a0,a0,1`		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	--	--	--	--	Ca	Cb	
`add a5,a4,a0`	Only 4 Fa stages are available on the same clock cycle												Stall until Ca and Cb is in-order							
`lbu a5,0(a5)`																				
`bne a5,zero,.L3`																				

L3:																				
`addi a0,a0,1`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
`add a5,a4,a0`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	Ca	Cb					
`lbu a5,0(a5)`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb	
`bne a5,zero,.L3`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	--	--	--	--	pk	rd	ex	Ca	Cb	
`L3:`																				
`addi a0,a0,1`		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	--	--	--	--	Ca	Cb	
`add a5,a4,a0`		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	--	--	--	Ca	Cb	
`lbu a5,0(a5)`																				
`bne a5,zero,.L3`																				

Stall 1 cycle until a0 is ready

Stall until Ca and Cb are in-order

L3:																				
`addi a0,a0,1`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
`add a5,a4,a0`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	Ca	Cb					
`lbu a5,0(a5)`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb	
`bne a5,zero,.L3`	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	--	--	--	--	pk	rd	ex	Ca	Cb	
`L3:`																				
`addi a0,a0,1`		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	--	--	--	--	Ca	Cb	
`add a5,a4,a0`		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	--	--	--	Ca	Cb	
`lbu a5,0(a5)`		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb
`bne a5,zero,.L3`						Stall 1 cycle until a5 is ready														

L3:																				
addi a0,a0,1	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	Ca	Cb						
add a5,a4,a0	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	Ca	Cb					
lbu a5,0(a5)	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb	
bne a5,zero,.L3	Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	--	--	--	--	pk	rd	ex	Ca	Cb	
L3:					Stall 4 cycles (Load-op) until a5 is ready															
addi a0,a0,1		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	pk	rd	ex	wb	--	--	--	--	Ca	Cb	
add a5,a4,a0		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	pk	rd	ex	wb	--	--	--	Ca	Cb	
lbu a5,0(a5)		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	pk	rd	ag	ma	mb	mc	wb	Ca	Cb
bne a5,zero,.L3		Fa	Fb	Fc	De	Fu	Al	Rn	Qu	--	--	--	--	--	--	pk	rd	ex	Ca	Cb