Lecture 8 slides

CE/CZ 3001:
Advanced Computer Architecture

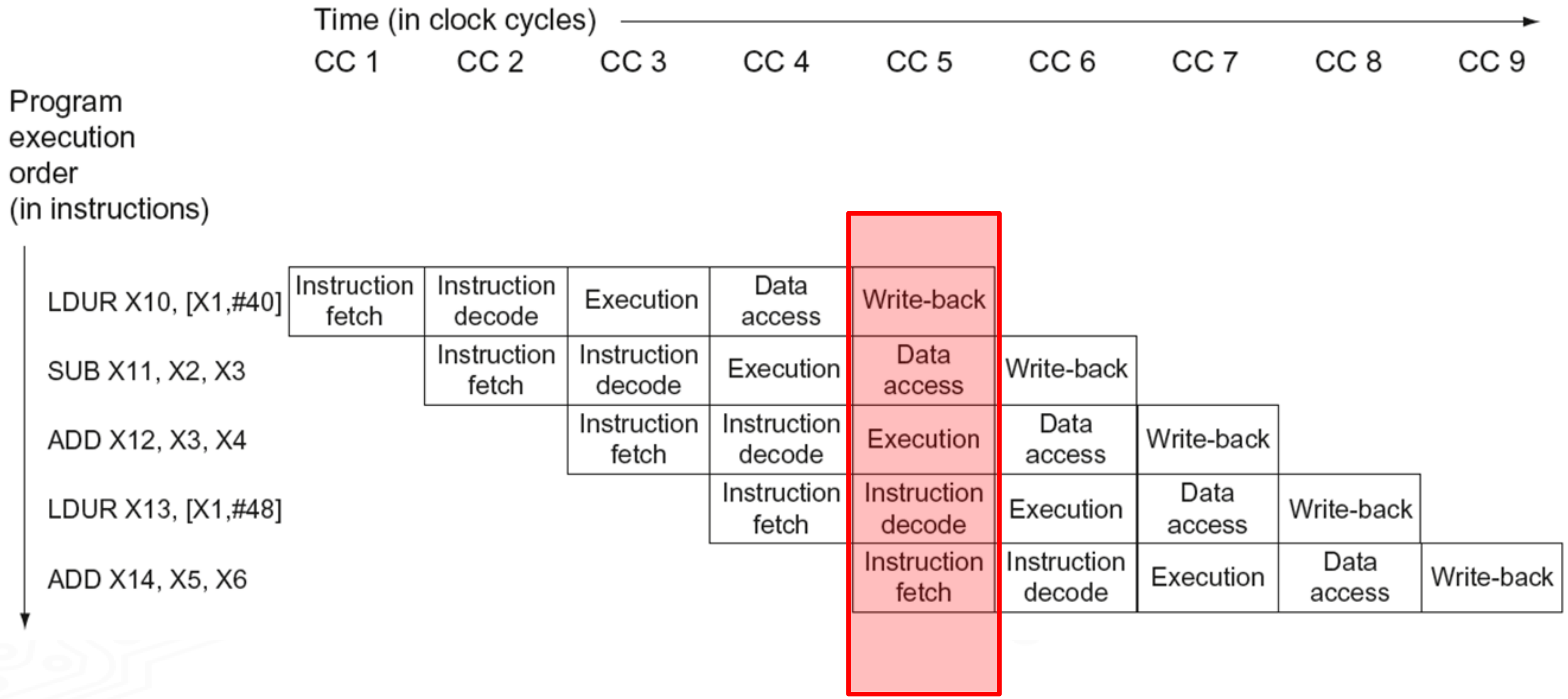
(Module 4: Instruction Level Parallelism(ILP))

Dr Smitha K. G.
School of Computer Science
And Engineering

Summary of video

- Data-dependence
- How to handle data dependencies?
  - Detect and Wait
  - Data forwarding through register
  - Detect and forward
- In order and out of order execution
- Instruction reordering and renaming
- Loop unrolling

CPI of a pipeline without stalls



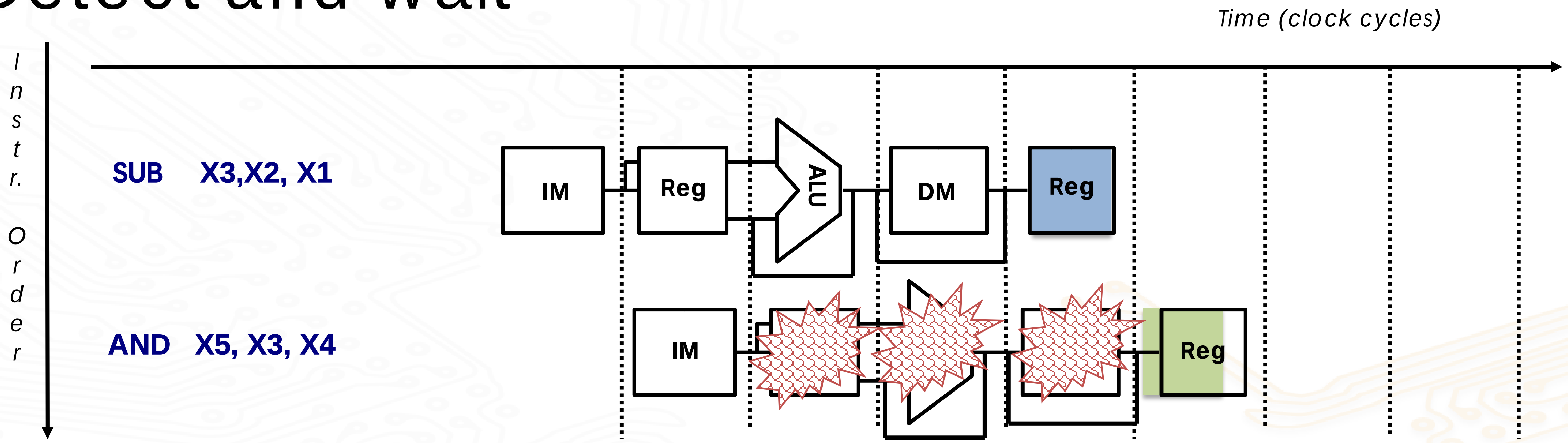

```
CPI= No of clock cycles/instruction
```

Steady state CPI = (No of instructions + no of stalls) / No of instructions

How to handle data dependencies

- Anti and output dependences are easier to handle
- True (Flow or RAW)dependences are more difficult to handle as they constitute true dependence on a value
  - Detect and wait until value is available in register file
    - Stall the program. <u>(HARDWARE)</u>
    - Compiler can also plug in the NOP instructions in between. <u>(SOFTWARE)</u>
  - Detect and forward / bypass data to dependent instruction

Detect and wait



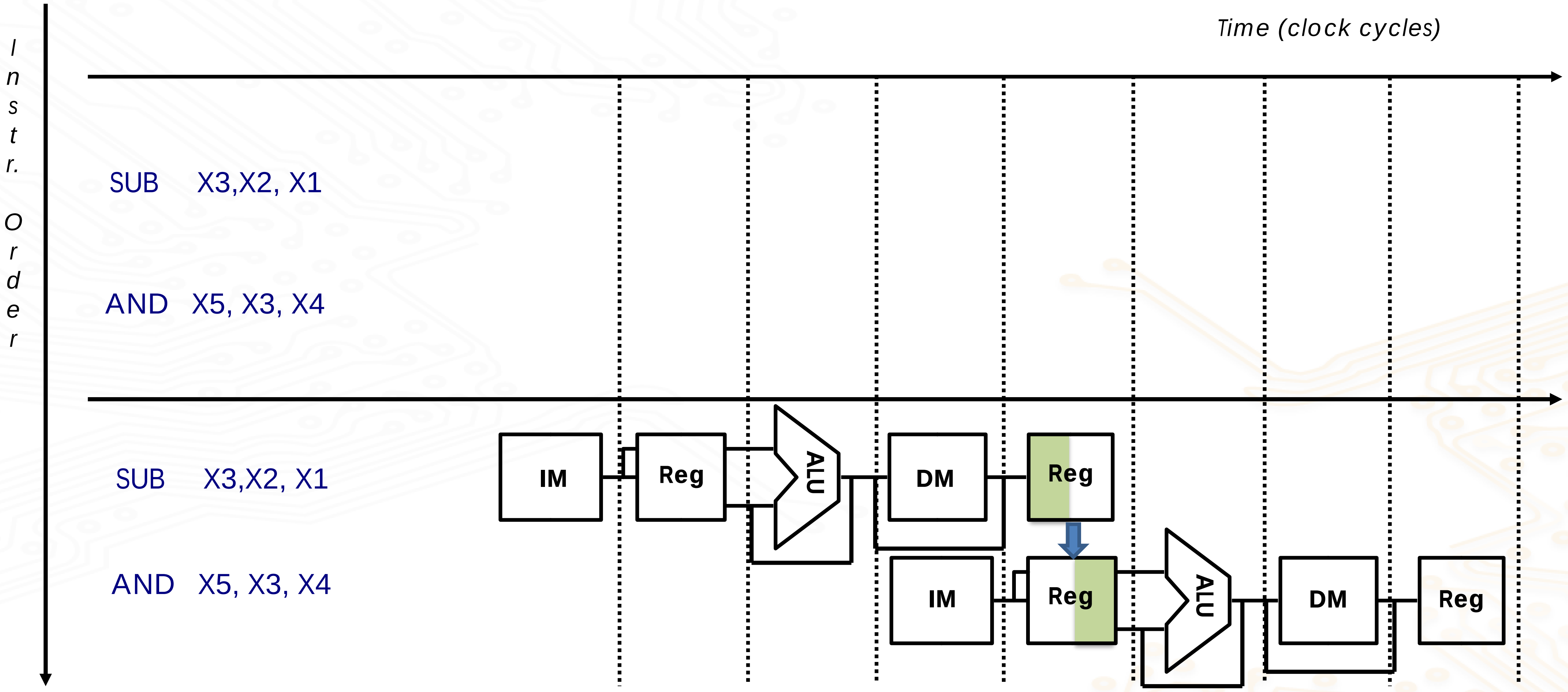
Software inserting NOPs

Instr.	1	2	3	4	5	6	7	8	9
I1									
I2									

Hardware stall

Instr.	1	2	3	4	5	6	7	8	9
I1									
I2									

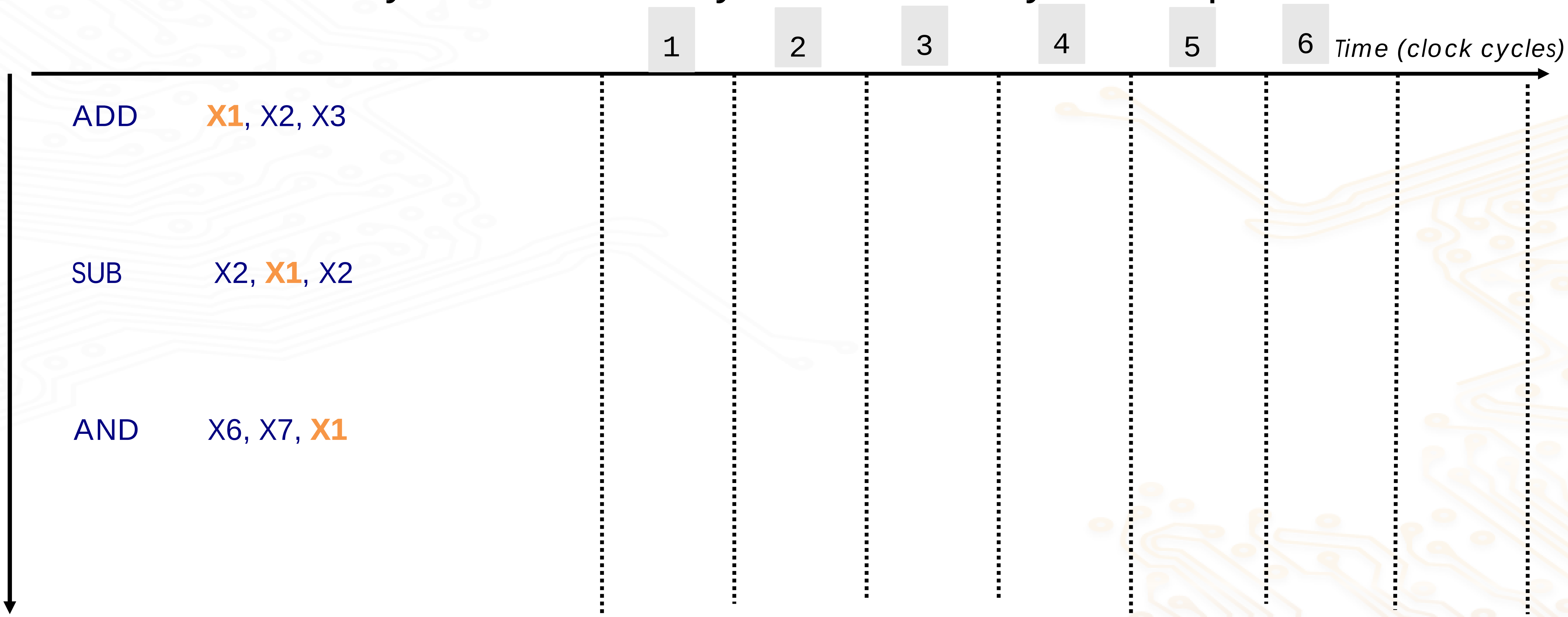
Data Forwarding – through register



Solution: write and read in the same cycle
Most processors have this as it is easy to implement

Detect and Forward / bypass

- Data forwarding.
- From pipeline stage registers to function units
- Forward data only in the clock cycle when they are required



Detect and Forward / bypass

From pipeline stage registers to function units

1 2 3 4 5 6 7

Time (clock cycles)

LDUR **X1**, [X2, #0]

SUB X4, **X1**, X5

AND X6, **X1**, X7

ORR X8, **X1**, X9

XOR X4, **X1**, X5

Forward data only in the clock cycle when they are required

Data forwarding – example 2

I1: ADD X1, X2, X3

I2: LDUR X2, [X1, #0]

I3: AND X6, X7, X1

Without forwarding
(writeback and decode can happen simultaneously)

Clocks	1	2	3	4	5	6	7	8	9	10
I1	IF	ID	EX	M	WB					
I2		IF	S	S	ID	EX	M	WB		
I3					IF	ID	EX	M	WB	

With forwarding

Clock cycle	1	2	3	4	5	6	7
I1							
I2							
I3							

Steady state CPI = (No of instructions + no of stalls) / No of instructions
Steady state CPI (no forwarding) =
Steady state CPI (forwarding) =

Performance improvement using Dynamic Scheduling

Assume full data forwarding

Out-of-order processors:

After instruction decode

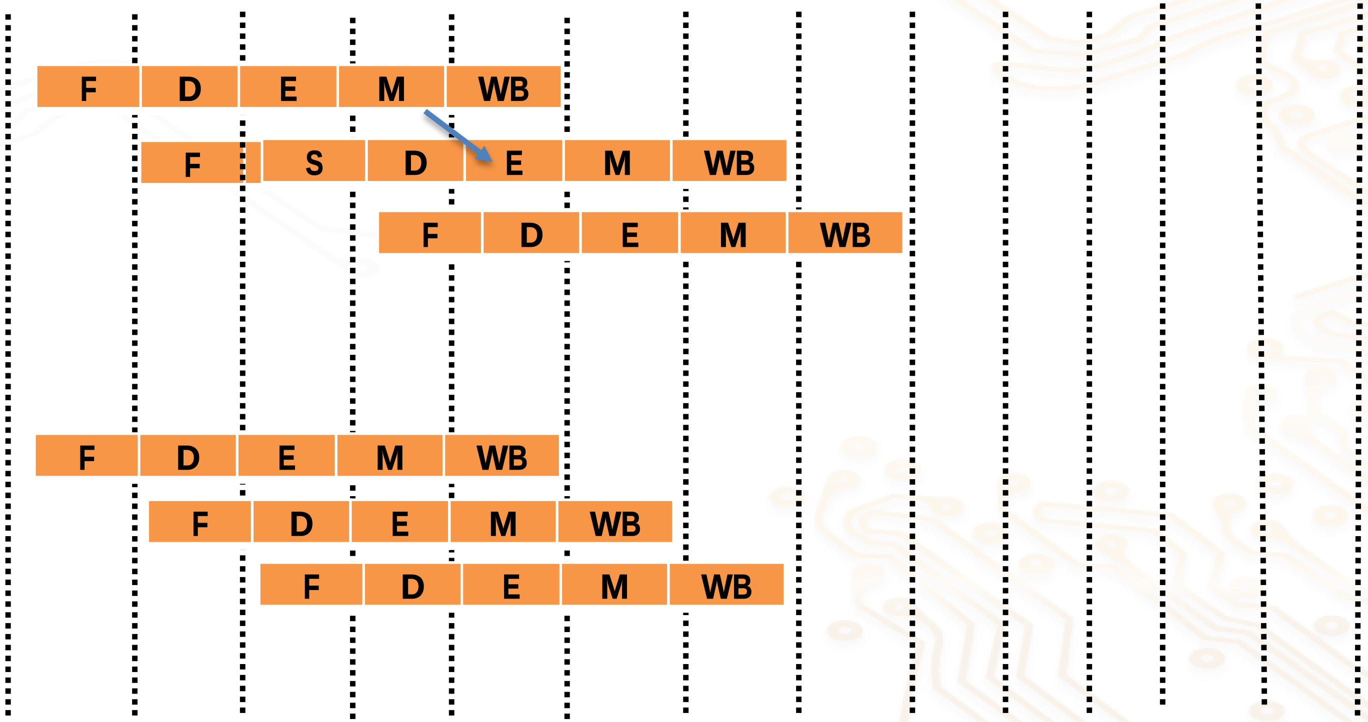
- don't wait for previous instructions to execute if this instruction does not depend on them, i.e., independent ready instructions can execute before earlier instructions that are stalled

in-order processors

LDUR **X1**, [X4, #100]
ADD X2, **X1**, X4
SUB X5, X6, X7

out-of-order processors

LDUR **X1**, [X4, #100]
SUB X5, X6, X7
ADD X2, **X1**, X4(no stalls)



Register Renaming

WAR	ADD	X4, X2, X1;	X4 $\leftarrow$ X2 + X1
	ANDI	X1, X0, #2;	X1 $\leftarrow$ X0 & 2
		Rename X1 to X3	
	ADD	X4, X2, X1;	X4 $\leftarrow$ X2 + X1
	AND	X3, X0, #2;	X3 $\leftarrow$ X0 & 2
WAW	ADD	X0, X2, X1;	X0 $\leftarrow$ X2 + X1
	SUB	X0, X3, X5;	X0 $\leftarrow$ X3 – X5
		Rename X0 to X4	
	ADD	X0, X2, X1;	X0 $\leftarrow$ X2 + X1
	SUB	X4, X3, X5;	X4 $\leftarrow$ X3 – X5

More register resources will be needed

Loop unrolling

- loop unrolling leads to multiple replications of the loop body
  - unrolling creates longer code sequences
  - goal is to execute iterations in parallel

- Example

```
for (i=0: i < 16; i++) {
c[i] = a[i]+ b[i];
}
```

Unroll once

```
for (i=0: i < 8; i++) {
c[2 * i] = a[2 * i] + b[2 * i];
c[2 * i+1] = a[2 * i+1] + b[2 * i+1];
}
```

- greater demand for registers
  - higher register pressure : more concurrency demands for more resources

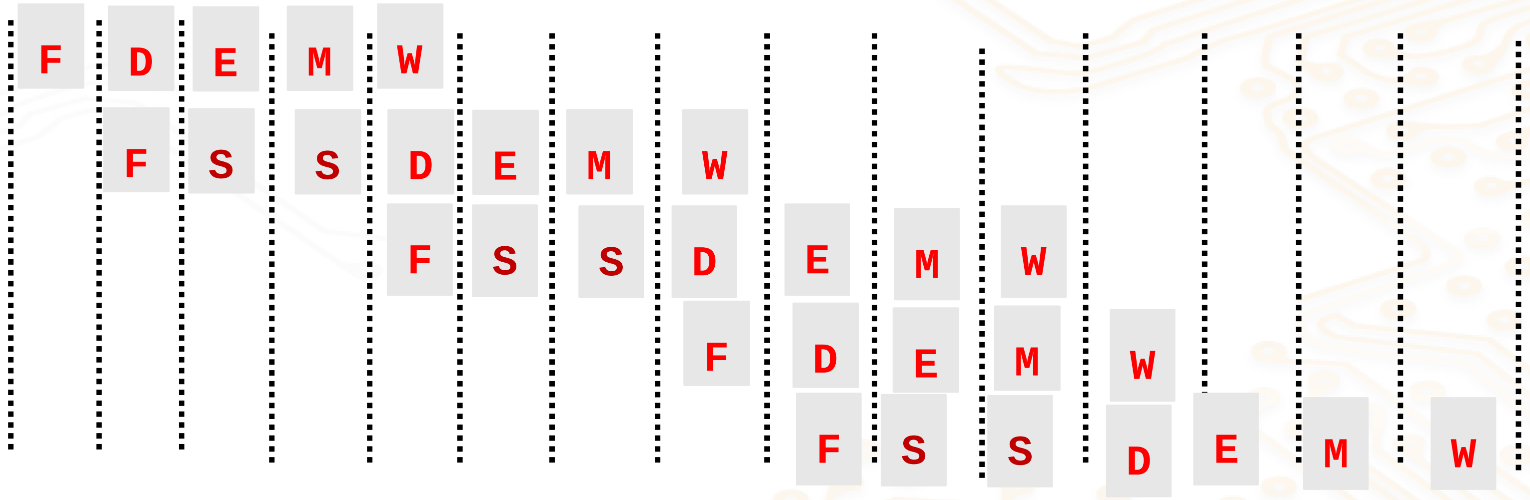
How can loop unrolling help us to reduce the stalls and to improve CPI?

Loop unrolling example

Loop:	LDUR	X0, [X1, #0];	load to X0 from mem[0+X1]
	ADD	X4, X0, X2;	add [X0]+[X2]
	STUR	X4, [X1, #0];	store X4 to mem[0+X1]
	SUBI	X1, X1, #8;	decrement pointer 8
	CBNZ	X1,Loop;	branch X1!=zero

If data forwarding is not allowed, write back and decode of two instructions can happen simultaneously

1. Loop:	**LDUR**	**X0, [X1, #0]**	
2.	**ADD**	**X4, X0, X2**	
3.	**STUR**	**X4, [X1, #0]**	
4.	**SUBI**	**X1, X1, #8;**	
5.	**CBNZ**	**X1,Loop;**	



CPI = (No of instructions + no of stall) / No. of instruction =

Loop unrolling example

```
Loop:   LDUR    X0, [X1, #0]
        ADD     X4, X0, X2        2 stall
        STUR    X4, [X1, #0]      2 stall      ;drop SUBI & CBNZ
        LDUR    X6, [X1, #-8]
        ADD     X8, X6, X2        2 stall
        STUR    X8, [X1, #-8]     2 stall      ;drop SUBI & BNEZ
        LDUR    X10, [X1,#-16]
        ADD     X12, X10, X2      2 stall
        STUR    X12, [X1, #-16]   2 stall      ;drop SUBI & BNEZ
        LDUR    X14, [X1, #-24]
        ADD     X16, X14, X2      2 stall
        STUR    X16, [X1, #-24]   2 stall
        SUBI    X1, X1, #32                    ;alter to 4*8
        CBNZ    X1,LOOP           2 stall
```

Rewrite loop to minimize stalls?

STRAIGHT FORWARD UNROLLING,
CPI=

Branch is not handled here

Loop unrolling with reordering

1 Loop:	LDUR	**X0**, [X1, #0]	
2	ADD	**X4**, **X0**, X2	2 stall
3	STUR	**X4**, [X1, #0]	2 stall
4	LDUR	**X6**, [X1, #-8]	
5	ADD	**X8, X6**, X2	2 stall
6	STUR	**X8**, [X1, #-8]	2 stall
7	LDUR	**X10**, [X1,#-16]	
8	ADD	**X12, X10**, X2	2 stall
9	STUR	**X12**, [X1, #-16]	2 stall
10	LDUR	**X14**, [X1,#-24]	
11	ADD	**X16, X14**, X2	2 stall
12	STUR	**X16, [X1, #**-24]	2 stall
13	SUBI	**X1**, X1, #32	
14	CBNZ	**X1**,LOOP	2 stall

1 Loop:	**LDUR**	**X0, [X1, #0]**	
2	**LDUR**	**X6, [X1, #-8]**	
3	**LDUR**	**X10, [X1,#-16]**	
4	**LDUR**	**X14, [X1,#-24]**	
5	**ADD**	**X4, X0, X2**	
6	**ADD**	**X8, X6, X2**	
7	**ADD**	**X12, X10, X2**	
8	**ADD**	**X16, X14, X2**	
9	**STUR**	**X4, [X1, 0]**	
10	**STUR**	**X8, [X1, #-8]**	
11	**STUR**	**X12, [X1, #-16]**	
12	**STUR**	**X16, [X1, #-24]**	
13	**SUBI**	**X1, X1, #32**	**2 stall here**
14	**CBNZ**	**X1,LOOP**	

No dataforwarding, WB and DEC happens simultaneously

How data hazards can be eliminated?

Summary

- **Detect and wait (stalling unnecessarily)**
- **Data forwarding**
- **Reordering (out of order)**
- **Renaming (to remove WAR and WAW hazard)**
- **Loop unrolling and reordering**