Lecture 9 slides

CE/CZ 3001:
Advanced Computer Architecture

(Module 4: Instruction Level Parallelism(ILP))

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

Performance improvement using Dynamic Scheduling

```
Assume full data forwarding
```

Out-of-order processors:

After instruction decode

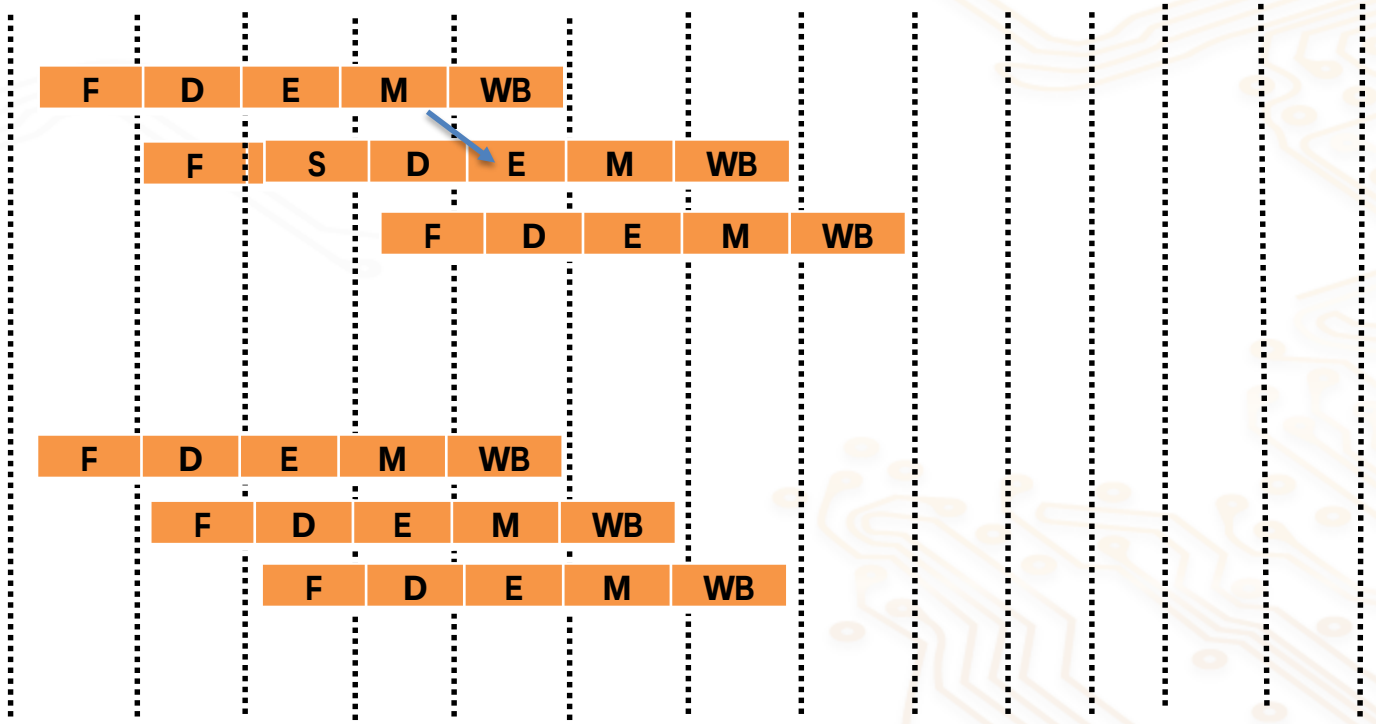
- don't wait for previous instructions to execute if this instruction does not depend on them, i.e., independent ready instructions can execute before earlier instructions that are stalled

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 ← X2 + X1
ANDI  X1, X0, #2;   X1 ← X0 & 2
```

⇩ Rename X1 to X3

```
ADD   X4, X2, X1;   X4 ← X2 + X1
AND   X3, X0, #2;   X3 ← X0 & 2
```

WAW

```
ADD   X0, X2, X1;   X0 ← X2 + X1
SUB   X0, X3, X5;   X0 ← X3 – X5
```

⇩ Rename X0 to X4

```
ADD   X0, X2, X1;   X0 ← X2 + X1
SUB   X4, X3, X5;   X4 ← 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];
}
```


```
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

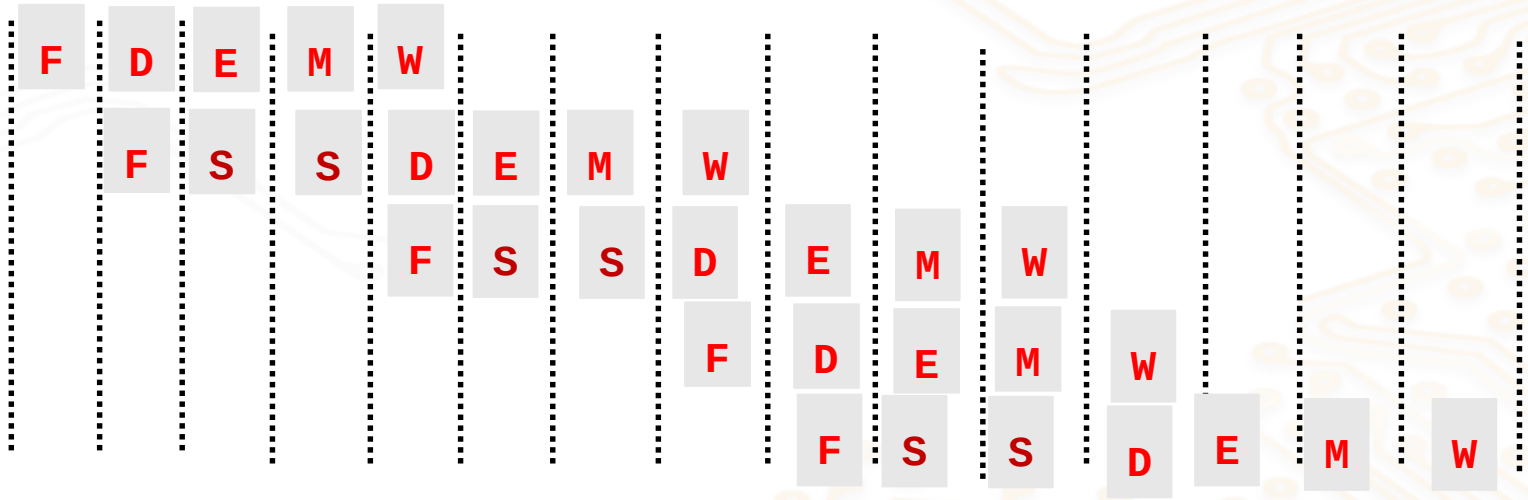
How can loop unrolling help us to reduce the stalls and to improve CPI?

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

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

Rewrite loop to minimize stalls?

STRAIGHT FORWARED UNROLLING,
CPI=

Loop unrolling with reordering

Branch is not handled here

```
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
14      CBNZ   X1,LOOP
```

2 stall here

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**

Summary of pre video

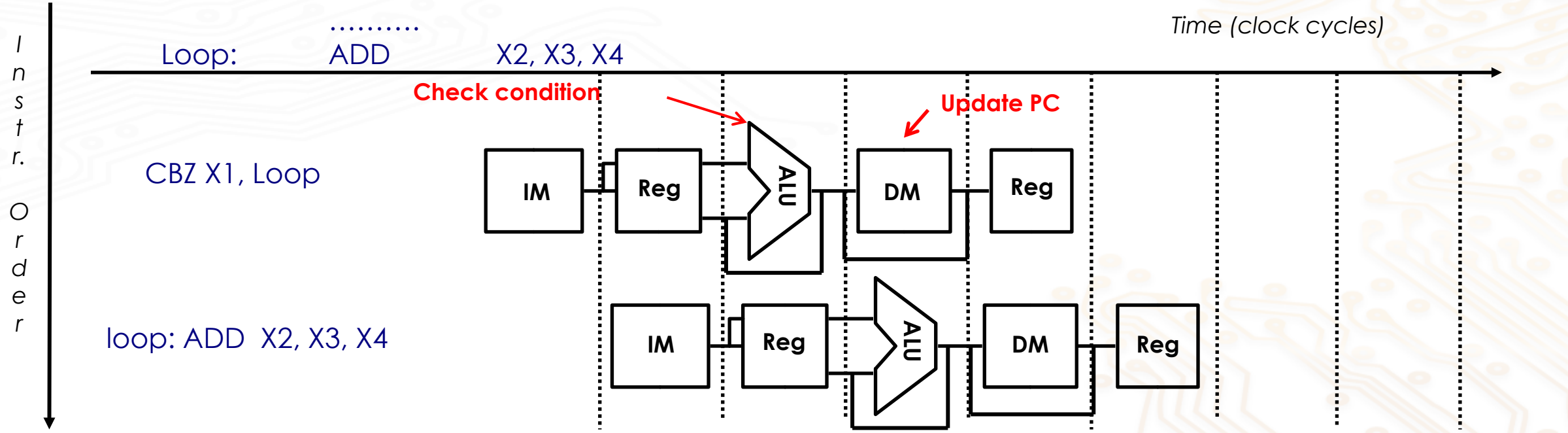
- Control Hazards
- How to tackle control hazards
- Conservative way
- Early evaluation of the content of PC (using hardware change)
- Branch prediction
  - Static branch prediction
  -

Control Hazards

- Branching instructions
  - pipeline to stall as it changes the execution sequence of instructions.

```
                CBZ     X1,Loop
                ORR     X7,X8, X9
                ……
Loop:           ADD     X2, X3, X4
```

If X1=0, then go to "loop" else go to next instruction



How to tackle control hazards

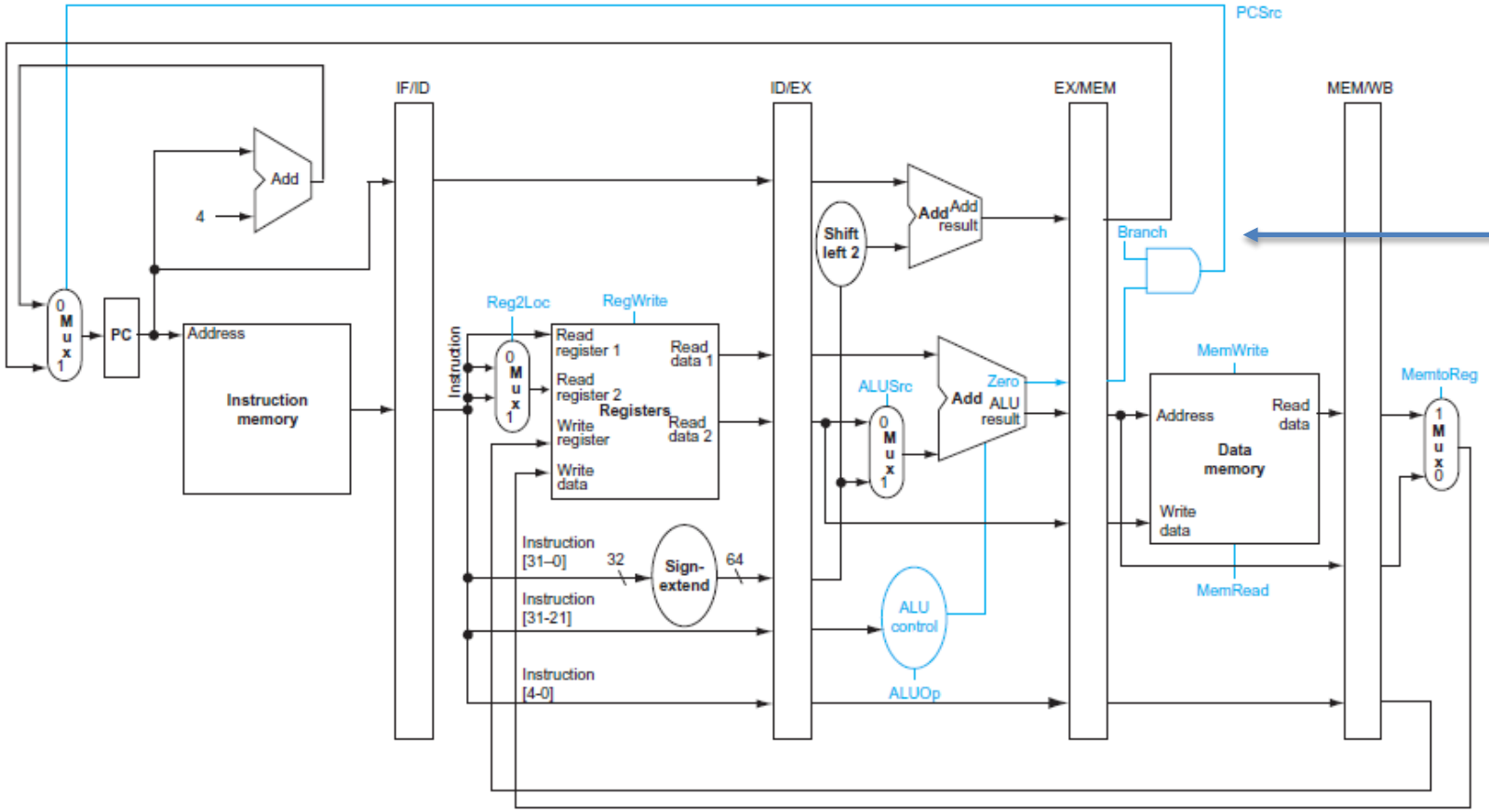
- **Conservative way**
- **Branch prediction**

Conservative method

Example

CBZ X1, loop
ADD X2, X3, X4
……
Loop: LDUR X3, [X0, #30]

Clock	1	2	3	4	5	6	7	8	9
I1	IF	ID	EX	M	WB				
I2		nop	nop	nop	nop	nop			
I3			nop	nop	nop	nop	nop		
I4				nop	nop	nop	nop	nop	
I5					IF	ID	EX	M	WB



CBZ updating PC in MEM stage Branch penalty= 3 stalls

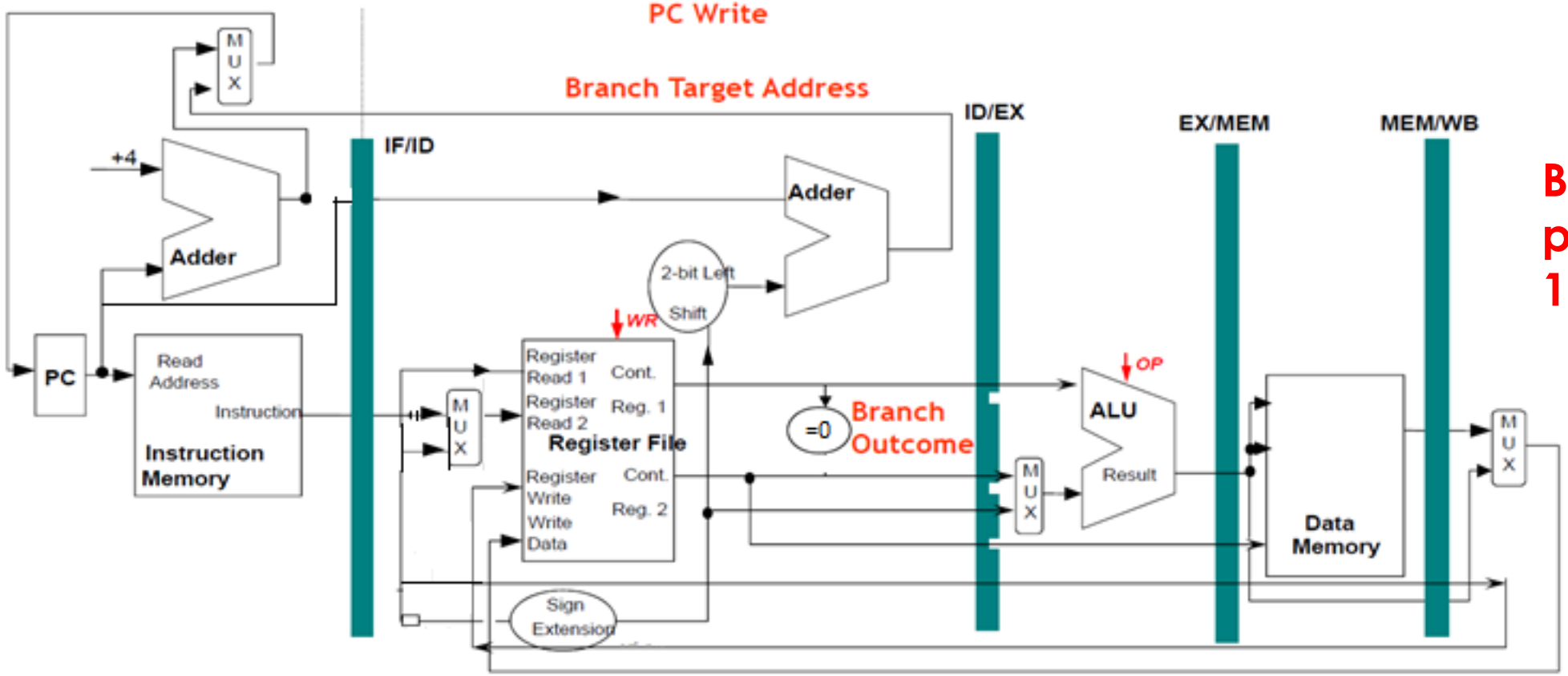
If PCSrc and Branch Target Address is updated in execute stage (hardware changes):
Then CBZ can update PC in EXE stage. Branch penalty= 2 stalls

Early Evaluation of PC for reducing branch stalls

Example

CBZ X1, loop
ADD X2, X3, X4
……
Loop: LDUR X3, [X0, #30]

Clock	1	2	3	4	5	6	7	8	9	10	11
I1	IF	ID	EX	M	WB						
I2		nop	nop	nop	nop	nop					
I5			IF	ID	EX	M	WB				



Branch penalty= 1 stalls

Another example- from exam paper

```
I1              ADDI X3, X31, 0X24
I2  Loop:       LDUR X2, [X1, #0]
I3               ADDI X2, X2, #10
I4               STUR X2, [X1, #0]
I5               ADDI X1, X1, #8
I6               SUBI X3, X3, #1
I7               CBNZ X3   Loop
```

find steady state CPI – no data forwarding (WB and dec simultaneously, Brach target address is updated in decode stage)

Branch Prediction

Branch prediction

Static branch prediction techniques:- The actions for the branch are fixed for each branch during the entire execution. (when behaviour is highly predictable).

Dynamic branch prediction techniques:- The prediction decision may change depending on the execution history (when behaviour is not predictable).

Static Prediction

a) Branch Always Not Taken – Predict-Not-Taken (Speculation)

- Execute successive instructions in sequence.
- Flush the pipeline and read correct instructions if branch actually taken

```
i1:        ADD    X3, X1, #2
i2:        CBZ    X3,  L1
i3:        ADD    X1, X0, X0
i4:        AND    X4, X1, X2
i5:   L1   SW     X4,[X5,#0]
```

(Assume branch solved in ID stage)

Instr.	1	2	3	4	5	6	7	8
I1	IF	ID	EX	M	WB			
I2		IF	ID	--	--	--		
I3			IF	F	F	F	F	
I5				IF	ID	EX	M	WB

Branch taken

1. Need to **flush** the next instruction already fetched.
2. Restart the execution by fetching the instruction at the branch target address – **One-cycle penalty.**

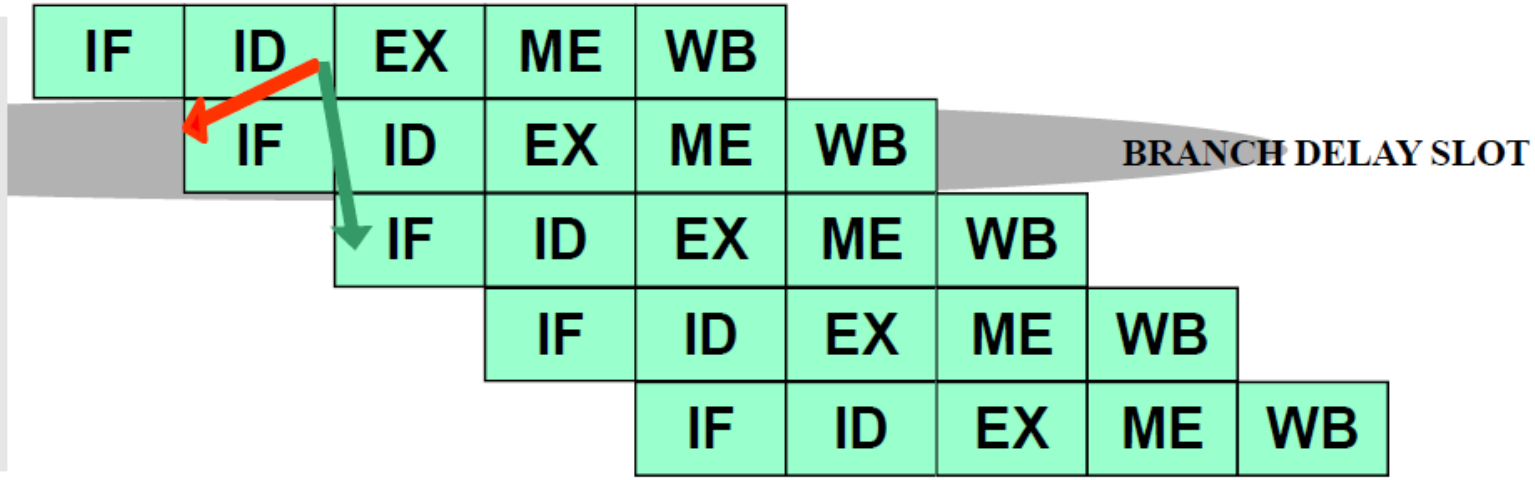
Instr.	1	2	3	4	5	6	7	8
I1	IF	ID	EX	M	WB			
I2		IF	ID	--	--	--		
I3			IF	ID	EX	M	WB	
I4				IF	ID	EX	M	WB
I5					IF	ID	EX	M

Branch not taken

Static Prediction – Delayed Branch

- The compiler statically schedules an independent instruction in the **branch delay slot**.

```
      CBZ   X1 L1
      ADD   X4,X5,X6
L1    LDUR  X3, [X0, #300]
      LDUR  X7, [X0, #400]
      LDUR  X8, [X0, #500]
```



A previous **add** instruction with no effects on the branch is scheduled in the **Branch Delay Slot.**