CSED311 Computer Architecture – Lecture 9

Pipelined CPU – Control Hazard

Eunhyeok Park

Department of Computer Science and Engineering
POSTECH

Review: Data Dependence

Data dependence

$$\begin{tikzcd}
r_3 \arrow[drr] & \leftarrow & r_1 & op & r_2 \\
r_5 & \leftarrow & r_3 & op & r_4
\end{tikzcd}$$

Read-after-Write (RAW)

Instruction must wait for all required input operands

Anti-dependence

$$\begin{tikzcd}
r_3 & \leftarrow & r_1 \arrow[dll] & op & r_2 \\
r_1 & \leftarrow & r_4 & op & r_5
\end{tikzcd}$$

Write-after-Read (WAR)

Later write must not affect a still-pending earlier read

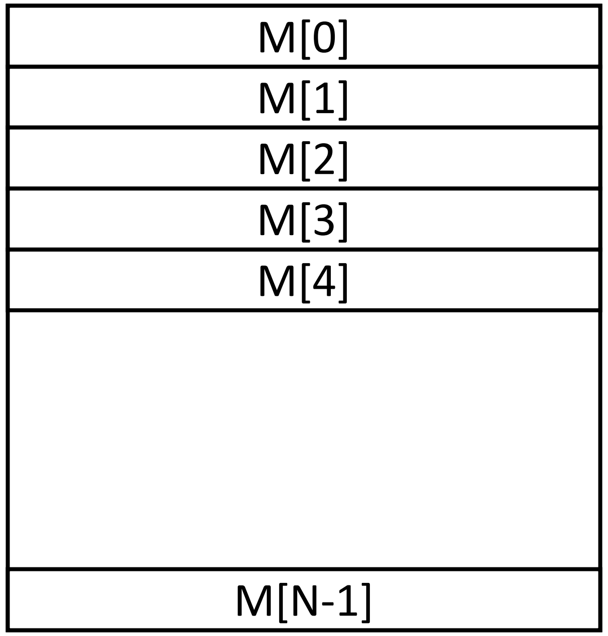
Output-dependence

$$\begin{tikzcd}
r_3 \arrow[dd, bend right] & \leftarrow & r_1 & op & r_2 \\
r_5 & \leftarrow & r_3 & op & r_4 \\
r_3 & \leftarrow & r_6 & op & r_7
\end{tikzcd}$$

Write-after-Write (WAW)

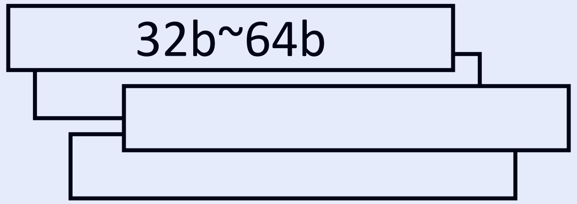
Earlier write must not affect an already-finished later write

Programmer Visible State



Memory

Array of storage locations indexed by an address



Registers

- Given special names in the ISA (as opposed to addresses)
- General vs. special purpose

Last lecture: data hazard (through registers)

Program Counter (32b~64b)

: Memory address of the current instruction

This lecture: control hazard (through PC)

Instructions (and programs) specify how to transform the values of programmer visible state

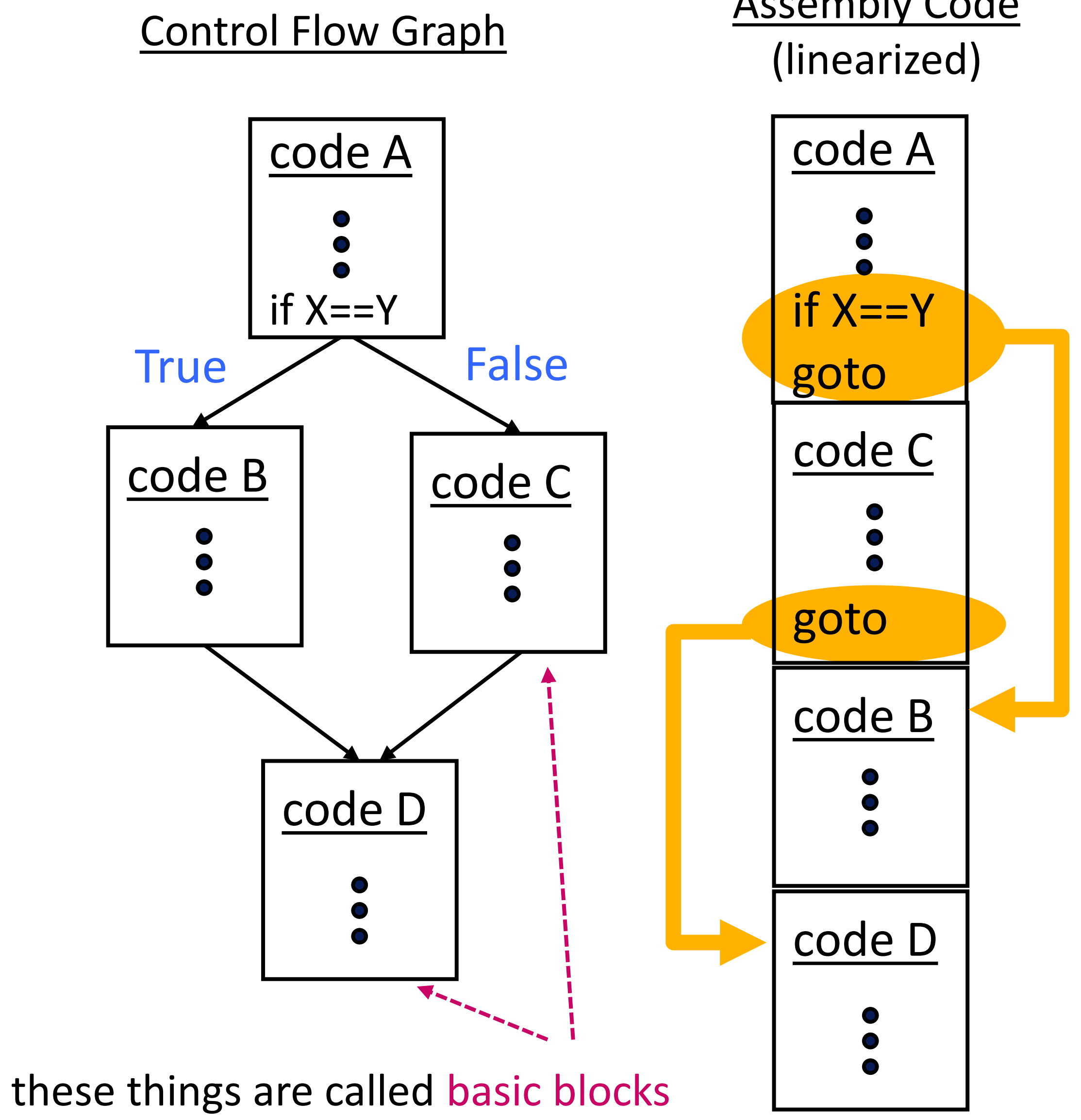
Review: Control Flow Instructions

- C-Code

```
{ code A }
if X==Y then
     { code B }
else
     { code C }
{ code D }
```

- A basic block is a sequence of instructions with
  - No embedded branches (except at end)
  - No branch targets (except at beginning)

- Control flow instructions: Bxx, JAL, and JALR



Instruction Ordering/Dependencies

Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Word	I	LW	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	LHU	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	SW	rs1,rs2,imm
Shifts	Shift Left	R	SLL	rd,rs1,rs2
	Shift Left Immediate	I	SLLI	rd,rs1,shamt
	Shift Right	R	SRL	rd,rs1,rs2
	Shift Right Immediate	I	SRLI	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt
Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm
Logical	XOR	R	XOR	rd,rs1,rs2
	XOR Immediate	I	XORI	rd,rs1,imm
	OR	R	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm

PC=PC+4

Branches	Branch =	SB	BEQ	rs1,rs2,imm
	Branch ≠	SB	BNE	rs1,rs2,imm
	Branch <	SB	BLT	rs1,rs2,imm
	Branch ≥	SB	BGE	rs1,rs2,imm
	Branch < Unsigned	SB	BLTU	rs1,rs2,imm
	Branch ≥ Unsigned	SB	BGEU	rs1,rs2,imm
Jump & Link	J&L	UJ	JAL	rd,imm
	Jump & Link Register	UJ	JALR	rd,rs1,imm

PC dependent on branch condition (PC=PC+4 or PC=PC+imm)

PC=PC+imm

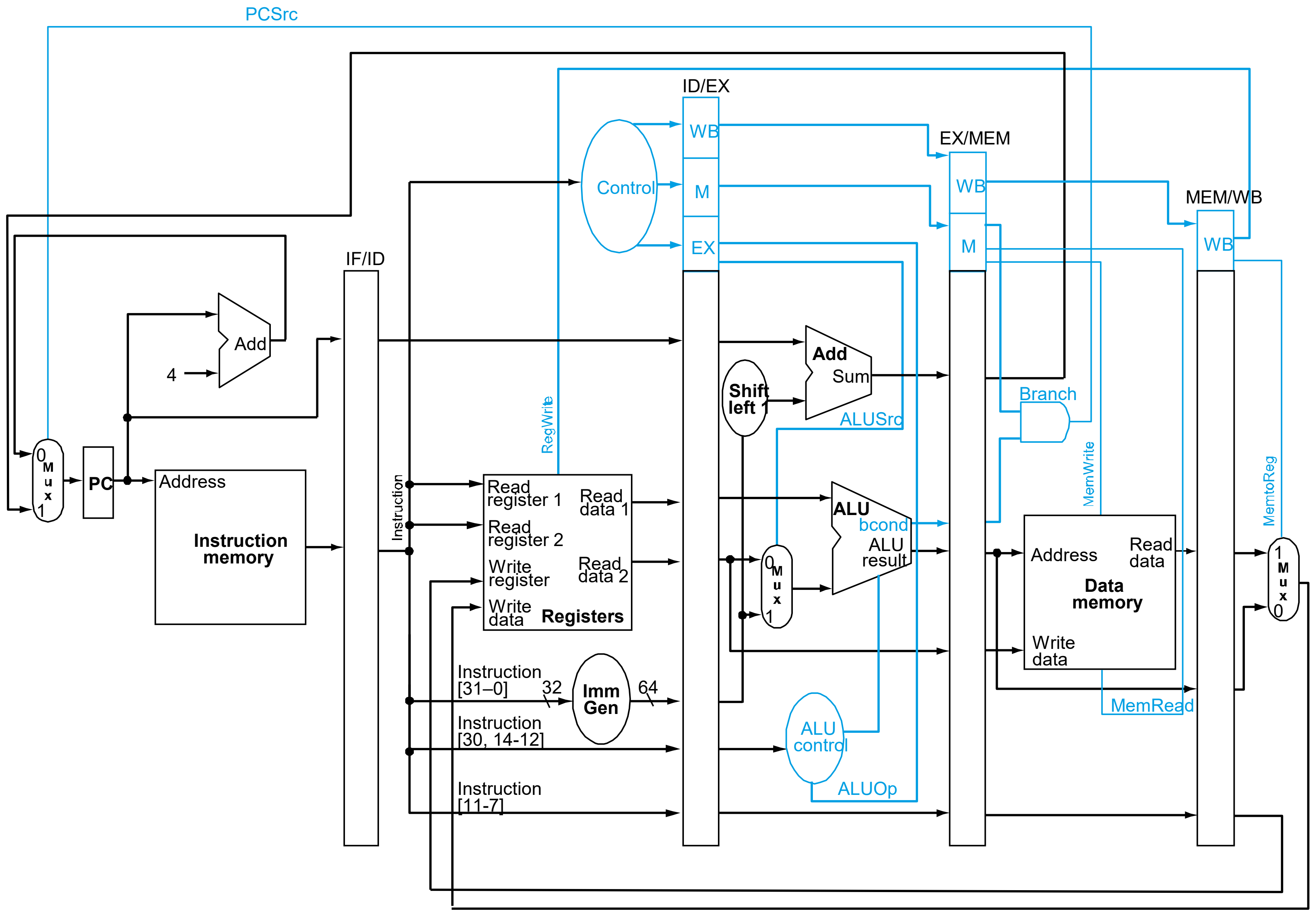
PC=GPR[rs1]+imm

- *Control* Dependence
  - All instructions are dependent by control flow
  - Every instruction uses and sets the PC

In other words, control dependence is **data dependence** on the PC

Source: RISC-V reference card

Review: Pipelined CPU



Based on figures from [P&H CO&D, COPYRIGHT 2017 Elsevier. ALL RIGHTS RESERVED.]

“PC” Hazard Analysis

Don’t confuse this with the earlier “data hazard” table which was about the operands!

	R/I-Type	LW	SW	Bxx	JAL	JALR
IF	use	use	use	use	use	use
ID	produce	produce	produce			
EX						
MEM				produce	produce	produce
WB						

- All instructions read and modify PC
- PC hazard distance is at least 1 – why?

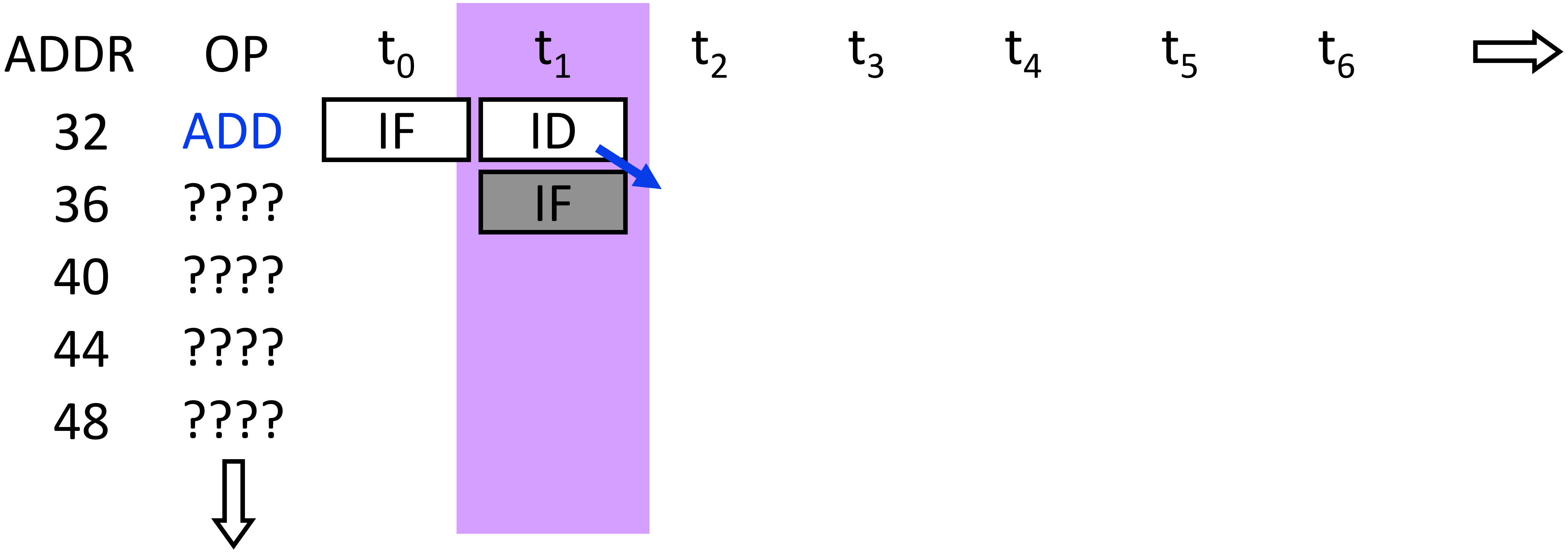
Control Hazard Resolved by Stalling

What's the next PC value?

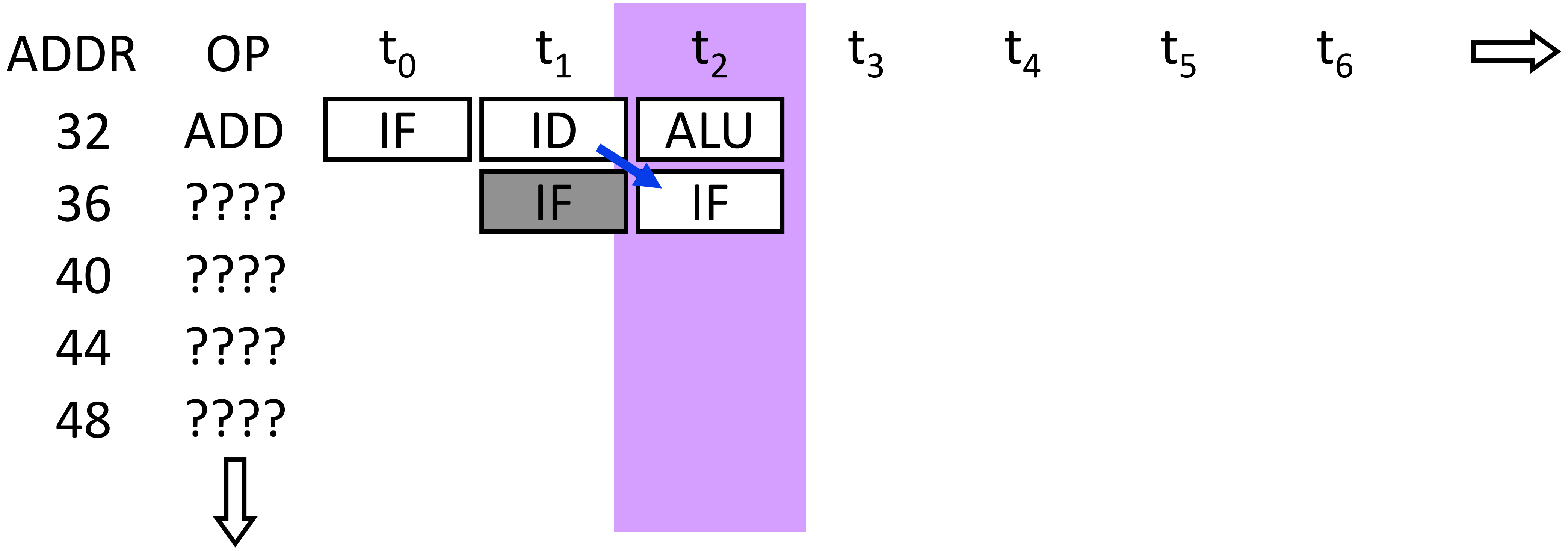
ADDR	OP	t_0	t_1	t_2	t_3	t_4	t_5	t_6
32	????	IF						
36	????							
40	????							
44	????							
48	????							

Control Hazard Resolved by Stalling

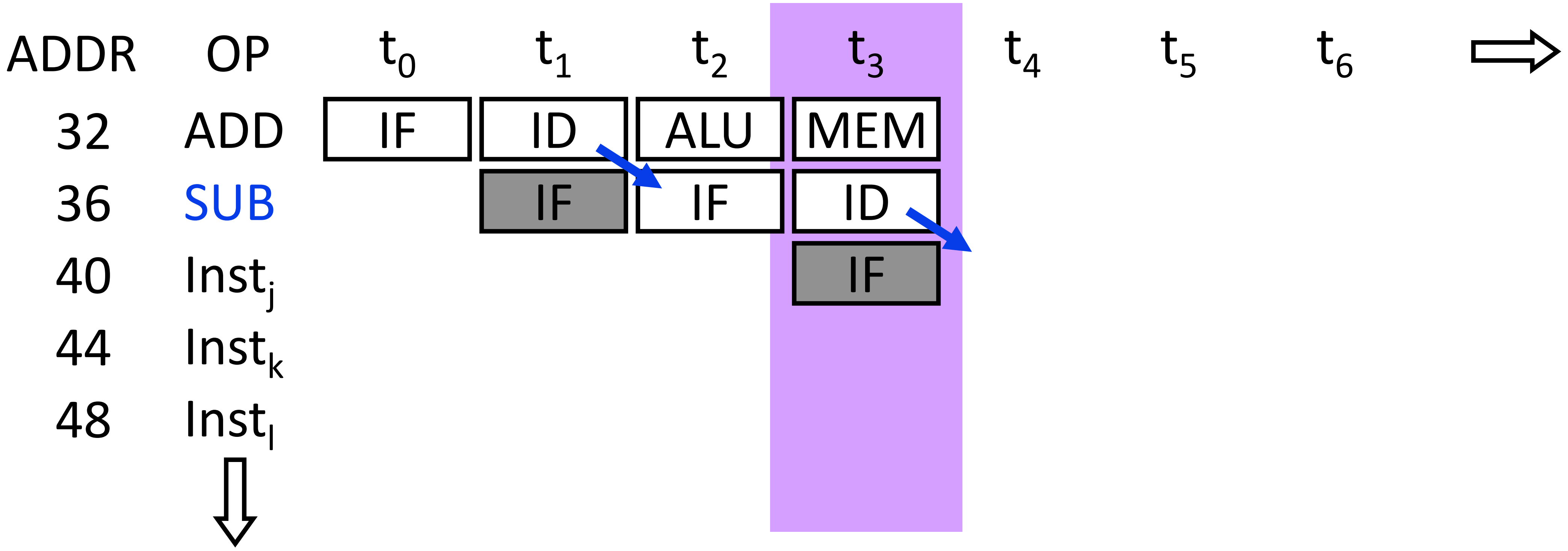
Now, what's the next PC value?



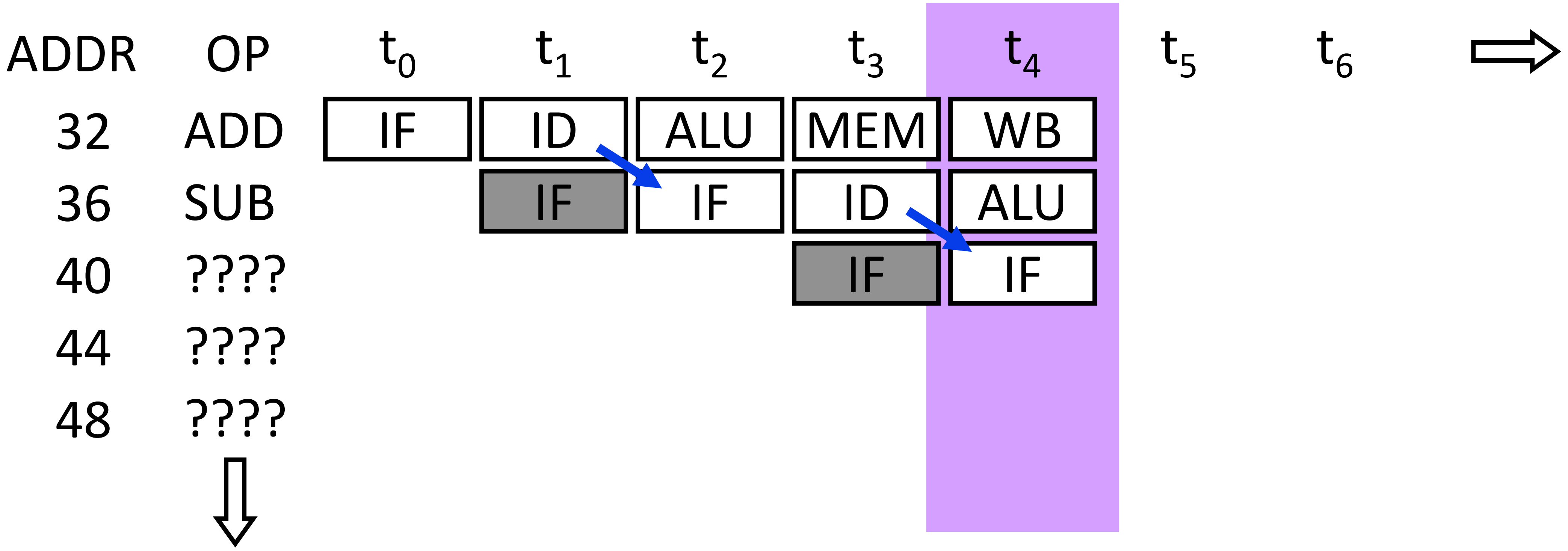
Control Hazard Resolved by Stalling



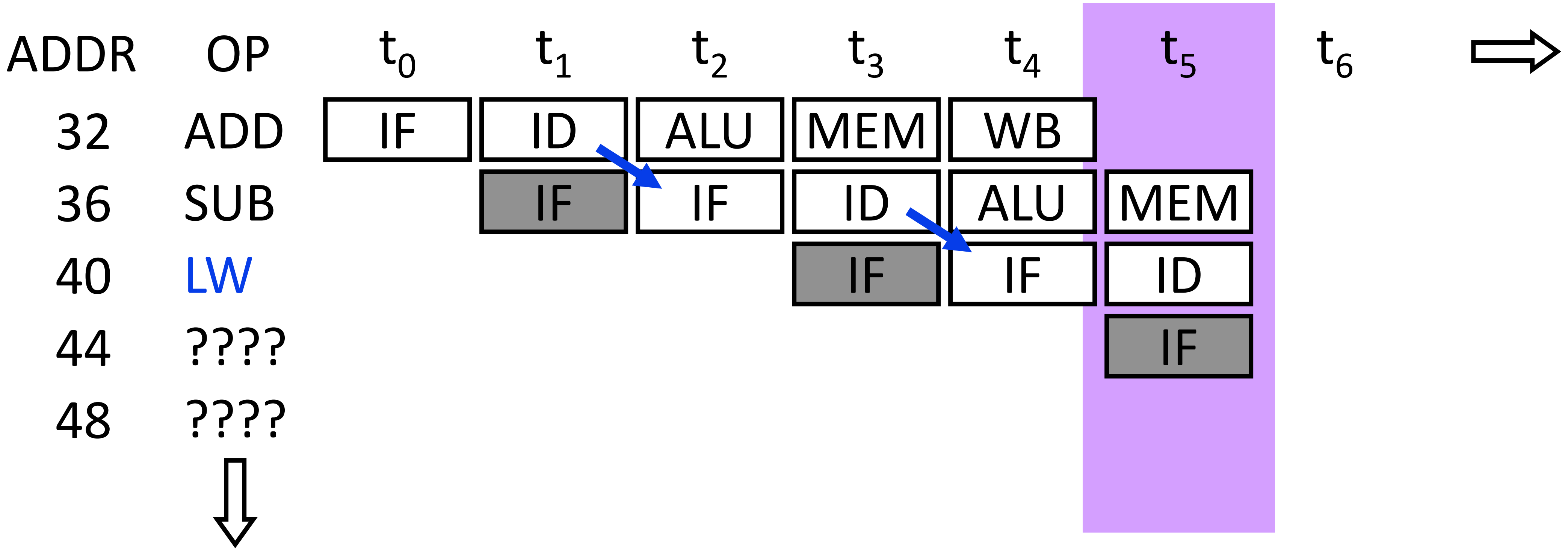
Control Hazard Resolved by Stalling



Control Hazard Resolved by Stalling

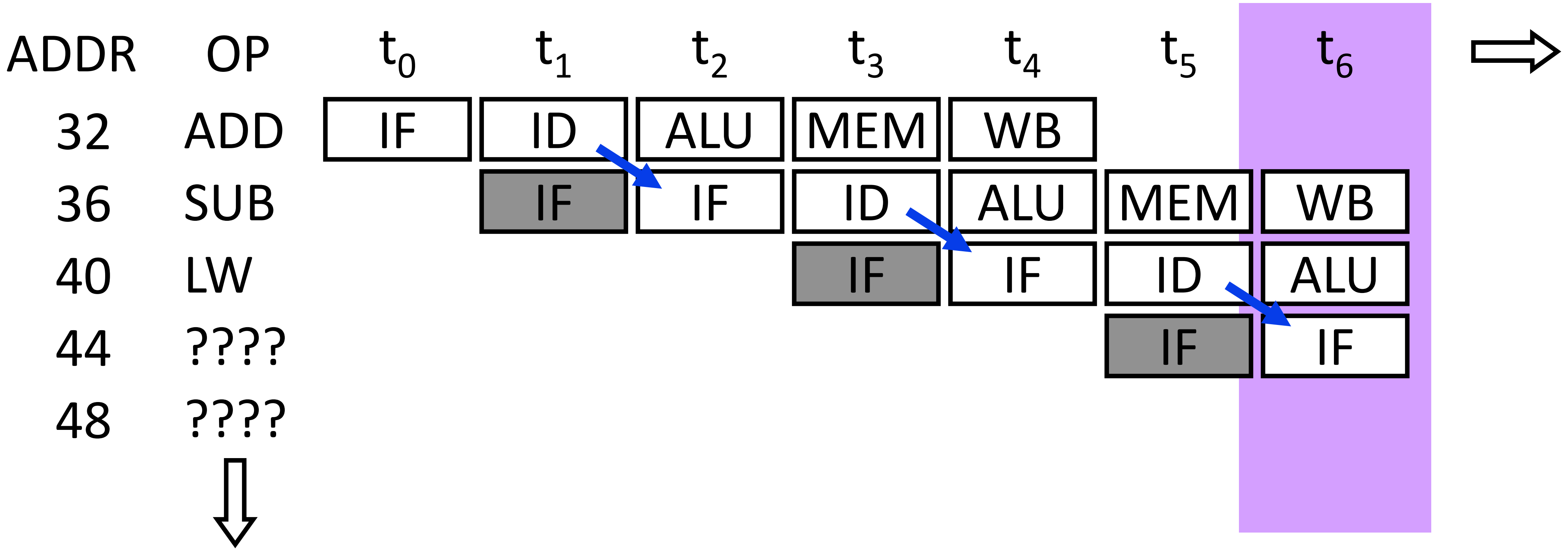


Control Hazard Resolved by Stalling



ADDR	OP	t_0	t_1	t_2	t_3	t_4	t_5	t_6
32	ADD	IF	ID	ALU	MEM	WB		
36	SUB		IF	IF	ID	ALU	MEM	
40	LW				IF	IF	ID	
44	????						IF	
48	????							

Control Hazard Resolved by Stalling

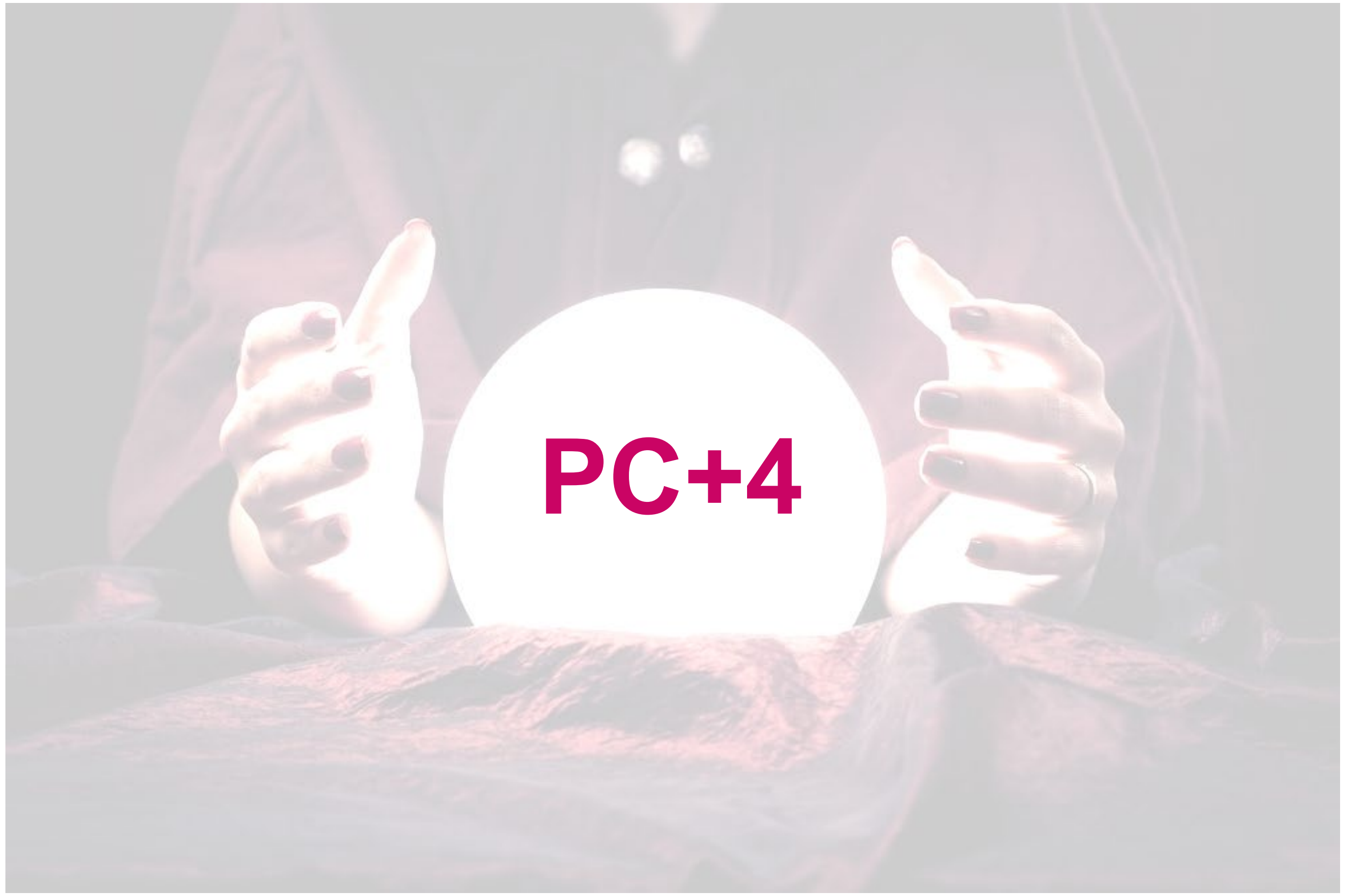


Performance reduced to 1/2 even if there's no control-flow instructions!

Is there a way to avoid such performance loss?

Observation: non-control-flow instructions are executed more often than control-flow instructions (i.e., <u>control flow instructions are only at the end of basic blocks</u>)

Predicting Next PC



Simplest Branch Prediction: Next PC = PC+4

- Typically, Only **~20%** of the instruction mix is control flow
- Out of the control flow instructions,
  - ~50% of "forward" control flow (i.e., if-then-else) is taken
  - ~90% of "backward" control flow (i.e., loop back) is taken

    Overall, typically ~70% taken and **~30%** not taken [Lee and Smith, 1984]

- Rather than waiting for true-dependence on PC to resolve,
  just **guess** nextPC = PC+4 (i.e., predict not-taken) to keep fetching every cycle

- Expect "nextPC = PC+4" ~86% of the time, but what happens for the remaining 14%?

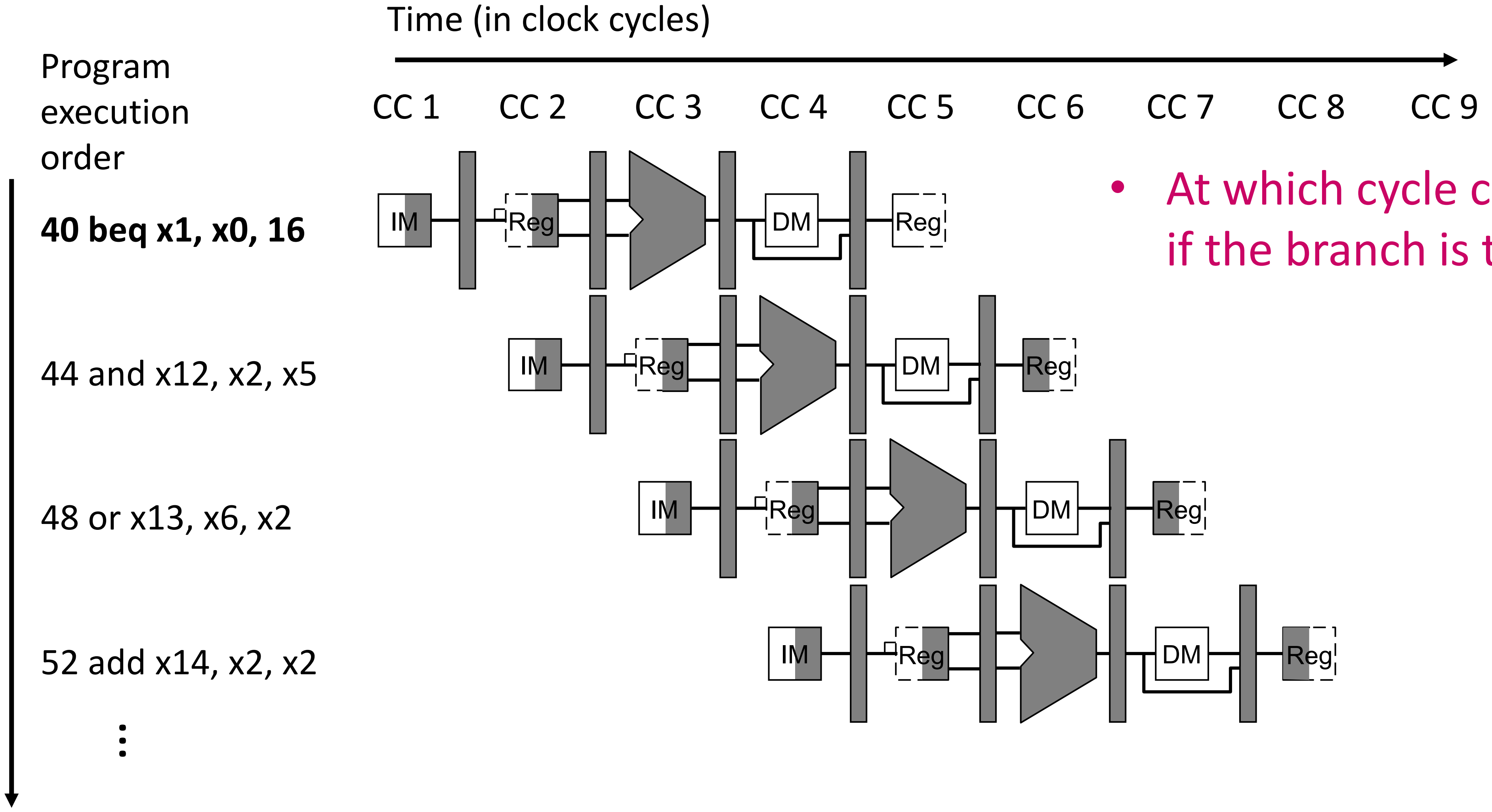
    What are the consequences of a wrong prediction?

    Performance? Correctness?

Control Speculation

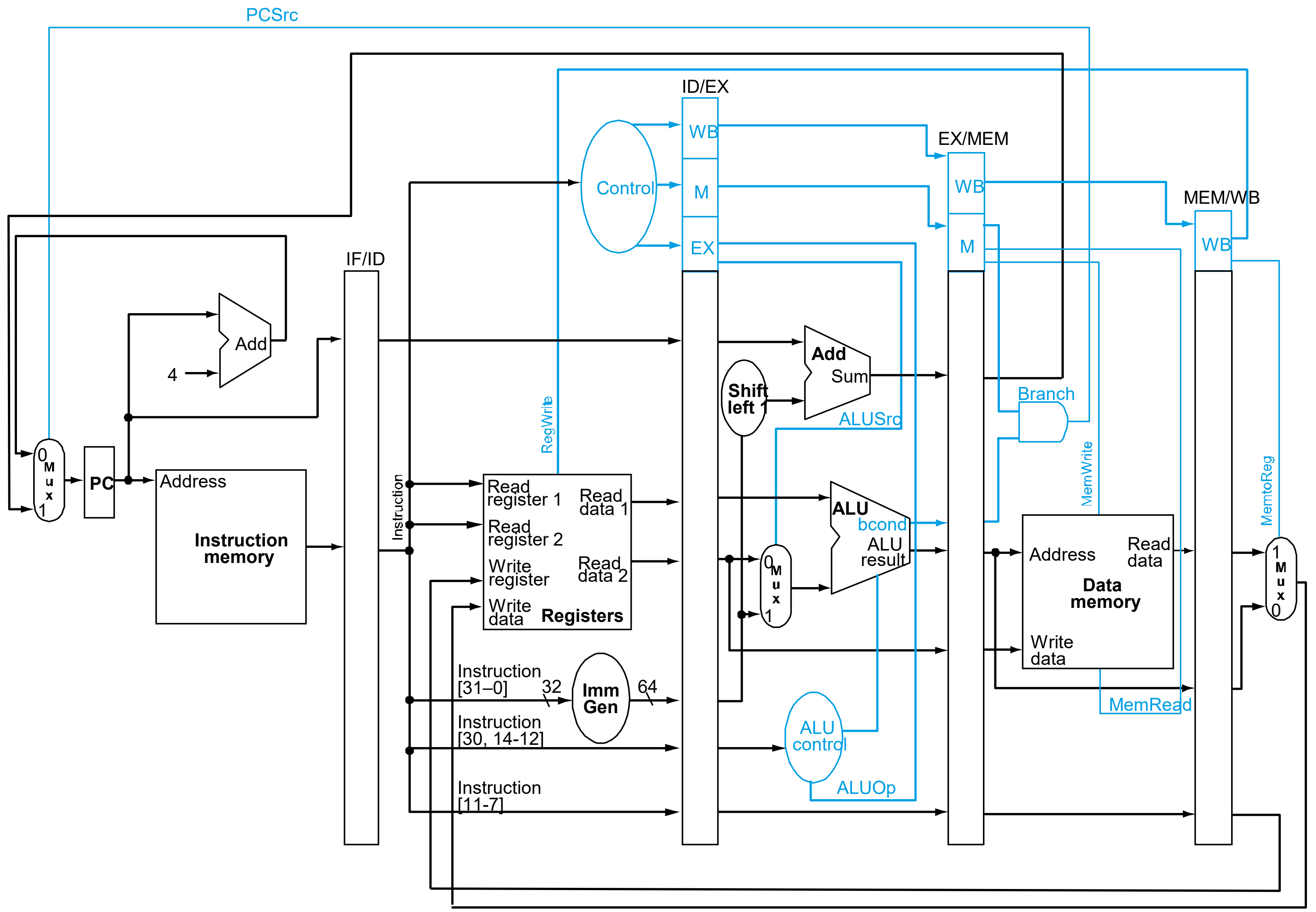
Time (in clock cycles)

Program execution order

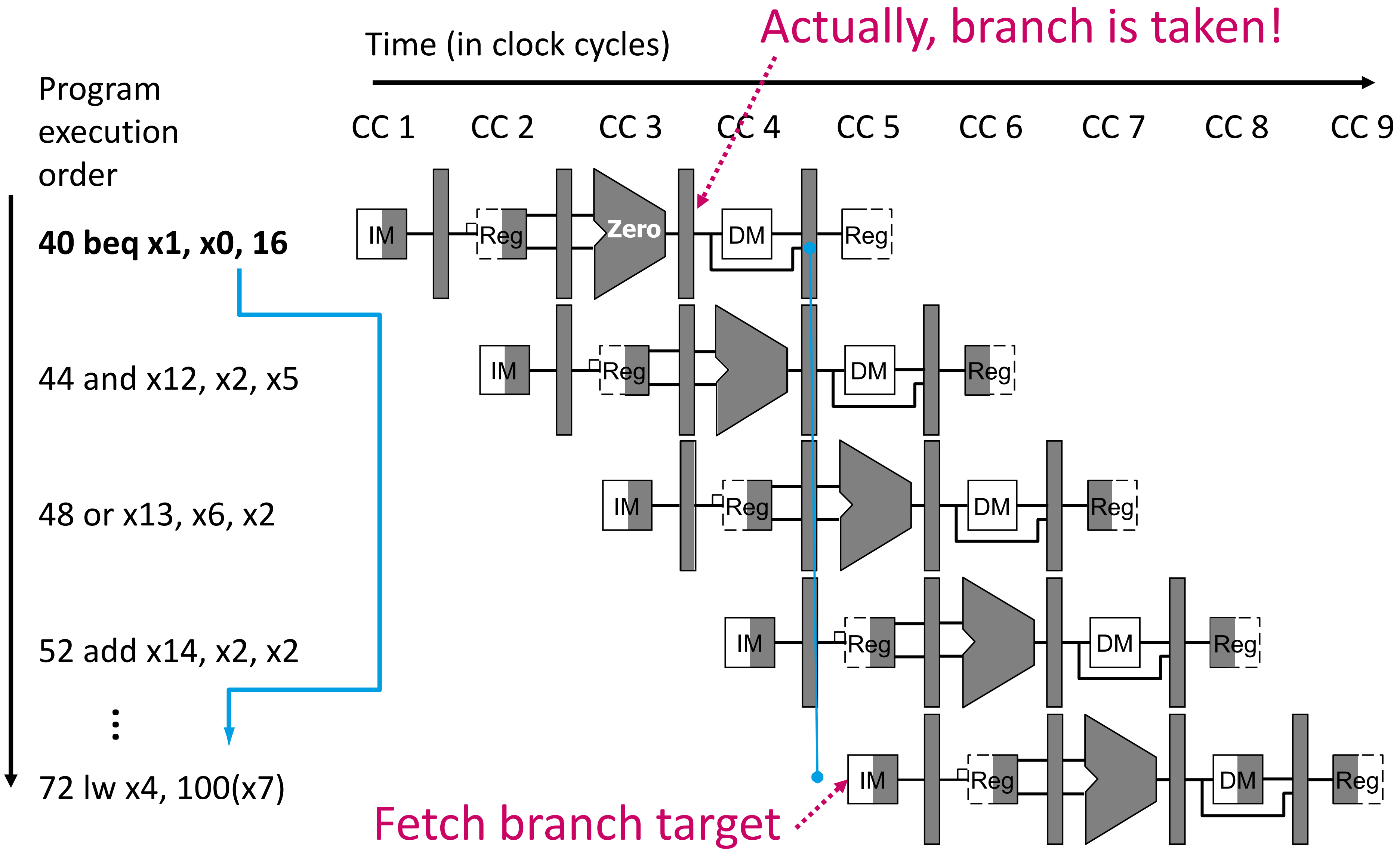


- At which cycle can we know if the branch is taken?

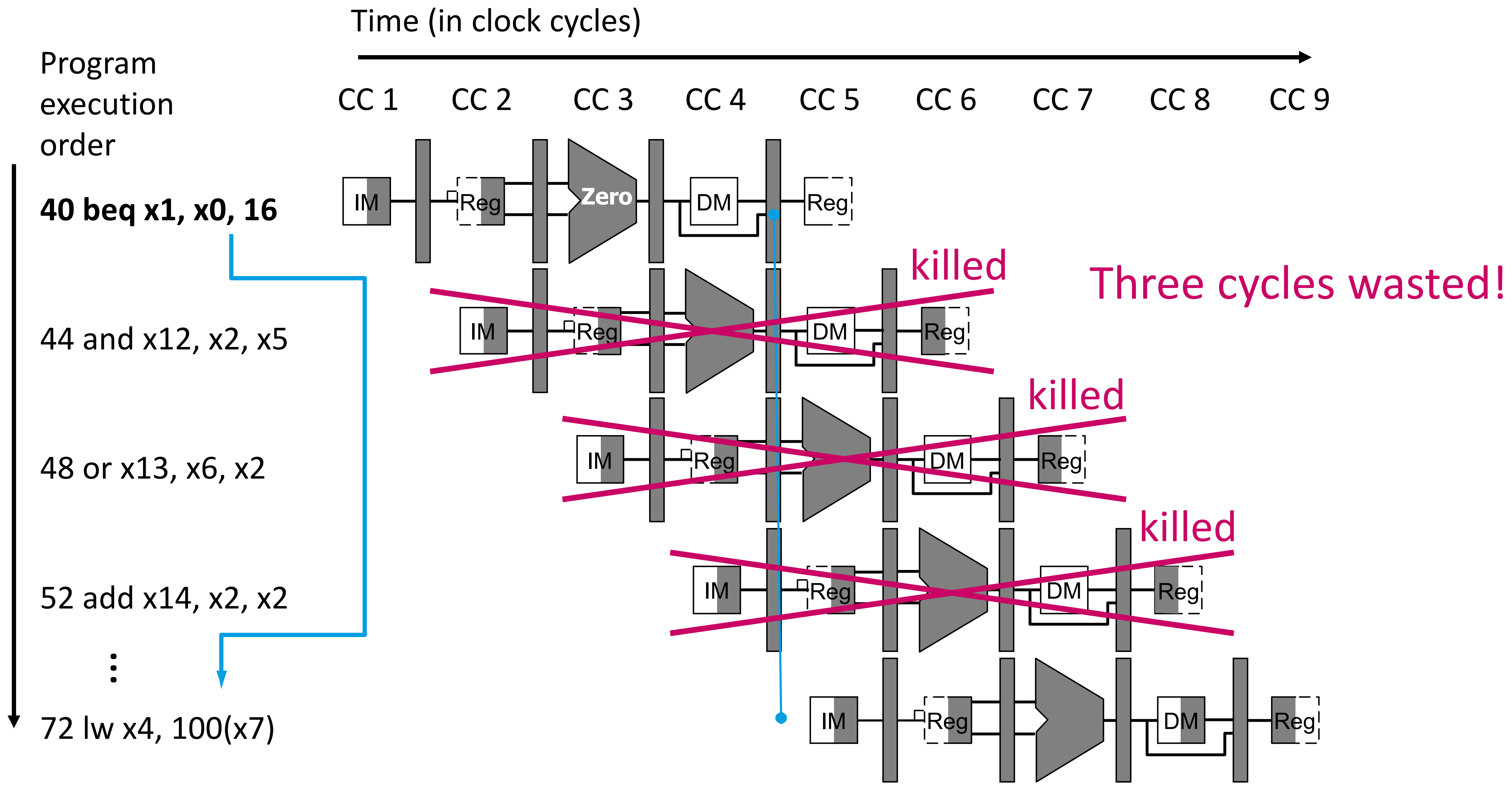
Recall the Pipelined CPU



Control Speculation

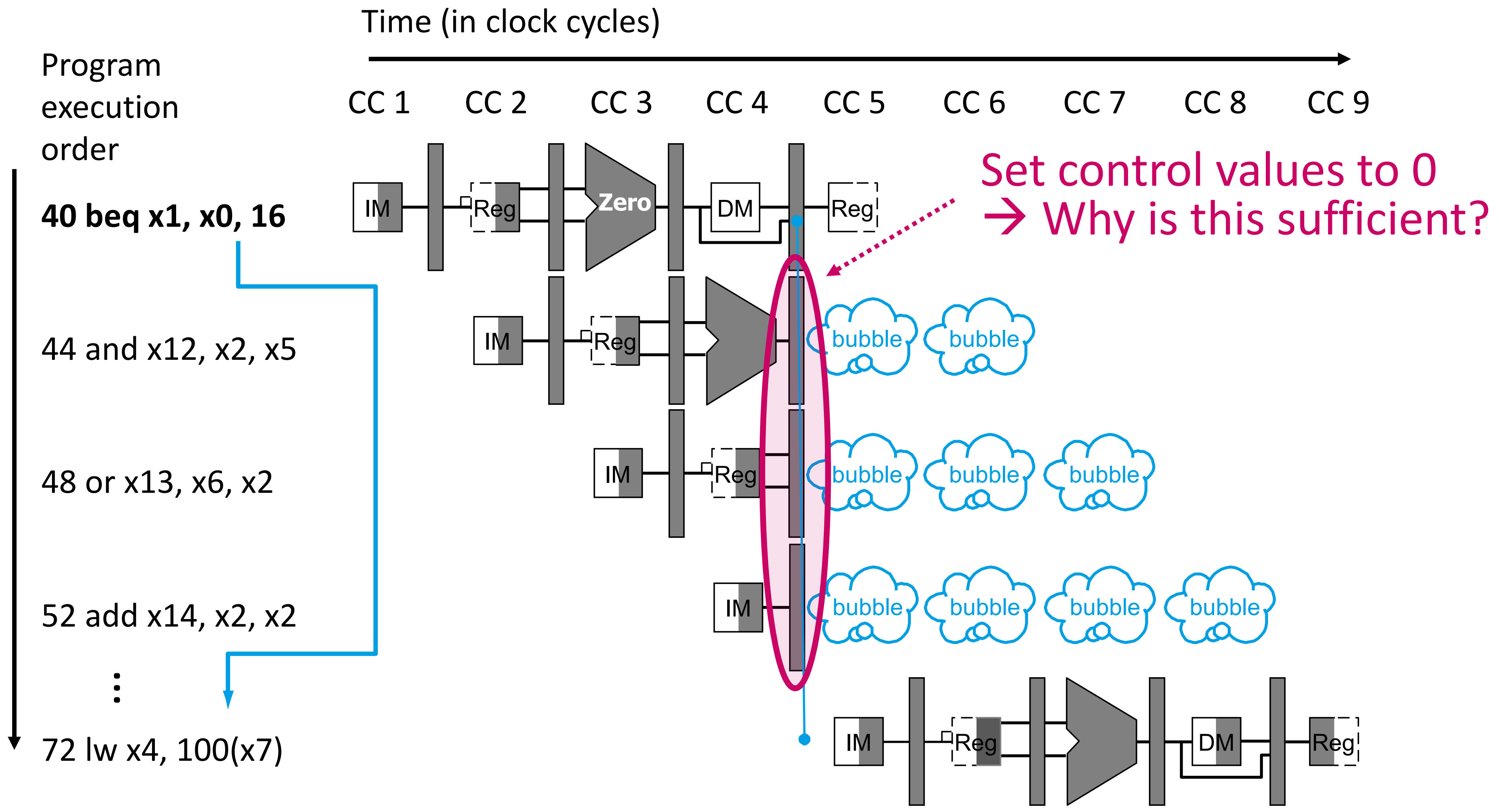


Control Speculation



Pipeline Flush on Misprediction

*Flush: to discard instructions in a pipeline, usually due to an unexpected event.



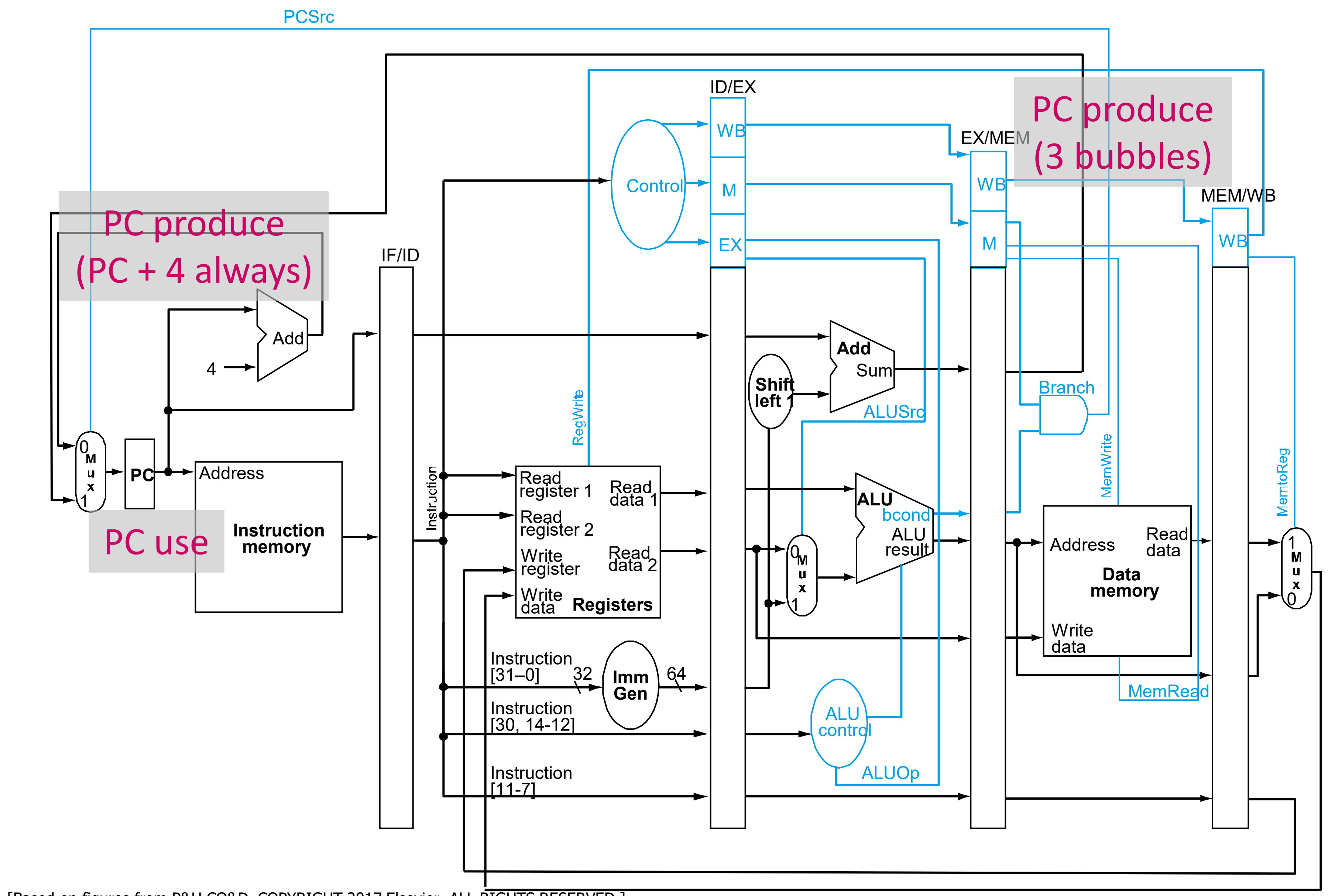
Performance Impact (PC+4 prediction)

- Correct guess → No penalty ~**86%** of the time
- Incorrect guess → 3 bubbles
- Assume
  - No data hazards
  - 20% are control flow instructions
  - 70% of control flow instructions are taken
  - IPC = 1 / [1 + (0.2 * 0.7) * 3] =

$$= 1 / [\, 1 + \underbrace{\mathbf{0.14}}_{\text{Probability of a wrong guess}} * \underbrace{3}_{\text{Penalty for a wrong guess}} \,] = 1 / 1.42 = 0.70$$

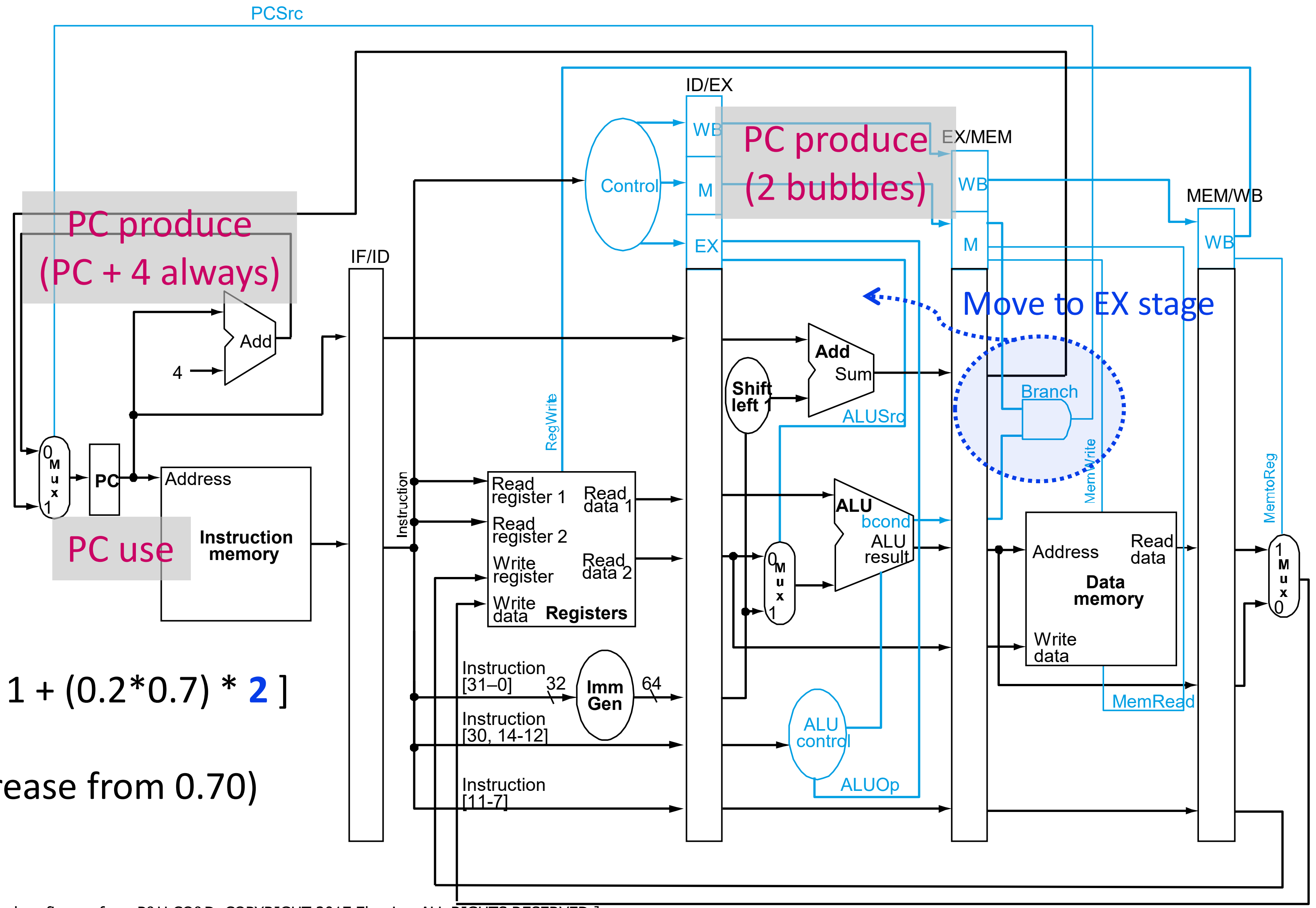
Can we reduce either of the two factors?

Reducing Misprediction Penalty



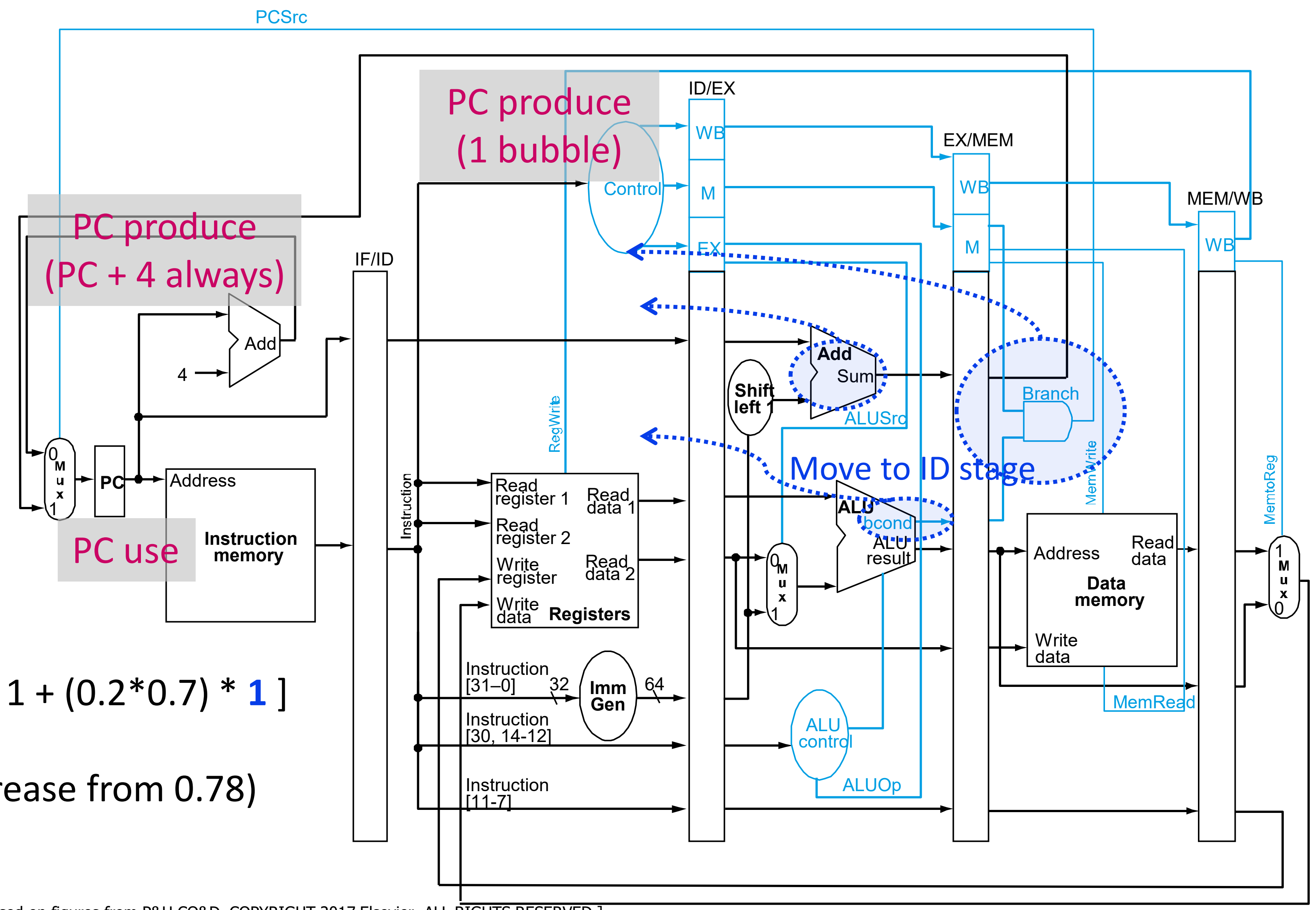
[Based on figures from P&H CO&D, COPYRIGHT 2017 Elsevier. ALL RIGHTS RESERVED.]

Reducing Misprediction Penalty



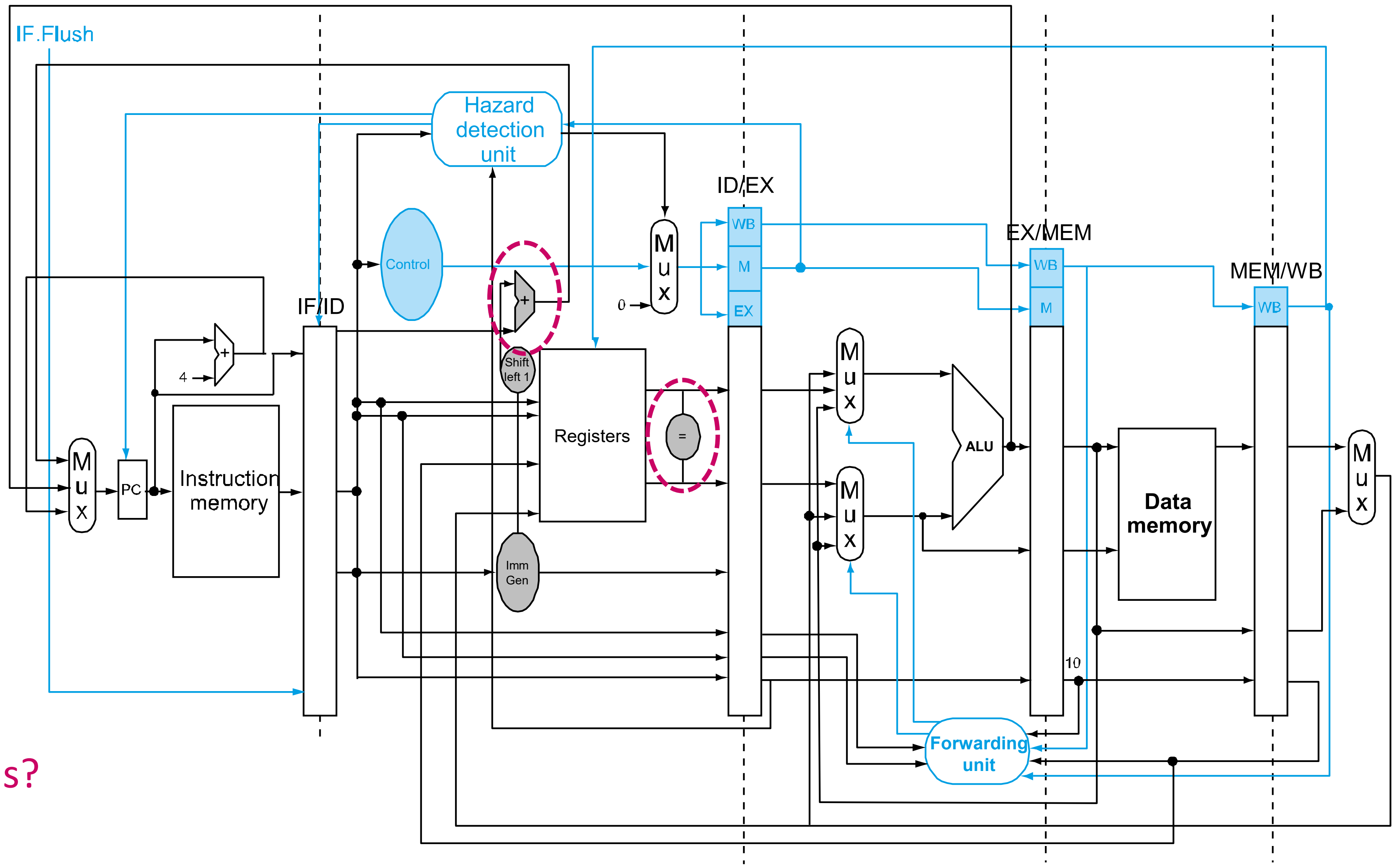
IPC = 1 / [1 + (0.2*0.7) * **2**]
= 0.78
(~11% increase from 0.70)

Reducing Misprediction Penalty Further



IPC = 1 / [1 + (0.2*0.7) * **1**]
= 0.88
(~12% increase from 0.78)

Branch Resolved in ID Stage



Additional considerations?

[Based on figures from P&H CO&D, COPYRIGHT 2017 Elsevier. ALL RIGHTS RESERVED.]

Final PC Hazard Analysis

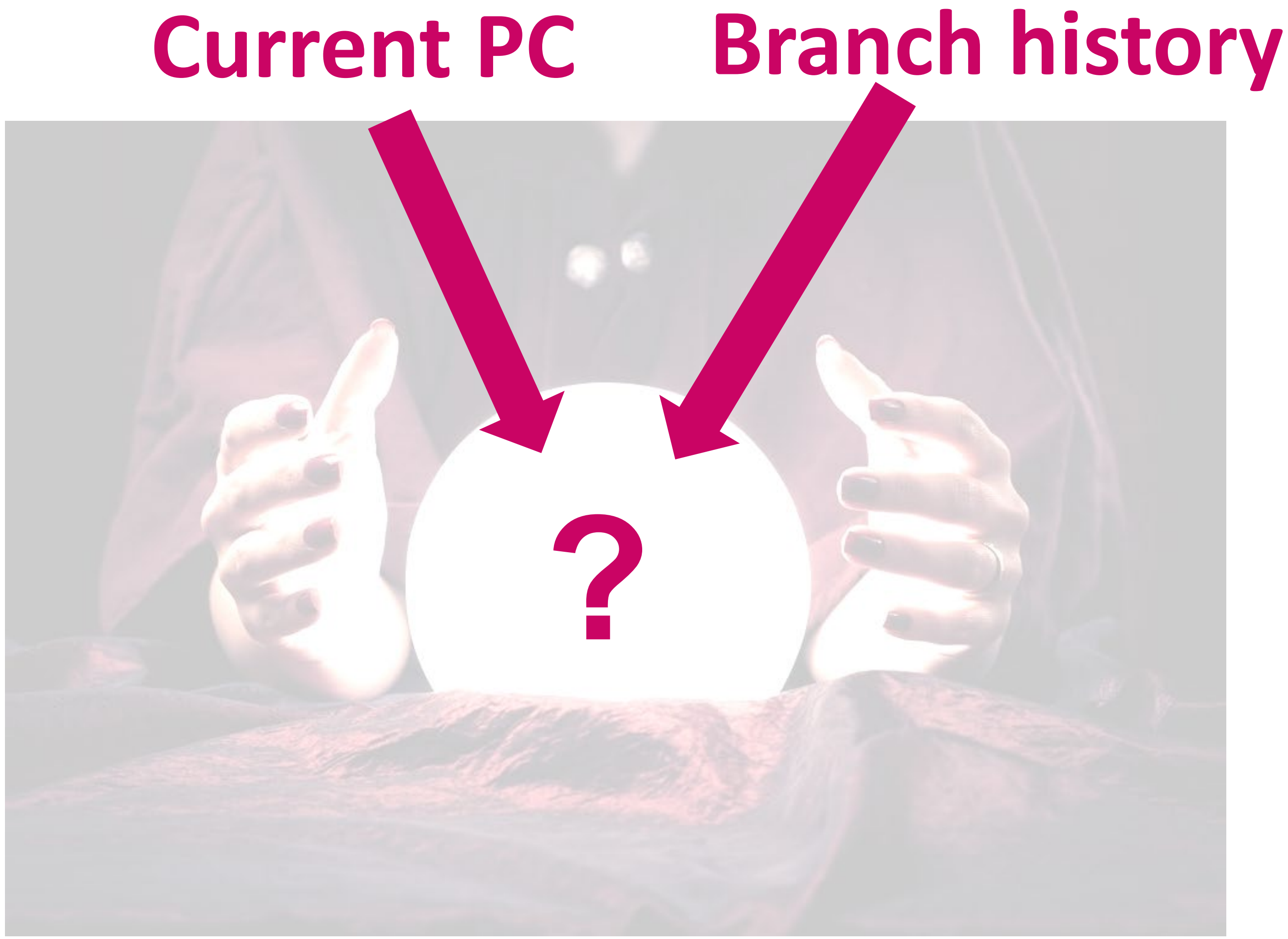
	R/I-Type	LW	SW	Bxx	JAL	JALR
IF	use (produce)	use (produce)	use (produce)	use	use	use
ID				produce	produce	?
EX						?
MEM						?
WB						?

- Hazard distance on a taken branch is 1
- Jump target for JAL calculated similar to Bxx
- Jump target for JALR still calculated at ? stage
- Hazard distance is greater in modern CPUs. Why?

Final Data Hazard Analysis (with Forwarding)

	R/I-Type	LW	SW	Bxx	JAL	JALR
IF						
ID				use		
EX	use produce	use	use	use	produce	use produce
MEM		produce	(use)			
WB						

Next Lecture: Making Better Predictions



Question?

Announcements

- Reading: P&H (RISC-V ed.) Ch 4.9