Universiteit Leiden

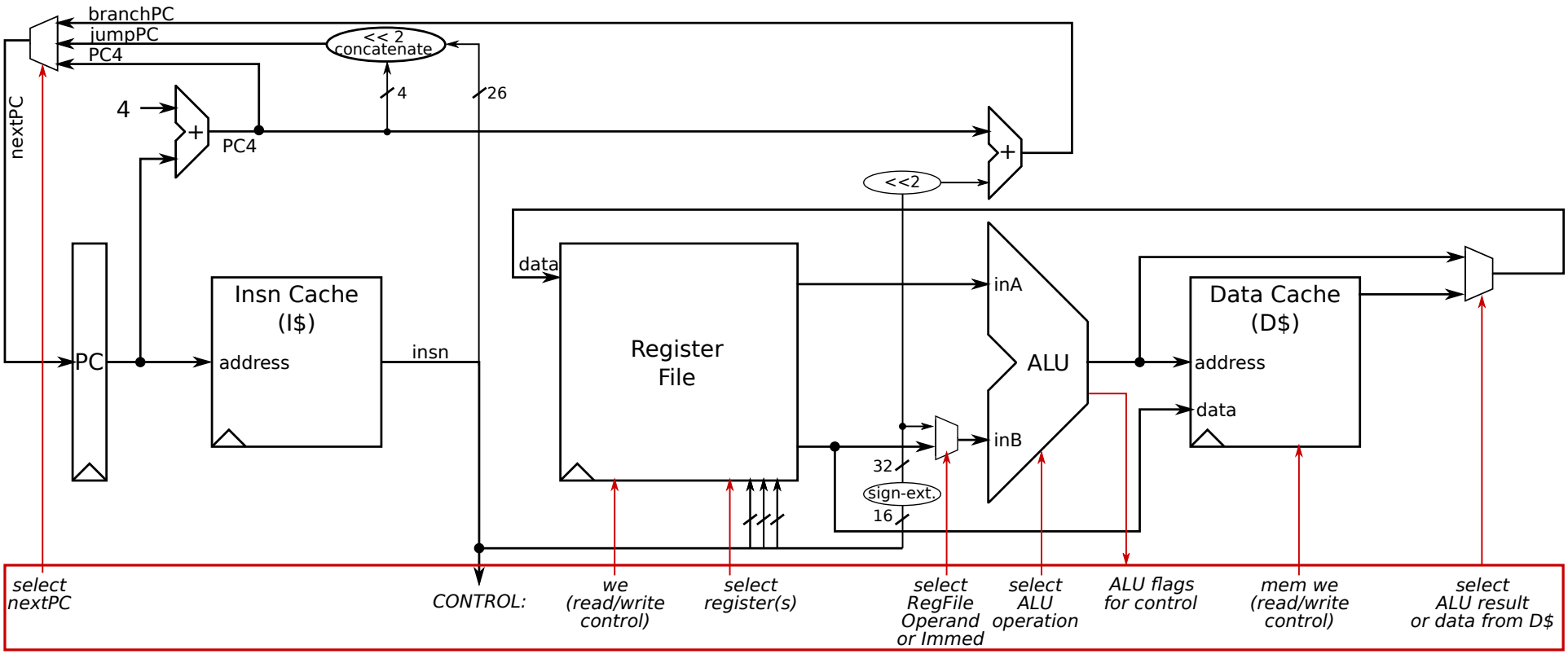
EssCS - Topic 2

Introduction to Computer Architecture

Lecture 7, 15.10.2024

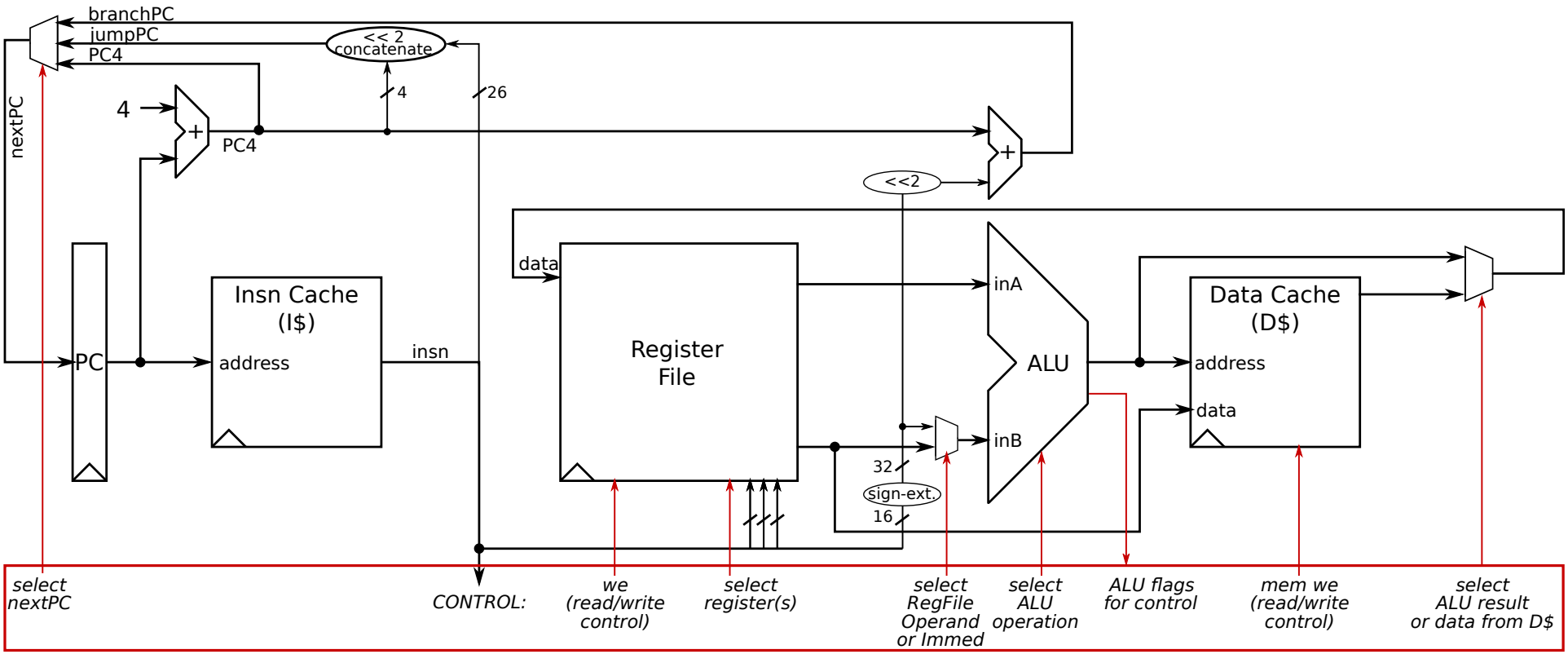
Nuša Zidarič

Single Cycle Datapath



insn syntax	operation	meaning	comments
add Rd, Rs, Rt	Rd $\leftarrow$ Rs + Rt	add	arithmetic op.
or Rd, Rs, Rt	Rd $\leftarrow$ Rs OR Rt	bitwise or	logic op.

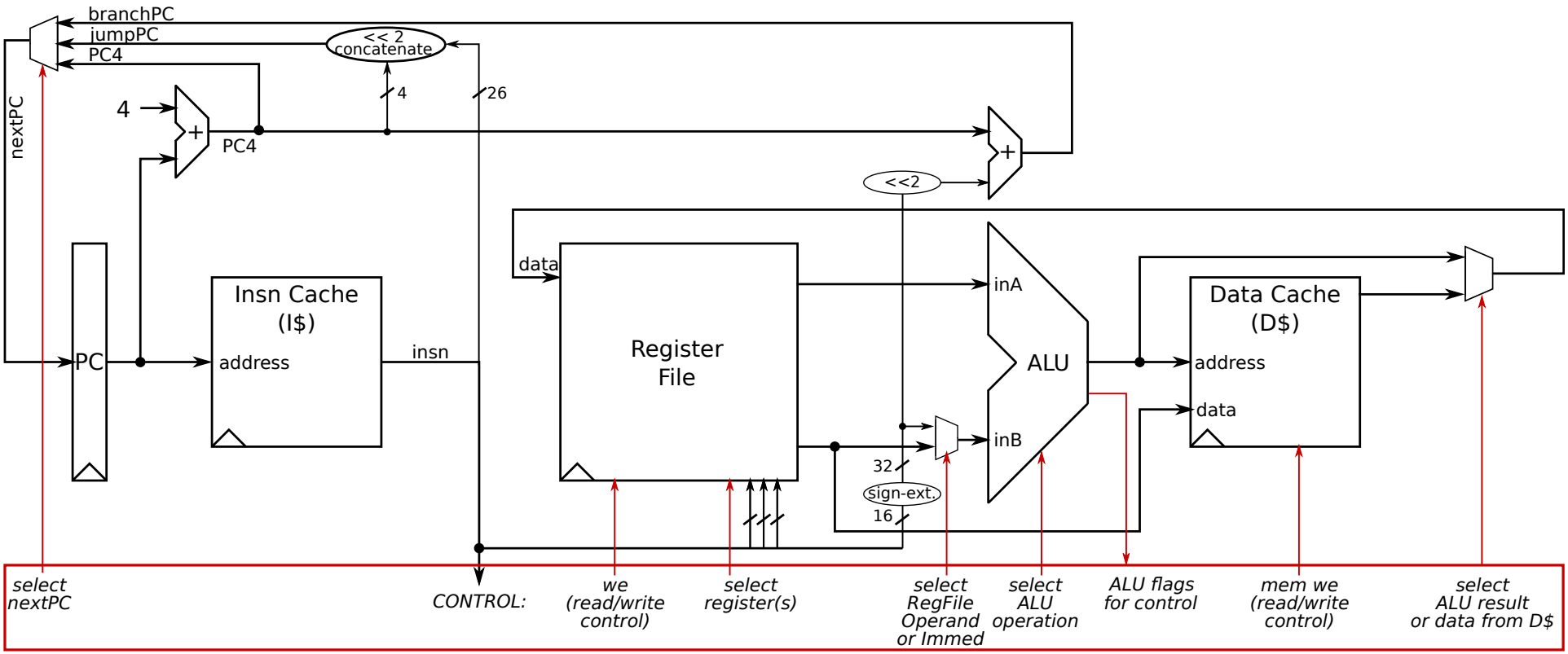
Single Cycle Datapath



insn syntax	operation	meaning	comments
lw Rt, offset(Rs)	Rt ← MEM[Rs+offset]	load word	data transfer
sw Rt, offset(Rs)	MEM[Rs+offset]← Rt	store word	data transfer

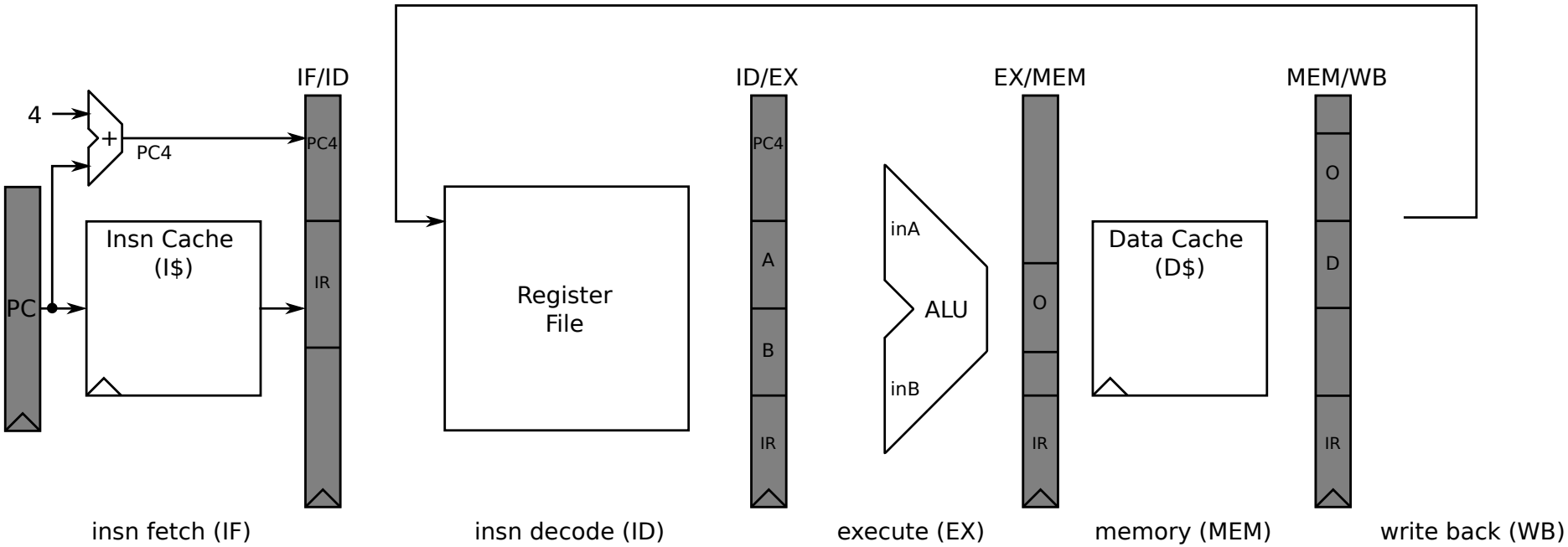
MEM refers to D$

Single Cycle Datapath



insn syntax	operation	meaning	comments
beq Rs, Rt, Label	if true: PC←new	branch if equal	conditional
j Label	PC←new	jump	unconditional

Pipelined Datapath

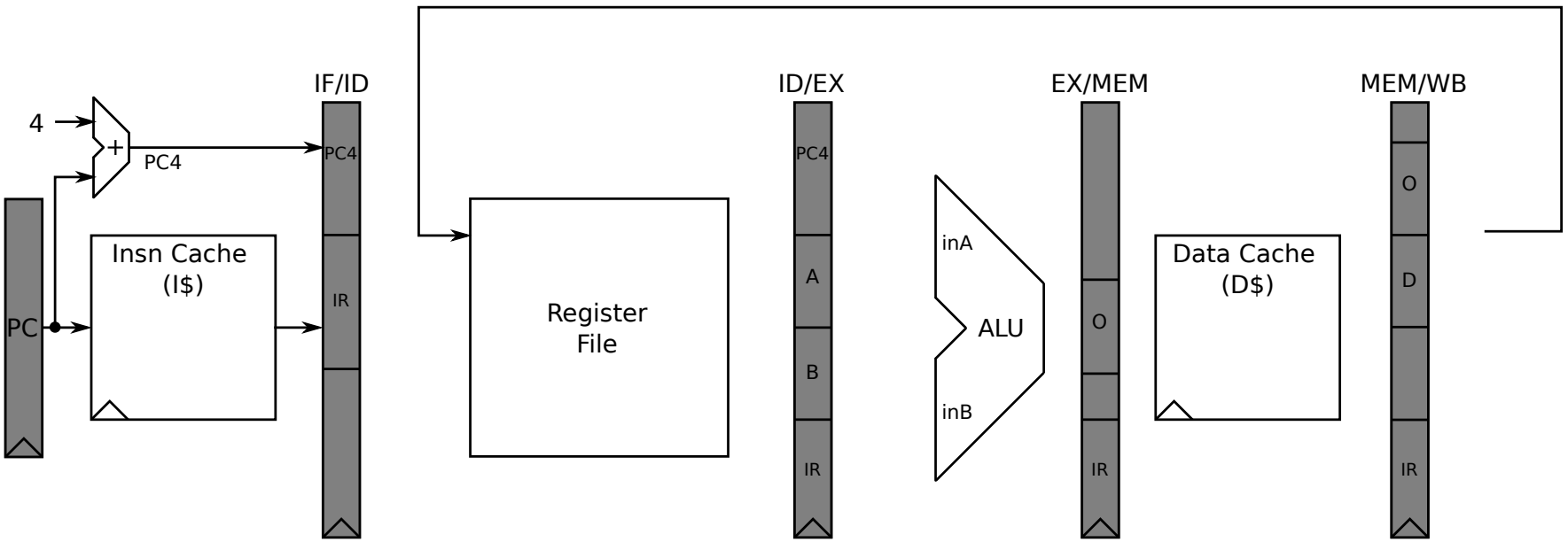


- similarities w.r.t. Topic 1: fully pipelined design (latency = , Tput =)
- keyword: replicate (each stage has separate hardware!)
- parcel: insn

- differences w.r.t. Topic 1: each stage has a different functionality

- different notation for timing diagram
  - before: clock cycles (columns) and hardware components (rows) on borders, parcels inside
  - now: clock cycles (columns) and insns (rows) on borders, stages (IF, ID, EX, MEM, WB) inside

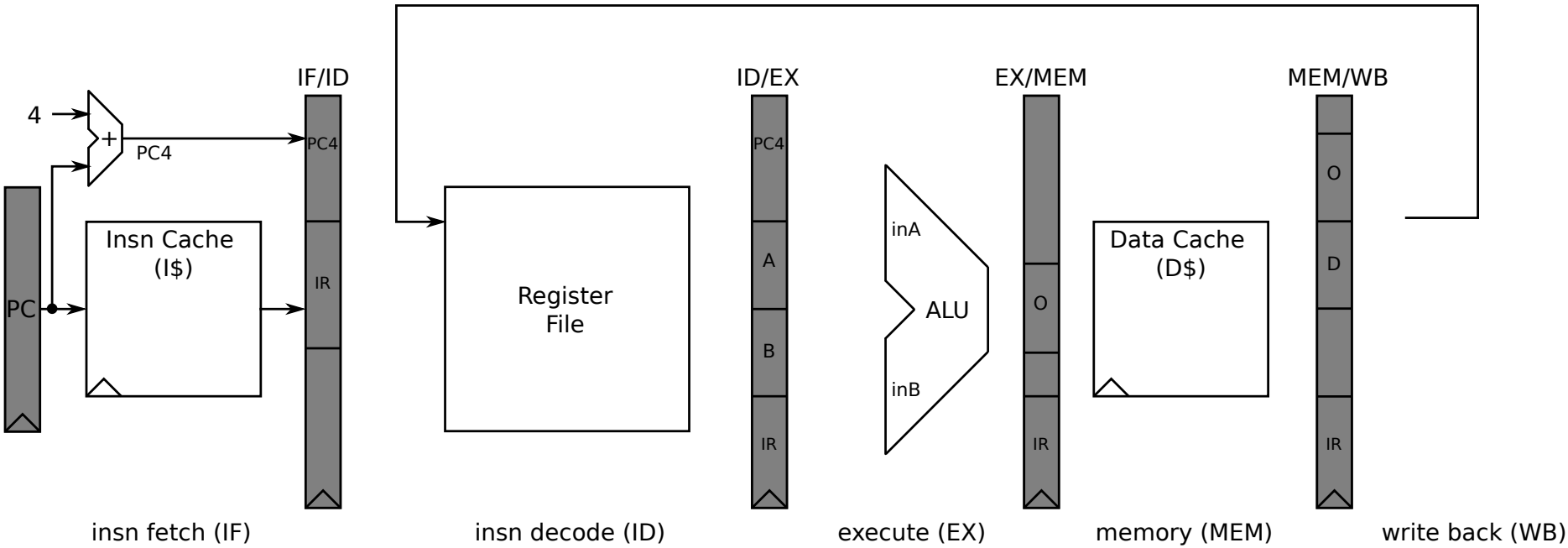
Pipelined Datapath



cycle	0	1	2	3	4	5	6	7	8
add Rd, Rs, Rt									

insn syntax	operation
add Rd, Rs, Rt	Rd $\leftarrow$ Rs + Rt.

Pipelined Datapath



insn syntax	operation
add Rd, Rs, Rt	Rd ← Rs + Rt.

cycle	0	1	2	3	4	5	6	7	8
add Rd, Rs, Rt									

cycle	0	1	2	3	4	5	6	7	8
add R3, R1, R2 add R7, R1, R3									

Data Dependencies

- Read-after-Write (RAW)

  $R3 \leftarrow R1 + R2$

  $R7 \leftarrow R1 + R3$

  insn 2 must wait for insn 1 to complete:

  insn 1 writes

  insn 2 read the **same** register

  solution1: stalling (inserting NOPs)

- Write-after-Read (WAR)

  $R3 \leftarrow R1 + R2$

  $R1 \leftarrow R7 + R5$

  insn 1 must read **old** value (not a problem if they execute in the correct order)

  insn 1 reads

  insn 2 writes the **same** register

- Write-after-Write (WAW)

  $R3 \leftarrow R1 + R2$

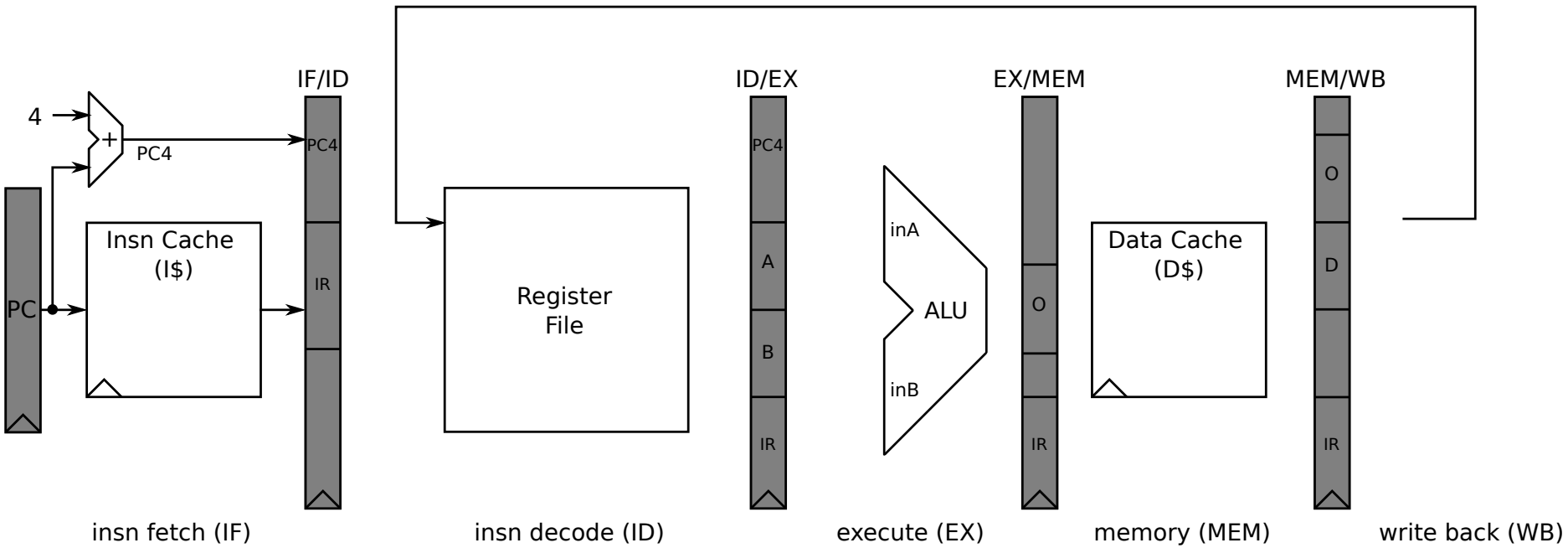
  $R3 \leftarrow R7 + R5$

  insn 2 must complete **after** insn 1 (otherwise wrong result overwrites correct)

  insn 1 writes

  insn 2 writes the **same** register

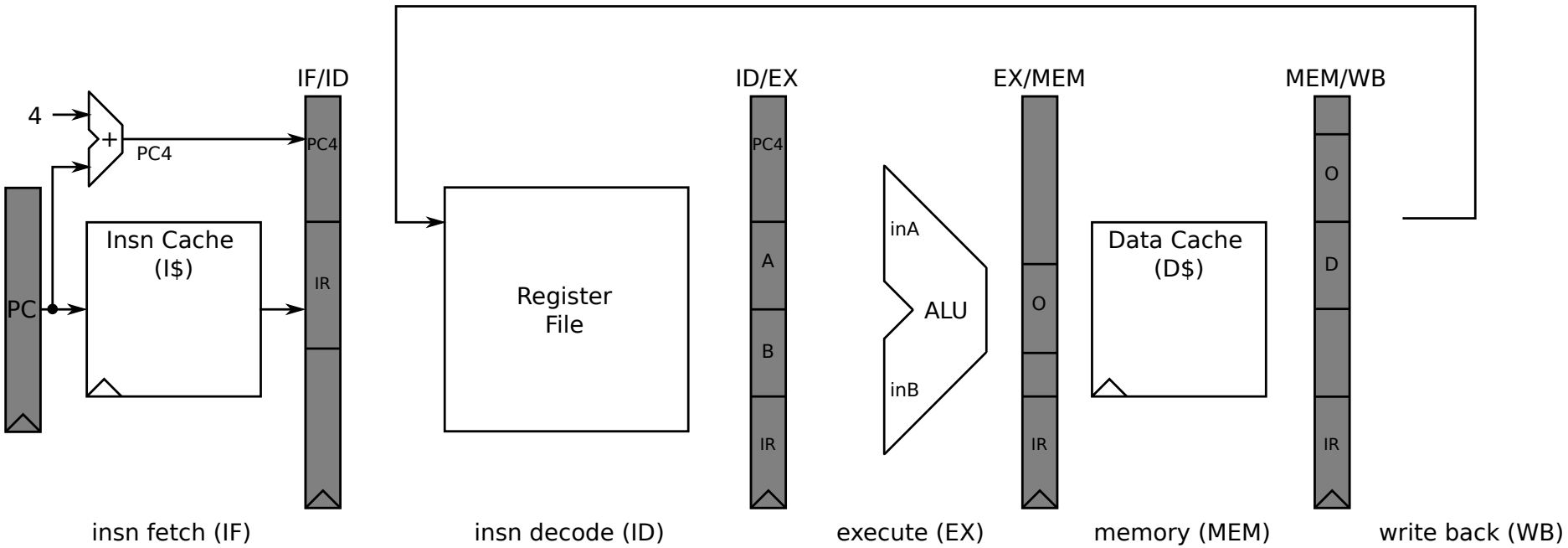
Pipelined Datapath



insn syntax	operation
add Rd, Rs, Rt	Rd ← Rs + Rt.
lw Rt, offset(Rs)	Rt ← MEM[Rs+offset]
sw Rt, offset(Rs)	MEM[Rs+offset]← Rt

cycle	0	1	2	3	4	5	6	7	8	9	10	11
add R3, R1, R2												
add R7, R1, R3												
lw R5, 12(R3)												
sw R5, 0(R7)												

Pipelined Datapath



Solution 2: BYPASSING (requires extra hardware):
EX/MEM or MEM/WB $\rightarrow$ inputs to EX, MEM/WB $\rightarrow$ data input to MEM

insn syntax	operation
add Rd, Rs, Rt	Rd $\leftarrow$ Rs + Rt.

cycle	0	1	2	3	4	5	6	7	8	9	10	11
add R3, R1, R2												
add R7, R1, R3												

CONTROL HAZARDS: branch instructions
condition is checked in EX and we already have two insns in IF and ID $\Rightarrow$ we have to "NOP them out"

Main Memory

- **Main memory:** one-dimensional array of words
  holds both the insns and data

- how bits are organized and how to access them ?
  2 parameters: memory word and memory address

- memory word: smallest number of bits with the same address
  word length = # bits in the word = n (one word can store 2^n different values)
  word length n is always a multiple of the CPU register length

  NOTE: we can access more then n bits at a time, but not less !

- address: a number that uniquely defines one memory word (location)
  address length = # bits in address = m max memory size = $2^m \cdot n$

  NOTE: too small m is one of the biggest design mistakes !

- Speed vs. Capacity: we want a fast and a big main memory
  $\Rightarrow$ solution: memory hierarchy

Memory Hierarchy

- **from the view-point of the memory:** in a given time-frame, certain addresses are more likely then others !
- the sequence of addresses for a given program is not random
- different programs will form different sequences of addresses and will exhibit different degrees of locality
  - usual: $A(i+1) = A(i) + 1$
  - deviations: jumps, branches, swapping between insn and data access
  - we are likely to find repeating insns (e.g. loops) and data (e.g. arrays)

- **temporal locality:** recently accessed will likely be accessed in the near future
- **spatial locality:** next address will be close to the current one

- **GOAL:** optimize average access time by making a big, slow main memory look like the small, fast L1 cache (we have seen a **split I$ and D$**, we will consider higher levels of hierarchy as unified cache)

hierarchy	L1		L2		M
coherence	L1	$\subset$	L2	$\subset$	M
size	L1	$<$	L2	$<$	M
speed	L1	$>$	L2	$>$	M