Pipeline Hazards

Hakim Weatherspoon

CS 3410, Spring 2013

Computer Science

Cornell University

See P&H Chapter 4.7

Goals for Today

Data Hazards

- Revisit Pipelined Processors
- Data dependencies
- Problem, detection, and solutions
  - (delaying, stalling, forwarding, bypass, etc)
- Hazard detection unit
- Forwarding unit

Next time

- Control Hazards

  What is the next instruction to execute if a branch is taken? Not taken?

MIPS Design Principles

Simplicity favors regularity

- 32 bit instructions

Smaller is faster

- Small register file

Make the common case fast

- Include support for constants

Good design demands good compromises

- Support for different type of interpretations/classes

Recall: MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats

R-type

`op`	`rs`	`rt`	`rd`	`shamt`	`func`
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-type

`op`	`rs`	`rt`	`immediate`
6 bits	5 bits	5 bits	16 bits

J-type

`op`	`immediate (target address)`
6 bits	26 bits

Recall: MIPS Instruction Types

Arithmetic/Logical

- R-type: result and two source registers, shift amount
- I-type: 16-bit immediate with sign/zero extension

Memory Access

- load/store between registers and memory
- word, half-word and byte operations

Control flow

- conditional branches: pc-relative addresses
- jumps: fixed offsets, register absolute

Recall: MIPS Instruction Types

Arithmetic/Logical

- ADD, ADDU, SUB, SUBU, AND, OR, XOR, NOR, SLT, SLTU
- ADDI, ADDIU, ANDI, ORI, XORI, LUI, SLL, SRL, SLLV, SRLV, SRAV, SLTI, SLTIU
- MULT, DIV, MFLO, MTLO, MFHI, MTHI

Memory Access

- LW, LH, LB, LHU, LBU, LWL, LWR
- SW, SH, SB, SWL, SWR

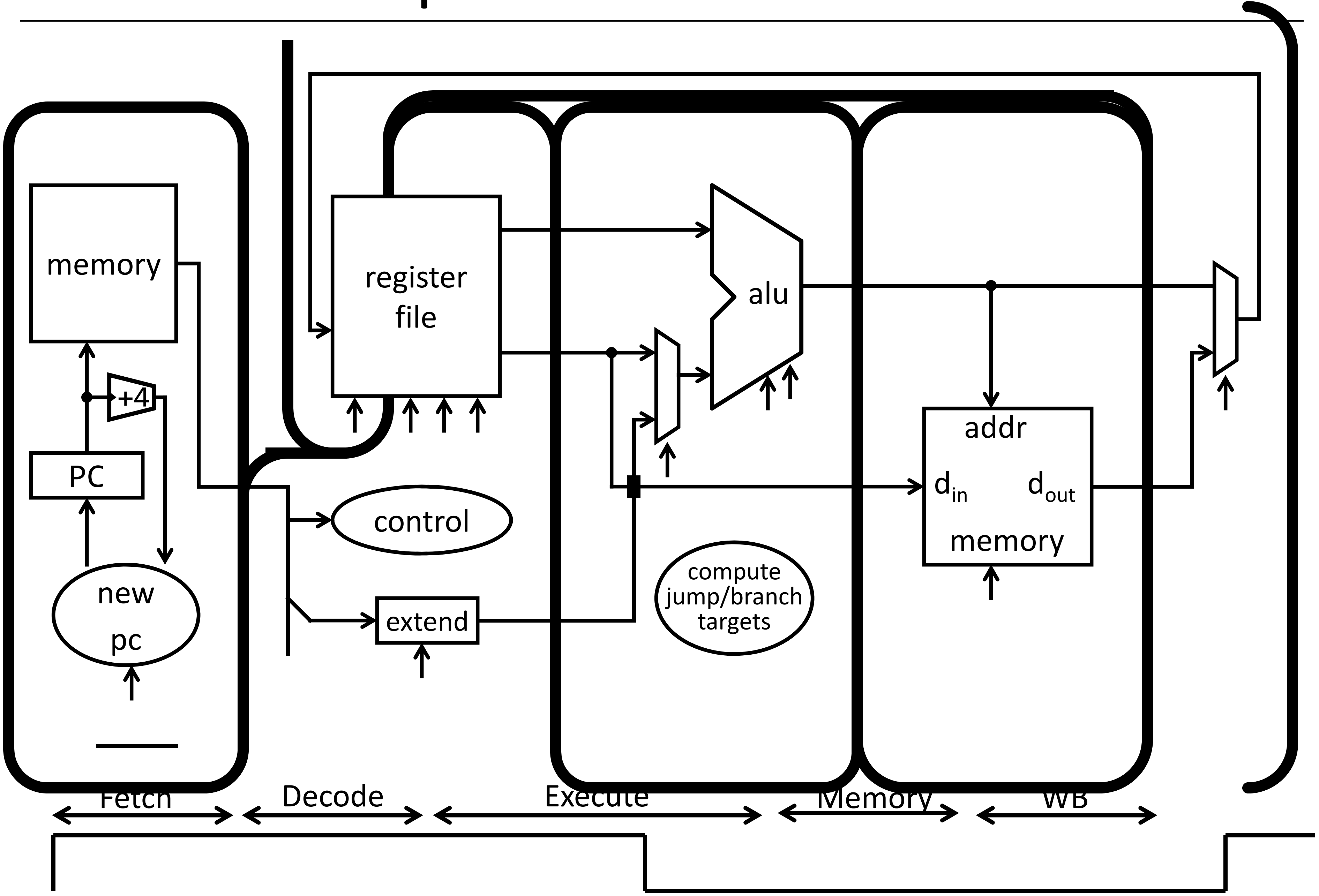
Control flow

- BEQ, BNE, BLEZ, BLTZ, BGEZ, BGTZ
- J, JR, JAL, JALR, BEQL, BNEL, BLEZL, BGTZL

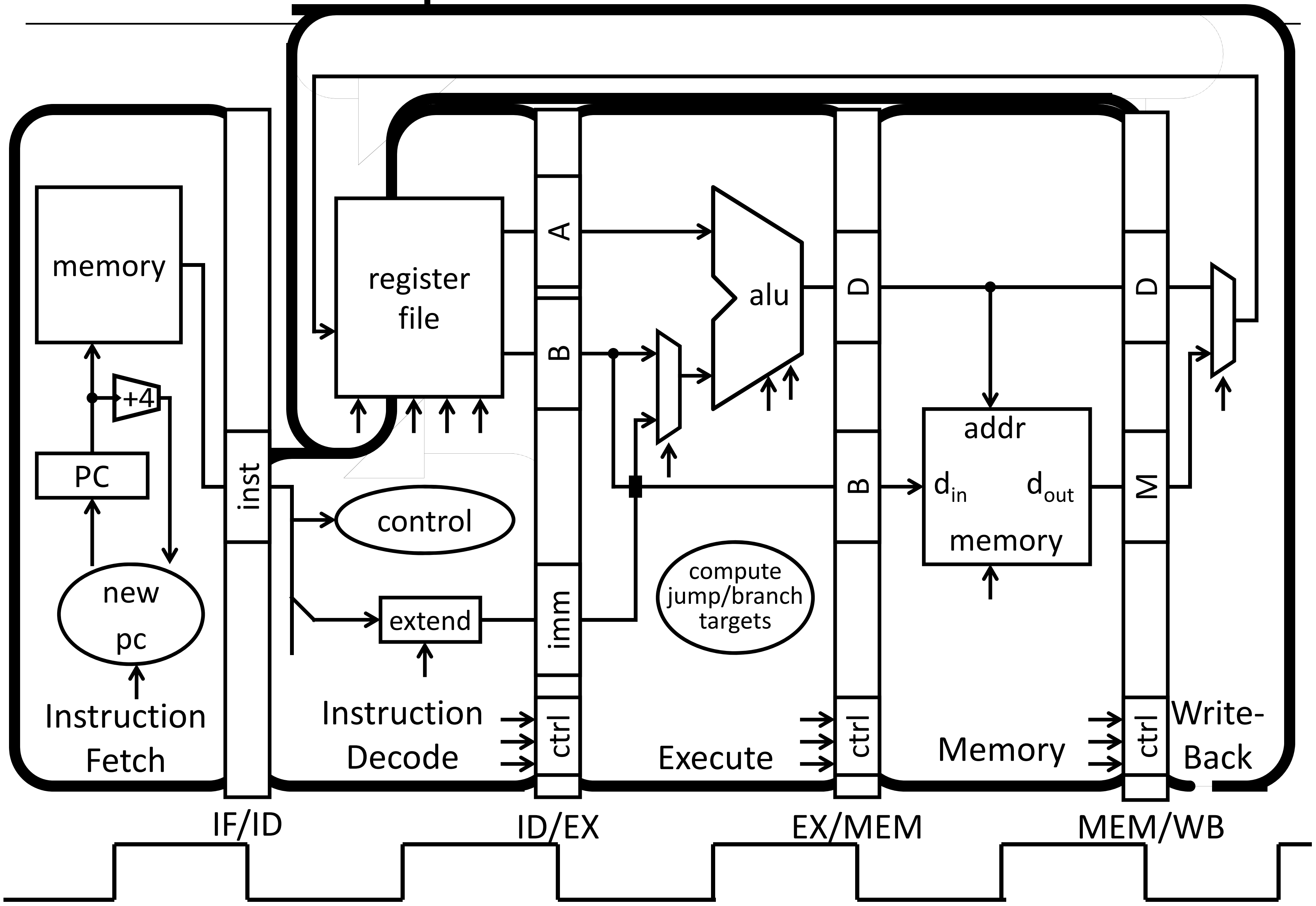
Special

- LL, SC, SYSCALL, BREAK, SYNC, COPROC

Pipelined Processor



Pipelined Processor



Example: : Sample Code (Simple)

```
add    r3, r1, r2;
nand   r6, r4, r5;
lw     r4, 20(r2);
add    r5, r2, r5;
sw     r7, 12(r3);
```

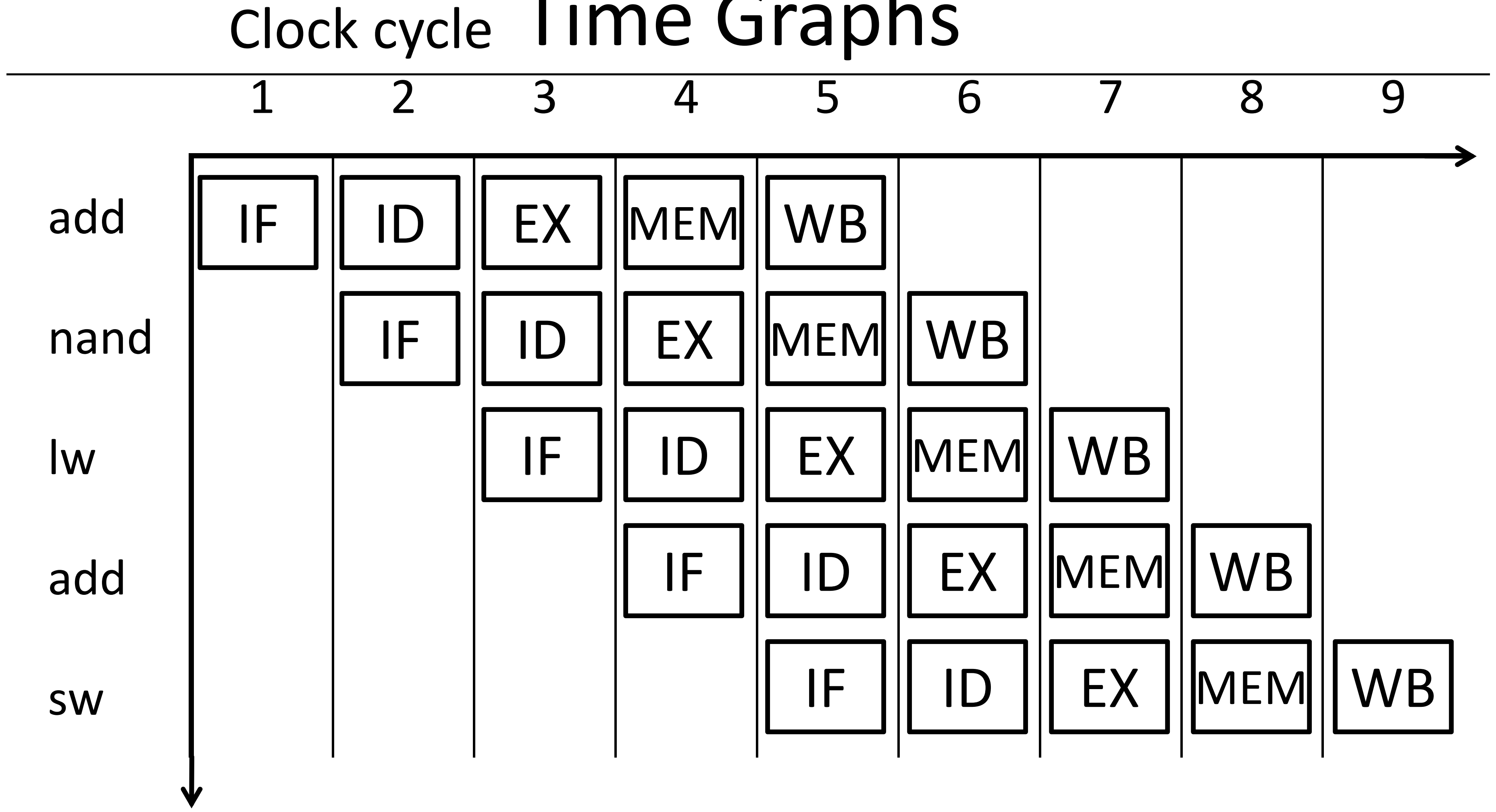
Example: Sample Code (Simple)

Assume eight-register machine

Run the following code on a pipelined datapath

```
add     r3  r1   r2    ;  reg 3 = reg 1 + reg 2
nand    r6   r4   r5   ;  reg 6 = ~(reg 4 & reg 5)
lw       r4   20 (r2)  ;  reg 4 =  Mem[reg2+20]
add     r5   r2   r5   ;  reg 5 = reg 2 + reg 5
sw      r7    12(r3)   ;  Mem[reg3+12] = reg 7
```

Time Graphs



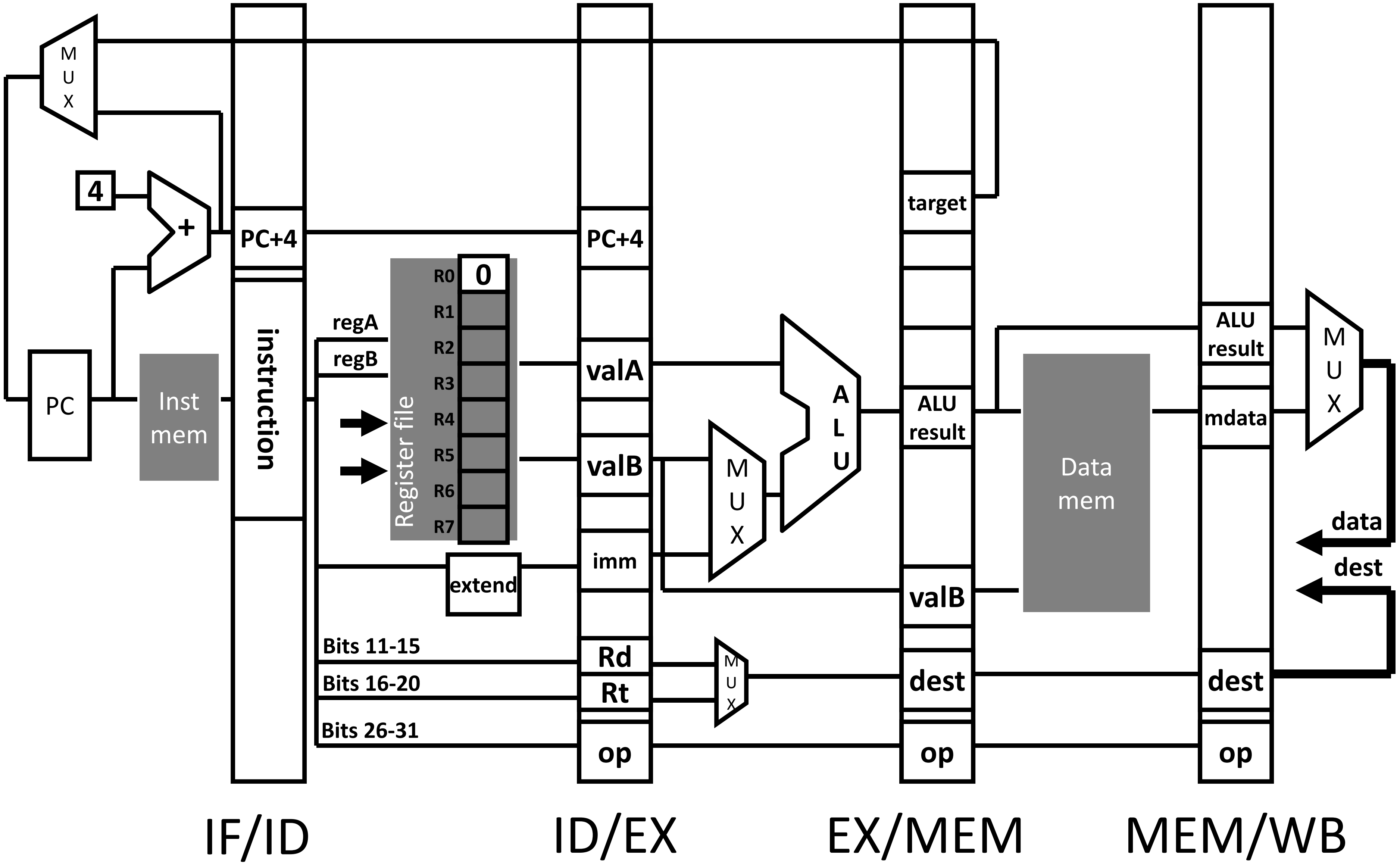
Clock cycle	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
nand		IF	ID	EX	MEM	WB			
lw			IF	ID	EX	MEM	WB		
add				IF	ID	EX	MEM	WB	
sw					IF	ID	EX	MEM	WB

Latency:

Throughput:

Concurrency:

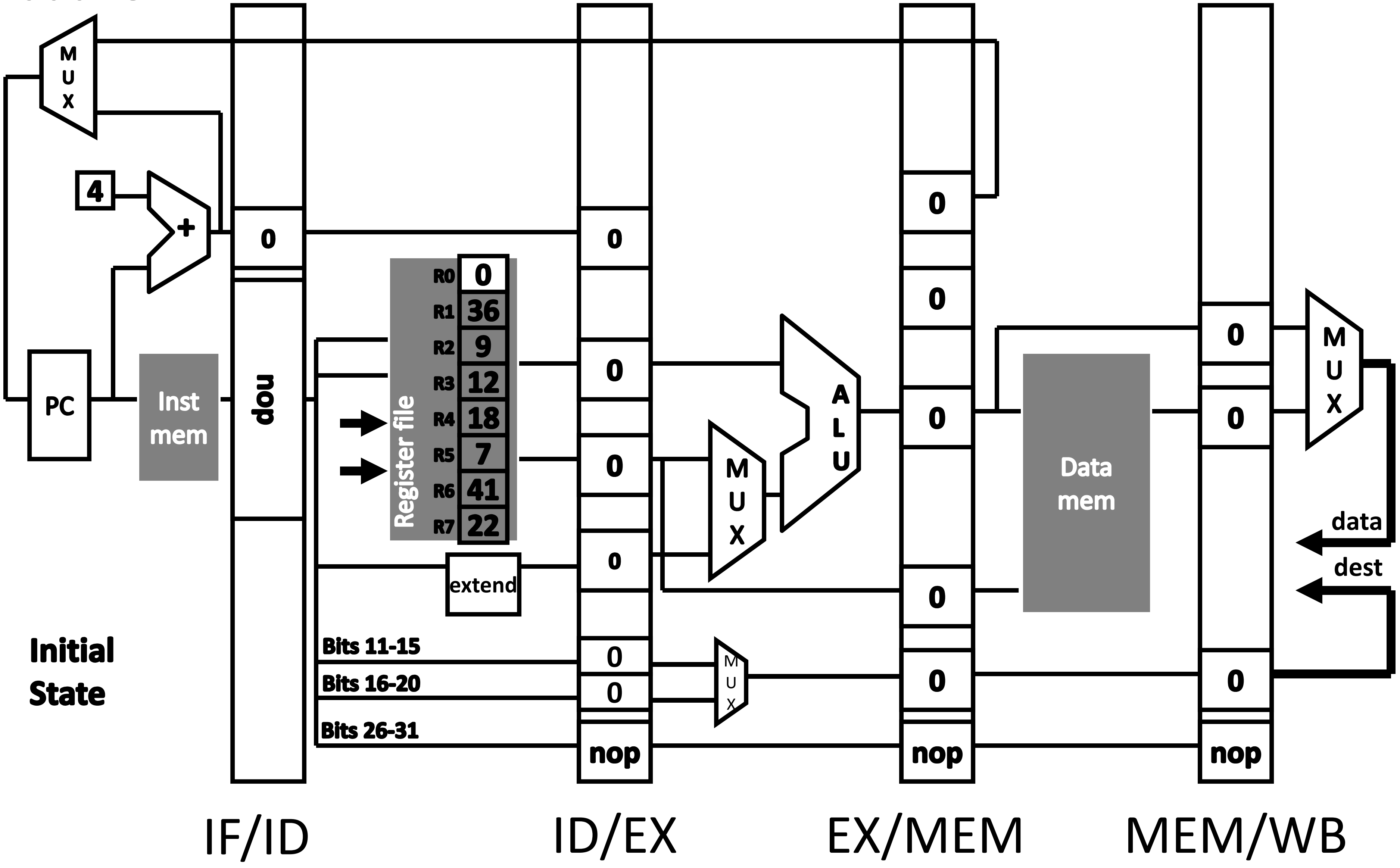
CPI =

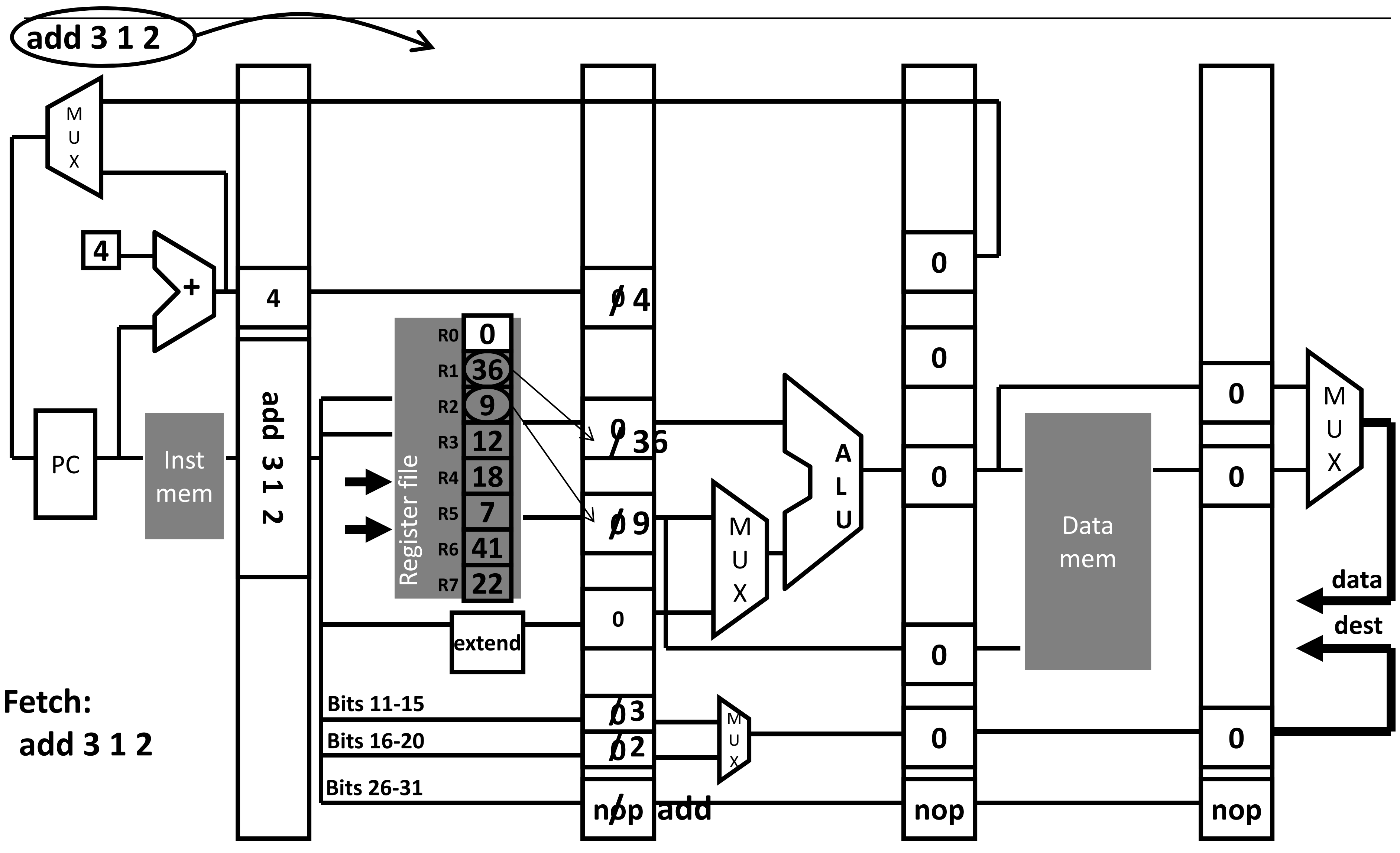


MUX
4
+
PC
Inst mem
PC+4
instruction
regA
regB
Register file
R0
0
R1
R2
R3
R4
R5
R6
R7
extend
Bits 11-15
Bits 16-20
Bits 26-31
PC+4
valA
valB
imm
Rd
Rt
op
MUX
ALU
MUX
target
ALU result
valB
dest
op
Data mem
ALU result
mdata
dest
op
MUX
data
dest
IF/ID
ID/EX
EX/MEM
MEM/WB

At time 1,
Fetch

add r3 r1 r2

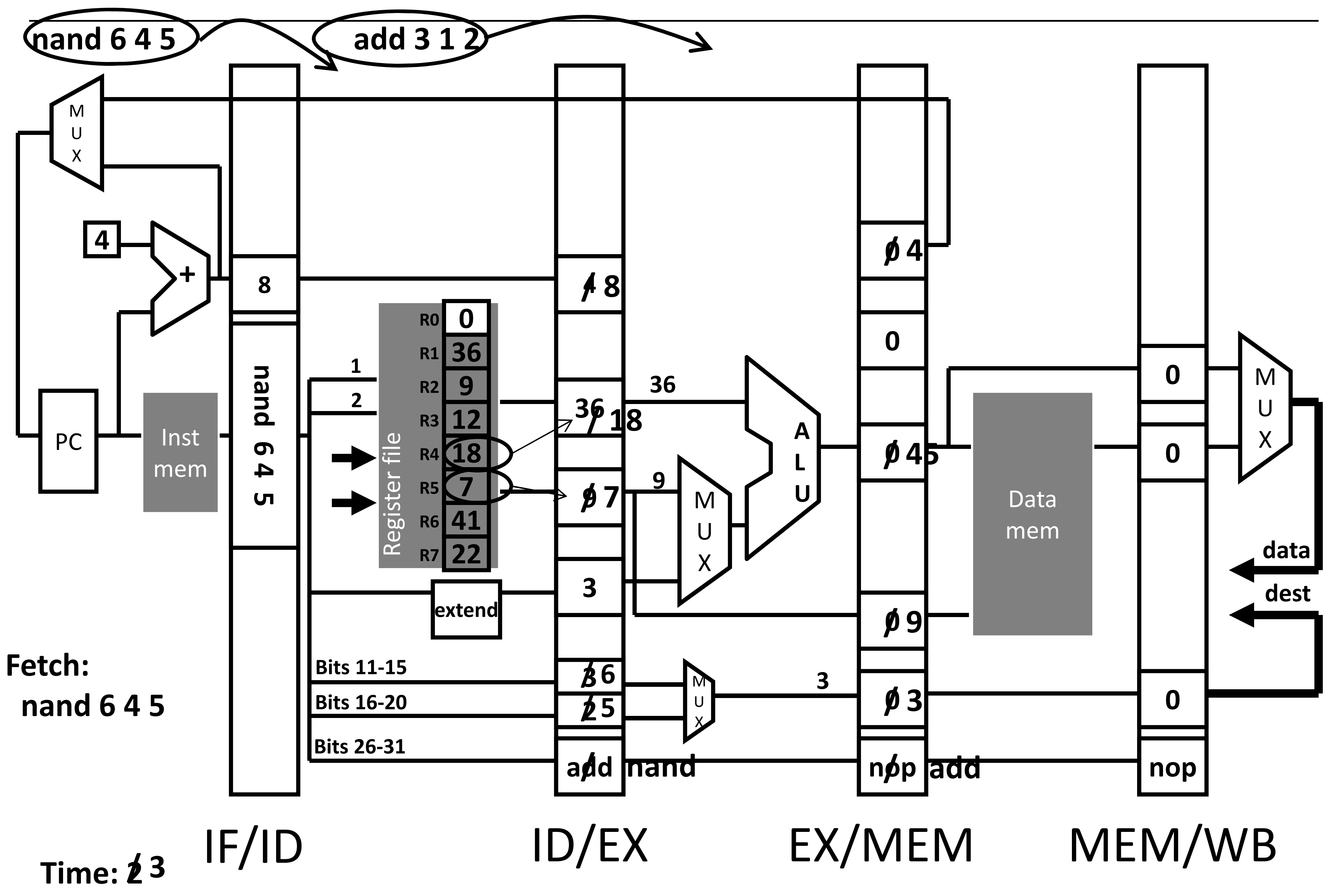




Fetch:
add 3 1 2

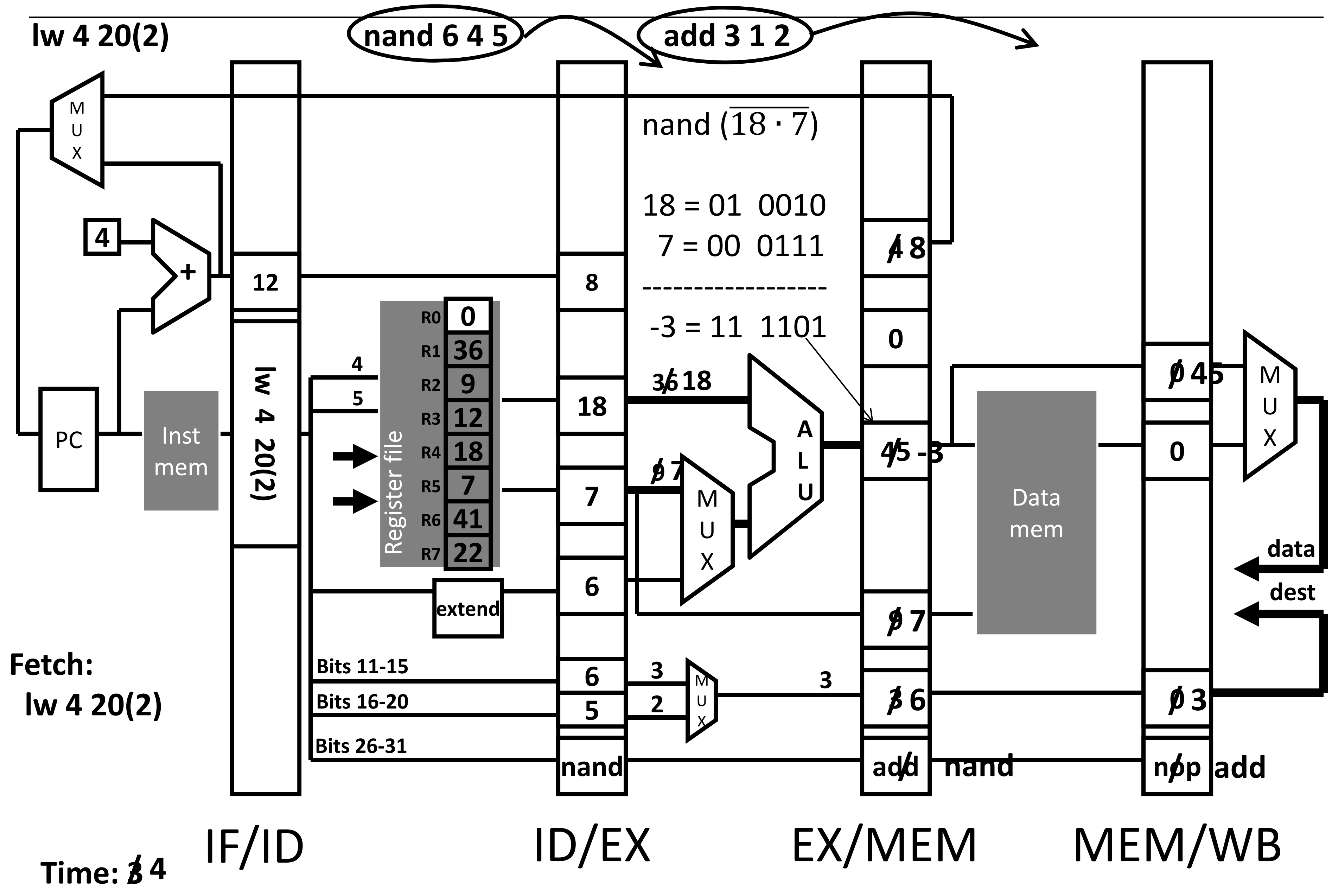
IF/ID ID/EX EX/MEM MEM/WB

Time: ~~1~~ 2

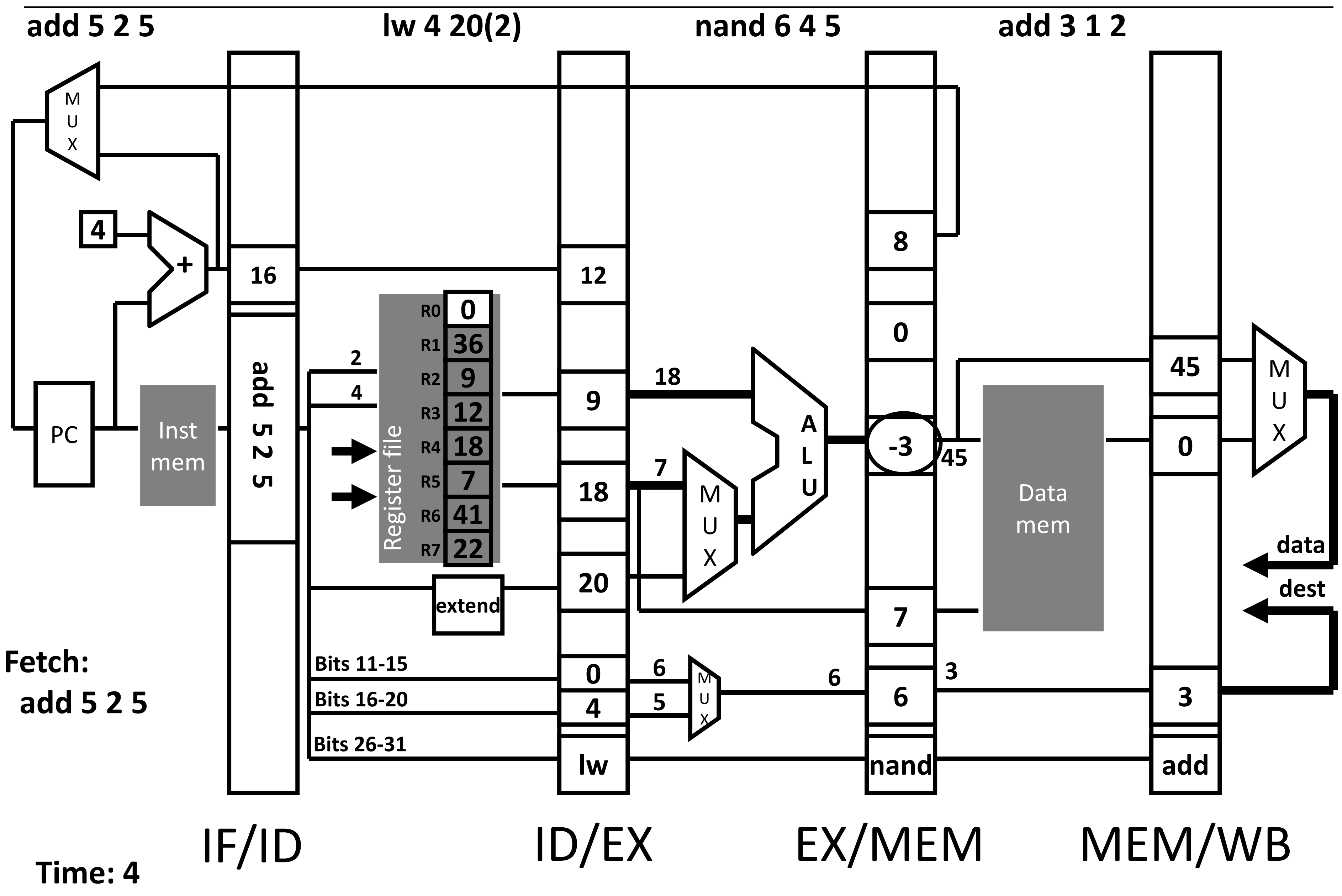


Fetch:
nand 6 4 5

Time: ~~2~~ 3

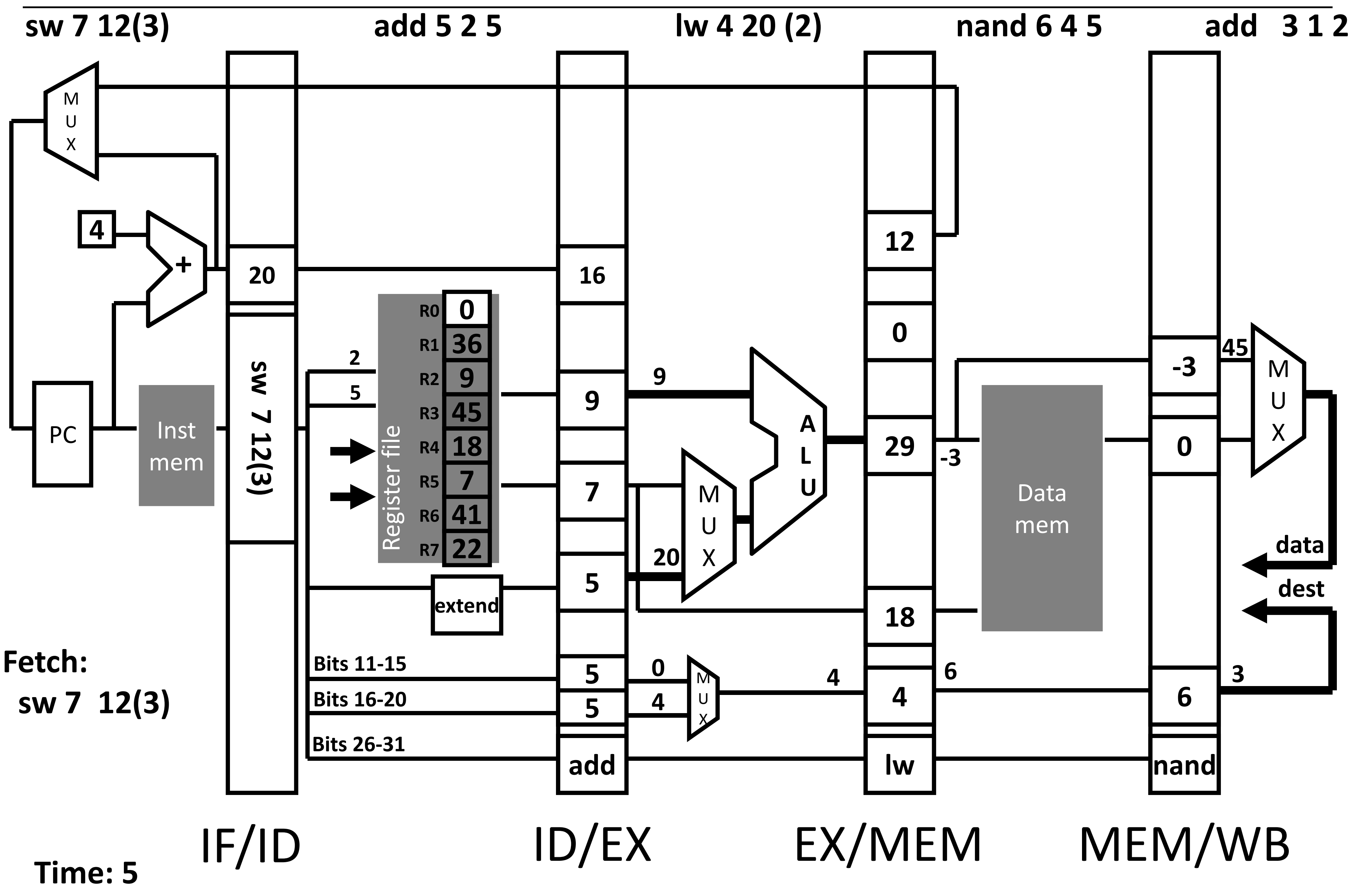

lw 4 20(2)
nand 6 4 5
add 3 1 2
MUX
4
+
12
PC
Inst mem
lw 4 20(2)
4
5
R0 0
R1 36
R2 9
R3 12
R4 18
R5 7
R6 41
R7 22
Register file
extend
Bits 11-15
Bits 16-20
Bits 26-31
8
18
7
6
6
5
nand
nand ($\overline{18 \cdot 7}$)
18 = 01 0010
7 = 00 0111

-3 = 11 1101
~~36~~ 18
~~9~~ 7
MUX
ALU
3
2
MUX
3
~~4~~ 8
0
~~45~~ -3
~~9~~ 7
~~3~~ 6
a~~dd~~ nand
Data mem
~~0~~ 45
0
~~0~~ 3
n~~op~~ add
MUX
data
dest
Fetch:
lw 4 20(2)
Time: ~~3~~ 4
IF/ID
ID/EX
EX/MEM
MEM/WB



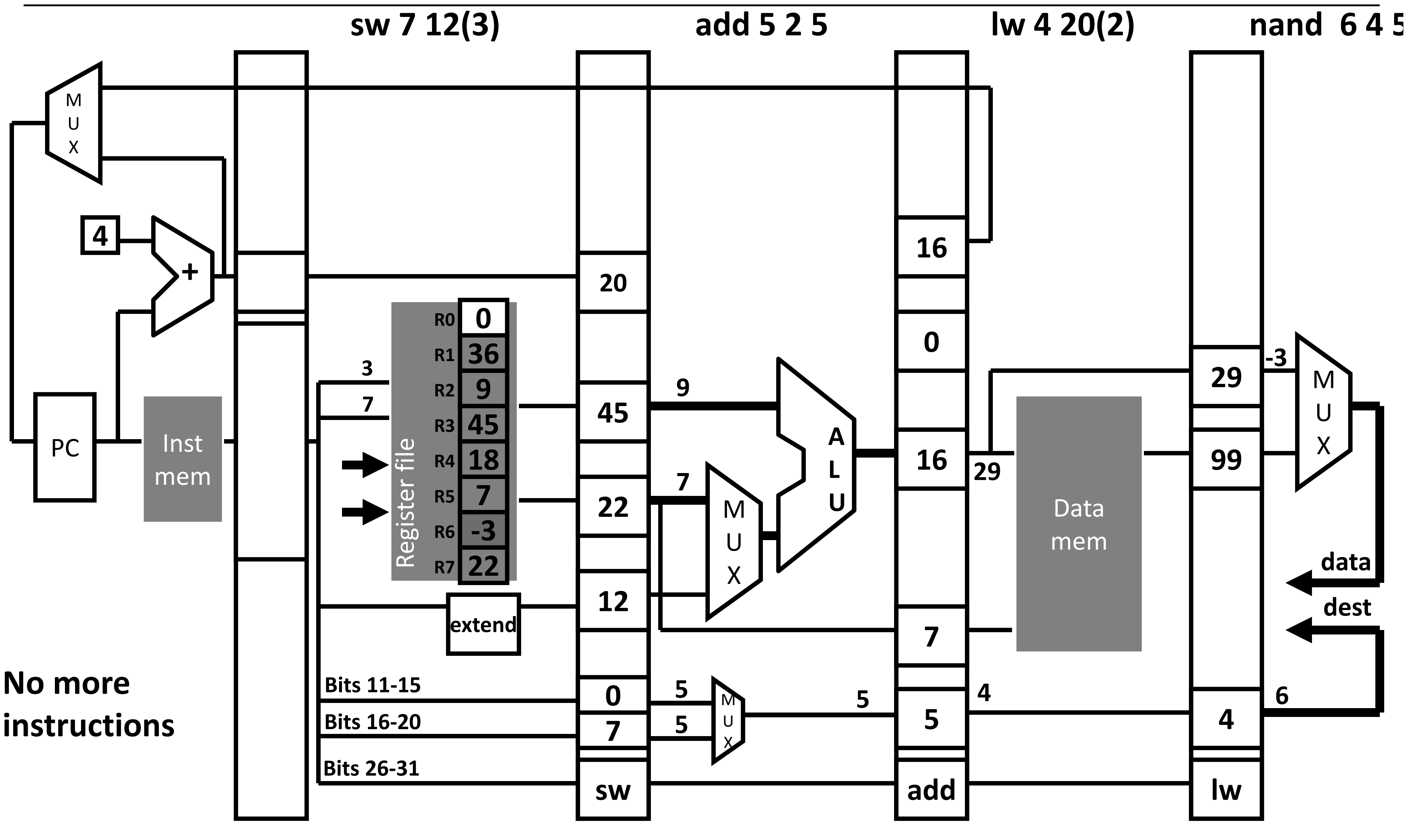
Fetch:
add 5 2 5

Time: 4



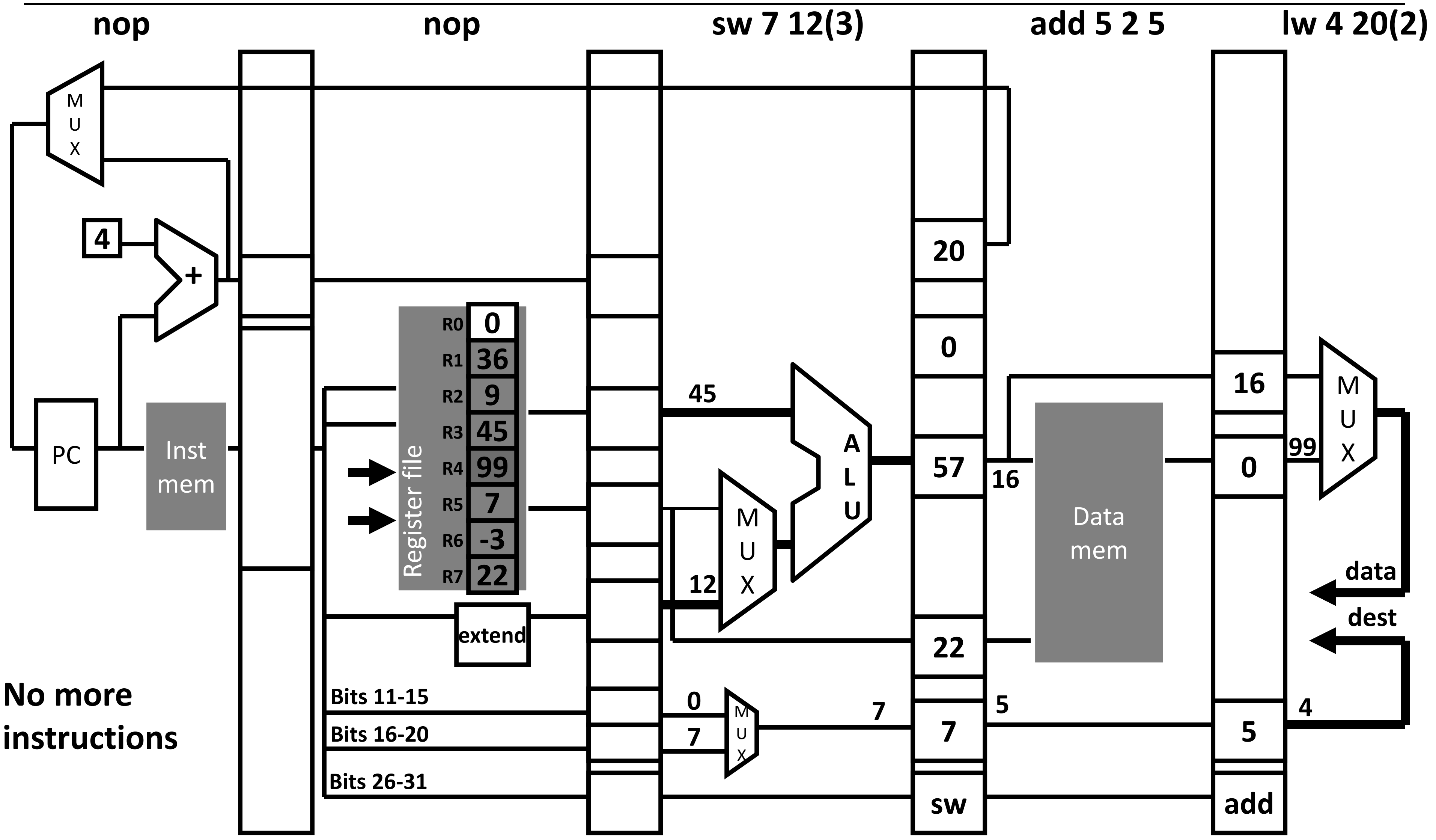
Fetch:
sw 7 12(3)

Time: 5



No more instructions

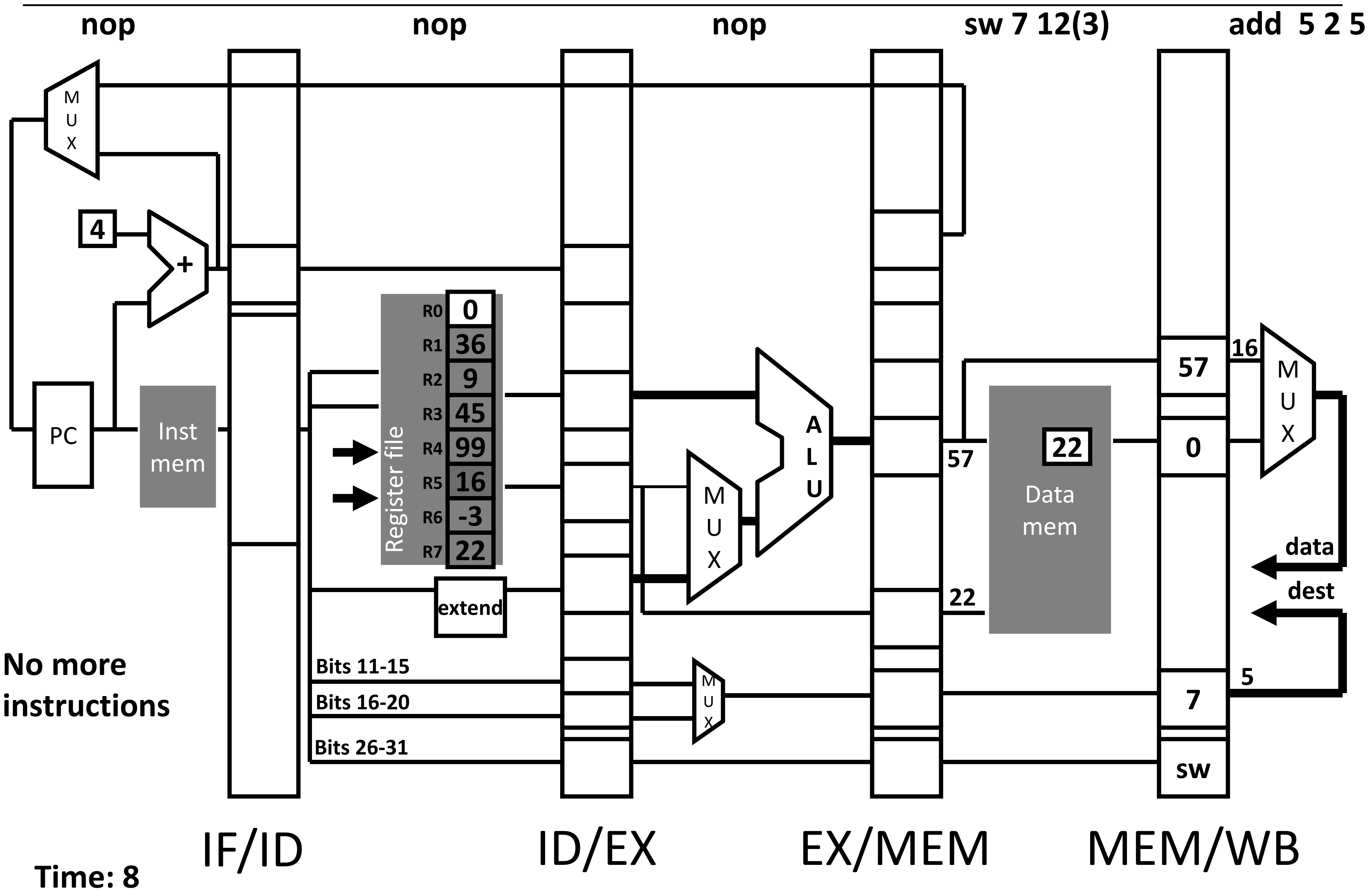
Time: 6



No more instructions

IF/ID ID/EX EX/MEM MEM/WB

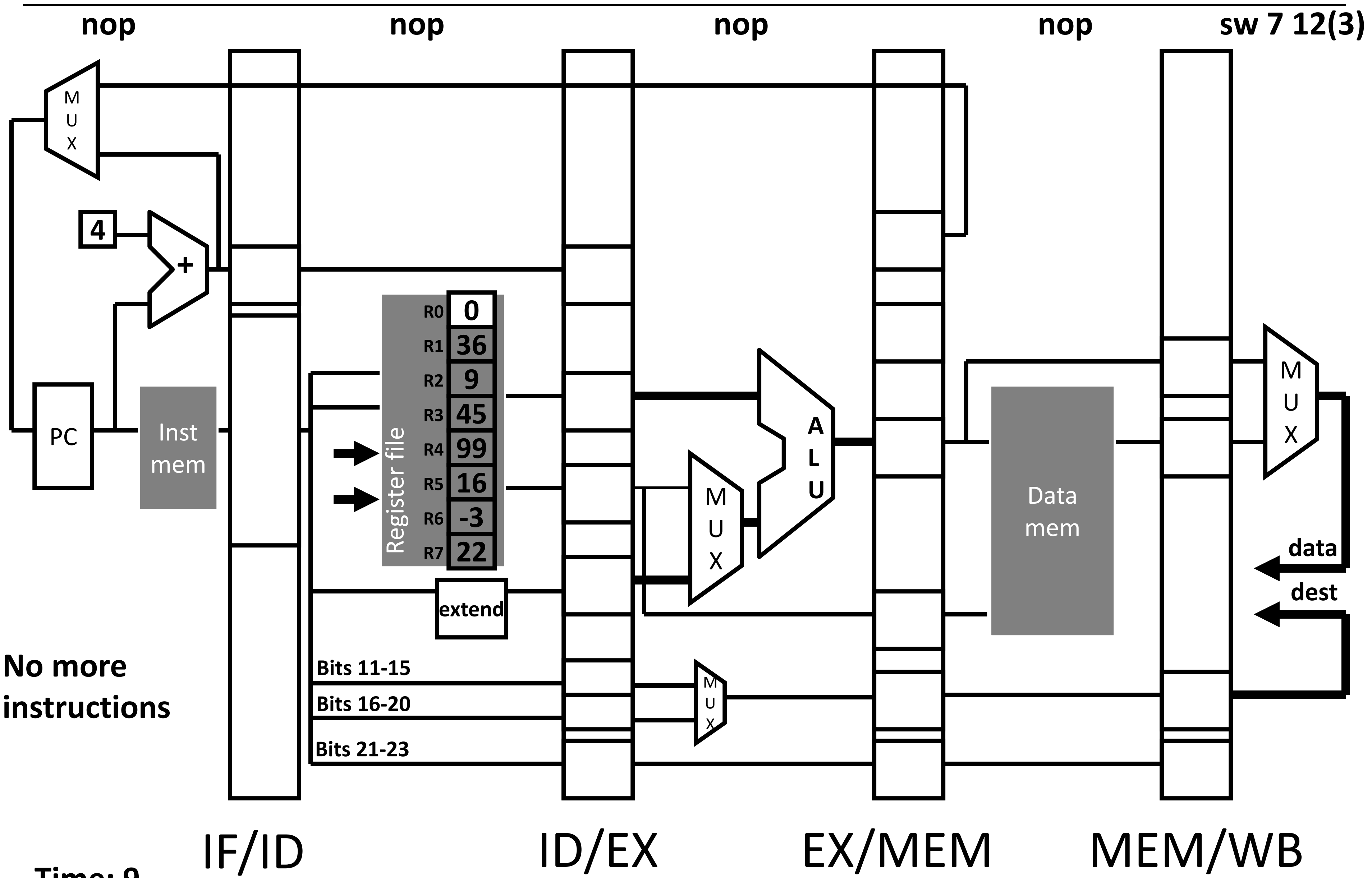
Time: 7



No more instructions

Time: 8

Slides thanks to Sally McKee



No more instructions

Time: 9

Takeaway

Pipelining is a powerful technique to mask latencies and increase throughput

- Logically, instructions execute one at a time
- Physically, instructions execute in parallel
  - Instruction level parallelism

Abstraction promotes decoupling

- Interface (ISA) vs. implementation (Pipeline)

Next Goal

What about data dependencies (also known as a data hazard in a pipelined processor)?

i.e. add r3, r1, r2

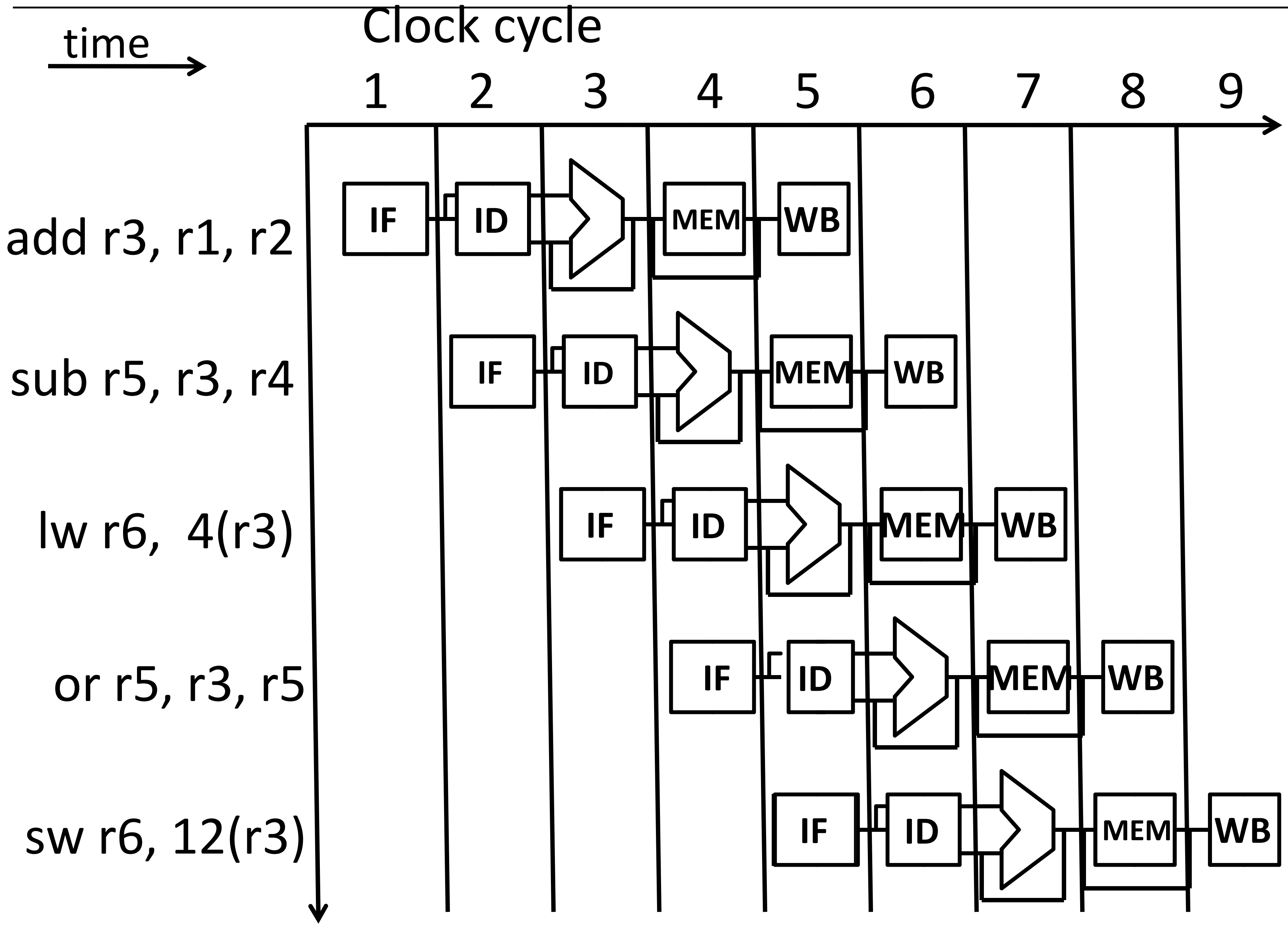
    sub r5, r3, r4

Data Hazards

Data Hazards

- register file reads occur in stage 2 (ID)
- register file writes occur in stage 5 (WB)
- next instructions may read values about to be written

Data Hazards



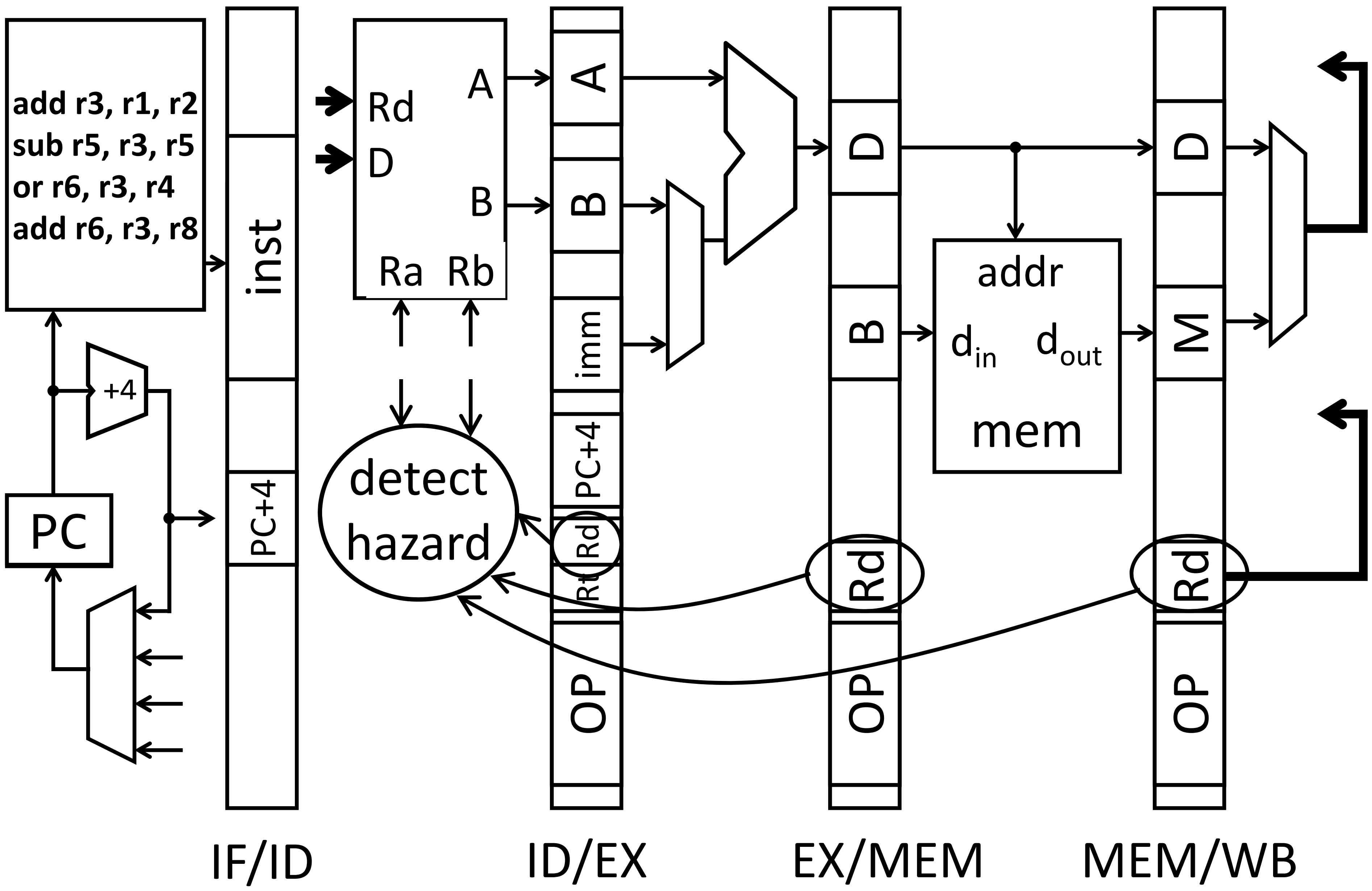
Data Hazards

Data Hazards

- register file reads occur in stage 2 (ID)
- register file writes occur in stage 5 (WB)
- next instructions may read values about to be written

How to detect?

Detecting Data Hazards



Takeaway

Data hazards occur when a operand (register) depends on the result of a previous instruction that may not be computed yet. A pipelined processor needs to detect data hazards.

Next Goal

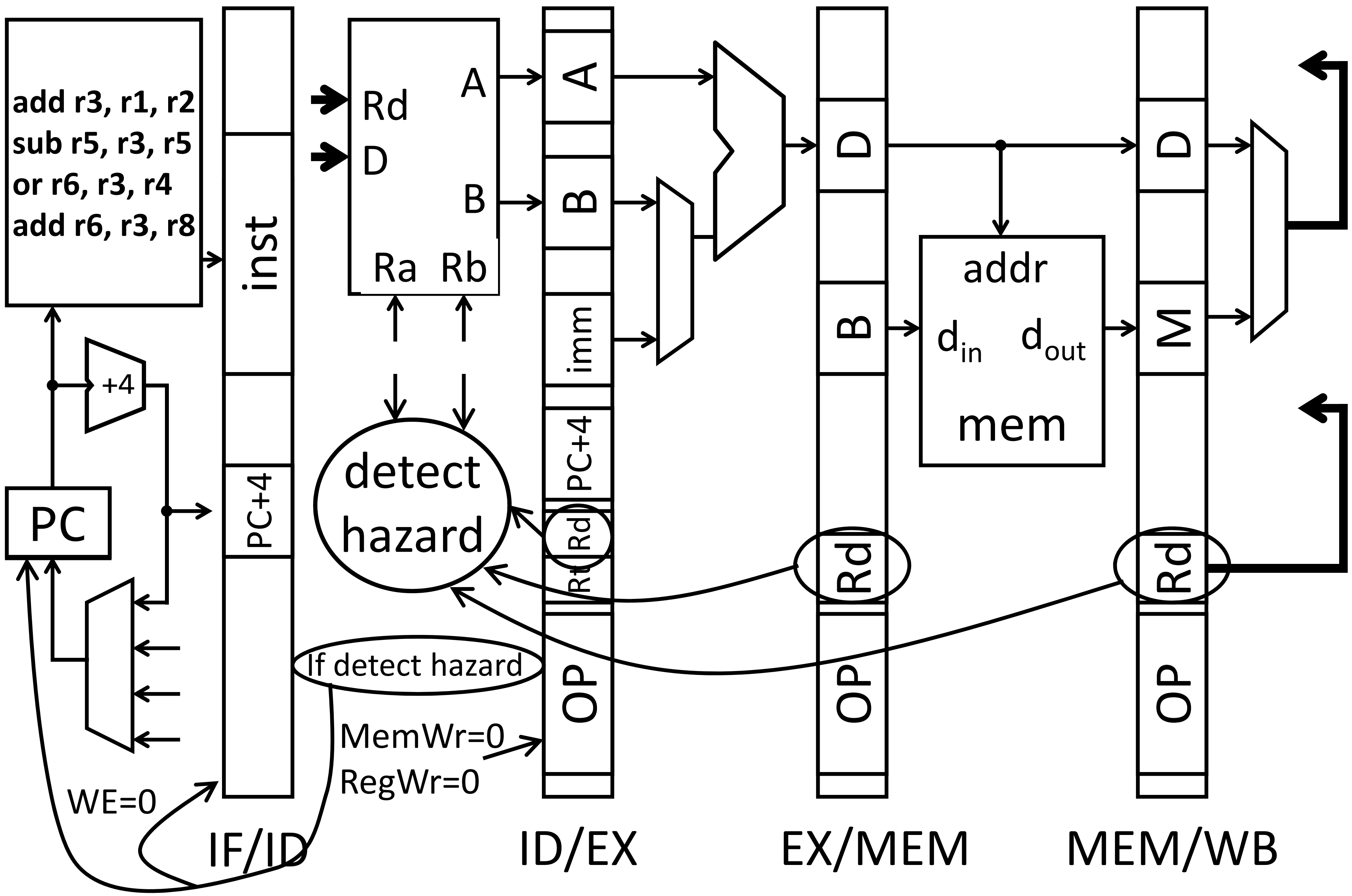
What to do if data hazard detected?

Stalling

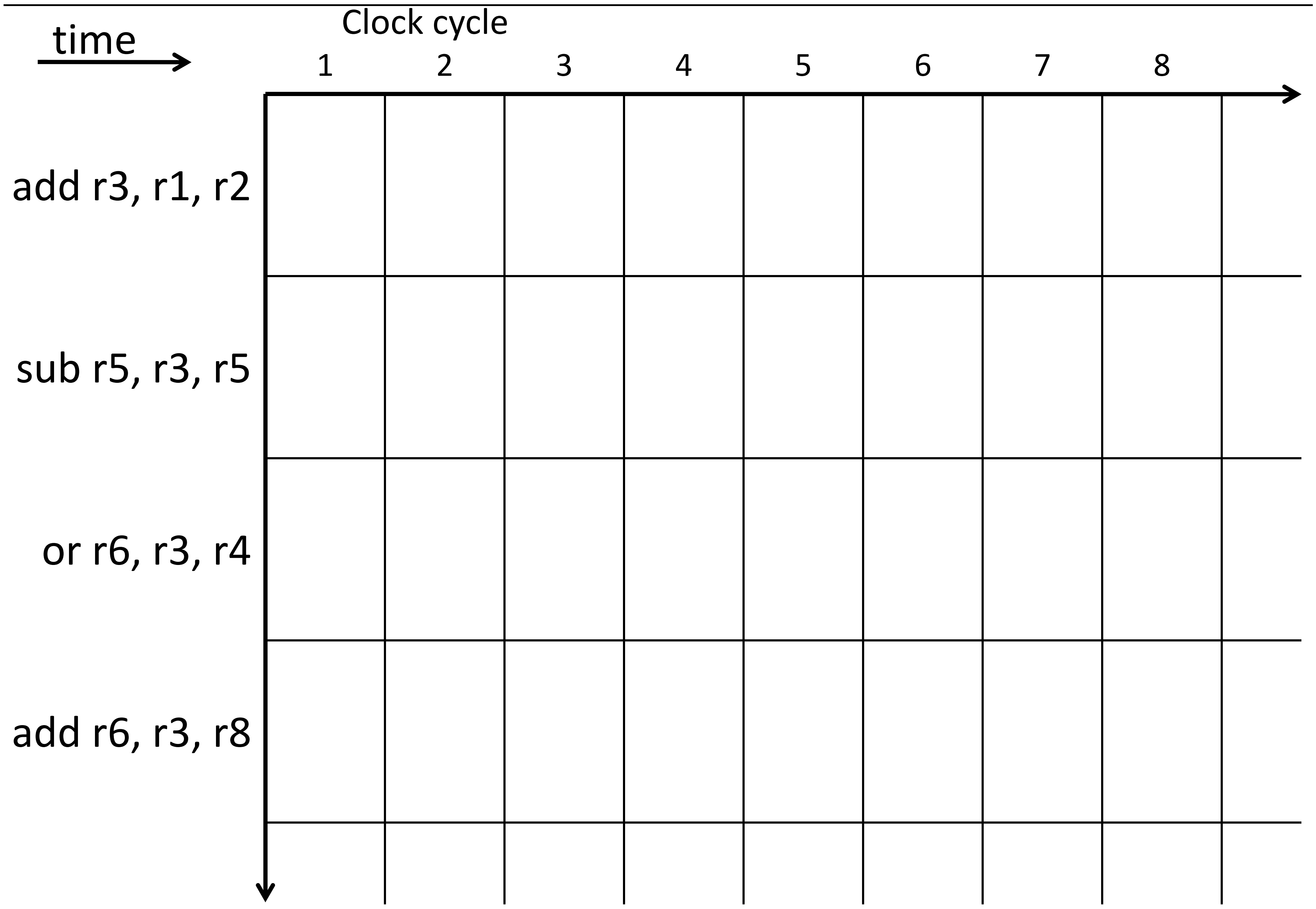
How to stall an instruction in ID stage

- prevent IF/ID pipeline register update
  - stalls the ID stage instruction
- convert ID stage instr into nop for later stages
  - innocuous "bubble" passes through pipeline
- prevent PC update
  - stalls the next (IF stage) instruction

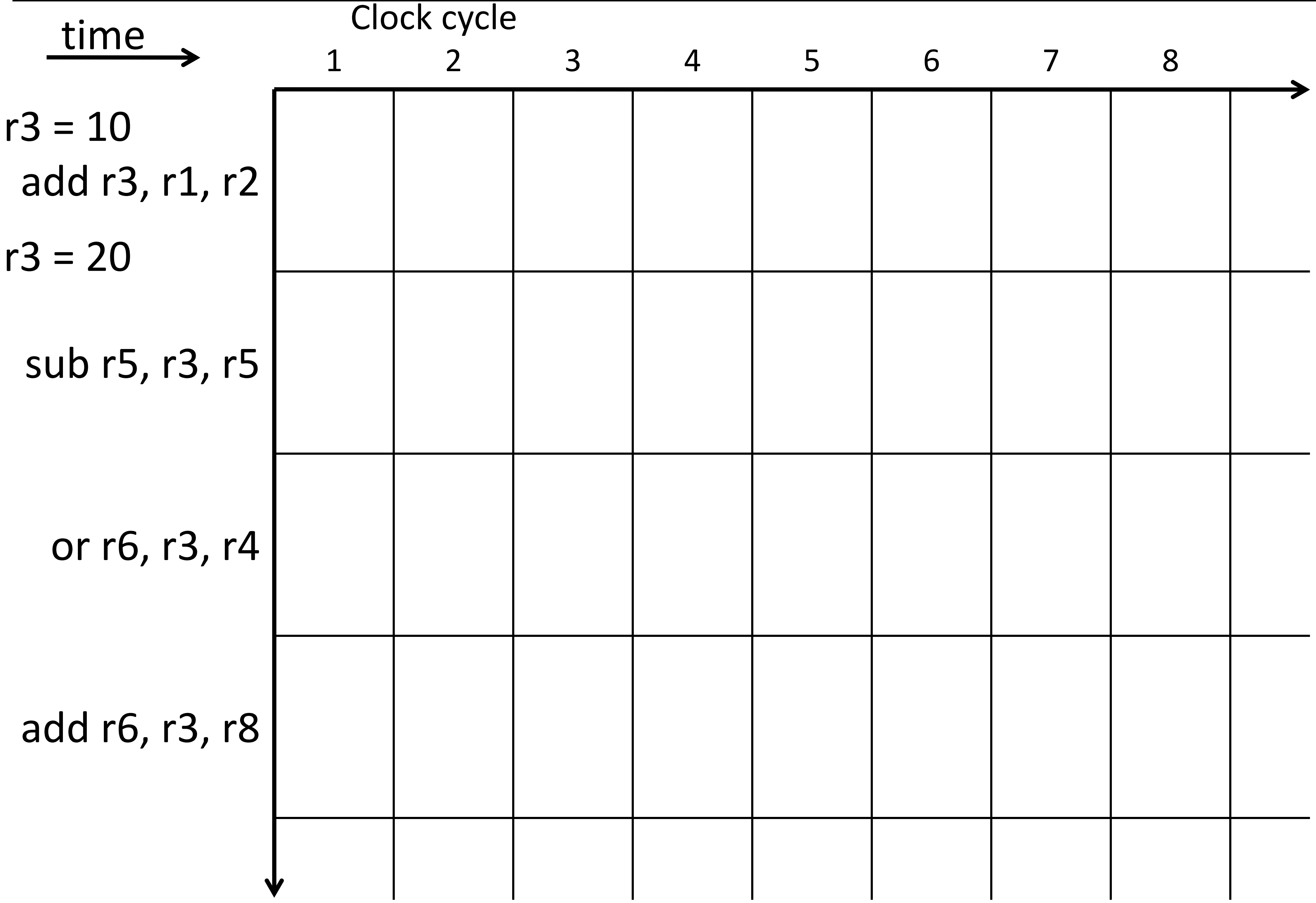
Detecting Data Hazards



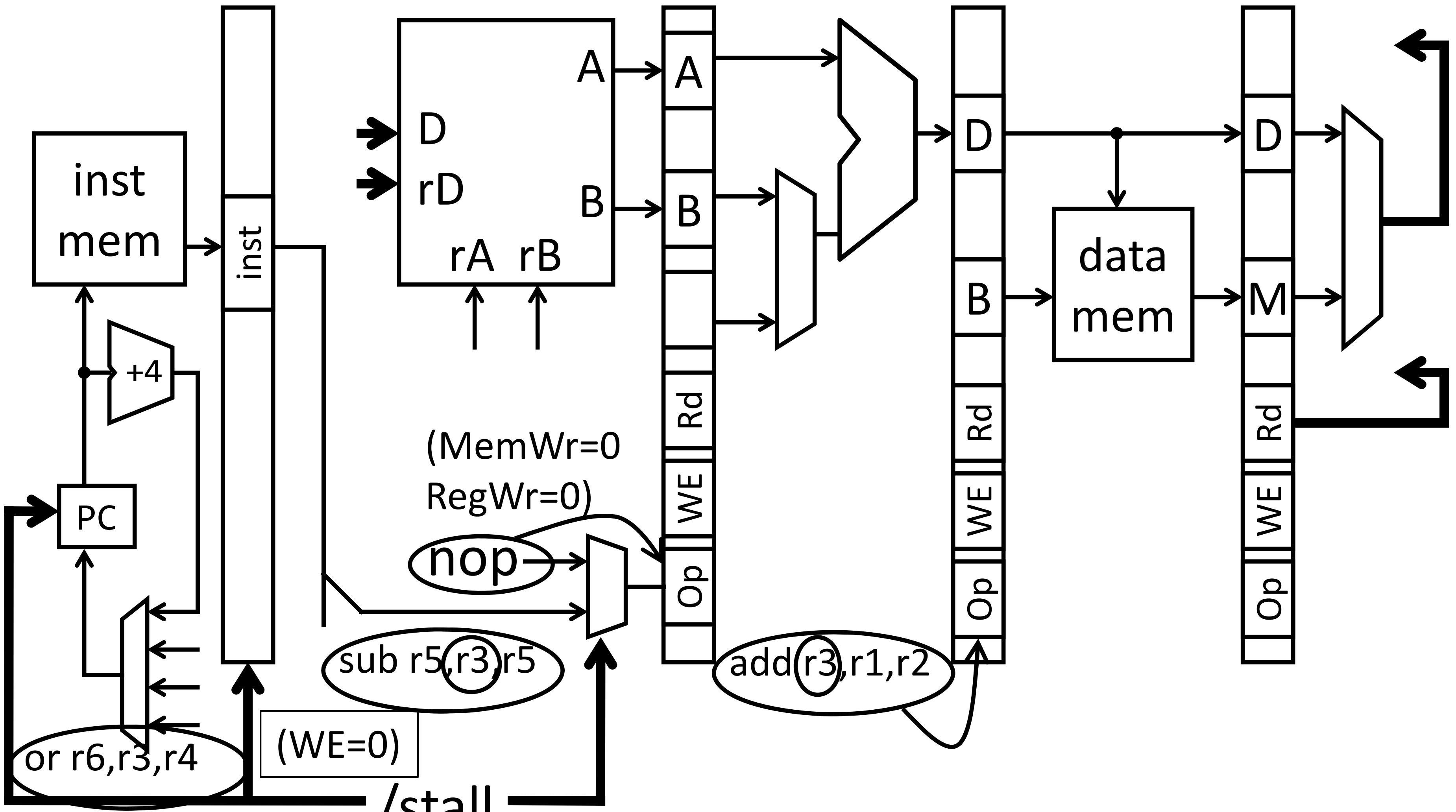
Stalling


time
Clock cycle
1
2
3
4
5
6
7
8
add r3, r1, r2
sub r5, r3, r5
or r6, r3, r4
add r6, r3, r8

Stalling

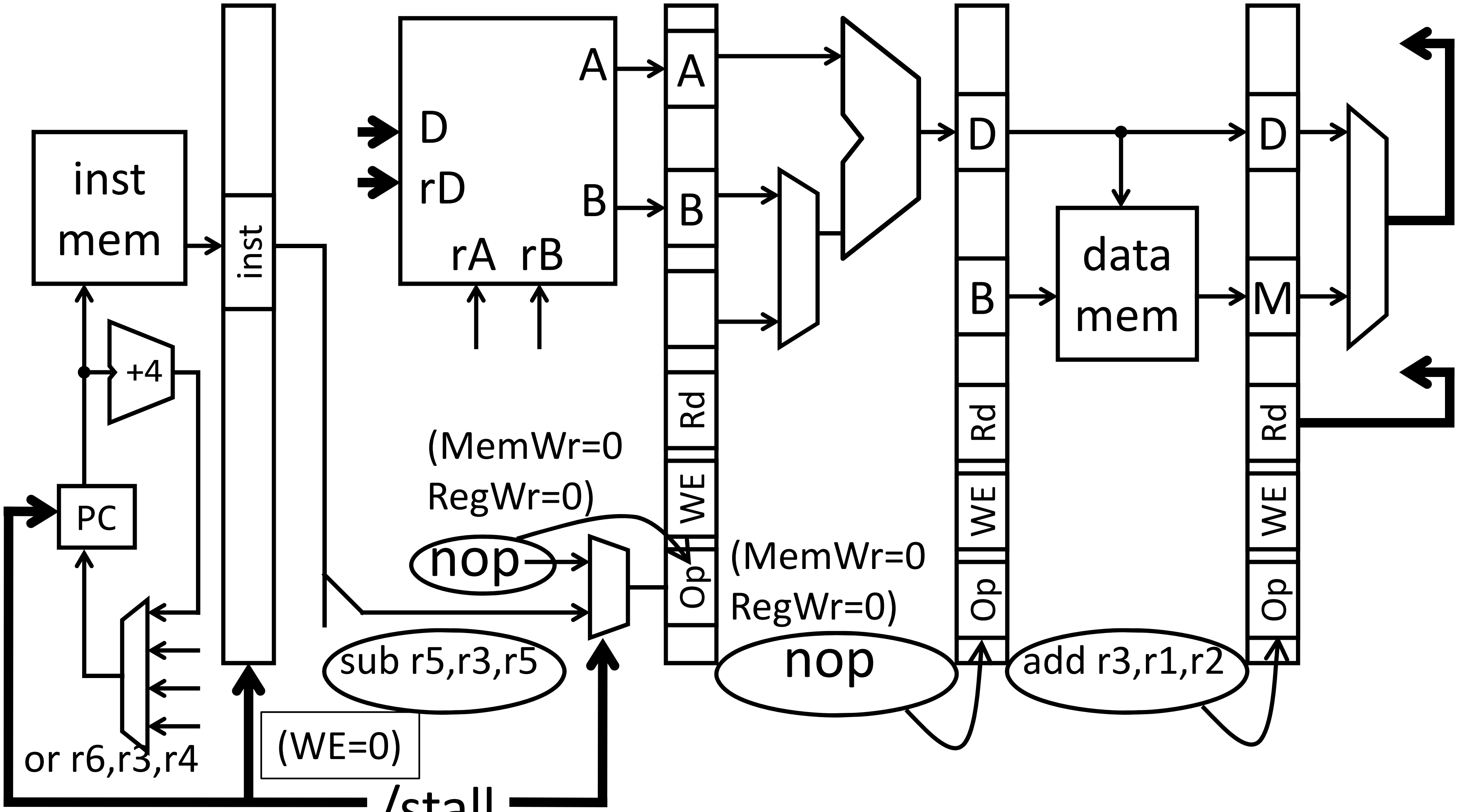

time
Clock cycle
1
2
3
4
5
6
7
8
r3 = 10
add r3, r1, r2
r3 = 20
sub r5, r3, r5
or r6, r3, r4
add r6, r3, r8

Stalling



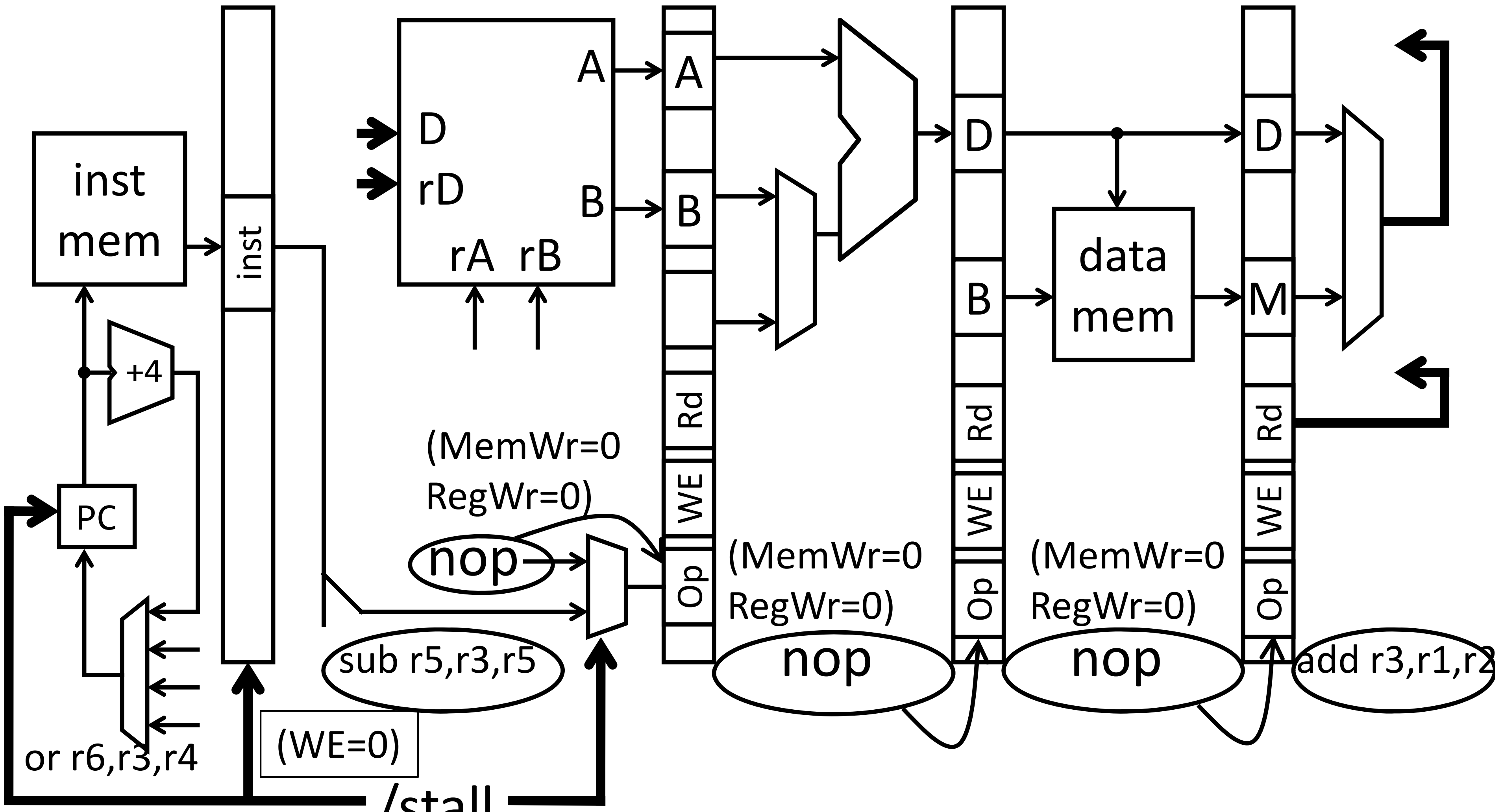
NOP = If(IF/ID.rA ≠ 0 &&
(IF/ID.rA==ID/Ex.Rd
IF/ID.rA==Ex/M.Rd
IF/ID.rA==M/W.Rd))

Stalling



NOP = If(IF/ID.rA ≠ 0 &&
(IF/ID.rA==ID/Ex.Rd
IF/ID.rA==Ex/M.Rd
IF/ID.rA==M/W.Rd))

Stalling



NOP = If(IF/ID.rA ≠ 0 &&
(IF/ID.rA==ID/Ex.Rd
IF/ID.rA==Ex/M.Rd
IF/ID.rA==M/W.Rd))

Stalling

How to stall an instruction in ID stage

- prevent IF/ID pipeline register update
  - stalls the ID stage instruction
- convert ID stage instr into nop for later stages
  - innocuous “bubble” passes through pipeline
- prevent PC update
  - stalls the next (IF stage) instruction

Takeaway

Data hazards occur when a operand (register) depends on the result of a previous instruction that may not be computed yet. A pipelined processor needs to detect data hazards.

Stalling, preventing a dependent instruction from advancing, is one way to resolve data hazards. Stalling introduces NOPs (“bubbles”) into a pipeline. Introduce NOPs by (1) preventing the PC from updating, (2) preventing writes to IF/ID registers from changing, and (3) preventing writes to memory and register file. Bubbles in pipeline significantly decrease performance.

Next Goal: Resolving Data Hazards via Forwarding

What to do if data hazard detected?

A) Wait/Stall

B) Reorder in Software (SW)

[✓ C) Forward/Bypass]

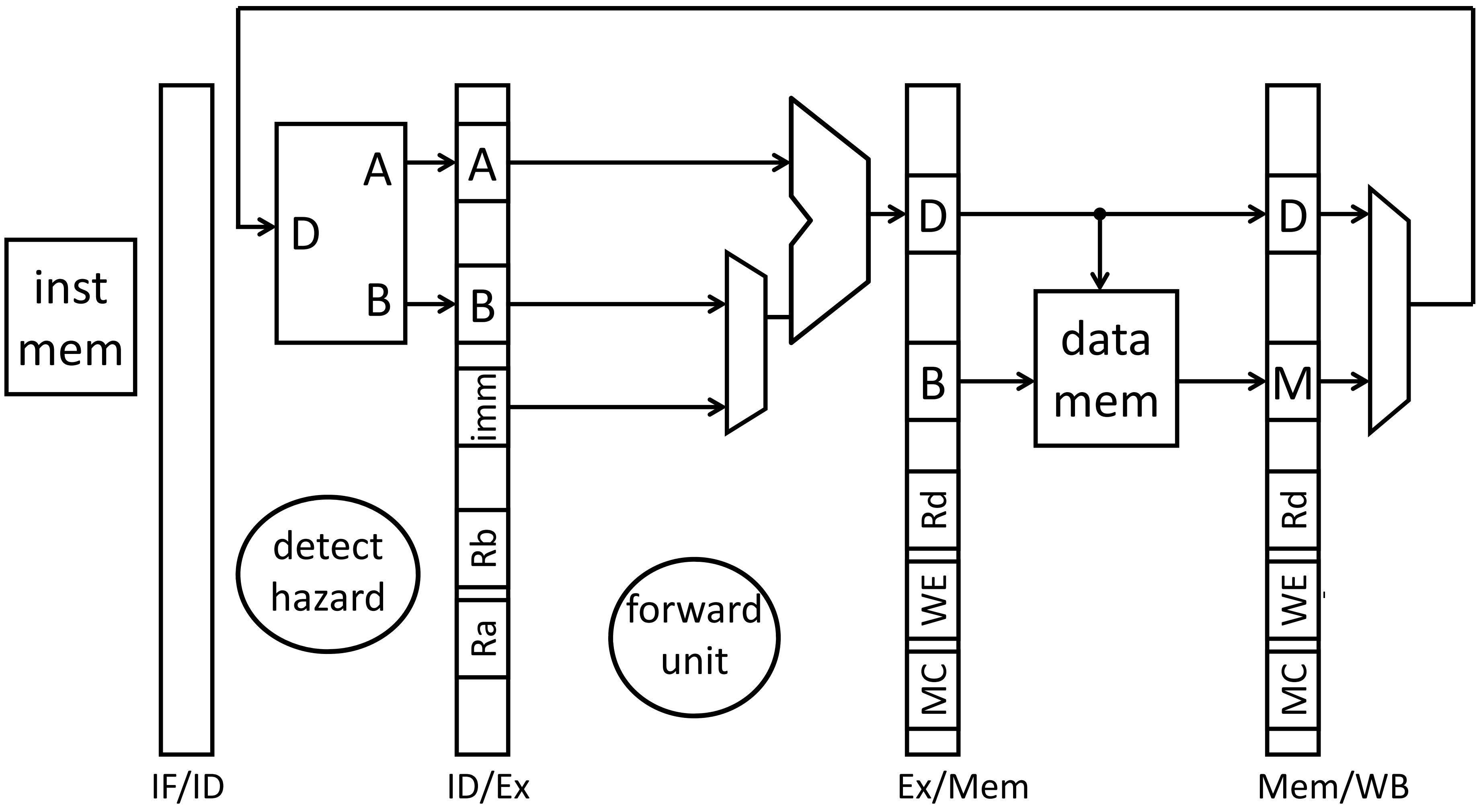
Forwarding

Forwarding bypasses some pipelined stages forwarding a result to a dependent instruction operand (register).

Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M$\rightarrow$Ex)
- Forwarding from Mem/WB register to Ex stage (W$\rightarrow$Ex)
- RegisterFile Bypass

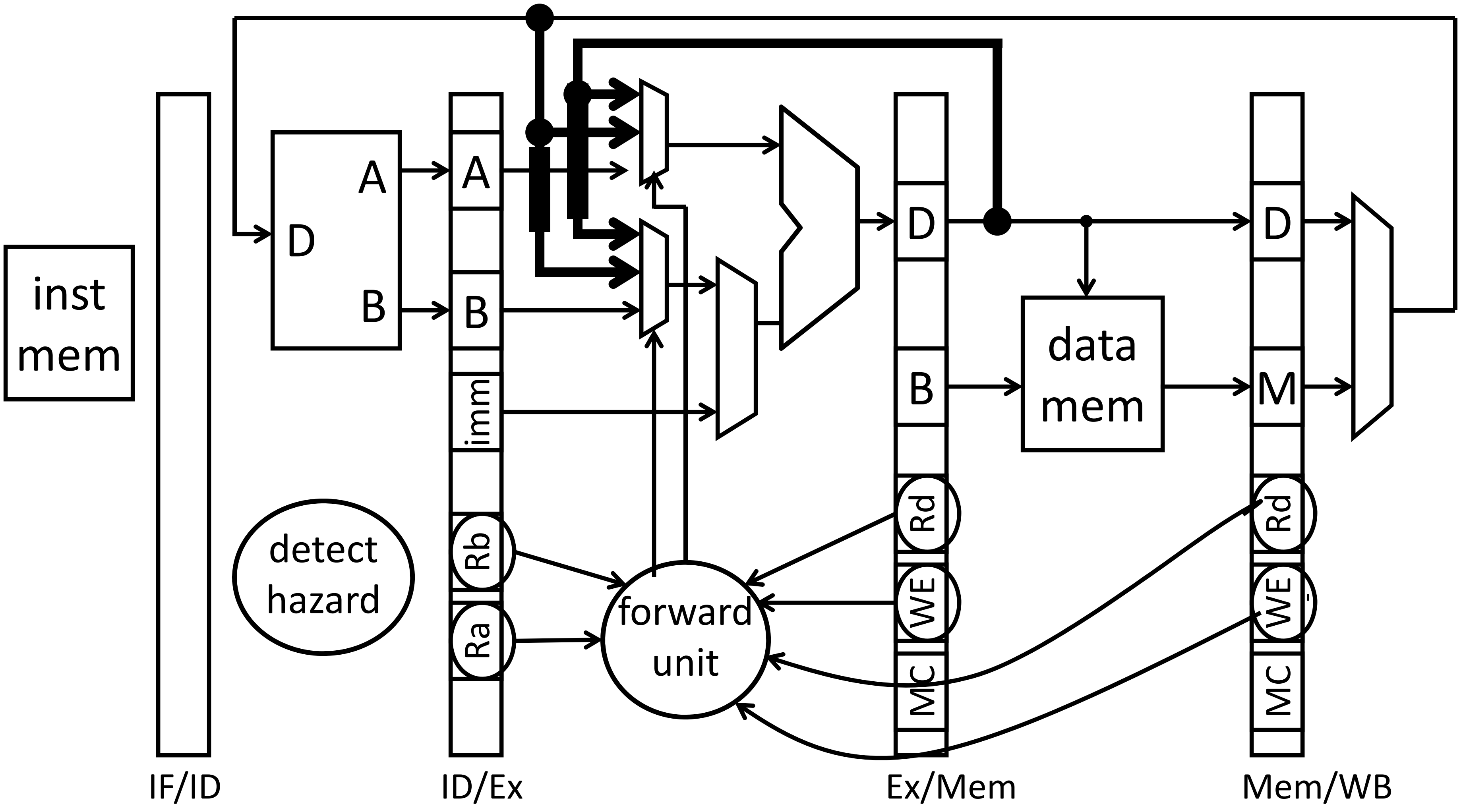
Forwarding Datapath



Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M$\rightarrow$Ex)
- Forwarding from Mem/WB register to Ex stage (W $\rightarrow$ Ex)
- RegisterFile Bypass

Forwarding Datapath



Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage ($M \rightarrow Ex$)
- Forwarding from Mem/WB register to Ex stage ($W \rightarrow Ex$)
- RegisterFile Bypass

Forwarding Datapath 1

Ex/MEM to EX Bypass

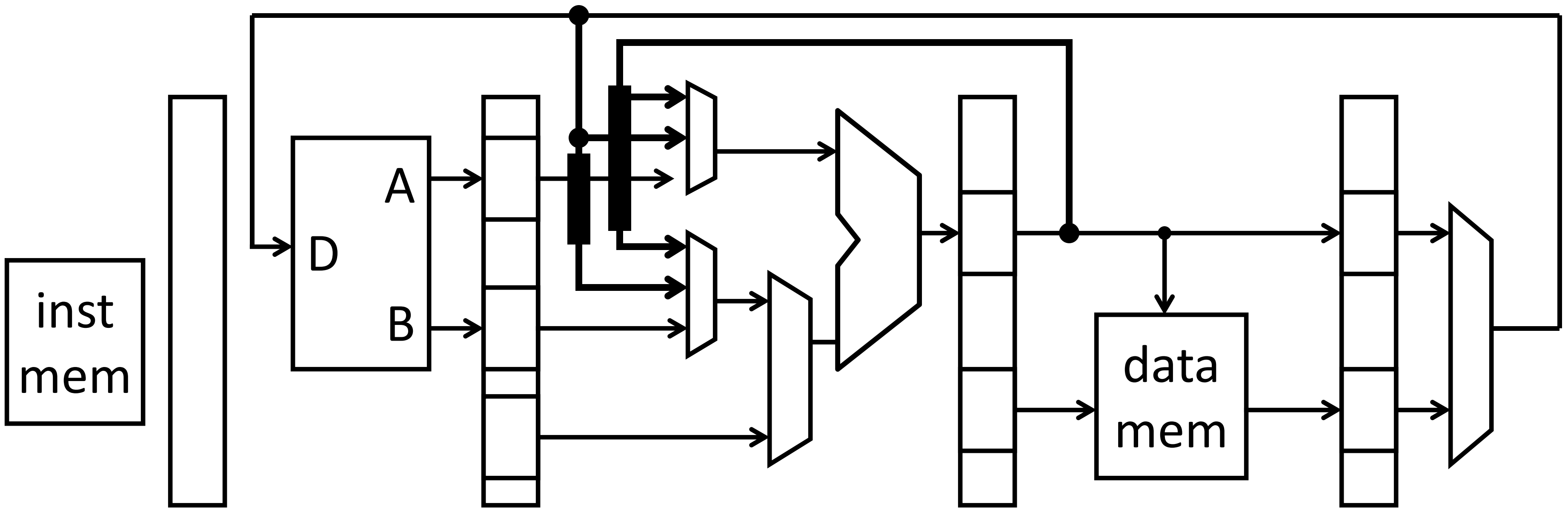
- EX needs ALU result that is still in MEM stage
- Resolve:

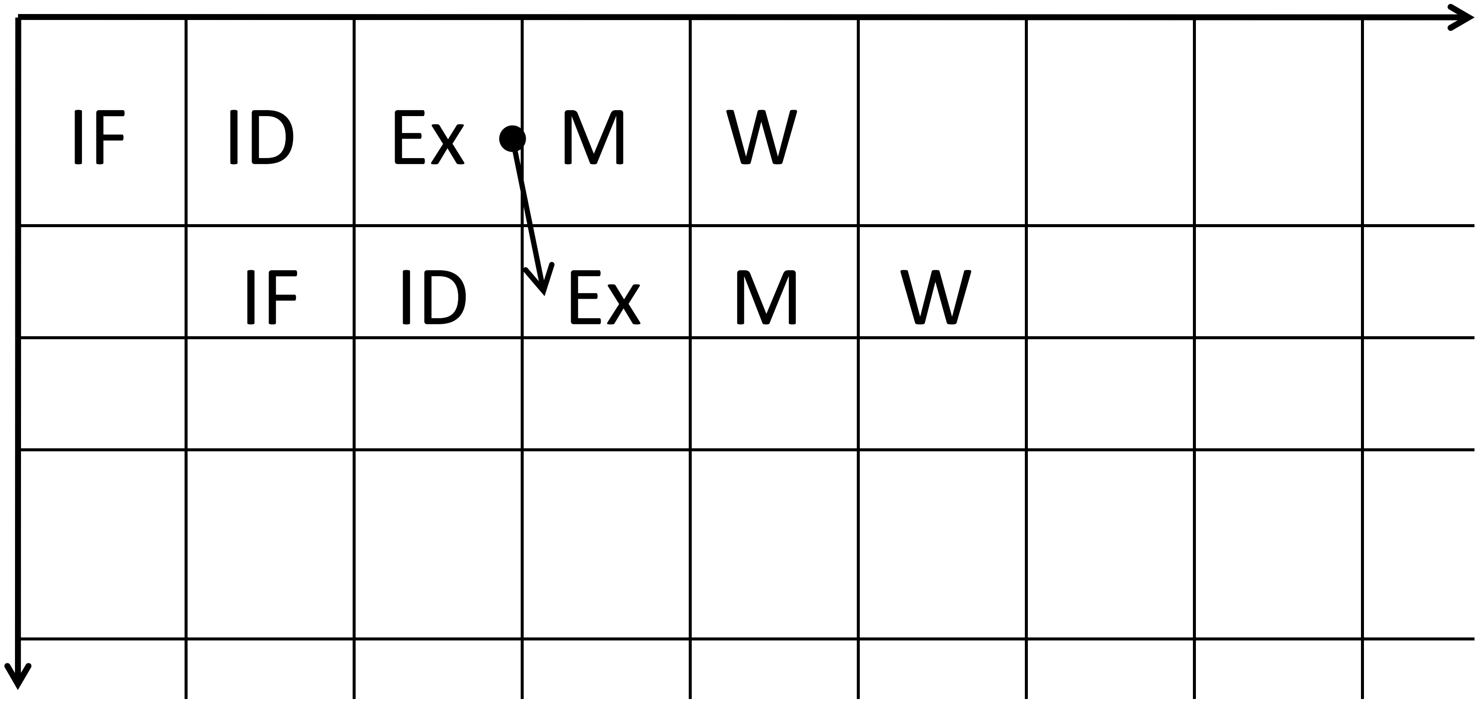
  Add a bypass from EX/MEM.D to start of EX

How to detect? Logic in Ex Stage:

```
forward = (Ex/M.WE && EX/M.Rd != 0 &&
           ID/Ex.Ra == Ex/M.Rd)
        || (same for Rb)
```

Forwarding Datapath 1





add r3, r1, r2	IF	ID	Ex	M	W			
sub r5, r3, r1		IF	ID	Ex	M	W		

Forwarding Datapath 2

Mem/WB to EX Bypass

- EX needs value being written by WB
- Resolve:

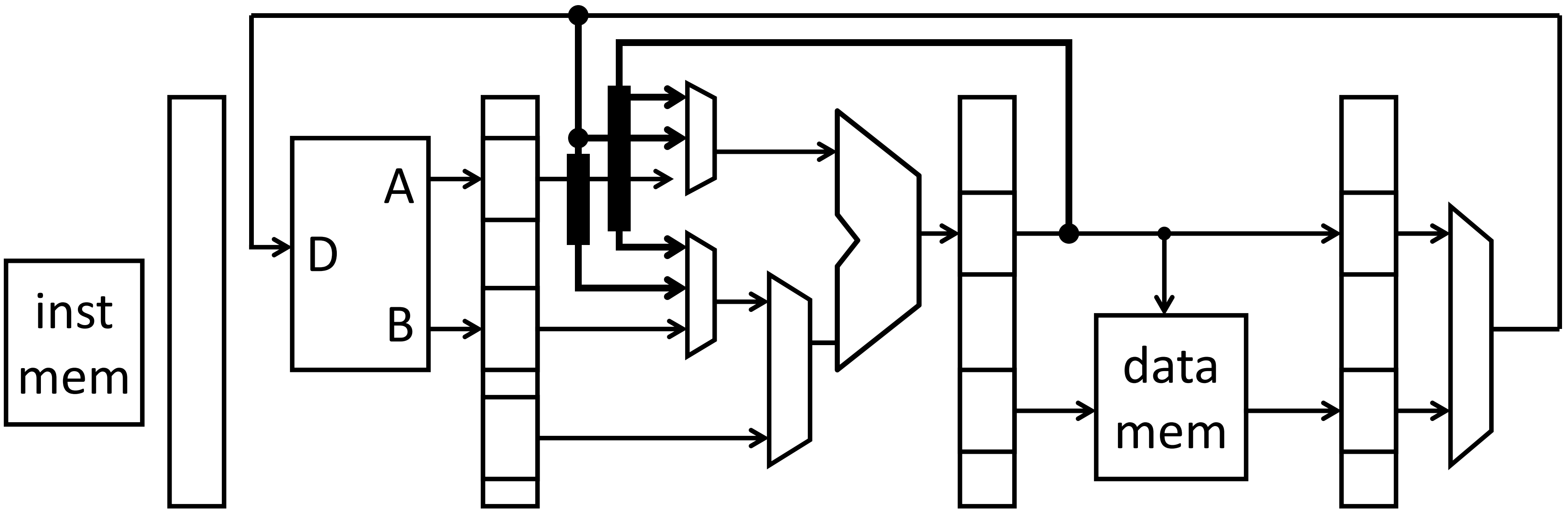
  Add bypass from WB final value to start of EX

How to detect? Logic in Ex Stage:

```
forward = (M/WB.WE && M/WB.Rd != 0 &&
              ID/Ex.Ra == M/WB.Rd &&
           not (ID/Ex.WE && Ex/M.Rd != 0 &&
                ID/Ex.Ra == Ex/M.Rd)
        || (same for Rb)
```

Check pg. 369

Forwarding Datapath 2



add r3, r1, r2	IF	ID	Ex	M	W				
sub r5, r3, r1		IF	ID	Ex	M	W			
or r6, r3, r4			IF	ID	Ex	M	W		

Register File Bypass

Register File Bypass

- Reading a value that is currently being written

Detect:

((Ra == MEM/WB.Rd) or (Rb == MEM/WB.Rd))
and (WB is writing a register)

Resolve:

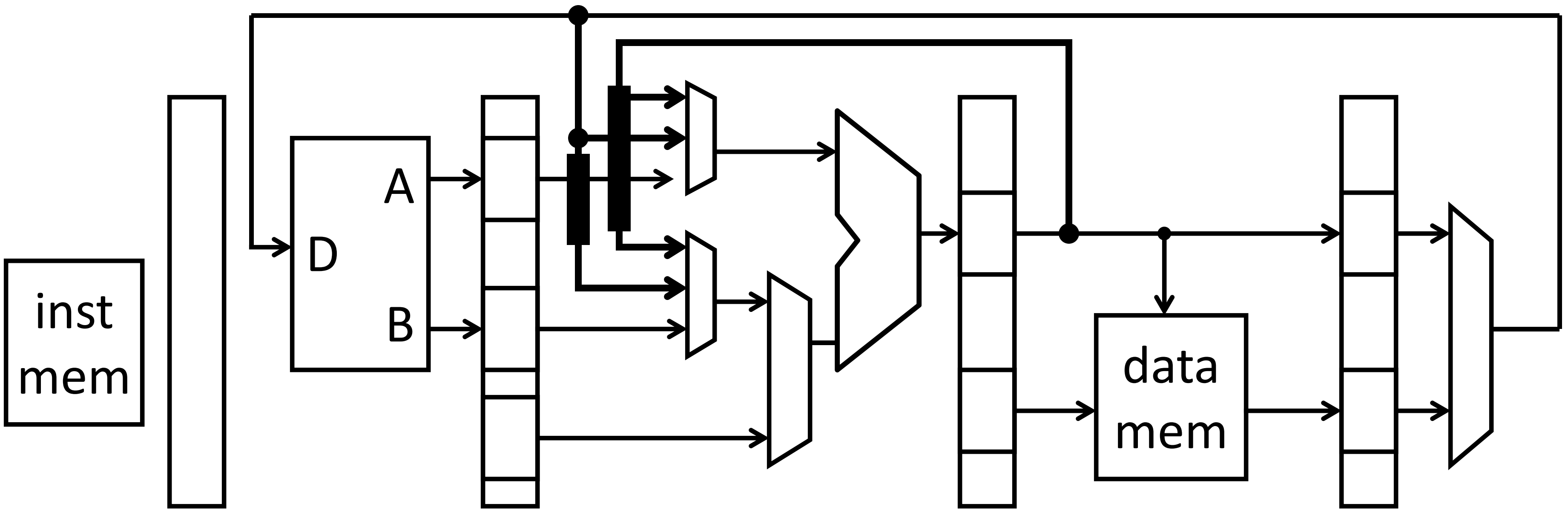
Add a bypass around register file (WB to ID)

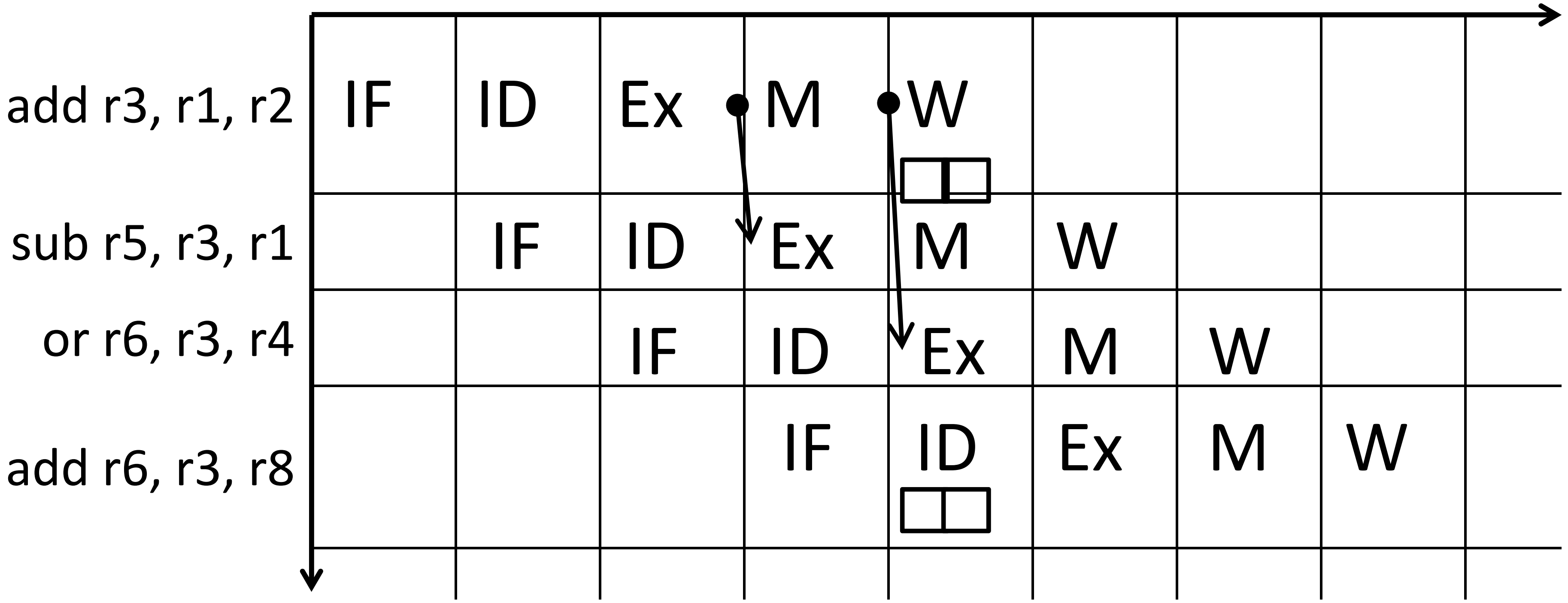
Better: (Hack) just negate register file clock

– writes happen at end of first half of each clock cycle

– reads happen during second half of each clock cycle

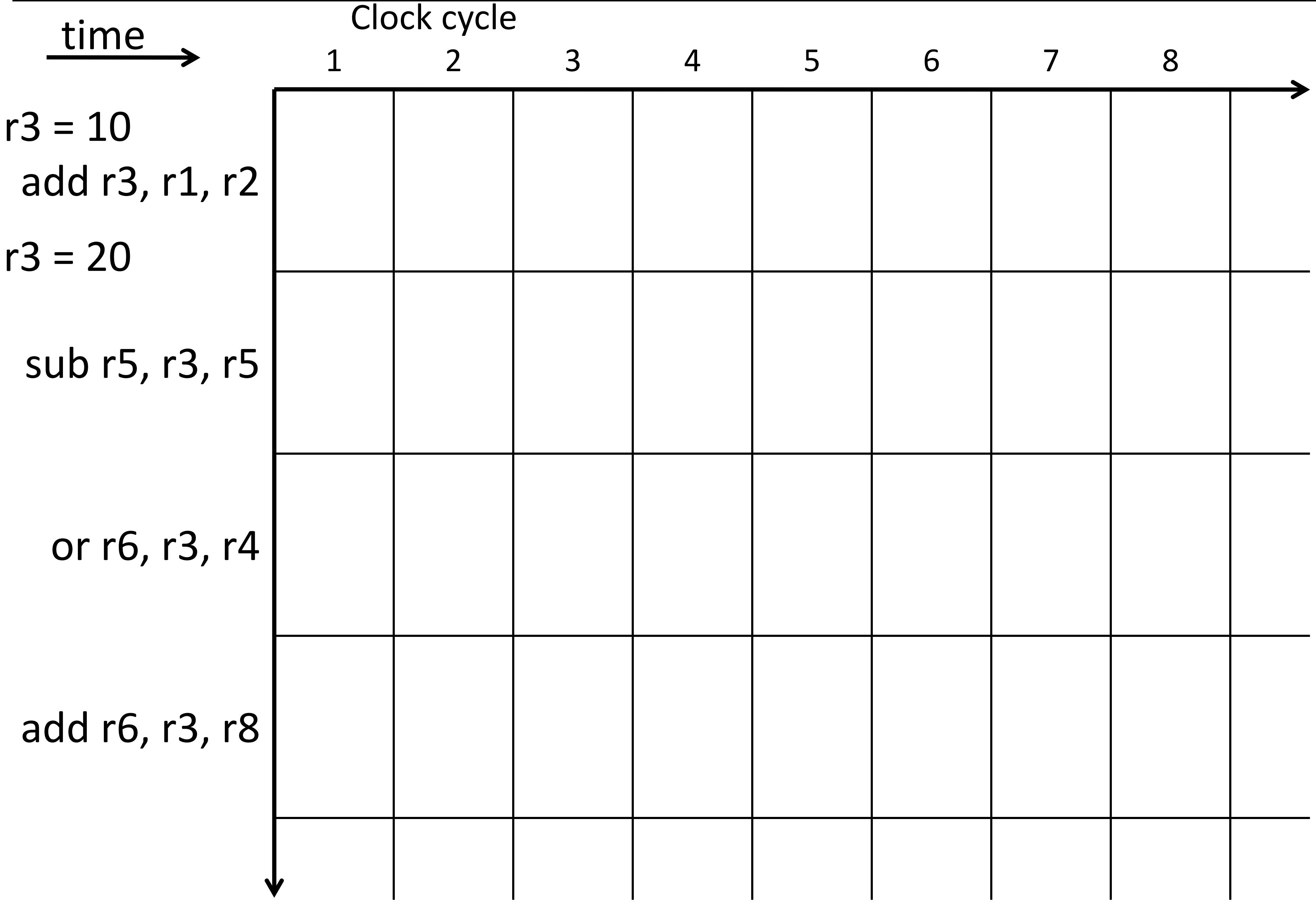
Register File Bypass





add r3, r1, r2	IF	ID	Ex	M	W			
sub r5, r3, r1		IF	ID	Ex	M	W		
or r6, r3, r4			IF	ID	Ex	M	W	
add r6, r3, r8				IF	ID	Ex	M	W

Forwarding Example



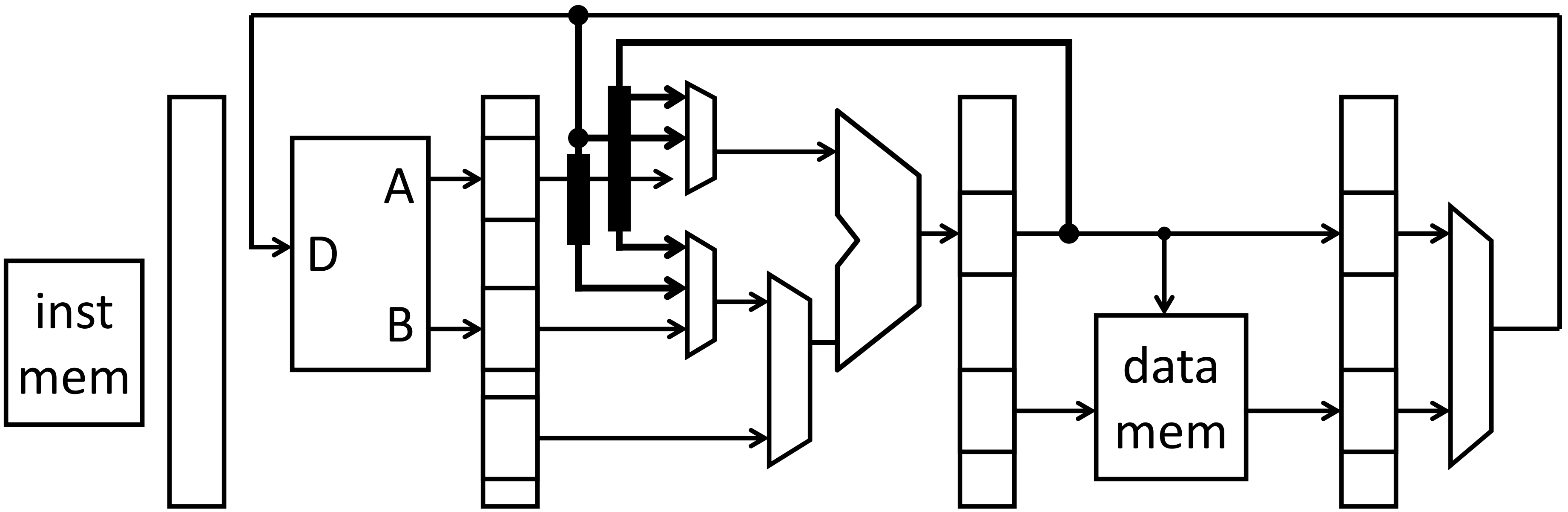
Forwarding Example 2

time →

Clock cycle

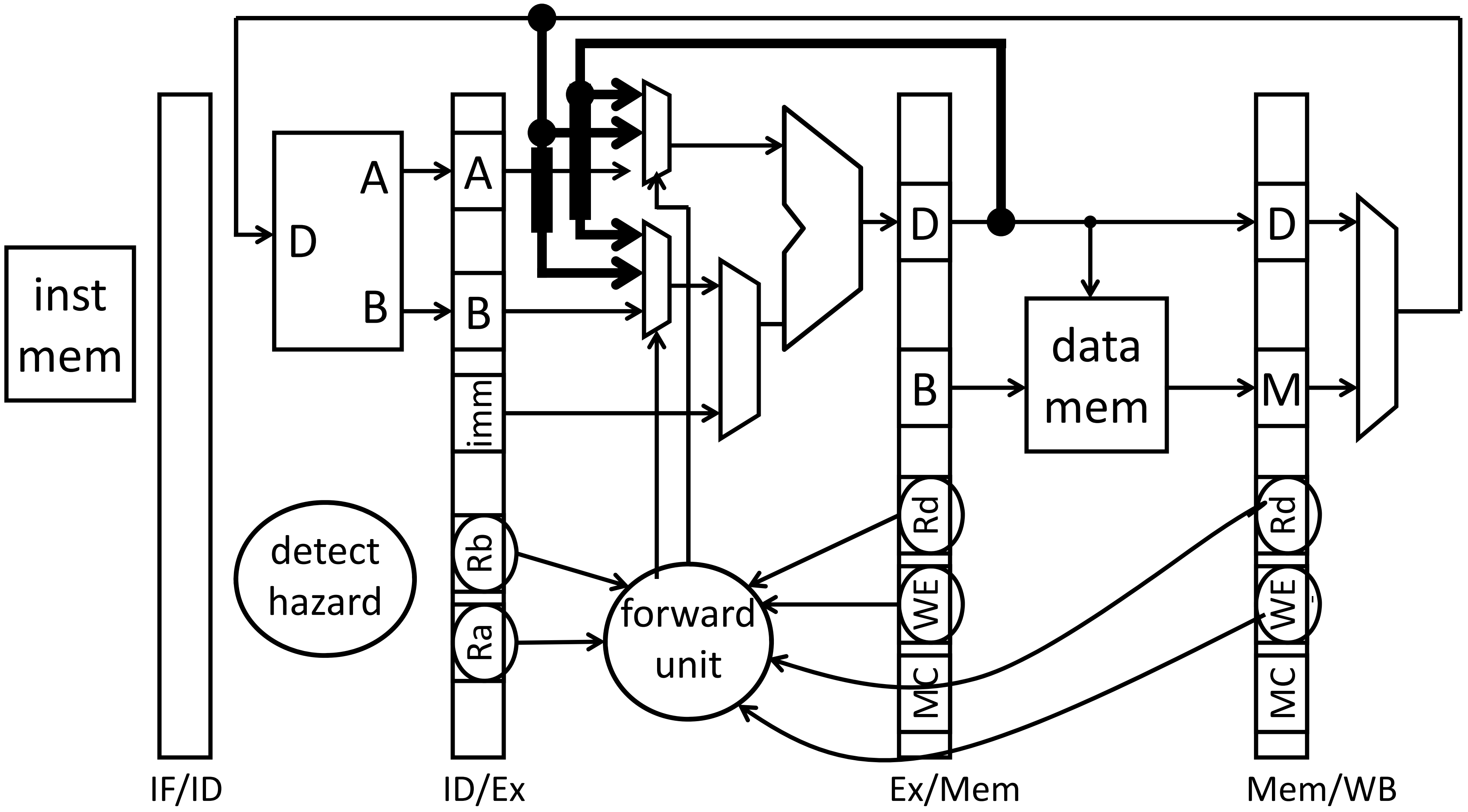
	1	2	3	4	5	6	7	8	
add r3, r1, r2	IF	ID	Ex	M	W				
sub r5, r3, r4		IF	ID	Ex	M	W			
lw r6, 4(r3)									
or r5, r3, r5									
sw r6, 12(r3)									

Tricky Example



add r1, r1, r2								
SUB r1, r1, r3								
OR r1, r4, r1								

Forwarding Datapath



Three types of forwarding/bypass

- Forwarding from Ex/Mem registers to Ex stage (M$\rightarrow$Ex)
- Forwarding from Mem/WB register to Ex stage (W $\rightarrow$ Ex)
- Register File Bypass

Takeaway

Data hazards occur when a operand (register) depends on the result of a previous instruction that may not be computed yet. A pipelined processor needs to detect data hazards.

Stalling, preventing a dependent instruction from advancing, is one way to resolve data hazards. Stalling introduces NOPs (“bubbles”) into a pipeline. Introduce NOPs by (1) preventing the PC from updating, (2) preventing writes to IF/ID registers from changing, and (3) preventing writes to memory and register file. Bubbles (nops) in pipeline significantly decrease performance.

Forwarding bypasses some pipelined stages forwarding a result to a dependent instruction operand (register). Better performance than stalling.

Administrivia

Prelim1: next Tuesday, February 26th in evening

- Time: We will start at 7:30pm sharp, so come early
- Prelim Review: This Thur 6-8pm in Upson B14 and Fri, 5-7pm in Phillips 203

- Closed Book
  - Cannot use electronic device or outside material
- Practice prelims are online in CMS
- Material covered everything up to end of this week
  - Appendix C (logic, gates, FSMs, memory, ALUs)
  - Chapter 4 (pipelined [and non-pipeline] MIPS processor with hazards)
  - Chapters 2 (Numbers / Arithmetic, simple MIPS instructions)
  - Chapter 1 (Performance)
  - HW1, HW2, Lab0, Lab1, Lab2

Administrivia

HW2 is due tomorrow

- ***Fill out Survey online***. Receive credit/points on homework for survey:
- Should have received email from Kathryn Dimiduk
- Survey is anonymous

Project1 (PA1) due week after prelim

- Continue working diligently. Use design doc momentum

Save your work!

- ***Save often***. Verify file is non-zero. Periodically save to Dropbox, email.
- Beware of MacOSX 10.5 (leopard) and 10.6 (snow-leopard)

Use your resources

- Lab Section, Piazza.com, Office Hours, Homework Help Session,
- Class notes, book, Sections, CSUGLab

Administrivia

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2013sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):

  - Tuesday, February 26th
  - Thursday, March 28th
  - Thursday, April 25th

Schedule is subject to change

Collaboration, Late, Re-grading Policies

“Black Board” Collaboration Policy

- Can discuss approach together on a “black board”
- Leave and write up solution independently
- Do not copy solutions

Late Policy

- Each person has a total of ***four*** “slip days”
- Max of ***two*** slip days for any individual assignment
- Slip days deducted first for *any* late assignment,
  cannot selectively apply slip days
- For projects, slip days are deducted from all partners
- 25% deducted per day late after slip days are exhausted

Regrade policy

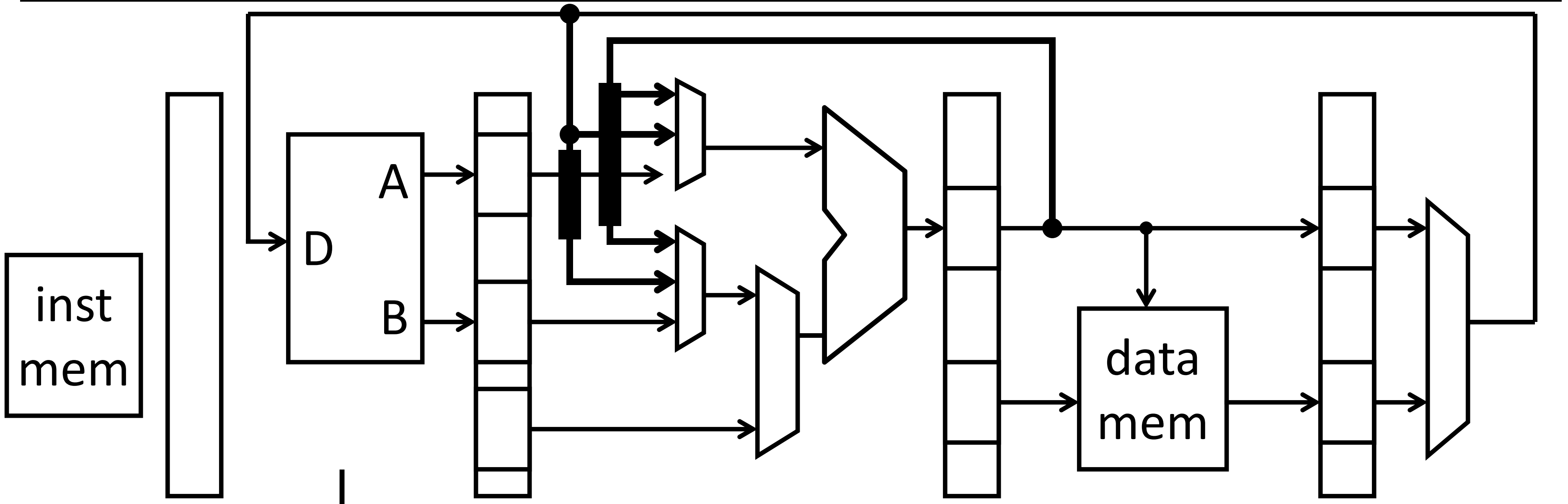
- Submit written request to lead TA,
  and lead TA will pick a different grader
- Submit another written request,
  lead TA will regrade directly
- Submit yet another written request for professor to regrade.

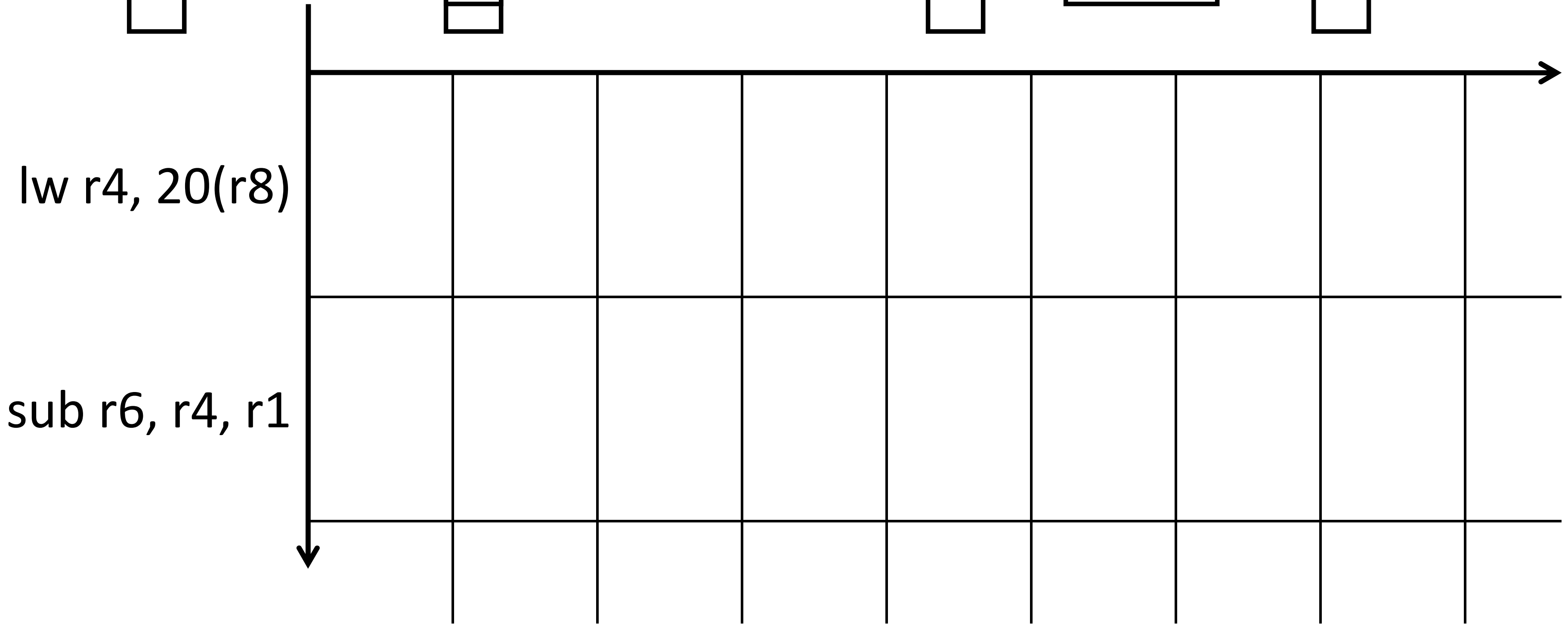
Quiz

Find all hazards, and say how they are resolved:

```
add r3, r1, r2
sub r3, r2, r1
nand r4, r3, r1
or  r0, r3, r4
xor r1, r4, r3
sb  r4, 1(r0)
```

Memory Load Data Hazard





Resolving Memory Load Hazard

Load Data Hazard

- Value not available until WB stage
- So: next instruction can’t proceed if hazard detected

Resolution:

- MIPS 2000/3000: one delay slot
  - ISA says results of loads are not available until one cycle later
  - Assembler inserts nop, or reorders to fill delay slot
- MIPS 4000 onwards: stall
  - But really, programmer/compiler reorders to avoid stalling in the load delay slot

Quiz 2

```
add  r3, r1, r2
nand r5, r3, r4
add  r2, r6, r3
lw  r6, 24(r3)
sw  r6, 12(r2)
```

Data Hazard Recap

Delay Slot(s)

- Modify ISA to match implementation

Stall

- Pause current and all subsequent instructions

Forward/Bypass

- Try to steal correct value from elsewhere in pipeline
- Otherwise, fall back to stalling or require a delay slot

Tradeoffs?