.

MIPS Pipeline

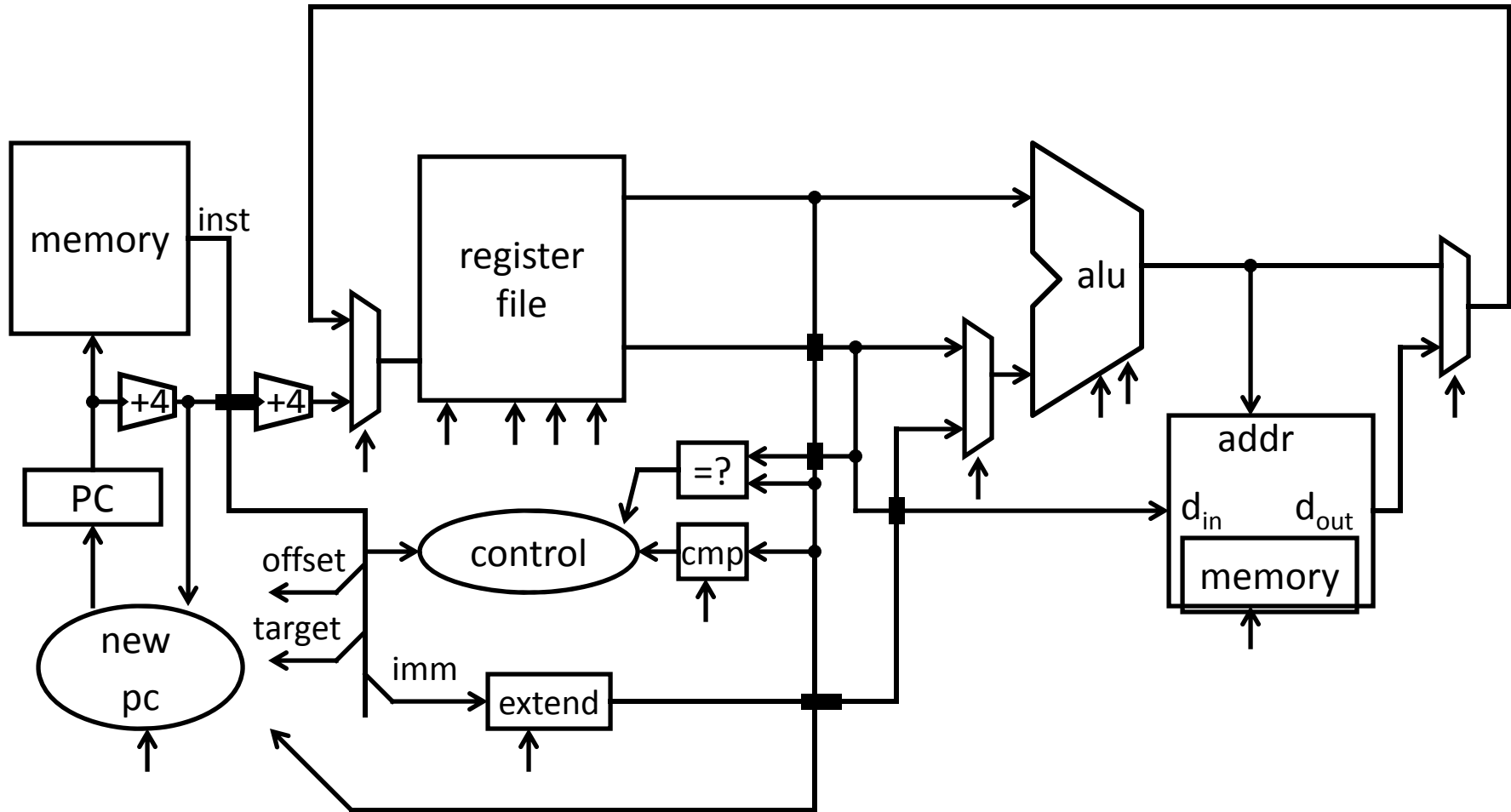**Hakim Weatherspoon**
CS 3410, Spring 2012
Computer Science
Cornell University

See P&H Chapter 4.6

A Processor

Review: Single cycle processor

What determines performance of Processor?

A) Critical Path

B) Clock Cycle Time

C) Cycles Per Instruction (CPI)

D) All of the above

E) None of the above

Review: Single Cycle Processor

Advantages

- Single Cycle per instruction make logic and clock simple

Disadvantages

- Since instructions take different time to finish, memory and functional unit are not efficiently utilized.

- Cycle time is the longest delay.

  – Load instruction

- Best possible CPI is 1

  – However, lower MIPS and longer clock period (lower clock frequency); hence, lower performance.

Review: Multi Cycle Processor

Advantages

- Better MIPS and smaller clock period (higher clock frequency)
- Hence, better performance than Single Cycle processor

Disadvantages

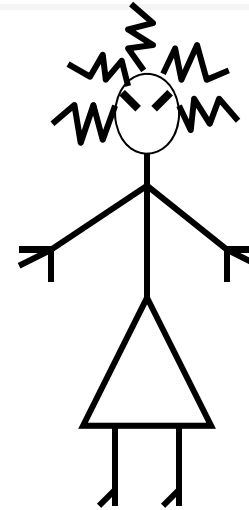- Higher CPI than single cycle processor

Pipelining: Want better Performance

- want small CPI (close to 1) with high MIPS and short clock period (high clock frequency)

- CPU time = instruction count x CPI x clock cycle time
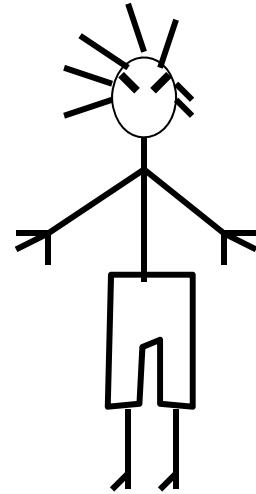
Single Cycle vs Pipelined Processor

See: P&H Chapter 4.5

The Kids

Alice

Bob
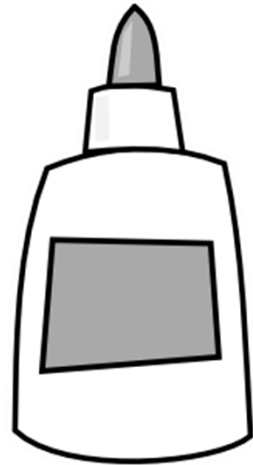
They don't always get along…

The Bicycle
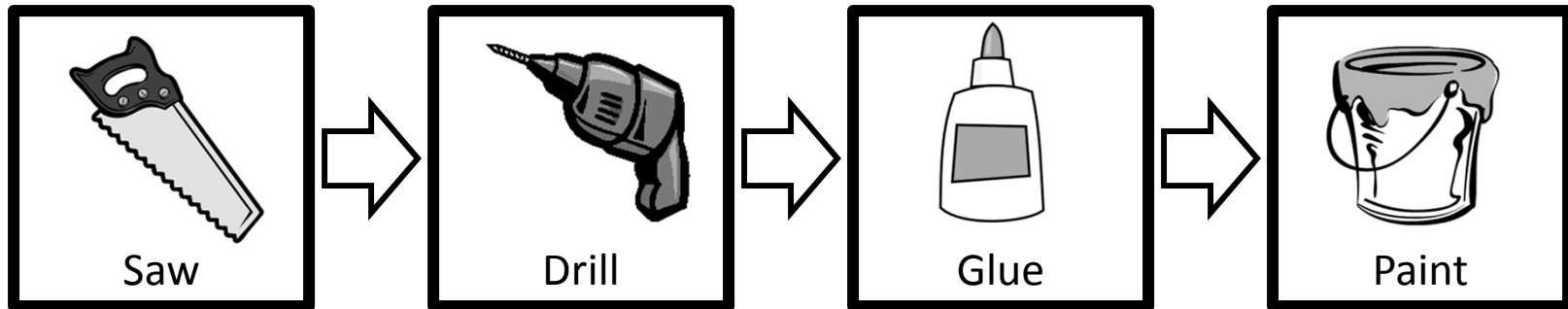
The Materials

Saw

Drill

Glue

Paint

The Instructions

N pieces, each built following same sequence:

Design 1: Sequential Schedule
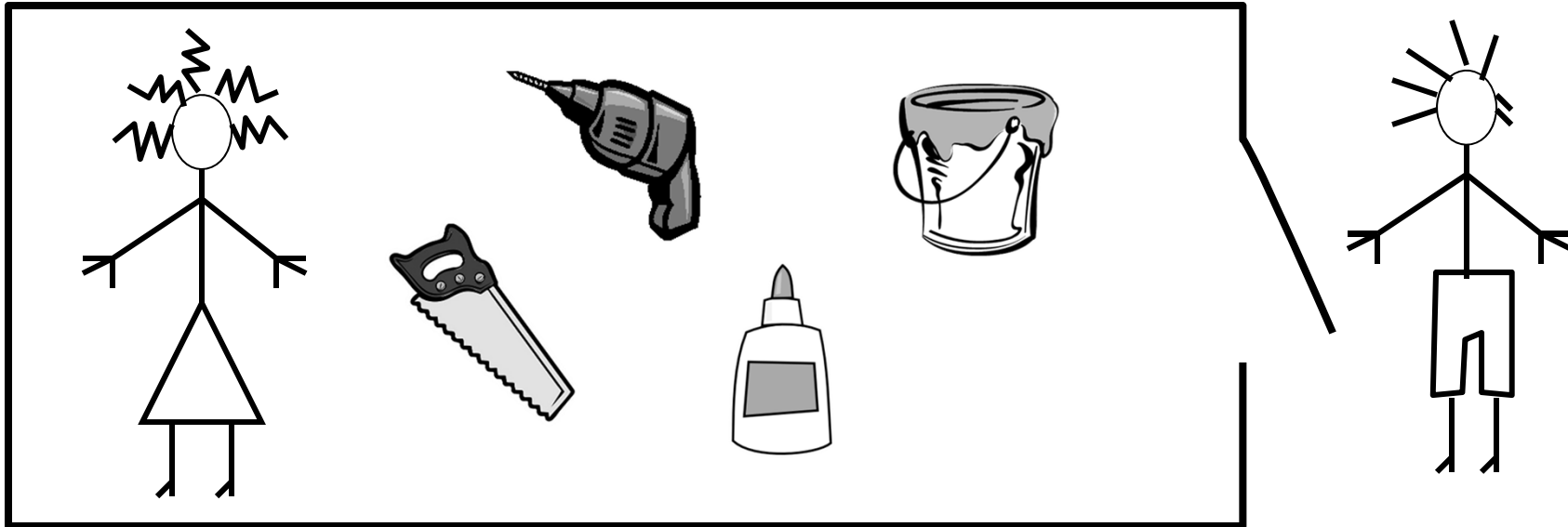
Alice owns the room

Bob can enter when Alice is finished

Repeat for remaining tasks

No possibility for conflicts

Sequential Performance
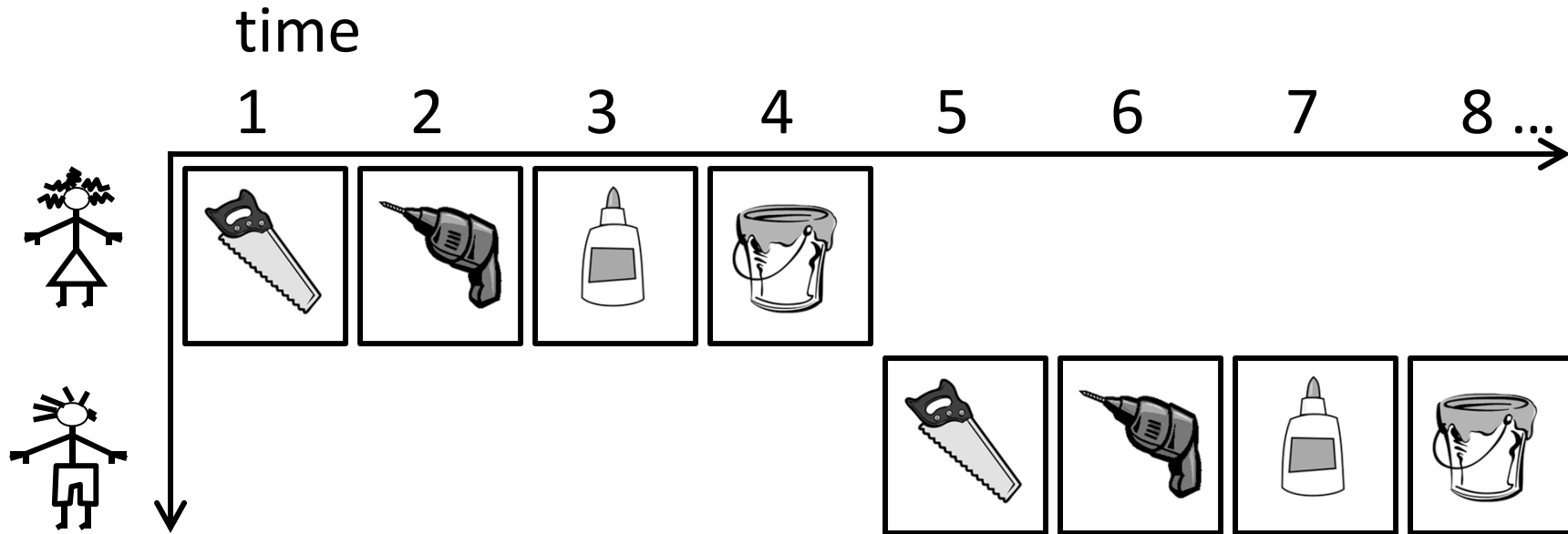
Latency: ~~4 hours / task~~

Throughput: ~~1 task / 4 hrs~~

Concurrency: 1

Can we do better?

CPI = 1

Design 2: Pipelined Design

Partition room into *stages* of a *pipeline*

Dave   Carol   Bob   Alice
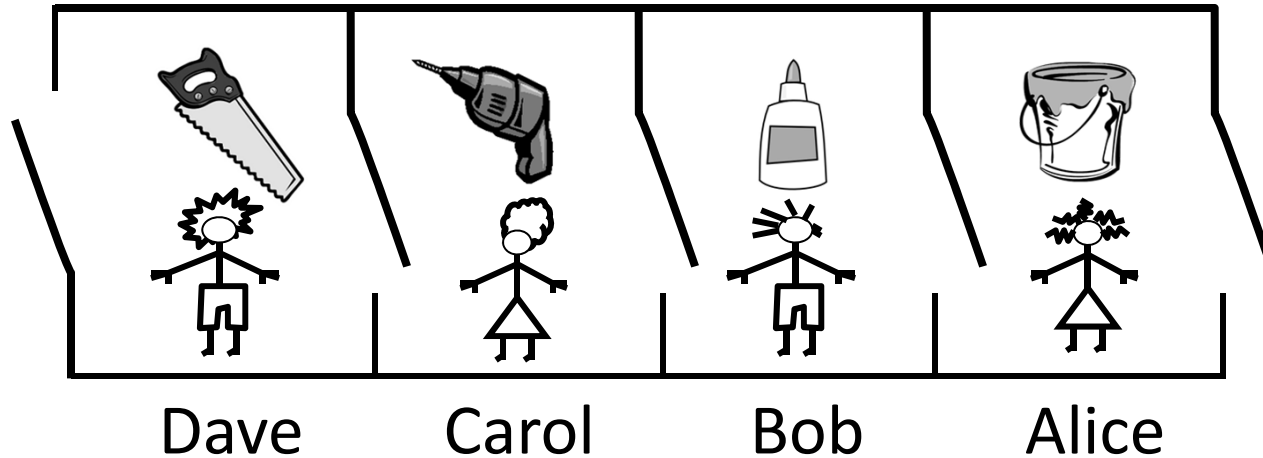
One person owns a stage at a time

4 stages

4 people working simultaneously

Everyone moves right in lockstep

time Pipelined Performance
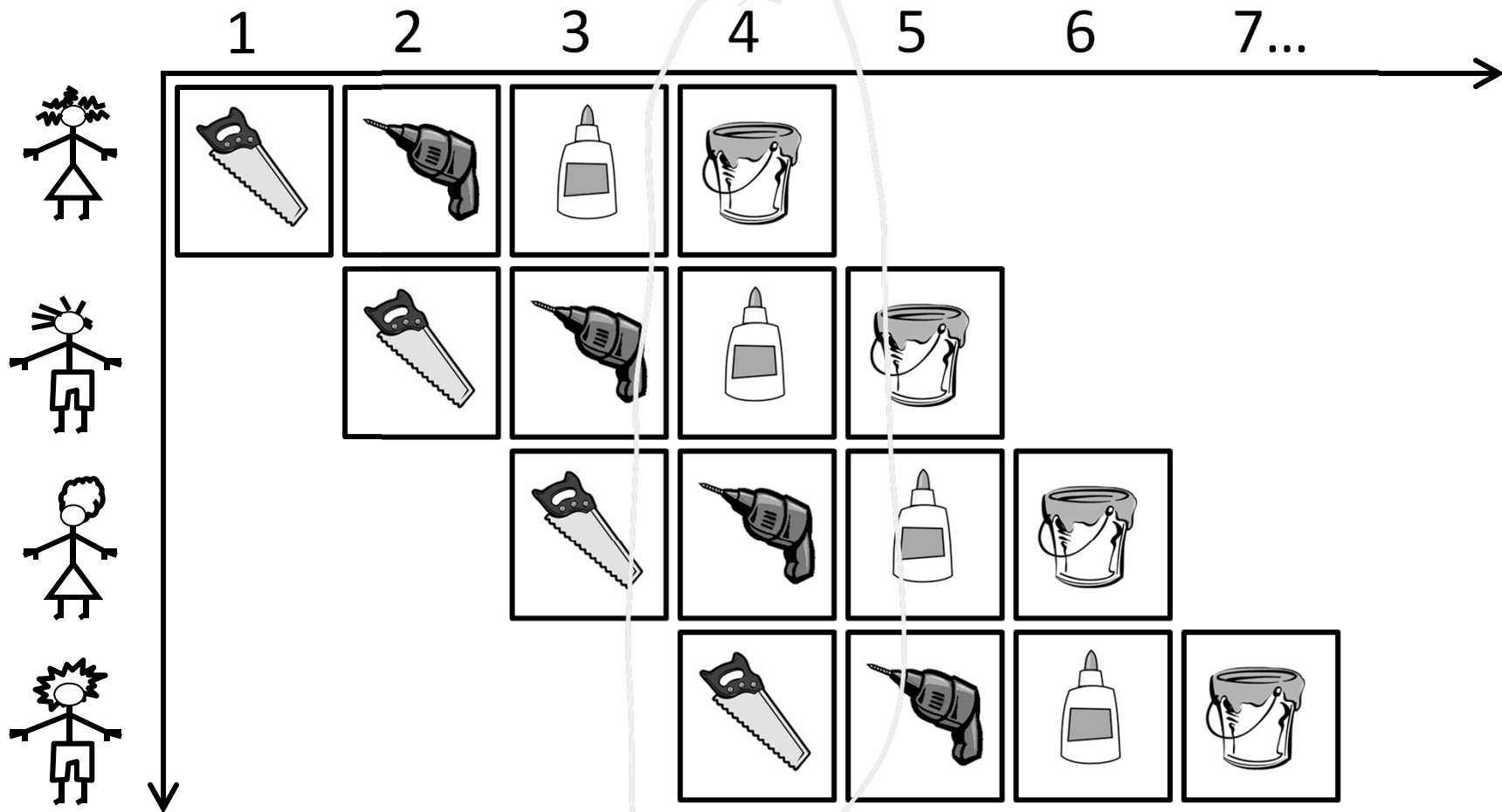
Latency: 4 hrs/task

Throughput: 1 task/hr

Concurrency: 4

CPI = 1

Lessons

Principle:

Throughput increased by parallel execution

Pipelining:

- Identify *pipeline stages*
- Isolate stages from each other
- Resolve pipeline *hazards* (Thursday)

A Processor

Review: Single cycle processor
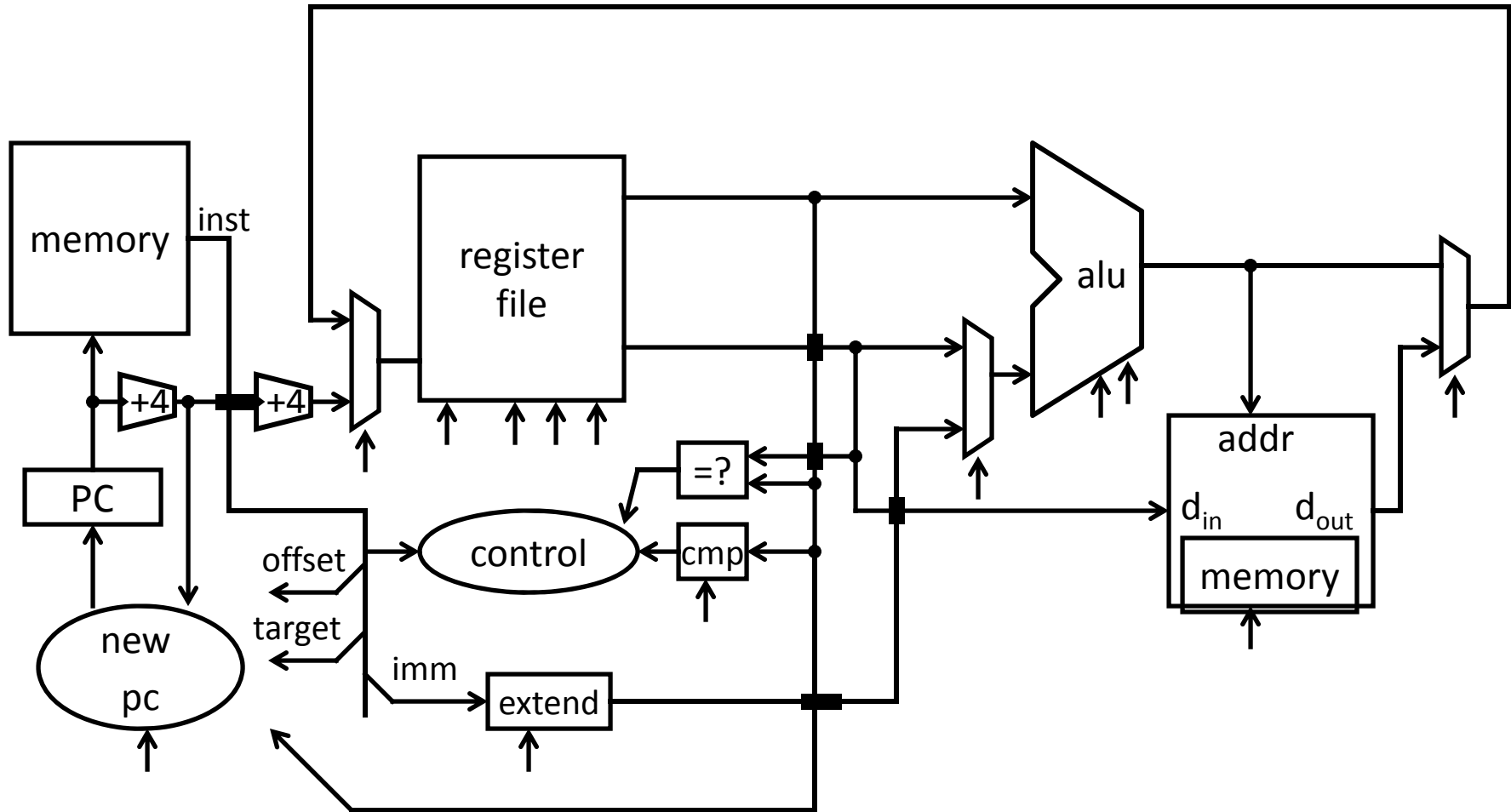
A Processor
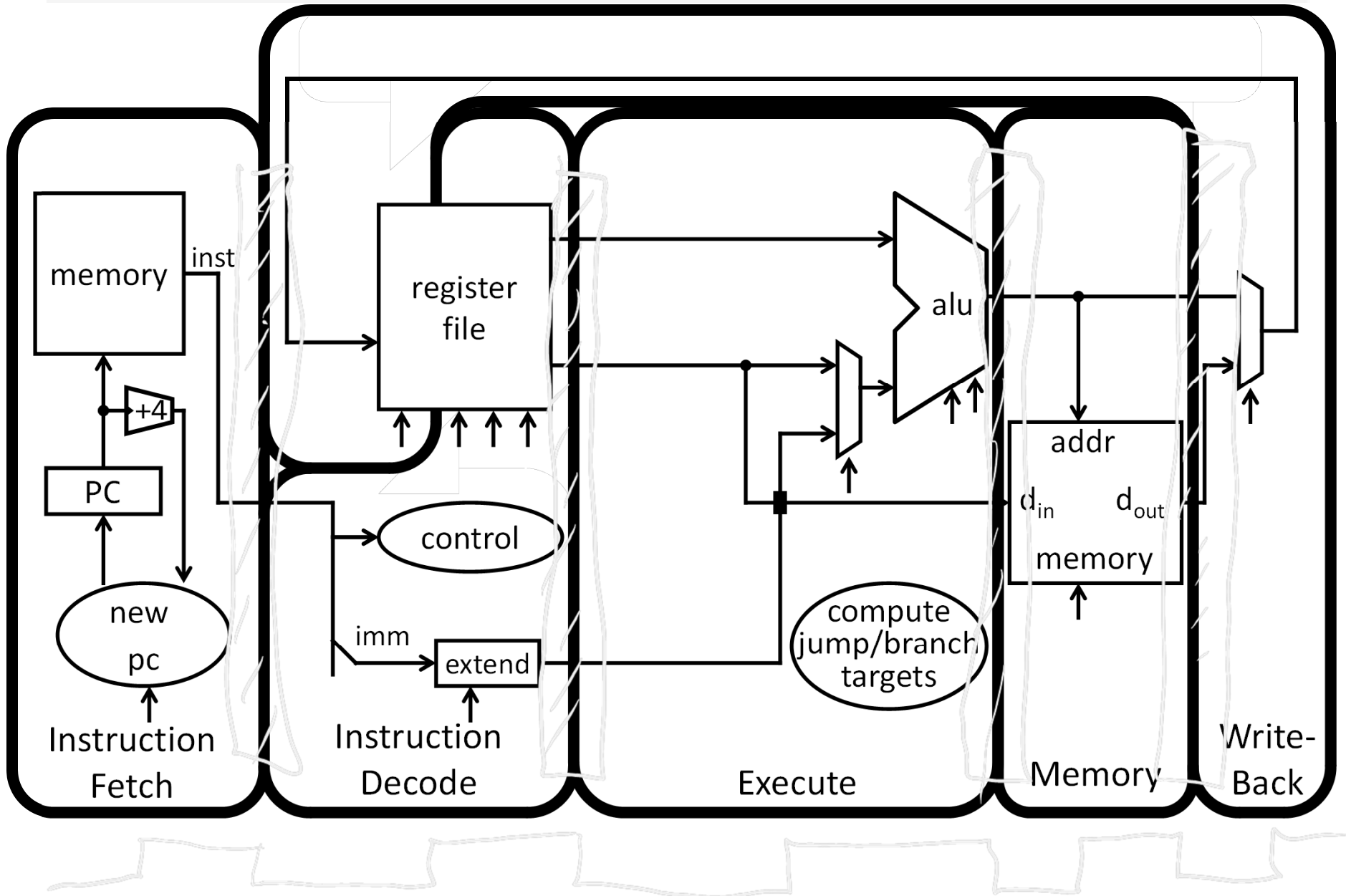
Basic Pipeline

Five stage "RISC" load-store architecture

1. Instruction fetch (IF)
   - get instruction from memory, increment PC

2. Instruction Decode (ID)
   - translate opcode into control signals and read registers

3. Execute (EX)
   - perform ALU operation, compute jump/branch targets

4. Memory (MEM)
   - access memory if needed

5. Writeback (WB)
   - update register file

Time Graphs

Clock cycle

	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
lw		IF	ID	EX	MEM	WB			
			IF	ID	EX	MEM	WB		
				IF	ID	EX	MEM	WB	
					IF	ID	EX	MEM	WB
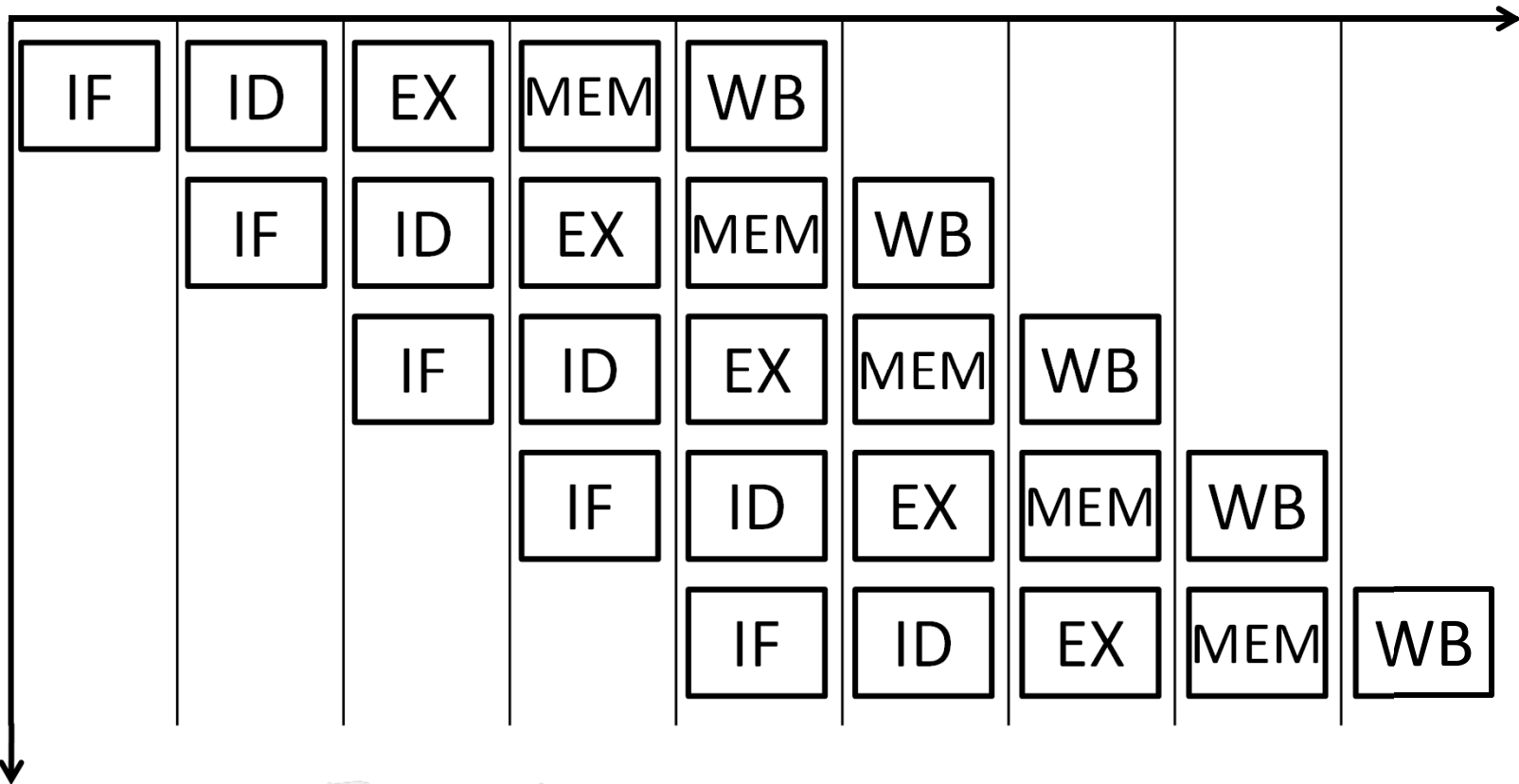
Latency: 5 cycles

Throughput: 1 instr/cycle   CPI = 1

Concurrency: 5

19

Principles of Pipelined Implementation

Break instructions across multiple clock cycles
(five, in this case)

Design a separate stage for the execution
performed during each clock cycle

Add pipeline registers (flip-flops) to isolate signals
between different stages

Pipelined Processor

See: P&H Chapter 4.6

Pipelined Processor

Instruction Fetch | Instruction Decode | Execute | Memory | Write-Back

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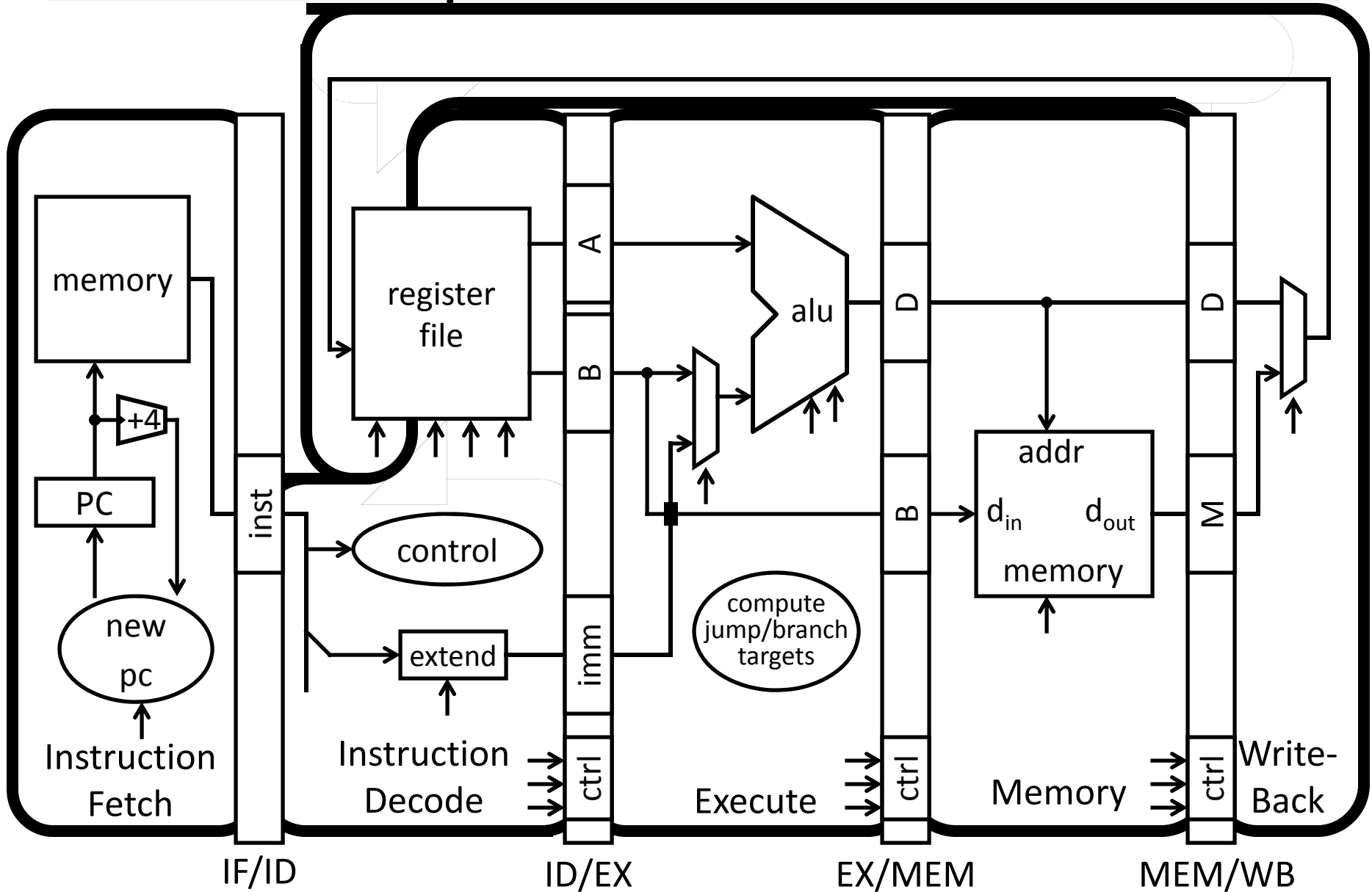
IF

Stage 1: Instruction Fetch

Fetch a new instruction every cycle
- Current PC is index to instruction memory
- Increment the PC at end of cycle (assume no branches for now)

$$2^{28} = 2^{20+8} = 2^{20} \cdot 2^{8}$$

Write values of interest to pipeline register (IF/ID)
$$= 256M$$
- Instruction bits (for later decoding)
- PC+4 (for later computing branch targets)

PC+4

BEQ   PC rel

J   PC abs $(PC+4)_{31...28}$
                   $(target2)$

JR   PC reg

instruction memory

addr   mc

+4

PC

new pc

inst

pc+4

pc+4
pc reg
pc abs
pc rel

IF

instruction memory

addr   mc

00 = read word

+4

PC

1→

1→ WE

inst

PC+4

Rest of pipeline

pcreg

pcrel

pcabs

pcsel

IF/ID

r   y

pc
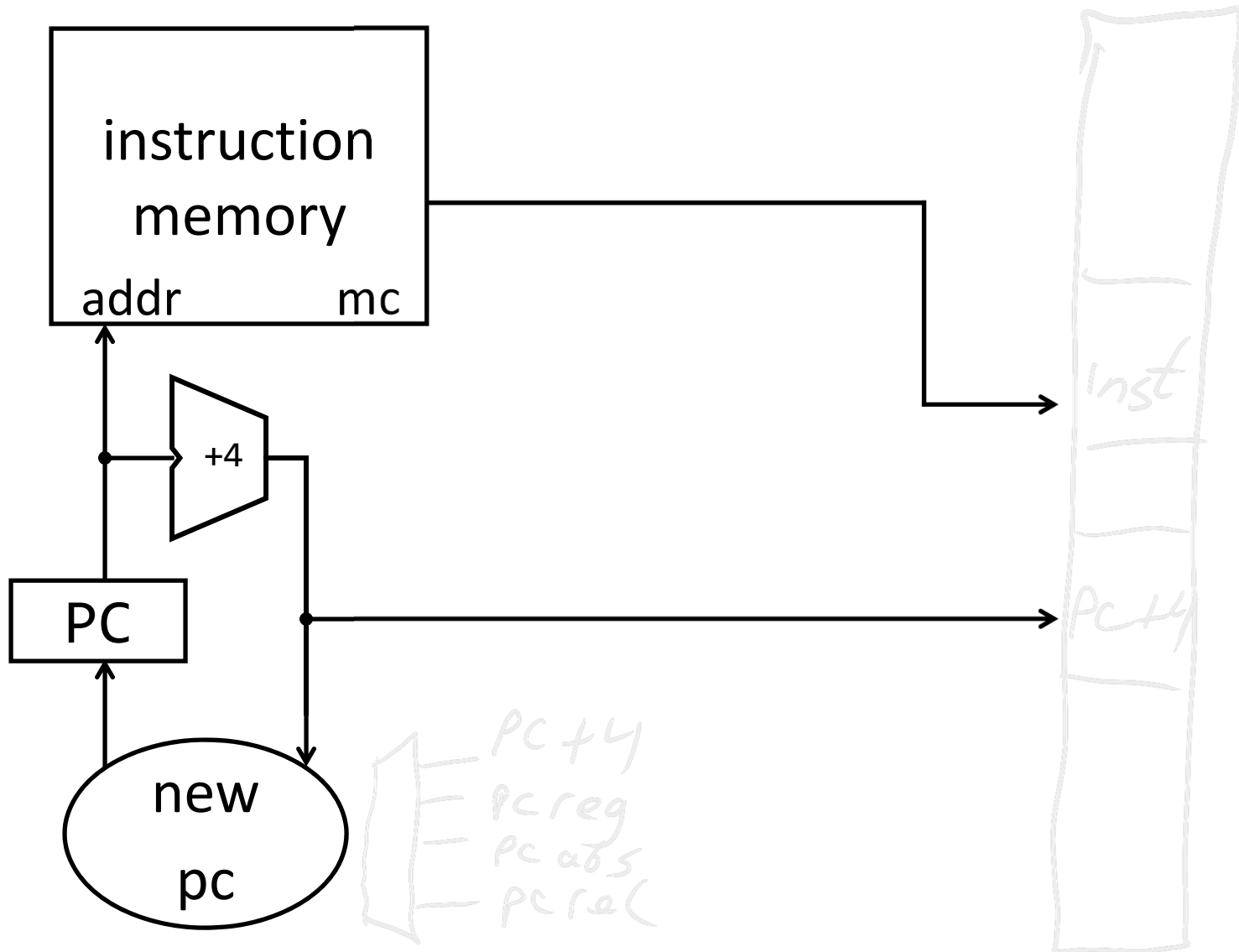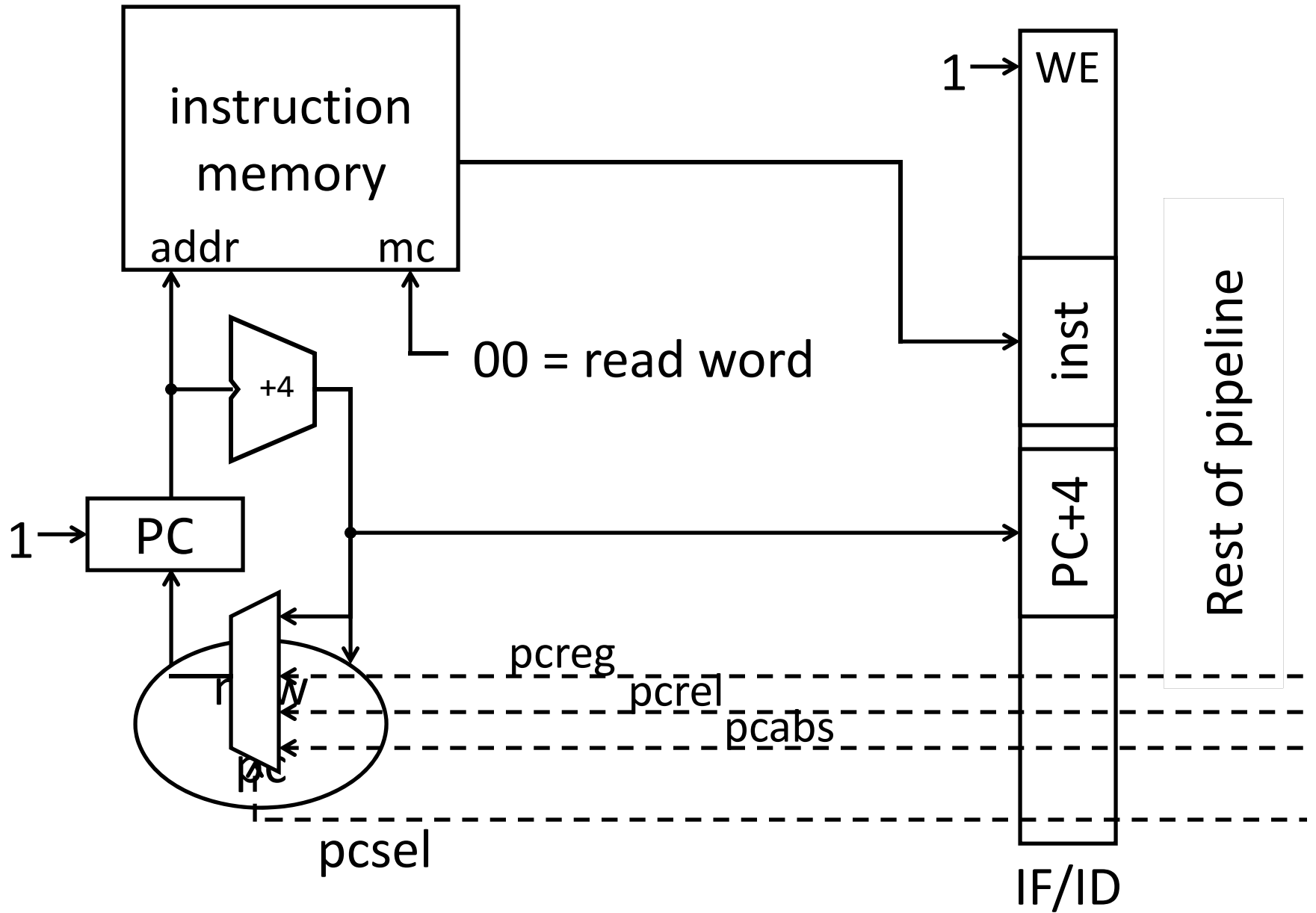
ID

Stage 2: Instruction Decode

On every cycle:
- Read IF/ID pipeline register to get instruction bits
- Decode instruction, generate control signals
- Read from register file

Write values of interest to pipeline register (ID/EX)
- Control information, Rd index, immediates, offsets, …
- Contents of Ra, Rb
- PC+4 (for computing branch targets later)

Stage 1: Instruction Fetch

register file

WE
Rd
D

A

B

Ra Rb

dest

mem/ALU result

decode

extend

IF/ID

inst

PC+4

ID/EX

A

B

imm

PC+4

ctrl

Rest of pipeline

ID — result — dest

register file

WE
Rd
D

A

B

Ra Rb

decode

extend

inst

PC+4

IF/ID

Stage 1: Instruction Fetch

A

B

imm

PC+4

ctrl
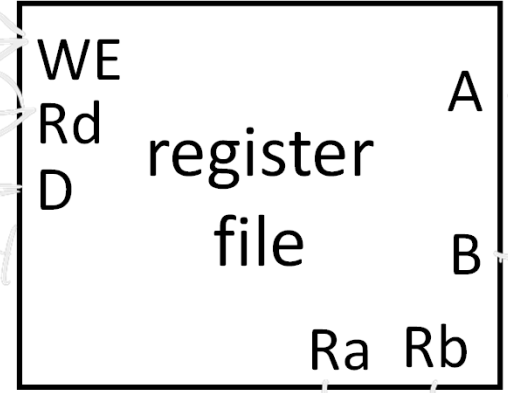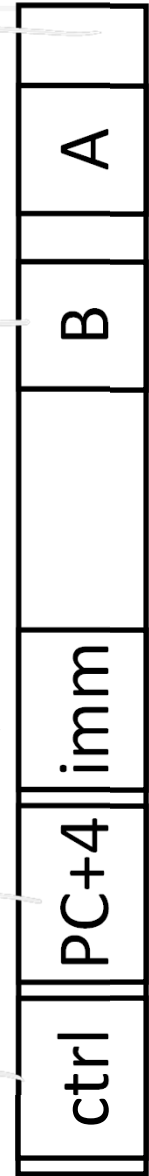
ID/EX

Rest of pipeline

28
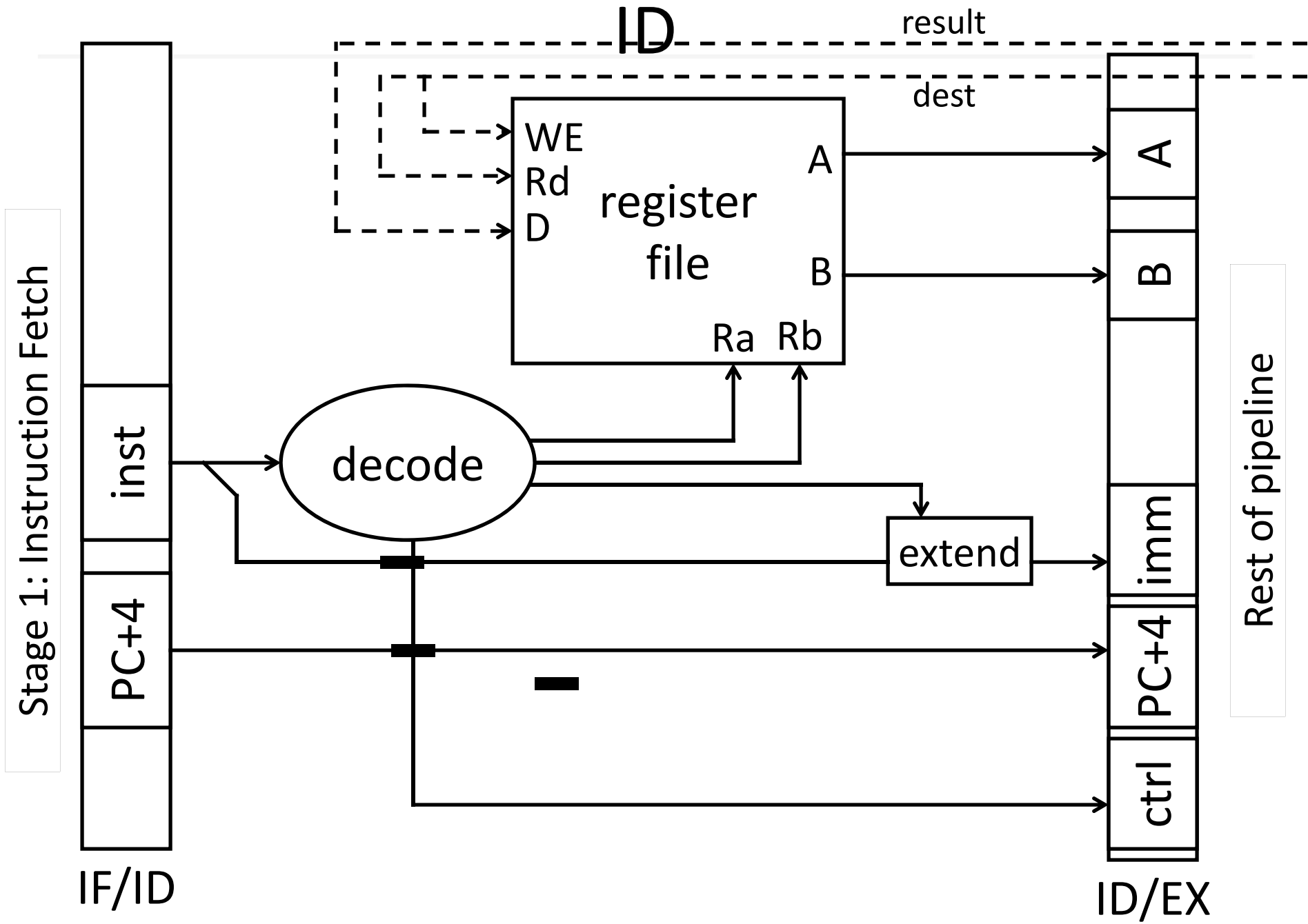
EX

Stage 3: Execute

On every cycle:
- Read ID/EX pipeline register to get values and control bits
- Perform ALU operation
- Compute targets (PC+4+offset, etc.) *in case* this is a branch
- Decide if jump/branch should be taken

Write values of interest to pipeline register (EX/MEM)
- Control information, Rd index, …
- Result of ALU operation
- Value *in case* this is a memory store instruction

EX

pcsel

branch?

pcreg

Stage 2: Instruction Decode

ID/EX

A

B

imm

PC+4

ctrl

=

alu

pcrel

pcabs

EX/MEM

D

B

target

ctrl

Rest of pipeline

30
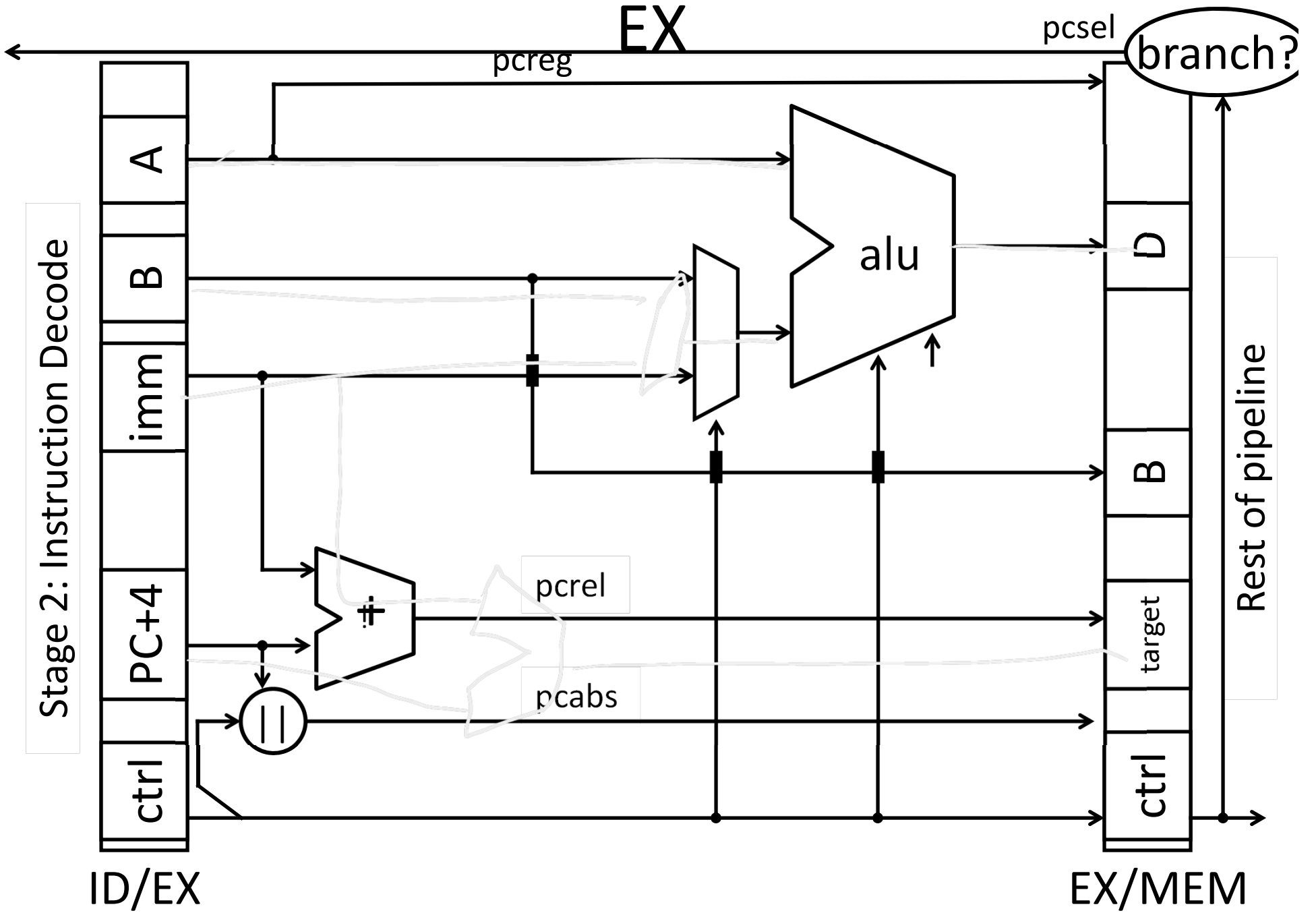
MEM

Stage 4: Memory

On every cycle:
- Read EX/MEM pipeline register to get values and control bits
- Perform memory load/store if needed
  - address is ALU result

Write values of interest to pipeline register (MEM/WB)
- Control information, Rd index, …
- Result of memory operation
- Pass result of ALU operation

MEM

Stage 3: Execute

EX/MEM

ctrl

target

B

D

ALU Result
~ Ra * Rb
~ sw/lw

addr

d_in   memory   d_out

mc

sw

lw

sw = wr
lw = rd

Rest of pipeline

MEM/WB

ctrl

M

D

MEM

pcsel
branch?
pcreg

Stage 3: Execute

D
B
target
ctrl
pcrel
pcabs

addr
d_{in} d_{out}
memory
mc

EX/MEM

D
M
ctrl

Rest of pipeline
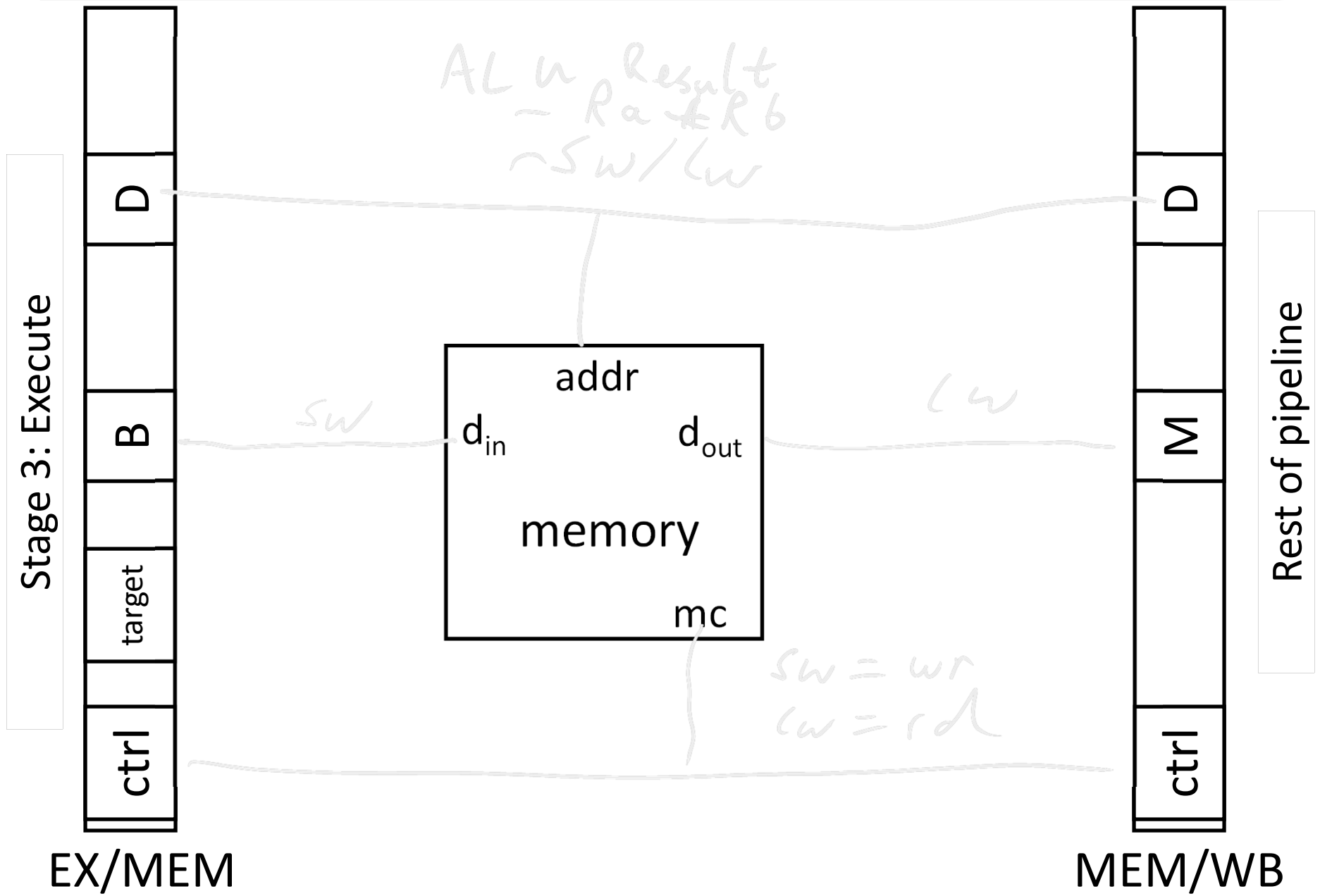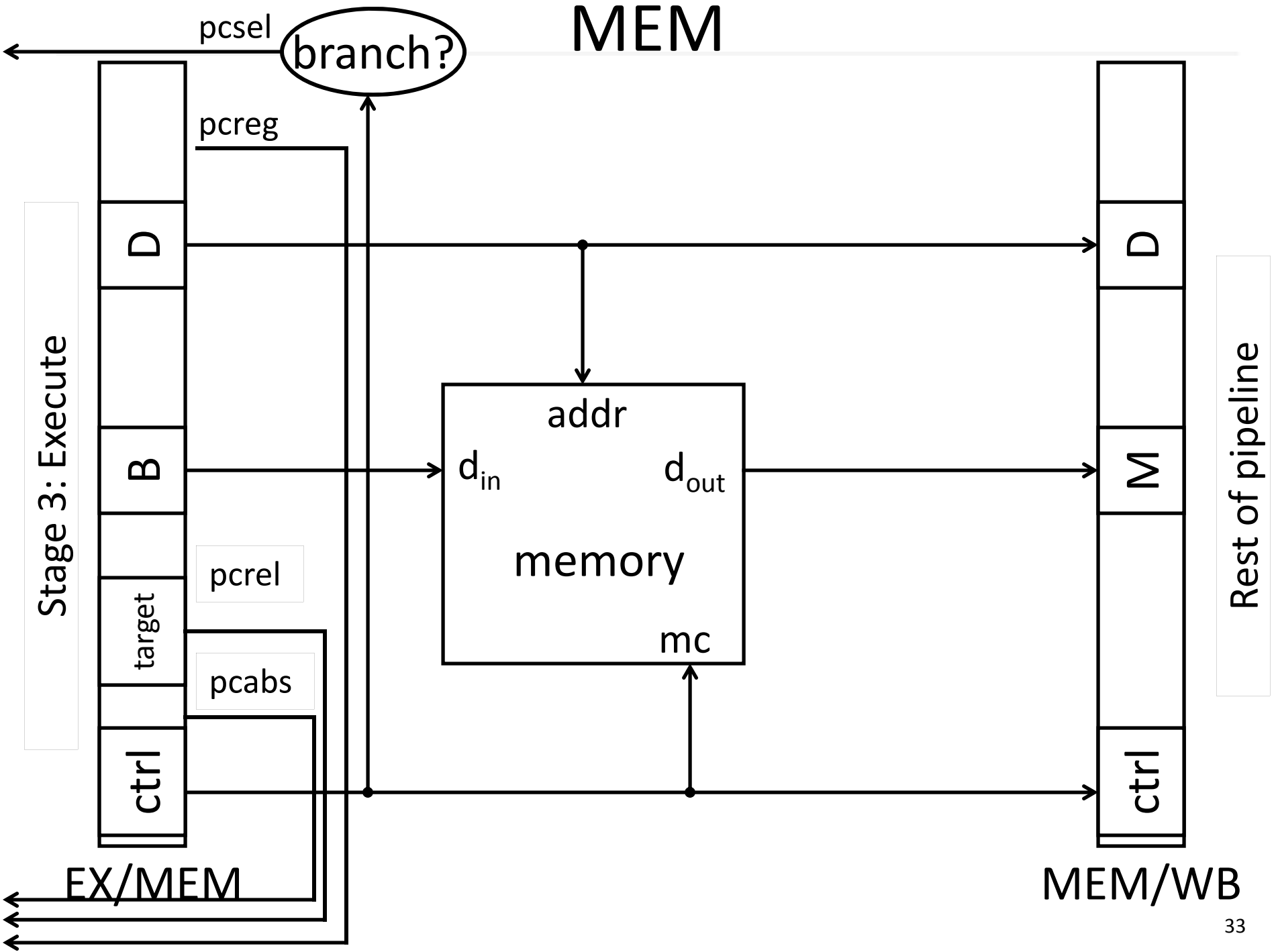
MEM/WB

WB

Stage 5: Write-back

On every cycle:
- Read MEM/WB pipeline register to get values and control bits
- Select value and write to register file

Stage 4: Memory

add

lw

Rd

WE

D

M

ctrl

MEM/WB

WB

result

D

Stage 4: Memory

M

dest

ctrl
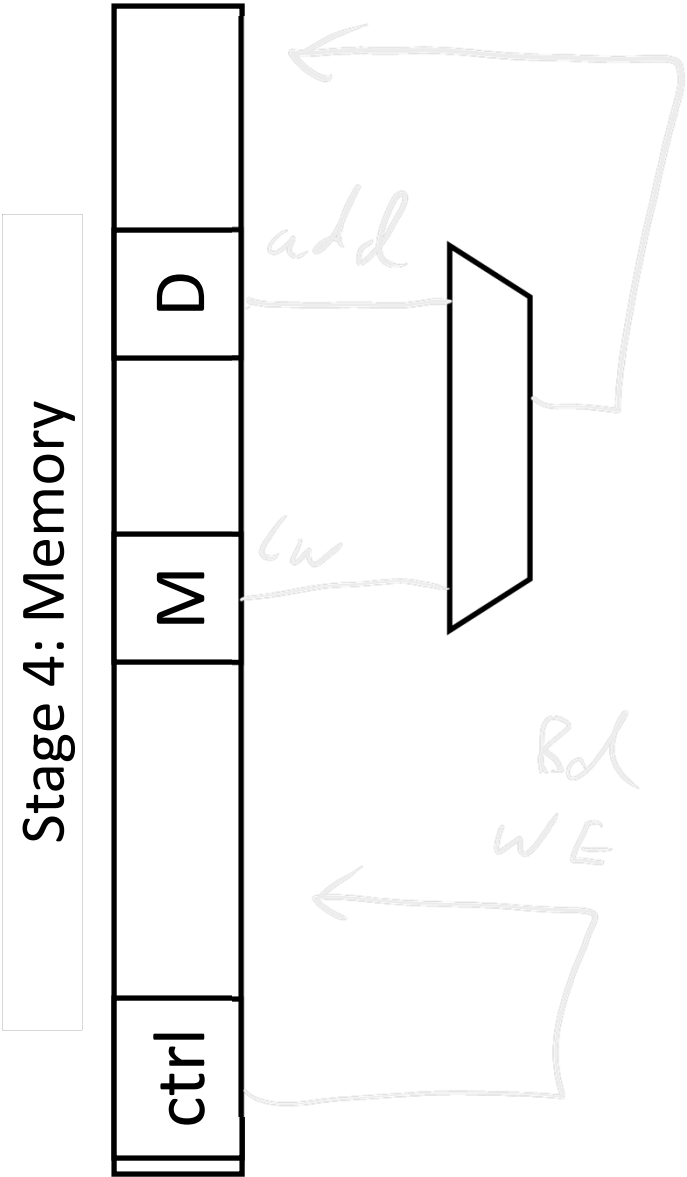
MEM/WB

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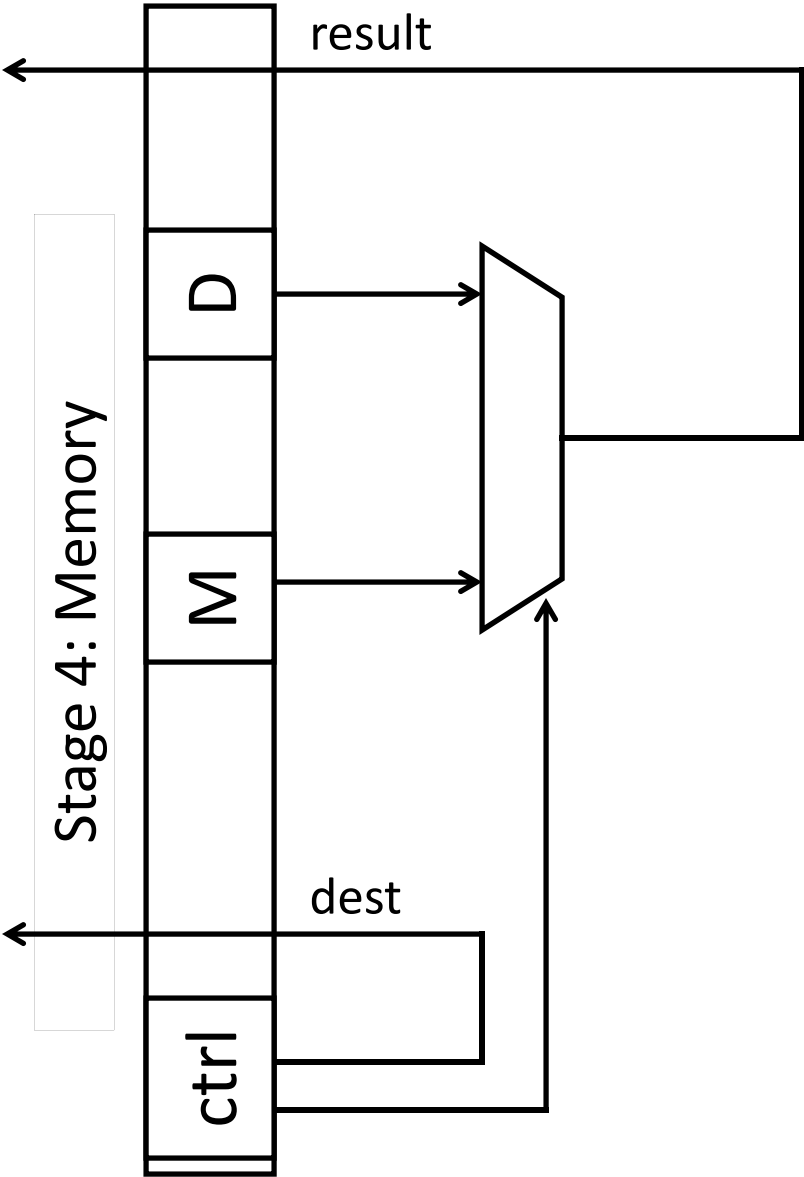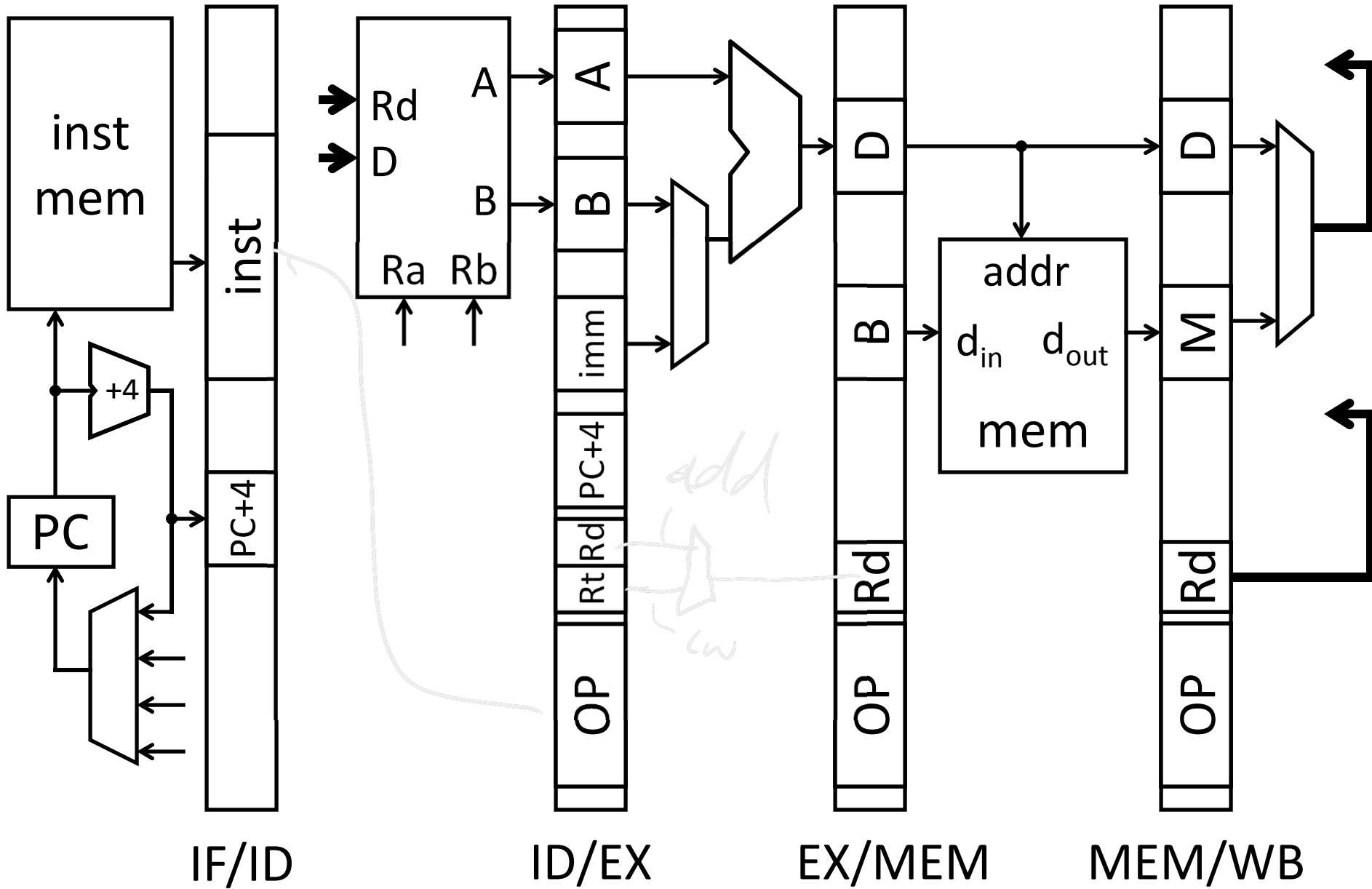
Administrivia

HW2 due today
- ***Fill out Survey online***.   Receive credit/points on homework for survey:
- https://cornell.qualtrics.com/SE/?SID=SV_5olFfZiXoWz6pKI
- 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

Prelim1: next Tuesday, February 28th in evening

- We will start at 7:30pm sharp, so come early
- Prelim Review: This Wed / Fri, 3:30-5:30pm, in 155 Olin
- 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

Check online syllabus/schedule

* http://www.cs.cornell.edu/Courses/CS3410/2012sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):

* Tuesday, February 28th
* Thursday, March 29th
* Thursday, April 26th

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
- 20% 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.

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
```

Example: : Sample Code (Simple)

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

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
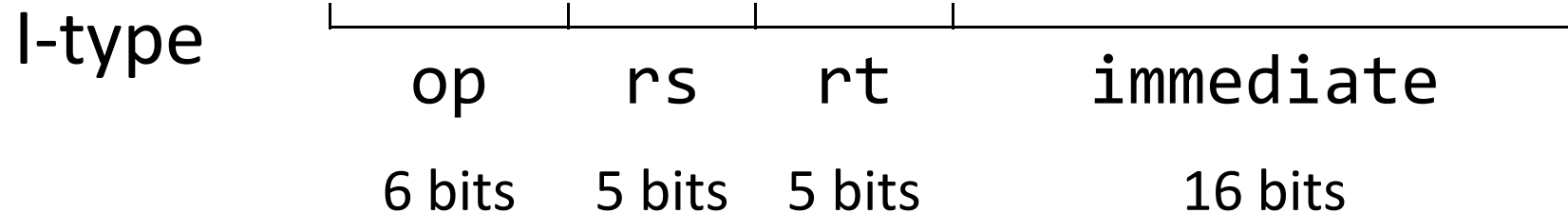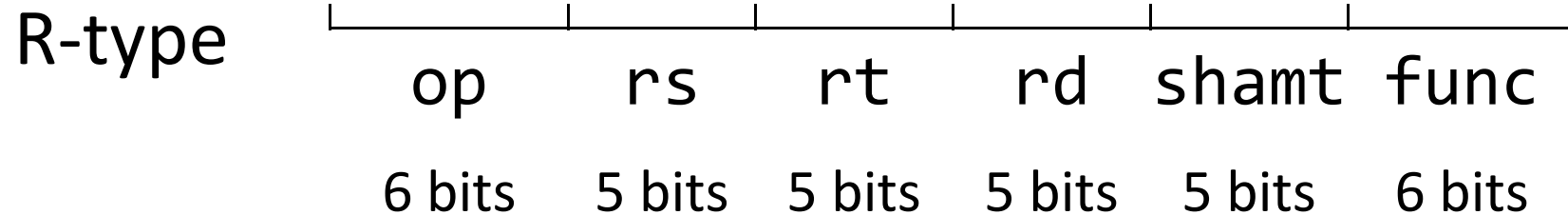
MIPS Instruction Types

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

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:

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

add r3 r1 r2

Initial State

Register	Value
R0	0
R1	36
R2	9
R3	12
R4	18
R5	7
R6	41
R7	22

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

add 3 1 2

4

4

add 3 1 2

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

PC

Inst mem

M U X

M U X

A L U

M U X

Data mem

M U X

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

nop

nop

nop

data

dest

Fetch:
 add 3 1 2

IF/ID

ID/EX

EX/MEM

MEM/WB

Time: 1

49

nand 6 4 5

add 3 1 2

MUX

4

+

8

PC

Inst mem

nand 6 4 5

R0	0
R1	36
R2	9
R3	12
R4	18
R5	7
R6	41
R7	22

Register file

1

2

extend

4

8

36

9

3

0

0

0

MUX

ALU

0

0

0

Data mem

0

0

MUX

data

dest

Fetch:
  nand 6 4 5

Bits 11-15

Bits 16-20

Bits 26-31

3

2

MUX

add

3

0

nop

0

nop

IF/ID

ID/EX

EX/MEM

MEM/WB

Time: 2

50

lw 4 20(2)

nand 6 4 5

add 3 1 2

M U X

4

+

12

PC

Inst mem

lw 4 20(2)

Register file

R0	0
R1	36
R2	9
R3	12
R4	18
R5	7
R6	41
R7	22

4

5

extend

Bits 11-15

Bits 16-20

Bits 26-31

8

18

7

6

6

5

nand

36

9

M U X

3

2

M U X

A L U

4

0

45

9

3

3

add

Data mem

M U X

0

0

data

dest

nop

Fetch:
 lw 4 20(2)

IF/ID

ID/EX

EX/MEM

MEM/WB

Time: 3

add 5 2 5   lw 4 20(2)   nand 6 4 5   add 3 1 2

Fetch:
add 5 2 5

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

Time: 4

sw 7 12(3)　　　add 5 2 5　　　lw 4 20 (2)　　　nand 6 4 5　　　add　3 1 2

4

+

20

16

M U X

PC

Inst mem

sw 7 12(3)

R0	**0**
R1	**36**
R2	**9**
R3	**45**
R4	**18**
R5	**7**
R6	**41**
R7	**22**

Register file

2

5

9

7

5

extend

9

A L U

20

M U X

12

0

29　-3

18

Data mem

-3

0

45

M U X

data

dest

Fetch:
　sw 7　12(3)

Bits 11-15

Bits 16-20

Bits 26-31

5

5

add

0

4

M U X

4

6

4

lw

6

6

nand

3

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

Time: 5

53

nop nop sw 7 12(3) add 5 2 5 lw 4 20(2)

MUX

4
+

R0 0
R1 36
R2 9
R3 45
R4 99
R5 7
R6 -3
R7 22

Register file

PC

Inst mem

extend

Bits 11-15
Bits 16-20
Bits 26-31

No more instructions

45

12

0
7

MUX

A L U

20

0

57   16

22

5

7

sw

Data mem

16

0

MUX

99

data

dest

4

5

add

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

Time: 7

55

nop   nop   nop   sw 7 12(3)   add 5 2 5

MUX

4

+

R0 0
R1 36
R2 9
R3 45
R4 99
R5 16
R6 -3
R7 22

PC

Inst mem

Register file

extend

A L U

M U X

57

22

Data mem

22

57

0

M U X

16

data

dest

No more instructions

Bits 11-15

Bits 16-20

Bits 26-31

M U X

5

7

sw

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

Time: 8

56

nop　　　　　**nop**　　　　　**nop**　　　　　**nop**　　　**sw 7 12(3)**

M U X

4

+

PC

Inst mem

R0 **0**
R1 **36**
R2 **9**
R3 **45**
R4 **99**
R5 **16**
R6 **-3**
R7 **22**

Register file

extend

M U X

A L U

M U X

Data mem

M U X

data

dest

No more instructions

Bits 11-15

Bits 16-20

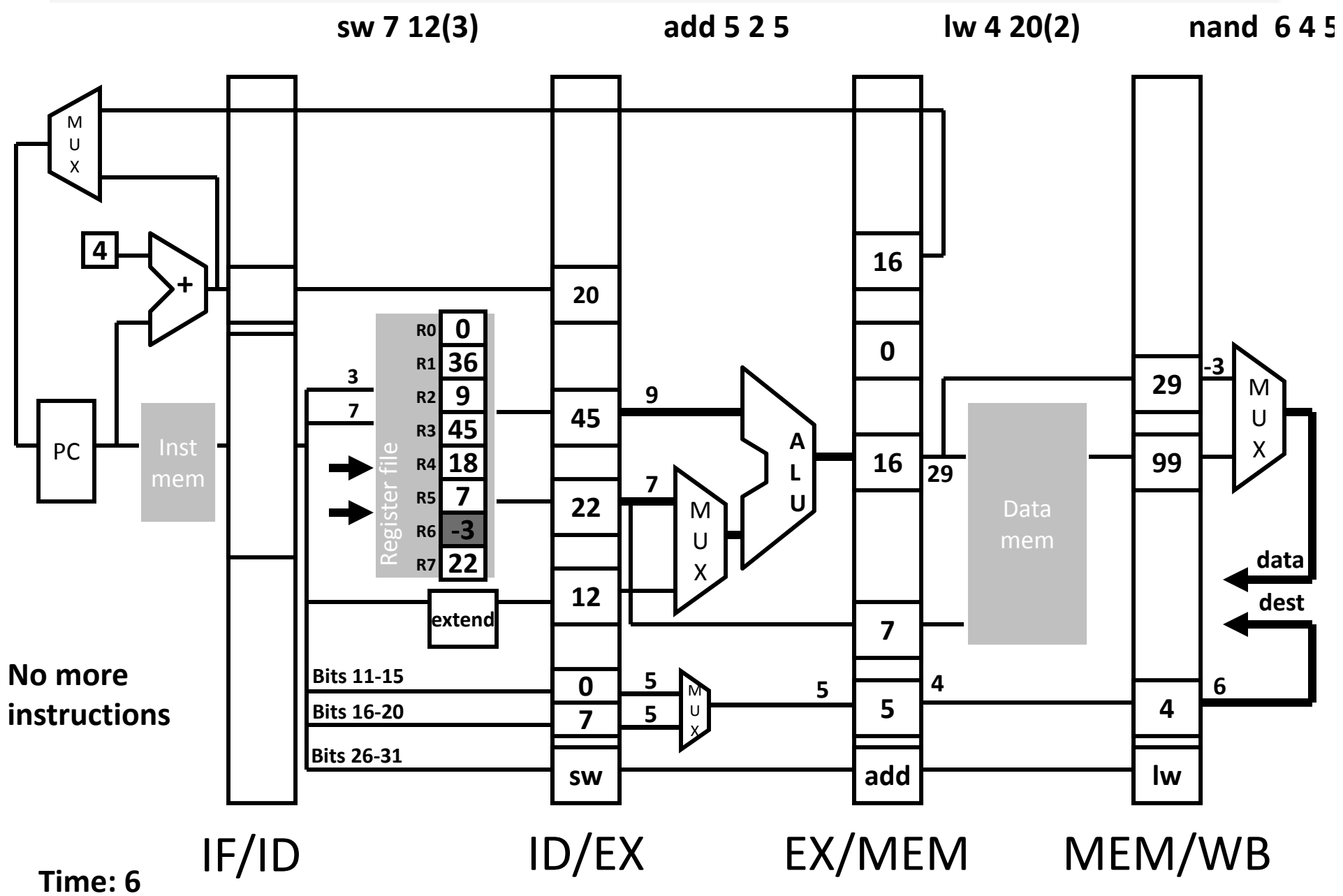Bits 21-23

M U X

IF/ID　　　　　**ID/EX**　　**EX/MEM**　　**MEM/WB**
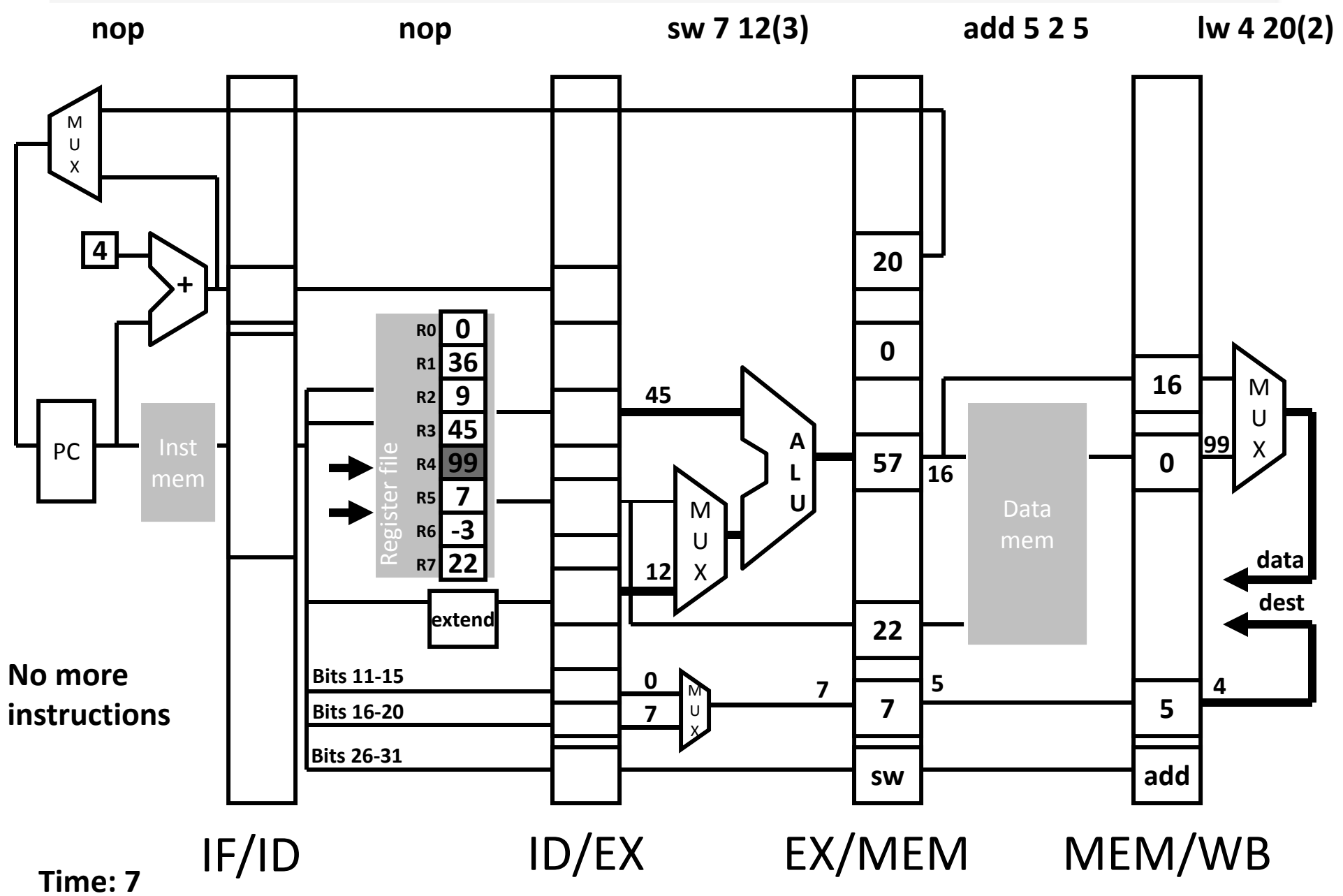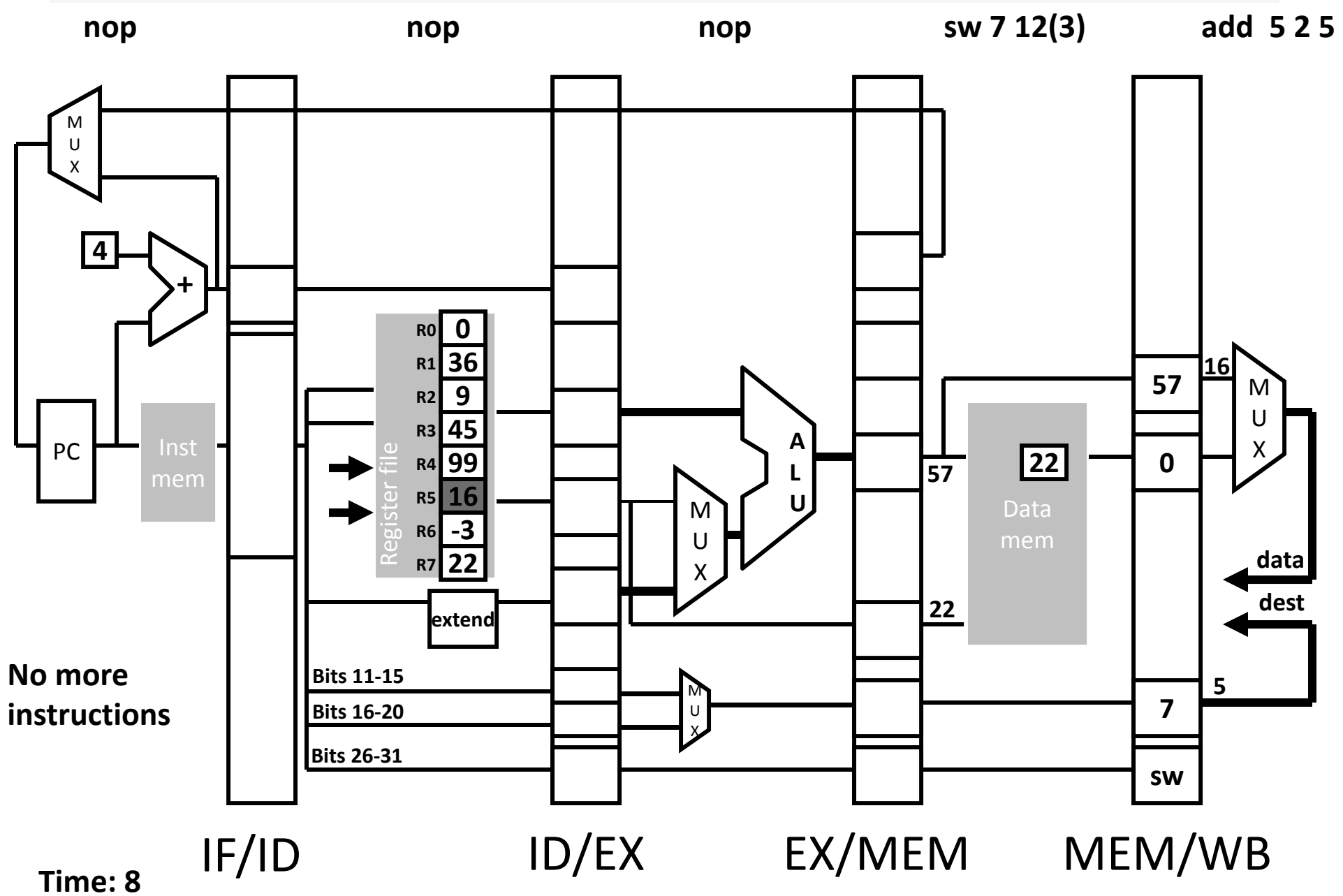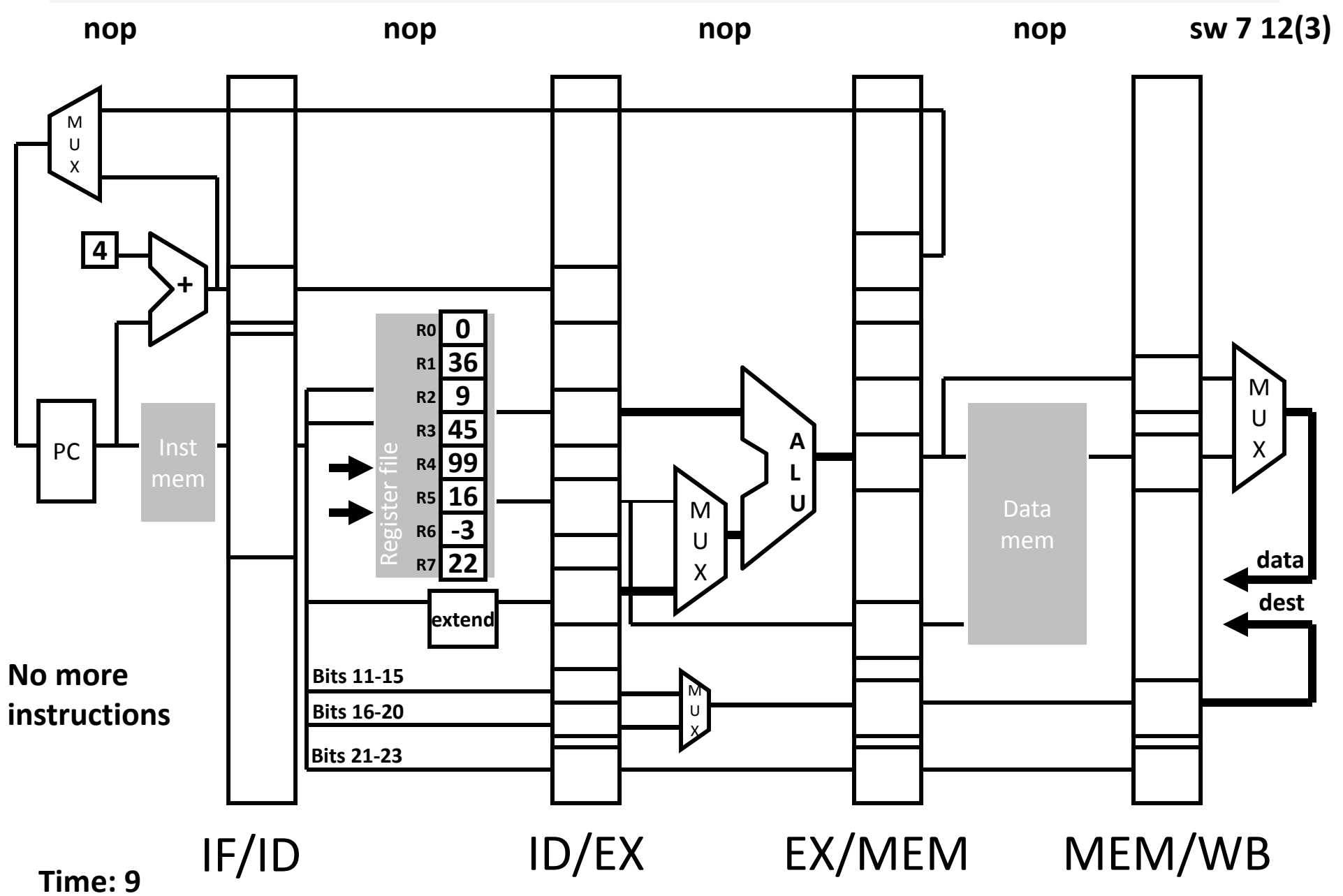
Time: 9

Pipelining Recap

Powerful technique for masking latencies

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

Abstraction promotes decoupling

- Interface (ISA) vs. implementation (Pipeline)

inst mem

PC

+4

IF/ID

inst

PC+4

Rd
D
A
B
Ra Rb

ID/EX

A
B
imm
PC+4
Rd
OP

EX/MEM

D
·
B
Rd
OP

addr

d_{in} d_{out}

mem

MEM/WB

D
M
Rd
OP