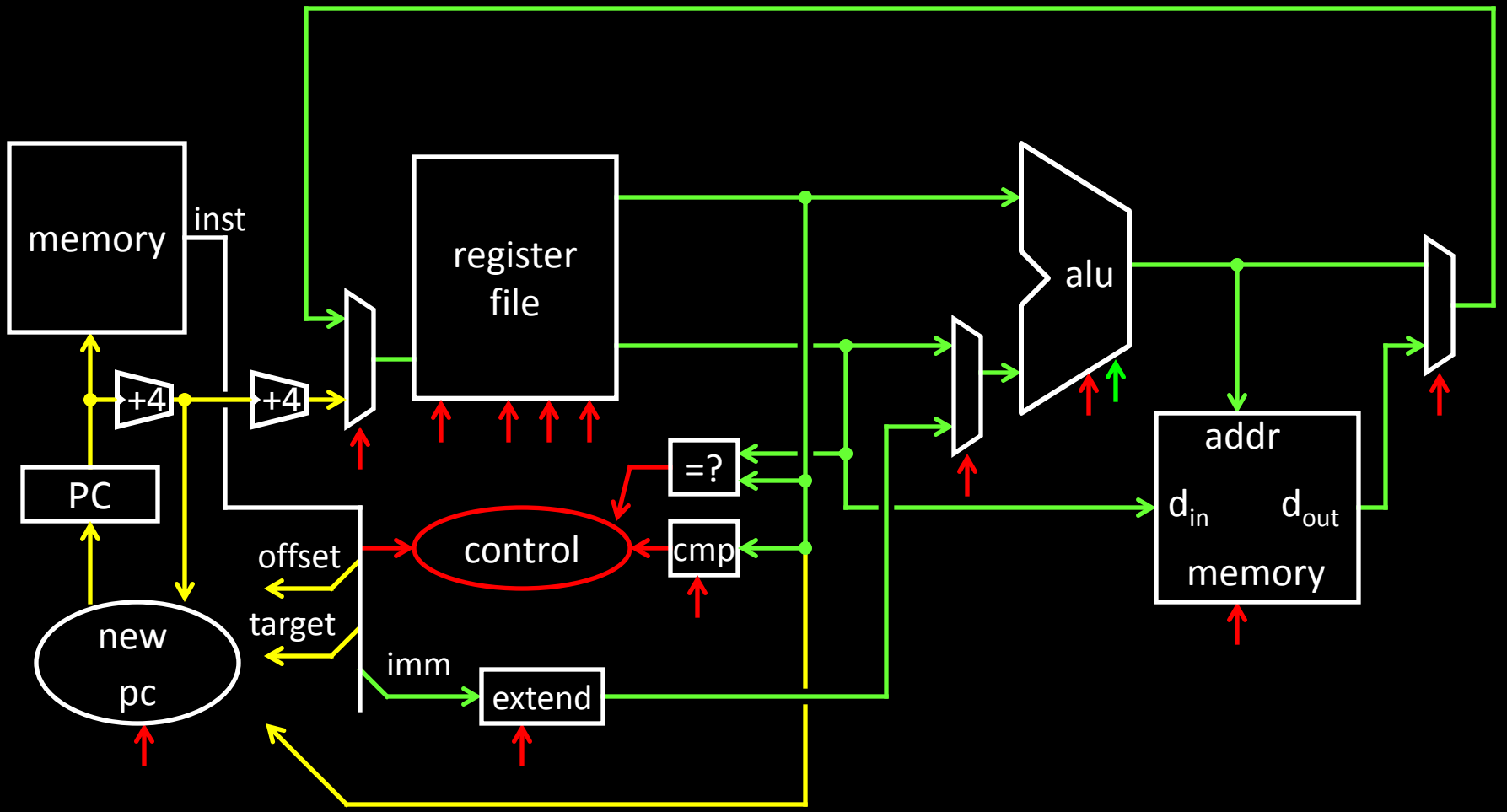
MIPS Pipeline

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

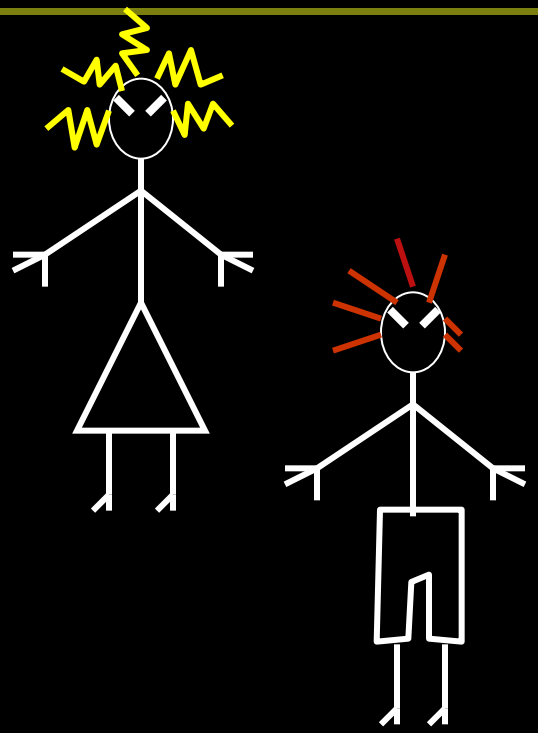
Single Cycle vs Pipelined Processor

See: P&H Chapter 4.5

The Kids

Alice

Bob



They don’t always get along…

The Bicycle

The Materials

Saw

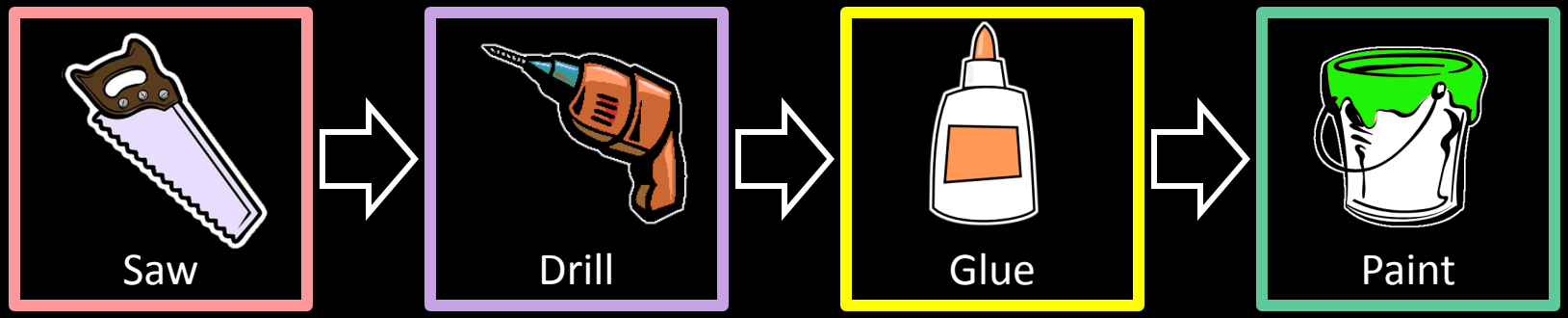
Drill

Glue

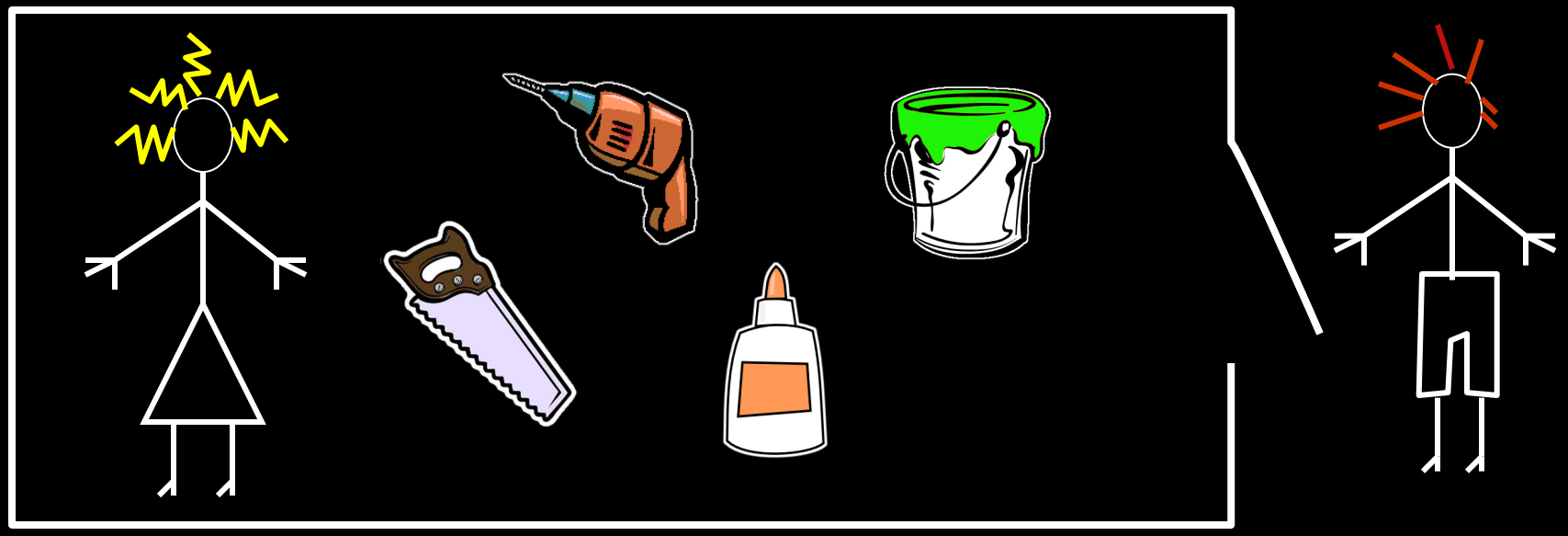
Paint

The Instructions

N pieces, each built following same sequence:



Design 1: Sequential Schedule



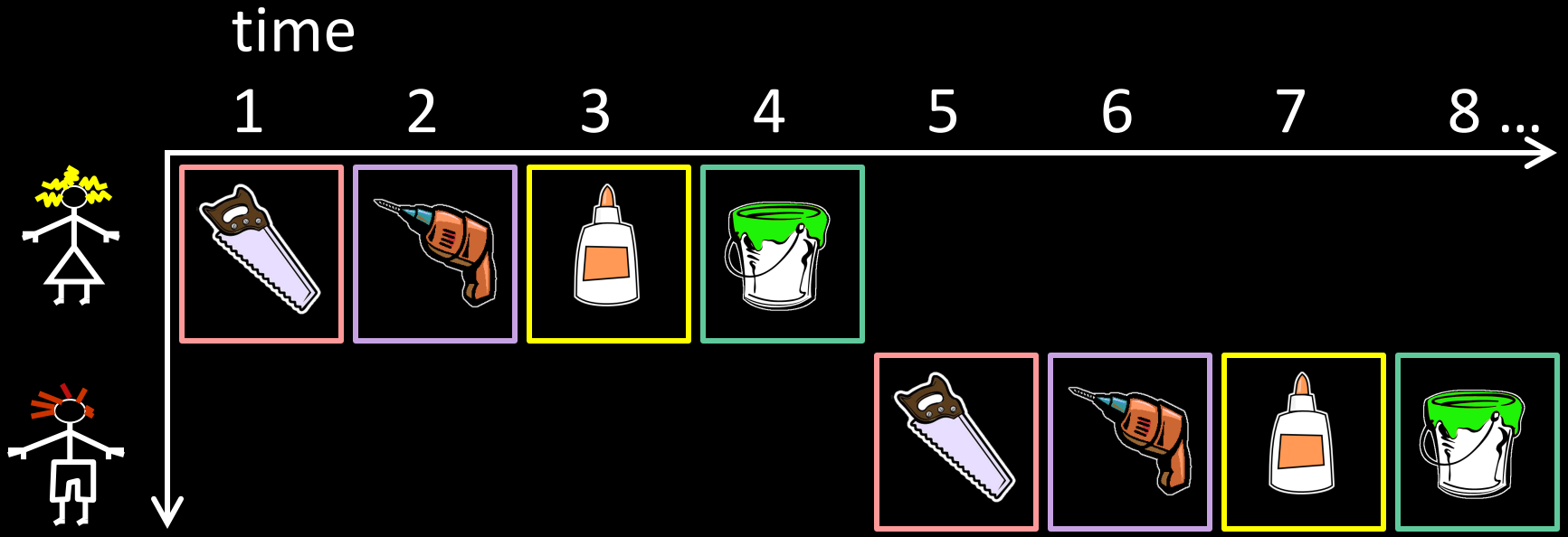
Alice owns the room

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

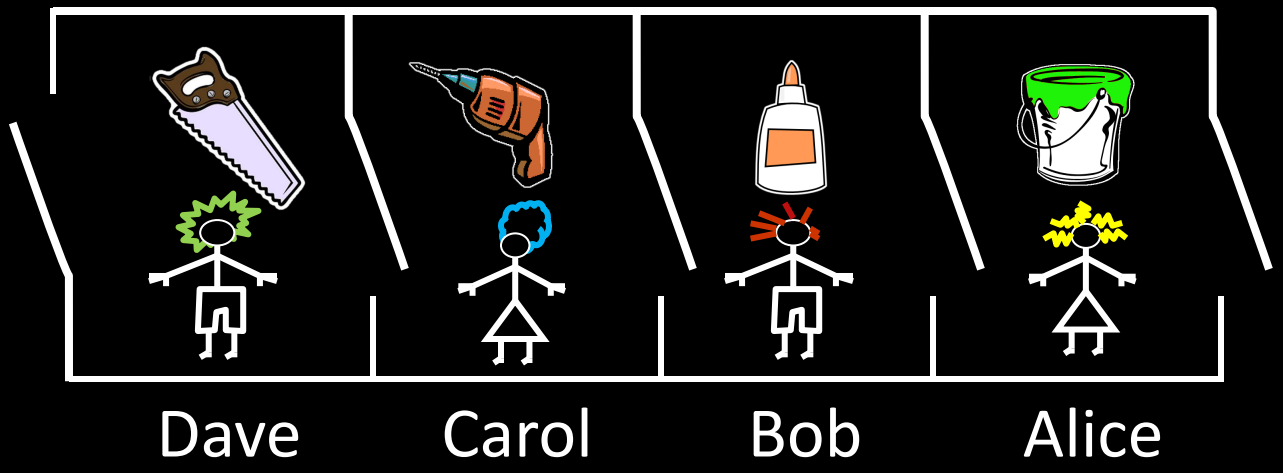
Concurrency: 1

Can we do better?

CPI = 1

Design 2: Pipelined Design

Partition room into *stages* of a *pipeline*



One person owns a stage at a time

4 stages

4 people working simultaneously

Everyone moves right in lockstep

Pipelined Performance

Latency: 4hrs/task

Throughput: 1 task/hr

Concurrency: 4

CPI = 1

Lessons

Principle:

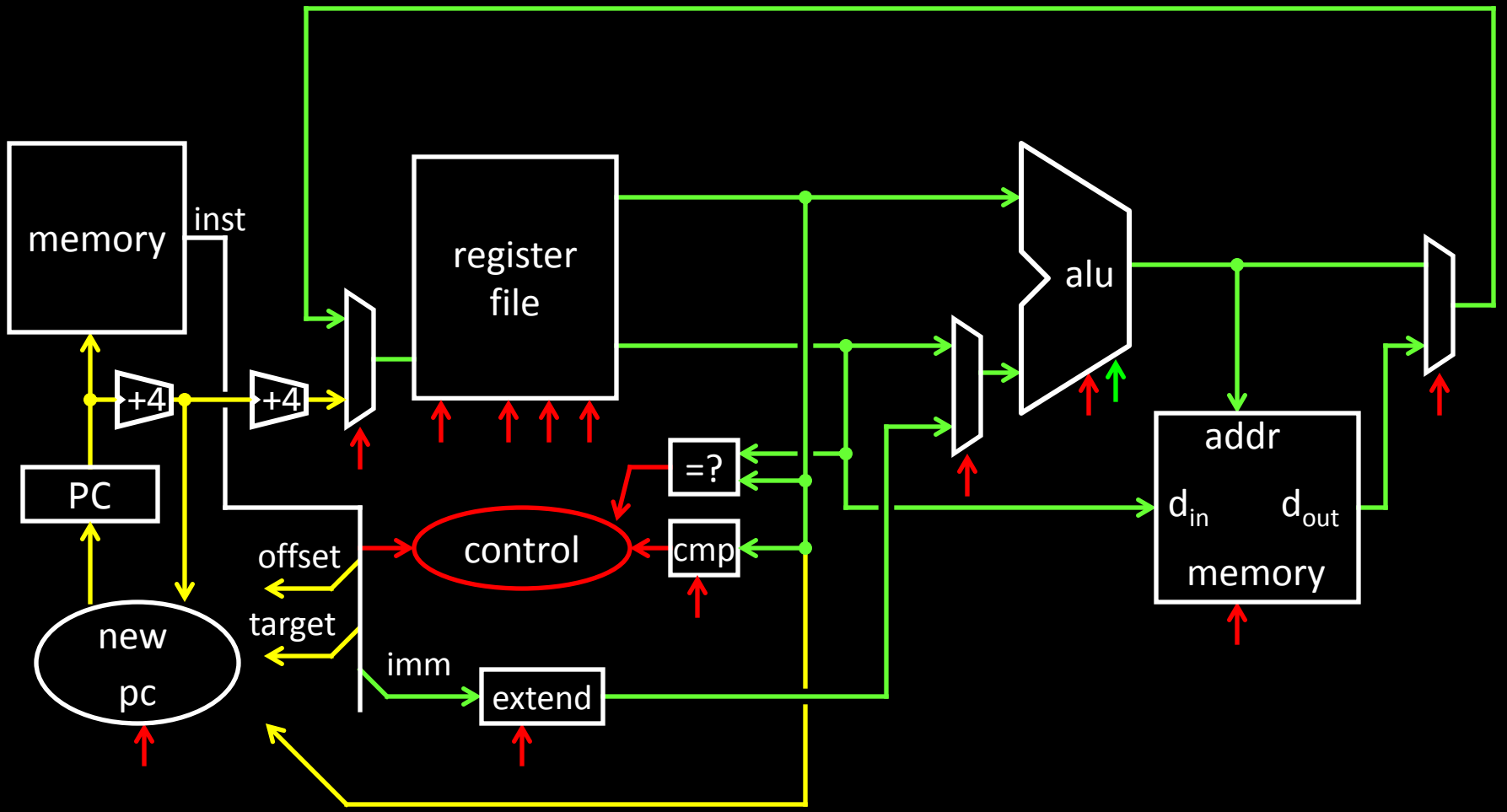
Throughput increased by parallel execution

Pipelining:

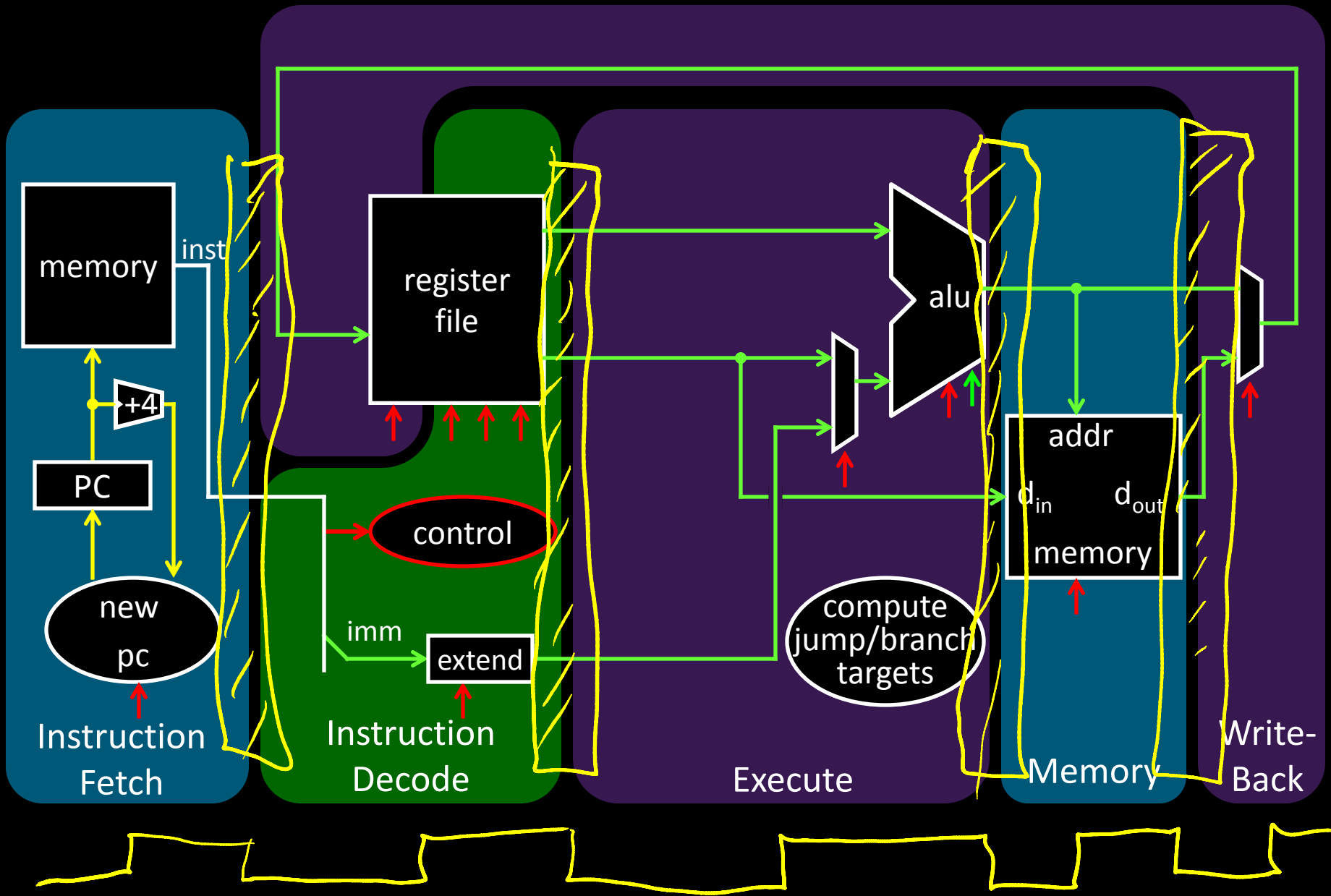
- Identify *pipeline stages*
- Isolate stages from each other
- Resolve pipeline *hazards* (Thursday)

A Processor

Review: Single cycle processor



A Processor

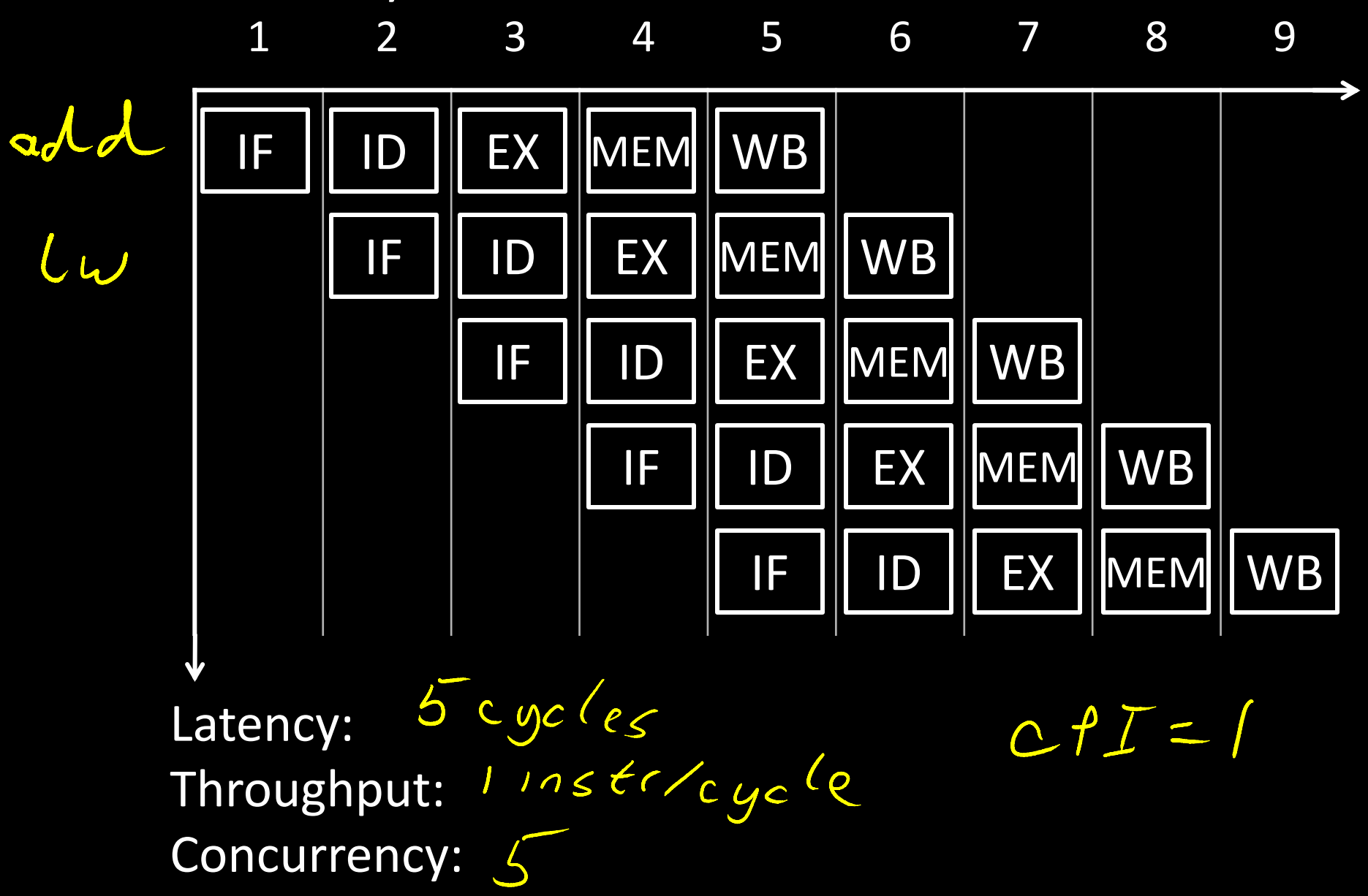

memory
inst
register file
alu
+4
PC
control
addr
d_{in}
d_{out}
memory
new pc
imm
extend
compute jump/branch targets
Instruction Fetch
Instruction Decode
Execute
Memory
Write-Back

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
lw
IF
ID
EX
MEM
WB
Latency: 5 cycles
Throughput: 1 instr/cycle
Concurrency: 5
CPI = 1

Principles of Pipelined Implementation

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

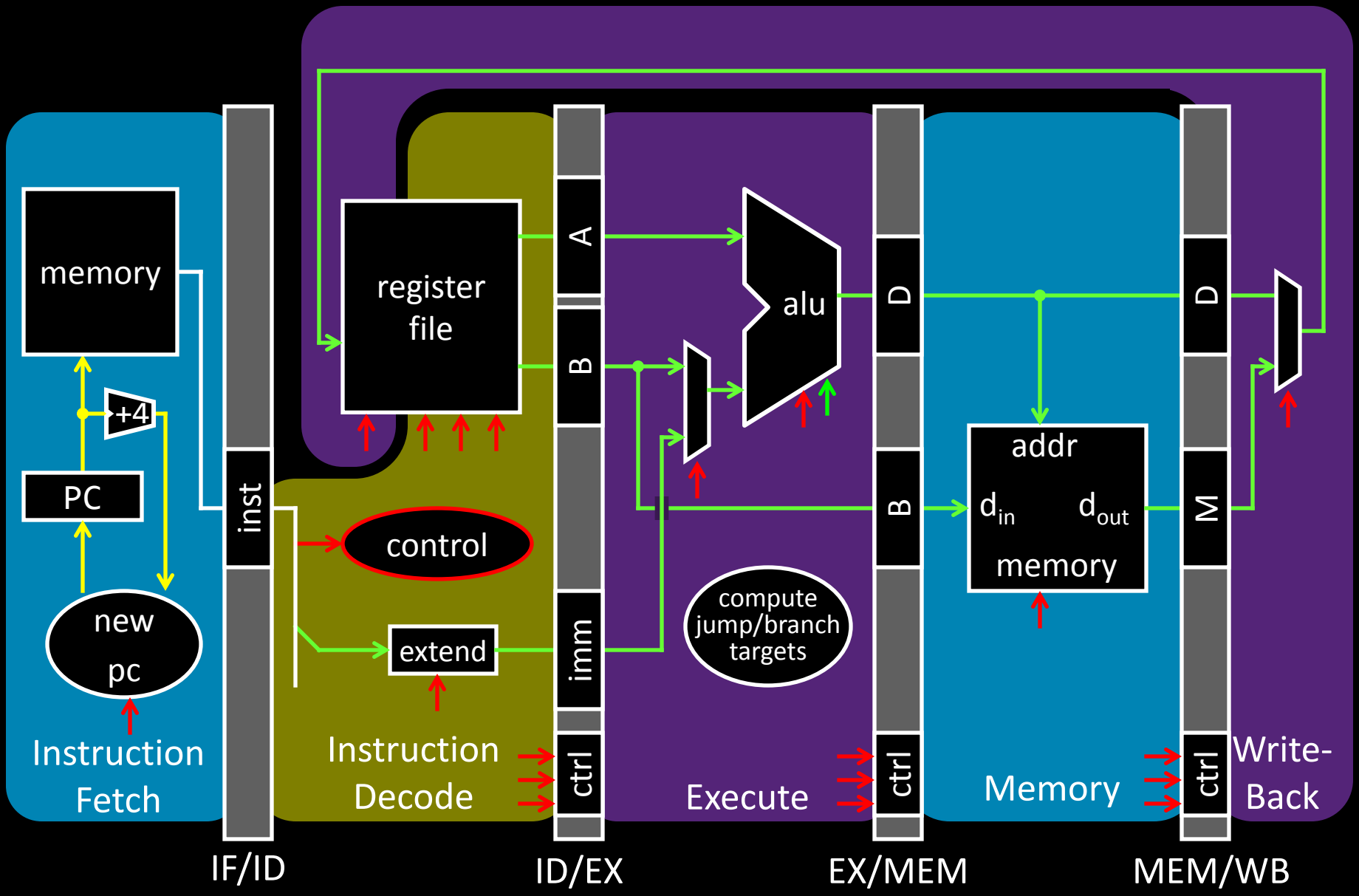
Design a separate stage for the execution performed during each clock cycle

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

Pipelined Processor

See: P&H Chapter 4.6

Pipelined Processor



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)

$2^{28} = 2^{20+8} = 2^{20}\,2^{8} = 256M$

Write values of interest to pipeline register (IF/ID)

- Instruction bits (for later decoding)
- PC+4 (for later computing branch targets)

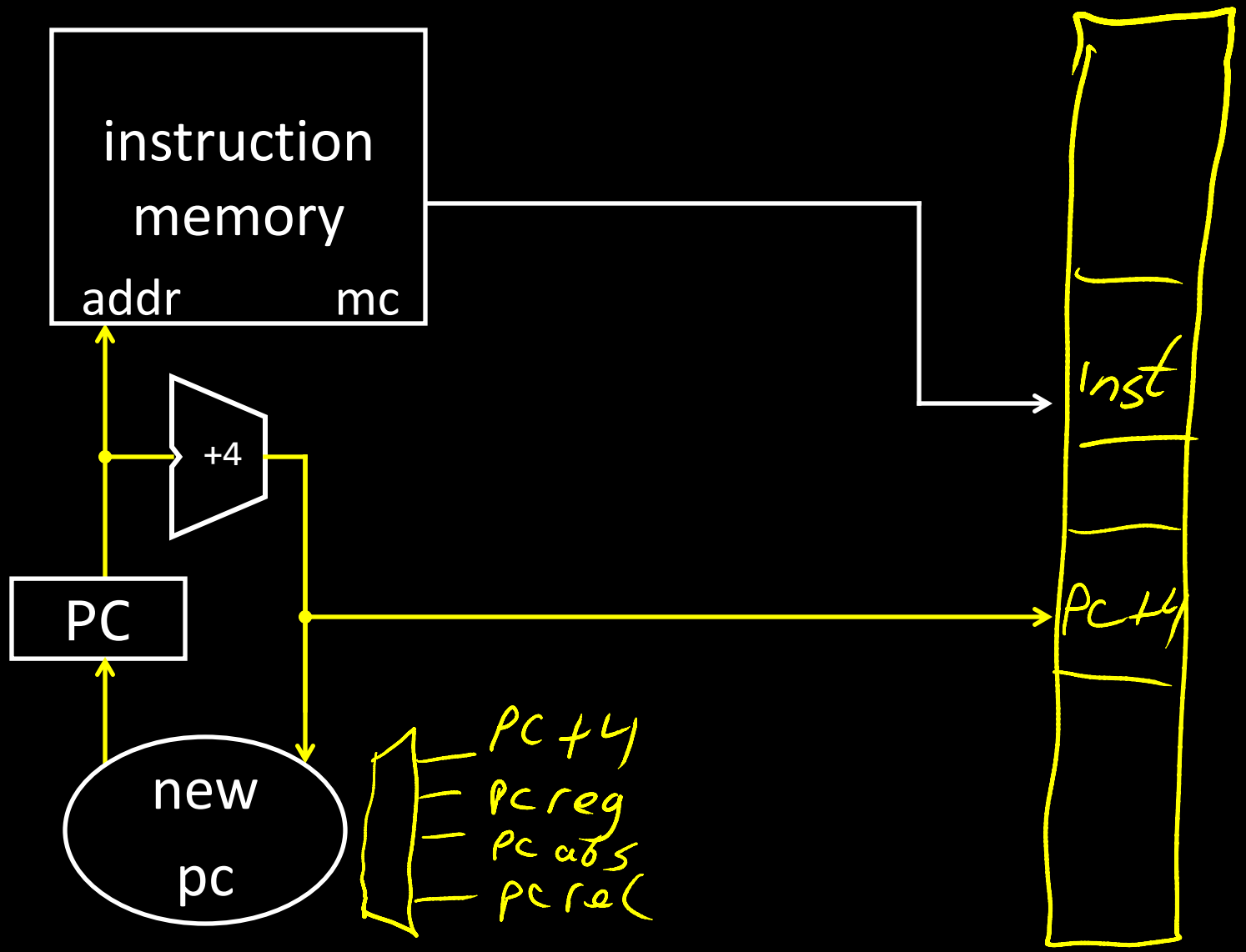
PC+4

BEQ PC rel

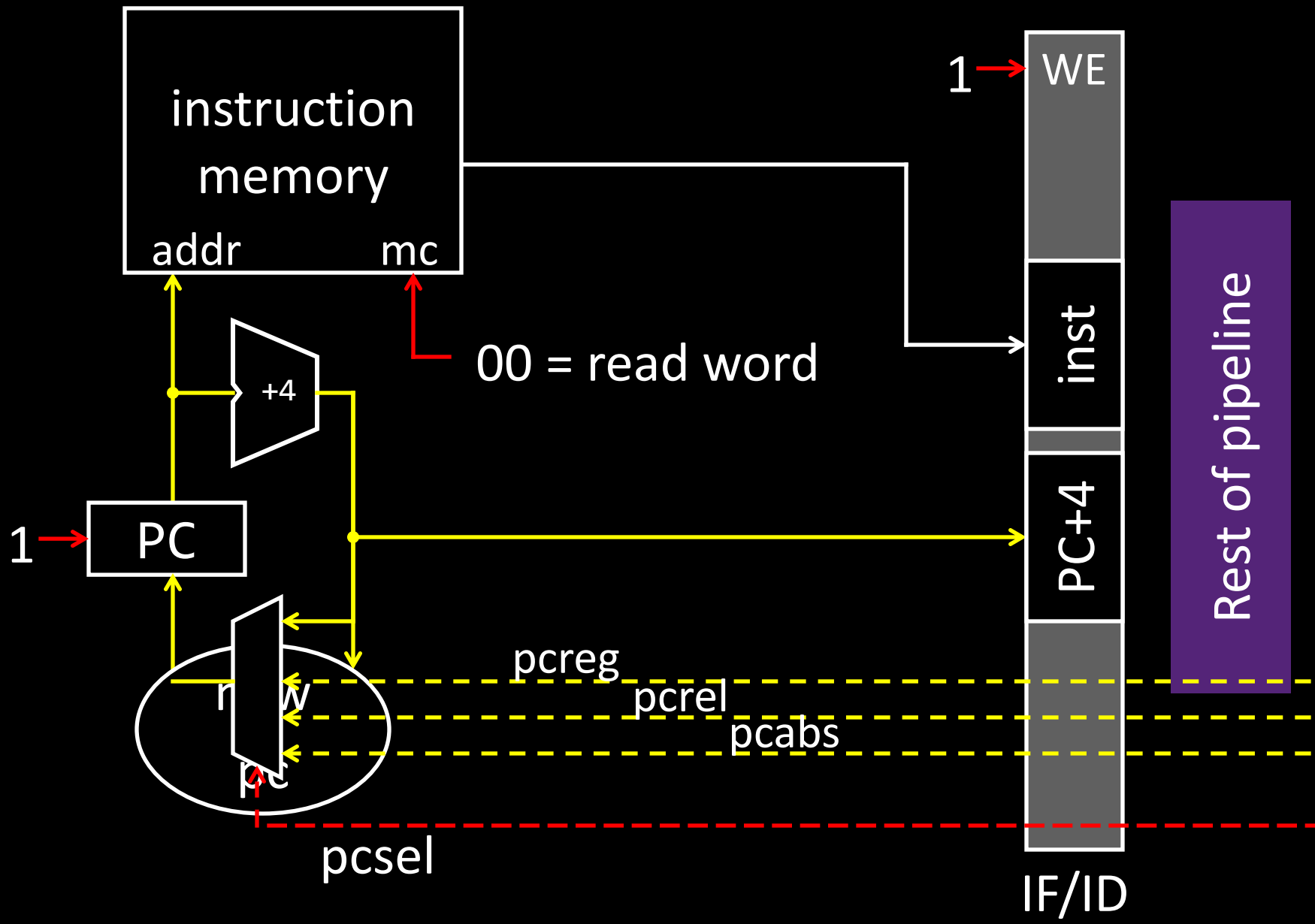
J PC abs $(PC+4)_{31\ldots28}$ (targ<<2)

JR PC reg

IF


instruction
memory
addr
mc
+4
PC
new
pc
PC+4
PCreg
PCabs
PCrel
Inst
PC+4

IF


instruction memory
addr
mc
00 = read word
1
WE
inst
PC+4
Rest of pipeline
+4
1
PC
pcreg
pcrel
pcabs
r
w
pc
pcsel
IF/ID

ID

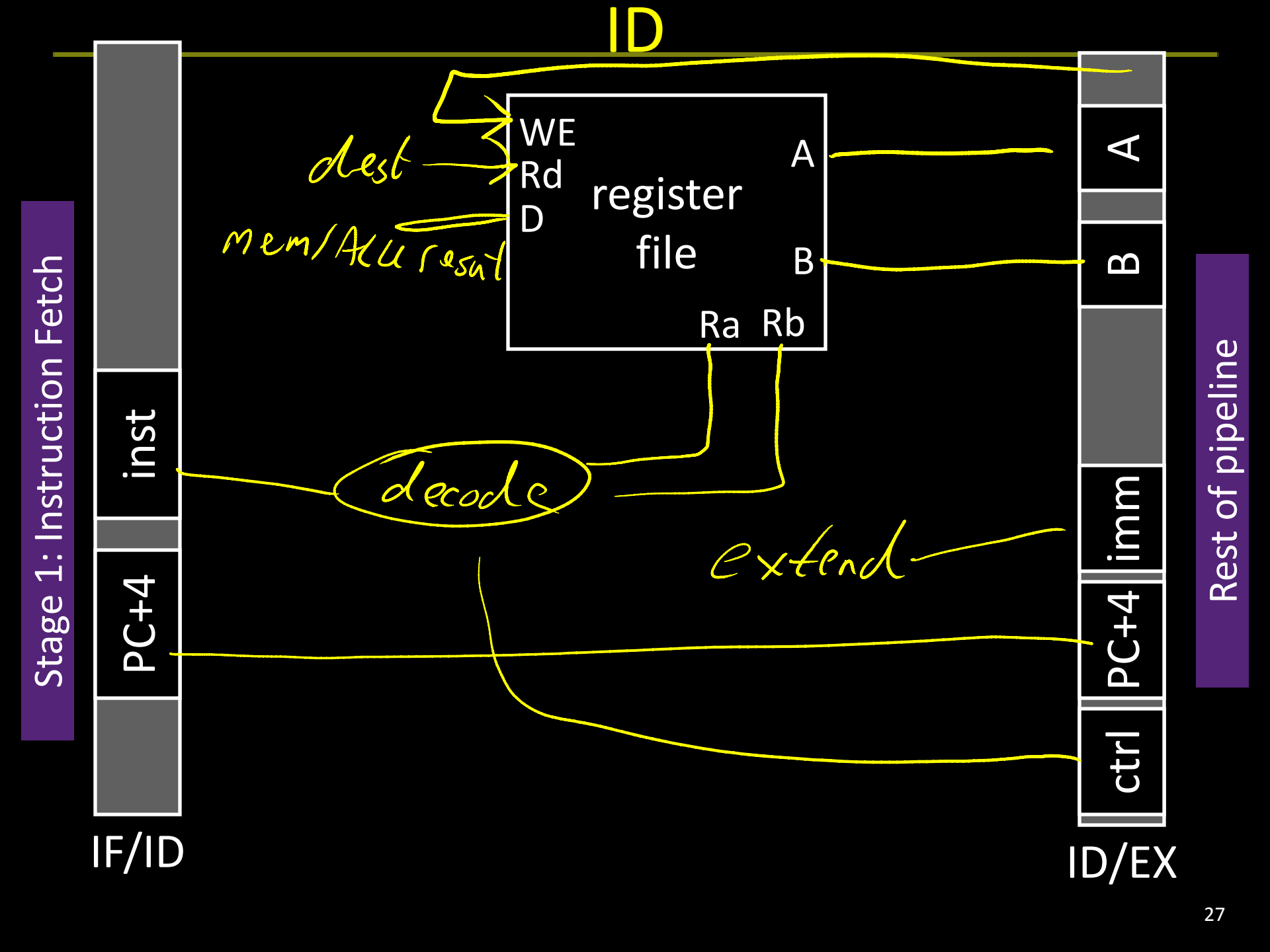
Stage 2: Instruction Decode

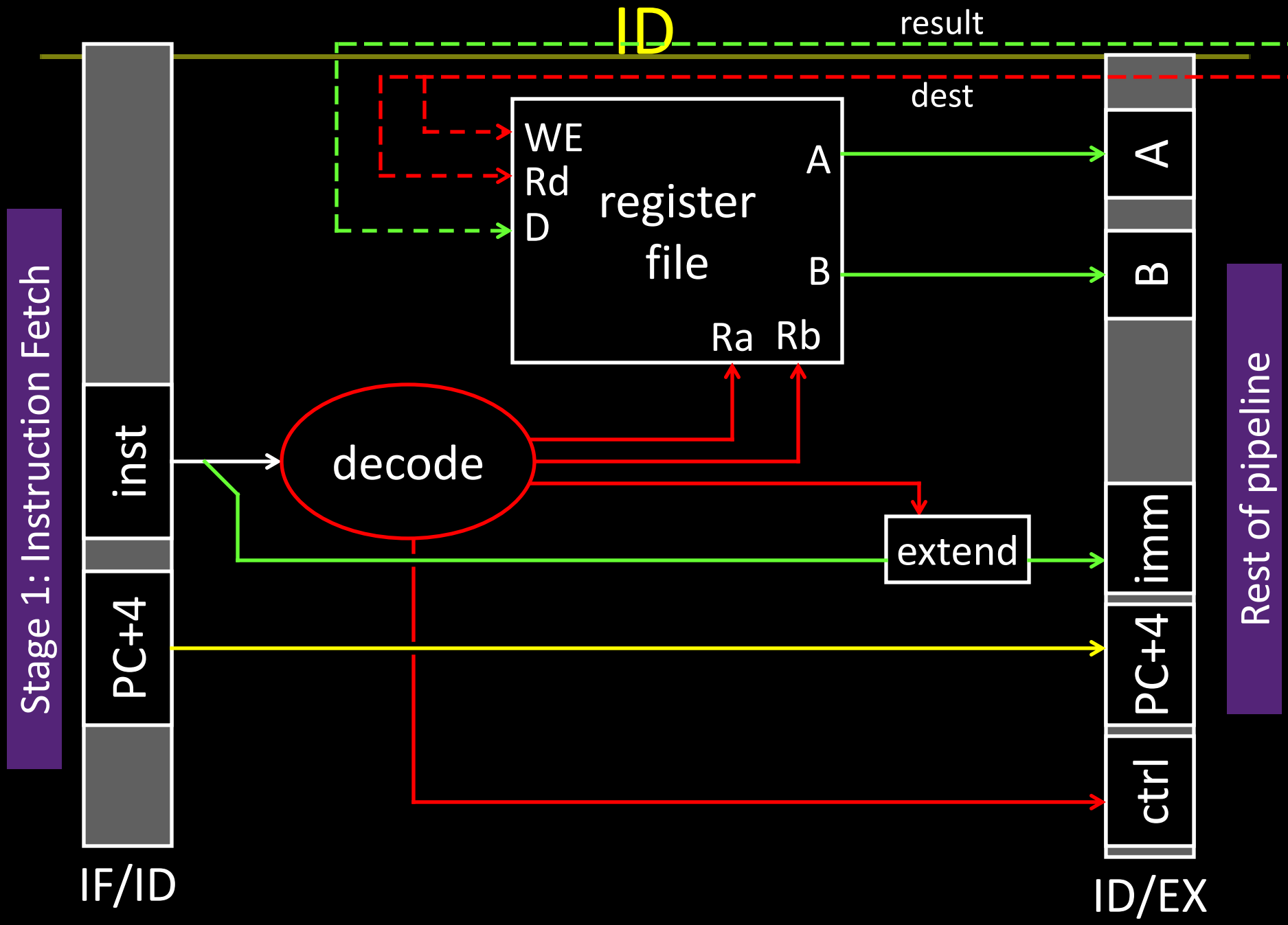
On every cycle:

- 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)


ID
WE
Rd
D
register file
A
B
Ra
Rb
dest
mem/ALU result
decode
extend
inst
PC+4
A
B
imm
PC+4
ctrl
Stage 1: Instruction Fetch
Rest of pipeline
IF/ID
ID/EX


ID
result
dest
WE
Rd
D
register file
A
B
Ra
Rb
inst
PC+4
decode
extend
A
B
imm
PC+4
ctrl
Stage 1: Instruction Fetch
Rest of pipeline
IF/ID
ID/EX

EX

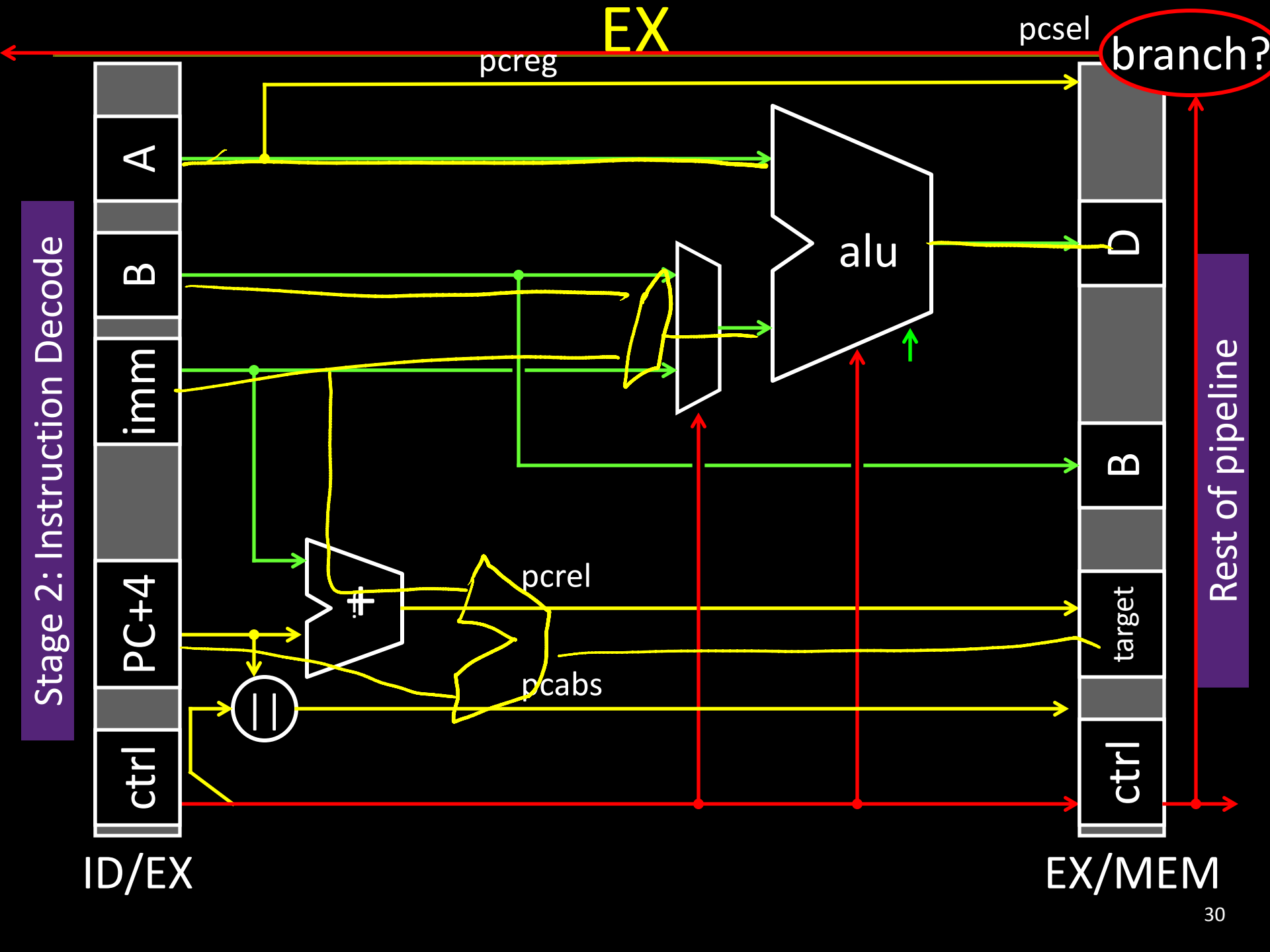
Stage 3: Execute

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
A
B
imm
PC+4
ctrl
alu
D
B
target
ctrl
pcrel
pcabs
+
=
Stage 2: Instruction Decode
Rest of pipeline
ID/EX
EX/MEM

MEM

Stage 4: Memory

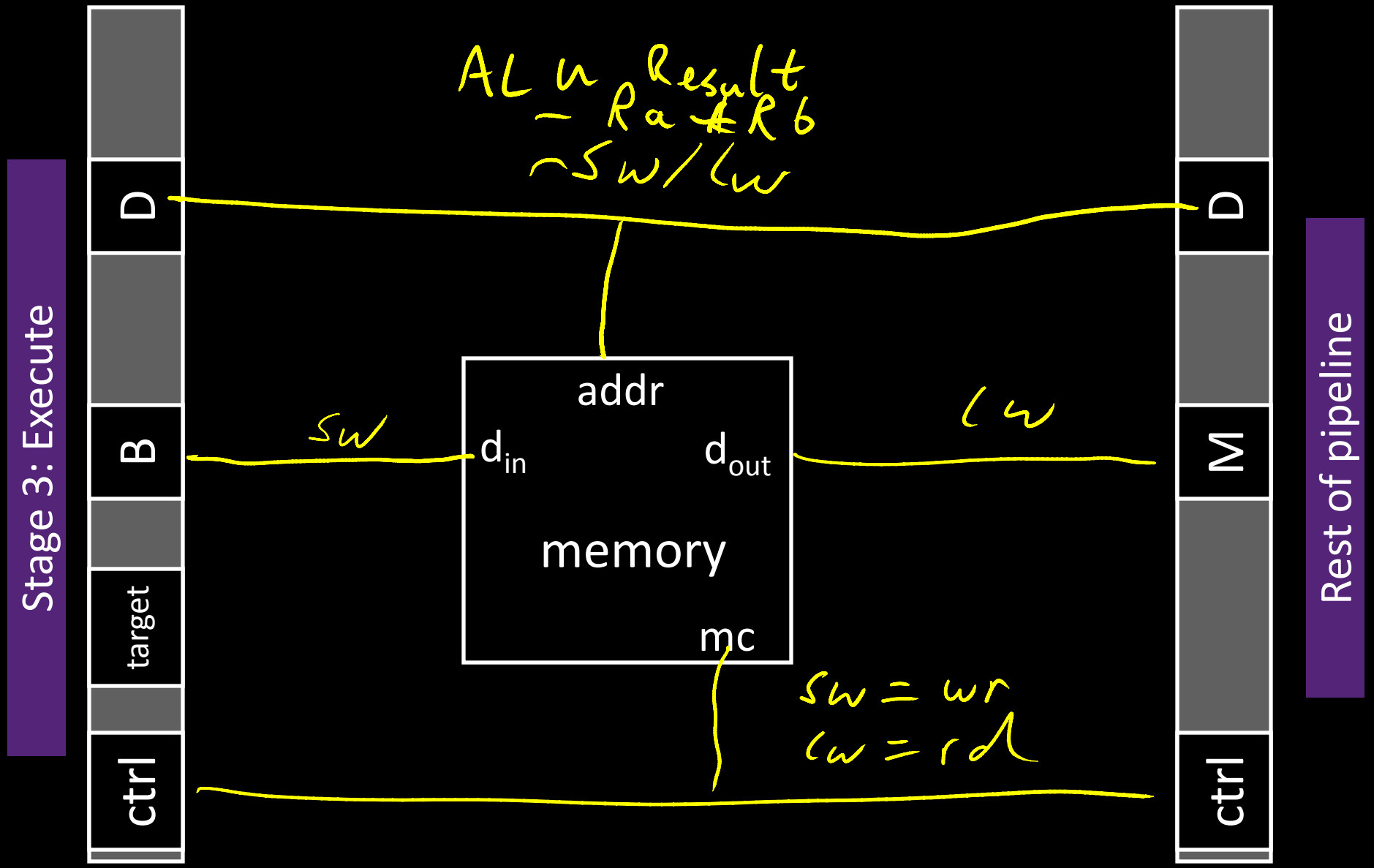
On every cycle:

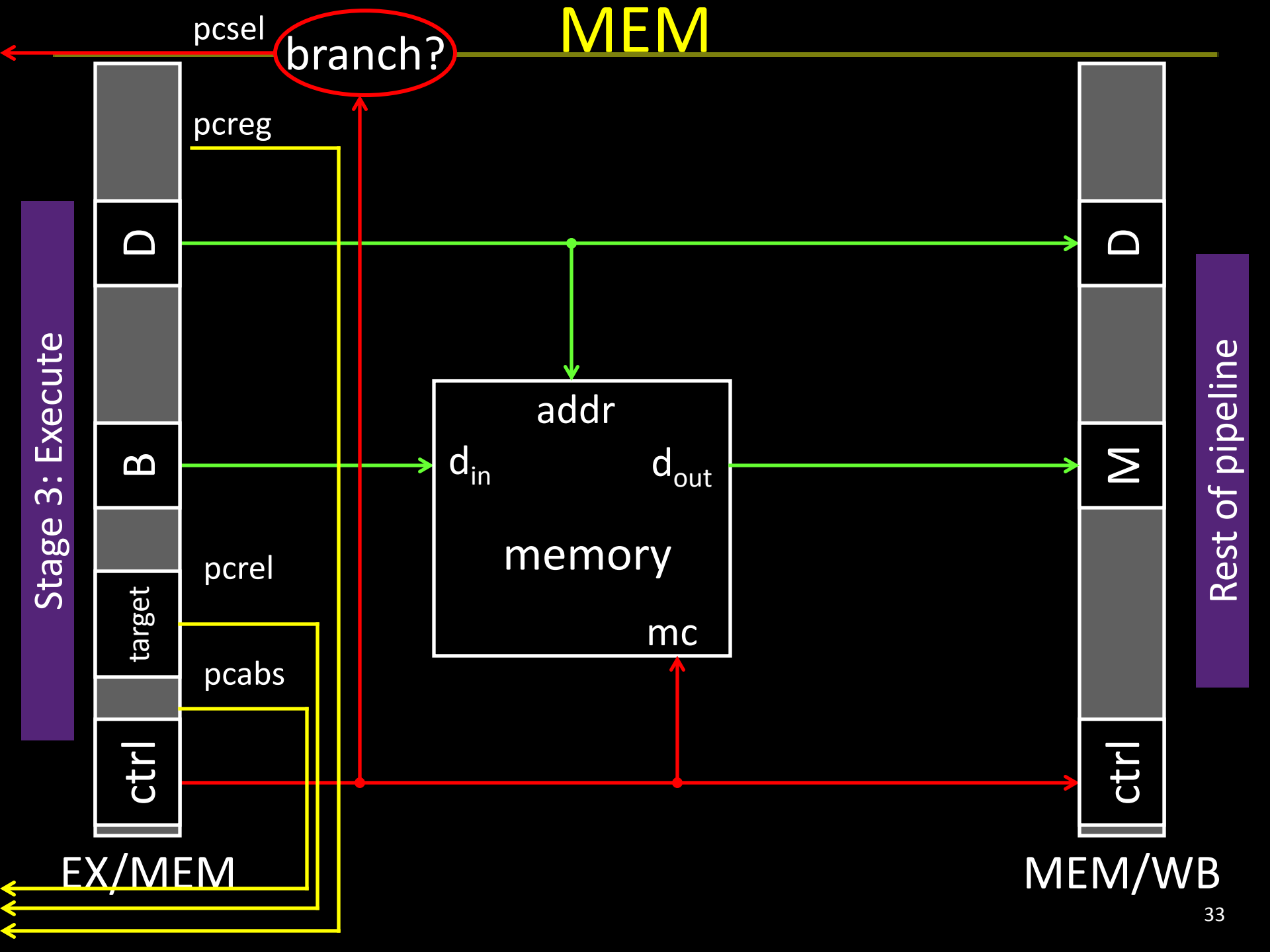
- Read EX/MEM pipeline register to get values and control bits
- Perform memory load/store if needed
  - address is ALU result

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
D
B
target
ctrl
EX/MEM
addr
d_{in}
d_{out}
memory
mc
sw
lw
ALU Result
= Ra + Rb
~sw/lw
sw = wr
lw = rd
D
M
ctrl
MEM/WB
Rest of pipeline


MEM
pcsel
branch?
pcreg
D
B
target
ctrl
pcrel
pcabs
addr
d_{in}
d_{out}
memory
mc
D
M
ctrl
Stage 3: Execute
Rest of pipeline
EX/MEM
MEM/WB

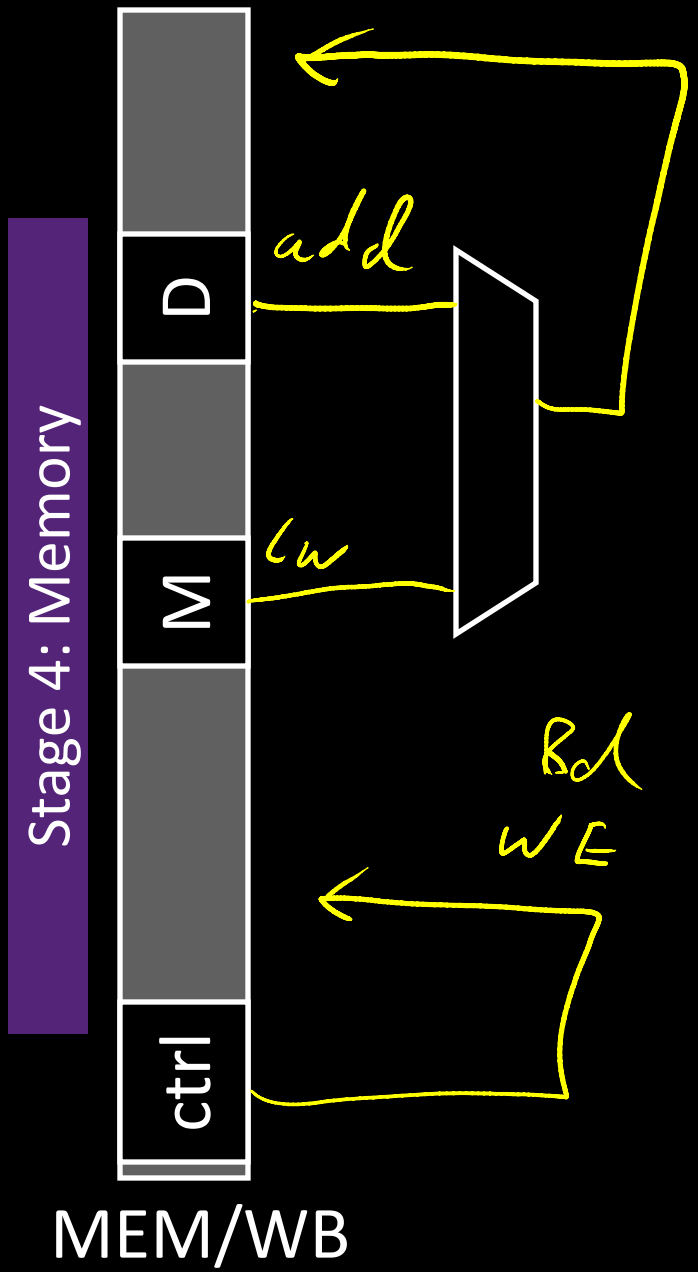
WB

Stage 5: Write-back

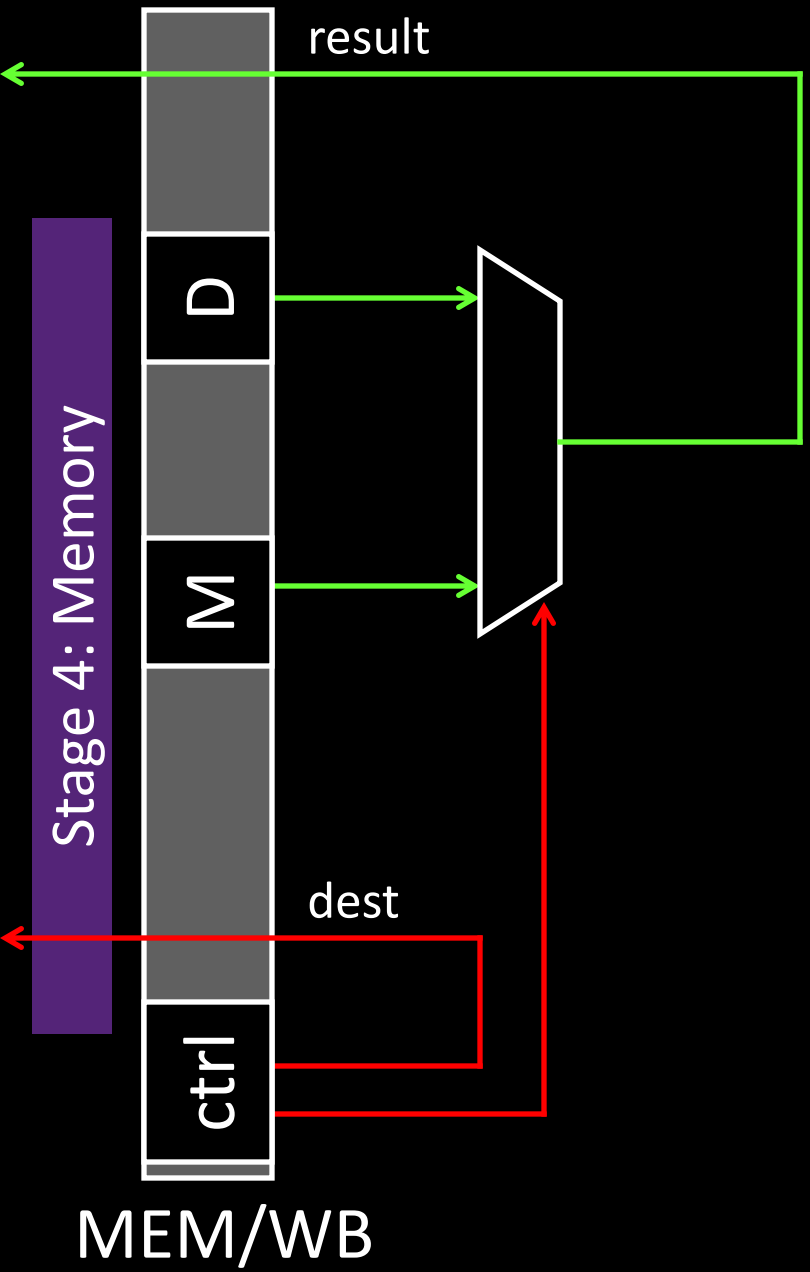
On every cycle:

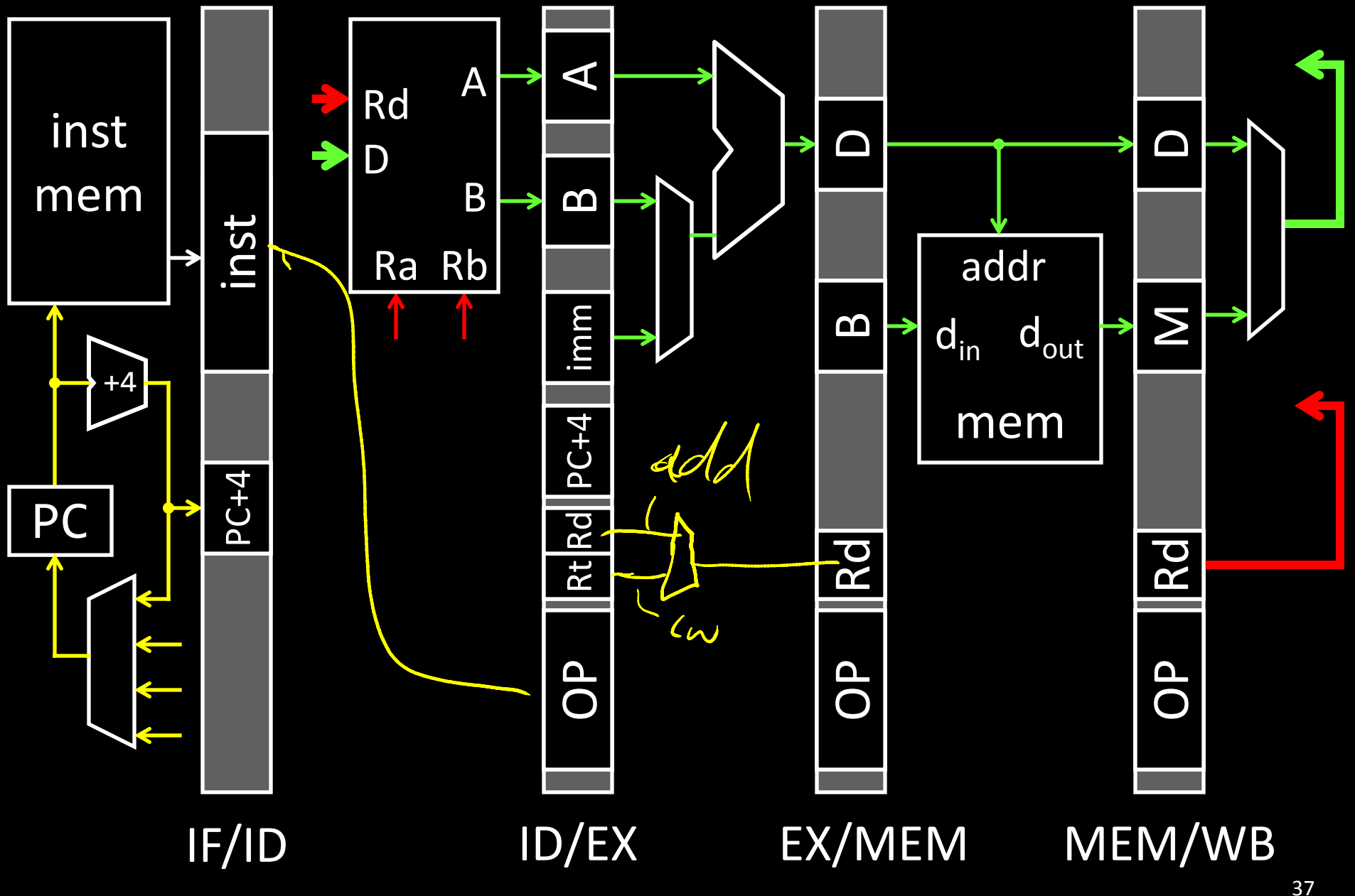
- Read MEM/WB pipeline register to get values and control bits
- Select value and write to register file

WB



WB




inst mem
+4
PC
inst
PC+4
Rd
D
A
B
Ra
Rb
A
B
imm
PC+4
Rt
Rd
OP
add
D
B
Rd
OP
addr
d_{in}
d_{out}
mem
D
M
Rd
OP
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);
```

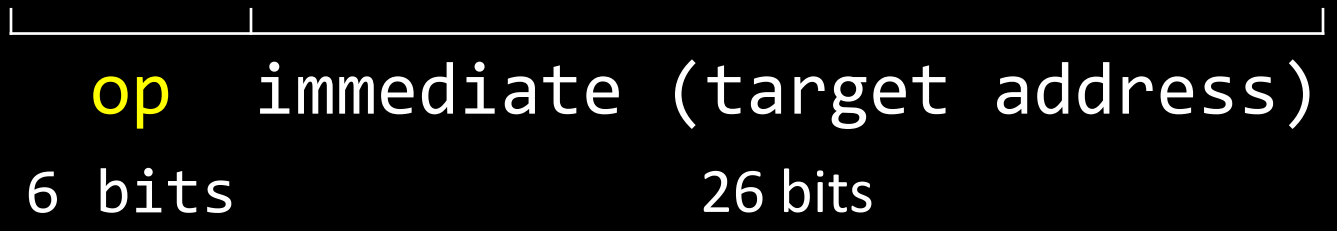
MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats

R-type

I-type

J-type



MIPS Instruction Types

Arithmetic/Logical

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

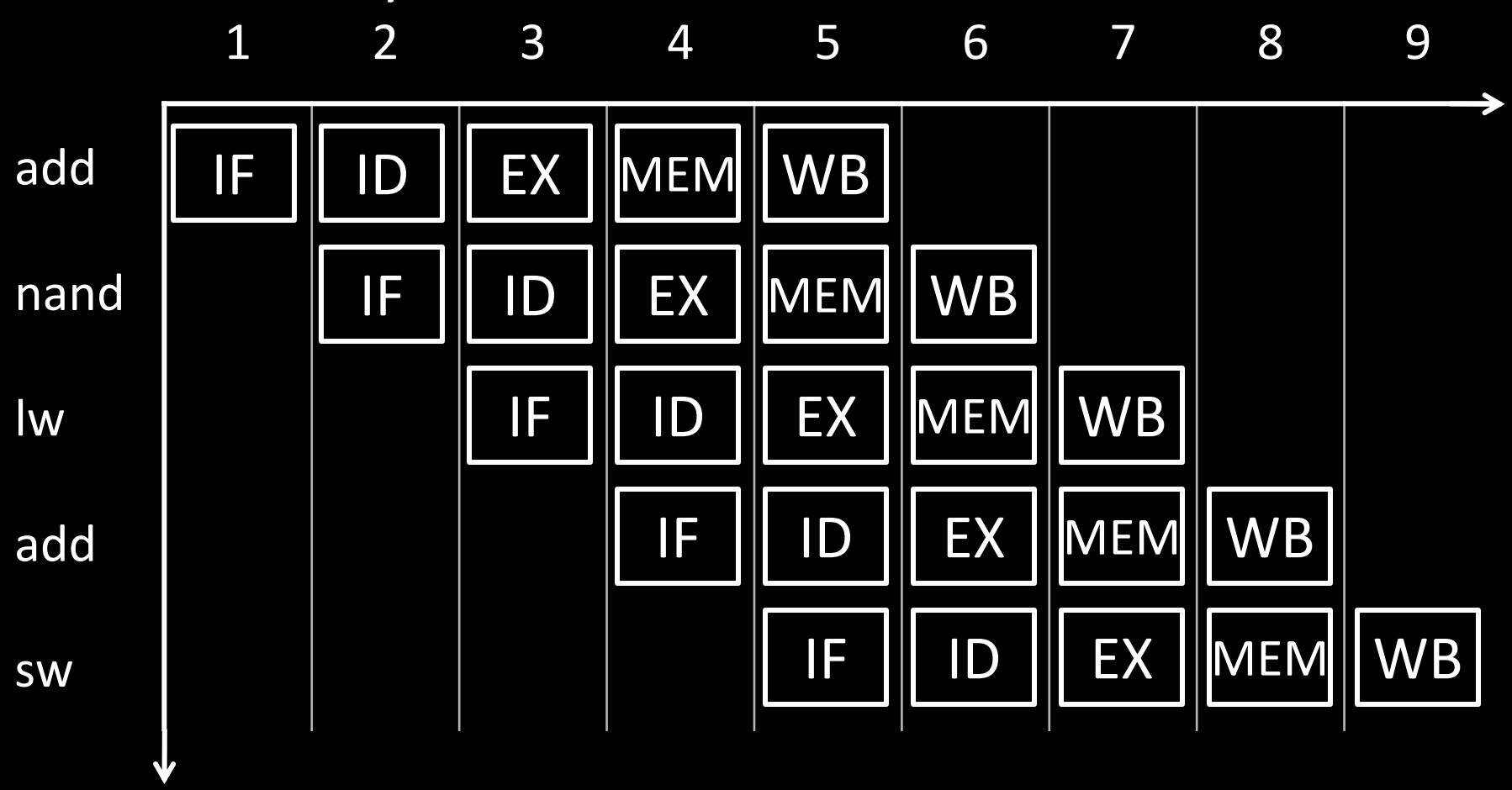
Memory Access

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

Control flow

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

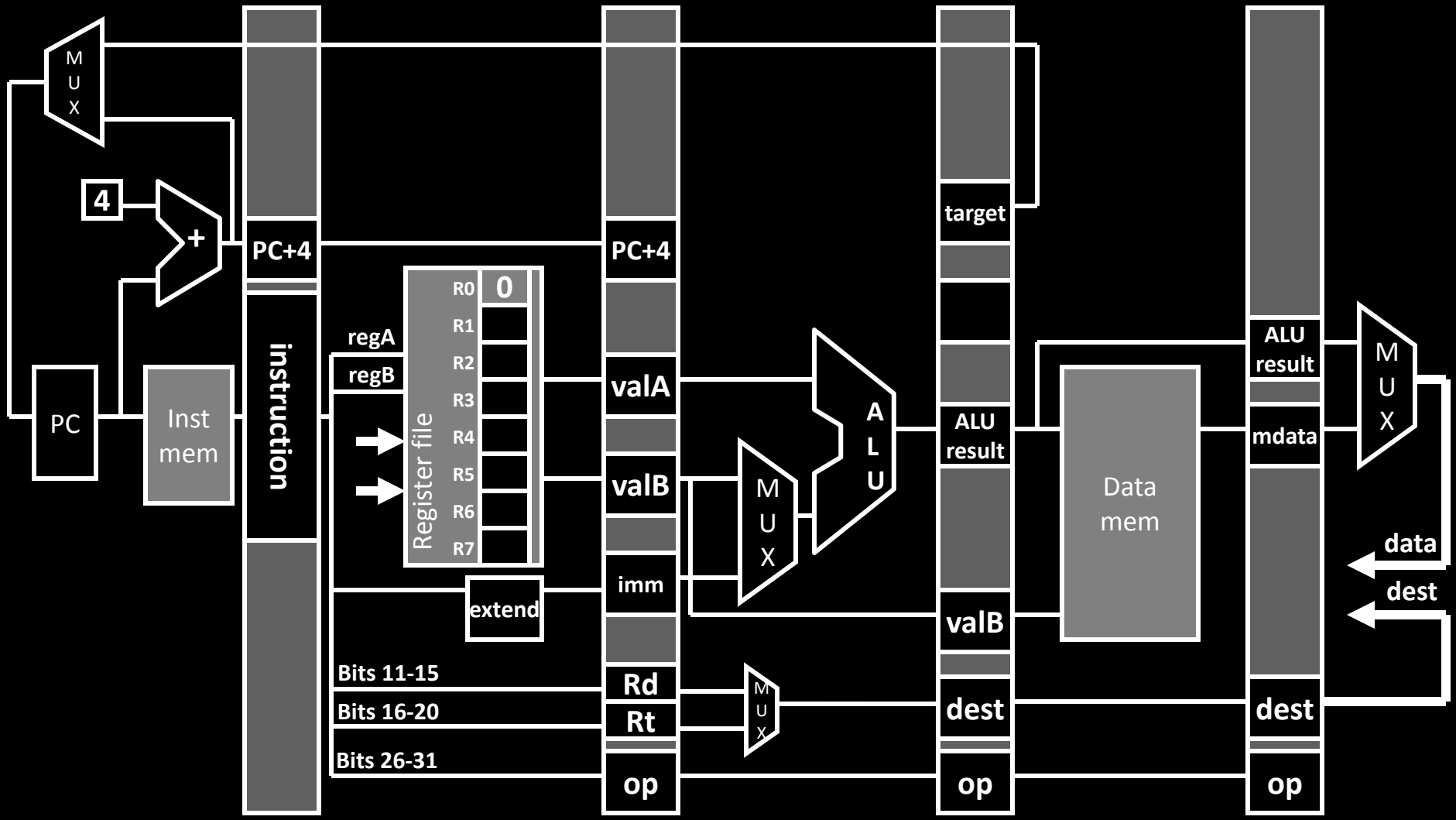
Time Graphs



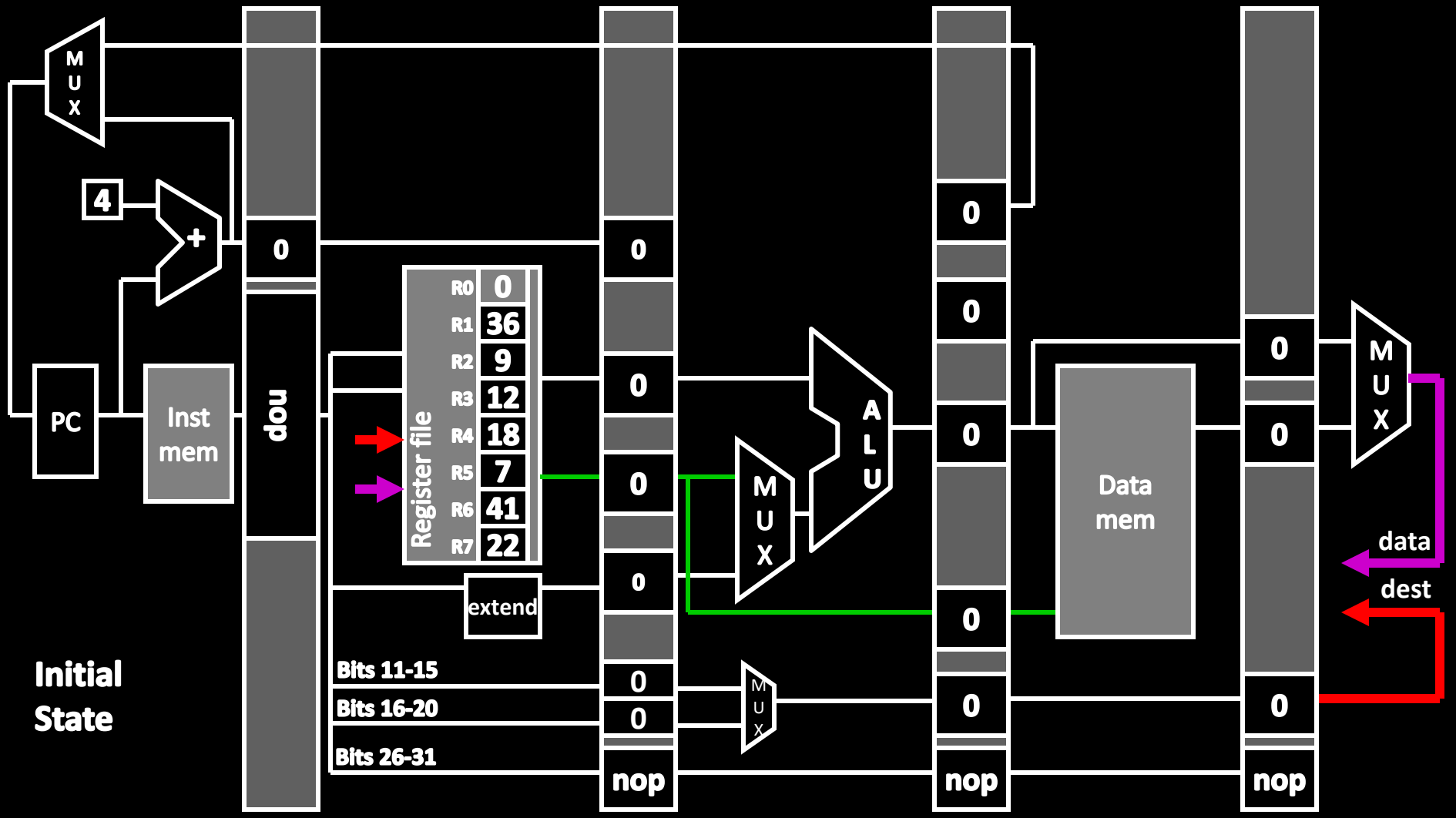
Latency:

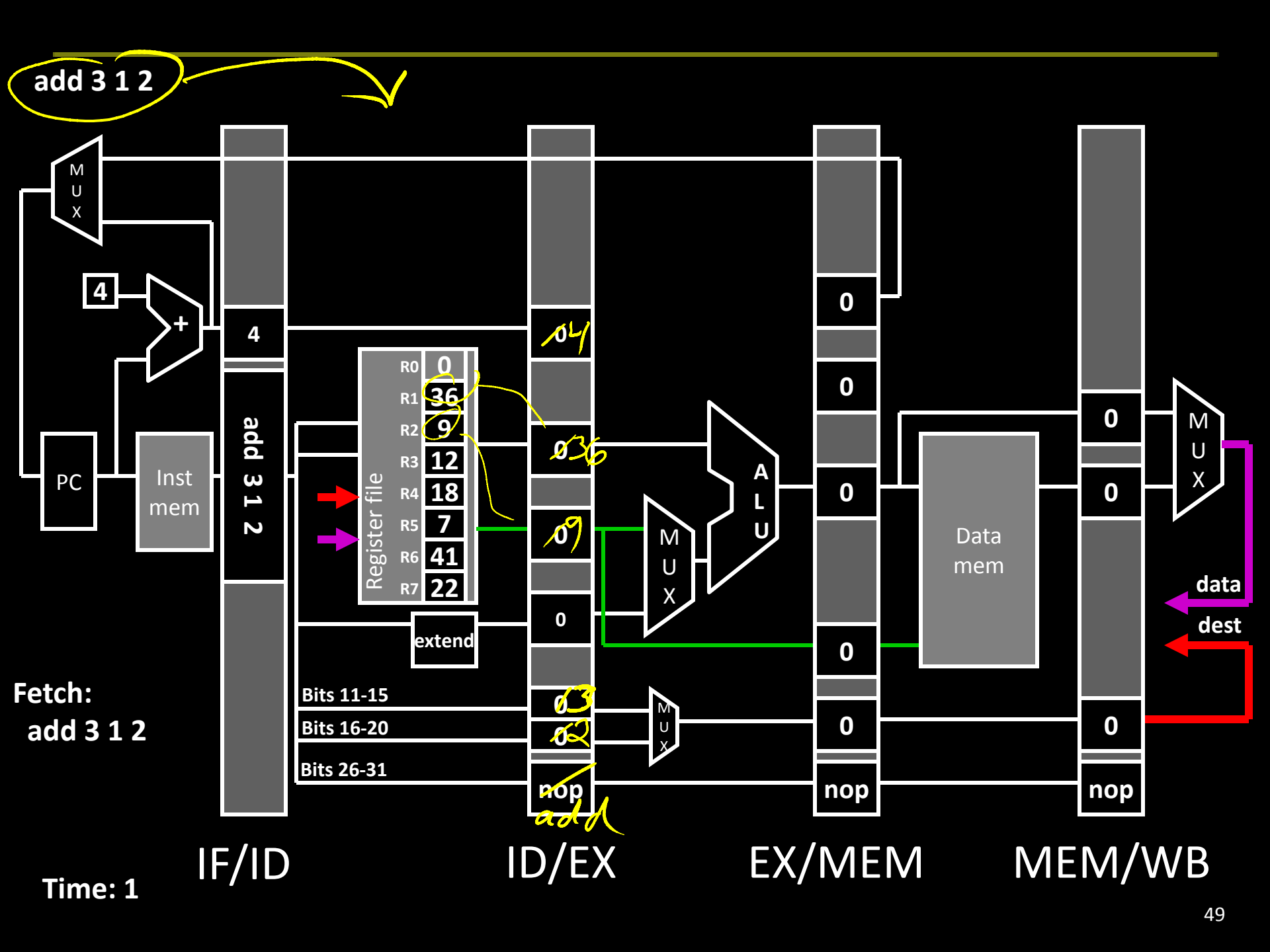
Throughput:

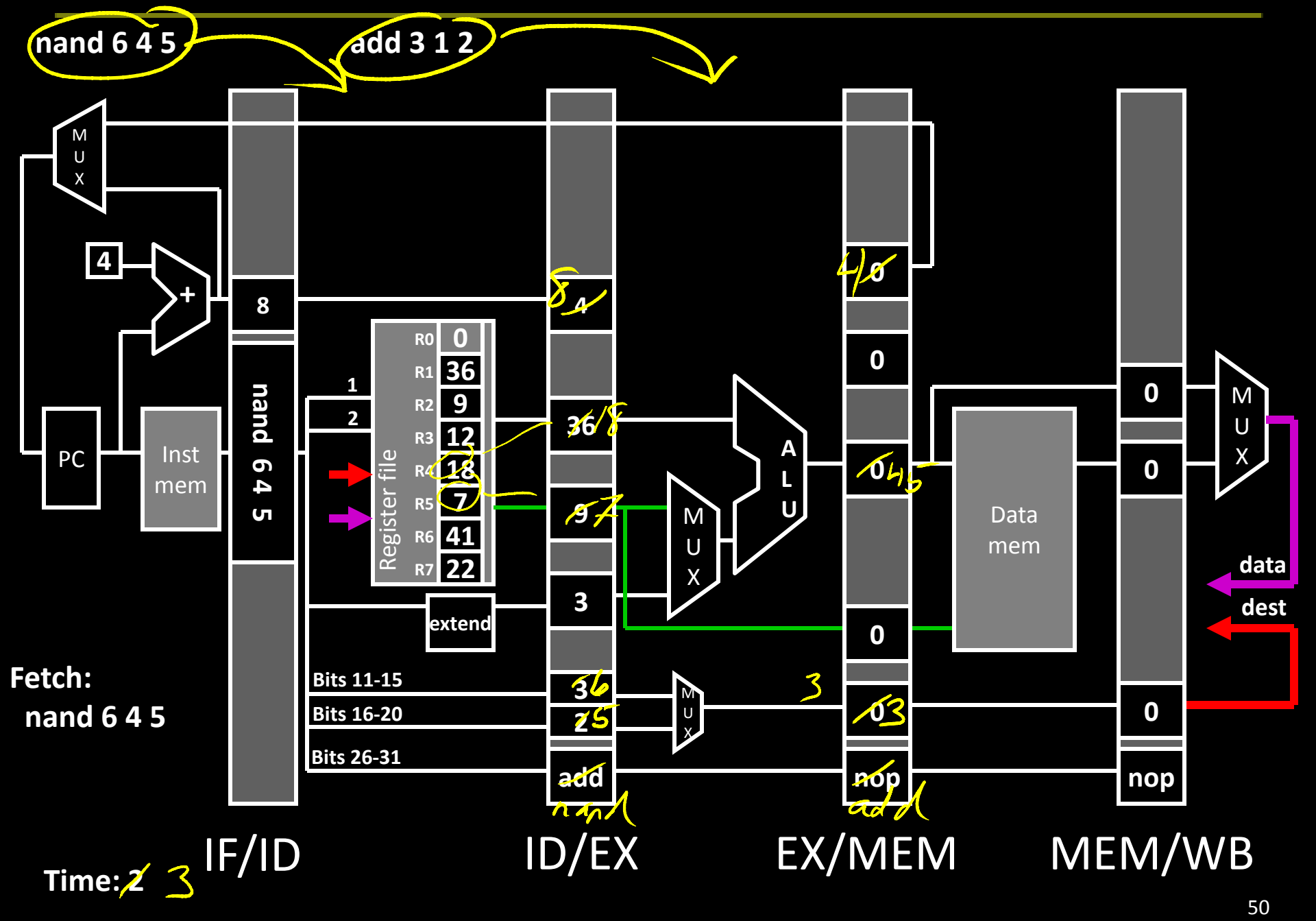
Concurrency:

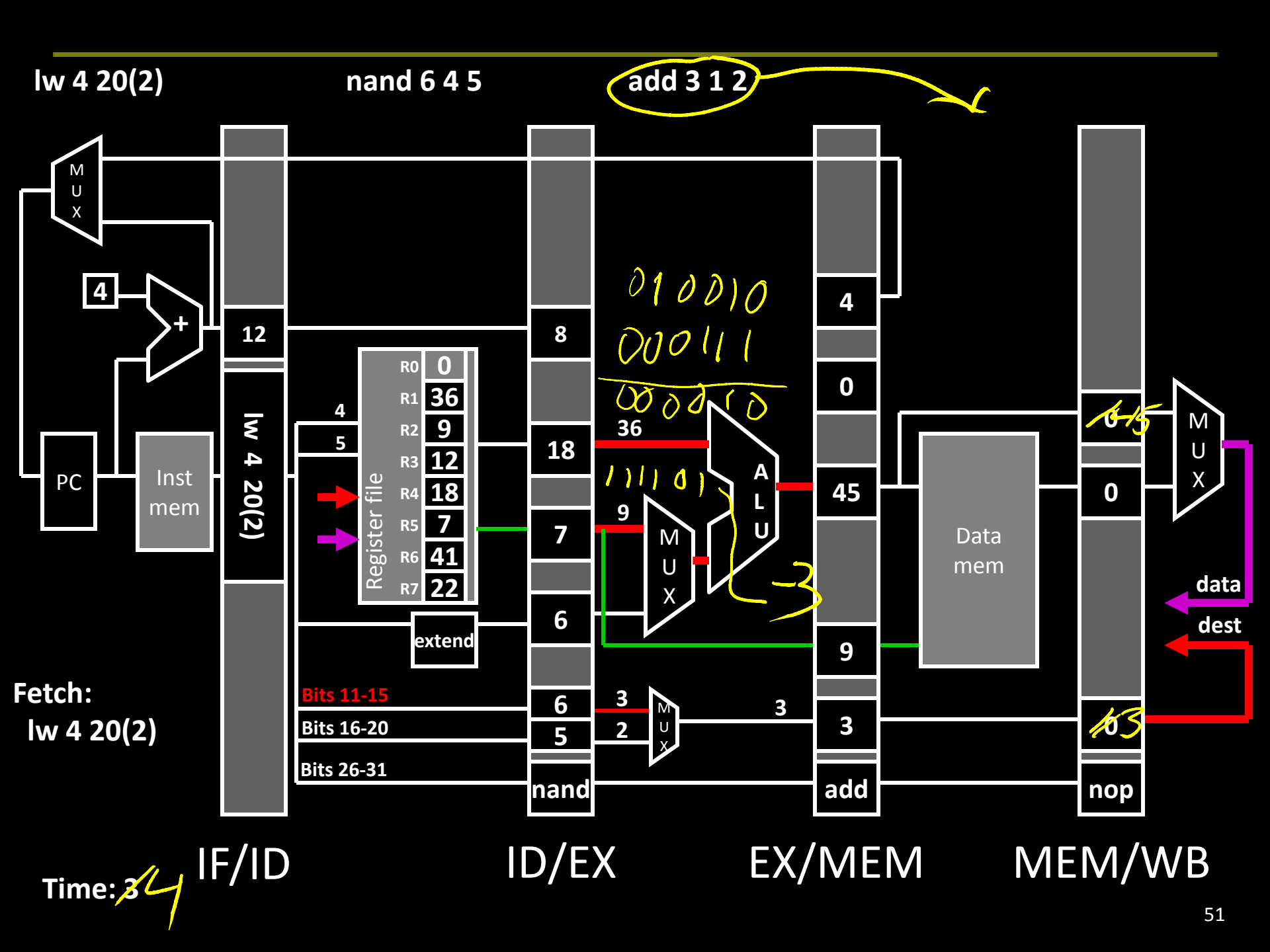

M
U
X
4
+
PC
Inst
mem
instruction
PC+4
regA
regB
Register file
R0
R1
R2
R3
R4
R5
R6
R7
0
extend
Bits 11-15
Bits 16-20
Bits 26-31
PC+4
valA
valB
imm
Rd
Rt
op
M
U
X
A
L
U
target
ALU
result
valB
dest
op
Data
mem
ALU
result
mdata
dest
op
M
U
X
data
dest
IF/ID
ID/EX
EX/MEM
MEM/WB

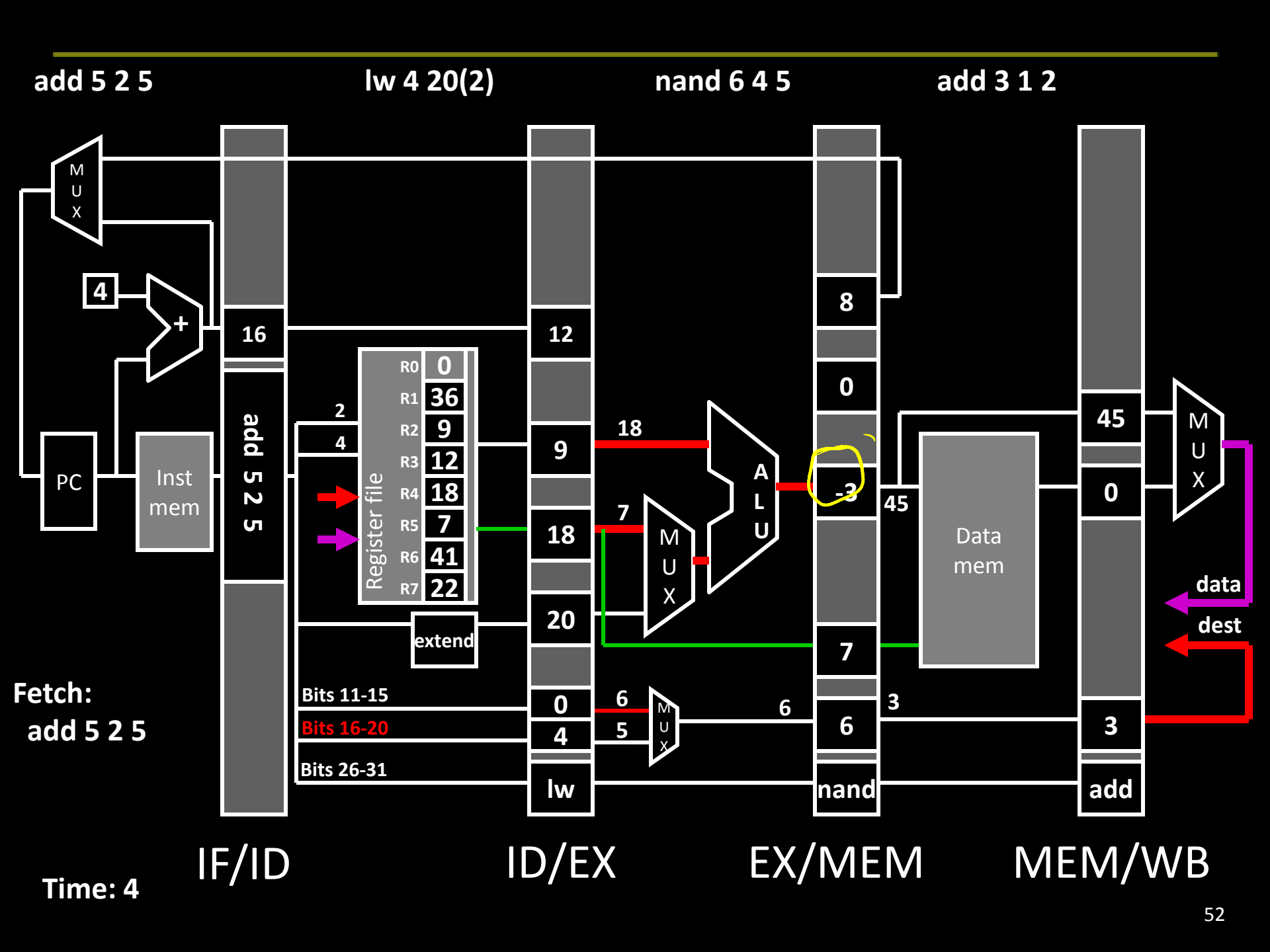
add r3 r1 r2

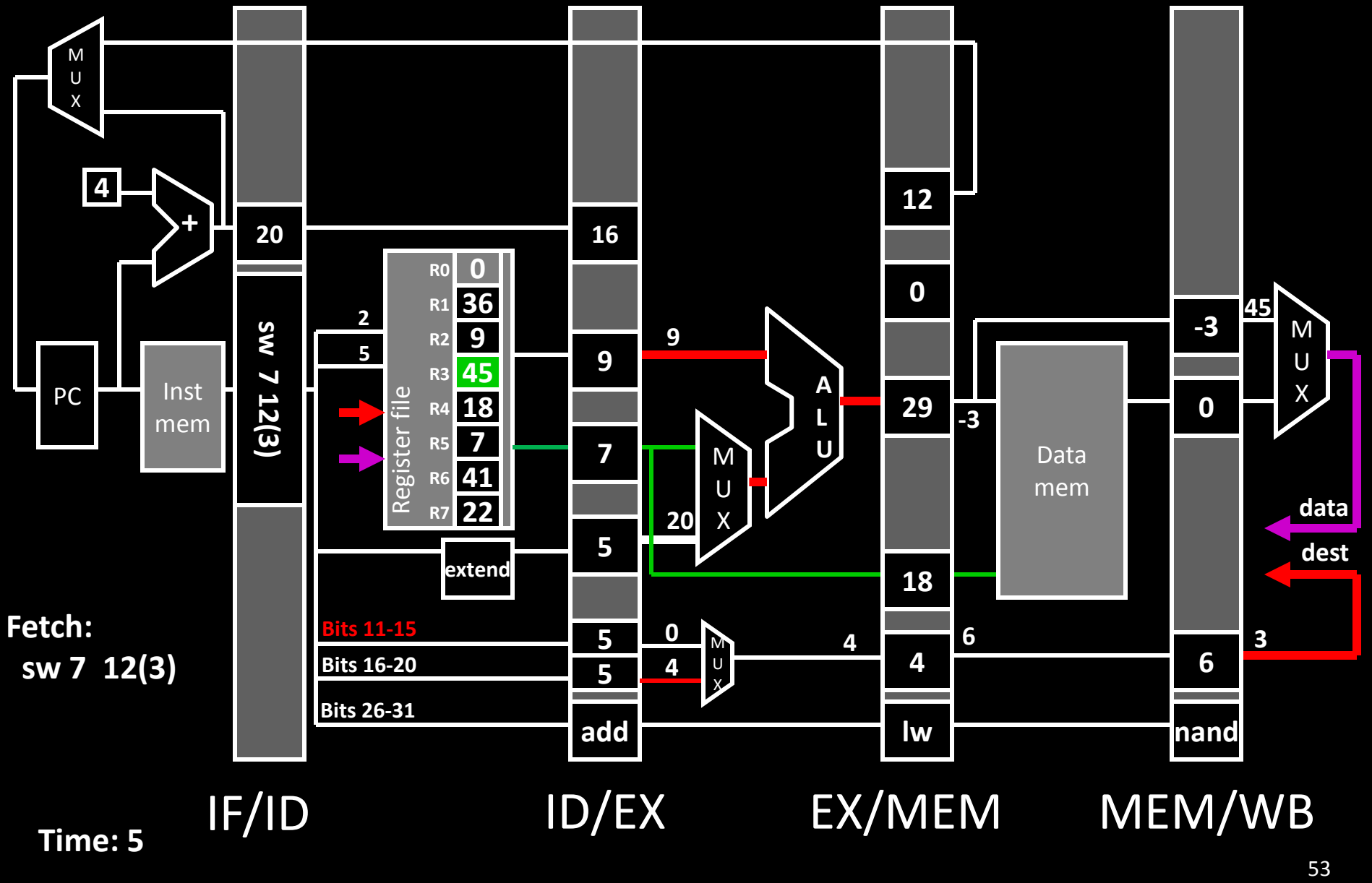

MUX
4
+
PC
Inst mem
dou
0
Register file
R0 0
R1 36
R2 9
R3 12
R4 18
R5 7
R6 41
R7 22
extend
Bits 11-15
Bits 16-20
Bits 26-31
MUX
ALU
MUX
Data mem
MUX
data
dest
nop
Initial State
IF/ID
ID/EX
EX/MEM
MEM/WB

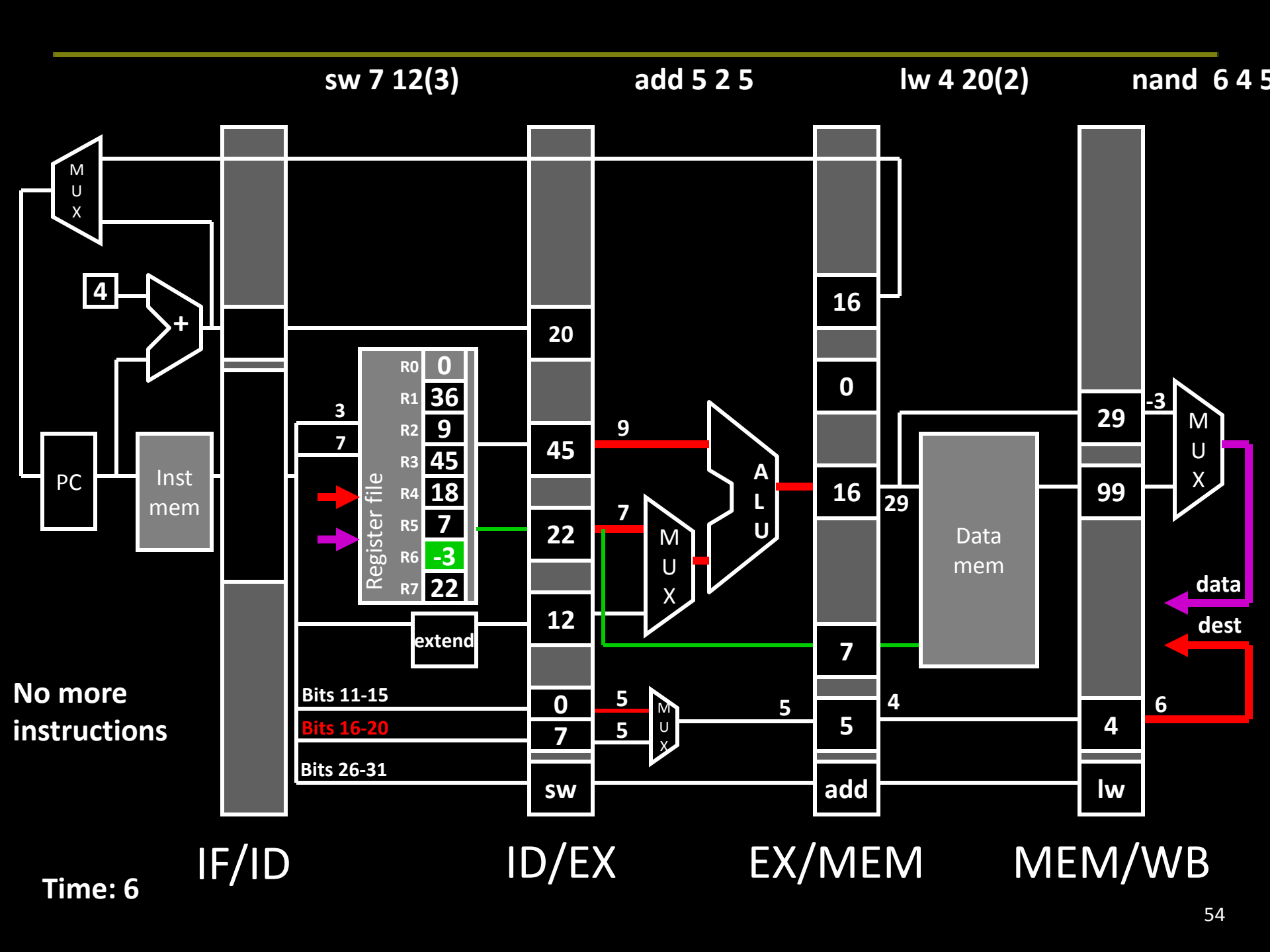

add 3 1 2
M U X
4
+
4
PC
Inst mem
add 3 1 2
Register file
R0 0
R1 36
R2 9
R3 12
R4 18
R5 7
R6 41
R7 22
extend
Bits 11-15
Bits 16-20
Bits 26-31
0
0
0
0
nop
M U X
A L U
0
0
0
0
0
nop
Data mem
0
0
0
nop
M U X
data
dest
Fetch:
add 3 1 2
Time: 1
IF/ID
ID/EX
EX/MEM
MEM/WB

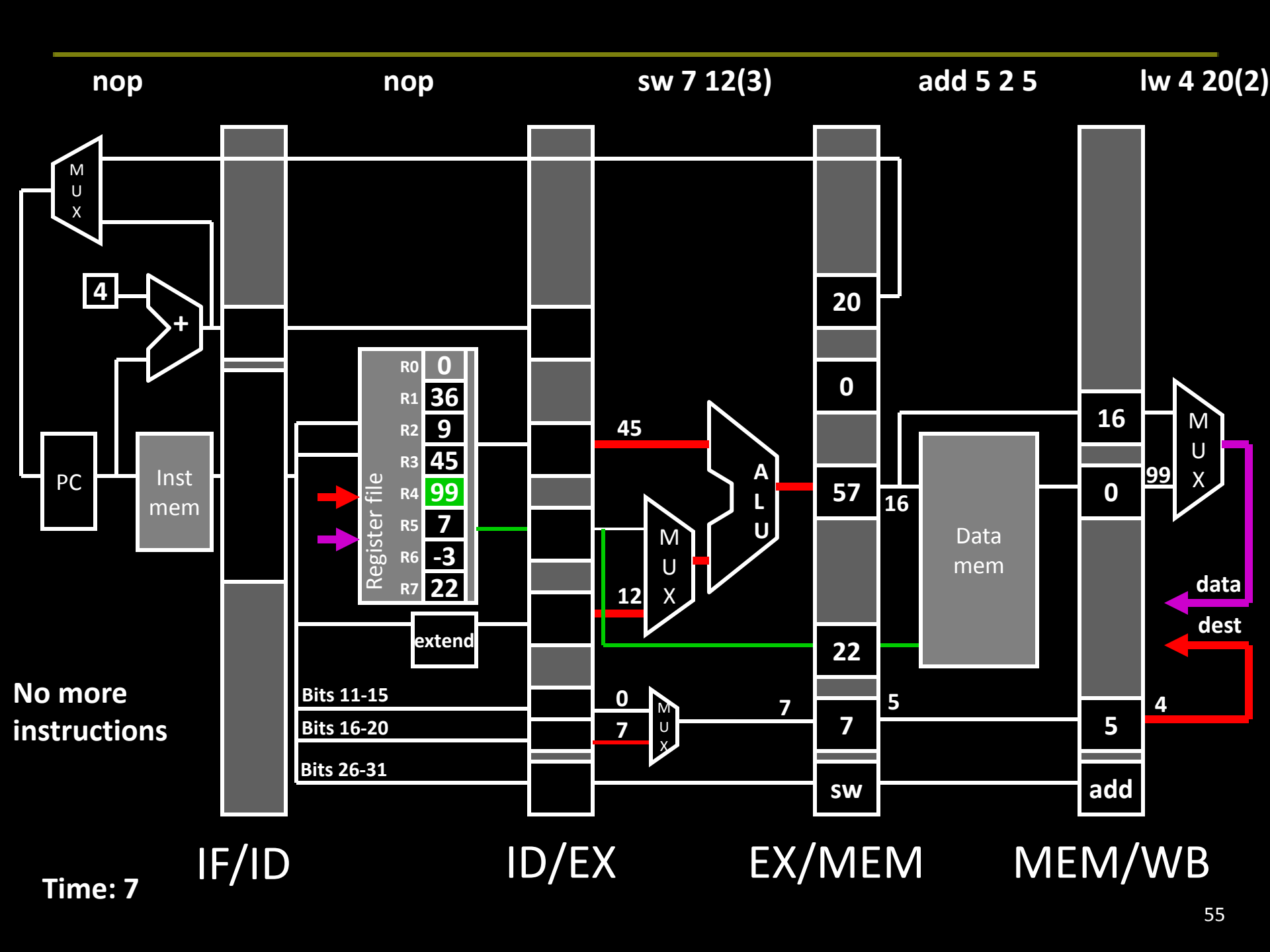

nand 6 4 5
add 3 1 2
MUX
4
+
8
PC
Inst mem
nand 6 4 5
1
2
Register file
R0 0
R1 36
R2 9
R3 12
R4 18
R5 7
R6 41
R7 22
extend
Bits 11-15
Bits 16-20
Bits 26-31
3
add
MUX
ALU
nop
Data mem
0
data
dest
Fetch:
nand 6 4 5
Time: 2 3
IF/ID
ID/EX
EX/MEM
MEM/WB

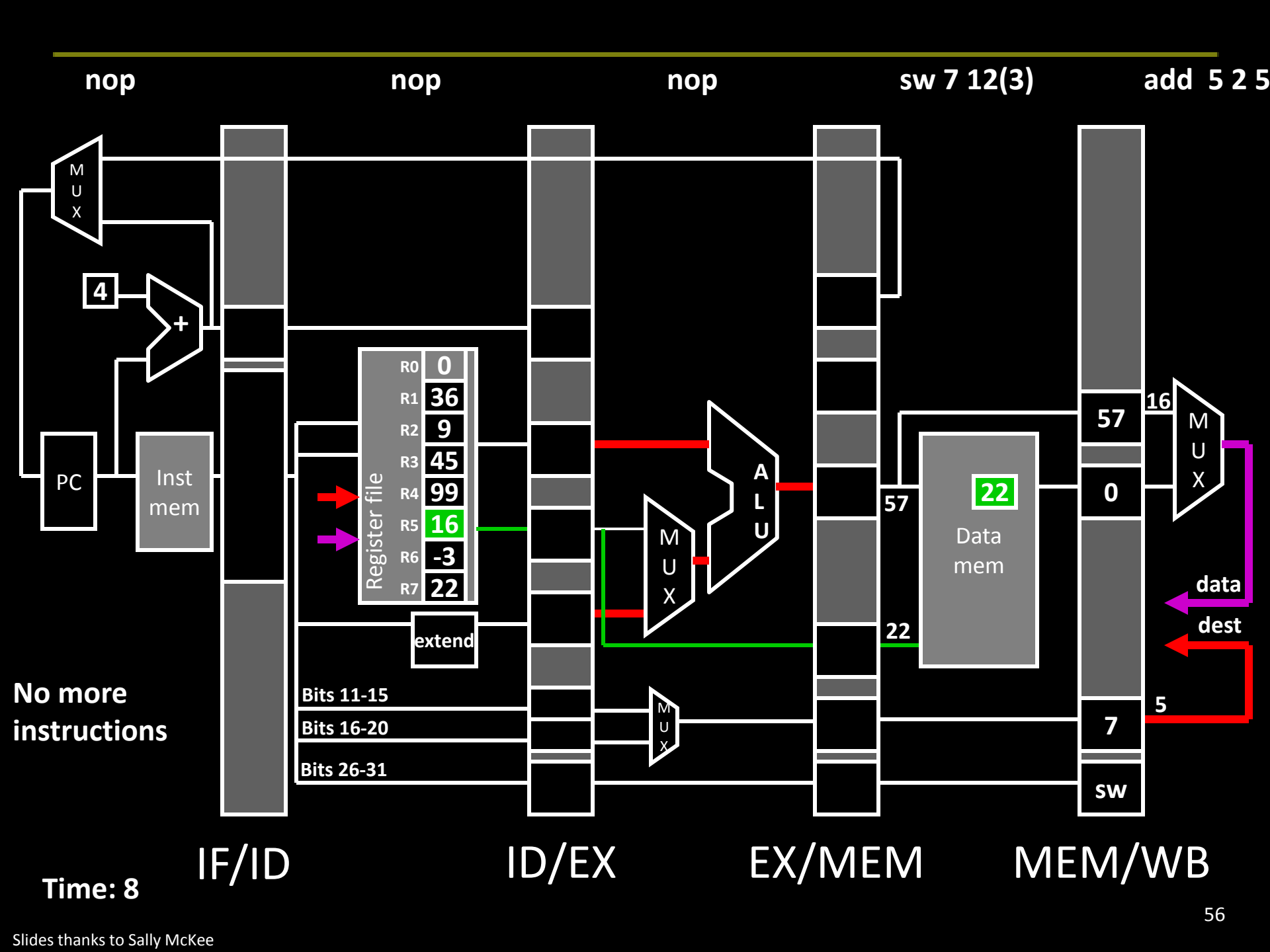

lw 4 20(2)
nand 6 4 5
add 3 1 2
M U X
4
+
PC
Inst mem
12
lw 4 20(2)
4
5
Register file
R0 0
R1 36
R2 9
R3 12
R4 18
R5 7
R6 41
R7 22
extend
Bits 11-15
Bits 16-20
Bits 26-31
8
18
7
6
6
5
nand
36
9
M U X
A L U
3
3
2
M U X
3
4
0
45
9
3
add
Data mem
0
0
0
M U X
data
dest
nop
Fetch:
lw 4 20(2)
Time: 3
IF/ID
ID/EX
EX/MEM
MEM/WB


add 5 2 5
lw 4 20(2)
nand 6 4 5
add 3 1 2
M U X
4
+
16
PC
Inst mem
add 5 2 5
2
4
Register file
R0 0
R1 36
R2 9
R3 12
R4 18
R5 7
R6 41
R7 22
extend
Bits 11-15
Bits 16-20
Bits 26-31
12
9
18
20
0
4
lw
18
7
M U X
A L U
6
5
M U X
8
0
-3
45
7
6
nand
6
3
Data mem
45
0
3
add
M U X
data
dest
Fetch:
add 5 2 5
Time: 4
IF/ID
ID/EX
EX/MEM
MEM/WB

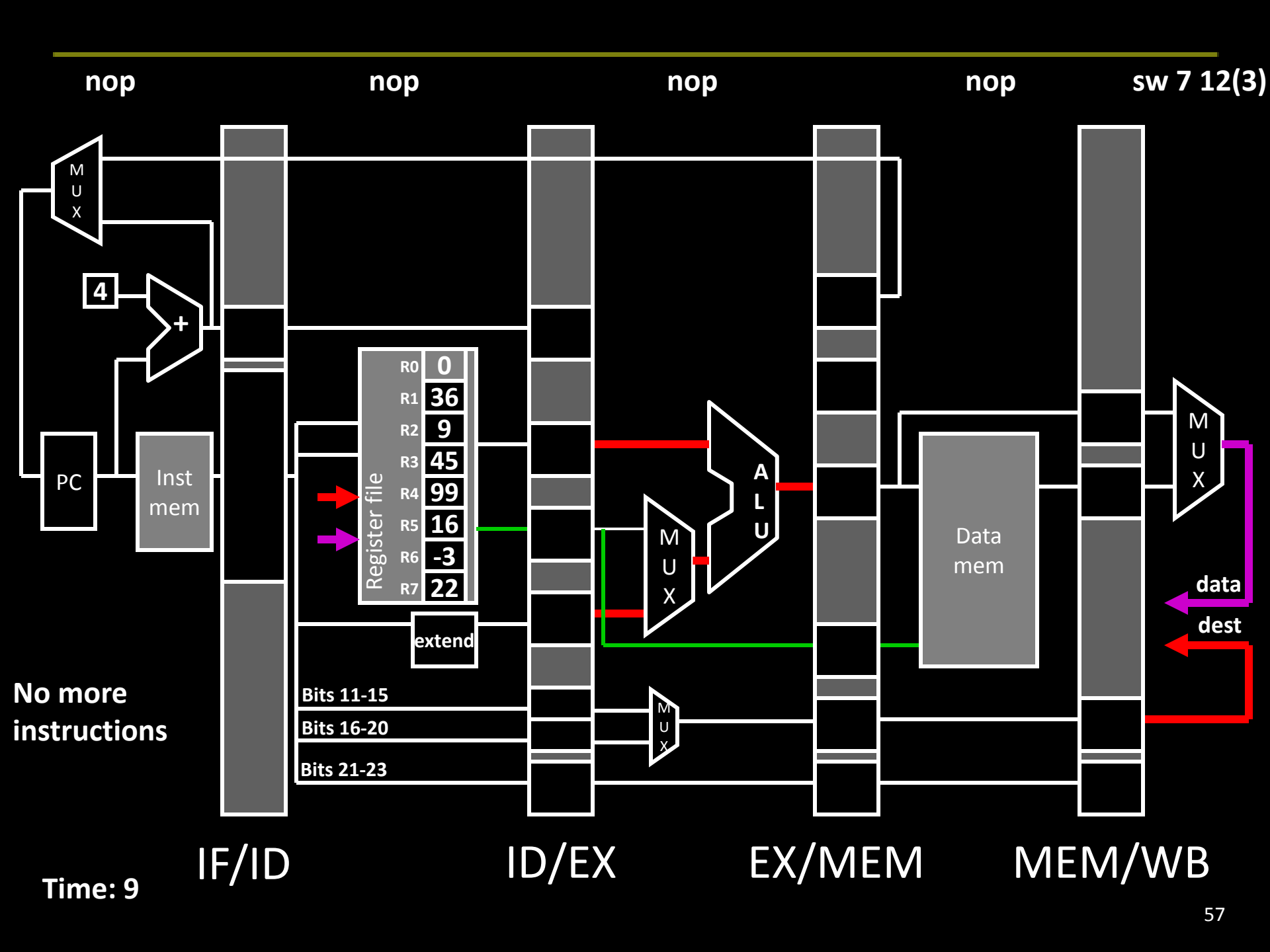

sw 7 12(3)
add 5 2 5
lw 4 20 (2)
nand 6 4 5
add 3 1 2
M U X
4
+
PC
Inst mem
20
sw 7 12(3)
2
5
Register file
R0 0
R1 36
R2 9
R3 45
R4 18
R5 7
R6 41
R7 22
extend
Bits 11-15
Bits 16-20
Bits 26-31
16
9
7
5
5
5
add
9
20
M U X
A L U
0
4
M U X
12
0
29
-3
18
4
lw
Data mem
-3
0
6
nand
45
M U X
data
dest
3
Fetch:
sw 7 12(3)
Time: 5
IF/ID
ID/EX
EX/MEM
MEM/WB


sw 7 12(3)
add 5 2 5
lw 4 20(2)
nand 6 4 5
M U X
4
+
PC
Inst mem
3
7
Register file
R0 0
R1 36
R2 9
R3 45
R4 18
R5 7
R6 -3
R7 22
extend
20
45
22
12
0
7
sw
Bits 11-15
Bits 16-20
Bits 26-31
9
7
M U X
A L U
5
5
M U X
16
0
16
29
7
5
add
5
Data mem
29
99
4
lw
-3
M U X
data
dest
6
No more instructions
Time: 6
IF/ID
ID/EX
EX/MEM
MEM/WB


nop
nop
sw 7 12(3)
add 5 2 5
lw 4 20(2)
M U X
4
+
PC
Inst mem
R0 0
R1 36
R2 9
R3 45
R4 99
R5 7
R6 -3
R7 22
Register file
extend
Bits 11-15
Bits 16-20
Bits 26-31
45
12
M U X
A L U
0
7
M U X
20
0
57
16
22
7
sw
Data mem
16
0
5
add
99
M U X
data
dest
4
5
7
No more instructions
Time: 7
IF/ID
ID/EX
EX/MEM
MEM/WB



Slides thanks to Sally McKee


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
Bits 11-15
Bits 16-20
Bits 21-23
M
U
X
A
L
U
M
U
X
Data
mem
M
U
X
data
dest
No more
instructions
IF/ID
ID/EX
EX/MEM
MEM/WB
Time: 9

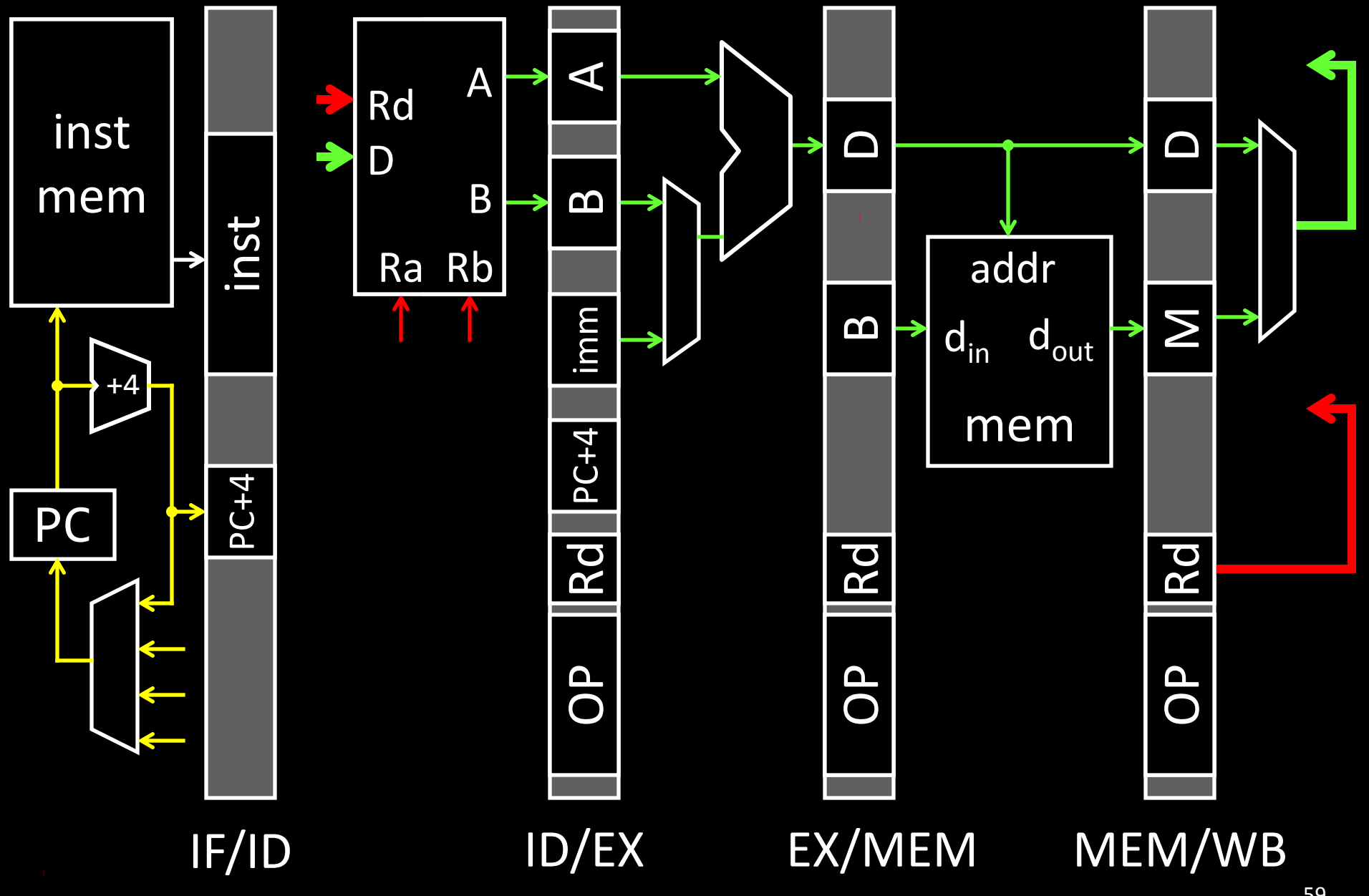
Pipelining Recap

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
+4
PC
inst
PC+4
Rd
D
A
B
Ra
Rb
A
B
imm
PC+4
Rd
OP
D
B
Rd
OP
addr
d_{in}
d_{out}
mem
D
M
Rd
OP
IF/ID
ID/EX
EX/MEM
MEM/WB