

MIPS Pipeline

Hakim Weatherspoon

CS 3410, Spring 2012

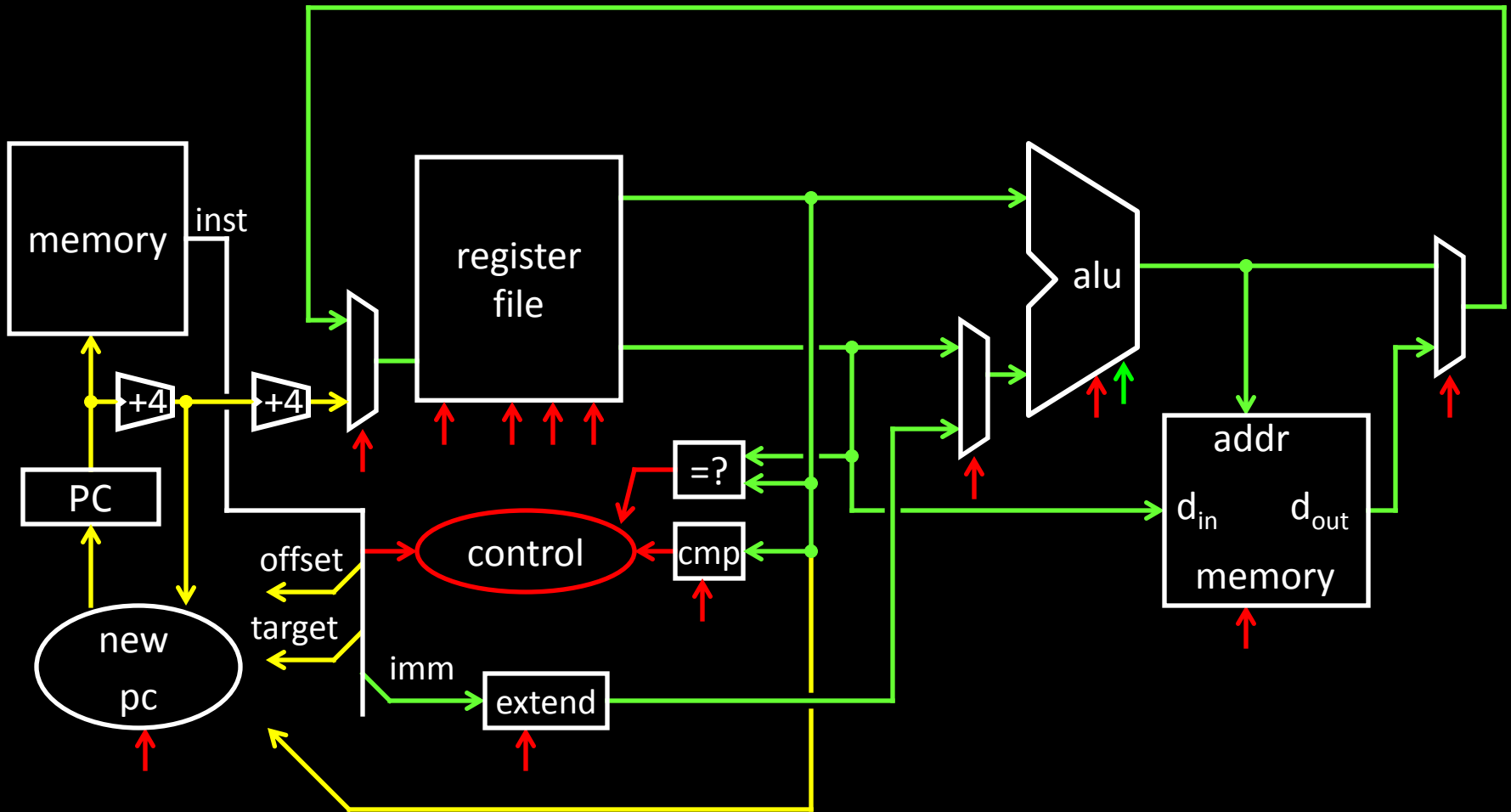
Computer Science

Cornell University

See P&H Chapter 4.6

A Processor

Review: Single cycle processor



Review: Single Cycle Processor

Advantages

- Single Cycle per instruction make logic and clock simple

Disadvantages

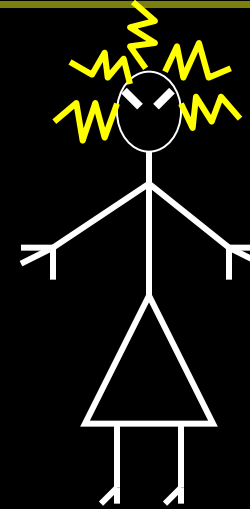
- Since instructions take different time to finish, memory and functional unit are not efficiently utilized.
- Cycle time is the longest delay.
 - Load instruction
- Best possible CPI is 1

Single Cycle vs Pipelined Processor

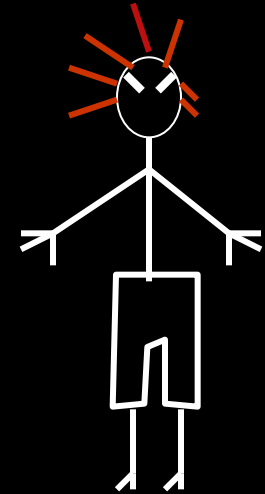
See: P&H Chapter 4.5

The Kids

Alice



Bob



They don't always get along...

The Bicycle



The Materials



Saw



Drill



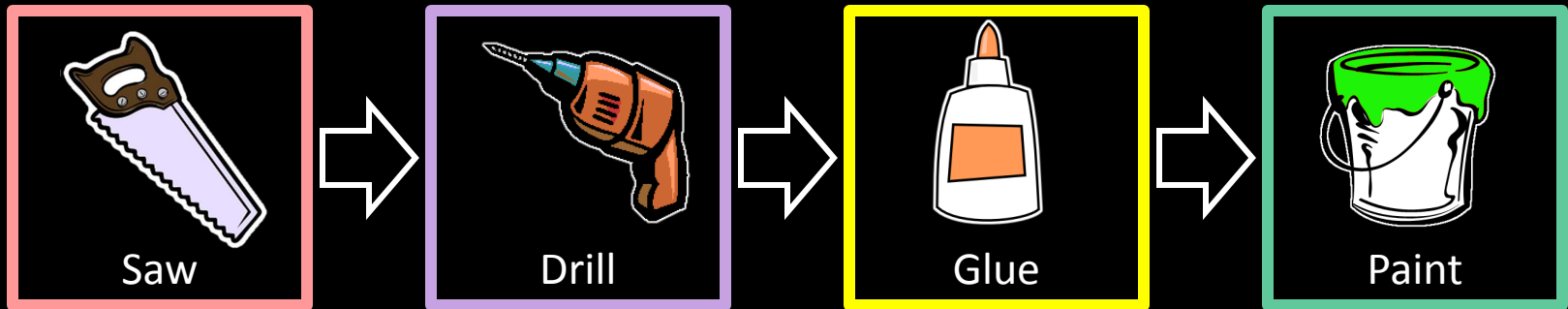
Glue



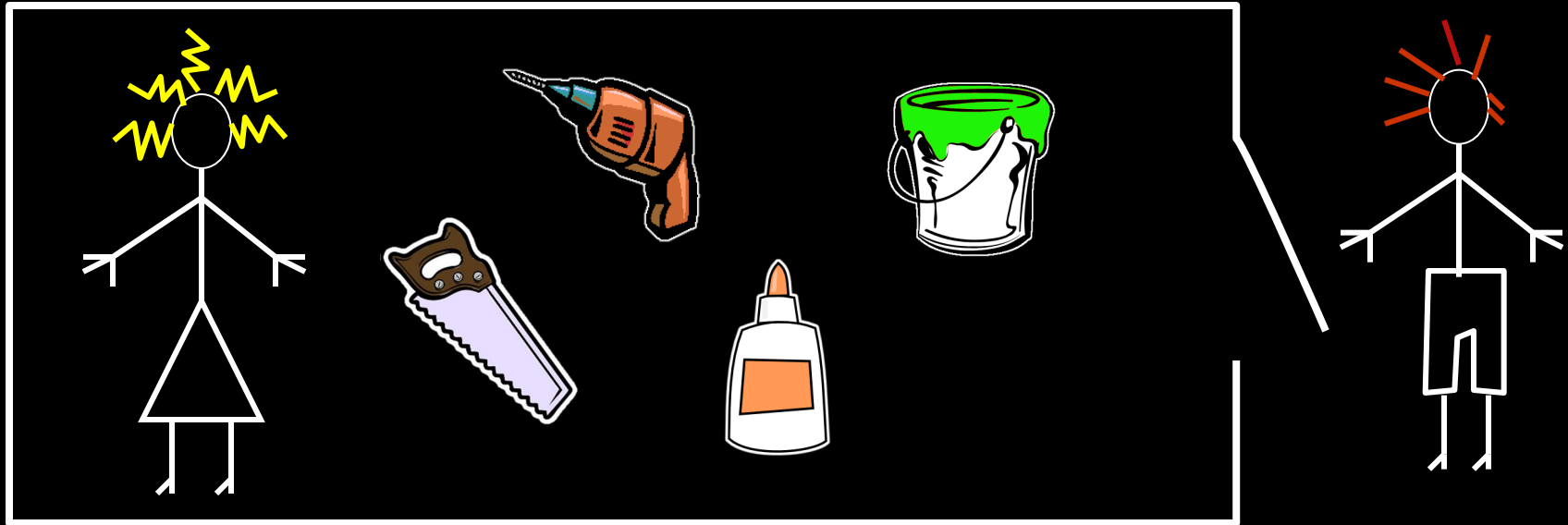
Paint

The Instructions

N pieces, each built following same sequence:



Design 1: Sequential Schedule



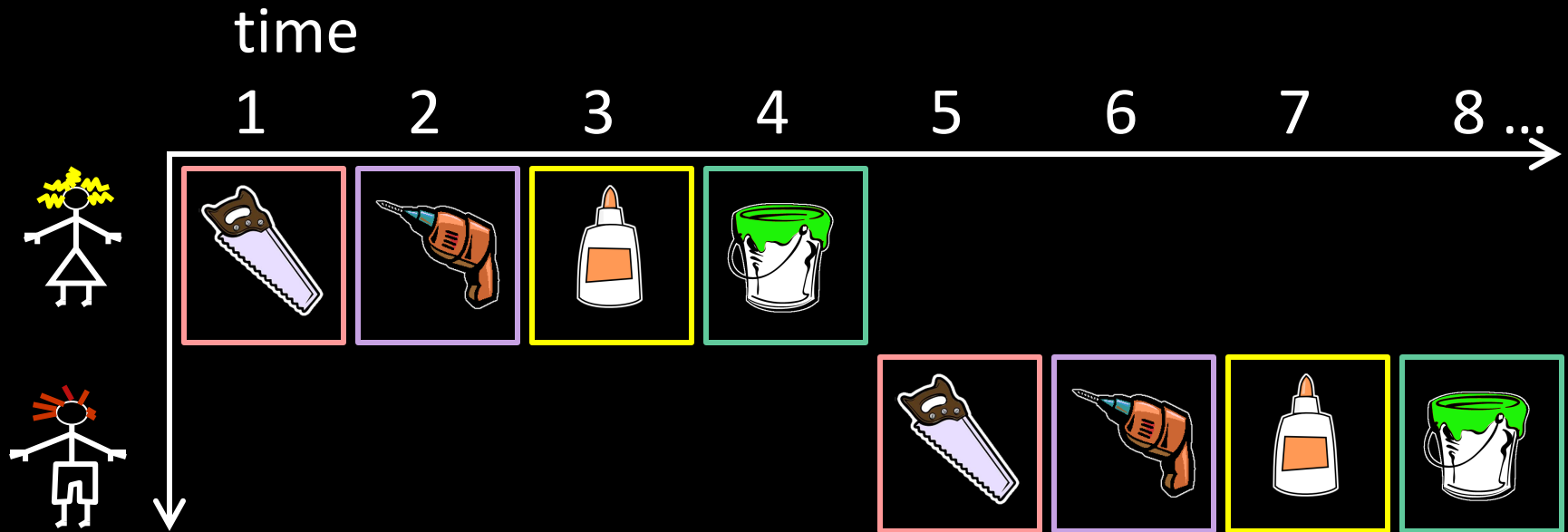
Alice owns the room

Bob can enter when Alice is finished

Repeat for remaining tasks

No possibility for conflicts

Sequential Performance



Latency:

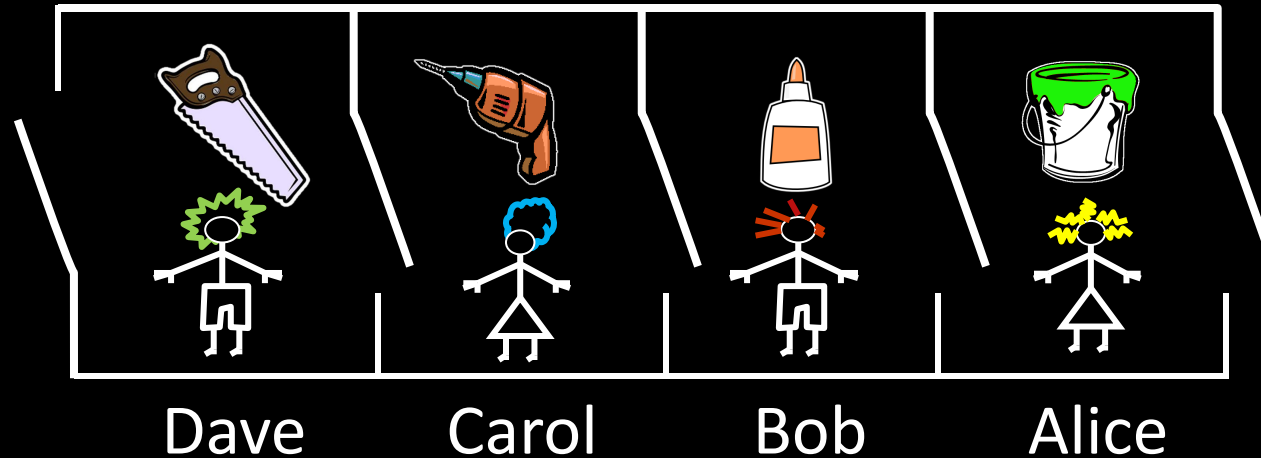
Throughput:

Concurrency:

Can we do better?

Design 2: Pipelined Design

Partition room into *stages* of a *pipeline*



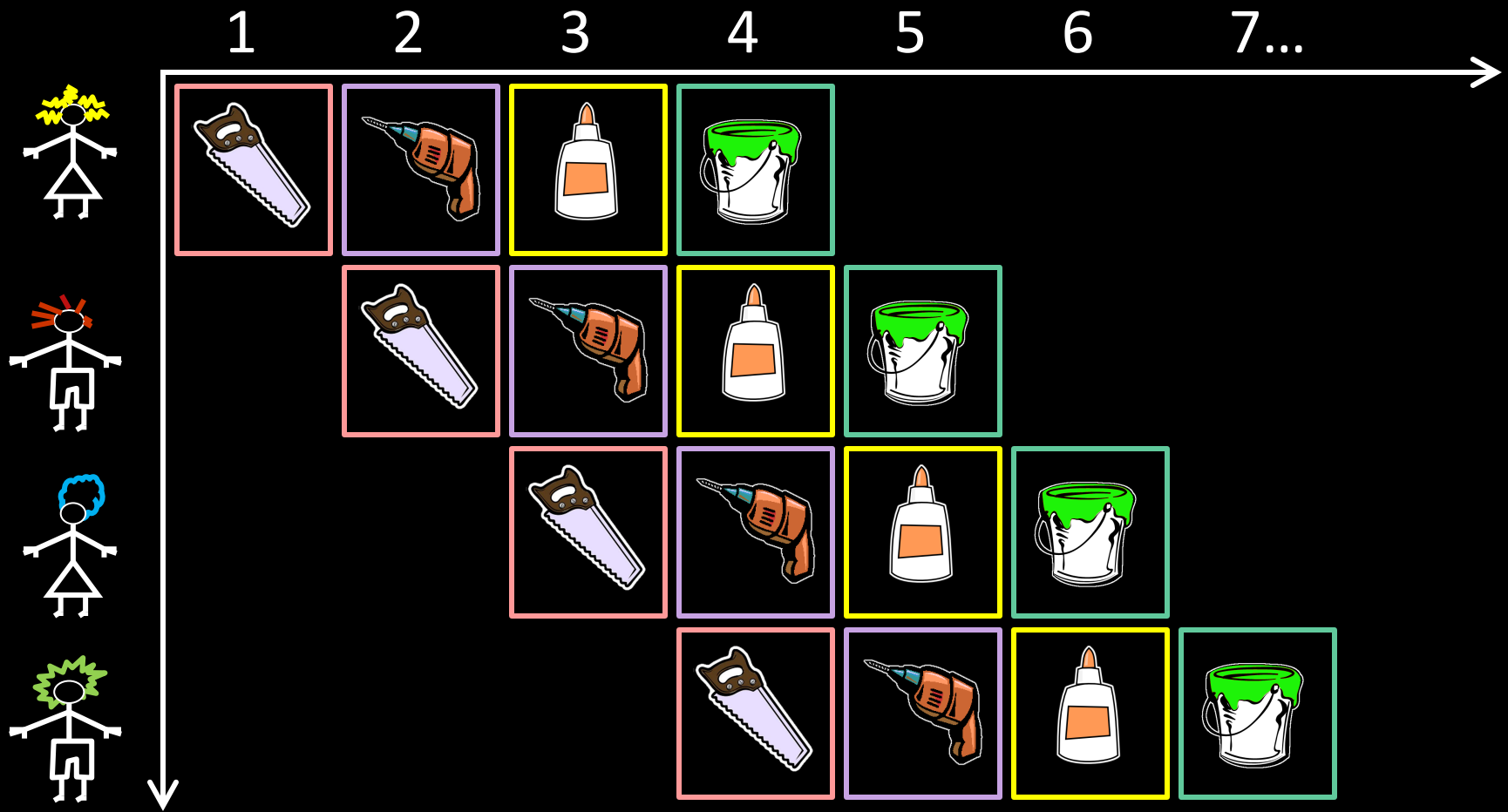
One person owns a stage at a time

4 stages

4 people working simultaneously

Everyone moves right in lockstep

time **Pipelined Performance**



Latency:

Throughput:

Concurrency:

Lessons

Principle:

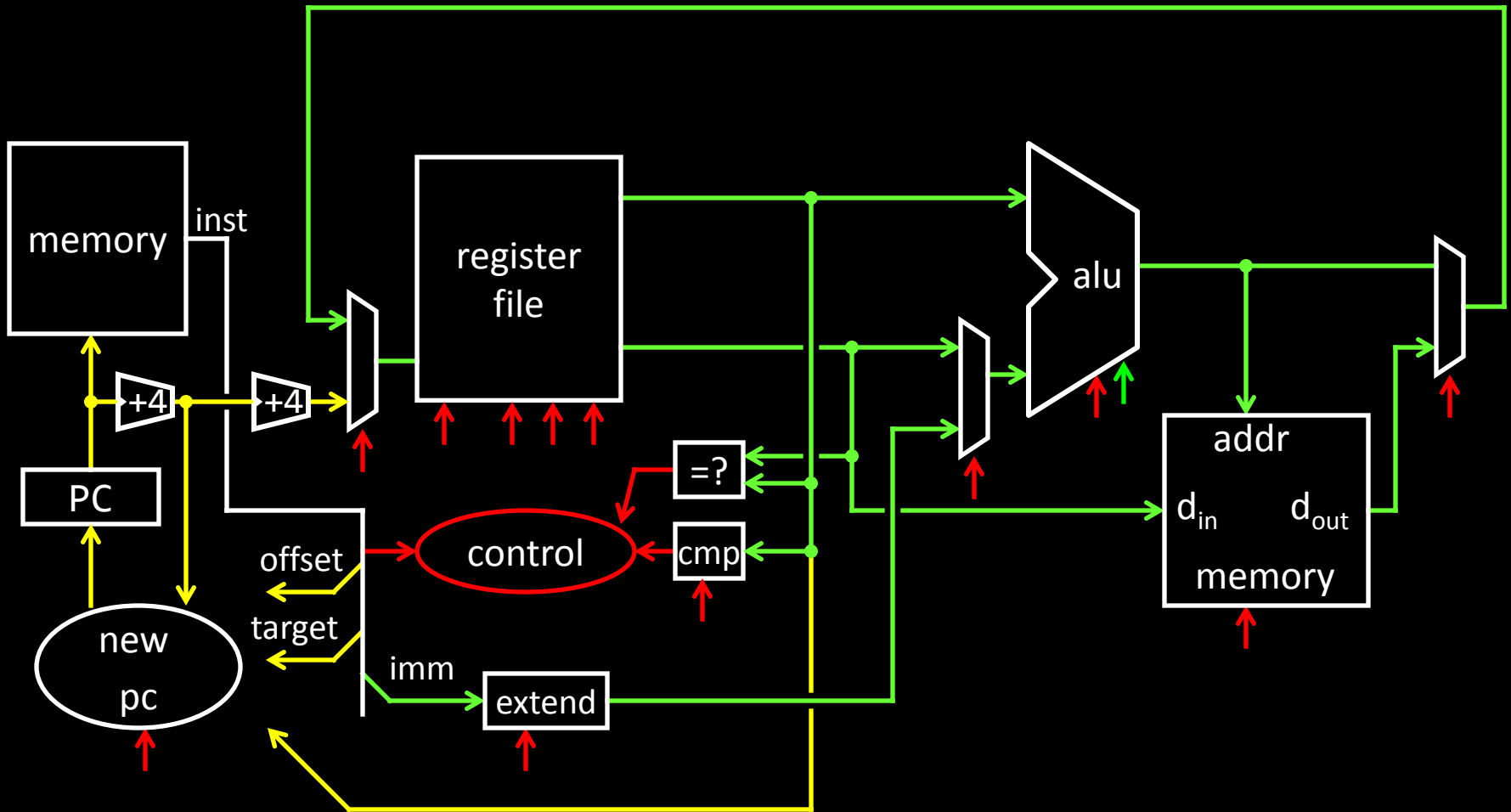
Throughput increased by parallel execution

Pipelining:

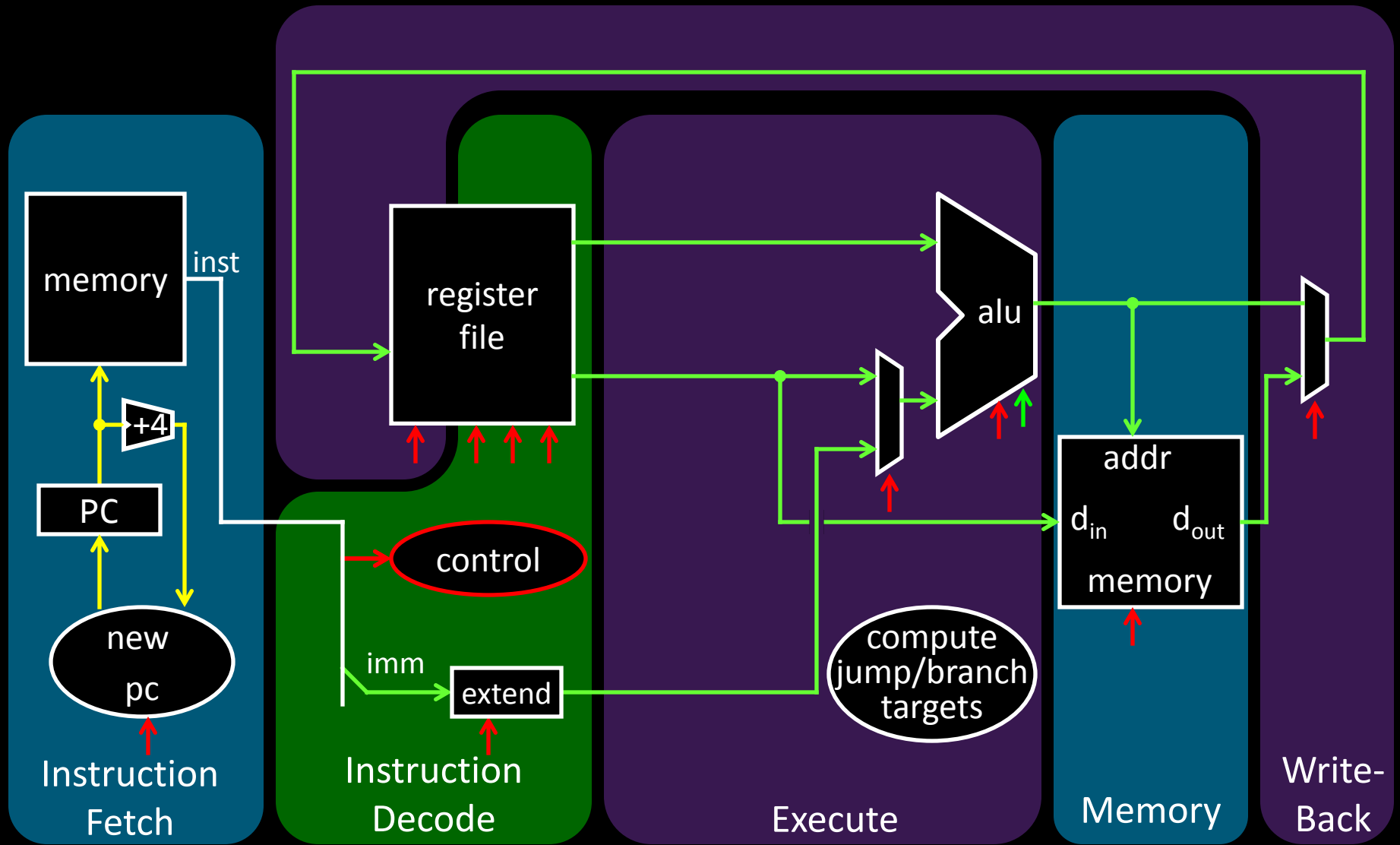
- Identify *pipeline stages*
- **Isolate** stages from each other
- Resolve pipeline *hazards* (Thursday)

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

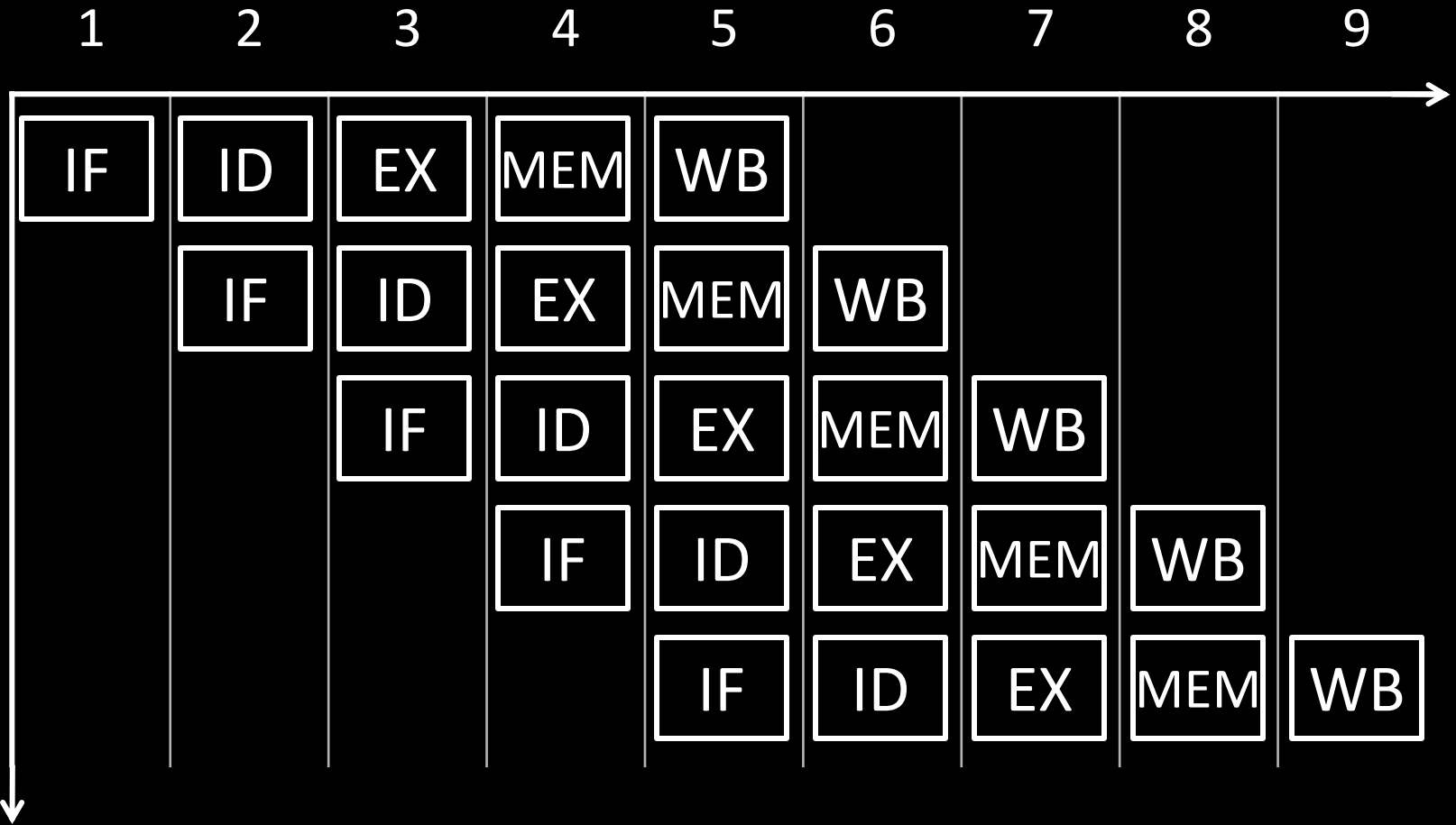
4. Memory (MEM)

- access memory if needed

5. Writeback (WB)

- update register file

Clock cycle Time Graphs



Latency:

Throughput:

Concurrency:

Principles of Pipelined Implementation

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

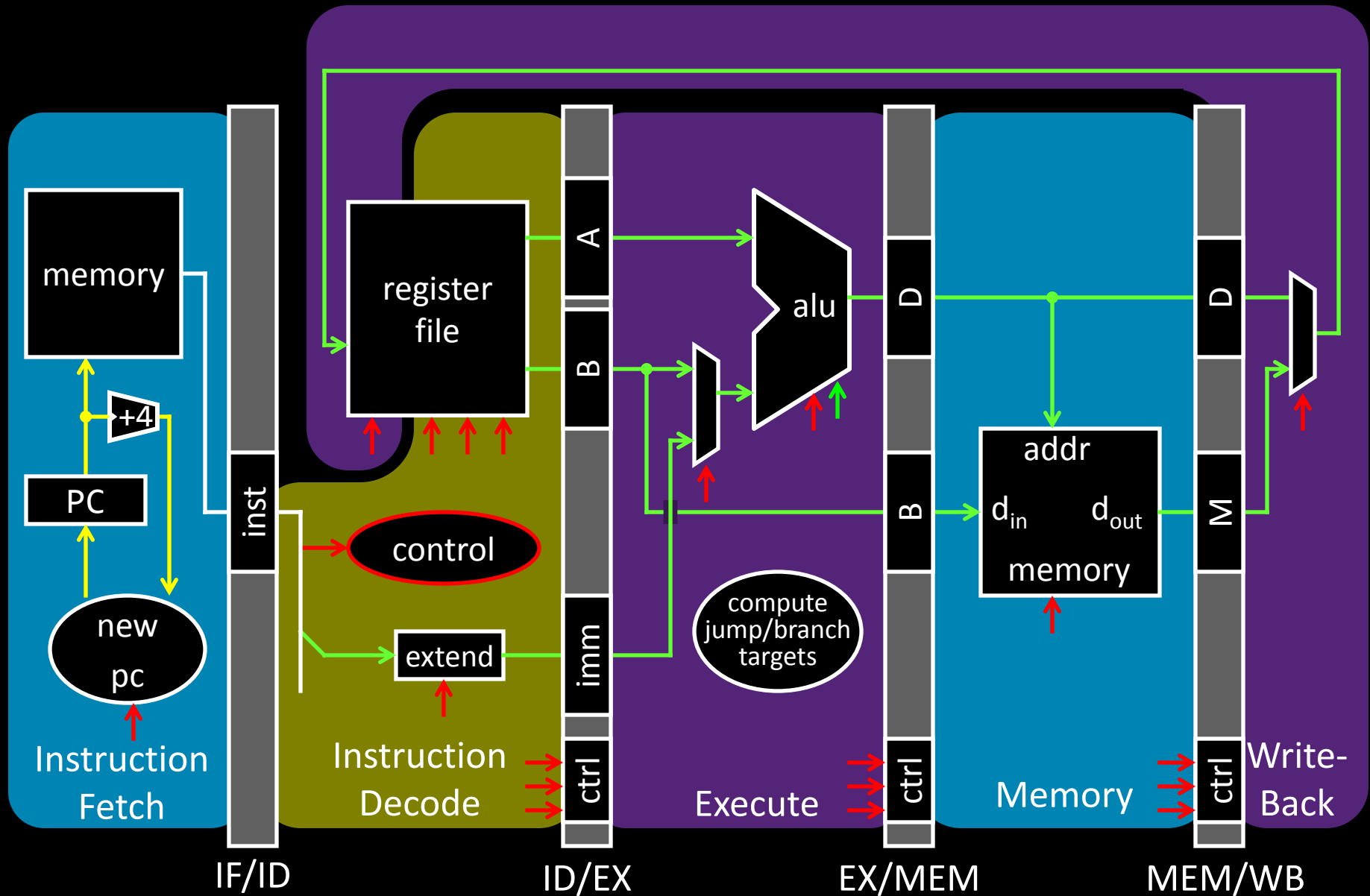
Design a separate **stage** for the execution
performed during each clock cycle

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

Pipelined Processor

See: P&H Chapter 4.6

Pipelined Processor



Stage 1: Instruction Fetch

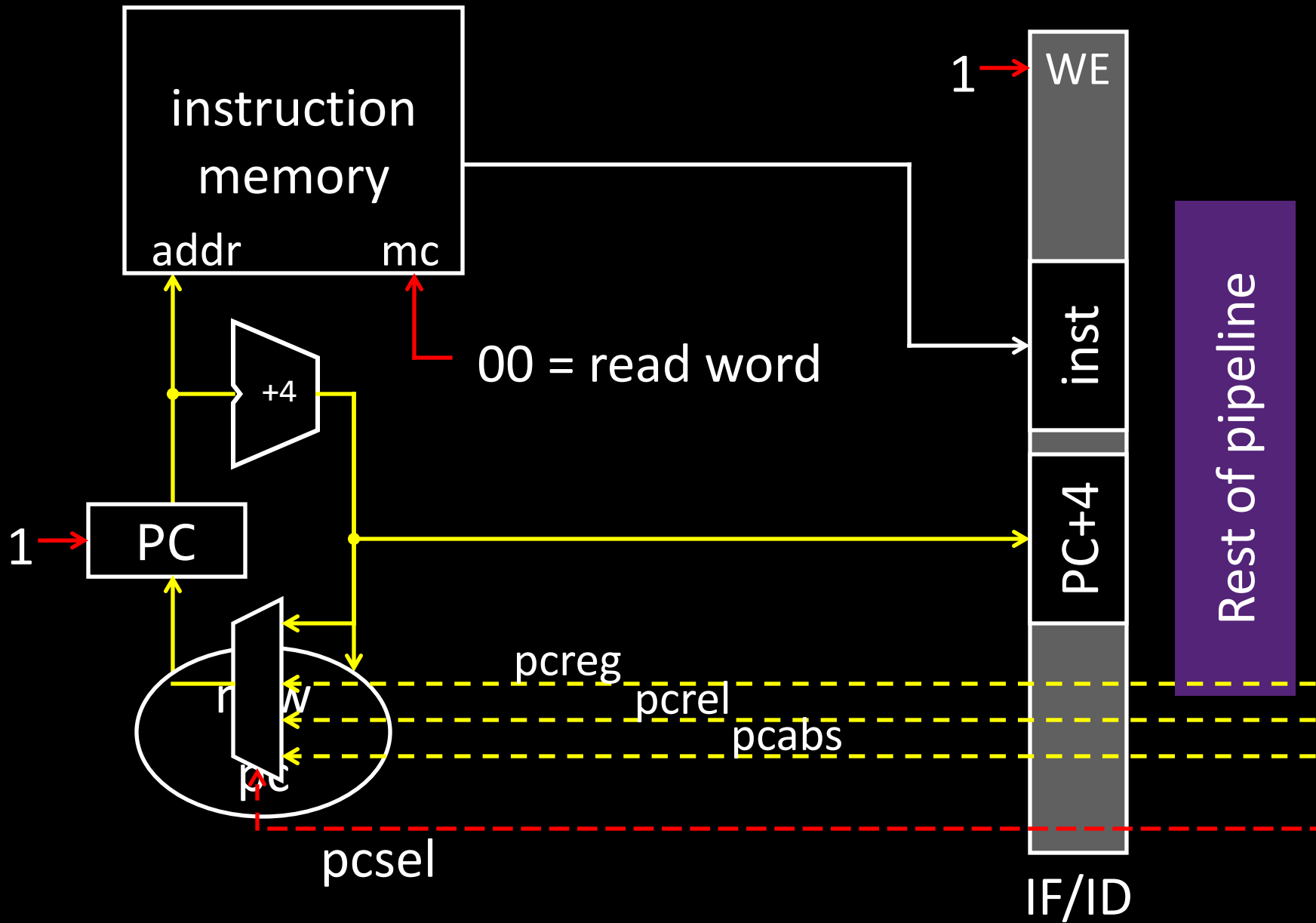
Fetch a new instruction **every** cycle

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

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

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

IF



Stage 2: Instruction Decode

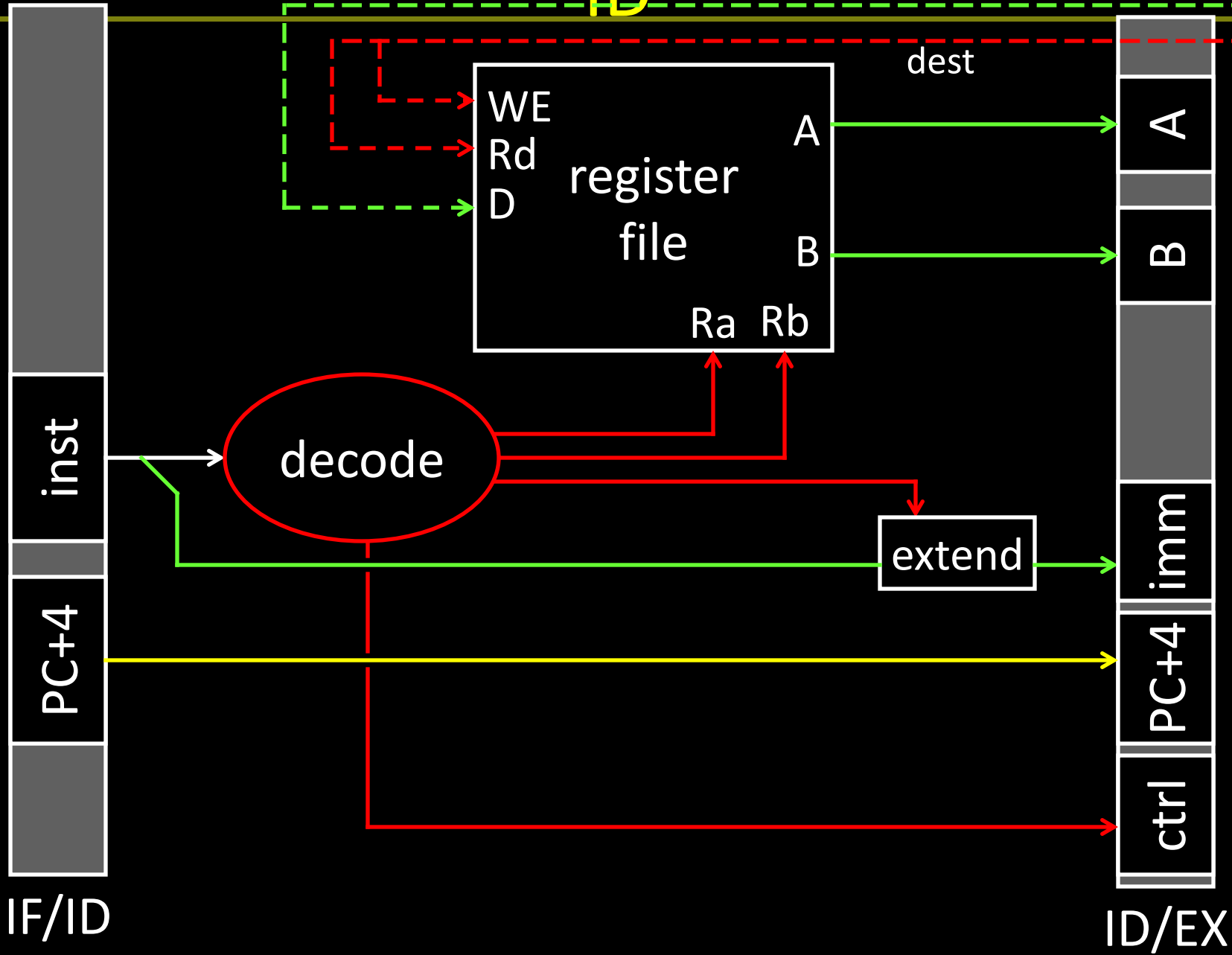
On **every** cycle:

- 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



Stage 3: Execute

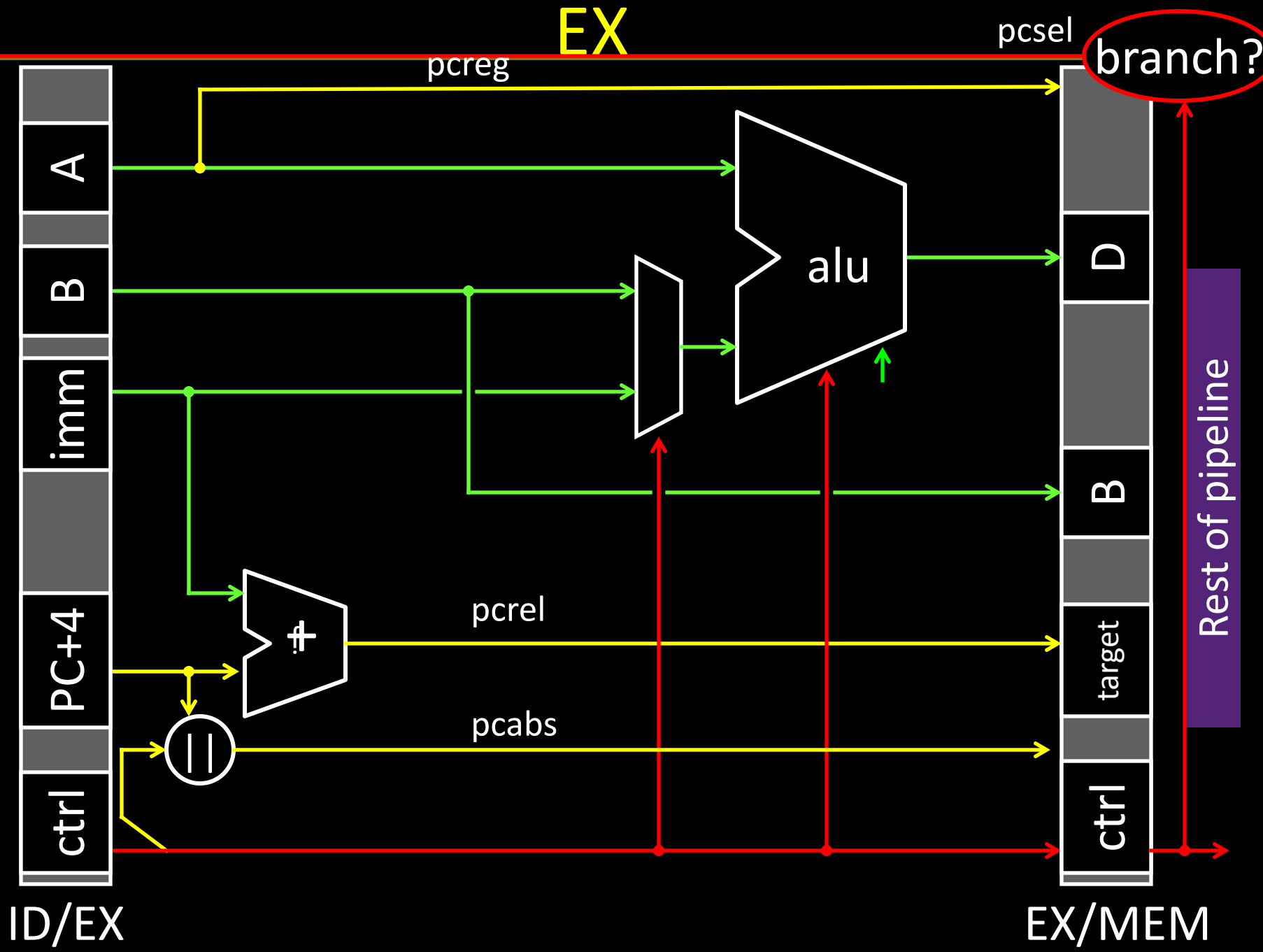
On **every** cycle:

- 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

Stage 2: Instruction Decode



MEM

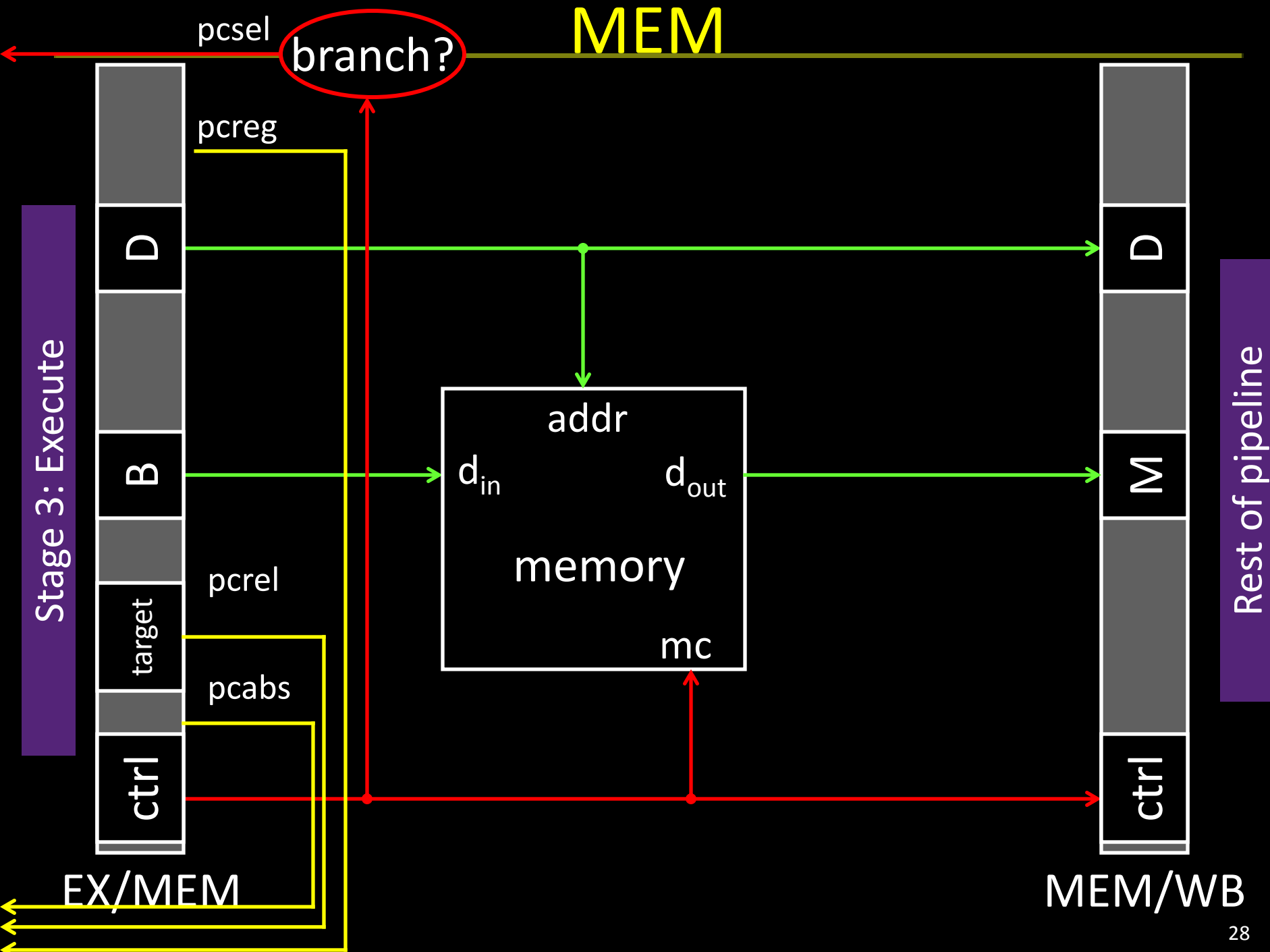
Stage 4: Memory

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

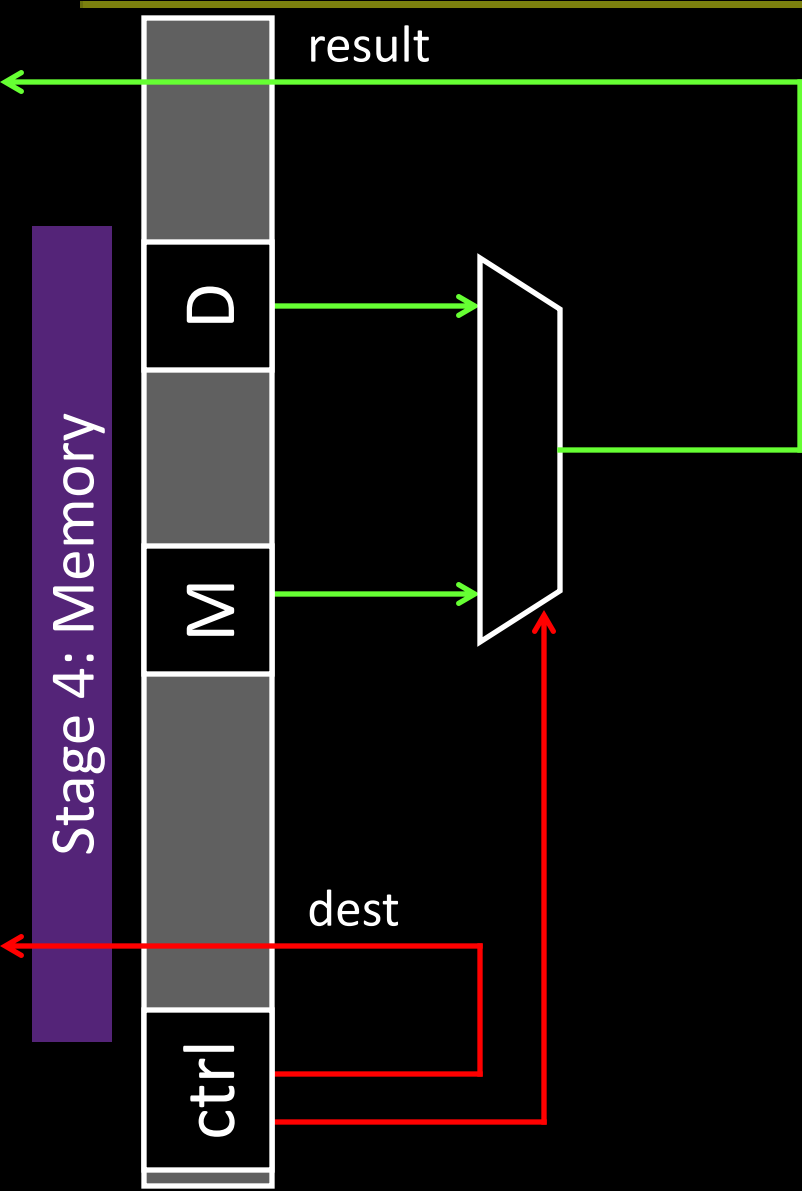


Stage 5: Write-back

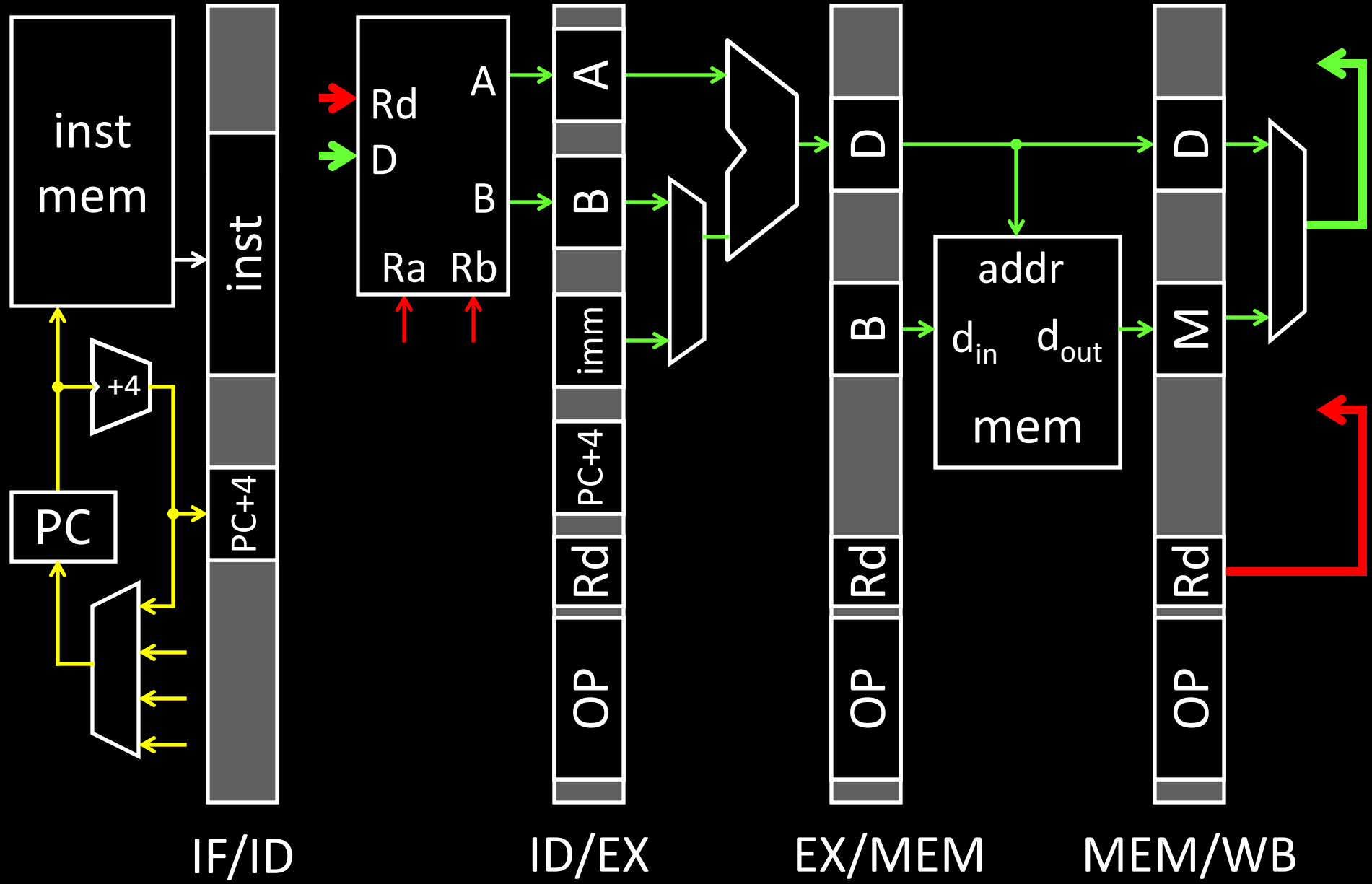
On **every** cycle:

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

WB



MEM/WB



Administrivia

HW2 due today

- **Fill out Survey online.** Receive credit/points on homework for survey.
- 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 3 1 2 ; reg 3 = reg 1 + reg 2

nand 6 4 5 ; reg 6 = ~(reg 4 & reg 5)

lw 4 20 (2) ; reg 4 = Mem[reg2+20]

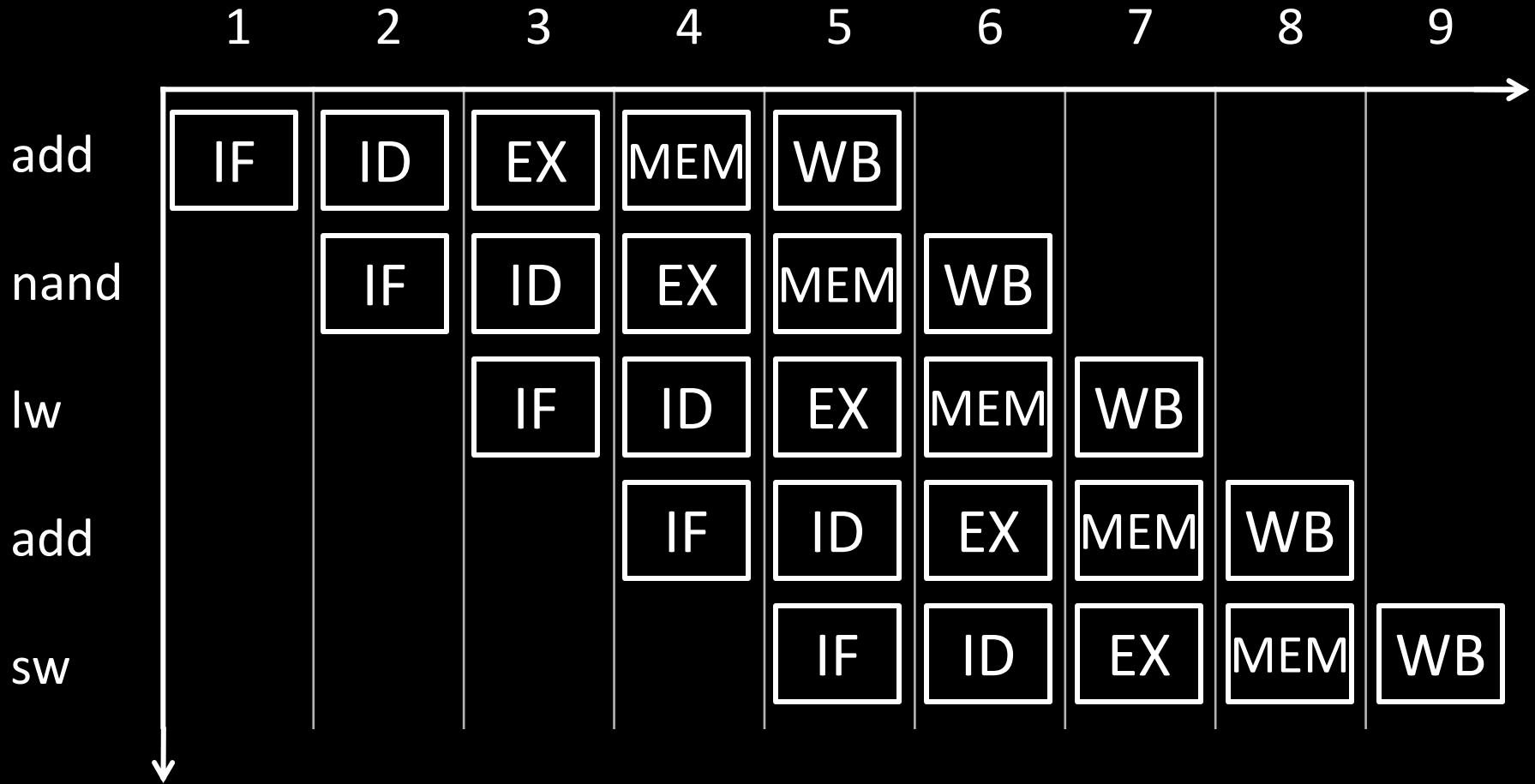
add 5 2 5 ; reg 5 = reg 2 + reg 5

sw 7 12(3) ; 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);
```

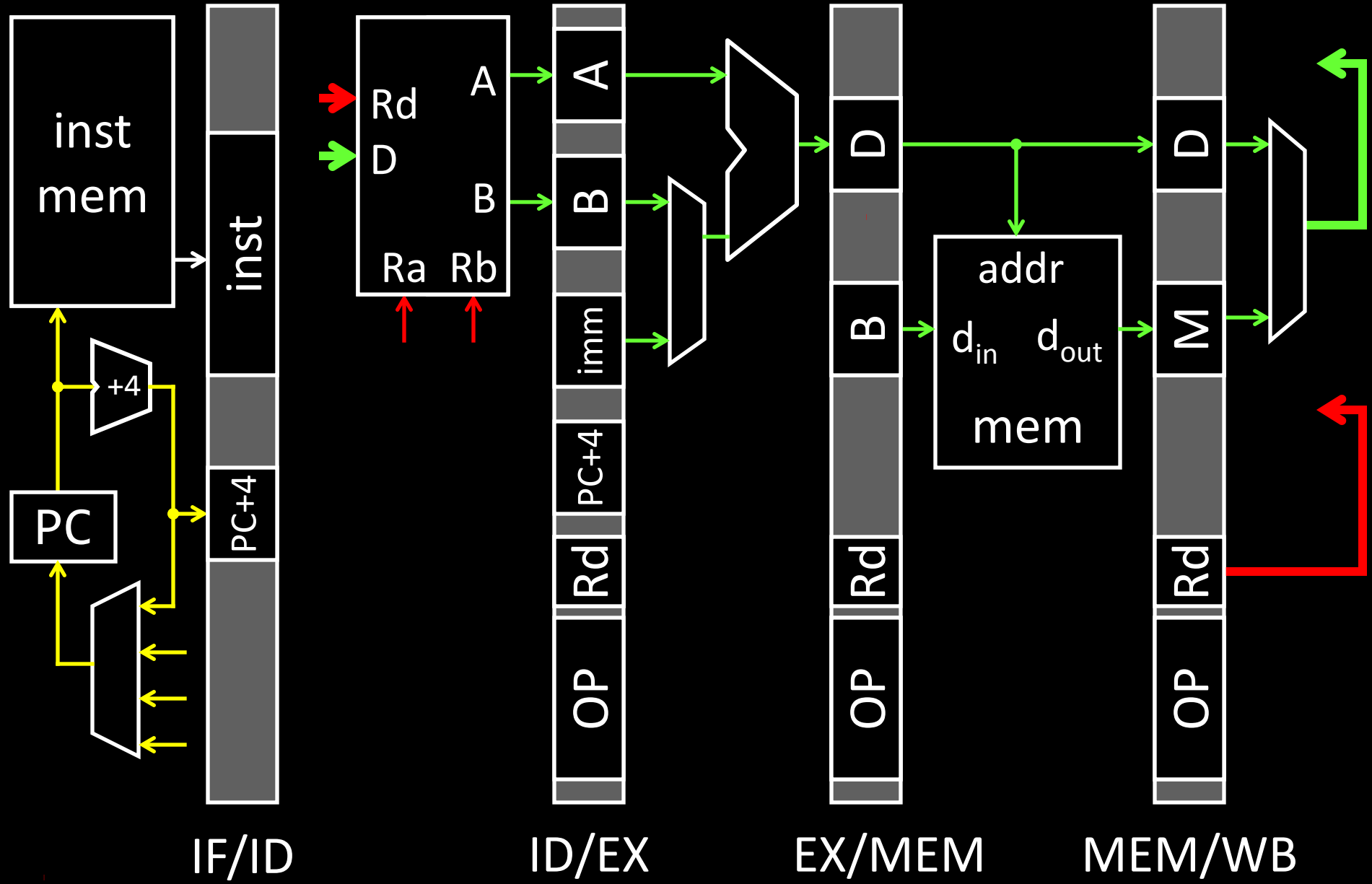
Clock cycle Time Graphs

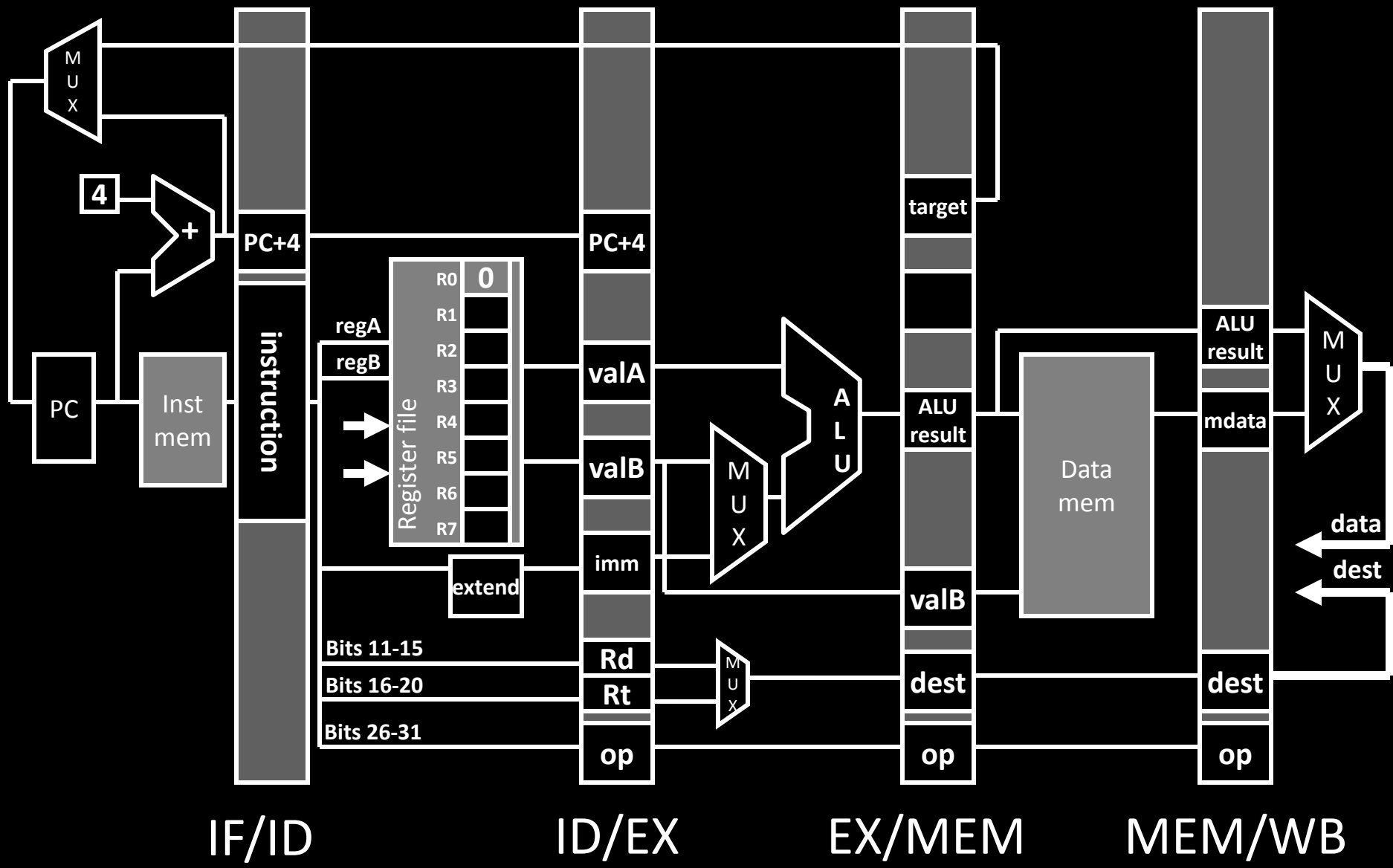


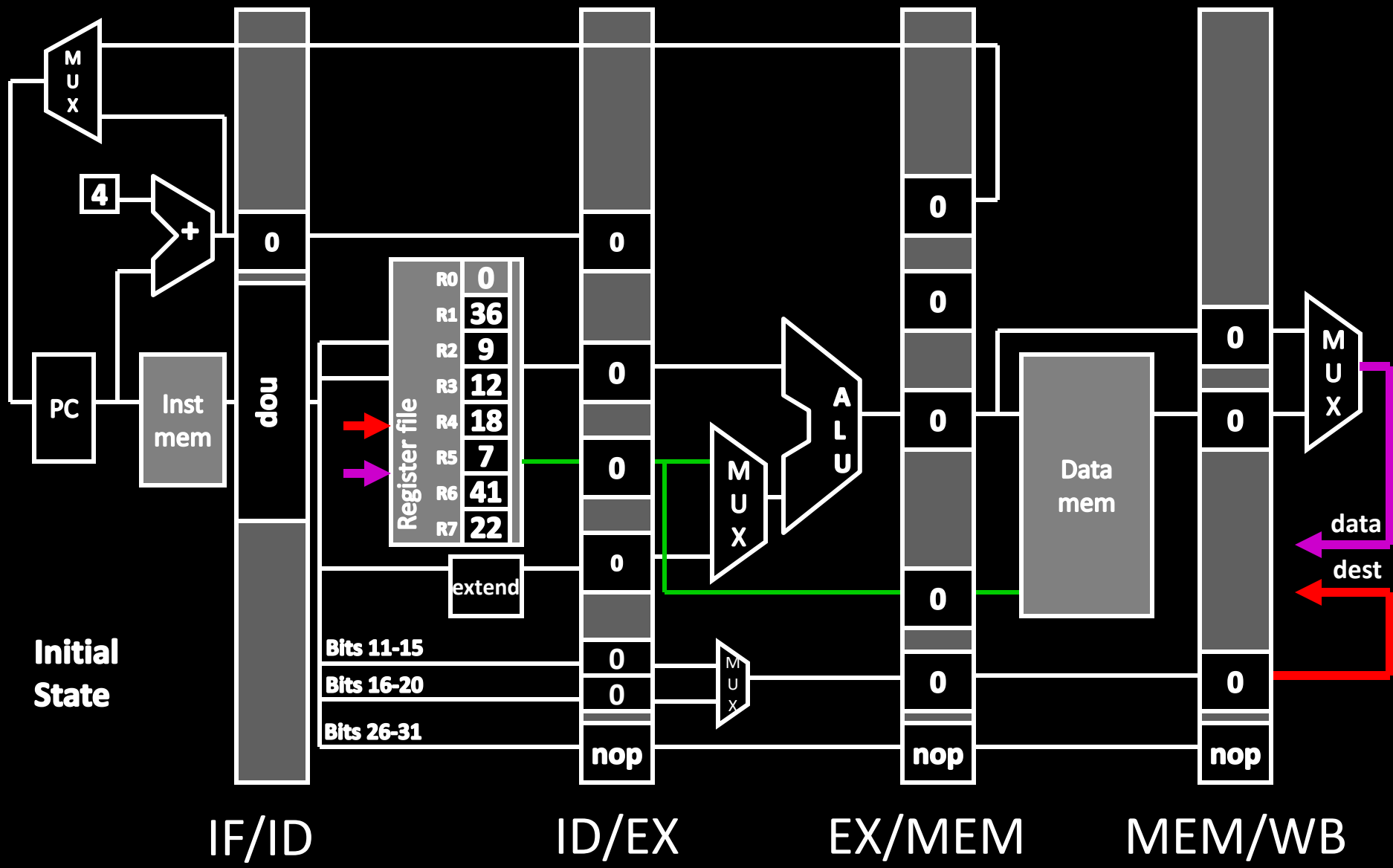
Latency:

Throughput:

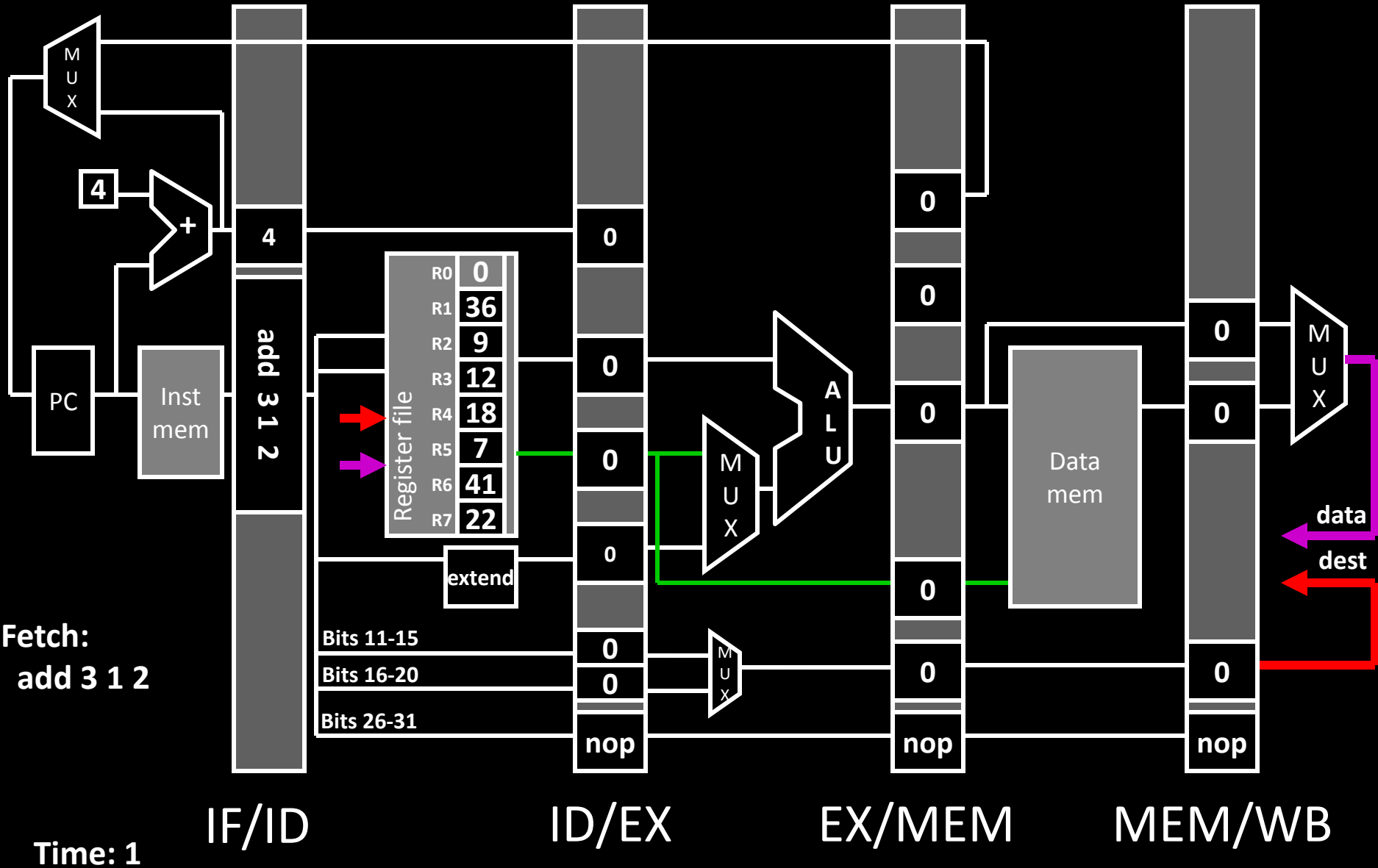
Concurrency:





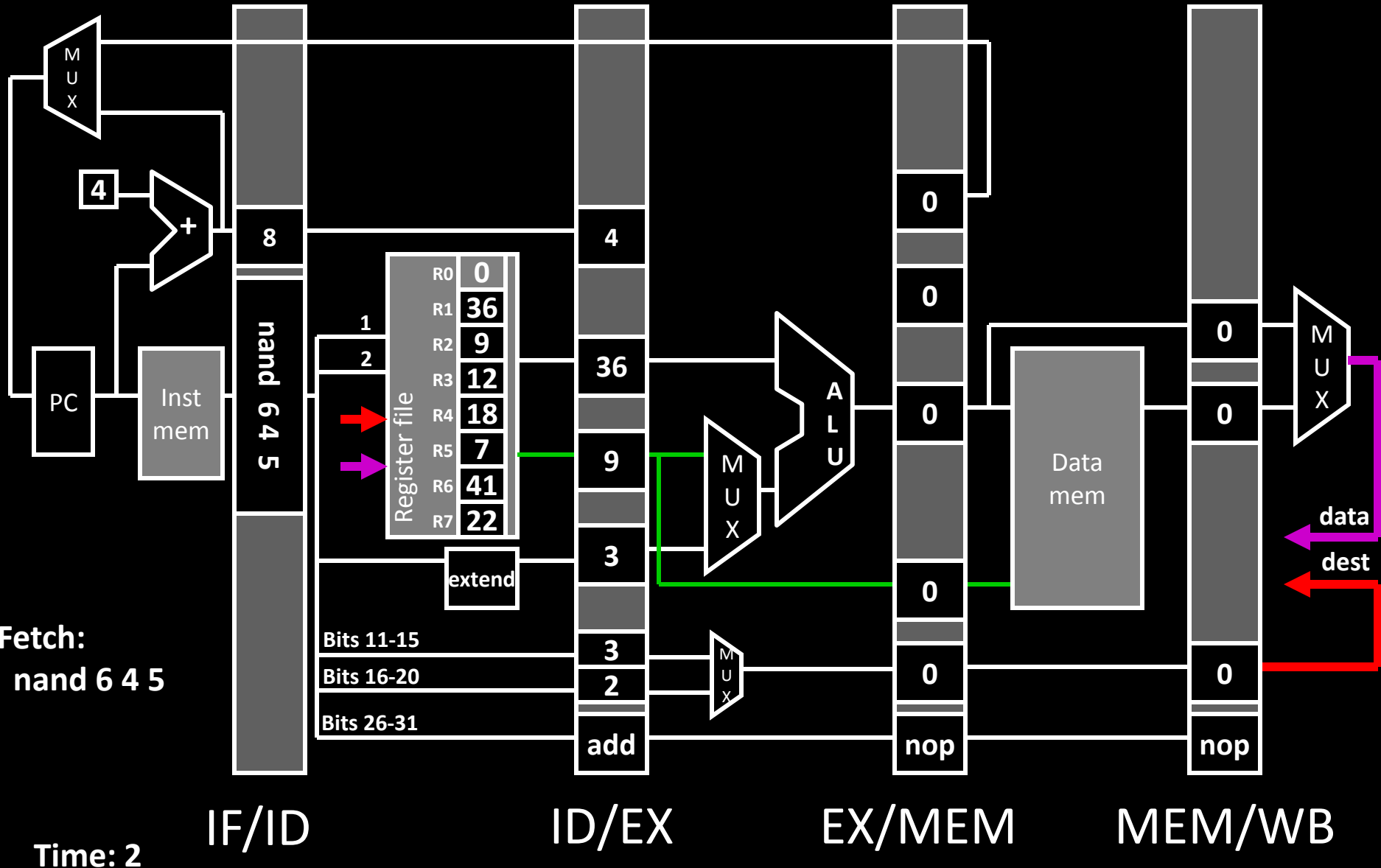


add 3 1 2



nand 6 4 5

add 3 1 2



Fetch:
nand 6 4 5

Time: 2

IF/ID

ID/EX

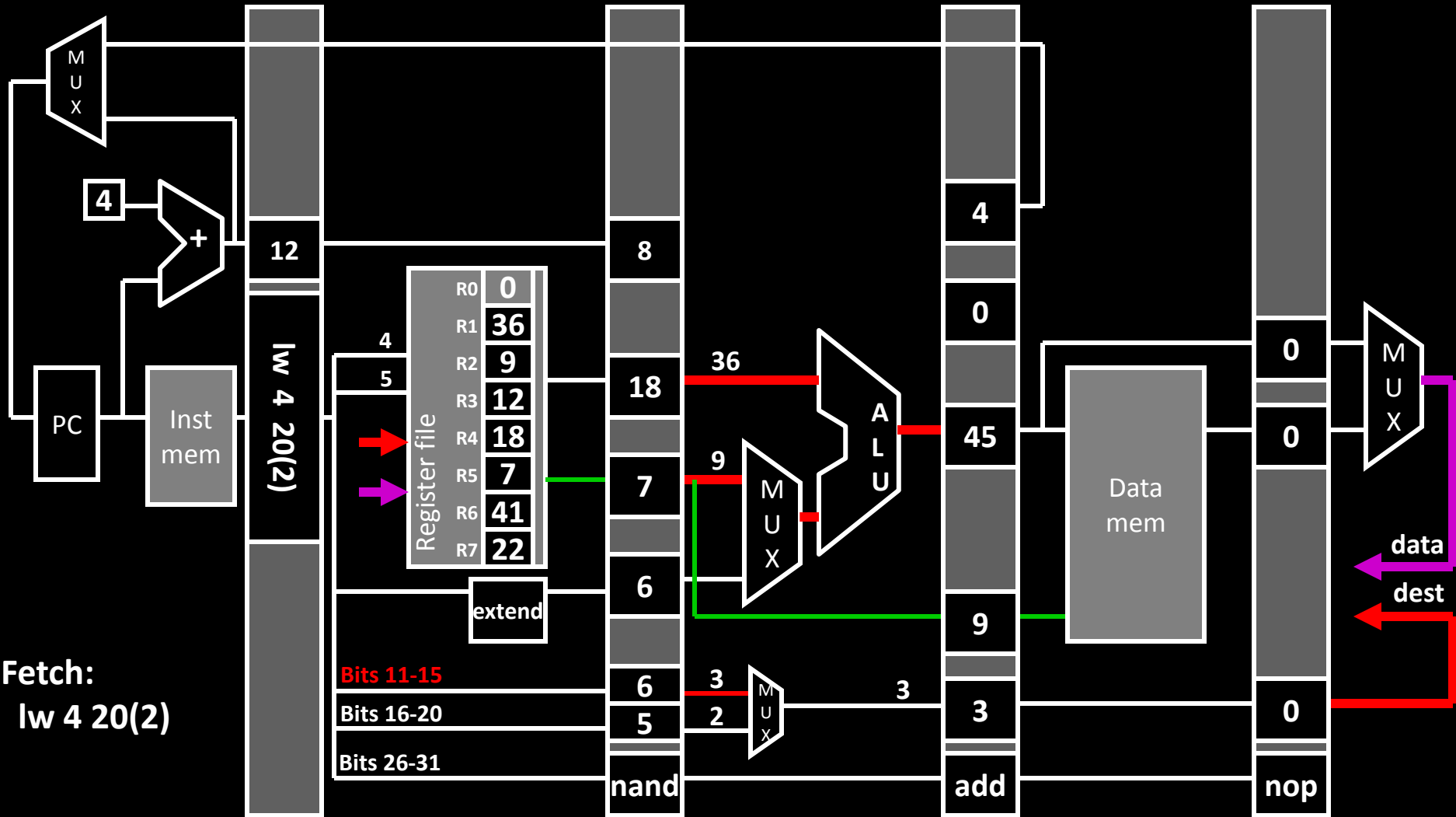
EX/MEM

MEM/WB

lw 4 20(2)

nand 6 4 5

add 3 1 2



Fetch:
lw 4 20(2)

Time: 3

IF/ID

ID/EX

EX/MEM

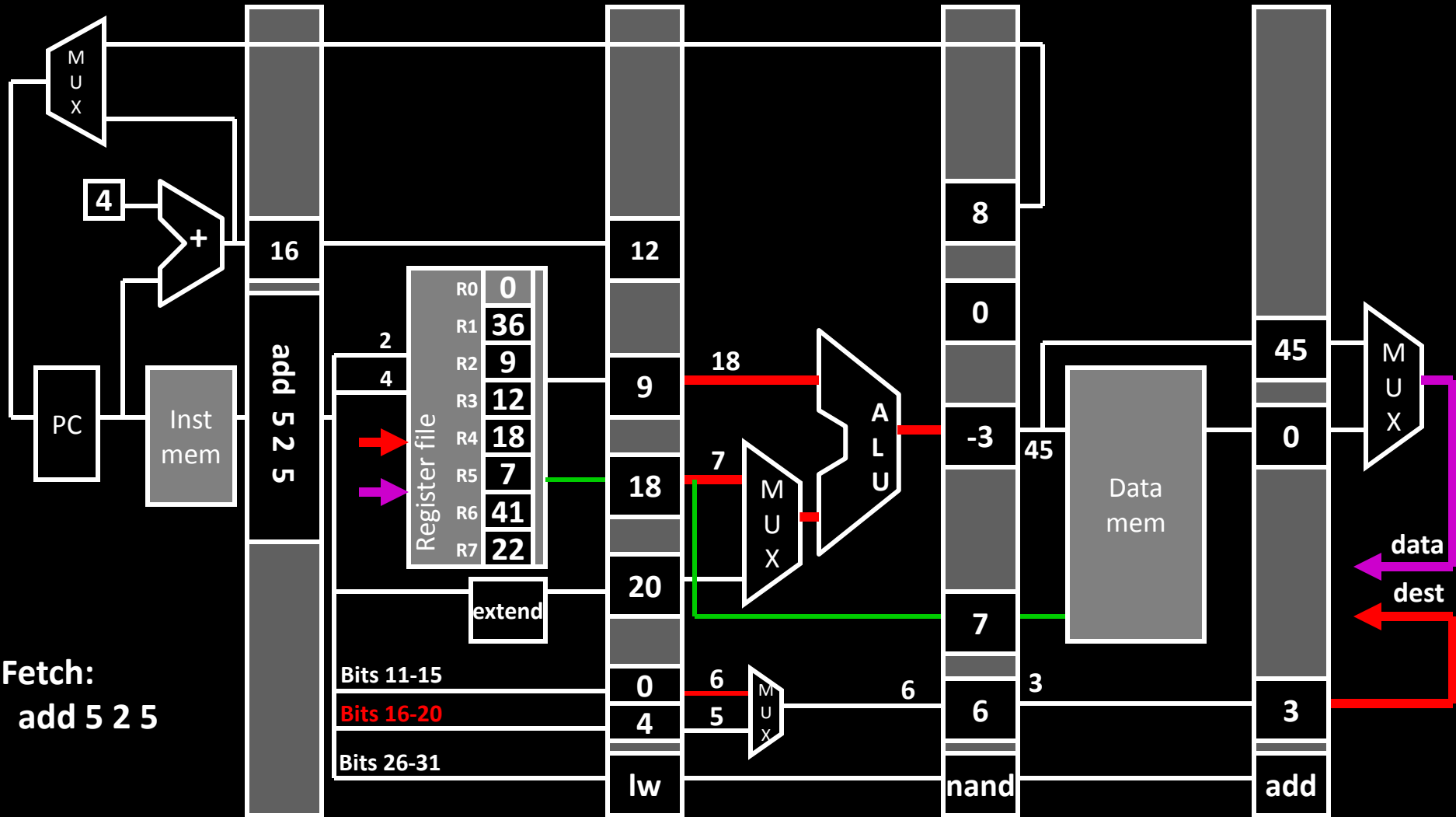
MEM/WB

add 5 2 5

lw 4 20(2)

nand 6 4 5

add 3 1 2



Fetch:
add 5 2 5

Time: 4

IF/ID

ID/EX

EX/MEM

MEM/WB

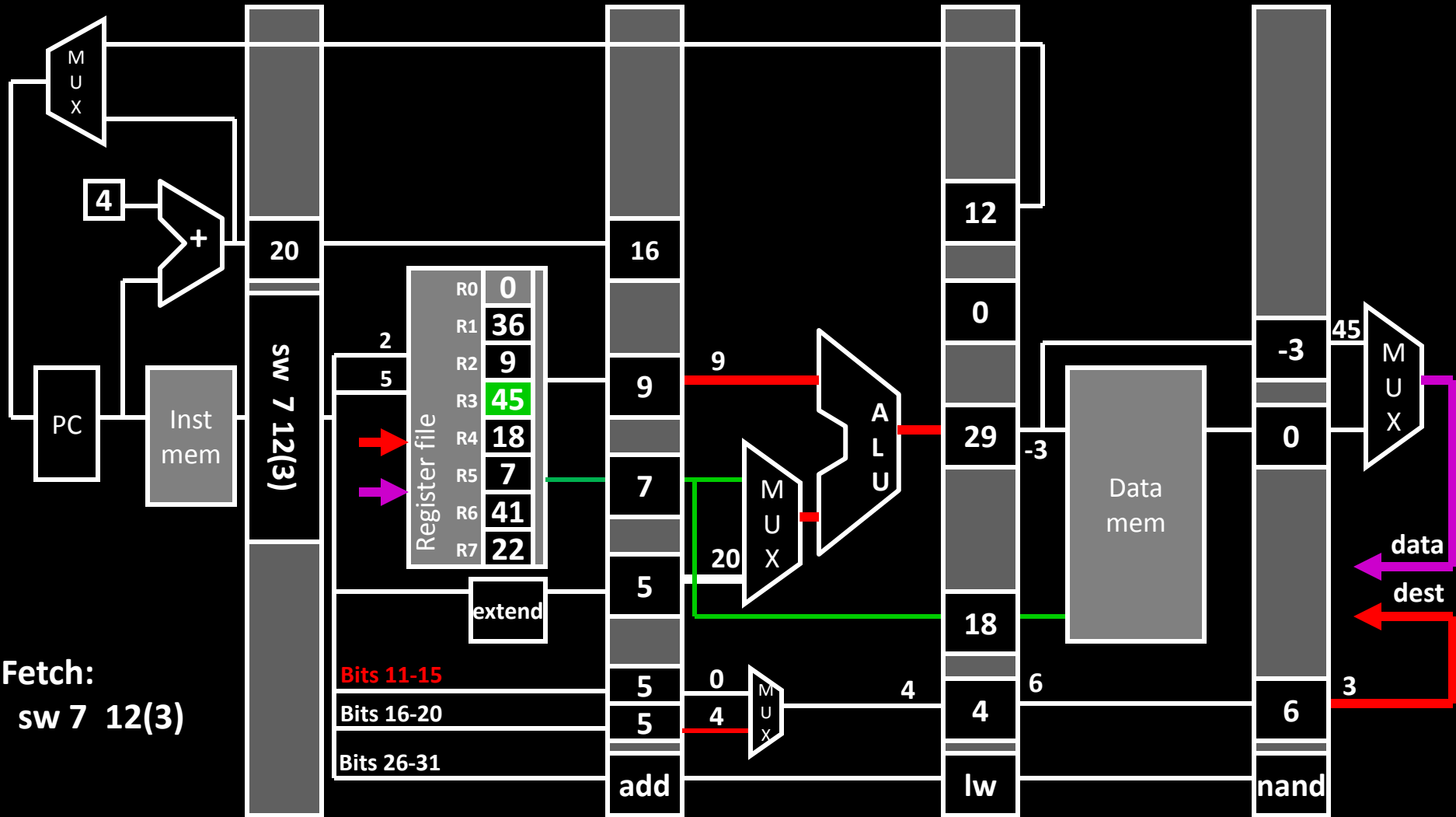
sw 7 12(3)

add 5 2 5

lw 4 20 (2)

nand 6 4 5

add 3 1 2



Fetch:
sw 7 12(3)

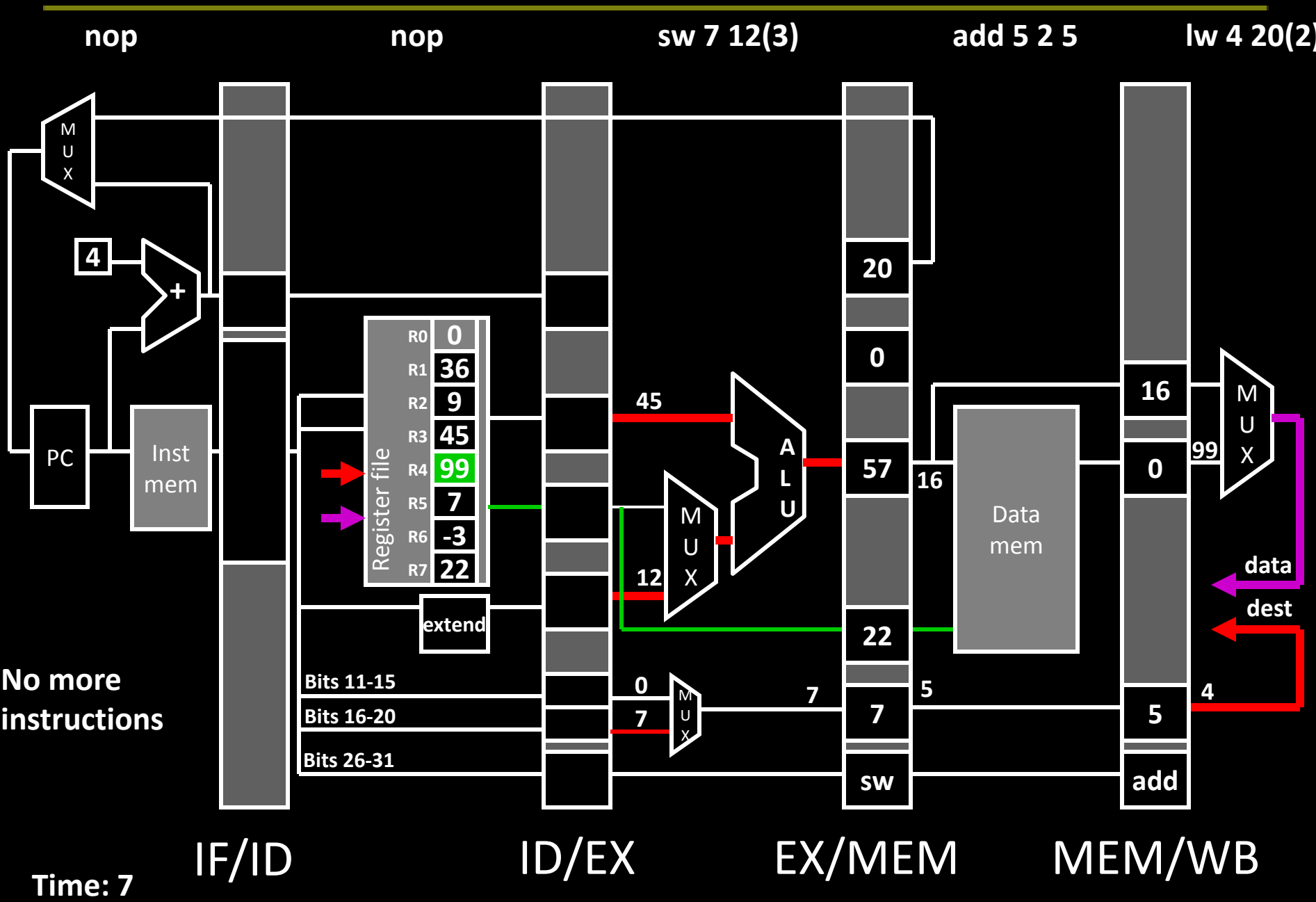
IF/ID

ID/EX

EX/MEM

MEM/WB

Time: 5



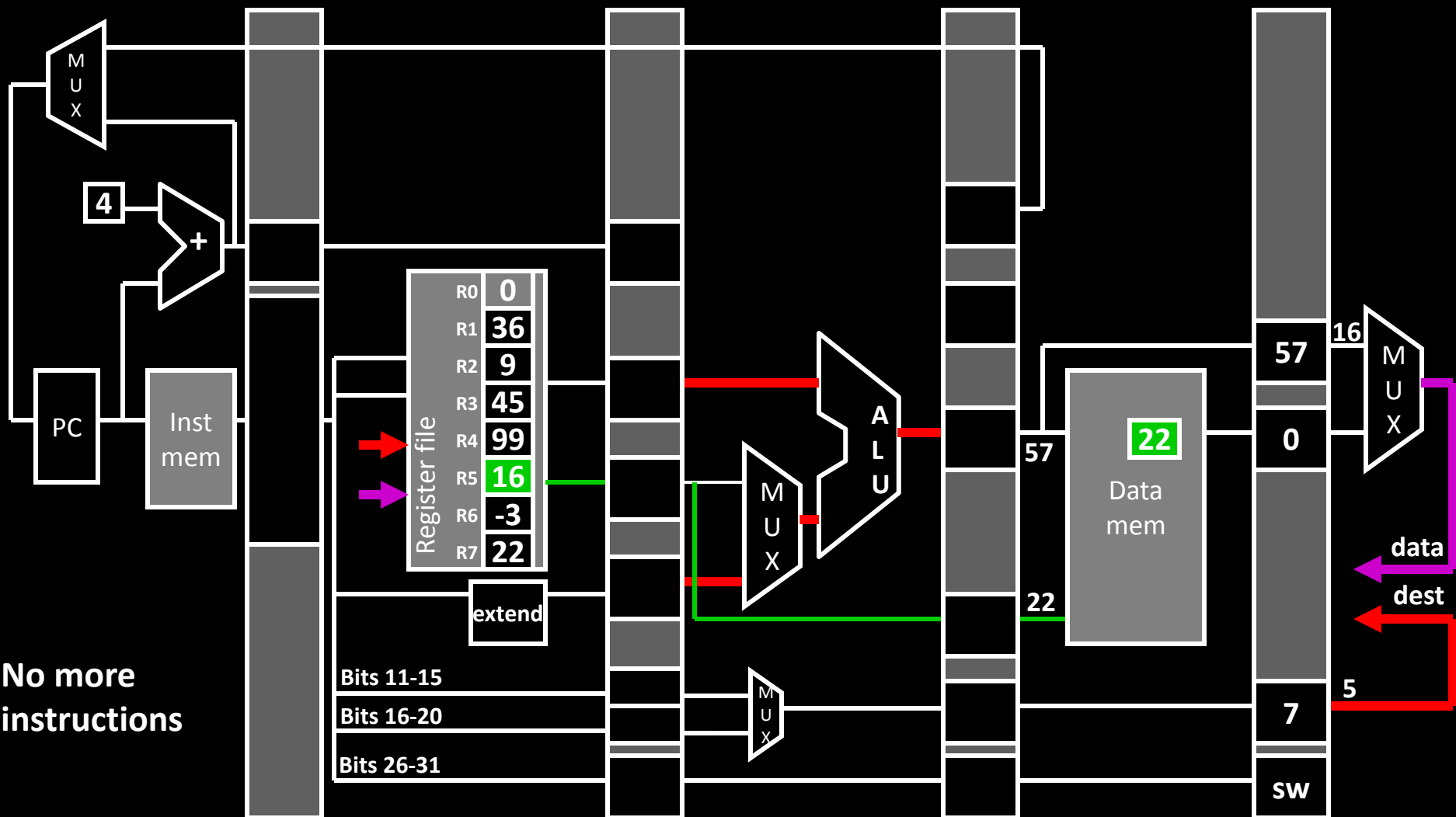
nop

nop

nop

sw 7 12(3)

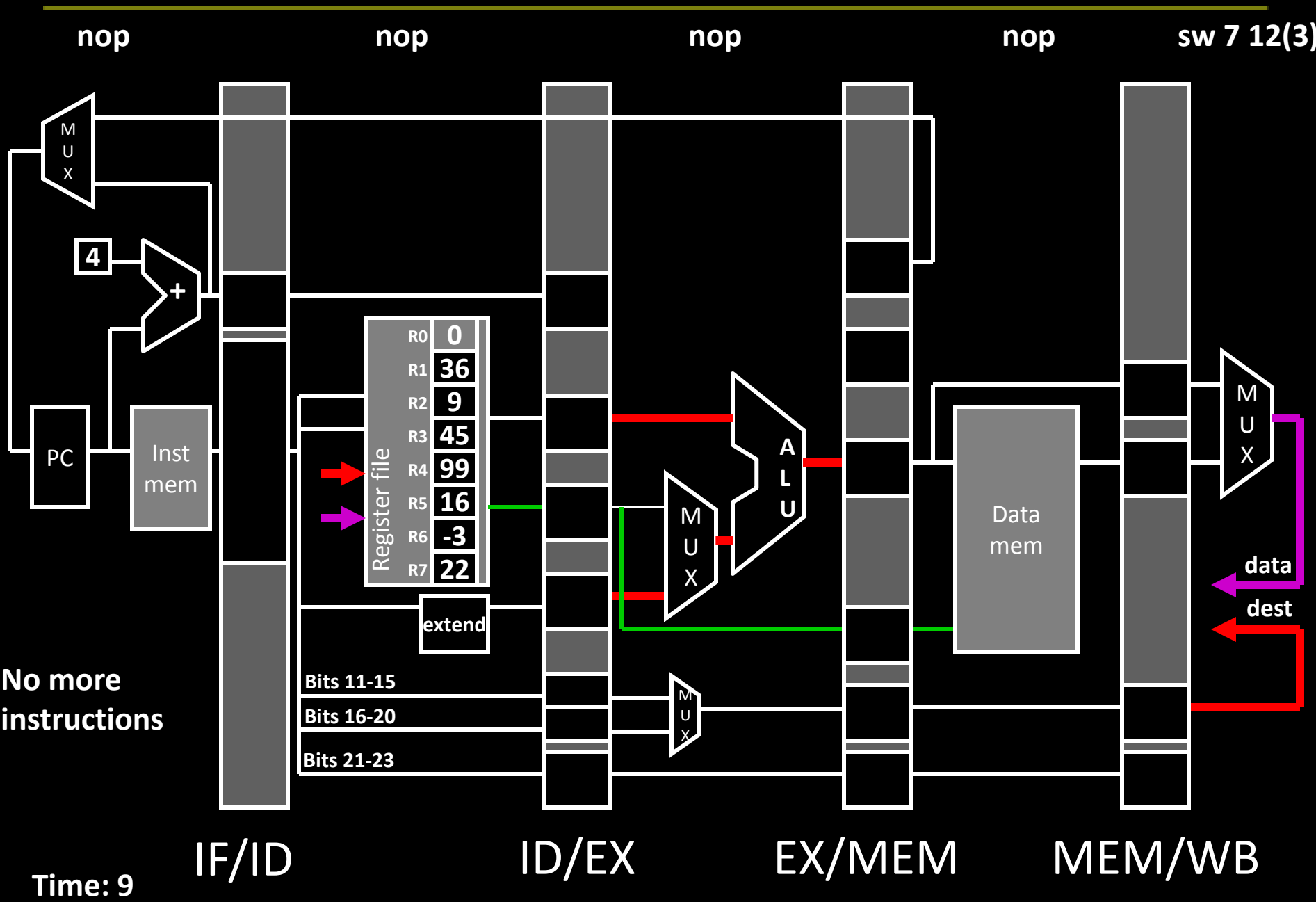
add 5 2 5



No more instructions

Time: 8

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



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)