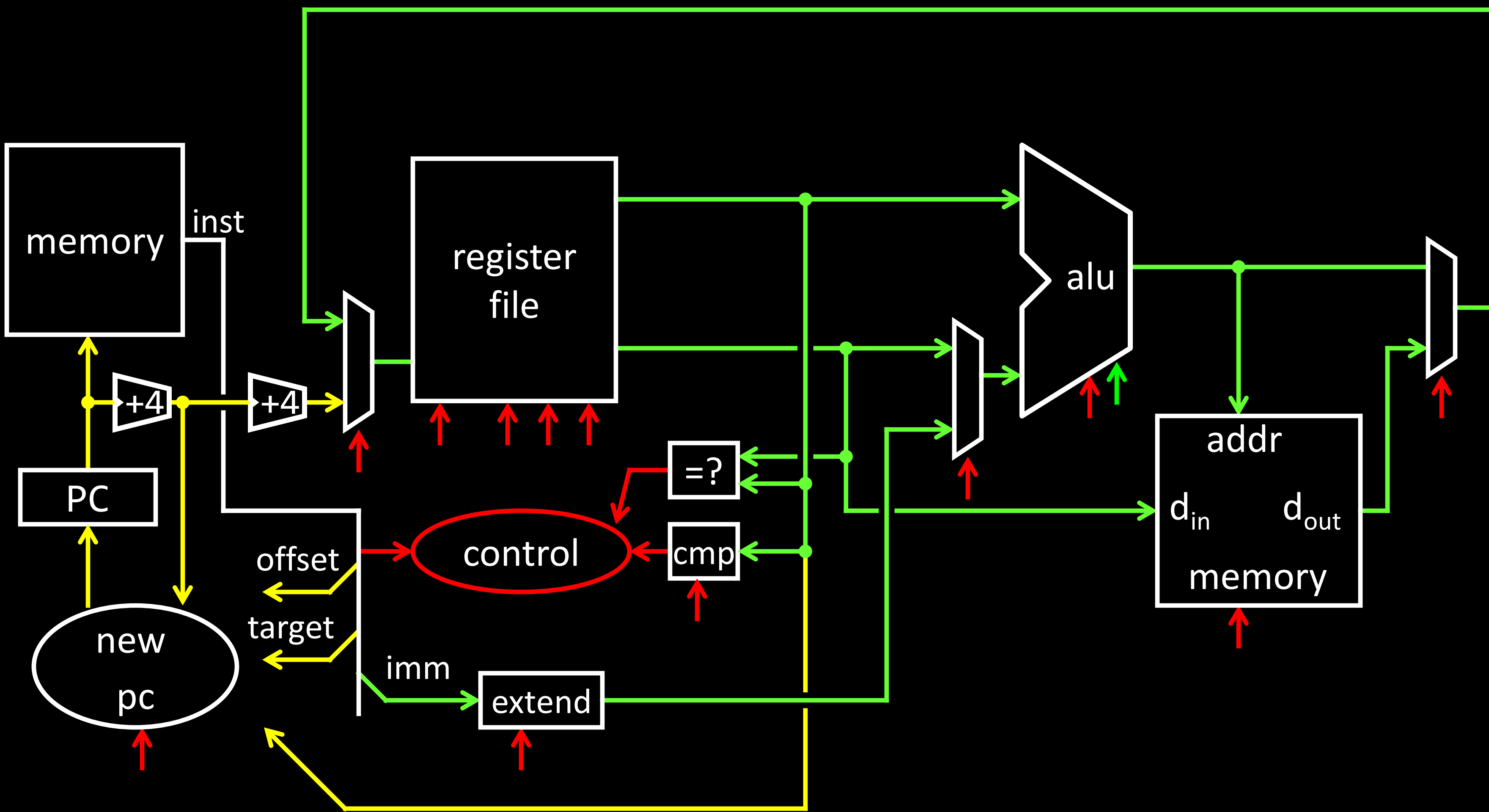
MIPS Pipeline

Hakim Weatherspoon
CS 3410, Spring 2013
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

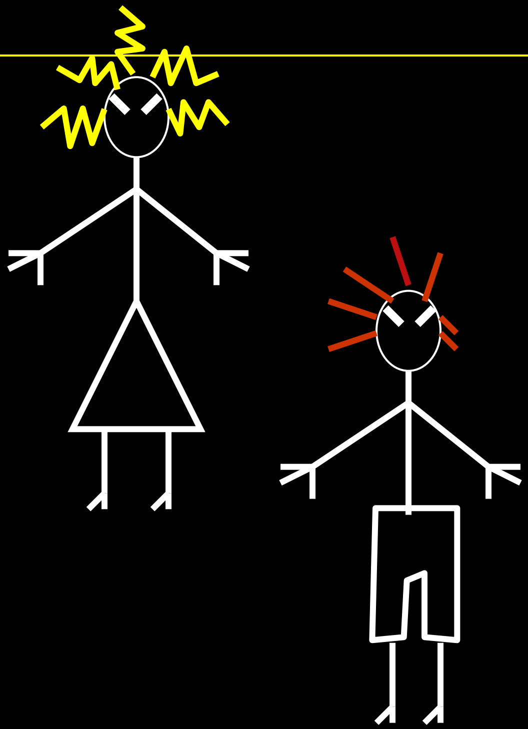
See: P&H Chapter 4.5

The Kids

Alice

Bob

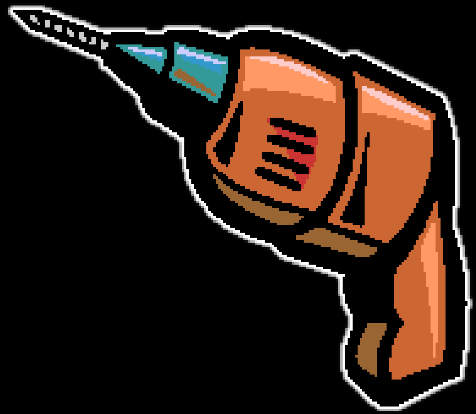
They don’t always get along…



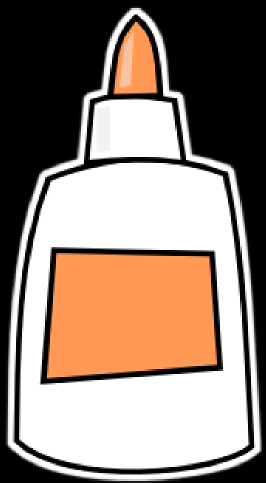
The Bicycle

The Materials

Saw



Drill

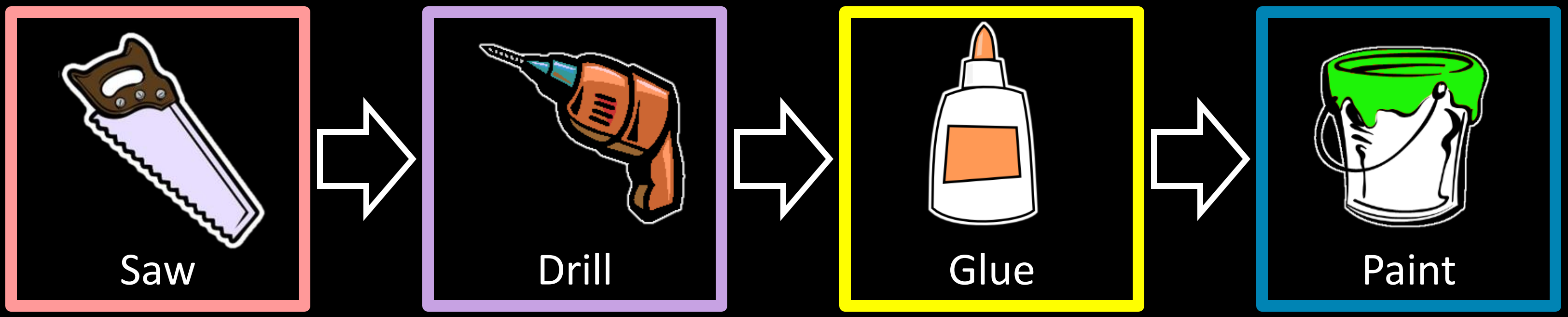


Glue

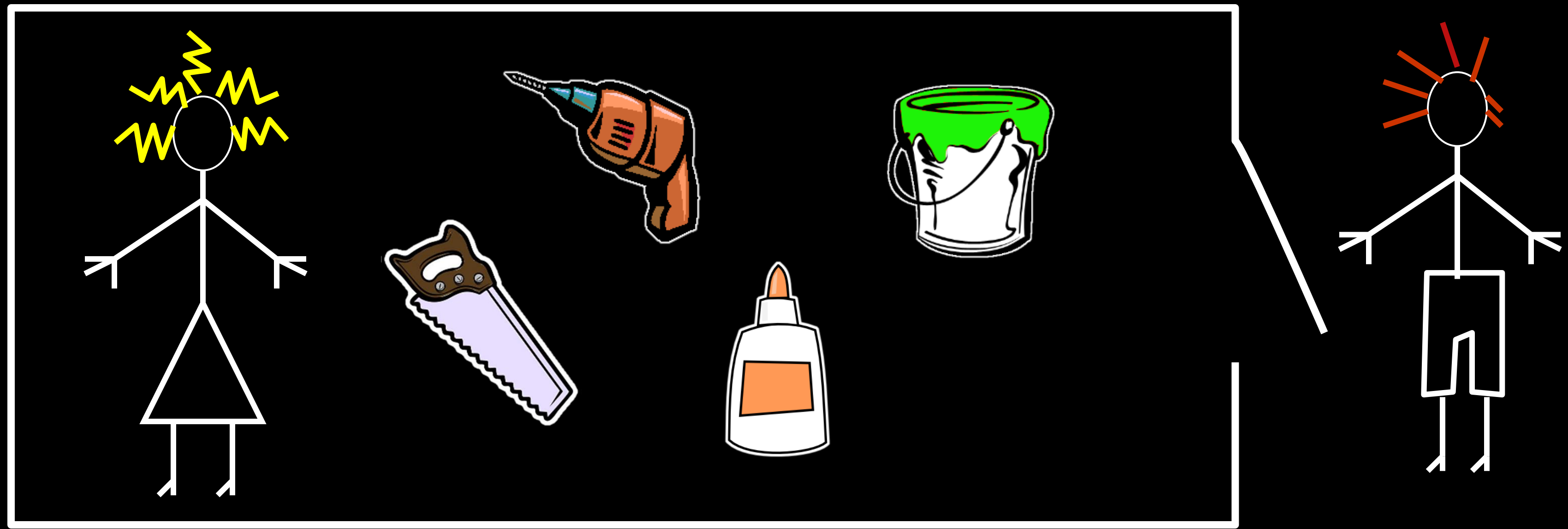
Paint

The Instructions

N pieces, each built following same sequence:



Design 1: Sequential Schedule



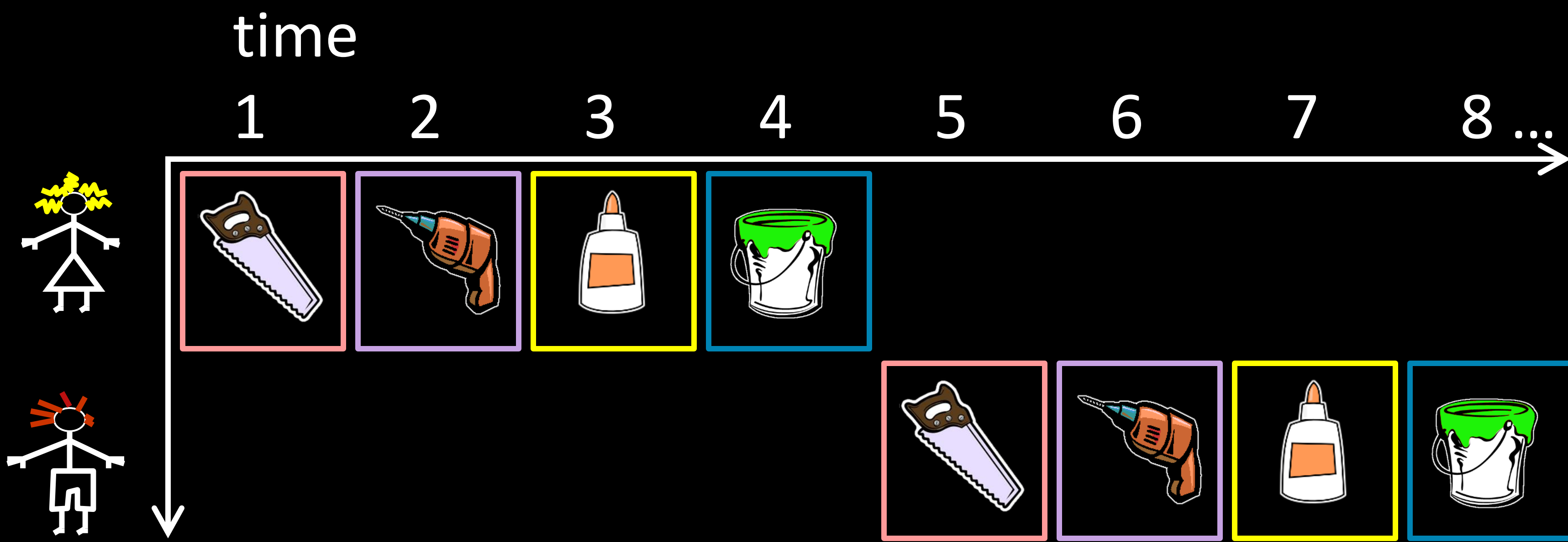
Alice owns the room

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

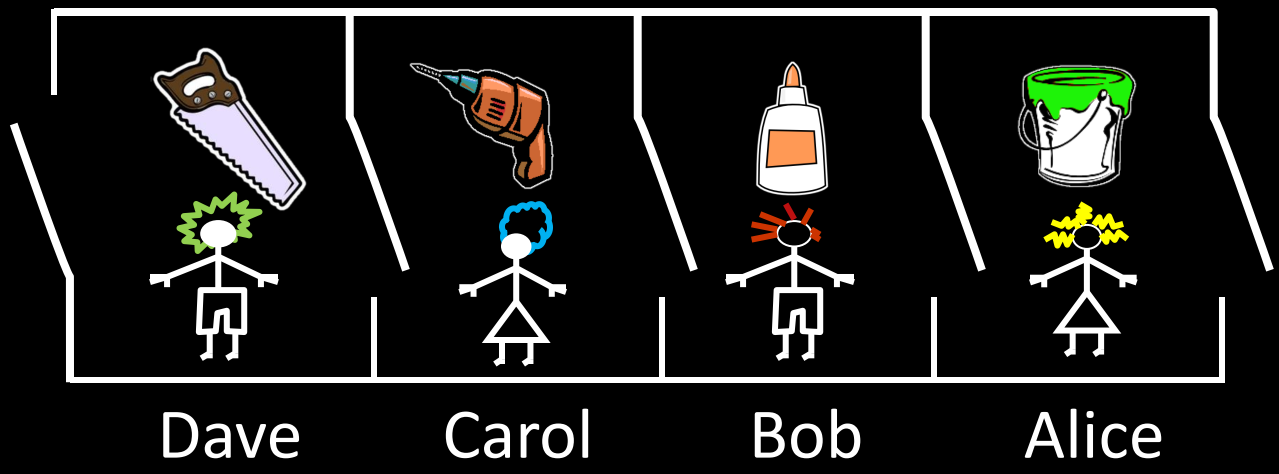
Concurrency: 1

CPI = 1

Can we do better?

Design 2: Pipelined Design

Partition room into *stages* of a *pipeline*



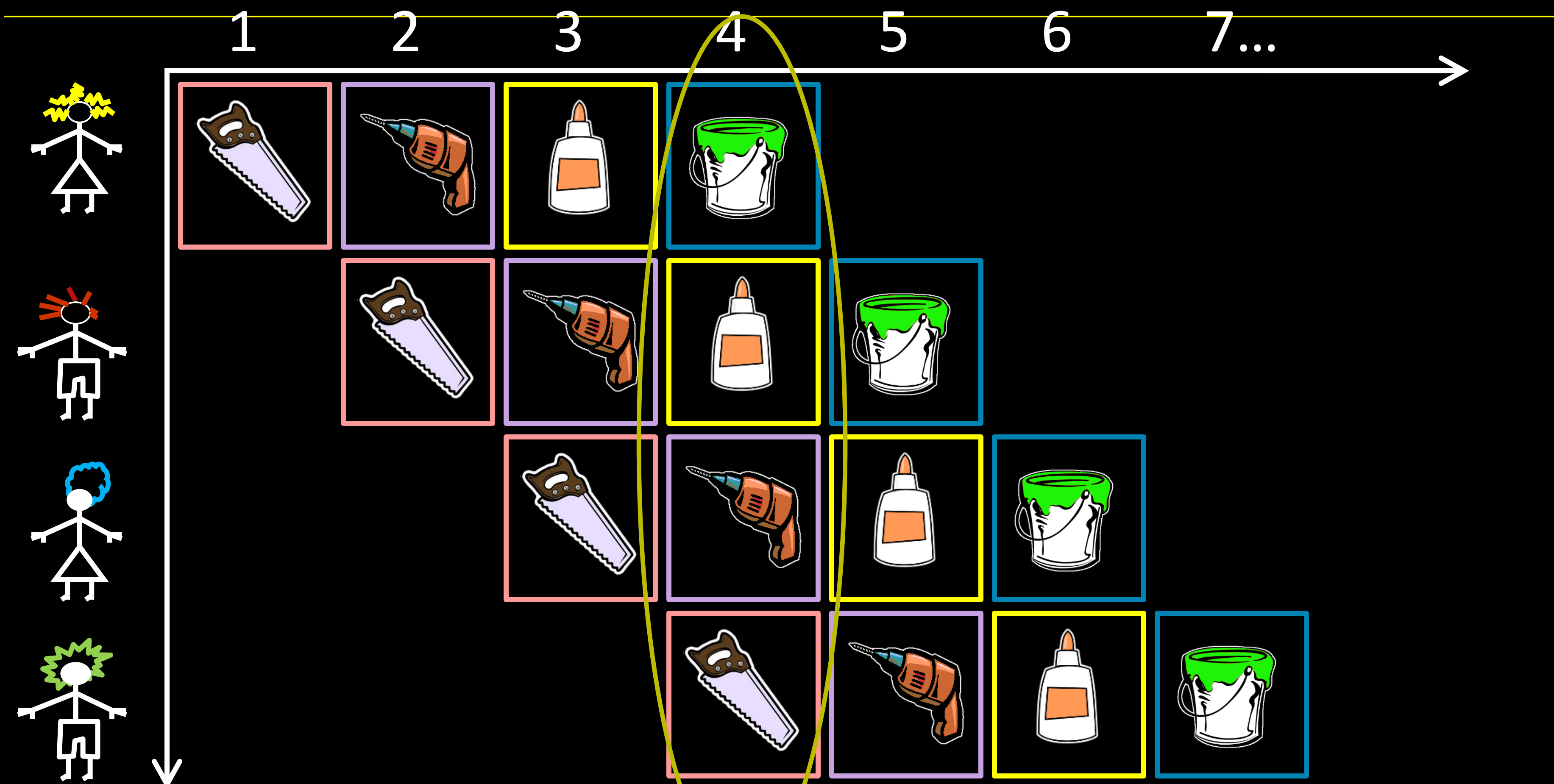
One person owns a stage at a time

4 stages

4 people working simultaneously

Everyone moves right in lockstep

Pipelined Performance



Latency: 4 hrs/task

Throughput: 1 task/hr

Concurrency: 4

CPI = 1

Lessons

Principle:

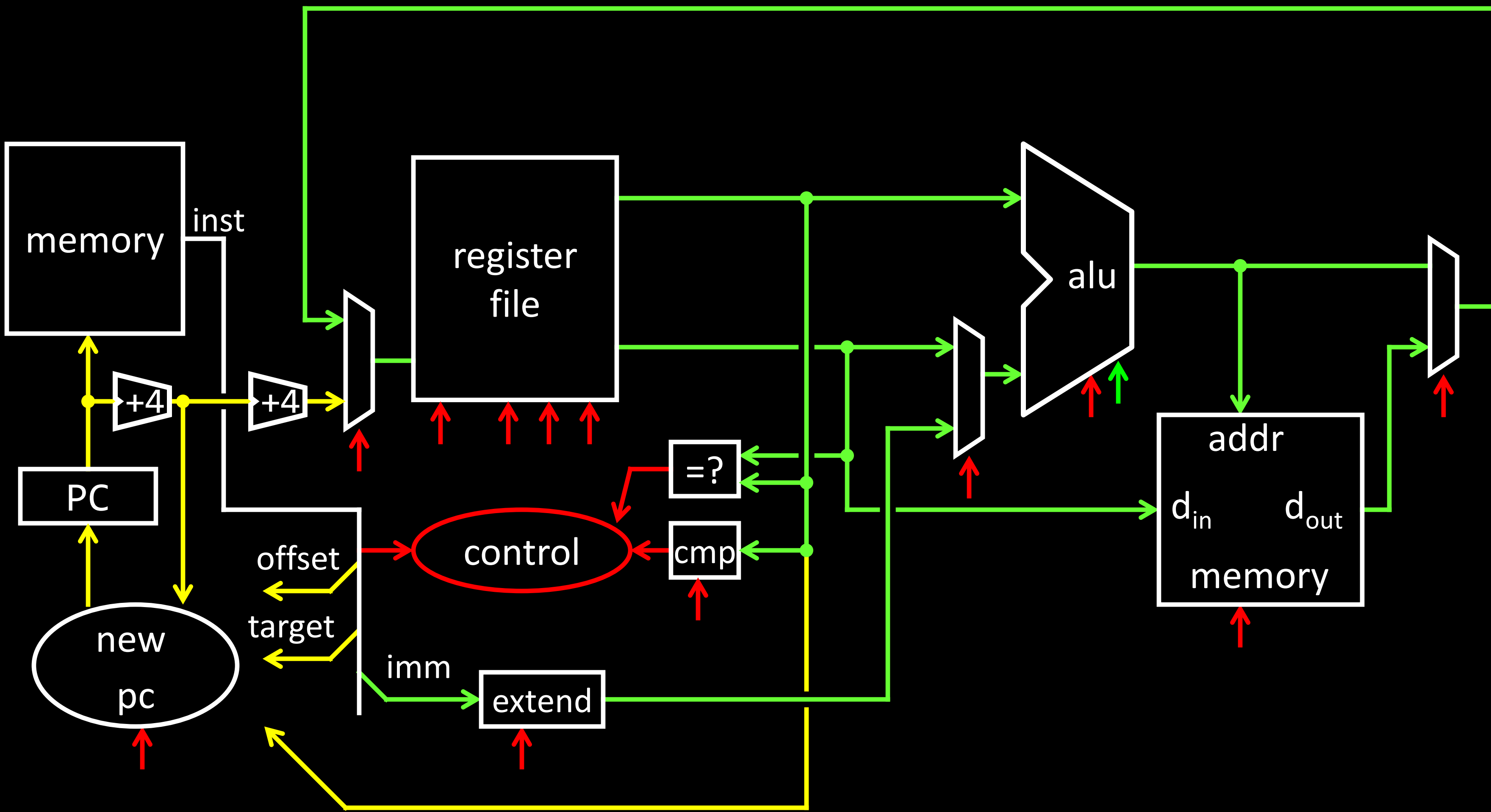
Throughput increased by parallel execution

Pipelining:

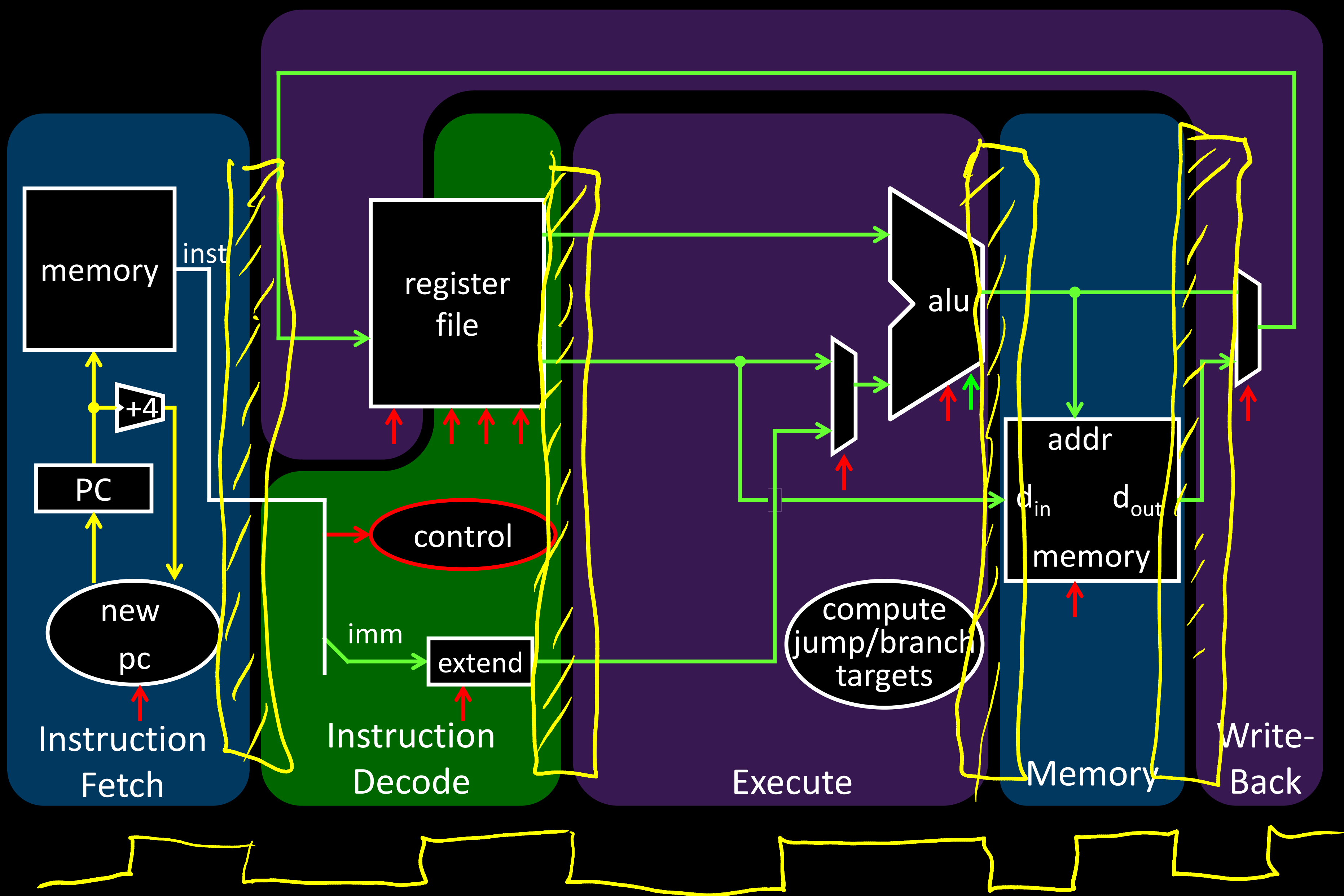
- Identify *pipeline stages*
- Isolate stages from each other
- Resolve pipeline *hazards* (Thursday)

A Processor

Review: Single cycle processor



A Processor

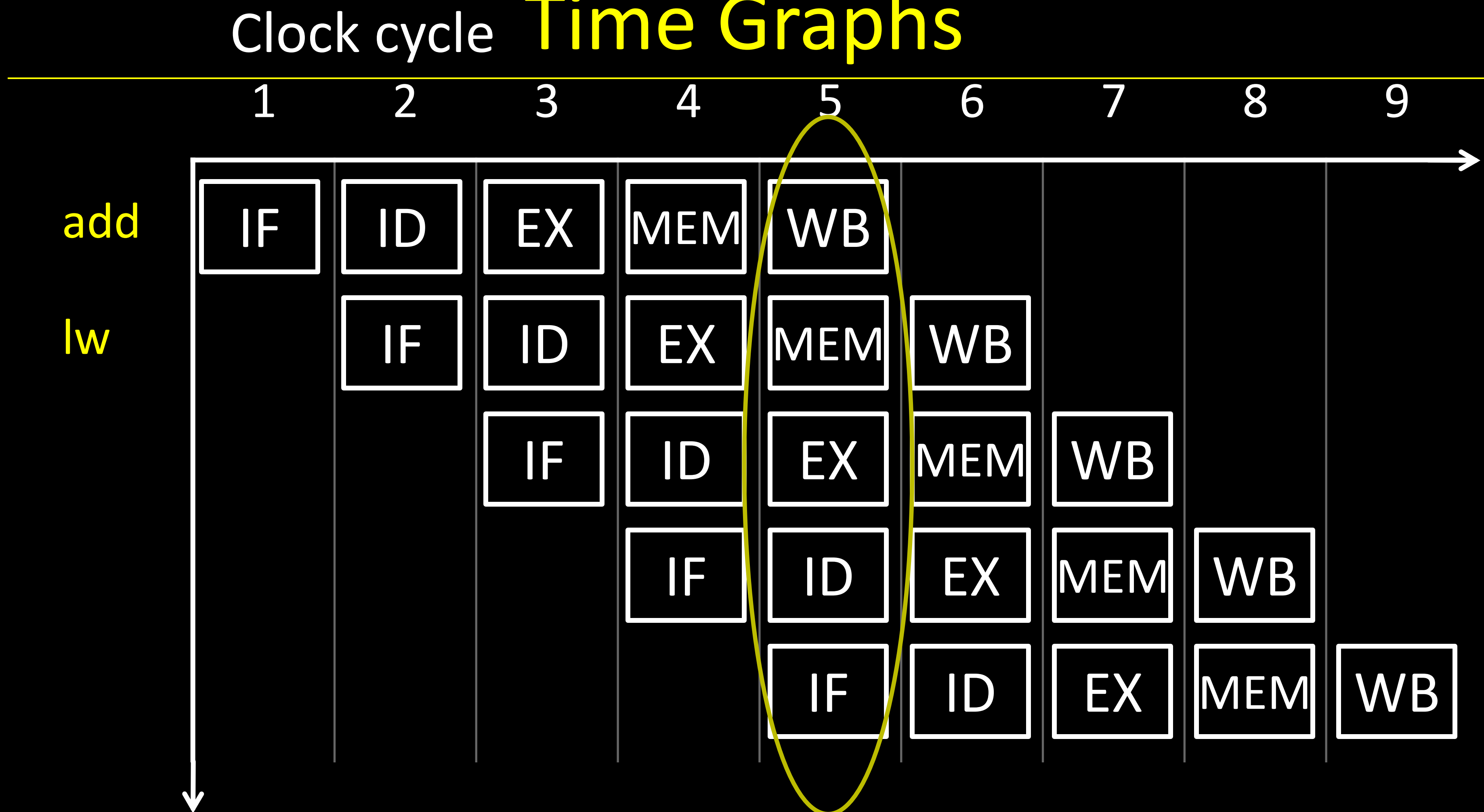

memory
inst
+4
PC
new
pc
Instruction
Fetch
register
file
control
imm
extend
Instruction
Decode
alu
compute
jump/branch
targets
Execute
addr
d_{in}
d_{out}
memory
Memory
Write-
Back

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



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)

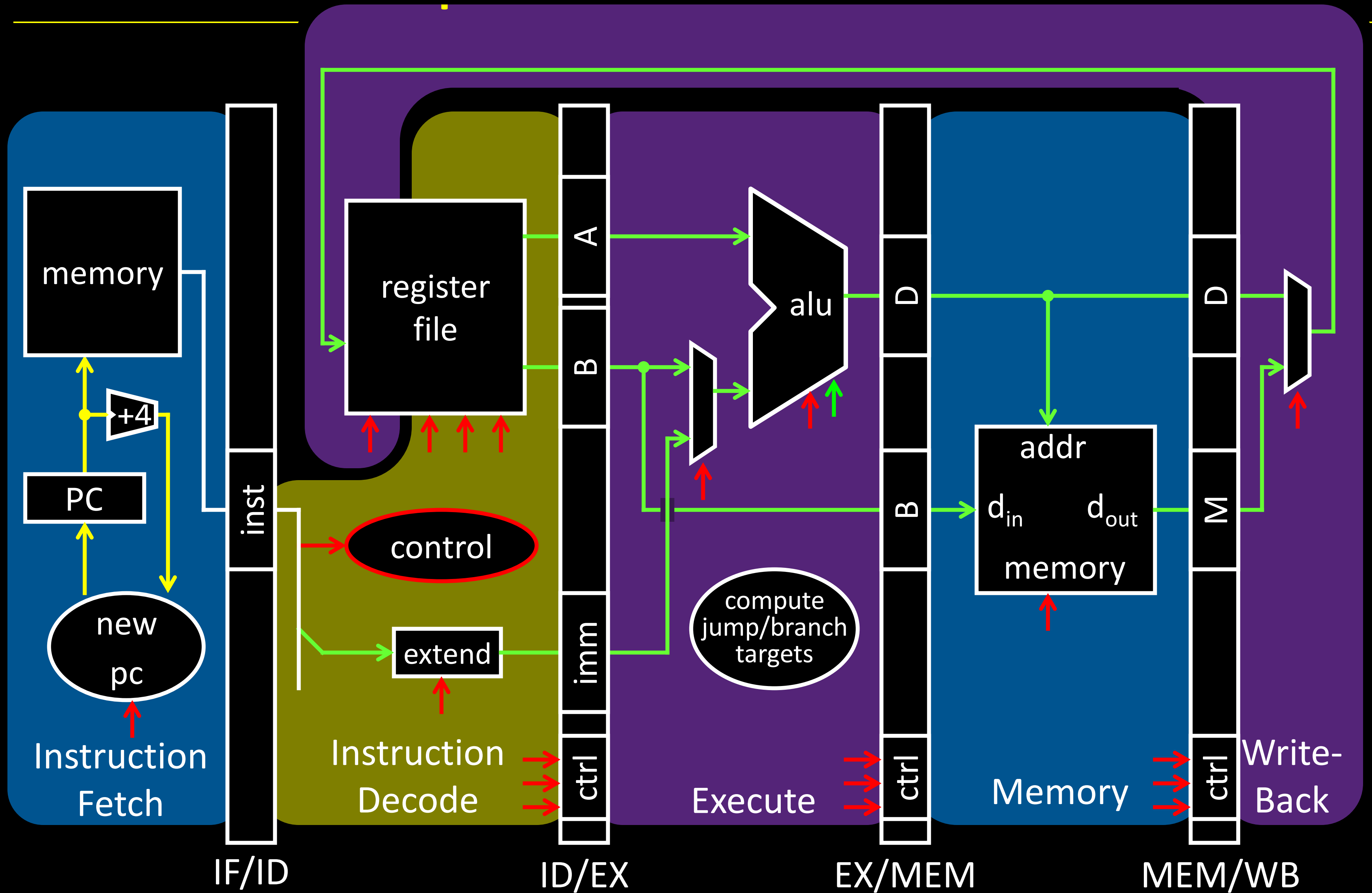
Design a separate stage for the execution performed during each clock cycle

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

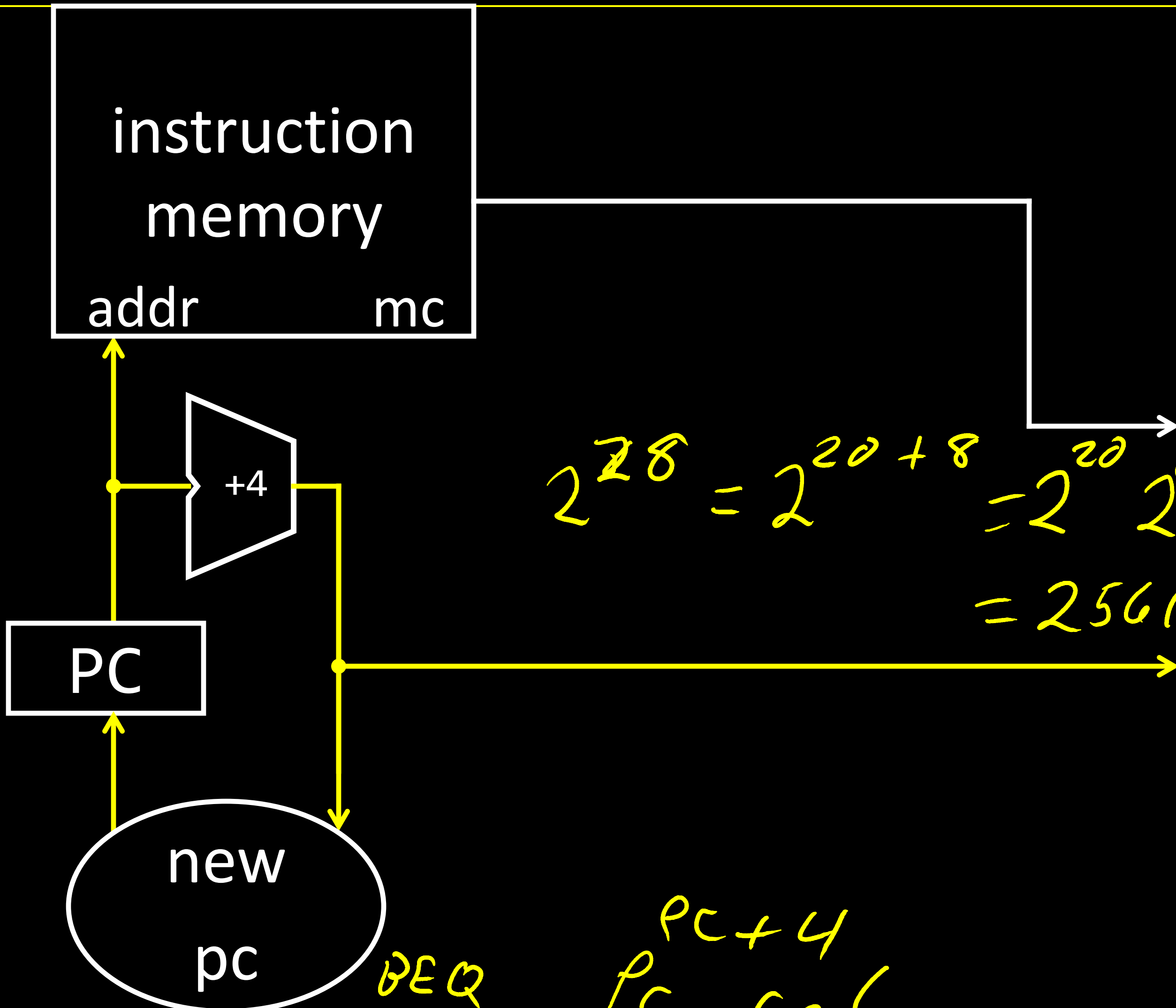
Fetch a new instruction every cycle

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

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

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

IF



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

$= 256M$

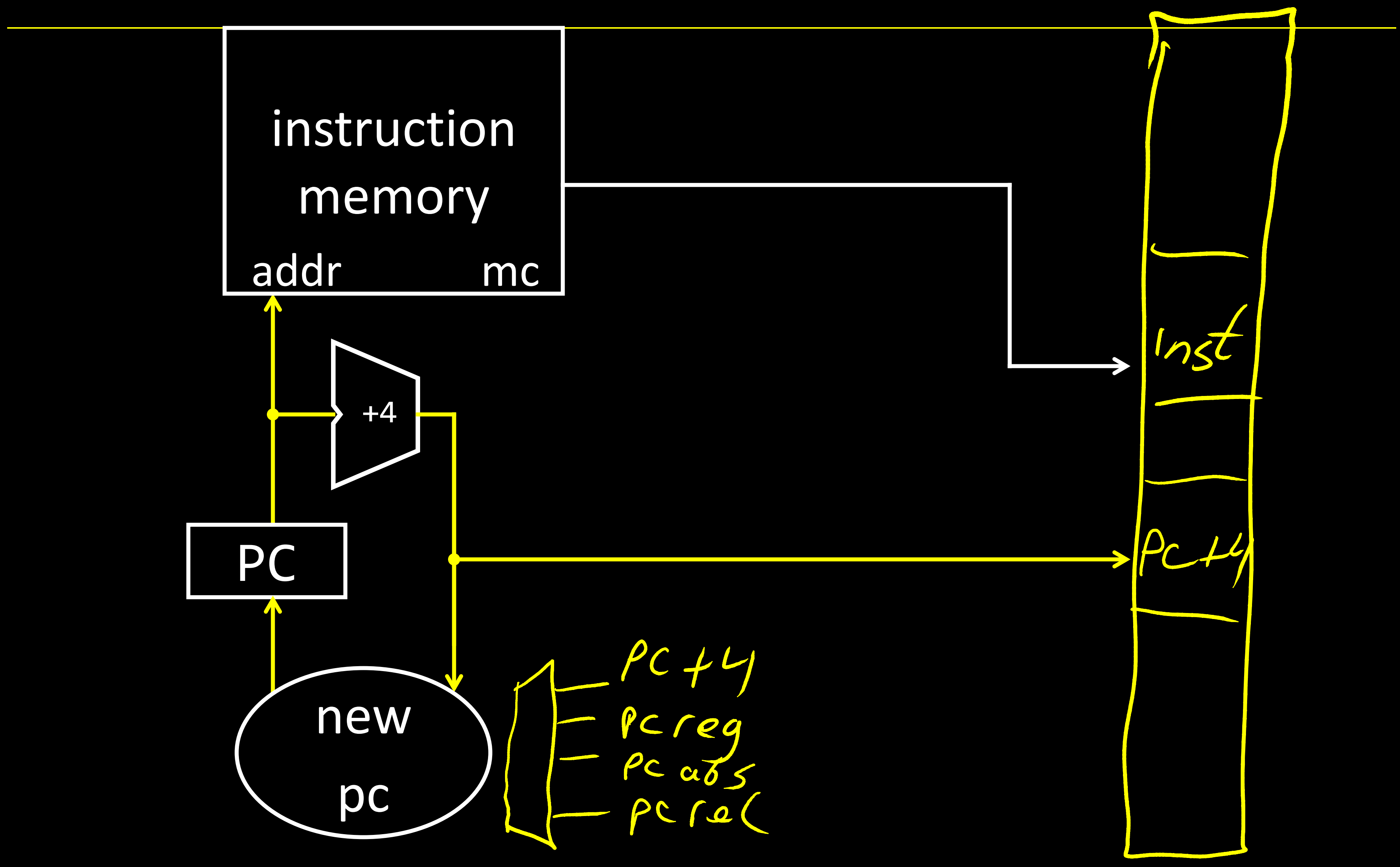
pc+4

BEQ pc rel

J pc abs $(pc+4)_{31..28}$ (targ<<2)

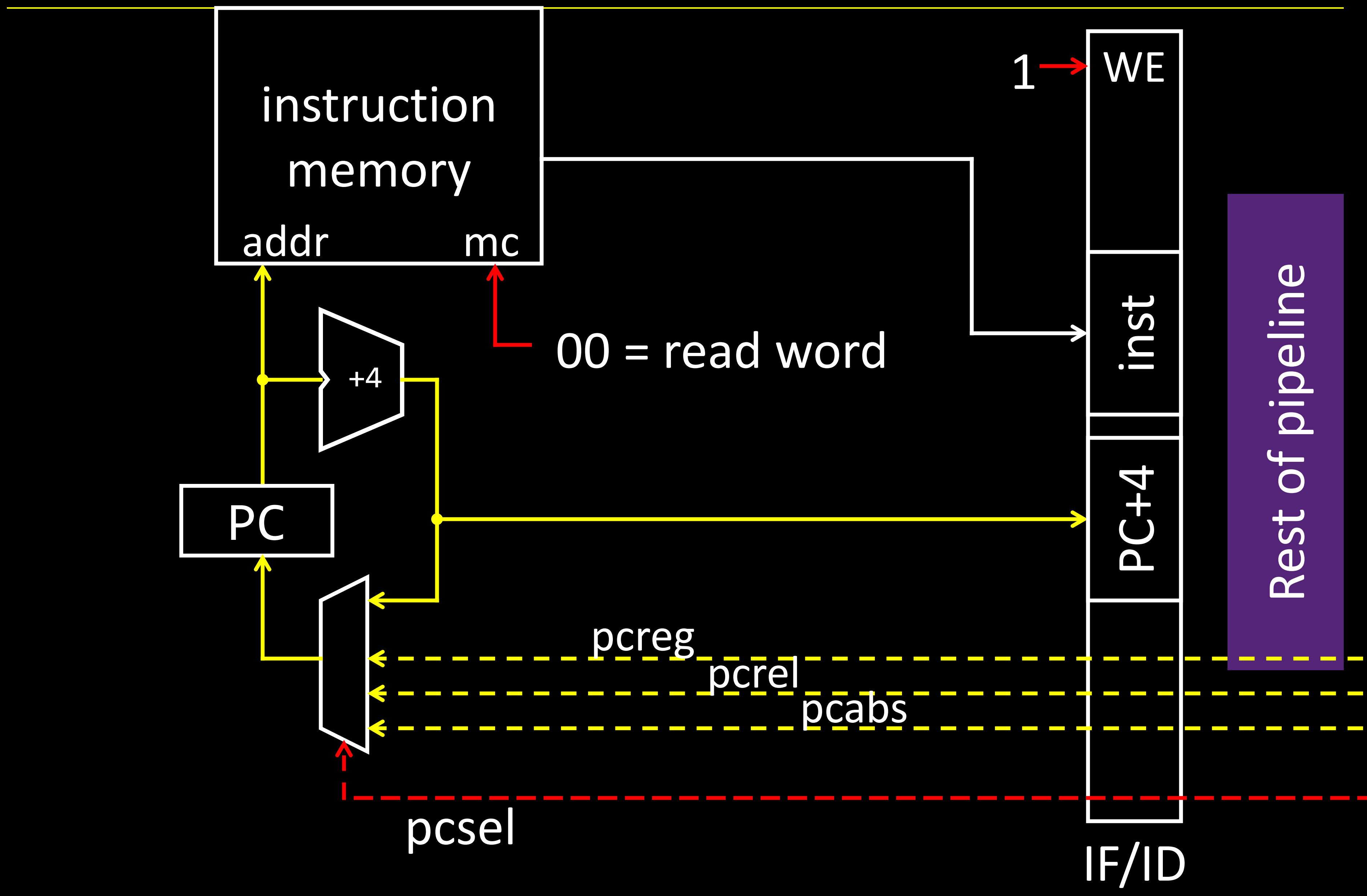
JR pc reg

IF



instruction memory
addr
mc
+4
PC
new pc
pc+4
pc reg
pc abs
pc rel
inst
pc+4

IF


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

ID

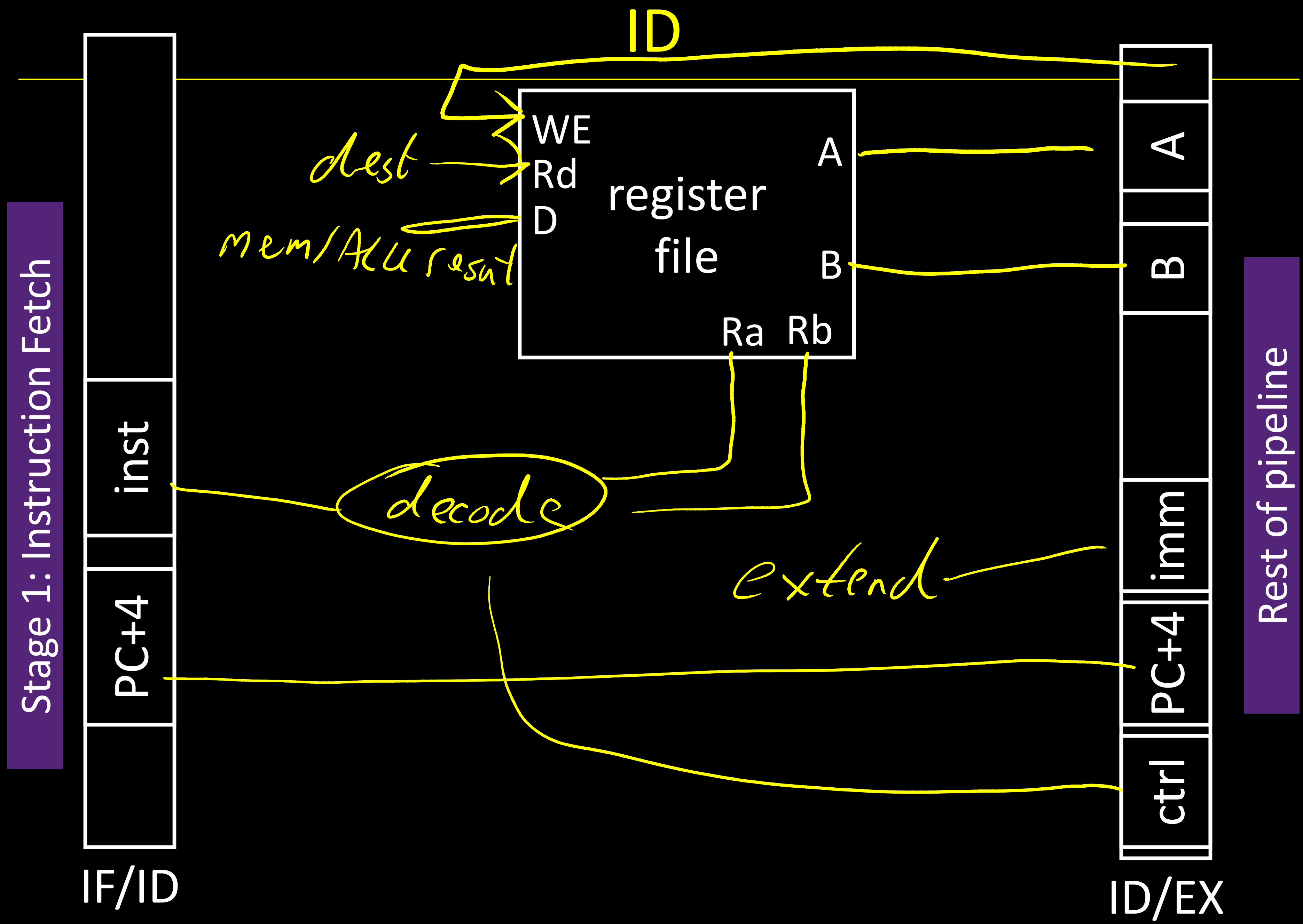
Stage 2: Instruction Decode

On every cycle:

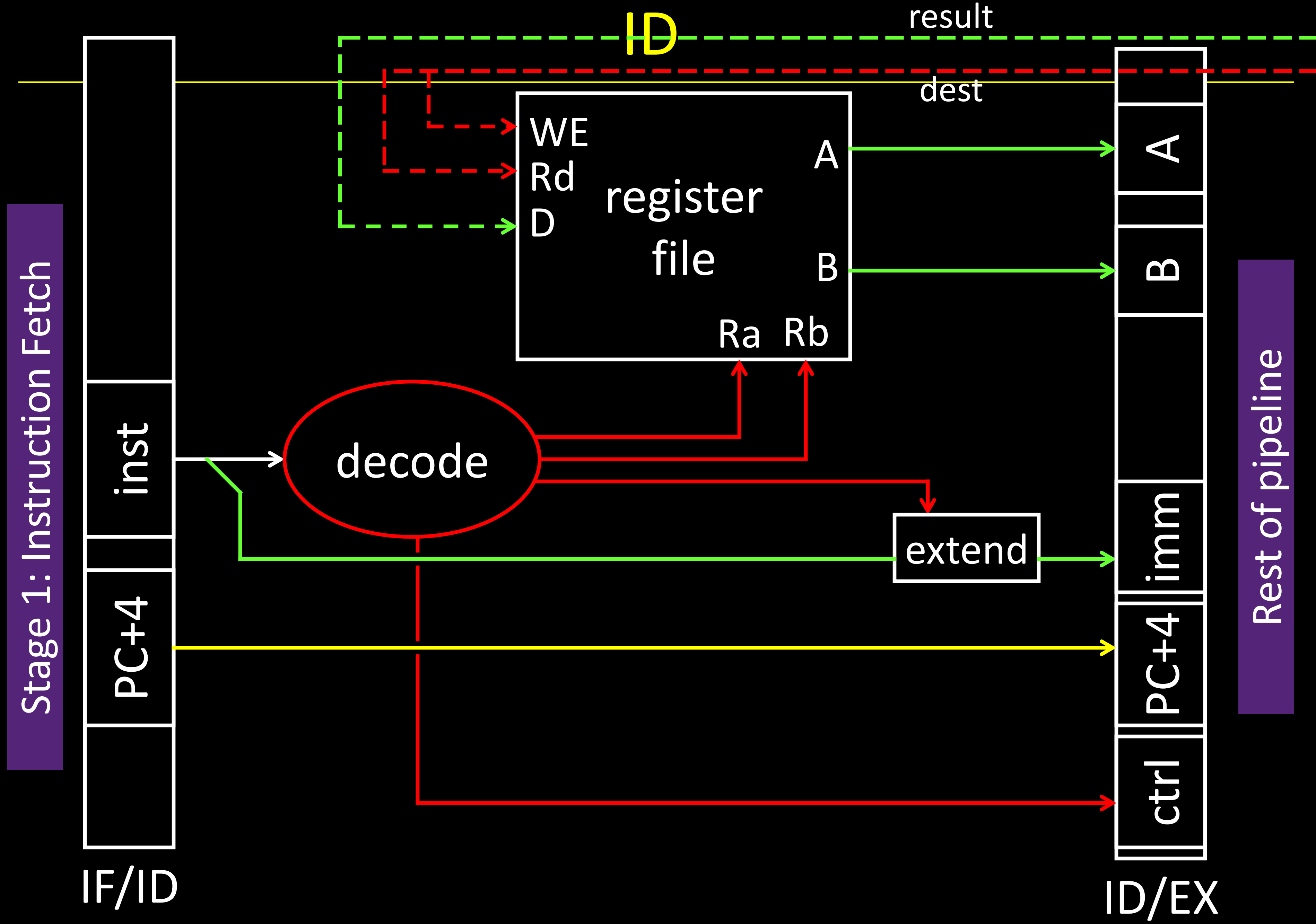
- Read IF/ID pipeline register to get instruction bits
- Decode instruction, generate control signals
- Read from register file

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


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

EX

Stage 3: Execute

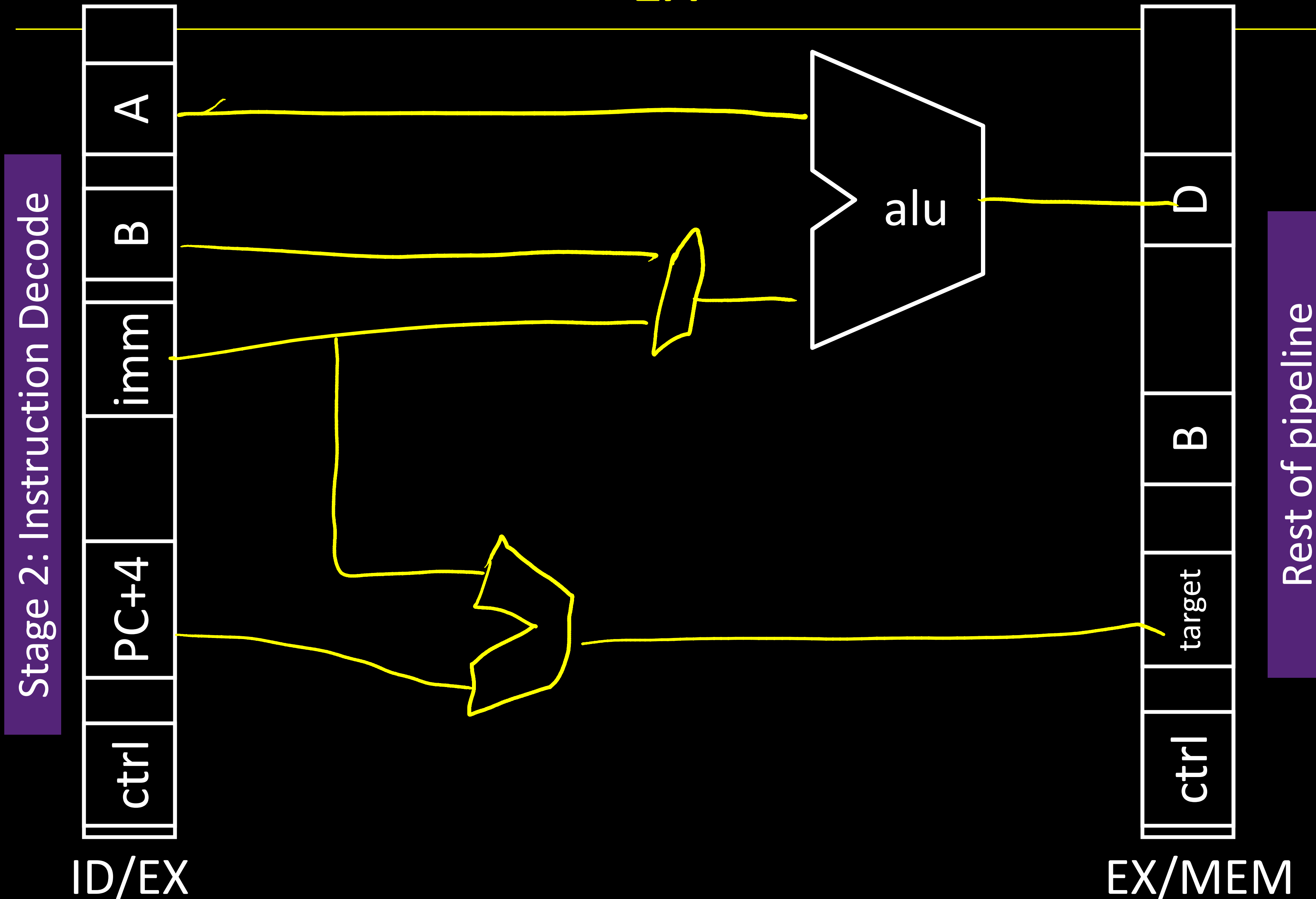
On every cycle:

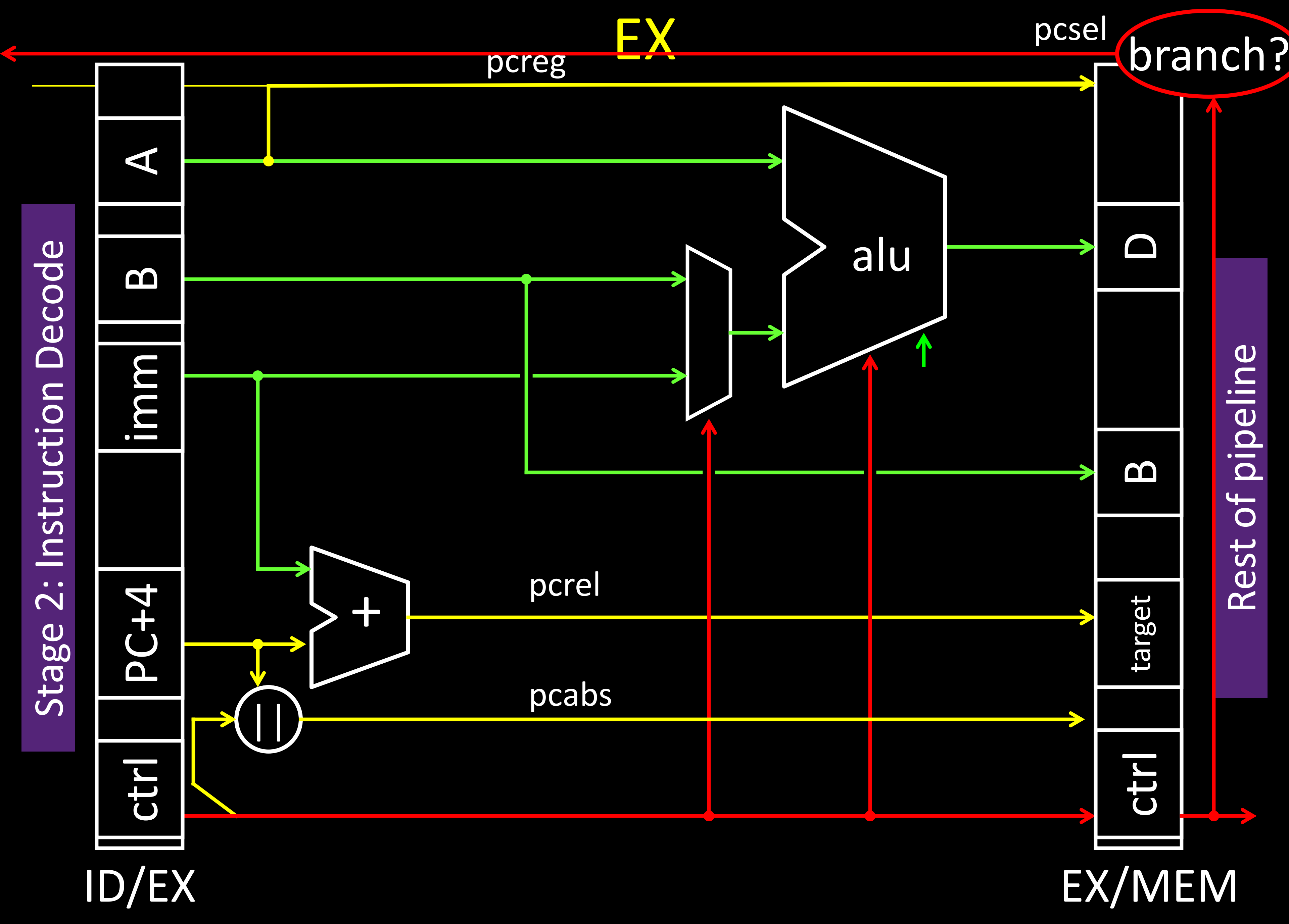
- Read ID/EX pipeline register to get values and control bits
- Perform ALU operation
- Compute targets (PC+4+offset, etc.) *in case* this is a branch
- Decide if jump/branch should be taken

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


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


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

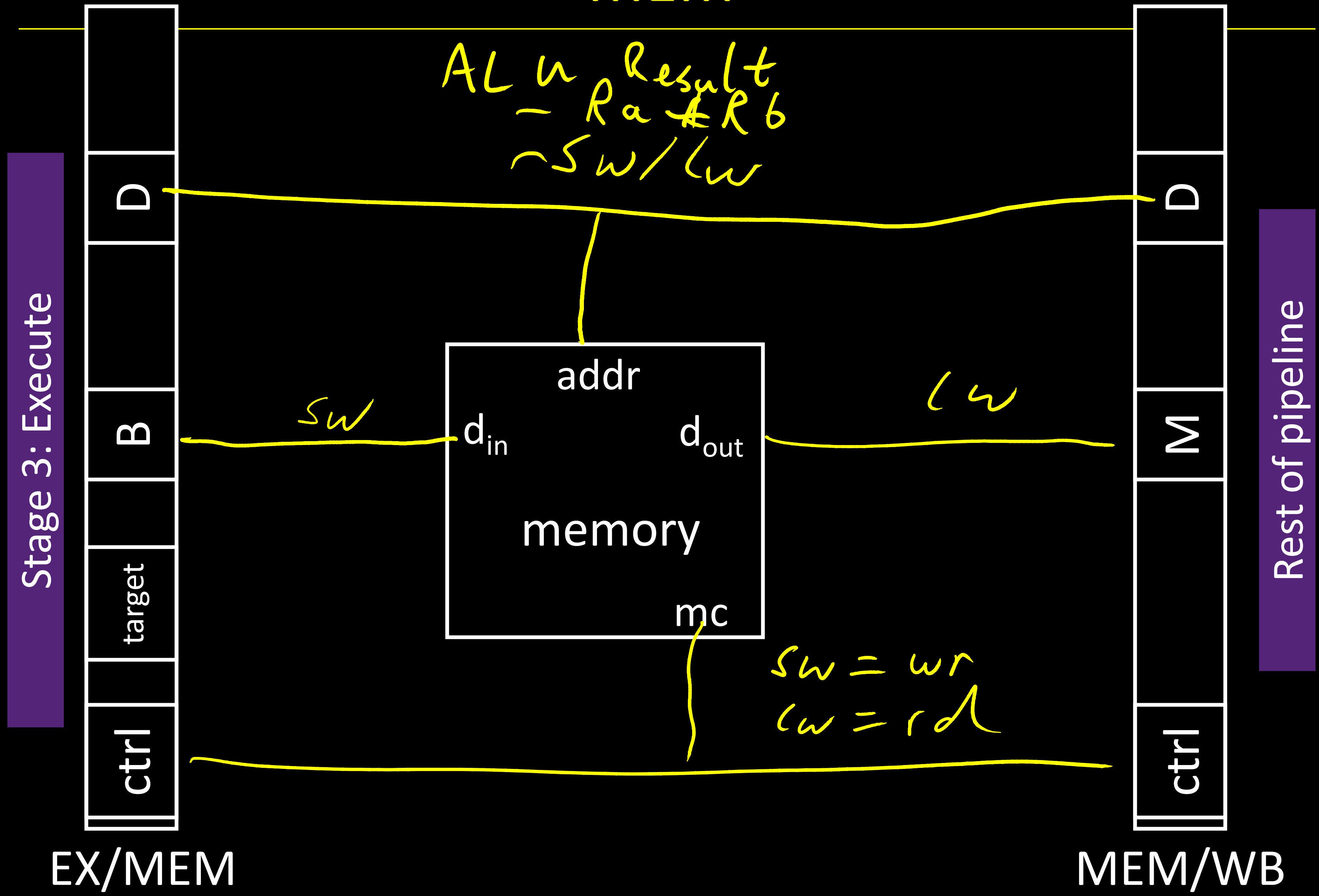
On every cycle:

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

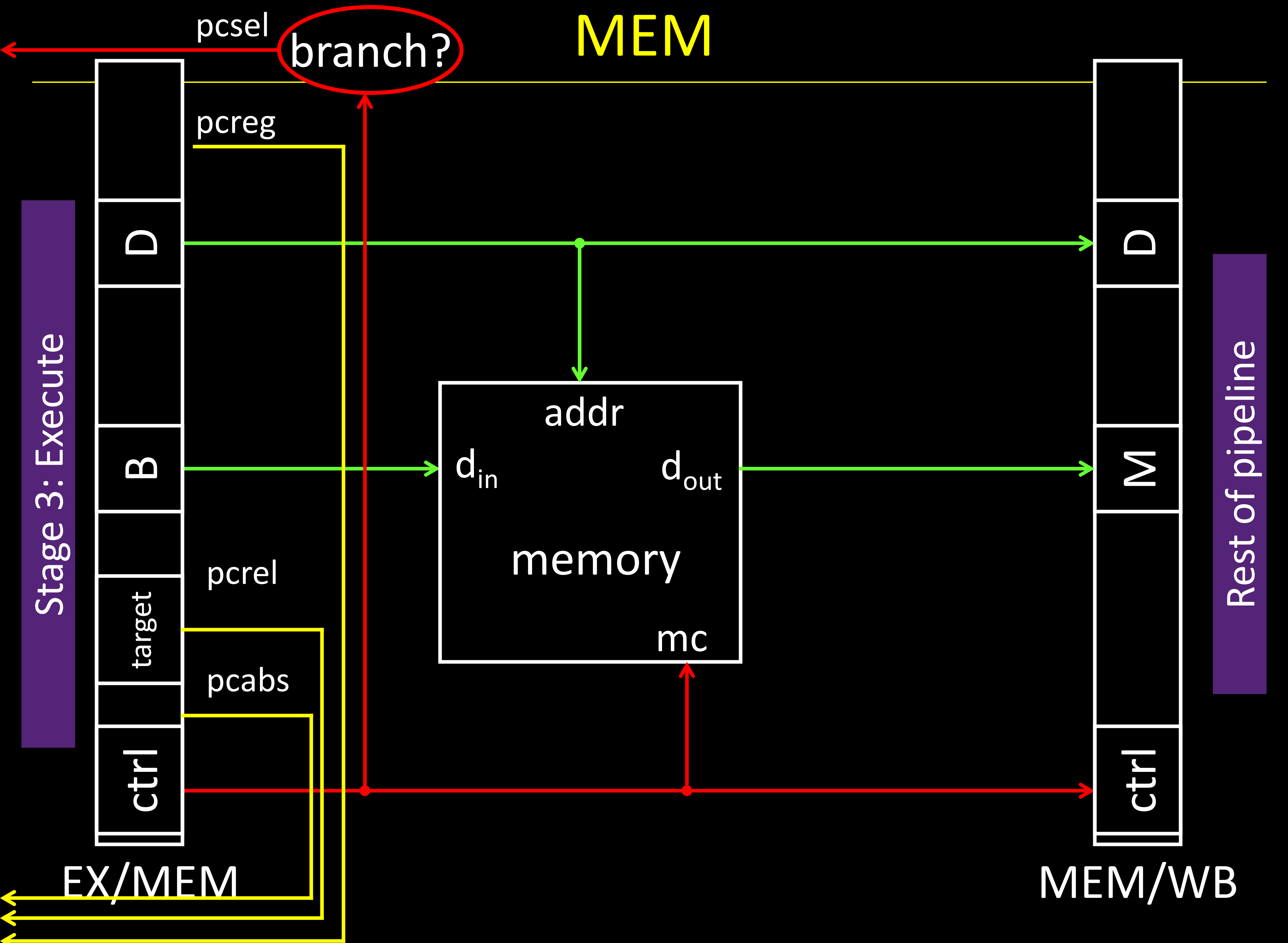
Write values of interest to pipeline register (MEM/WB)

- Control information, Rd index, …
- Result of memory operation
- Pass result of ALU operation

MEM



Stage 3: Execute
D
B
target
ctrl
ALU Result
– Ra & Rb
– sw/lw
addr
sw
d_{in}
d_{out}
lw
memory
mc
sw = wr
lw = rd
D
M
ctrl
Rest of pipeline
EX/MEM
MEM/WB


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

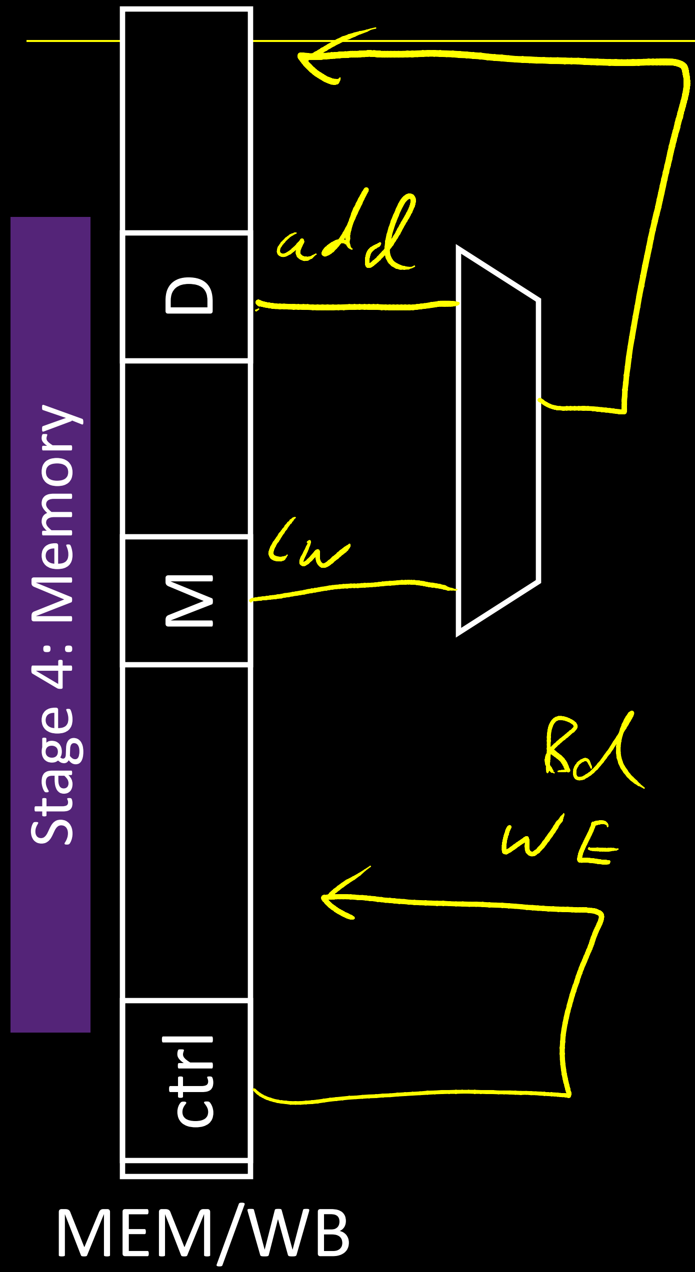
WB

Stage 5: Write-back

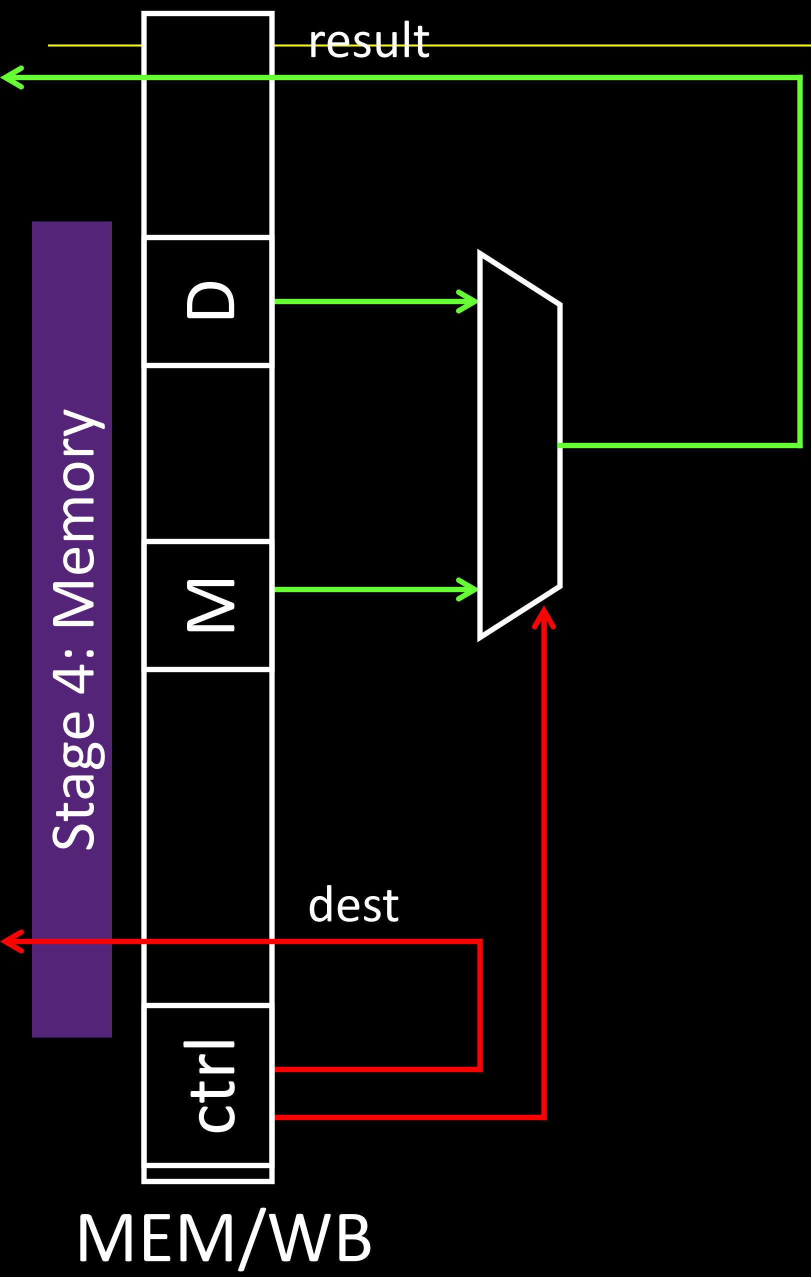
On every cycle:

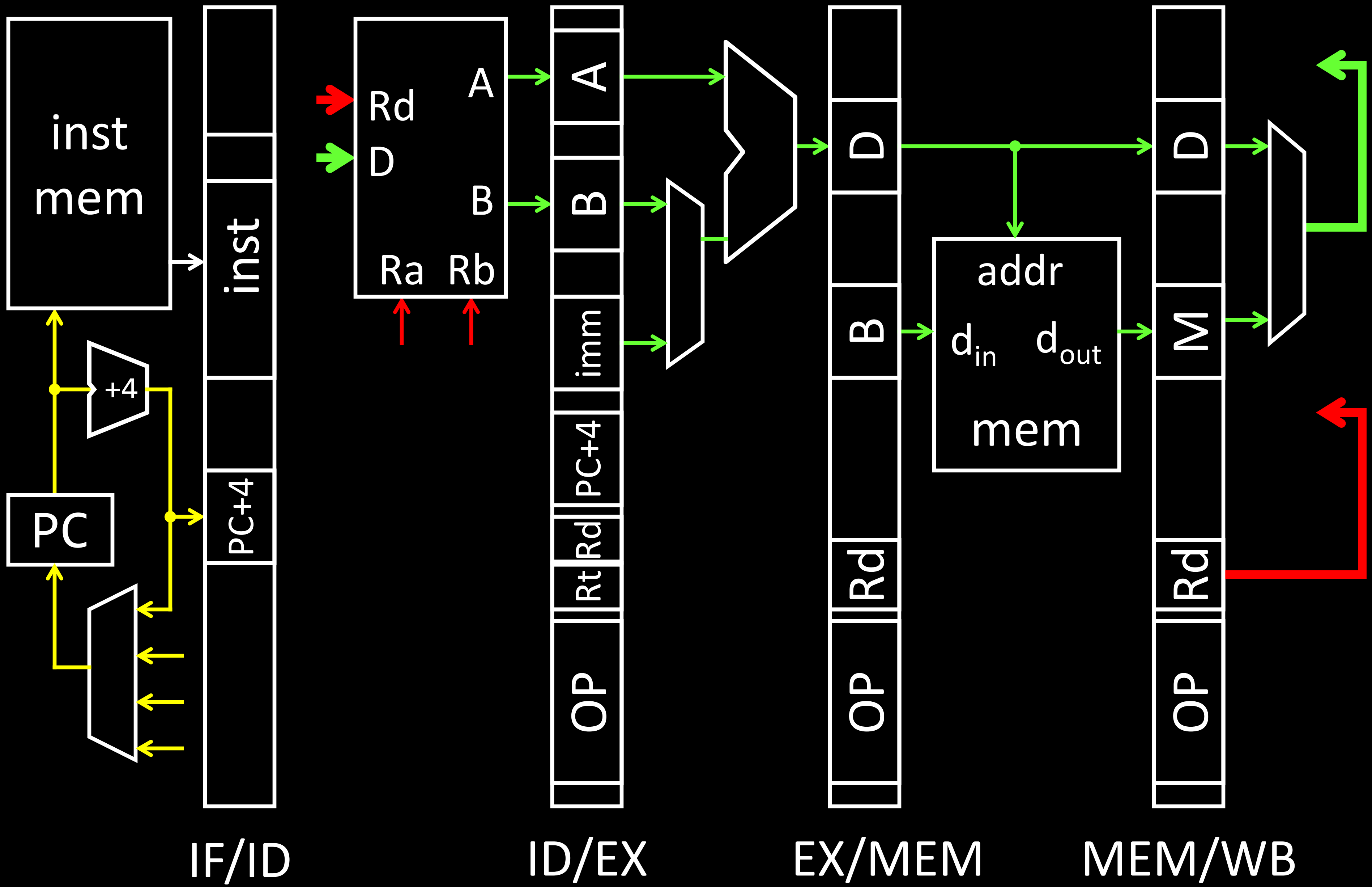
- Read MEM/WB pipeline register to get values and control bits
- Select value and write to register file

WB



WB




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

Administrivia

Required: partner for group project

Project1 (PA1) and Homework2 (HW2) are both out

PA1 Design Doc and HW2 due in one week, start early

Work alone on HW2, but in group for PA1

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
- 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

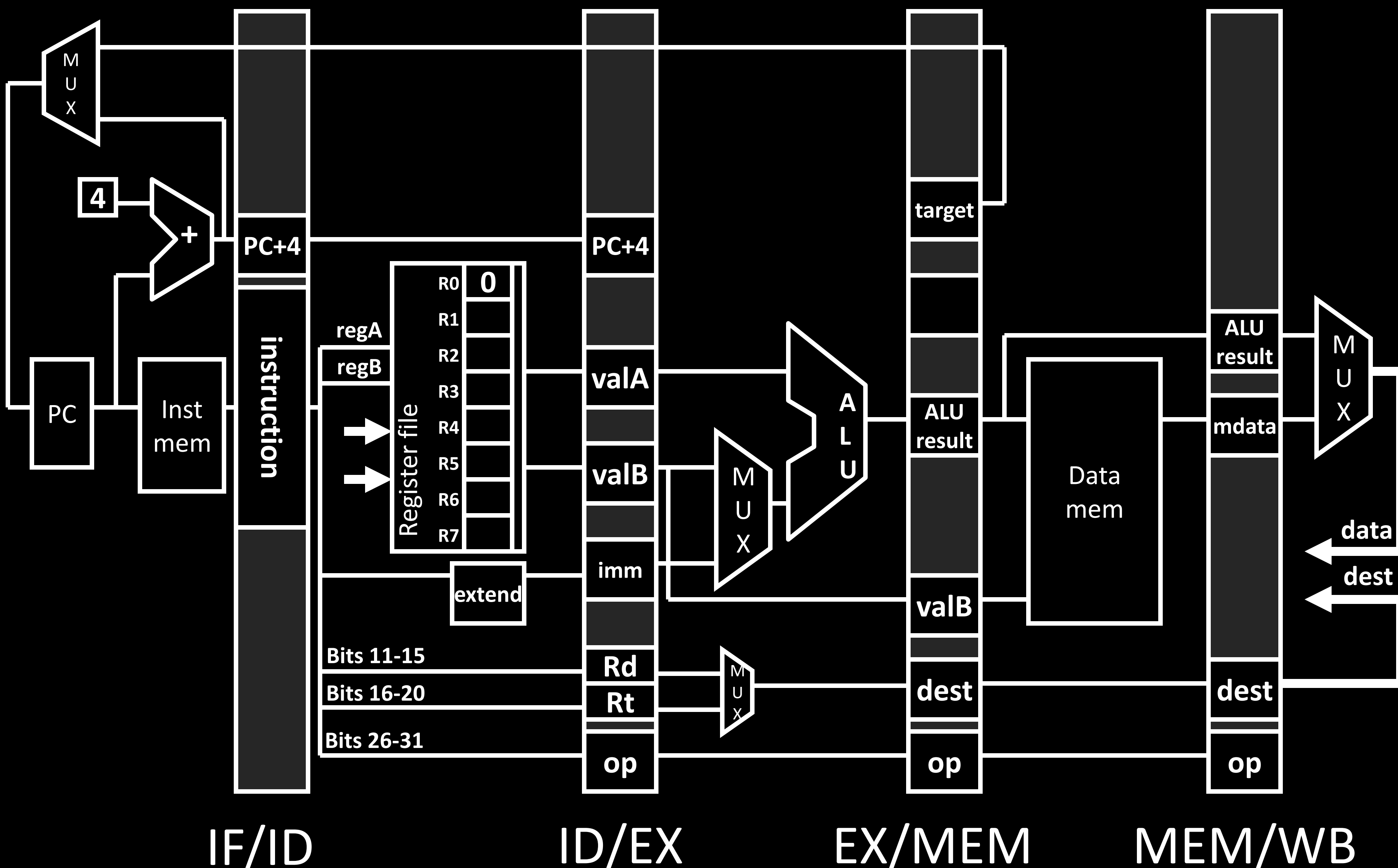
Clock cycle

	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:

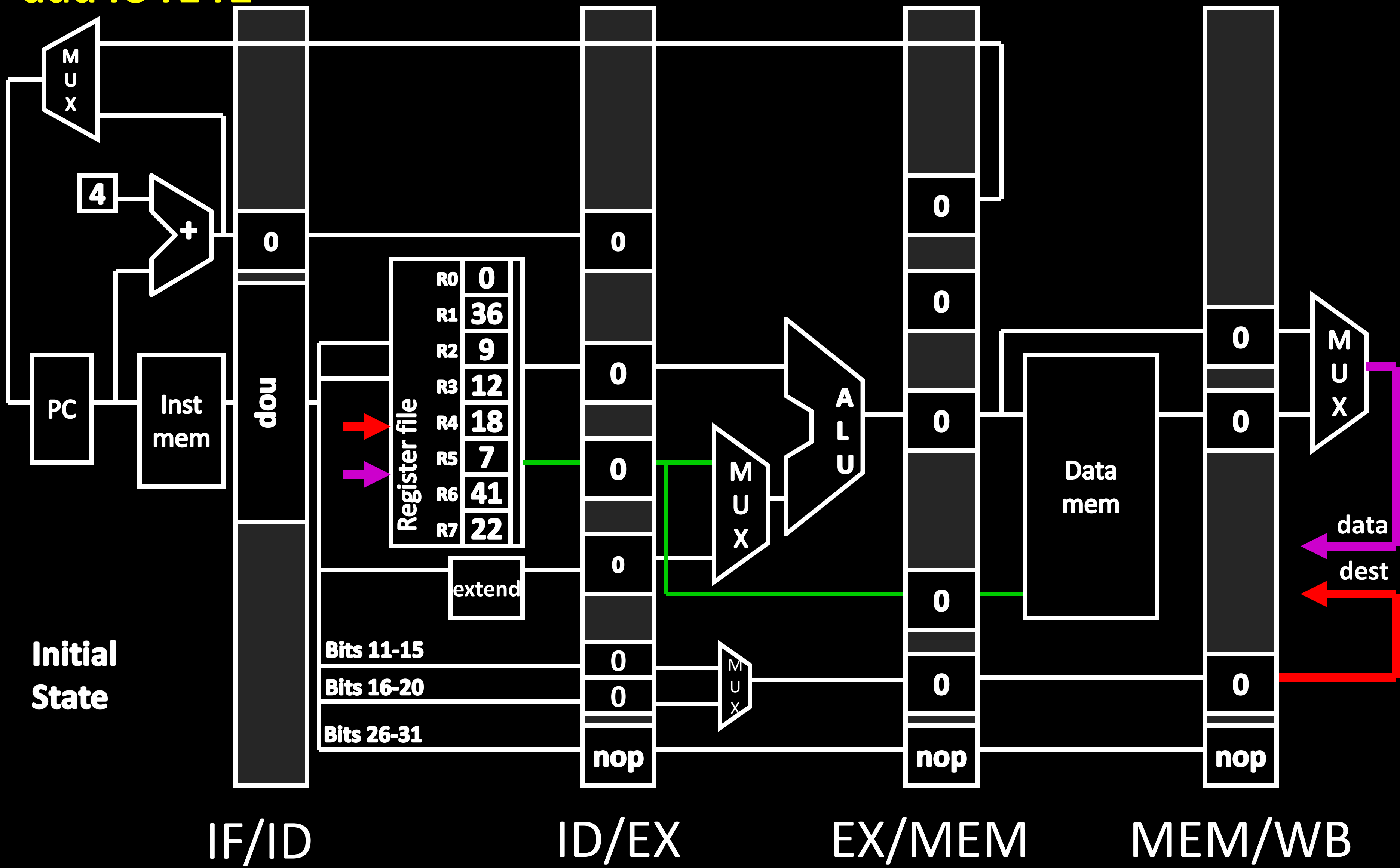
Throughput:

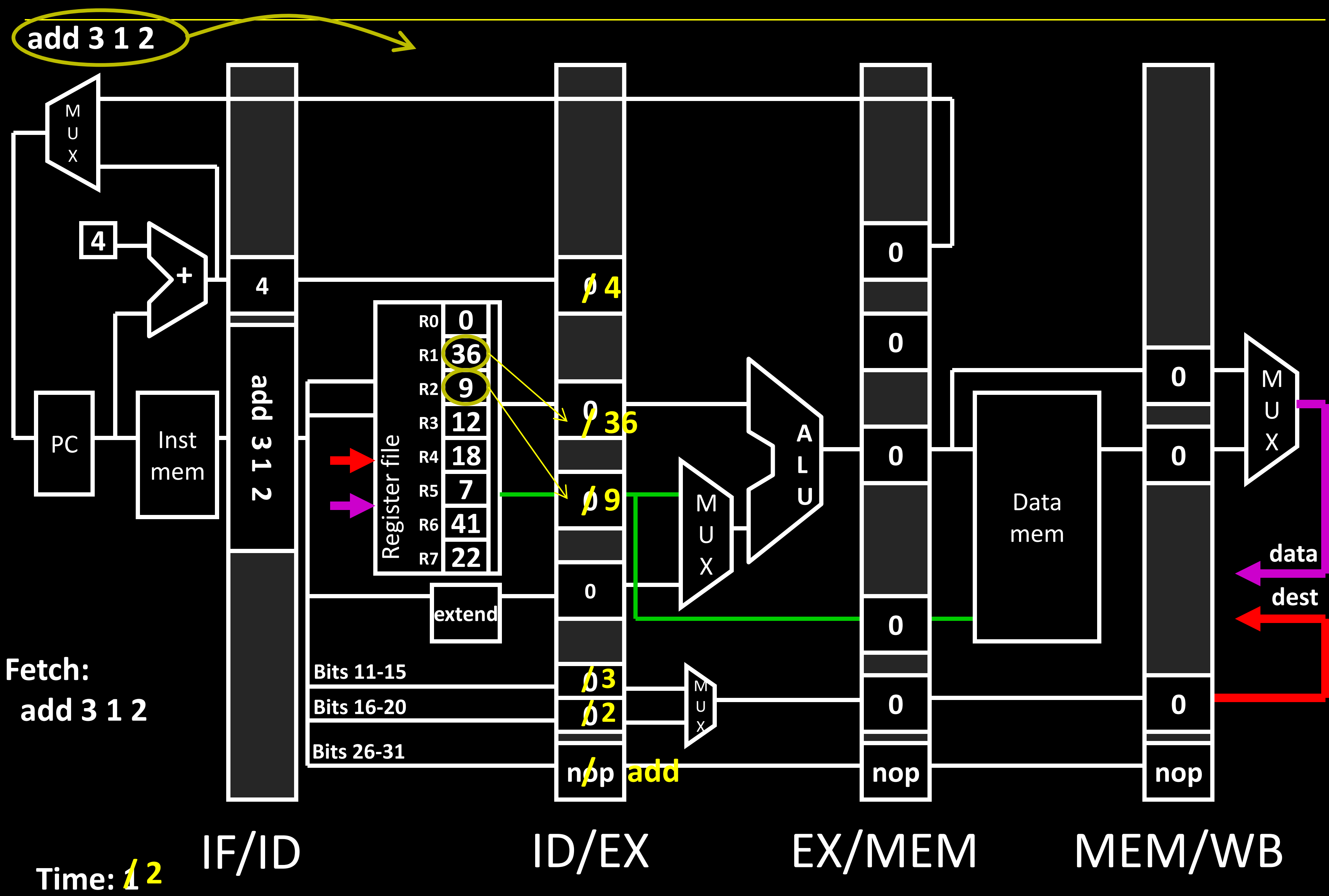
Concurrency:

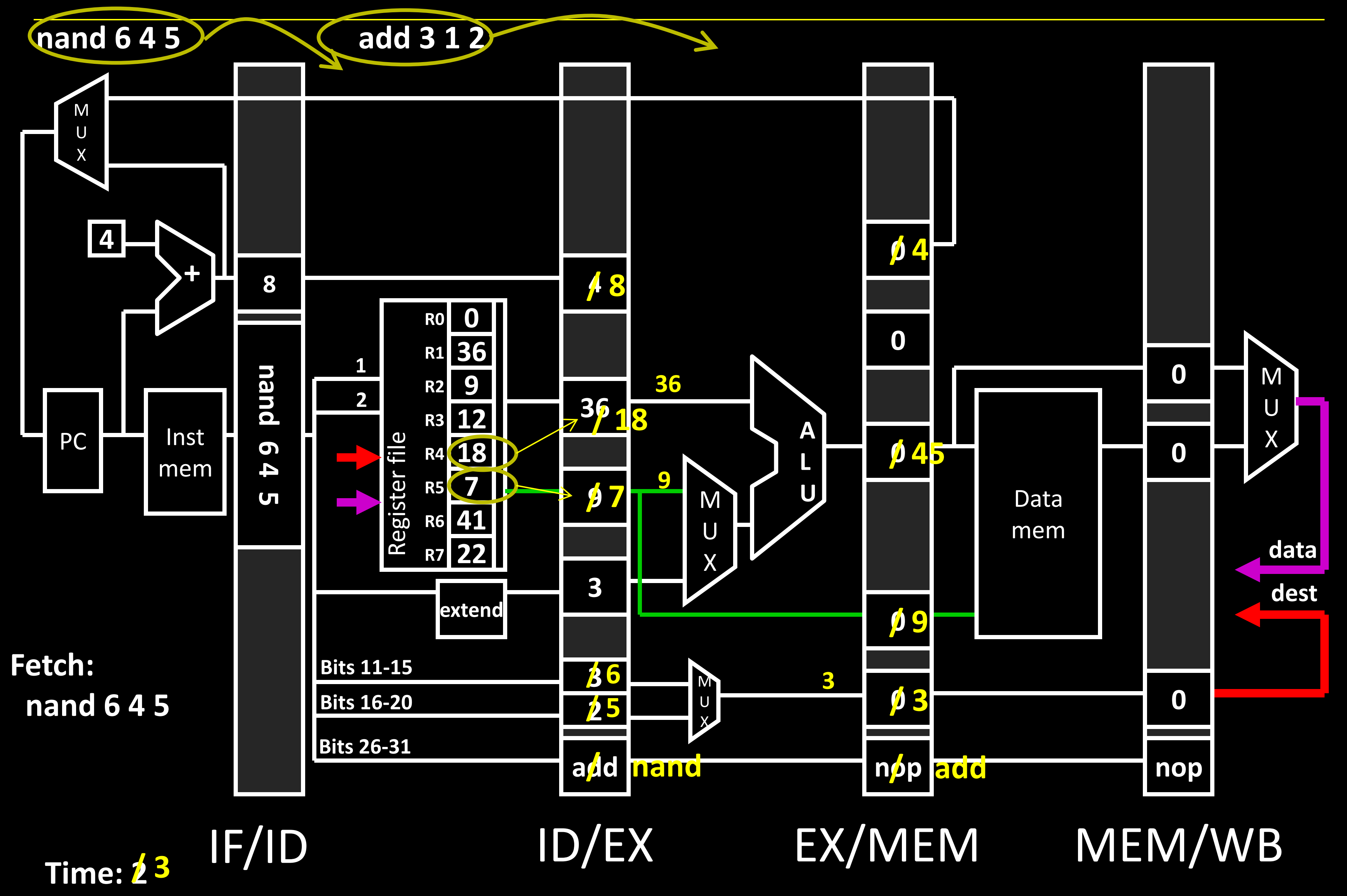

MUX
4
+
PC
Inst mem
PC+4
instruction
regA
regB
Register file
R0
0
R1
R2
R3
R4
R5
R6
R7
extend
Bits 11-15
Bits 16-20
Bits 26-31
PC+4
valA
valB
imm
Rd
Rt
op
MUX
ALU
MUX
target
ALU result
valB
dest
op
Data mem
ALU result
mdata
dest
op
MUX
data
dest
IF/ID
ID/EX
EX/MEM
MEM/WB

At time 1,
Fetch

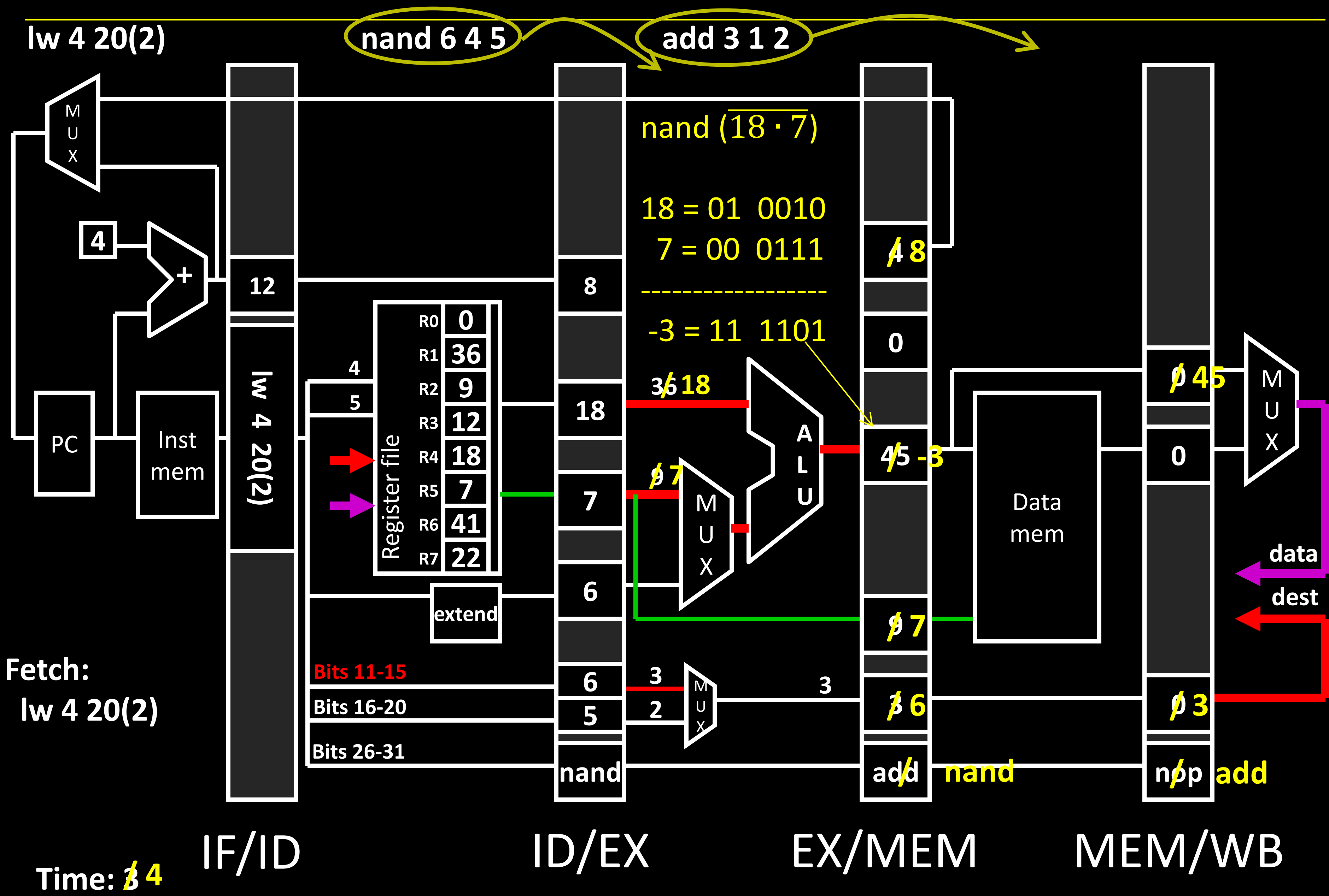
add r3 r1 r2




add 3 1 2
MUX
4
+
PC
Inst mem
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
~~0~~ 4
~~0~~ 36
~~0~~ 9
0
~~0~~ 3
~~0~~ 2
~~nop~~ add
MUX
ALU
MUX
0
0
0
0
0
nop
Data mem
0
0
0
nop
MUX
data
dest
IF/ID
ID/EX
EX/MEM
MEM/WB
Fetch:
add 3 1 2
Time: ~~1~~ 2



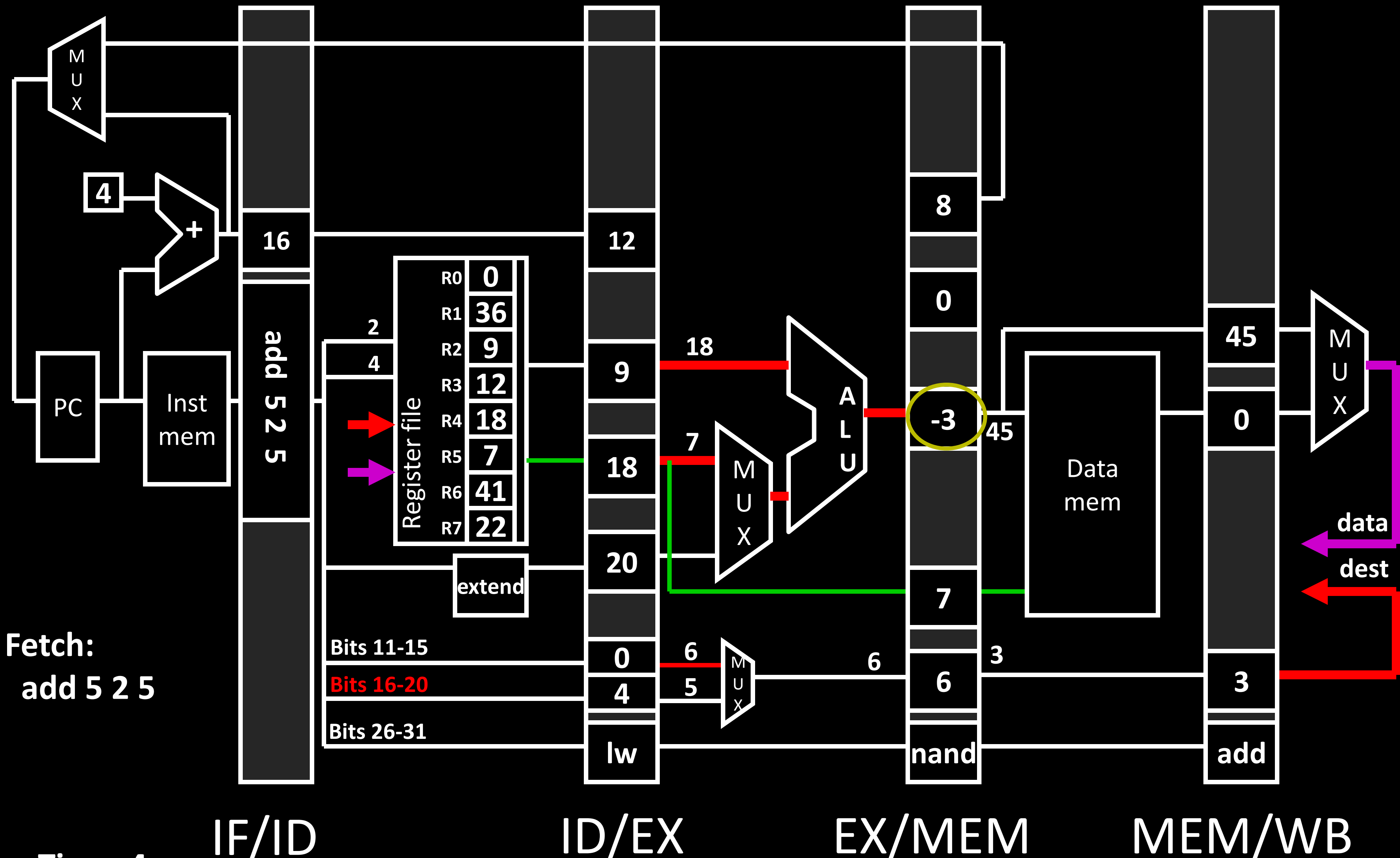
nand 6 4 5
add 3 1 2
MUX
4
+
PC
Inst mem
8
nand 6 4 5
Register file
R0 0
R1 36
R2 9
R3 12
R4 18
R5 7
R6 41
R7 22
1
2
extend
Bits 11-15
Bits 16-20
Bits 26-31
4 8
36 18
9 7
3
3 6
2 5
add nand
36
9
MUX
ALU
3
0 4
0
0 45
0 9
0 3
nop add
Data mem
0
0
0
nop
MUX
data
dest
Fetch:
nand 6 4 5
Time: 2 3
IF/ID
ID/EX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



lw 4 20(2)
nand 6 4 5
add 3 1 2
MUX
4
+
PC
Inst mem
12
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
nand $(\overline{18 \cdot 7})$
18 = 01 0010
7 = 00 0111

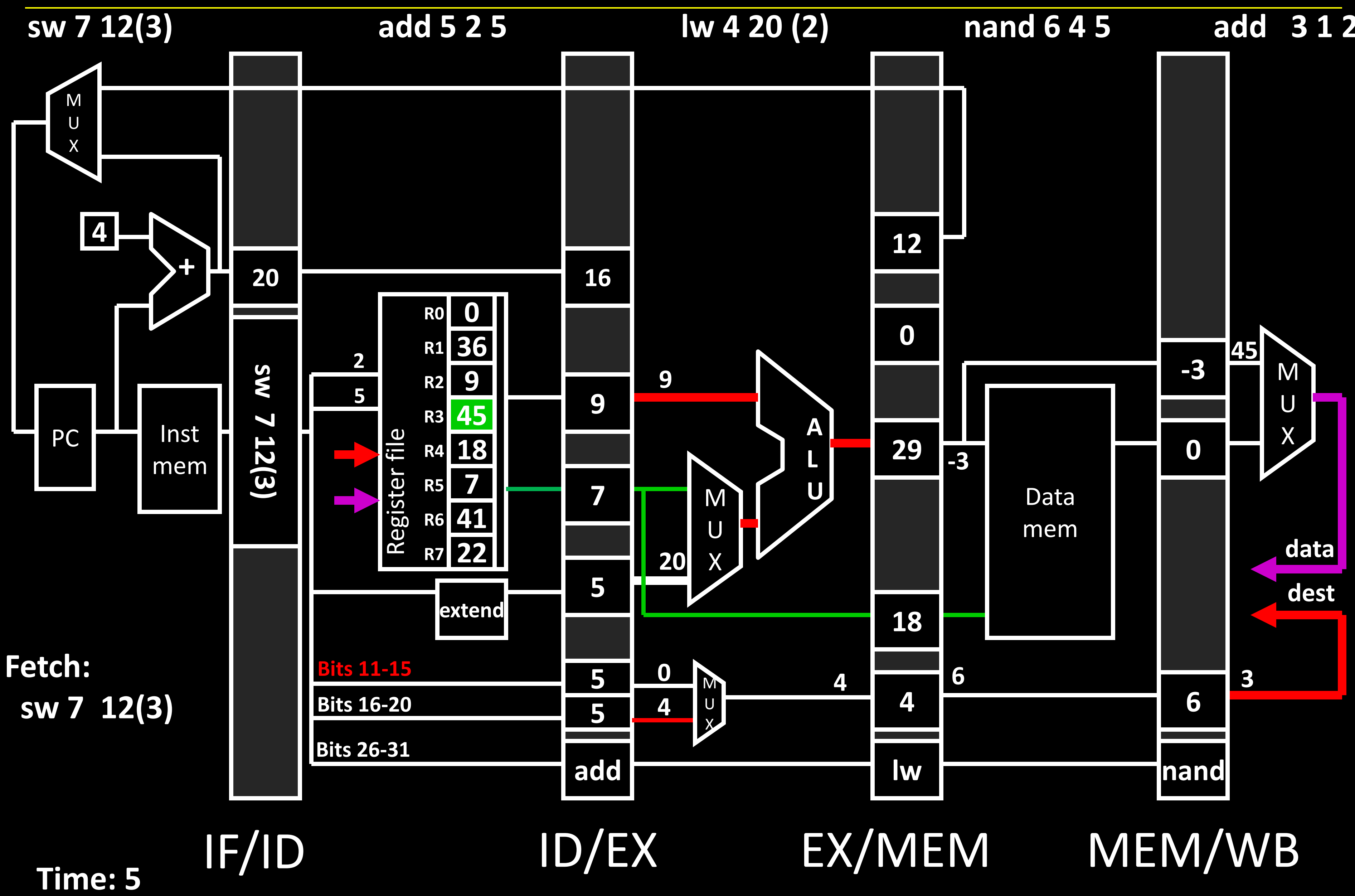
-3 = 11 1101
~~36~~ 18
~~9~~ 7
MUX
ALU
3
2
MUX
3
~~4~~ 8
0
~~45~~ -3
~~9~~ 7
~~3~~ 6
~~add~~ nand
Data mem
~~0~~ 45
0
~~0~~ 3
~~nop~~ add
MUX
data
dest
Fetch:
lw 4 20(2)
Time: ~~3~~ 4
IF/ID
ID/EX
EX/MEM
MEM/WB

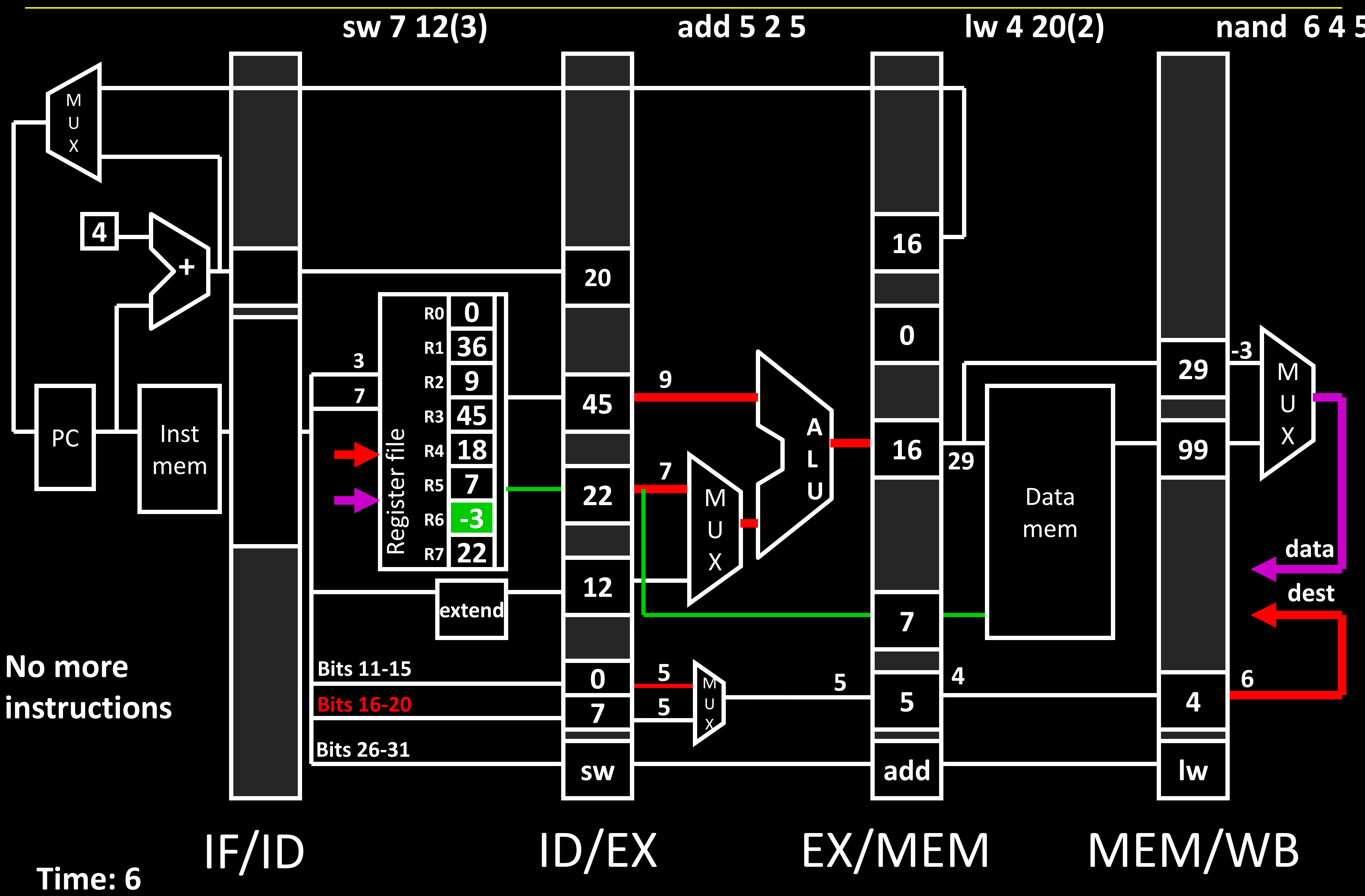
add 5 2 5 | lw 4 20(2) | nand 6 4 5 | add 3 1 2



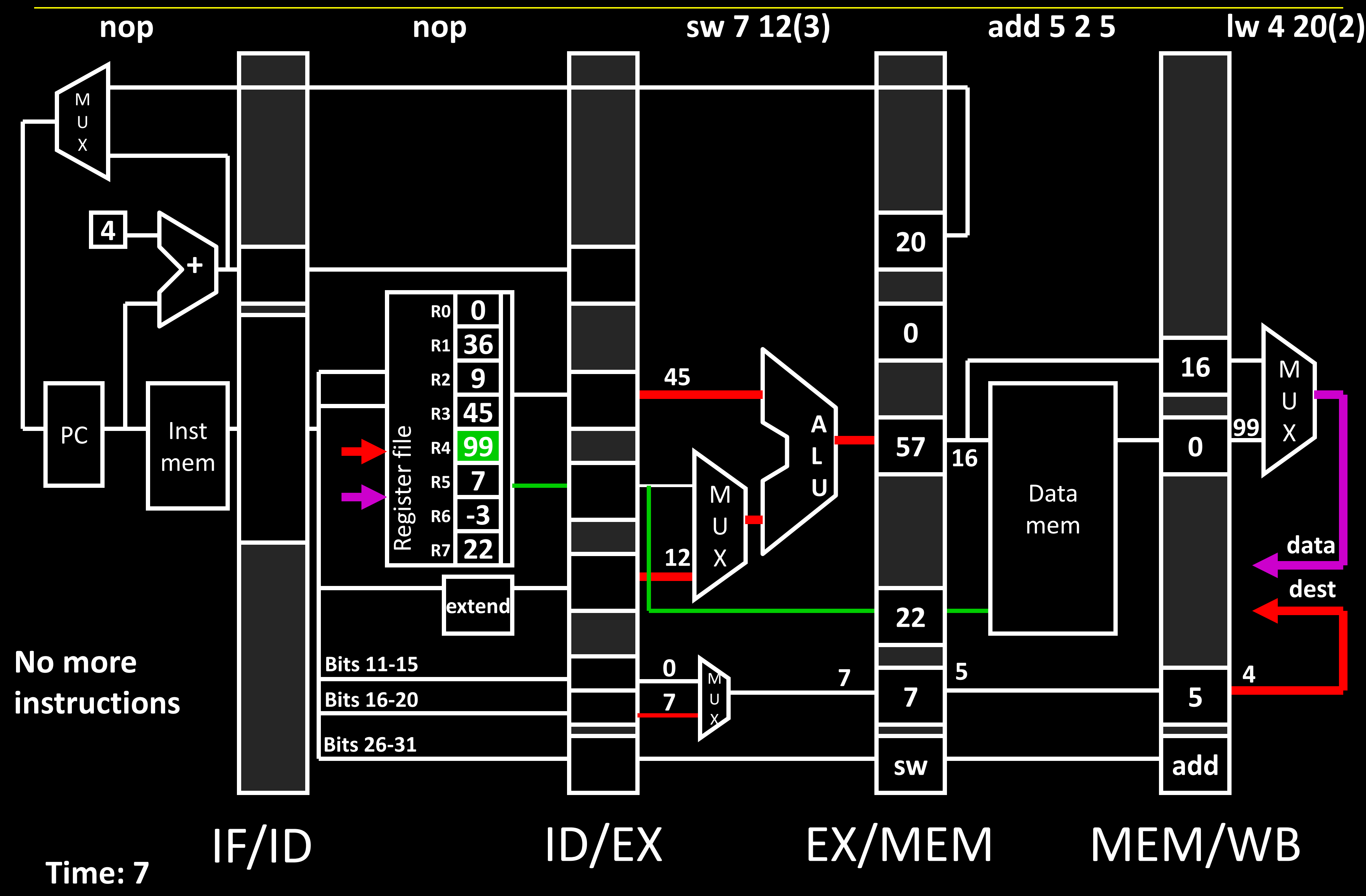
Fetch:
add 5 2 5

Time: 4

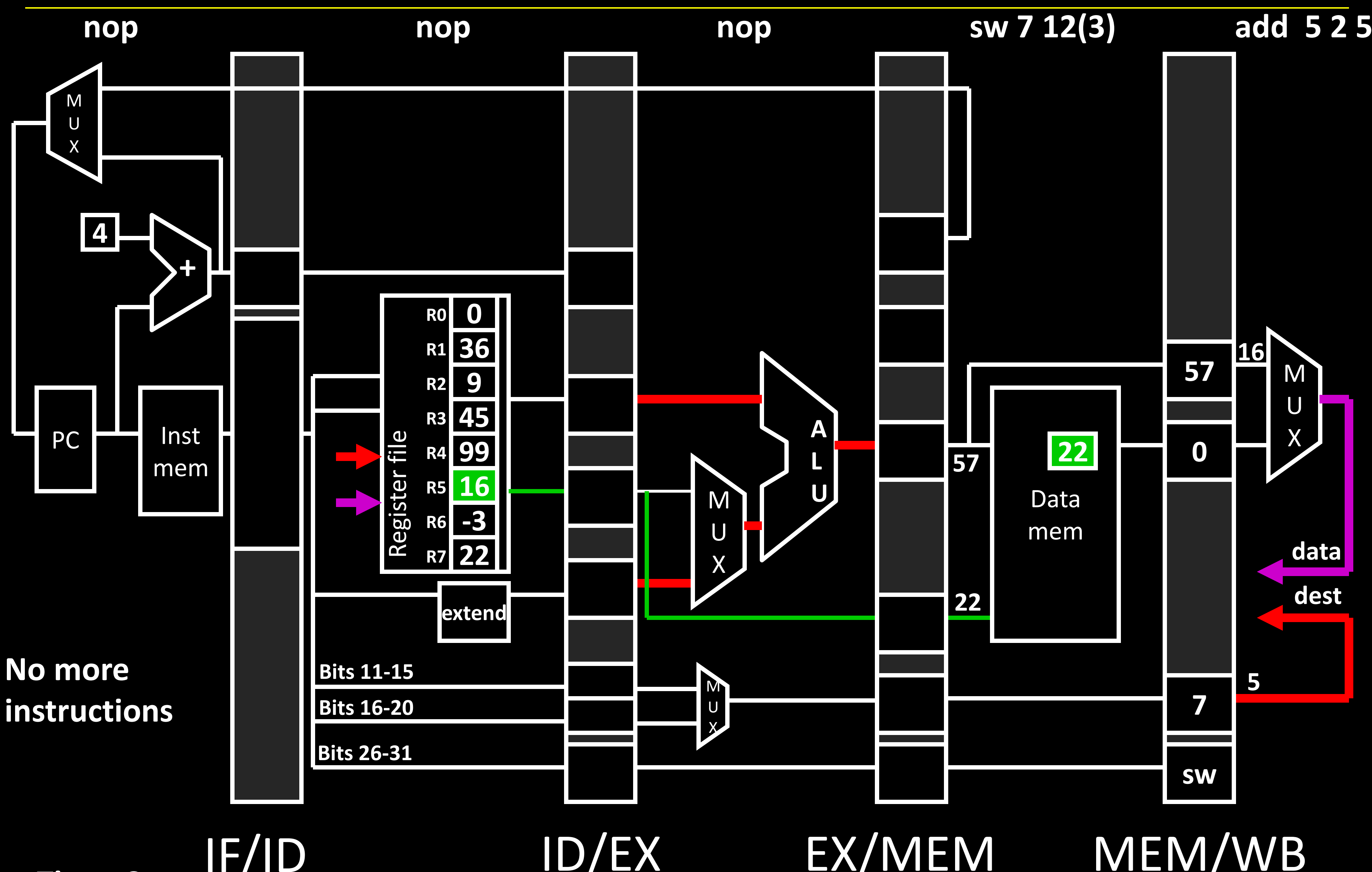

sw 7 12(3)
add 5 2 5
lw 4 20 (2)
nand 6 4 5
add 3 1 2
M U X
4
+
PC
Inst mem
20
sw 7 12(3)
Register file
R0 0
R1 36
R2 9
R3 45
R4 18
R5 7
R6 41
R7 22
2
5
extend
Bits 11-15
Bits 16-20
Bits 26-31
16
9
7
5
5
5
add
9
20
M U X
A L U
0
4
M U X
4
12
0
29
18
4
lw
-3
Data mem
6
-3
0
6
nand
45
M U X
3
data
dest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



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



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

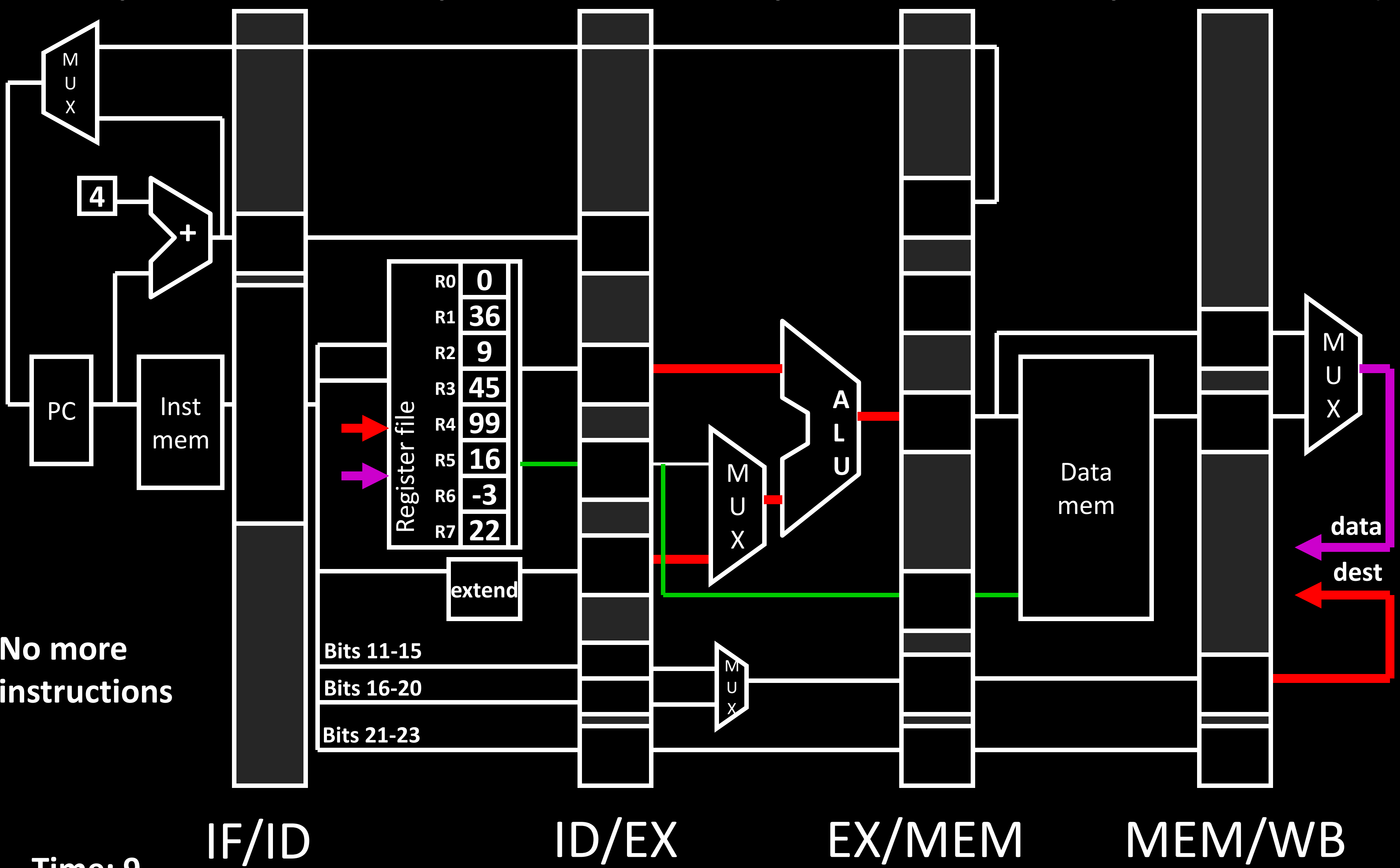


No more instructions

Time: 8

Slides thanks to Sally McKee

nop nop nop nop sw 7 12(3)



No more instructions

Time: 9

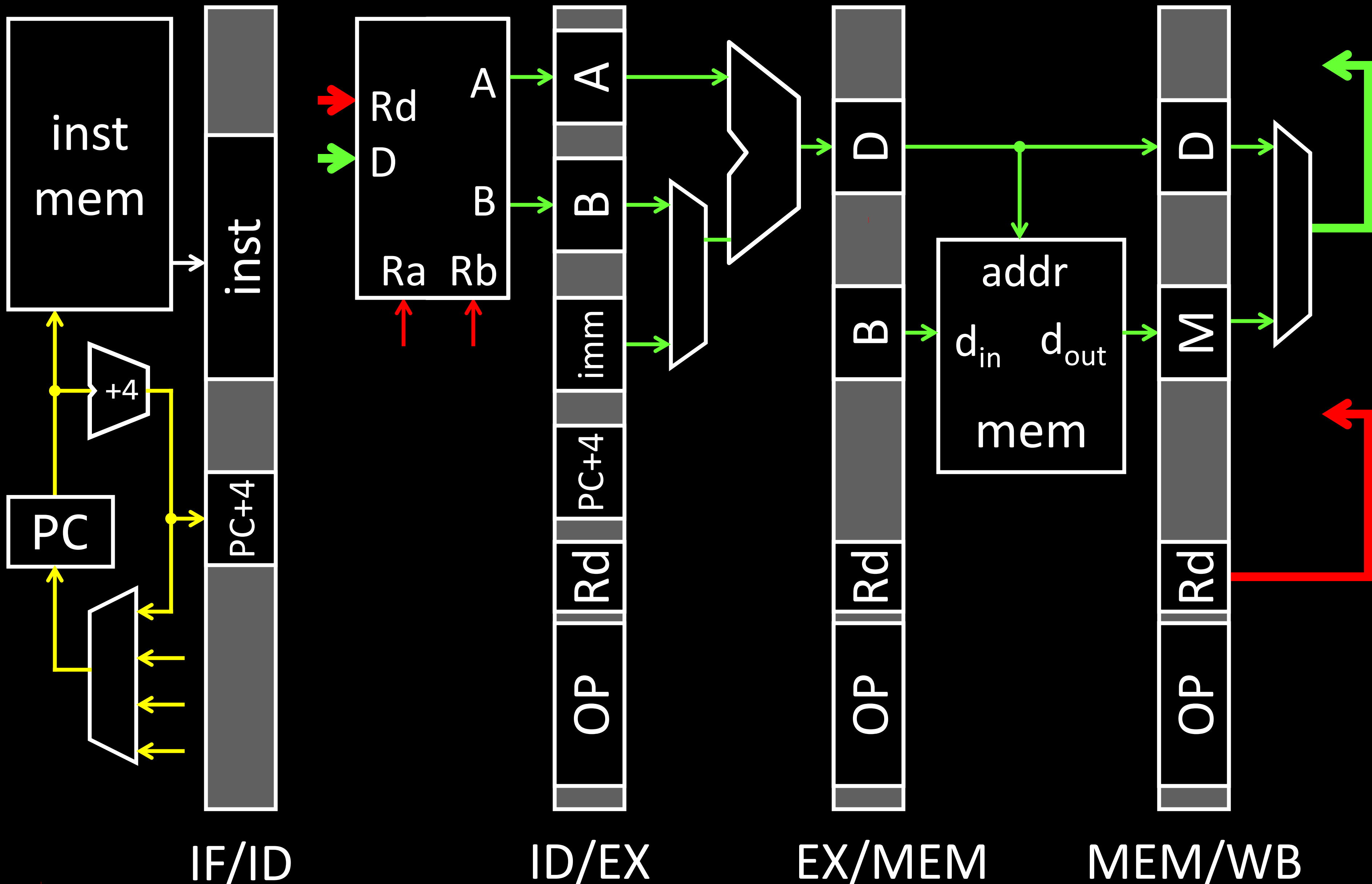
Pipelining Recap

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
inst
PC+4
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