12. Basic Processor Design – Introduction to Pipelining

EECS 370 – Introduction to Computer Organization - Winter 2016
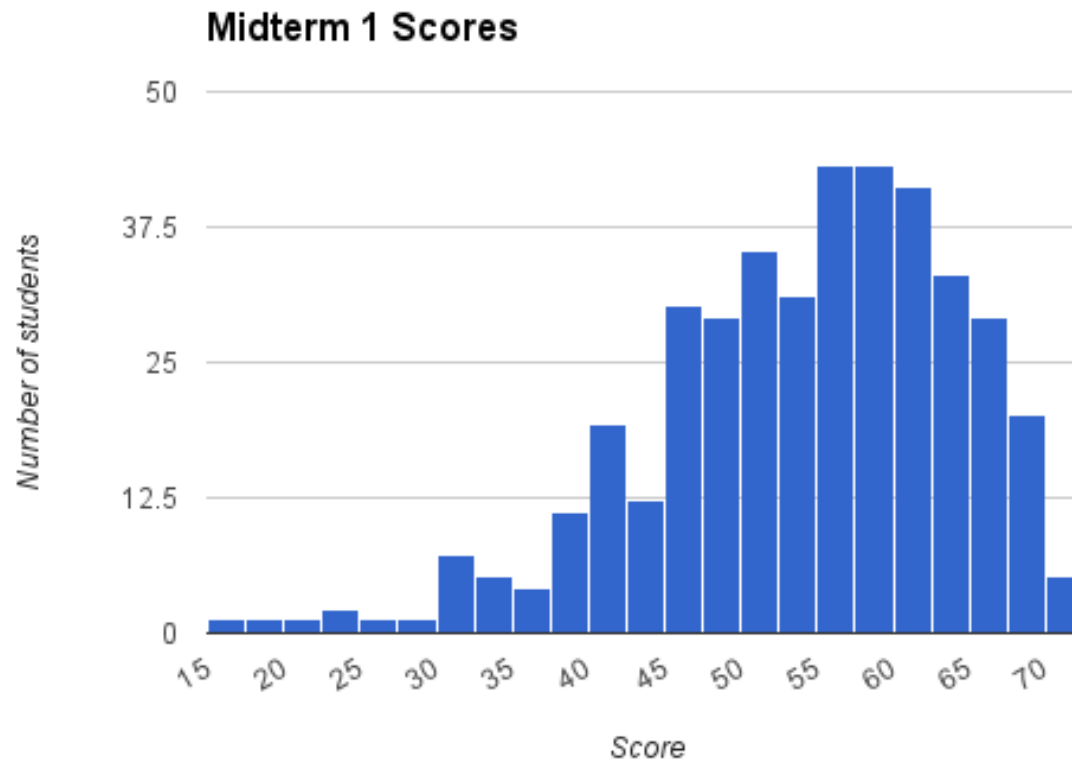
Profs. Valeria Bertacco & Reetu Das

EECS Department
University of Michigan in Ann Arbor, USA

Exam Statistics

Midterm 1 Scores

- ❑ Average: 54 / 71
- ❑ Std. Dev: 10

Exam Details

❑ Average score per question

- Q1 – 5.1/6   Q2 – 6.8/10
- Q3 – 7.2/9   Q4 – 5.3/10
- Q5 – 5.6/9   Q6 – 5.8/6
- Q7 – 9.6/11   Q8 – 7.5/10
- Q2 – ARM Assembly
- Q4 – Caller/Callee
- Q5 - Location

❑ Exam Re-grade Requests

- Written form available on website and Piazza
- Need detailed description of re-grade request, sign form to acknowledge everything will be re-graded
- Turn in before/after class or Prof. Office Hours
- Must receive by 3/8

x86 vs. MIPS vs. ARM vs. LC2K

MOBILE SALES SKYROCKET ... / ... LEAVING INTEL CHIPS BEHIND

Source: https://semiwiki.com
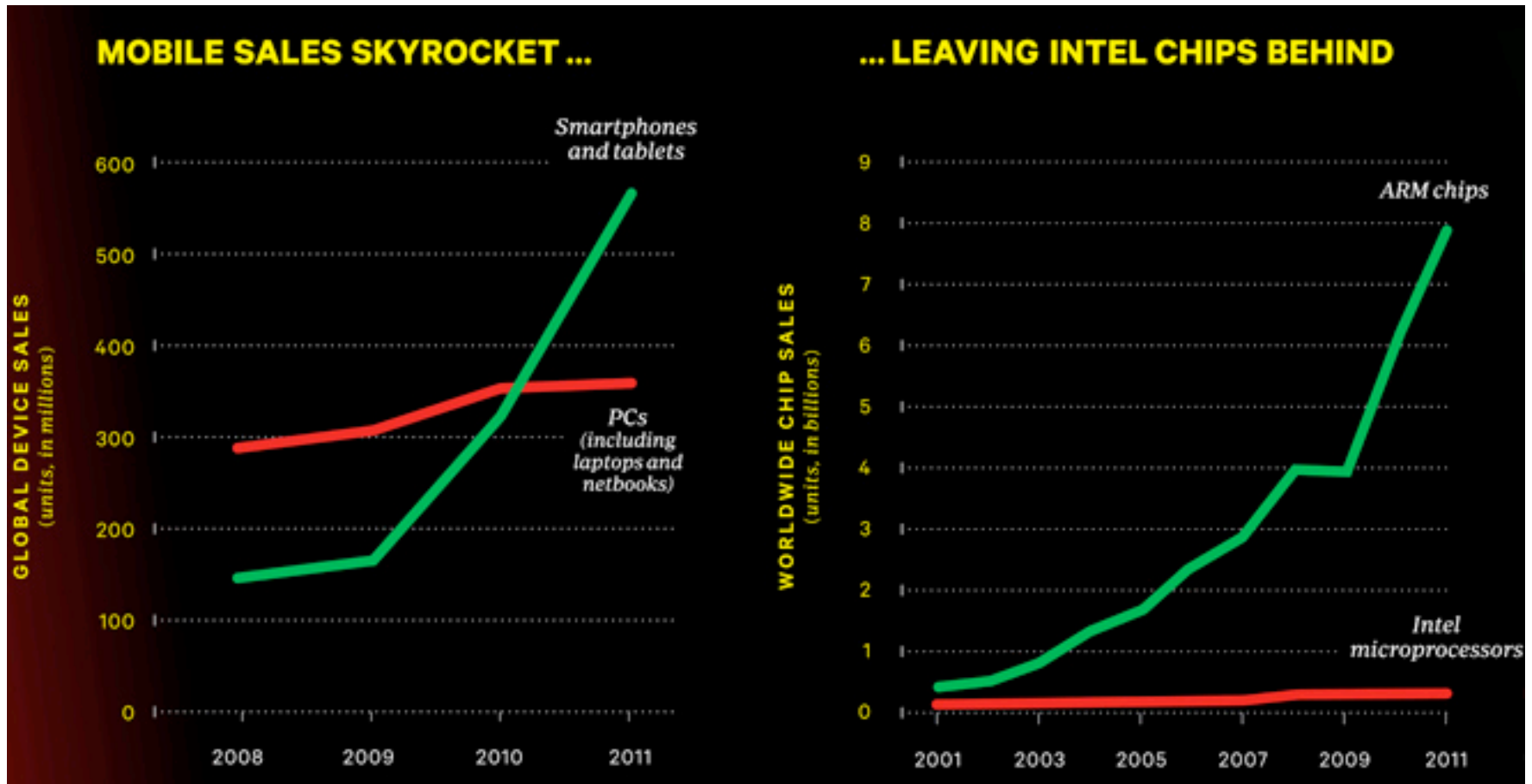
x86 vs. MIPS vs. ARM vs. LC2K

❏ x86:
- Complex instruction set (harder to reason about, difficult for projects)

❏ MIPS
- No major commercial examples
- Best book (most widely used) on computer architecture written with it

❏ ARM
- Gives exposer to real world ISA
- Added skill line for value on resume

❏ LC2K
- Limited ISA (8 instructions) makes writing project much easier
- Simpler syntax to enable parsing by project (no #, r1, etc.)

Review: Single-Cycle LC2Kx Datapath
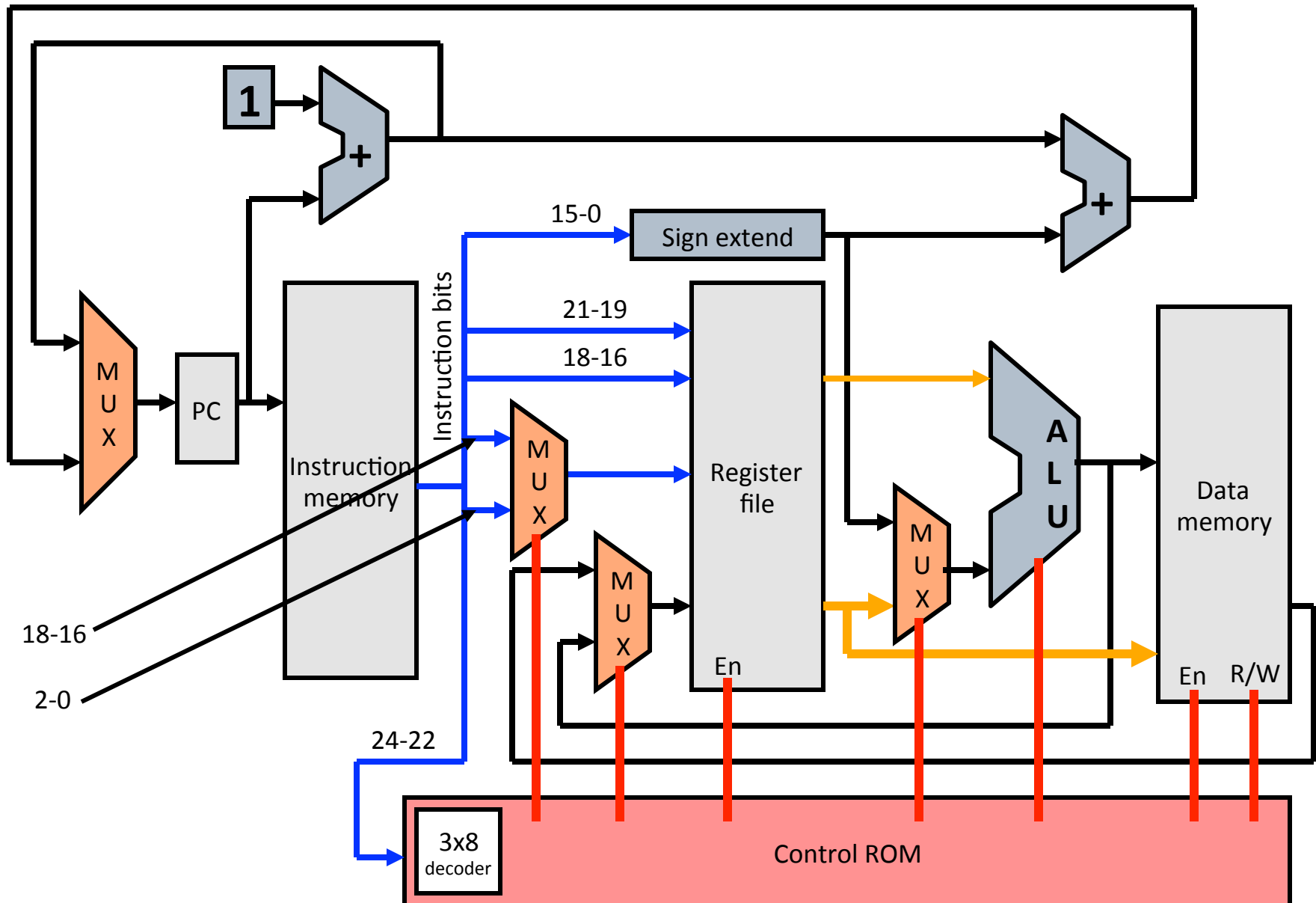
Review: Multi-Cycle LC2Kx Datapath
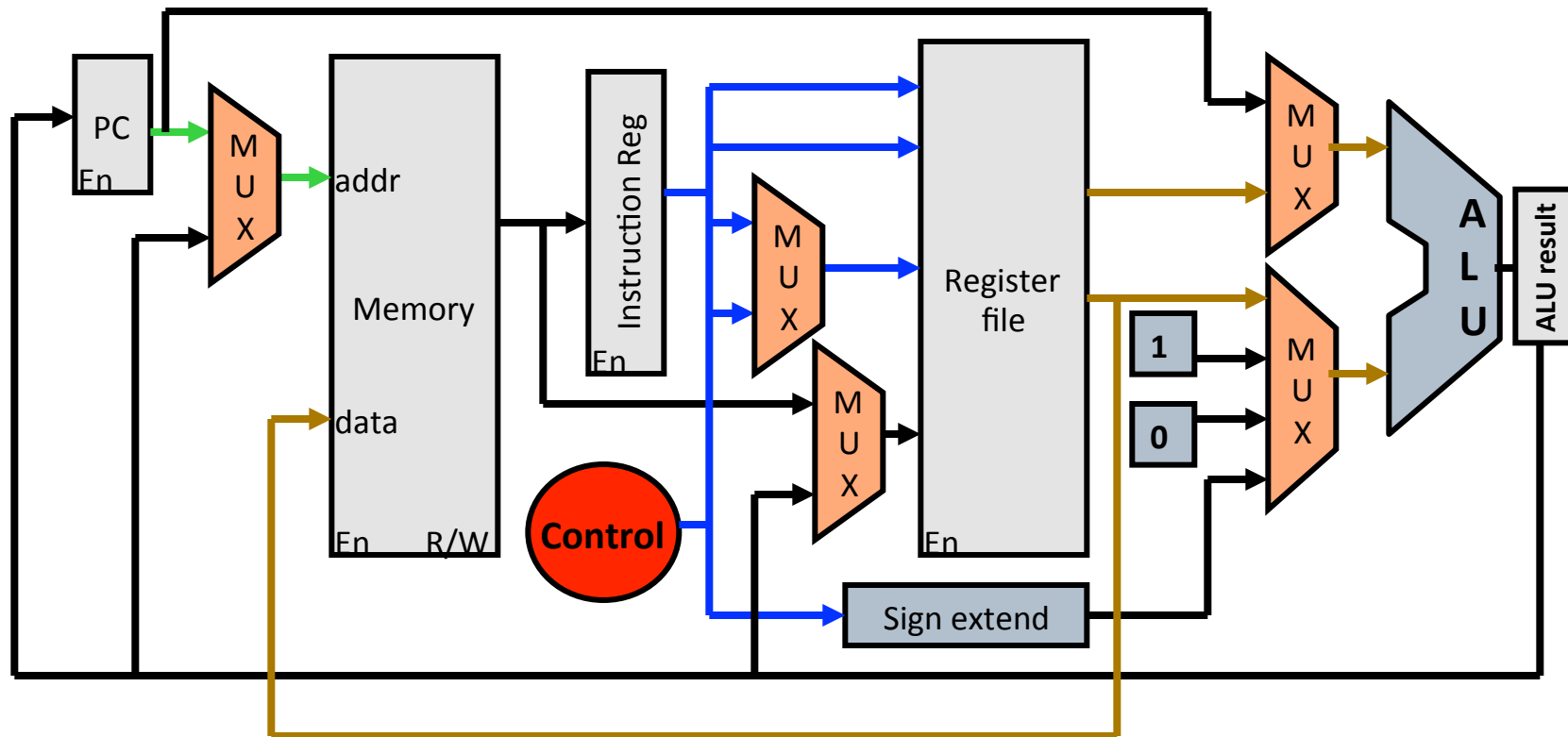
Review: Single and Multicycle Performance

1 ns – Register File read/write time

2 ns – ALU/adder

2 ns – memory access

0 ns – MUX, PC access, sign extend, ROM

1. Assuming the above delays, what is the best cycle time that the LC2k multicycle datapath could achieve?

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

Review: Single and Multicycle Performance

1 ns – Register read/write time

2 ns – ALU/adder

2 ns – memory access

0 ns – MUX, PC access, sign extend, ROM

1. Assuming the above delays, what is the best cycle time that the LC2k multicycle datapath could achieve?

MC: MAX(2, 1, 2, 2, 1) = 2ns

SC: 2 + 1 + 2 + 2 + 1 = 8 ns

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

SC: 100 cycles * 8 ns = 800 ns

MC: (25*5 + 10*4 + 45*4 + 20*4)cycles * 2ns = 850 ns

Performance Metrics

1. **Response time**: when is my job done (time)?
   - When will my books arrive from amazon.com?
   - How long will this program/instruction take?

2. **Throughput**: how much work can get done within a specified time (work/time)?
   - How many books will amazon.com sell this week?
   - How many programs/instructions complete per hour?
   - Improved relatively easily by using multiprocessors.

Performance Metrics – Execution Time

❑ Response time for a program is its **execution time**

Execution time (for an application):

= total instructions executed x CPI x clock period

- **Called the "Iron Law" of performance**

❑ CPI = avg number of clock cycles per instruction *for an application*

❑ For multi-cycle processor implementations we need:

- Cycles necessary for each type of instruction

- Mix of instructions executed in the application (dynamic instruction execution profile)

How far have we come?

❑ Single-cycle processor implementations

- CPI = ?

- clock period = ?

❑ Multi-cycle processor implementations

- CPI = ?

- clock period = ?

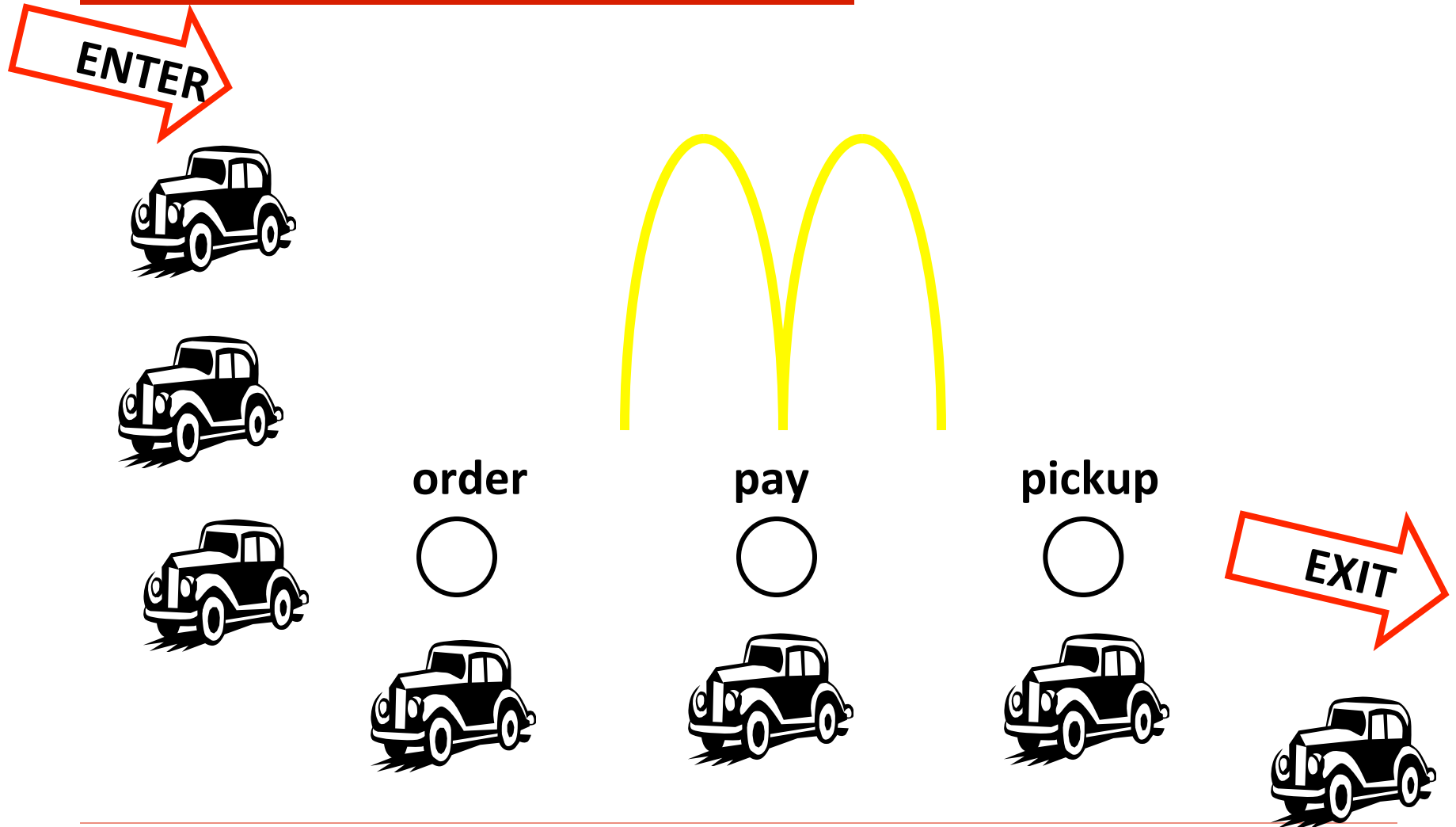❑ Next step: improve CPI without impacting clock period

- The easiest thing to do is to work on multiple instructions at the same time.
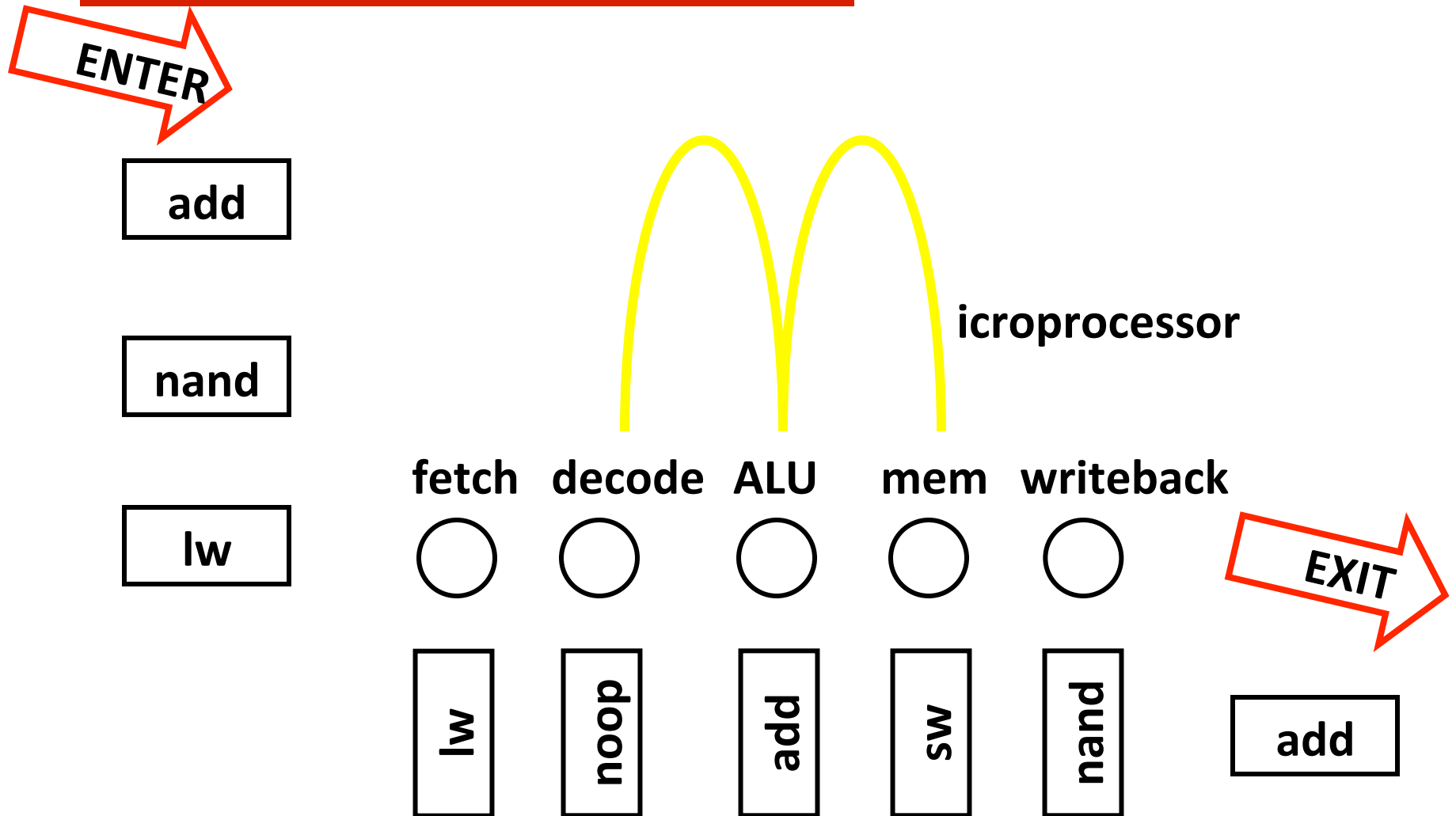
Pipelining

❑ Want to execute an instruction?

- Build a processor (multi-cycle)

- Find instructions

- Line up instructions (1, 2, 3, …)

- Overlap execution

  - Cycle #1:   Fetch 1

  - Cycle #2:   Decode 1   Fetch 2

  - Cycle #3:   ALU 1   Decode 2   Fetch 3

  - . . . . . .

- This is called pipelining instruction execution.

- Used extensively for the first time on IBM 360 (1960s).

- CPI approaches 1.

Pipelining

ENTER

order   pay   pickup

EXIT

Pipelining

ENTER

add

nand

icroprocessor

lw

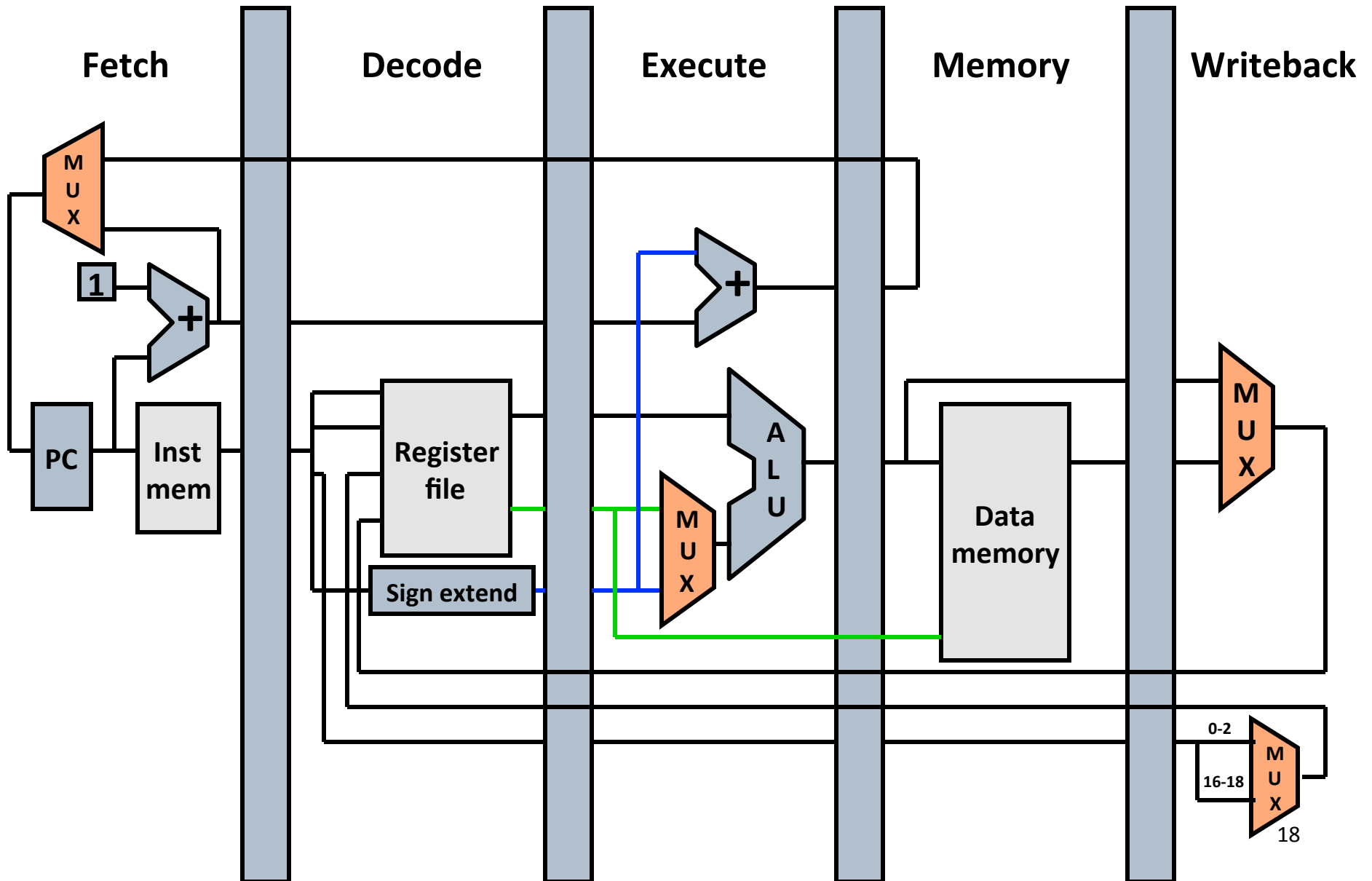| fetch	decode	ALU	mem	writeback
○	○	○	○	○
lw	noop	add	sw	nand

EXIT

add

Pipelining Today

❑ Execute as many instructions at the same time as possible.

- Pipelining: 12-20+ cycles
- Multiple pipelines

❑ Pentium:

- 2 pipelines, 5 cycles each   (10 instructions "in flight")

❑ Pentium Pro/II/III

- 3 pipelines (kinda), 12 cycles each (kinda)
- Instructions can execute out of their original program order

❑ Pentium IV

- 4 pipelines, 20 cycles deep
- Prescott: 4 pipelines, 31 cycles deep (could be clocked up to 8 GHz with special cooling)

❑ Core i7 (Nehalem)

- 4 pipelines, 16 cycles deep

Pipelined implementation of LC2Kx

❑ Break the execution of the instruction into cycles.

- Similar to the multi-cycle datapath

❑ Design a separate datapath stage for the execution performed during each cycle.

- Build pipeline registers to communicate between the stages.

Our new pipelined datapath

Fetch   Decode   Execute   Memory   Writeback

MUX

1

+

PC

Inst mem

Register file

Sign extend

MUX

ALU

MUX

+

Data memory

MUX

0-2

16-18

MUX

18

Stage 1: Fetch

❑ Design a datapath that can fetch an instruction from memory every cycle.

- Use PC to index memory to read instruction
- Increment the PC (assume no branches for now)

❑ Write everything needed to complete execution to the pipeline register (IF/ID)

- The next stage will read this pipeline register.
- Note that pipeline register must be edge-triggered

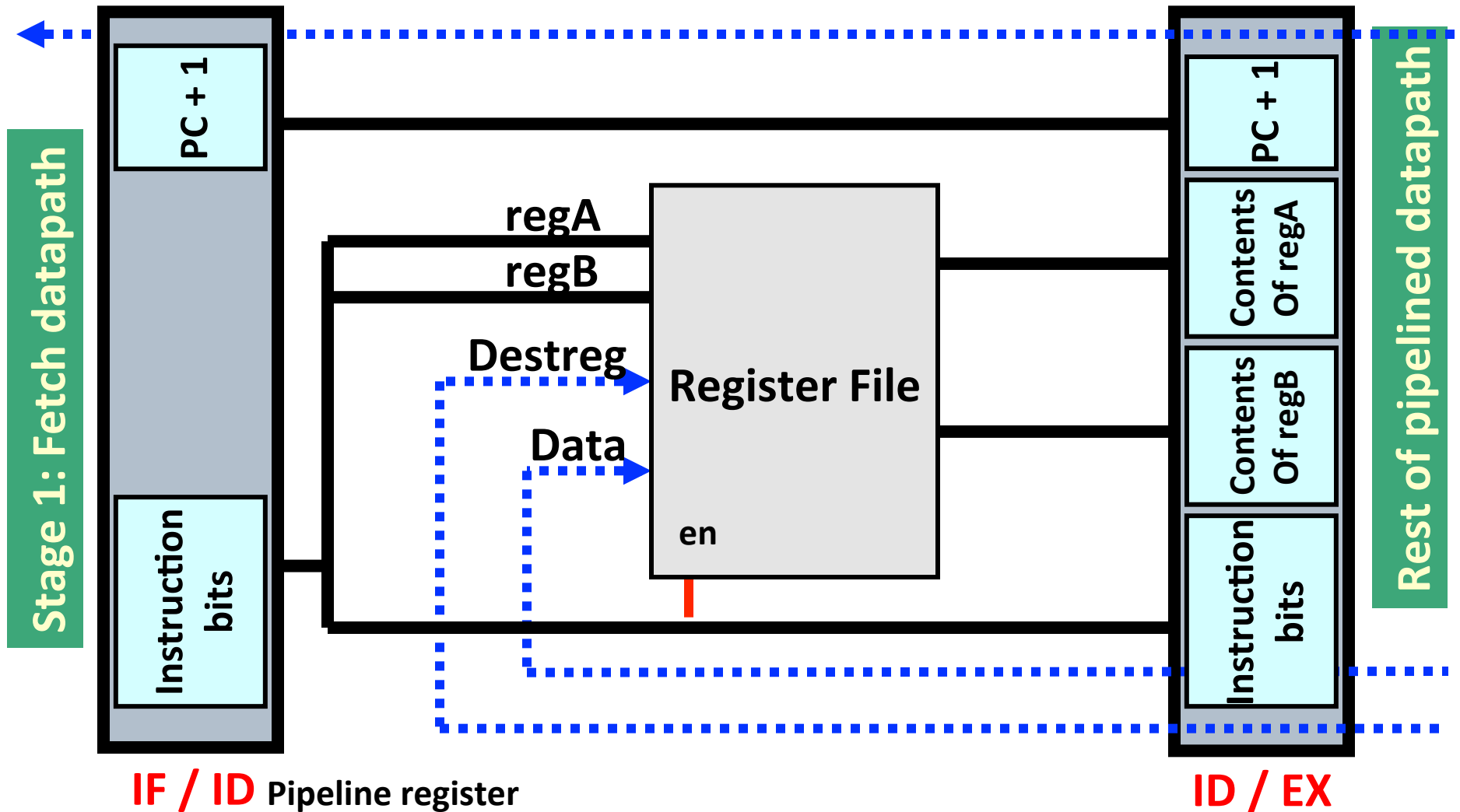# Pipeline datapath – Fetch stage

IF / ID Pipeline register
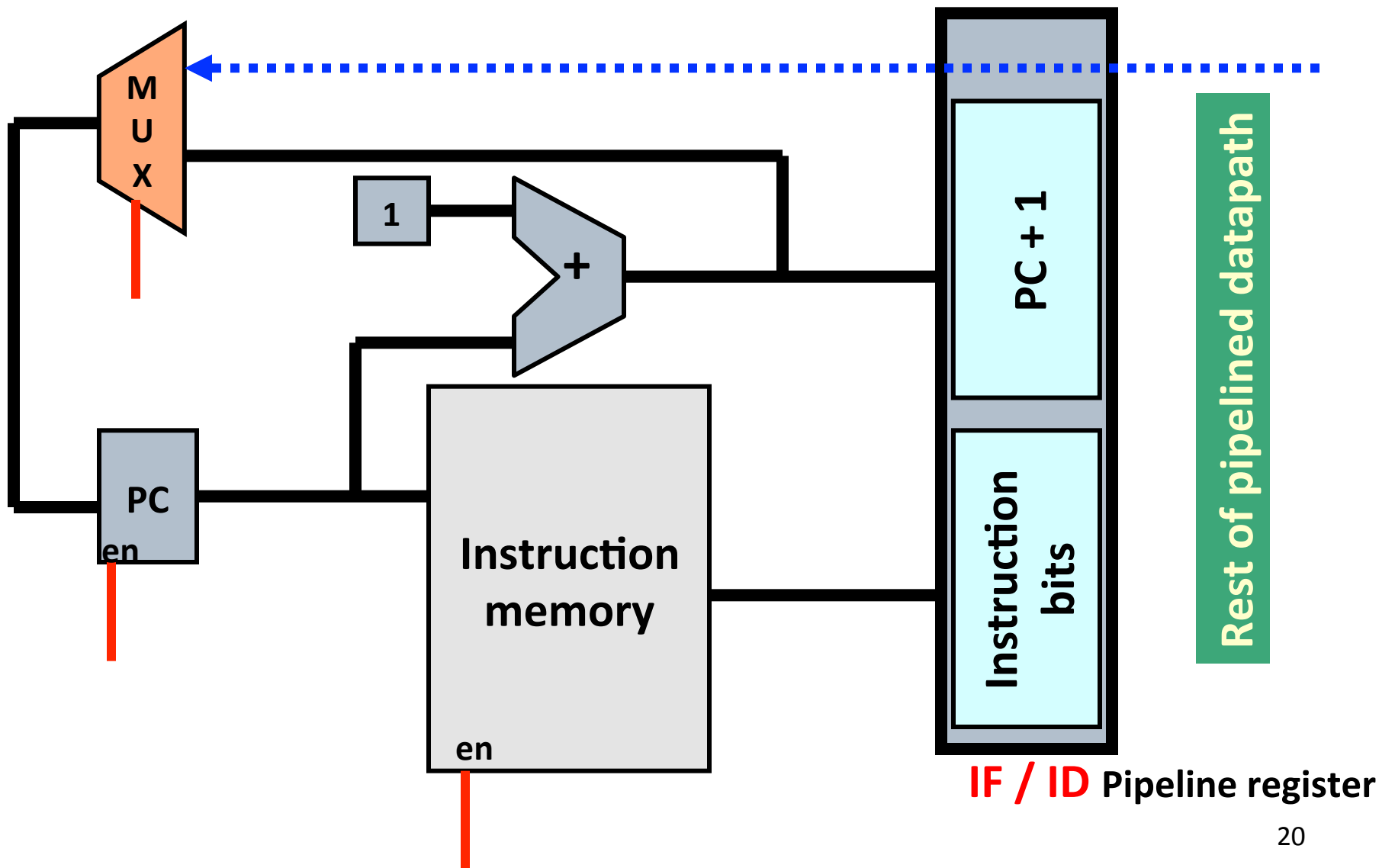
Stage 2: Decode

❑ Design a datapath that reads the IF/ID pipeline register, decodes instruction and reads register file (specified by regA and regB of instruction bits).

• Decode is easy, just pass on the opcode and let later stages figure out their own control signals for the instruction.

❑ Write everything needed to complete execution to the pipeline register (ID/EX)

• Pass on the offset field and both destination register specifiers (or simply pass on the whole instruction!).

• Including PC+1 even though decode didn't use it.

Pipeline datapath – Decode stage

Stage 1: Fetch datapath

PC + 1

regA
regB

Destreg

Data

Register File

en

Instruction bits

IF / ID Pipeline register

PC + 1

Contents Of regA

Contents Of regB

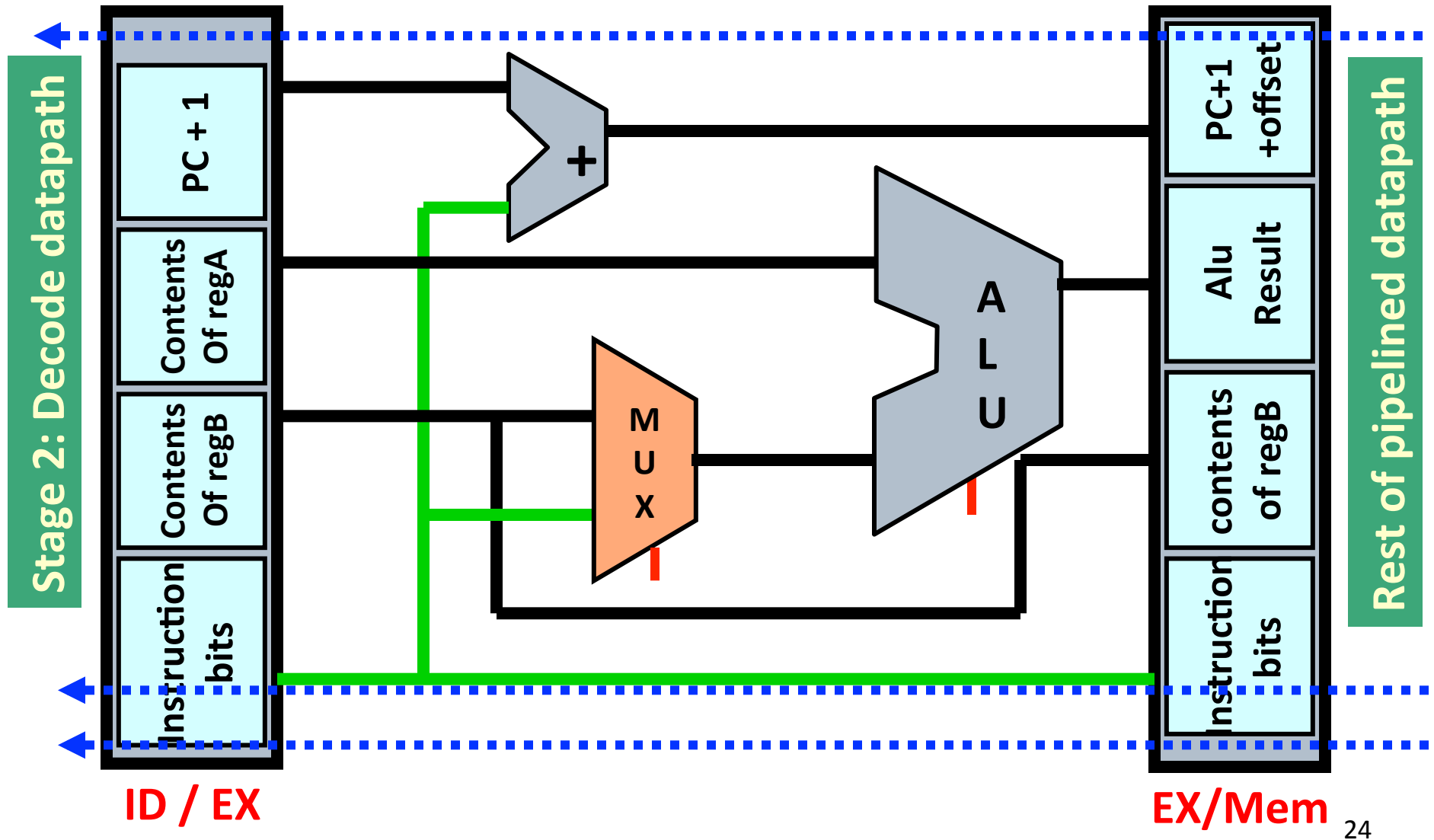Instruction bits

ID / EX

Rest of pipelined datapath

22

Stage 3: Execute

❑ Design a datapath that performs the proper ALU operation for the instruction specified and the values present in the ID/EX pipeline register.

- The inputs are the contents of regA and either the contents of regB or the offset field on the instruction.
- Also, calculate PC+1+offset in case this is a branch.

❑ Write everything needed to complete execution to the pipeline register (EX/Mem)

- ALU result, contents of regB and PC+1+offset
- Instruction bits for opcode and destReg specifiers
- Result from comparison of regA and regB contents

Pipeline datapath – Execute stage

Stage 2: Decode datapath

PC + 1

Contents Of regA

Contents Of regB

Instruction bits

+

MUX

ALU

PC+1 +offset

Alu Result

contents of regB

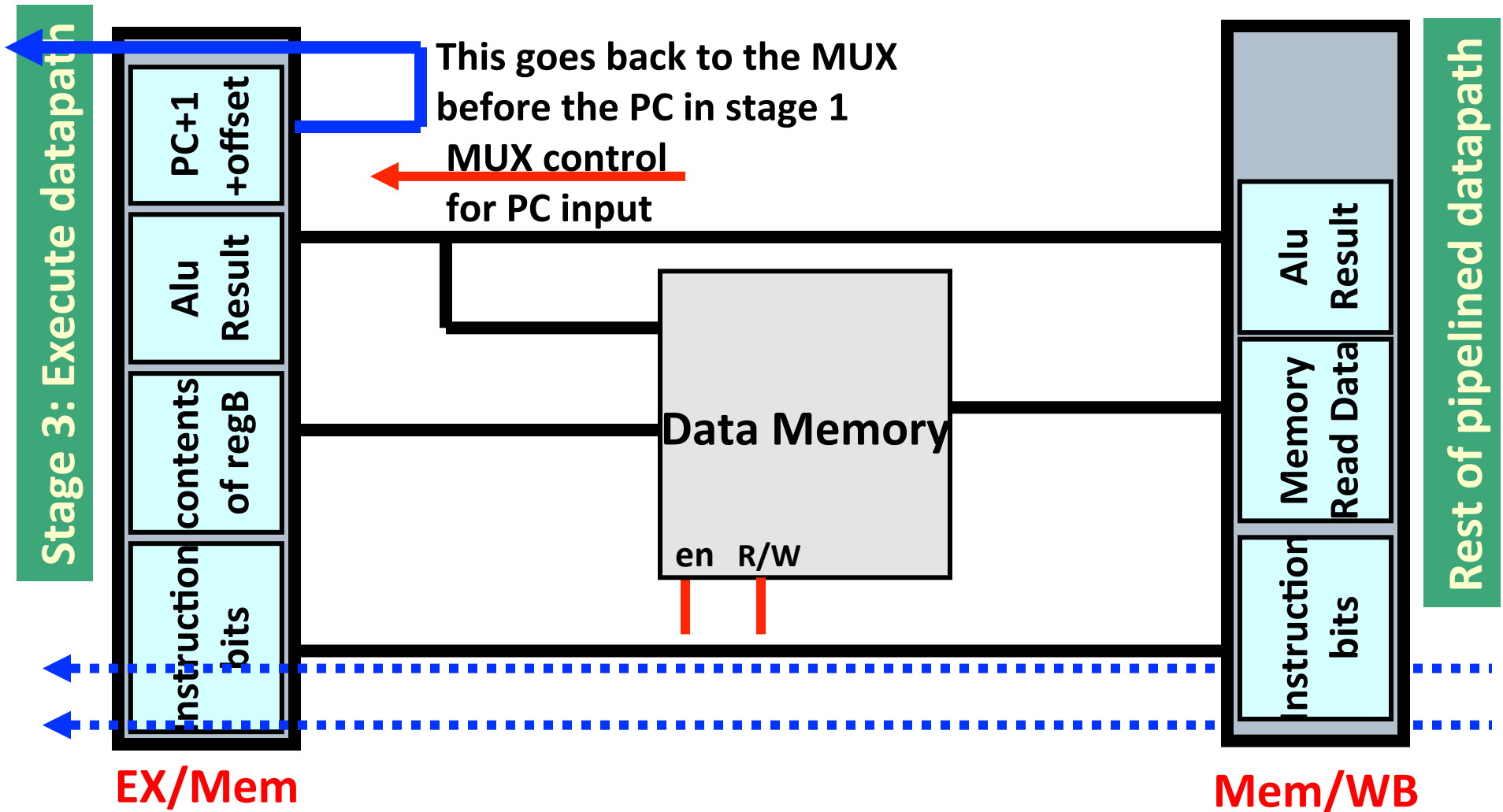Instruction bits

Rest of pipelined datapath

ID / EX

EX/Mem

24

Stage 4: Memory Operation

❑ Design a datapath that performs the proper memory operation for the instruction specified and the values present in the EX/Mem pipeline register.

  • ALU result contains address for **ld** and **st** instructions.

  • Opcode bits control memory R/W and enable signals.

❑ Write everything needed to complete execution to the pipeline register (Mem/WB)

  • ALU result and MemData
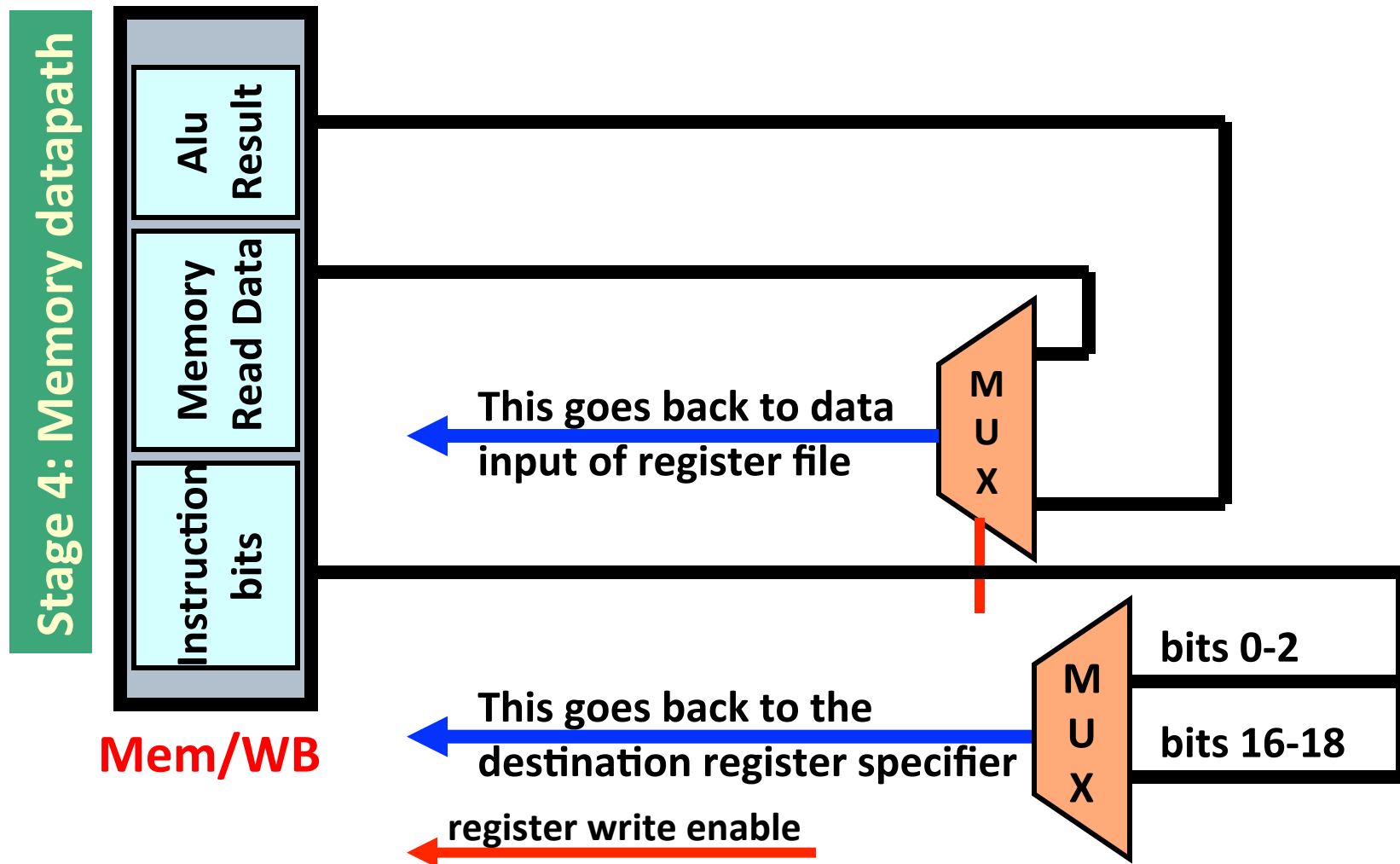
  • Instruction bits for opcode and destReg specifiers

Pipeline datapath – Memory stage

Stage 3: Execute datapath

PC+1 +offset

Alu Result

contents of regB

Instruction bits

This goes back to the MUX before the PC in stage 1

MUX control for PC input

Data Memory

en   R/W

Alu Result

Memory Read Data

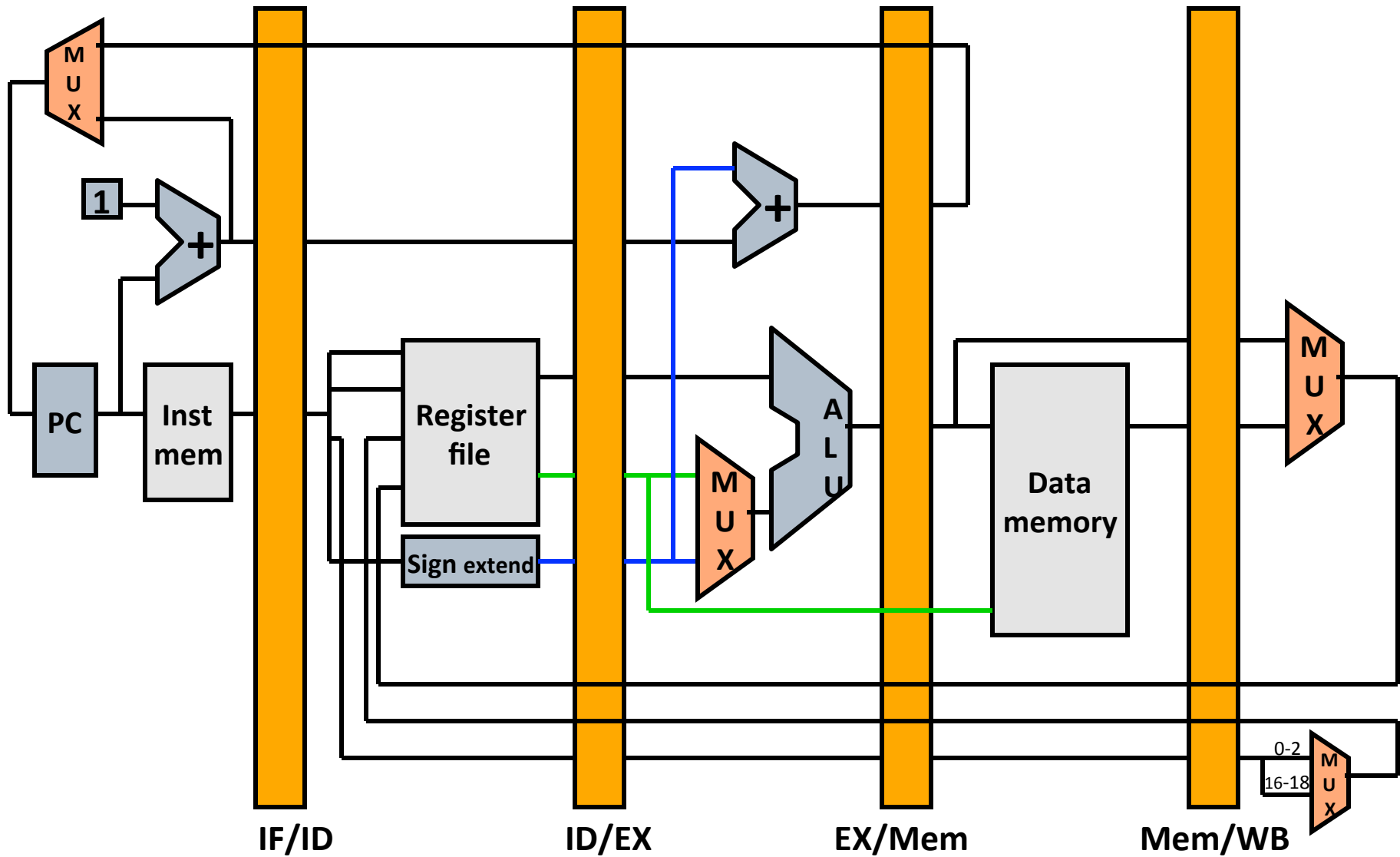Instruction bits

Rest of pipelined datapath

EX/Mem

Mem/WB

Stage 5: Write back

❑ Design a datapath that completes the execution of this instruction, writing to the register file if required.

- Write MemData to destReg for ld instruction

- Write ALU result to destReg for add or nand instructions.

- Opcode bits also control register write enable signal.
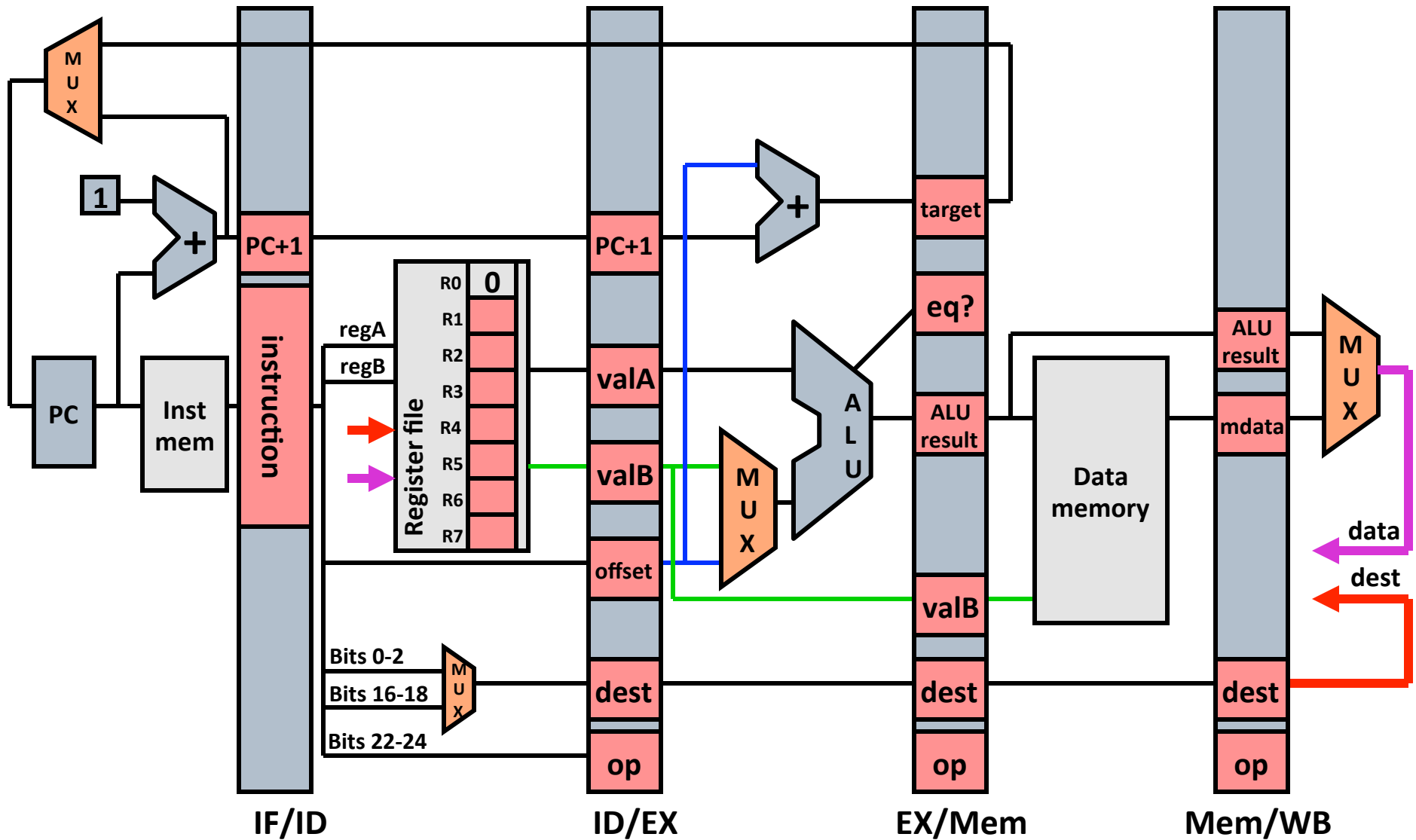
Pipeline datapath – Writeback stage

Stage 4: Memory datapath

Alu Result

Memory Read Data

Instruction bits

Mem/WB

This goes back to data input of register file

M U X

This goes back to the destination register specifier

register write enable

M U X

bits 0-2

bits 16-18

Putting all together

IF/ID ID/EX EX/Mem Mem/WB

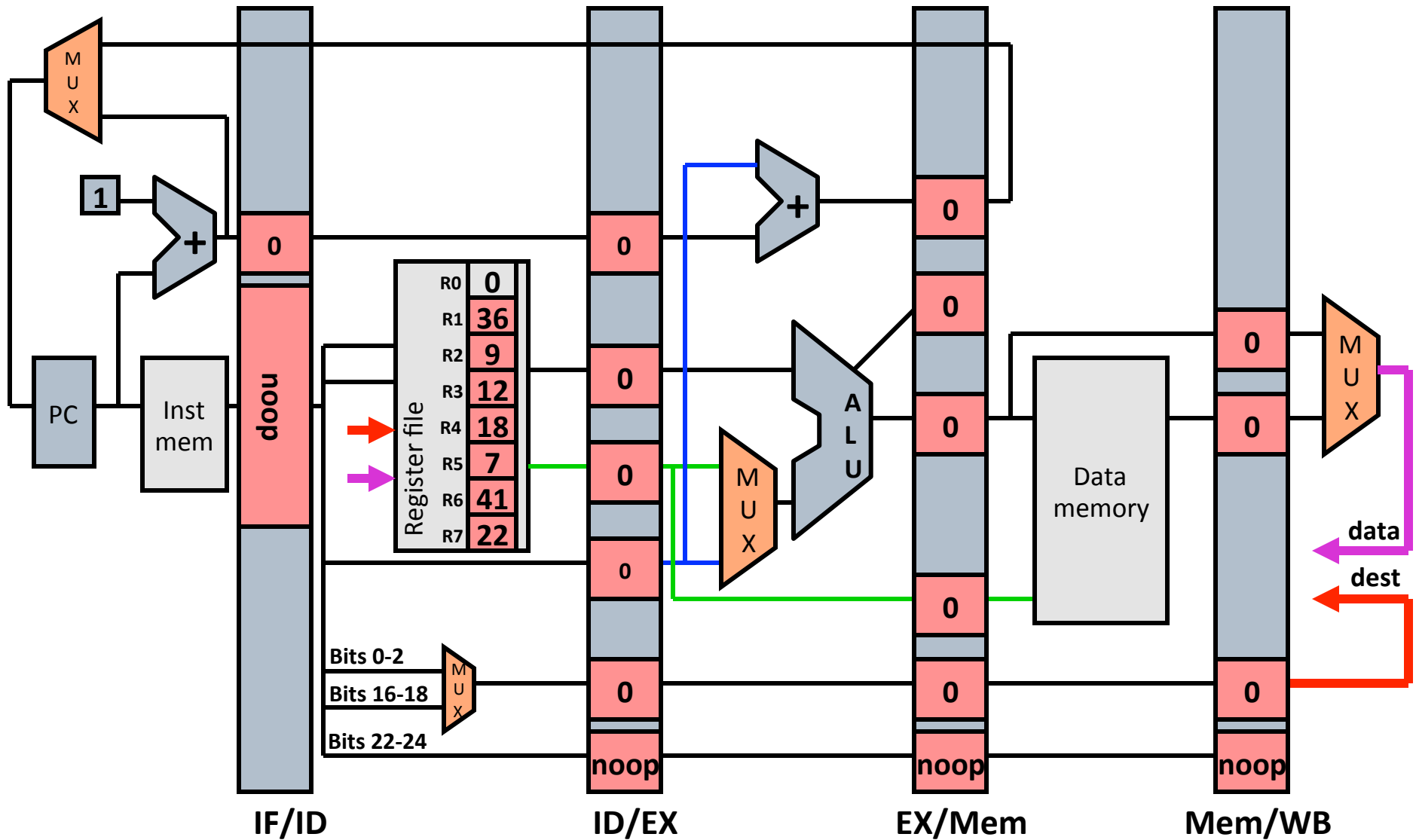# Sample Code (Simple)

Let's run the following code on pipelined LC2K2x:

- add   1   2   3   ;   reg 3 = reg 1 + reg 2
- nor   4   5   6   ;   reg 6 = reg 4 nor reg 5
- lw   2   4   20   ;   reg 4 = Mem[reg2+20]
- add   2   5   5   ;   reg 5 = reg 2 + reg 5
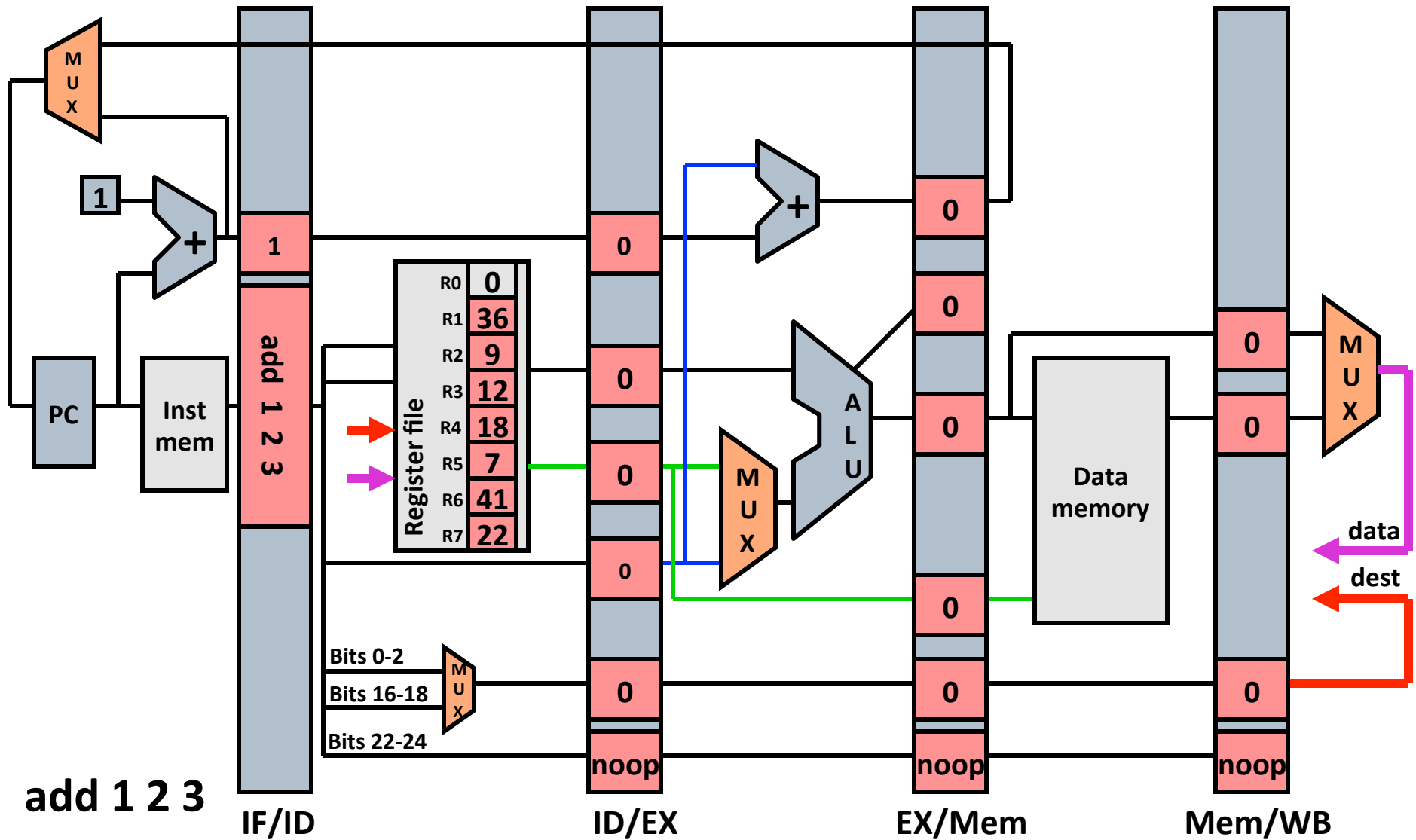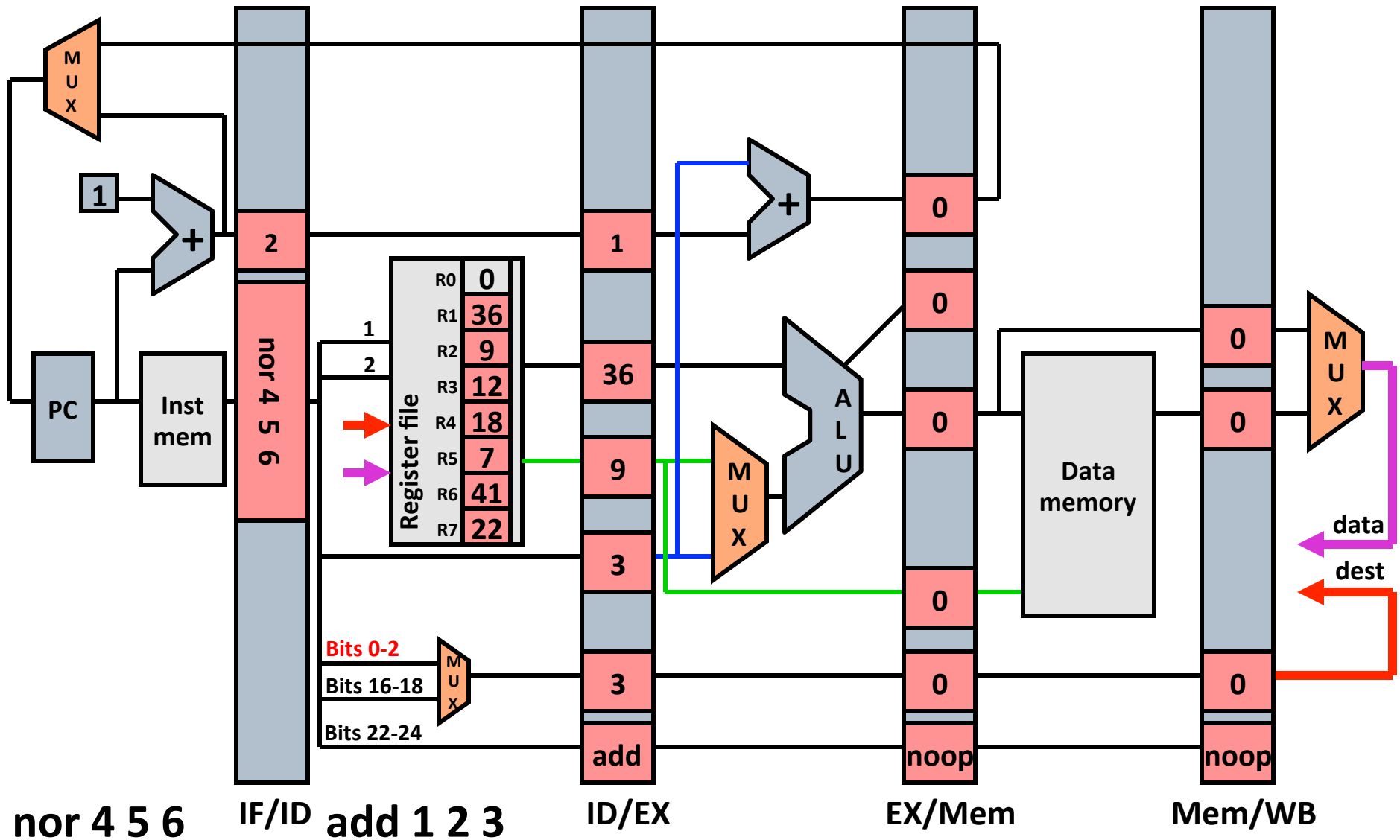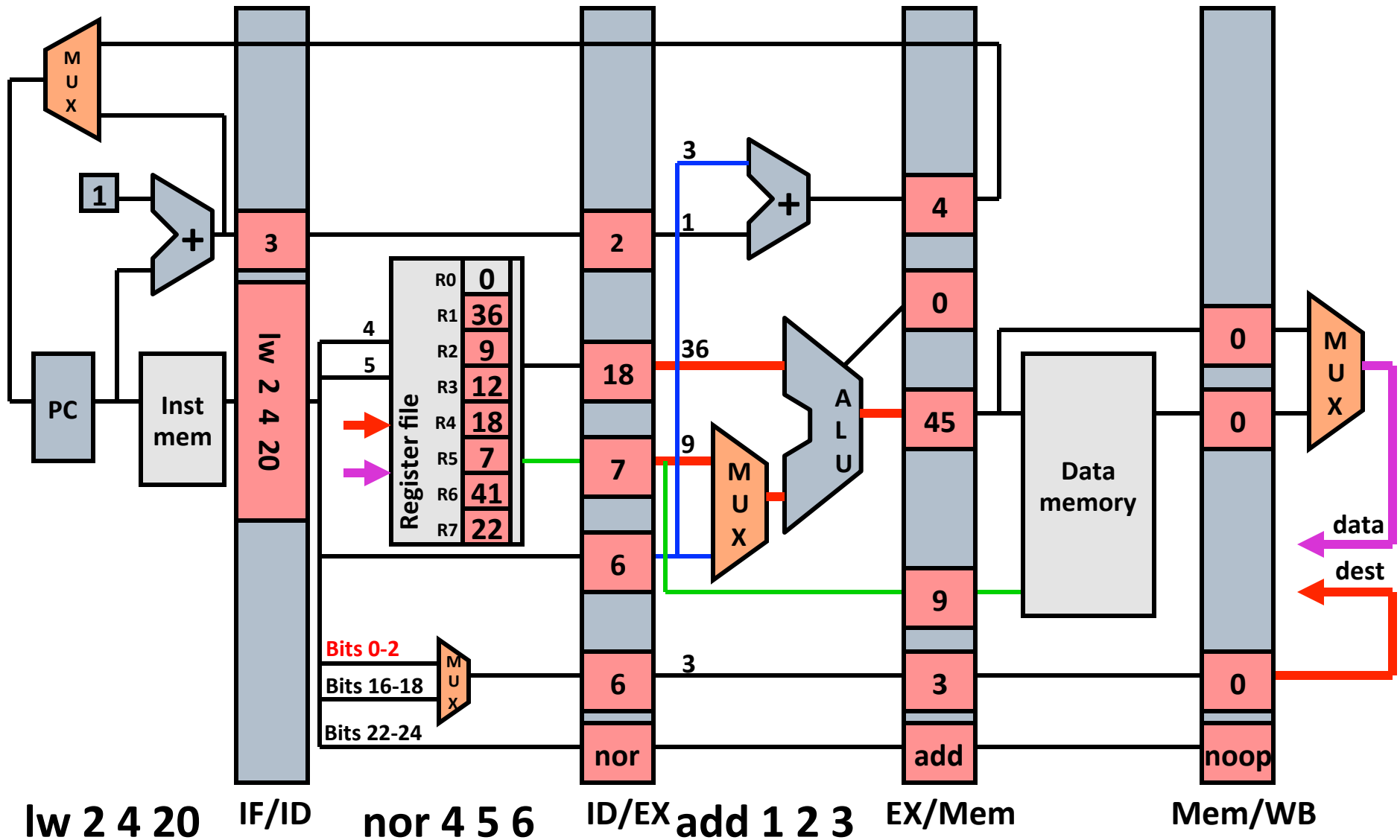- sw   3   7   10   ;   Mem[reg3+10] =reg 7

Pipeline datapath

IF/ID ID/EX EX/Mem Mem/WB

31

Time 0 - Initial state

32

Time 1 - Fetch: add 1 2 3

add 1 2 3

IF/ID   ID/EX   EX/Mem   Mem/WB

Time 2 - Fetch: nand 4 5 6

Time 3 - Fetch: lw 2 4 20
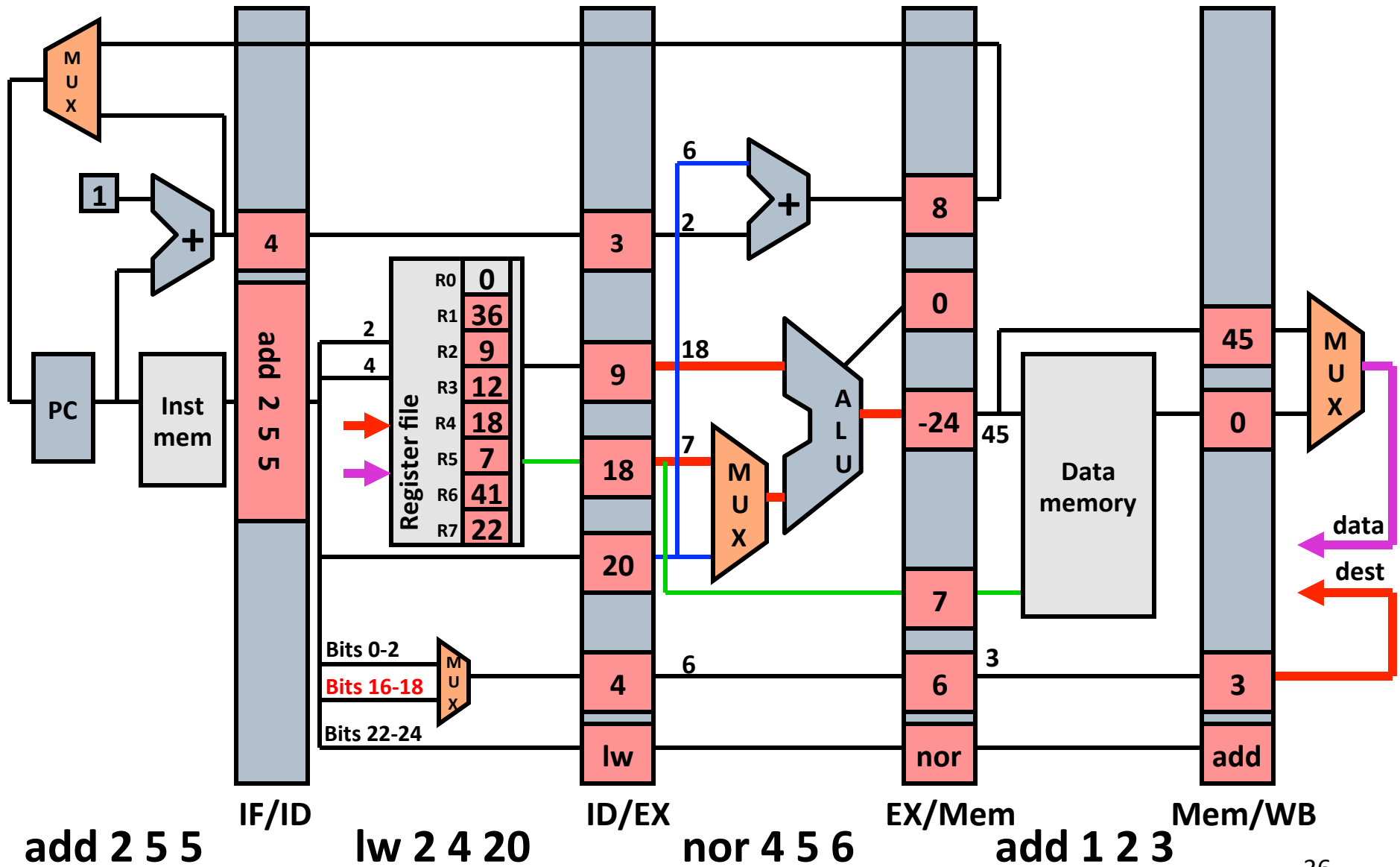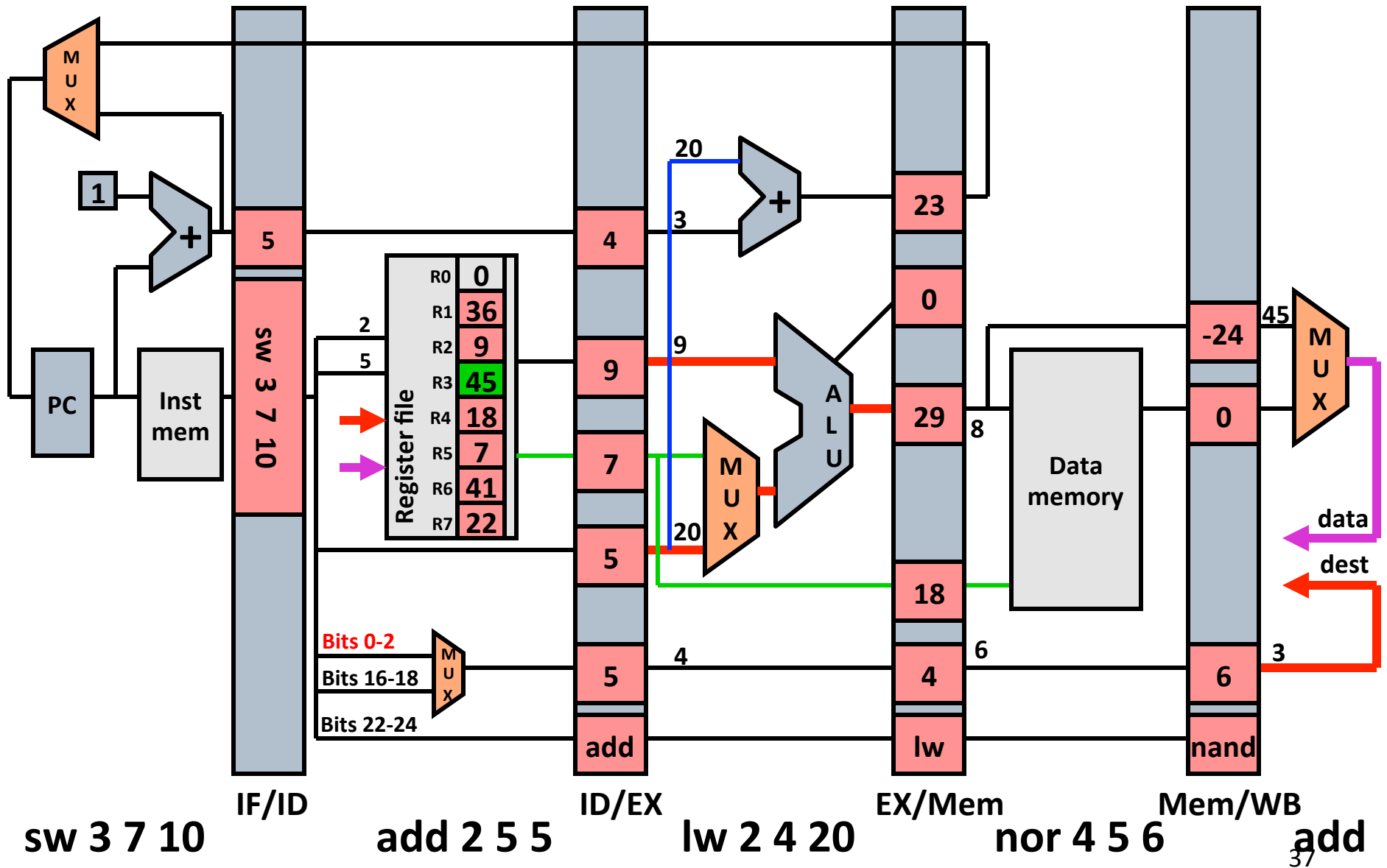
lw 2 4 20   IF/ID   nor 4 5 6   ID/EX   add 1 2 3   EX/Mem   Mem/WB

35

Time 4 - Fetch: add 2 5 5

IF/ID   **ID/EX**   **EX/Mem**   **Mem/WB**

add 2 5 5   **lw 2 4 20**   **nor 4 5 6**   **add 1 2 3**

36

Time 5 - Fetch: sw 3 7 10

IF/ID   ID/EX   EX/Mem   Mem/WB

sw 3 7 10   add 2 5 5   lw 2 4 20   nor 4 5 6   add

37

Time 6 – no more instructions

IF/ID   ID/EX   EX/Mem   Mem/WB

sw 3 7 10   **add 2 5 5**   **lw 2 4 20**   **nor**

Time 7 – no more instructions

IF/ID   **ID/EX**   **EX/Mem**   **Mem/WB**

sw 3 7 10   **add 2 5 5**   **lw**

Time 8 – no more instructions

IF/ID   **ID/EX**   **EX/Mem**   **Mem/WB**

sw 3 7 10   add

Register file:
R0 0
R1 36
R2 9
R3 45
R4 99
R5 16
R6 -24
R7 22

Bits 0-2
Bits 16-18
Bits 22-24

Data memory: 22

55 16
55 0
22
7 5
sw

data
dest
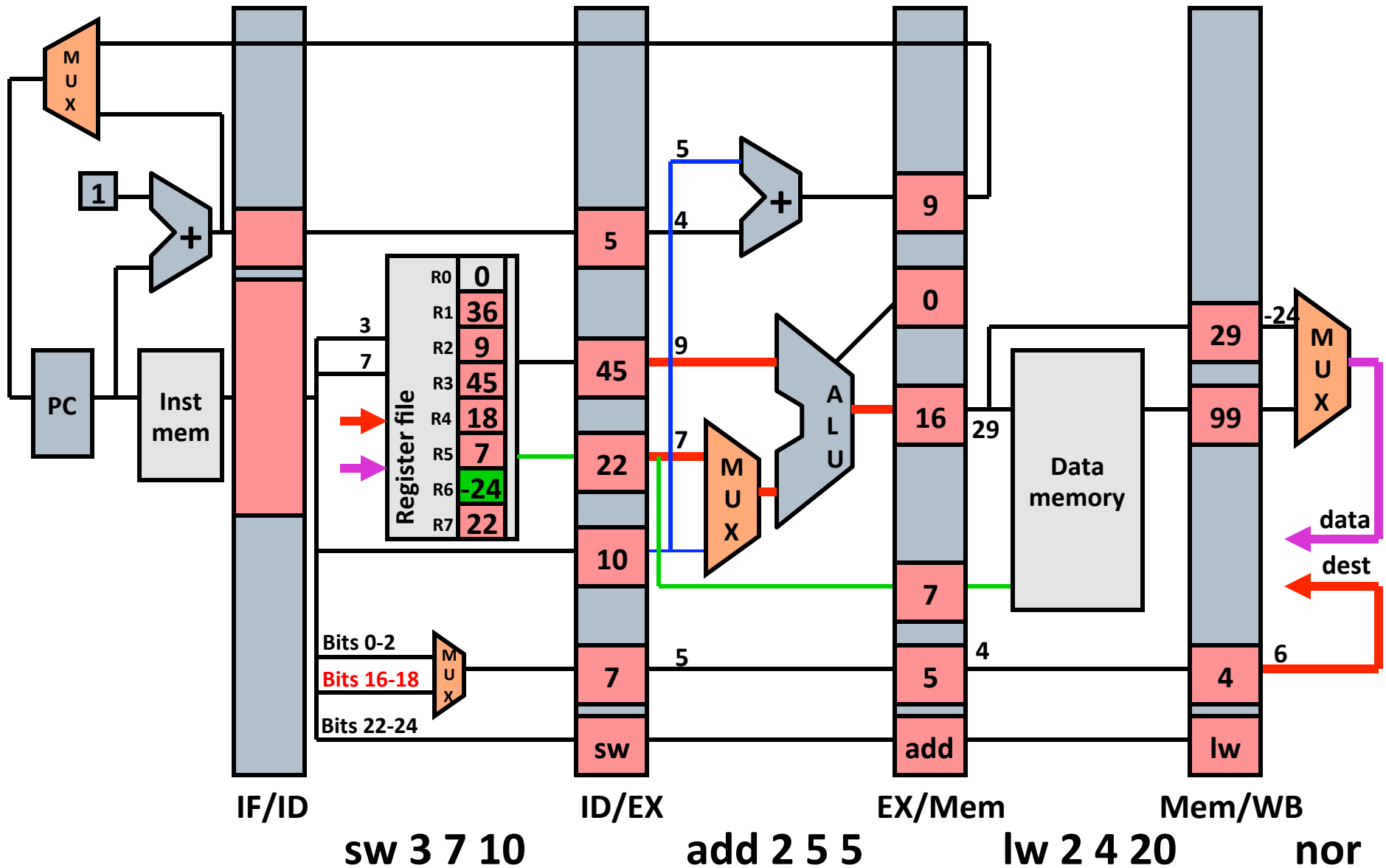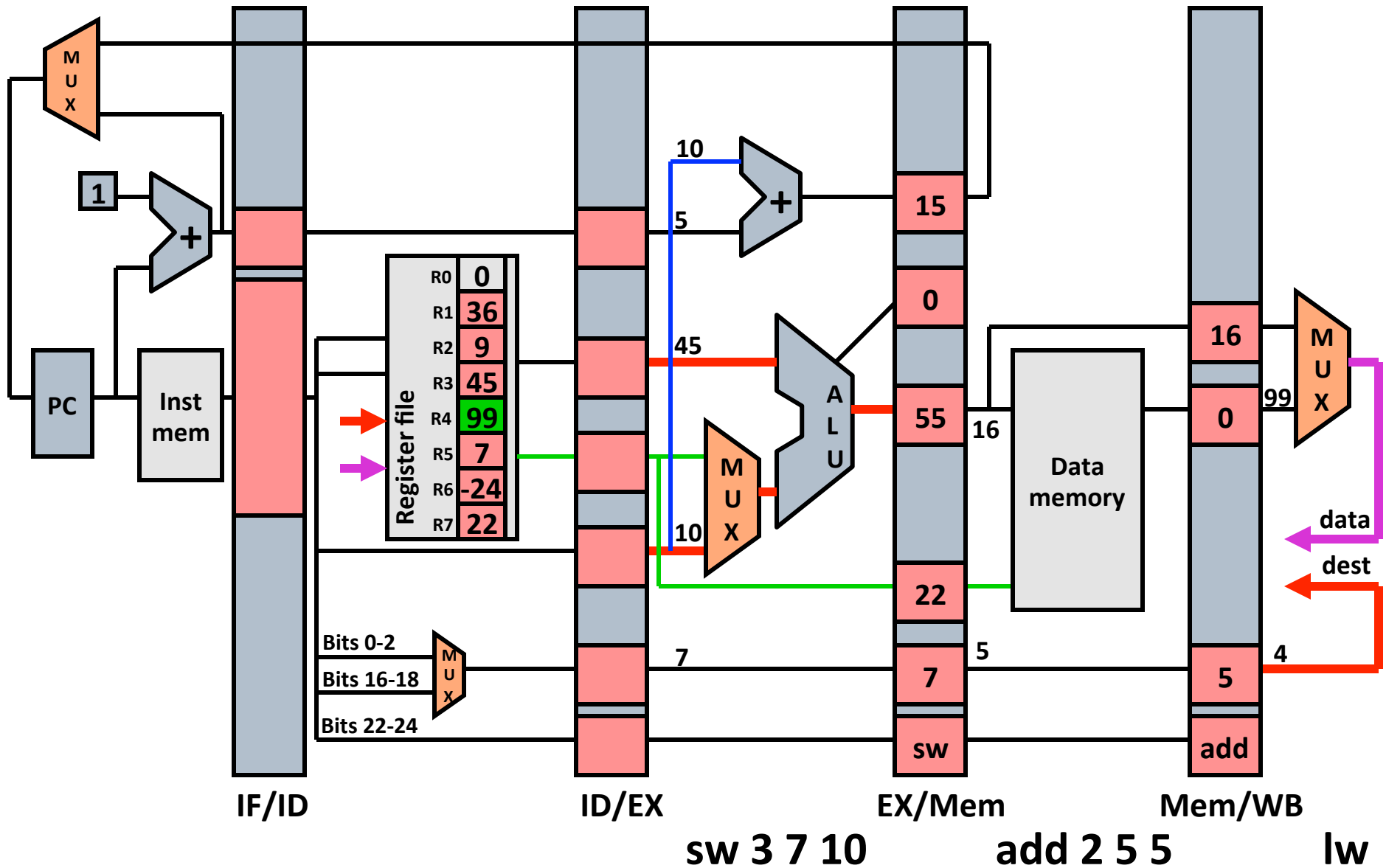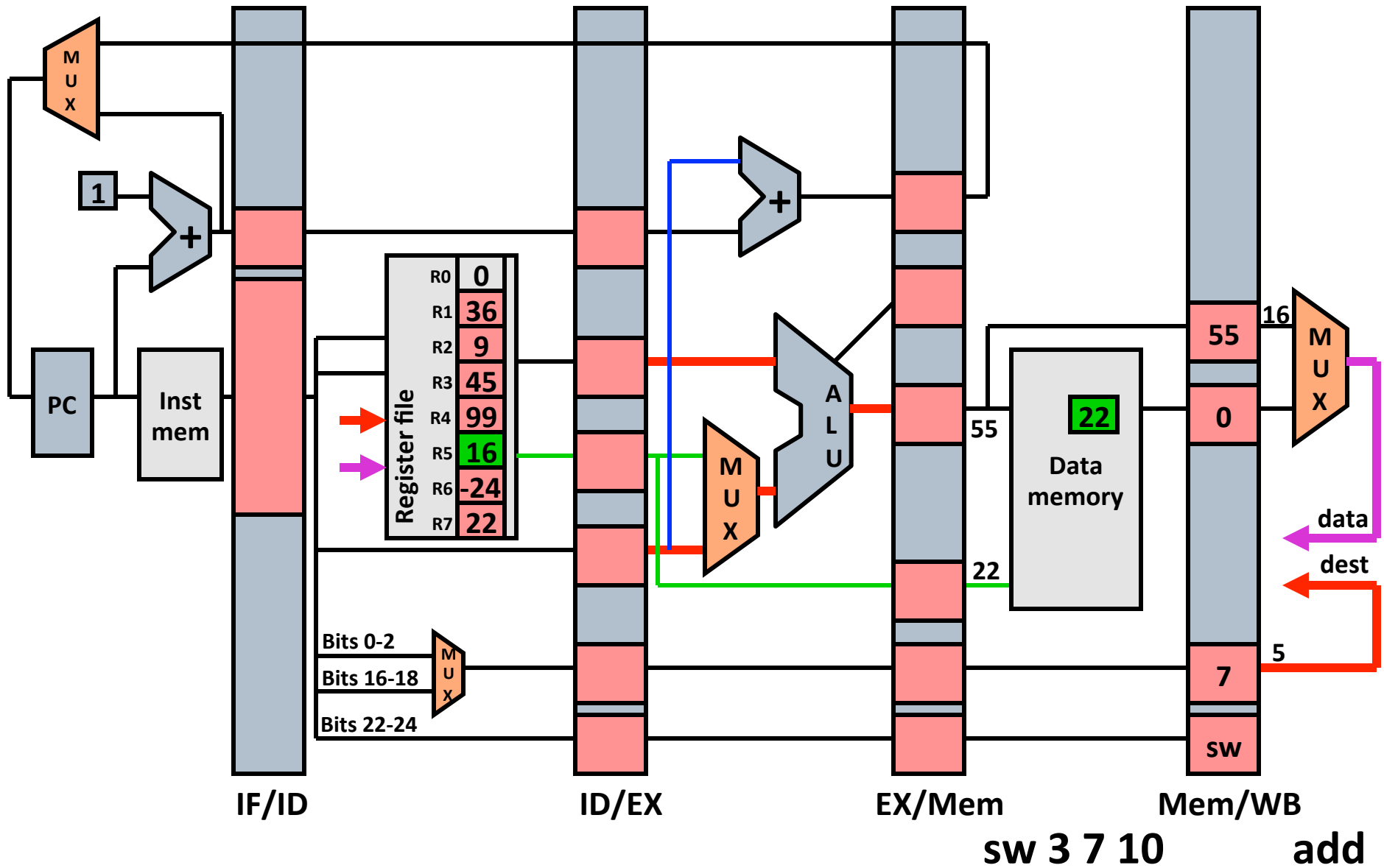
Time 9 – no more instructions
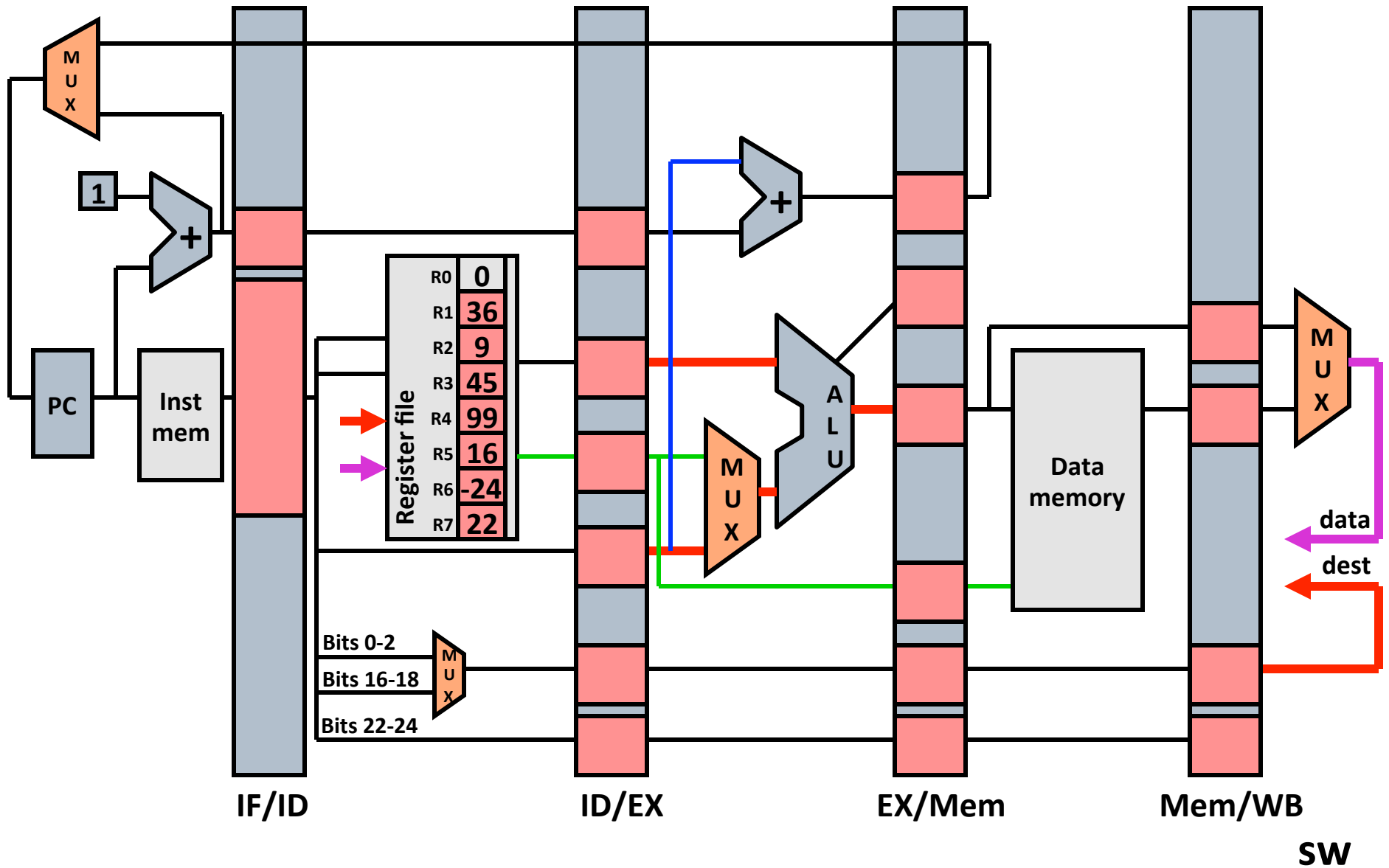
Time graphs (a.k.a. pipe trace)

Time:	1	2	3	4	5	6	7	8	9
add	fetch	decode	execute	memory	writeback				
nor		fetch	decode	execute	memory	writeback			
lw			fetch	decode	execute	memory	writeback		
add				fetch	decode	execute	memory	writeback	
sw					fetch	decode	execute	memory	writeback

A vertical slice reports the entire activity of the pipeline at time 5

What can go wrong?

❑ Data hazards: since register reads occur in stage 2 and register writes occur in stage 5 it is possible to read the wrong value if it is about to be written.

❑ Control hazards: A branch instruction may change the PC, but not until stage 4. What do we fetch before that?

❑ Exceptions: How do you handle exceptions in a pipelined processor with 5 instructions in flight?

❑ Next Lecture: Data Hazards