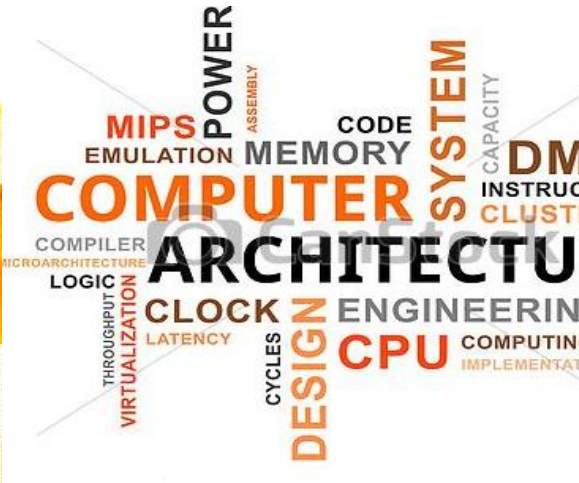
UNIT III
PROCESSOR AND PIPELINING

Fundamental concepts – Execution of a complete instruction – Multiple bus organization – Hardwired control – Micro programmed control – **Pipelining: Basic concepts** – Data hazards – Instruction hazards – Influence on Instruction sets – Data path and control consideration.



Recap the previous Class

Five instruction execution steps

- Instruction fetch
- Instruction decode and register read
- Execution, memory address calculation, or branch completion
- Memory access or R-type instruction completion
- Write-back

Instruction execution takes 3~5 cycles!

Goal of pipelining is...

- Throughput!
- Observation
  - Instructions follow “steps”
  - Some steps are common (e.g., fetch, decode)

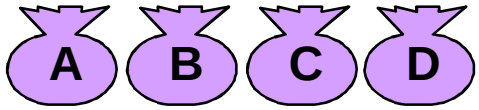
    Because we “program” ALU, the execution step is also common
- Pipelining
  - Basic idea: overlap instruction execution

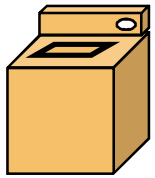
    While instruction #1 is in the decode stage, fetch instruction #2

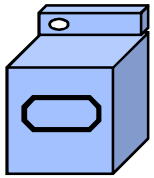
    Provide more resources such that overlapping is not hindered by sharing of resources

What Is Pipelining

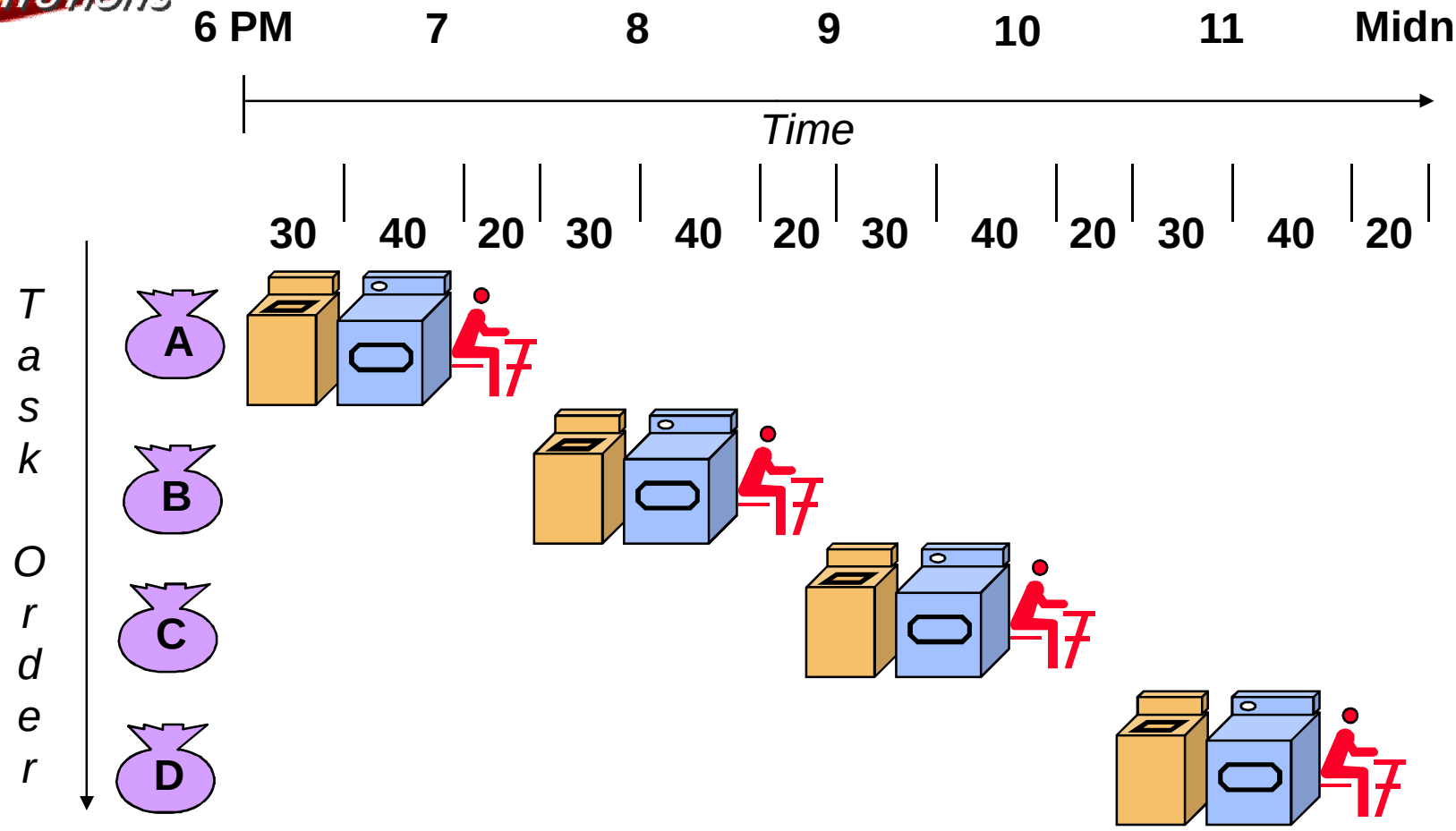
- Laundry Example
- A, B, C, D each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- “Folder” takes 20 minutes







What Is Pipelining

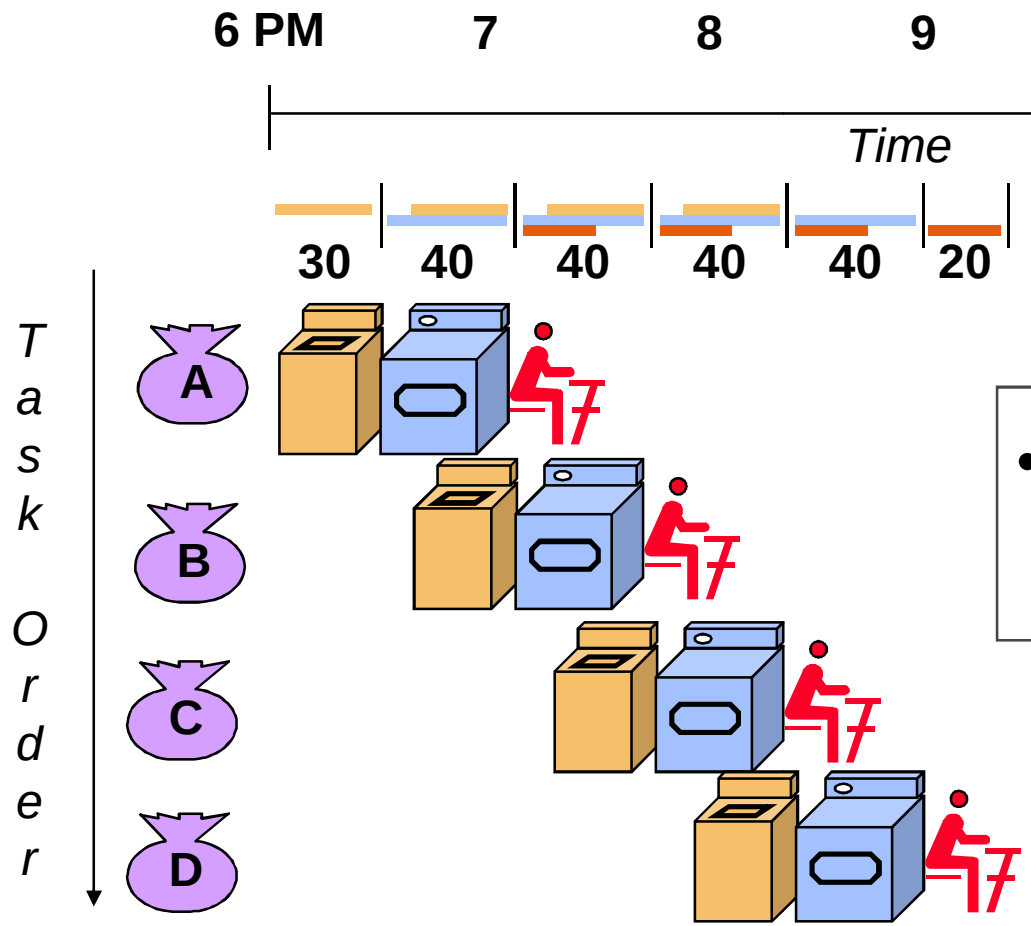


Sequential laundry takes 6 hours for 4 loads

If they learned pipelining, how long would laundry take?

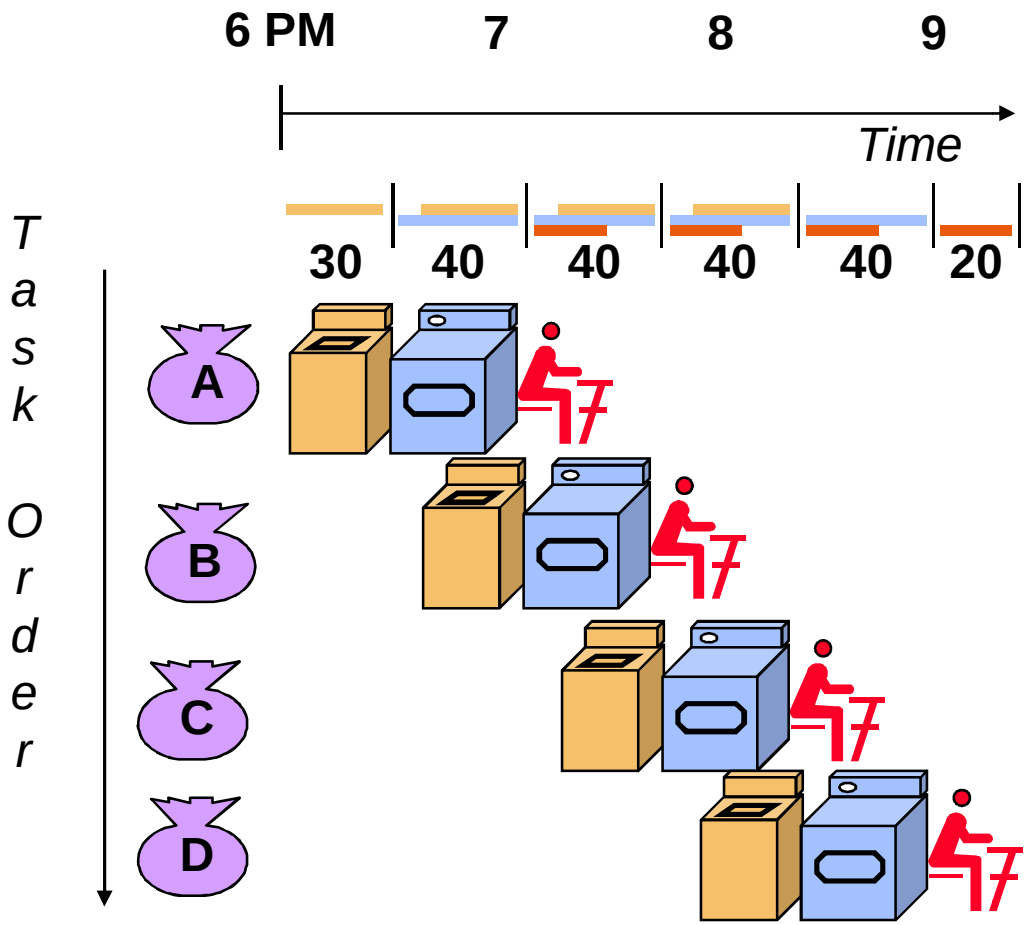
What Is Pipelining

Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Lessons



- Pipelining doesn't help latency of single task, **it helps throughput** of entire workload
- Pipeline **rate limited** by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

Five pipeline stages

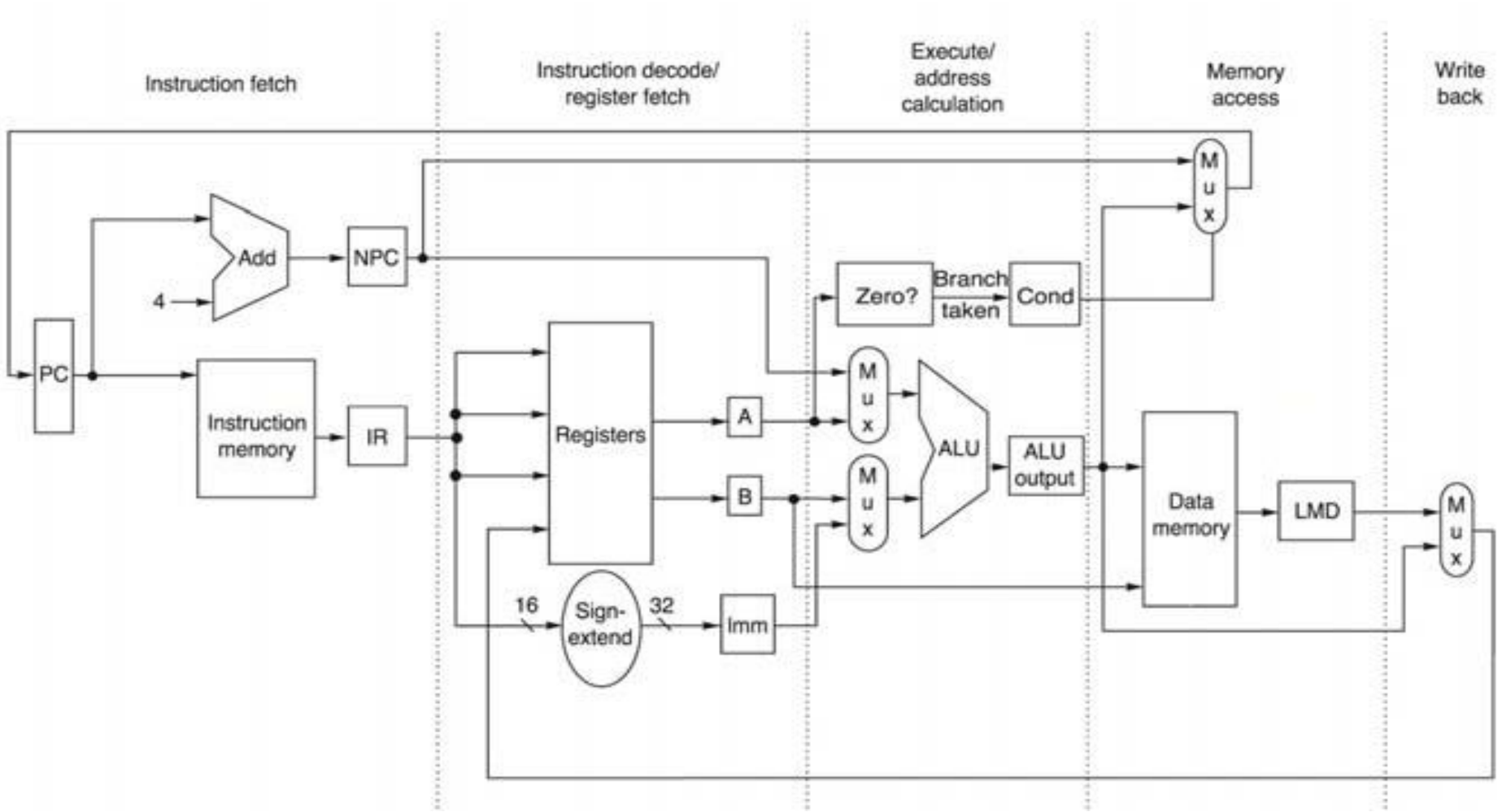
- In fact, some commercial processors follow this design
  - Early MIPS design
  - ARM9 (very popular embedded processor core)
- Fetch (F) - Fetch instruction
- Decode (D) - Decode instruction and read operands
- Execute (X)
  - ALU operation, address calculation in the case of memory access
- Memory access (M)
- Write back (W) - Update register file

Pipelining instruction execution

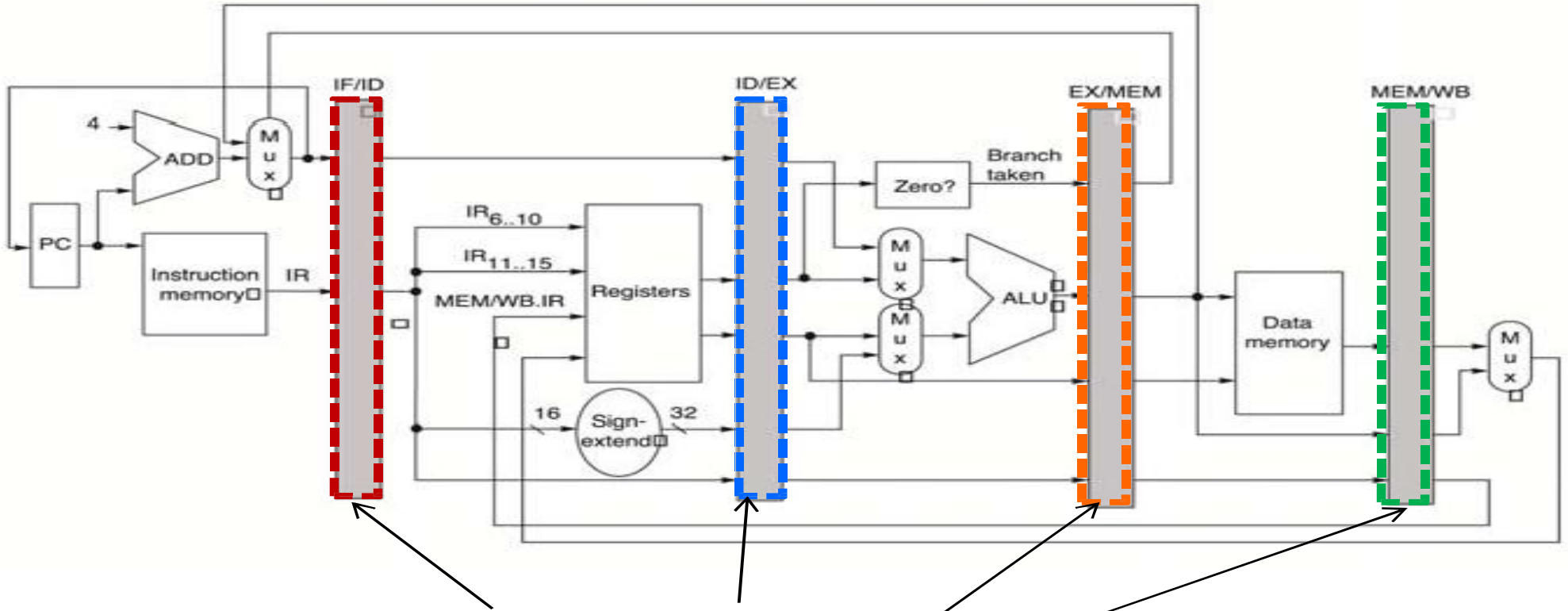
IF ID EX MEM WB

	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9
ADD	IF	ID	EX	MEM	WB				
SUB		IF	ID	EX	MEM	WB			
LOAD			IF	ID	EX	MEM	WB		
AND				IF	ID		EX	MEM	WB
OR					IF		ID	EX	MEM

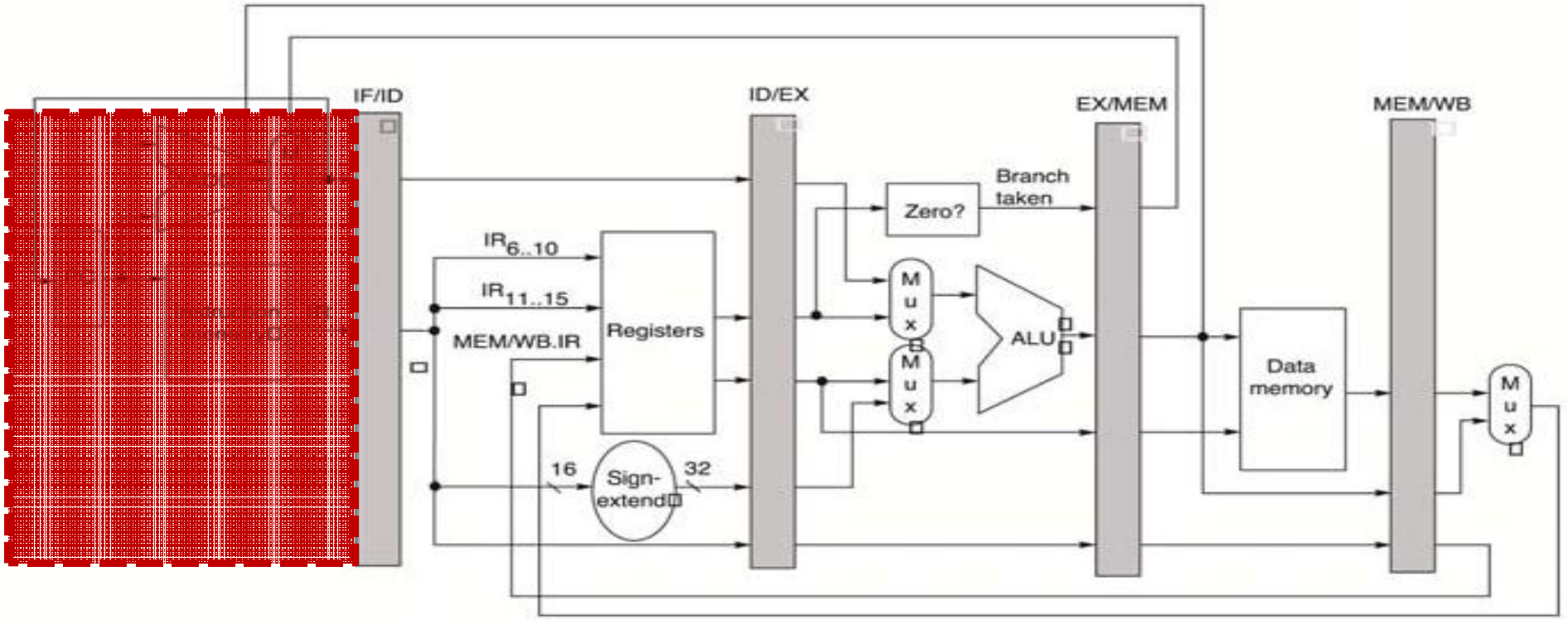
Multi-cycle to pipelined datapath



Multi-cycle to pipelined datapath

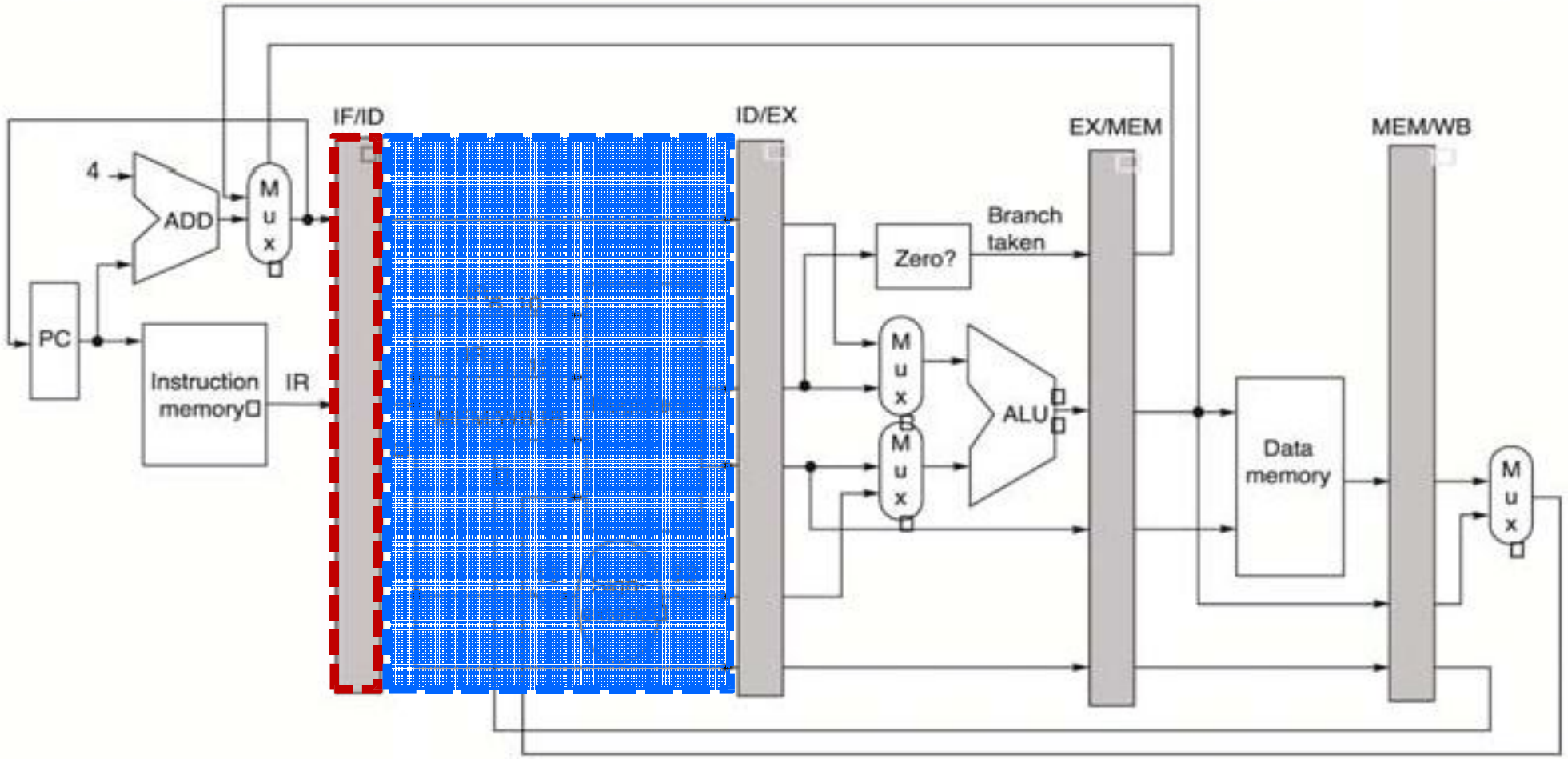


lw in the “F” stage



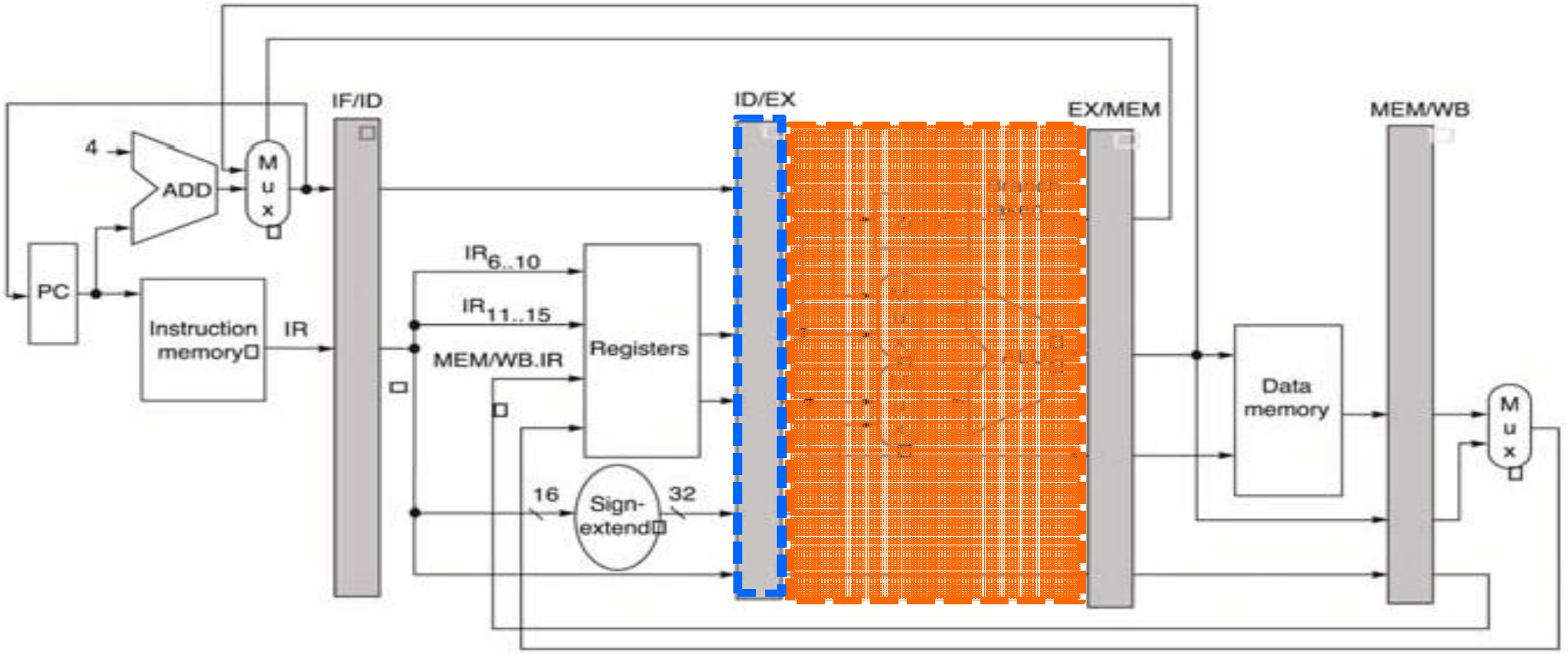
Read an instruction from instruction memory; address is in PC; compute (PC+4) to update PC

lw in the “D” stage



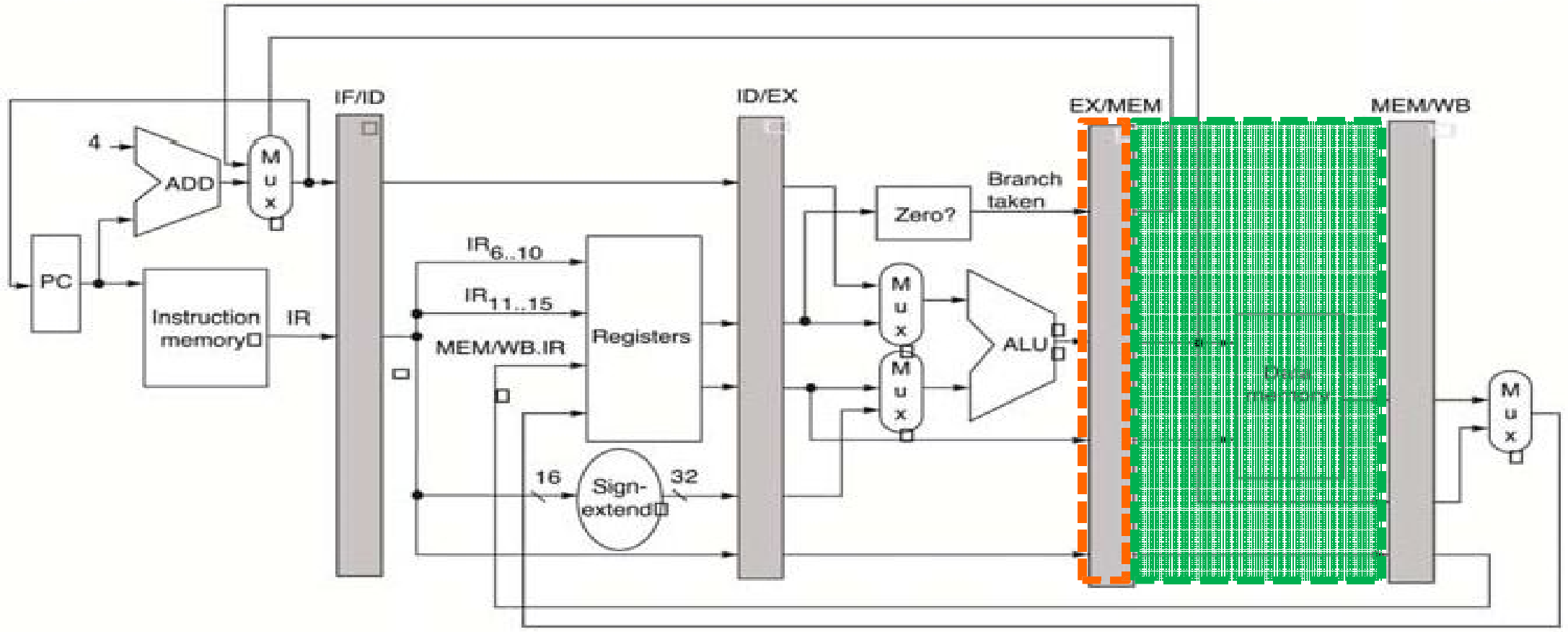
Read operands from register file; sign-extend the immediate field

lw in the “X” stage



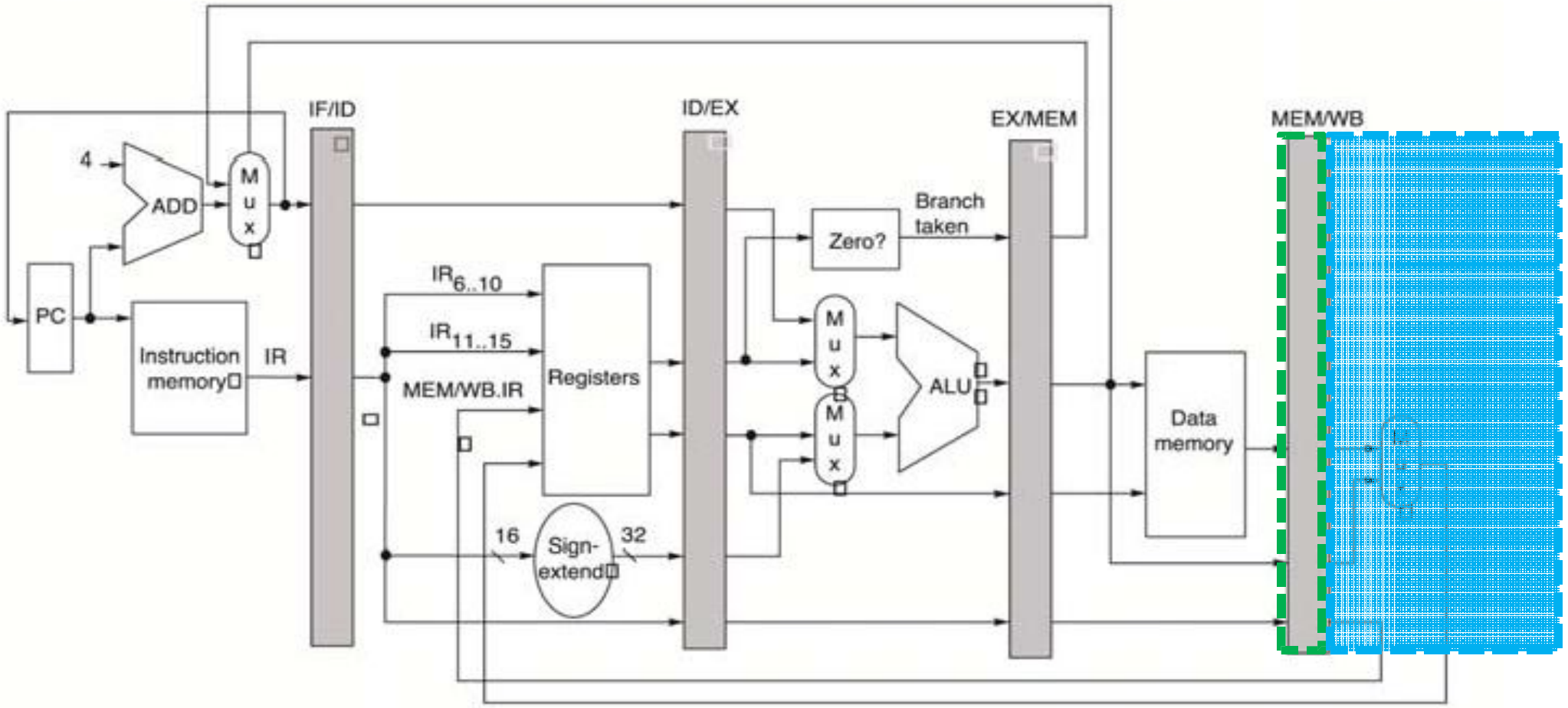
Add the base register value and the immediate value to form memory access address;

lw in the “M” stage



Read a value from memory

lw in the "W" stage

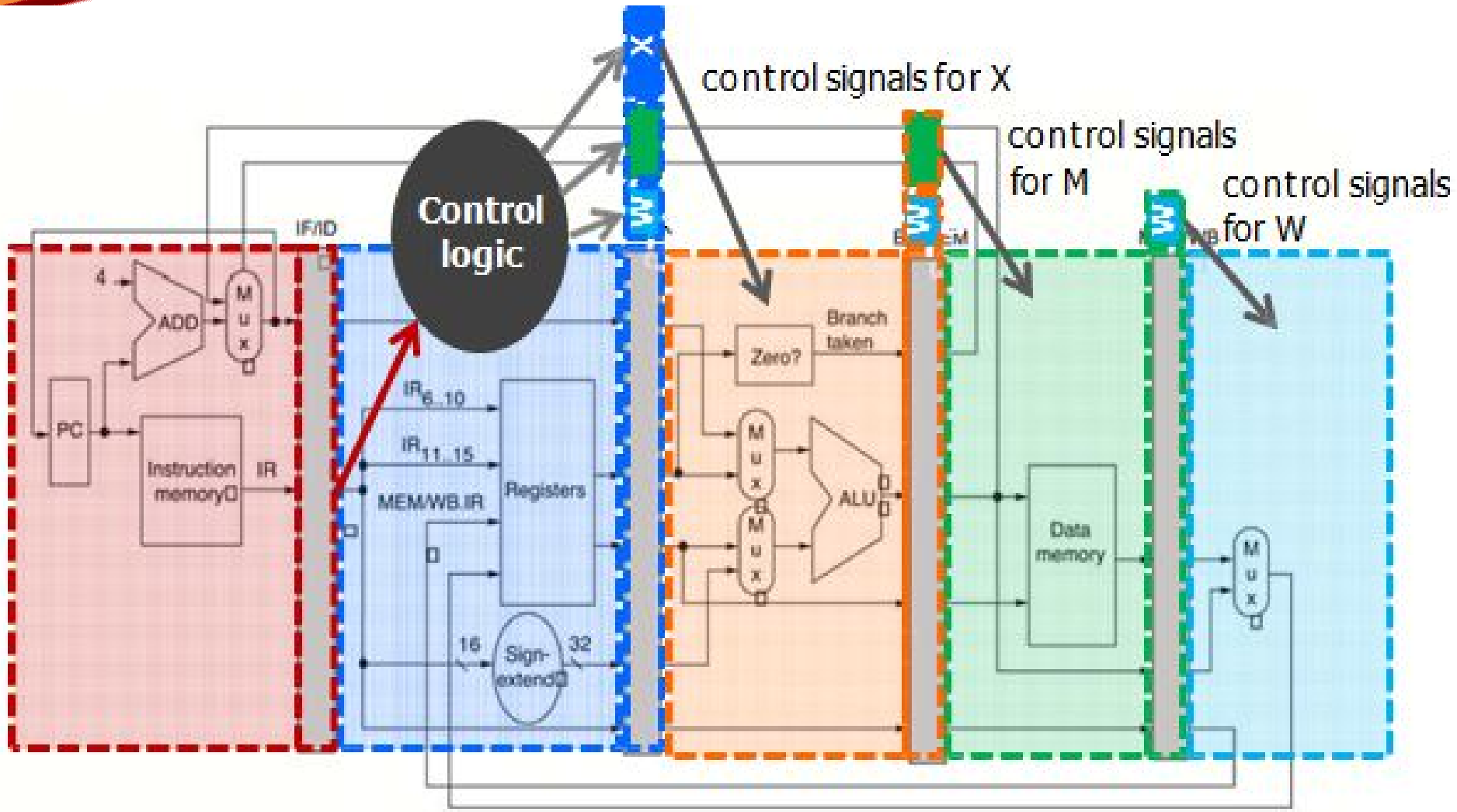


Update register file

Pipeline control

- There are multiple instructions in flight (in different pipeline stages)
- Hence, control signals for an instruction should flow through the pipeline stages with the instruction
- Alternatively, with the instruction information in each pipeline register, one can generate control signals by decoding the information
- Pipeline control becomes more complex than previous designs because of potential dependences between instructions in flight

Pipeline control



Pipelining performance

- Time between instructions
  - Pipelining: 1 cycle

    ~Time for one instruction (# cycles) / # pipeline stages
  - Non-pipelined

    Single-cycle: 1 cycle (but it is LONG)

    Multi-cycle: N cycles (depending on instruction)
- Pipelining does NOT improve latency of a single instruction
  - In fact, instruction execution latency can be even longer (why?)
  - Pipelining improves “throughput” (what is throughput?)
  - It improves the program execution time (why?)

TEXT BOOK

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, McGraw-Hill, 6th Edition 2012.

REFERENCES

1. David A. Patterson and John L. Hennessey, “Computer organization and design”, MorganKauffman ,Elsevier, 5th edition, 2014.
2. William Stallings, “Computer Organization and Architecture designing for Performance”, Pearson Education 8th Edition, 2010
3. John P.Hayes, “Computer Architecture and Organization”, McGraw Hill, 3rd Edition, 2002
4. M. Morris R. Mano “Computer System Architecture” 3rd Edition 2007
5. David A. Patterson “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann; 5th edition 2011

Courtesy : **University of Pittsburgh**

THANK YOU