Roadmap

Instruction Level Parallelism (ILP)

1. Review of Simple Pipelined Processors
2. Dynamically Scheduled Pipelines
3. Superscalar and Multithreaded Processors

1. Simple Pipelining

- Baseline design:
  - Introduce a simple RISC Instruction Set Architecture (ISA) (called DLX)
  - Non-pipelined DLX implementation
- 5-Stage Pipelined DLX Processor
- Hazards and their solutions

Roadmap

Instruction Level Parallelsim (ILP)

Overlapped execution of instructions from a sequential program

- Programmer obliviousness
- Single-threaded applications
- Compiler support

Limits

ILP Wall

Memory Wall

Power Wall

Simplified DLX Instruction Set

DLX: Idealized RISC processor (similar to MIPS, ARM)

- Load/Store architecture using base + offfset addressing
- 32 bit word size aligned at word address boundaries
- 32-bit memory addresses (aligned)

Registers

- 32 32-bit Integer GPRs R0 …. R31
- 32 32-bit Floating Point Registers F0, .. F31
- No condition codes

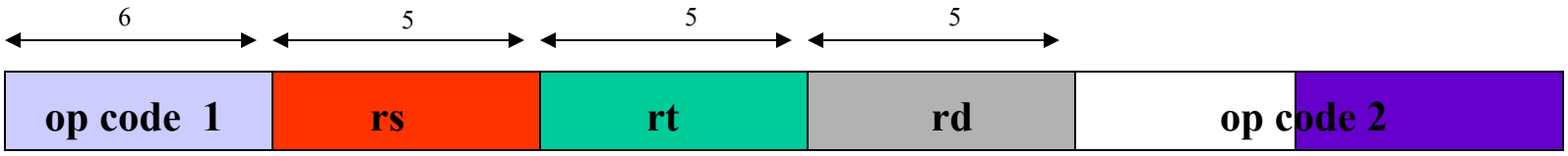
Instructions

- Integer ALU
- Floating Point (FP)
- Memory Access
- Program Control :

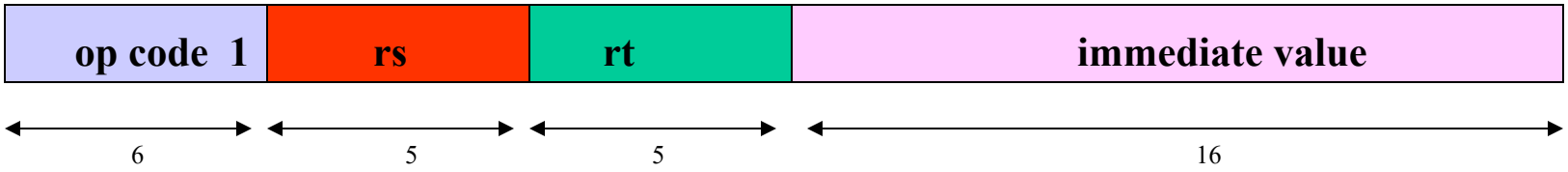
Instruction Formats

All instructions are one word long (32 bit aligned)

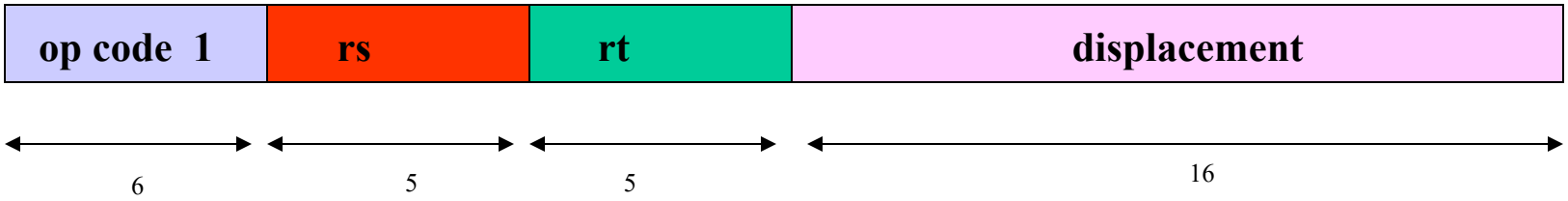
R-R Instructions: add rd, rs, rt



R-I Instructions: addi rt, rs, d



Load/Store/Branch Instructions: lw rt, d(rs), sw rt, d(rs), beq rs, rt, d(PC)



Simplified Integer DLX Instruction Set

1. ALU instructions: (ADD, SUB, AND, OR, XOR, ….)

RR mode: ADD rd, rs, rt | [rd] = [rs] + [rt]

Example: ADD R2, R4, R5

The 32-bit contents of registers rs and rt are added, and the sum is written to register rd.

RI mode: ADDI rt, rs, d | [rt] = [rs] + EXT(d)

Example: ADDI R2, R4, 1000

The 16-bit value d is sign-extended to 32 bits and added to the 32-bit contents of register rs. The sum is written to register rt.

Simplified Integer DLX Instruction Set

2. Memory reference instructions: Load (LW) and Store (SW)

LW rt, d(rs)	ea = EXT(d) + (rs); (rt) = MEM[ea]
SW rt, d(rs)	ea = EXT(d) + (rs); MEM[ea] = (rt)

Example:

LW R5, 0(R2)	R5 gets the word whose *address* is stored in R2
SW R6, 1000(R2)	R5 written to memory location with address 1000 + value in R2

- LW: Reads a word from memory at effective address [ea] and writes it to register rt
- SW: Writes the contents of register rt to memory at effective address [ea]

- Effective Address ea
  - 16-bit displacement d is sign-extended to 32 bits and added to the contents of base register rs to get the effective address.

Simplified Integer DLX Instruction Set

3. Control instructions

Conditional Branch

if condition is TRUE

go to Target Address;

else

continue with next in-line instruction

- Compare values in two registers (e.g. BEQ, BNE, BGT, BLT, BGW, BLE, …)
- Compare with 0 (e.g. BEQZ, BNEZ, BGTZ, BLTZ, BGEZ, BLEZ, …)

Example

BEQ rs, rt, d(PC) | Go to Target Address if contents of registers rs and rt are equal

BEQZ rs, d(PC) | Go to Target Address if contents of register rs equals zero

Target Address = PC + 4 + Extended(d)

16-bit displacement d is sign-extended to 32 bits

PC is address of the branch instruction

Blank Slide

DLX Implementation

Functional Modules

- Register File (REG)
- Instruction Memory (IM)
- Data memory (DM)
- Arithmetic Logic Unit (ALU)
- Decoder, Program Counter

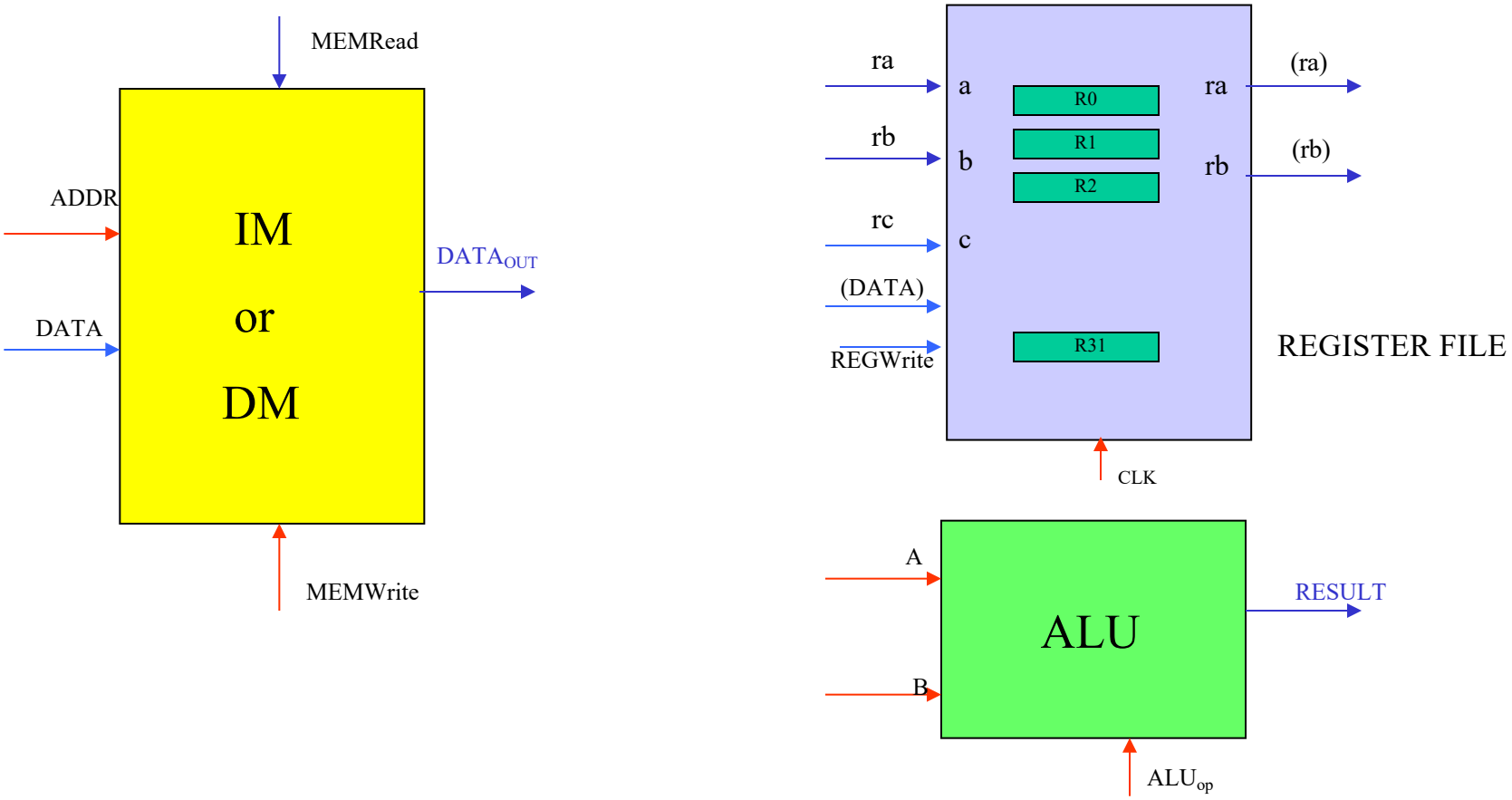
DataPaths

Path of information in executing an instruction: sequence of FMs

Timing and Control

Enforce sequencing in the datapath

Functional Units



Decoder: Combinatorial Circuit that generates control signals from instruction

PC: Register holds address of current instruction. Changes value at next clock edge

Hypothetical Single-cycle Implementation of DLX

Each instruction completes in 1 clock cycle

- Registers (PC + Register File) have stable values after clock edge

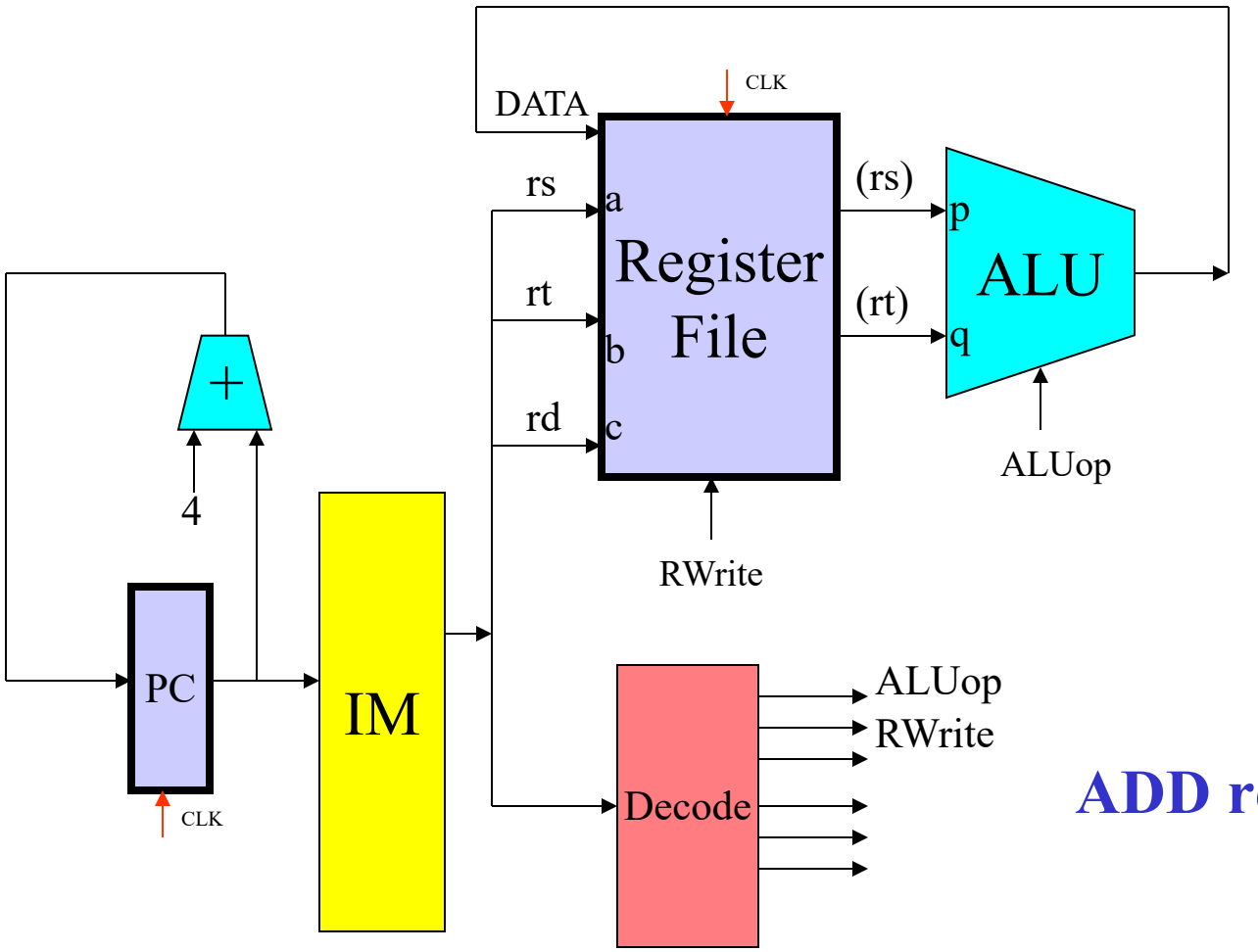
During clock cycle:

1. Read Instruction from Instruction memory (IM)
2. Decode/ Generate control signals
3. Read source register values
4. Generate ALU outputs
5. Read or Write Data Memory for Load or Store instruction
6. New PC value is computed

Update destination register and PC at next clock edge.

Datapaths

R-R Instruction:

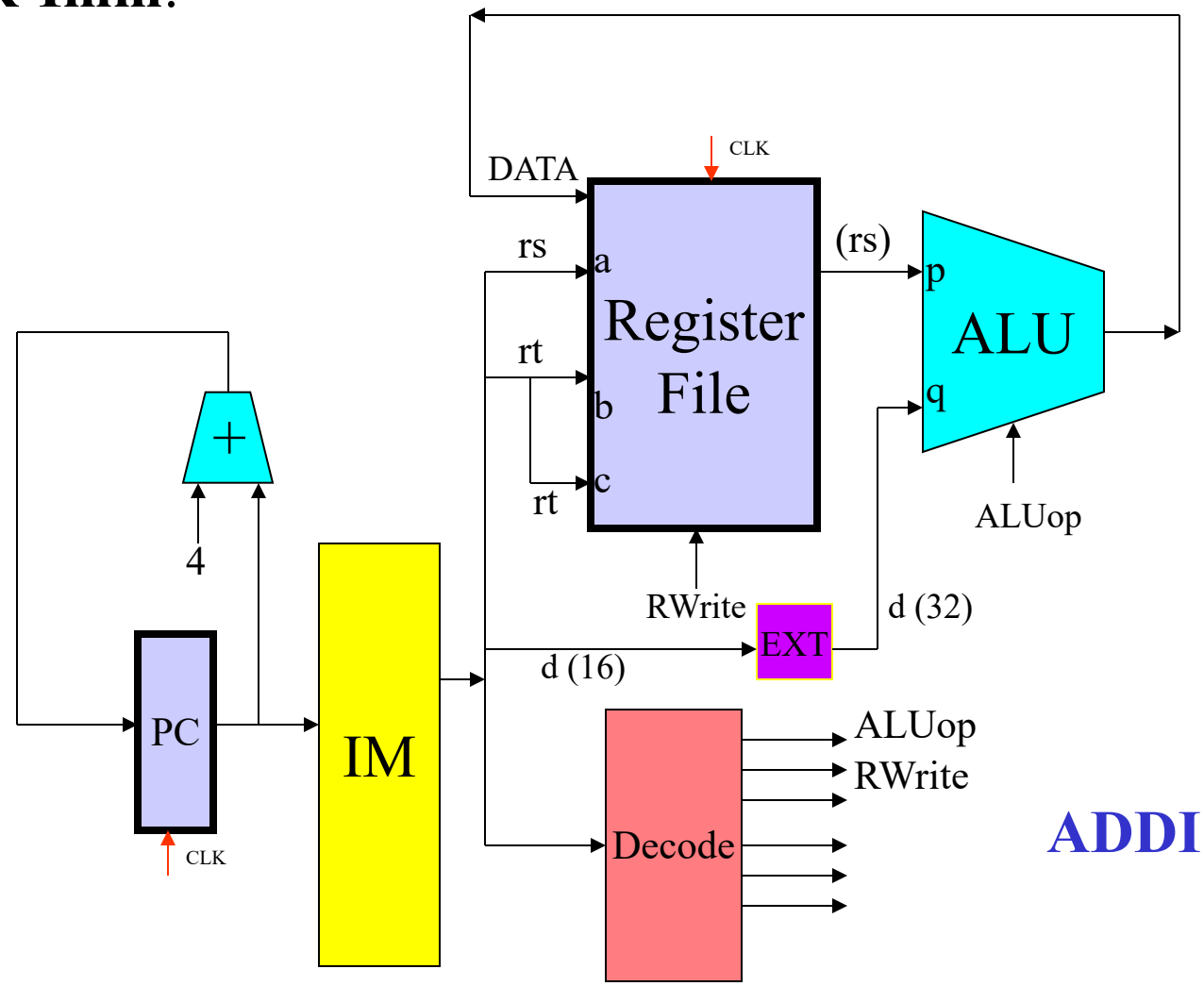


a, b: Read port
c: Write port

ADD rd, rs, rt

Datapath: Register-Immediate

R-Imm:

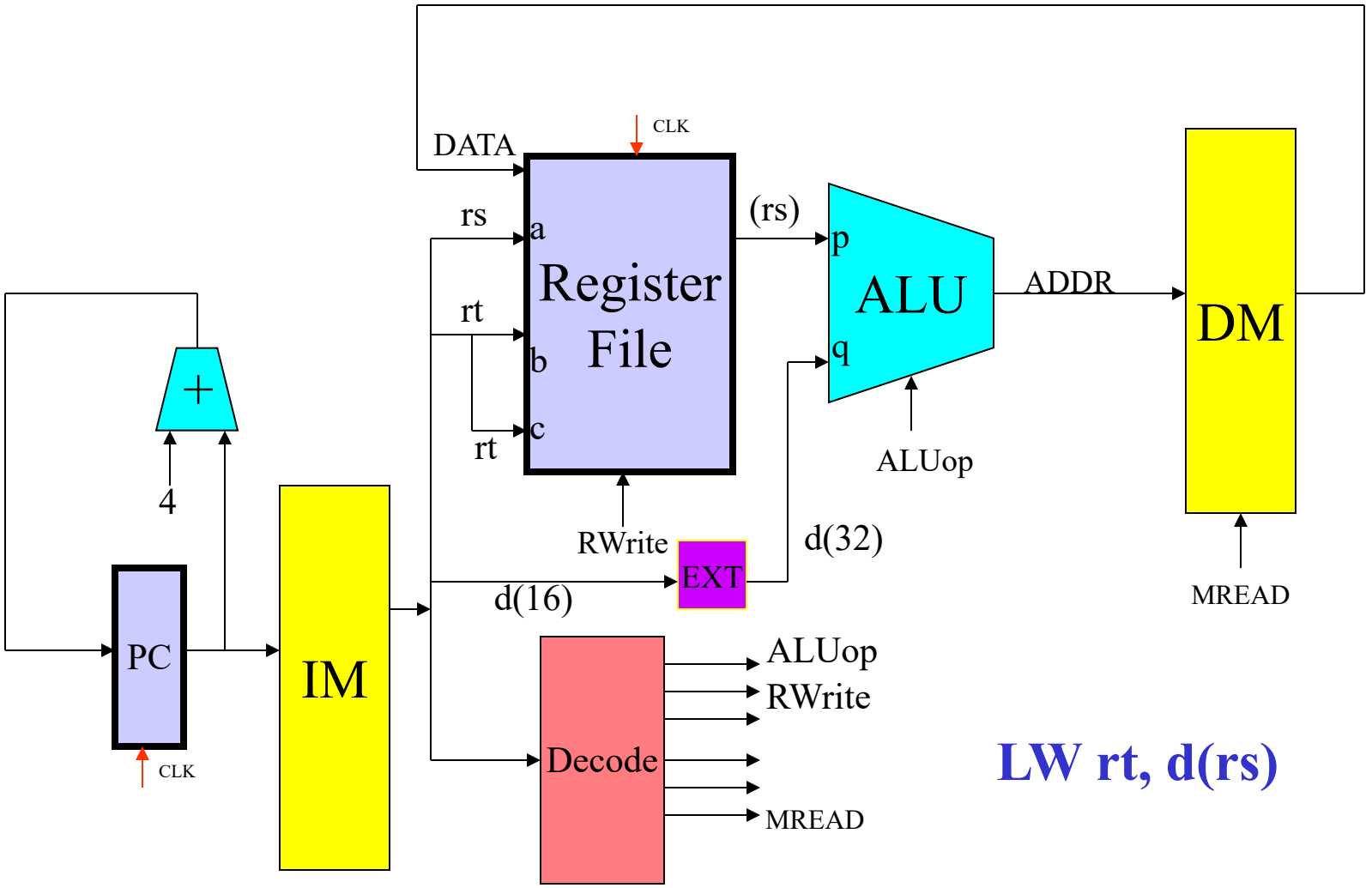


a, b: Read port
c: Write port
d: Imm data

ADDI rt, rs, Imm

Datapaths

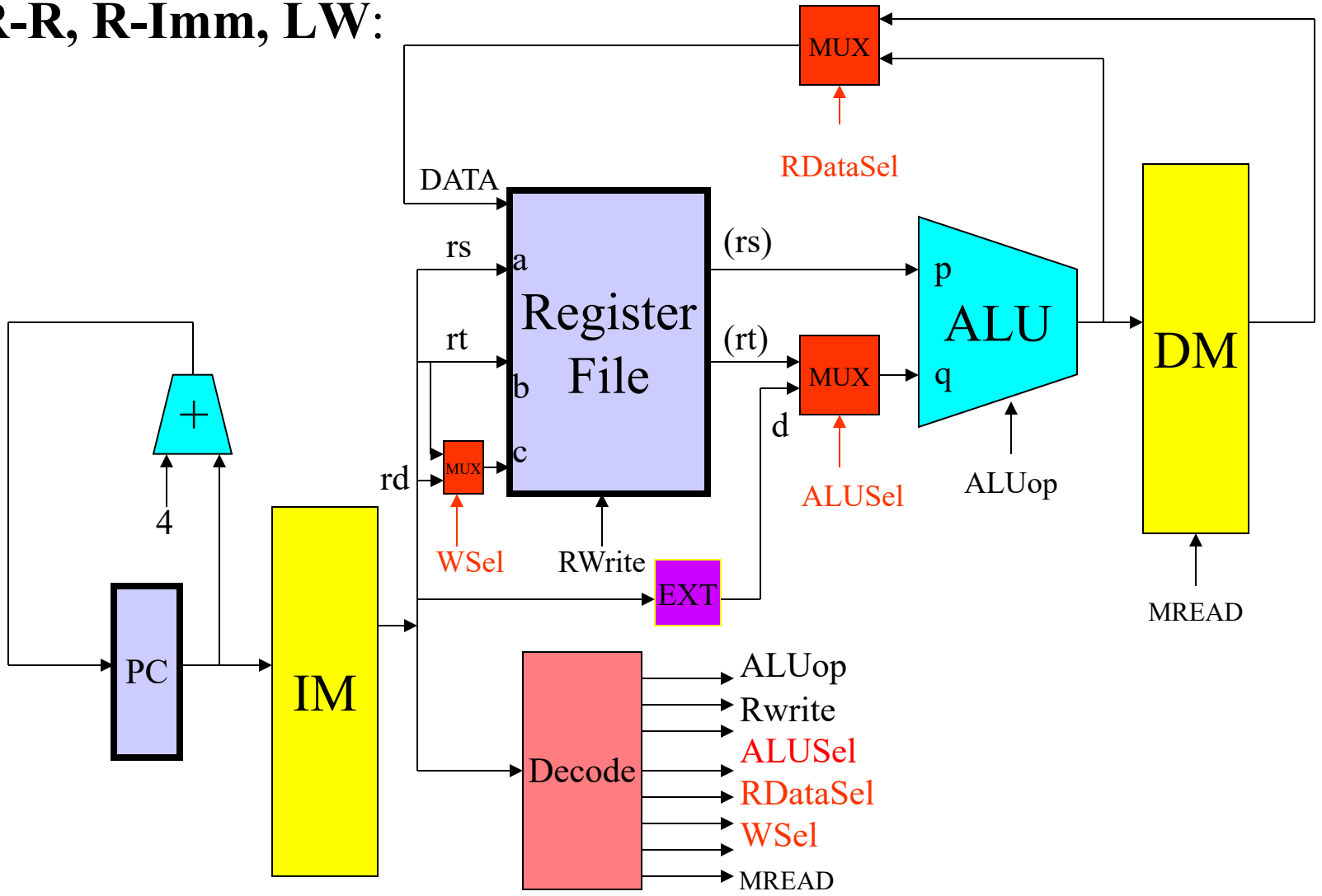
LW Instruction:



LW rt, d(rs)

Datapaths

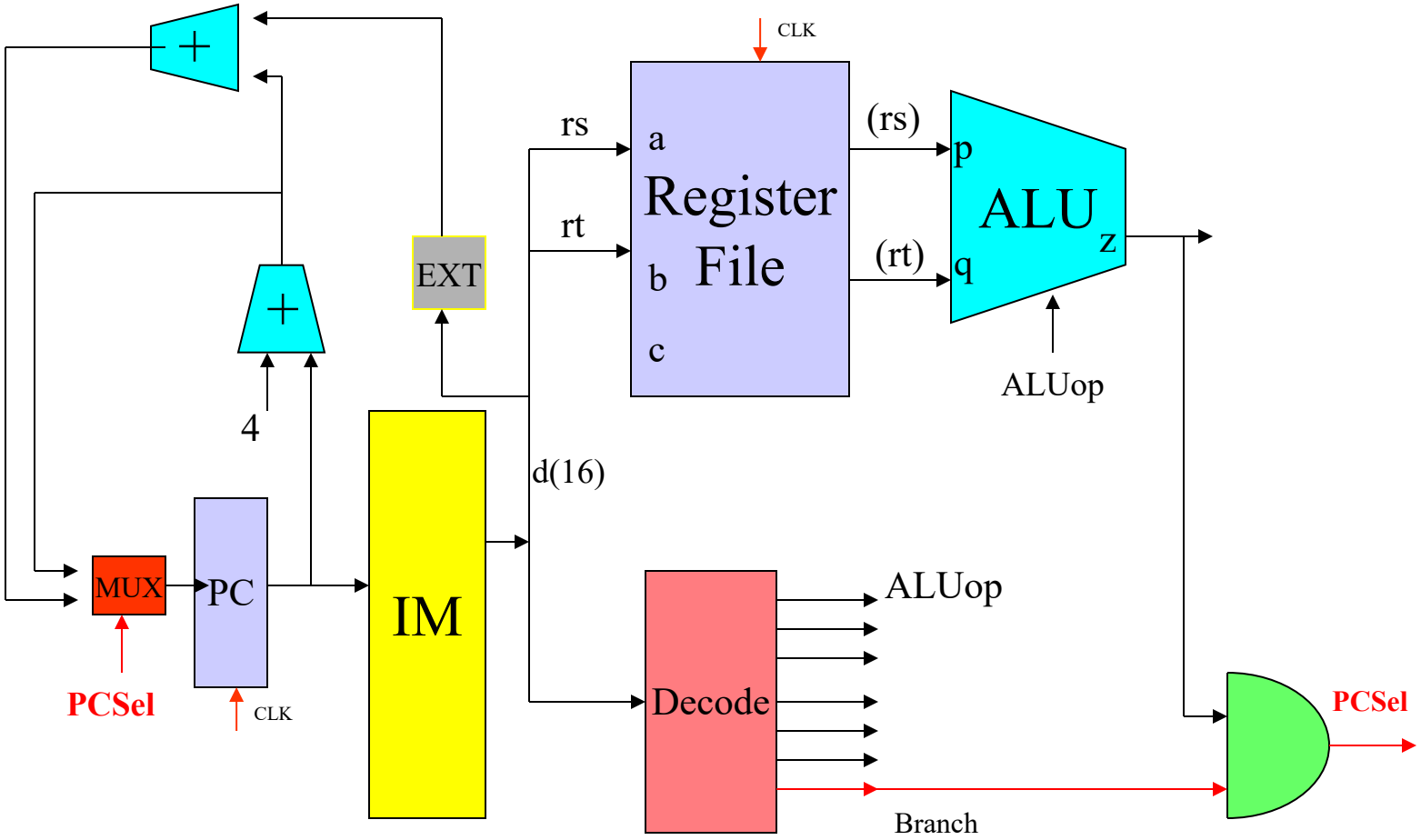
R-R, R-Imm, LW:



Datapaths

BEQ Instruction:

BEQ rs, rt, d



Performance Model

Instruction Throughput (MIPS rating):

- Clock Frequency (cycles/sec) x IPC (instructions/cycle) x 10^{-6}

CPI (Cycles per Instruction)

Measure by counting instruction completion rate

Single-cycle design: Each instruction takes 1 clock cycle (CPI = 1)

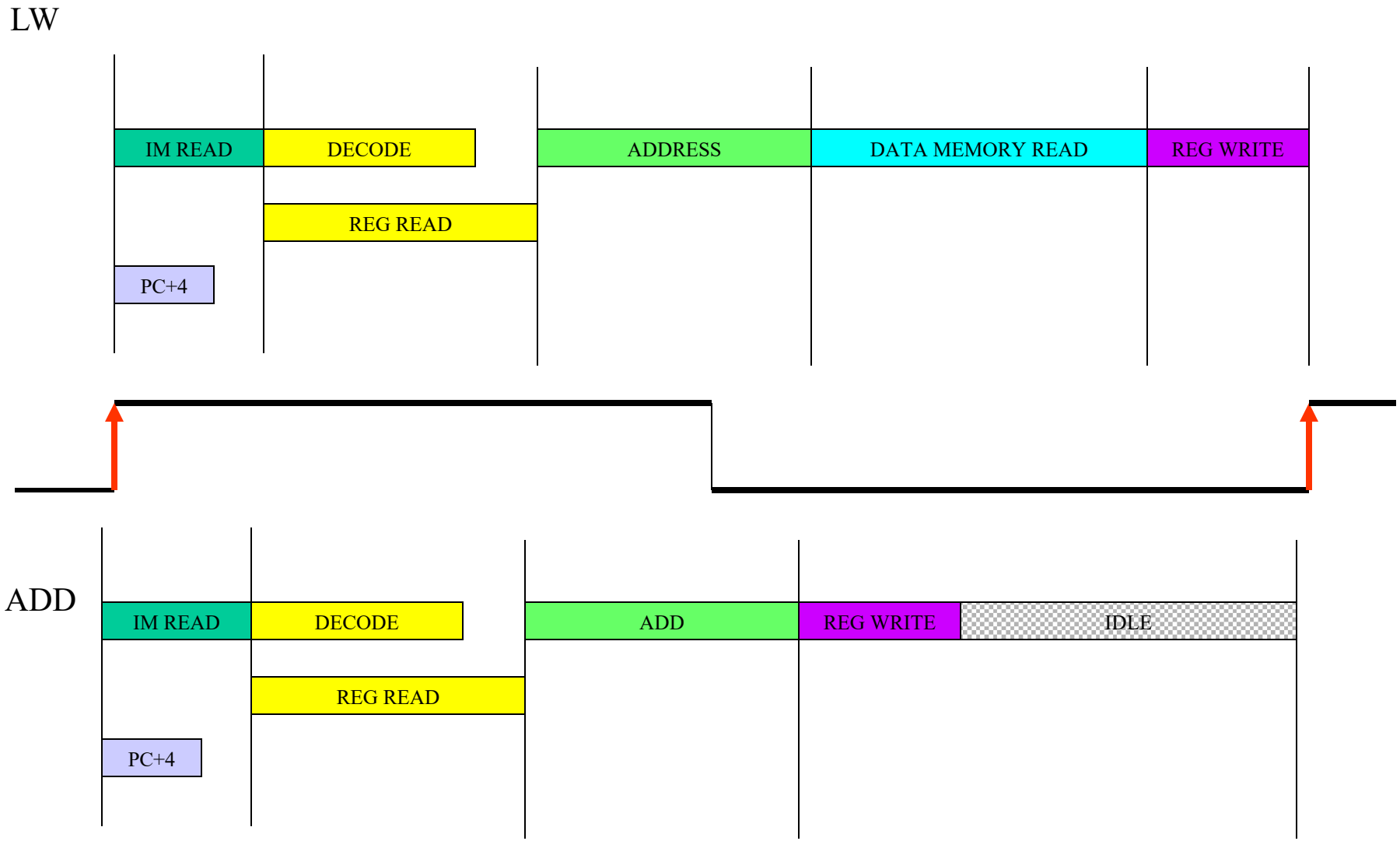
IPC (Instructions/Cycle) = 1/CPI

How do we improve performance?

- Increase Frequency
- Increase IPC
  - Workload to measure IPC?
  - SPEC Benchmarks

Single Cycle Design

- Cycle time determined by longest instruction
- No reuse of Functional Modules (Separate IM and DM, ALU etc)



Blank Slide

Pipelined Processor Design

Instruction execution decomposed into 5 stages

IF: Instruction Fetch

ID/REG: Instruction Decode and Register Read

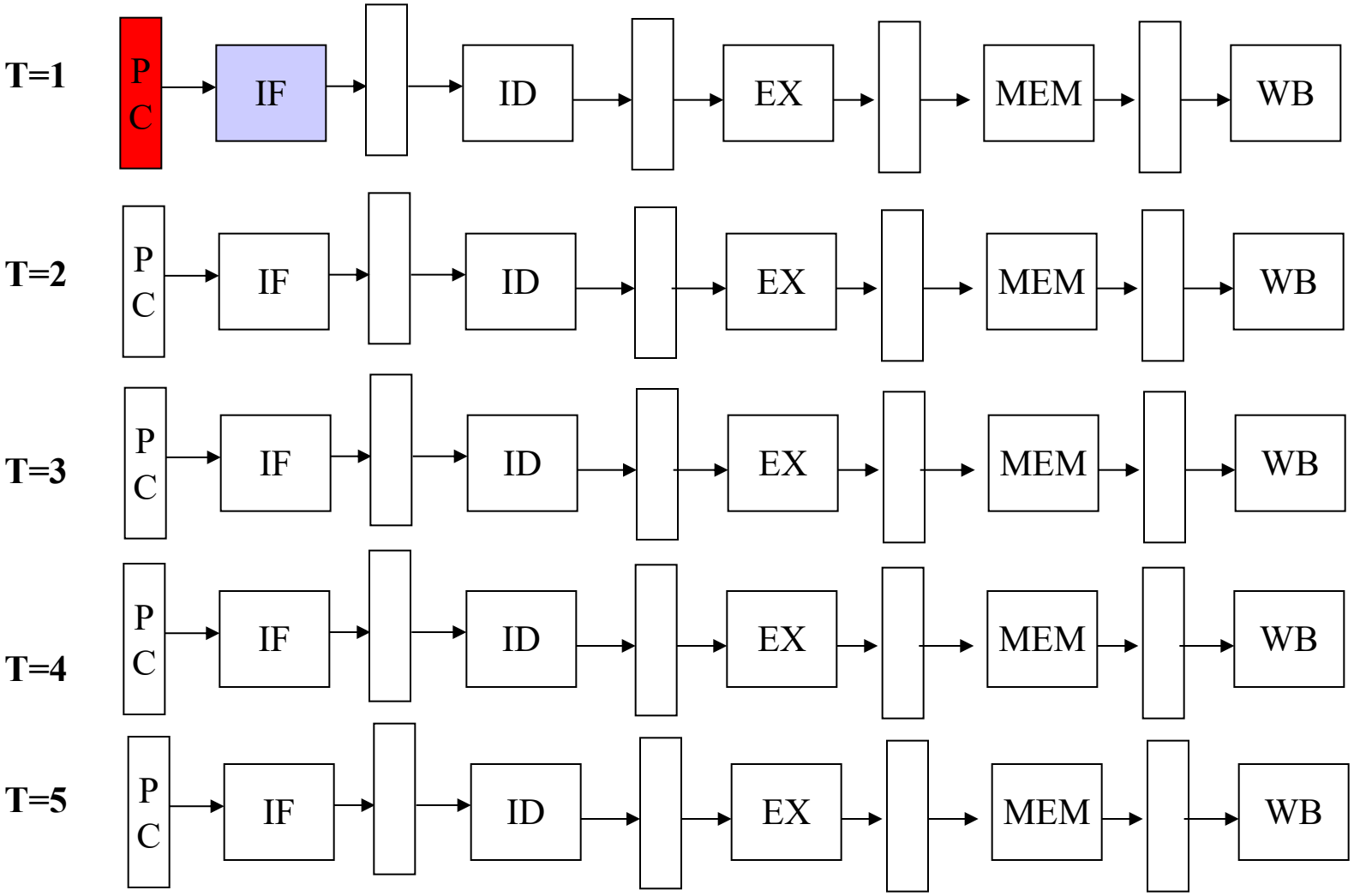
ALU: ALU operation

MEM: Data memory Read or Write

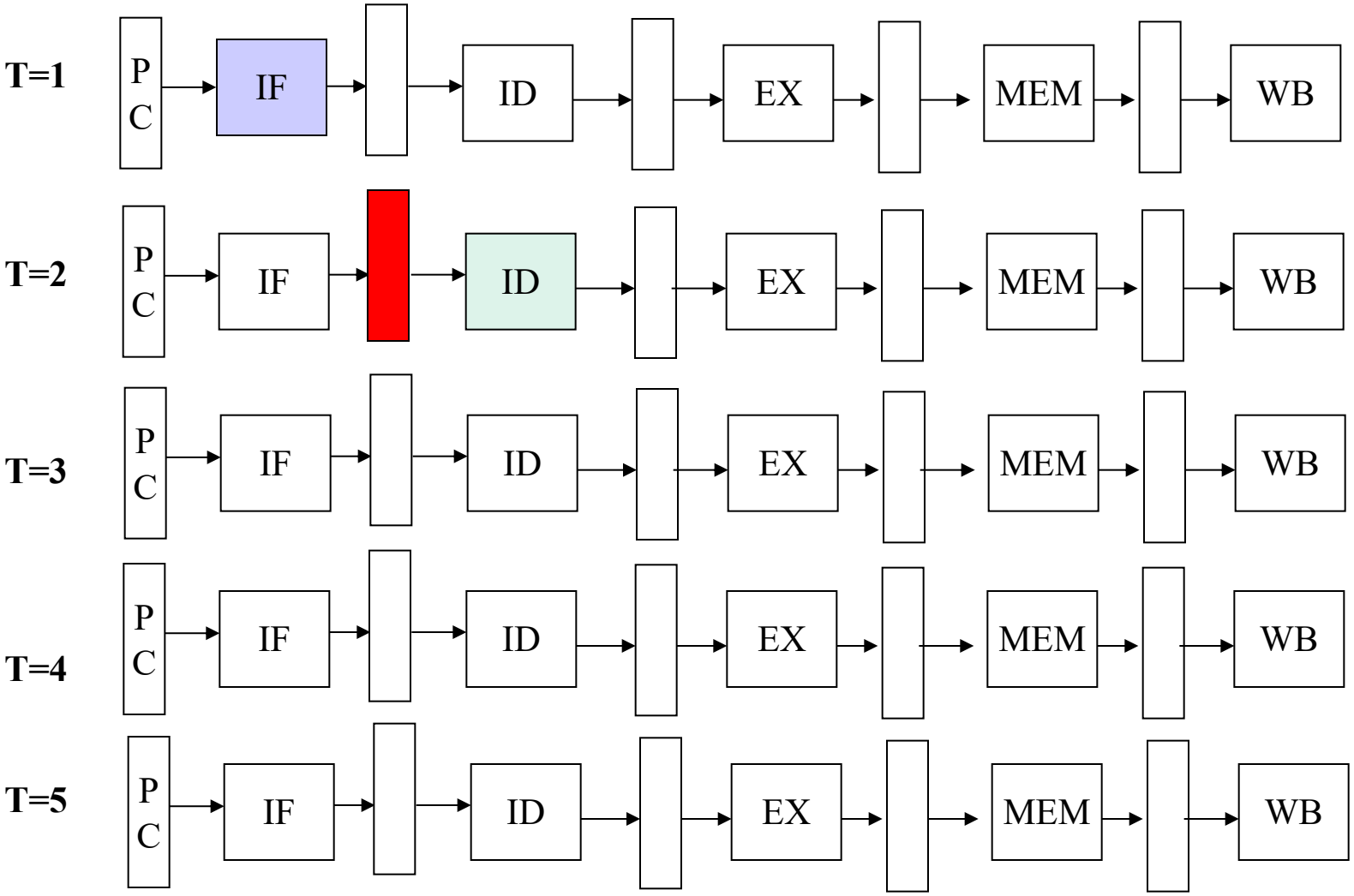
WB: Writeback to Register

- Successive stages of the pipeline separated by Pipeline Registers
- Each stage takes 1 clock cycle
  - Inputs read from the pipeline register on the left
  - Results clocked into pipeline register on the right

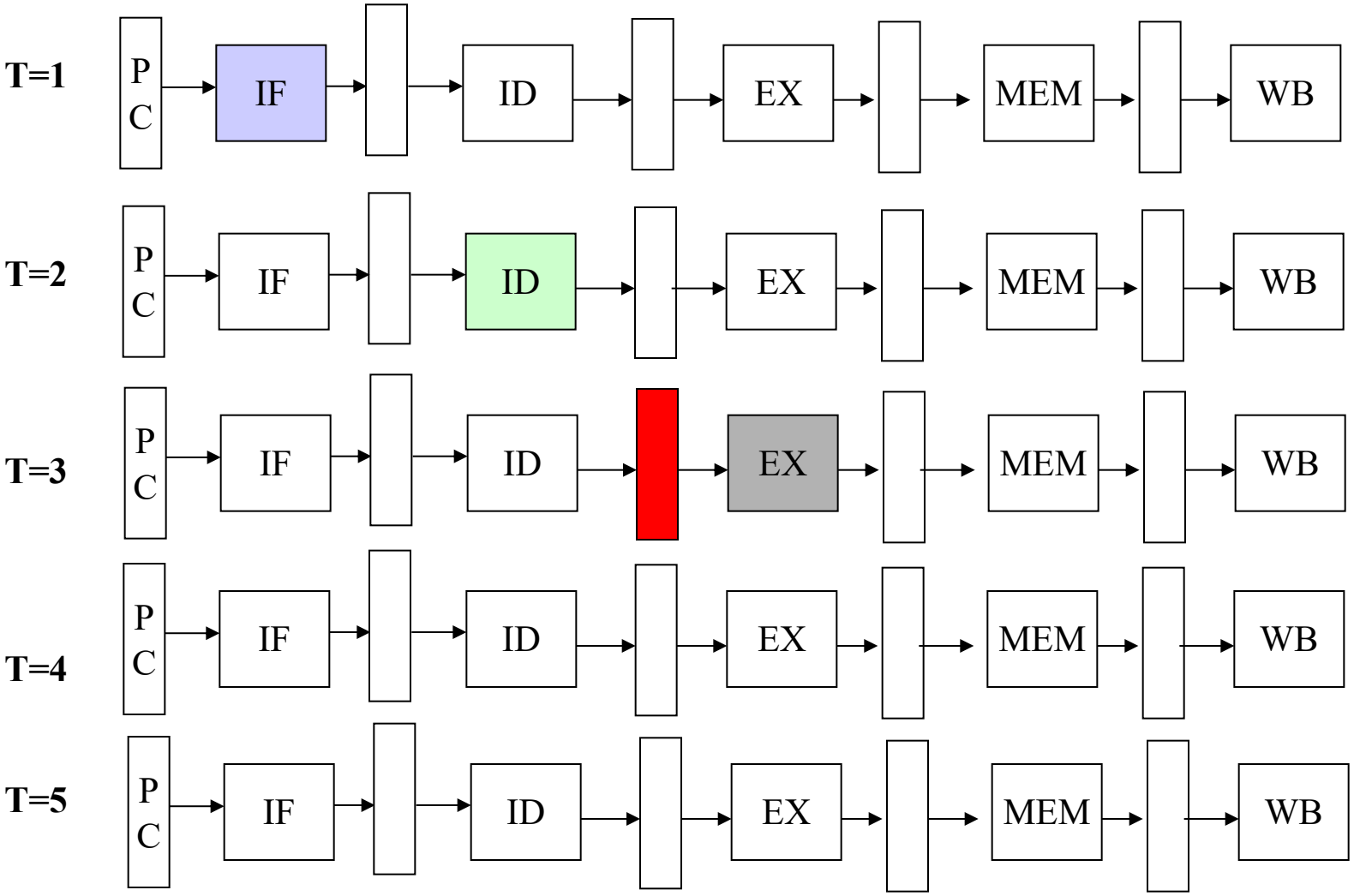
5-Stage Processor Pipeline



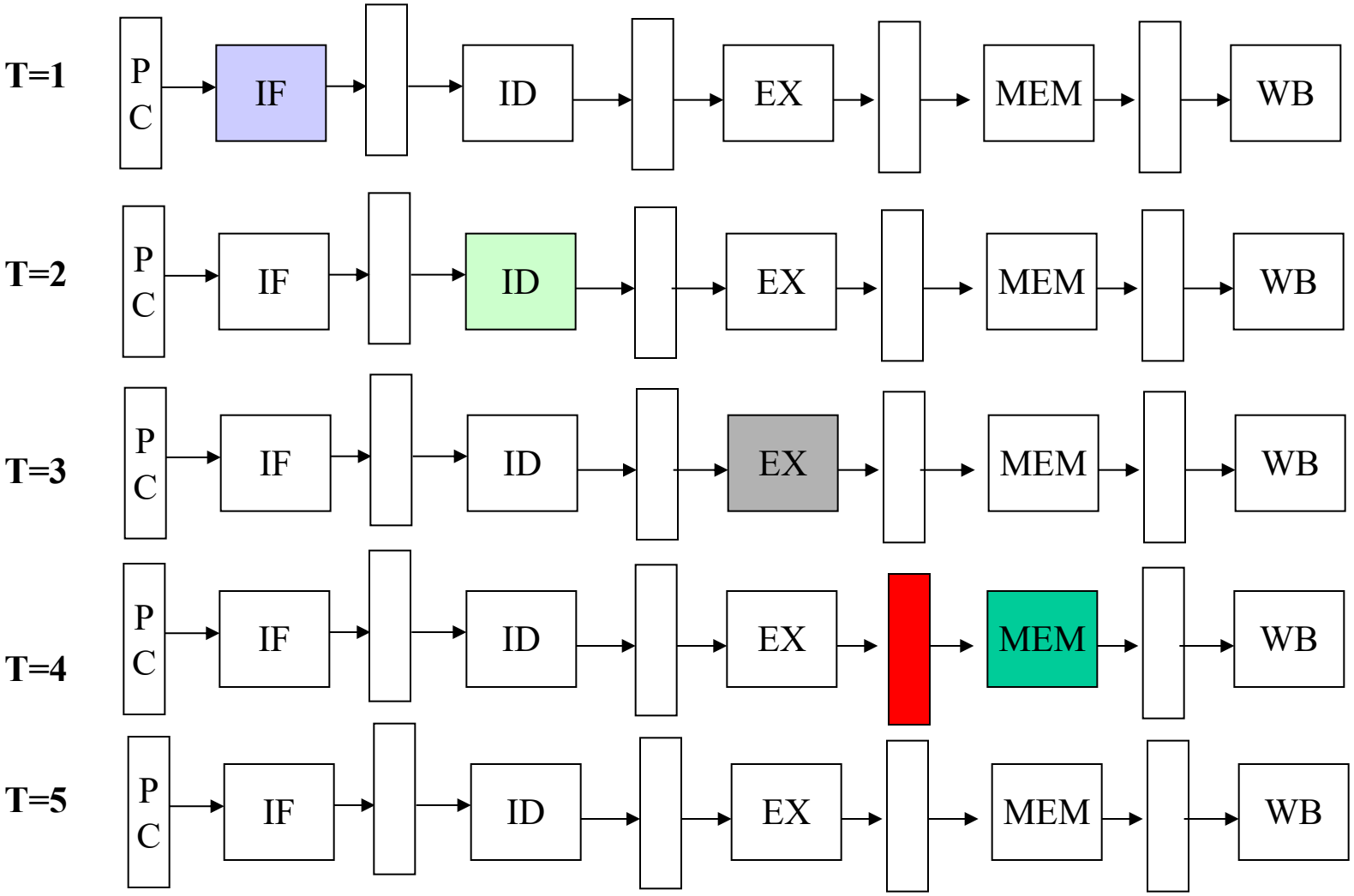
5-Stage Processor Pipeline



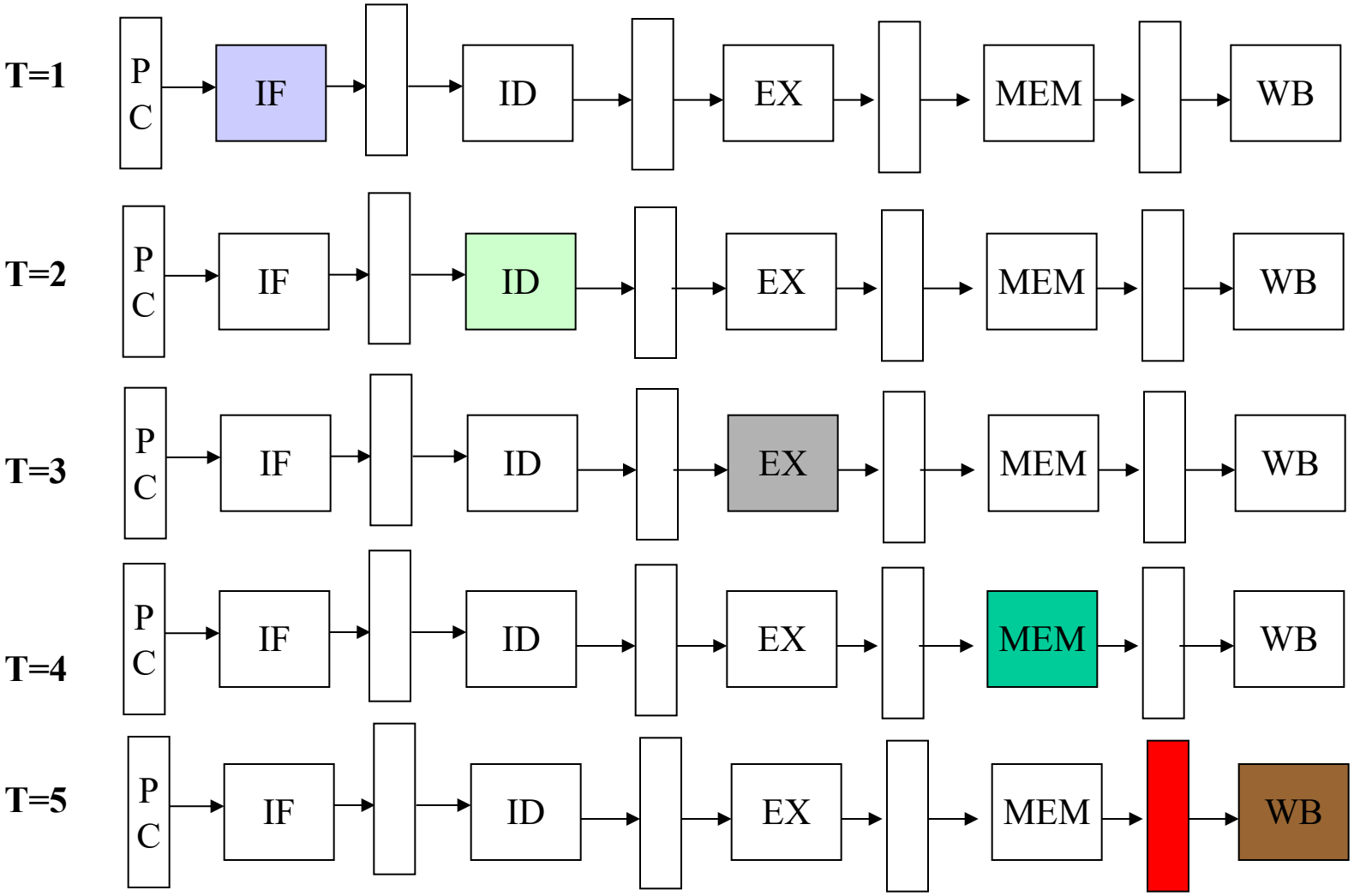
5-Stage Processor Pipeline



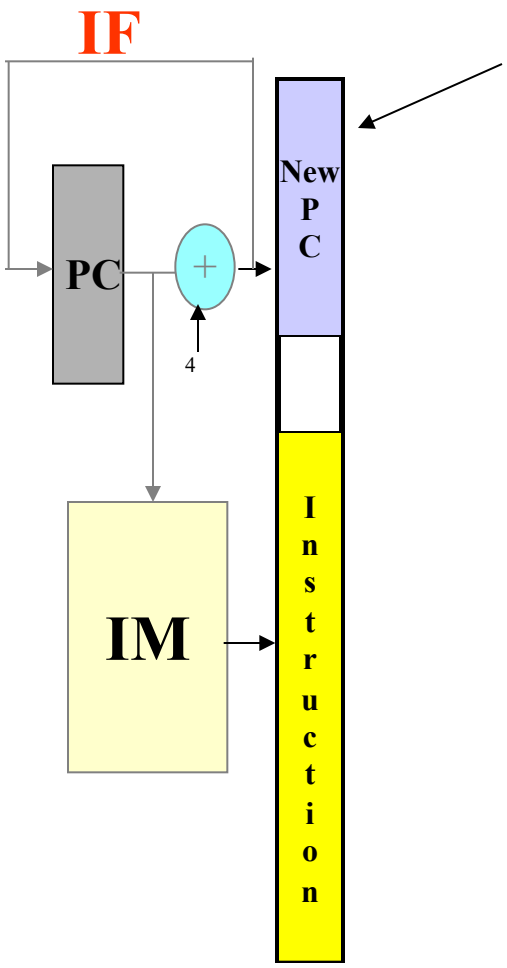
5-Stage Processor Pipeline



5-Stage Processor Pipeline



Execution of LW Instruction: Cycle 1



Pipeline Register

PC had address of current instruction

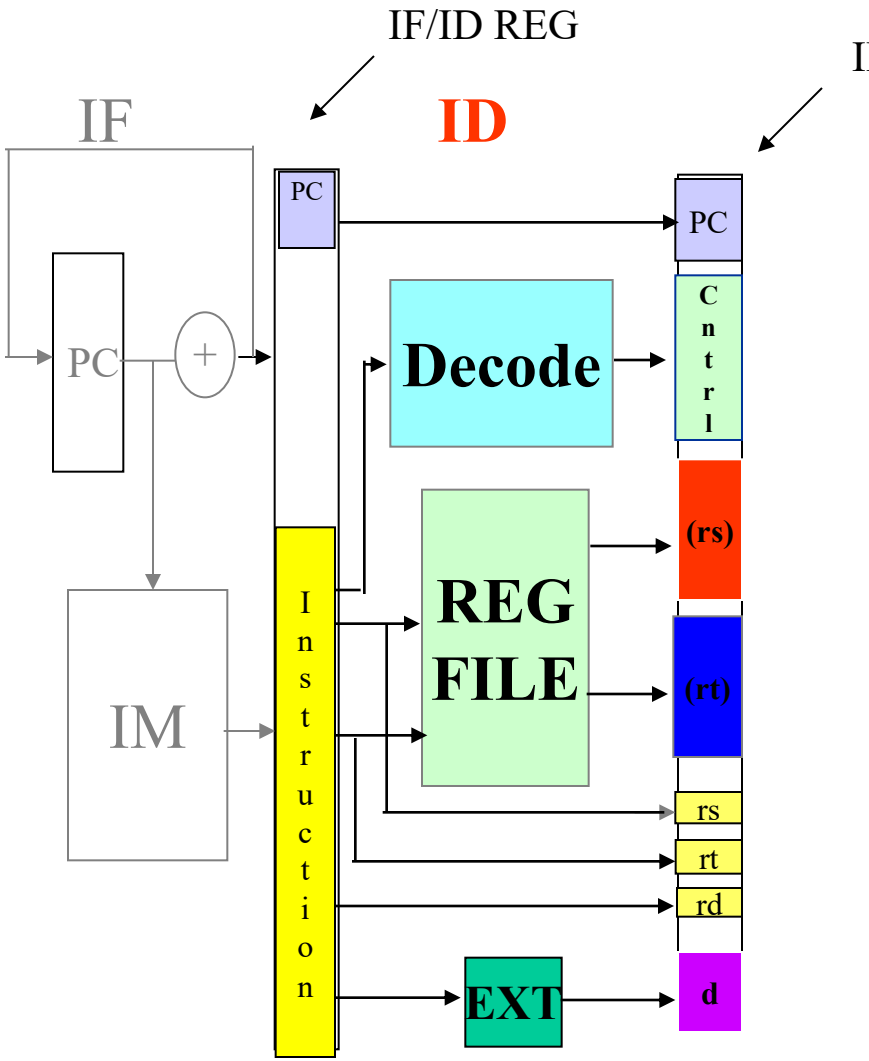
- Read instruction from Instruction Memory (IM)
- Increment PC

At clock edge:

Write updated PC value into PC register and into Pipeline Register

Write the instruction into the Pipeline Register

Execution of LW Instruction: Cycle 2



IF/ID REG had current instruction

- Decode instruction
- Read registers

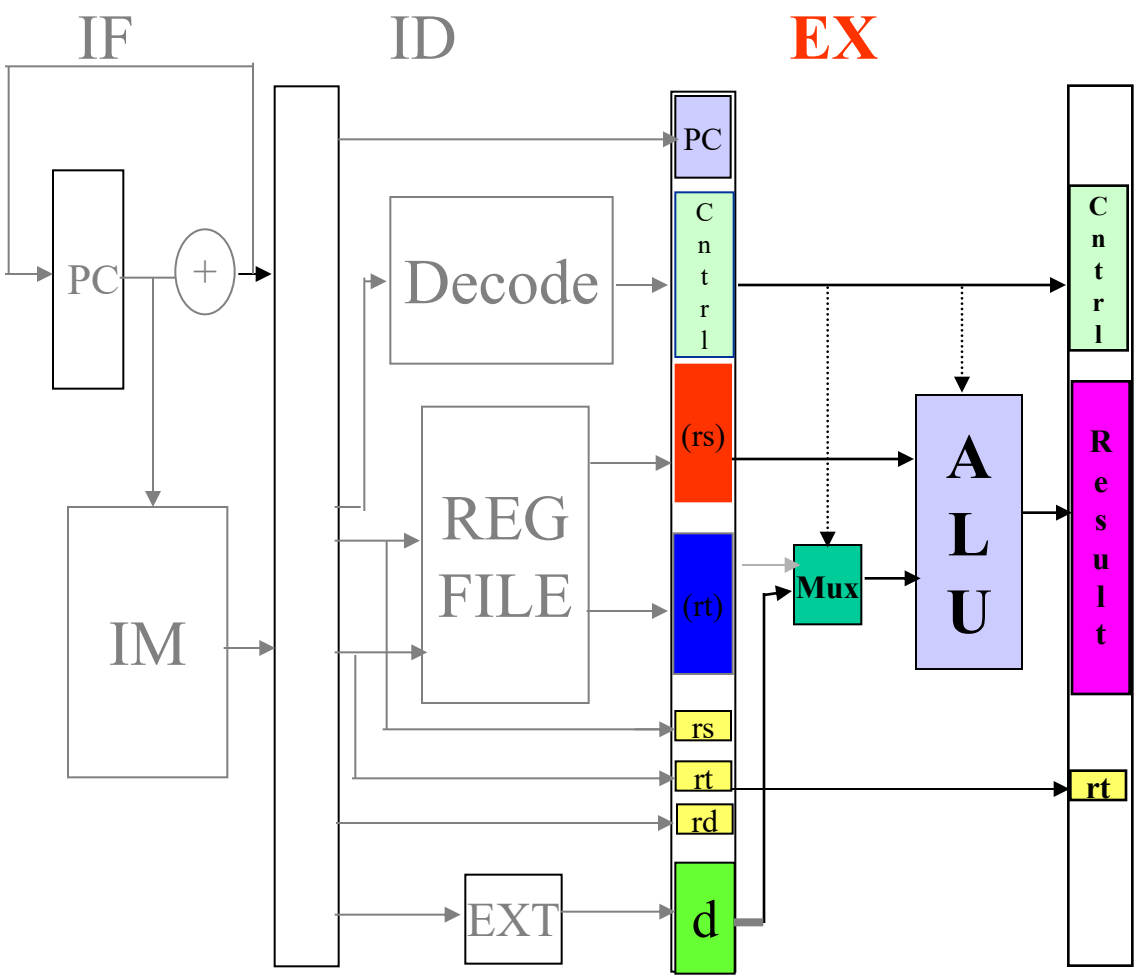
At clock edge:

Write decoded control signals and selecgted fields of instruction into ID/EX Pipeline Register

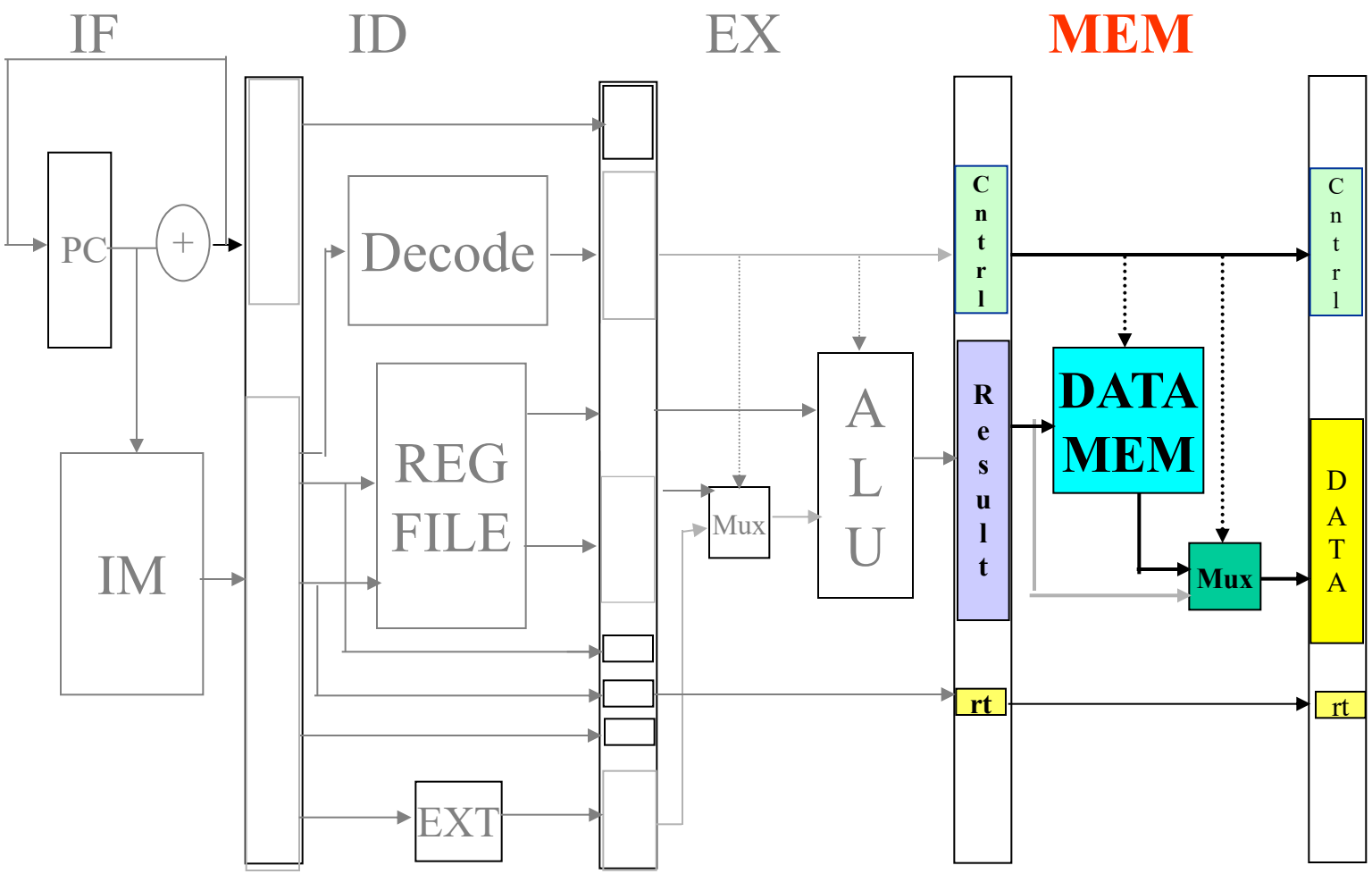
Write register values rs and rt into ID/EX Pipeline Register

Transfer PC value from IF/ID register to ID/EX register

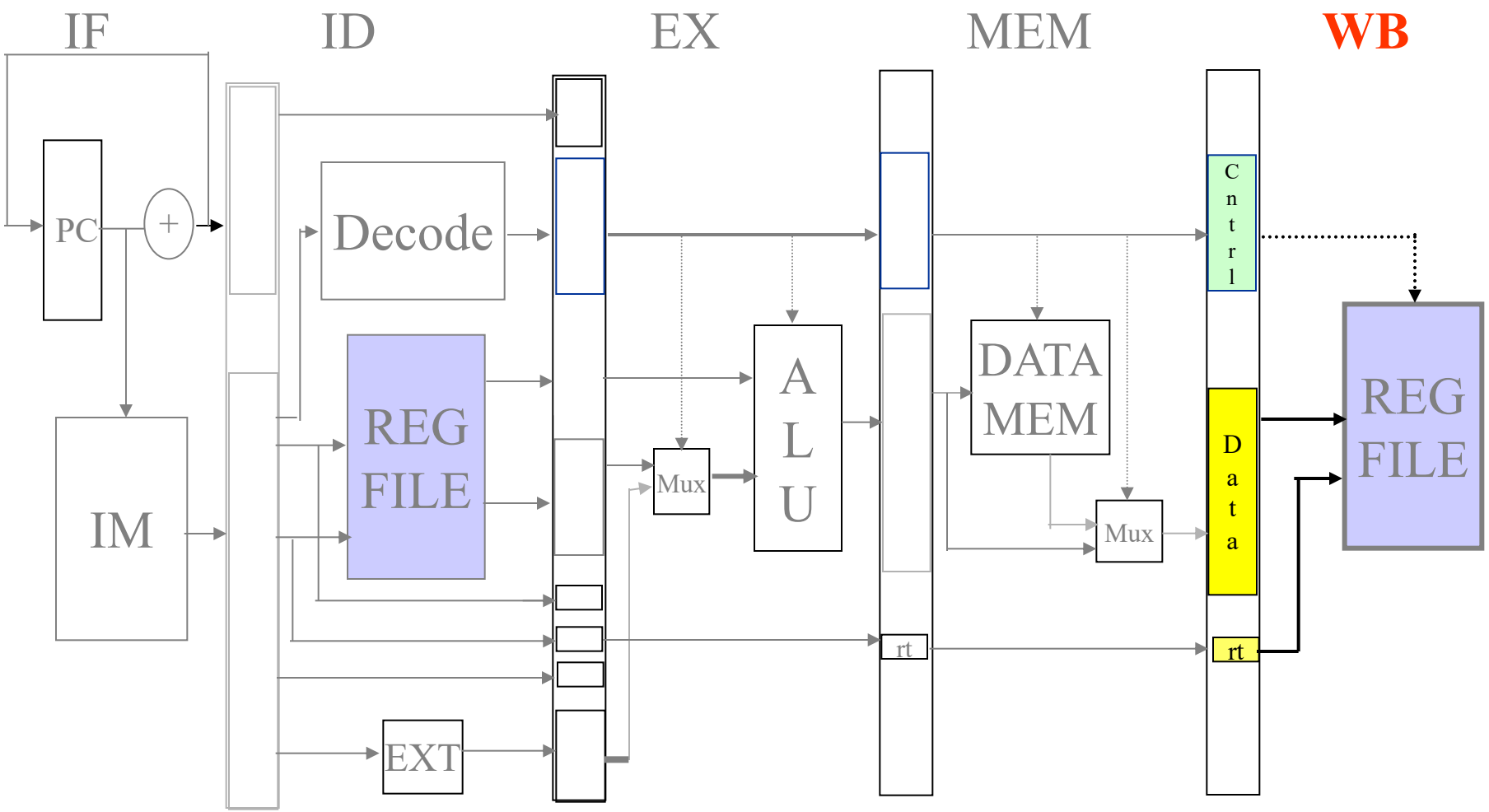
Execution of LW Instruction: Cycle 3



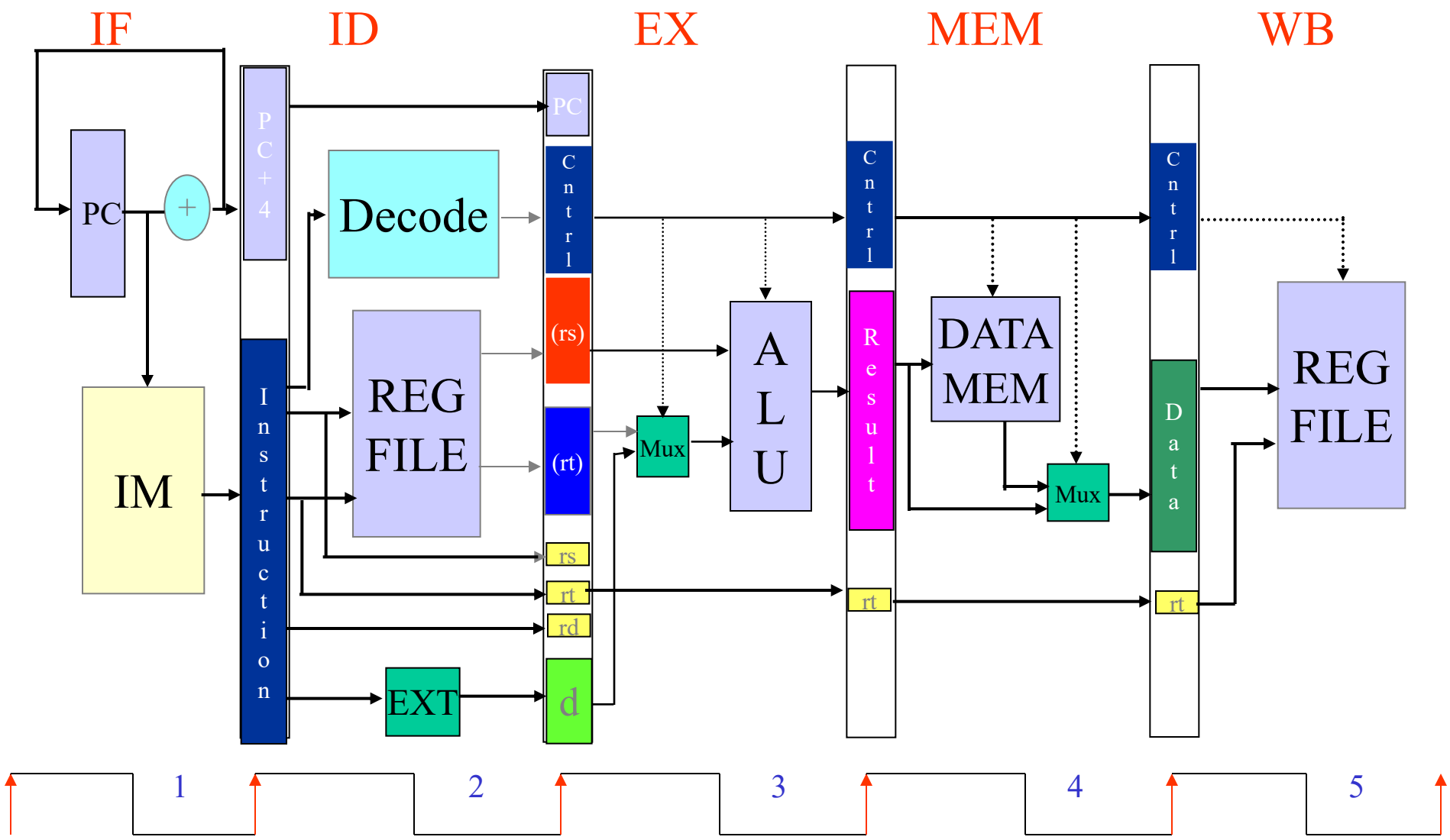
Execution of LW Instruction: Cycle 4



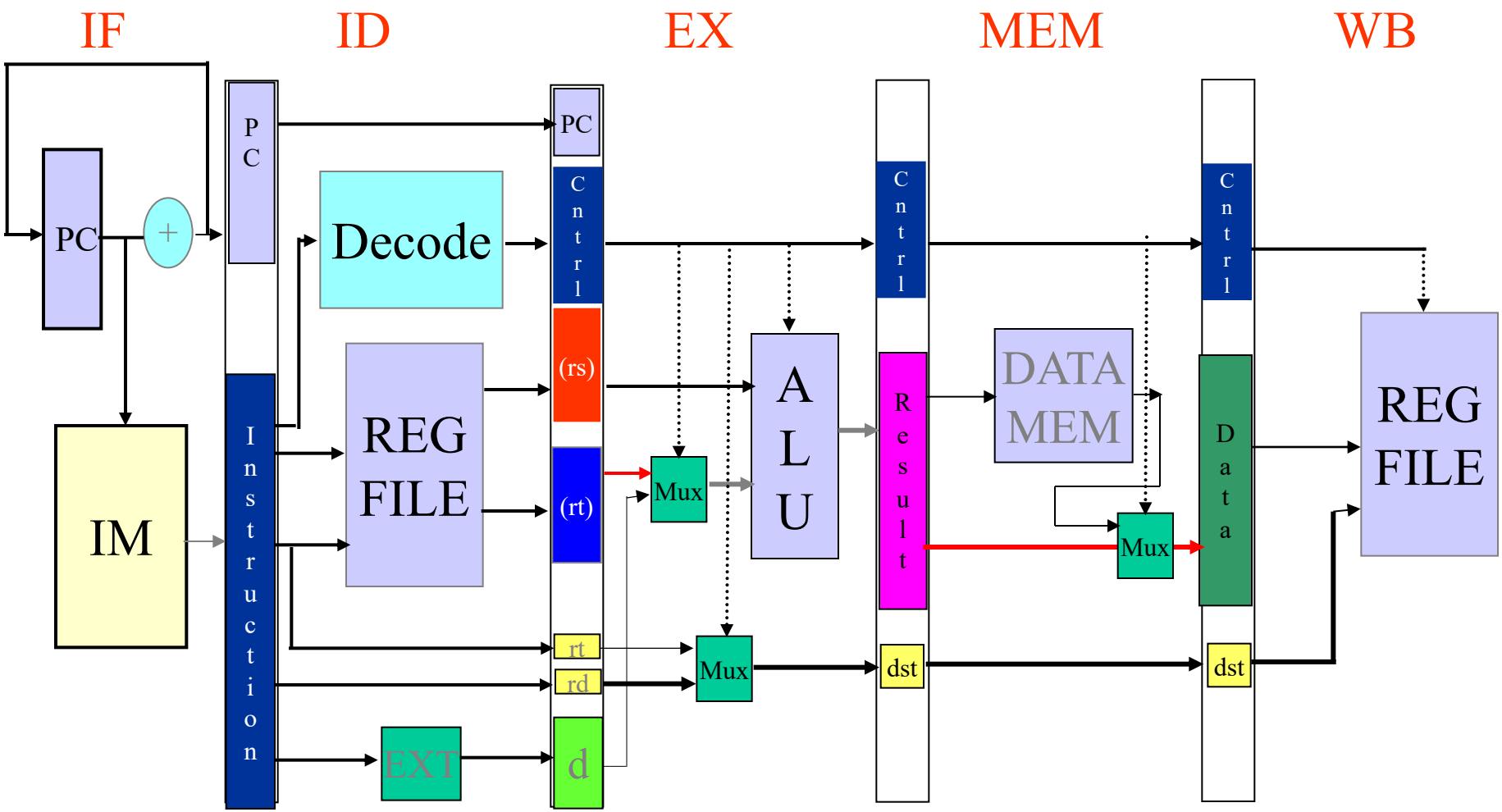
Execution of LW Instruction: Cycle 5



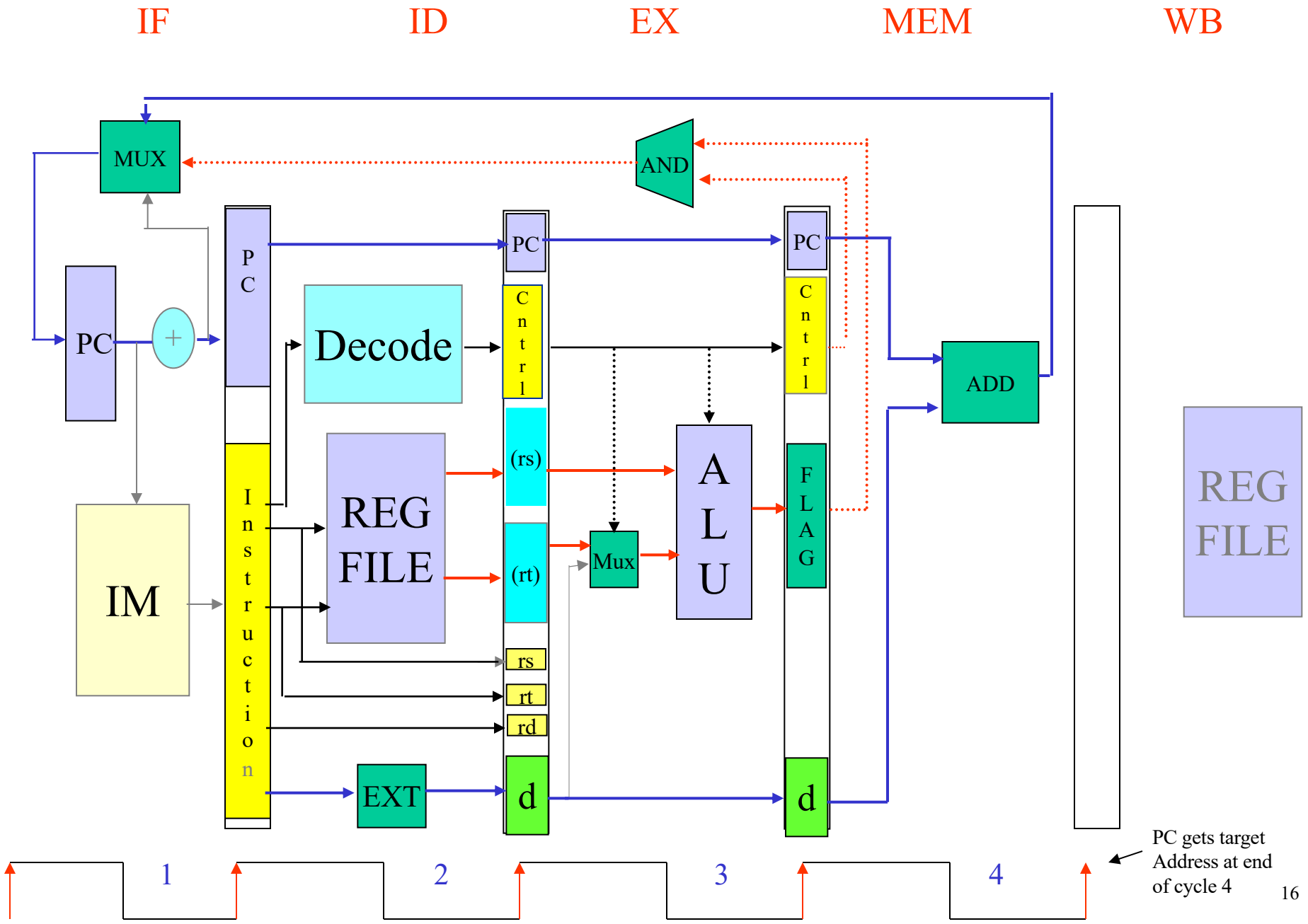
Execution of Load Instruction: LW rt, d(rs)



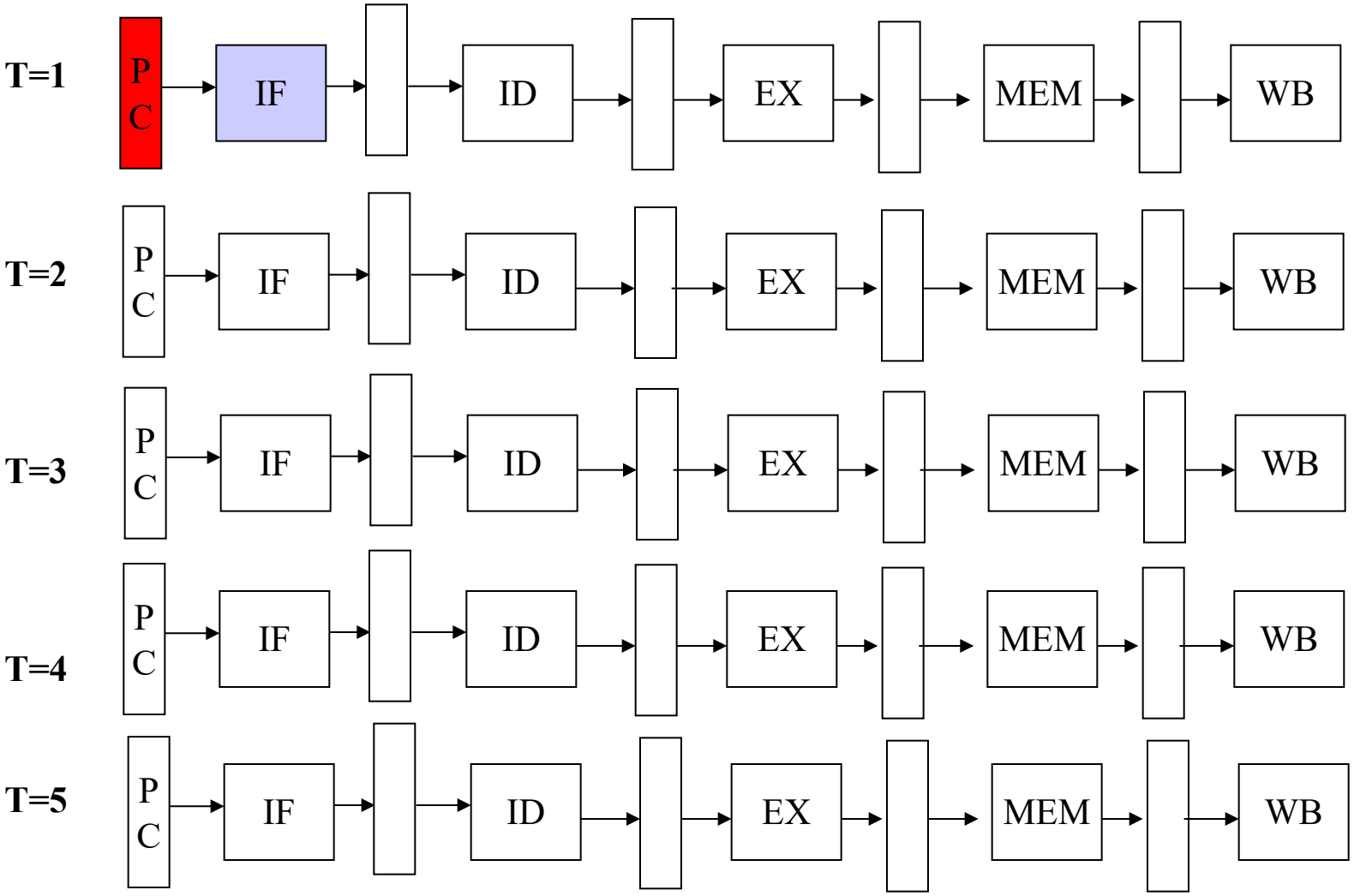
Execution of R-R Instruction: **ADD rd, rt, rs**



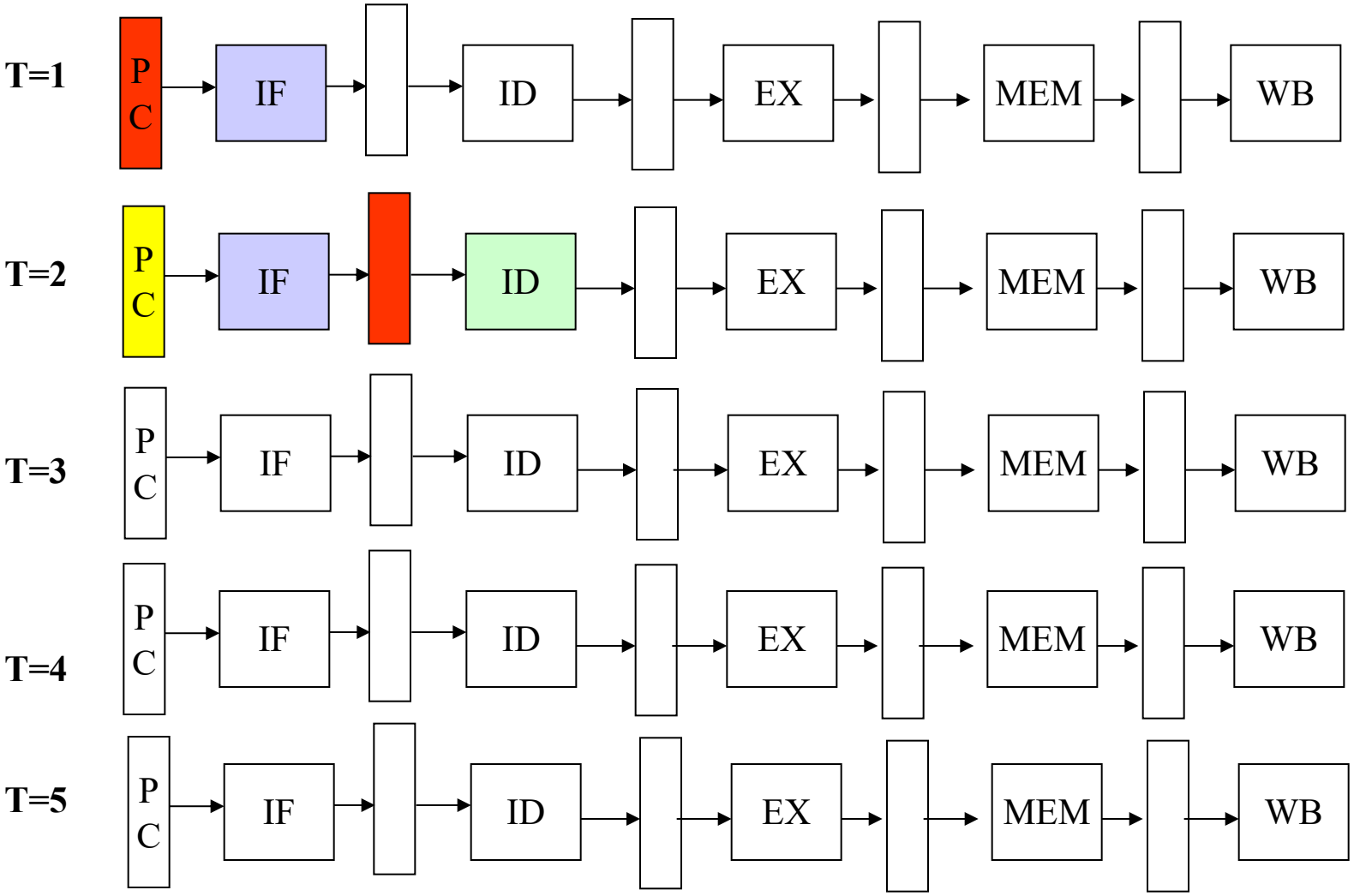
Lazy Execution of Branch Equal Instruction: **BEQ rt, rs, d**



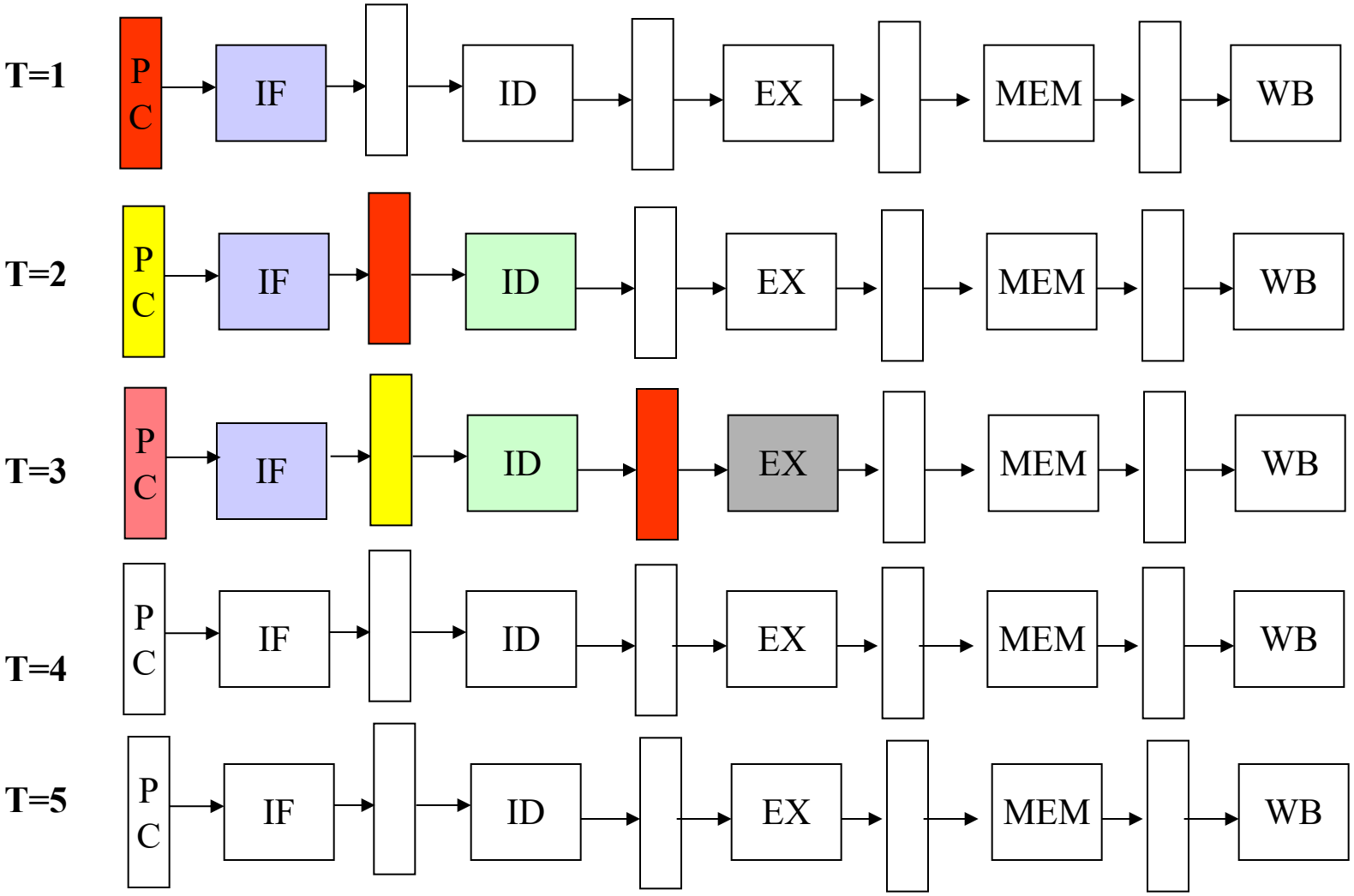
5-Stage Processor Pipeline



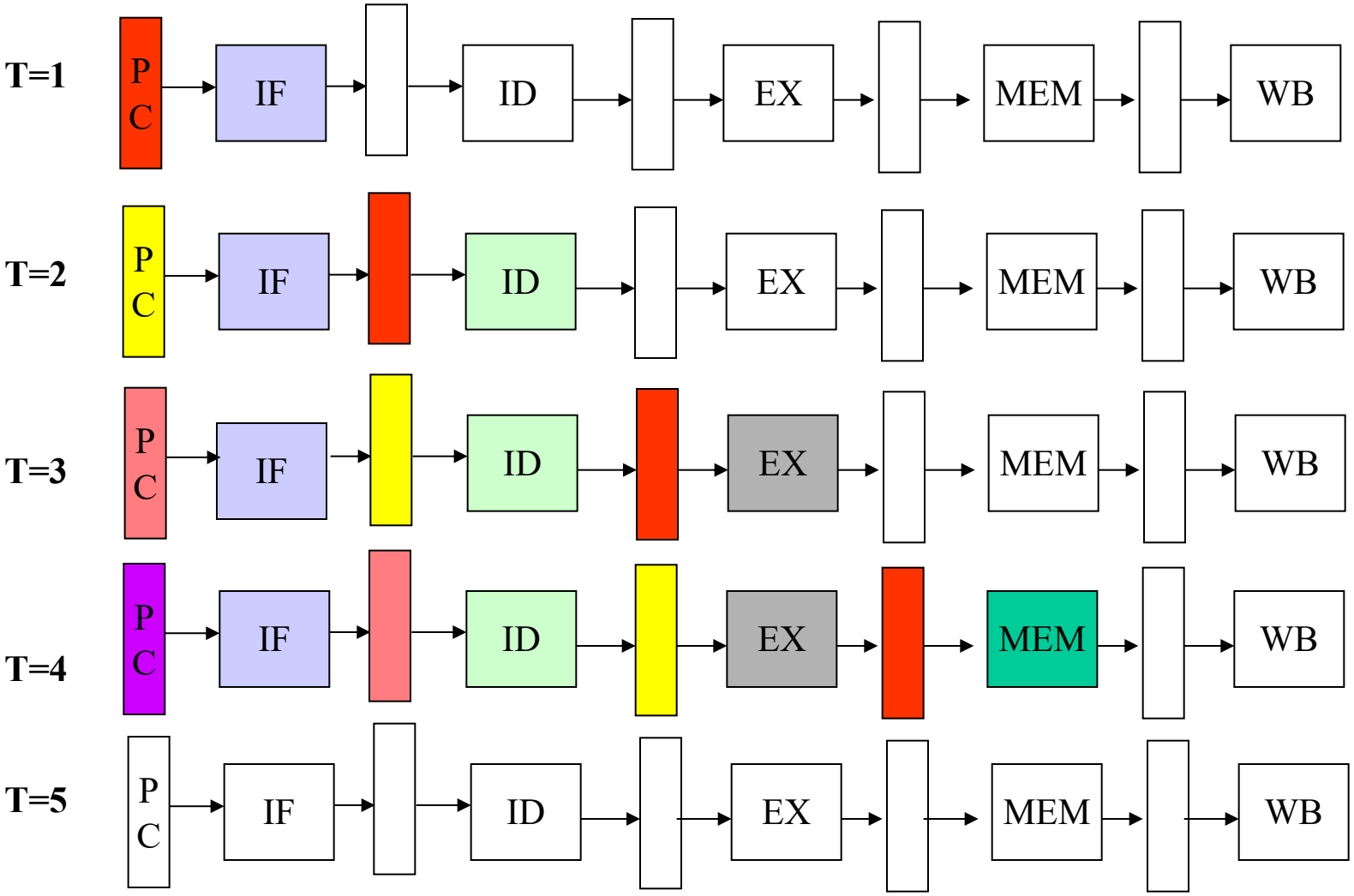
5-Stage Processor Pipeline



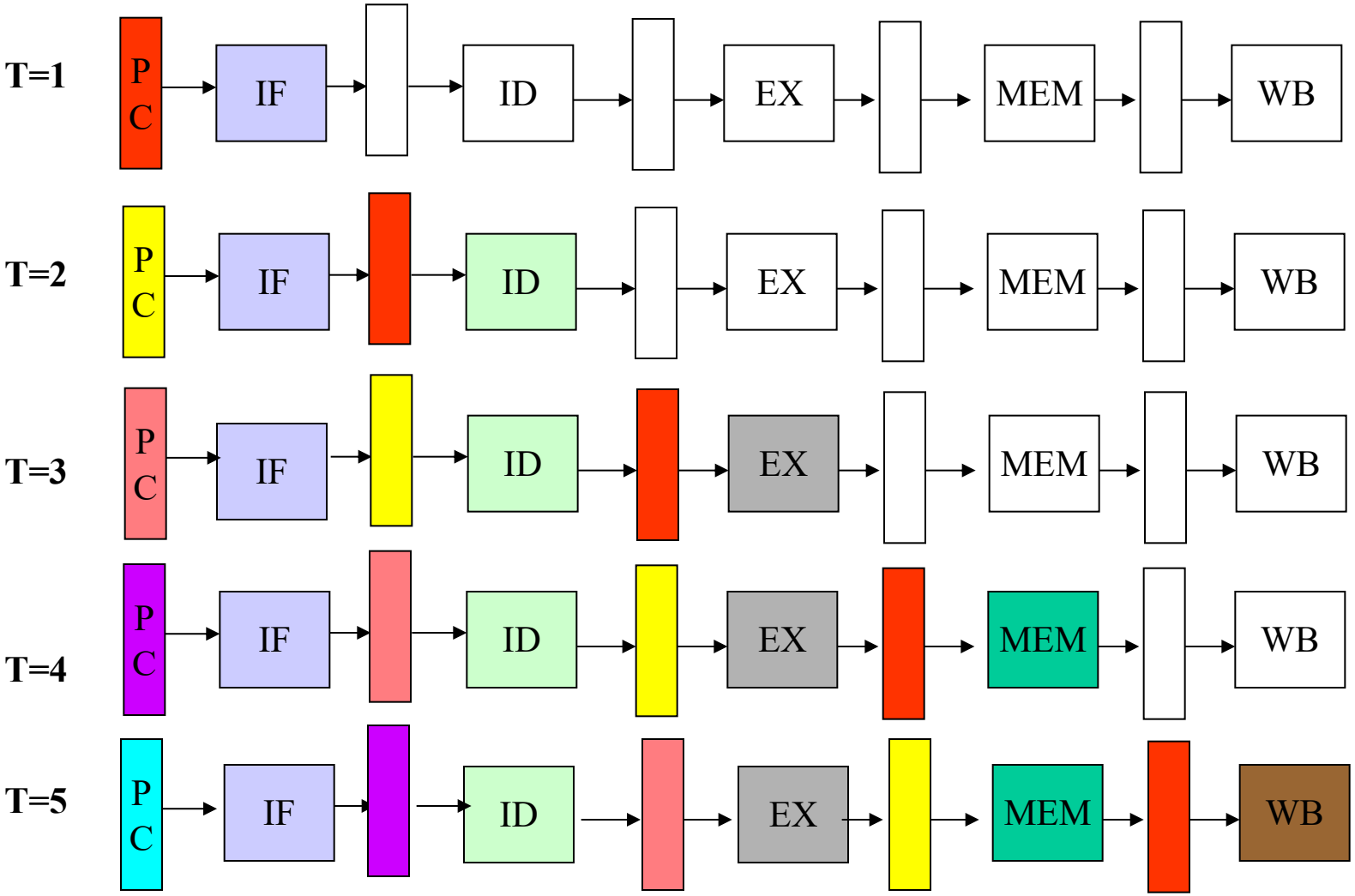
5-Stage Processor Pipeline



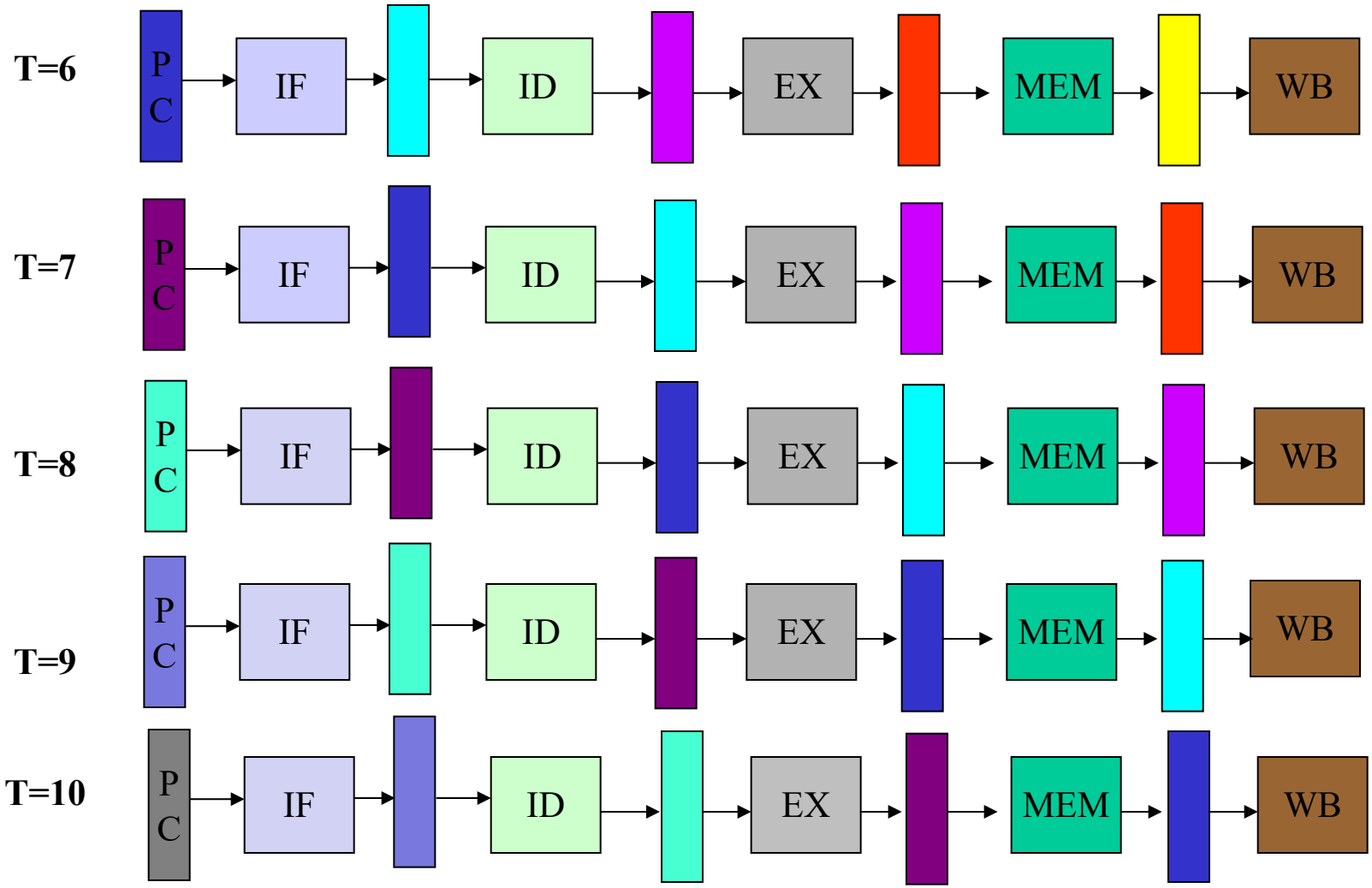
5-Stage Processor Pipeline



5-Stage Processor Pipeline



5-Stage Processor Pipeline



Instruction 6 will finish at cycle 6 + 5 - 1 = 10 Instruction n at cycle: n +4

Ideal Pipeline Summary

MIPS = IPC x Clock Frequency (Hz) x 10^{-6}

CPI = n+4 (cycles) / n (instructions) = 1 + 4/n

Approximately CPI = 1 for large n with no stalls

Clock frequency : Roughly 5 times that of base design

- Limited by the delay through the pipeline stage
  - Faster circuitry
  - Less circuitry with more stages
  - Will be ultimately limited by pipeline register delays
    - In practice useful functions require at least 10 gate delays

- Deep pipelines with very thin stages
  - Clock rate (frequency) increase
  - Instruction throughput (MIPS) increases in ideal pipeline
  - Single instruction latency increase
  - May cause greater number of stall cycles with data and control hazards

Blank Slide

Pipeline Hazards

- What factors force deviations from this ideal pipeline?

- Hazard
  - Condition that disrupts the orderly flow of instructions
  - Requires special attention by hardware and/or software to maintain program correctness and performance

1. Structural Hazards

   Contention for hardware resources

2. Data Hazards

   Data dependencies between instructions

3. Control Hazards

   Disruptions caused by program control flow

Structural Hazards

Contention for hardware resources

- Memory Contention

  Instructions in IF and MEM stages may contend for memory
  We will assume separate instruction and data memories to avoid conflict

- ALU Contention

  Only 1 instruction doing an ALU operation at any time

  No contention in this pipeline

- Register File (RF) Contention
  - Instruction writes to RF while other instruction is reading RF

Memory Contention

Instructions in IF and MEM stages may contend for memory

Example

10% of instructions are LOADS and 5% are STORES.

A single-ported unified Instruction and Data Memory.

What is the effective CPI?

Each LOAD and STORE will add 1 stall cycle to the pipeline

CPI = 1.0 (nominal CPI) + 10% x 1 + 5% x 1 = 1.15

We will assume separate instruction and data memories to avoid conflict

Memory Contention

Example 1: 1% instructions encounter a Cache Miss in the IM.

Miss Penalty 100 cycles. What is the effective CPI?

Each IM cache miss adds 100 cycle stall. CPI = 1.0 + 1% x 100 = 2.0

Example 2 10% of instructions are LOADS and have a Data Miss Rate of 20% in DM. What is the effective CPI?

Worst Case: Miss in the IM and DM never occur together

A data miss adds 100 additional stall cycles i.e 10% x 20% x 100 cycles = 2 cycles

CPI = 2.0 + 2.0 = 4.0

Best Case: Assuming instruction and data misses can be served concurrently

All instructions misses (1%) overlap with a data miss (2%): All instruction miss stall times subsumed completely by data miss stalls.

Stall cycles is only for the data miss stalls: 2 cycles

CPI = 1.0 + 2.0 = 3.0