# Computer Architecture

# Instruction Pipelining

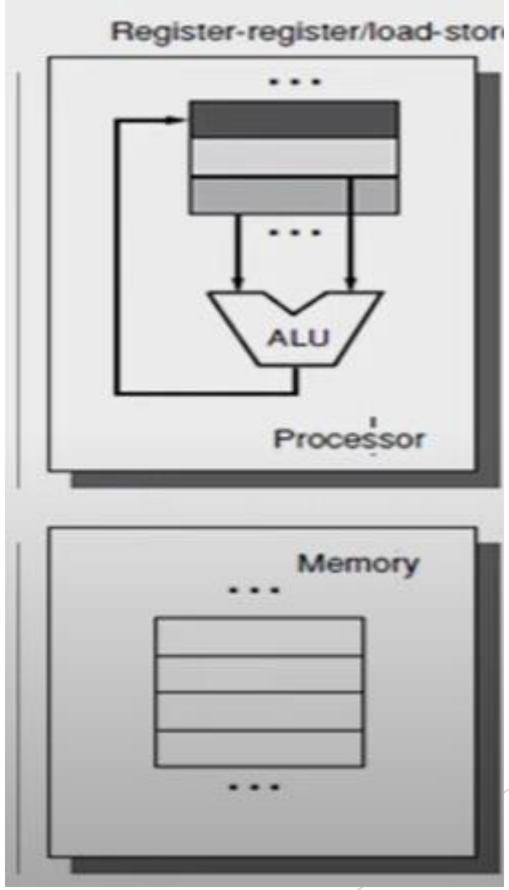
Dr. Mohammad Reza Selim

# Lecture Outline

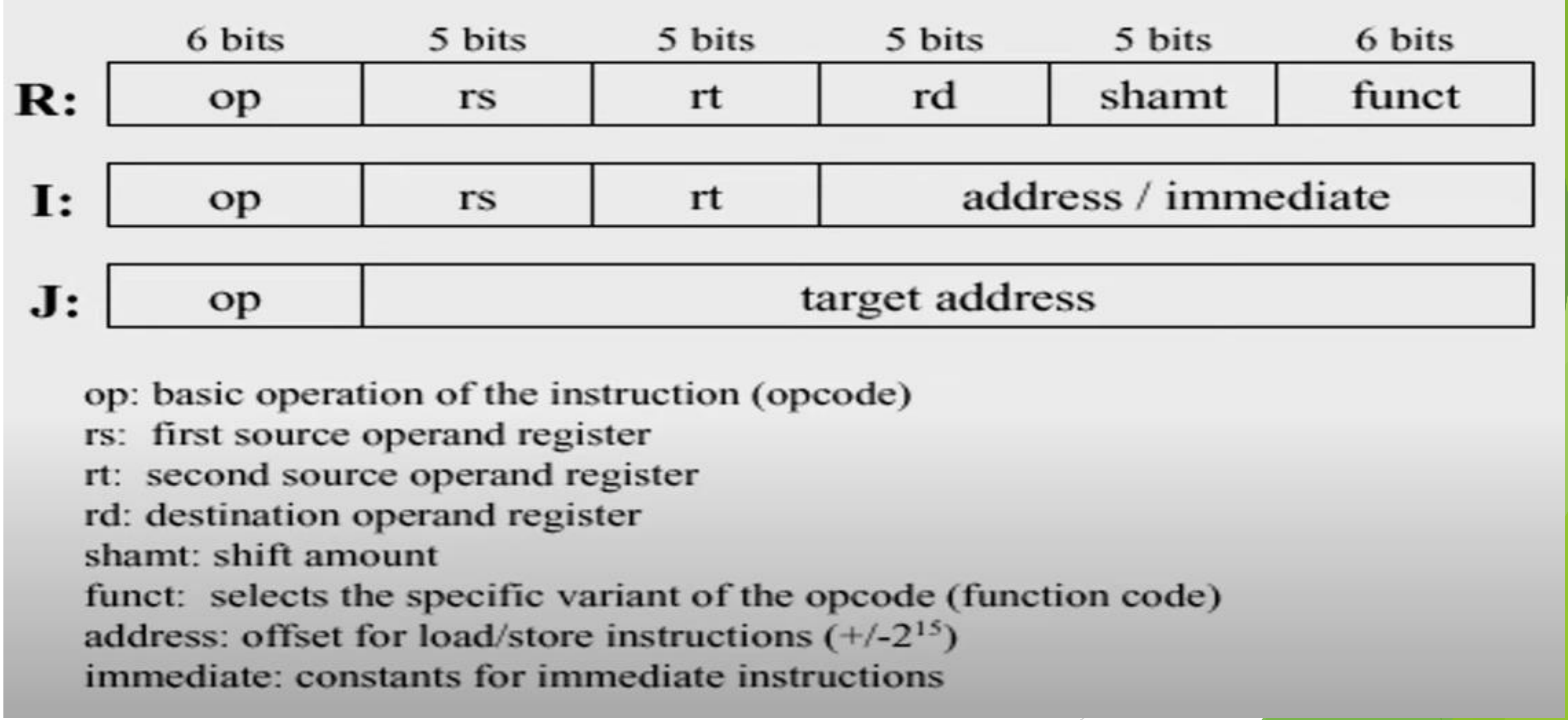
- Instruction Representation in MIPS
- What is pipelining?
- Pipeline characteristics
- Pipeline RISC Datapath
- Five Stages of Pipelined RISC Datapath
- Pipeline Issues

# MIPS

- Microprocessor without Interlocked Pipelined Stages (MIPS)
- 32 registers, 32-bit each
- Uniform length instructions
- RISC load-store architecture



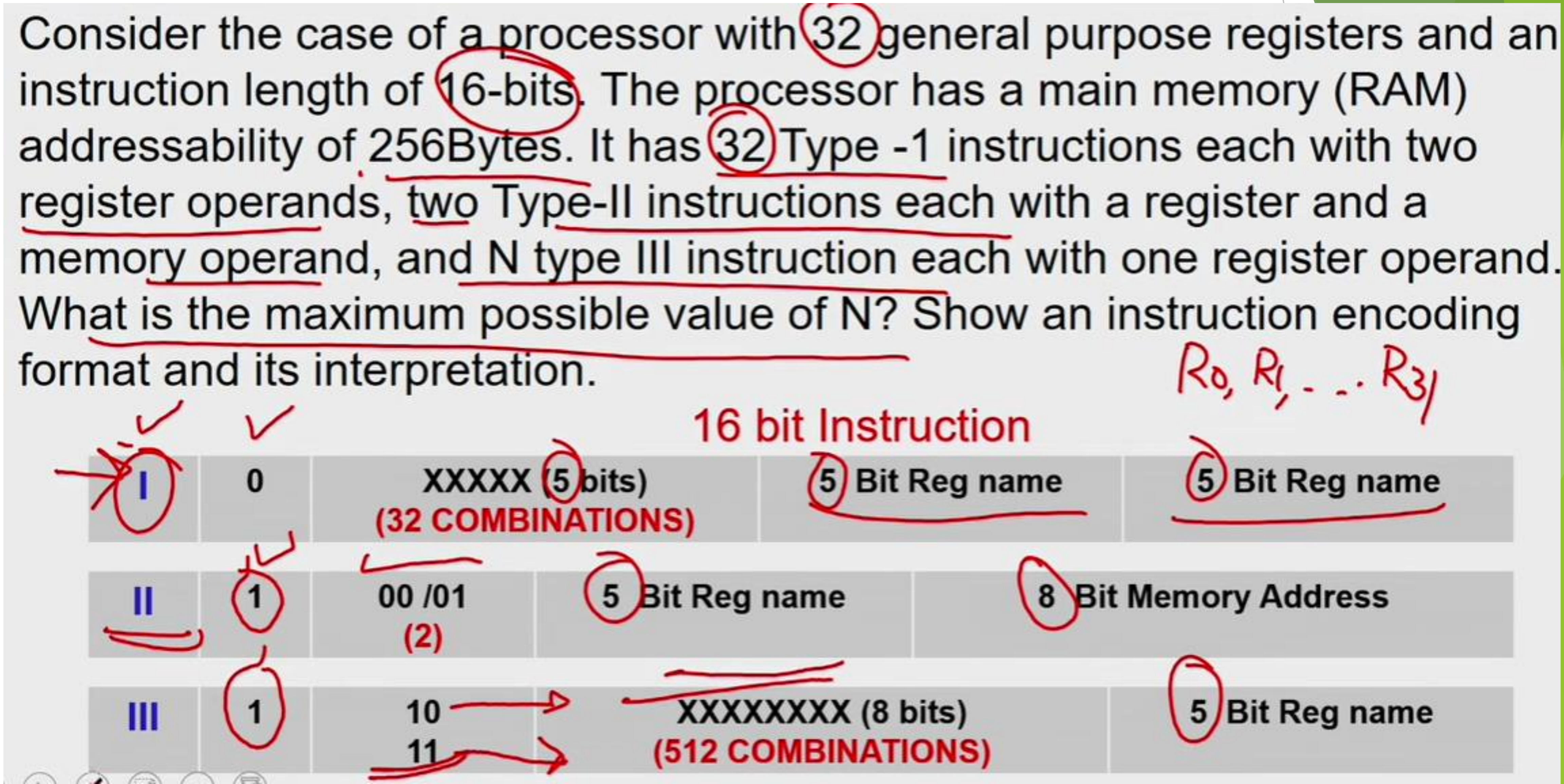
# How instructions are represented in MIPS processors



| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |
|---|---|---|---|---|---|---|
| R: | op | rs | rt | rd | shamt | funct |
| I: | op | rs | rt | address / immediate | | |
| J: | op | target address | | | | |

op: basic operation of the instruction (opcode)
rs: first source operand register
rt: second source operand register
rd: destination operand register
shamt: shift amount
funct: selects the specific variant of the opcode (function code)
address: offset for load/store instructions (+/-$2^{15}$)
immediate: constants for immediate instructions

# Problem: Instruction Encoding



Consider the case of a processor with 32 general purpose registers and an instruction length of 16-bits. The processor has a main memory (RAM) addressability of 256Bytes. It has 32 Type -1 instructions each with two register operands, two Type-II instructions each with a register and a memory operand, and N type III instruction each with one register operand. What is the maximum possible value of N? Show an instruction encoding format and its interpretation.

$R_0, R_1, \ldots R_{31}$

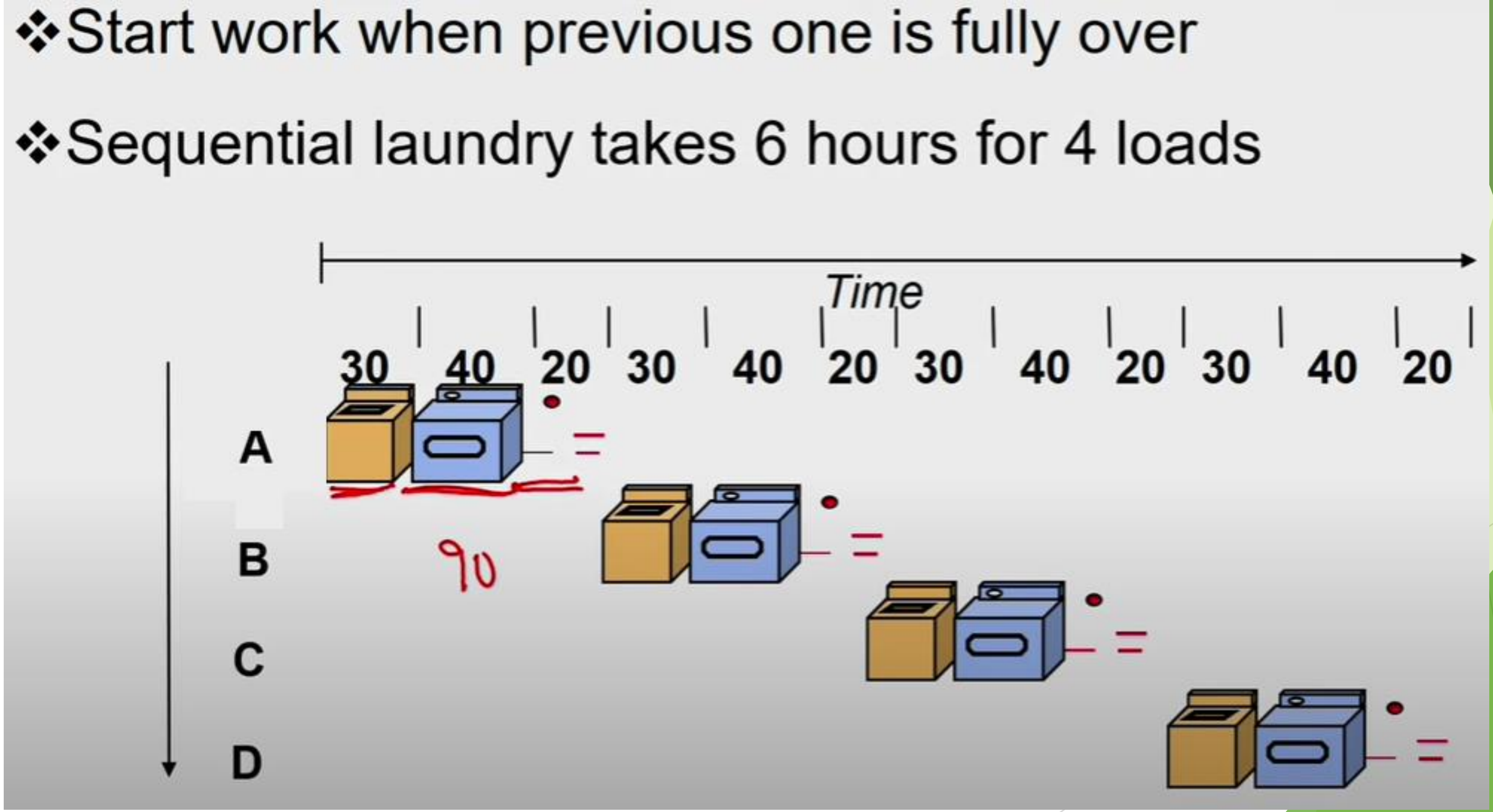
16 bit Instruction

| | | | | |
|---|---|---|---|---|
| I | 0 | XXXXX (5 bits)<br>(32 COMBINATIONS) | 5 Bit Reg name | 5 Bit Reg name |
| II | 1 | 00 /01<br>(2) | 5 Bit Reg name | 8 Bit Memory Address |
| III | 1 | 10<br>11 | XXXXXXXX (8 bits)<br>(512 COMBINATIONS) | 5 Bit Reg name |

# What is Pipelining?

- Pipelining is an implementation technique whereby multiple instructions are overlapped in execution.
- It takes advantage of parallelism that exists among the actions needed to execute an instruction.
- Today, pipelining is the key implementation technique used to make fast CPUs.

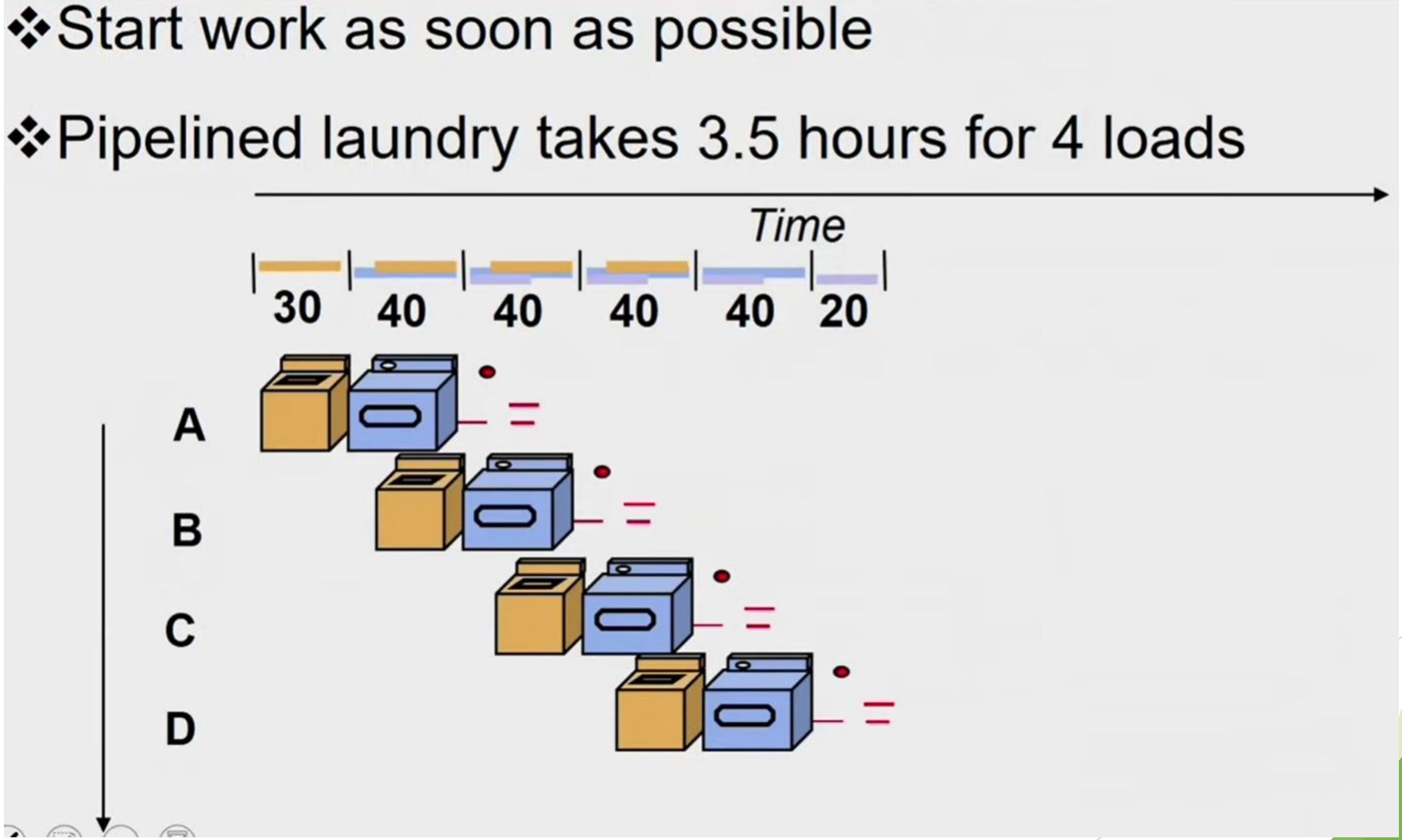
# Un-pipelined Workflow Example



❖Start work when previous one is fully over

❖Sequential laundry takes 6 hours for 4 loads

# Pipeline Workflow Example

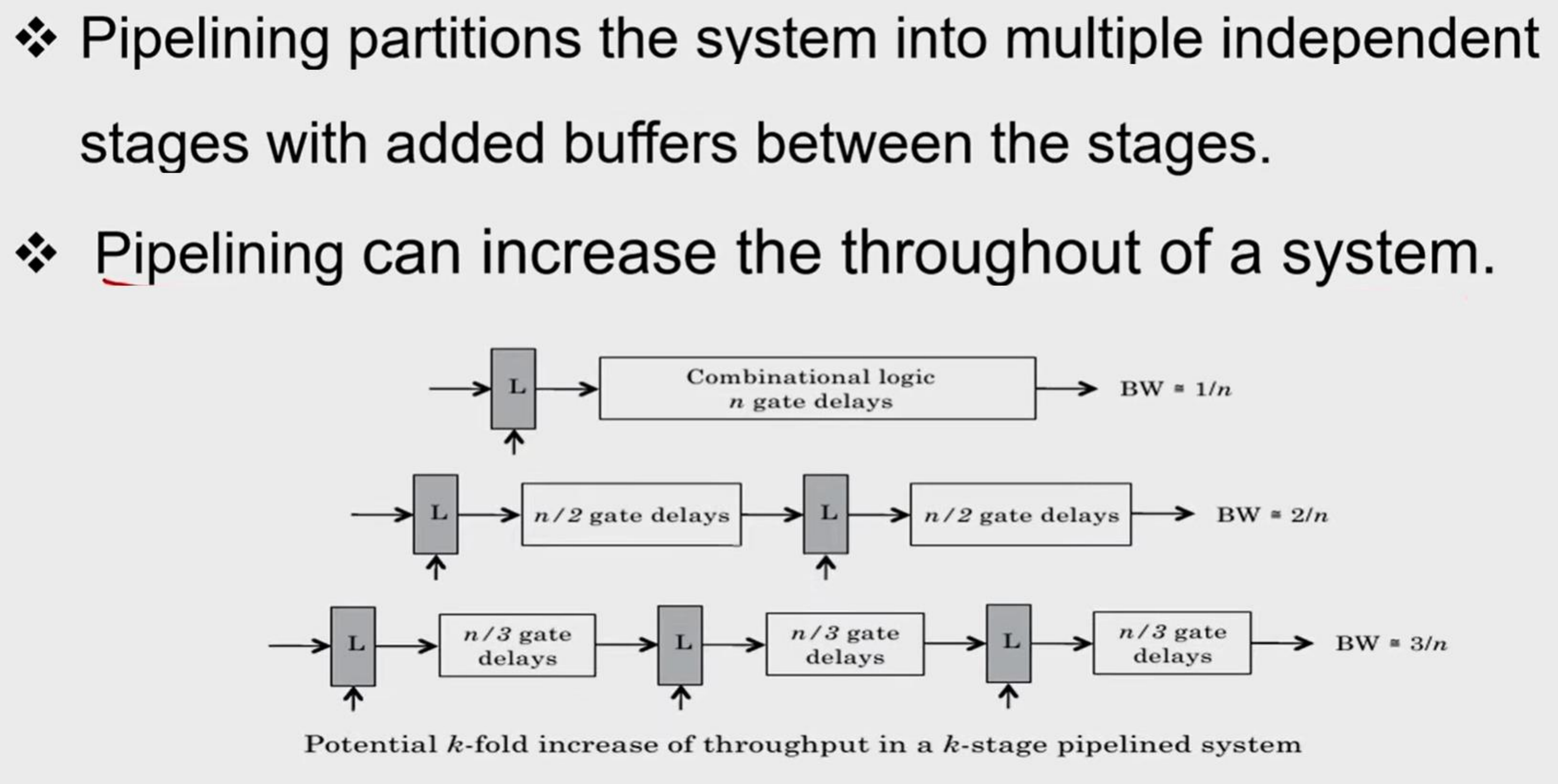


- Start work as soon as possible
- Pipelined laundry takes 3.5 hours for 4 loads

# Pipeline Characteristics

- Pipeline does not reduce the latency of a single task, its improves throughput of entire workload
- Pipeline rate is limited by the slowest pipeline stage
- Potential speedup = No. of pipe stages
- Unbalanced length of pipe stages reduces speedup
- Time to fill and time to drain pipeline reduces speedup

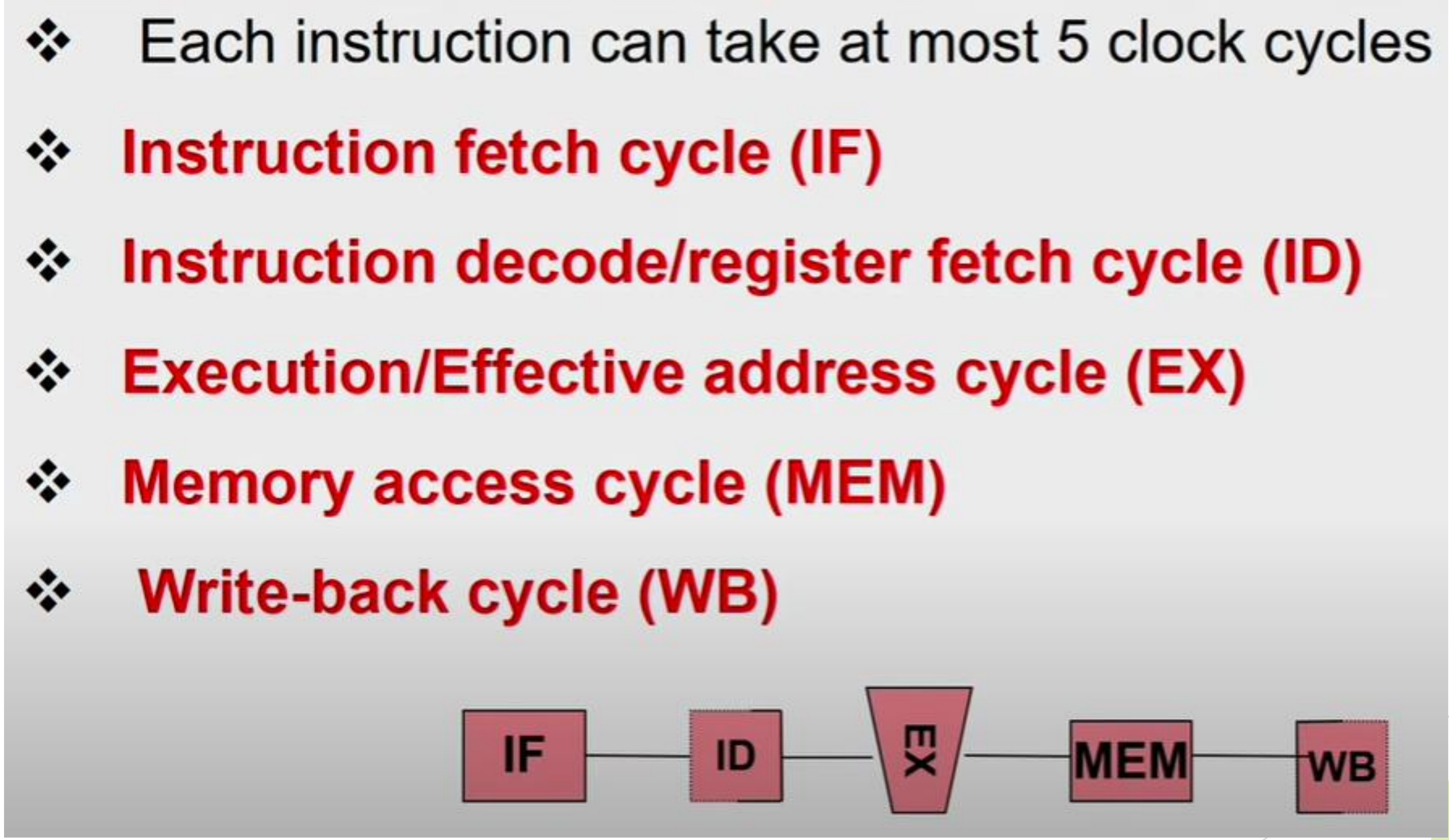
# Pipelining in Circuits



- Pipelining partitions the system into multiple independent stages with added buffers between the stages.
- Pipelining can increase the throughout of a system.

Potential $k$-fold increase of throughput in a $k$-stage pipelined system

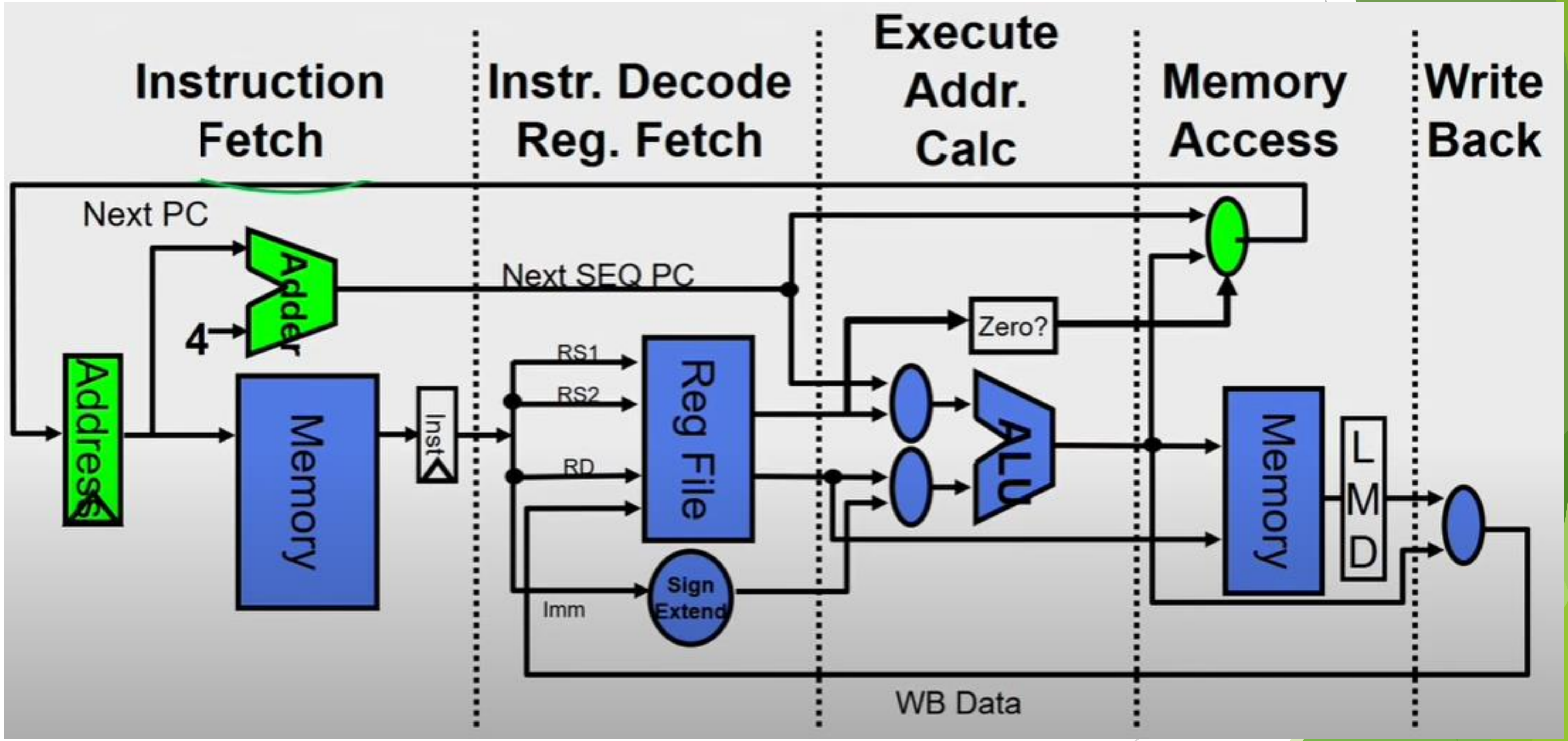
# Instruction Execution Cycle



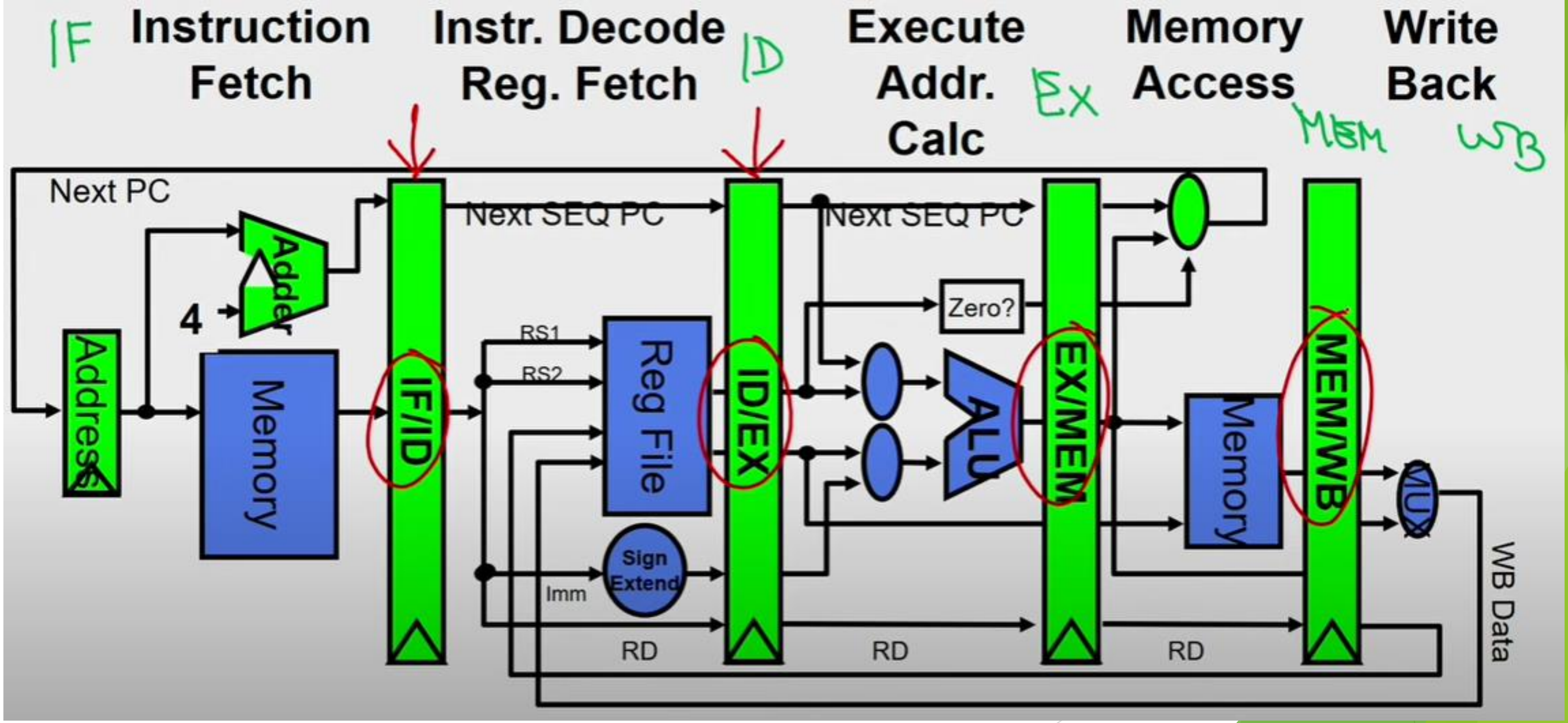
- Each instruction can take at most 5 clock cycles
- **Instruction fetch cycle (IF)**
- **Instruction decode/register fetch cycle (ID)**
- **Execution/Effective address cycle (EX)**
- **Memory access cycle (MEM)**
- **Write-back cycle (WB)**

IF
ID
EX
MEM
WB

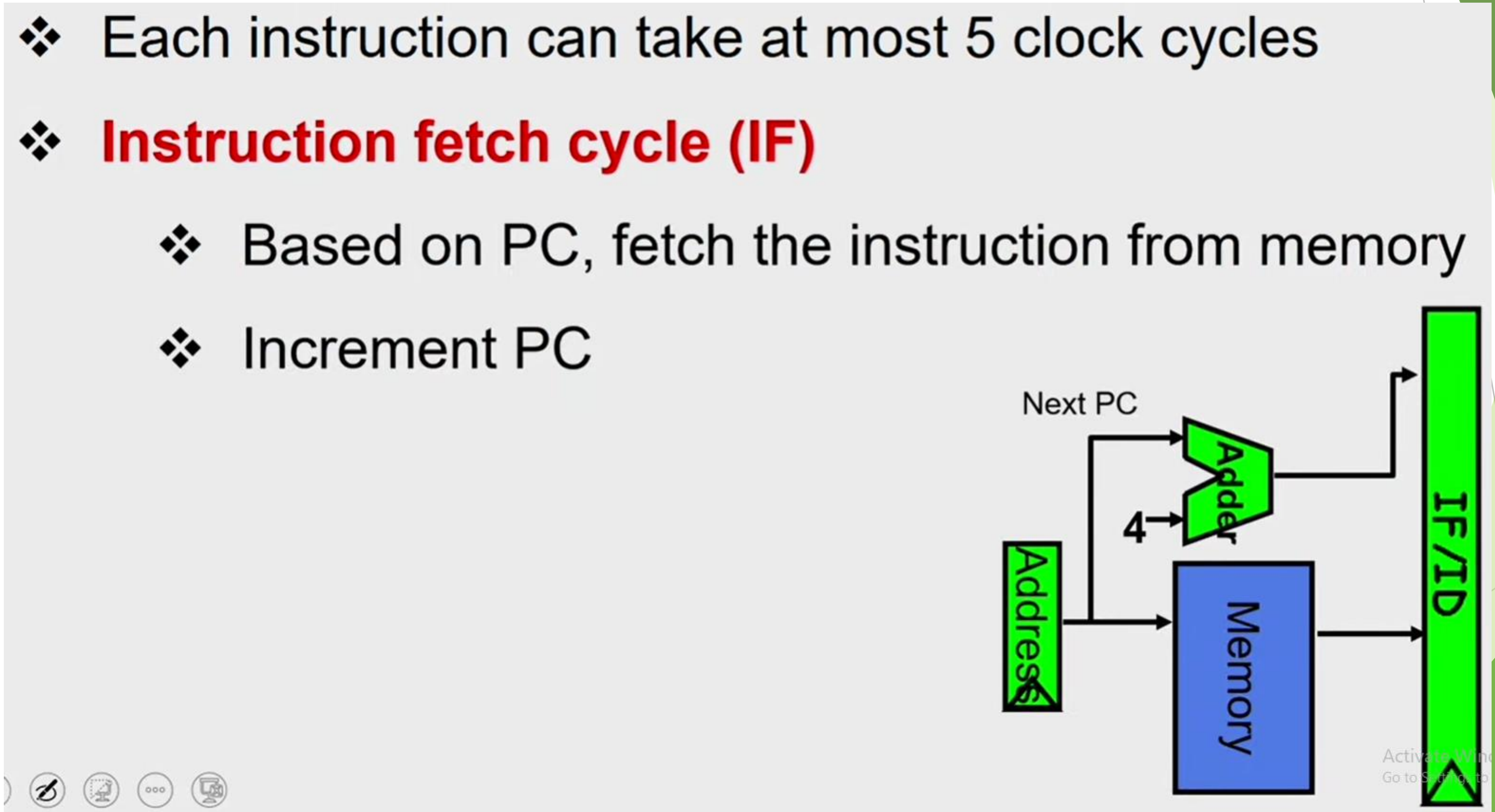
# Un-Pipelined RISC Datapath



# Pipelined RISC Datapath



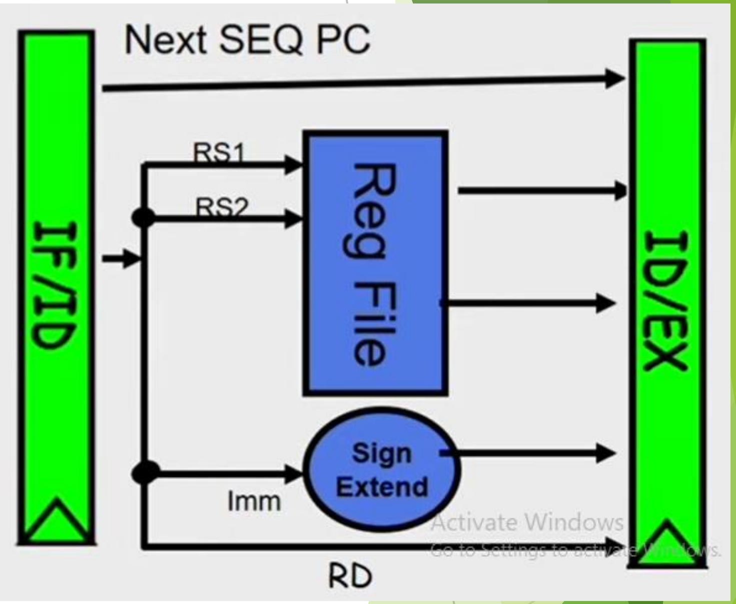
# Pipelined RISC Datapath



- Each instruction can take at most 5 clock cycles
- **Instruction fetch cycle (IF)**
  - Based on PC, fetch the instruction from memory
  - Increment PC

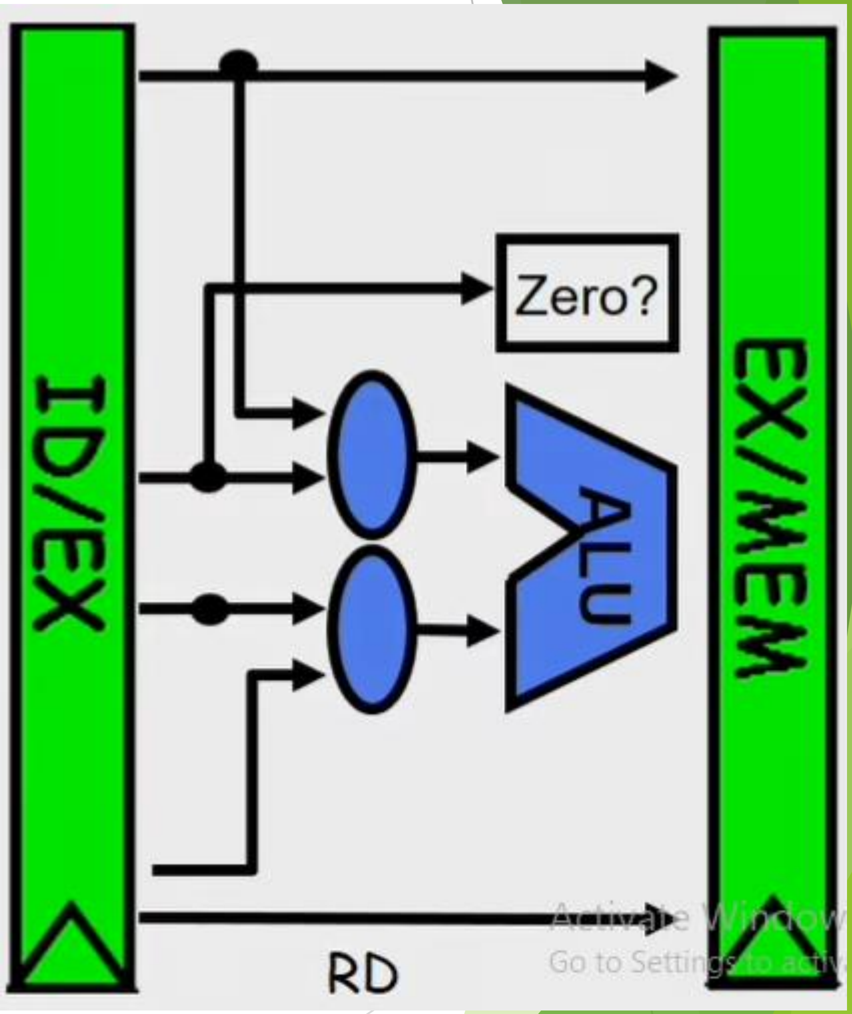
# Pipelined RISC Datapath

- Instruction Decode / Register Fetch Cycle (ID)
  - Decode the instruction + Register read operation
  - Example: ADD R1, R2, R3 = A3.04.02.03
  - = 10100011 00000001 00000010 00000011
  - Example LW R1, 8(R2) = 86.01.02.03
  - 10000110 00000001 00001000 00000010

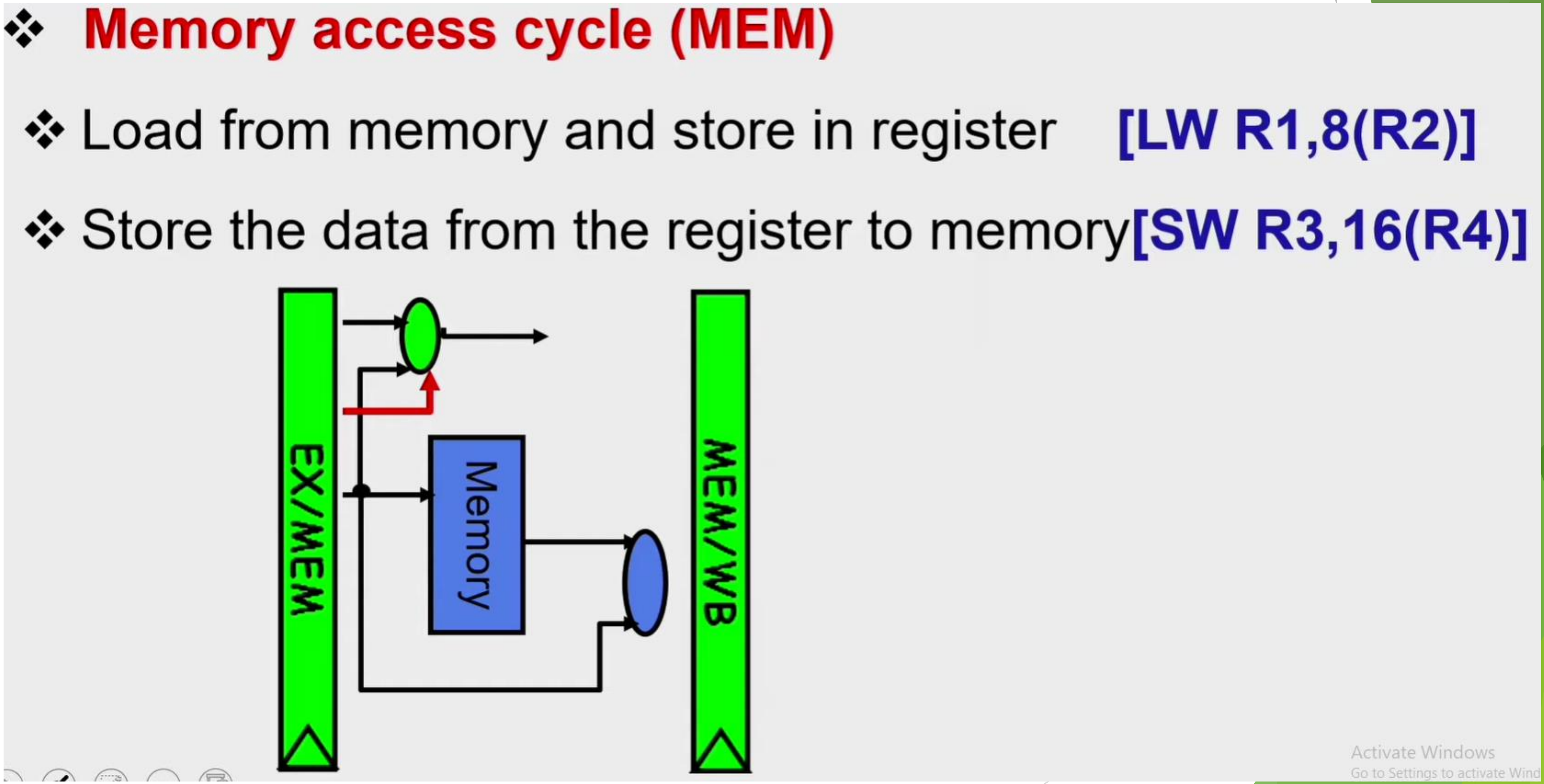


# Pipelined RISC Datapath

- Execution / Effective Address Cycle (Ex)
  - Memory Reference instructions: Calculate Effective Address
  - Example: LW R1, 8(R2), Effective address = [R2]+8
  - Execute Reg-Reg ALU Instructions
  - Example: Add R1, R2, R3



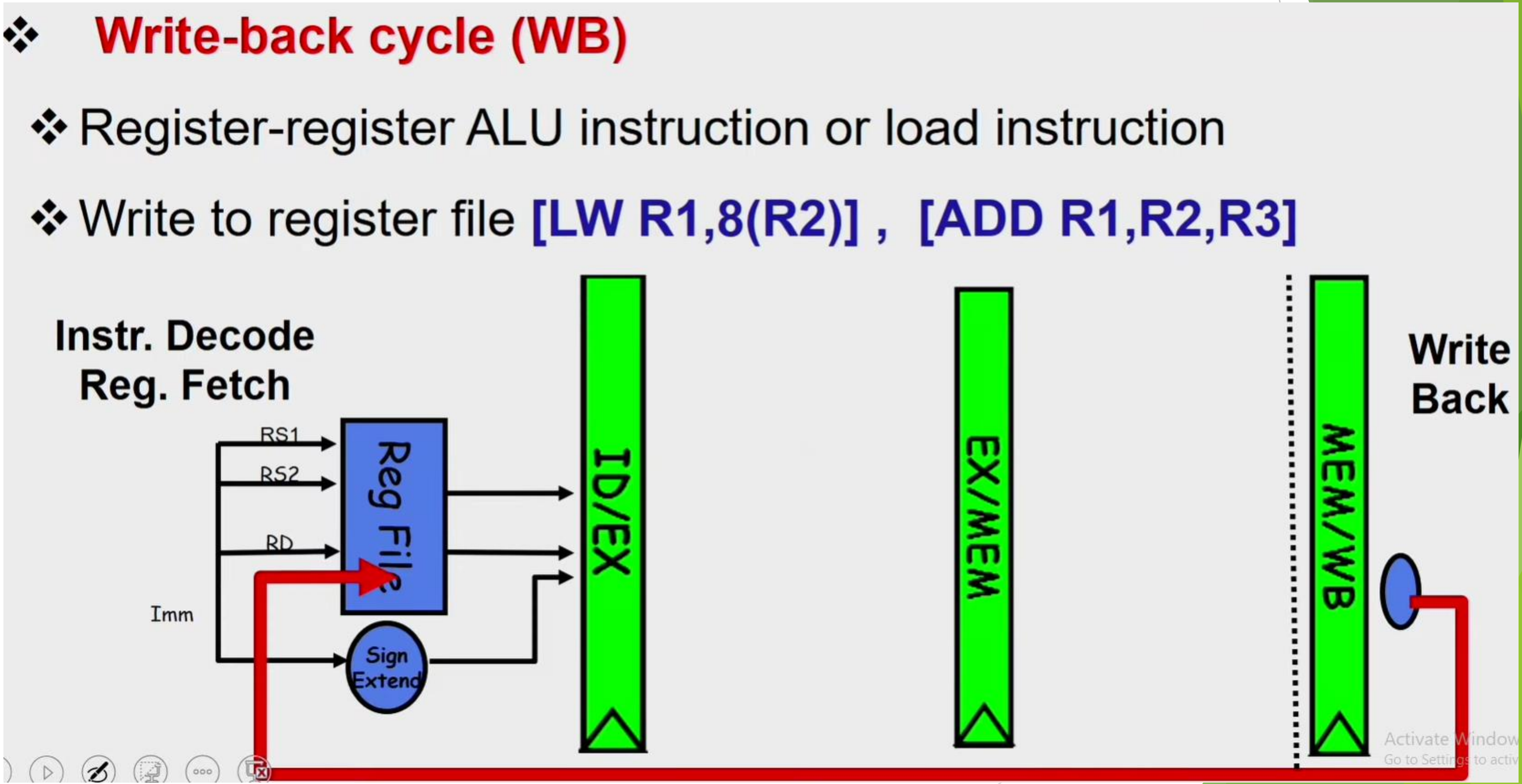
# Pipelined RISC Datapath



❖ **Memory access cycle (MEM)**

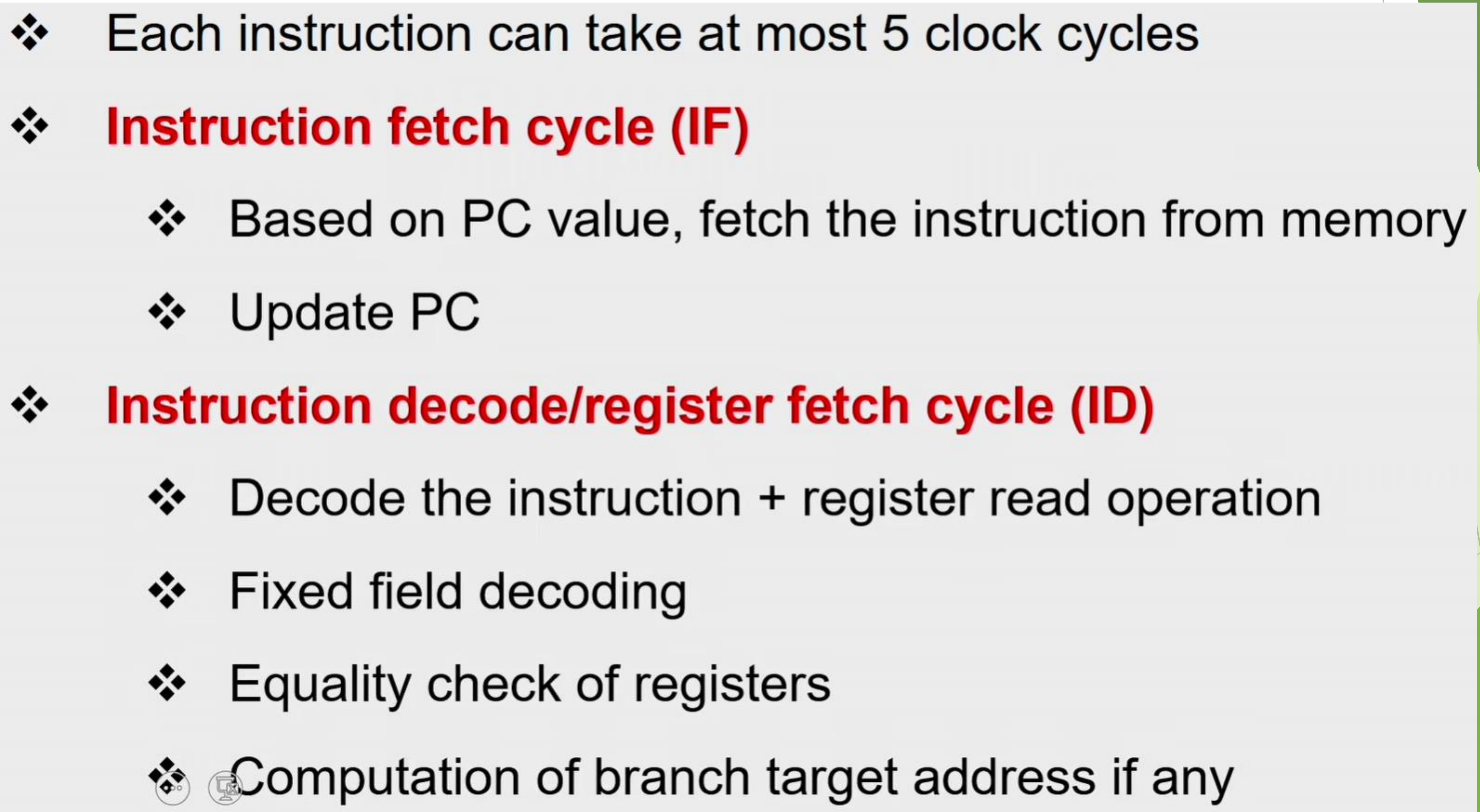
- ❖ Load from memory and store in register **[LW R1,8(R2)]**
- ❖ Store the data from the register to memory**[SW R3,16(R4)]**

# Pipelined RISC Datapath



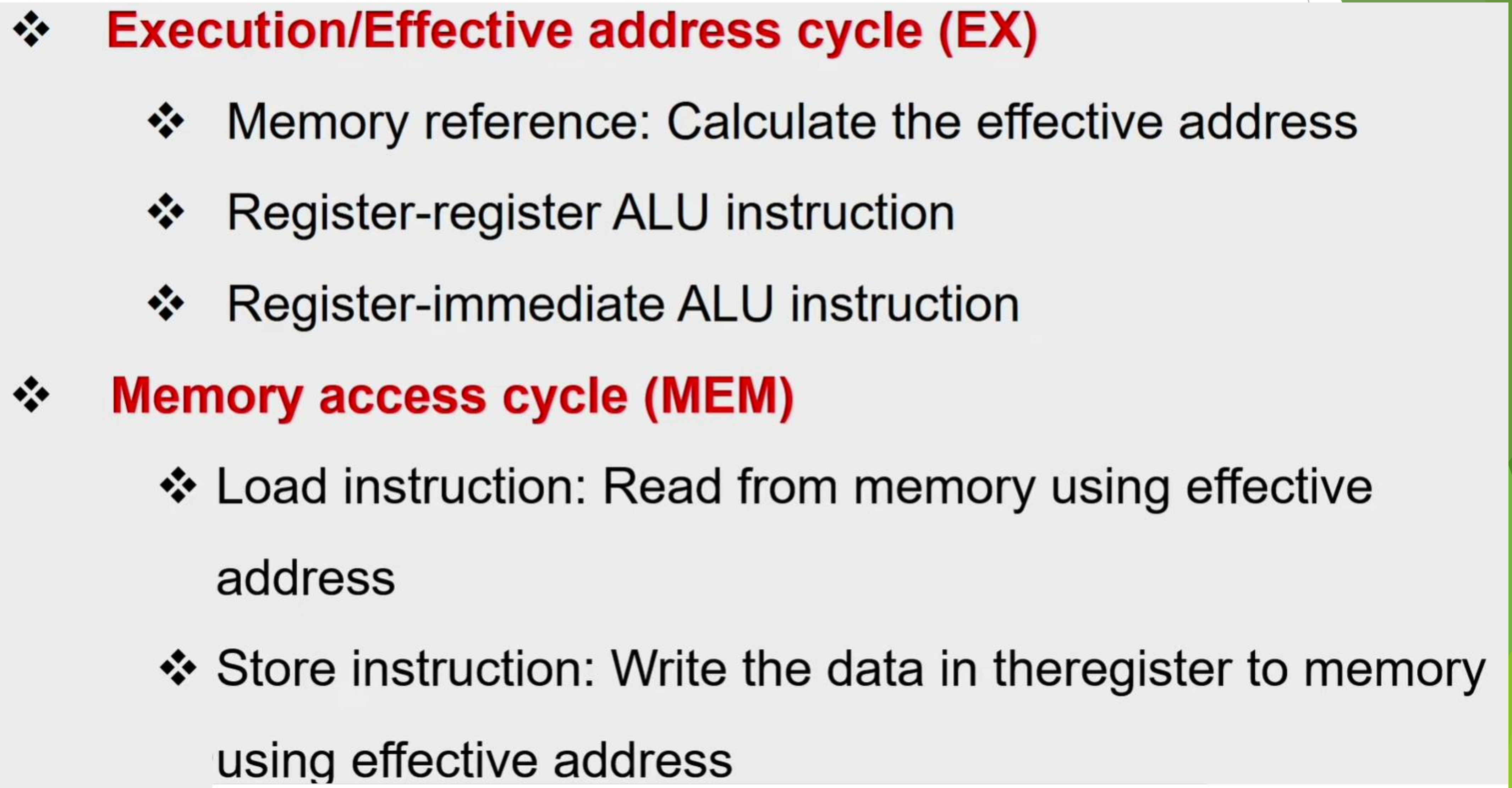
- **Write-back cycle (WB)**
  - Register-register ALU instruction or load instruction
  - Write to register file **[LW R1,8(R2)] , [ADD R1,R2,R3]**

# Five Stages of Pipelined RISC Datapath



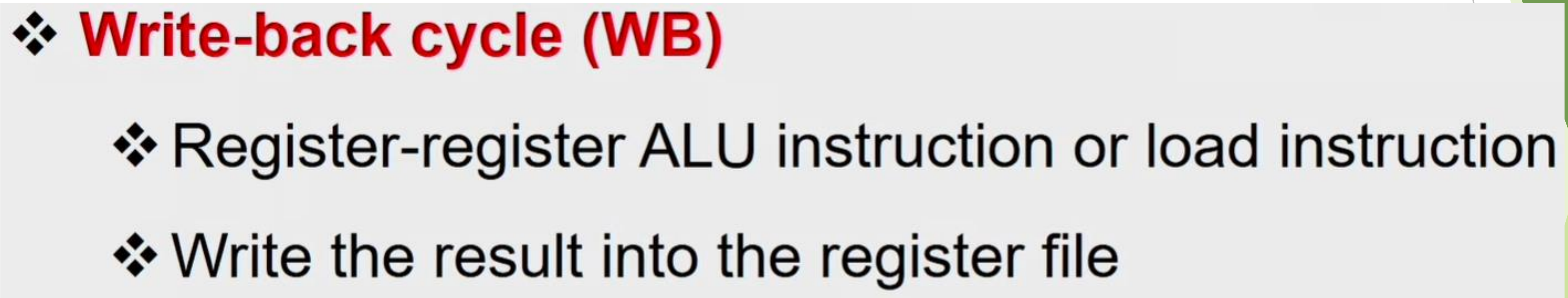
- Each instruction can take at most 5 clock cycles
- **Instruction fetch cycle (IF)**
  - Based on PC value, fetch the instruction from memory
  - Update PC
- **Instruction decode/register fetch cycle (ID)**
  - Decode the instruction + register read operation
  - Fixed field decoding
  - Equality check of registers
  - Computation of branch target address if any

# Five Stages of Pipelined RISC Datapath (Contd.)



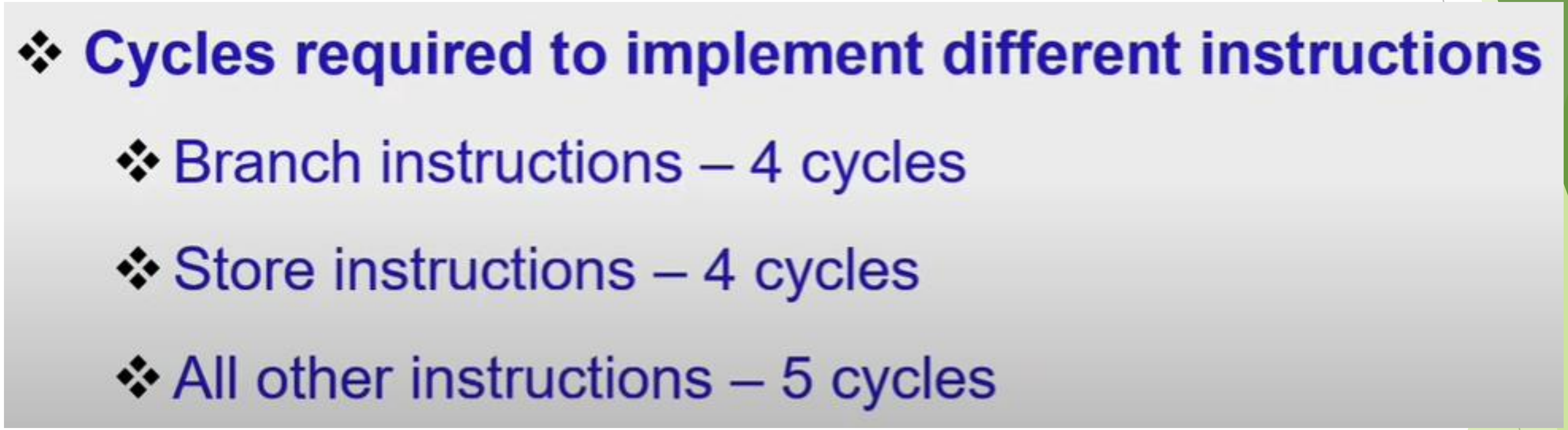
- **Execution/Effective address cycle (EX)**
  - Memory reference: Calculate the effective address
  - Register-register ALU instruction
  - Register-immediate ALU instruction
- **Memory access cycle (MEM)**
  - Load instruction: Read from memory using effective address
  - Store instruction: Write the data in theregister to memory using effective address

# Five Stages of Pipelined RISC Datapath (Contd.)



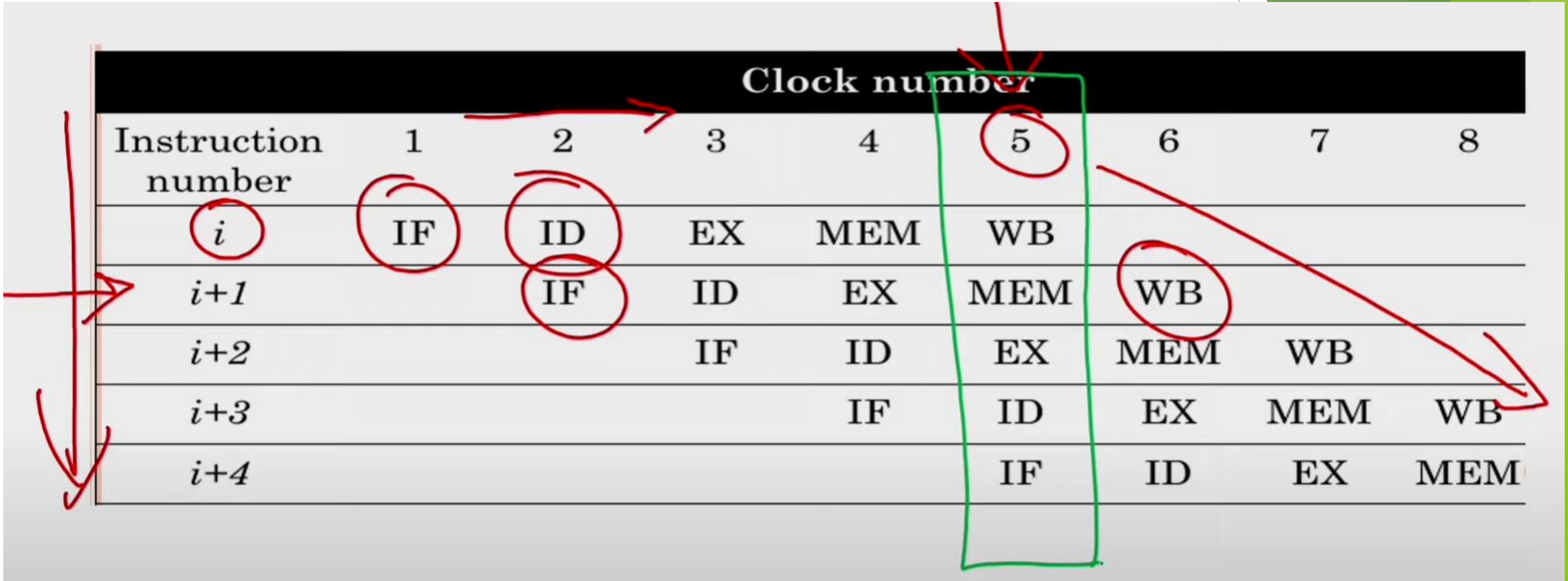
- **Write-back cycle (WB)**
  - Register-register ALU instruction or load instruction
  - Write the result into the register file

# Five Stages of Pipelined RISC Datapath (Contd.)



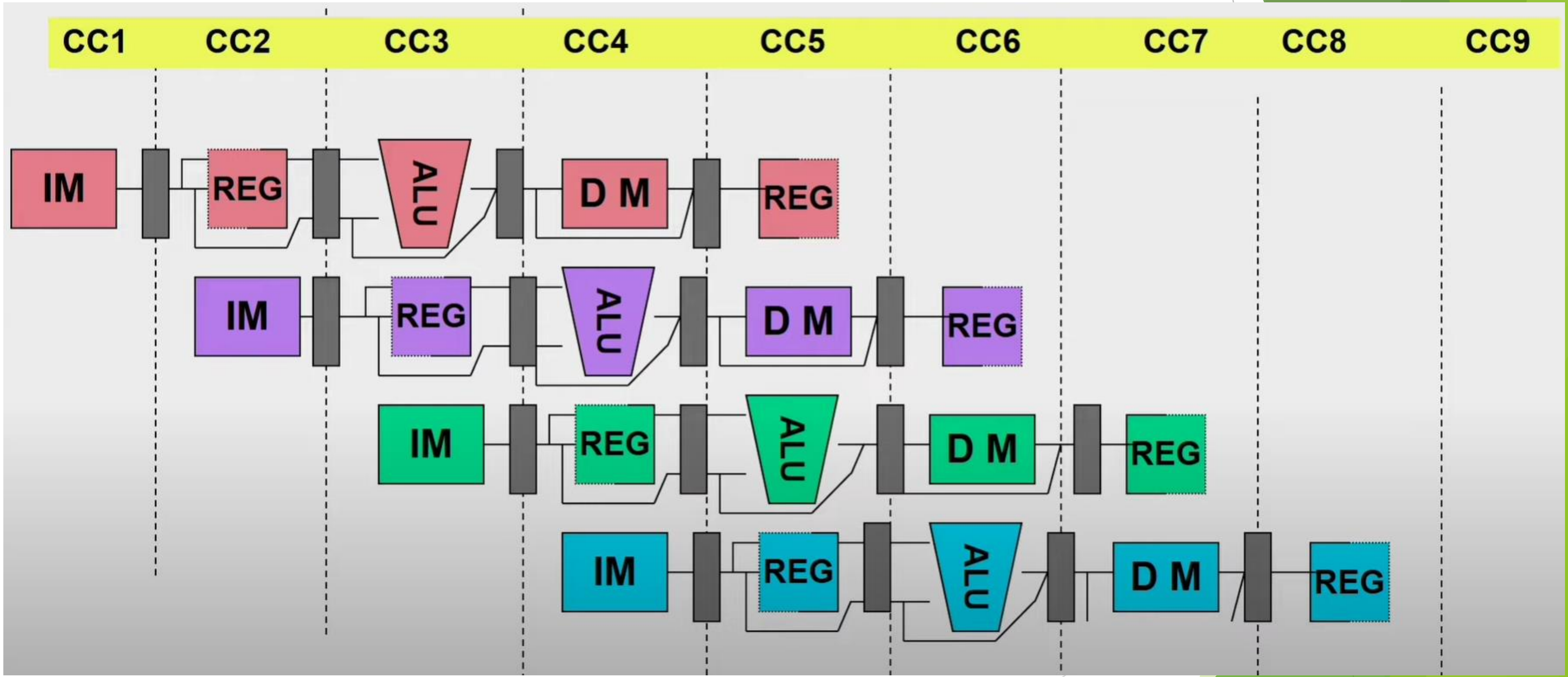
- **Cycles required to implement different instructions**
  - Branch instructions – 4 cycles
  - Store instructions – 4 cycles
  - All other instructions – 5 cycles

# Visualizing the Pipeline

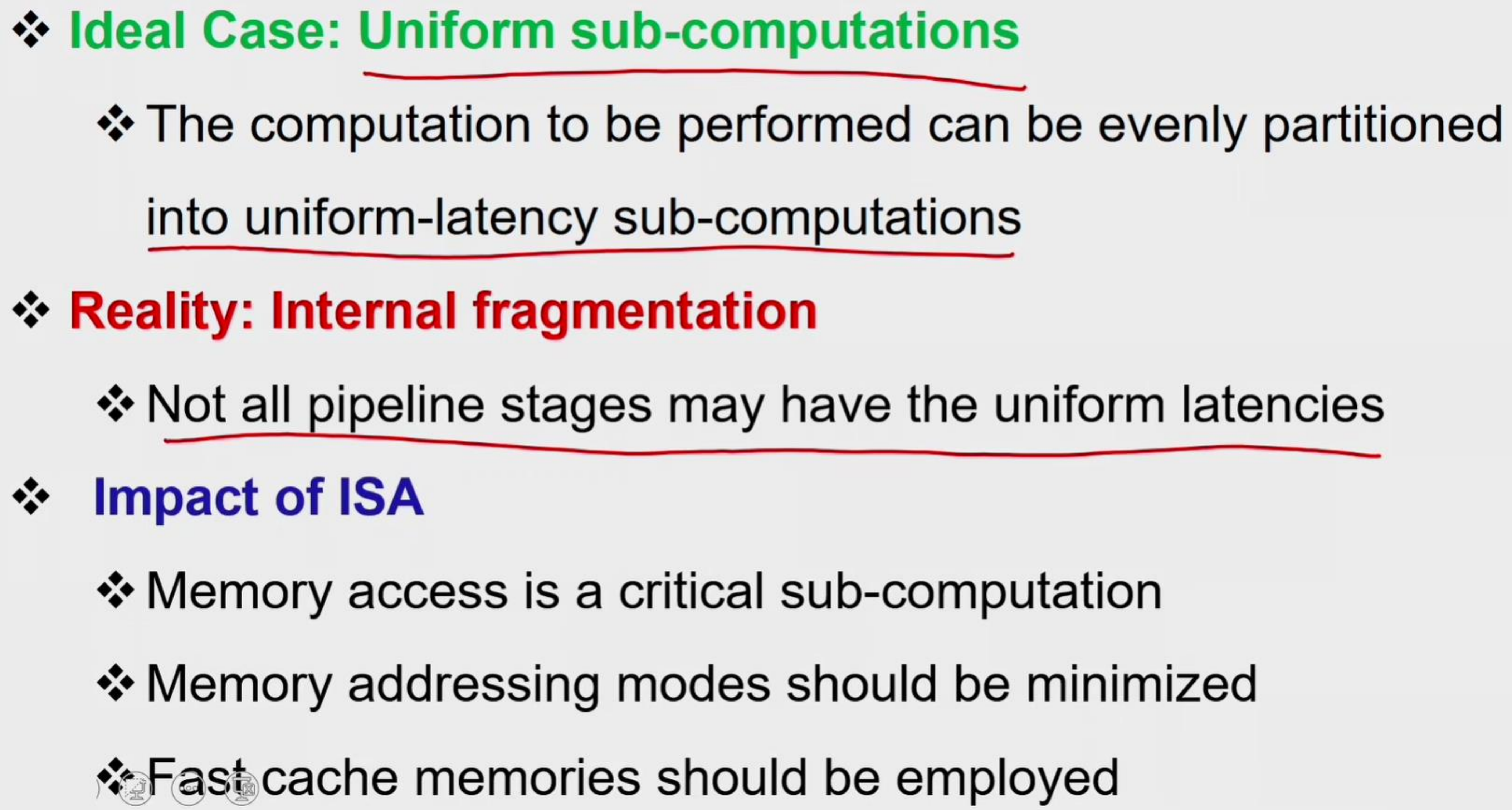


|  | Clock number |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| Instruction number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *i* | IF | ID | EX | MEM | WB |  |  |  |
| *i+1* |  | IF | ID | EX | MEM | WB |  |  |
| *i+2* |  |  | IF | ID | EX | MEM | WB |  |
| *i+3* |  |  |  | IF | ID | EX | MEM | WB |
| *i+4* |  |  |  |  | IF | ID | EX | MEM |

# Visualizing the Pipeline

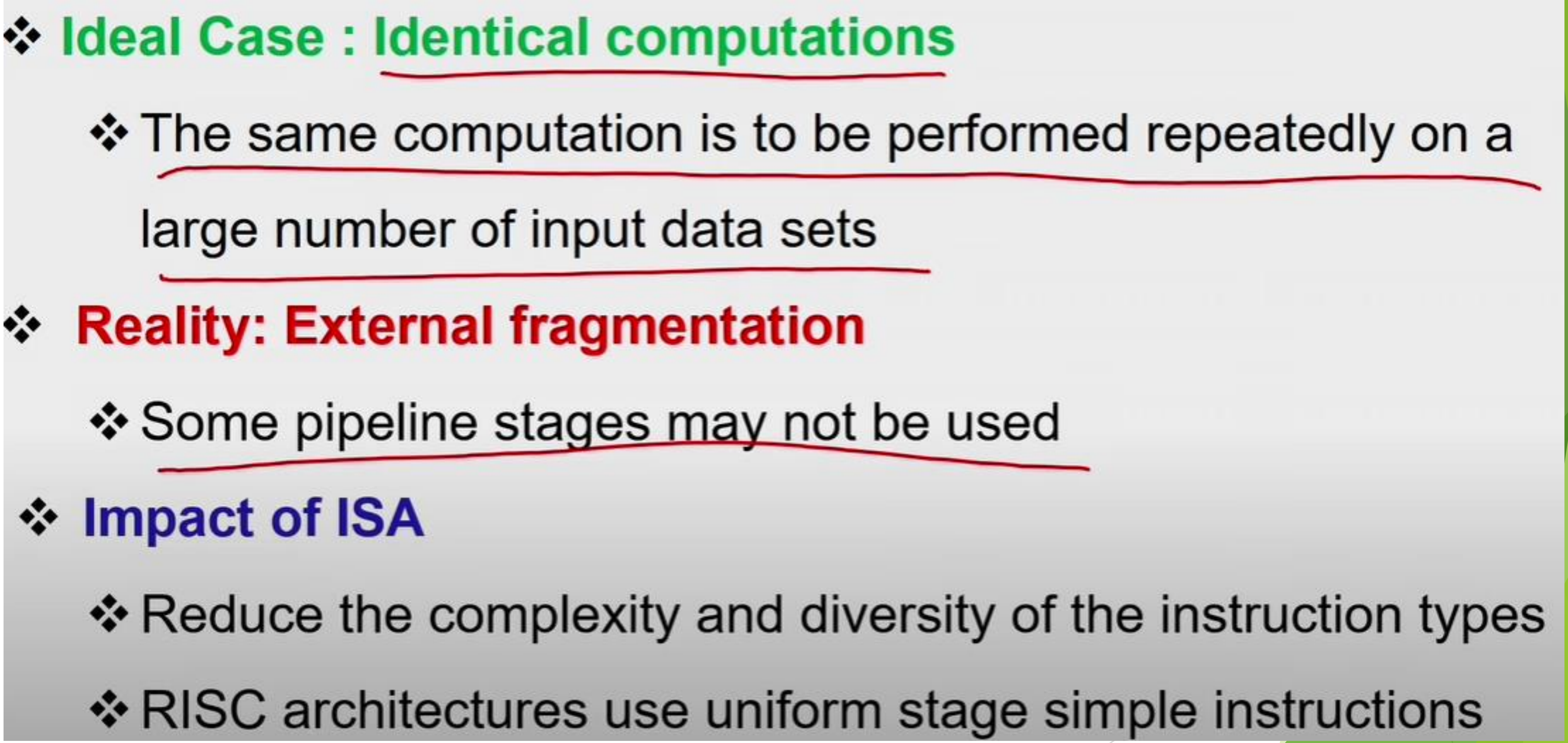


# Pipeline Issues (1)



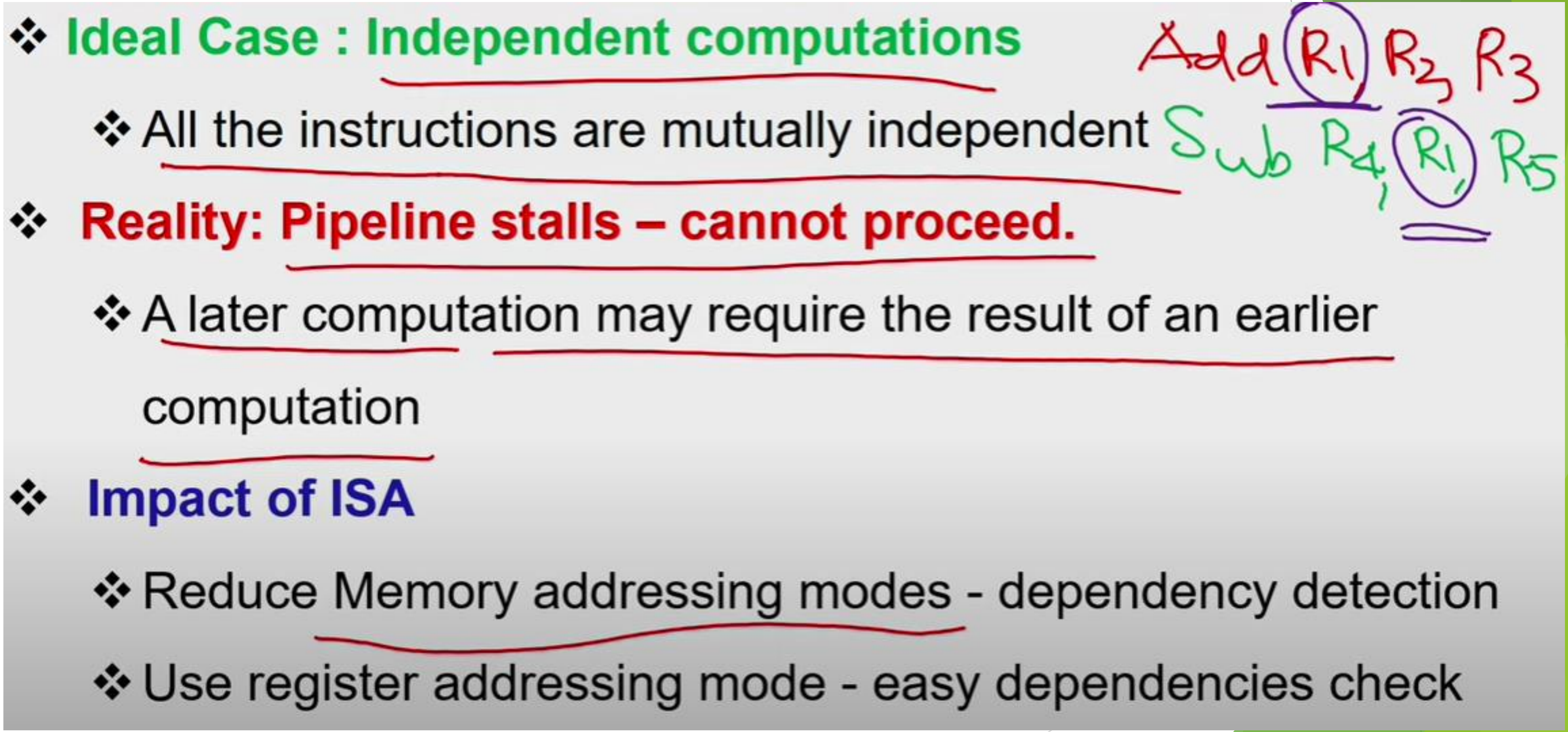
- ❖ **Ideal Case: Uniform sub-computations**
  - ❖ The computation to be performed can be evenly partitioned into uniform-latency sub-computations
- ❖ **Reality: Internal fragmentation**
  - ❖ Not all pipeline stages may have the uniform latencies
- ❖ **Impact of ISA**
  - ❖ Memory access is a critical sub-computation
  - ❖ Memory addressing modes should be minimized
  - ❖ Fast cache memories should be employed

# Pipeline Issues (2)



- **Ideal Case : Identical computations**
  - The same computation is to be performed repeatedly on a large number of input data sets
- **Reality: External fragmentation**
  - Some pipeline stages may not be used
- **Impact of ISA**
  - Reduce the complexity and diversity of the instruction types
  - RISC architectures use uniform stage simple instructions

# Pipeline Issues (3)



❖ **Ideal Case : Independent computations**

- ❖ All the instructions are mutually independent

❖ **Reality: Pipeline stalls – cannot proceed.**

- ❖ A later computation may require the result of an earlier computation

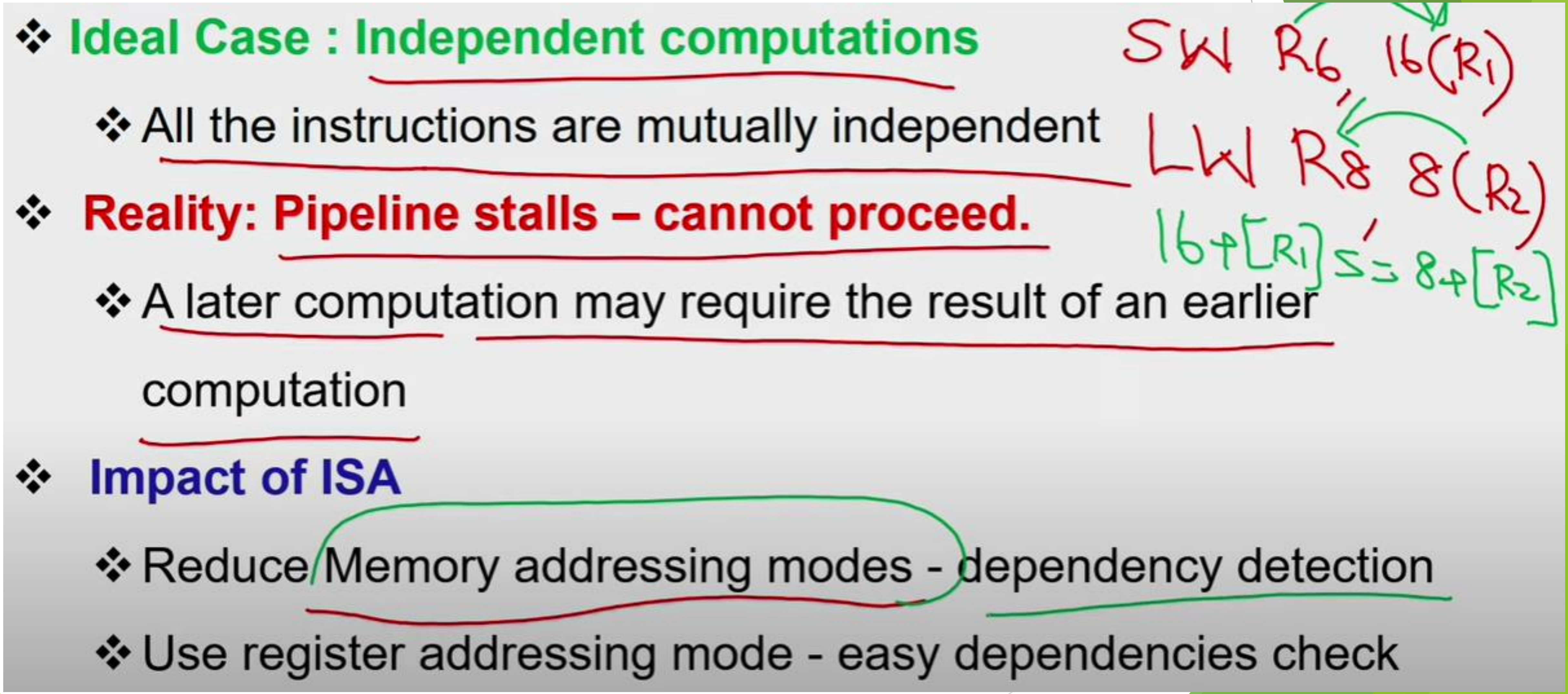
❖ **Impact of ISA**

- ❖ Reduce Memory addressing modes - dependency detection
- ❖ Use register addressing mode - easy dependencies check

Add R1 R2, R3

Sub R4, R1, R5

# Pipeline Issues (4)



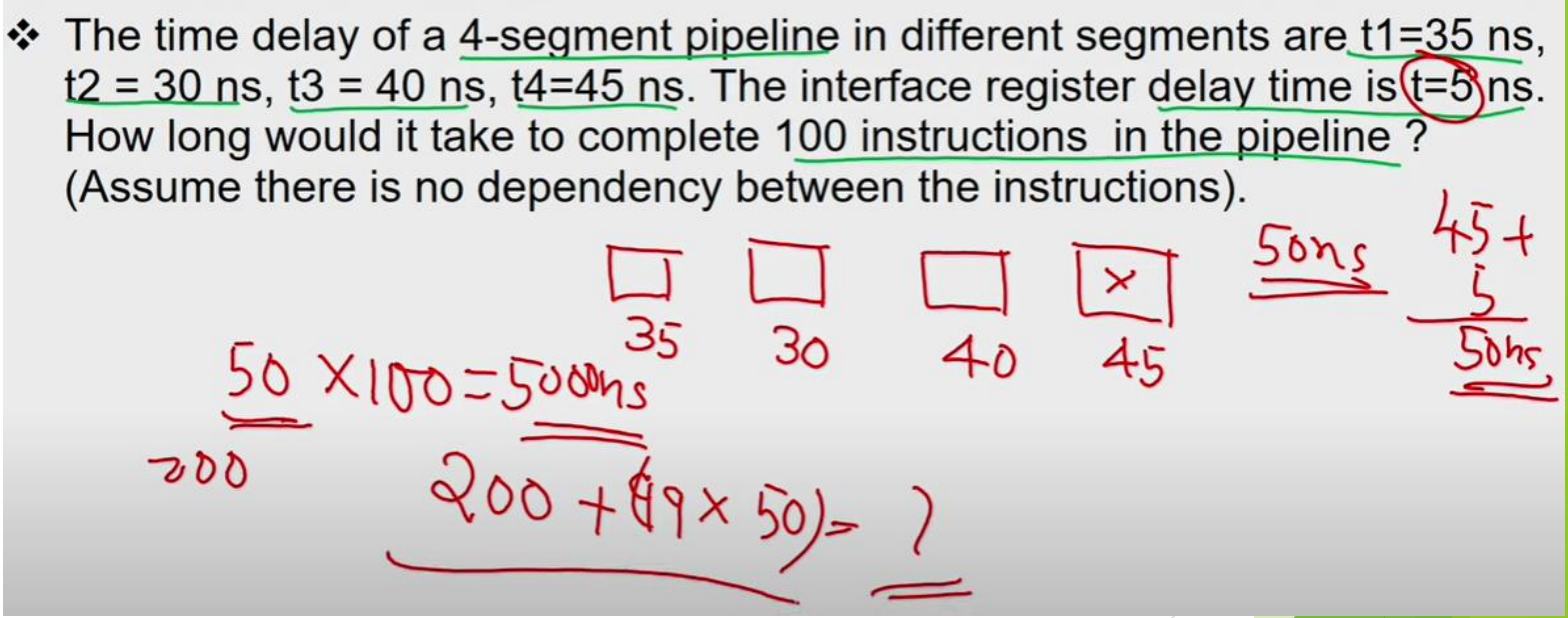
- **Ideal Case : Independent computations**
  - All the instructions are mutually independent
- **Reality: Pipeline stalls – cannot proceed.**
  - A later computation may require the result of an earlier computation
- **Impact of ISA**
  - Reduce Memory addressing modes - dependency detection
  - Use register addressing mode - easy dependencies check

SW $R_6$, $16(R_1)$

LW $R_8$ $8(R_2)$

$16+[R_1]$ S= $8+[R_2]$

# Problem: Pipelined Design



- The time delay of a 4-segment pipeline in different segments are t1=35 ns, t2 = 30 ns, t3 = 40 ns, t4=45 ns. The interface register delay time is t=5 ns. How long would it take to complete 100 instructions in the pipeline ? (Assume there is no dependency between the instructions).

35
30
40
45
50ns

$$\begin{array}{r} 45+ \\ 5 \\ \hline 50ns \end{array}$$

$50 \times 100 = 5000ns$

200

$200 + (99 \times 50) = ?$