CS 223 Computer Organization & Architecture

Lecture 24 [30.03.2020]

Introduction to RISC Instruction Pipeline

John Jose

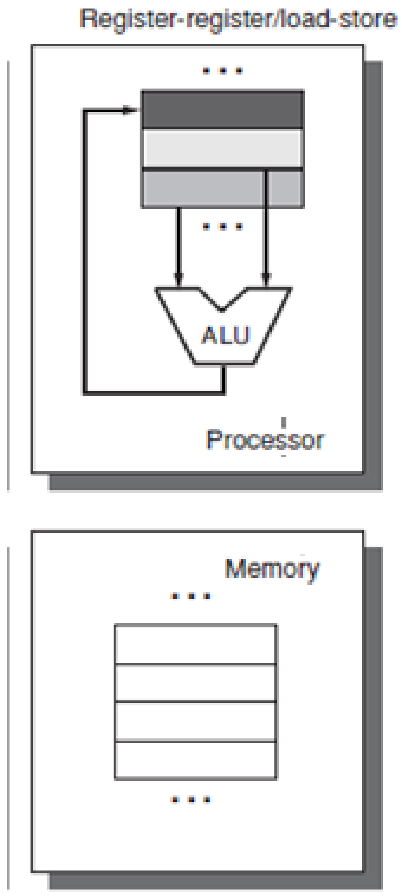
Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

Introduction to MIPS-RISC Architecture

- ❖Microprocessor without Interlocked Pipelined Stages
- ❖32 registers (32 bit each)
- ❖Uniform length instructions [4B]
- ❖RISC- Load store architecture



Main Types of Instructions

- Arithmetic
  - Integer
  - Floating Point
- Memory access instructions
  - Load & Store
- Control flow
  - Jump
  - Conditional Branch
  - Call & Return

MIPS arithmetic

- ❖ Most instructions have 3 operands
- ❖ Operand order is fixed (destination first)

Example C code: A = B + C

MIPS code: add $s0, $s1, $s2

($s0, $s1 and $s2 are associated with variables by compiler)

MIPS arithmetic

C code:

```
A = B + C + D;

E = F - A;
```

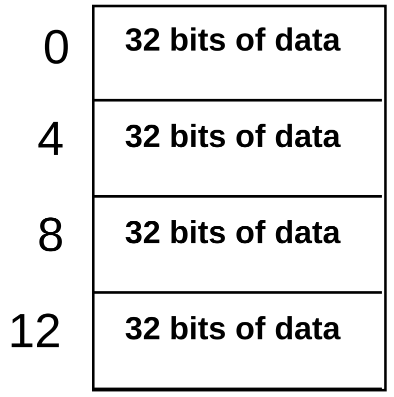
MIPS code:

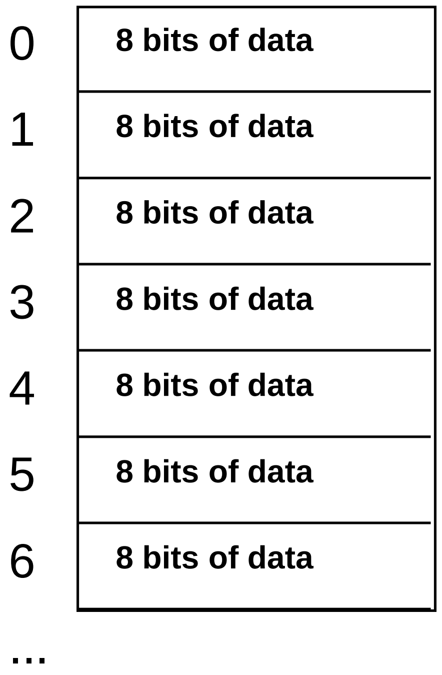
```
add $t0, $s1, $s2

add $s0, $t0, $s3

sub $s4, $s5, $s0
```

- Arithmetic instruction operands must be registers.
- Only 32 registers provided.
- Compiler associates variables with registers.
- Compiler keeps as many variables in registers as possible

Memory Organization

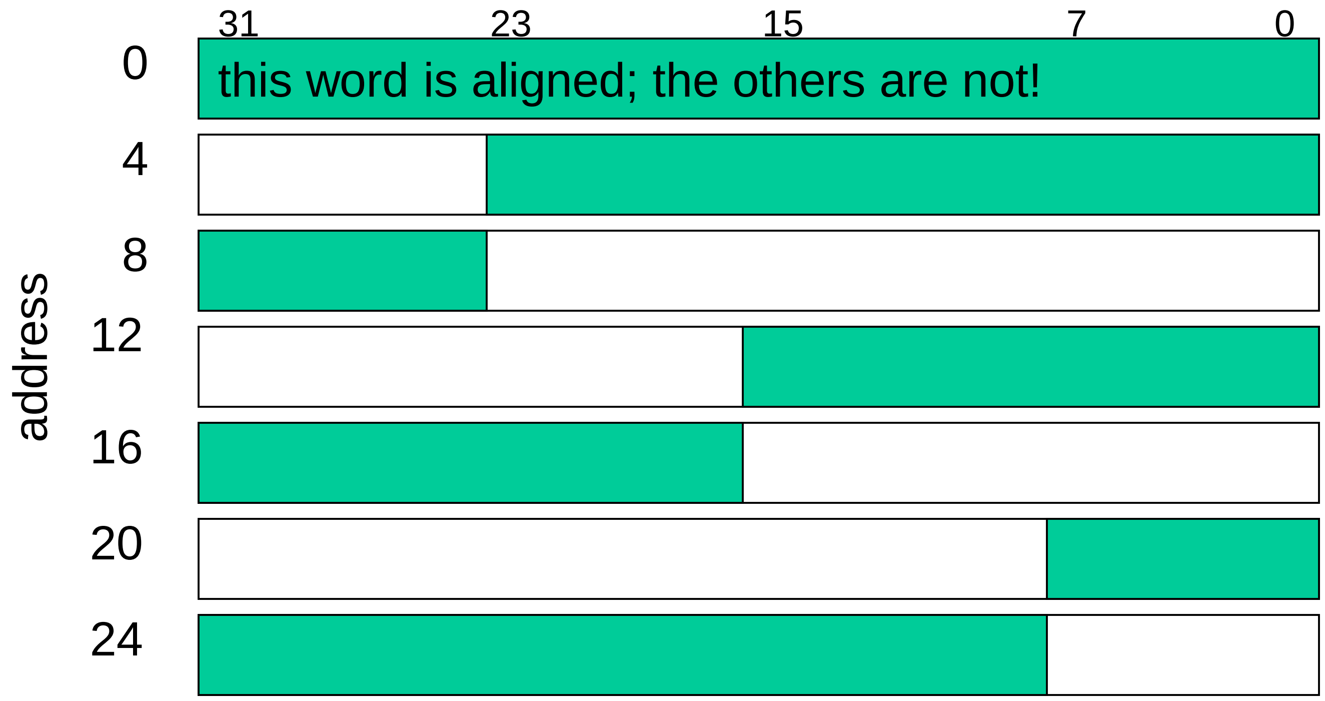
- Viewed as a large, single-dimension array, with an address
- A memory address is an index into the array
- For MIPS, a word is 32 bits or 4 bytes.





Memory Organization

- Words are aligned



Instructions & Meanings

Instruction	Meaning
add $s1, $s2, $s3	$s1 = $s2 + $s3
sub $s1, $s2, $s3	$s1 = $s2 – $s3
lw $s1, 100($s2)	$s1 = Memory[$s2+100]
sw $s1, 100($s2)	Memory[$s2+100] = $s1

Instructions: load and store

Example: C code: A[8] = h + A[8];

MIPS code:

```
lw  $t0, 32($s3)
add $t0, $s2, $t0
sw  $t0, 32($s3)
```

❖ Store word operation has no destination (reg) operand

MIPS Instruction Encoding

	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

MIPS Instruction Encoding : R-type

ADD $R17, $R2, $R8

op	rs	rt	rd	shamt	funct
000000	10001	00010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

MIPS Instruction Encoding : I-type

- Load-word and store-word instructions,
- Example: lw $s3, 32($s2)

30	3	2	32

op	rs	rt	16 bit number

MIPS Instruction Encoding : I-type

- ❖ Decision making instructions

```
bne $t0, $t1, Label
beq $t0, $t1, Label
```

- ❖ Example: if (i==j)

  h = i + j;

```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label: .....
```

bne	$t0	$t1	Label

Op	$t0	$t1	Label

MIPS Instruction Encoding : J-type

- ❖ MIPS unconditional branch instructions: j label
- ❖ Example:

```
if (i!=j)                        beq $s4, $s5, Lab1
    h=i+j;                       add $s3, $s4, $s5
else                             j    Lab2
    h=i-j;              Lab1:  sub $s3, $s4, $s5
xxx                     Lab2:  xxx
```

J	Lab 2

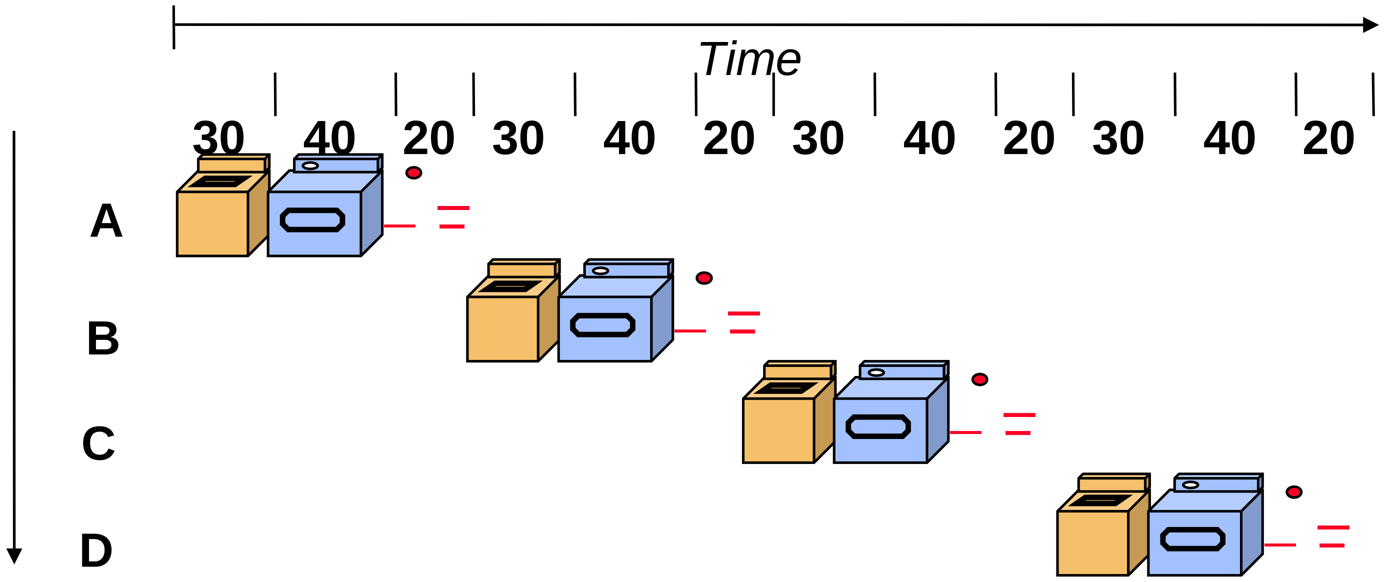
Introduction to Instruction Pipeline Principles

Automobile Production Pipeline

- ❖A Pipelining is a series of stages, where some work unit is done at each stage in parallel.
- ❖The stages are connected one to the next to form a pipe.
- ❖Work module enter at one end, progress through the stages, and exit at the other end.

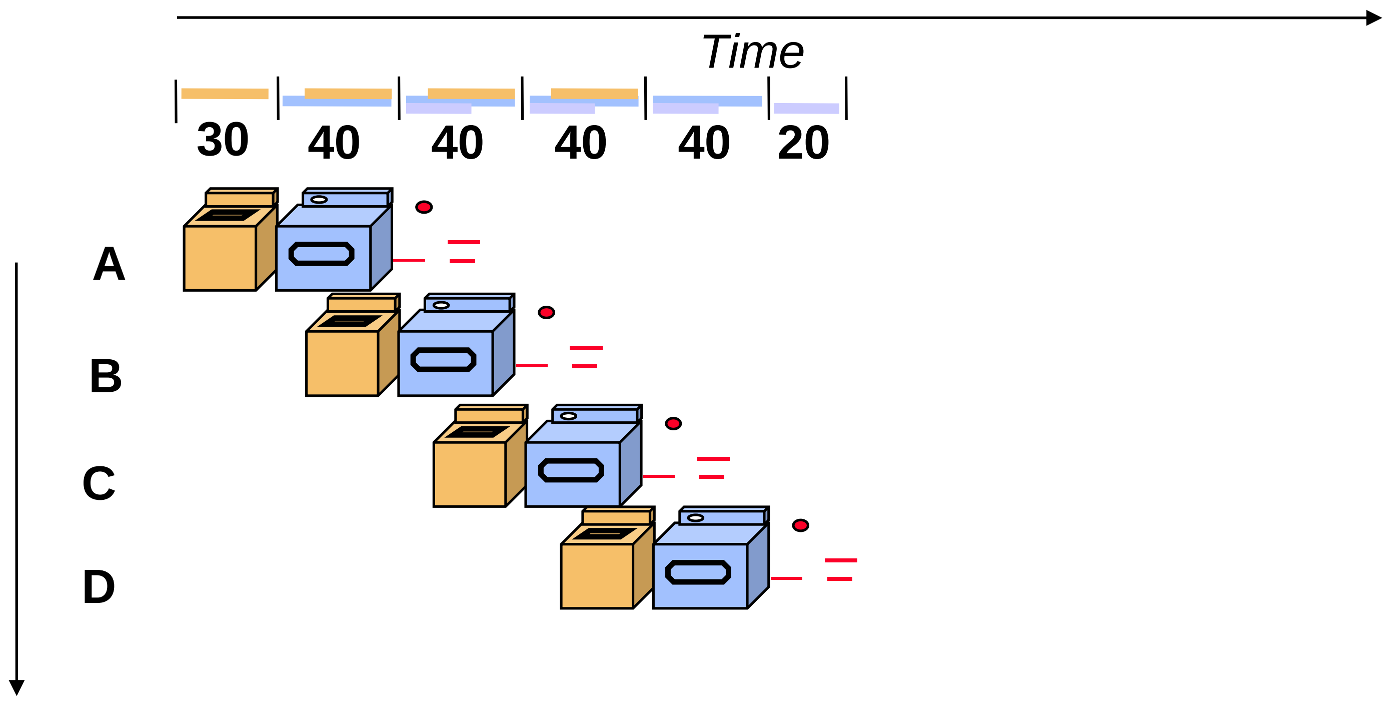
Unpipelined Work flow

- ❖Start work when previous one is fully over
- ❖Sequential laundry takes 6 hours for 4 loads



Pipelined Work flow

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

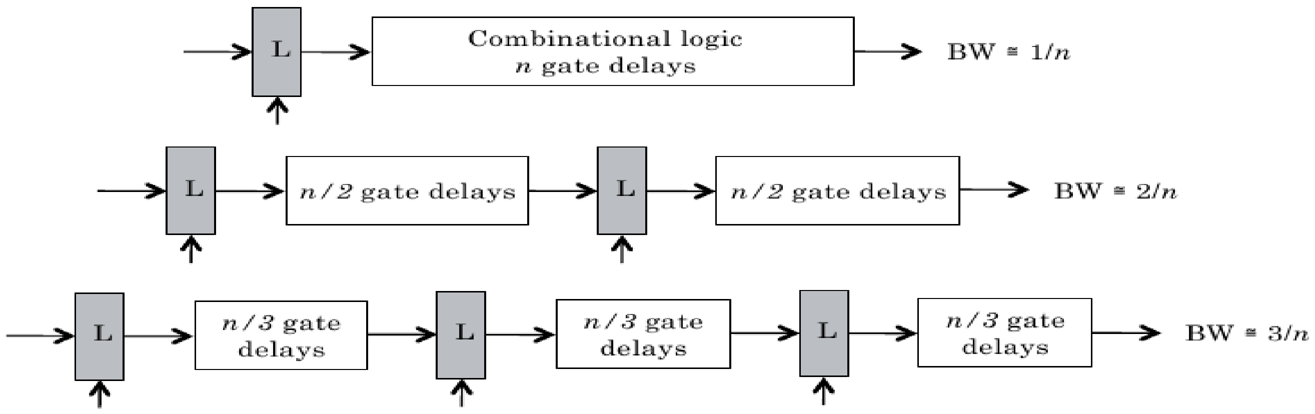


Pipelining Characteristics

- ❖Pipelining doesn’t reduce latency of single task, it improves throughput of entire workload
- ❖Pipeline rate limited by slowest pipeline stage
- ❖Potential speedup = Number of pipe stages
- ❖Unbalanced lengths of pipe stages reduces speedup
- ❖Time to fill pipeline and time to drain it reduces speedup

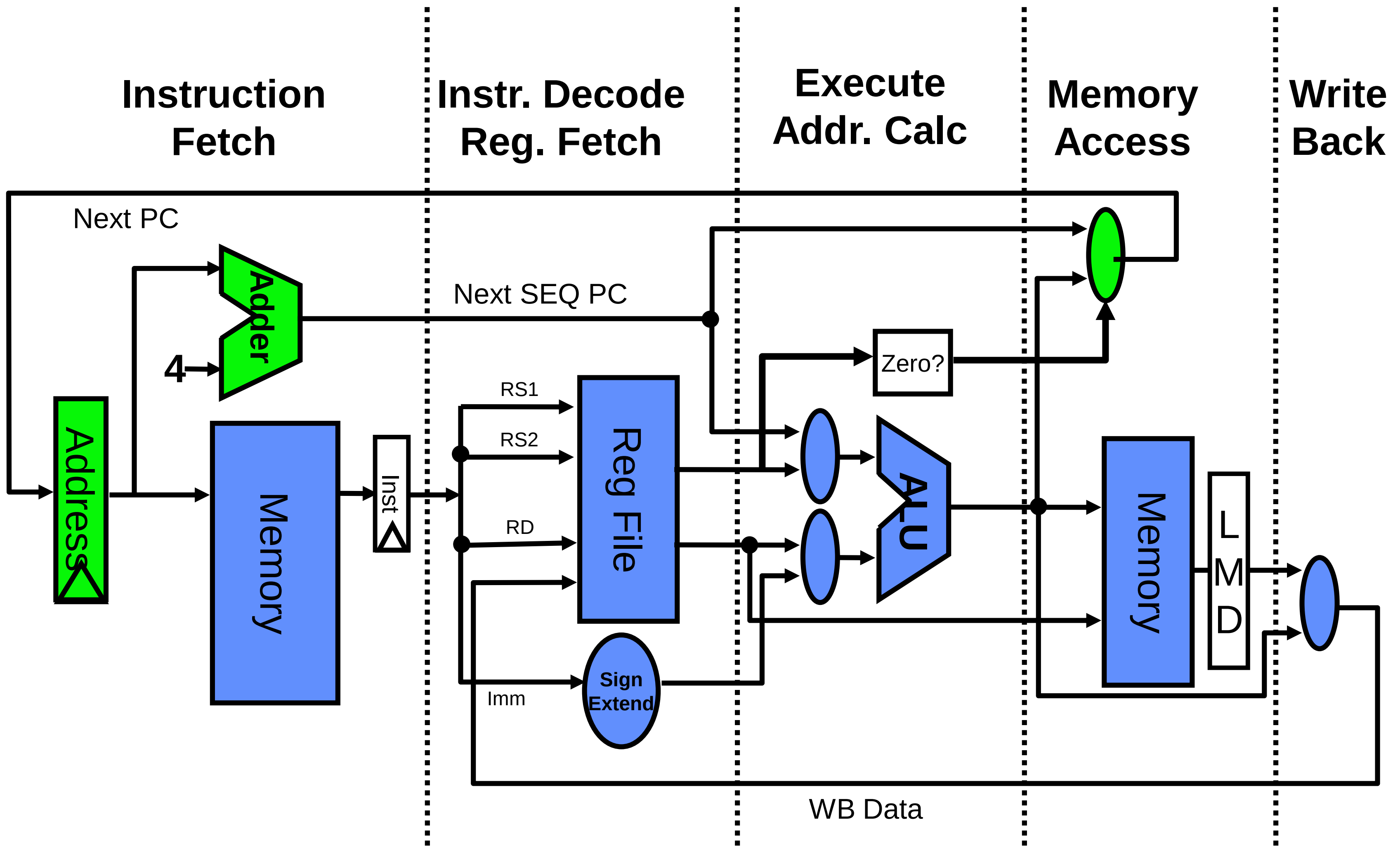
Pipelining in Circuits

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

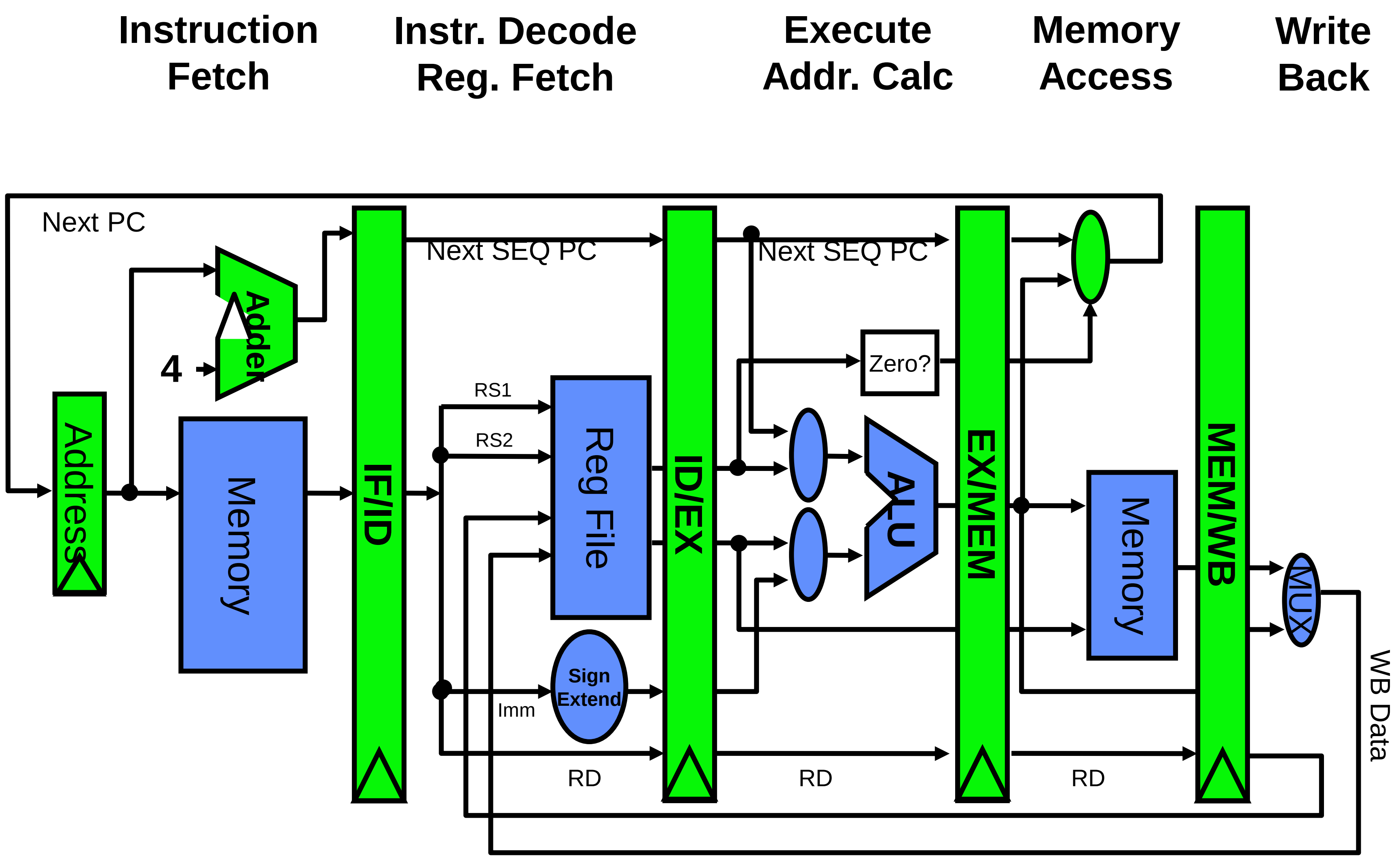


Potential k-fold increase of throughput in a k-stage pipelined system

Unpipelined RISC Data path



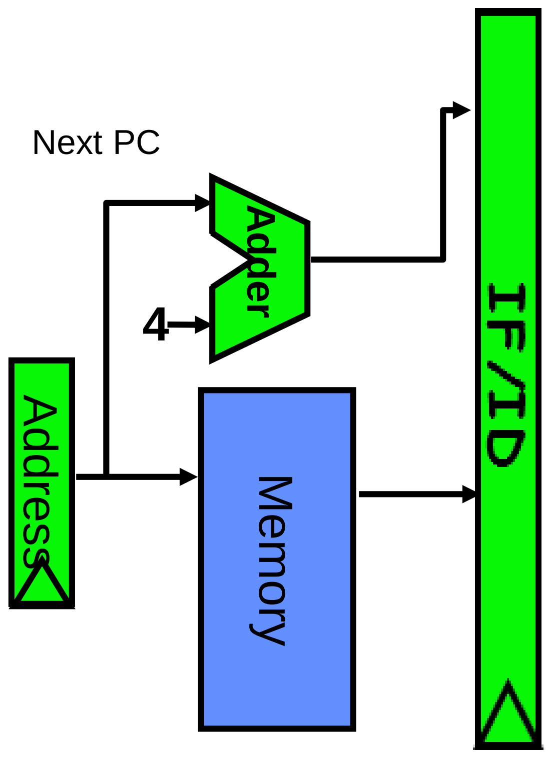
Pipelined RISC Data path



RISC MIPS Instruction Pipeline

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

Instruction Fetch

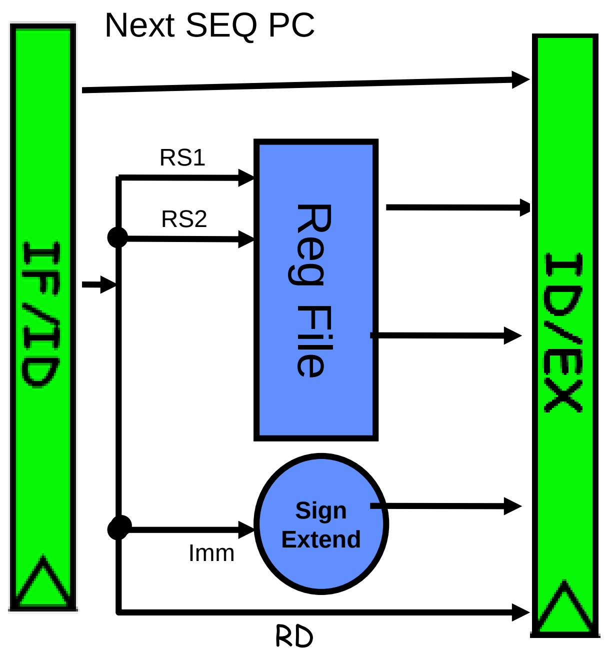


RISC MIPS Instruction Pipeline

- **Instruction decode/register fetch cycle (ID)**
  - Decode the instruction + register read operation
  - Fixed field decoding **[ADD R1,R2,R2] OR [LW R1,8(R2)]**

    Ex: A3.01.02.03 : 10100011 00000001 00000010 00000011
  - **Ex: 86.01.02.03 : 10000110 00000001 00001000 00000010**

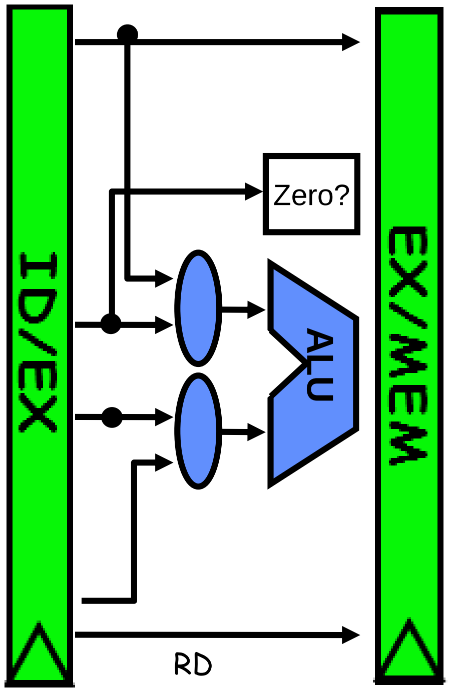
Instr. Decode Reg. Fetch



RISC MIPS Instruction Pipeline

- **Execution/Effective address cycle (EX)**
  - Memory reference: Calculate the effective address
    - **[LW R1,8(R2)]** **EFF ADDR= [R2] +8**
  - Register-register ALU instruction **[ADD R1,R2,R2]**

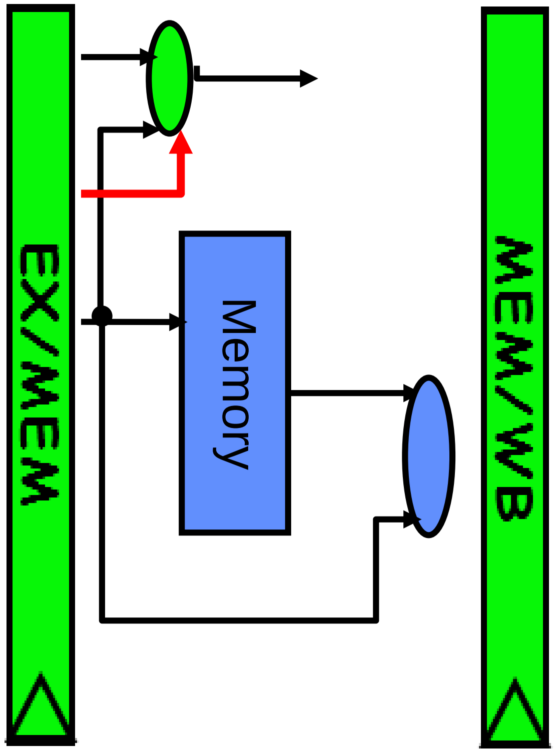
Execute Addr. Calc



RISC MIPS Instruction Pipeline

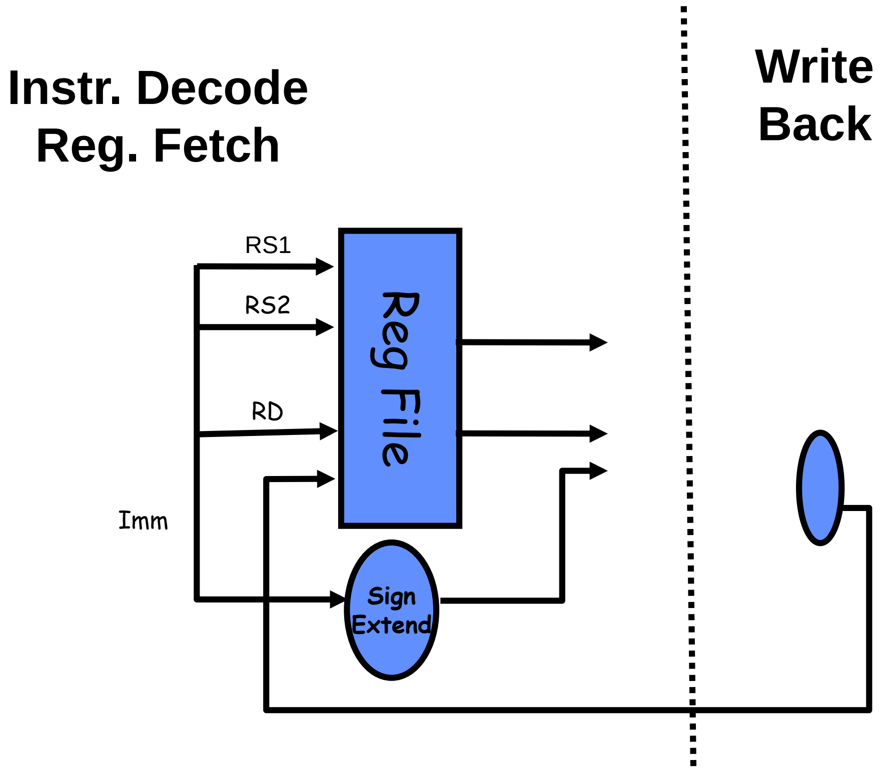
- **Memory access cycle (MEM)**
  - Load from memory and store in register **[LW R1,8(R2)]**
  - Store the data from the register to memory **[SW R3,16(R4)]**

Memory Access

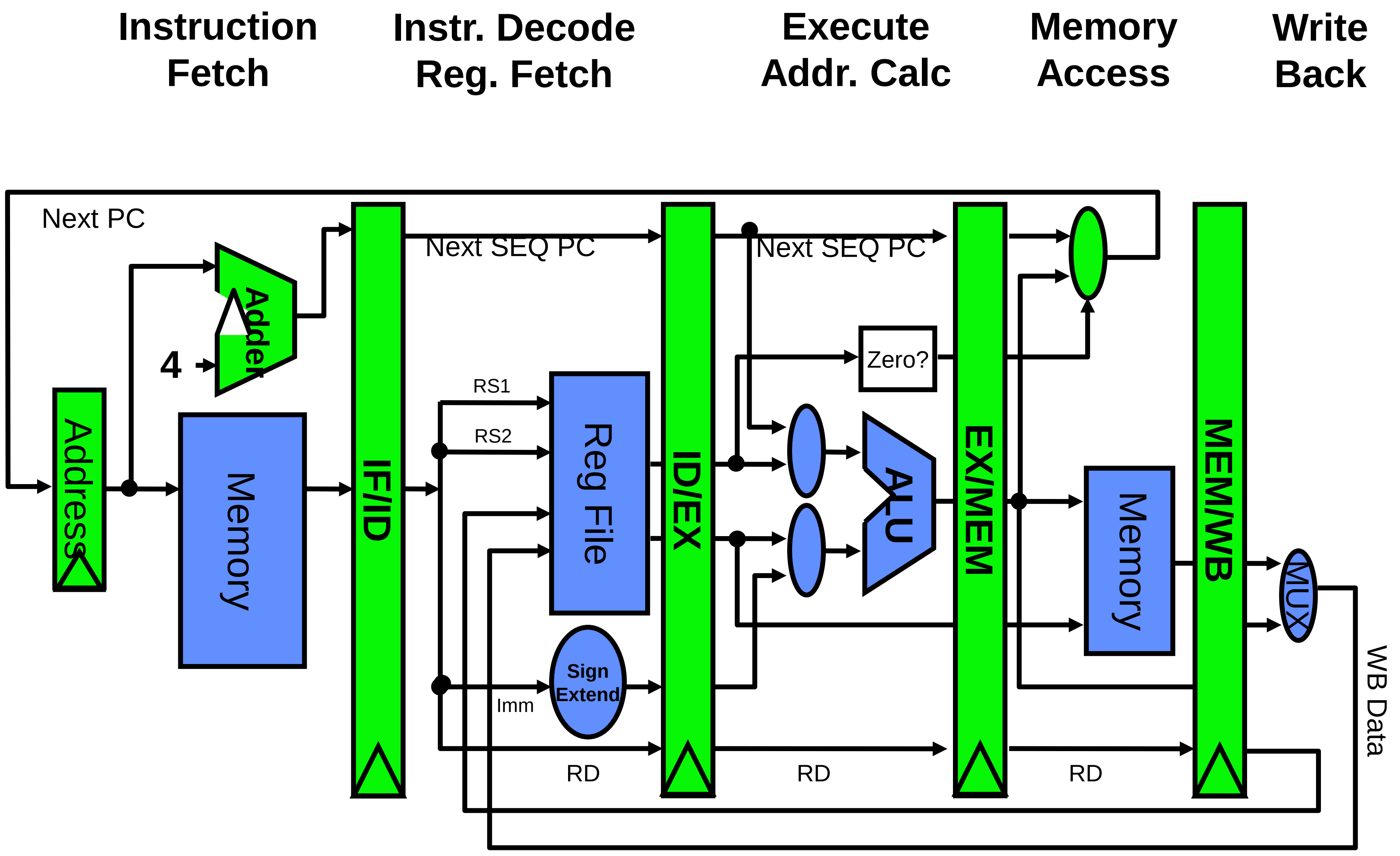


RISC Instruction Pipeline

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



Pipelined RISC Data path



5 Steps of RISC Data path

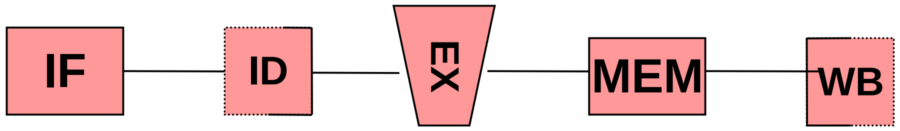
- Each instruction can take at most 5 clock cycles
- **Instruction fetch cycle (IF)**
  - Based on PC value, fetch the instruction from memory
  - Update PC
- **Instruction decode/register fetch cycle (ID)**
  - Decode the instruction + register read operation
  - Fixed field decoding
  - Equality check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

5 Steps of MIPS Data path

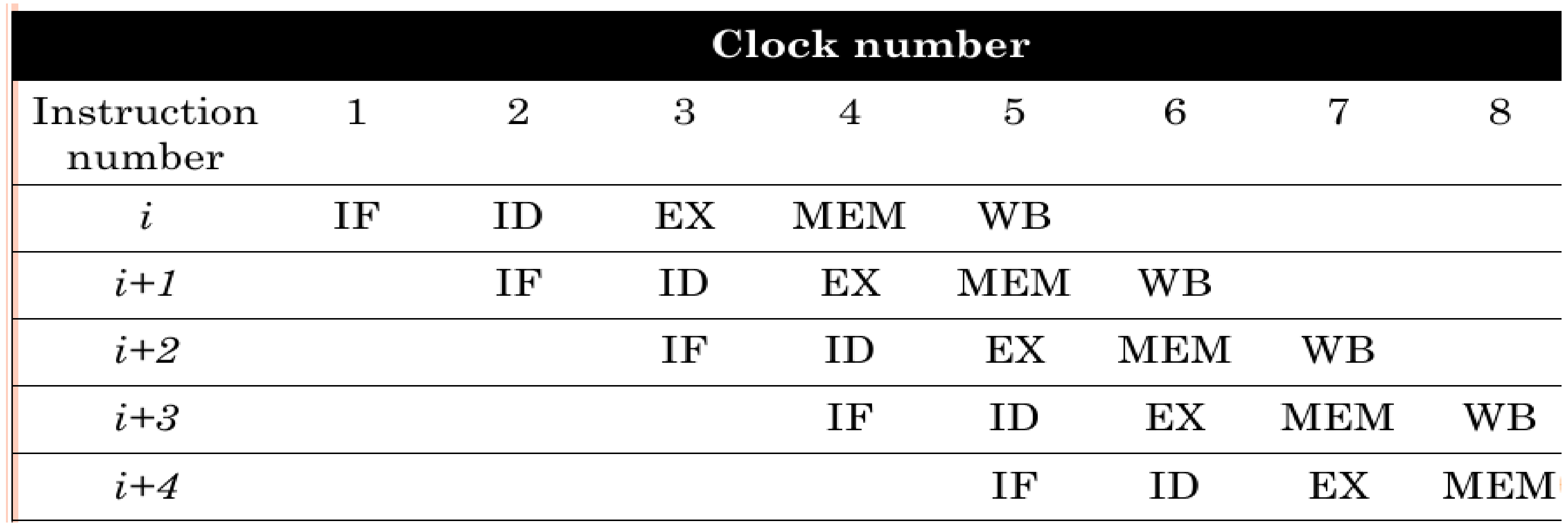
- **Execution/Effective address cycle (EX)**
  - Memory reference: Calculate the effective address
  - Register-register ALU instruction
  - Register-immediate ALU instruction
- **Memory access cycle (MEM)**
  - Load instruction: Read from memory using effective address
  - Store instruction: Write the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

5 Steps of MIPS Data path

- **Write-back cycle (WB)**
  - Register-register ALU instruction or load instruction
  - Write the result into the register file
- **Cycles required to implement different instructions**
  - Branch instructions – 4 cycles
  - Store instructions – 4 cycles
  - All other instructions – 5 cycles

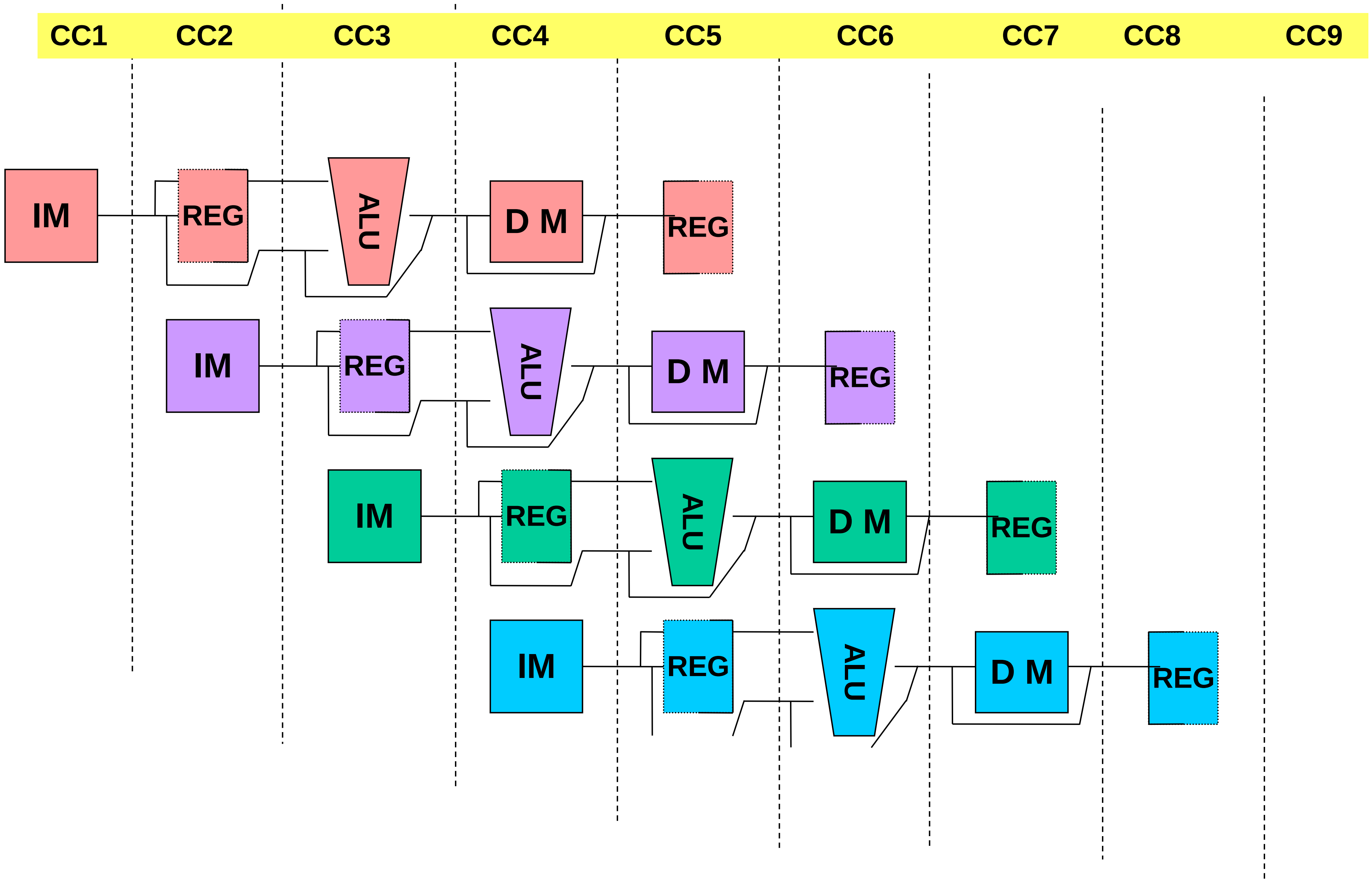


Visualizing Pipelining

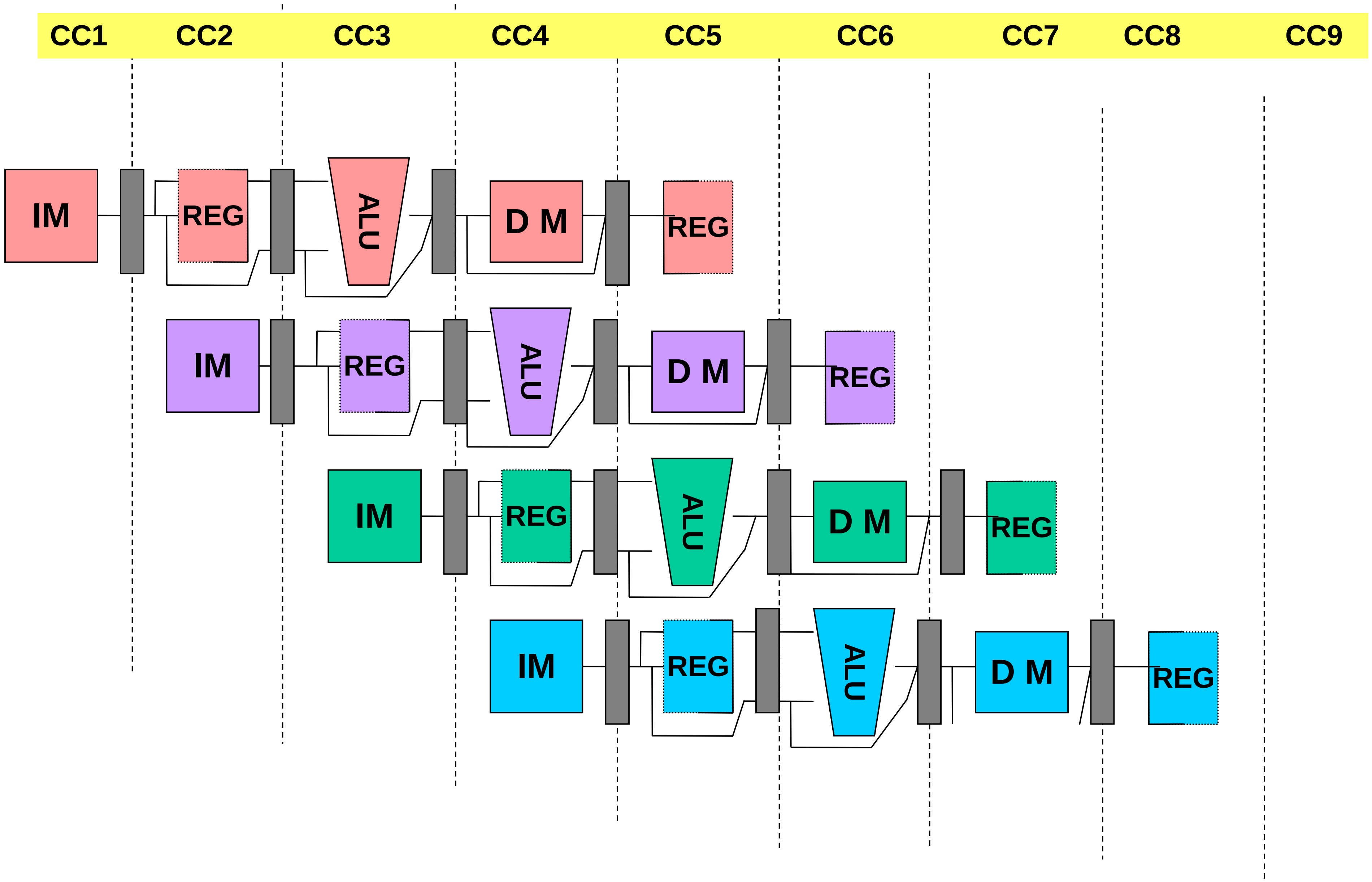


	Clock number							
Instruction number	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB
i+4					IF	ID	EX	MEM

Visualizing Pipelining



Visualizing Pipelining



Pipelining Issues

- **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

Pipelining Issues

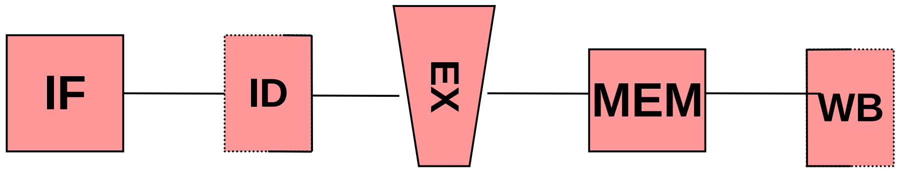
- **Ideal Case : Identical computations**
  - The same computation is to be performed repeatedly on a large number of input data sets
- **Reality: External fragmentation**
  - Some pipeline stages may not be used
- **Impact of ISA**
  - Reduce the complexity and diversity of the instruction types
  -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

Pipelining Issues

- **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

RISC MIPS Instruction Pipeline

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



johnjose@iitg.ac.in

http://www.iitg.ac.in/johnjose/