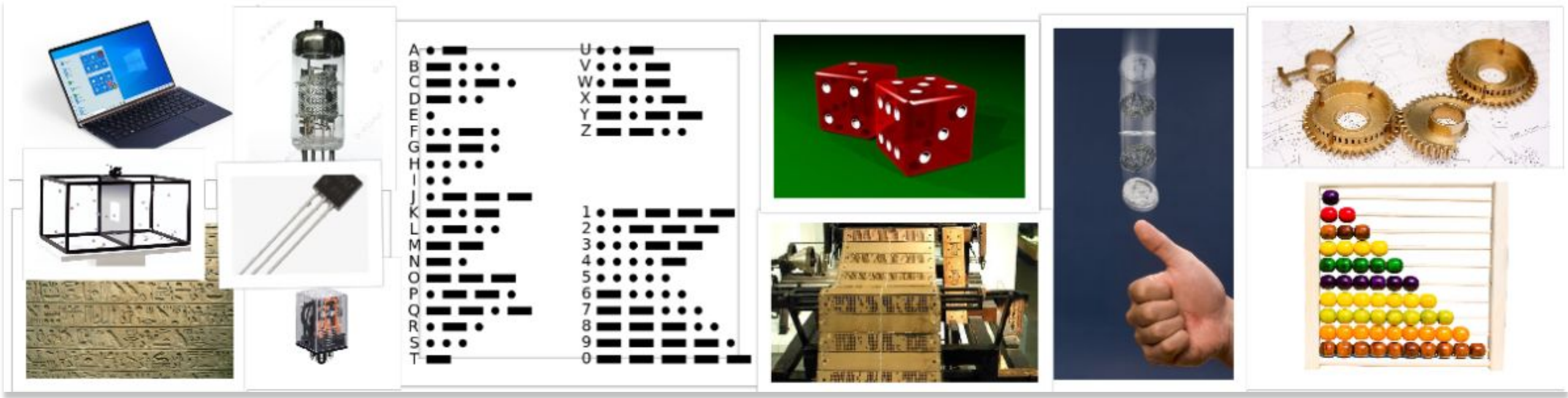


Computer Architecture

Design, Analysis, Execution and Optimization of Instructions

Datapath & CU for Pipelined Microprocessor: MIPS

Objectives

- Design the processor such that
  - Clock period (T) should be lesser than a single-cycle processor [similar to a multi-cycle one]
  - CPI should be 1

Comparison of Single & Multi-Cycle MIPS Processor

Performance analysis of Single-Cycle implementation

75

- XYZ-organization is contemplating building the single-cycle MIPS processor in a 65-nm CMOS manufacturing process. The organization has determined that the logic elements have the delays given in Table. Help the organization compute the execution time for a program with 100 billion instructions.
- $T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFSetup}$
  = 30 + 2(250) + 150 + 200 + 25 + 20 = **925 ps**
- The total execution time =
  (100 * 10^9 instrs.) * (1 cycle/instrs.) * (925 * 10^{-12} s/cycle)
  = 92.5 seconds

Parameter	Delay (ps)
t_{pcq_PC}	30
t_{mem}	250
t_{RFread}	150
t_{ALU}	200
t_{mux}	25
$t_{RFSetup}$	20

Other Benchmark: Standard Performance Evaluation Corporation (SPEC)
Princeton Application Repository for Shared-Memory Computers (PARSEC)
Stanford Parallel Applications for Shared-Memory (SPLASH-2) programs.

Performance Analysis of Multi-Cycle Implementation

64

- XYZ-organization is contemplating building the multi-cycle MIPS processor instead of the single-cycle processor. For both designs, the organization plans on using a 65-nm CMOS manufacturing process. The organization has determined that the logic elements have the delays given in Table. Help the organization compare each processor's execution time for a program with 100 billion instructions
- $T_c = t_{pcq\_PC} + t_{mux} + \max\{t_{ALU} + t_{mux} + t_{mux}, t_{mem}\} + t_{setup}$
- T_c = 30 + 25 + 250 + 20 = **325 ps**
- Execution time =
  (100 * 10^9 instrs.) * (**4.12** cycle/instrs.) * (325 * 10^{-12} s/cycle)
  = 133.9 seconds

Parameter	Delay (ps)
t_{pcq_PC}	30
t_{mem}	250
t_{RFread}	20
t_{ALU}	200
t_{mux}	25
t_{setup}	20

Problems of Multi-cycle Processor

- The fundamental problem
  - Split the slowest instruction, *lw*, 5-steps
  - The processor's clock cycle time does not improve 5-times, 185 ps
- The steps take unequal length of time
- Only one stage is busy and the remaining stages are idle
- 5-non-architectural registers and an additional multiplexer

Instructions	Multi-cycle (Clock-cycle)	Single-cycle (Clock-cycle)
LW	5	1
SW	4	1
R-type	4	1
BEQ	3	1
ADDI	4	1
J	3	1
CPI	**> 1**	**1**
CLK Period: T	**Lesser than Single-cycle**	**More than Multi-Cycle**

Single Cycle: Non-shared FUs, **CPI =1 or IPC=1**, clock period (T_{single})= slowest instr. in ISA

Multi-cycle: Shared FUs, CPI > 1 or IPC <1, **clock period**: $T_{multi} < T_{single}$

Can we have a microprocessor like: IPC=1 & clock period [< Tmulti < Tsingle]?

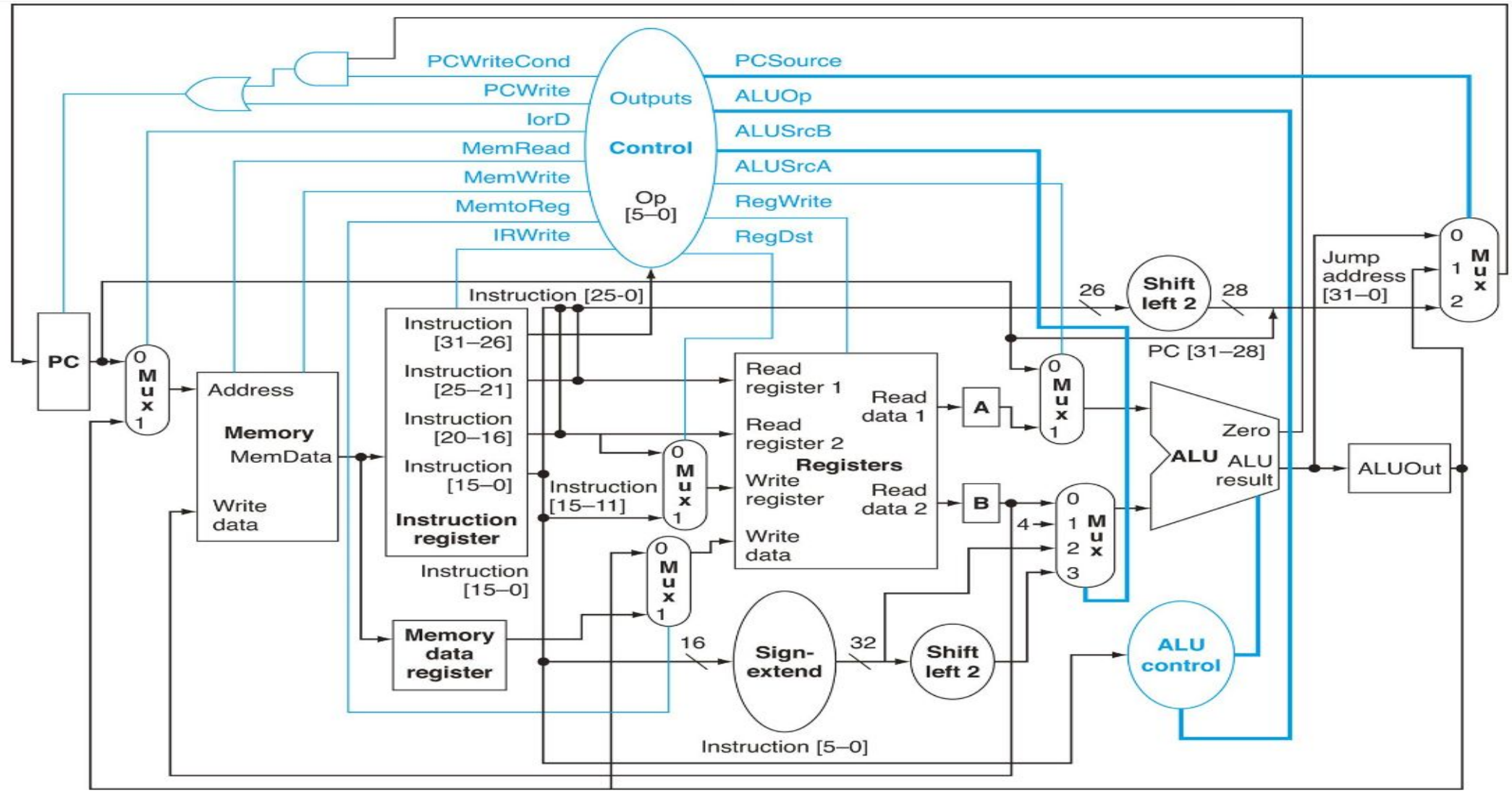
Cycles Per Instruction (CPI)

Program Execution time: #instr. x CPI x Clk (T)

Instructions Per Cycle/Seconds (IPC) = 1/CPI

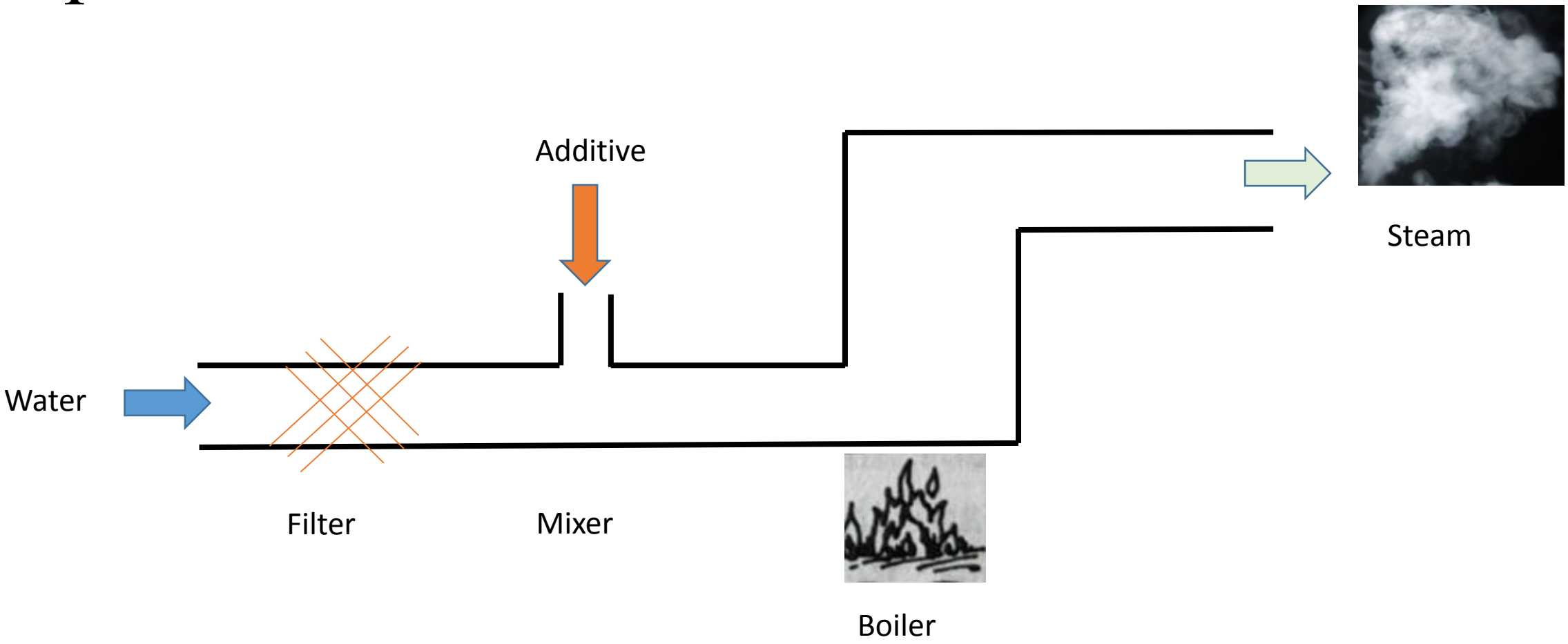
Problems of Multi-cycle Processor

Only one stage is busy and remaining stages are idle at anytime

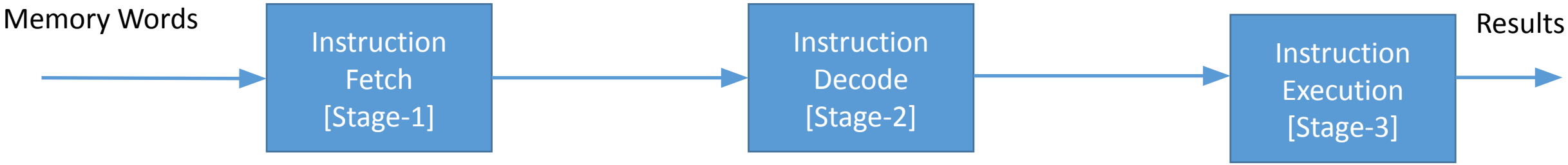


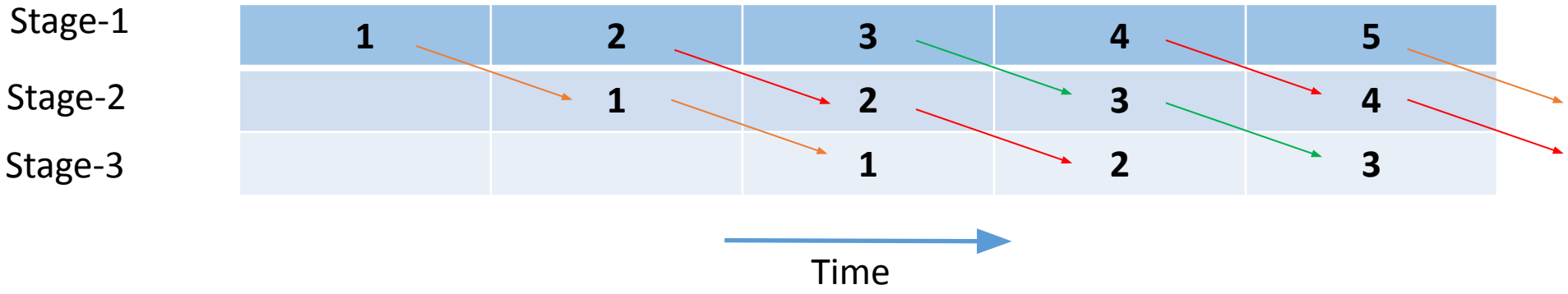
Book- P&H-COD

Pipeline in a Chemical Plant



Pipeline in the Instruction Execution





What is the difference in this Analogical or Parallel reasoning?

Pipelined MIPS-based processor

- Partitioning the Instruction Executional cycle (function)
  - Subfunctions
  - Input of one subfunction TOTALLY comes from output of previous subfunctions
  - Other than inputs & outputs, there are no interrelationships between subfunctions
  - Hardware may be developed (stage) to execute each subfunction
  - Each hardware units' evaluations are usually approximately equal

Pipelined MIPS-based processor

- Powerful way to improve the *throughput*
- Divide the single-cycle implementation
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- A commercial MIPS processor: R2000/R3000

- Partitioning the Instruction Executional cycle (function)
  - Subfunctions
  - Input of one subfunction TOTALLY comes from output of previous subfunctions
  - Other than inputs & outputs, there are no interrelationships between subfunctions
  - Hardware may be developed (stage) to execute each subfunction
  - Each hardware units' evaluations are usually approximately equal

Latency of each instructions is unchanged, but throughput is ideally 5-times better

Pipelined MIPS-based processor

- Stage elements
  - Reading & writing the memory
  - Register file
  - ALU operation
- Each stage takes almost same amount of time
  - Consists of one element

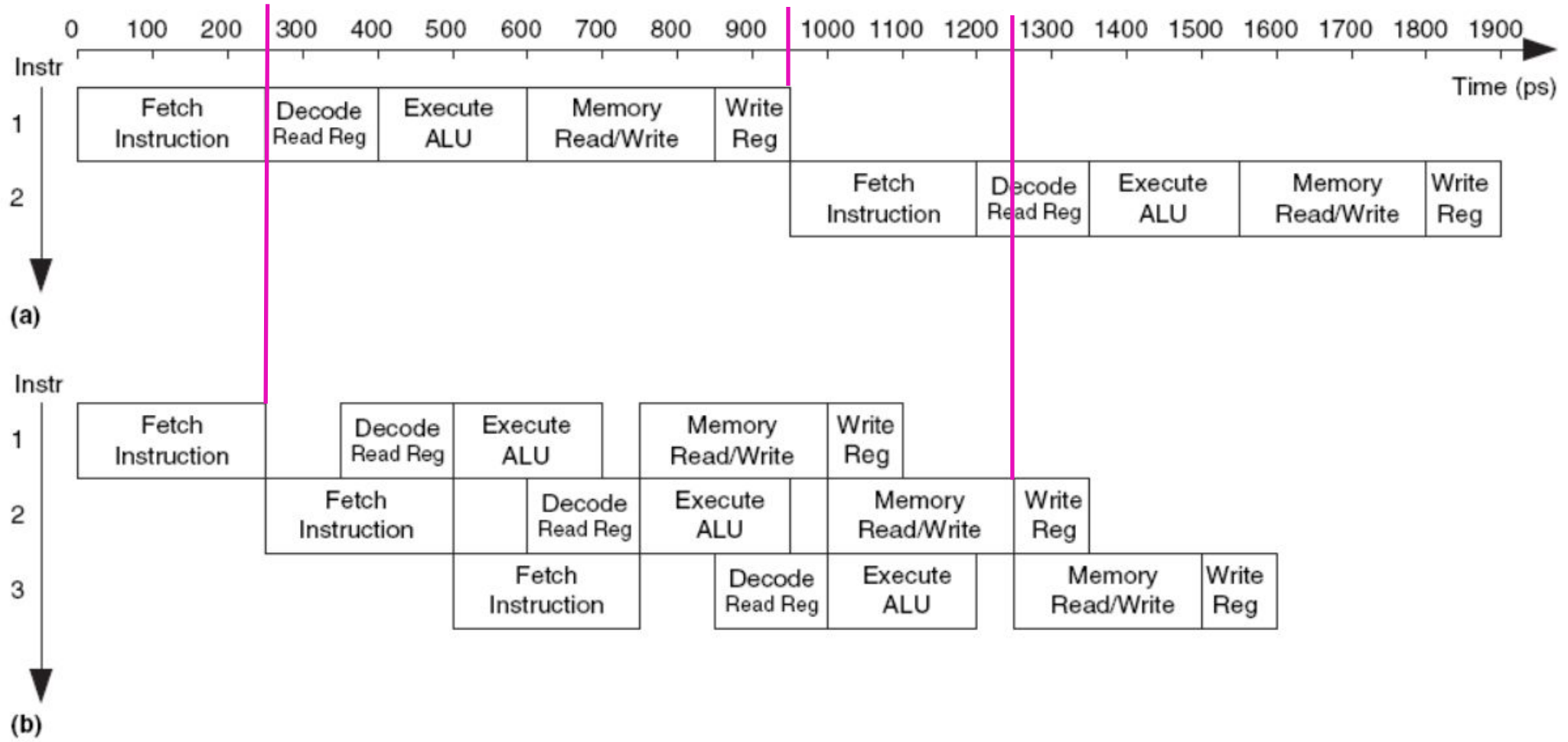
Comparison of timing diagram

- Delay of the elements

Element	Parameter	Delay (ps)
Register clk-to-Q	T_{pcq}	30
Register setup	T_{setup}	20
Multiplexer	T_{mux}	25
ALU	T_{ALU}	200
Memory read	T_{mem}	250
Register file read	t_{RFread}	150
Register file write	t_{RWrite}	100
Register file setup	$t_{RFsetup}$	20

Comparison of timing diagram

- Delay of MUX & register is not included



Timing diagram of (a) single-cycle processor (b) pipelined processor

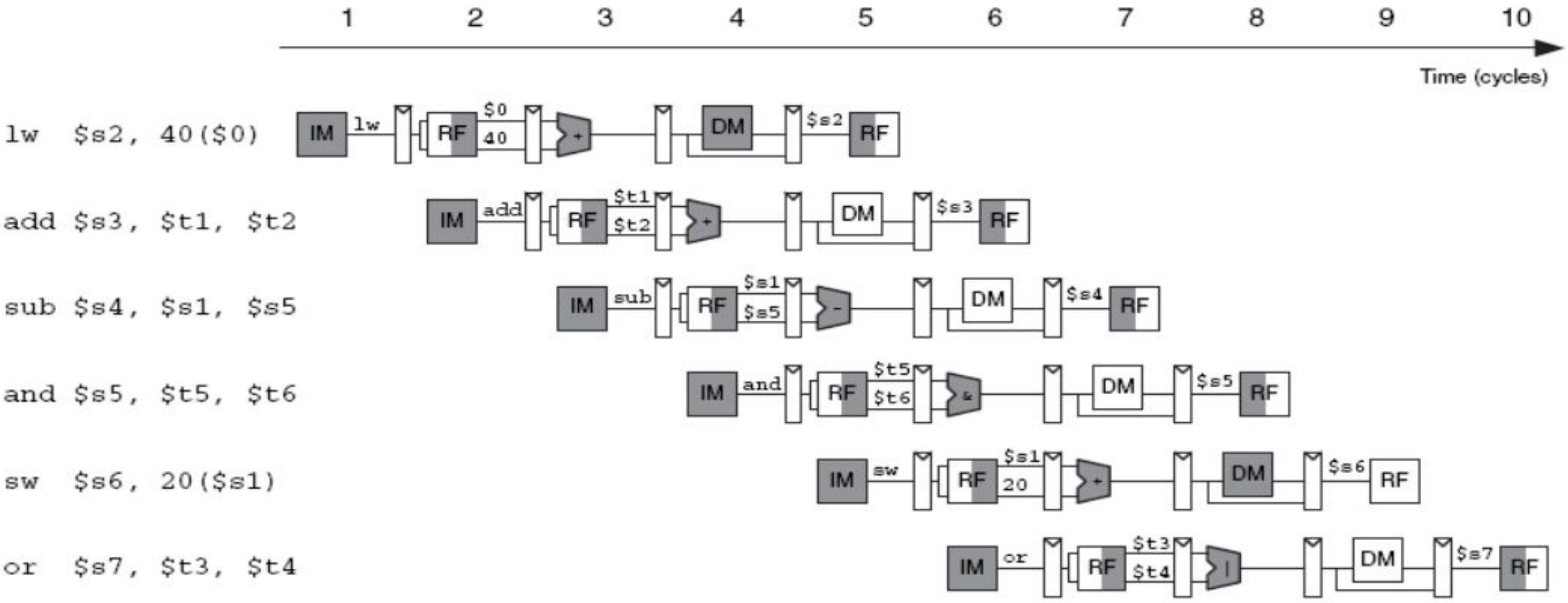
Book- P&H-COD

Comparison of timings

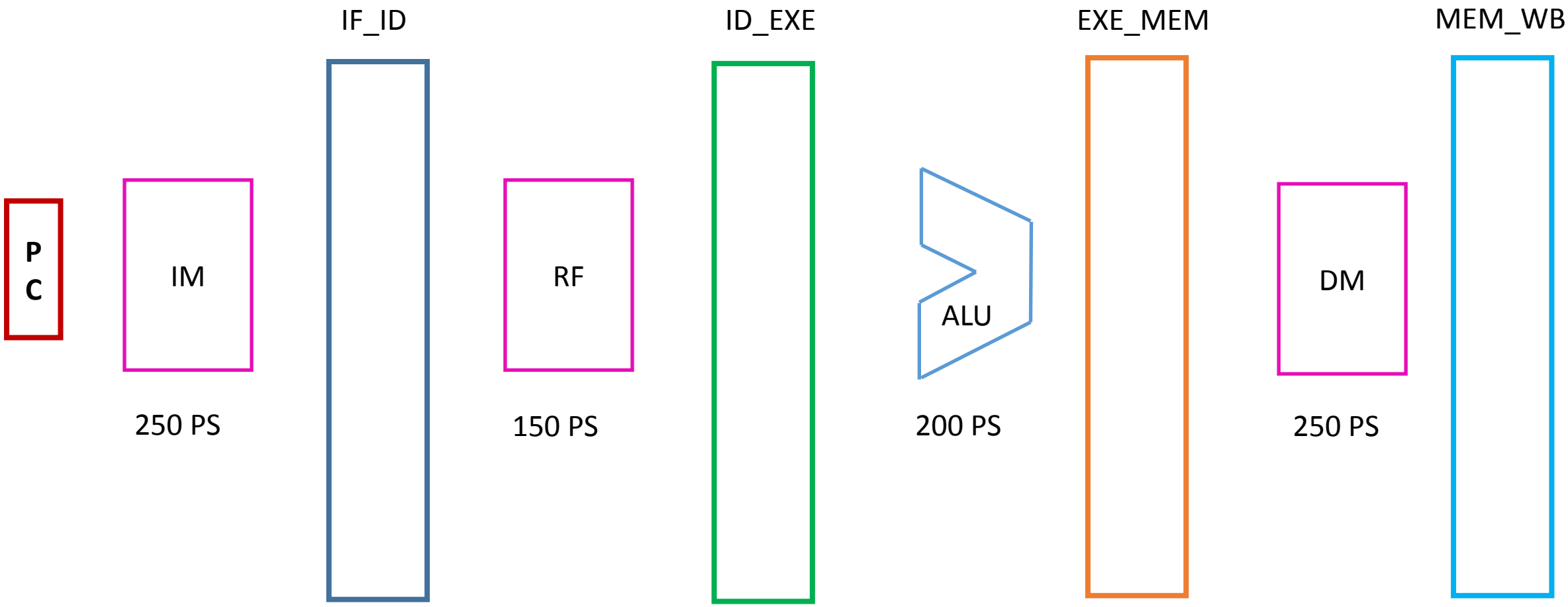
- **Single-cycle processor**
  - Instruction latency is 950 ps
  - Throughput 1 instruction per 950 ps
  - 1.05 billions instruction per second
- **Pipelined processor**
  - Length of pipeline stage is 250 ps (mem. access)
  - Instruction latency is 5*250 = 1250 ps
  - Throughput 1 instruction per 250 ps
  - 4 billions instructions per seconds

A view of pipeline in operation

- Resource utilization



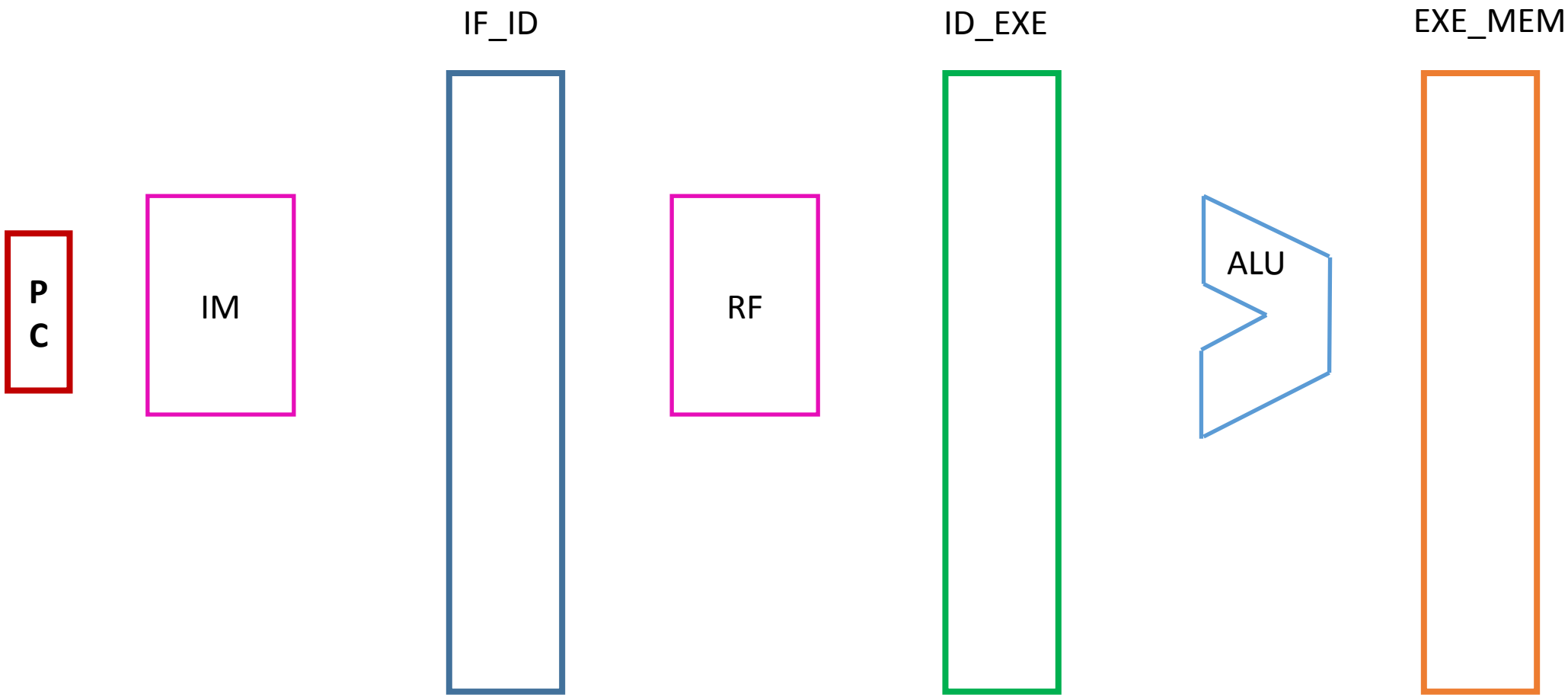
Delay elements and stage registers



Delay values are taken from the previous table.

Datapath for R-type: ADD R1, R2, R3

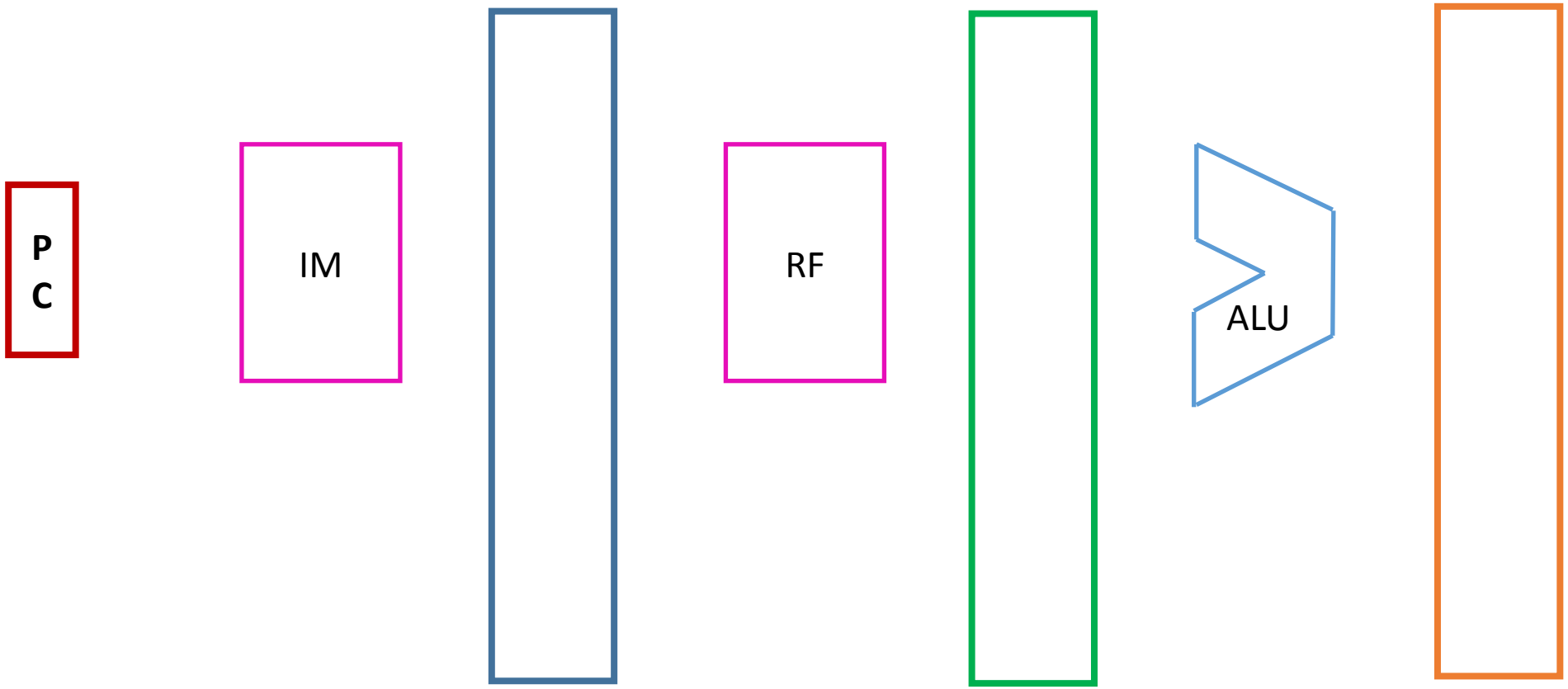
op	rs	rt	rd	shamt	funct
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)



Datapath for B-type: BEQ R1, R3, offset

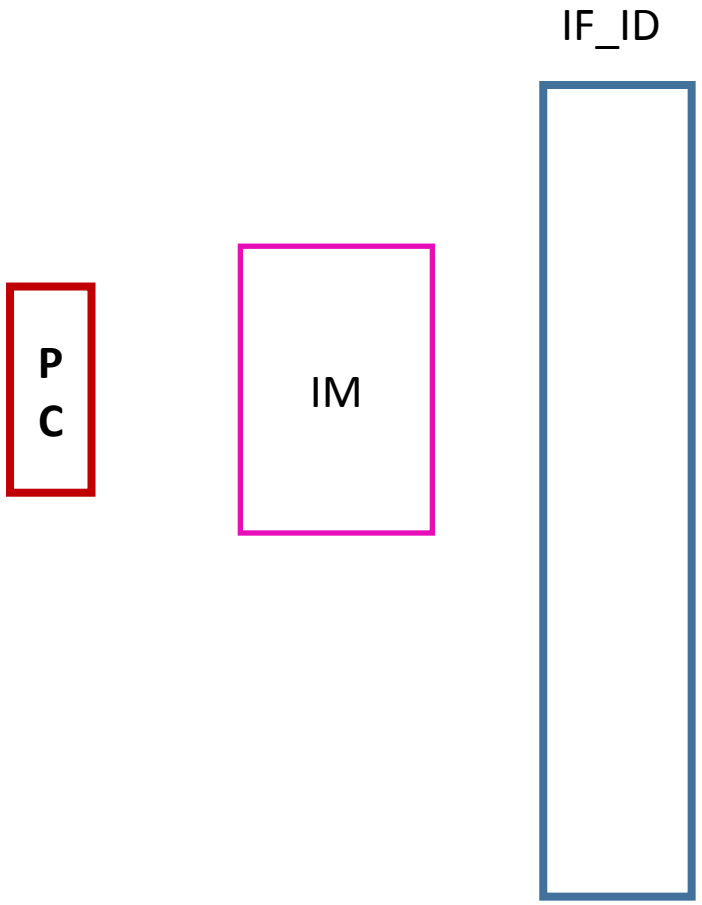
op	rs	rt	offset		
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)

IF_ID ID_EXE EXE_MEM



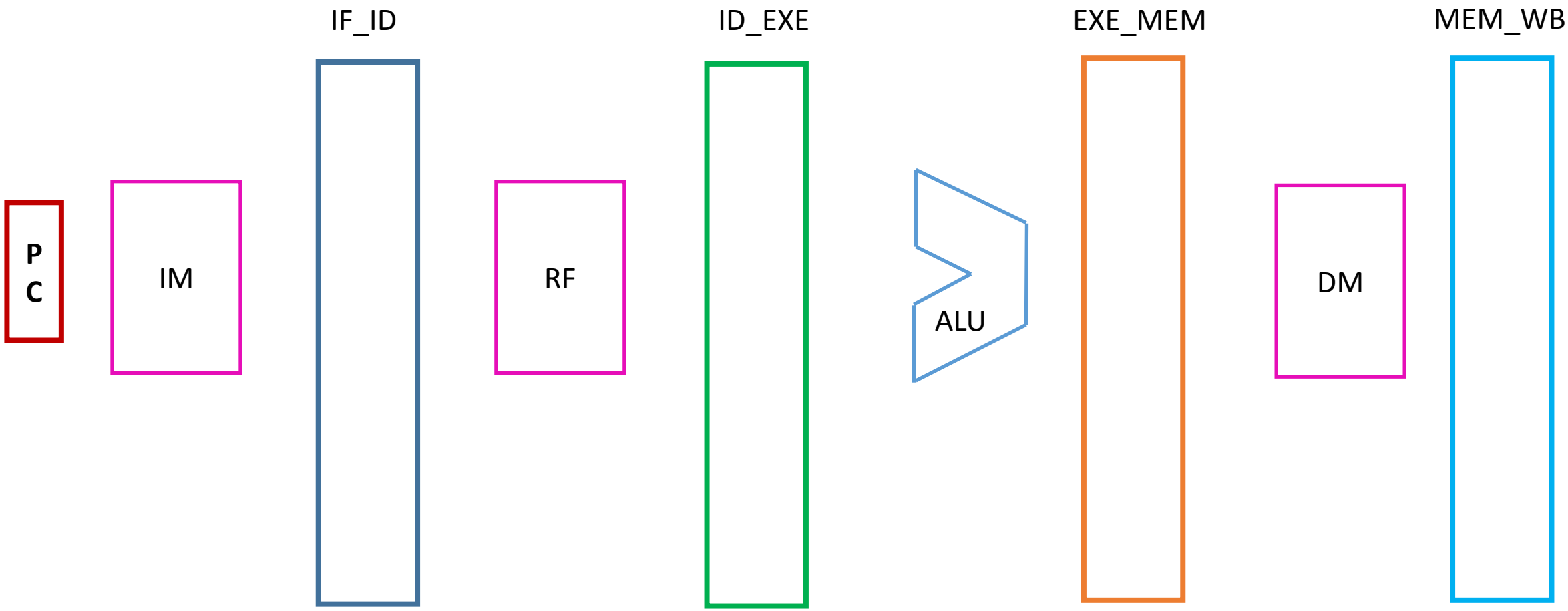
Datapath for J-type: J Offset

op	address				
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)



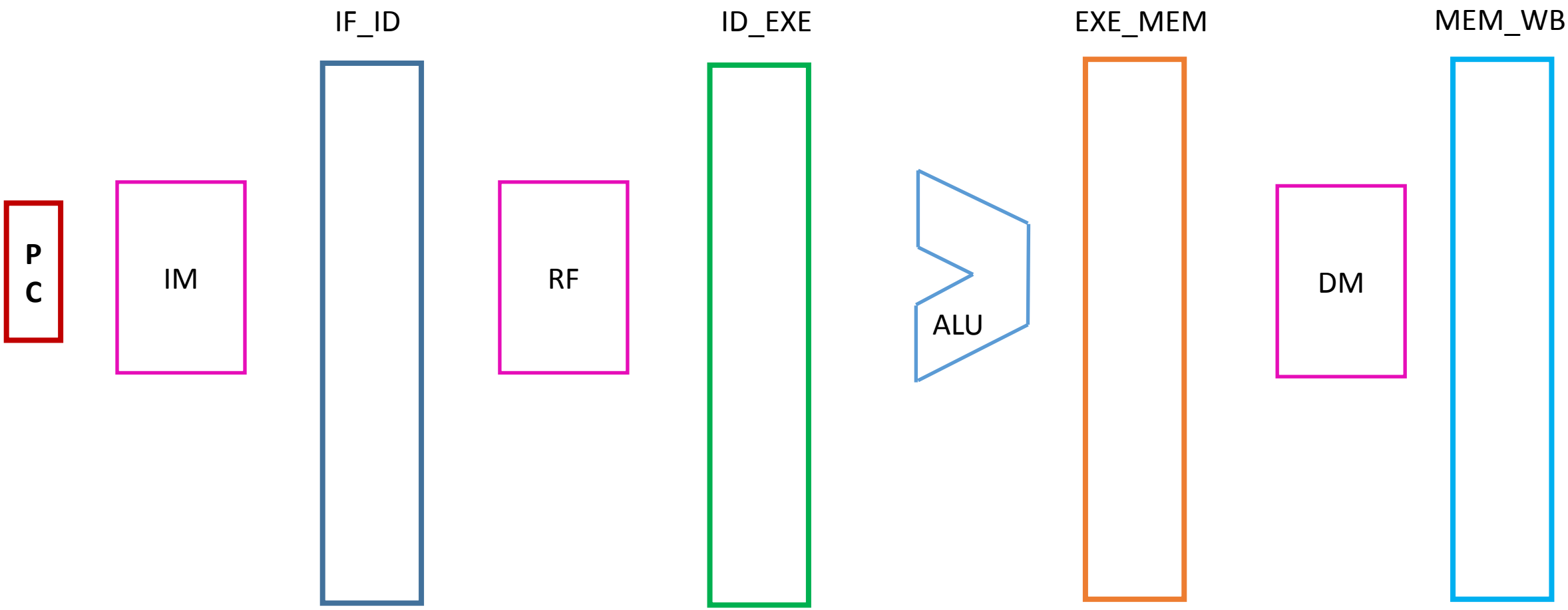
Datapath for I-type: LW R1, #5(R3)

op	rs	rt	Offset		
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)



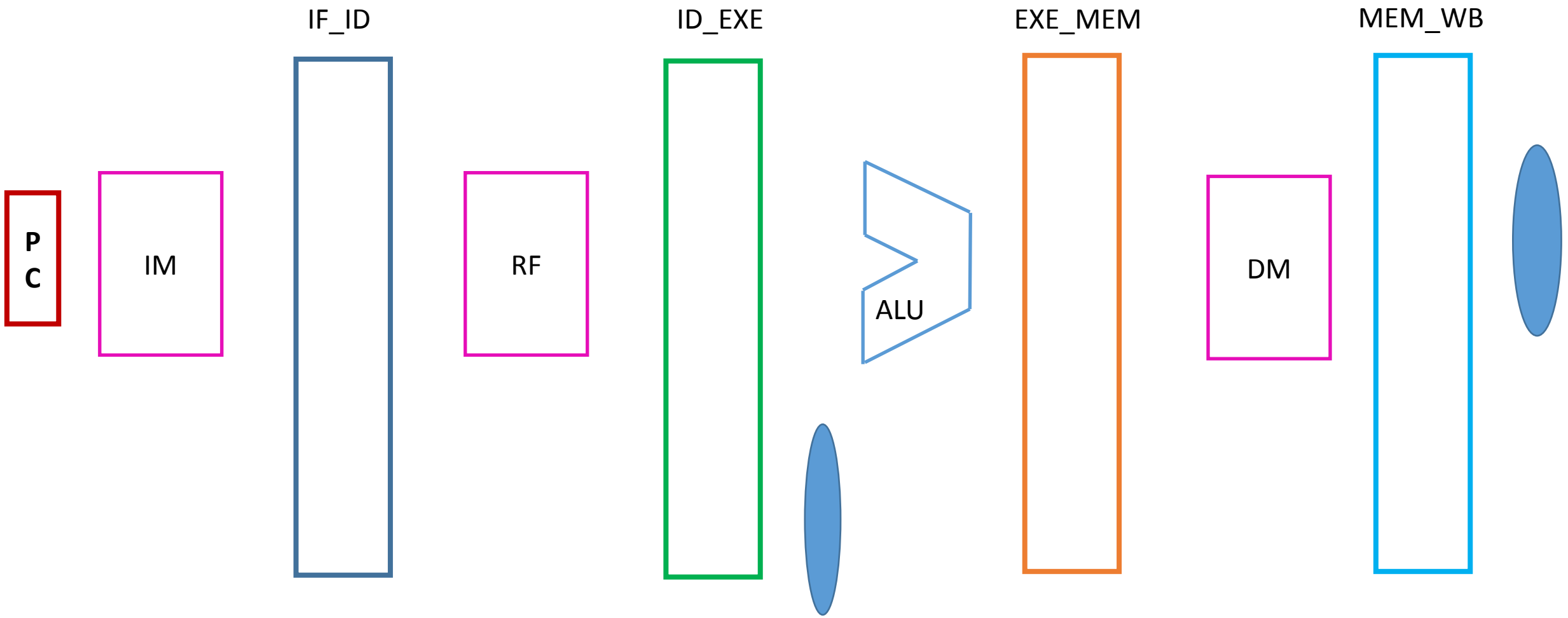
Datapath for I-type: SW R1, #5(R3)

op	rs	rt	Offset		
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)



Datapath for LW R1, #5(R3) & R-type

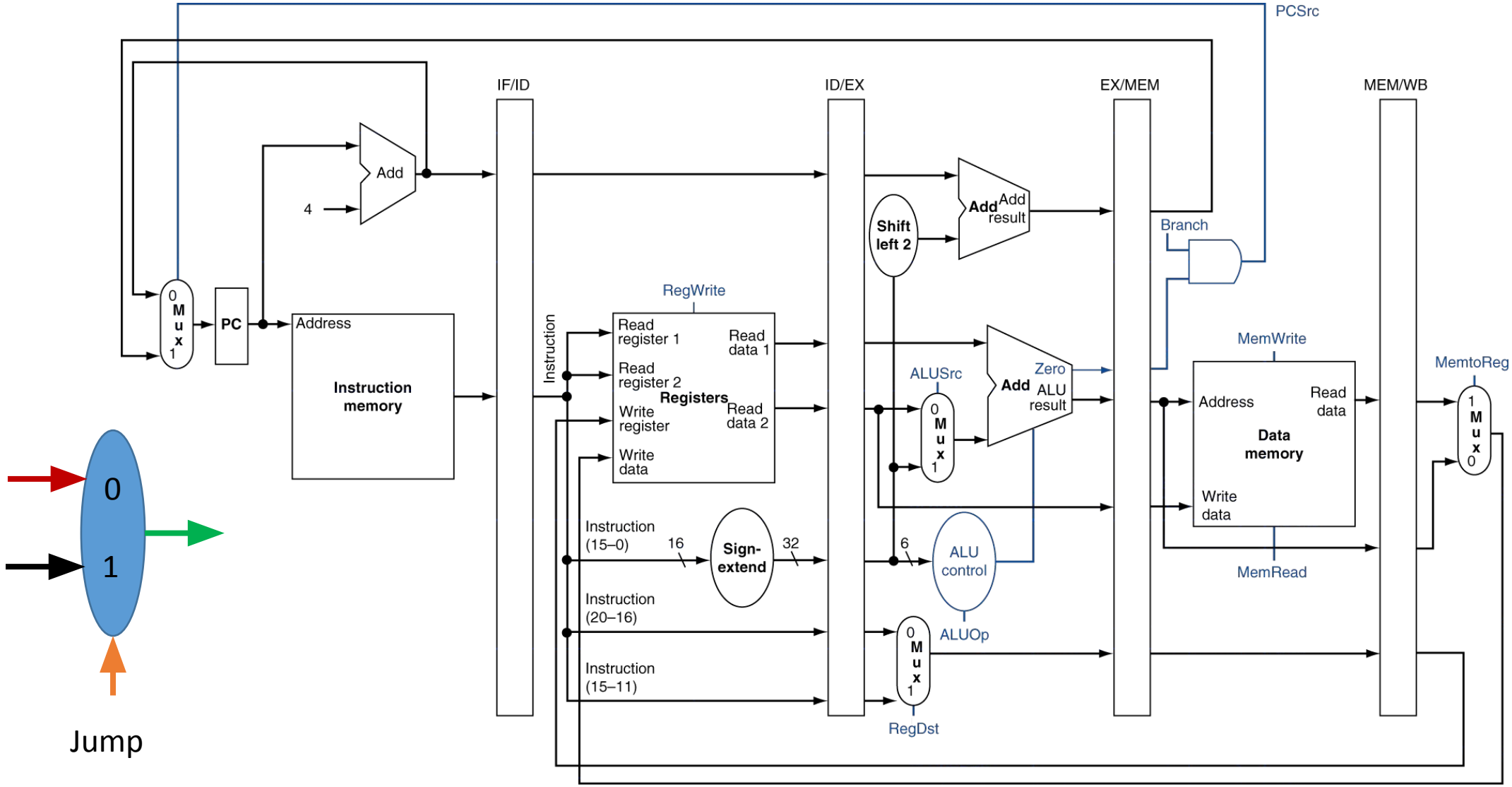
op	rs	rt	rd	shamt	funct
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)



Combined Datapath

- Stages
  - Insert multiplexer
- Stage registers
  - Union of registers added for each instruction

Combined Datapath



Jump

Book- P&H-COD

Control unit for Pipelined MIPS processor

- Identify the control signals
  - Jump
  - RegDst
  - RegWrite
  - ALUSrc
  - Branch
  - ALUOp
  - MemRead
  - MemWrite
  - MemtoReg

Fetch	Decode	Execute	Memory	Write Back
jump		ALUSrc	Branch	MemtoReg
		ALUOp	MemRead	RegWrite
		RegDst	MemWrite	

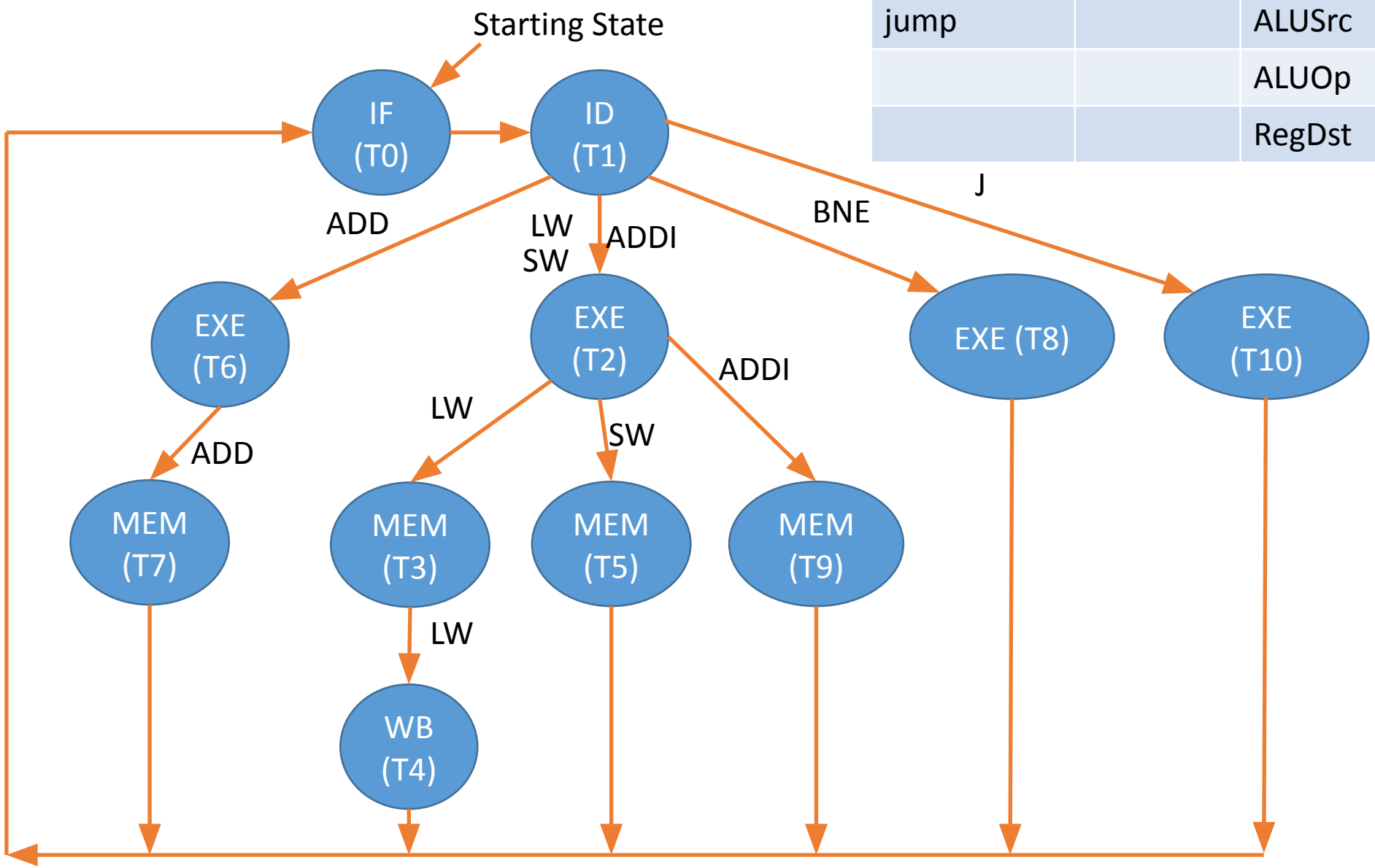
- How does one generate the control signals?
  - We want to add a minarals liquid with the water flowing through a pipeline after every 1 KM
  - How?

Control generation: Single-cycle

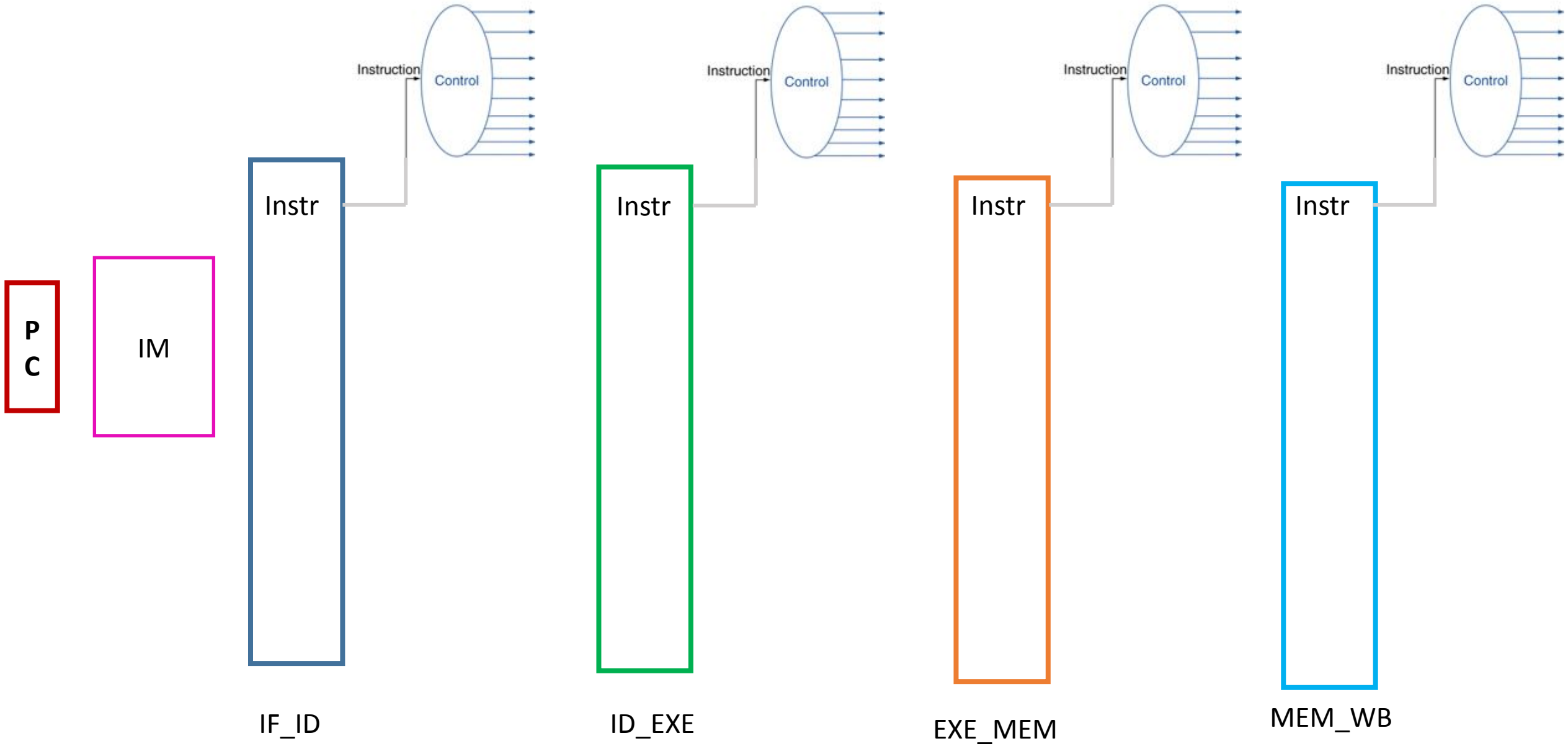
Instr.	Jump	RegDst	RegWrite	ALUSrc	Branch	ALUOp1	ALUOp0	MemRead	MemWrite	MemtoReg
R-type	0	1	1	0	0	1	0	0	0	0
lw	0	0	1	1	0	0	0	1	0	1
sw	0	x	0	1	0	0	0	0	1	x
addi	0	0	1	1	0	0	0	0	0	0
B-type	0	x	0	0	1	0	1	0	0	x
J-type	1	x	0	x	x	x	x	0	0	x

Control generation: Multi-cycle

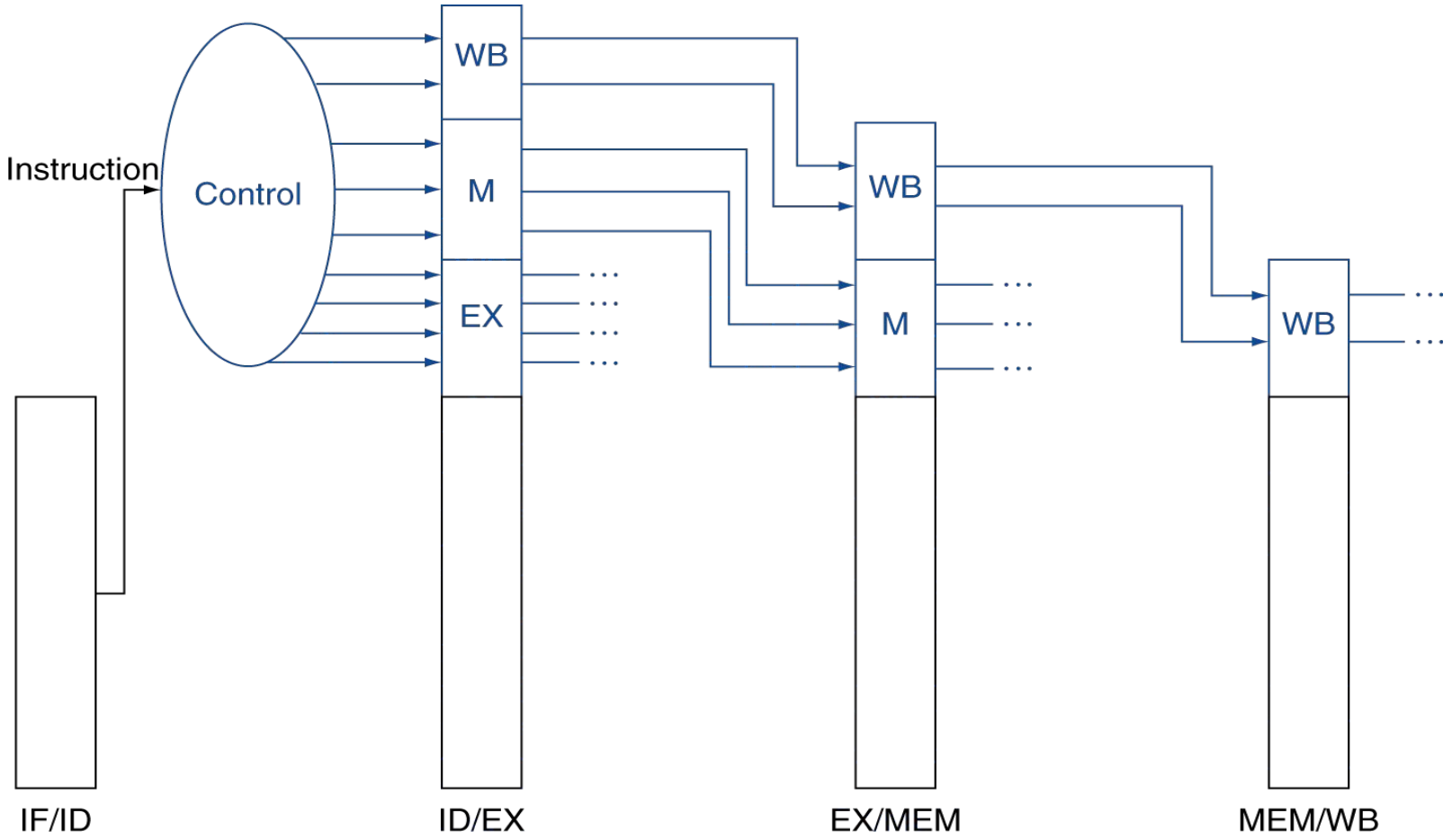
Fetch	Decode	Execute	Memory	Write Back
jump		ALUSrc	Branch	MemtoReg
		ALUOp	MemRead	RegWrite
		RegDst	MemWrite	



Control generation: Strategy - 1



Control generation: Strategy - 2



Book- P&H-COD

Control unit for Pipelined MIPS processor

Instr	Execution/Address Calc stage control lines					Memory access stage control lines			Write-back control lines	
Instr	**Jump**	**RegDst**	**ALUOp1**	**ALUOp0**	**ALUSrc**	**Branch**	**MemRead**	**MemWrite**	**RegWrite**	**MemtoReg**
R-format	0	1	1	0	0	0	0	0	1	0
lw	0	0	0	0	1	0	1	0	1	1
sw	0	x	0	0	1	0	1	0	0	x
beq	0	x	0	1	0	1	0	0	0	x

Single cycle MIPS processor

Instr	Jump	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg
R-format	0	1	1	0	0	0	0	0	1	0
lw	0	0	0	0	1	0	1	0	1	1
sw	0	x	0	0	1	0	1	0	0	x
beq	0	x	0	1	0	1	0	0	0	x

Control unit for Pipelined MIPS processor

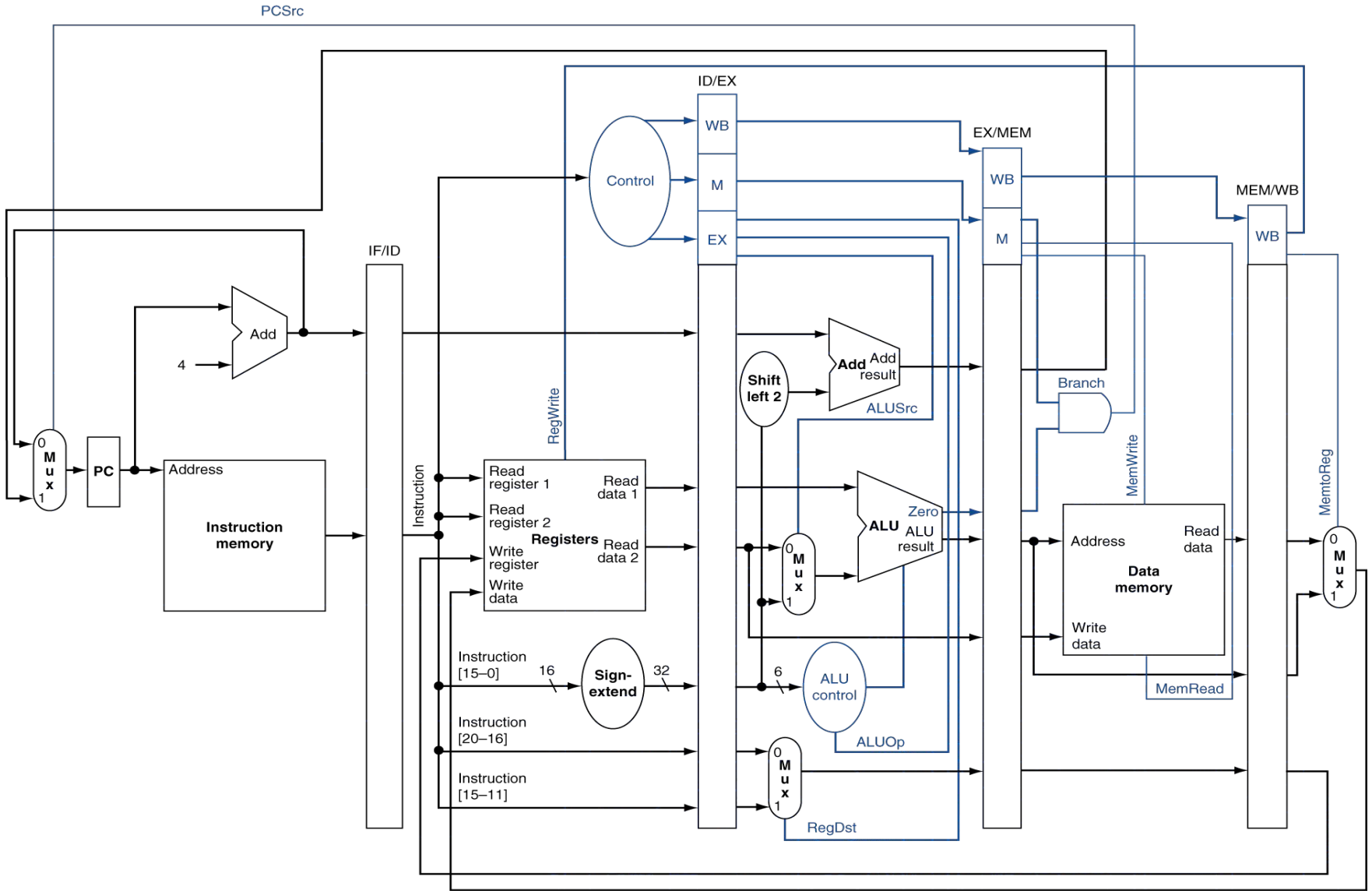
- How to generate such control signals?
  - Settings the 10 control lines in each stage for each instruction
  - Simplest way is same as in single cycle
  - Most the controls can be generated at the same time or decoding stage

- How to manage the control signals generated for i-th instruction and control signal will be generated for (i+1)-th instructions?
  - Erroneous control signals can be generated

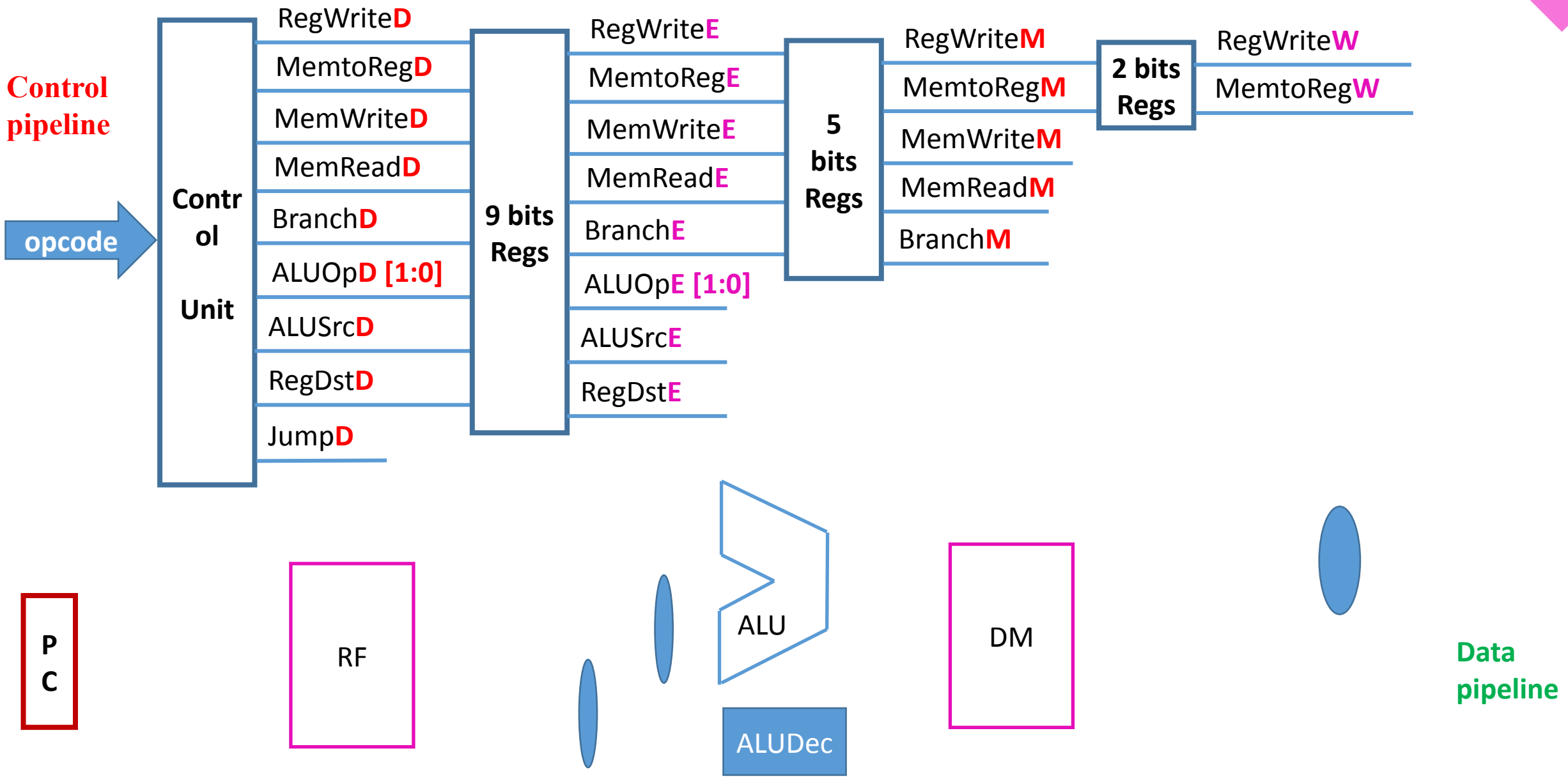
Control unit for Pipelined MIPS processor

- How to manage the control signals generated for i-th instruction and control signal will be generated for (i+1)-th instructions?
  - Erroneous control signals can be generated
  - Extension of the pipeline registers for storing the control signals' values

Pipelined Datapath & Control



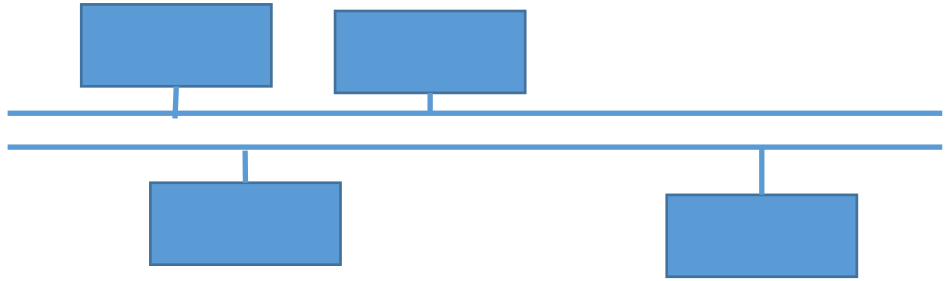
Pipelined Control Signals

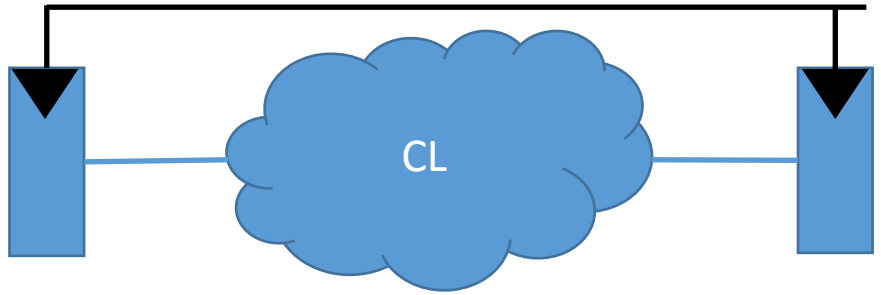


Designing Instruction Sets for Pipelining

- MIPS's instructions are same length
- X86's instructions vary 1 byte to 15 byte, is pipelining challenging ??
- MIPS has a few addressing modes
- Memory operands only appear in loads or stores in MIPS
- Operand are aligned in memory

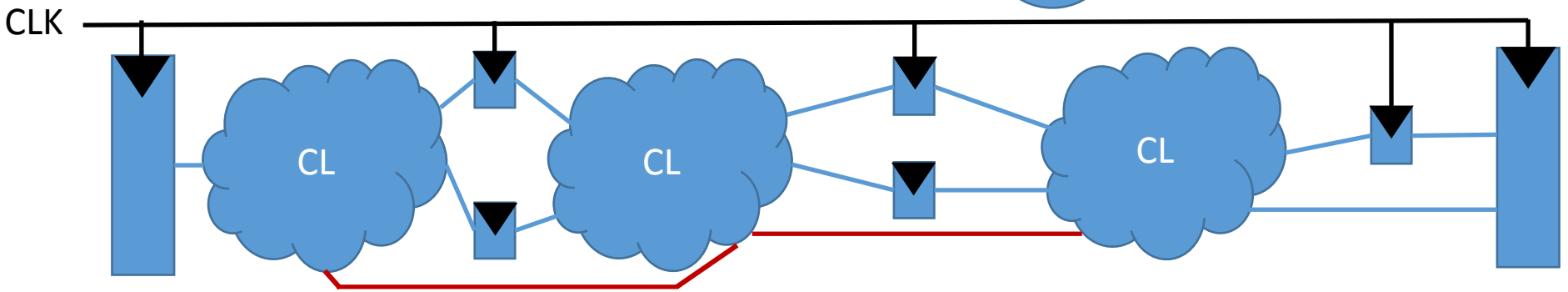
Comparison of datapaths





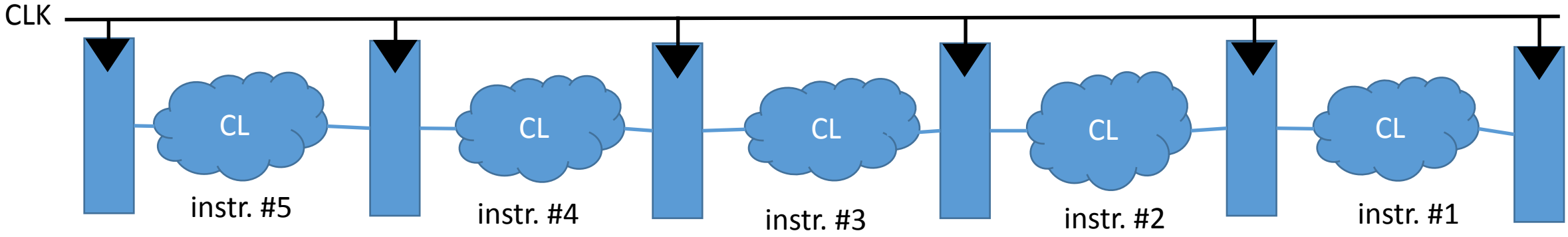
CL: Combinational Logic

Only one instr. In the datapath at an instant of time



Only one instr.

What if one instruction is here.



Microprocessor Design Trade-offs: Interconnects Vs Functional Units Vs IPC

- (clock) Cycle Per Instruction (CPI)
- Instructions Per (clock) Cycle/Seconds (IPC) = 1/CPI

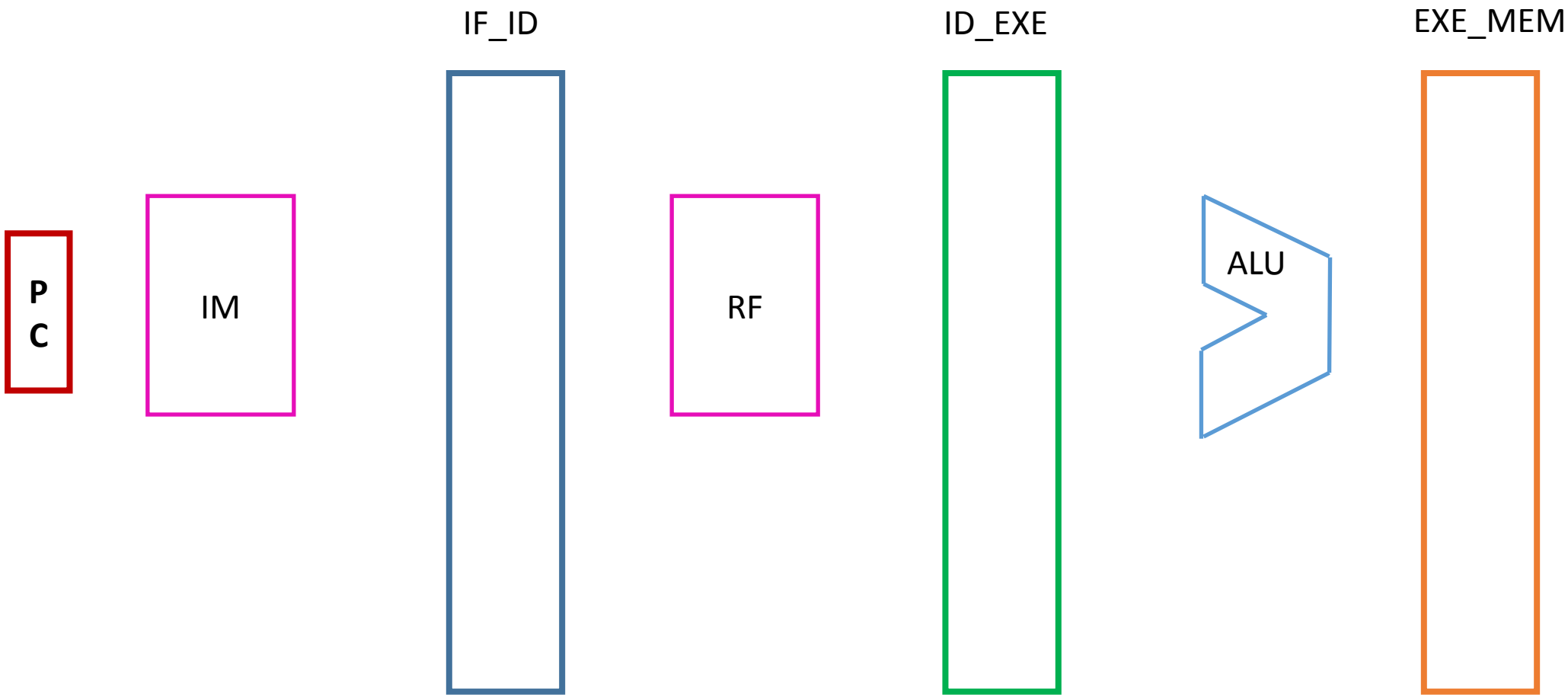
Interconnects (Bus)	Functional Units (FUs)	
	Less	More
Less	Single-bus & Single-FU (Multi-Cycle, IPC < 1)	Single-bus & Many-FUs (Multi-Cycle, IPC < 1)
More	Many-bus & Single-FU (Multi-Cycle, IPC < 1)	Many-bus & Many-FUs (Single-Cycle or Pipeline, IPC = 1)

Methods/Algorithms:
1) Multi-Cycle
2) Single-Cycle
3) Pipelined

- Pipeline: IPC = 1 (borrowed from Single-Cycle) and less clock period (T) (borrowed from Multi-Cycle), shared the Buses & FUs by more than one instruction.
- **Program Execution time: #instr. x (1/IPC) x Clk (T)**
- Can we have the IPC > 1?

Datapath for R-type: ADD R1, R2, R3

op	rs	rt	rd	shamt	funct
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)



Reg. File's write operation @posedge

&

Stage Reg.'s write operation @negedge

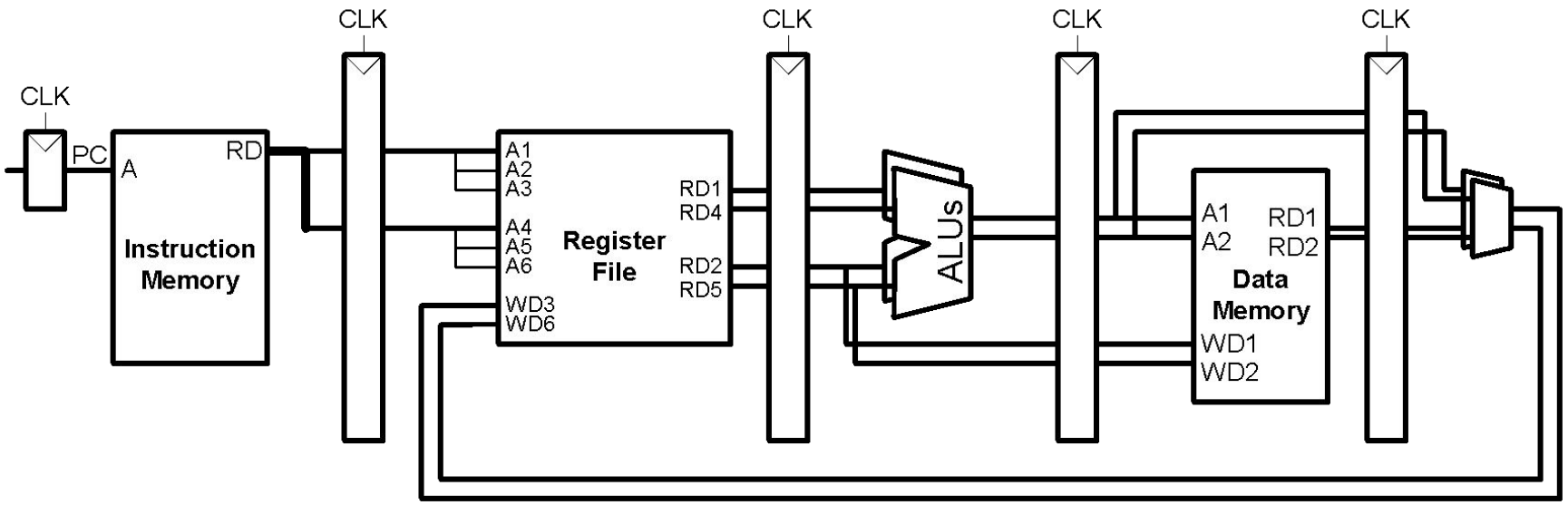
Can we have IPC > 1?

Multiple issue processor

Superscalar and VLIW

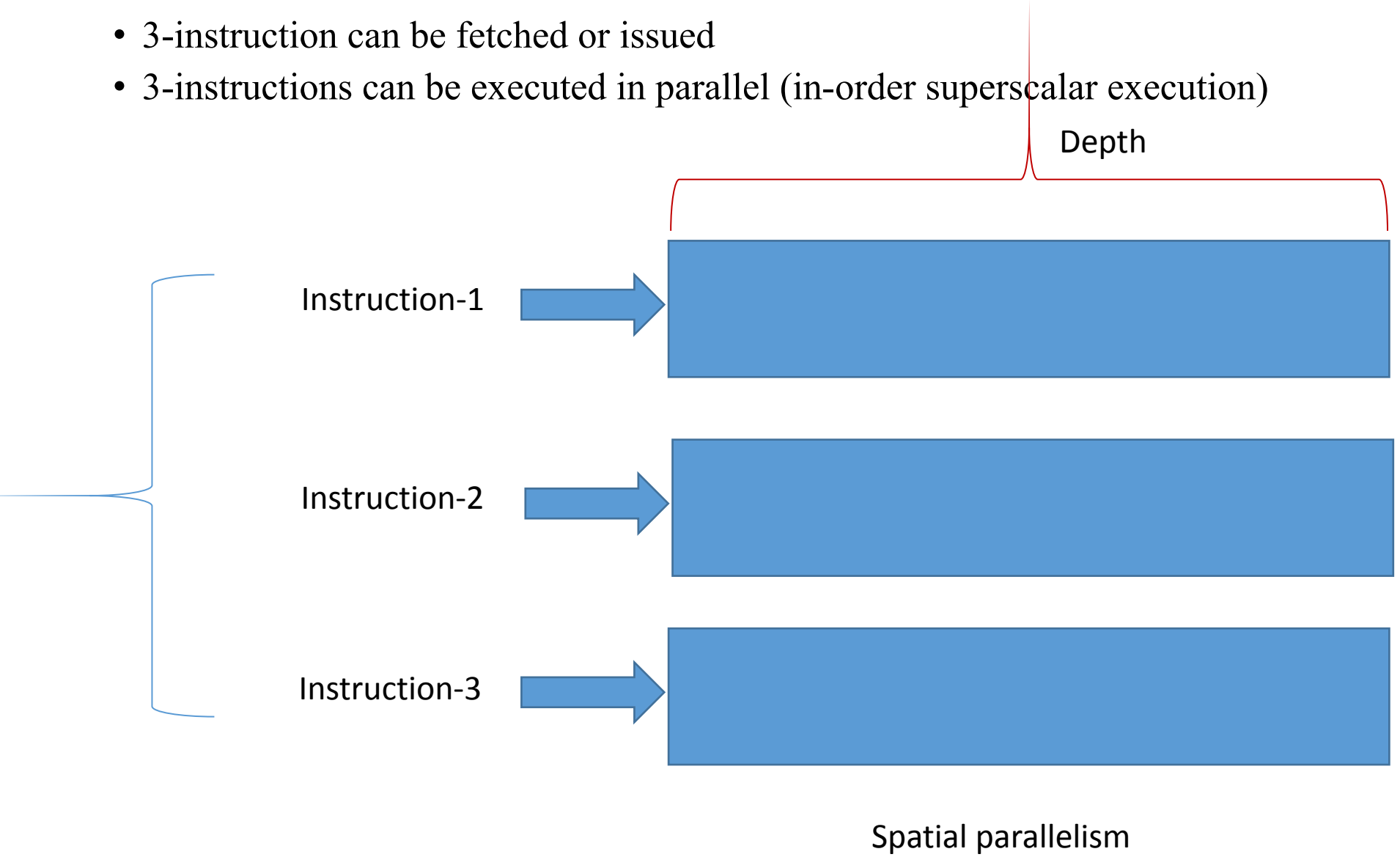
IPC = 2, Superscalar or multiple issue processor

IPC = 2, Superscalar or multiple issue processor



IPC = 3, Superscalar or 3 issue processor

- 3-instruction can be fetched or issued
- 3-instructions can be executed in parallel (in-order superscalar execution)



Static Multiple Issue MIPS Processor

- Two issue Instruction
- Integer ALU operations – using one datapath
  - ADD, BNE, etc
- Data transfer operations – using 2nd datapath
  - LW & SW
- Instruction format

Static Multiple Issue Processor

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

Book-COD-P&H, CH-4

Static Multiple Issue Processor

Very Long Instruction Word (VLIW)

If one instruction of the pair cannot be used, we require that it be replaced with a noop. Thus, the instructions always issue in pairs, possibly with a noop in one slot.

In some designs, the compiler takes full responsibility for removing *all* hazards, scheduling the code and inserting no-ops so that the code executes without any need for hazard detection or hardware-generated stalls.

Static Multiple Issue Processor

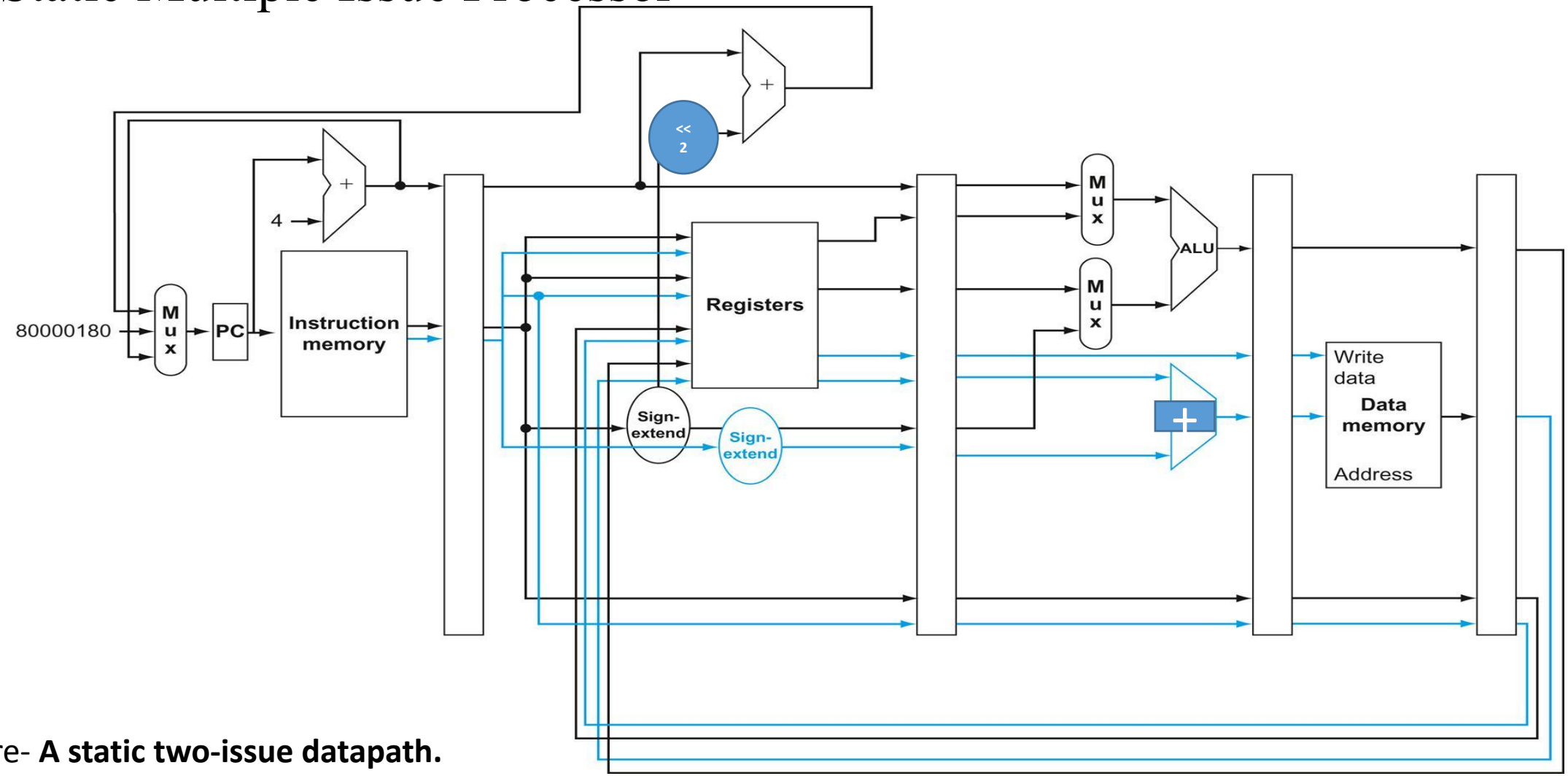
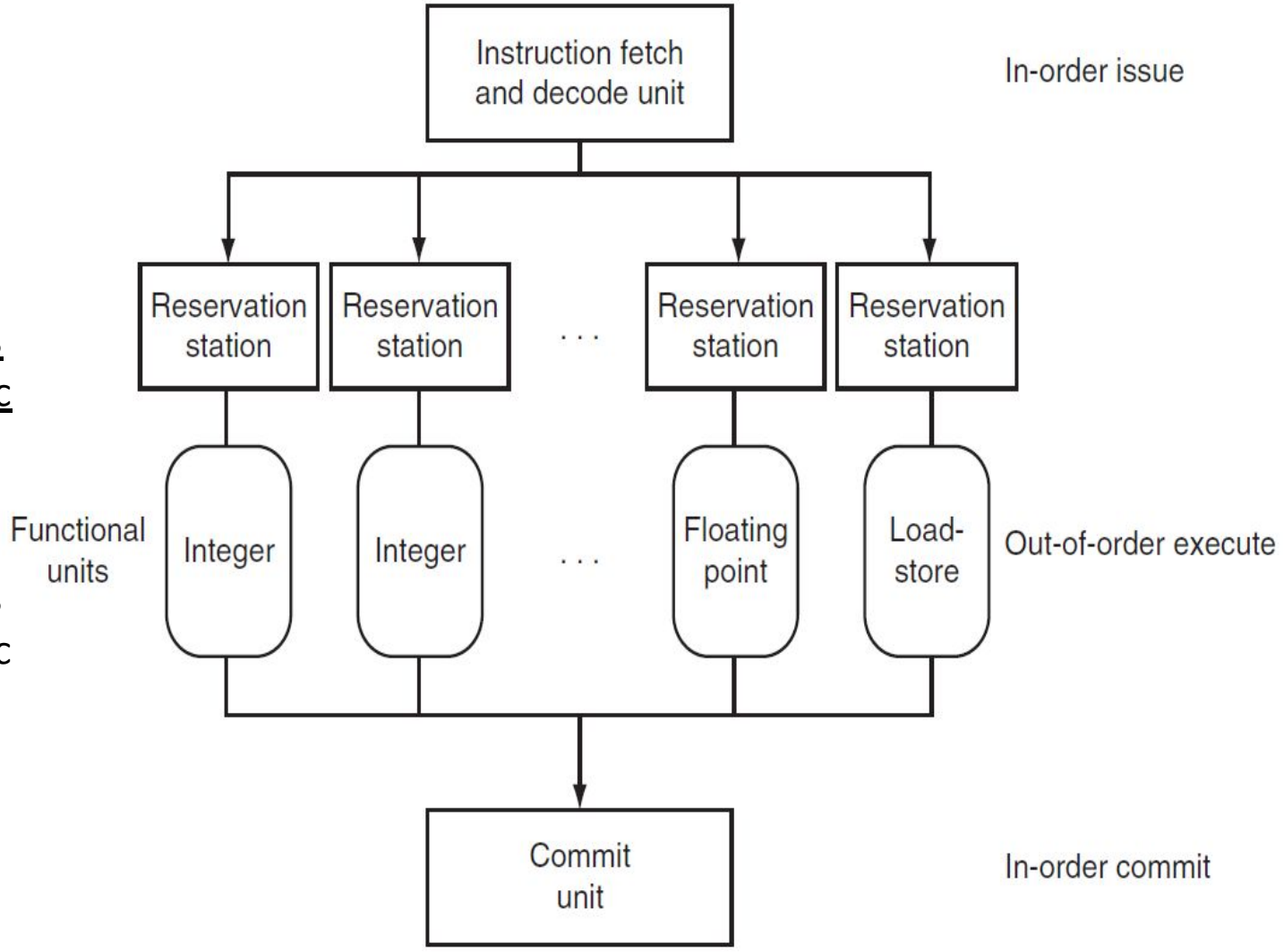


Figure- **A static two-issue datapath.**
The additions needed for double issue are highlighted: another 32 bits from instruction memory, two more read ports and one more write port on the register file, and another ALU. Assume the bottom ADDER handles address calculations for data transfers and the top ALU handles everything else.

Dynamic Multiple Issue or Superscalar

1. Soreboard Technique
https://www.cs.umd.edu/~meesh/411/website/projects/dynamic/scoreboard.html

2. Tomasulo Technique:
https://www.cs.umd.edu/~meesh/411/website/projects/dynamic/tomasulo.html



Book-COD-P&H, CH-4

Difference between Superscalar and VLIW

- General Vs special instruction
- Dynamic Vs. Static pipeline scheduling
- Hazard detection
- Instruction format
- Different VLIW processor needs compilation of the application

ISA design steps

- Step-1:
  - Find out the instructions for the **Algorithm(s)**
- Step-2: [Microarchitecture design]
  - Find out the strategy (Sharedbus/Singlecycle/Multicycle/Pipeline[in order]/etc) for datapath and next
  - Design the datapath and its components for each instructions
- Step-3:
  - Design the combined datapaths for all instructions
- Step-4:
  - Decide the clock period based on the critical path [timing analysis]
  - Add setup time, clock-to-Q and etc. to the decided clock period [satisfy hold time constraints]
- Step-5:
  - Identify the control signals on the combined datapath
- Step-6:
  - Design the Control Unit (H/W or S/W) for generating the such control signals based on the strategy (Sharedbus/Singlecycle/Multicycle/Pipeline[in order]/etc) decided for datapath
- Step-7:
  - Test & verification of the designed processor

How about Single-purpose microprocessor like MinMax microprocessor?

Applying Pipeline Technique in Other Processors

- MinMax Processor
  - Recording is available in the Google-classroom

- Simple CPU
  - Recording is available in the Google-classroom

Homework

- Design the Pipelined MIPS ISA using Verilog HDL and C++
- Convert
  - MinMax microprocessor in Pipelined MinMax
- Design the Pipelined MinMax microprocessor using Verilog HDL and C++
- How does Intel manages to run CISC-type code onto RISC-based pipeline?

Summary

- Limitation of Multi-cycle approach
- CPI Vs IPC
- Comparison between single-cycle and pipelined approaches
- Views of pipeline in operation
- Comparison of datapaths
- Design tradeoffs of microprocessors
- Datapath and CU for pipelined processor