



Computer Architecture (CS F342)

Design and Analysis of Instructions

Design of Control Unit for Reduced Instruction Set Computer
(RISC)

Problems of Single-cycle Datapath Design

- Single-cycle design works well but inefficient design
- Clock length (worst-case delay) is same for all instructions
- It is not a balanced design
- CPI is 1
- Use more resources:
 - Adder
 - Memory
- Necessity of balanced datapath design technique by focusing on common-case design & analysis principle

Balanced datapath design: Multi-cycle approach

- Instruction execution can be broken down to smaller steps
 - Is it similar to CISC-style approach?
- Simple instruction can complete the execution earlier than the complex instructions
- One can design the multi-cycle datapath as similar in single-cycle
 - Connecting architectural elements with the memory using combinational logic
 - Next, design the controller

Each clock cycle is now dedicated to perform a single task:

IF, ID, Execute (ALU), Data Memory, Write Back

3

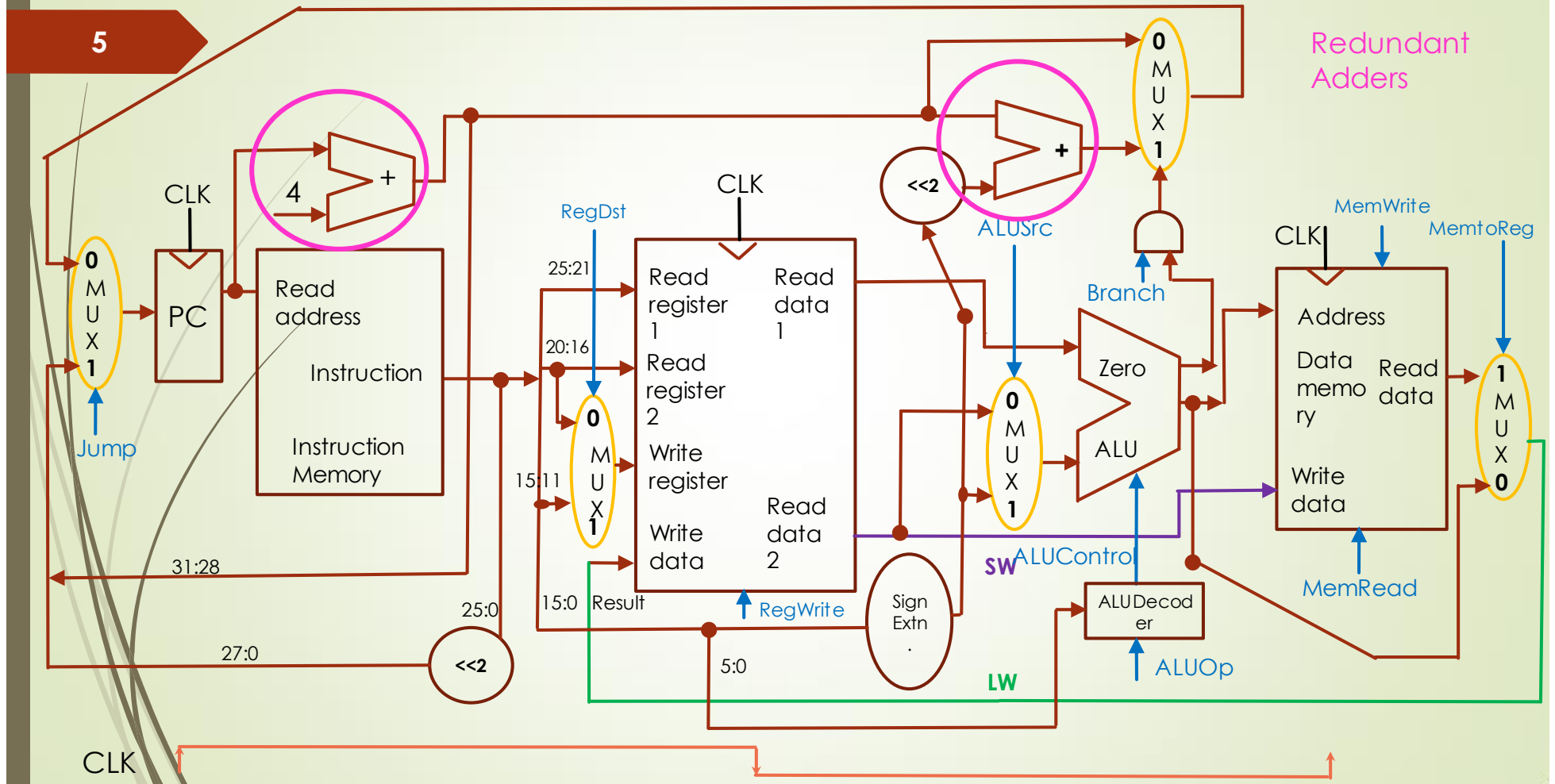
Balanced datapath design: Multi-cycle approach

- The key difference is
 - Controller produces different signals on different steps/states
 - A finite state machine approach as in CISC-style

Balanced datapath design: Multi-cycle approach

- Combined the instruction and data memory
- Remove the redundant adders
- Incorporate the non-architectural state elements

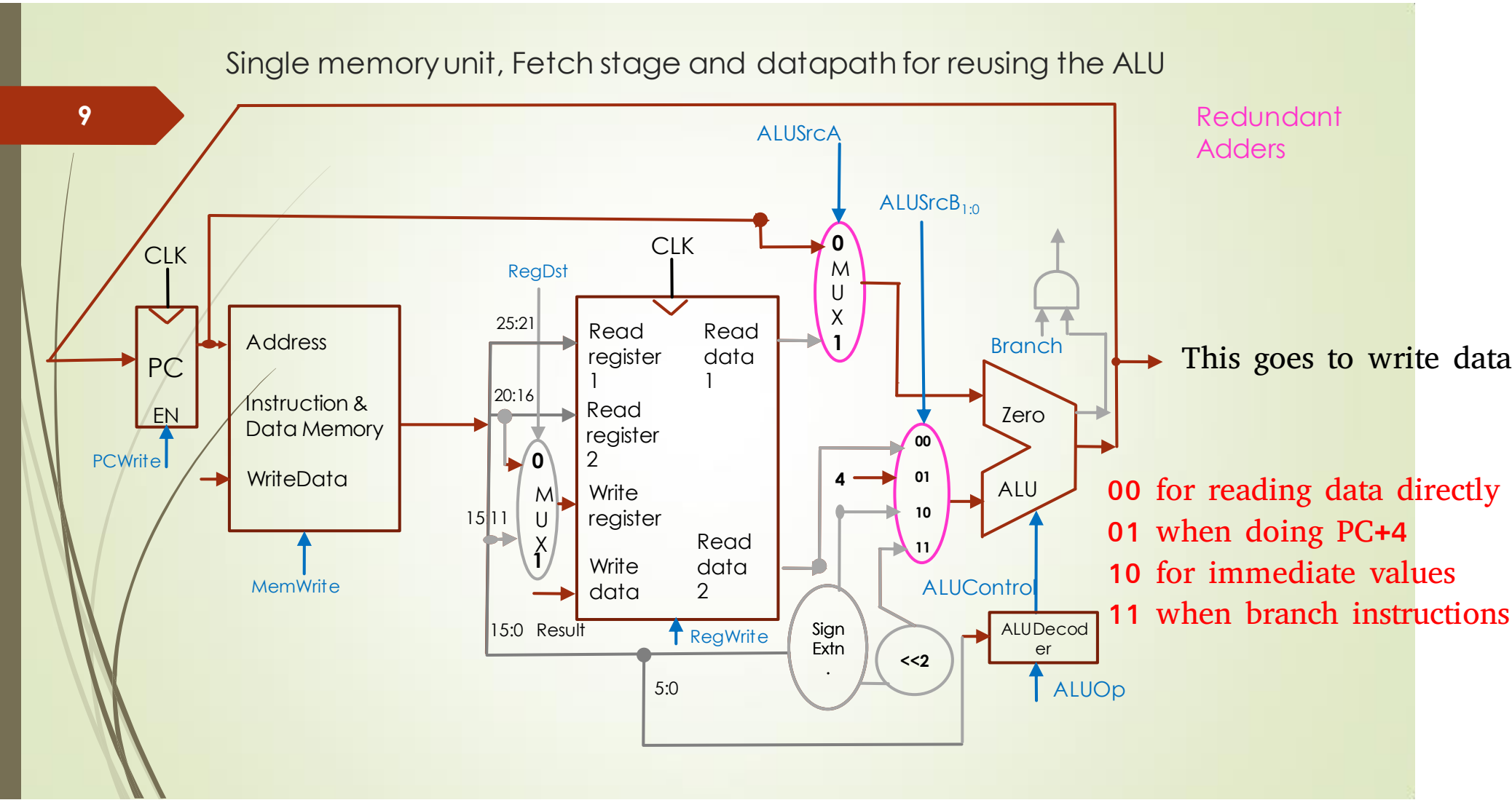
Single-cycle datapath



Balanced datapath design: Multi-cycle approach

- Remove the redundant adders
- Where to place this operations?
- How does one control such operations?

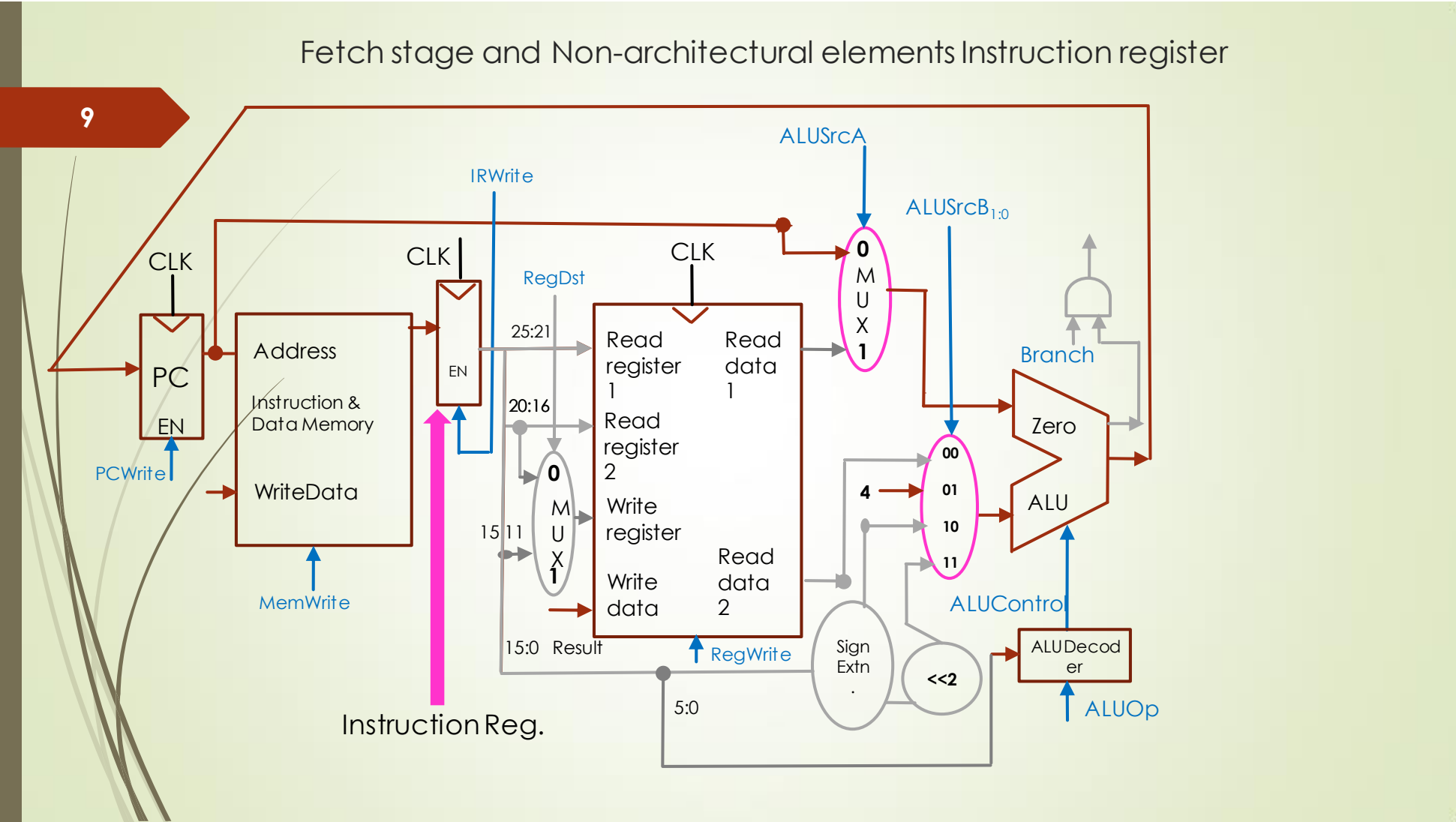
PCWrite used to control if PC is being incremented or not.
Also note that here instruction and data memory are merged into one



Balanced datapath design: Multi-cycle approach

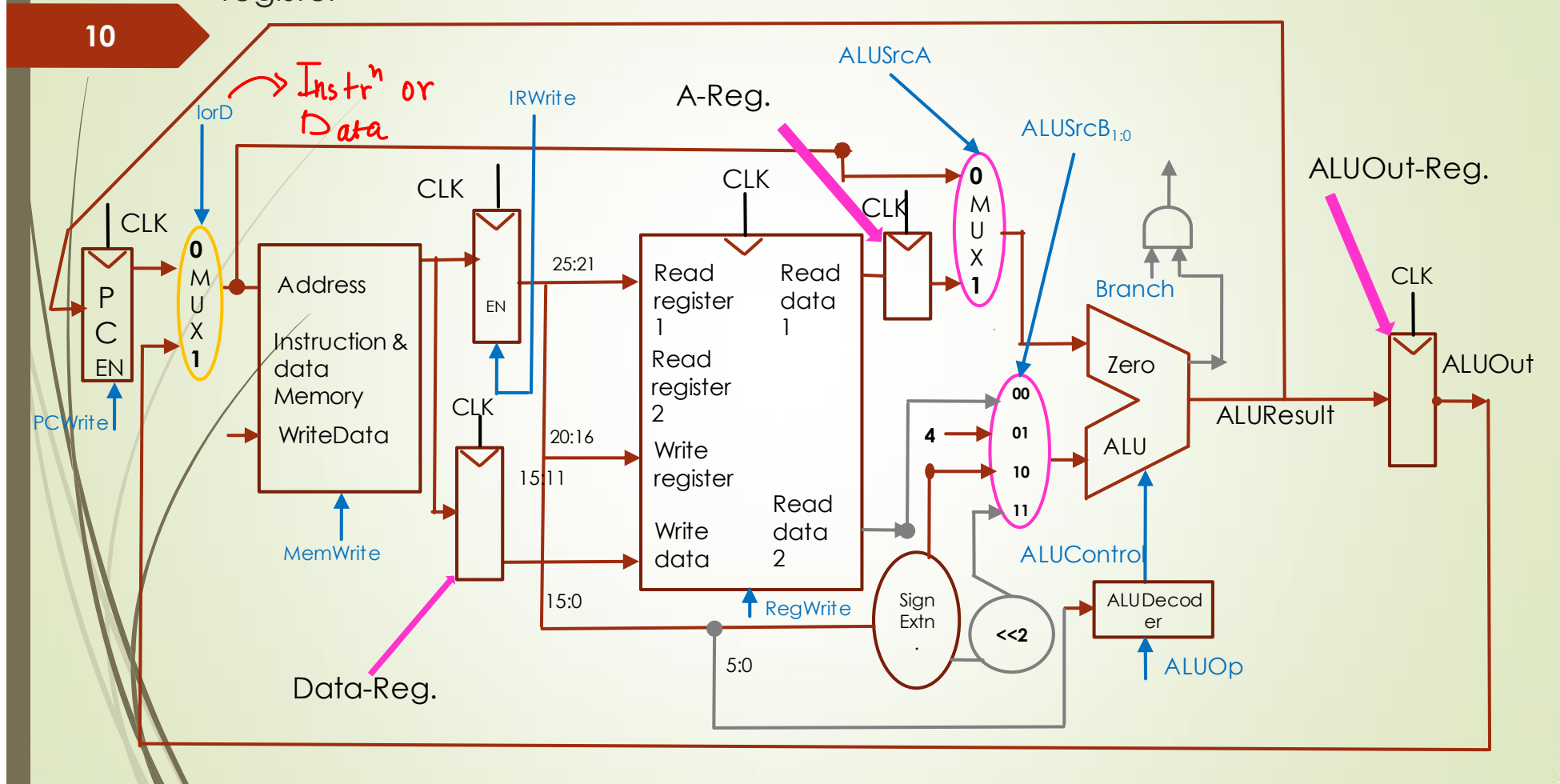
- Combined the instruction and data memory
- Remove the redundant adders
- Incorporate the non-architectural state elements

During the IF cycle: IRWrite is set. So the instruction stored in the address corresponding to PC is stored inside the Instr Reg. After this the IRWrite is unset and PCWrite is set. So the PC becomes PC+4 ready to get to next instruction when the full execution of this one is finished.

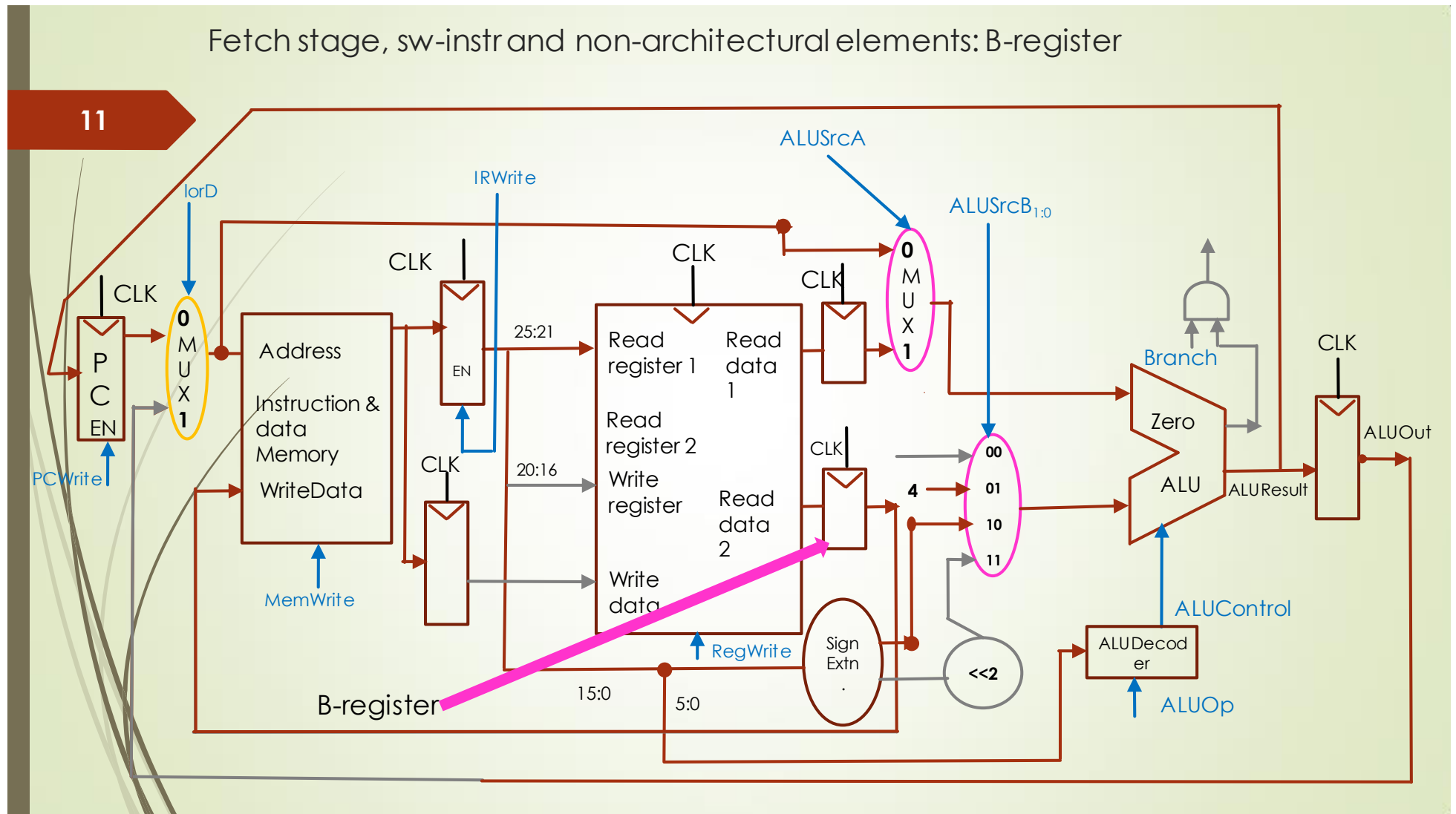


A data register is also needed so as to distinguish clearly when the mem element is acting like data or instr memory. We basically require a register to save information from the previous clock cycle. Each of them has an enable signal also. Here we have used A-Reg to store the data 1 which we can use in the exec cycle later. If this isn't done, then there is a possibility that some other things change in the previous wires so as to give wrong values for read data and others also.

Fetch stage, lw-instr. and non-architectural elements: A, Data and ALUOut-register

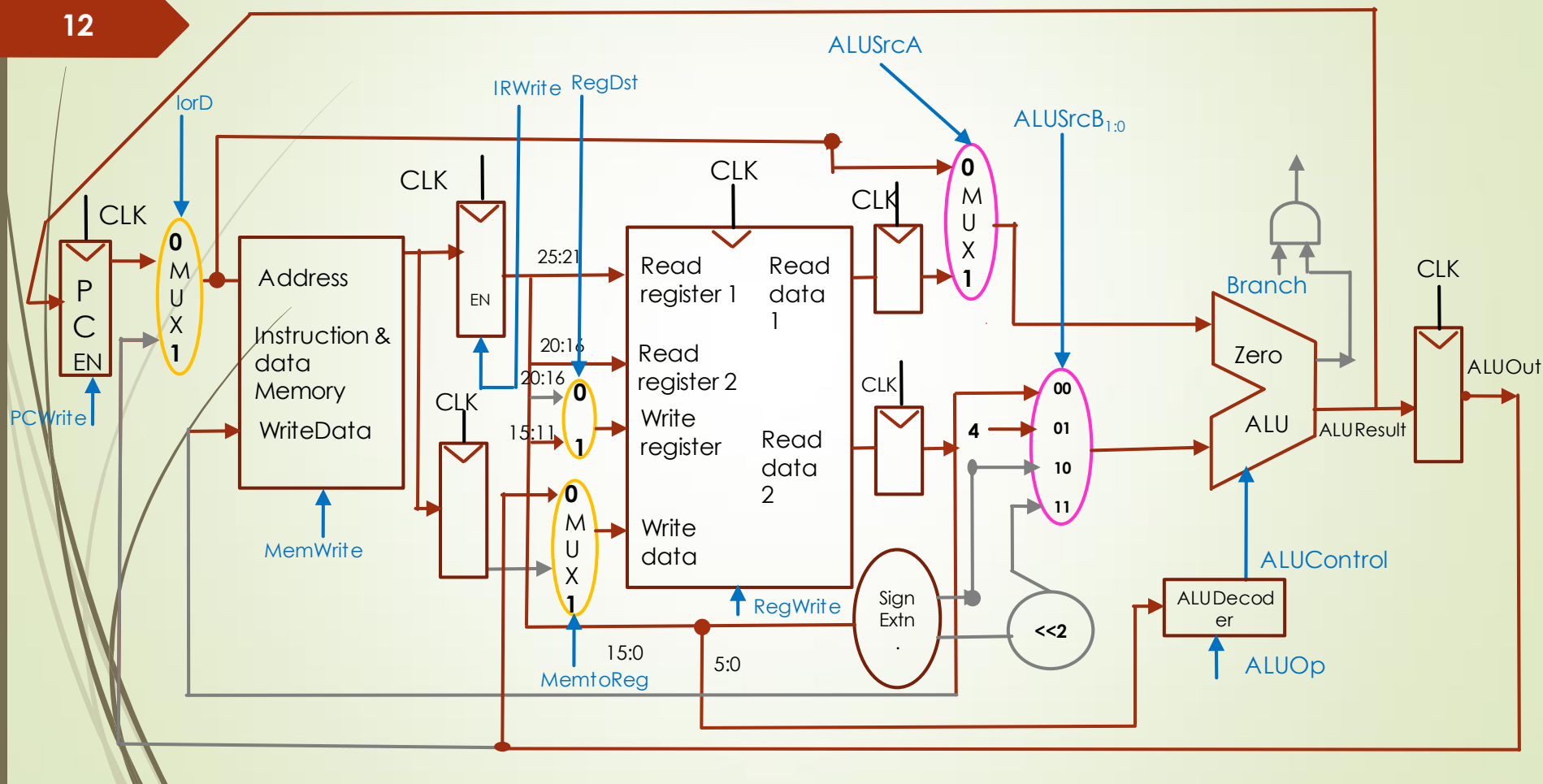


Similar to A-reg, B-reg also needed



Fetch stage and R-type instruction

12

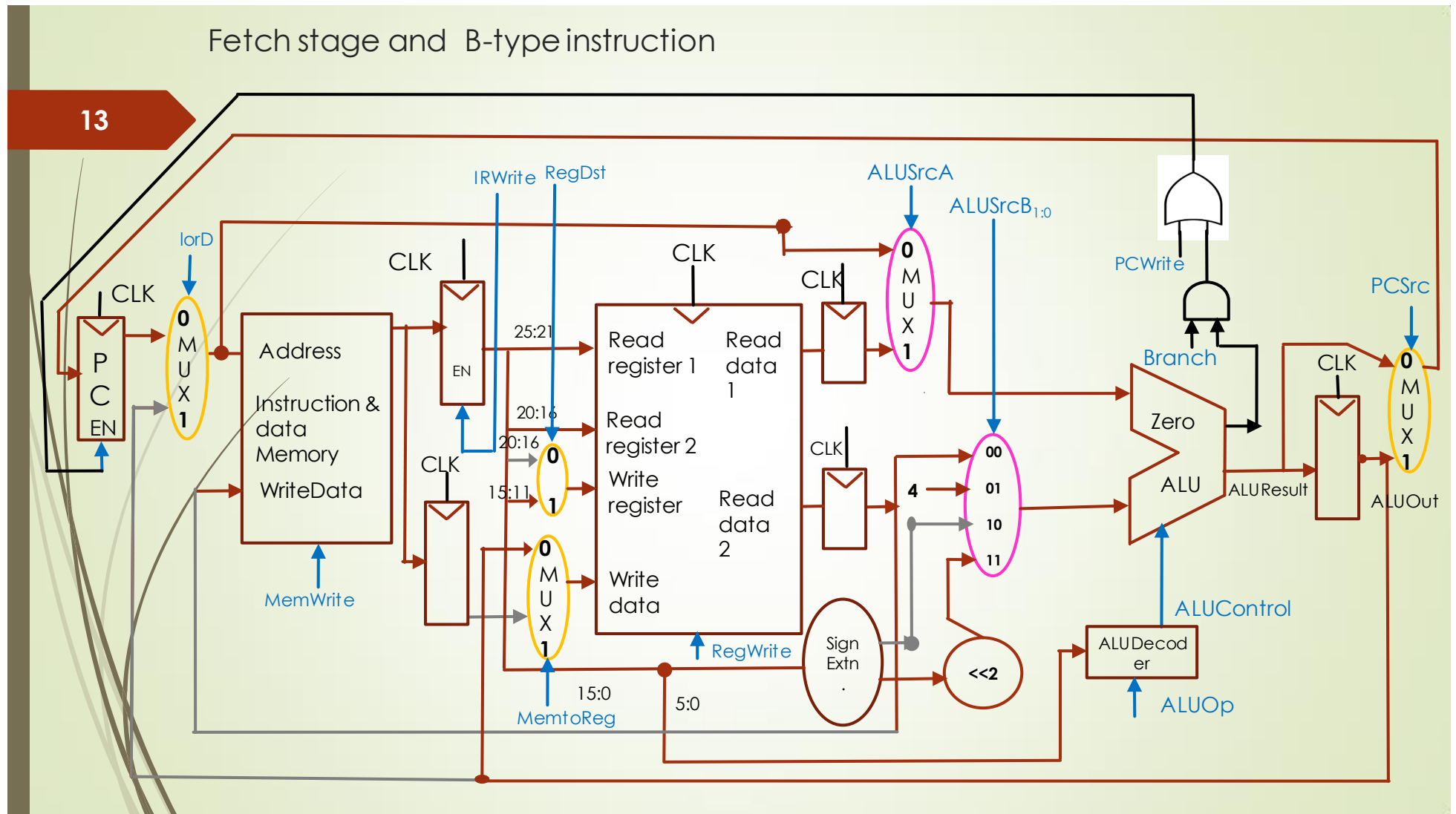


When branch instruction is called, the PC should be enabled. So, either PCWrite will decide or Branch \wedge Zero.

Steps: 1. IF

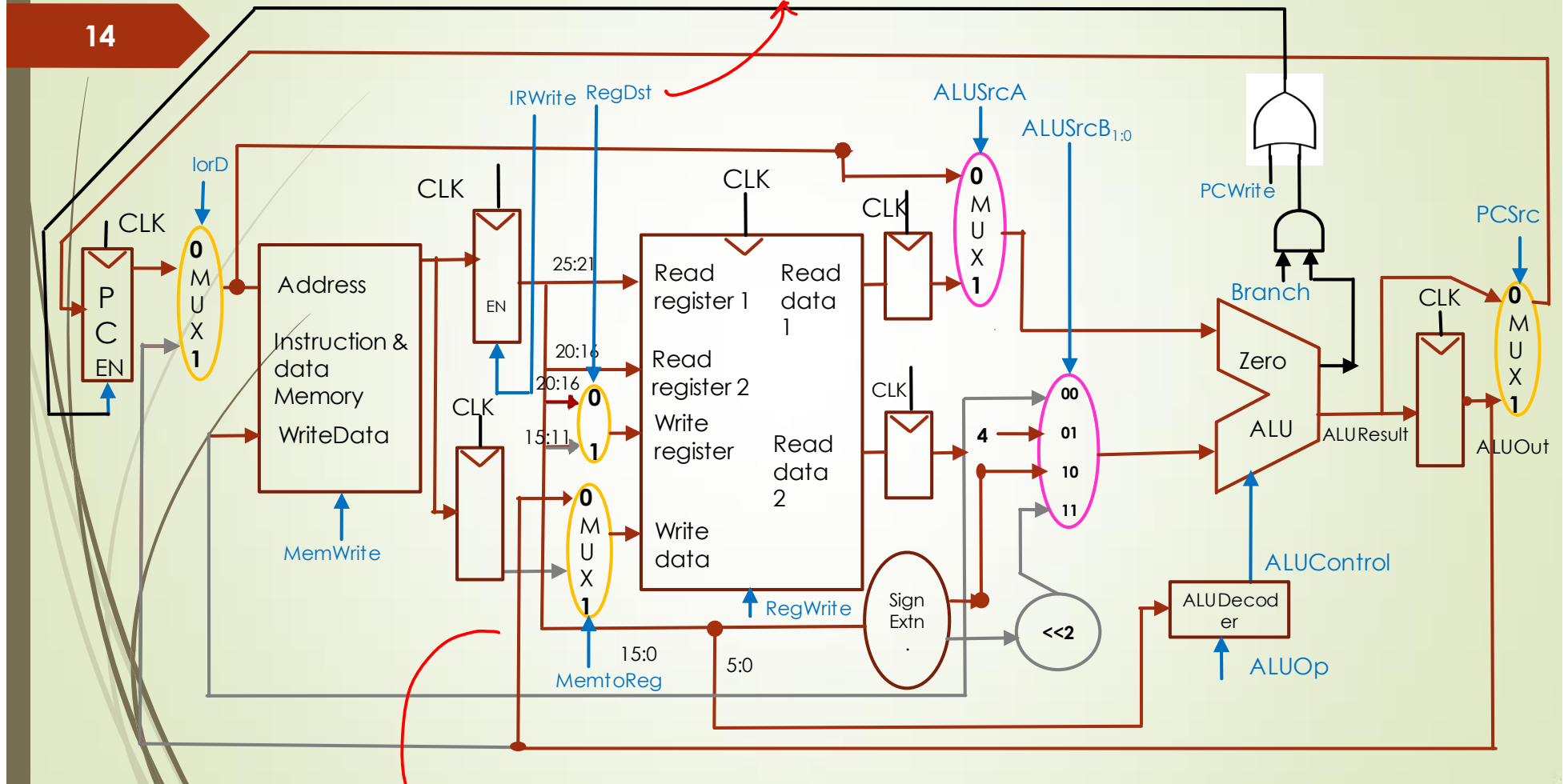
2. ID: In this stage itself $ALUResult = PC + 4 + offset$ is calculated and stored in ALUOut

3. the A-reg and B-reg will be compared and accordingly PC will be enabled or disabled. $PCSrc = 1$ for branch so, the next address is taken from the ALUOut



Fetch stage and I-type instruction

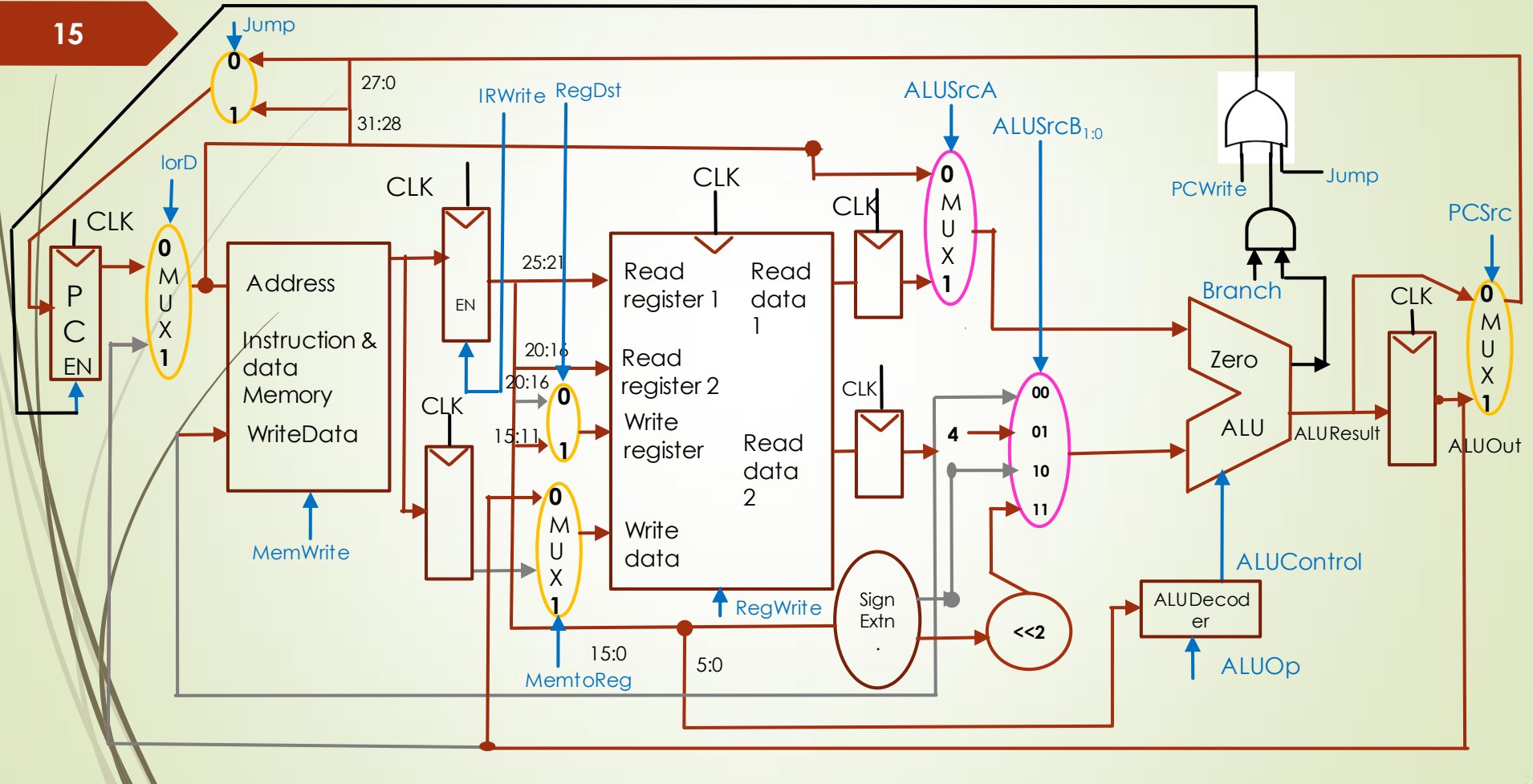
14



This wire will carry the final data and write.

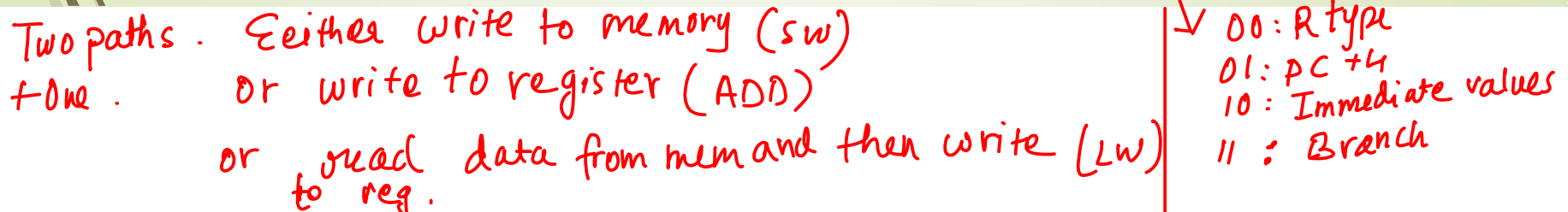
Fetch stage and Jump instruction

15



$\rightarrow PC + 4$
 $\{IF, Jump, Branch\}$
 Enabler for PC

A 32 bit wire



Control signals

- IorD
- Jump
- Memwrite
- IRWrite
- RegDst
- MemtoReg
- RegWrite
- ALUSrcA
- ALUSrcR1:0
- PCWrite
- Branch
- PCSrc
- ALUOp
- ALUControl

Fetch stage

Machine state	Operation	Control signals
T0	$\text{InsR} \leftarrow \text{M}[\text{PC}];$ $\text{PC} \leftarrow \text{PC} + 4$	$\text{IorD}=0, \text{IRWrite}=1,$ $\text{ALUSrcA}=0, \text{ALUSrc}=01, \text{ALUOp}=00, \text{PCSrc}=0, \text{PCWrite}=1$

Decode stage

Machine state	Operation	Control signals
T1	(PC+4) + SigExt _n (offset)	ALUSrcA=0, ALUSrcB _{1:0} = 11, ALUOp=00

Useful for BRZ instruction

LW type instruction

Machine state	Operation	Control signals
T2	$A + \text{sigEx}(\text{offset})$	$\text{ALUSrcA}=1, \text{ALUSrcB}_{1:0} = 10, \text{ALUOp}=00$
T3	$\text{Data} \leftarrow M[A + \text{sigEx}(\text{off})]$	$\text{lorD}=1$
T4	$\text{RF}[\text{dest}] \leftarrow \text{Data}$	$\text{RegDst}=0, \text{MemtoReg}=1, \text{RegWrite}=1, \text{T0}$

SW type instruction

Machine state	Operation	Control signals
T2	$A + \text{sigEx}(\text{offset})$	$\text{ALUSrcA}=1, \text{ALUSrcB}_{1:0} = 10, \text{ALUOp}=00$
T3	$M[A + \text{sigEx}(\text{offset})] \leftarrow B$	$\text{lorD}=1, \text{MemWrite}=1, \text{TO}$

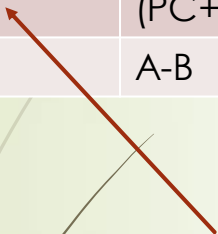
R-type instruction

Machine state	Operation	Control signals
T2	A Op B	ALUSrcA=1, ALUSrcB _{1:0} = 00, ALUOp=00
T3	RF[dstn] \leftarrow A Op B	RegDst=1, MemtoReg=0, RegWrite=1, T0

B-type instruction

Machine state	Operation	Control signals
T1	(PC+4) + SigExtn(offset)	ALUSrcA=0, ALUSrcB _{1:0} = 11, ALUOp=00
T2	A-B	ALUSrcA=1, ALUSrcB _{1:0} = 00, ALUOp=01, Branch=1, T0

Decoding stage



ADDI instruction

24

Machine state	Operation	Control signals
T2	A OP SigExtn(offset)	ALUSrcA=1, ALUSrcB _{1:0} = 10, ALUOp=00
T3	RF[Destn] ← A OP SigExtn(offset)	RegDst=0, MemtoReg=0, T0

Jump instruction

25

Machine state	Operation	Control signals
T2	A OP SigExtn(offset)	Jump=1, T0

Clock-cycle needed for the instructions

Instructions	Clock-cycle
LW	5
SW	4
R-type	4
BEQ	3
ADDI	4
J	3

All
 IF, ID, Execute, Mem.
 , write Back
 " " " , write Back.
 " " " , write Back.
 " " "

What is next?

- All the ALU operations (ALUOp) similar as in Single-cycle approach
- Generate the control signals using:
 - Hardwired-based approach
 - Microprogrammed-based approach
- Inputs of the control unit similar as in Single-cycle approach

Performance Analysis

- The program consists of approximately 25% loads, 10% stores, 11% branches, 2% jumps, and 52% R-type instructions. Determine the average CPI for this program.

Performance Analysis

- The program consists of approximately 25% loads, 10% stores, 11% branches, 2% jumps, and 52% R-type instructions. Determine the average CPI for this program.
- The average CPI is the sum over each instruction of the CPI for that instruction multiplied by the fraction of the time that instruction is used
- $\text{Average CPI} = (0.11 + 0.02)(3) + (0.52 + 0.10)(4) + (0.25)(5) = 4.12$
- For Single-cycle approach, Avg. CPI is 1

Performance Analysis

- Each cycle involved one ALU operation, memory access, or register file access
- Assumptions:
 - the register file is faster than the memory and
 - writing memory is faster than reading memory
- Datapath has two possible critical paths:
- $T_c = t_{pcq_PC} + t_{mux} + \max\{t_{ALU} + t_{mux}, t_{mem}\} + t_{registerRead}$

Performance Analysis

- 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

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

Performance Analysis

- 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
- $T_c = t_{pcq_PC} + t_{mux} + \max\{t_{ALU} + t_{mux}, t_{mem}\} + t_{registerRead}$
- $T_c = 30 + 25 + 250 + 20 = 350$ ps
- Execution time =
 $(100 * 10^9 \text{ instrs.}) * (4.12 \text{ cycle/instrs.}) * (350 * 10^{-12} \text{ s/cycle})$
 $= 133.9 \text{ seconds}$

Parameter	Delay (ps)
t_{pcq_PC}	30
regSet up	20
t_{mem}	250
t_{RFread}	20
t_{ALU}	200
t_{mux}	25
$t_{RFwrite}$	20

Performance Analysis: A Comparison

- For multi-cycle, $T_c = 350$ ps and $CPI = 4.12$
- For single-cycle, $T_c = 925$ ps and $CPI = 1$
- For multi-cycle, execution time = 133.9 seconds
- For single-cycle, execution time = 92.5 seconds
- This example shows multi-cycle processor is slow than the single-cycle processor; why is it so?
- Multi-cycle processor is less expensive
- It has 5-nonarchitectural elements

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

Summary

- Disadvantages of Single-cycle processor
- Necessity of balanced design approach and multi-cycle processor
- Datapath design of multi-cycle processor
- Design of Controls for multi-cycle processor
- Performance analysis of multi-cycle processor
- Comparison between single-cycle and multi-cycle processor
- Disadvantages of multi-cycle processor