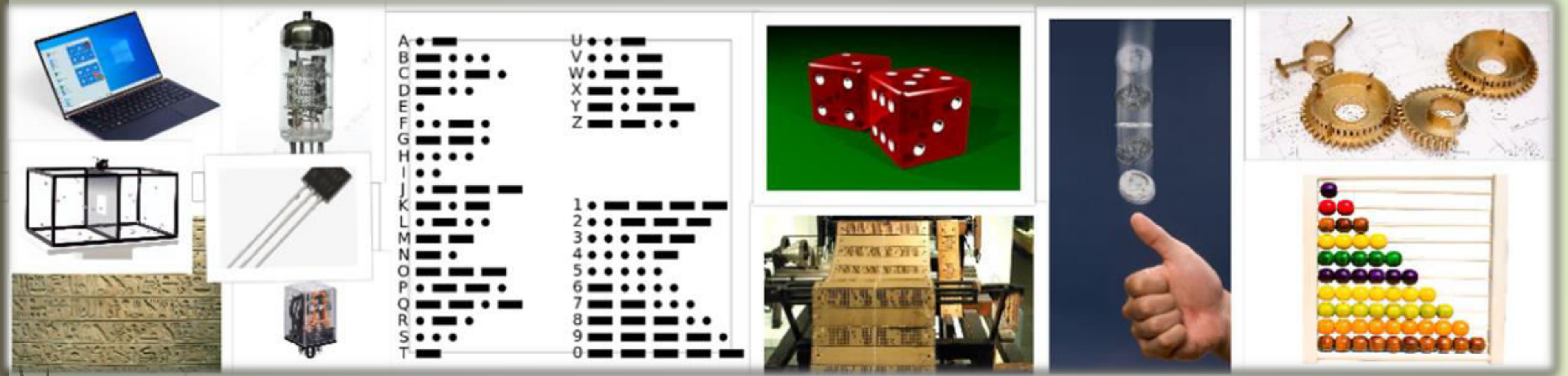


Computer Architecture (CS F342)

Design and Analysis of Instructions

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

Performance improvement of CPU

- To improve the performance of any processor by analyzing
  - Not only the Control Unit
  - But also the data-path
  - How does one analyze that?

How does one analyse the data-path?

- Consider the different stages of instruction cycle
- Identify the components

How does one analyse the data-path?

- Consider a few of the MIPS [*] instructions and design the data path
- Data instruction of type register or R-type instruction
- Memory type or M-type instruction or immediate type or I-type instruction
- Branch type or B-type

[*] Microprocessor without Interlocked Pipeline Stages (MIPS) and RISC-I to RISC-V are the example of RISC-style processor

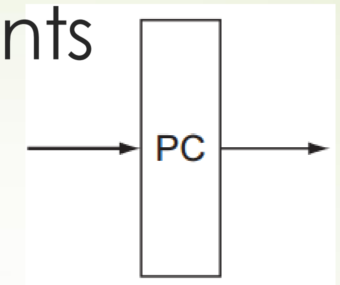
Instruction format for MIPS-based processor

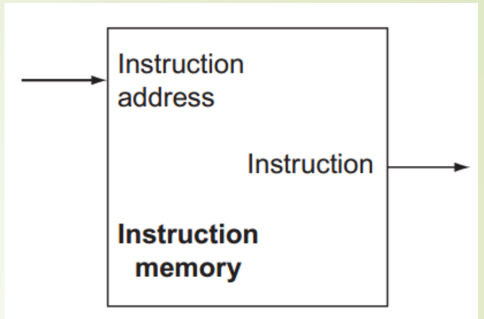
op	rs	rt	rd	shamt	funct
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)

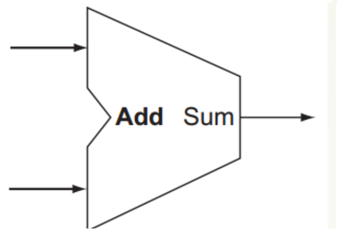
- *op:* Basic operation of the instruction, called the opcode
- *rs:* The first register source operand
- *rt:* The second register source operand
- *rd:* The register destination operand. It gets the result of the operation.
- *shamt:* Shift amount. (to be used for shift instructions. Otherwise, the field contains zero in this section.)
- *funct:* Function. This field, often called the *function code,* selects the specific variant of the operation in the op field

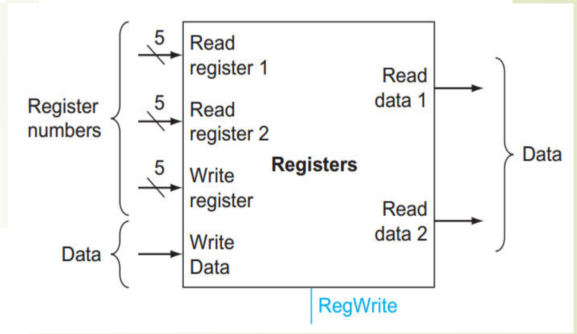
Data-path elements

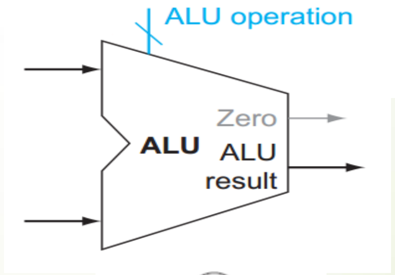
- Program counter (PC)
- Instruction memory
- Adder
- Register file
- ALU
- Signed-extension unit
- Data memory

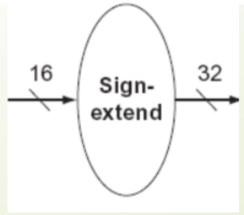


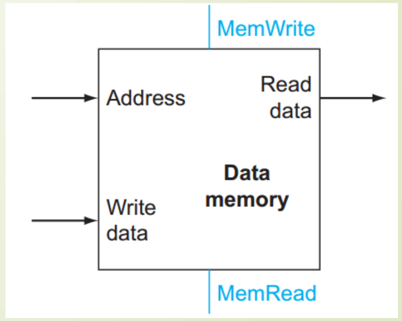




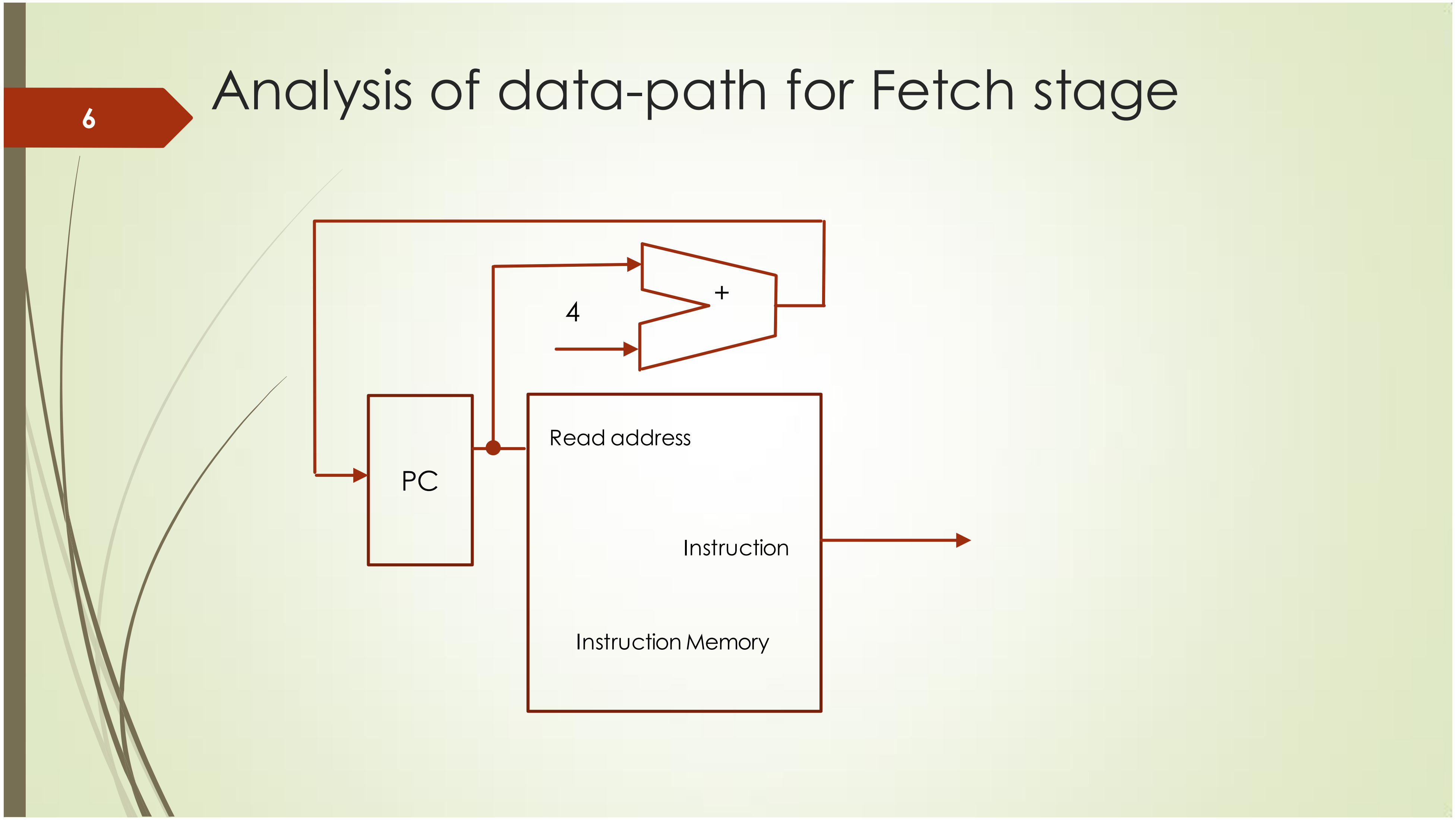






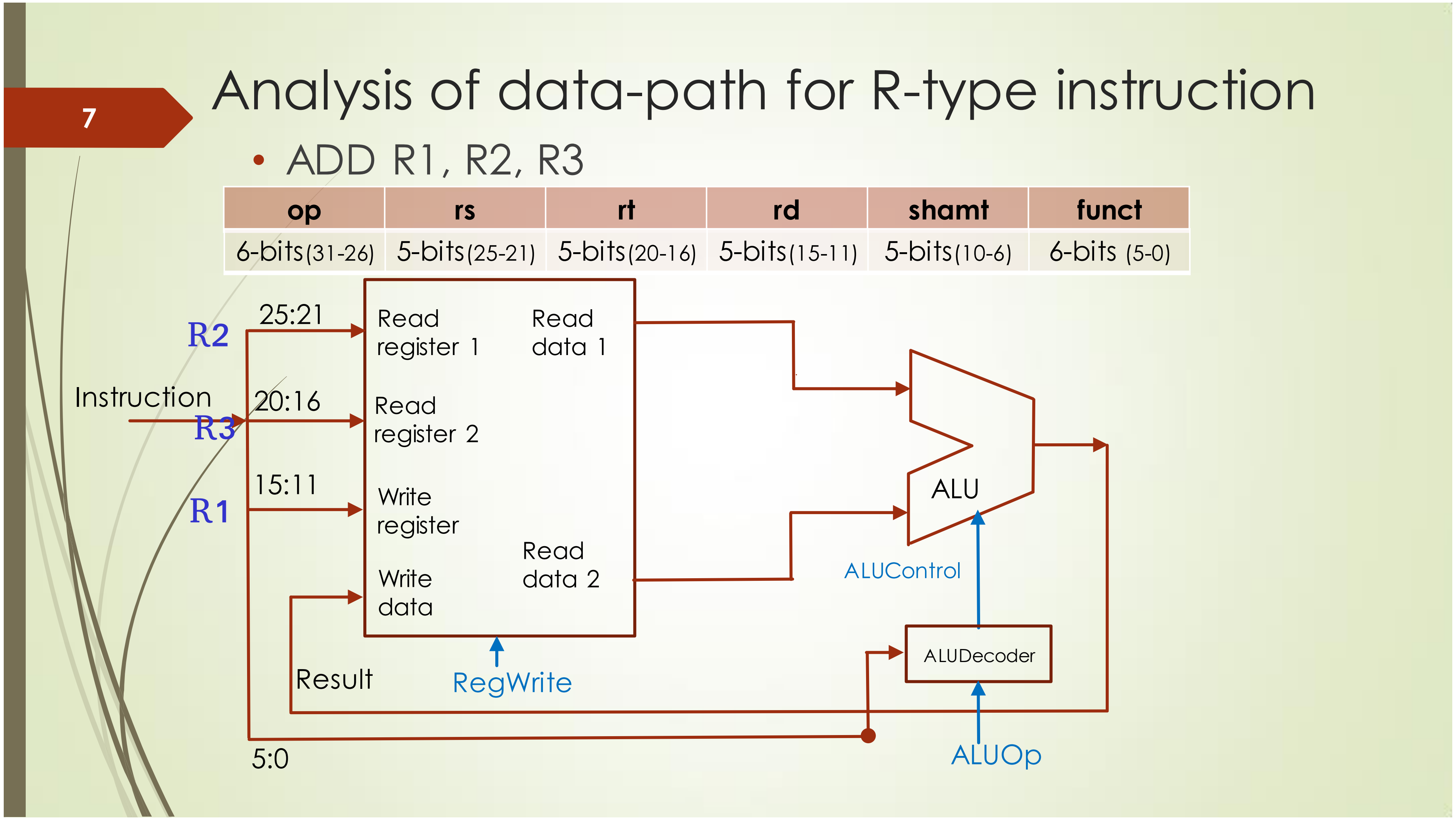


Program Counter stores the address to the instruction location.
Instruction memory reads this address and stores the instruction.
After every instruction, **+4** is added to PC to get to the next instruction.



Analysis of data-path for Fetch stage

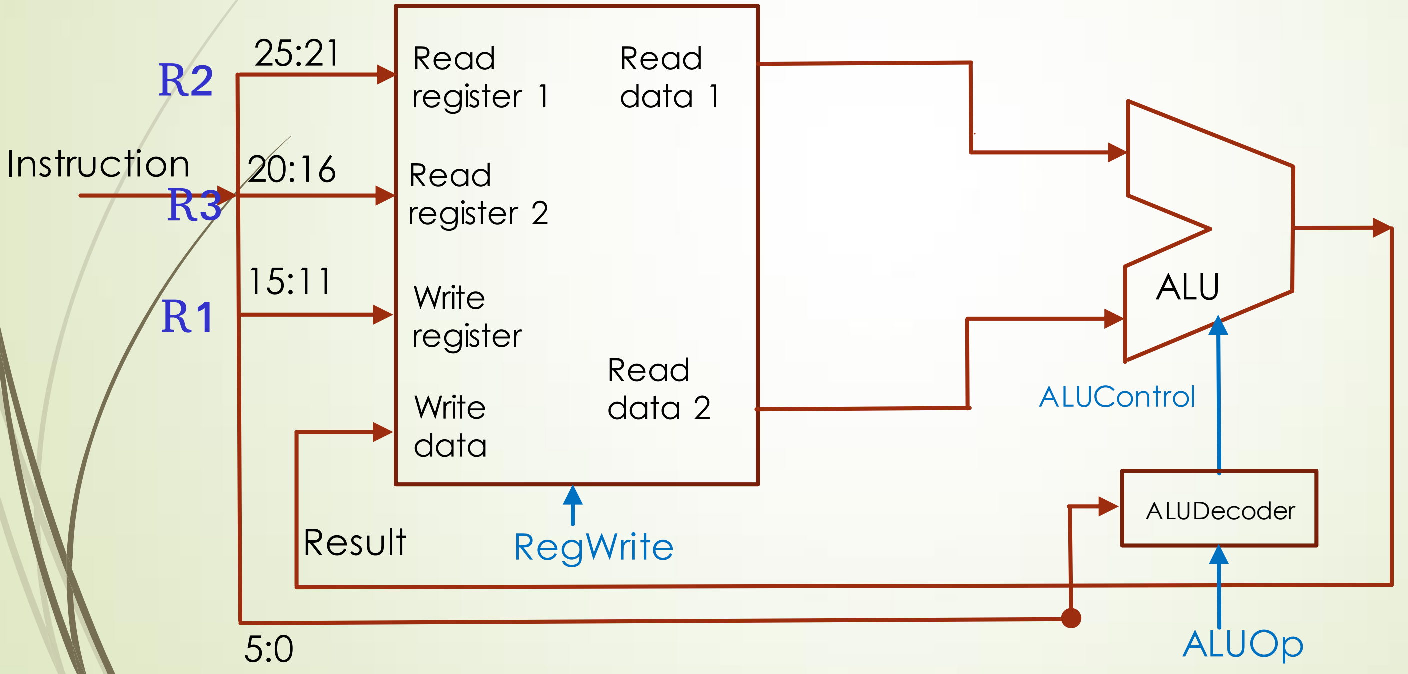
rs, rt, rd will store the register number (assigned to each register)



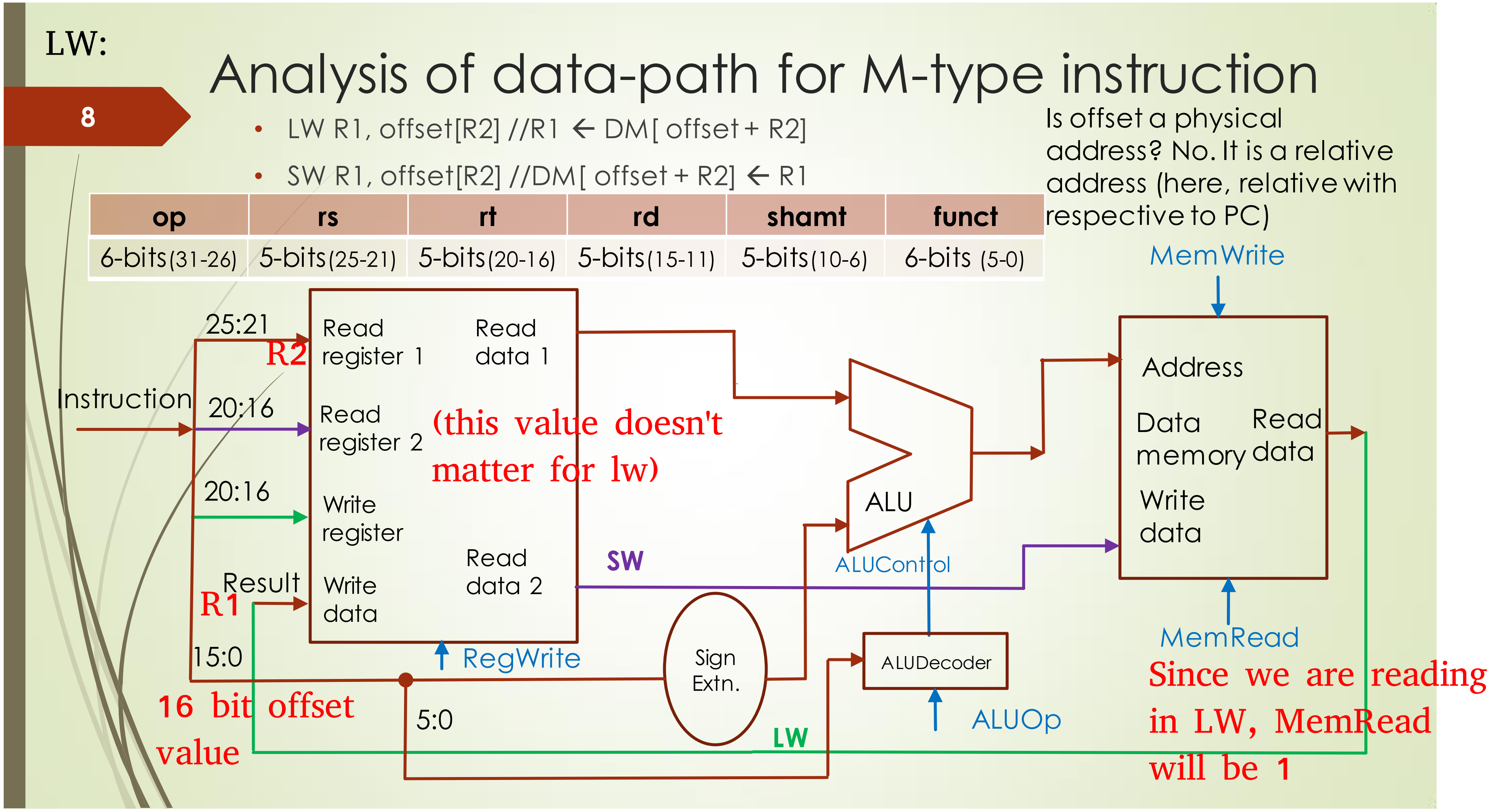
Analysis of data-path for R-type instruction

- 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)



Load Word copies data from memory locn to register. So, it only has to write to the register R1 and reads only the value in R2 which has the base address stored in it.



LW:

8

Analysis of data-path for M-type instruction

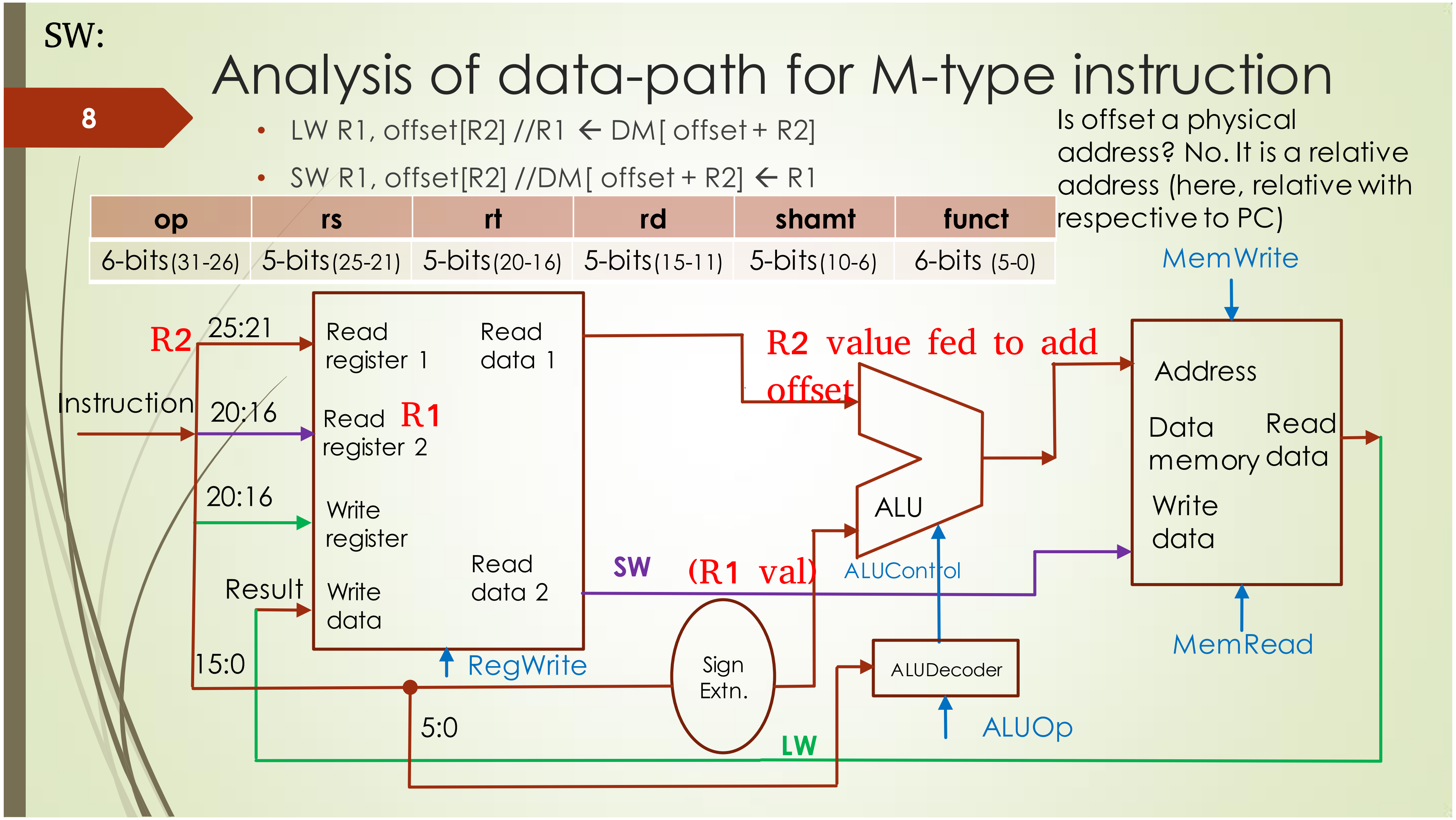
- LW R1, offset[R2] //R1 ← DM[offset + R2]
- SW R1, offset[R2] //DM[offset + R2] ← R1

Is offset a physical address? No. It is a relative address (here, relative with respective to PC)

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)

16 bit offset values has to be converted to 32 bit so sign extended.
Also, the last 6 bits which denote the funct doesn't matter here because ALUOp will only generate for Addition of offset to value in R2

SW will copy the register value in R1 to a memory location at addresss -> [offset + R2]



SW:

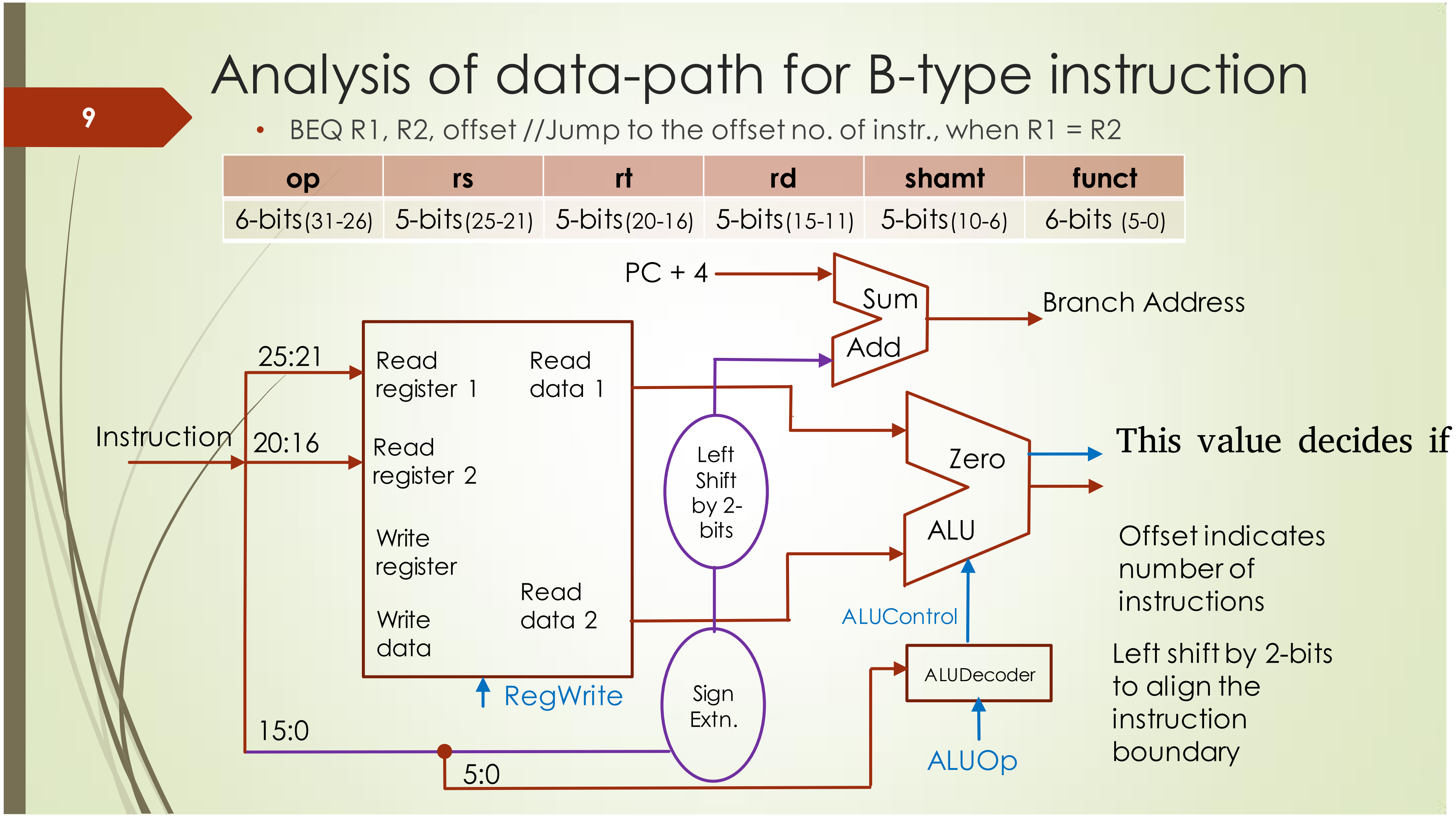
Analysis of data-path for M-type instruction

- LW R1, offset[R2] //R1 ← DM[offset + R2]
- SW R1, offset[R2] //DM[offset + R2] ← R1

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)

Is offset a physical address? No. It is a relative address (here, relative with respective to PC)

Conditional jump. Reads the value R1 and R2. Offset value is read (basically the label used in code). This address is added to the program counter value to get the new address of the branch.



9

Analysis of data-path for B-type instruction

- BEQ R1, R2, offset //Jump to the offset no. of instr., when R1 = R2

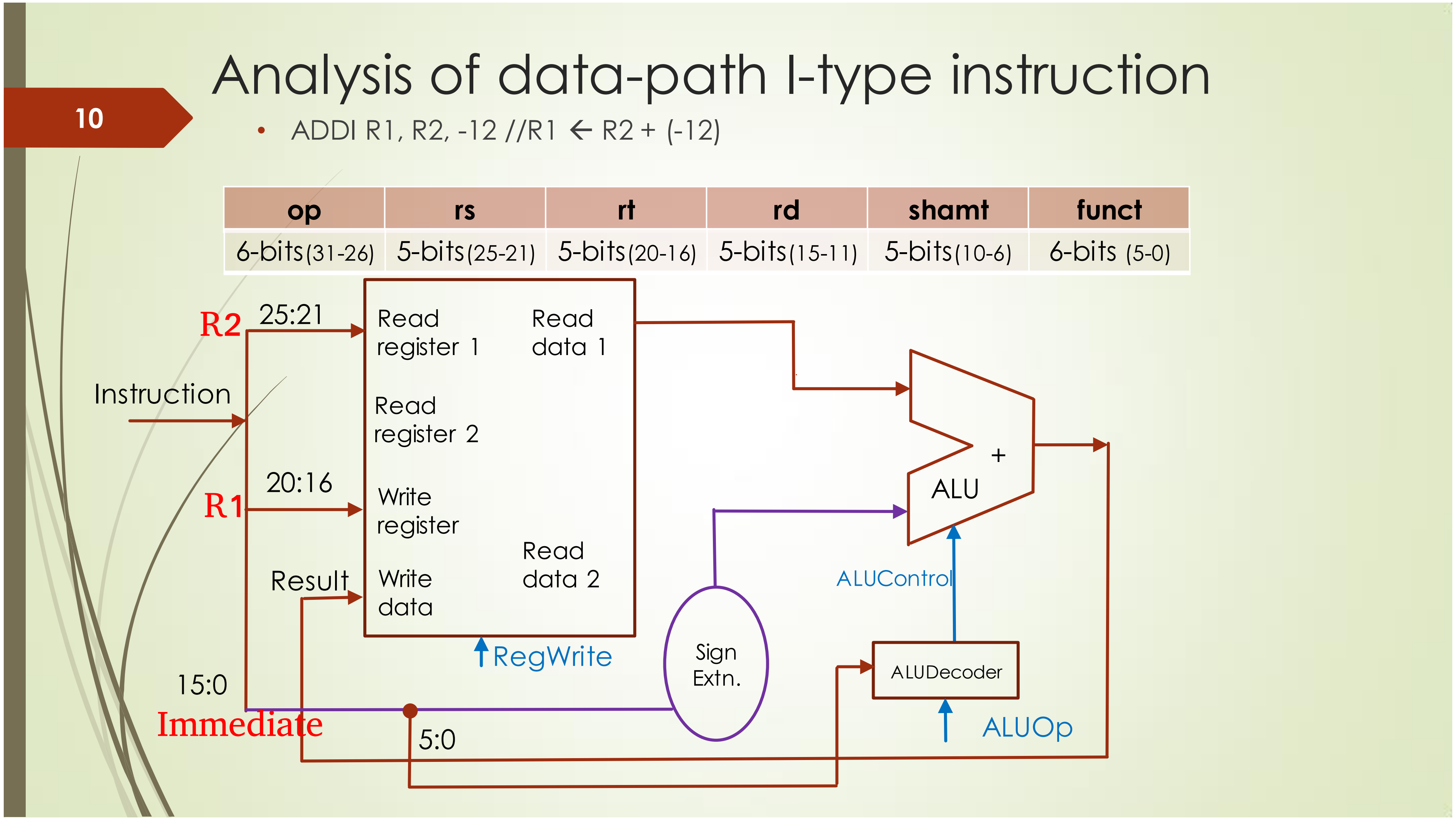
op	rs	rt	rd	shamt	funct
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)

This value decides if equal

Offset indicates number of instructions

Left shift by 2-bits to align the instruction boundary

Note that here, 20:16 gives the register to be written rather than read.



Analysis of data-path I-type instruction

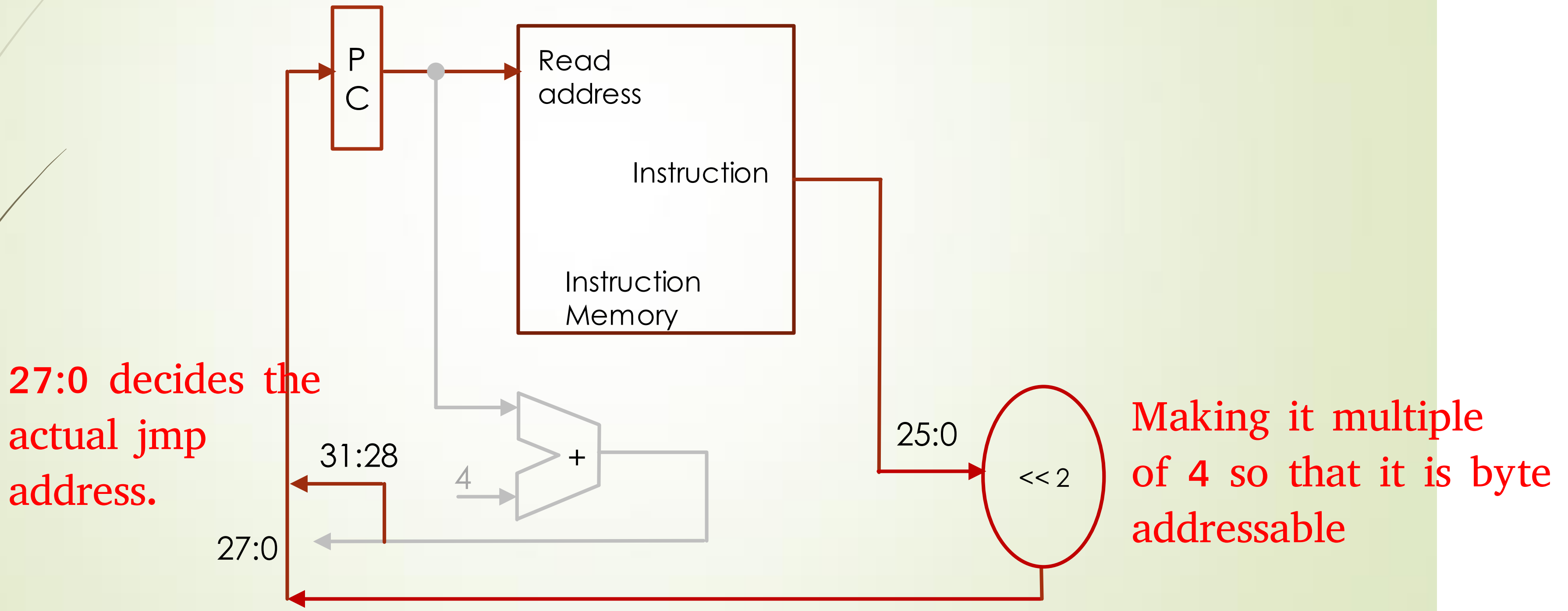
- ADDI R1, R2, -12 //R1 ← R2 + (-12)

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)

Analysis of data-path j-type instruction

- J addrs //PC ← PC[31:28]addrs[27:0]

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)

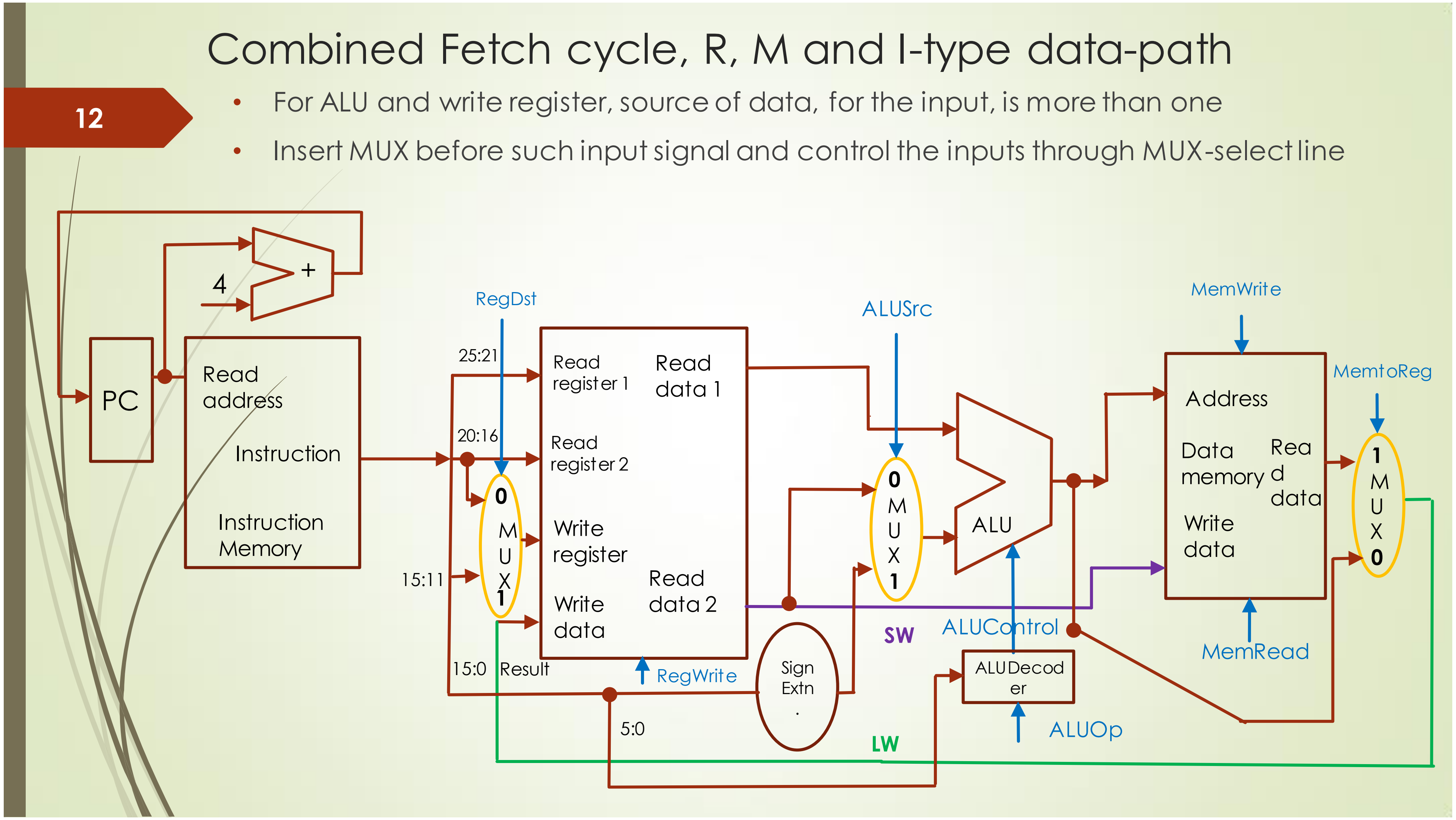


27:0 decides the actual jmp address.

Making it multiple of 4 so that it is byte addressable

The 4 most significant bits are from the previous address value

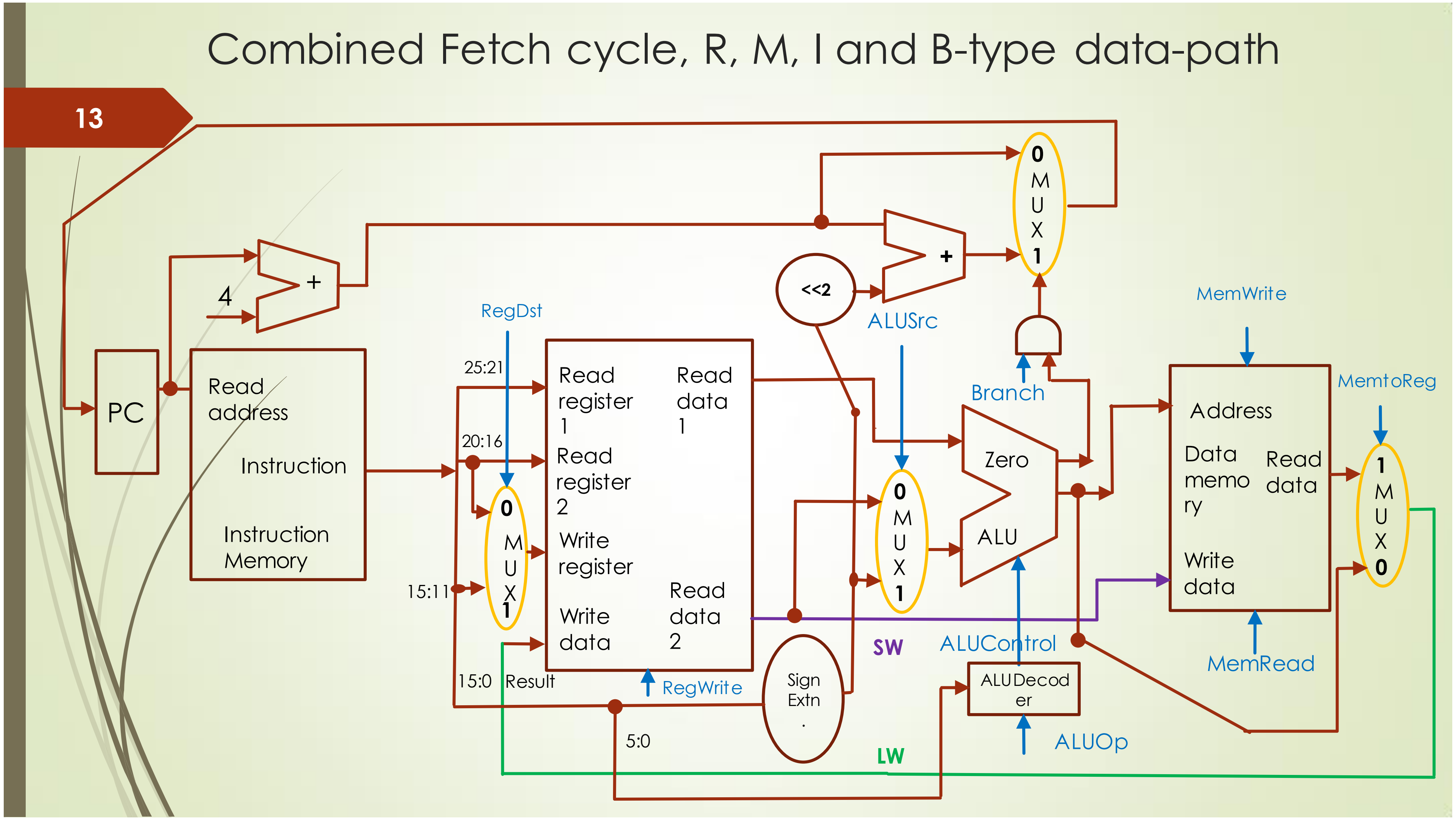
RegDst signal decides whether register 20:16 is being written or 15:11 is being written
ALUSrc decides whether Immediate value should be used or the read data 2 value.
MemToReg is set if register is being written



Combined Fetch cycle, R, M and I-type data-path

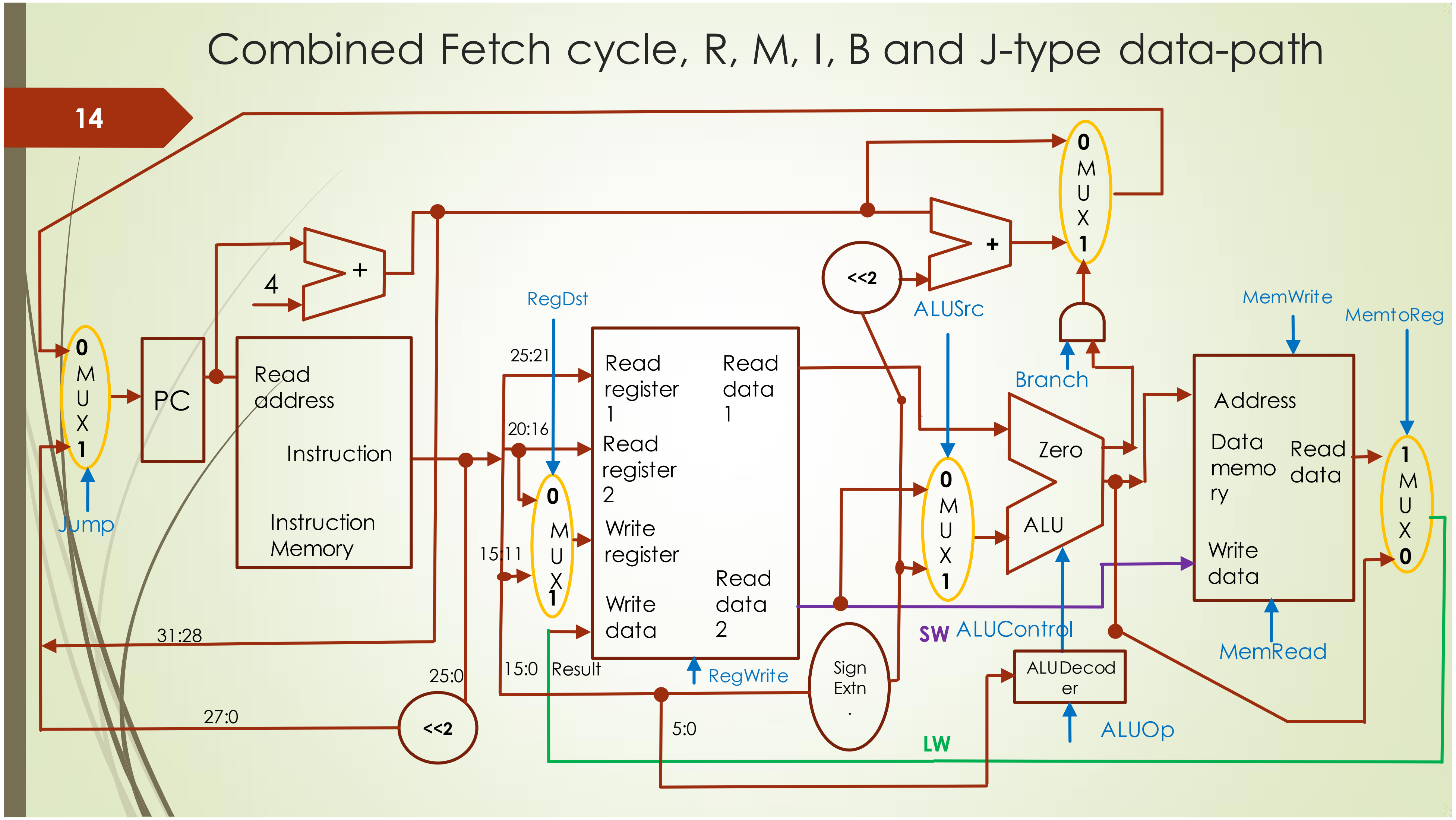
- For ALU and write register, source of data, for the input, is more than one
- Insert MUX before such input signal and control the inputs through MUX-select line

The branch signal decides whether the next address is PC+4 or the value PC+4+offset.
The zero value from the ALU must also be set (that is the comparator should satisfy the branch condition)

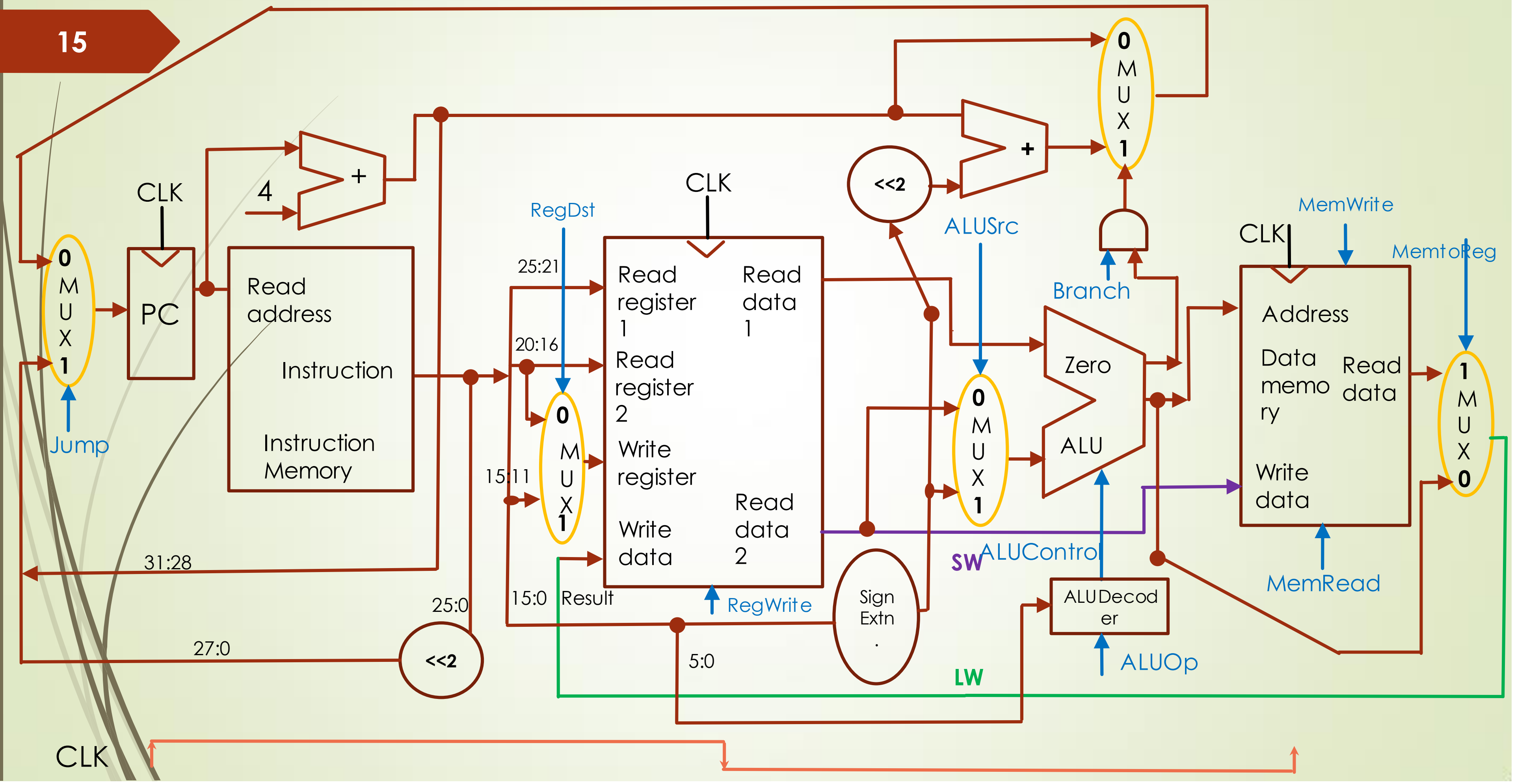


Combined Fetch cycle, R, M, I and B-type data-path

Jump signal decides if PC **+4** or the jump address.



Combined Fetch cycle, R, M, I, B and J-type data-path and clock



Identify the control signals

- Jump
- RegDst
- RegWrite
- ALUSrc
- Branch
- ALUOp
- MemRead
- MemWrite
- MemtoReg

Generation of Controls

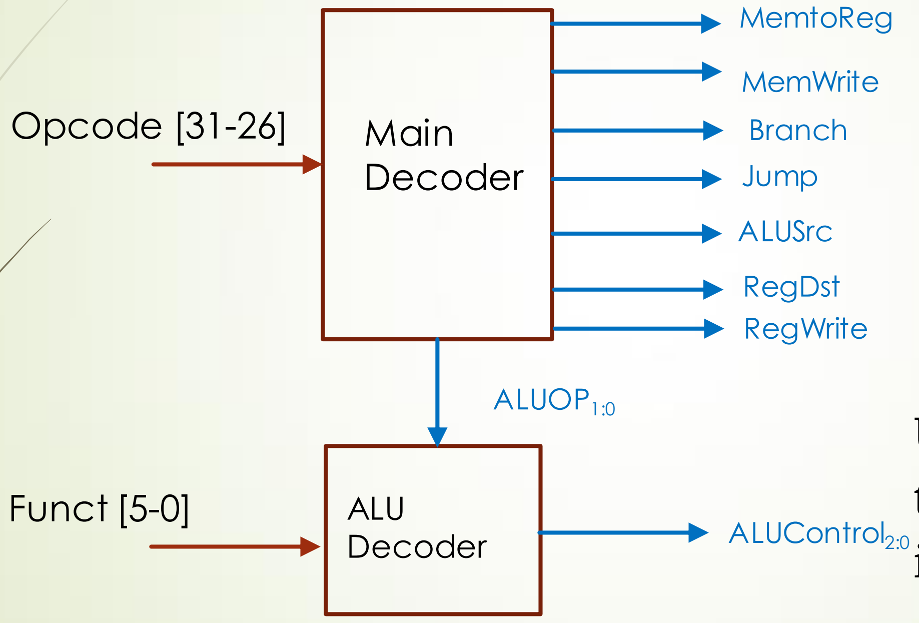
Inputs to the control unit: op-code part [31:26] and funct part [5:0] of the instruction

ALUOp	Meaning
00	add
01	subtract
10	Look at *funct* field
11	n/a

Output of the control unit:

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 Unit



Uses ALUop signal and funct to decide which function to actually invoke

Generation of Controls

Inst. opcode	ALUOp	Instr. operation	Funct field	Desired ALU action	ALUControl
100010 (LW)	00	load word	xxxxxx	add	0010
100011 (SW)	00	store word	xxxxxx	add	0010
000100 (BEQ)	01	branch equal	xxxxxx	subtract	0110
000000 (R-type)	10	add	100000	add	0010
R-type	10	Subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111
001000 (addi)	00	Immediate	xxxxxx	add	xxxx
000010 (j)	xx	jump	xxxxxx	jump	xxxx

Single-cycle implementation

- The previous design is called single-cycle implementation
- The instruction memory, register file and data memory are all read combinationally
- The new instruction appears to output of instruction memory after some propagation delay, if the address changes
- Operations are done on rising edge of the clock
- The single-cycle microarchitecture executes an entire instruction in one clock cycle
- Simple control unit (why?)

Performance analysis of Single-cycle implementation

- Need some quantity (or metric) for comparison of two design
- How does one measure the effectiveness of new design?

Performance analysis of Single-cycle implementation

- Execution time of a program is a metric
- $Execution\ Time = (\#instructions)\left(\frac{Cycles}{instruction}\right)\left(\frac{Seconds}{cycle}\right)$
- #instructions or length of a program depends on ISA
- Complicated Vs. Simple ISA
- The number of cycle per instruction (on an average) is called *CPI*
- Throughput = 1/CPI
- Assumption: an ideal memory model
- The number of seconds per cycle is the clock period (?)

Performance analysis of Single-cycle implementation

- CPI = 1, for single-cycle implementation
- lw-instruction decides Critical path (T_c)
- $T_c = t_{pcq\_PC} + t_{mem} + \max\{t_{RFread}, t_{select} + t_{mux}\} + t_{ALU} + t_{mem} + t_{mux} + t_{RFwrite}$
- Register read takes longer time than mux selection
- $T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFwrite}$

Performance analysis of Single-cycle implementation

- 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.

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

Performance analysis of Single-cycle implementation

- 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.
- $T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFwrite}$

  = 30 + 2(250) + 150 + 200 + 25 + 20 = 925
- 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
$t_{RFwrite}$	20

Summary

- Disadvantages of CISC-style processor
- RISC-style processor organization
- Design of datapath for Single-cycle processor
- Design of controls for Single-cycle processor
- Performance analysis of Single-cycle processor