ECE 485/585
Computer Organization and Design

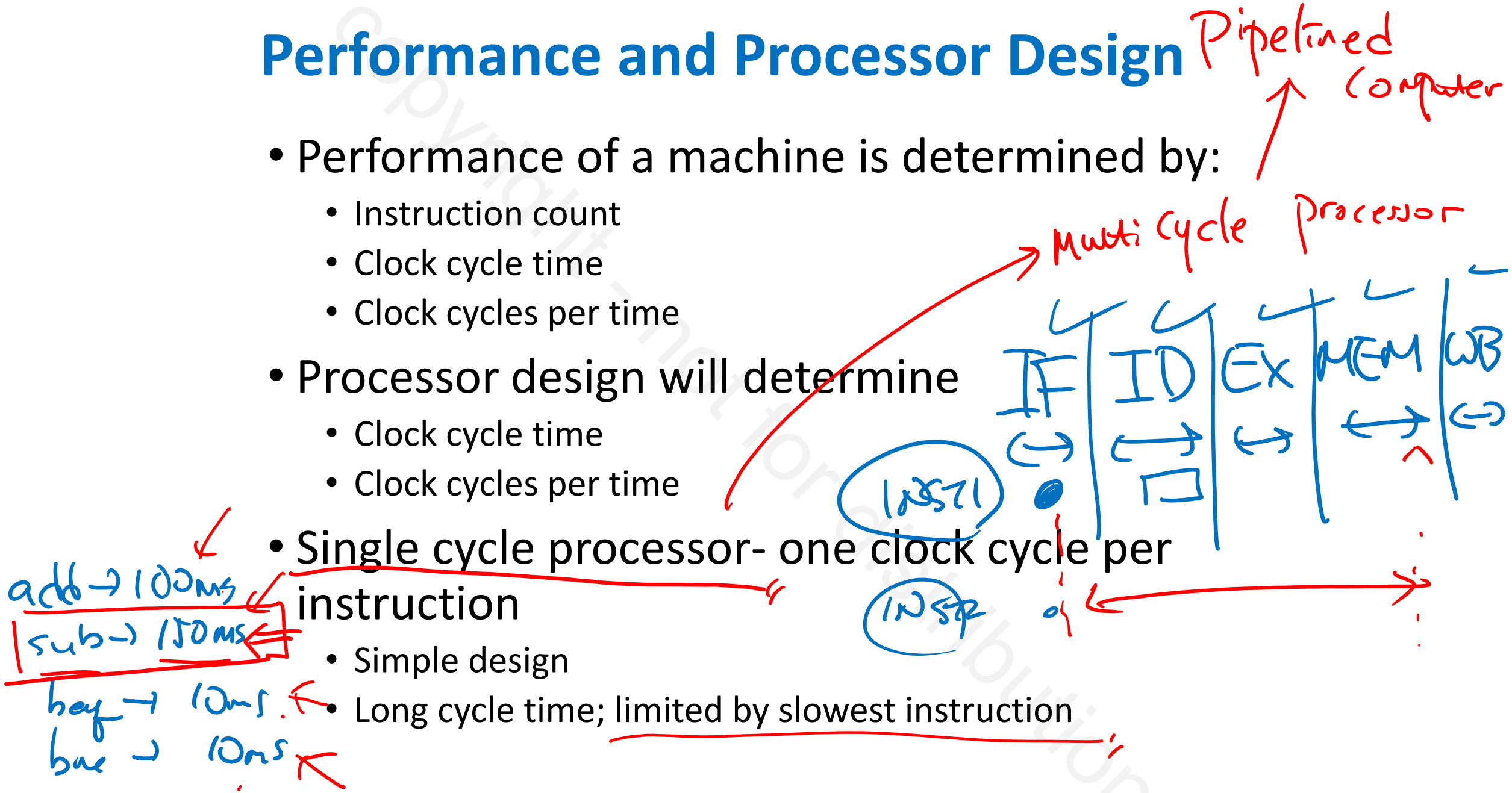
Lecture 7: Datapath Design – Part 1
Fall 2022

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Performance and Processor Design

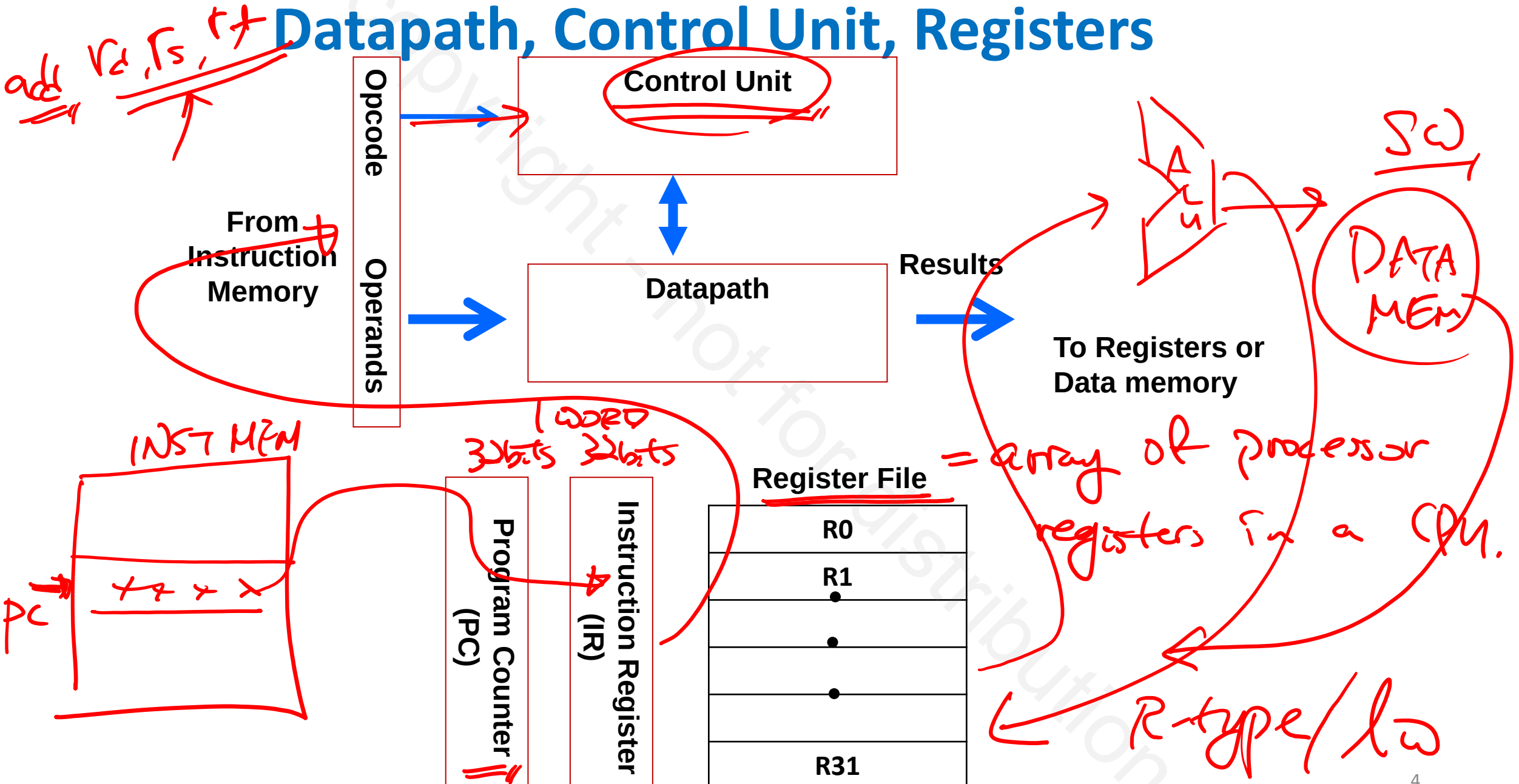
- Performance of a machine is determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per time
- Processor design will determine
  - Clock cycle time
  - Clock cycles per time
- Single cycle processor- one clock cycle per instruction
  - Simple design
  - Long cycle time; limited by slowest instruction



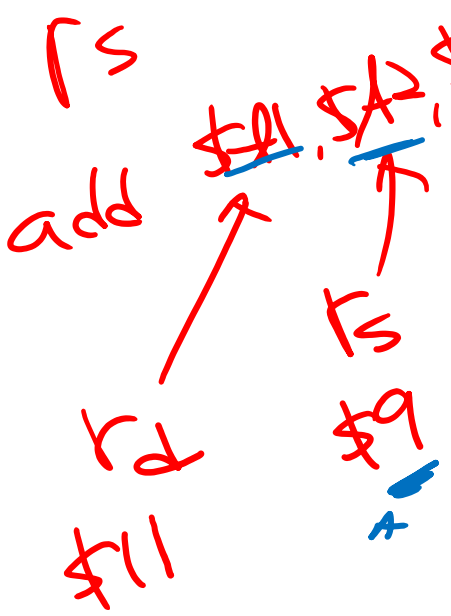
Central Processing Unit (CPU)

- Datapath
  - The hardware through which the data being processed and the results generated travel through
  - Associated signals for operations identified
- Control Unit (CU)
  - Interpretation of the instruction
  - Receiving and providing Signals to the Datapath
- Set of Registers
- Data Processing, Computation, Decision Making
  - Arithmetic and Logic Unit (ALU)

Datapath, Control Unit, Registers



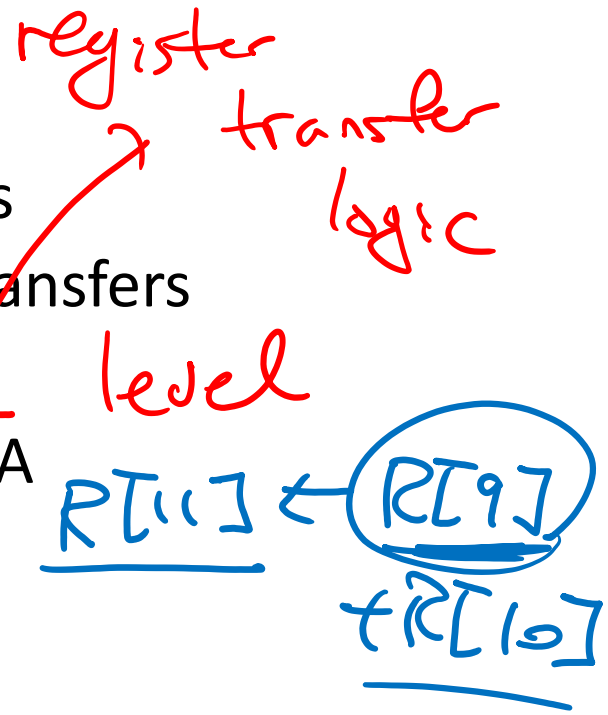
How to Design a Processor



- Analyze instruction set => datapath requirements
  - Meaning of instructions is given by register transfers

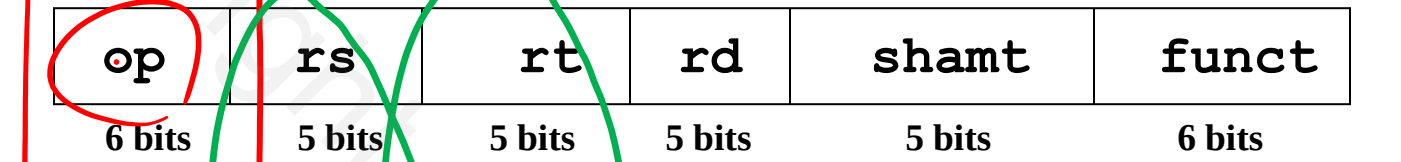
R[rd]<-R[rs]+R[rt]

  - Datapath must include storage element for ISA registers
  - Datapath must support each register transfer
- Select datapath elements and establish clocking methodology
- Design datapath to meet the requirements
- Analyze implementation of each instruction to determine control points
- Design the control logic



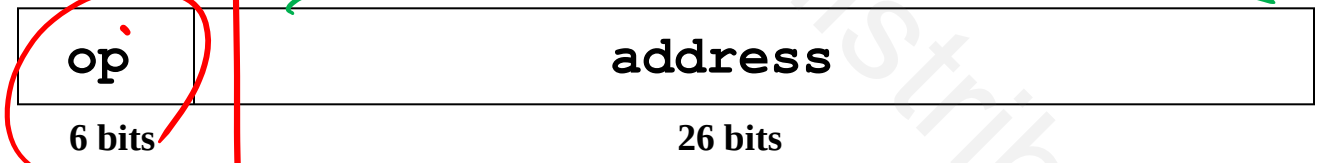
MIPS Instruction Formats

- R-type:



- I-type:

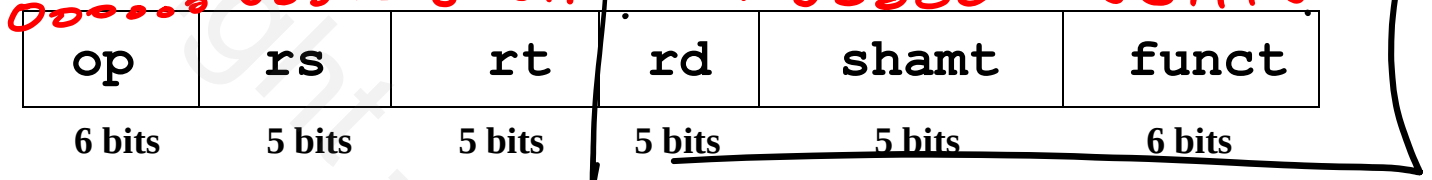
- J-type:



design a H/W that satisfies/decodes/executes all 3 types MIPS INST.

MIPS subset

- ADD and SUB

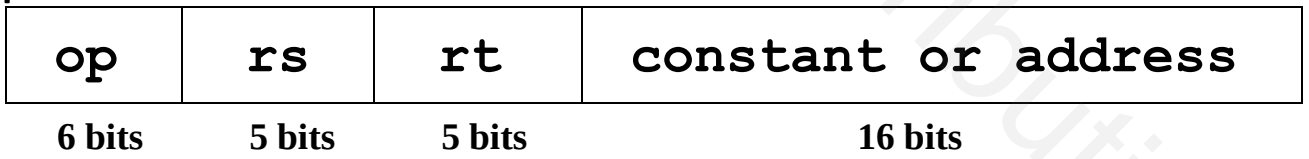



```
addu rd,rs,rt
subu rd,rs,rt
```

- Load and Store

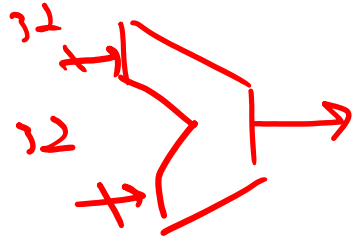

```
lw rt,rs,Imm16
sw rt,rs,Imm16
```

- Branch




```
beq rs,rt,Imm16
```

Register Transfer Logic (RTL)



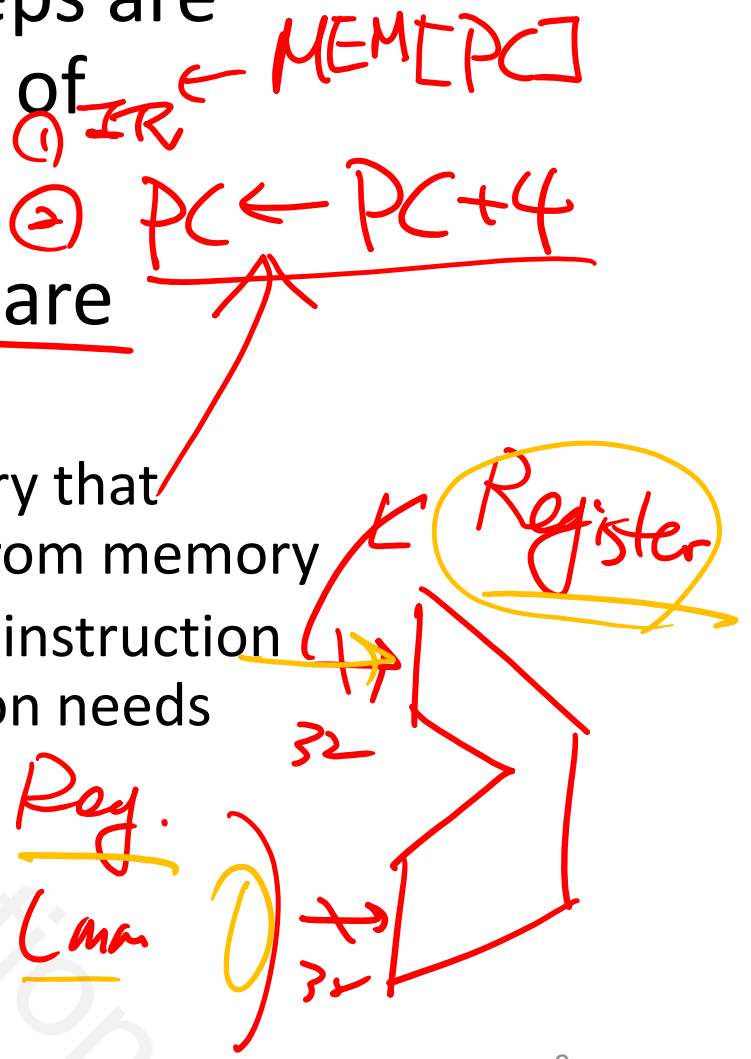
- RTL gives the meaning of instructions
- All instructions start by fetching the instruction:

```
op | rs | rt | rd | shamnt | funct = MEM[PC]
op | rs | rt | Imm16 = MEM[PC]
```

Inst	Register Transfers	
Addu	R[rd]<-R[rs]+R[rt];	PC <- PC+4
subu	R[rd]<-R[rs]-R[rt];	PC <- PC+4
load	R[rt]<-MEM[R[rs]+sign_ext(imm16)];	PC <- PC+4
store	MEM[R[rs]+sign_ext(imm16)]<-R[rt];	PC <- PC+4
beq	if (R[rs]==R[rt]) then PC<-PC+4+(sign_ext(imm16)<<2) else PC <- PC+4	

Implementation Overview

- Regular structure – almost all the steps are same independent of the exact class of information
- For every instruction, first two steps are identical:
  - Send the program counter (PC) to the memory that contains the code and fetch the instruction from memory
  - Read one or two registers, using fields of the instruction to select the registers to read. Load instruction needs only one register read.



Implementation Overview

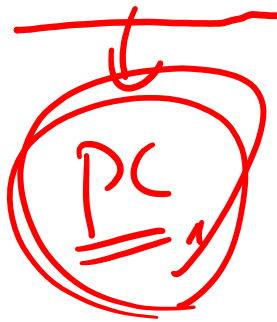
- After these two steps, the next steps depend on the instruction class
- However the actions are largely same for all three instruction classes (memory-reference, arithmetic-logical, branches)
  - All instructions except **`jump`** use ALU after reading the registers
  - WHY?
    - Memory-reference instructions for address calculation
    - Branches for comparison
    - Arithmetic for operation execution

Implementation Overview

- After using ALU:
  - Memory reference instruction will access memory either to write data for a **`store`** or read data for **`load`**
  - Arithmetic-logical instruction must write data from ALU back into register
  - Branch instruction may need to change next instruction address based on the comparison result

I type (lw) (sw)

R type.

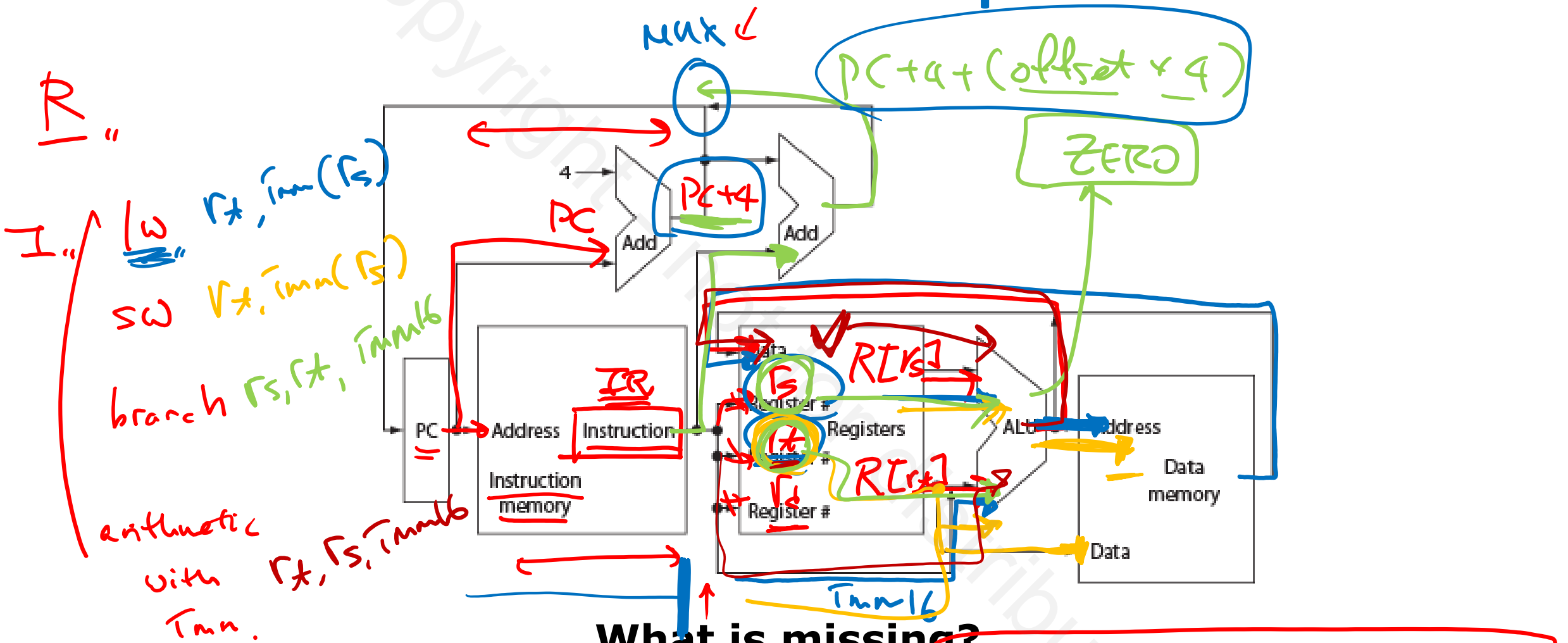


$rs, $t

① PC+4+(Offset x4)

② PC+4

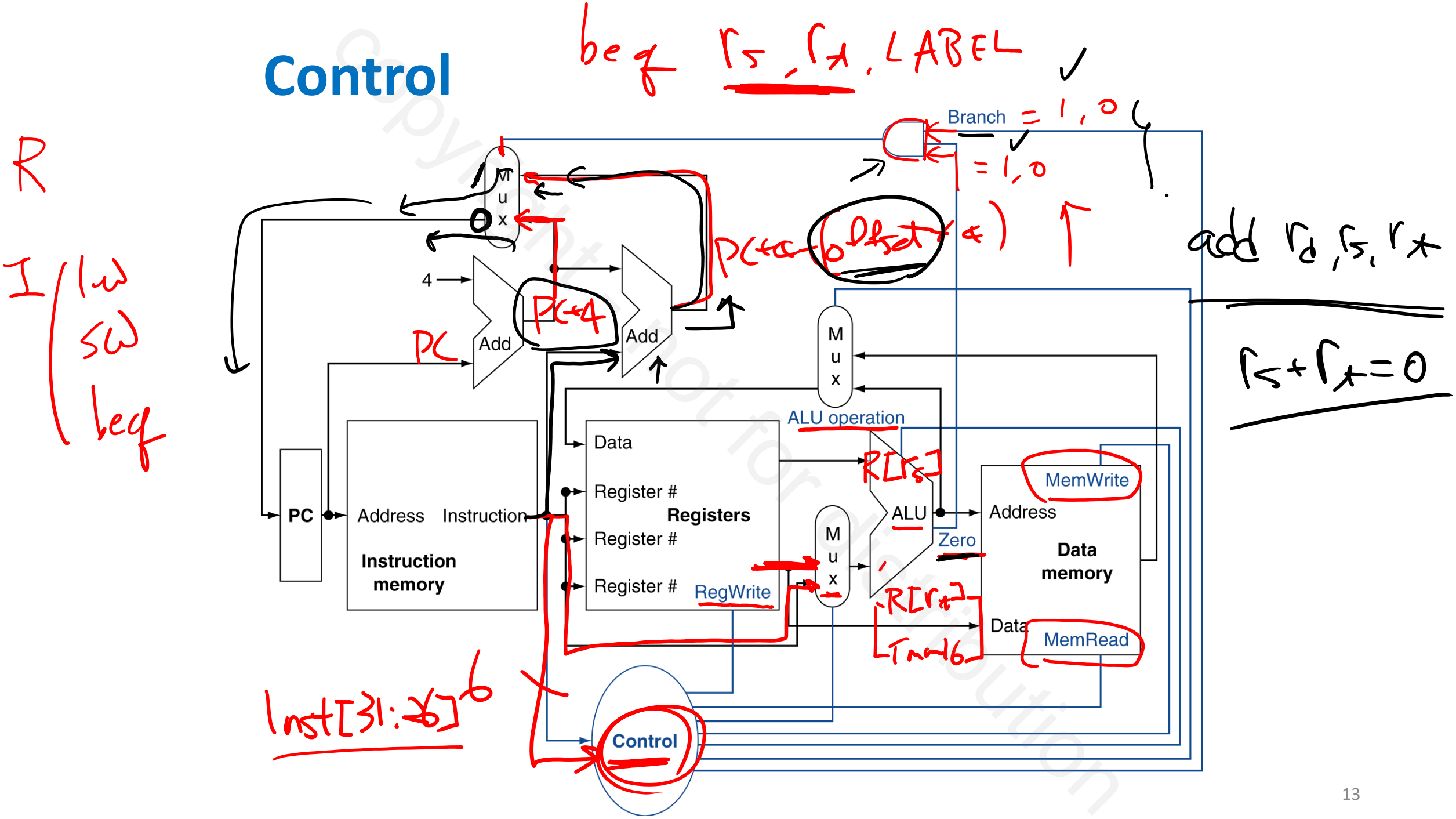
Abstract View of MIPS Implementation



What is missing?

CONTROL SIGNALS

Control

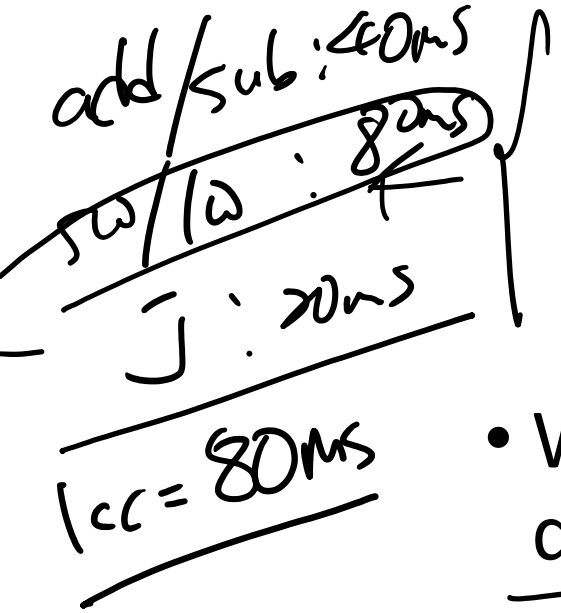


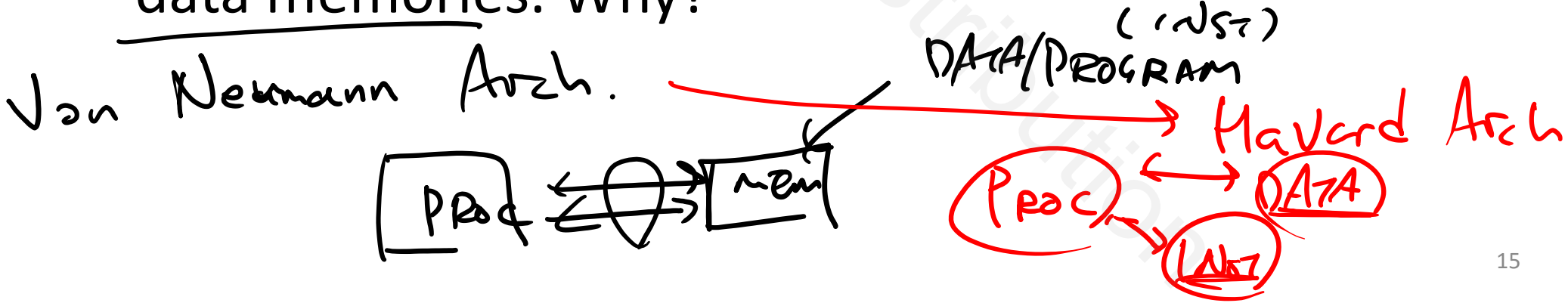
Next

- More detail for functional units
- Increase connections between units
- Add control unit to control what actions are taken depending on the instruction classes

Single Clock Cycle Implementation

- Start with a simple implementation
- Single clock cycle design:
  - Every instruction begins execution at one clock edge and completes execution on next clock edge
  - Easy to understand but not practical
  - Slower than multicycle design that uses multiple clock cycles for each instruction
- We need separate instruction and memory & data memories. Why?





Why seperate INST MEM & DATA MEM

IF ID Ex mem WB

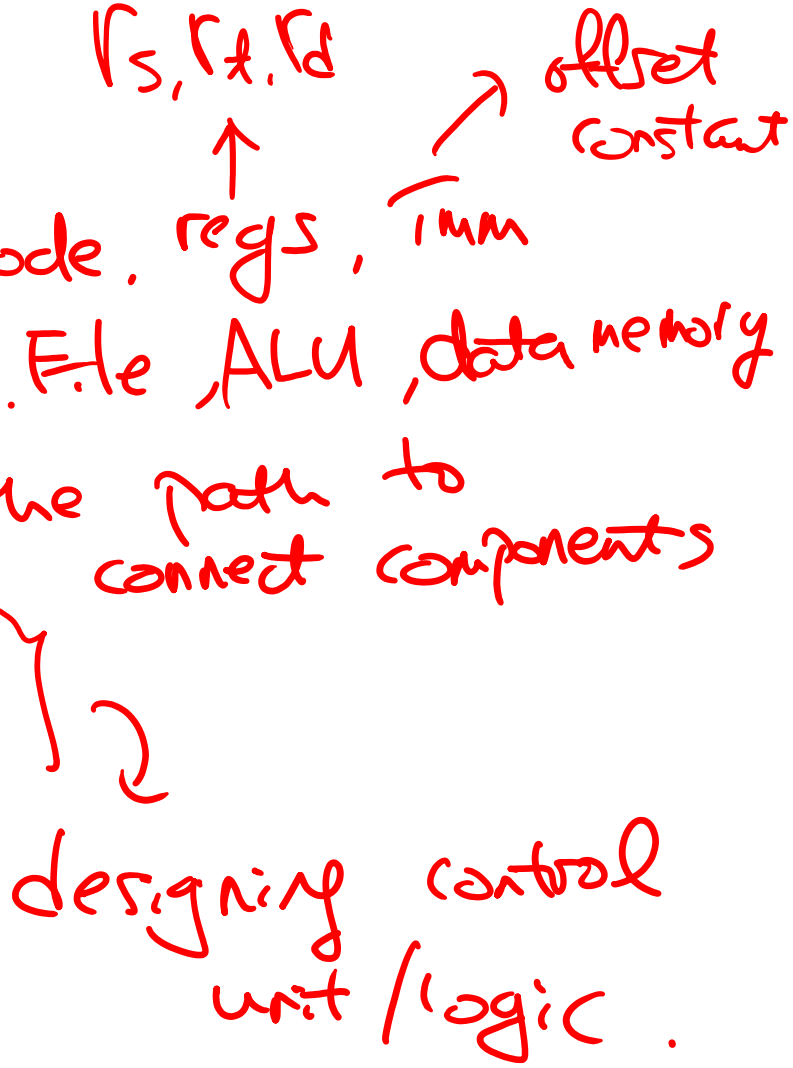
① to fetch next Instruction While current Instruction is executing

② Instruction MEM → Read only
data MEM → R/W

} not to overwrite current. Instruction

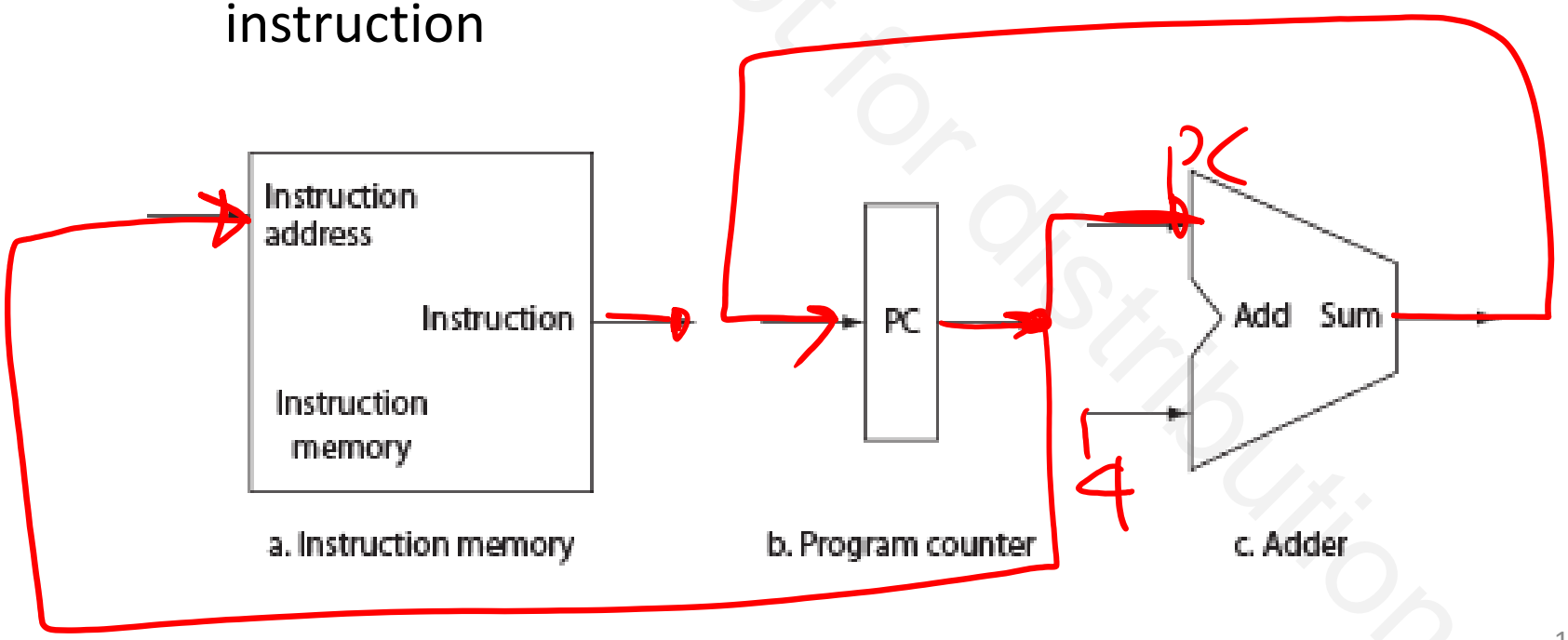
Building a Datapath

- Step 1: Instruction Set analysis
- Step 2: Datapath components
- Step 3: Datapath design
- Step 4: Control points determined
- Step 5: Design of the control unit



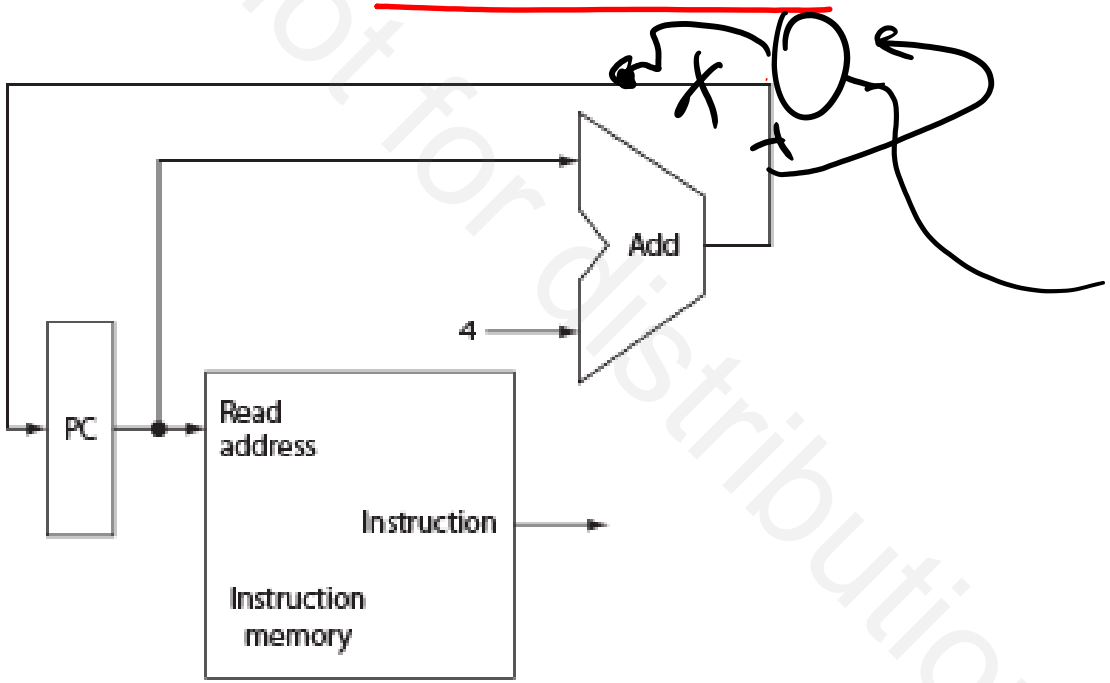
Components of the Datapath

- Instruction fetching:
  - Memory unit to store and supply instructions
  - Program counter register used to hold the address of the current instruction
  - Adder to increment the PC to the address of the next instruction



Instruction Fetch Unit

- Instruction fetch: mem[PC]
- Update program counter:
  - Sequential code: PC <-PC+4
  - Branch and Jump: PC <- destination address



R-type Instruction Implementation

- Let's consider R-type instructions

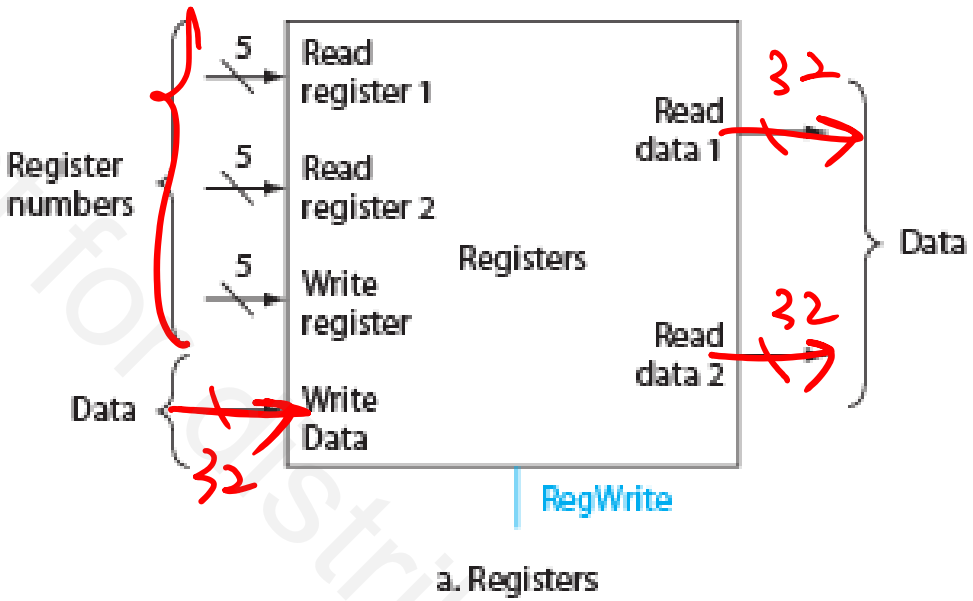
  (arithmetic–logical instructions)

```
R[rd] <- R[rs] op R[rt]
```

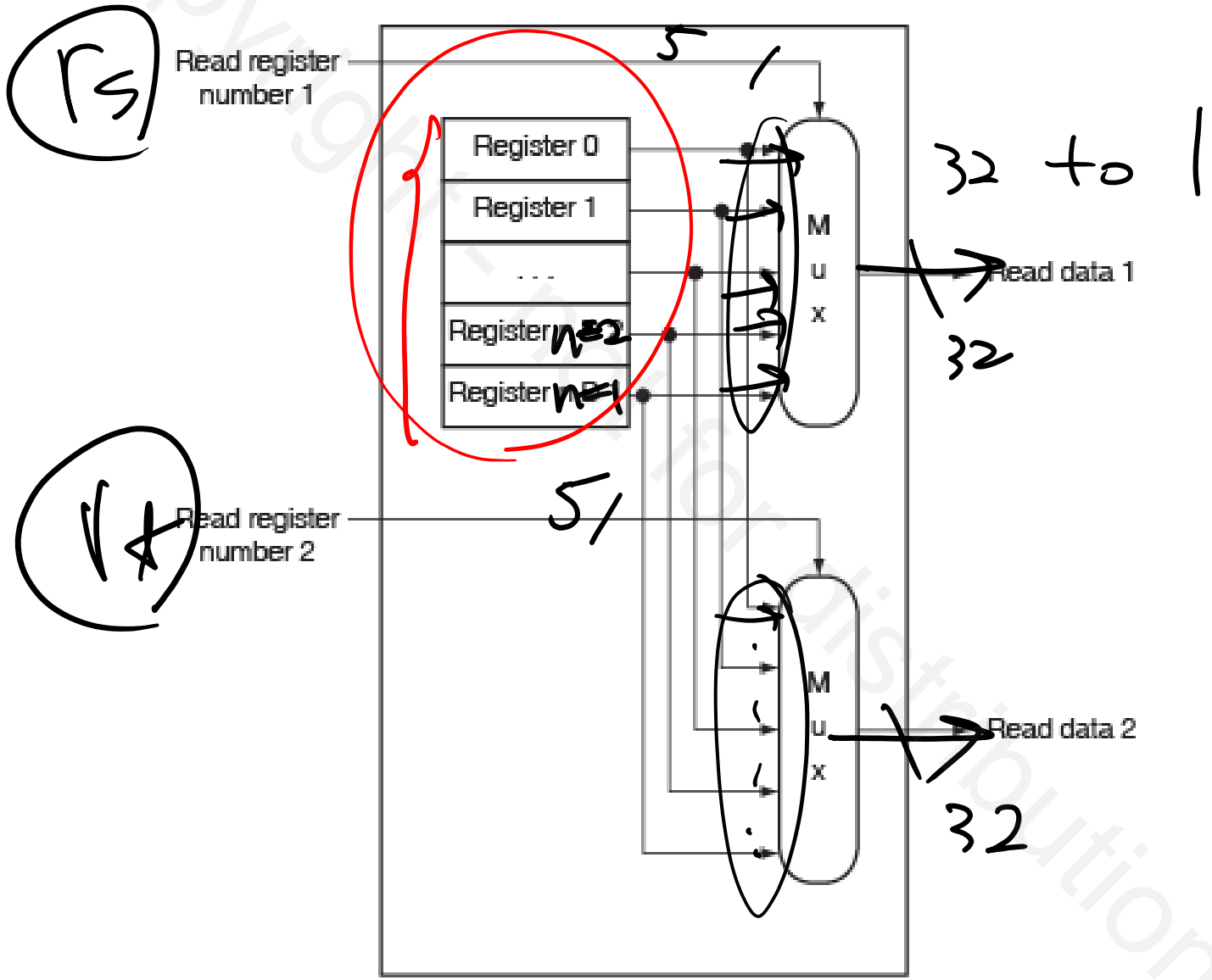
- Read two registers , perform an ALU operation and write the results to destination register
- We need a register file
- We need an Arithmetic Logic Unit (ALU)

Storage Element: Register File

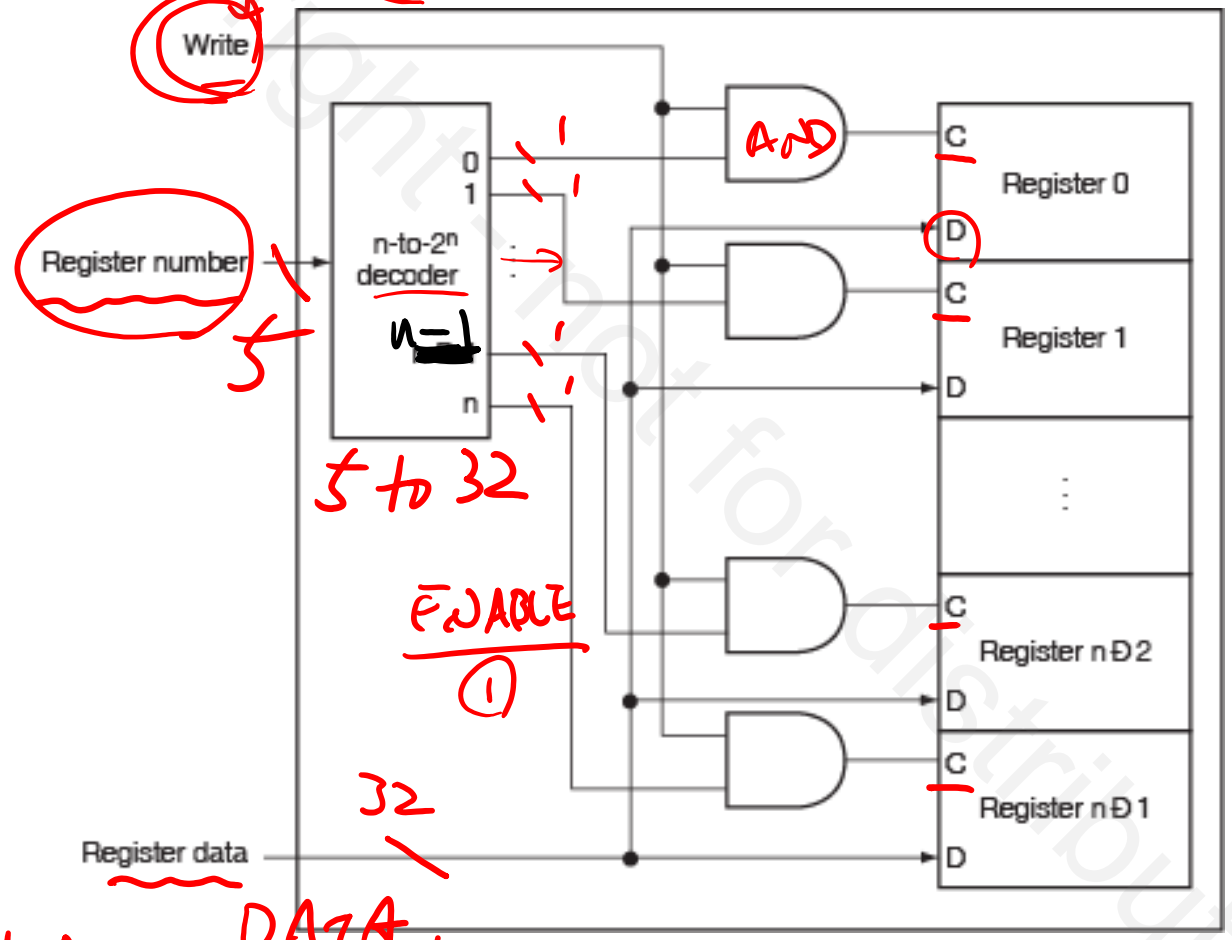
- Register file contains 32 registers
- Two 32-bit output buses
- One 32-bit input bus
- Registers are selected by 5-bit control signals
- Register file always outputs the contents of Read register inputs
- Writes are however controlled by write control signal which must be asserted for a write to occur at the clock edge



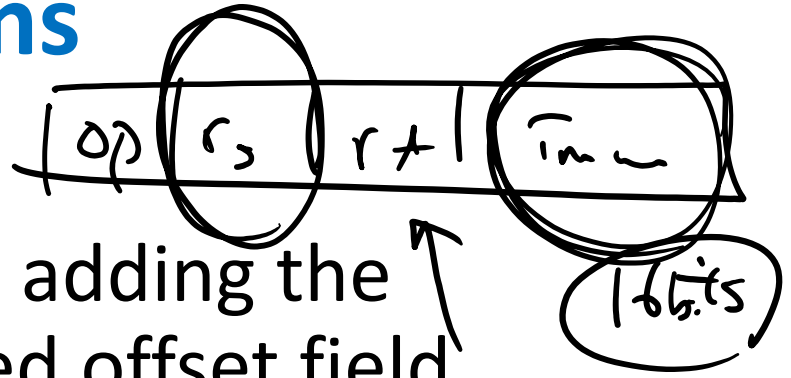
Register File – Read Access



Register File – Write Access



Load and Store Instructions Implementation



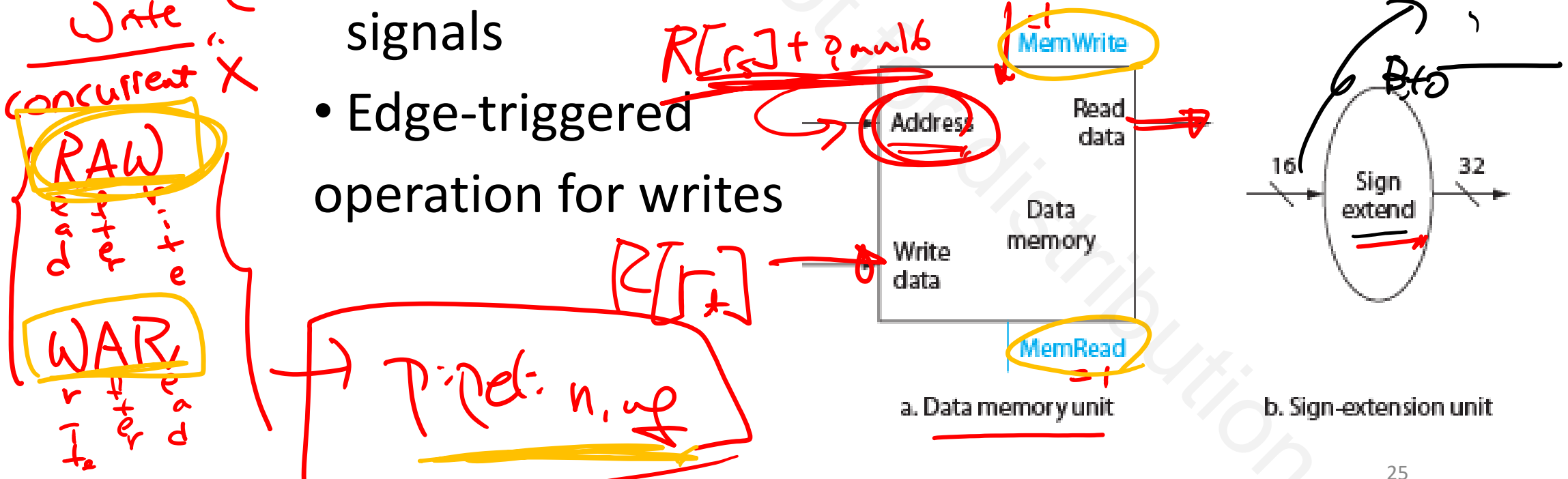
- Compute a memory address by adding the base register to the 16-bit signed offset field contained in the instruction. Either read from the register file for **`store`** instruction or write to the register file for **`load`** instruction:

`R[rt]<-MEM[R[rs]+sign_ext(imm16)]`

- Both register file and ALU is required
- In addition:
  - a unit to sign-extend the 16-bit offset field in instruction to 32-bit
  - A data memory unit to read from or write to.

Data Memory Unit

- Inputs for address and write data
  - Address selects data to put on the bus
- Single output for read result
- There are separate read and write control signals
- Edge-triggered operation for writes



a. Data memory unit

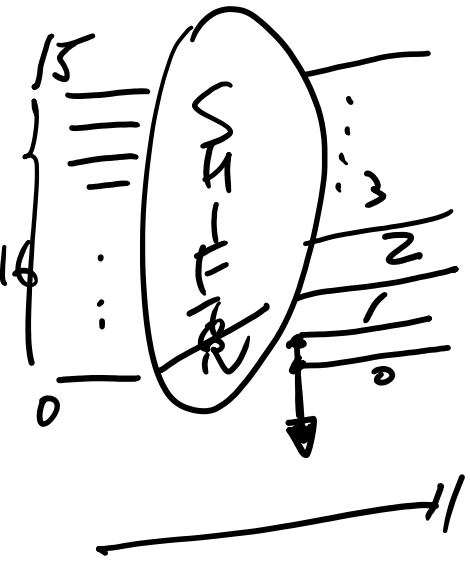
b. Sign-extension unit

Branch Instruction Implementation

- Branch instruction **`beq`** has three operands; two registers and a 16-bit offset

```
beq rs,rt,Imm16
```

- To implement this instruction, compute branch target address by adding sign-extended offset to PC
  - Base for the branch address calculation is the address of the next instruction following the branch
  - Offset field is shifted left 2 bits so that it is a word offset, increasing the effective range of the offset field by a factor of 4



BRANCH

PC ← PC+4;

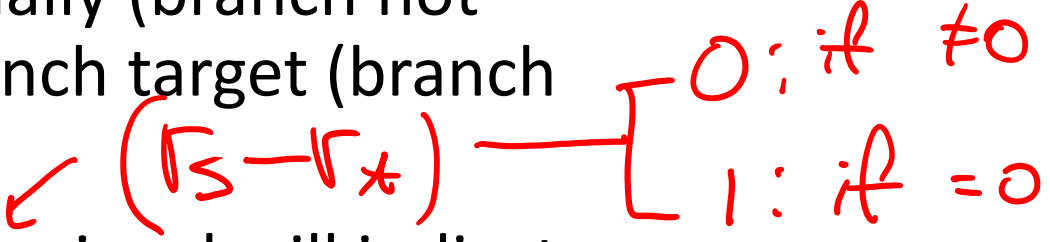
PC ← (PC+4) + (Imm<<2)

IR[15:0]<<2

x4

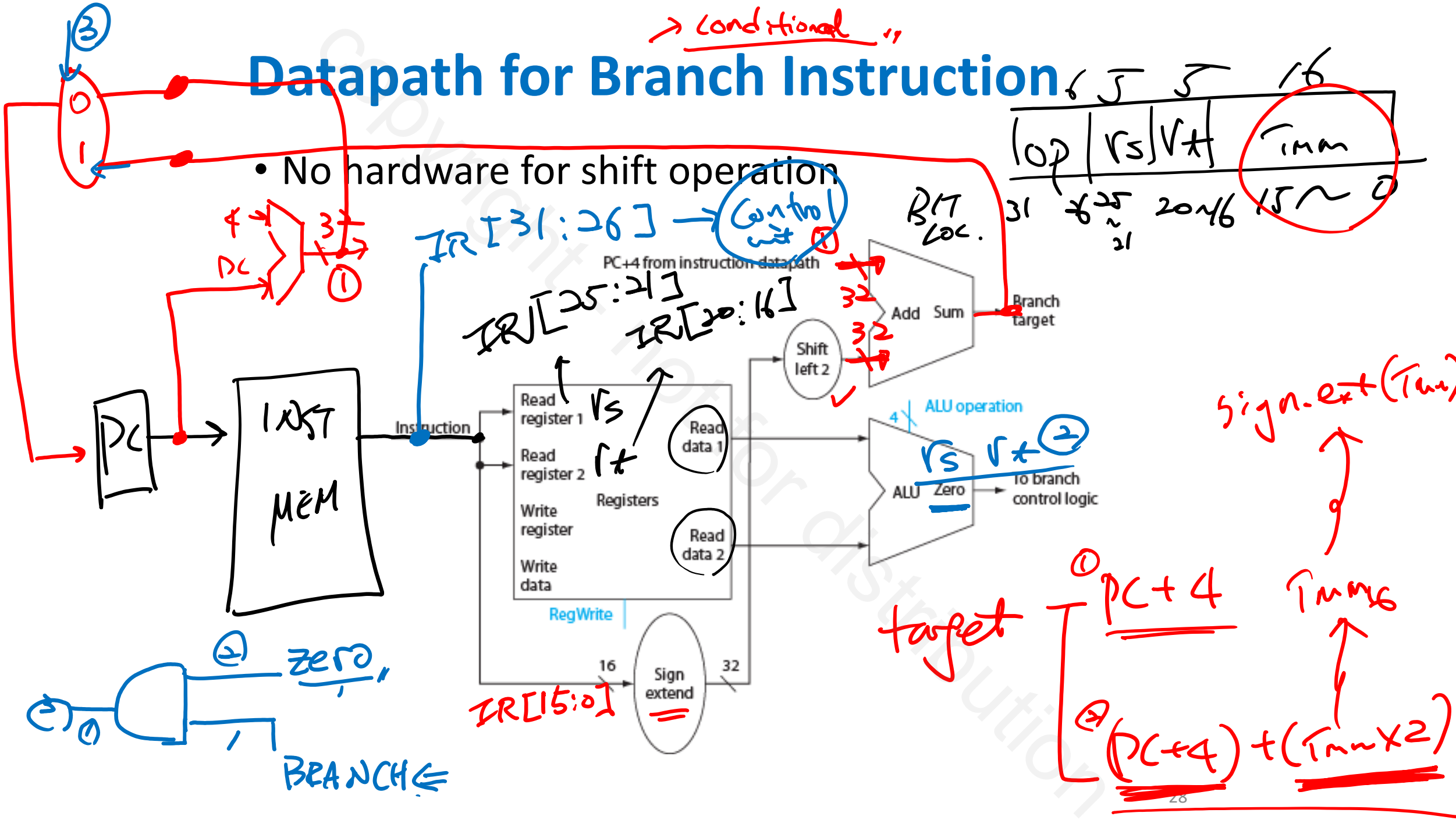
Branch Instruction Implementation

- Determine whether next instruction is the instruction that follow sequentially (branch not taken) or instruction at the branch target (branch taken)
- Use ALU to do comparison: Zero signal will indicate if the result was 0
- For computing the destination address, use sign extension unit and adder



Datapath for Branch Instruction

- No hardware for shift operation

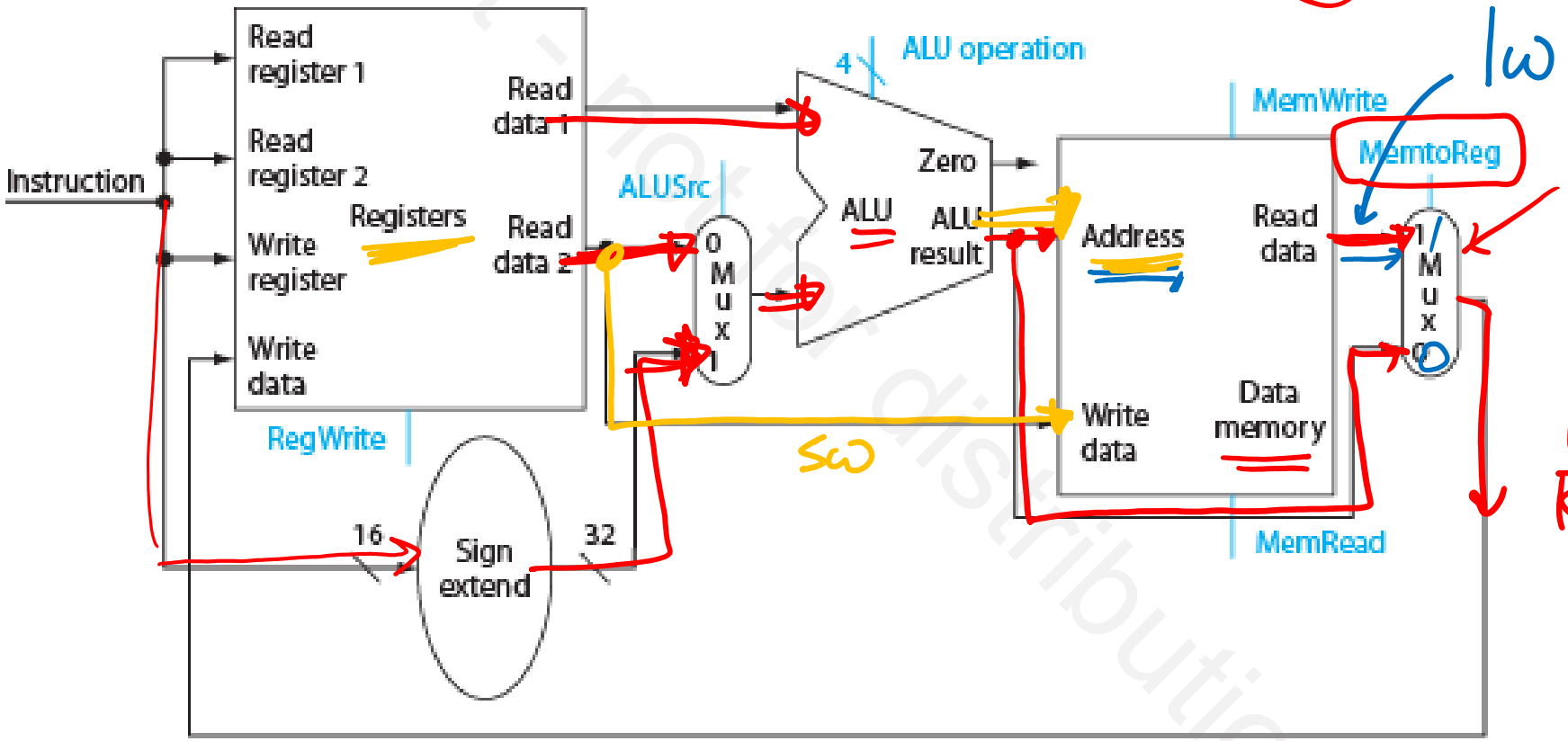


Combining the Datapath

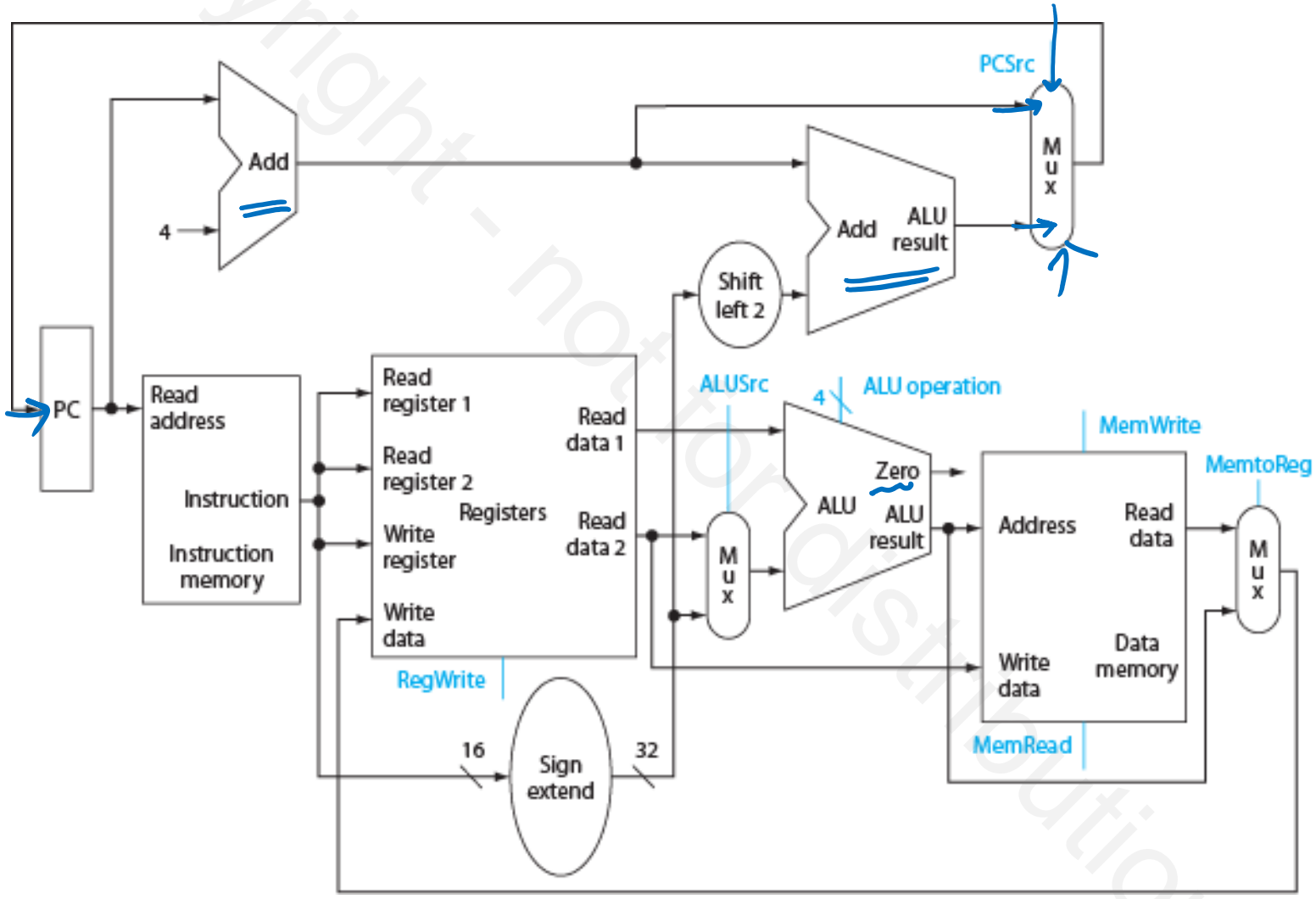
- Single clock cycle means no datapath resource can be used more than once per instruction
- Duplicate any element that is used more than once
  - Separate memories for data and instruction
  - Multiple functional units
- In order to share datapath elements between two instruction classes , use multiplexors and control signals to allow multiple connections to the input of an element

Datapath for Memory and R-type Instructions

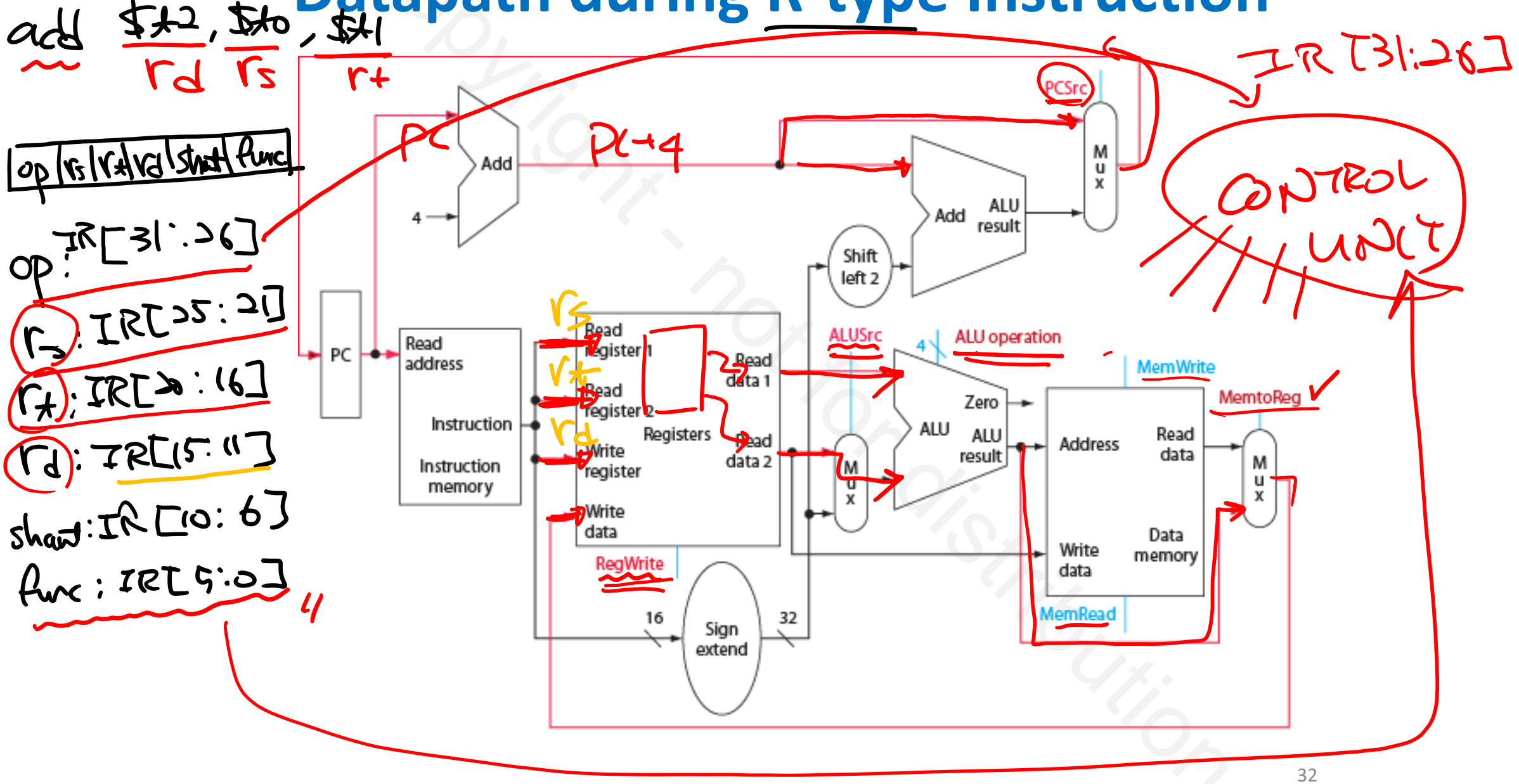
- Two multiplexors added



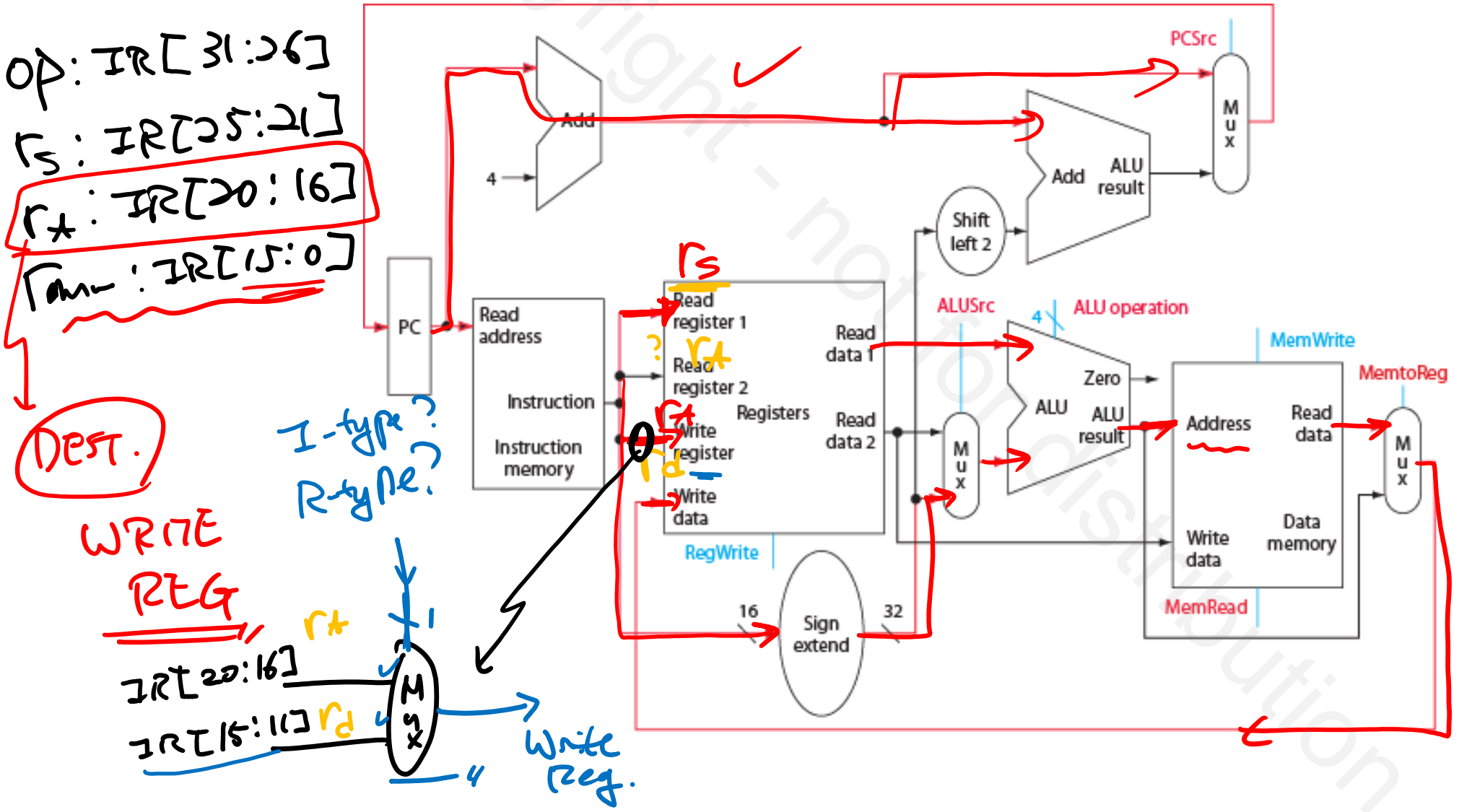
Single Cycle Datapath



Datapath during R-type Instruction

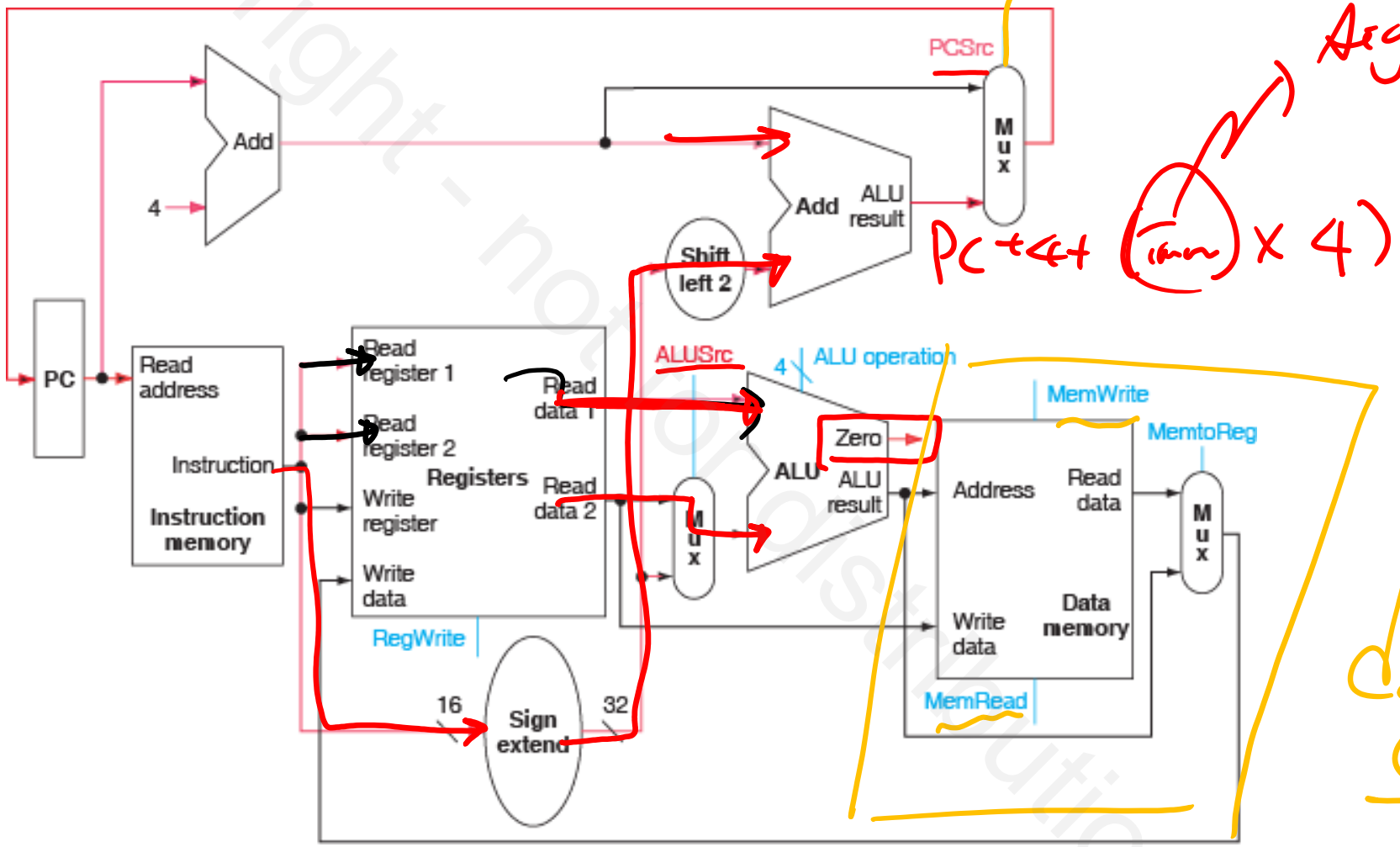


Datapath during Load Instruction



Datapath during Branch Instruction

beq $t0, $t1, label



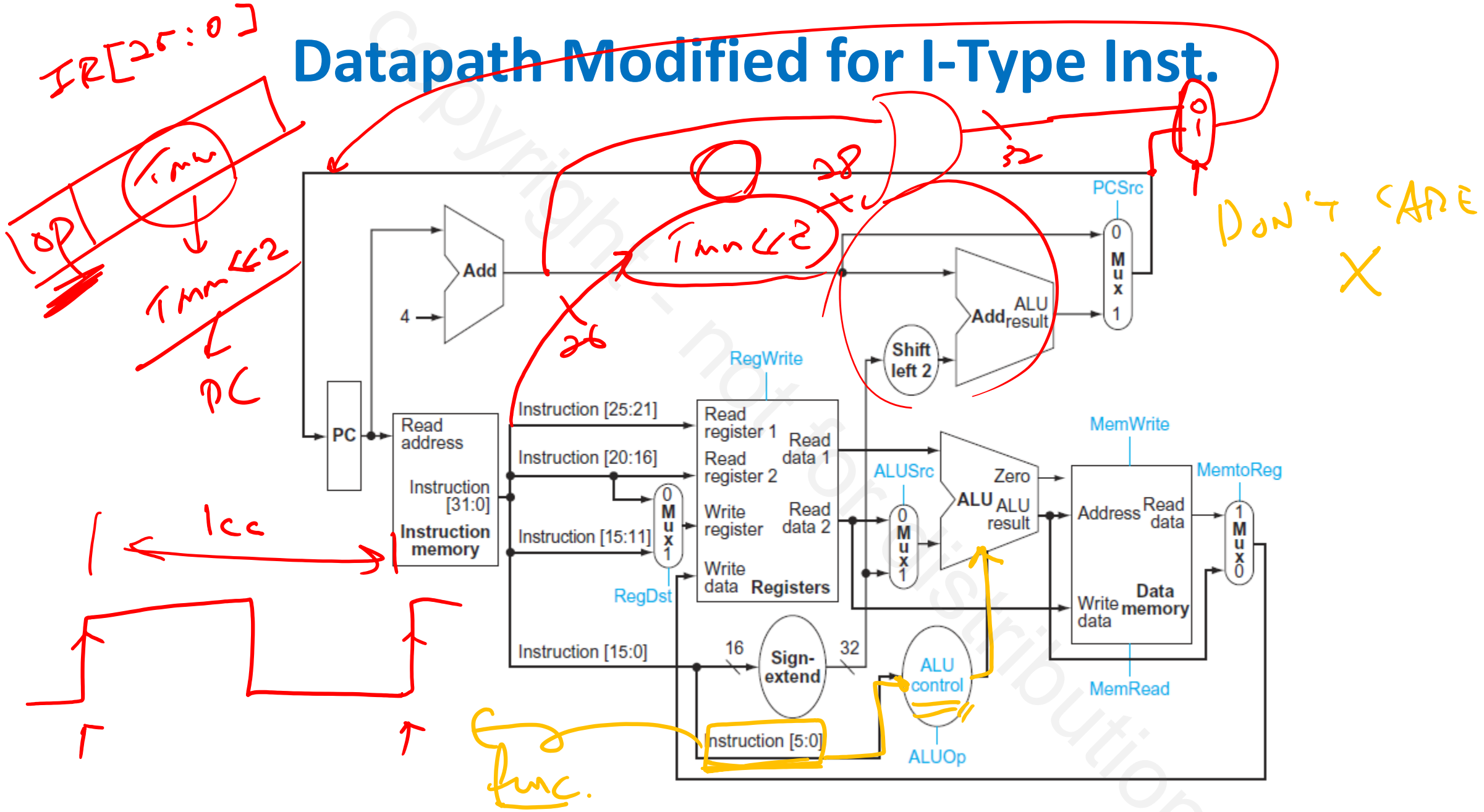
"ZERO" = "BRANCH"

Assert

PC+4+(imm x 4)

don't Care

Datapath Modified for I-Type Inst.



Summary

- MIPS make it easier:
  - Instructions same size
  - Source registers always same place
  - Immediates same size, location
  - Operations always on registers and immediates
- Single cycle datapath:
  - CPI=1; clock cycle time => long