# Computer System Architecture

H. SOUBRA

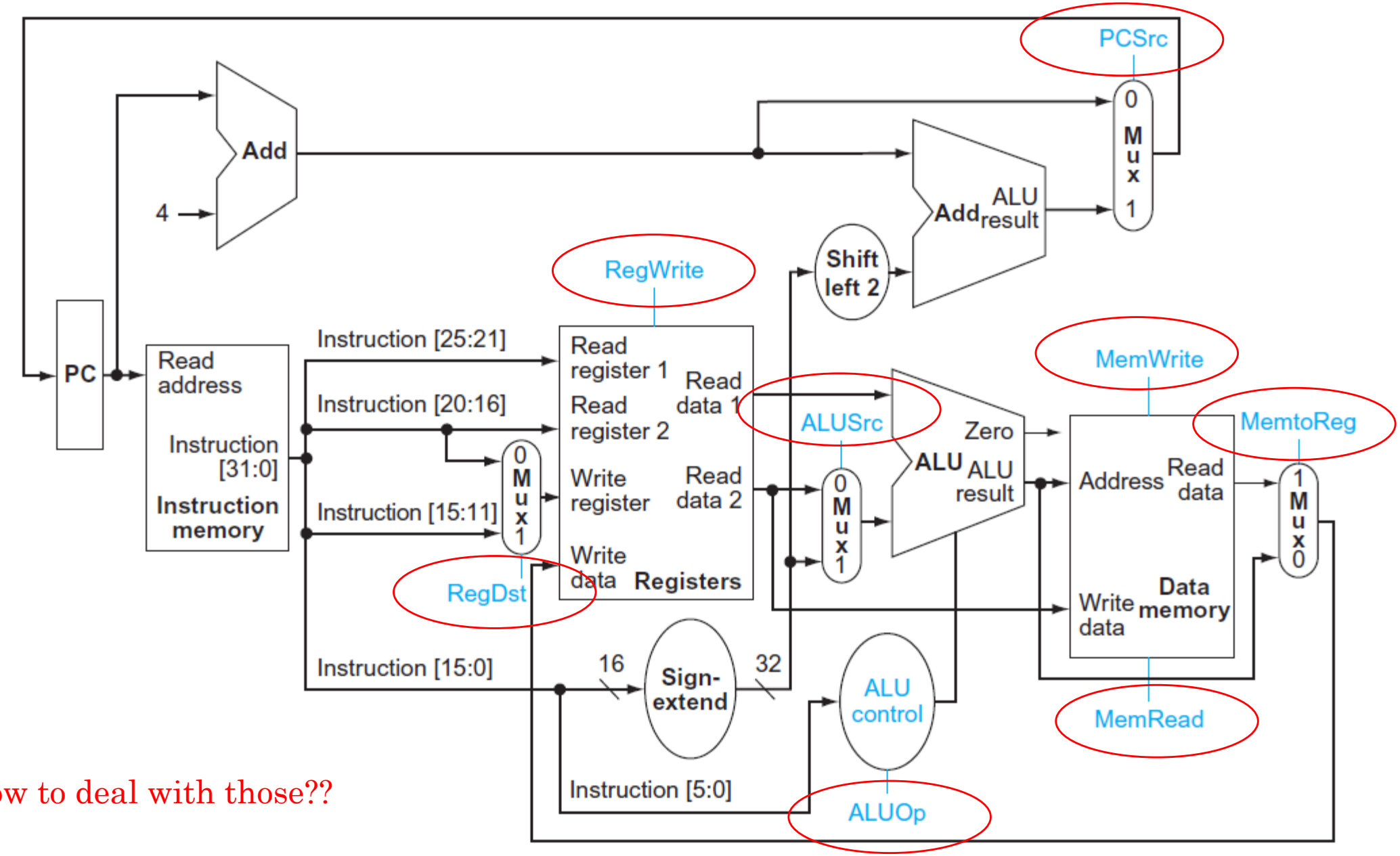
# Disclaimer

This course contains copyrighted material the use of which has not always been specifically authorized by the copyright owner. They are used strictly for educational purposes. The principle of fair use applies.

# Lecture 8: Hardware Implementation

- How to deal with control signals?
- Pipelining!

# Simple Hardware Implementation



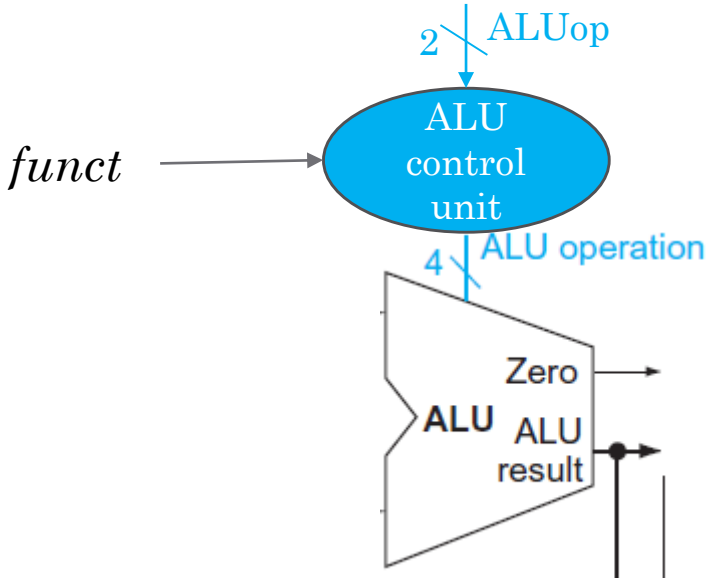

How to deal with those??

SOUBRA© 2022

# Control Signal Table

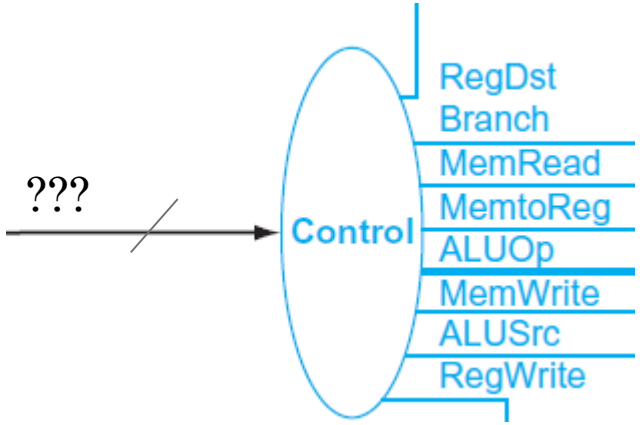
| Instruction | ALUop | RegDst | ALUSrc | RegWrite | MemRead | MemWrite | Branch | MemtoReg |
|---|---|---|---|---|---|---|---|---|
| LW | | | | | | | | |
| SW | | | | | | | | |
| BEQ | | | | | | | | |
| R-TYPE | | | | | | | | |

| Instruction | ALUop 2-bit control | ALU operation | ALU operation 4-bit Input |
|---|---|---|---|
| LW/SW | 00 | ADD | 0010 |
| BEQ | 01 | SUB | 0110 |
| R-TYPE | 10 | *funct* | *funct* |

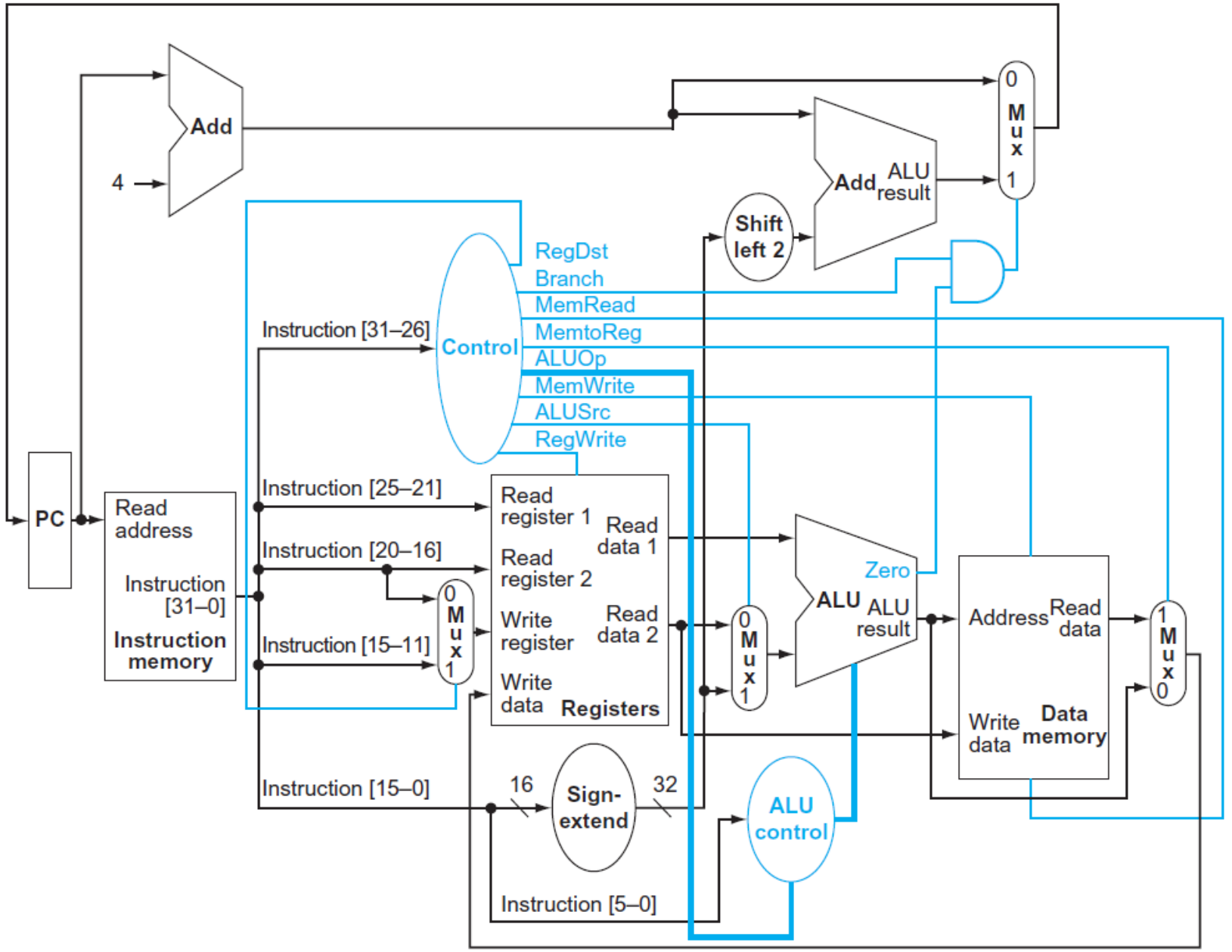


# Control Signal Table

| Instruction | ALUop | RegDst | ALUSrc | RegWrite | MemRead | MemWrite | Branch | MemtoReg |
|---|---|---|---|---|---|---|---|---|
| LW | 00 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| SW | 00 | X | 1 | 0 | 0 | 1 | 0 | X |
| BEQ | 01 | X | 0 | 0 | 0 | 0 | 1 | X |
| R-TYPE | 10 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |



# Full Simple Hardware Implementation



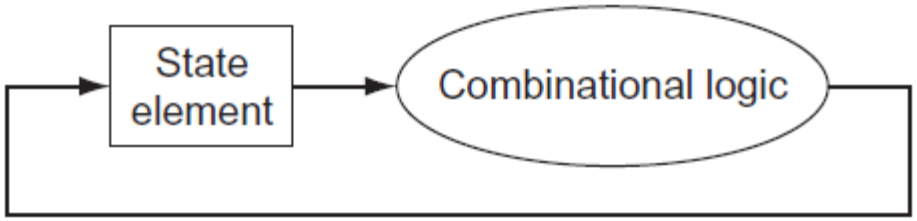
# Control function for the simple implementation

| OPCODE | Instruction | ALUop | RegDst | ALUSrc | RegWrite | MemRead | MemWrite | Branch | MemtoReg |
|---|---|---|---|---|---|---|---|---|---|
| 100011 | LW | 00 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 101011 | SW | 00 | X | 1 | 0 | 0 | 1 | 0 | X |
| 000100 | BEQ | 01 | X | 0 | 0 | 0 | 0 | 1 | X |
| 000000 | R-TYPE | 10 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

SOUBRA© 2022

# Instruction cycles

- How many cycles to execute an instruction? Why?
- Edge triggered
- Memory for instructions separate from one for data!
- And any other element needed more than once must be duplicated



# Performance of the Single-Cycle Implementation

$$Execution\ time\ (in\ \text{Seconds/Program}) = \frac{Instructions}{Program} * \frac{Clock\ cycles}{Instruction} * \frac{Seconds}{Clock\ cycle}$$

1

How big should it be?

SOUBRA© 2022

# Performance of the Single-Cycle Implementation

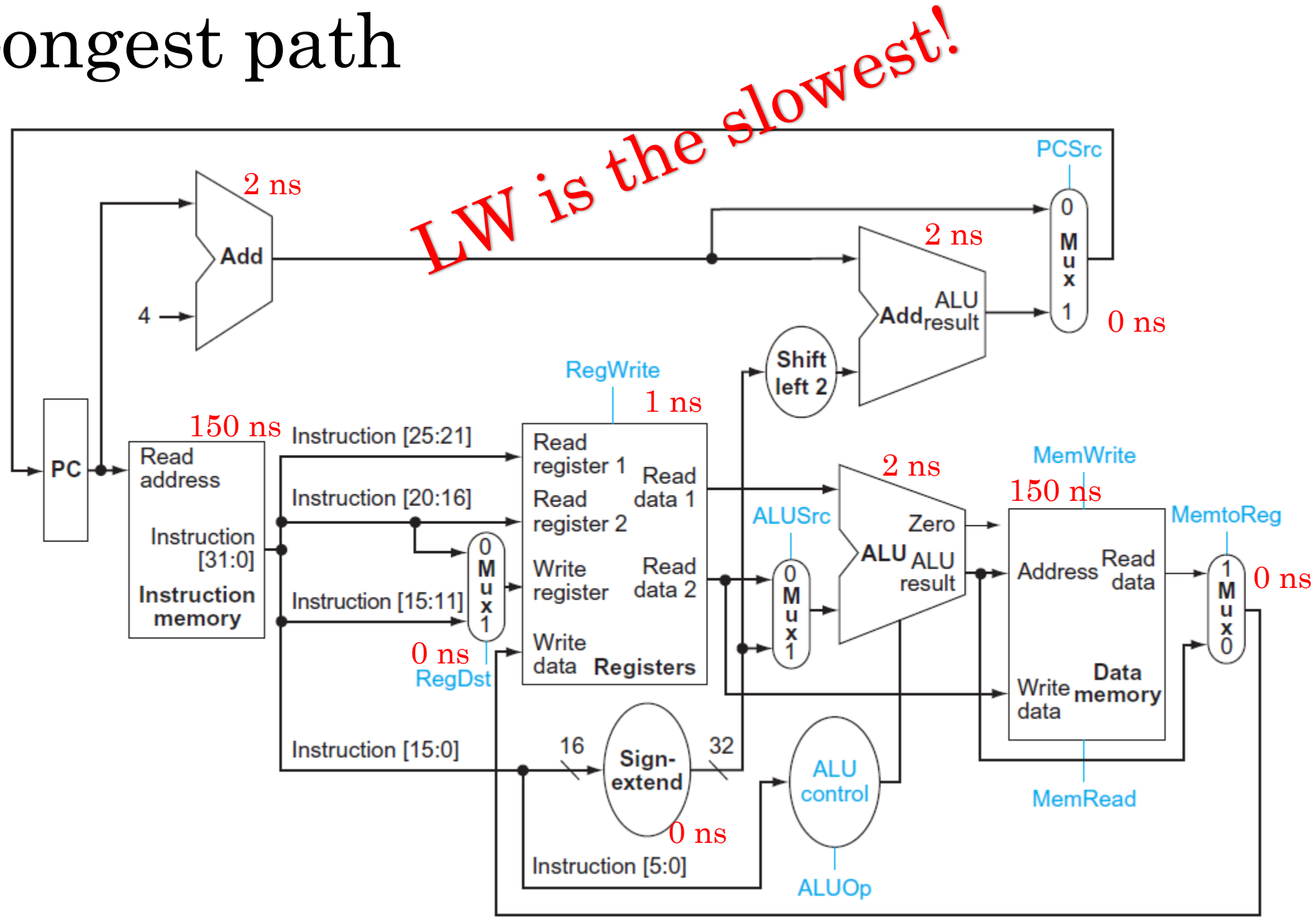
- Single-cycle design will work correctly but inefficiently
- Clock cycle must have the same length for every instruction
- The longest possible path in the processor determines the clock cycle
- Which instruction is the slowest?

$$Execution\ time\ (in\ \text{Seconds/Program}) = \frac{Instructions}{Program} * \frac{Clock\ cycles}{Instruction} * \frac{Seconds}{Clock\ cycle}$$

1

How big should it be?

# Longest path



# Performance of the Single-Cycle Implementation

$$Execution\ time\ (in\ \text{Seconds/Program}) = \frac{Instructions}{Program} * \frac{Clock\ cycles}{Instruction} * \frac{Seconds}{Clock\ cycle}$$

1 How big should it be?

- Lw is the slowest:
  1. reading the instruction memory: 150ns
  2. reading the base register: 1ns
  3. computing memory address: 2ns
  4. reading the data memory: 150ns
  5. storing data back in a register: 1ns

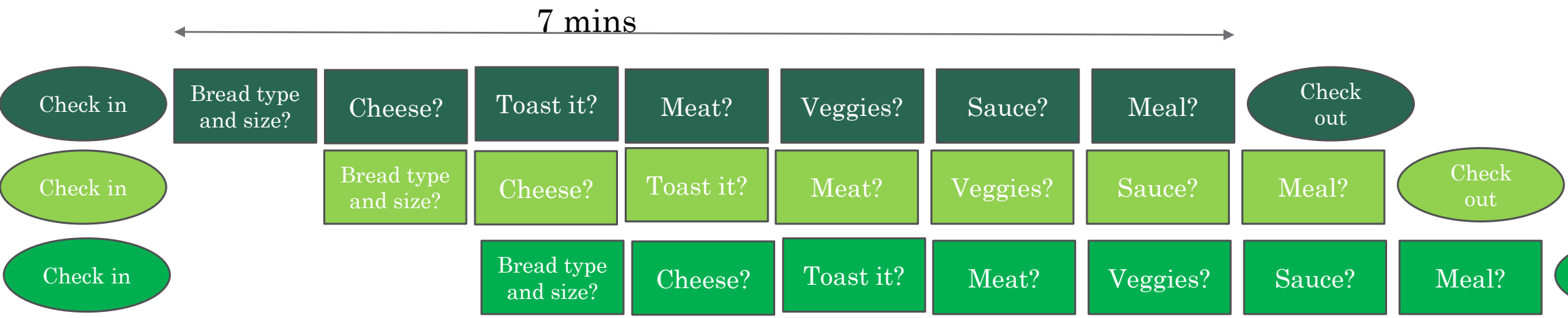
So it takes 304 ns

So the clock should be 1/304*10^-9= 3 MHZ


What to do?

SOUBRA© 2022

# Sandwich making Pipelining

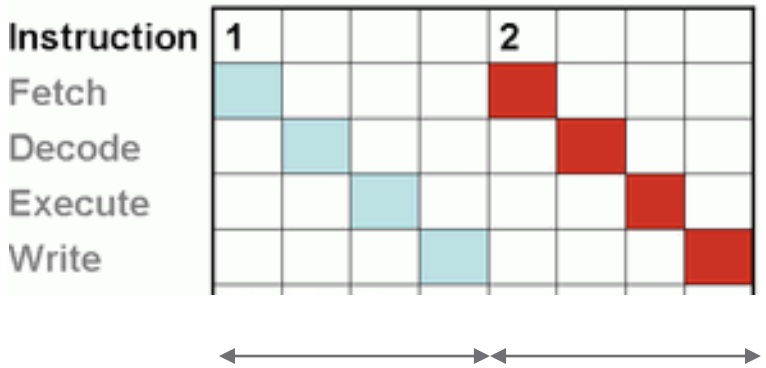




SOUBRA© 2022

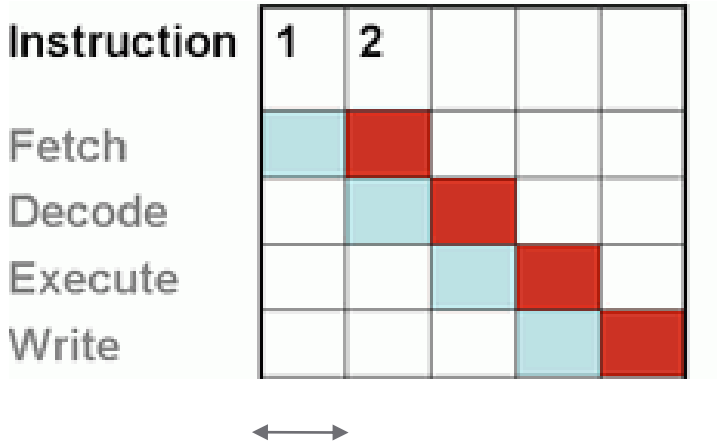
# Pipelining

- What are the instruction stages:
  - Fetch
  - Decode (+Read registers)
  - Execute (+ calculate an address)
  - Read memory
  - Write result into a register
- In single cycle: instructions must wait for their turn
- Pipelining: eliminate the wait and overlap instructions!
- Single-Cycle versus Pipelined Performance?

Tutorial!




All stages time = Clock Cycle




Stage time = Clock Cycle

SOUBRA© 2022

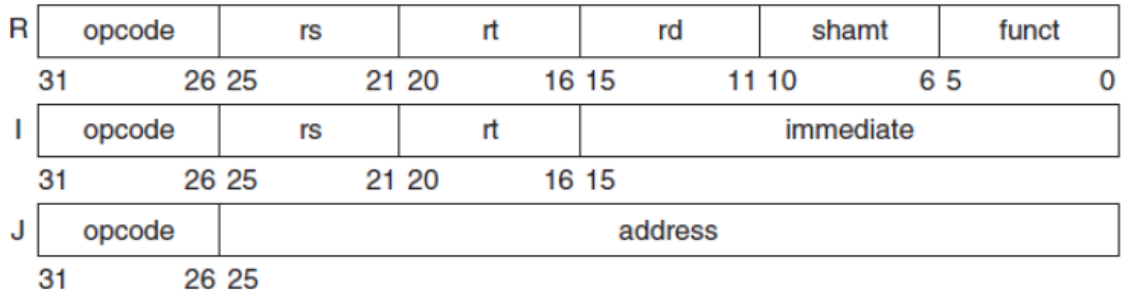
# Designing ISA for Pipelining

What makes fetch and decode easier?

1. Fixed-sized instructions.
2. Instruction format where source register is in the same place in each instruction

What makes execute and memory read easy?

1. Memory operands only appear in loads or stores
2. Memory alignment

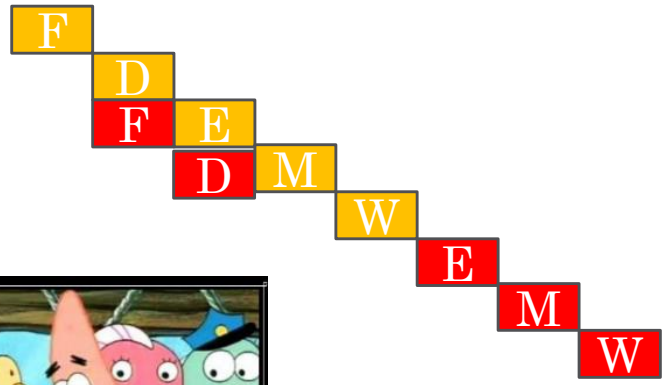


# Limitations of pipelining

**Data Hazards**

- Consider the following code, and try to pipeline it:

```
sub $s1, $t1, $t2
add $t0, $s1, $s0
```




**Control Hazards**

- Consider the following code, and try to pipeline it:

```
sub $s0, $t1, $t2
bne $s3, $s4, else
addi $t0, $zero, 1
…
else: addi $s1, $zero, 2
…
```

Next instruction?!?

**Structure Hazards**

- What if instruction fetching and data reading were not ‘separated’?

SOUBRA© 2022

# How to deal with pipelining Hazards

- Control Hazards: a.k.a Branch hazard:
  - Stall? =>Delayed decision!
  - Prediction?

- Data hazards:
  - Reorder instructions (hazards removed by the compiler)
  - Forwarding/bypassing

SOUBRA© 2022

# Research

- How much technology has influenced time spent in the different components?

SOUBRA© 2022