Computer Architecture (Lab). Week 12

Muhammad, Munir, Vladislav, Alena, Hamza, Manuel

Innopolis University

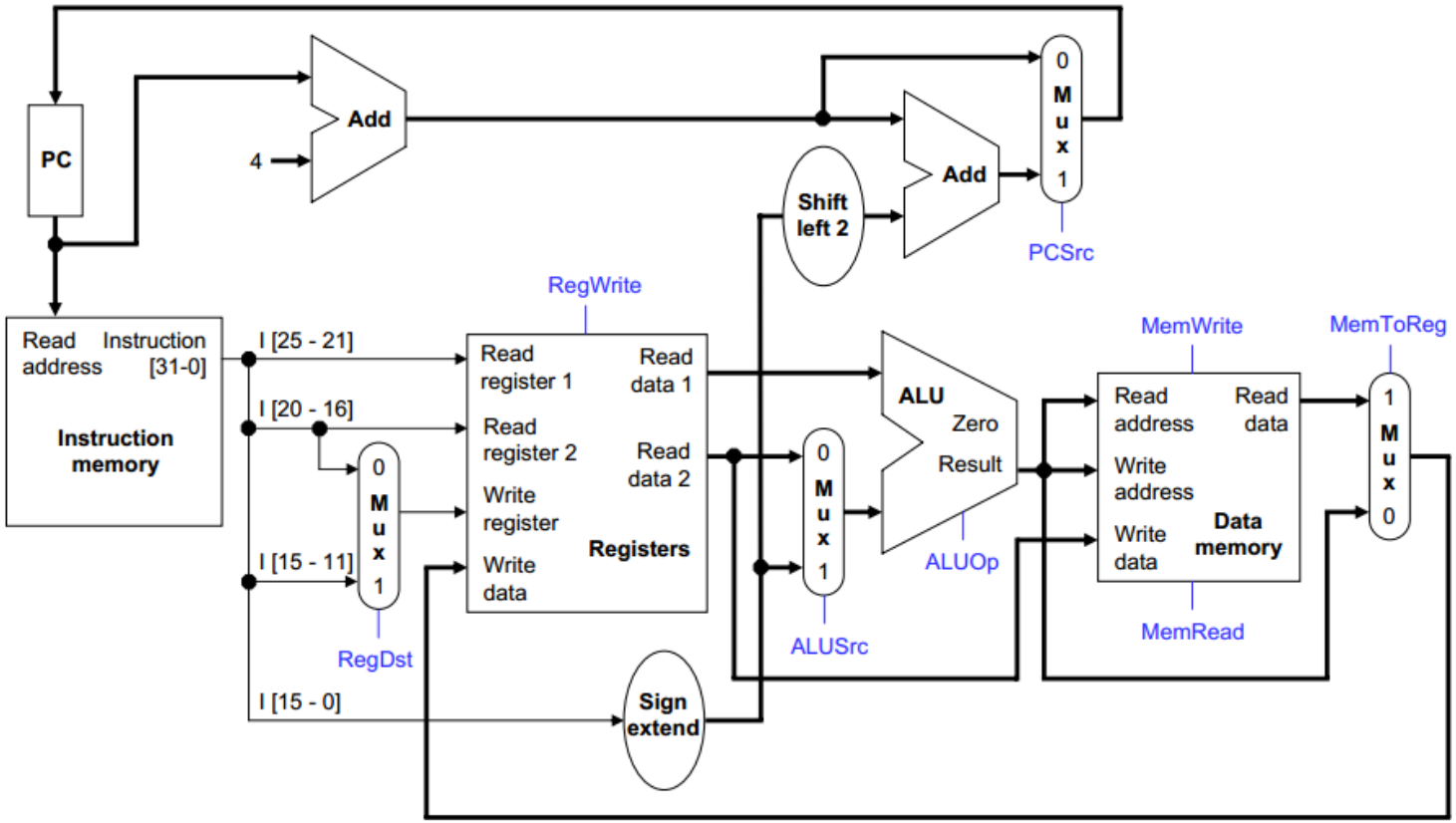
m.fahim@innopolis.ru
m.makhmutov@innopolis.ru
v.ostankovich@innopolis.ru
a.yuryeva@innopolis.ru
h.salem@innopolis.university
m.rodriguez.osuna@innopolis.university

November 19, 2020

Topic of the Lab

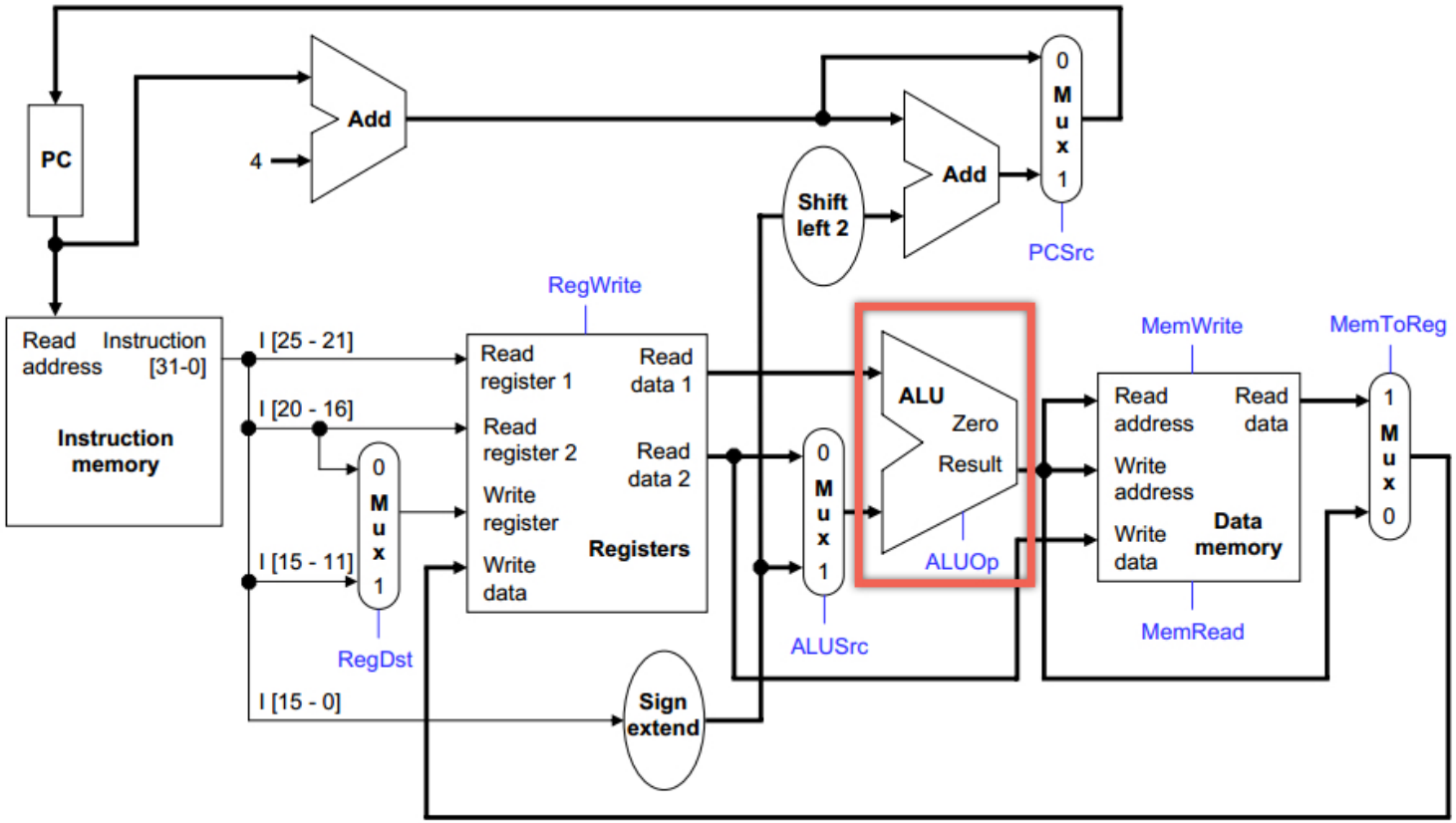
- Delving further into a single-cycle processor
- Implementing simple ALU

Single-Cycle Processor (1/2)



Overview of a single-cycle processor architecture. Can you explain the purpose of these blocks?

Single-Cycle Processor (2/2)



Focus on ALU for now. How can we implement MIPS compatible ALU in Verilog?

ALU Operations (1/2)

In MIPS, ALU is accessed during execution of R-type operations

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Opcode for ALU operations is `0x00`. The meaning of an operation is specified by `funct` field. R-type operations include:

- `add`
- `sub`
- `srl`
- `xor`
- `jr`
- `...`

ALU Operations (2/2)

In MIPS, ALU is accessed during execution of R-type operations

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

ALU operations have the same `op`, but different `funct`:

- `add $rd,$rs,$rt`

000000	xxxxx	xxxxx	xxxxx	00000	100000

- `sub $rd,$rs,$rt`

000000	xxxxx	xxxxx	xxxxx	00000	100010

- `srl $rd,$rt,3`

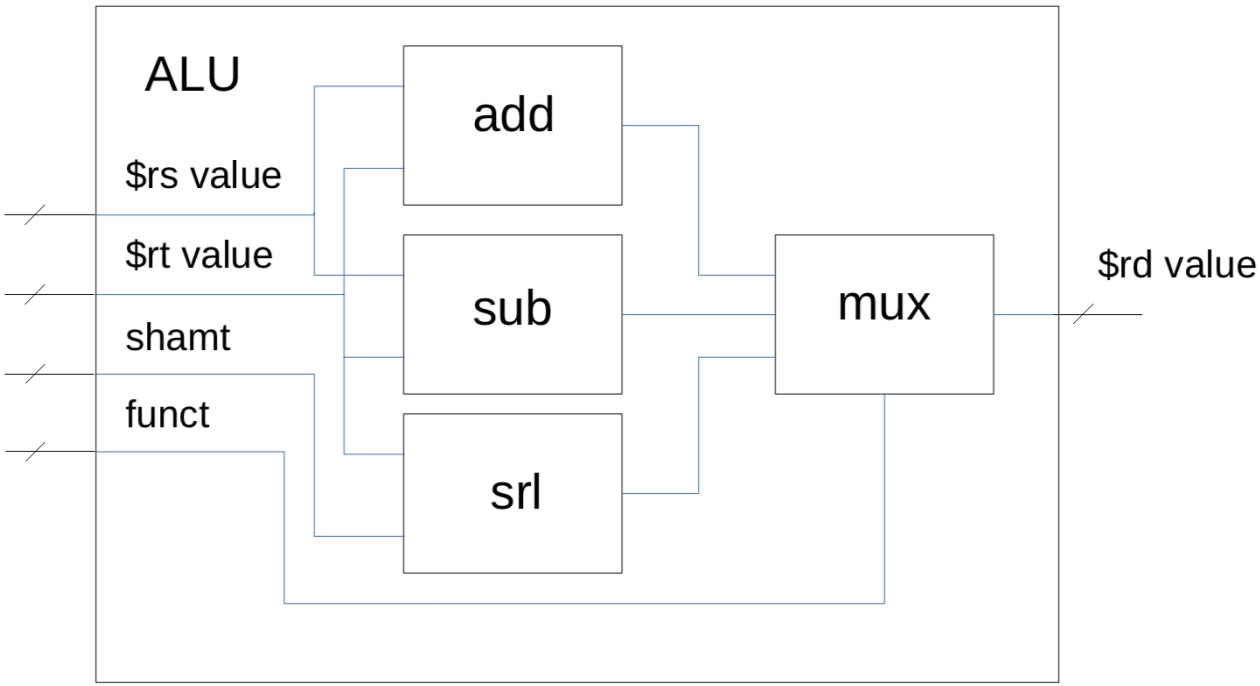
000000	xxxxx	xxxxx	xxxxx	00011	000010

ALU Data Flow (1/2)

Suppose ALU that supports three operations: Addition, Subtraction, and Shift Right Logical. To build this ALU we need:

- Input data lines for the values of `$rs` and `$rt`
- Input line `funct` for choosing function type
- Input line for specifying `shamt` (Shift amount)
- Addition module
- Subtraction module
- Shift module
- Multiplexer that decides what goes to the output

ALU Data Flow (2/2)



In the simplest implementation, all ALU functions are always calculated, but only one result is routed to the output.

MUX implementation

MUX can be implemented using `case` operator in Verilog. Important properties of `case`:

- Assign operation can be performed only on wires
- `case` can work only within `always@(...)` block

```
always@(*) begin
    case (register)
        value1: begin
            ...
            end
        value2: begin
            ...
            end
        default: begin
            ...
            end
    endcase
end
```

Exercise

Implement an ALU that supports three operations: Addition, Subtraction, and Shift Right Logical. The inputs of the ALU are:

- `$rs` address (5-bit)
- `$rt` address (5-bit)
- `$rd` address (5-bit)
- `shamt` value (5-bit)
- `funct` value (6-bit)

Hint: Use `funct` codes from the slides above.
You are provided with a useful testbench module and expected output file.

Exercise (cont.)

You need to implement following modules:

- `glob`: Module containing 32 32-bits MIPS registers. You should initialize array of 32 32-bits registers with zero values.
- `add`: Module for addition
- `sub`: Module for subtraction
- `srl`: Module for Shift Right Logical operation
- `mux`: Multiplexer module
- `alu`: ALU module

Acknowledgements

- This lab was created and maintained by Vitaly Romanov, Aidar Gabdullin, Munir Makhmutov, Ruzilya Mirgalimova, Muhammad Fahim, Vladislav Ostankovich, Alena Yuryeva and Artem Burmyakov