Lab Session 5
ALU and Register Files

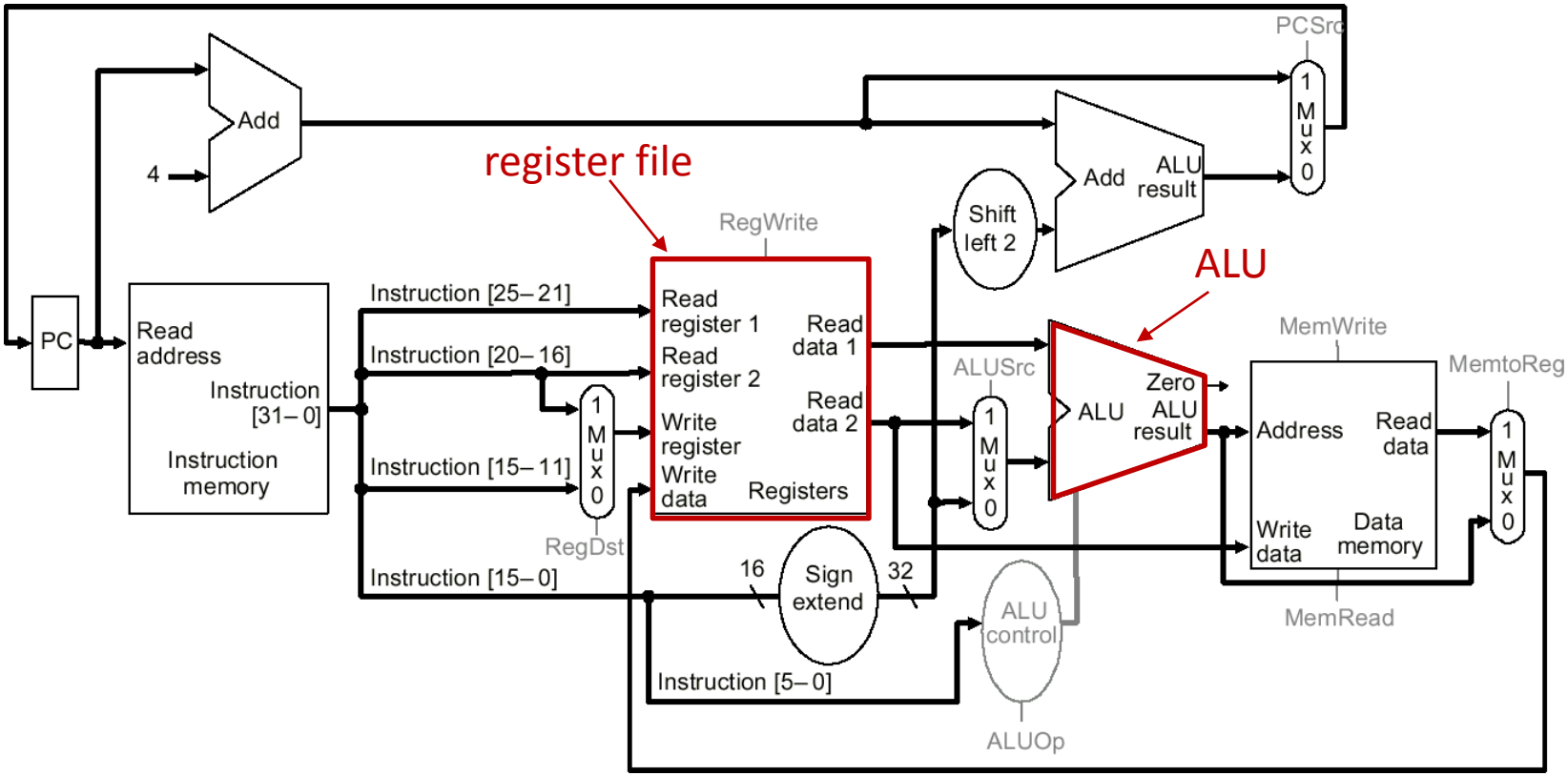
Advisor: Lih-Yih Chiou
Speaker: Patty
Date: 03/23/2016

Outline

- Introduction
- Lab A : Arithmetic Logic Unit
- Lab B : Register File
- Lab C : Serial-In Parallel-Out Register File
- Lab D : Register File + ALU
- Lab5 Homework
- Copy Reference code

Introduction

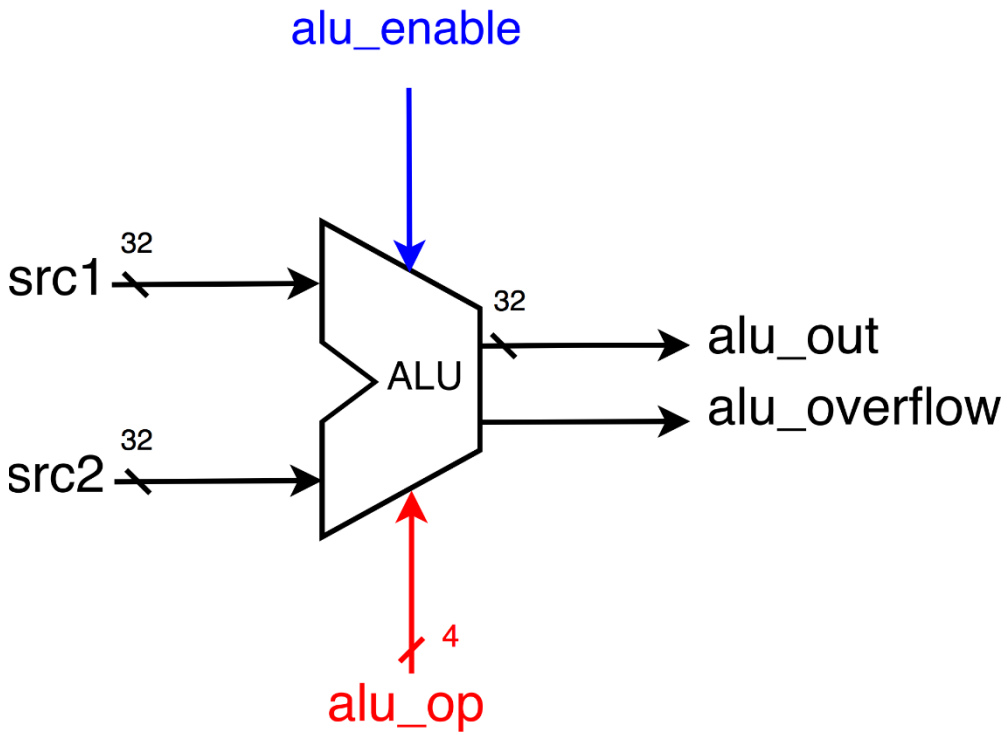
Single-Cycle CPU



Computer Organization and Design. THE HARDWARE/SOFTWARE INTERFACE. David A. Patterson, John L. Hennessy

Lab A : Arithmetic Logic Unit

- A digital electronic circuit that performs arithmetic and bitwise logical operations on integer binary numbers
- A fundamental building block of the CPU.



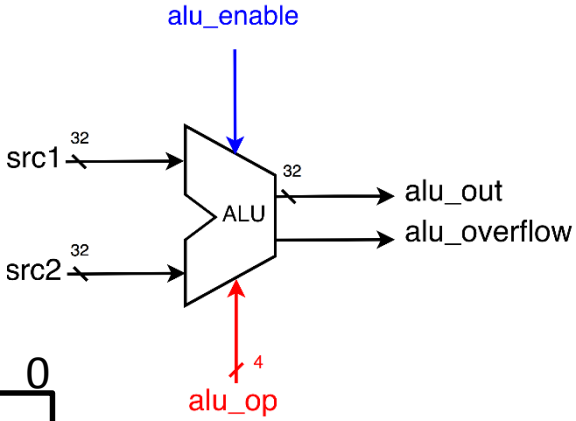
Lab A : Arithmetic Logic Unit

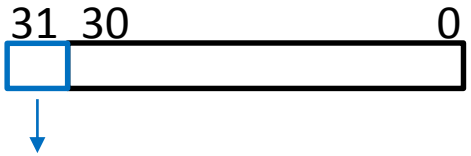
Signal	Type	Bits	Description
alu_enable	input	1	0→close 1→open
alu_op	input	4	Operation code select which operation to be executed
src1	input	32	ALU source 1
src2	input	32	ALU source 2
alu_out	output	32	ALU result
alu_overflow	output	32	0→no overflow 1→overflow

NOTE :

Definition of overflow : the result is greater than which a given register can store or represent.

ADD pos + pos = neg
ADD neg + neg = pos
SUB pos - neg = neg
SUB neg - pos = pos





sign bit (0 -> pos, 1 -> neg)

Lab A : Arithmetic Logic Unit

Category	alu_op	Operation	Description
Arithmetic	0000	ADD	alu_out = src1 + src2
	0001	SUB	alu_out = src1 - src2
Logical	0010	AND	alu_out = src1 & src2
	0011	OR	alu_out = src1 | src2
	0100	XOR	alu_out = src1 ^ src2
	0101	NOR	alu_out = ~(src1 | src2)
Barrel shifter	0110	SRL	alu_out = src1 >> src2
	0111	ROTR	alu_out = src1 rotate right by “src2 bits”

NOTE :

Difference between shift and rotate :

SRL 8’b0000_1111 >> 8’b0000_0011 = 8’b0000_0001

ROTR 8’b0000_1111 rotate right 8’b0000_0011 = 8’b1110_0001

Lab A : Reference Code (1/2)

```
1  `timescale 1ns/10ps
2
3  // ---------------------- define ---------------------- //
4
5  `define DataSize     32
6  `define ALUopSize    4
7  //define ALUop
8  `define ADDop        4'b0000
9  `define SUBop        4'b0001
10 `define ANDop        4'b0010
11 `define ORop         4'b0011
12 `define XORop        4'b0100
13 `define NORop        4'b0101
14 `define SRLop        4'b0110
15 `define ROTRop       4'b0111
16
17 module ALU (alu_enable, alu_op, src1, src2, alu_out, alu_overflow);
18
19 // ---------------------- input  ---------------------- //
20 input                  alu_enable;
21 input  [`ALUopSize-1:0] alu_op;
22 input  [`DataSize-1:0]  src1;
23 input  [`DataSize-1:0]  src2;
24
25 // --------------------- output ---------------------- //
26 output [`DataSize-1:0]  alu_out;
27 output                 alu_overflow;
28
29 // ---------------------  reg   ---------------------- //
30 reg    [`DataSize-1:0]  alu_out;
31 reg                    alu_overflow;
32 reg    [63:0]           temp;
```

Increase modifiability and readability

Lab A : Reference Code (1/2)

If there is any change on the **RHS** of "=", the always block will be executed

readability

```
33
34  always@(*)begin
35      alu_overflow = 1'b0;
36      if(alu_enable)begin
37          case(alu_op)
38              `ADDop  :   begin
39                              alu_out = src1 + src2;
40                              if((src1[31]==0 && src2[31]==0 && alu_out[31]==1)||
41                                 (src1[31]==1 && src2[31]==1 && alu_out[31]==0))
42                                  alu_overflow = 1'b1;
43                              else
44                                  alu_overflow = 1'b0;
45                              end
46              `SUBop  :   begin
47                              alu_out = src1 - src2;
48                              if((src1[31]==0 && src2[31]==1 && alu_out[31]==1)||
49                                 (src1[31]==1 && src2[31]==0 && alu_out[31]==0))
50                                  alu_overflow = 1'b1;
51                              else
52                                  alu_overflow = 1'b0;
53                              end
54              `ANDop  :   alu_out = src1 & src2;
55              `ORop   :   alu_out = src1 | src2;
56              `XORop  :   alu_out = src1 ^ src2;
57              `NORop  :   alu_out = ~(src1 | src2);
58              `SRLop  :   alu_out = src1 >> src2;
59              `ROTRop :   begin
60                              temp = {src1,src1};
61                              temp = temp >> src2;
62                              alu_out = temp[31:0];
63                              end
64              default :   alu_out = 32'b0;
65          endcase
66      end
67      else
68          alu_out = 32'b0;
69      end
70  endmodule
```

case

full case

Lab B : Register File

D Latch

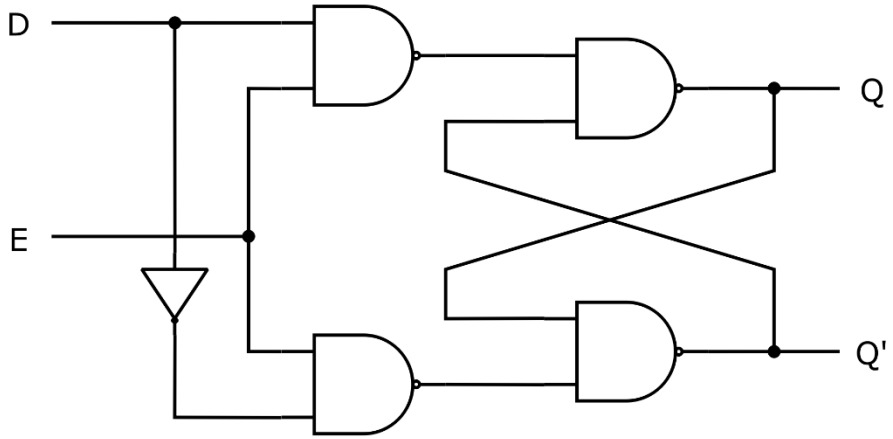


Fig. D latch with enable

E	D	Q	Q_{next}	Q'_{next}
0	x	0	0	1
0	x	1	1	0
1	0	x	0	1
1	1	x	1	0

Lab B : Register File

D flip-flop

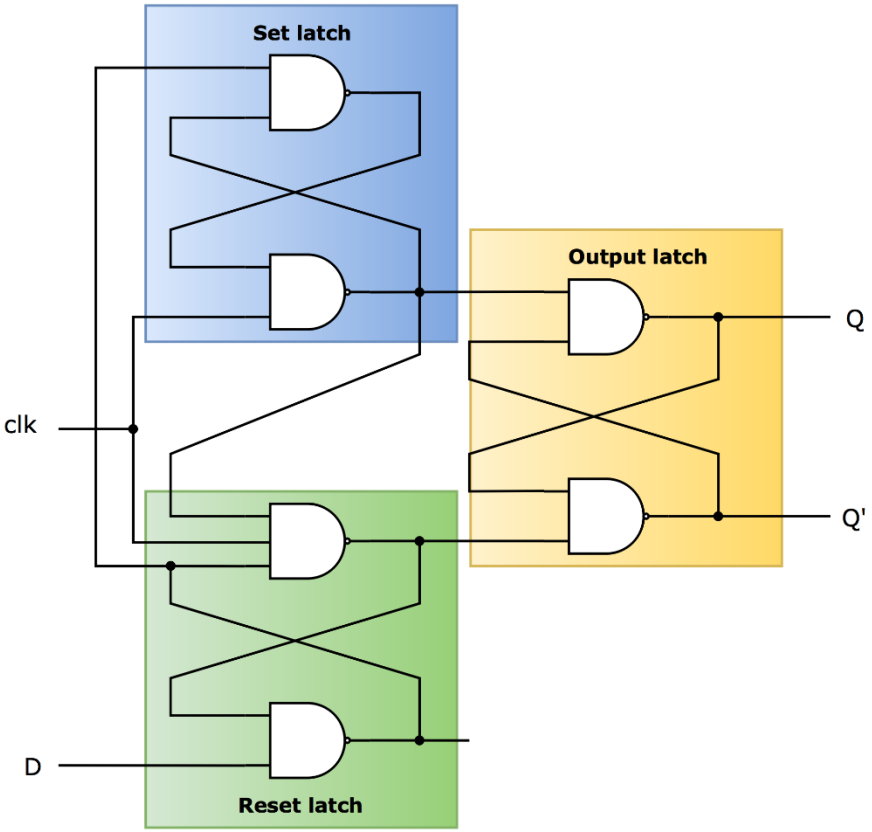
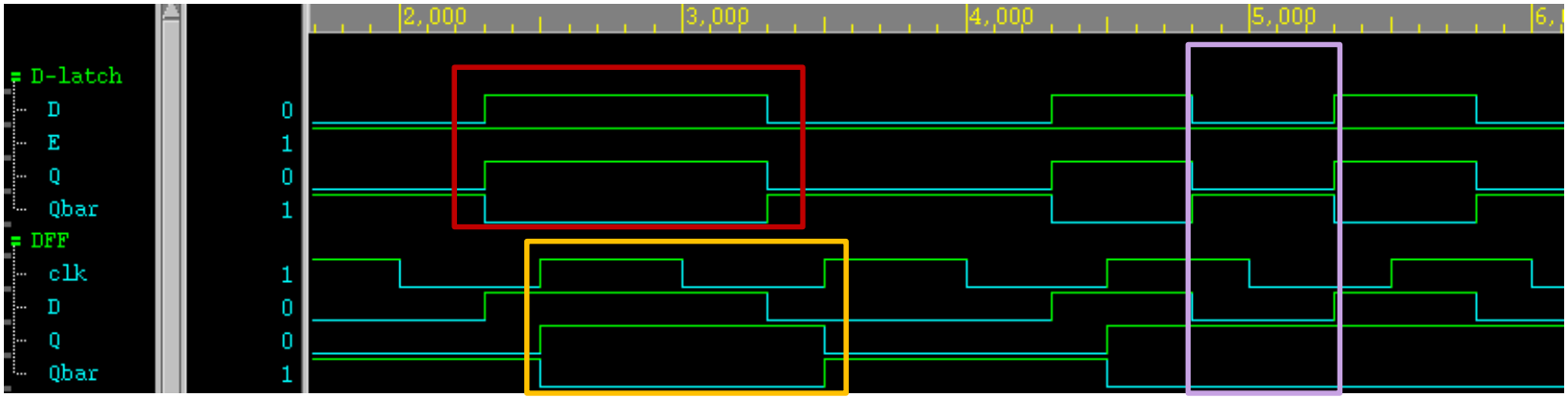


Fig. Positive-edge-triggered D flip-fliop

clk	D	Q	Q_{next}	Q'_{next}
0	x	0	0	1
0	x	1	1	0
1	x	0	0	1
1	x	1	1	0
↑	0	x	0	1
↑	1	x	1	0

Lab B : Register File

- The main difference between latch and flip-flop
  - Latch
    - Outputs are constantly affected by the inputs as long as the enable signal is asserted.
  - Flip-Flop
    - Outputs change only at the rising edge of the clock signal.



Lab B : Register File

- **Register file is an array of processor registers in a CPU.**
  - A 32 X 32 register file means it has 32 registers and each of them is 32 bits.
- **Components**
  - n to 1 Decoder
  - Array
  - m to n Multiplexer
- **Multiple output ports allow us to read several data in one cycle.**

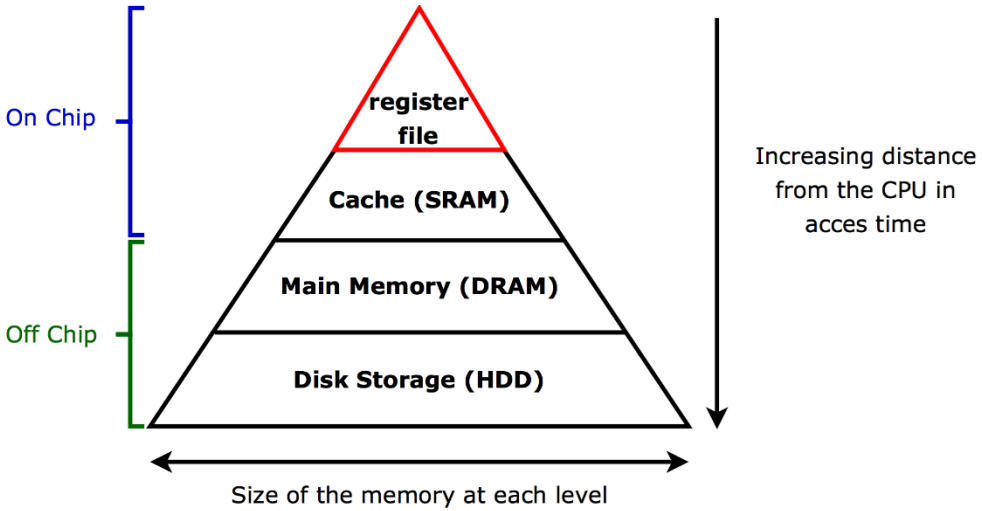


Fig. Memory Hierarchy

Lab B : Register File

How does it work ?

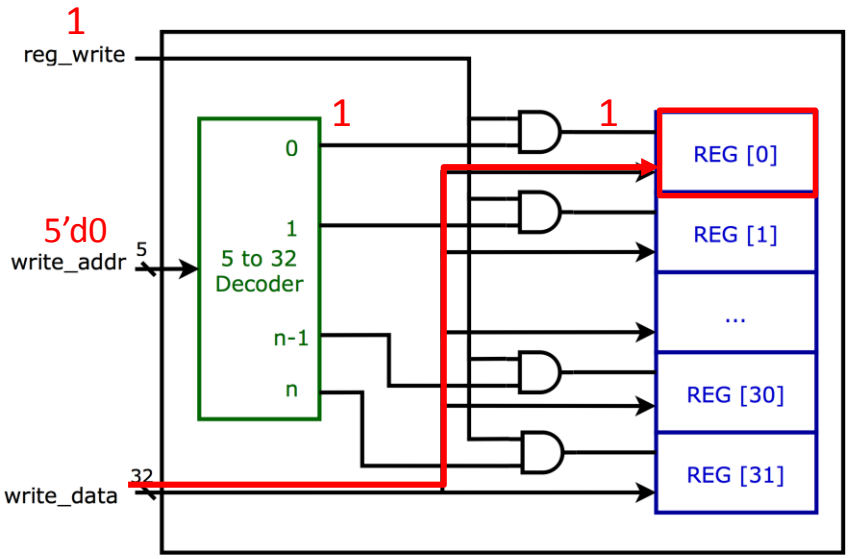


Fig. Write data into register file

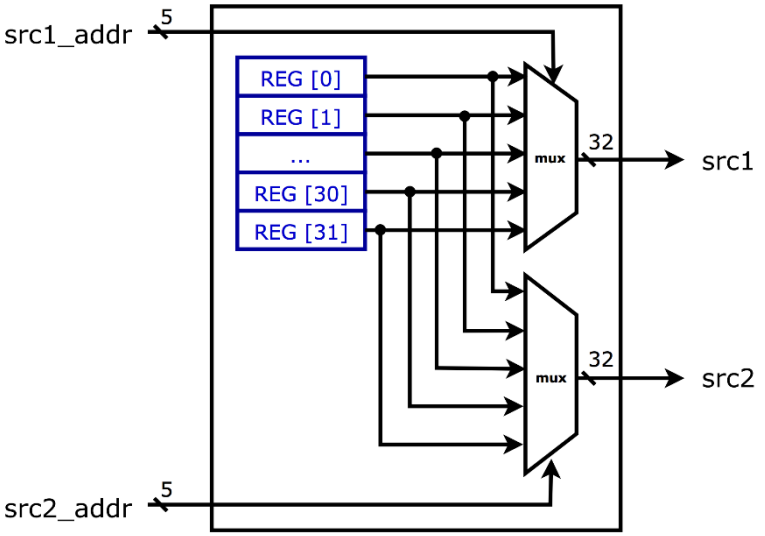
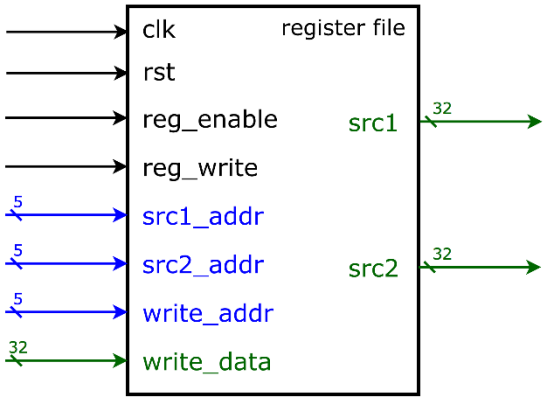


Fig. Read data from register file

Lab B : Register File

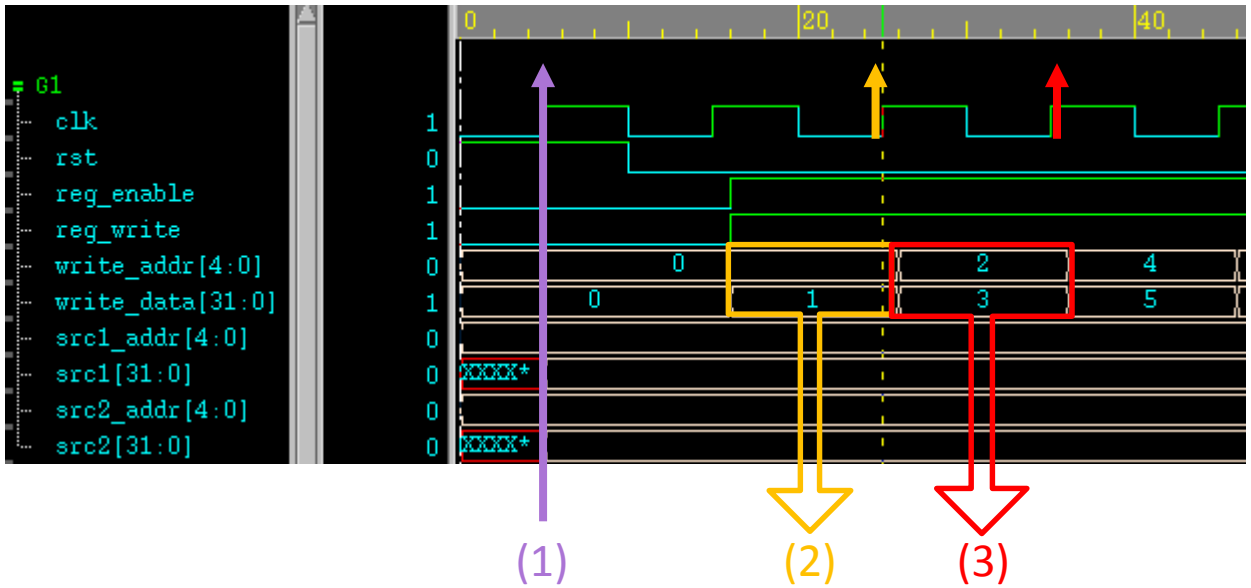
Port List

Signal	Type	Bits	Description
clk	input	1	clock
rst	input	1	reset
reg_enable	input	1	0 → off 1 → on
reg_write	input	1	0 → read 1 → write
src1_addr	input	5	source1 address
src2_addr	input	5	source2 address
write_addr	input	5	write address
write_data	input	32	write data
src1	output	32	read data source1
src2	output	32	read data source2



Lab B : Register File

How does it write?



(1) clock positive edge↑, reset signal is set to 1, all registers are reset to zero

(2) clock positive edge↑, register file is enable and in write mode

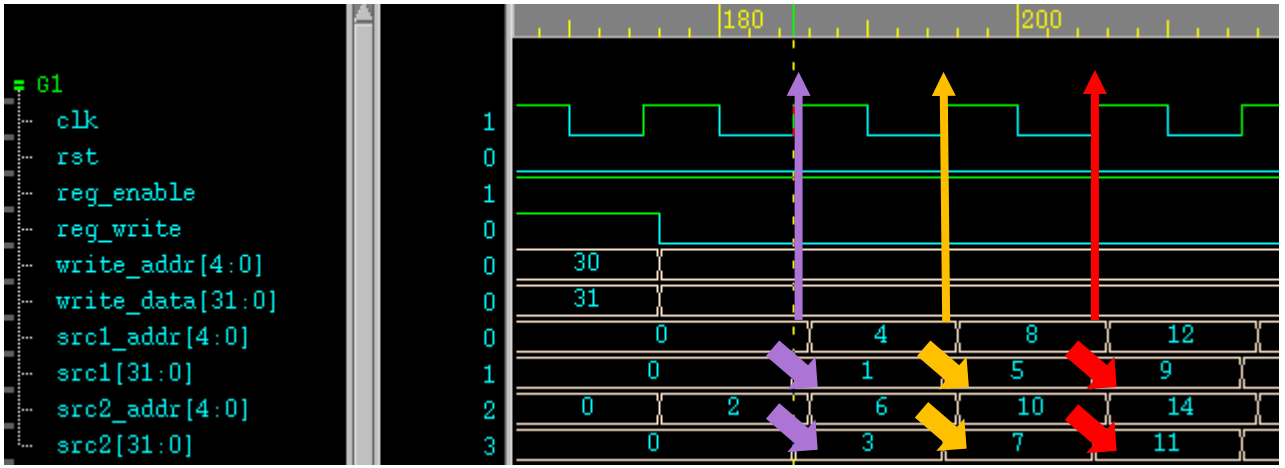
→ 32'd1 is written into REG[0]

(3) clock positive edge↑, register file is enable and in write mode

→ 32'd3 is written into REG[2]

Lab B : Register File

How does it read?



(1) clock positive edge↑, register file is enable and in read mode

↑ → src1 = REG [0] = 32'd1, src2 = REG [2] = 32'd3

(2) clock positive edge↑, register file is enable and in read mode

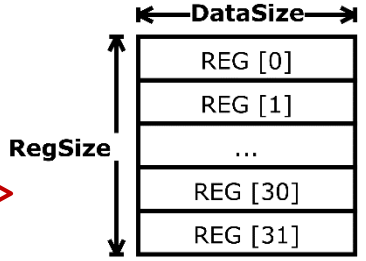
↑ → src1 = REG [4] = 32'd5, src2 = REG [6] = 32'd7

(3) clock positive edge↑, register file is enable and in read mode

↑ → src1 = REG [8] = 32'd9, src2 = REG [10] = 32'11

Lab B : Reference Code (1/2)

```
1   `timescale 1ns/10ps
2
3   // ---------------------- define ---------------------- //
4   `define DataSize    32
5   `define RegSize     32
6   `define AddrSize    5
7
8   module regfile (clk, rst, reg_enable, reg_write, src1_addr, src2_addr,
9                   write_addr, write_data, src1, src2);
10
11  // ---------------------- input  ---------------------- //
12  input                       clk;
13  input                       rst;
14  input                       reg_enable;
15  input                       reg_write;
16  input   [`AddrSize-1:0]     src1_addr;
17  input   [`AddrSize-1:0]     src2_addr;
18  input   [`AddrSize-1:0]     write_addr;
19  input   [`DataSize-1:0]     write_data;
20
21  // ---------------------- output ---------------------- //
22  output  [`DataSize-1:0]     src1;
23  output  [`DataSize-1:0]     src2;
24
25  // ----------------------  reg   ---------------------- //
26  reg     [`DataSize-1:0]     src1;
27  reg     [`DataSize-1:0]     src2;
28  reg     [`DataSize-1:0]     REG [`RegSize-1:0];
29
30  integer i;
```



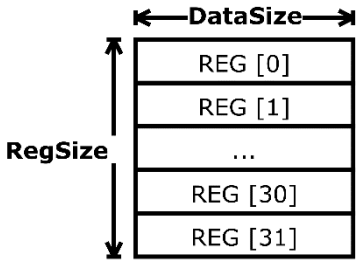
Lab B : Reference Code (2/2)

```
31
32  always@(posedge clk)begin
33      if(rst)begin
34          for(i=0; i<`RegSize; i=i+1)
35              REG[i] <= 32'b0;
36          src1 <= 32'b0;
37          src2 <= 32'b0;
38          end
39      else begin
40          if(reg_enable)begin
41              if(reg_write)
42                  REG[write_addr] <= write_data;
43              else begin
44                  src1 <= REG[src1_addr];
45                  src2 <= REG[src2_addr];
46                  end
47              end
48          else begin
49                  src1 <= 32'b0;
50                  src2 <= 32'b0;
51                  end
52          end
53      end
54  endmodule
```

Using for loop to reset the register array

register write

register read



Lab C : Serial-In Parallel-Out Register File

- A special design of register file which use only one address to get more than one data in a register file simultaneously.
- Take a serial-in parallel-out 32 X 32 register file as an example:
  - ➔ It only has one read address port but three output ports.

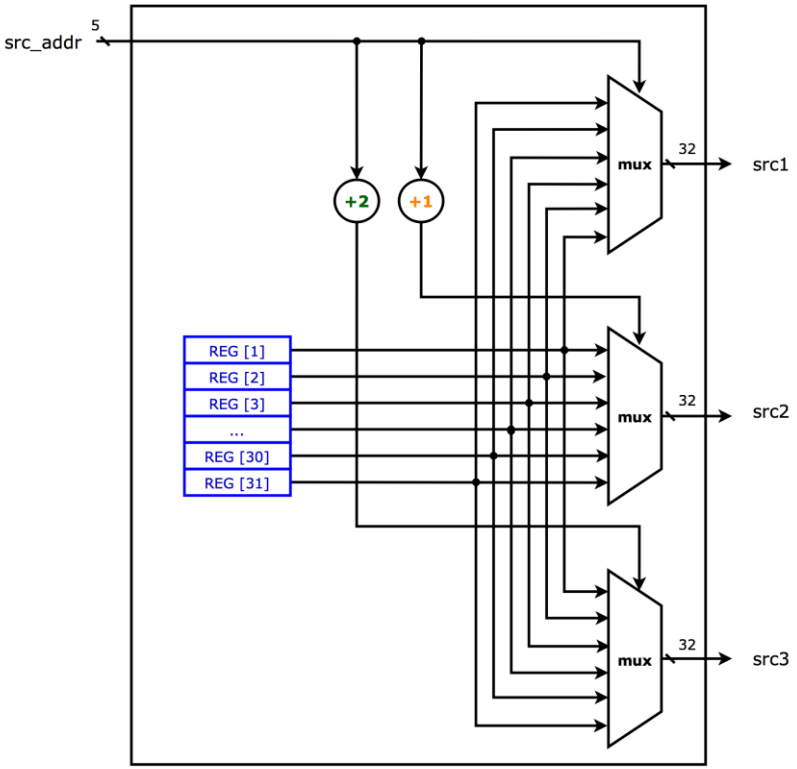
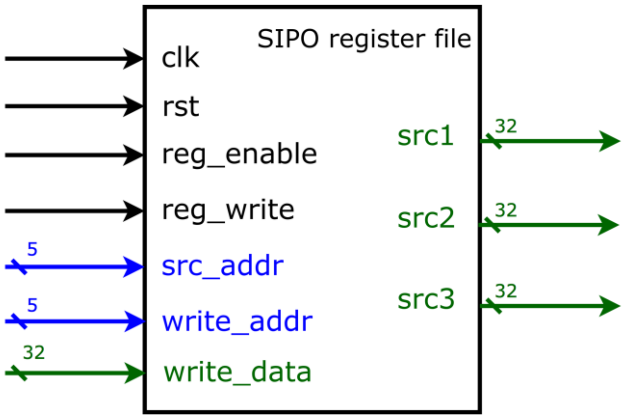


Fig. Read data from SIPO register file

Lab C : Serial-In Parallel-Out Register File

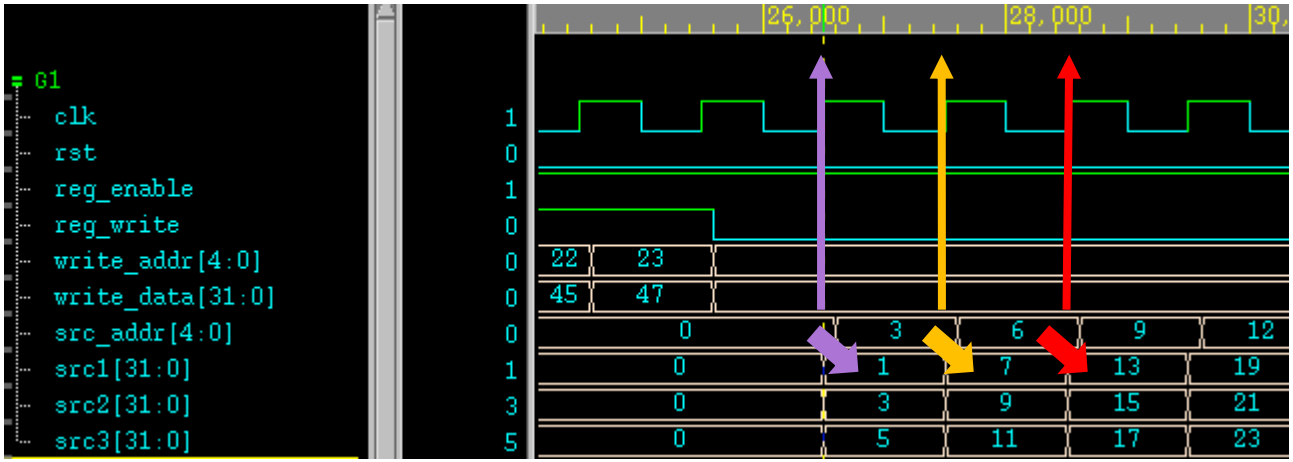
Port List

Signal	Type	Bits	Description
clk	input	1	clock
rst	input	1	reset
reg_enable	input	1	register file enable
reg_write	input	1	0 → read 1 → write
src_addr	input	5	source address
write_addr	input	5	write address
write_data	input	32	write data
src1	output	32	read data source1
src2	output	32	read data source2
src3	output	32	read data source3



Lab C : Serial-In Parallel-Out Register File

How does it read?



(1) clock positive edge ↑, register file is enable and in read mode

↑ → src1 = REG [0] = 32'd1, src2 = REG [0+1] = 32'd3, src3 = REG [0+2] = 32'd5

(2) clock positive edge ↑, register file is enable and in read mode

↑ → src1 = REG [3] = 32'd7, src2 = REG [3+1] = 32'd9, src3 = REG [3+2] = 32'd11

(3) clock positive edge ↑, register file is enable and in read mode

↑ → src1 = REG [6] = 32'd13, src2 = REG [6+1] = 32'd15, src3 = REG [6+2] = 32'd17

Lab C : Reference Code (1/2)

```
1   `timescale 1ns/10ps
2
3   // ---------------------- define ---------------------- //
4   `define DataSize    32
5   `define RegSize     32
6   `define AddrSize    5
7
8   module regfile_sipo (clk, rst, reg_enable, reg_write, src_addr, write_addr,
9                        write_data, src1, src2, src3);
10
11  // ---------------------- input  ---------------------- //
12  input                       clk;
13  input                       rst;
14  input                       reg_enable;
15  input                       reg_write;
16  input   [`AddrSize-1:0]     src_addr;
17  input   [`AddrSize-1:0]     write_addr;
18  input   [`DataSize-1:0]     write_data;
19
20  // ---------------------- output ---------------------- //
21  output  [`DataSize-1:0]     src1;
22  output  [`DataSize-1:0]     src2;
23  output  [`DataSize-1:0]     src3;
24
25  // ----------------------  reg   ---------------------- //
26  reg     [`DataSize-1:0]     src1;
27  reg     [`DataSize-1:0]     src2;
28  reg     [`DataSize-1:0]     src3;
29  reg     [`DataSize-1:0]     REG [`RegSize-1:0];
30
31  integer i;
```

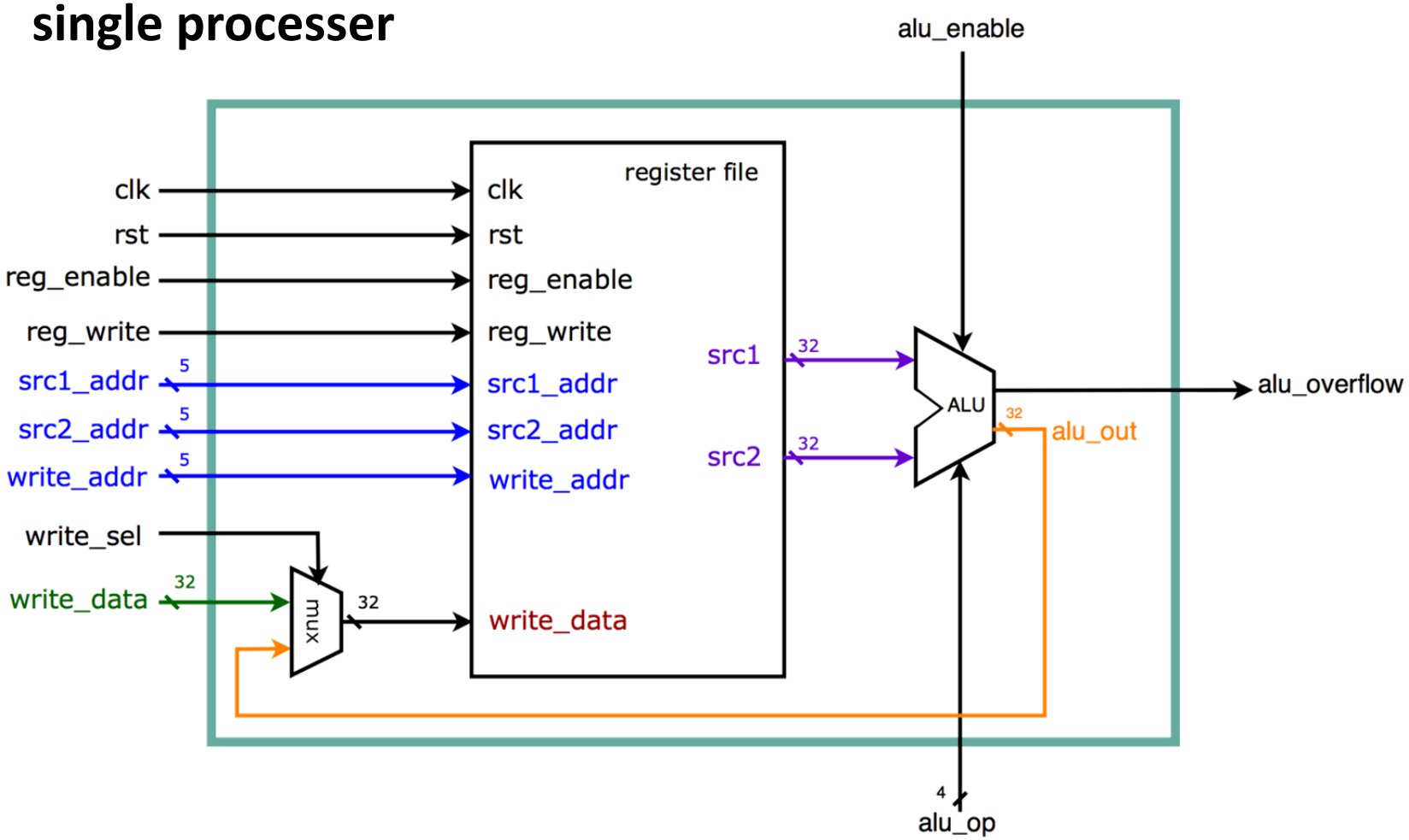
Lab C : Reference Code (1/3)

```
32
33  always@(posedge clk)begin
34      if(rst)begin
35          for(i=0; i<`RegSize; i=i+1)
36              REG[i] <= 32'b0;
37          src1 <= 32'b0;
38          src2 <= 32'b0;
39          src3 <= 32'b0;
40          end
41      else begin
42          if(reg_enable)begin
43              if(reg_write)
44                  REG[write_addr] <= write_data;
45              else begin
46                  src1 <= REG[src_addr];
47                  src2 <= REG[src_addr+1];
48                  src3 <= REG[src_addr+2];
49                  end
50              end
51          else begin
52                  src1 <= 32'b0;
53                  src2 <= 32'b0;
54                  src3 <= 32'b0;
55                  end
56          end
57      end
58  endmodule
```

register read

Lab D : Register File + ALU

- Combining a 32 x 32 Register File and an ALU to form a subsystem that could carry out the basic operation of a single processer



Lab D : Register File + ALU

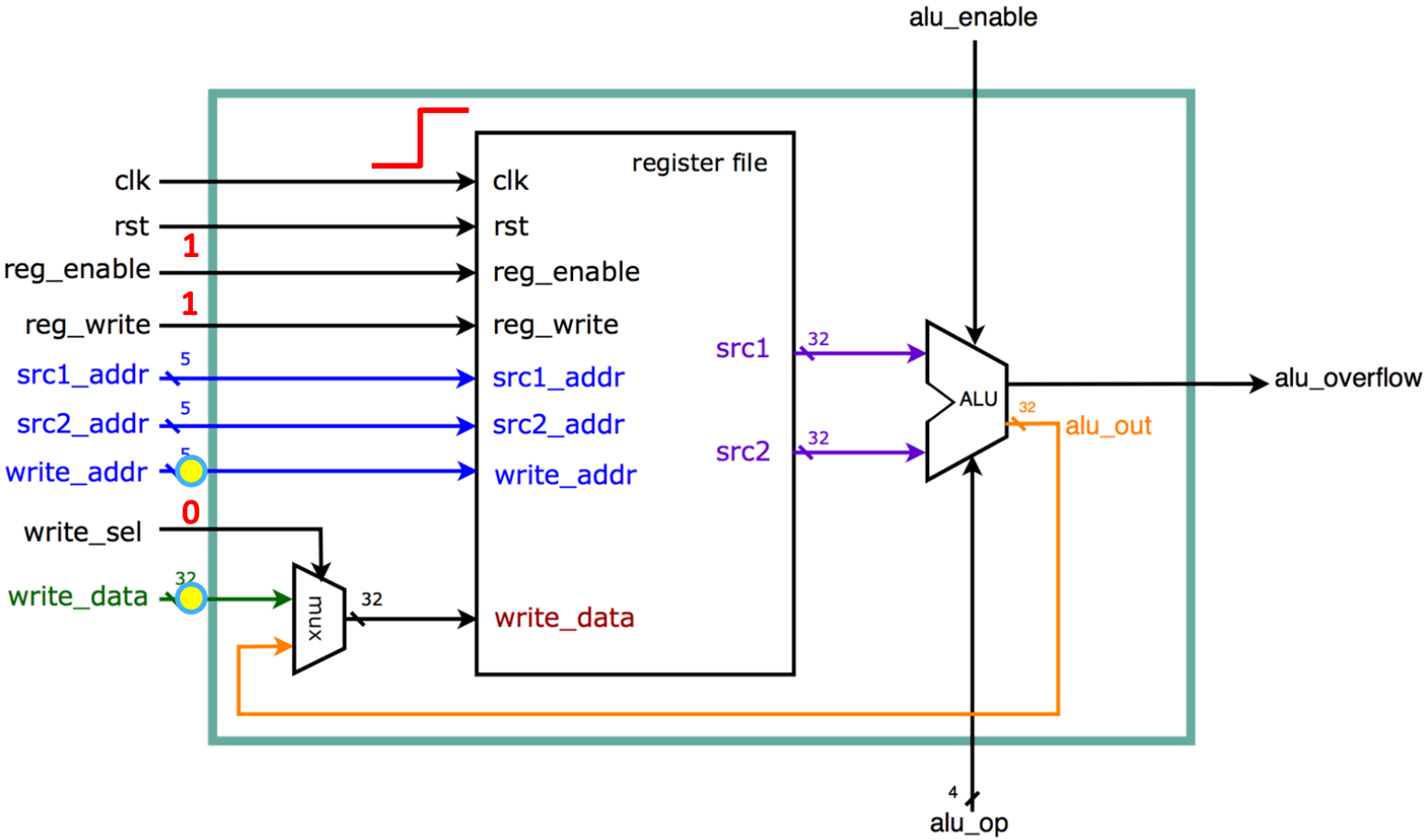
Port List

Signal	Type	Bits	Description
clk	input	1	clock
rst	input	1	reset
reg_enable	input	1	register file enable
reg_write	input	1	0 → read 1 → write
src1_addr	input	5	source1 address
src2_addr	input	5	source2 address
write_addr	input	5	write address
write_data	input	32	write data
alu_enable	input	1	0→close 1→open
alu_op	input	4	ALU operation code
write_sel	input	1	0→write_data 1→alu_out
alu_overflow	output	1	0→no overflow 1→overflow

Lab D : Register File + ALU

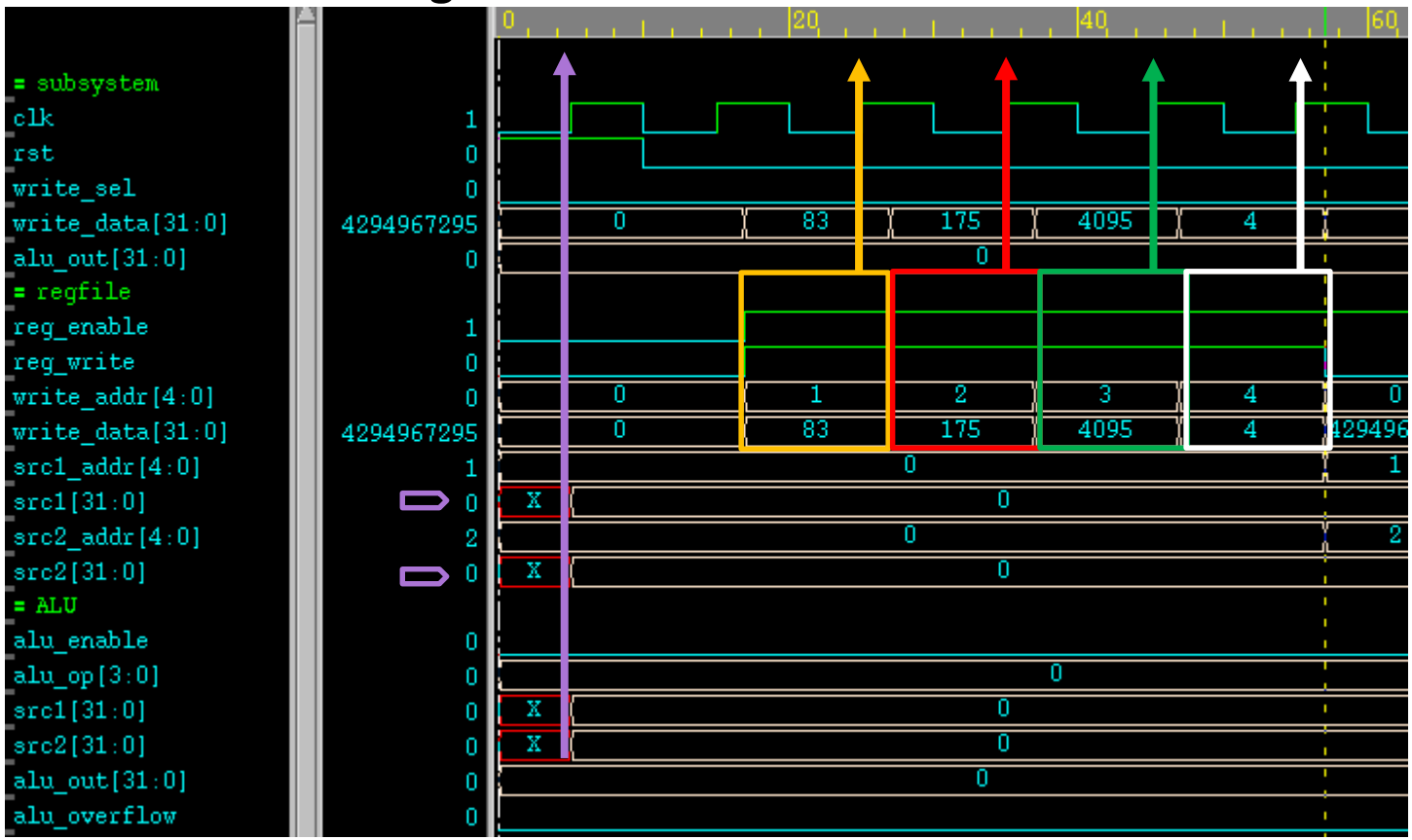
How does it work ?

➔ Write data into register file



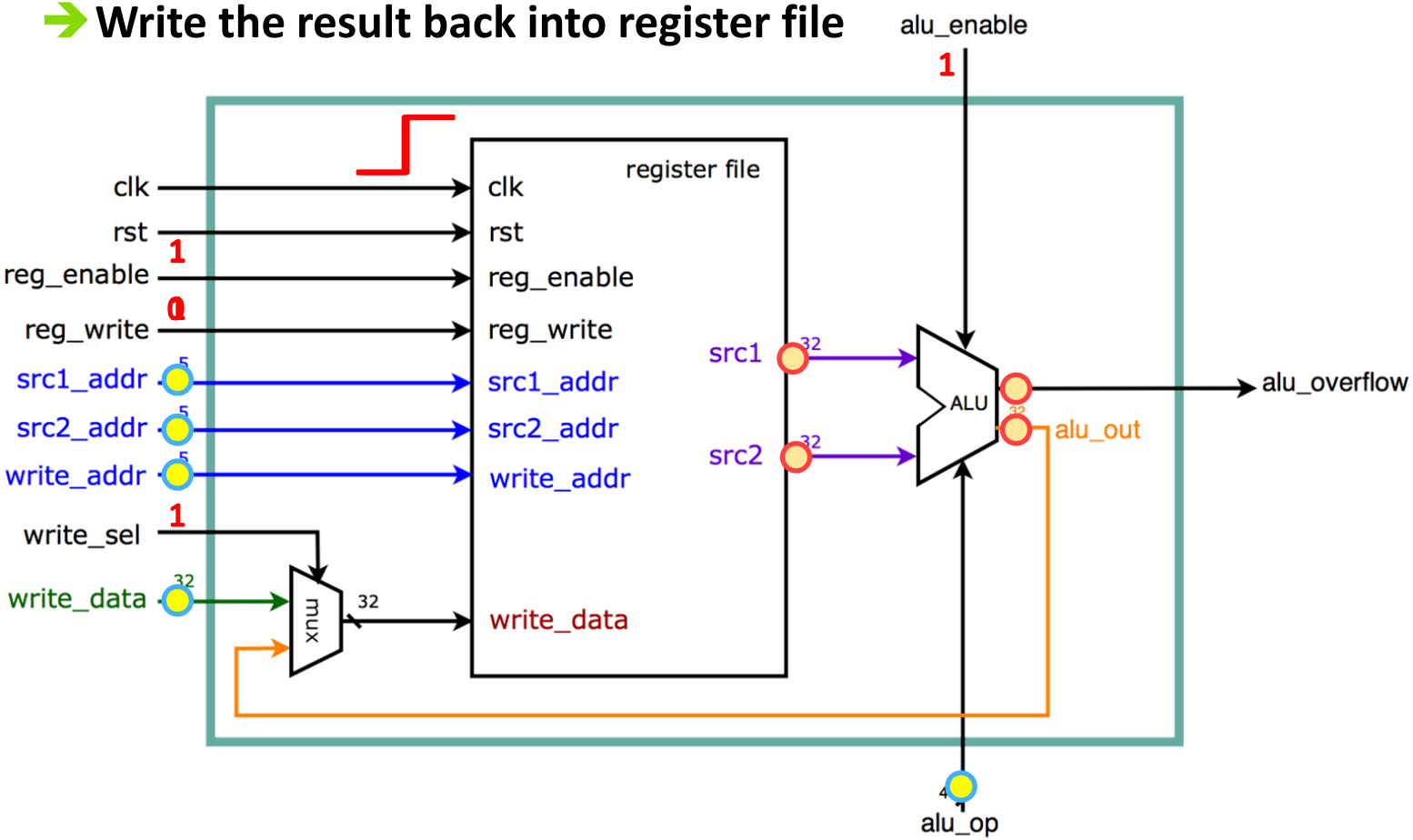
Lab D : Register File + ALU

- How does it work ?
  - ➔ Write data into register file



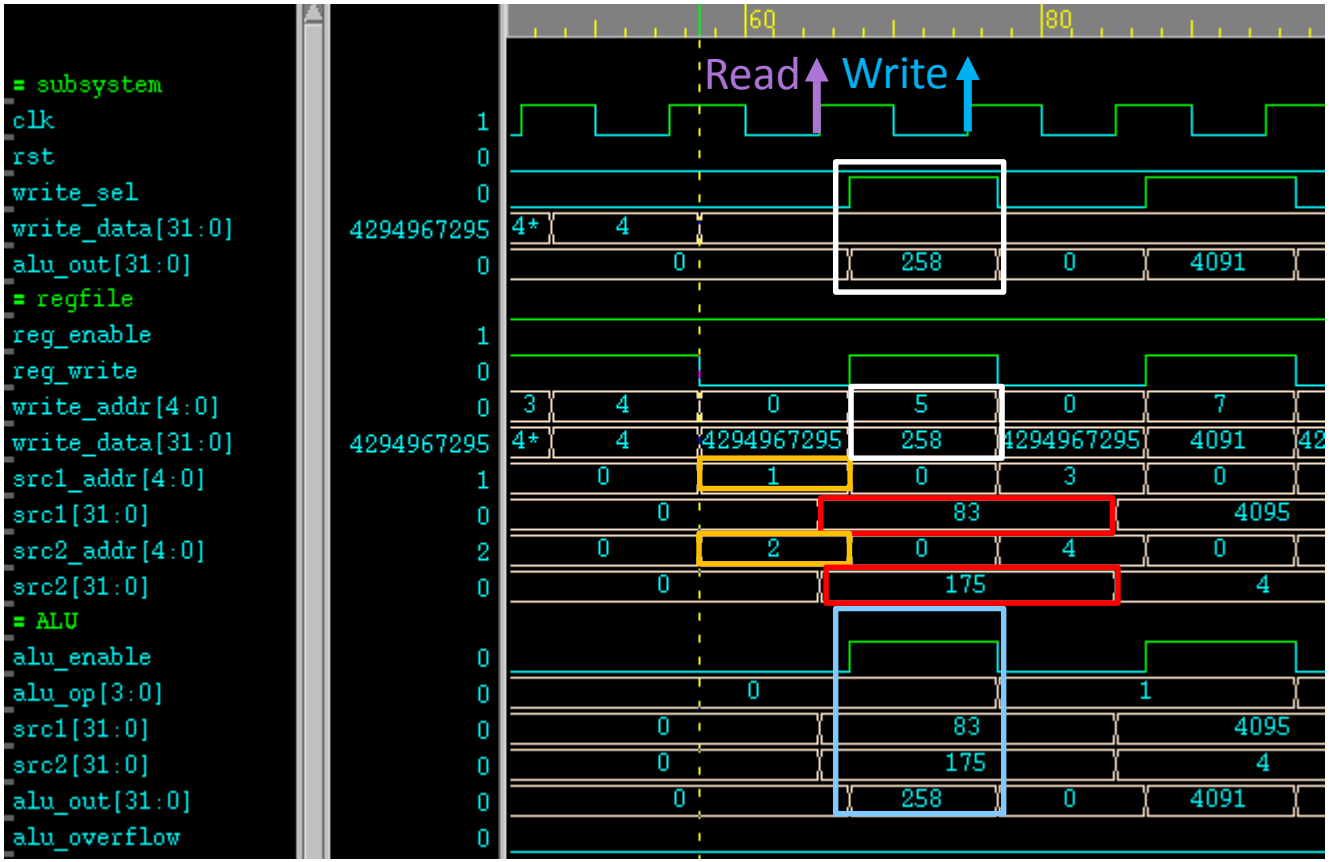
Lab D : Register File + ALU

- How does it work ?
  - Read data from register file and execute by ALU
  - Write the result back into register file



Lab D : Register File + ALU

- How does it work ?



Lab5 Homework

- **ProbA**
  - **Implement another 8 instructions based on the ALU in LabA.**
- **ProbB**
  - **Design a 64 X 32 register file based on LabB's structure.**
- **ProbC**
  - **Design a 128 X 32 SIPO-register file based on LabC's structure with 5 output ports.**
- **ProbD**
  - **Implement the subsystem in LabD using the components your designed in ProbA and ProbB.**

Attention :

1. Make sure all your verilog code can be compiled in SOC lab.
2. Behavioral descriptions are acceptable.

Copy Reference code

- Steps :
  - Open the terminal command-line interface.
  - Login the workstations. (ssh –X vlsicad5(9))
  - % cp -r /home/user2/vlsi16/vlsi1680/lab5 .

01~15:vlsicad5
16~30:vlsicad9

Point