Parallel and Reconfigurable VLSI Computing (8)

RTL Design: Tiny Processor

Hiroki Nakahara

Tokyo Institute of Technology

The lecture is based on a part of "CPUの創り方",
(in Japanese)

Outline

- Processor Specification
- Behavior Design of Tiny Processor
- RTL Design
- Example

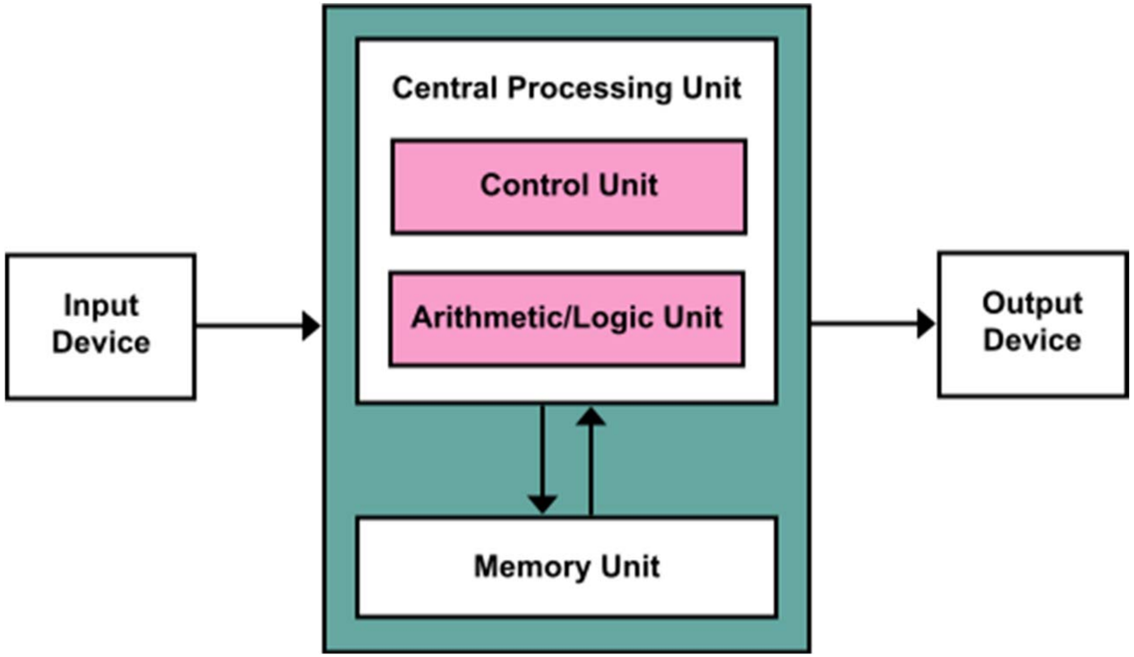
Processor Specification

Stored Program Computer

- Operate according to "program" stored in memory
  - Run various applications on the same hardware
- Its idea can be traced back to the 1936 theoretical concept of a universal Turing machine
- Von Neumann was aware of the paper, and he impressed it on his collaborators as well

von Neumann Architecture

- It also known as the von Neumann model and Princeton architecture
- Based on the 1945 description by the mathematician and physicist John von Neumann and others in the First Draft of a Report on the EDVAC

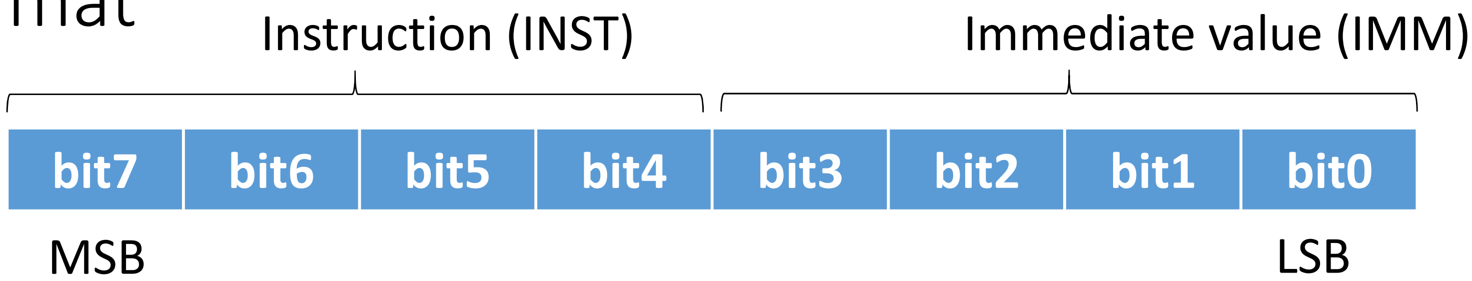


4bit Tiny Processor

- Operation and data: 4bit (8bit word length)
- 4bit PC (Program Counter)
  - Up to 16 steps (ROM Size) and reset to all zero "0000"
- Two registers A and B
- Carry flag (CF) only by addition
  - Used for a jump operation
- I/O: Only 4bit input and output
- Arithmetic operation: Only 4bit addition

Instruction

- Format



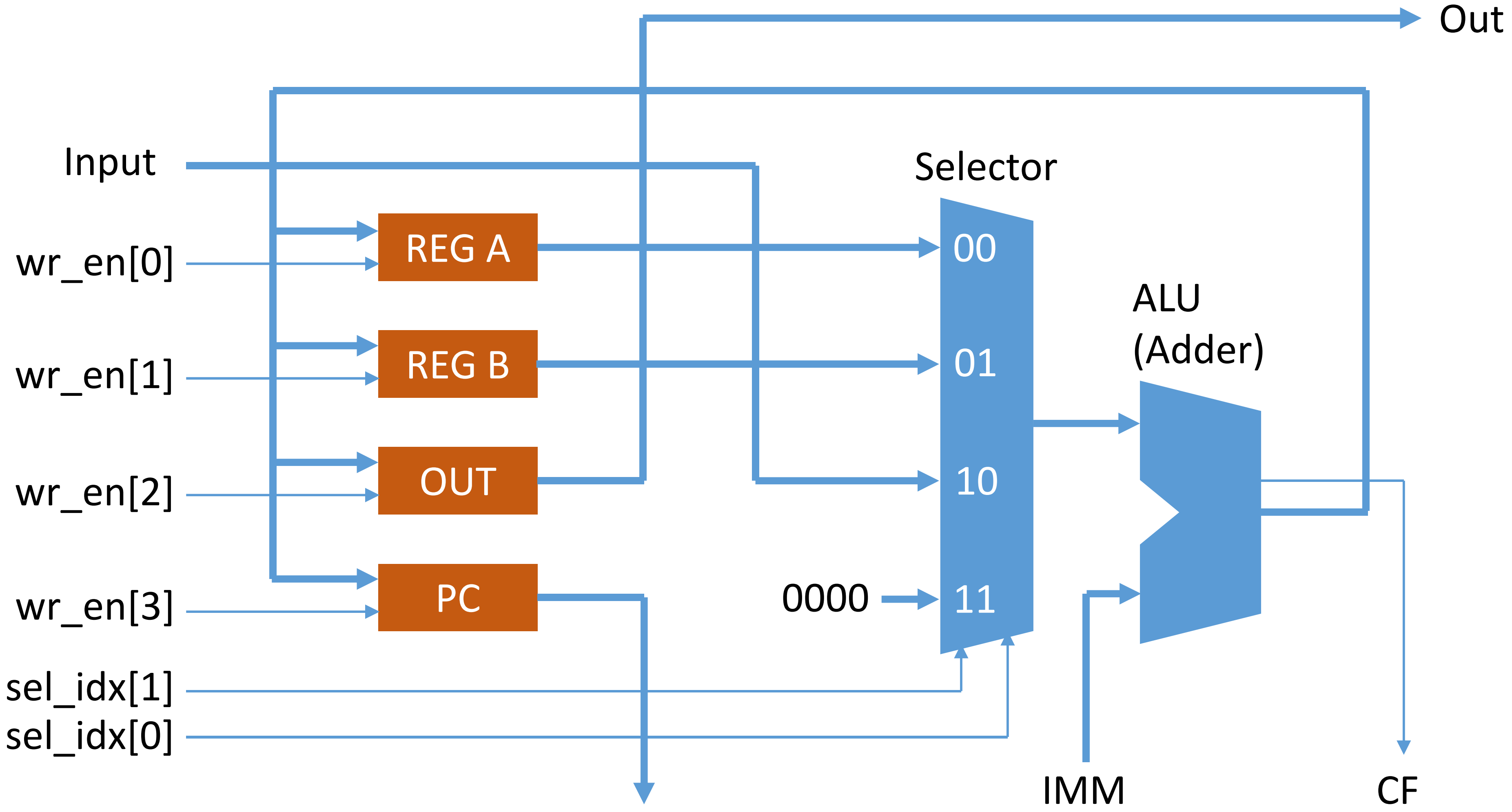
- "00001001" denotes MOV A,1001 as assembly code
  - $(0001001)_b$ is a machine code, and it can be written by a hexadecimal value
  - "MOV" is a mnemonic (Instruction)
  - "1001" is immediate value

Instruction Set

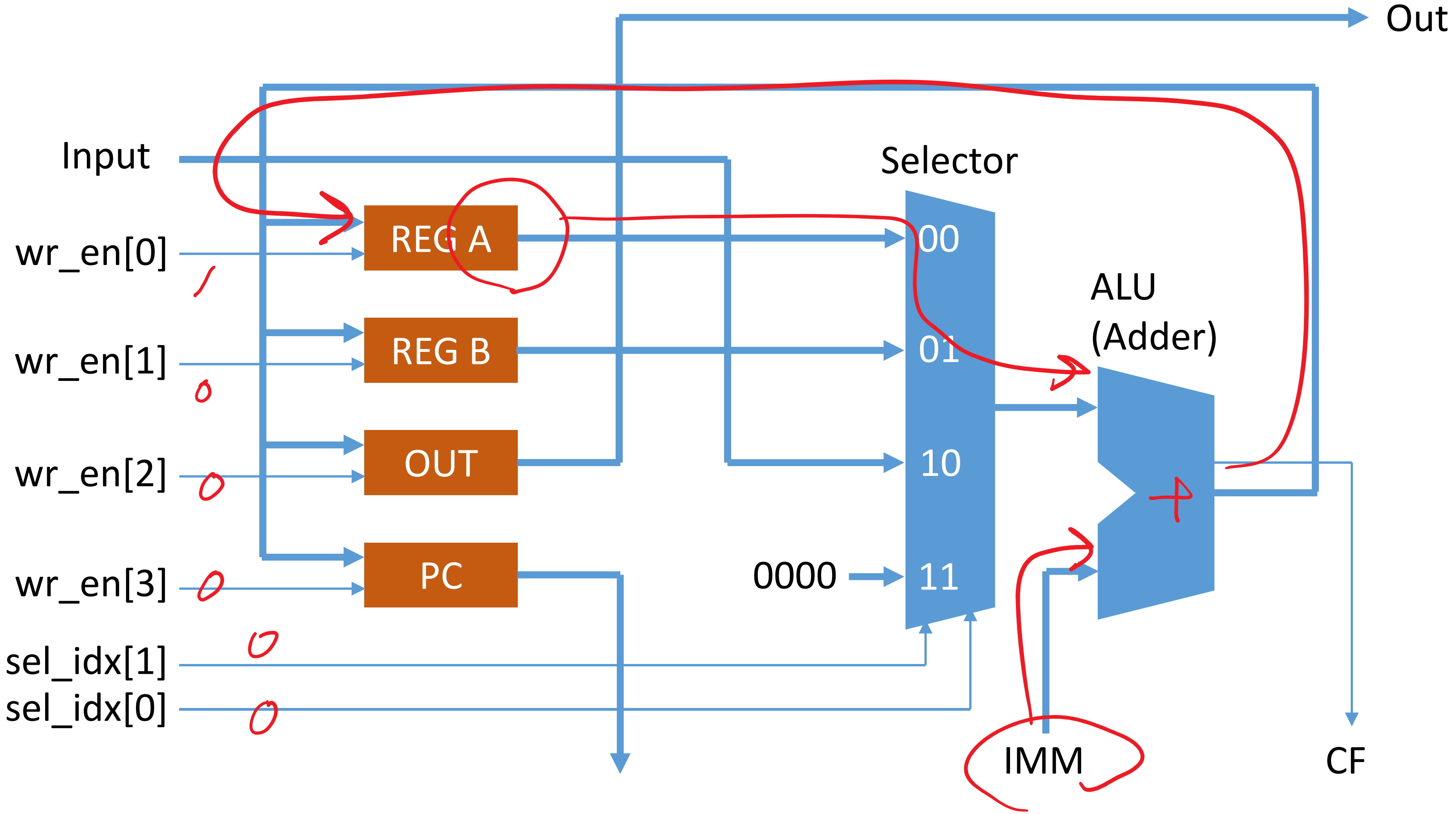
- MOV A,IMM A<-IMM
- MOV B,IMM B<-IMM
- MOV A,B A<-B
- MOV B,A B<-A
- ADD A,IMM A<-A+IMM, and CF<-Carry bit
- ADD B,IMM B<-B+IMM, and CF<-Carry bit
- IN A A<-INPUT (GPIO)
- IN B B<-INPUT (GPIO)
- OUT IMM OUTPUT(GPIO)<-IMM
- OUT B OUTPUT(GPIO)<-B
- JMP IMM PC<-IMM
- JNC IMM IF CF==0 THEN PC<-IMM ELSE PC<-PC+1

A: Register A
B: Register B
PC: Program Counter
IMM: Immediate value
CF: Carry Flag

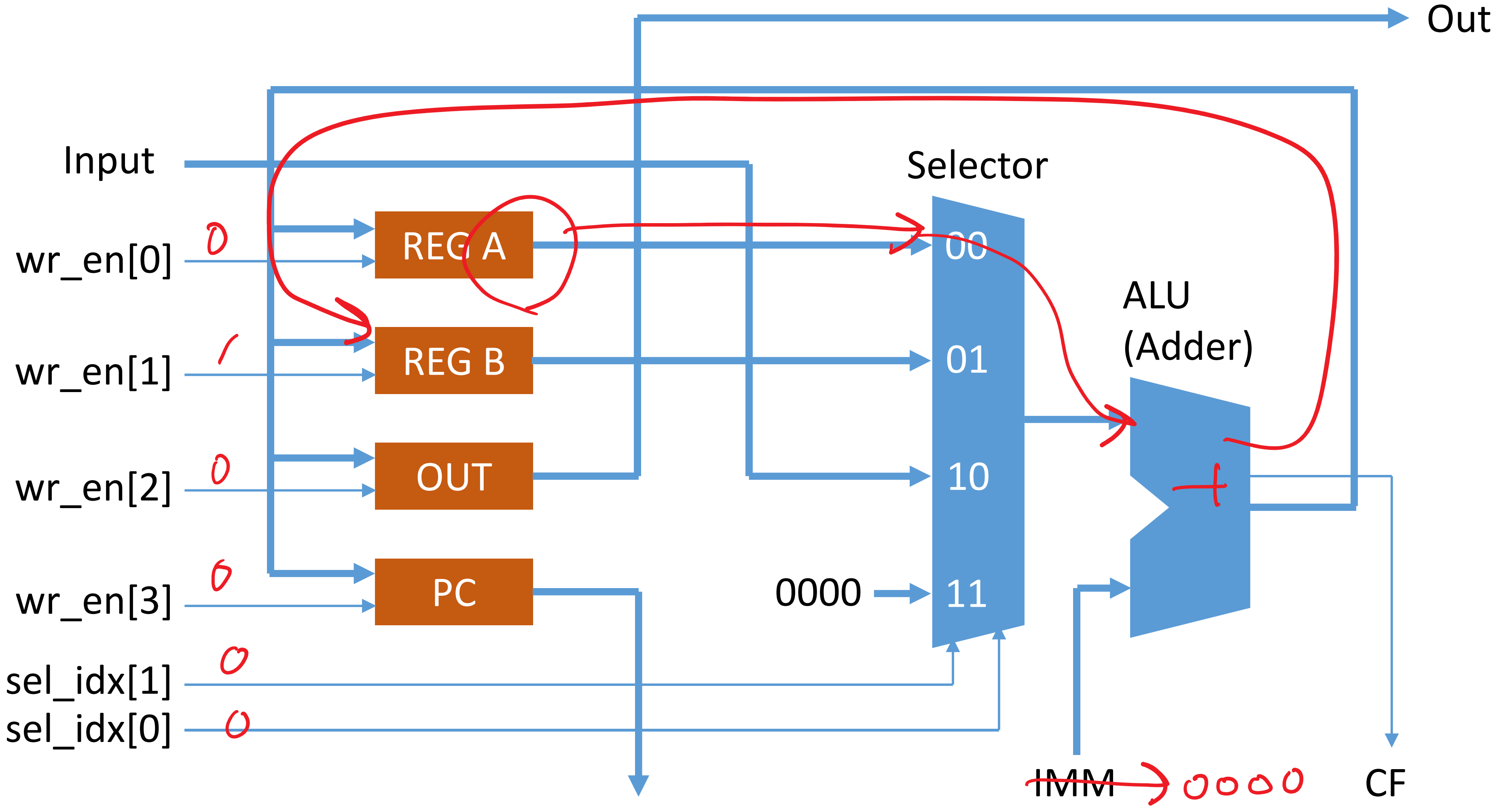
Data Path (Architecture)



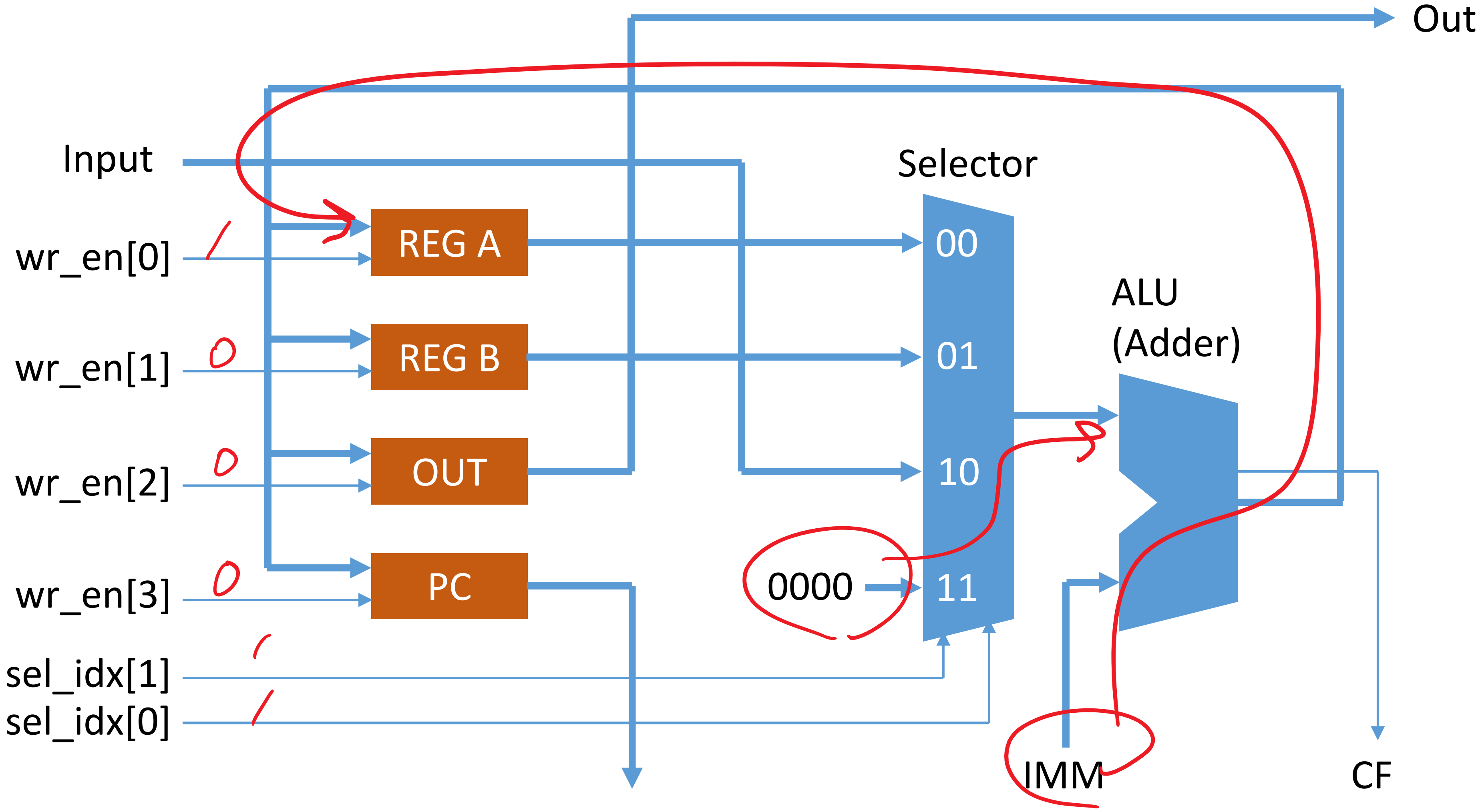
Add Operation (ADD A,IMM)



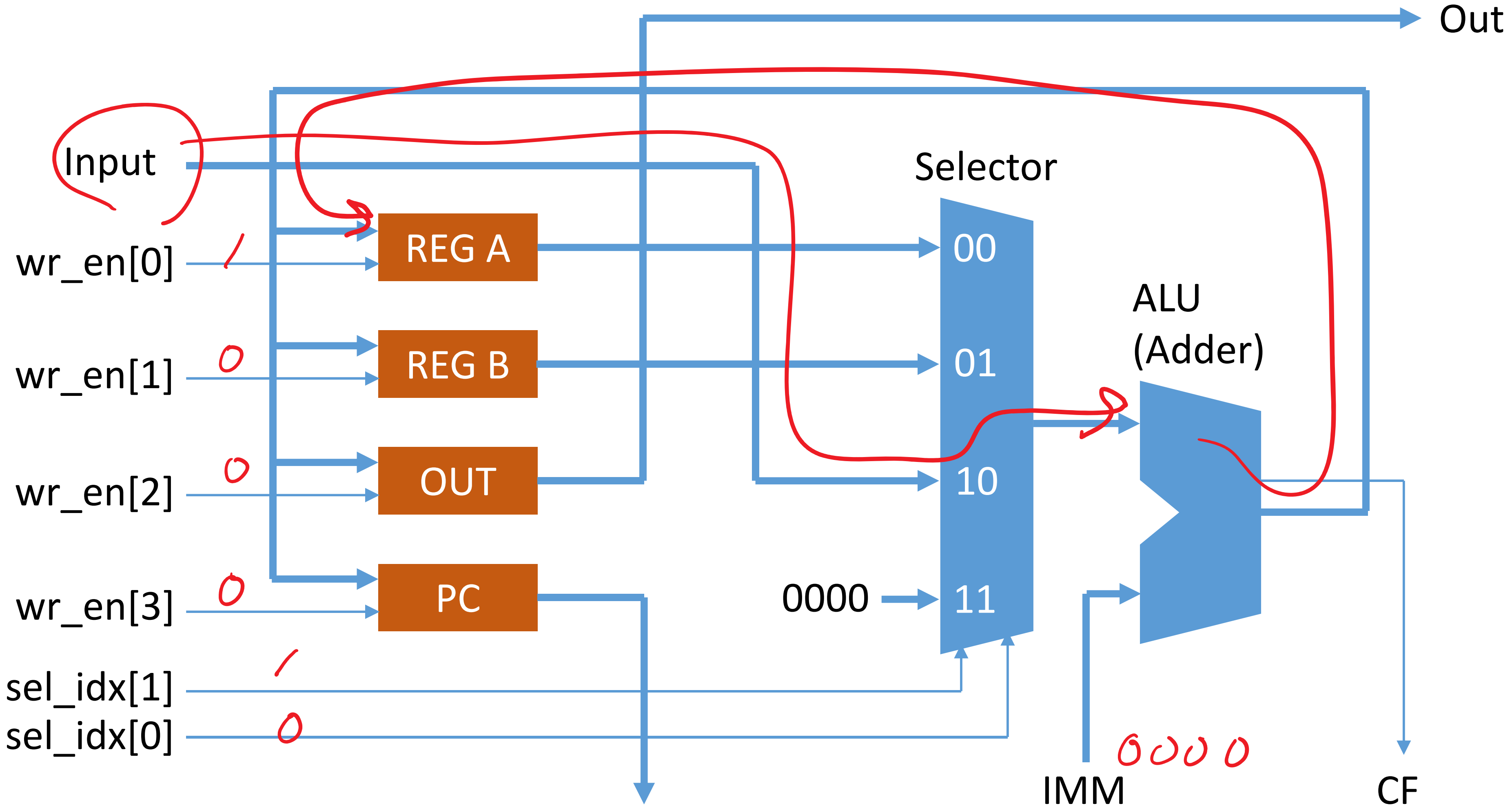
MOV Operation (MOV A,B)



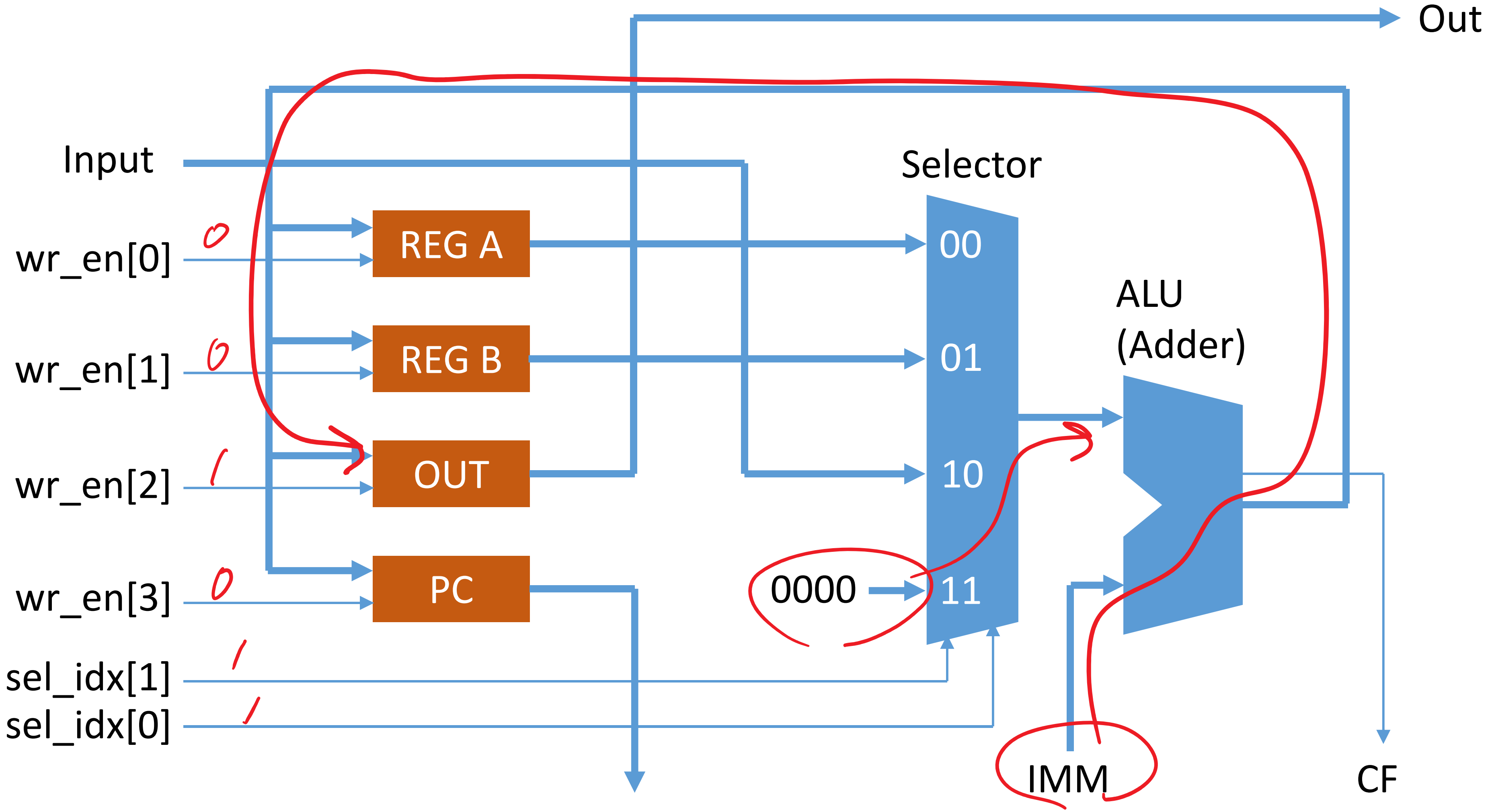
MOV Operation (MOV A,IMM)



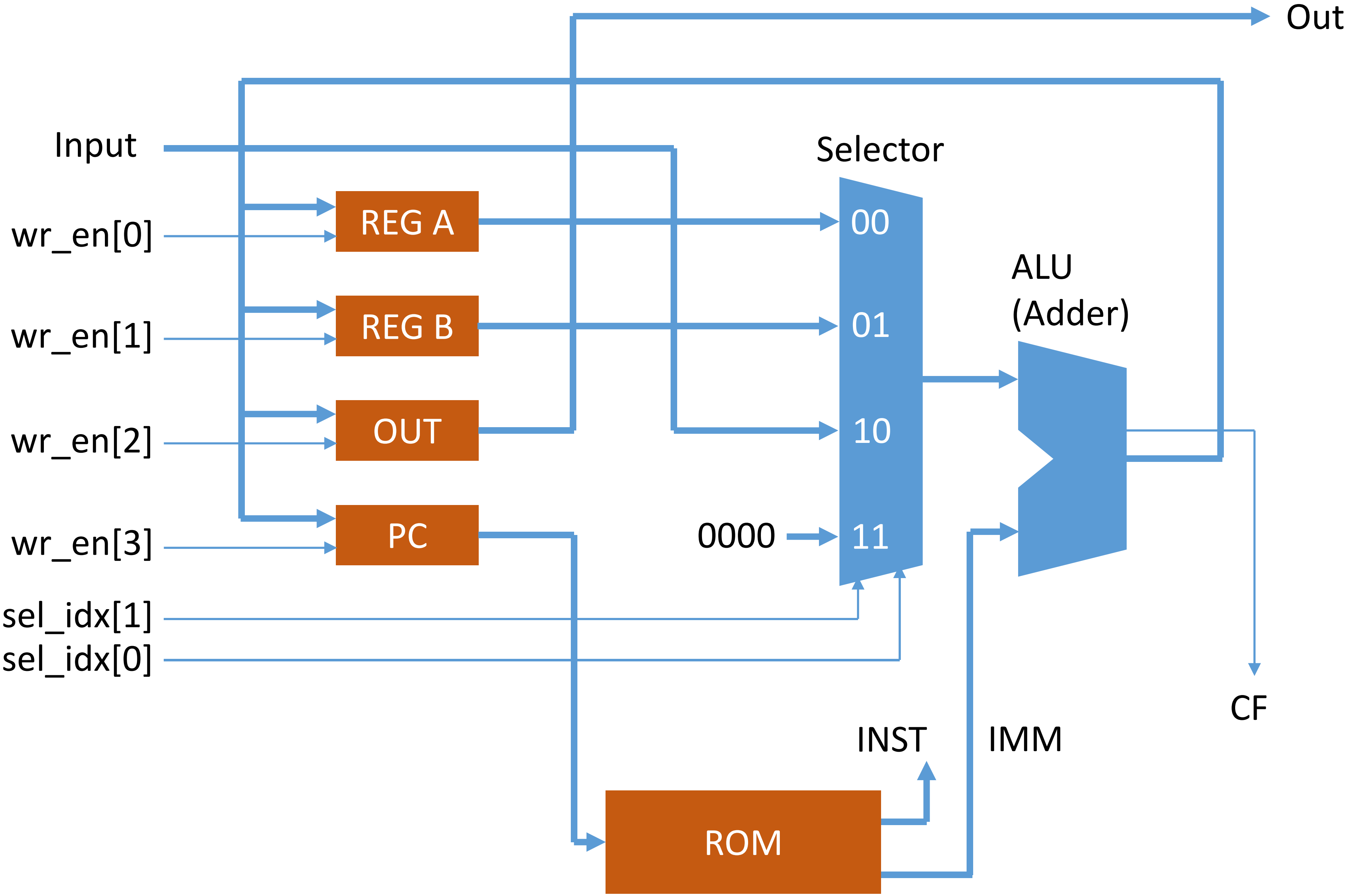
Read Input Operation (IN A)



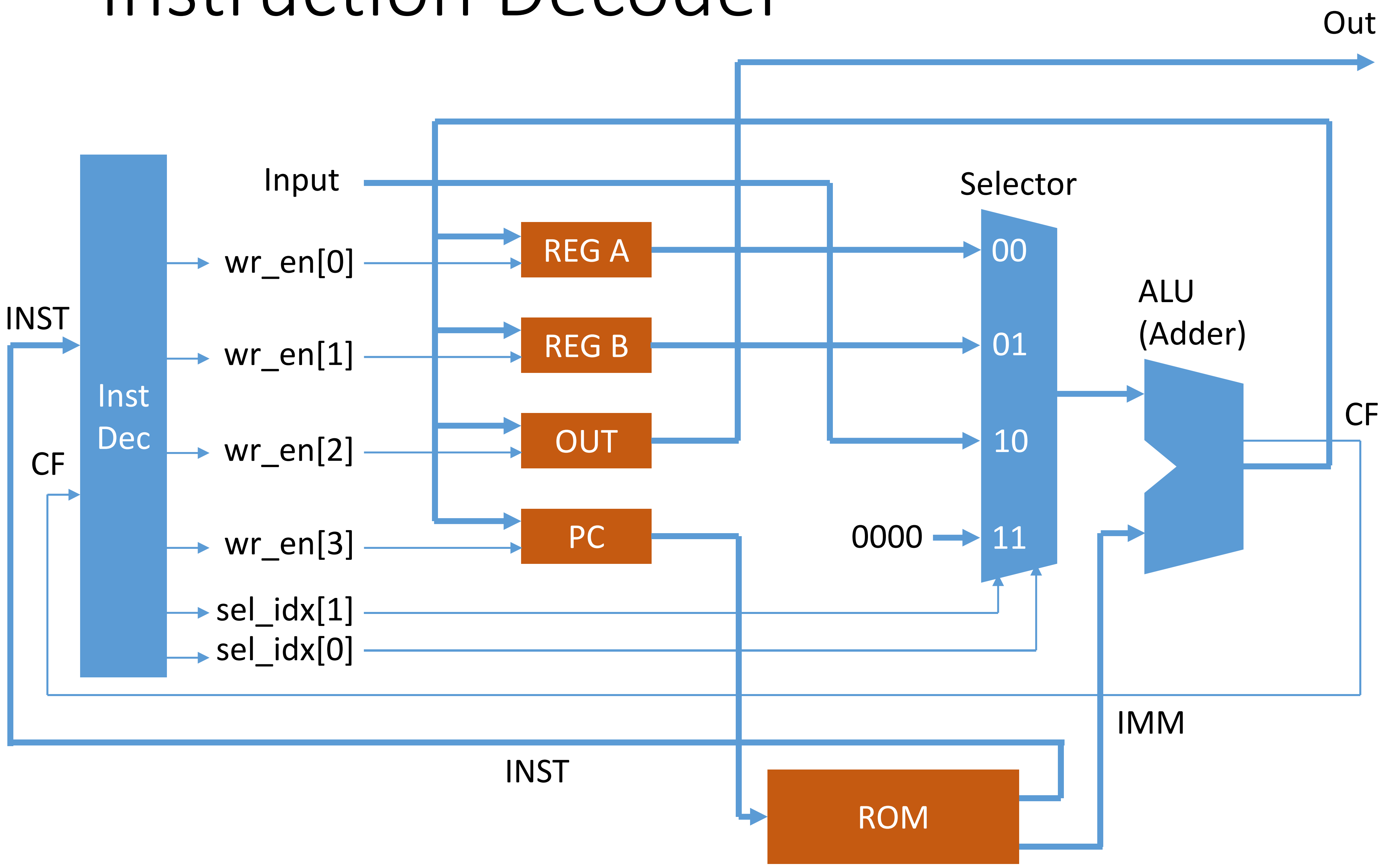
Write Output Operation (Out IMM)



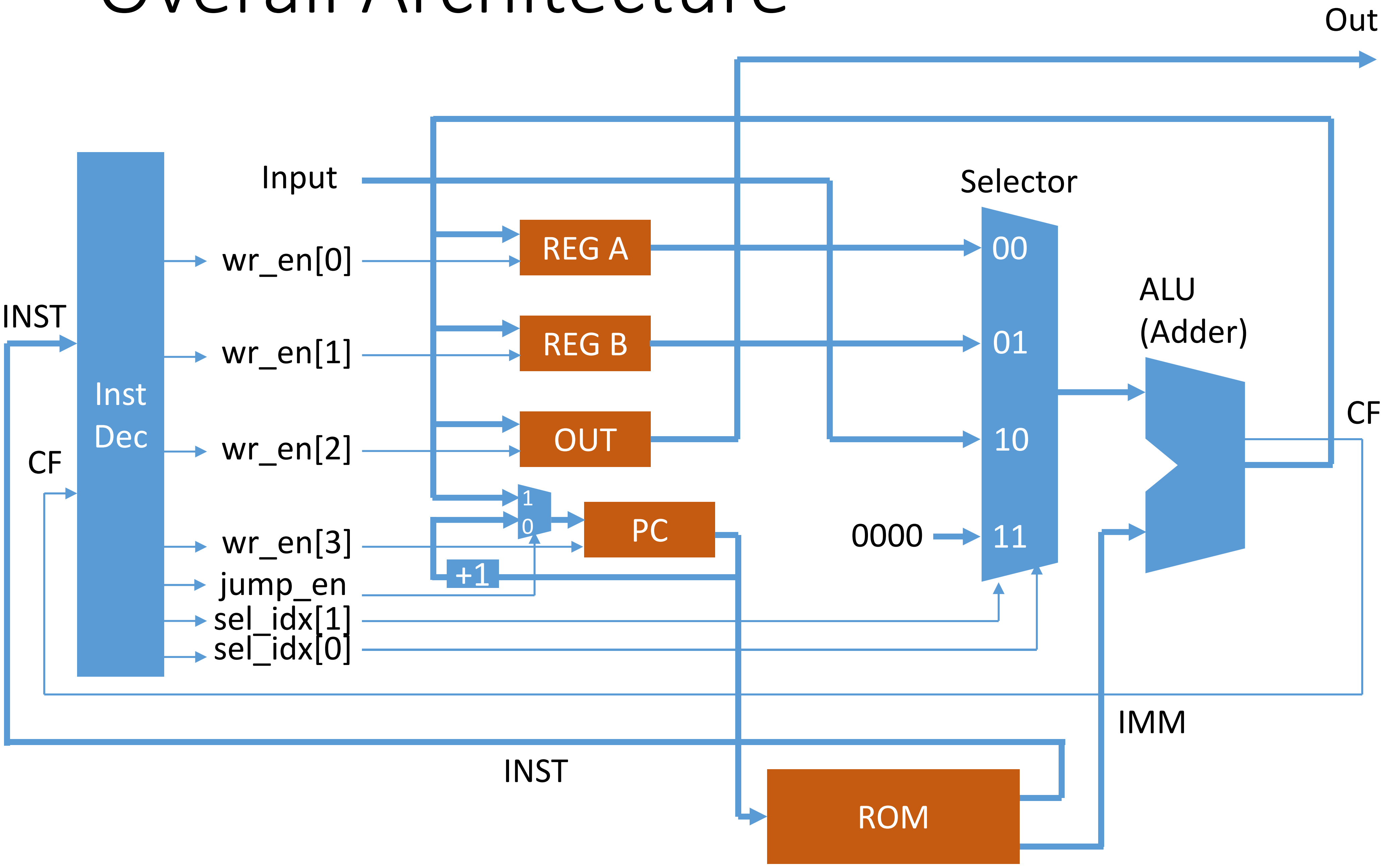
Program Memory (ROM)



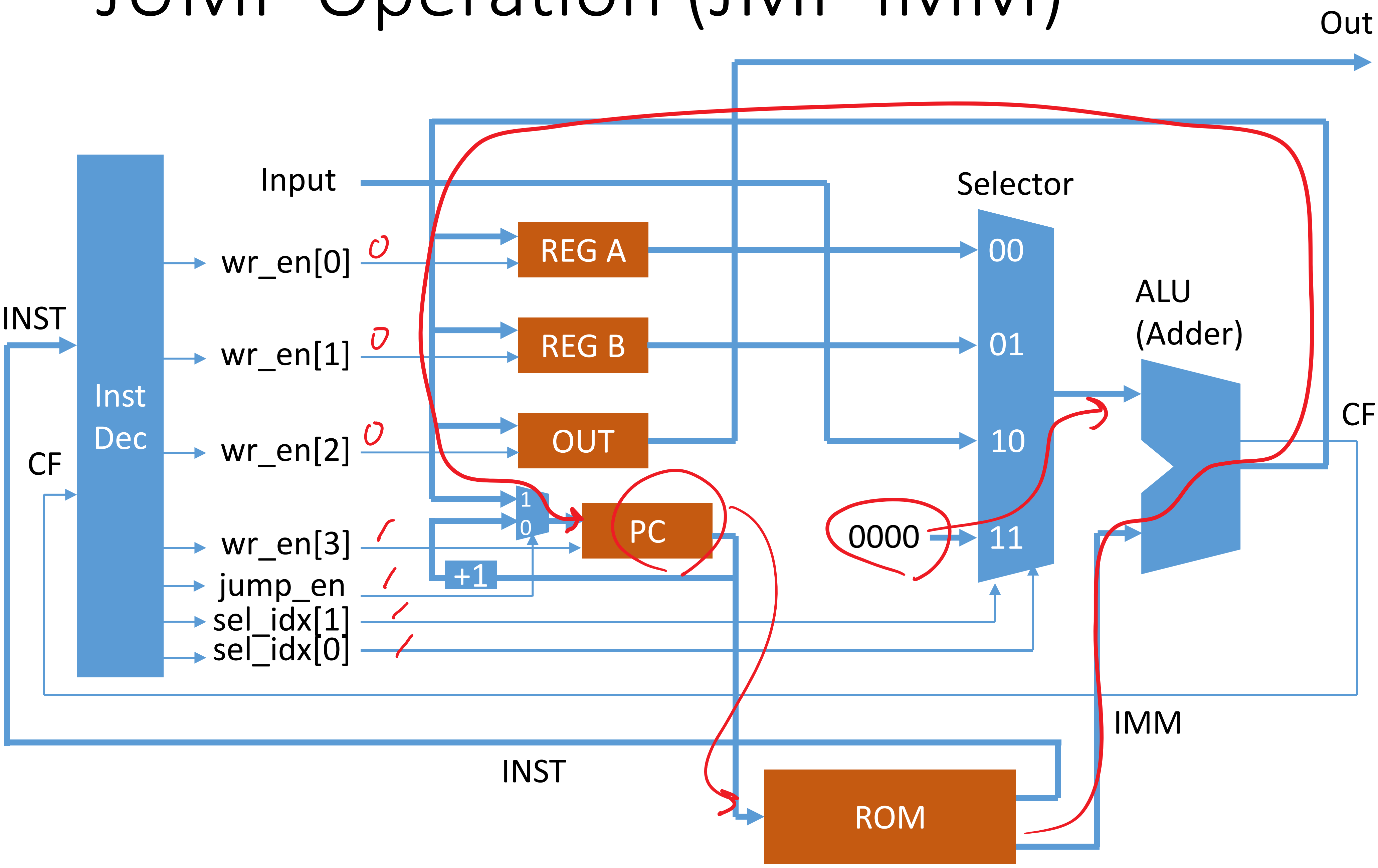
Instruction Decoder



Overall Architecture



JUMP Operation (JMP IMM)

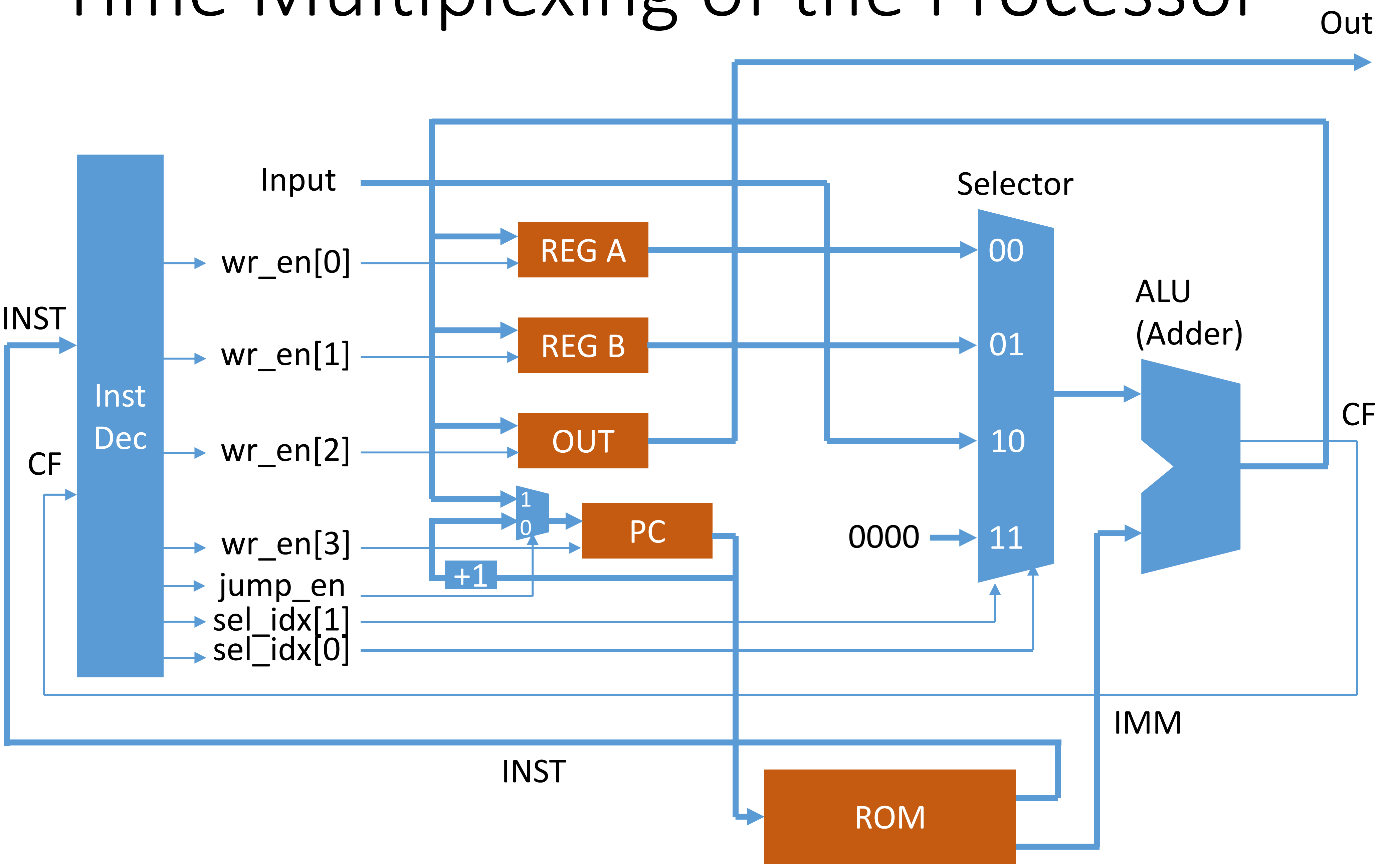


Truth Table for an Instruction Decoder

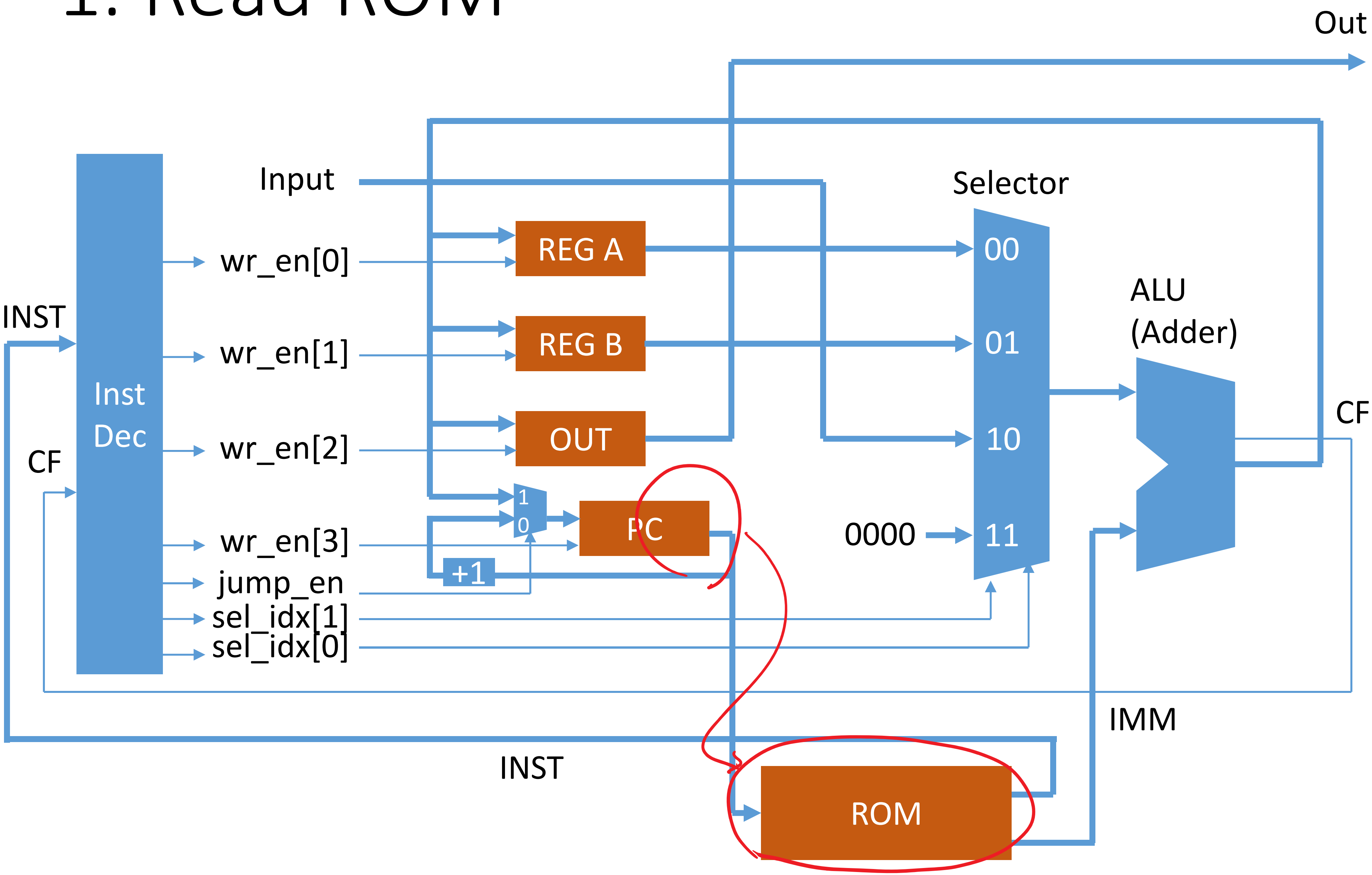
	Inst[3:0]	CF	wr_en[3:0]	jump_en	sel_idx[1:0]
MOV A,IMM	0000	X	1001	0	11
MOV B,IMM	0001	X	1010	0	11
MOV A,B	0010	X	1001	0	01
MOV B,A	0011	X	1010	0	00
ADD A,IMM	0100	X	1001	0	00
ADD B,IMM	0101	X	1001	0	01
IN A	0110	X	1001	0	10
IN B	0111	X	1010	0	10
OUT IMM	1000	X	1100	0	11
OUT B	1001	X	1100	0	01
JMP IMM	1010	X	1000	1	11
JNC IMM	1011	0	1000	1	11
JNC IMM	1011	1	1000	0	11

Behavior Design of Tiny Processor

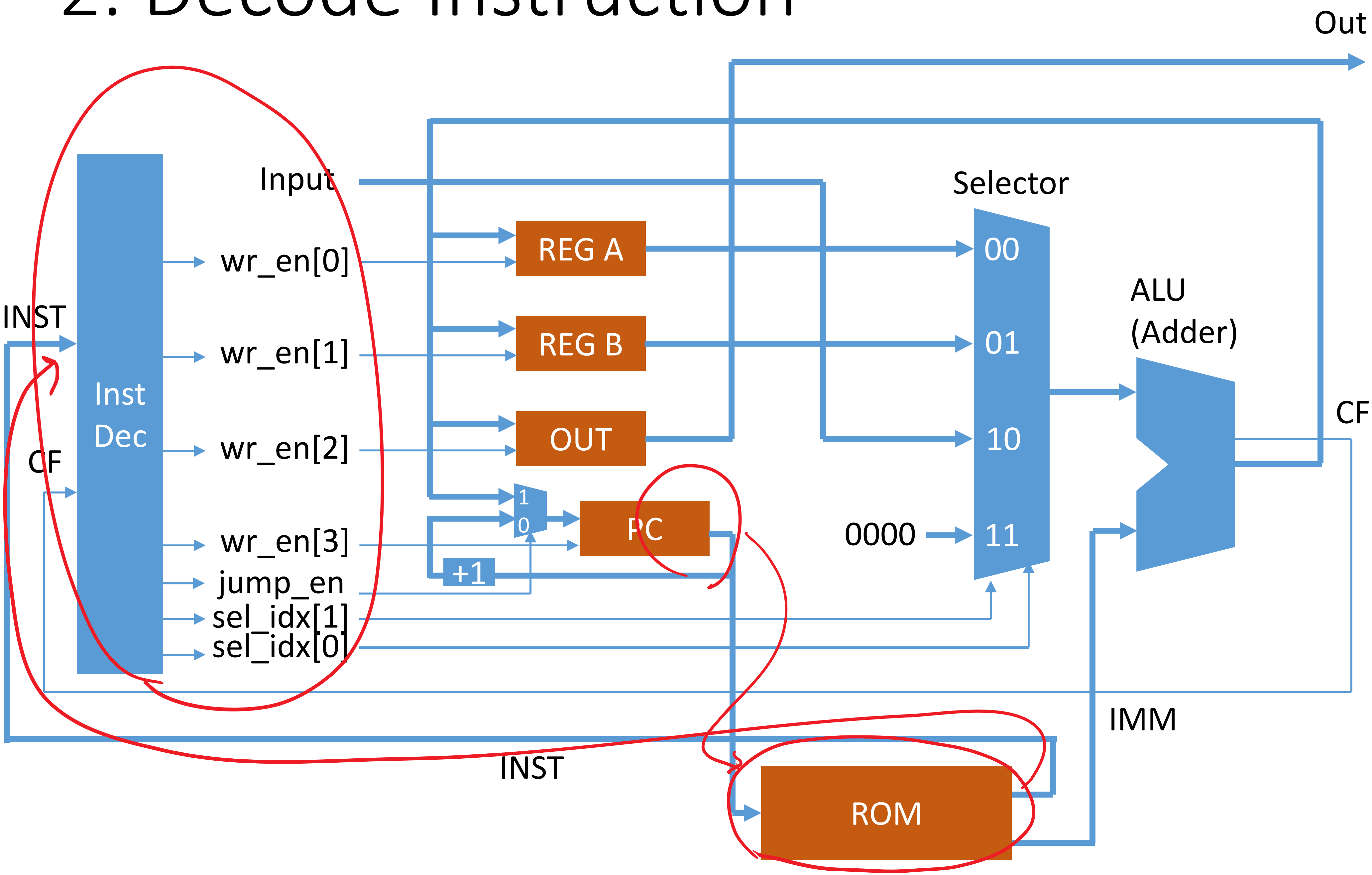
Time Multiplexing of the Processor



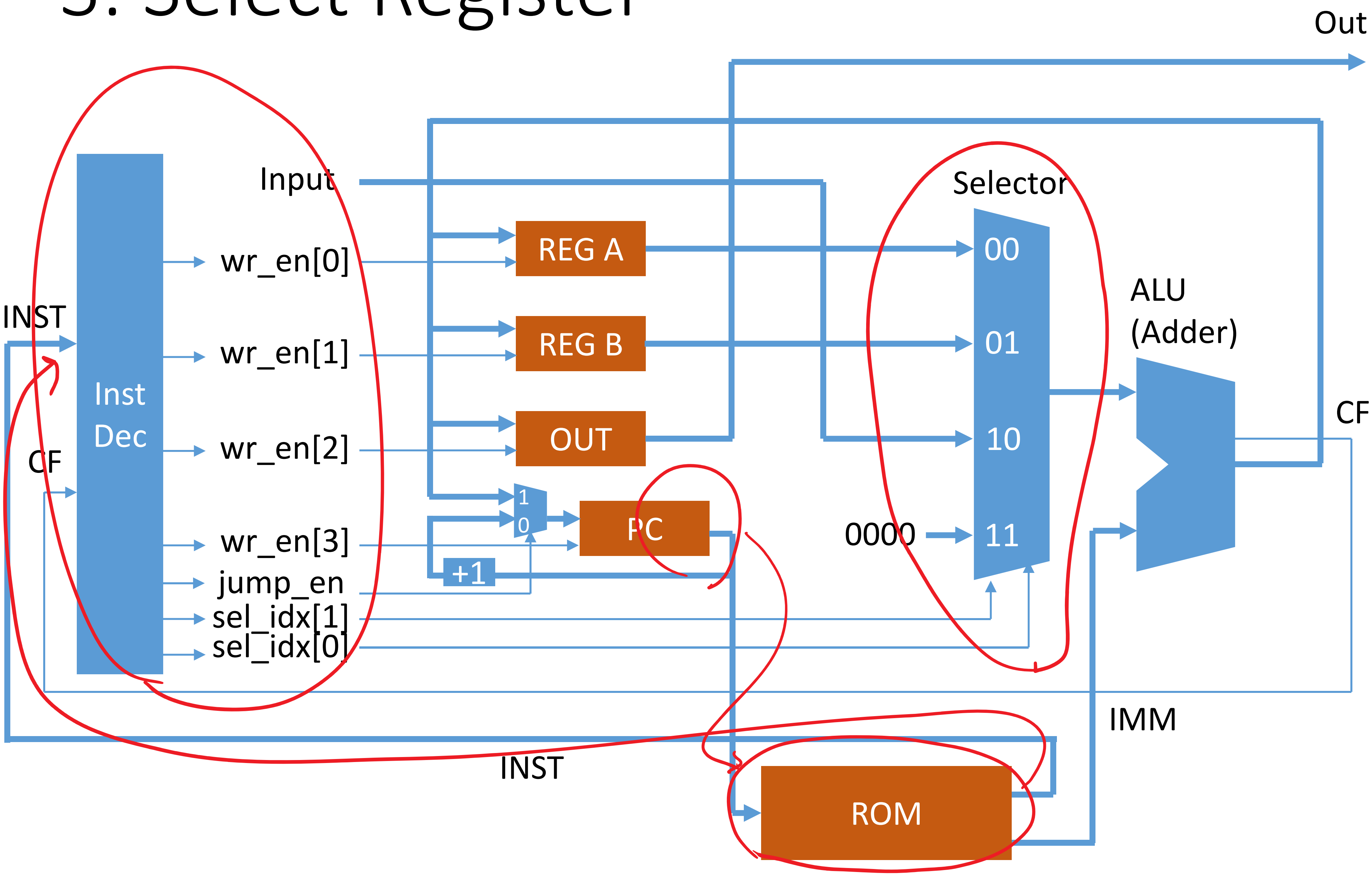
1. Read ROM


Out
Input
Selector
REG A
00
wr_en[0]
INST
REG B
01
ALU
(Adder)
wr_en[1]
Inst
Dec
OUT
10
CF
wr_en[2]
CF
1
0
PC
0000
11
wr_en[3]
+1
jump_en
sel_idx[1]
sel_idx[0]
IMM
INST
ROM

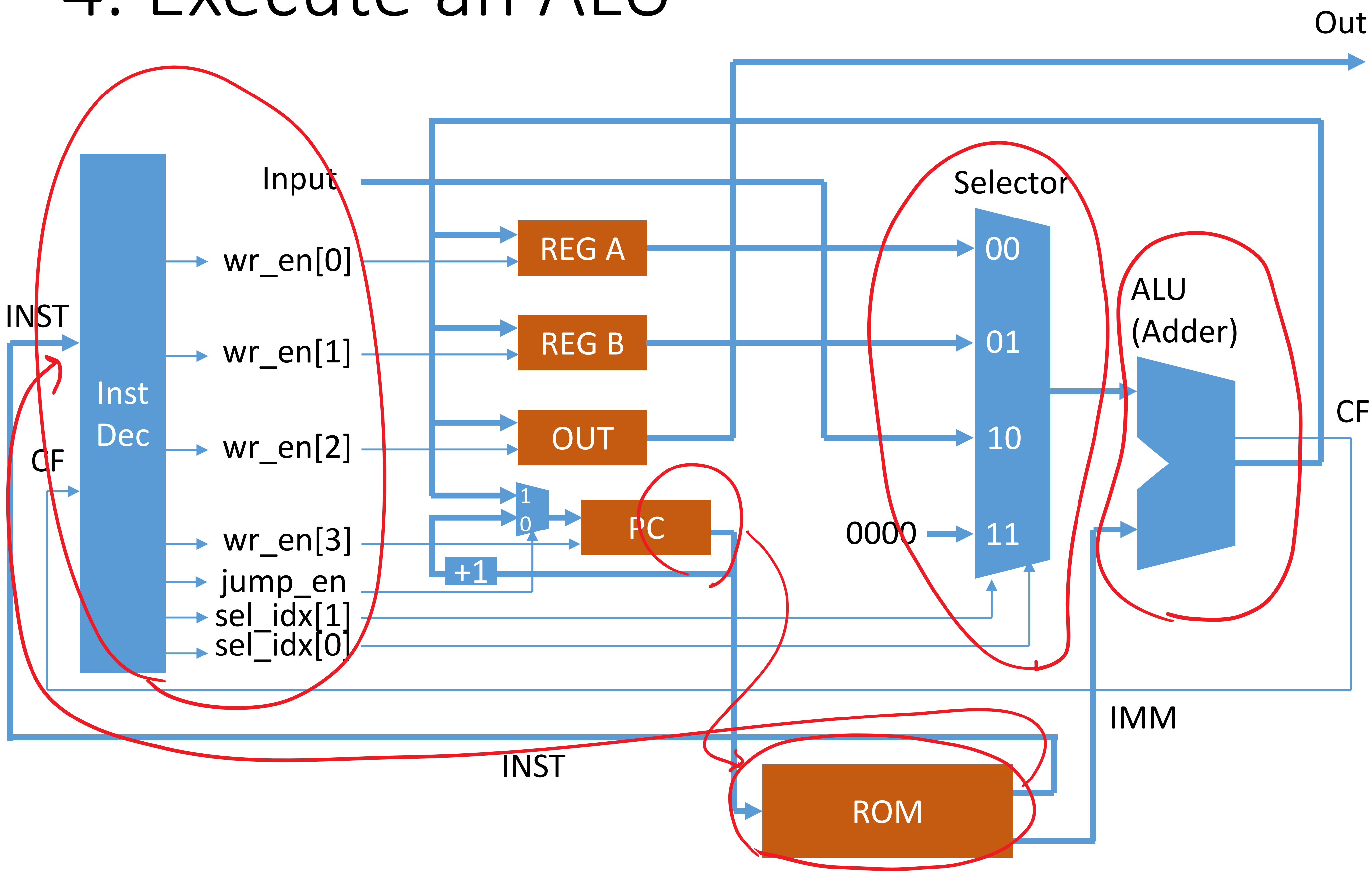
2. Decode Instruction



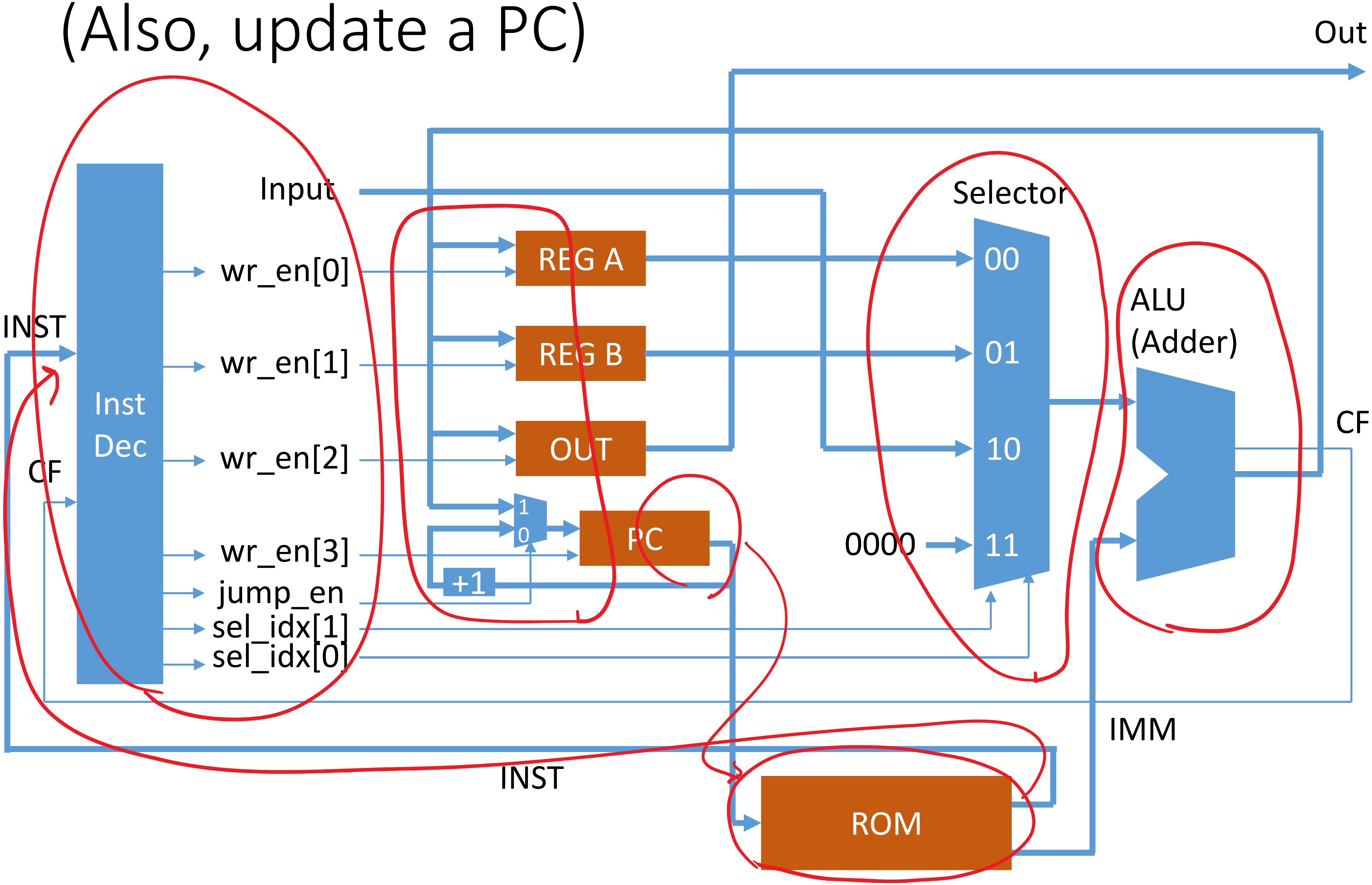
3. Select Register



4. Execute an ALU

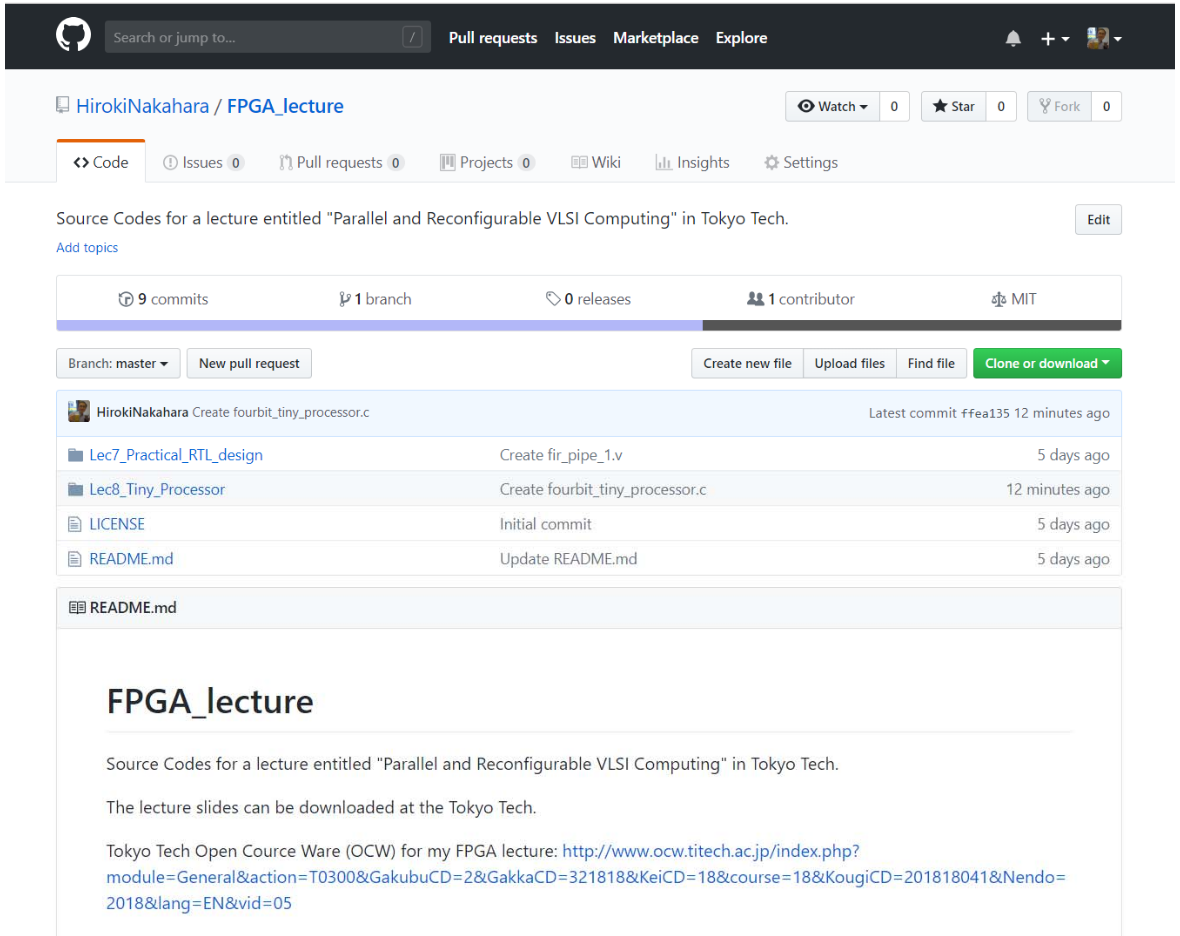


5. Write Back to Registers
(Also, update a PC)



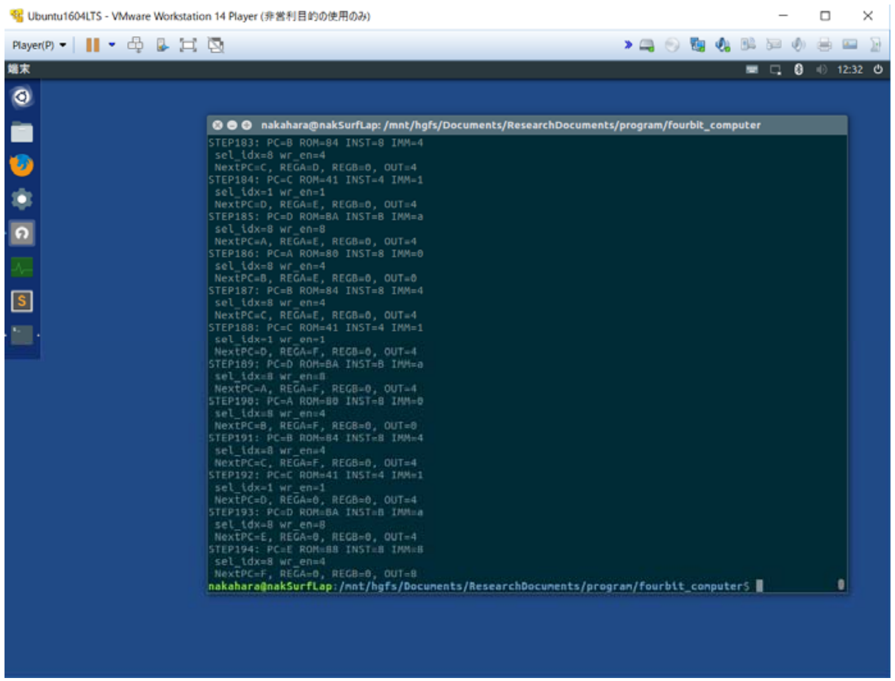
C/C++ Source Code

- see,http://https://github.com/HirokiNakahara/FPGA_lecture/blob/master/Lec8_Tiny_Processor/fourbit_tiny_processor.c



Case Study: LED Controller

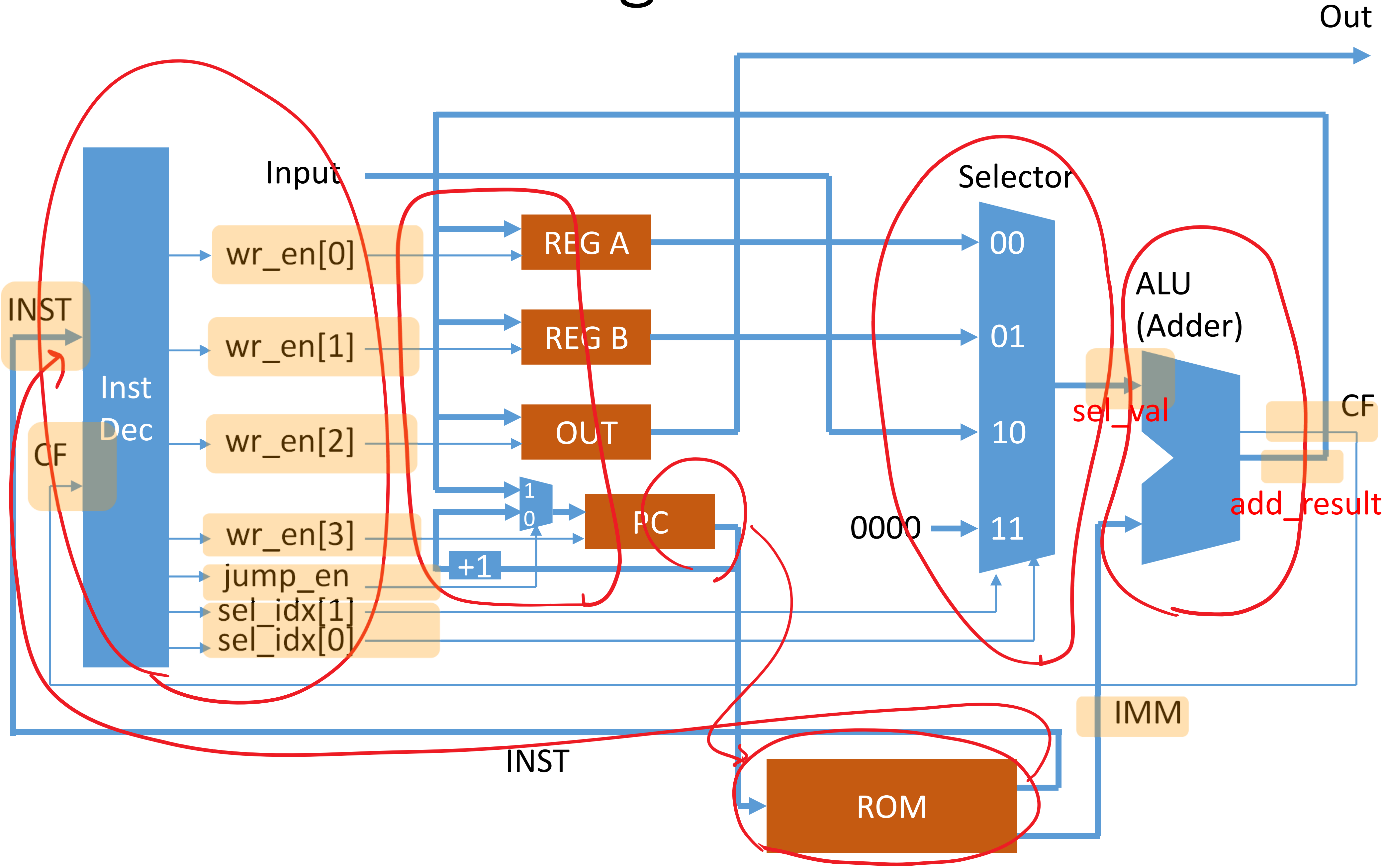
```
0x81,   // OUT 0001
0x82,   // OUT 0010
0x84,   // OUT 0100
0x88,   // OUT 1000
0x84,   // OUT 0100
0x82,   // OUT 0010
0xA0,   // JMP to 0000
0x00,   // Followings are NOP
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
```



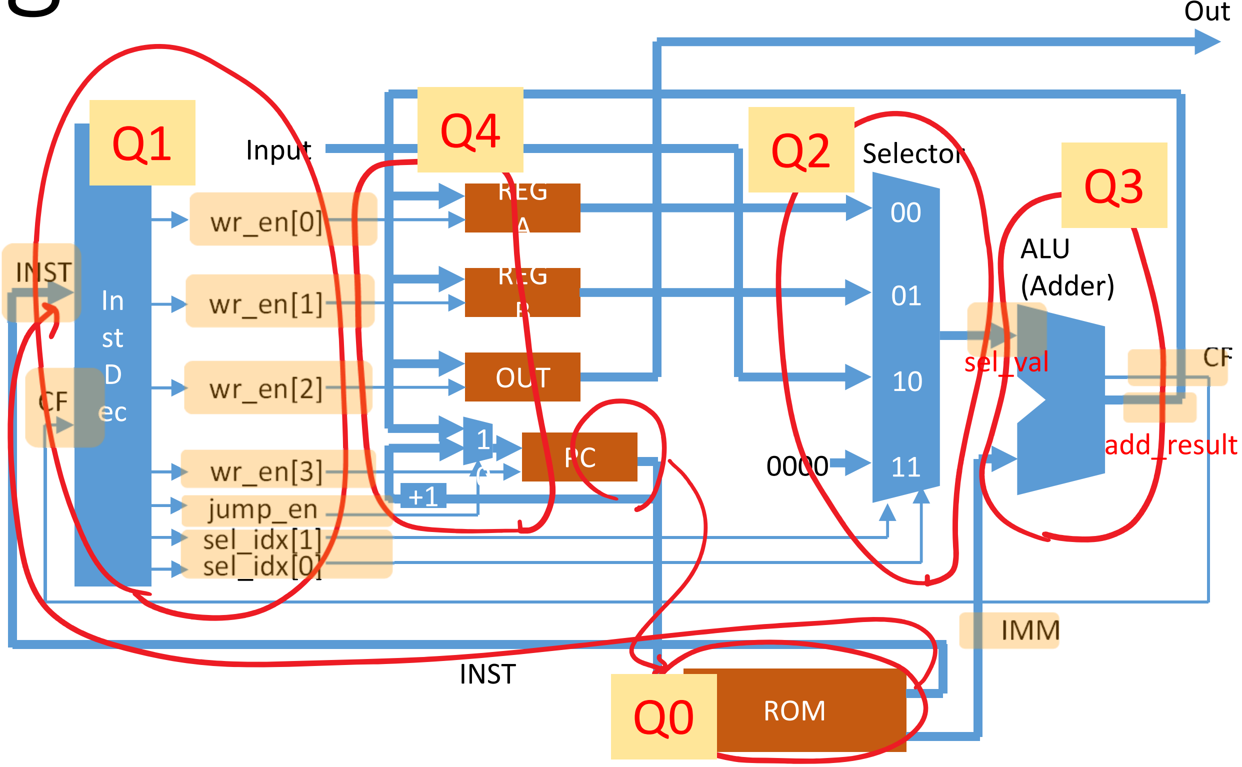
Execute a C behavior code on your PC

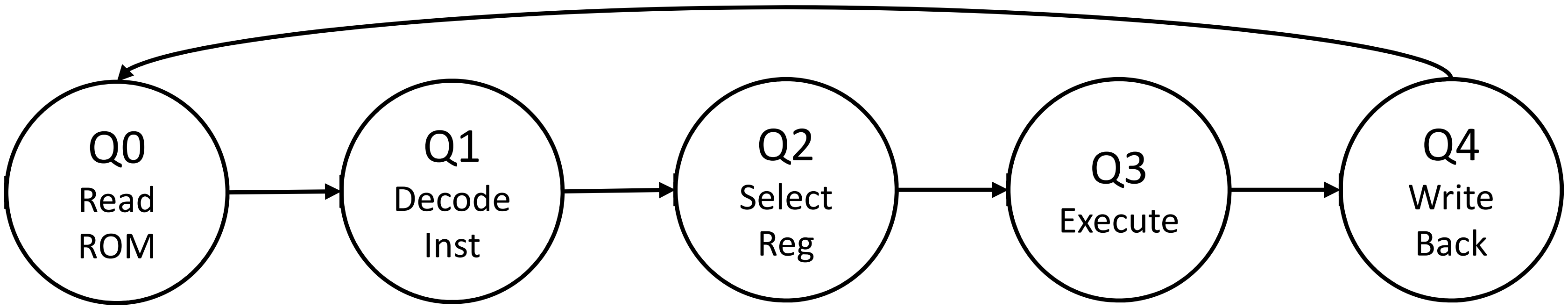
RTL Design

Insert Virtual Register



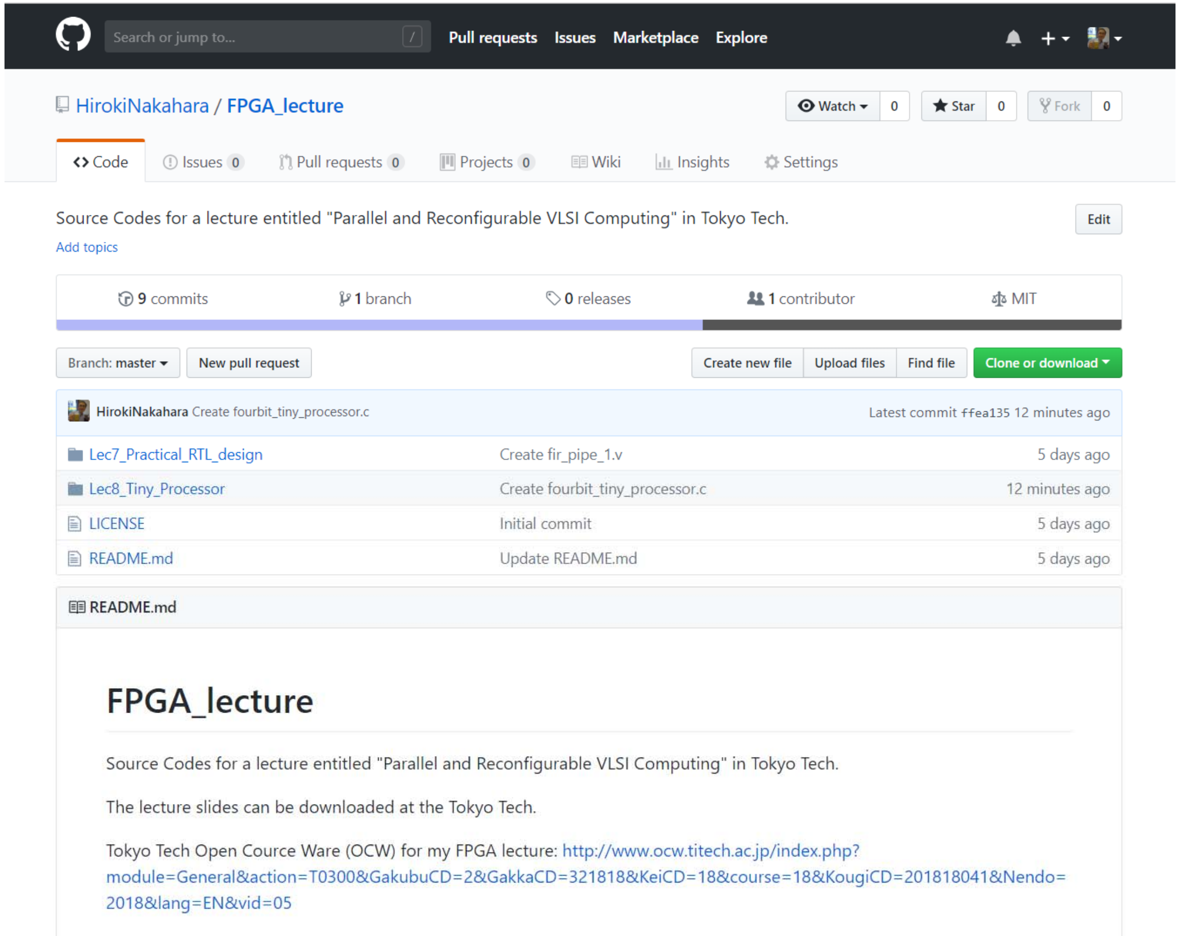
Design an FSM





RTL Source Code

- see,http://https://github.com/HirokiNakahara/FPGA_lecture/blob/master/Lec8_Tiny_Processor/
    - RTL: fourbit_tiny_processor.v
    - Testbench: testbench_tiny_processor.v

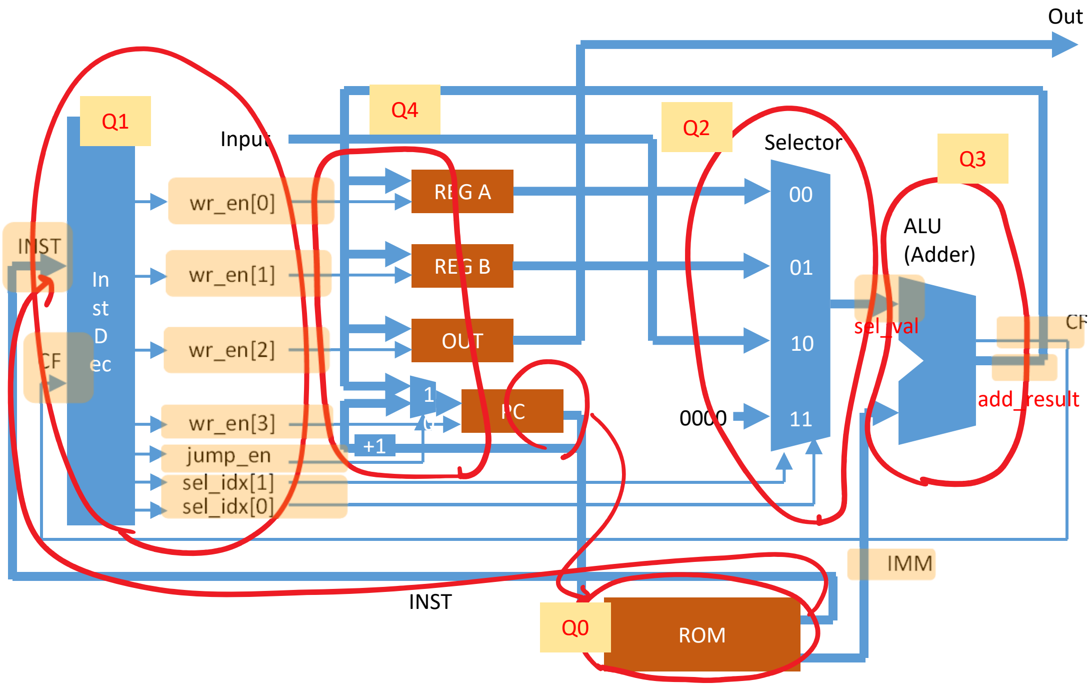


RTL Code for Reading ROM

```
        case( state)
        3'b000:begin // read ROM
            state <= 3'b001;
            {INST, IMM} <= ROM( PC);
        end
```

⋮

```
function [7:0]ROM( input [3:0]address);
begin
    case( address)
    4'b0000: ROM = 8'h81;
    4'b0001: ROM = 8'h82;
    4'b0010: ROM = 8'h84;
    4'b0011: ROM = 8'h88;
    4'b0100: ROM = 8'h84;
    4'b0101: ROM = 8'h82;
    4'b0110: ROM = 8'hA0;
    4'b0111: ROM = 8'h00;
    4'b1000: ROM = 8'h00;
    4'b1001: ROM = 8'h00;
    4'b1010: ROM = 8'h00;
    4'b1011: ROM = 8'h00;
    4'b1100: ROM = 8'h00;
    4'b1101: ROM = 8'h00;
    4'b1110: ROM = 8'h00;
    4'b1111: ROM = 8'h00;
    endcase
end
endfunction
```



RTL for an Instruction Decoder

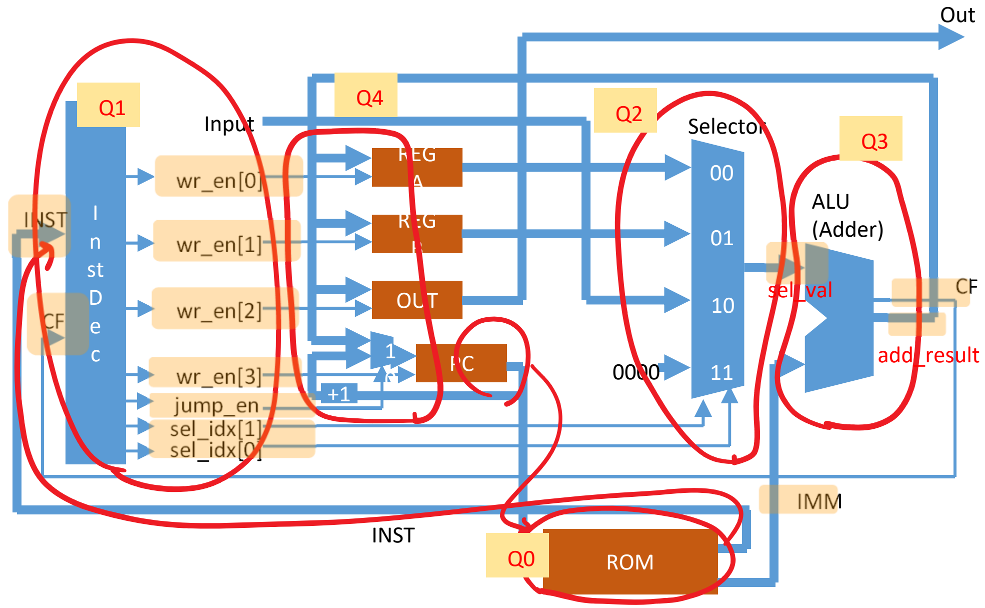
```
        3'b001:begin // decode inst
            state <= 3'b010;
            {wr_en, jump_en, sel_idx} <= INST_DEC( INST, CF);
        end
```

⋮

```
function [6:0]INST_DEC( input [3:0]inst, input CF);
begin
```

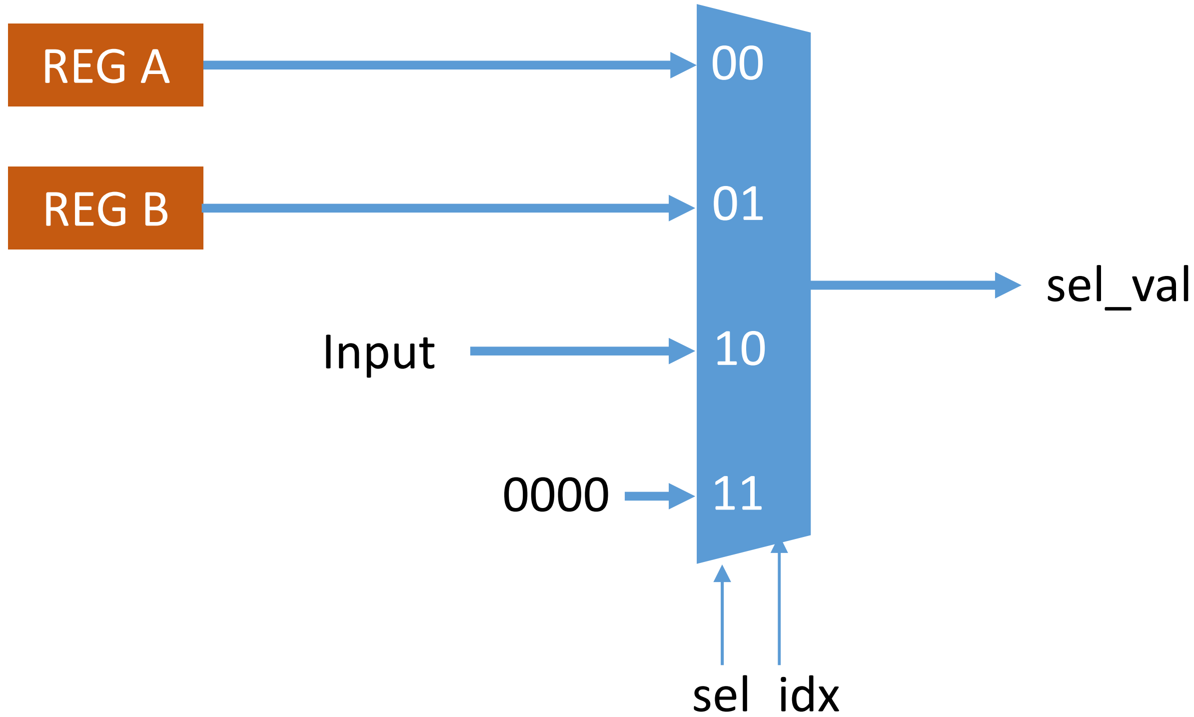
"casex" accept a don't care "x"

```
    casex( {inst, CF})
    5'b0000_x: INST_DEC = 7'b1001_0_11;
    5'b0001_x: INST_DEC = 7'b1010_0_11;
    5'b0010_x: INST_DEC = 7'b1001_0_01;
    5'b0011_x: INST_DEC = 7'b1010_0_00;
    5'b0100_x: INST_DEC = 7'b1001_0_00;
    5'b0101_x: INST_DEC = 7'b1001_0_01;
    5'b0110_x: INST_DEC = 7'b1001_0_10;
    5'b0111_x: INST_DEC = 7'b1010_0_10;
    5'b1000_x: INST_DEC = 7'b1100_0_11;
    5'b1001_x: INST_DEC = 7'b1100_0_01;
    5'b1010_x: INST_DEC = 7'b1000_1_11;
    5'b1011_0: INST_DEC = 7'b1000_1_11;
    5'b1011_1: INST_DEC = 7'b1000_0_11;
    default:   INST_DEC = 7'b0000_0_00;
    endcase
end
endfunction
```



RTL for a Register Selector

```
            3'b010:begin // select reg
                state <= 3'b011;
                sel_val <= sel_idx == 2'b00 ? REGA :
                                   sel_idx == 2'b01 ? REGB :
                                   sel_idx == 2'b10 ? gpio_in :
                                   4'b0000;
            end
```

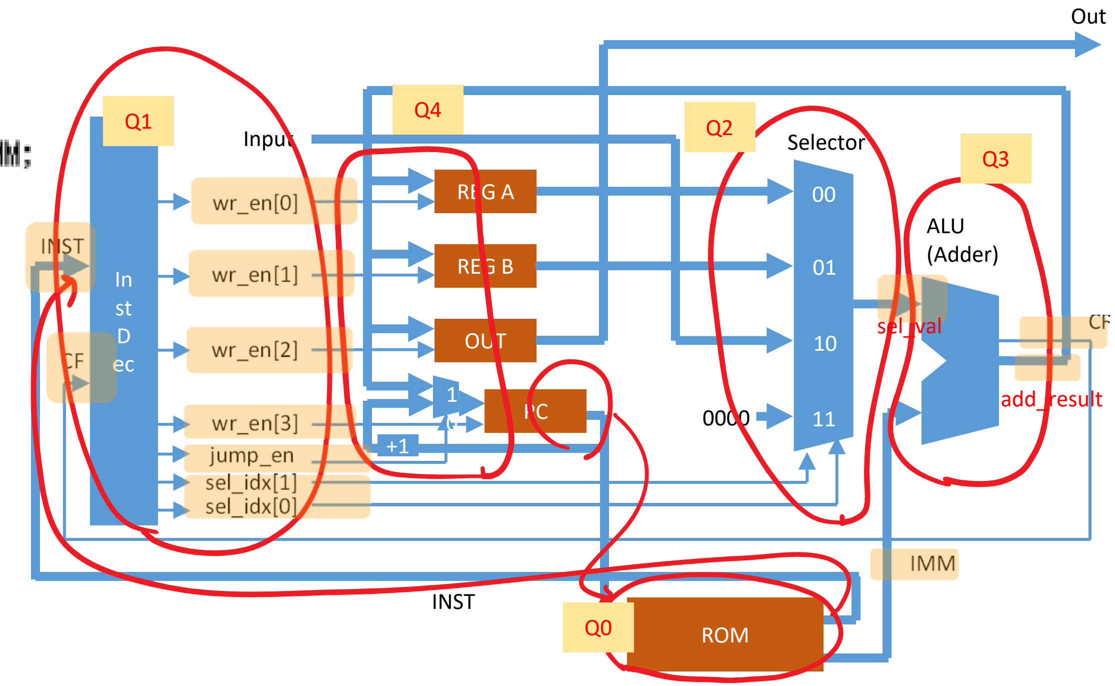


RTL for an ALU (4-bit Adder with CF)

```
3'b011:begin // execute
    state <= 3'b100;
    add_result <= tmp_add_result[3:0];
    CF <= tmp_add_result[4];
end
```

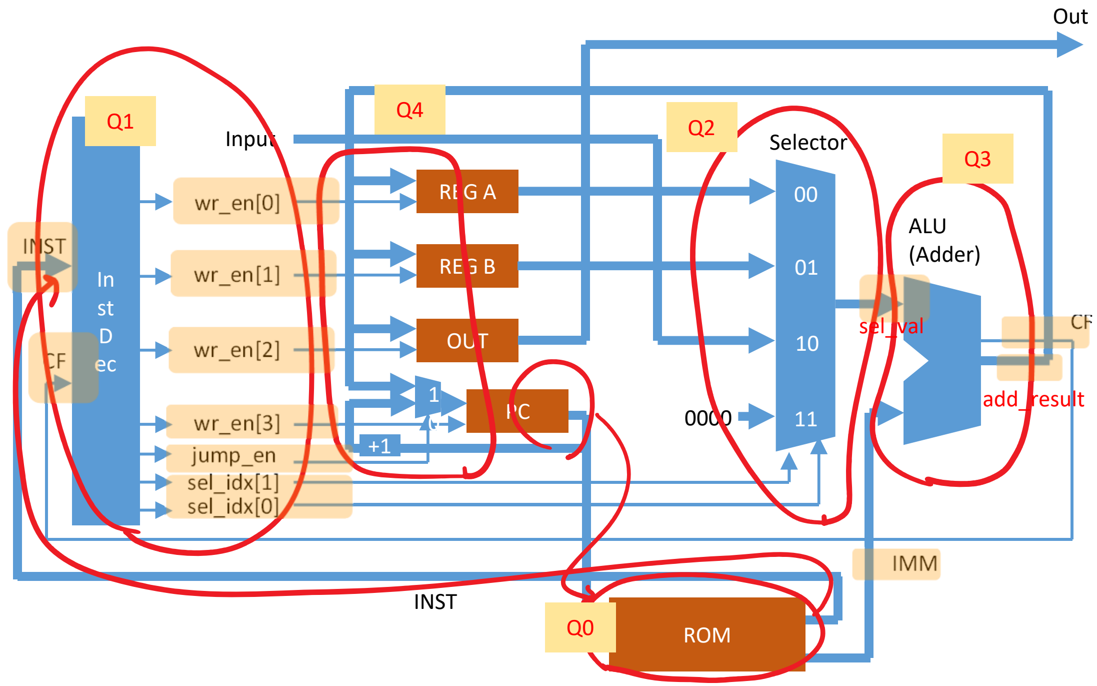
⋮

```
assign tmp_add_result = sel_val + IMM;
```



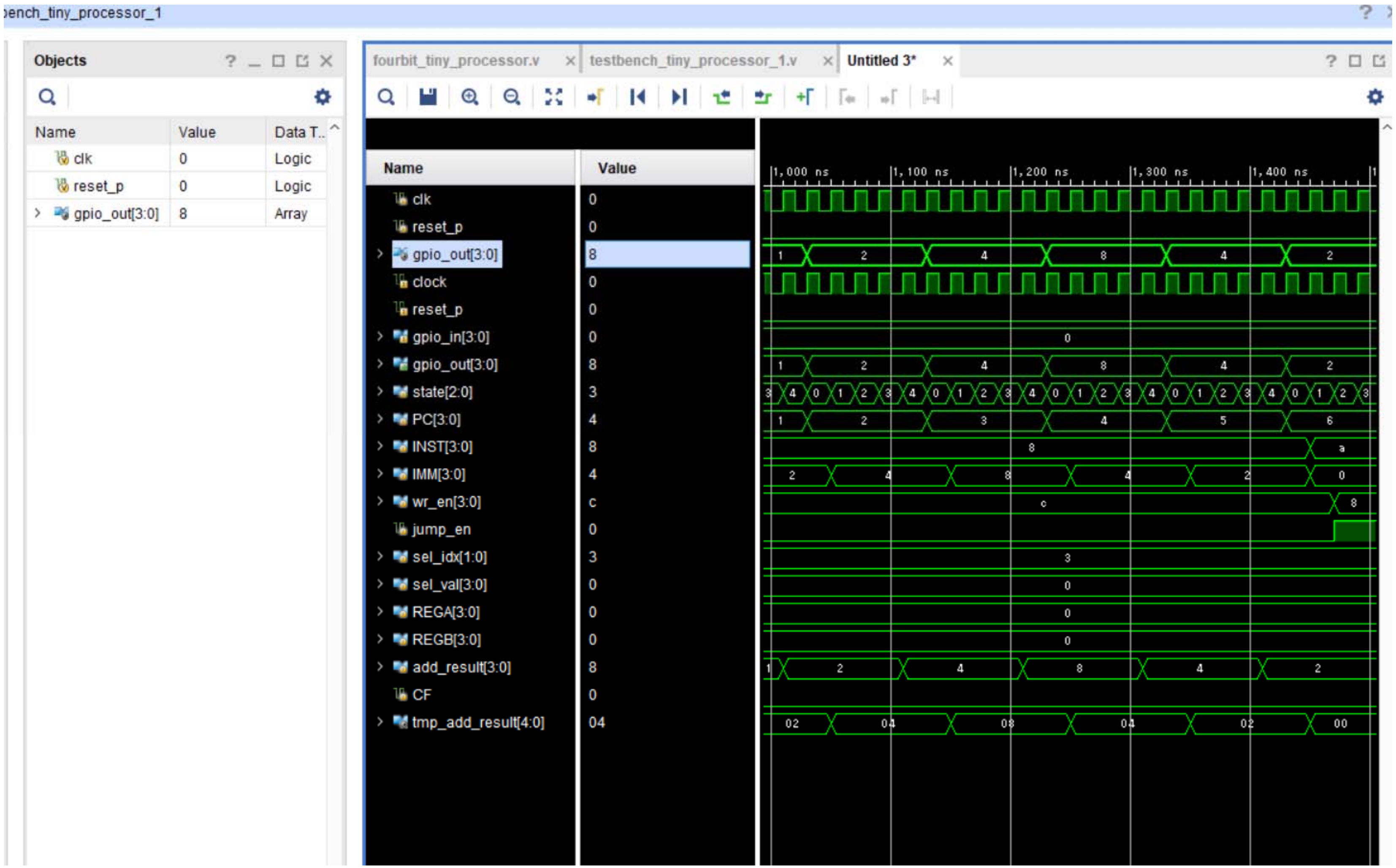
RTL for a Write Back to Register

```
3'b100:begin // write back regs
    state <= 3'b000;
    if( wr_en[0] == 1'b1) REGA <= add_result;
    if( wr_en[1] == 1'b1) REGB <= add_result;
    if( wr_en[2] == 1'b1) gpio_out <= add_result;
    if( jump_en == 1'b1)
        PC <= add_result;
    else
        PC <= PC + 1'b1;
end
```



Behavior Simulation

- See, testbench_tiny_processor.v



Wrapper of a Topmodule

- see, github. Source: topmodule_fourbit_tiny_processor.v

```
module topmodule_fourbit_tiny_processor(
    input sysclk,
    input [0:0]btn,
    output [3:0]led
    );

    reg [24:0]count; // 1sec ~ 8ns(125MHz) * 125,000,000

    always@( posedge sysclk or posedge btn[0])begin
        if( btn[0] == 1'b1) begin
            count <= 25'b0;
        end else begin
            count <= count + 1'b1;
        end
    end

    fourbit_tiny_processor tiny_processor_inst(
        .clock( count[24]),
        .reset_p( btn[0]),
        .gpio_in( 4'b0000),
        .gpio_out( led)
    );

endmodule
```

Example

3min Timer

0x87, // OUT 0111 LED: ○●●●
0x41, // ADD A,0001
0xB1, // JNC 0001
0x41, // ADD A,001
0xB3, // JNC 0011
0x86, // OUT 0110 LED: ○●●○
0x41, // ADD A,0001
0xB6, // JNC 0110
0x41, // ADD A,0001
0xB8, // JNC 1000
0x80, // OUT 0000 LED: ○○○○
0x84, // OUT 0100 LED: ○●○○
0x41, // ADD 0001
0xBA, // JNC 1010
0x88, // OUT 1000 LED: ○●○○
0xAF, // JMP 1111

Conclusion

- Tiny Processor by RTL Design
  - Specification
  - Block Diagram Design
  - C/C++ Behavior Design
  - FSM Design
  - RTL Design & Verification
  - FPGA Implementation
- Software Example
  - LED Controller
  - 3min Timer

Exercise

- (Mandatory) Implement the tiny processor on the Zybo board
- (Optional 1) Extend the memory size to 32 words
- (Optional 2) Add a subtractor as a new operation
- (Optional 3) Develop an assembler for the tiny processor

Send a report to nakahara@ict.e.titech.ac.jp

Deadline is 20^{th}, July, 2018 (At the beginning of the lecture)