

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

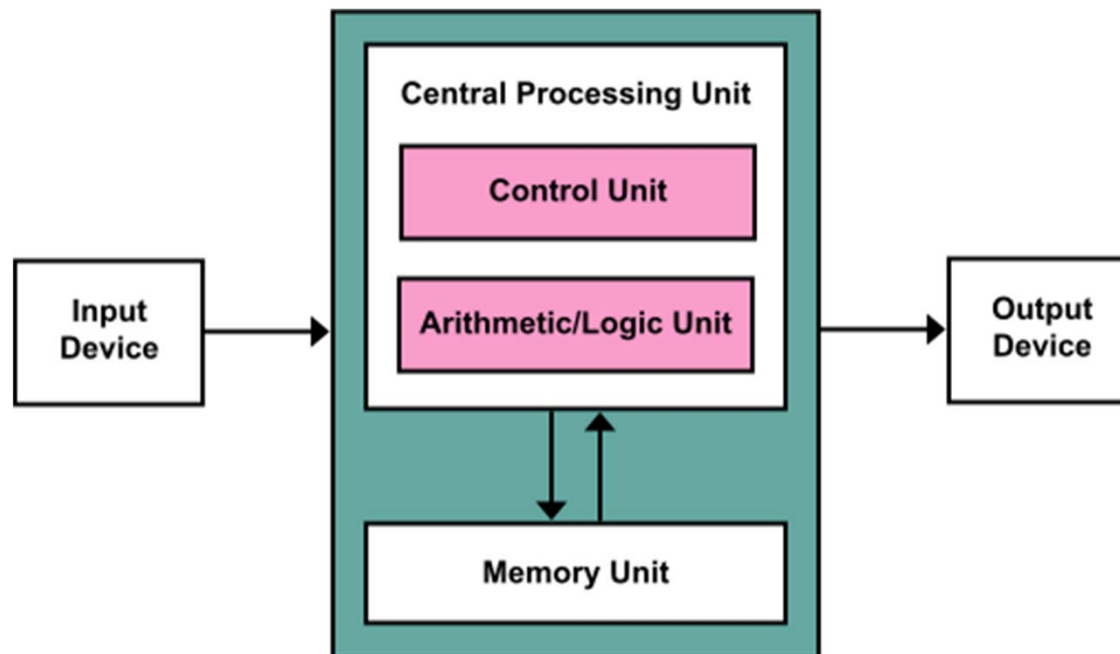
# Processor Specification

# 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

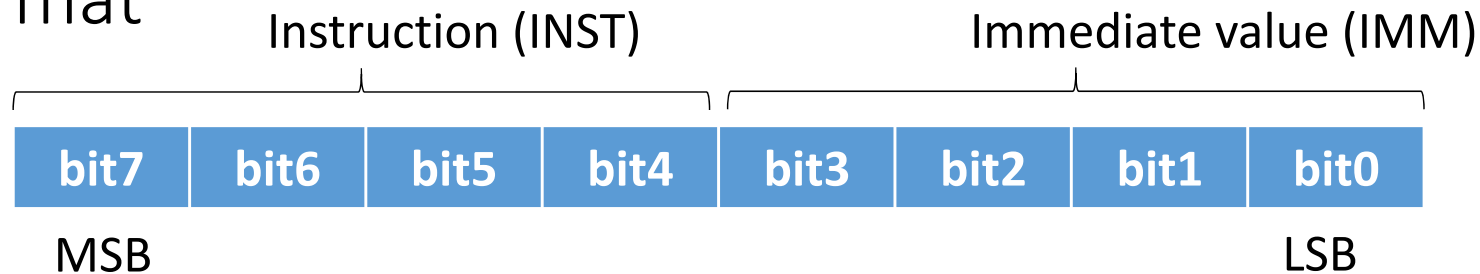


# 4bit Tiny Processor

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

A: Register A

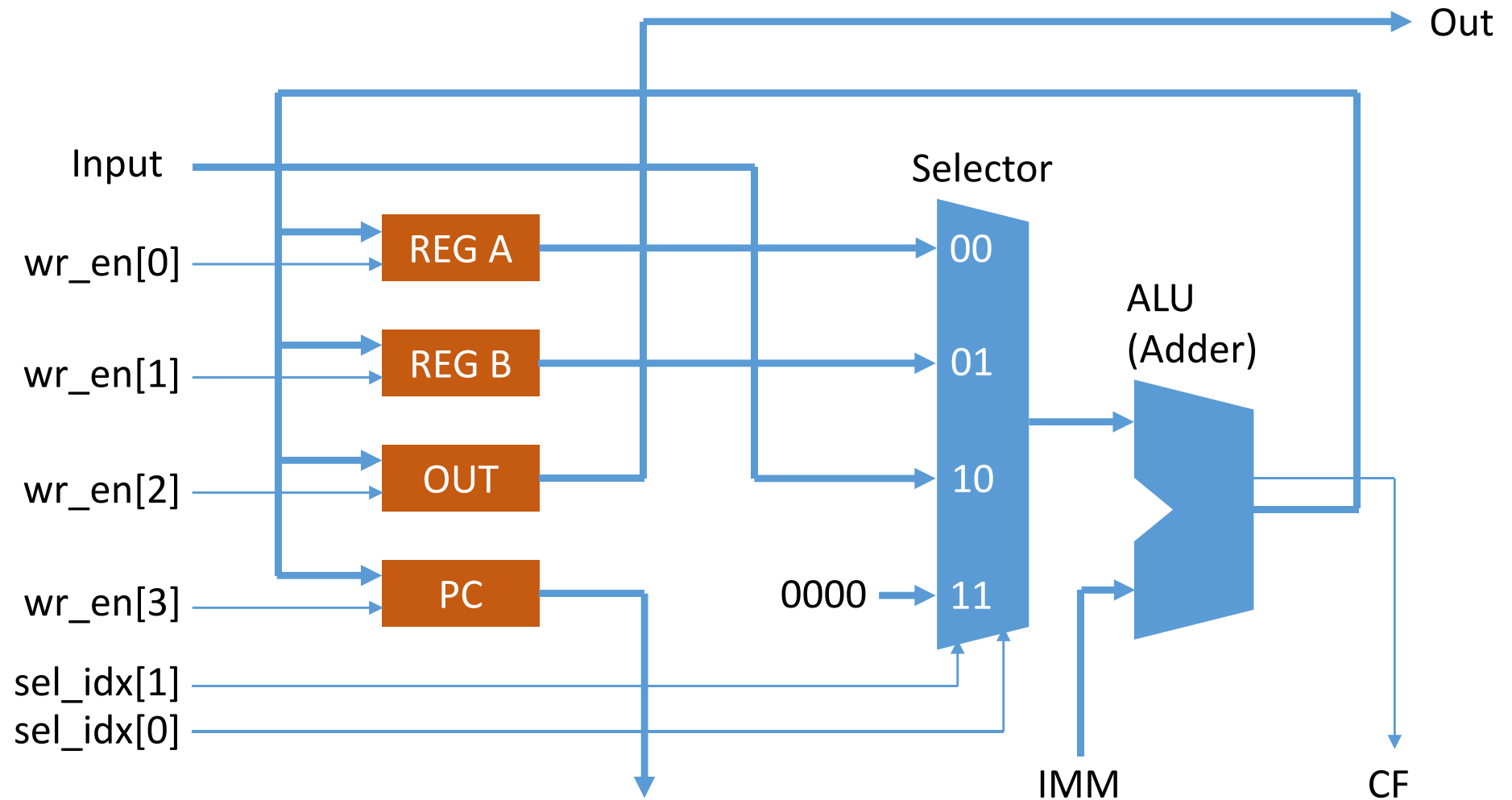
B: Register B

PC: Program Counter

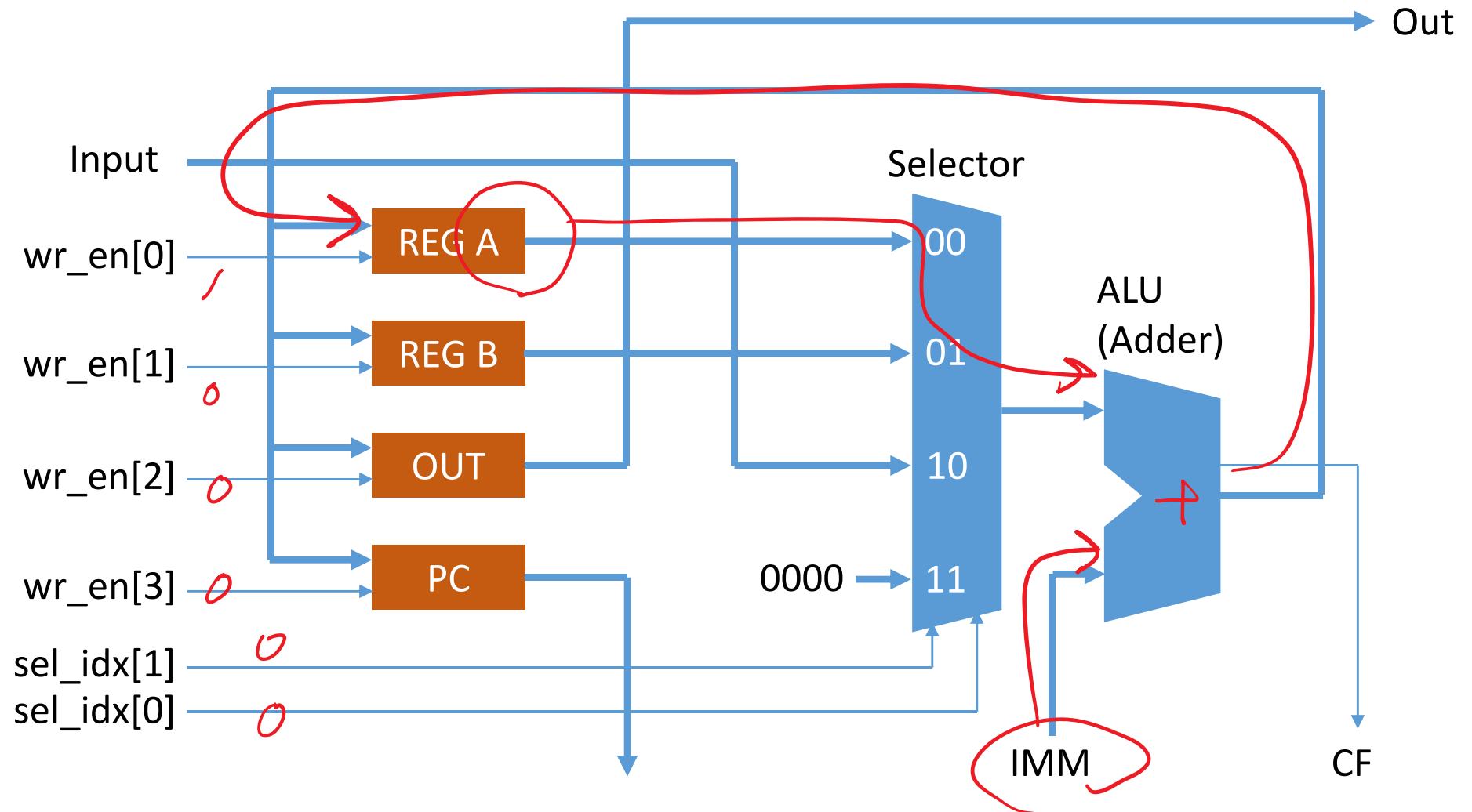
IMM: Immediate value

CF: Carry Flag

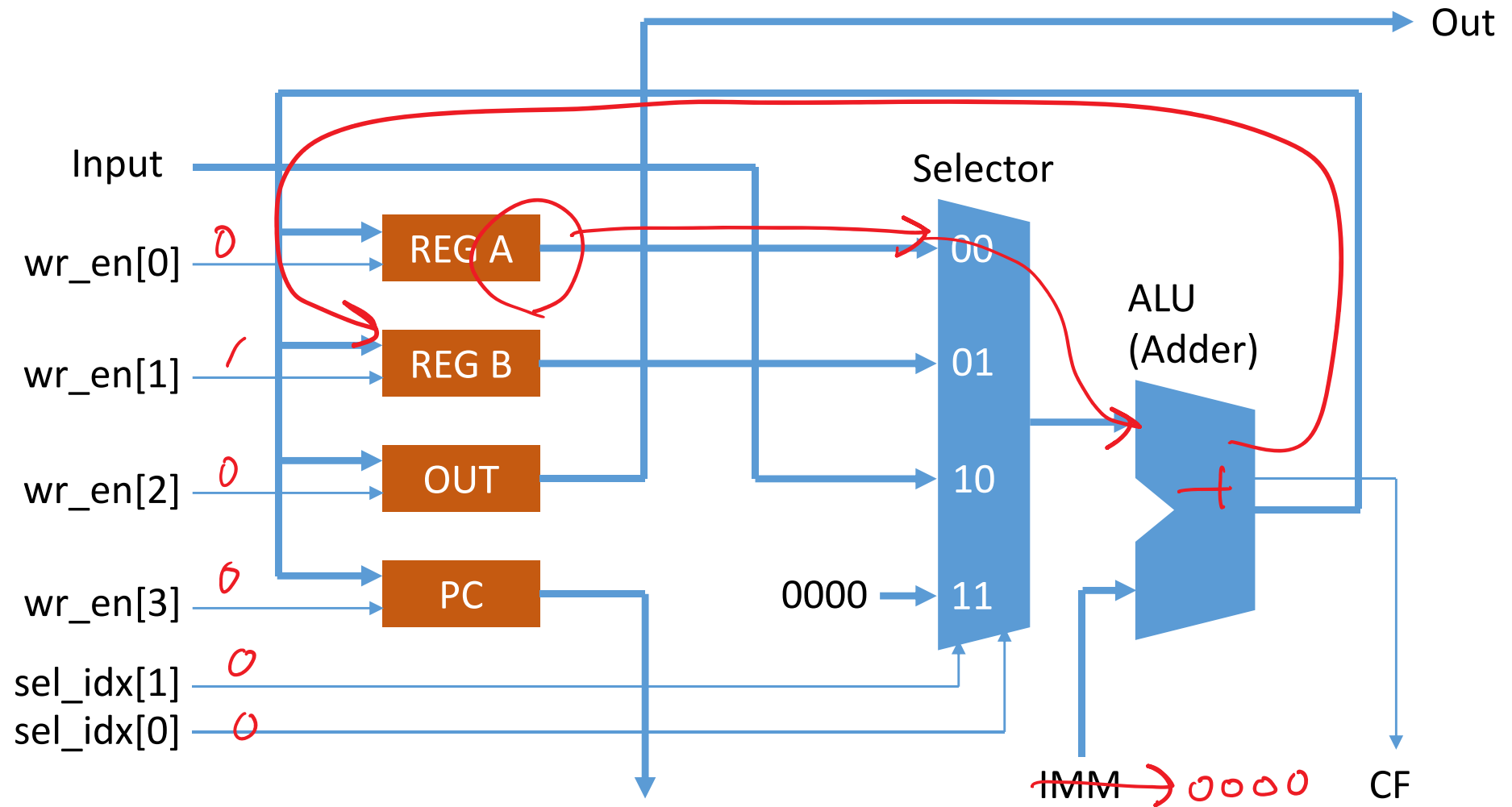
# Data Path (Architecture)



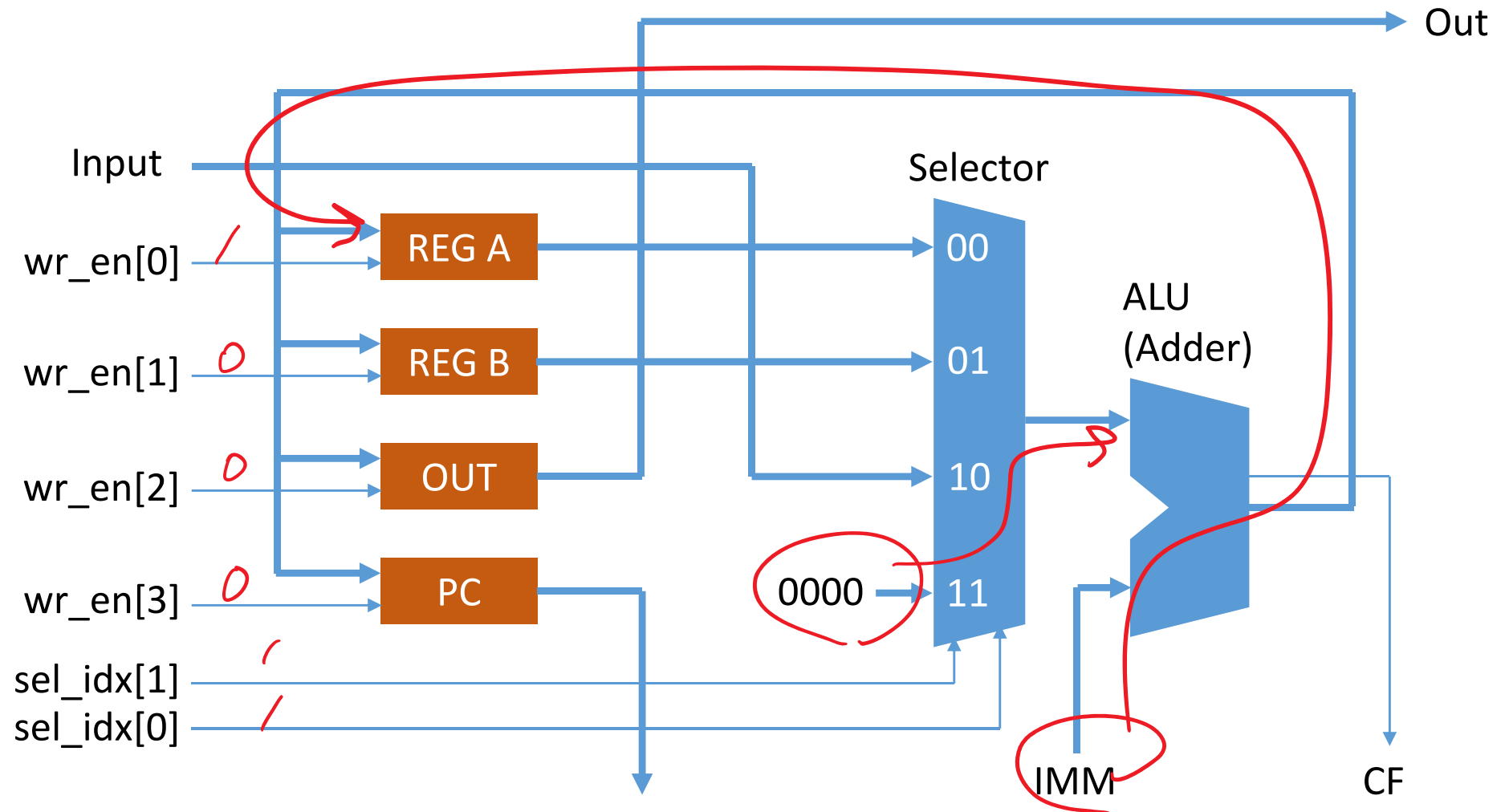
# Add Operation (ADD A,IMM)



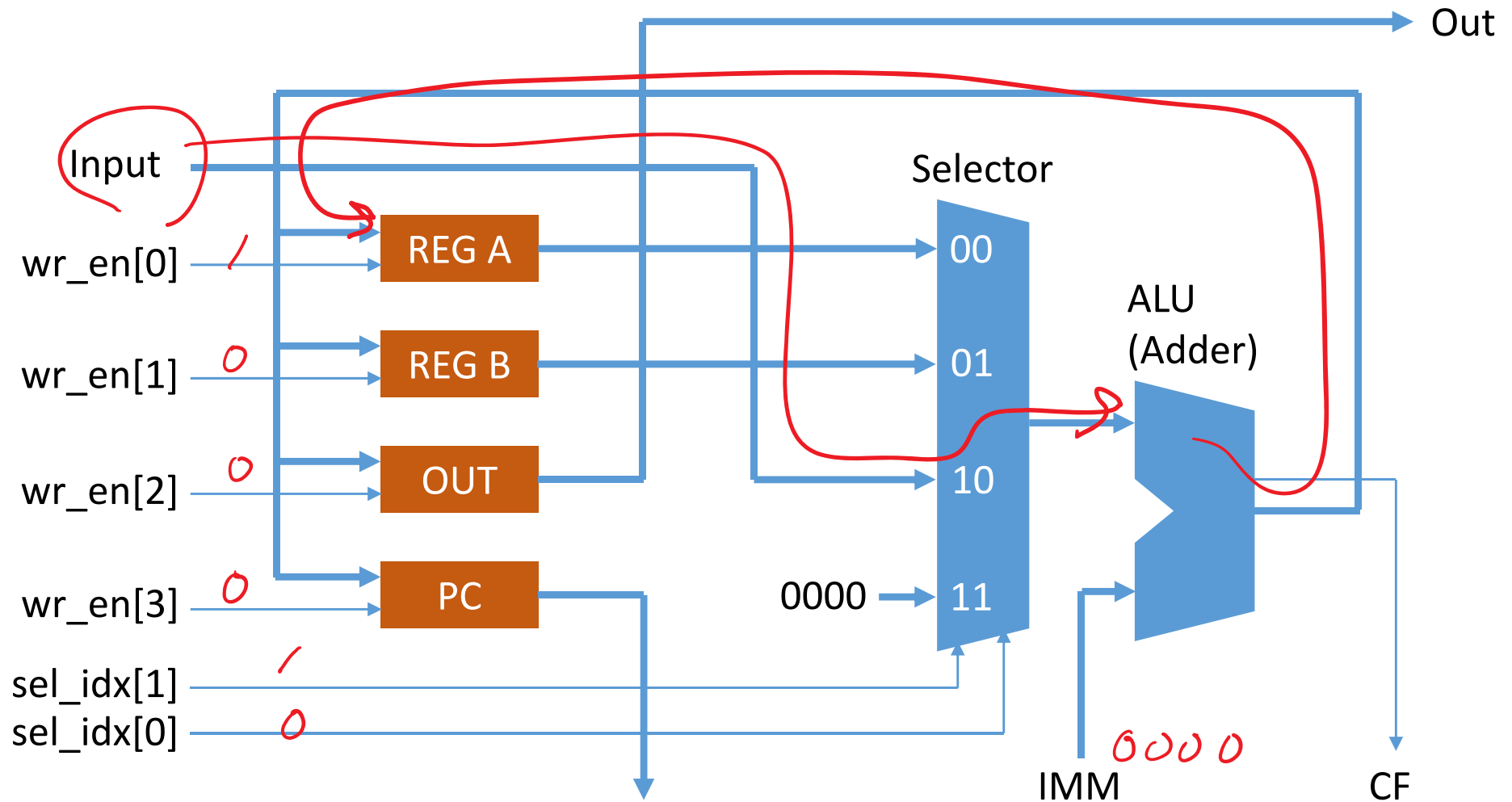
# MOV Operation (MOV A,B)



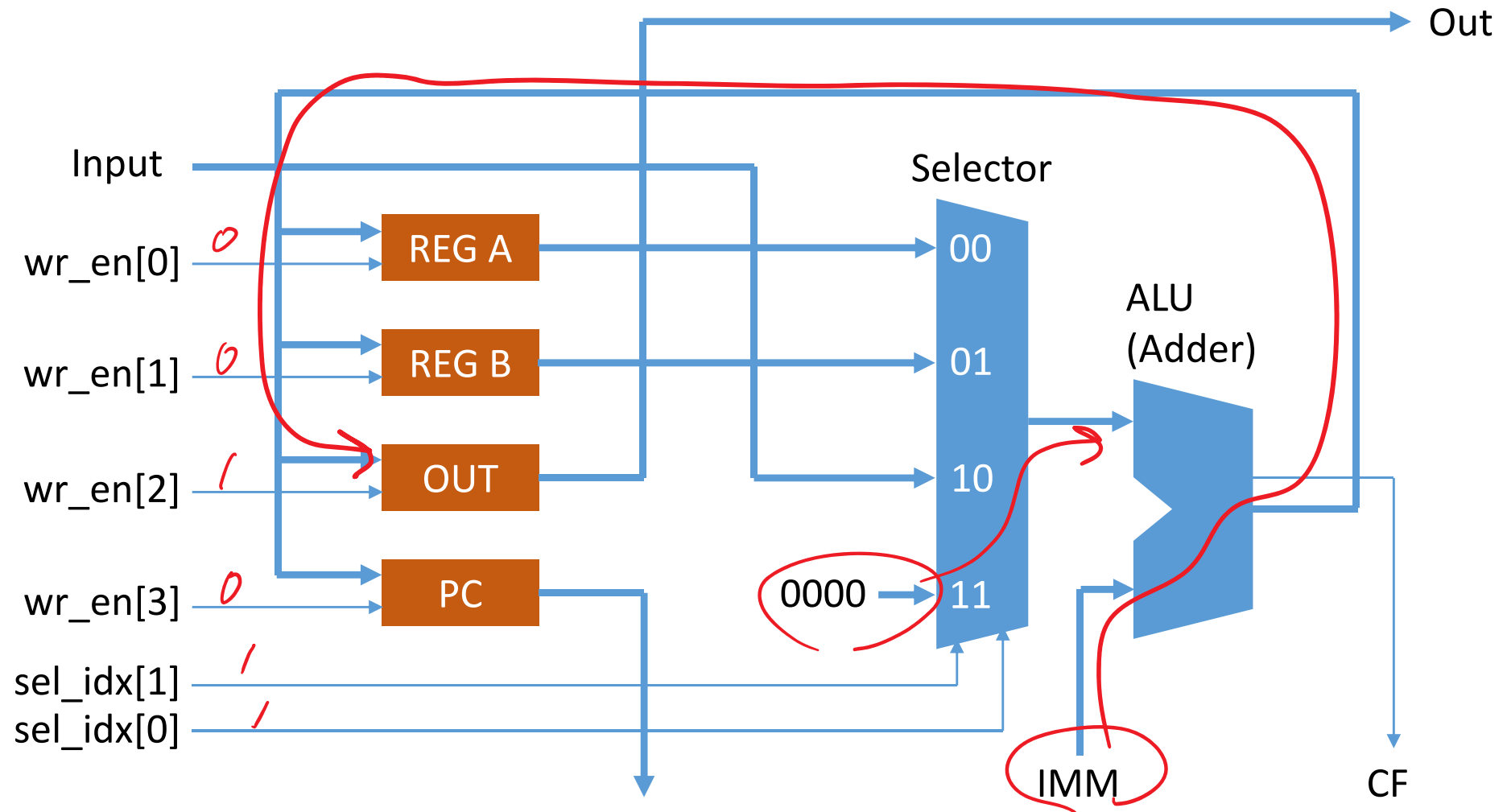
# MOV Operation (MOV A,IMM)



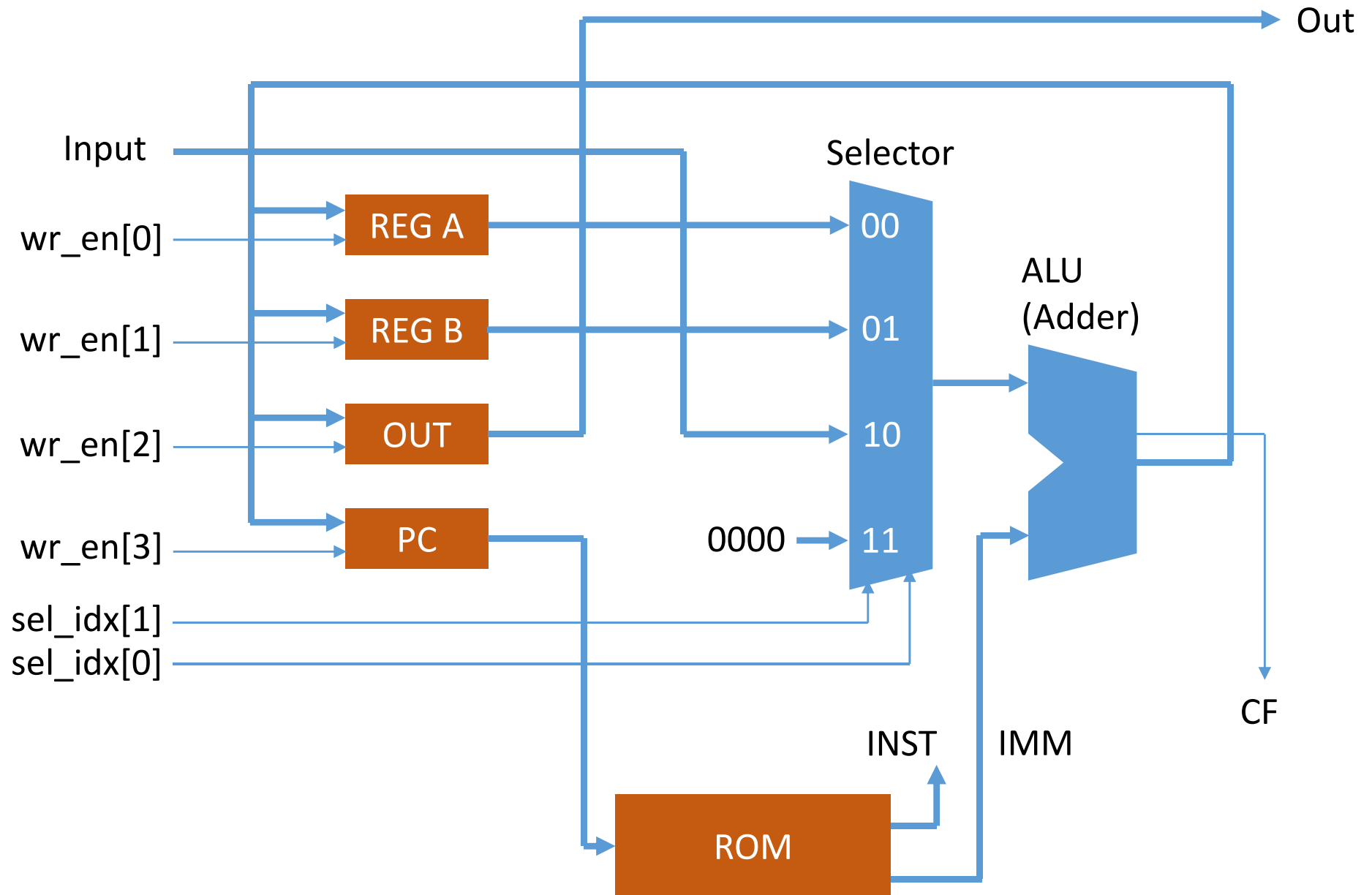
# Read Input Operation (IN A)



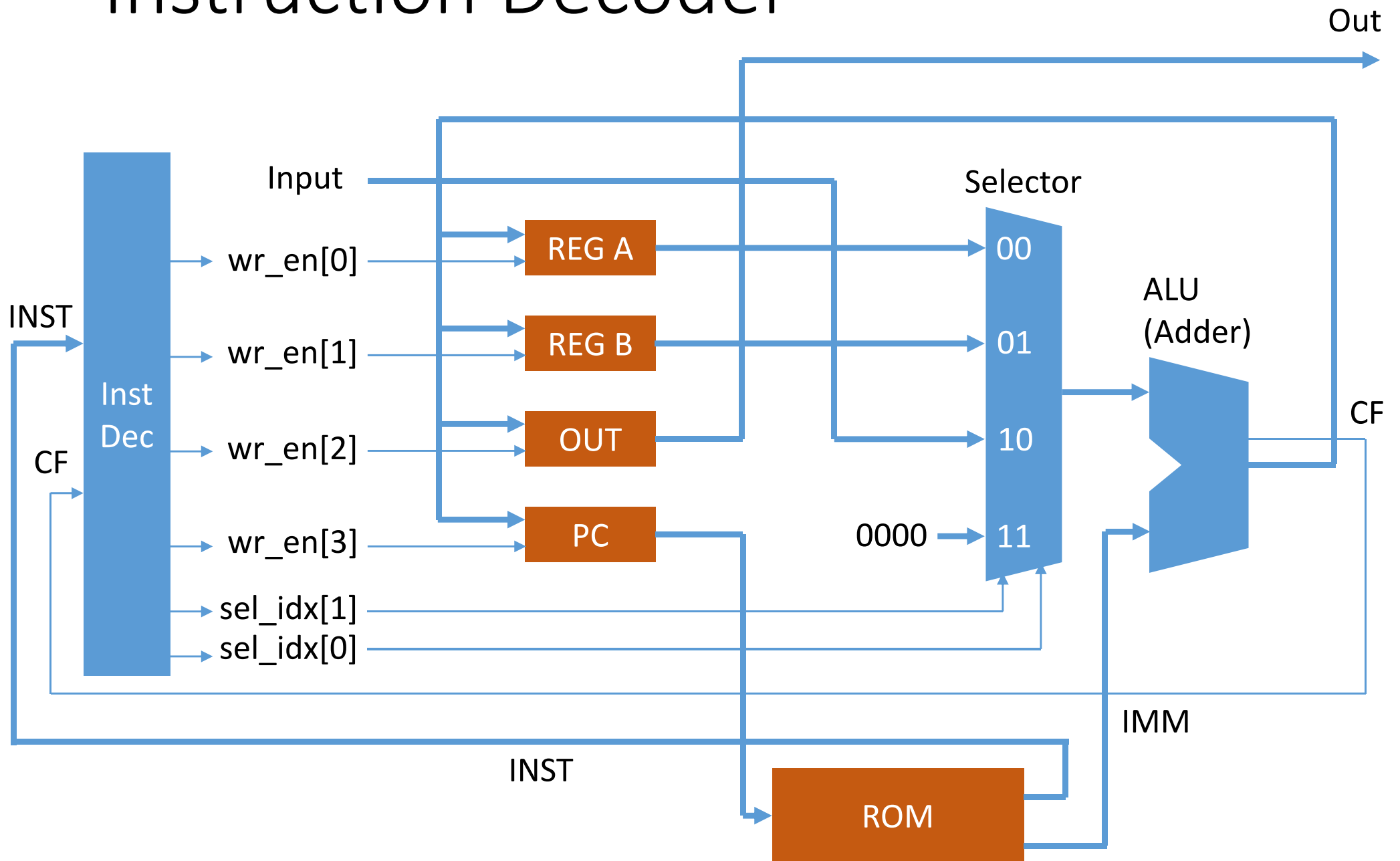
# Write Output Operation (Out IMM)



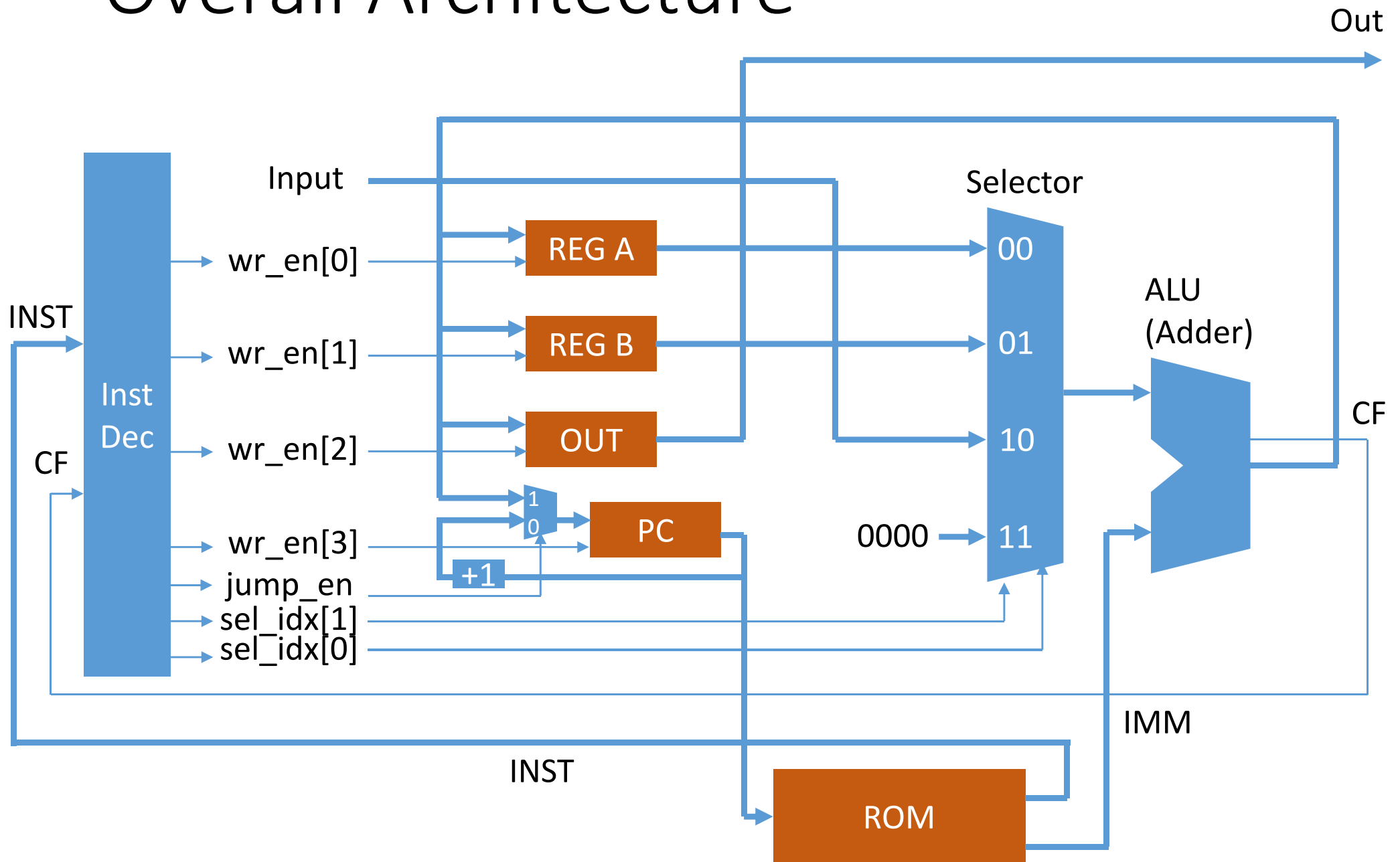
# Program Memory (ROM)



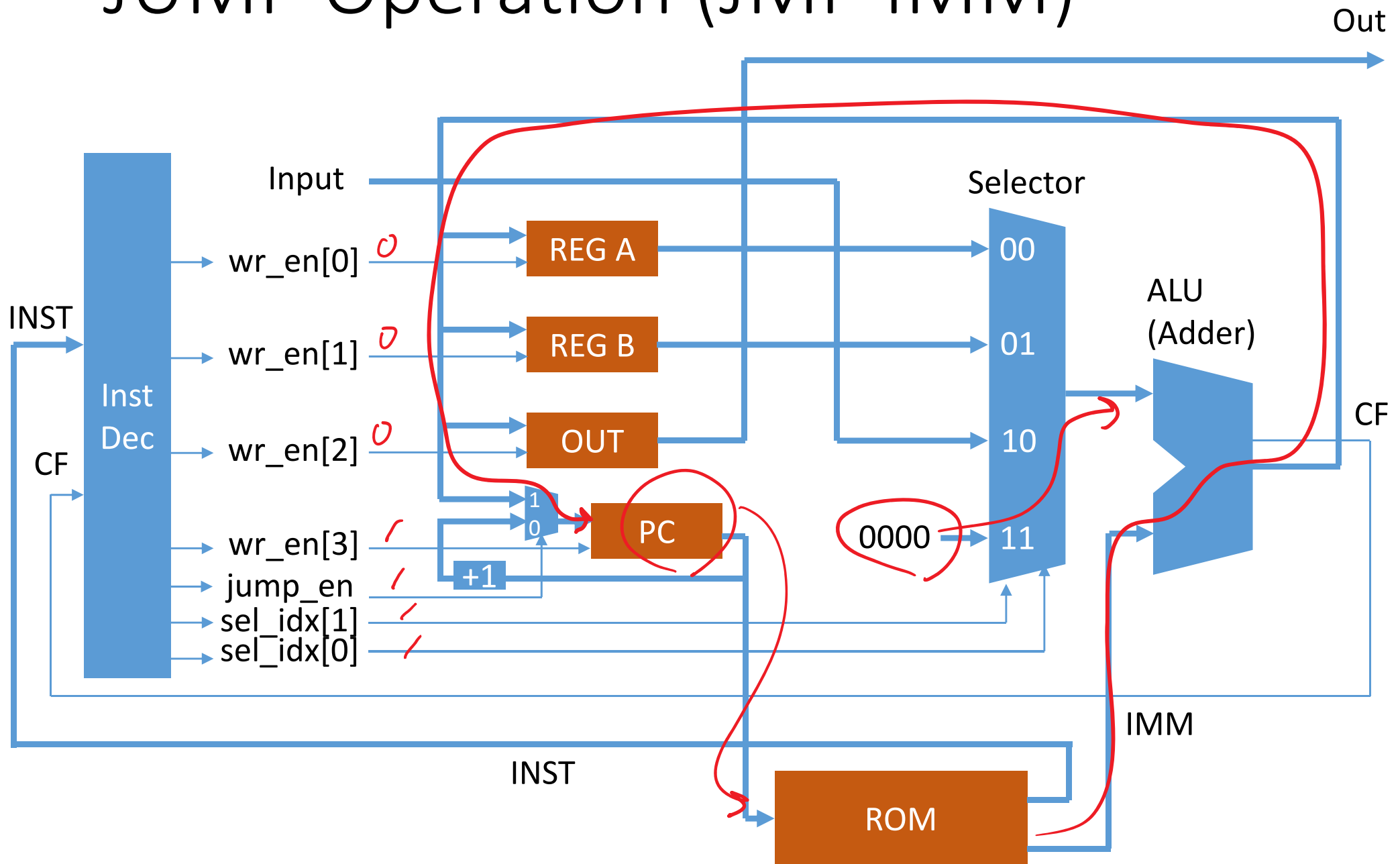
# Instruction Decoder



# Overall Architecture



# JUMP Operation (JMP IMM)

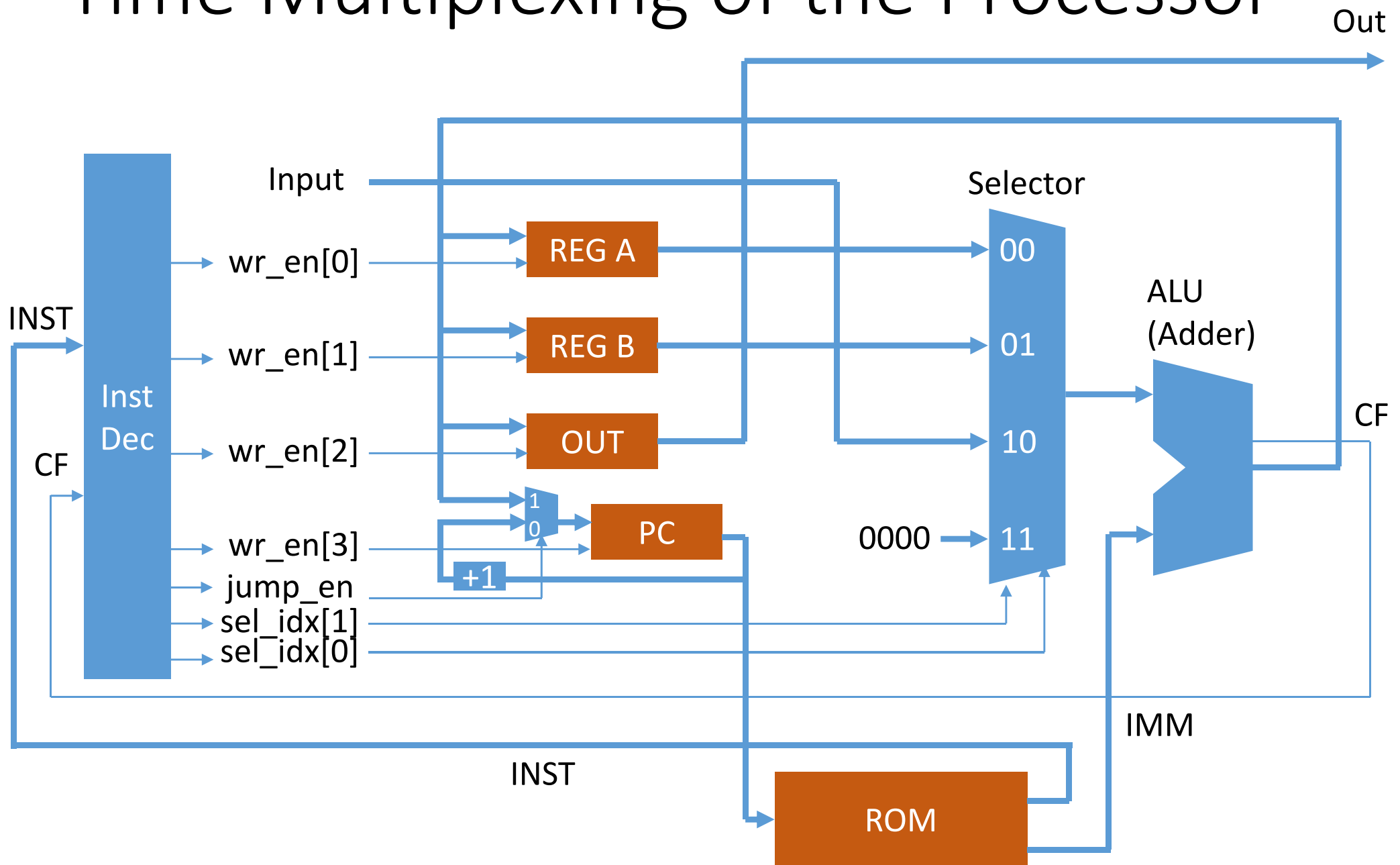


# Truth Table for an Instruction Decoder

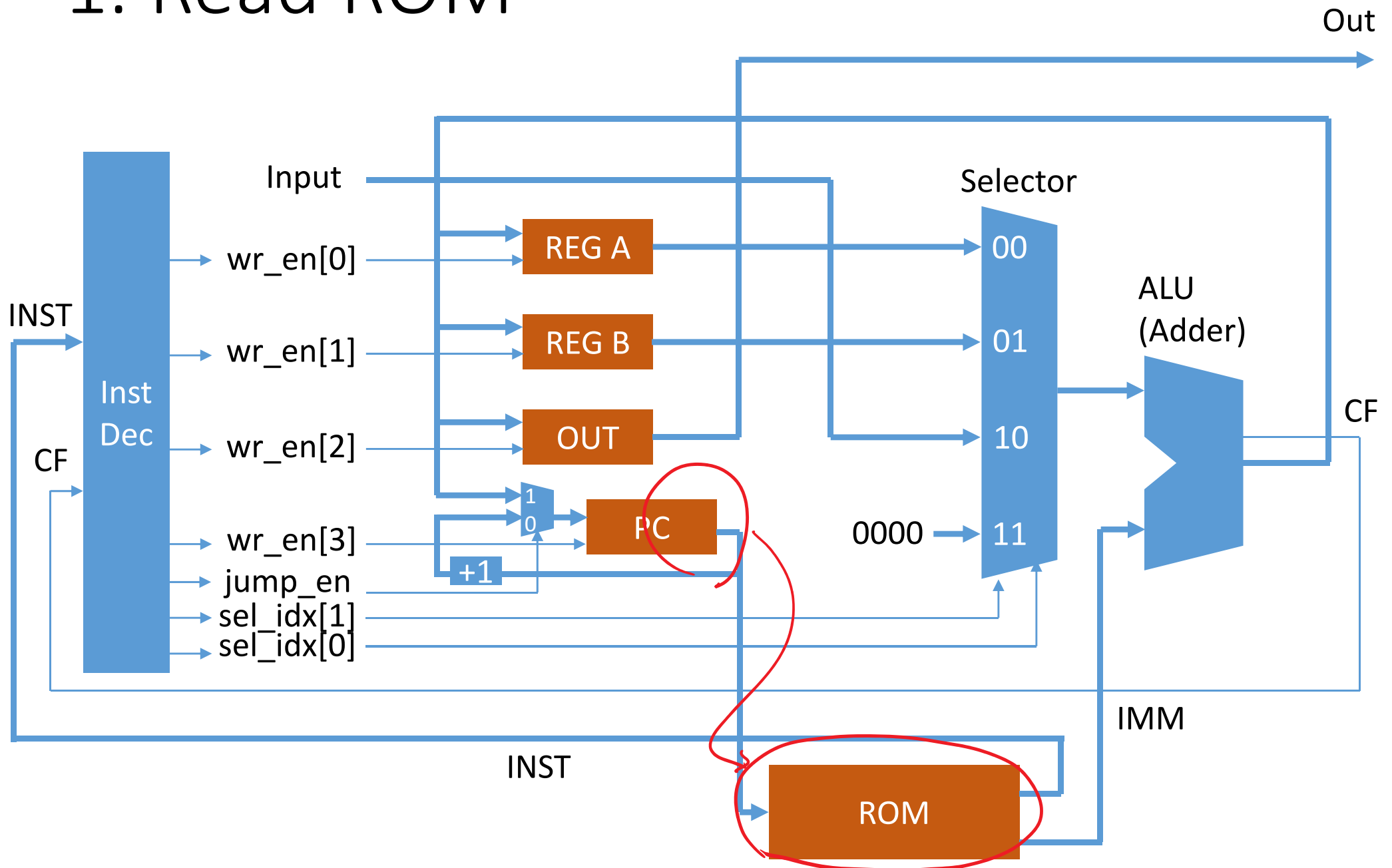
|           | Inst[3:0] | CF | wr_en[3:0] | jump_en | sel_idx[1:0] |
|-----------|-----------|----|------------|---------|--------------|
| MOV A,IMM | 0000      | X  | 1001       | 0       | 11           |
| MOV B,IMM | 0001      | X  | 1010       | 0       | 11           |
| MOV A,B   | 0010      | 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

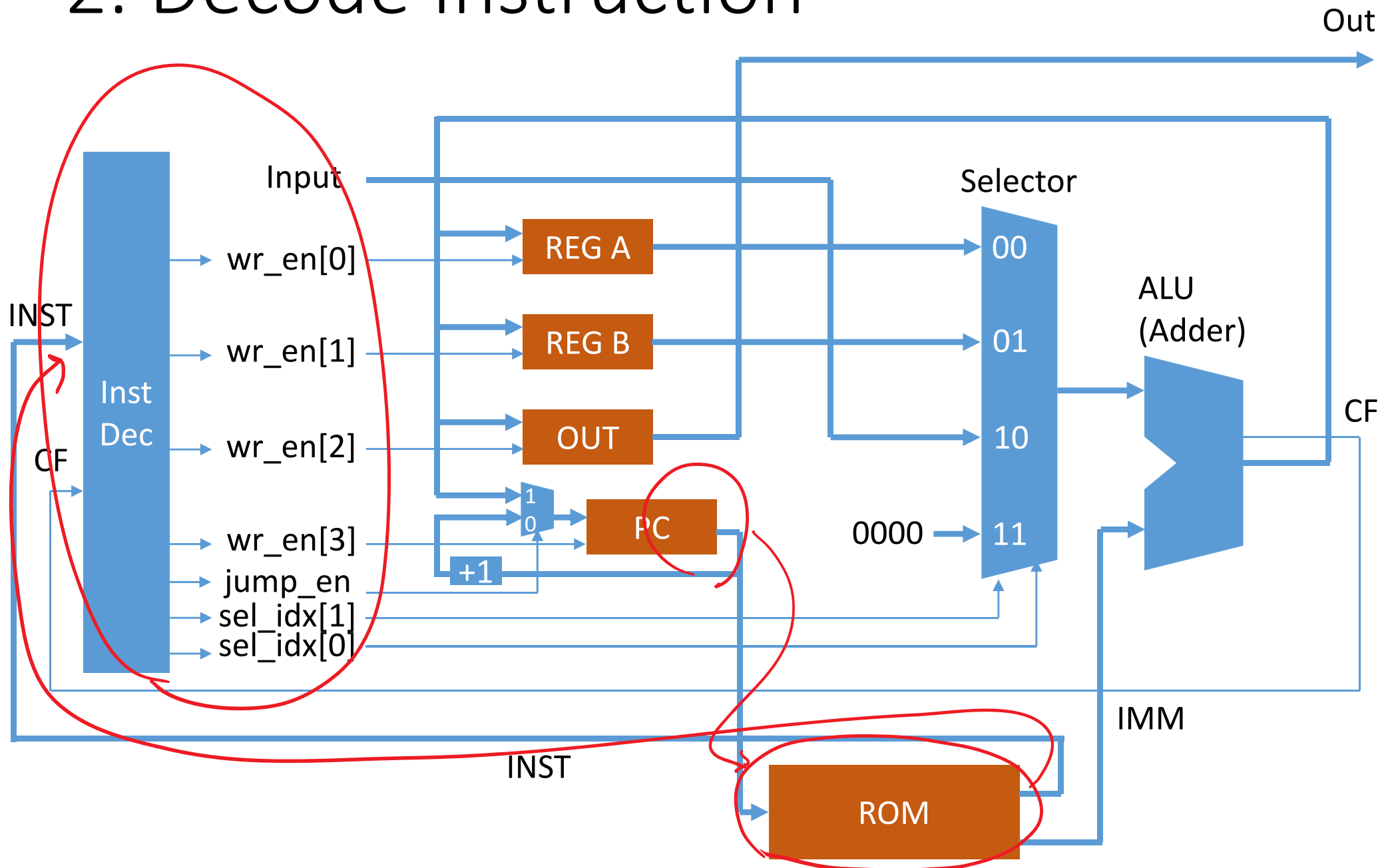
# Time Multiplexing of the Processor



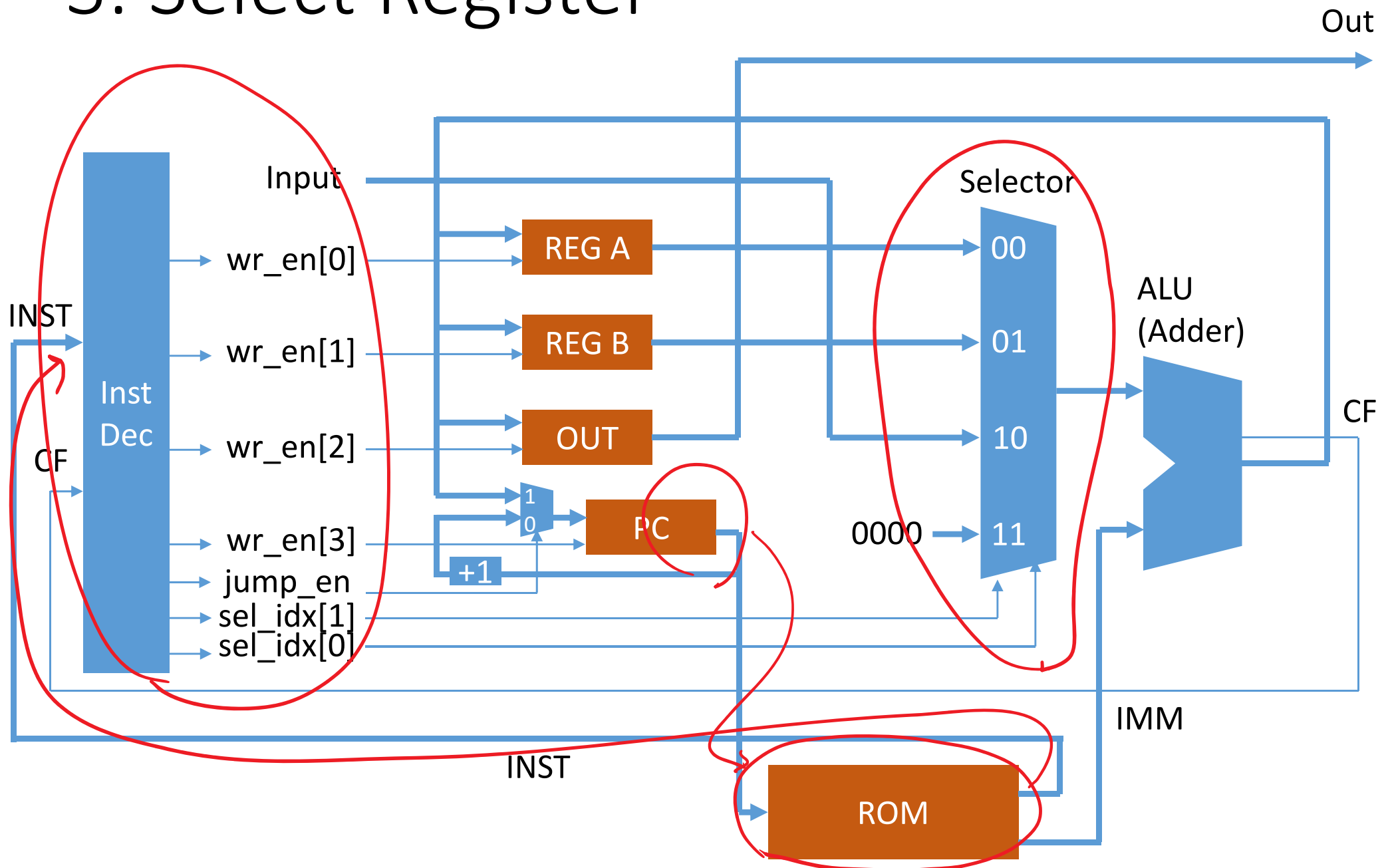
# 1. Read ROM



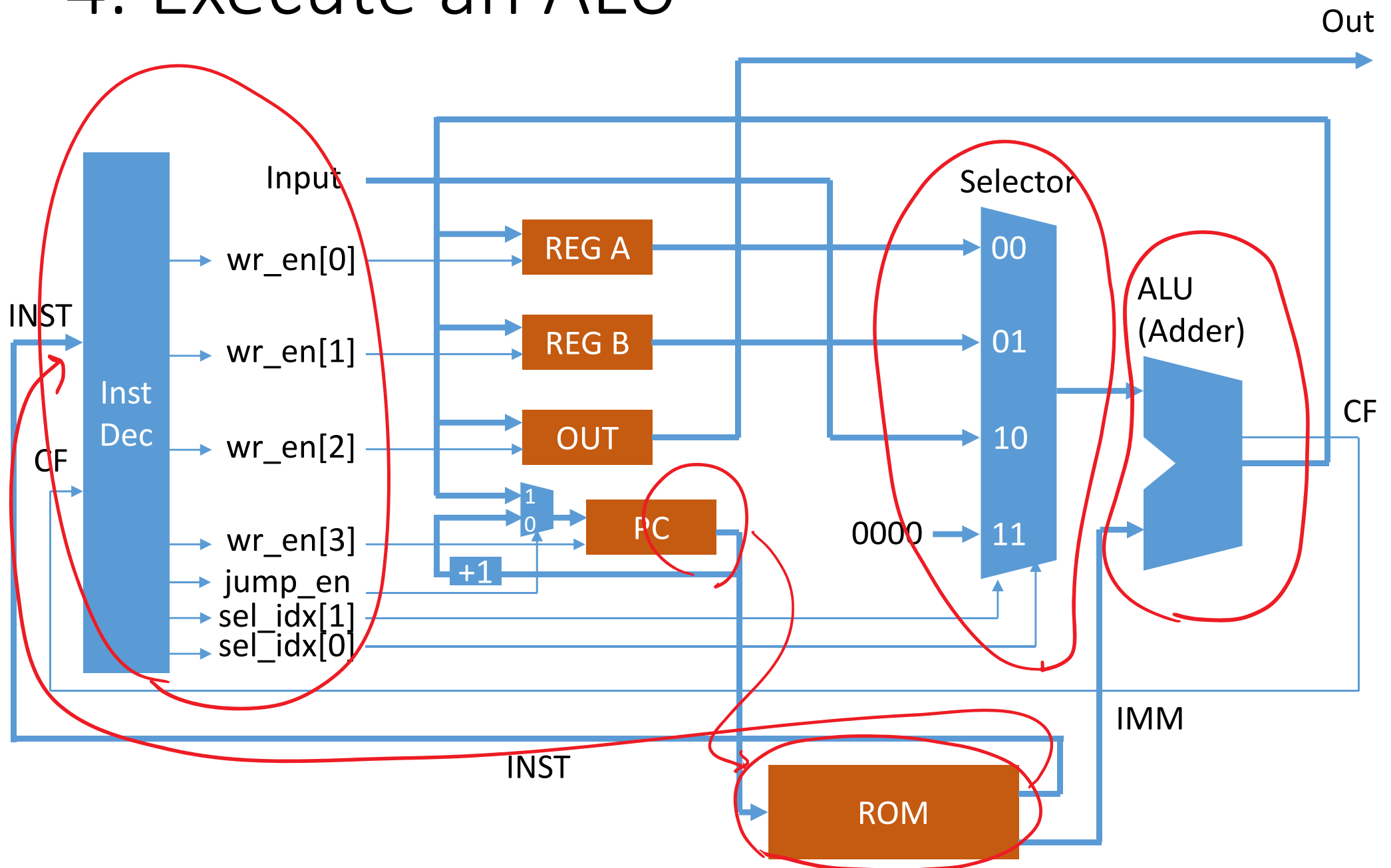
## 2. Decode Instruction



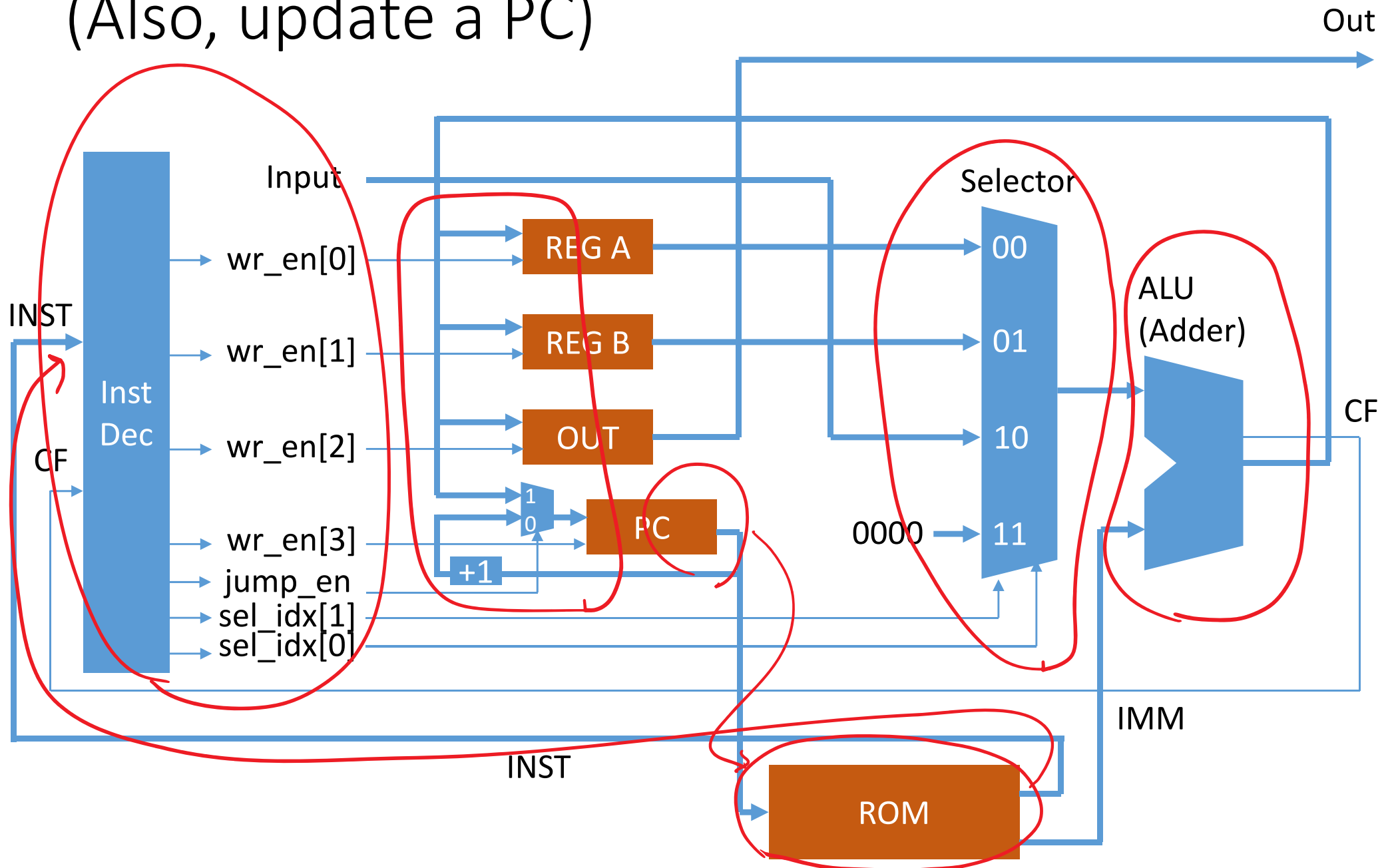
### 3. Select Register



## 4. Execute an ALU



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



# C/C++ Source Code

- see, [https://github.com/HirokiNakahara/FPGA\\_lecture/blob/master/Lec8\\_Tiny\\_Processor/fourbit\\_tiny\\_processor.c](https://github.com/HirokiNakahara/FPGA_lecture/blob/master/Lec8_Tiny_Processor/fourbit_tiny_processor.c)

The screenshot shows the GitHub repository page for HirokiNakahara/FPGA\_lecture. The repository has 9 commits, 1 branch, 0 releases, 1 contributor, and is licensed under MIT. The latest commit is ffea135, made 12 minutes ago. The repository contains the following files:

| File                      | Commit                          | Time           |
|---------------------------|---------------------------------|----------------|
| Lec7_Practical_RTL_design | Create fir_pipe_1.v             | 5 days ago     |
| Lec8_Tiny_Processor       | Create fourbit_tiny_processor.c | 12 minutes ago |
| LICENSE                   | Initial commit                  | 5 days ago     |
| README.md                 | Update README.md                | 5 days ago     |

The README.md file contains the following text:

## FPGA\_lecture

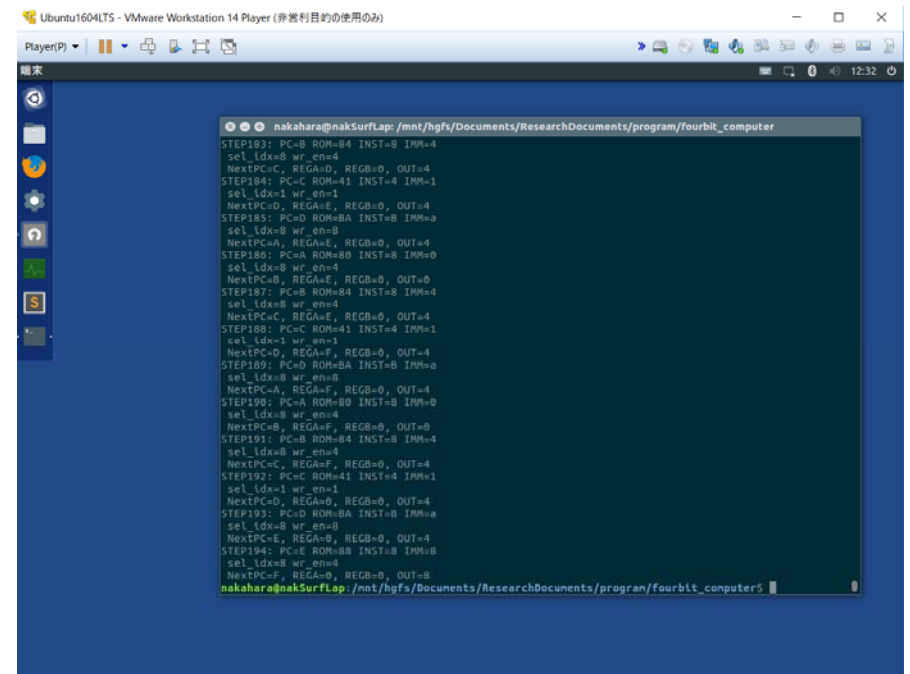
Source Codes for a lecture entitled "Parallel and Reconfigurable VLSI Computing" in Tokyo Tech.

The lecture slides can be downloaded at the Tokyo Tech.

Tokyo Tech Open Course Ware (OCW) for my FPGA lecture: <http://www.ocw.titech.ac.jp/index.php?module=General&action=T0300&GakubuCD=2&GakkaCD=321818&KeiCD=18&course=18&KougiCD=201818041&Nendo=2018&lang=EN&vid=05>

# Case Study: LED Controller

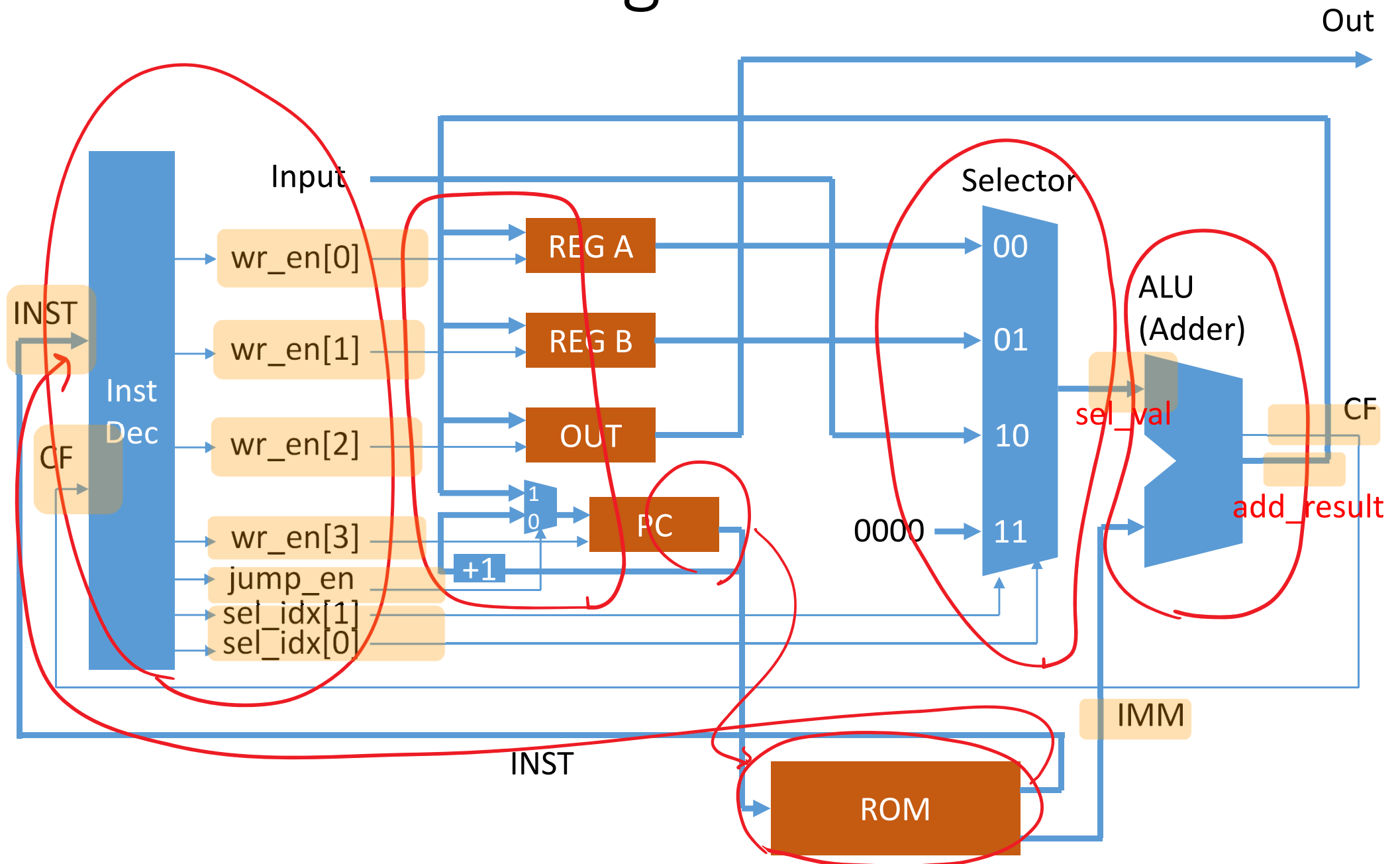
```
0x81, // OUT 0001
0x82, // OUT 0010
0x84, // OUT 0100
0x88, // OUT 1000
0x84, // OUT 0100
0x82, // OUT 0010
0xA0, // JMP to 0000
0x00, // Followings are NOP
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
```

A screenshot of a terminal window titled 'nakahara@nakSurfLap: /mnt/hgfs/Documents/ResearchDocuments/program/fourbit\_computer'. The terminal displays a series of assembly instructions and their execution steps, including PC, ROM, INST, IMM, and register values. The instructions are: STEP183: PC=B ROM=84 INST=8 IMM=4; sel\_ldx=B wr\_en=4; NextPC=C, REGA=D, REGB=0, OUT=4; STEP184: PC=C ROM=41 INST=4 IMM=1; sel\_ldx=1 wr\_en=1; NextPC=D, REGA=E, REGB=0, OUT=4; STEP185: PC=D ROM=BA INST=B IMM=a; sel\_ldx=B wr\_en=8; NextPC=A, REGA=E, REGB=0, OUT=4; STEP186: PC=A ROM=80 INST=8 IMM=0; sel\_ldx=B wr\_en=4; NextPC=B, REGA=E, REGB=0, OUT=4; STEP187: PC=B ROM=84 INST=8 IMM=4; sel\_ldx=B wr\_en=4; NextPC=C, REGA=E, REGB=0, OUT=4; STEP188: PC=C ROM=41 INST=4 IMM=1; sel\_ldx=1 wr\_en=1; NextPC=D, REGA=F, REGB=0, OUT=4; STEP189: PC=D ROM=BA INST=B IMM=a; sel\_ldx=B wr\_en=8; NextPC=A, REGA=F, REGB=0, OUT=4; STEP190: PC=A ROM=80 INST=8 IMM=0; sel\_ldx=B wr\_en=4; NextPC=B, REGA=F, REGB=0, OUT=4; STEP191: PC=B ROM=84 INST=8 IMM=4; sel\_ldx=B wr\_en=4; NextPC=C, REGA=F, REGB=0, OUT=4; STEP192: PC=C ROM=41 INST=4 IMM=1; sel\_ldx=1 wr\_en=1; NextPC=D, REGA=0, REGB=0, OUT=4; STEP193: PC=D ROM=BA INST=B IMM=a; sel\_ldx=B wr\_en=8; NextPC=E, REGA=0, REGB=0, OUT=4; STEP194: PC=E ROM=8B INST=B IMM=B; sel\_ldx=B wr\_en=4; NextPC=F, REGA=0, REGB=0, OUT=0. The prompt at the bottom is 'nakahara@nakSurfLap: /mnt/hgfs/Documents/ResearchDocuments/program/fourbit\_computer\$'.

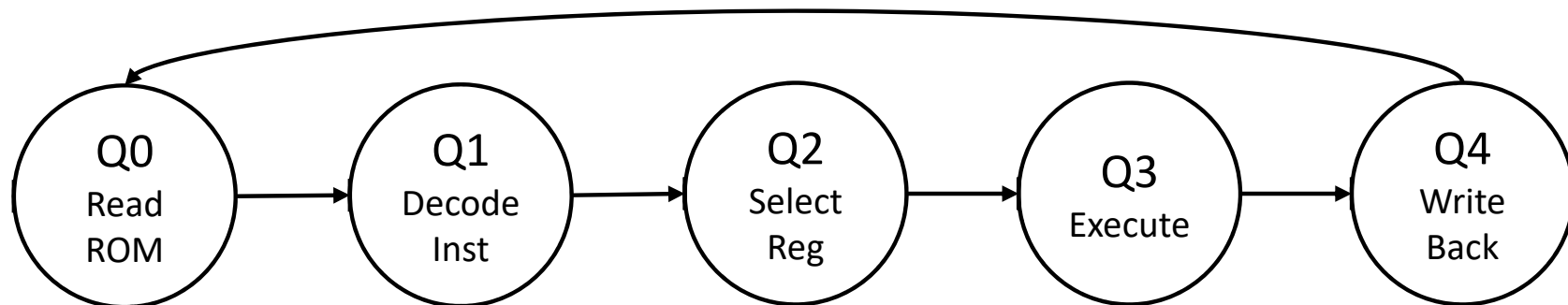
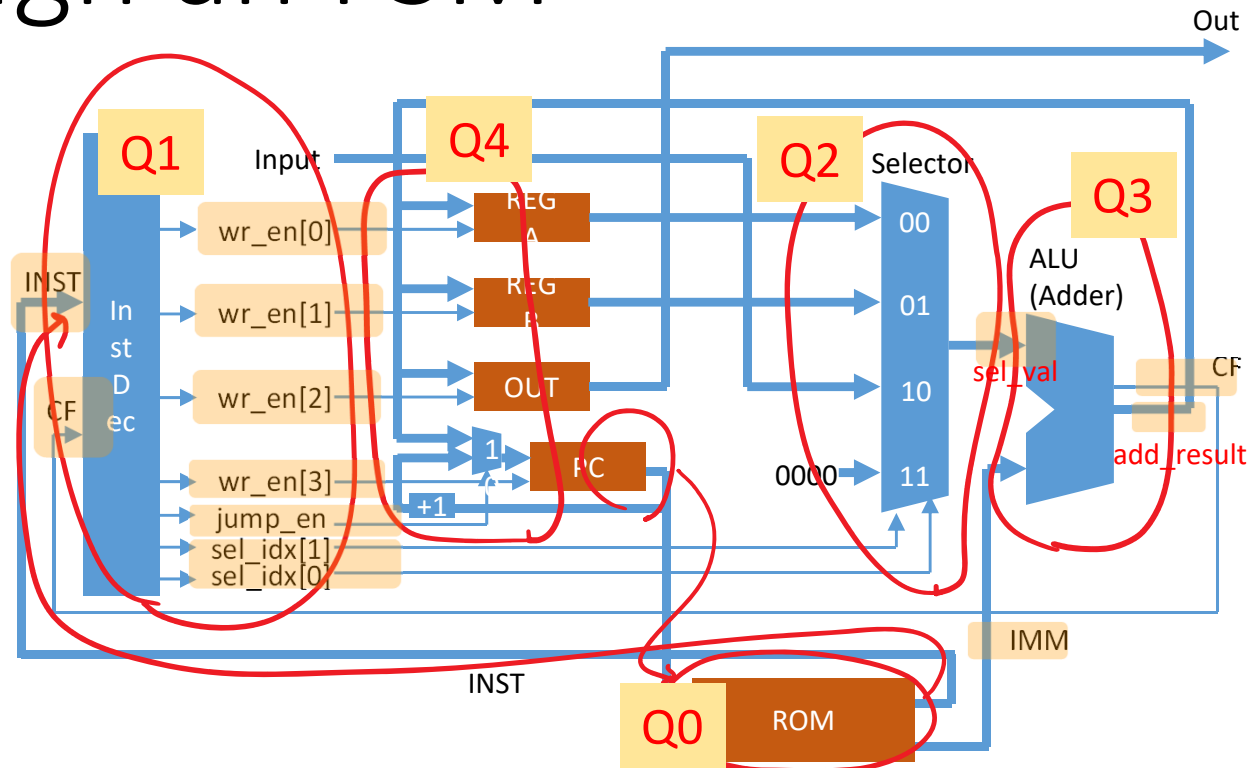
Execute a C behavior code on your PC

# RTL Design

# Insert Virtual Register



# Design an FSM



# RTL Source Code

- see, [https://github.com/HirokiNakahara/FPGA\\_lecture/blob/master/Lec8\\_Tiny\\_Processor/](https://github.com/HirokiNakahara/FPGA_lecture/blob/master/Lec8_Tiny_Processor/)
  - RTL: fourbit\_tiny\_processor.v
  - Testbench: testbench\_tiny\_processor.v

The screenshot shows the GitHub repository page for HirokiNakahara/FPGA\_lecture. The repository is described as "Source Codes for a lecture entitled 'Parallel and Reconfigurable VLSI Computing' in Tokyo Tech." It has 9 commits, 1 branch, 0 releases, and 1 contributor. The repository is licensed under MIT. The commit history shows the following:

| Commit  | Author         | Message                         | Time           |
|---------|----------------|---------------------------------|----------------|
| flea135 | HirokiNakahara | Create fourbit_tiny_processor.c | 12 minutes ago |
| flea135 | HirokiNakahara | Create fir_pipe_1.v             | 5 days ago     |
| flea135 | HirokiNakahara | Create fourbit_tiny_processor.c | 12 minutes ago |
| flea135 | HirokiNakahara | Initial commit                  | 5 days ago     |
| flea135 | HirokiNakahara | Update README.md                | 5 days ago     |

The README.md file contains the following text:

## FPGA\_lecture

Source Codes for a lecture entitled "Parallel and Reconfigurable VLSI Computing" in Tokyo Tech.

The lecture slides can be downloaded at the Tokyo Tech.

Tokyo Tech Open Course Ware (OCW) for my FPGA lecture: <http://www.ocw.titech.ac.jp/index.php?module=General&action=T0300&GakubuCD=2&GakkaCD=321818&KeiCD=18&course=18&KougiCD=201818041&Nendo=2018&lang=EN&vid=05>

# RTL Code for Reading ROM

```

60     case( state)
61     3'b000:begin // read ROM
62         state <= 3'b001;
63         {INST, IMM} <= ROM( PC);
64     end

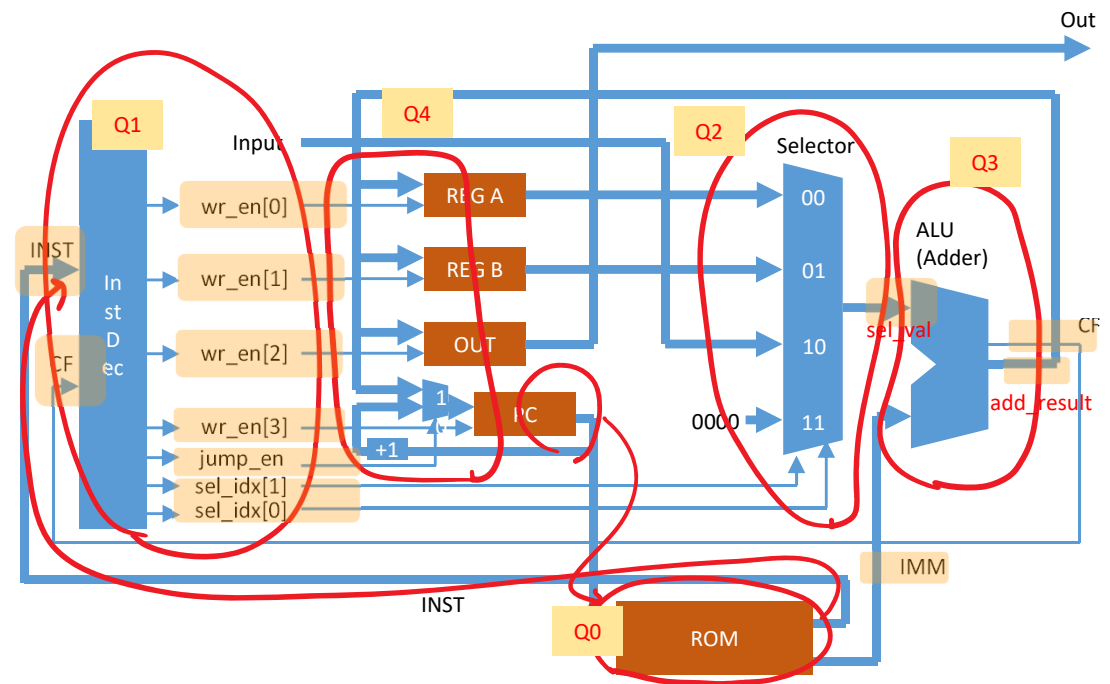
```

⋮

```

98     function [7:0]ROM( input [3:0]address);
99     begin
100         case( address)
101         4'b0000: ROM = 8'h81;
102         4'b0001: ROM = 8'h82;
103         4'b0010: ROM = 8'h84;
104         4'b0011: ROM = 8'h88;
105         4'b0100: ROM = 8'h84;
106         4'b0101: ROM = 8'h82;
107         4'b0110: ROM = 8'hA0;
108         4'b0111: ROM = 8'h00;
109         4'b1000: ROM = 8'h00;
110         4'b1001: ROM = 8'h00;
111         4'b1010: ROM = 8'h00;
112         4'b1011: ROM = 8'h00;
113         4'b1100: ROM = 8'h00;
114         4'b1101: ROM = 8'h00;
115         4'b1110: ROM = 8'h00;
116         4'b1111: ROM = 8'h00;
117     endcase
118     end
119 endfunction

```



# RTL for an Instruction Decoder

```

65 3'b001:begin // decode inst
66     state <= 3'b010;
67     {wr_en, jump_en, sel_idx} <= INST_DEC( INST, CF);
68 end

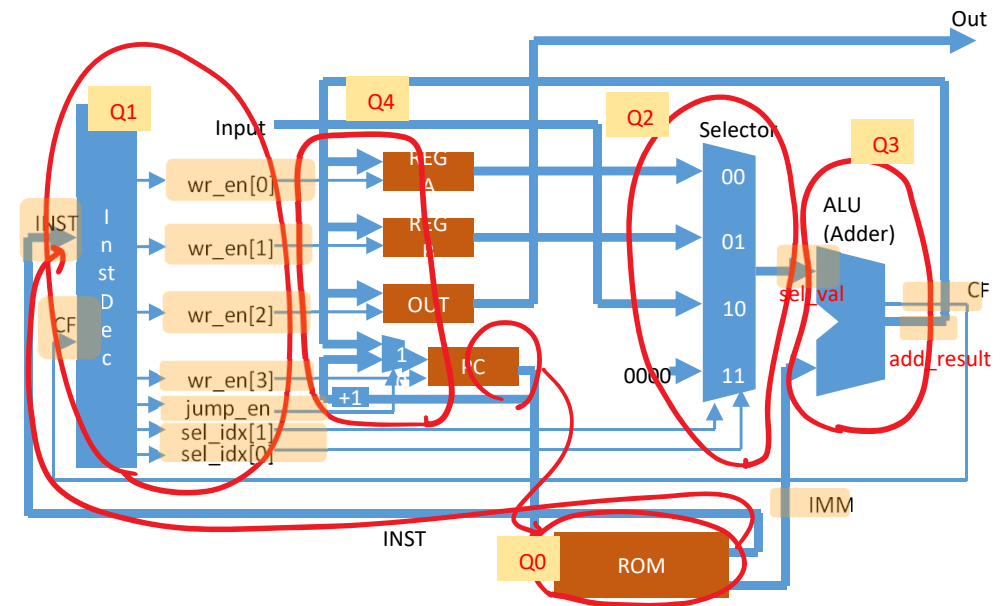
```

⋮

```

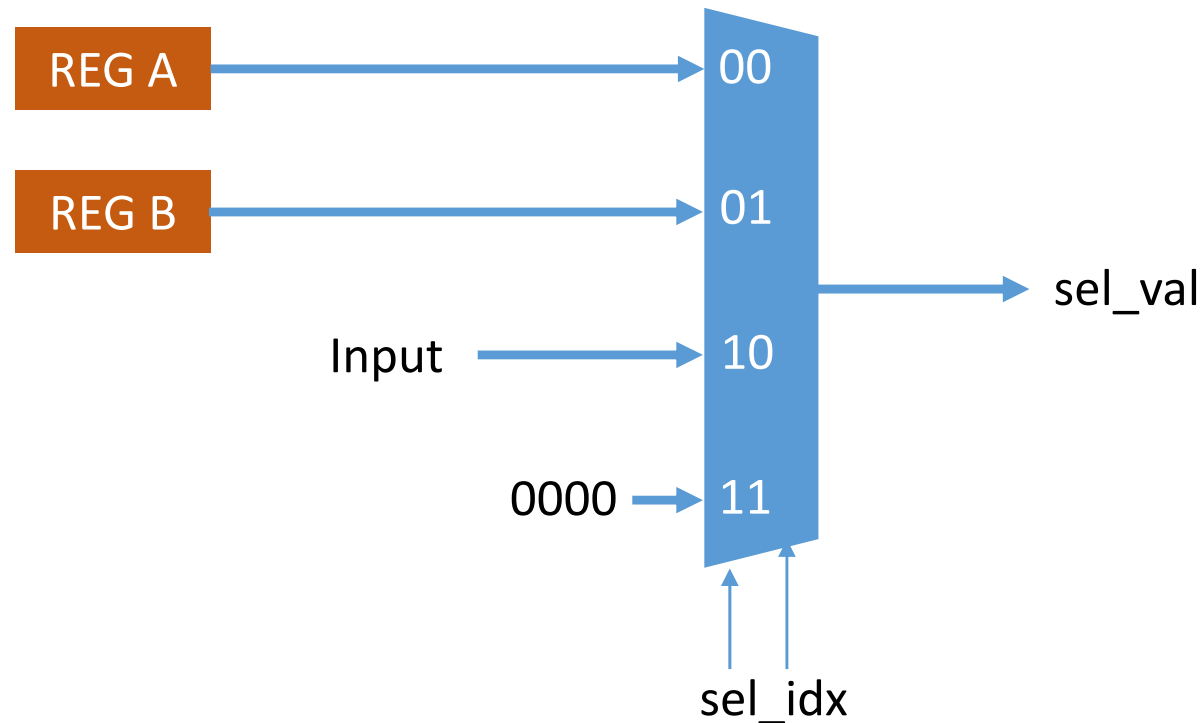
140 function [6:0]INST_DEC( input [3:0]inst, input CF);
141 begin "case" accept a don't care "x"
142     casex( {inst, CF})
143         5'b0000_x: INST_DEC = 7'b1001_0_11;
144         5'b0001_x: INST_DEC = 7'b1010_0_11;
145         5'b0010_x: INST_DEC = 7'b1001_0_01;
146         5'b0011_x: INST_DEC = 7'b1010_0_00;
147         5'b0100_x: INST_DEC = 7'b1001_0_00;
148         5'b0101_x: INST_DEC = 7'b1001_0_01;
149         5'b0110_x: INST_DEC = 7'b1001_0_10;
150         5'b0111_x: INST_DEC = 7'b1010_0_10;
151         5'b1000_x: INST_DEC = 7'b1100_0_11;
152         5'b1001_x: INST_DEC = 7'b1100_0_01;
153         5'b1010_x: INST_DEC = 7'b1000_1_11;
154         5'b1011_0: INST_DEC = 7'b1000_1_11;
155         5'b1011_1: INST_DEC = 7'b1000_0_11;
156         default: INST_DEC = 7'b0000_0_00;
157     endcase
158 end
159 endfunction

```



# RTL for a Register Selector

```
69 3'b010:begin // select reg
70   state <= 3'b011;
71   sel_val <= sel_idx == 2'b00 ? REGA :
72               sel_idx == 2'b01 ? REGB :
73               sel_idx == 2'b10 ? gpio_in :
74               4'b0000;
75 end
```

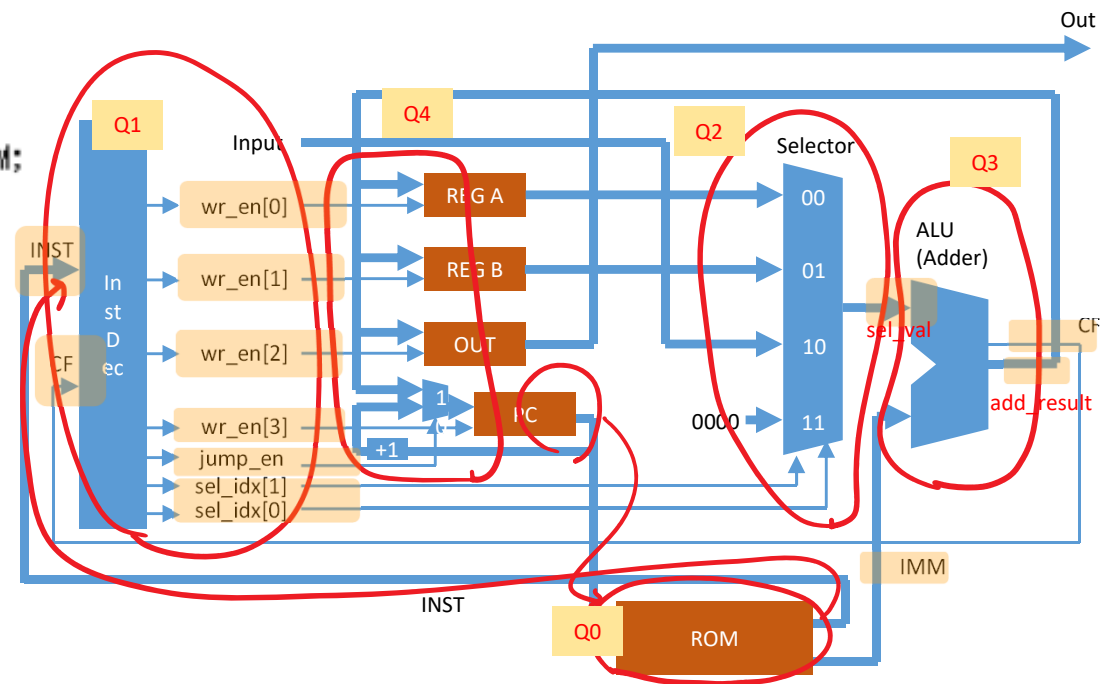


# RTL for an ALU (4-bit Adder with CF)

```
76 3'b011:begin // execute
77   state <= 3'b100;
78   add_result <= tmp_add_result[3:0];
79   CF <= tmp_add_result[4];
80 end
```

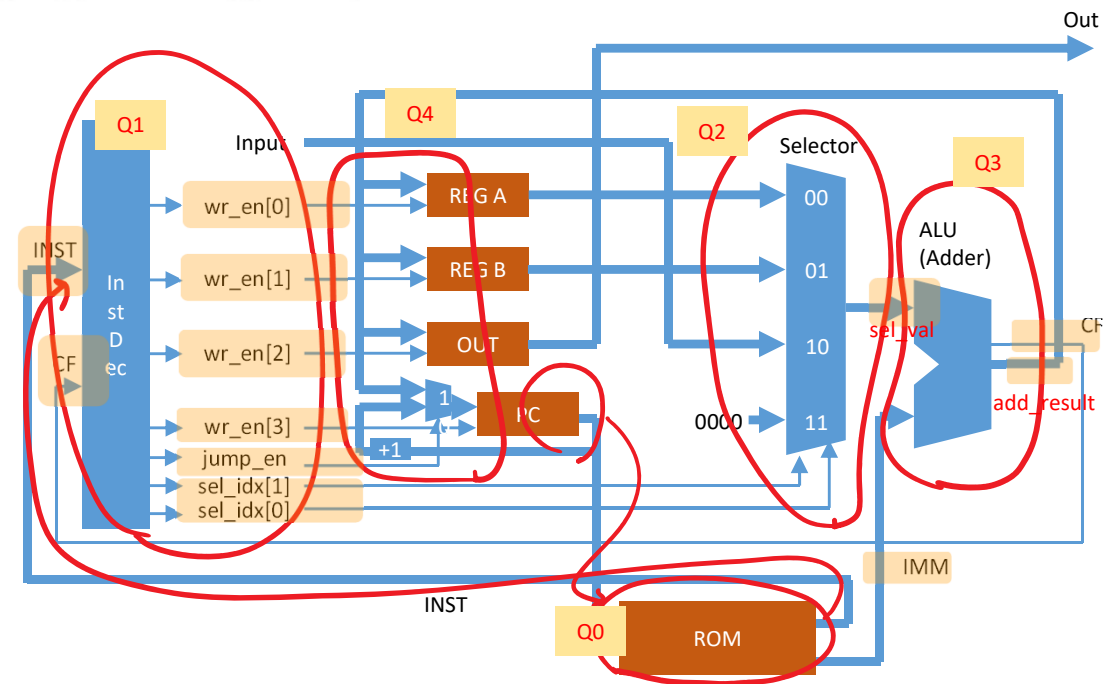
⋮

```
160
161 assign tmp_add_result = sel_val + IMM;
162
```



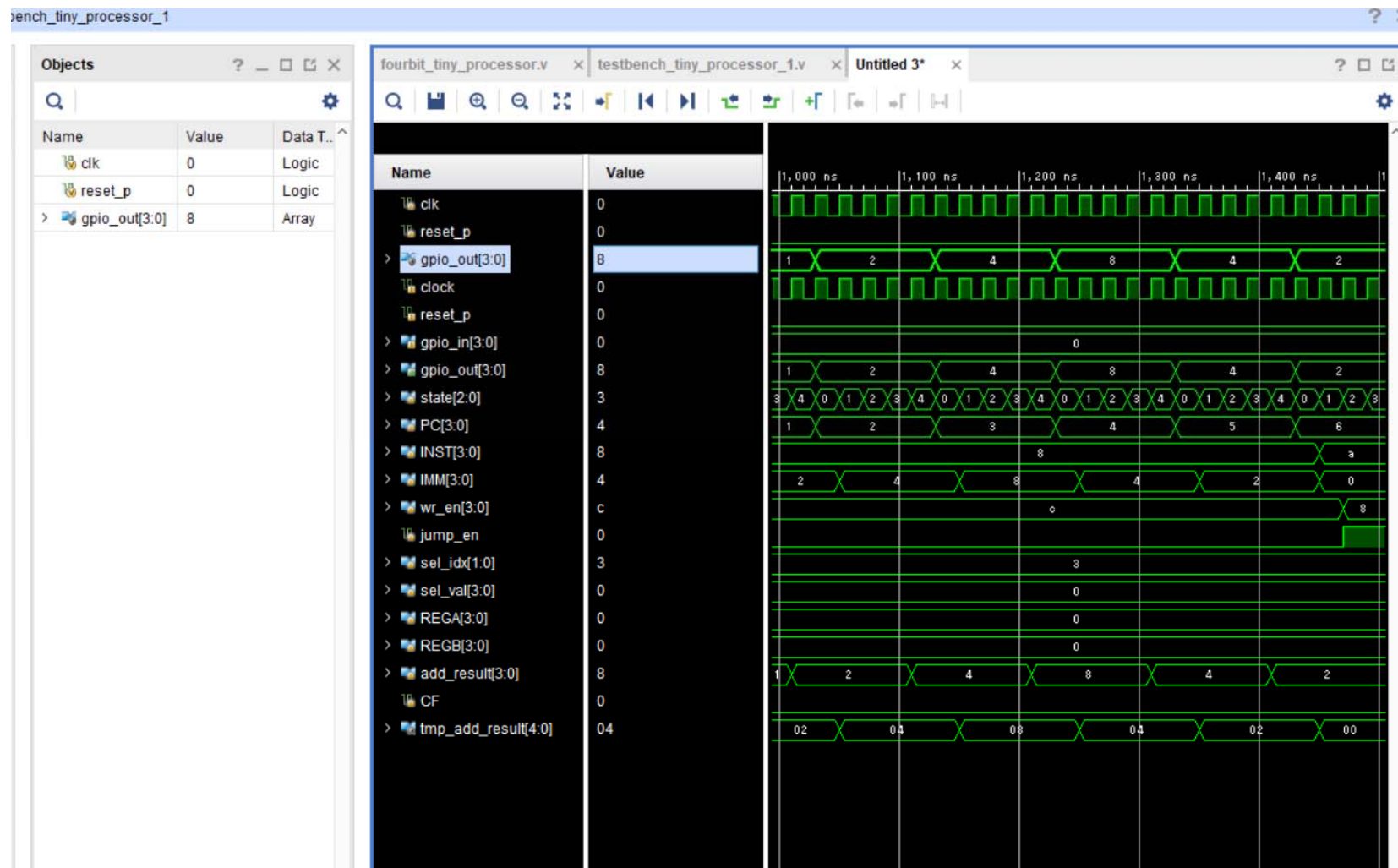
# RTL for a Write Back to Register

```
81 3'b100:begin // write back regs
82   state <= 3'b000;
83   if( wr_en[0] == 1'b1) REGA <= add_result;
84   if( wr_en[1] == 1'b1) REGB <= add_result;
85   if( wr_en[2] == 1'b1) gpio_out <= add_result;
86   if( jump_en == 1'b1)
87     PC <= add_result;
88   else
89     PC <= PC + 1'b1;
90 end
```



# Behavior Simulation

- See, testbench\_tiny\_processor.v



# Wrapper of a Topmodule

- see, github. Source: topmodule\_fourbit\_tiny\_processor.v

```
23 module topmodule_fourbit_tiny_processor(  
24     input sysclk,  
25     input [0:0]btn,  
26     output [3:0]led  
27 );  
28  
29     reg [24:0]count; // 1sec ~ 8ns (125MHz) * 125,000,000  
30  
31     always@(posedge sysclk or posedge btn[0])begin  
32         if( btn[0] == 1'b1) begin  
33             count <= 25'b0;  
34         end else begin  
35             count <= count + 1'b1;  
36         end  
37     end  
38  
39     fourbit_tiny_processor tiny_processor_inst(  
40         .clock( count[24]),  
41         .reset_p( btn[0]),  
42         .gpio_in( 4'b0000),  
43         .gpio_out( led)  
44     );  
45  
46 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

0x84, // OUT 0100

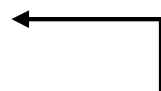
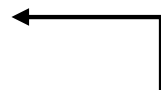
0x41, // ADD 0001

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](mailto:nakahara@ict.e.titech.ac.jp)

Deadline is 20<sup>th</sup>, July, 2018 (At the beginning of the lecture)