CC-SiMP-32 **C**ity **C**ollege **Si**mple **M**IPS **P**rocessor **32**-bit

Zheng Peng

City College of New York

Objectives

- Understand the MIPS ISA
- Understand the structure of single cycle processor
- Understand the principals of processor design
- Practice the VHDL programming skill

Tasks

- Implementing a basic 32-bit MIPS processor using VHDL
- Successfully running a MIPS program on the processor

CC-SiMP-32

- 32-bit MIPS processor
- 32 32-bit registers
- 256-byte instruction memory (ROM)
- 256-byte data memory (RAM)

Supported instructions

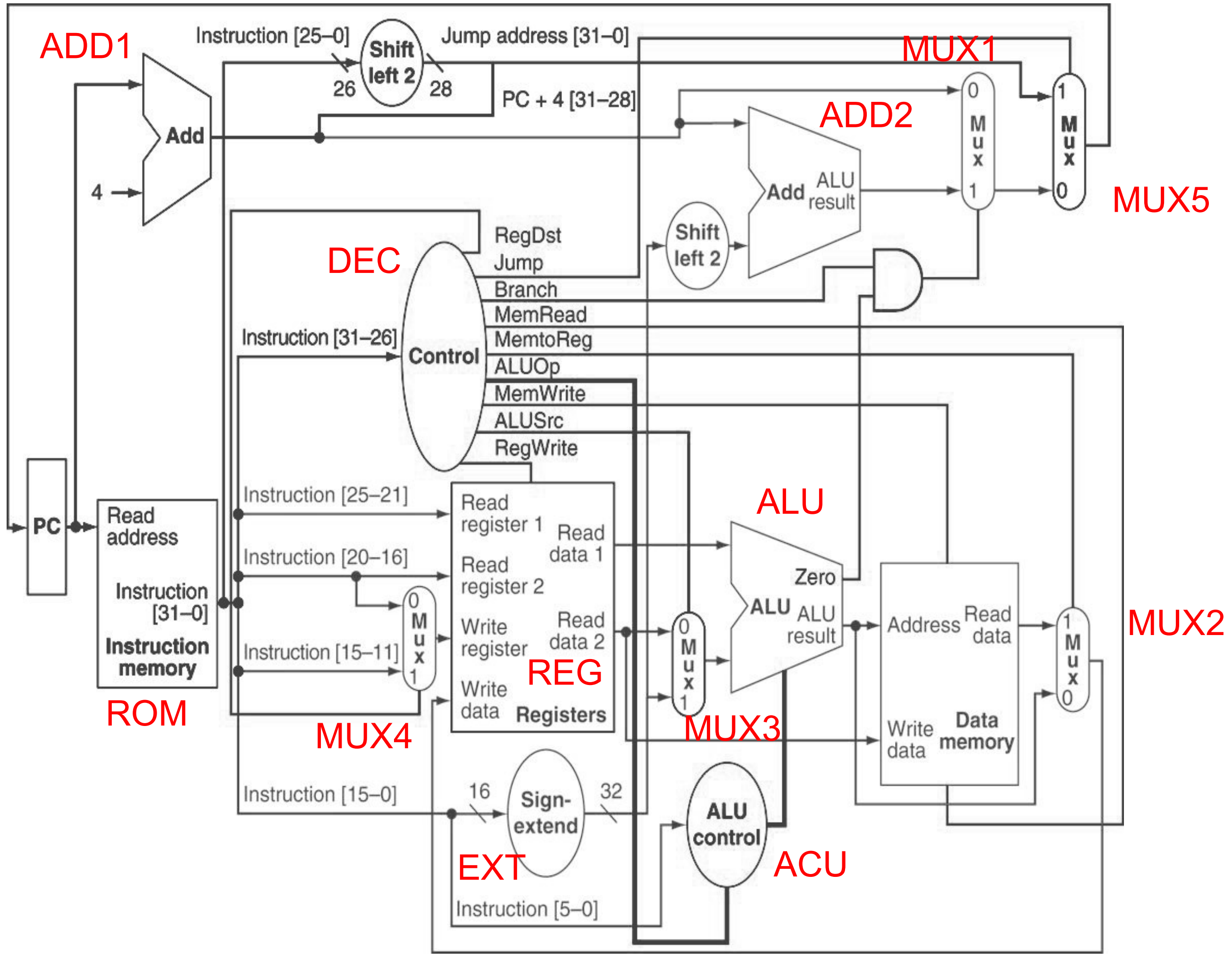
- Your processor needs to support at least the following instructions:

Instruction Name	Instruction Format
`addu`	R-format
`addi/addiu`	I-format
`beq/bne`	I-format
`lw/sw`	I-format
`j`	I-format

  – Note that in this class, you are not expected to fully support `syscall`. However, your processor should be able to recognize the `syscall` instruction to stop the program execution.

Basic Design

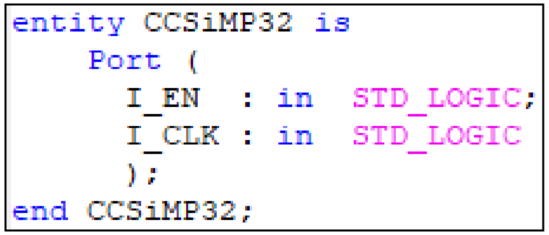
- Your design will be based on the following diagram



* Note that you need to make modifications to support both branch instructions.

Top Module

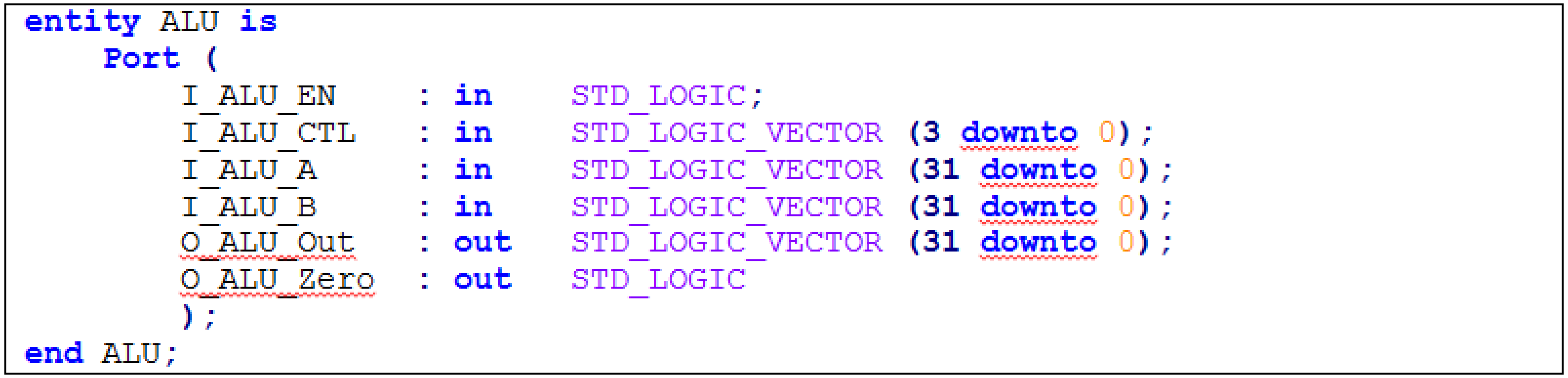
- The top module is the root of the design hierarchy.
- It hosts all system components and defines the interconnection between them.
- It also describes the inputs/outputs of the system.
- The VHDL entity declaration of the ALU module is given as follows.



```
entity CCSiMP32 is
    Port (
      I_EN  : in  STD_LOGIC;
      I_CLK : in  STD_LOGIC
      );
end CCSiMP32;
```

ALU Module

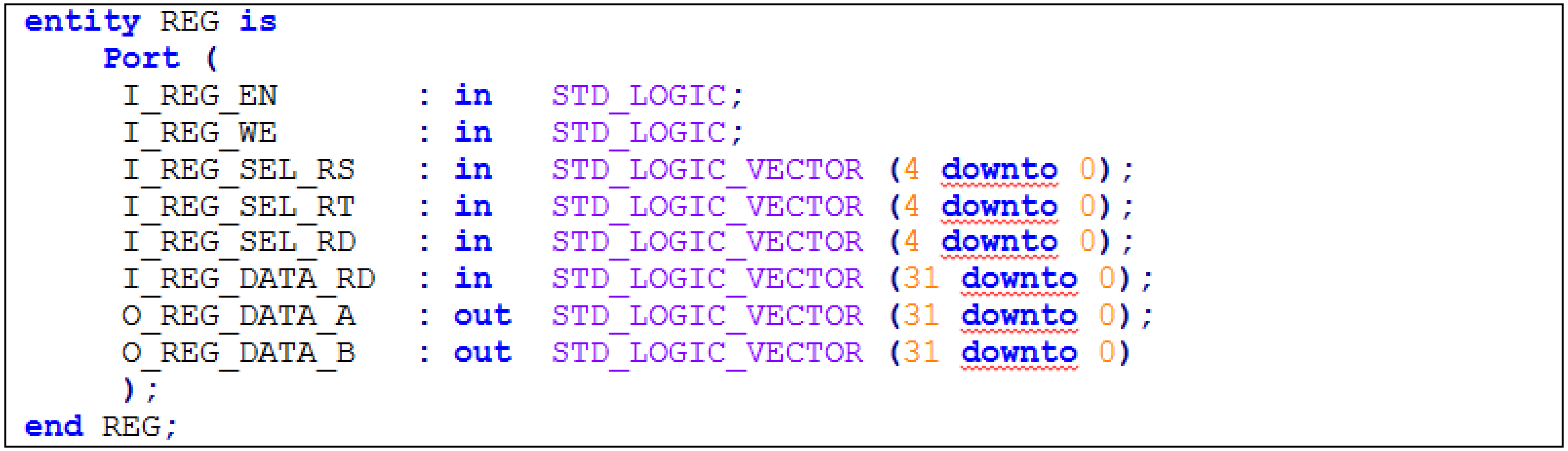
- ALU is the heart of the processor.
- In this lab, the ALU should be able to perform additions and subtractions.
- The VHDL entity declaration of the ALU module is given as follows.



```
entity ALU is
    Port (
        I_ALU_EN     : in     STD_LOGIC;
        I_ALU_CTL    : in     STD_LOGIC_VECTOR (3 downto 0);
        I_ALU_A      : in     STD_LOGIC_VECTOR (31 downto 0);
        I_ALU_B      : in     STD_LOGIC_VECTOR (31 downto 0);
        O_ALU_Out    : out    STD_LOGIC_VECTOR (31 downto 0);
        O_ALU_Zero   : out    STD_LOGIC
        );
end ALU;
```

Register Module

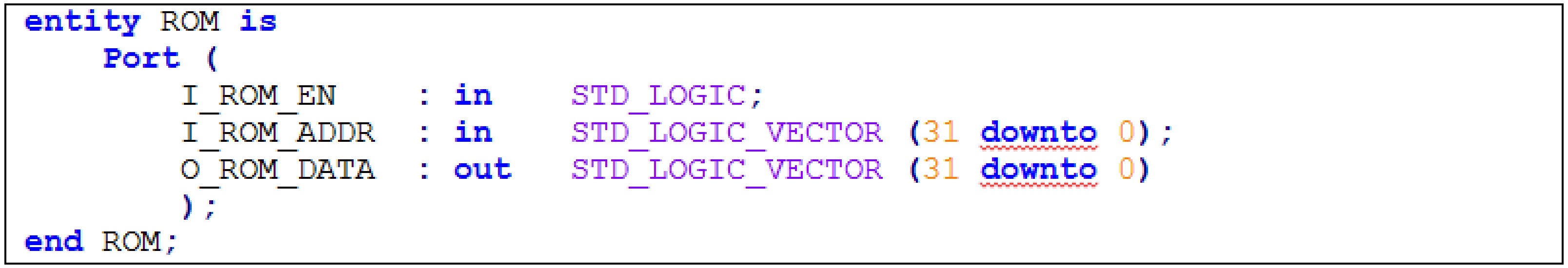
- Note that in a MIPS processor, the register module includes 32 registers. Each of the register is 32-bit in width.
- The VHDL entity declaration of the register module is given as follows.



```
entity REG is
    Port (
     I_REG_EN       : in   STD_LOGIC;
     I_REG_WE       : in   STD_LOGIC;
     I_REG_SEL_RS   : in   STD_LOGIC_VECTOR (4 downto 0);
     I_REG_SEL_RT   : in   STD_LOGIC_VECTOR (4 downto 0);
     I_REG_SEL_RD   : in   STD_LOGIC_VECTOR (4 downto 0);
     I_REG_DATA_RD  : in   STD_LOGIC_VECTOR (31 downto 0);
     O_REG_DATA_A   : out  STD_LOGIC_VECTOR (31 downto 0);
     O_REG_DATA_B   : out  STD_LOGIC_VECTOR (31 downto 0)
     );
end REG;
```

Instruction Memory

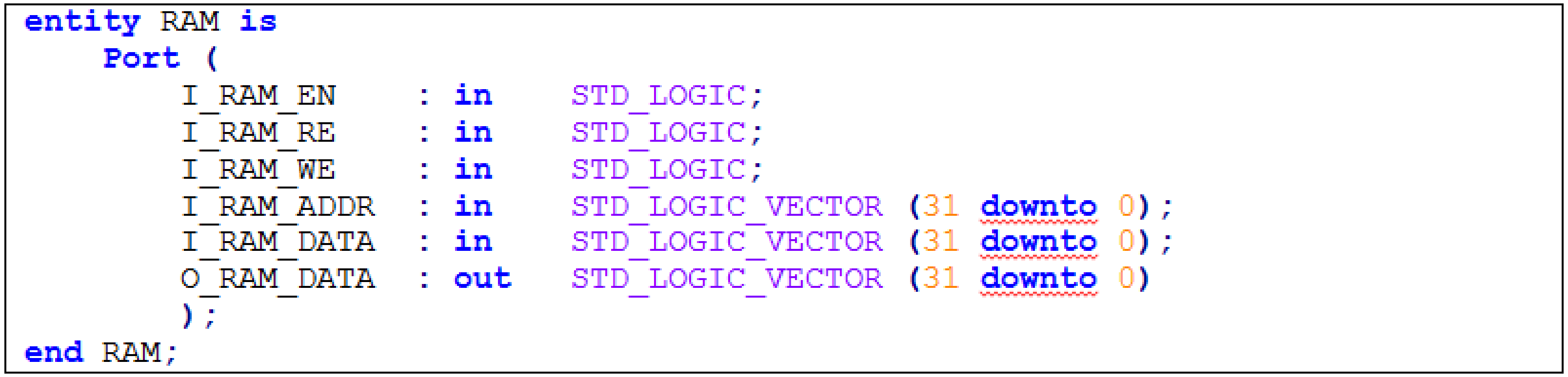
- This MIPS processor is expected to have 256 bytes of instruction memory.
- The VHDL entity declaration of the instruction memory is given as follows.



```
entity ROM is
    Port (
        I_ROM_EN     : in    STD_LOGIC;
        I_ROM_ADDR   : in    STD_LOGIC_VECTOR (31 downto 0);
        O_ROM_DATA   : out   STD_LOGIC_VECTOR (31 downto 0)
        );
end ROM;
```

Data Memory

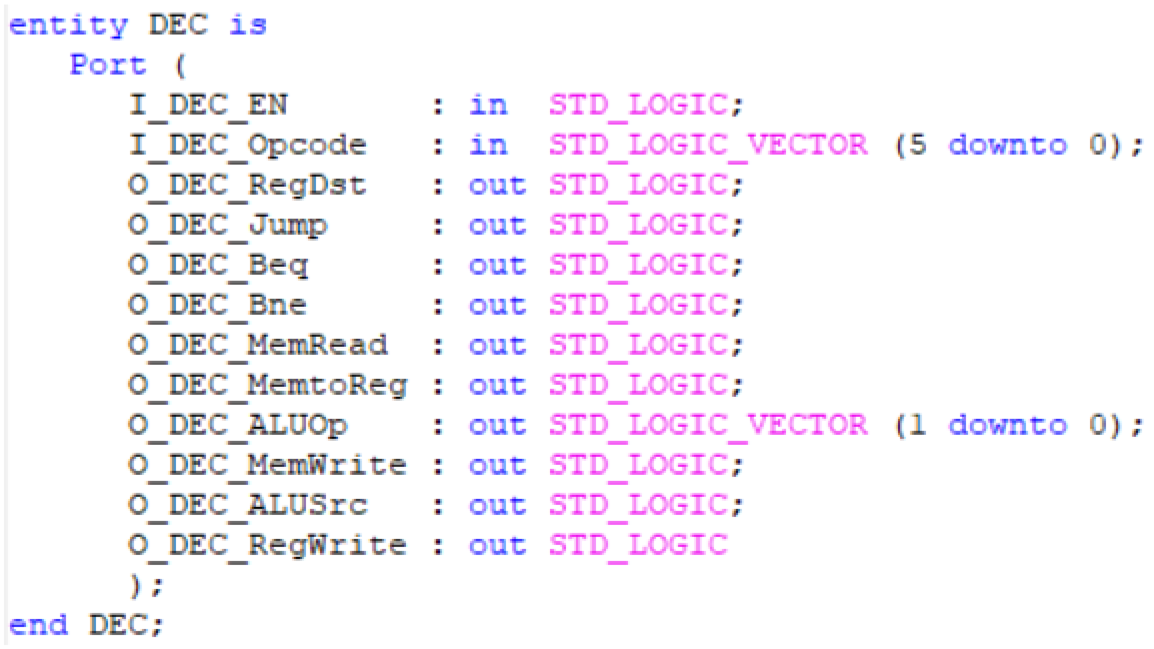
- The MIPS processor is expected to have 256 bytes of data memory



```
entity RAM is
    Port (
        I_RAM_EN     : in    STD_LOGIC;
        I_RAM_RE     : in    STD_LOGIC;
        I_RAM_WE     : in    STD_LOGIC;
        I_RAM_ADDR   : in    STD_LOGIC_VECTOR (31 downto 0);
        I_RAM_DATA   : in    STD_LOGIC_VECTOR (31 downto 0);
        O_RAM_DATA   : out   STD_LOGIC_VECTOR (31 downto 0)
        );
end RAM;
```

Instruction Decoder

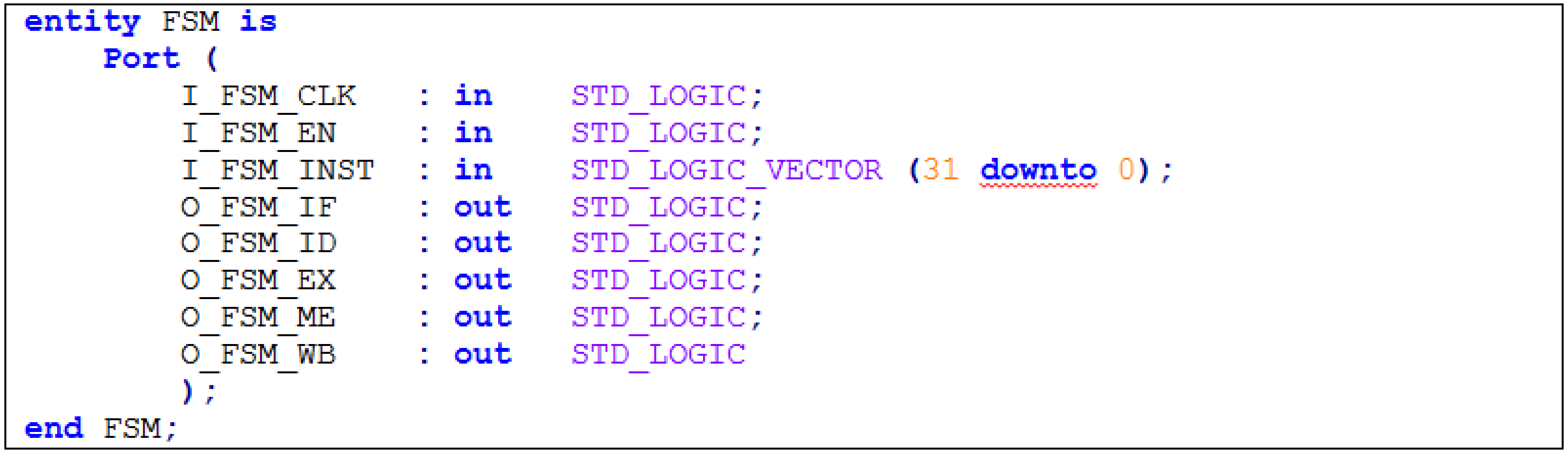
- The instruction decoder generates the control signals based on the opcode of the instructions.



```
entity DEC is
   Port (
      I_DEC_EN          : in   STD_LOGIC;
      I_DEC_Opcode      : in   STD_LOGIC_VECTOR (5 downto 0);
      O_DEC_RegDst      : out  STD_LOGIC;
      O_DEC_Jump        : out  STD_LOGIC;
      O_DEC_Beq         : out  STD_LOGIC;
      O_DEC_Bne         : out  STD_LOGIC;
      O_DEC_MemRead     : out  STD_LOGIC;
      O_DEC_MemtoReg    : out  STD_LOGIC;
      O_DEC_ALUOp       : out  STD_LOGIC_VECTOR (1 downto 0);
      O_DEC_MemWrite    : out  STD_LOGIC;
      O_DEC_ALUSrc      : out  STD_LOGIC;
      O_DEC_RegWrite    : out  STD_LOGIC
      );
end DEC;
```

State Machine Module

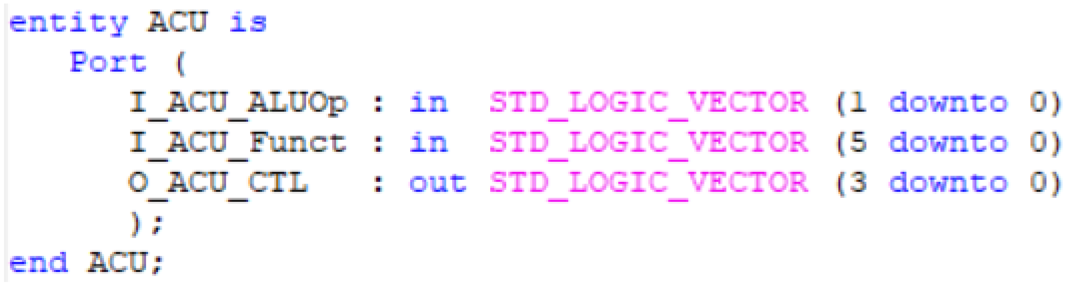
- The state machine module is the brain of the processor.
- It coordinates the operations among processor components.



```
entity FSM is
    Port (
        I_FSM_CLK    : in    STD_LOGIC;
        I_FSM_EN     : in    STD_LOGIC;
        I_FSM_INST   : in    STD_LOGIC_VECTOR (31 downto 0);
        O_FSM_IF     : out   STD_LOGIC;
        O_FSM_ID     : out   STD_LOGIC;
        O_FSM_EX     : out   STD_LOGIC;
        O_FSM_ME     : out   STD_LOGIC;
        O_FSM_WB     : out   STD_LOGIC
        );
end FSM;
```

ALU Control Unit

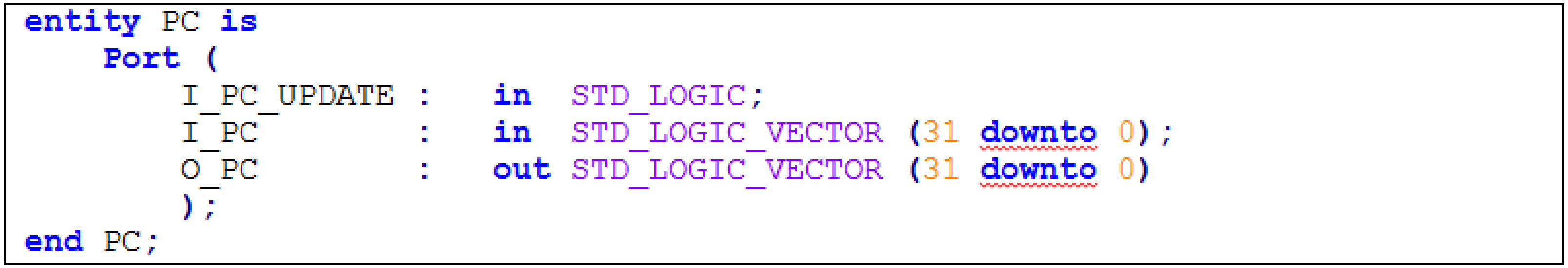
- The ALU control unit (ACU) determines the ALU operations.
- It takes inputs from the `ALUOp` signal from the instruction decoder, as well as the `Funct` code of the current instruction.
- Its outputs are the control signals going to the ALU.



```
entity ACU is
   Port (
      I_ACU_ALUOp : in  STD_LOGIC_VECTOR (1 downto 0);
      I_ACU_Funct : in  STD_LOGIC_VECTOR (5 downto 0);
      O_ACU_CTL   : out STD_LOGIC_VECTOR (3 downto 0)
      );
end ACU;
```

PC Module

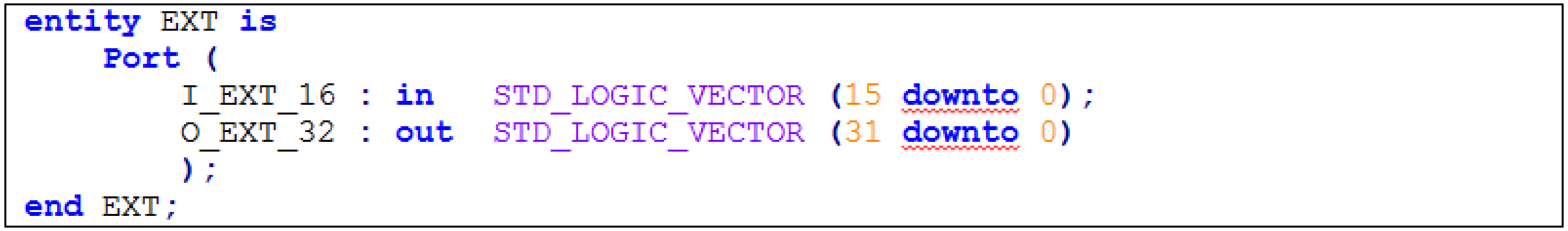
- The PC (Program Counter) is a register to store the instruction address.



```
entity PC is
    Port (
        I_PC_UPDATE :   in  STD_LOGIC;
        I_PC        :   in  STD_LOGIC_VECTOR (31 downto 0);
        O_PC        :   out STD_LOGIC_VECTOR (31 downto 0)
        );
end PC;
```

Sign Extension Module

- The sign extension module performs a signed extension on the immediate number encoded in I-format instructions.



```
entity EXT is
    Port (
        I_EXT_16 : in   STD_LOGIC_VECTOR (15 downto 0);
        O_EXT_32 : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end EXT;
```

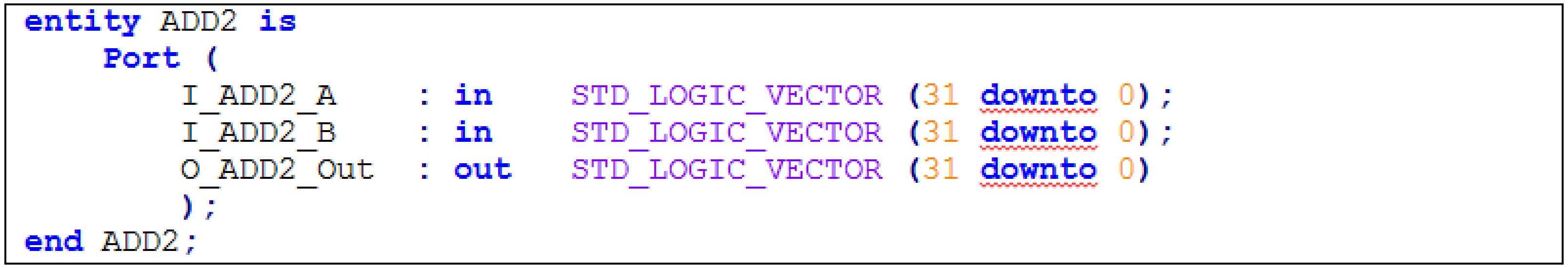
Adder Module (1)

- `ADD1` in the processor design diagram
- This adder module increments the PC value by 4.

```vhdl
entity ADD1 is
    Port (
        I_ADD1_A     : in     STD_LOGIC_VECTOR (31 downto 0);
        O_ADD1_Out   : out    STD_LOGIC_VECTOR (31 downto 0)
        );
end ADD1;
```

Adder Module (2)

- ADD2 in the processor design diagram
- This adder module computes the branch target address.



```
entity ADD2 is
    Port (
        I_ADD2_A     : in    STD_LOGIC_VECTOR (31 downto 0);
        I_ADD2_B     : in    STD_LOGIC_VECTOR (31 downto 0);
        O_ADD2_Out   : out   STD_LOGIC_VECTOR (31 downto 0)
        );
end ADD2;
```

Multiplexers

- In the processor design, there are four 2-1 multiplexers.
- Note that, depending on the width of the input bus, there are two different types of multiplexers.

```
entity MUX32 is
    Port (
        I_MUX_0      : in     STD_LOGIC_VECTOR (31 downto 0);
        I_MUX_1      : in     STD_LOGIC_VECTOR (31 downto 0);
        I_MUX_Sel    : in     STD_LOGIC;
        O_MUX_Out    : out    STD_LOGIC_VECTOR (31 downto 0)
        );
end MUX32;
```

```
entity MUX5 is
    Port (
        I_MUX_0      : in     STD_LOGIC_VECTOR (4 downto 0);
        I_MUX_1      : in     STD_LOGIC_VECTOR (4 downto 0);
        I_MUX_Sel    : in     STD_LOGIC;
        I_MUX_Out    : out    STD_LOGIC_VECTOR (4 downto 0)
        );
end MUX5;
```