The **Microelectronics Training Center**

The MTC is an initiative within the INVOMEC division

Industrialization & Training in Microelectronics

Lab-exercise

Lab 4:

Design of the ALU

Cluster: Cluster1
Module: Module2a

Target group: Students

Version: 1.1
Date: 21/03/06
Author: Osman Allam
Modified by: Geert Vanwijnsberghe
History : clarified difference between std_logic_1164 and std_numeric_std

This material was developed with support of the European Social Fund.
ESF: Prevent and combat unemployment by promoting employability, entrepreneurship, adaptability and equal opportunities between women and men, and by investment in people.
http://www.esf-agentschap.be

Introduction

The Arithmetic and Logic Unit (ALU) is the functional unit of any computer system. It performs the arithmetic, logical and shift operations demanded by user's programs running on the microprocessor.

The condition flags store information about the latest operation performed by the ALU. The condition flags in our microprocessor are negative (neg), overflow (ovf) and zero (zro).

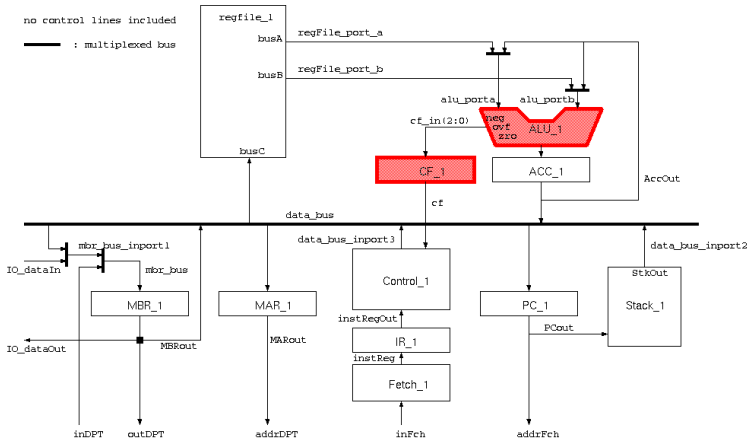


Figure 1: CPU architecture

Objectives

After completing this module, the student should be able to:

- Use and possibly overload standard functions;
- Convert from one type to another in VHDL;
- Describe arithmetic and logic operations in VHDL;
- Explain different types of shift and rotate operations.

Knowledge background

- Understanding of the ALU function
- Basic knowledge of VHDL

Classification

- Level: 3
- Duration: 90 minutes

Input

Specifications of the functions to be developed. VHDL templates are provided.

The lab

The IEEE library (IEEE.std_logic_1164.all) provides a wide range of logic functions.
The IEEE library (IEEE.std_numeric_std) provides many arithmetic functions, but most of them take `(un)signed` and natural arguments.

Since all data ports in your design are `std_logic_vector`, you need to convert them into `(un)signed` and `natural` in order to be able to use the arithmetic functions.

Converting a `std_logic_vector` signal into a `natural` one goes in 2 steps:

a. Casting into `(un)signed`.
b. Converting into `natural` by using the function:

```
function TO_INTEGER (ARG: UNSIGNED) return NATURAL;
```

Example:

```
signal arg_std : std_logic_vector (7 downto 0);
signal arg_uns : unsigned (7 downto 0);
signal arg_nat : natural;
…
arg_uns <= unsigned (arg_std);
arg_nat <= to_integer (arg_uns);
```

or you can do the conversion in one statement:

```
arg_nat <= to_integer (unsigned (arg_std));
```

Converting a `natural` signal into a `std_logic_vector` one goes in 2 steps too:

1. Converting into `(un)signed` by using the function:

```
function TO_UNSIGNED (ARG, SIZE: NATURAL) return UNSIGNED;
```

and

```
function TO_SIGNED (ARG, SIZE: NATURAL) return SIGNED;
```


2. Casting into std_logic_vector.

Example:

```
arg_uns <= to_unsigned (arg_nat, 8);
arg_std <= std_logic_vector (arg_uns);
```

or you can do the converstion in one statement:

```
arg_std <= std_logic_vector (to_unsigned (arg_nat, 8));
```

Our ALU in micor6 will be capable of performing the following functions : ADD_OP, SUB_OP, MULT_OP, (*DIV_OP, REM_OP*), AND_OP, OR_OP, XOR_OP, INV_OP, INC_OP, DEC_OP, ZRO_OP, PASS_A, PASS_B, SHR_ARTH, SHR_LGC, SHL_ARTH, SHL_LGC, ROTR, ROTL . All off these function names are combined in an enumerated type alu_op that you find in the micro_pk.vhd package.

1. Arithmetic functions

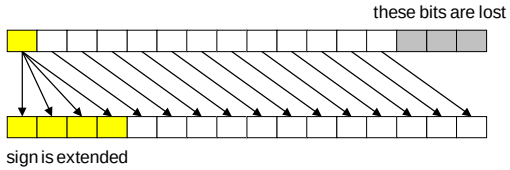
Addition, subtraction .. available in IEEE.std_numeric_std operate on signed operands and return signed results. Therefore the above conversions and castings will be needed to apply these functions on the ports that are all of type std_logic_vector.

2. Logic functions:

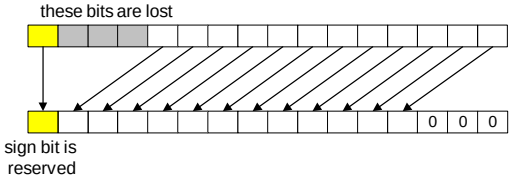
AND, OR, NOT, XOR operate on `std_logic_vector` operands directly and return `std_logic_vector` results. No conversion or casting is required.

The functionality of the shift and rotate functions is shown below:

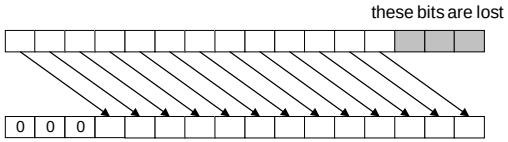
1. Arithmetic shift right by count : SHR_ARTH



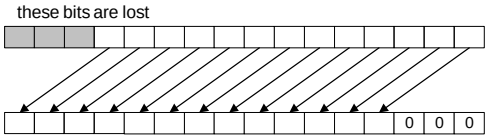
2. Arithmetic shift left by count : SHL_ARTH



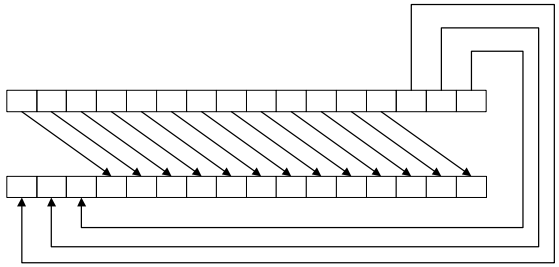
3. Logic shift right by count : SHR_LGC



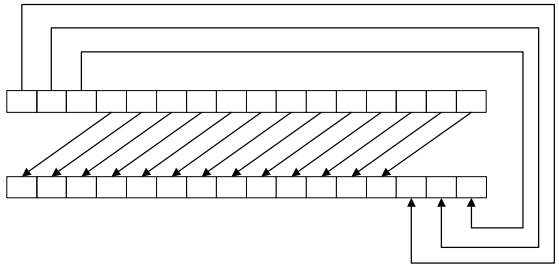
4. Logic shift left by count : SHL_LGC



5. Rotate right by count : ROTR

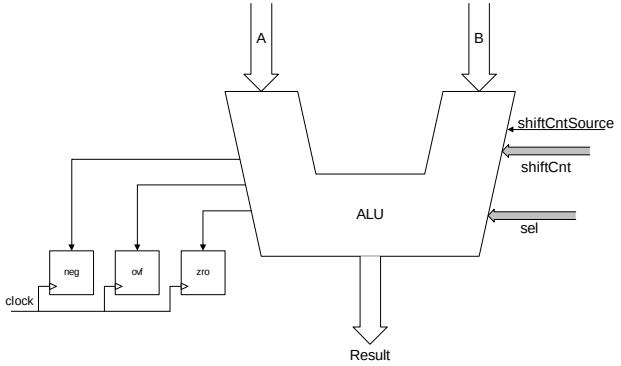


6. Rotate left by count : ROTL



Exercise

Design an Arithmetic and Logic Unit (ALU). Apart from performing a function it also updates 3 condition flags; negative (`neg`), overflow (`ovf`) and zero (`zro`). The ALU is purely combinational logic. It contains no storage elements. The condition flags are stored in 3 D-flip-flops outside the ALU.



The ALU implements the following functions:

1. Addition (result = A + B)
2. Subtraction (result = A – B)
3. AND (result = A AND B)
4. OR (result = A OR B)
5. XOR (result = A XOR B)
6. NOT (result = NOT A)
7. Increment by 1 (result = A + 1)
8. Decrement by 1 (result = A – 1)
9. Zero the result (result = 0)
10. Arithmetic/Logic shift by a given count to the right/left
11. Logic shift right by a count
12. Rotate by a given count to the right/left
13. Pass the left operand (result = A)
14. Pass the right operand (result = B)

The ALU has 3 data inputs:

1. A : the left operand
2. B: the right operand
3. `shiftCnt`: Shift/rotate count

The ALU has 2 control inputs:

1. `sel`: selects which operation to be performed. Possible values are declared by constants of type alu_op in the package `micro_pk` (constants of type `alu_op`).
2. `shiftCntSrc`: Shift count source: when active, the shifter uses the slice [5:0] of the input B as the shift count. Otherwise, the shift count is the shiftCnt input.

The ALU has 2 outputs:

1. `result`: the computation result
2. 3 condition flag outputs

Use the template provided in the file `alu.vhd`
In this module you only comple the alu.vhd file and compile it. The complete verification of the code will be accomplished in the next module.
`Note : DIV_OP and REM_OP will not be implemented in this lab.`