The **Microelectronics Training Center**

*The MTC is an initiative within the INVOMEC division*

Industrialization & Training in Microelectronics

Lab-exercise

# *Lab 4:*

# Design of the stack

Cluster: Cluster1
Module: Module3c

Target group: Students

Version: 1.1
Date: 15/12/06
Author: Osman Allam
Modified by: Geert Vanwijnsberghe
History : Testbench added + small changes to the text

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 stack in Micro6 holds the address of the instruction next to a subroutine call instruction to insure that the calling program is continued after the subroutine returns.

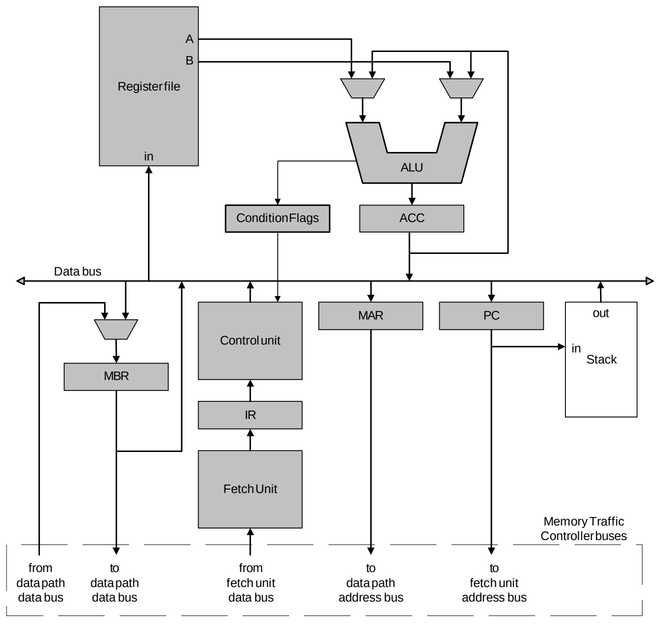


**Figure 1: CPU architecture**

## Objectives

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

- Build a storage system in VHDL
- Describe how hardware stacks work

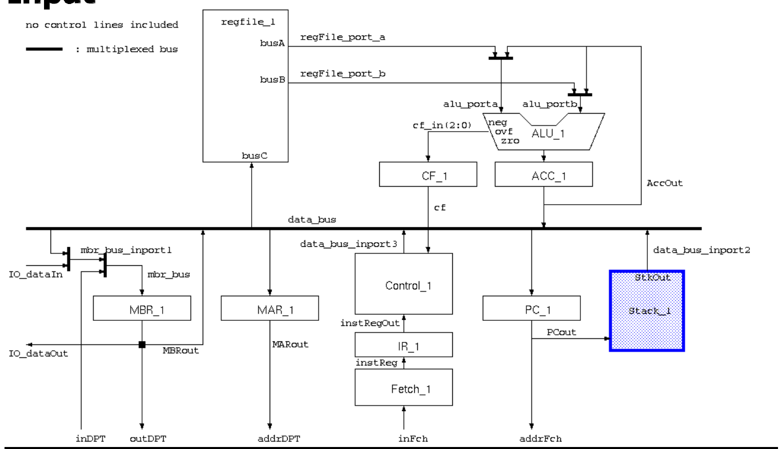
## Knowledge background

Basic VHDL knowledge

## Classification

- Level: 3
- Duration: 90 minutes

## Input



VHDL template for the stack and a complete testbench tb_stack.vhd.

## The lab

Stacking is a data storage technique in which data is accessed in a last-in-first-out (LIFO) manner.
There are 2 stack constructs in Micro6.

1. Hardware stack: this is a stand-alone, small stack used to save the contents of the program counter (PC) when a jump or a call to a subroutine is encountered so that returning to the calling routine is possible.
2. Memory stack: this is a segment in the main memory. The memory stack is much larger than the hardware stack. It is used in Micro6 for passing parameters to subroutines.

In this exercise, we will focus on designing the hardware stack. The memory stack is simply implemented by a stack pointer pointing to the next slot (location) of the stack segment. Its operation is controlled directly by the control unit.

### Stack structure

The basic components of the hardware stack are:

1. A set of registers (known as the stack slots). The number of the registers is the depth of the stack.
2. A stack pointer (sp): conventionally, the stack pointer points to the next free slot of the stack.
3. An output register

### Stack operations

1. Push: Writing data into the stack is performed in 2 clock cycles:
   a. Writing data into the slot pointed to by the stack pointer
   b. Incrementing the stack pointer
2. Pop: Reading data from the stack is performed in 2 clock cycles:

   a. Decrementing the stack pointer
   b. Reading data from the slot pointed to by the stack pointer

## Exercise

Using the template provided in the `stack.vhd` file, design a stack with the following specifications:

1. The depth of the stack is 16.
   sp=0 =>empty
   sp=16 => full
2. The width of the stack is unconstrained.
3. The stack operations, push and pop, are possible only when the enable input is asserted.
4. The stack can be asynchronously reset without asserting the enable input.
5. Control inputs:
   a. PUSH: higher priority
   b. POP: lower priority
   c. When the stack is full, push operations are ignored. And similarly, when it is empty, pop operations are ignored. In both cases, the stack pointer and the stack contents remain unchanged.

Use a finite state machine (FSM) to model your stack. There are 5 states:

1. Idle: the stack is in steady state.
2. Writing (wr): when the PUSH input is asserted, the stack input data is stored into the Stack Slot pointed to be the Stack Pointer and additionally into the Output Register.
3. Incrementing (inc) the Stack Pointer.
4. Decrementing (dec) the stack pointer: when the POP input is asserted, the Stack Pointer is decremented.
5. Reading (rd): following the Decrementing state, the Output Register is loaded by the data in the Stack Slot pointed to by the updated Stack Pointer.

The state transition table is shown below.

| Current | Inputs | | | Stack condition | | Next State |
|---|---|---|---|---|---|---|
| State | EN | PUSH | POP | Empty(sp=0) | Full(sp=16) | |
| idle | 1 | 1 | x | x | NO | wr |
| | 1 | 0 | 1 | NO | x | dec |
| | ELSE | | | | | idle |
| inc | 1 | 1 | x | x | NO | wr |
| | 1 | 0 | 1 | NO | x | dec |
| | ELSE | | | | | idle |
| rd | 1 | 1 | x | x | NO | wr |
| | 1 | 0 | 1 | NO | x | dec |
| | ELSE | | | | | idle |
| wr | x | x | x | x | x | inc |
| dec | x | x | x | x | x | rd |

Use the available testbench to verify your design.