The **Microelectronics Training Center**

The MTC is an initiative within the INVOMEC division

Industrialization & Training in Microelectronics

Lab-exercise

Lab 4:

Design of the decode function inside the control unit

Cluster: Cluster1
Module: Module4a

Target group: Students

Version: 1.0
Date: 18/12/06
Author: Osman Allam
Modified by: Geert Vanwijnsberghe
History : figures added and modified

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 Control Unit of Micro6 is split into 3 sub-units: Fetch Unit, Decode Unit and Execute Unit. Splitting the control unit in this way makes the code easier to understand and debug.

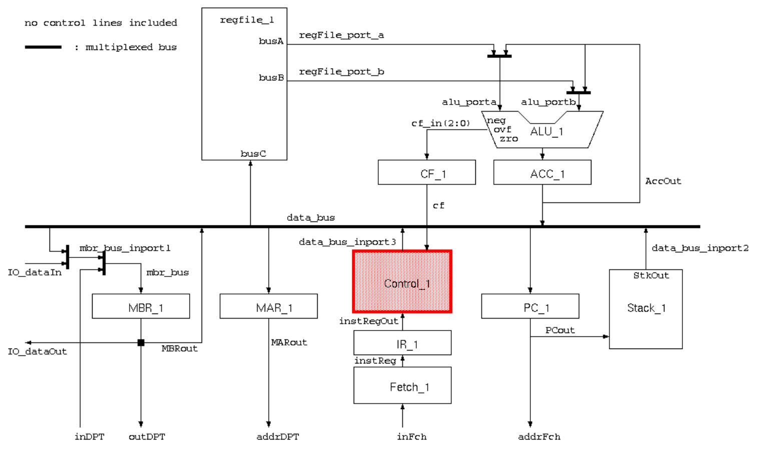


Figure 1: CPU architecture

Objectives

After completing this module, you should be able to:

- Use record data types
- Describe decoding functionality

Knowledge background

- Basic VHDL knowledge
- Understanding of the operation of control units within any computer system

Classification

- Level: 3
- Duration: 2 hours

Input

VHDL template

The lab

The basic communication between control unit and fetch unit uses a kind of handshake mechanism allowing both units to run as independently as possible.
When the fetch unit has received a ReadInstr pulse from the control unit it will start reading the next instruction from the memory while de-asserting the vldInstr signal. After the instruction is received by the fetch unit the vldInstr signal is asserted until a new ReadInstr pulse is given by control unit.

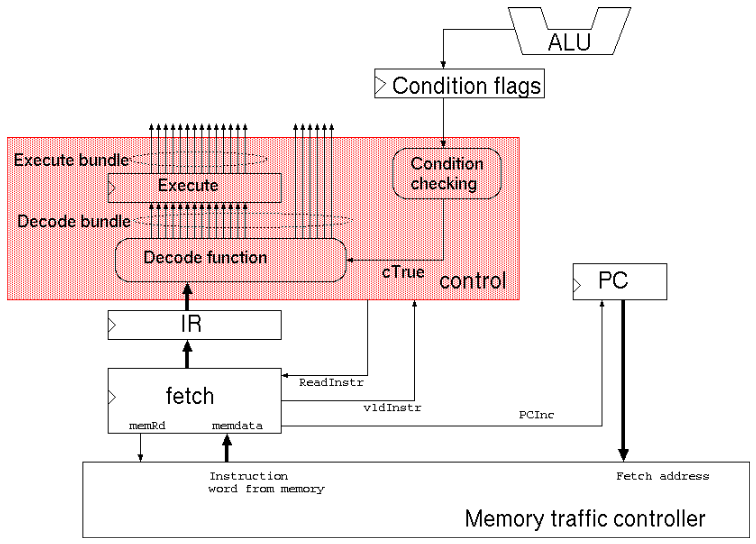


Figure 2 : fetch and control path

The control unit consists of a large FSM but the basic structure for an instruction group gx is show in figure 3. The "Reading" and the "Decoding" states are only "on" for 1 clock cycle and the ReadInstr signal is asserted while the FSM is in the "Reading" state. You can see that the FSM goes to an idle ste (= stall) when for some reason the next instruction is not yet available from the fetch unit.

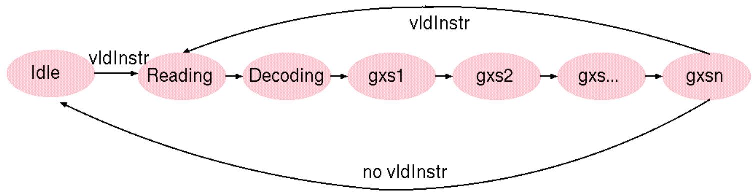


Figure 3

In one of the next modules we will have a closer look at the control unit and its state machine.

In this module, you will design the Decode function of the microprocessor as a single VHDL function in the `micro_control_pk` package. Figure 2 shows where this function is located in the control unit.

Before we proceed with writing the decode function, we would like to demonstrate a couple of VHDL features that will greatly help you accomplish your task.

Record data types

If your design contains many signals that you are no longer able to interprete your code easily, it might be helpful if you group these signals in records. Signals may be grouped according to their functionality and/or direction. Grouping signals in records provides two advantages:

1. Improving readability.
2. Facilitating adding and removing ports.

```
type <new_type> is record
  <signal1> : <signal1_type>;
  <signal2> : <signal2_type>;
  <signal3> : <signal3_type>;
  ..
end record;
```

Note that signals belonging to records can be of any data type including records as well.

Aliases

An alias of a data object (signal, variable, constant, … etc) provides an alternate identifier to refer to that object. Aliases can be used in different ways but in this module, we are interested in using aliases to denote elements or slices of arrays.

```
alias <identifier> : <data_type> is <array_data (<element>)>;
```

```
alias <identifier> : <data_type> is <array_data (<slice>)>;
```

Example: the opcode is the most significant 5 bits of the instruction word (32-bits). We can express this information by declaring an alias.

```
alias opcode : std_logic_vector (4 downto 0) is instrWord (31
downto 27);
```

Exercise

1. Start writing the package `micro_control_pk` with declaring 2 record types, 1 for the decode bundle and 1 for the execute bundle (look at table below).

Signal name	Type	Range	Decode bundle	Execute bundle
instrGroup	instrGroup_t	NA	✓	
Asel	std_logic_vector	4:0	✓	✓
Bsel	std_logic_vector	4:0	✓	✓
Csel	std_logic_vector	4:0	✓	✓
ALUSel	alu_op	NA	✓	✓
shiftCnt	std_logic_vector	4:0	✓	

shiftCntSrc	std_logic		✓	
portAsel	std_logic		✓	
portBsel	std_logic		✓	
CFen	std_logic		✓	✓
STKen	std_logic		✓	
DATAsel	std_logic_vector	1:0	✓	✓
MBRsel	std_logic		✓	✓
memAddr	std_logic_vector	15:0	✓	
RegFileWr	std_logic			✓
stkInc	std_logic			✓
stkDec	std_logic			✓
ACCen	std_logic			✓
MARen	std_logic			✓
MBRen	std_logic			✓
PCen	std_logic			✓
IRen	std_logic			✓
STKpop	std_logic			✓
STKpush	std_logic			✓
memRd	std_logic			✓
memWr	std_logic			✓

ⓘ Instruction Groups: The Decode function divides instructions into a number of groups. This removes some decoding burden from the Execute Unit by combining all the instructions with the same clocked control signals (generated by the Execute Unit) in one group.

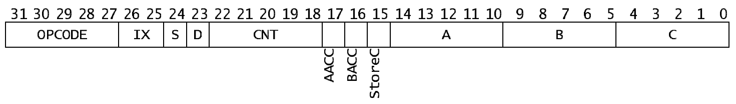
Note that some signals belong to both the decode bundle and the execute bundle. These signals are initialized by the decode unit and appear as default values in the execute unit state machine. The execute unit may update (change) the values of these signals.

2. Design the function `decodeInstr` (decode instruction) which takes an instruction word (`instReg`) from the fetch unit and an indication whether the branch condition is true (`cTrue`) and returns the decode bundle. In the declarative part of the function, declare aliases for all fields of the instruction formats.

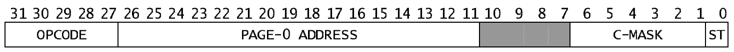
ⓘ `cTrue` is generated by a condition checking function in the control unit. It will be explained in a later module.

Micro6 supports 3 different instruction formats:

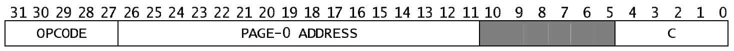
Format1



Format2



Format3



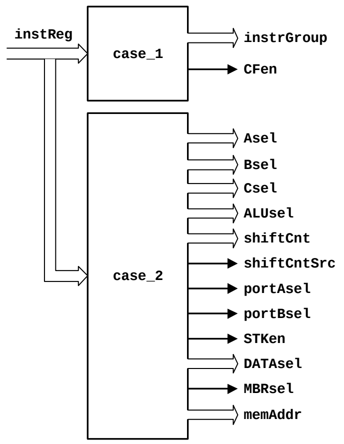
Field	Description
`OPCODE`	Opcode
`IX`	Index register
`S`	Shift count source
`D`	Shift direction
`CNT`	Shift count
`AACC`	ALU port A is ACC (Accumulator)
`BACC`	ALU port B is ACC (Accumulator)
`StoreC`	Store the result in C
`A`	Register file port A selection lines
`B`	Register file port B selection lines
`C`	Register file input port selection lines
`Page-0 address`	Address of a location in the first memory page
`C-MASK`	Condition mask
`ST`	Enable stack

a. Use aliases to describe instruction formats.

The operation of the decode unit can be split into 2 steps:

1. Decoding the opcode of the instruction;
2. Extracting the instruction parameters (for example, references to its operands).

The decode unit of Micro6 is described in VHDL by two case statements. The first one (`case_1`) is compact in the sense that a single choice of the case expression may correspond to more than 1 opcode. This case statement generates two outputs only.
Each case choice of the second case statement (`case_2`) corresponds to a single opcode. This case statement generates all other outputs. See the figure below.



For simplicity and ease of debug you may consider writing the default values of some of the signals of the decode bundle. Some values do not change in the case statement, those values are default values. This is already done for you in the VHDL template. Recall that not all parameters are relevant to all instructions, and the execute unit ignores all irrelevant information. This fact gives your more freedom when assigning default values.

b. Write the code of the first case statement (`case_1`) using the provided template.

Use the template provided in the file `micro_control_pk.vhd`.

Complete this code and compile. No testbench is given to verify this package.