

## Laboratory 7

### 7. Single-Cycle MIPS CPU Design (4): 16-bits version – One clock cycle per instruction

#### 7.1. Objectives

Study, design, implement and test

- **Instruction Execute Unit for the 16-bit Single-Cycle MIPS CPU**
- **Testing of the Arithmetical-Logical Instructions**

Familiarize the students with

- Single-Cycle CPU design: Defining the instructions / writing the test program (MIPS assembly language, machine code)
- Xilinx® ISE WebPack
- Digilent Development Boards (**DDB**)
  - [Digilent Basys Board – Reference Manual](#)
  - [Digilent Basys 2 Board – Reference Manual](#)
  - [Digilent Basys 3 Board – Reference Manual](#)

#### 7.2. Reduced Size MIPS Processor Description

(!) Read Lectures 3 and 4 in order to understand the works needed in this laboratory.

Remember that an instruction execution cycle (lecture 4) has the following phases:

- |         |   |                                    |
|---------|---|------------------------------------|
| • IF    | – | Instruction Fetch                  |
| • ID/OF | – | Instruction Decode / Operand Fetch |
| • EX    | – | Execute                            |
| • MEM   | – | Memory                             |
| • WB    | – | Write Back                         |

Your own Single-Cycle MIPS 16 processor implementation (that you will continue and hopefully finish in this laboratory) will be partitioned in 5 (five) components (new entities). These components will be declared and instantiated in the “test\_env” project.

The utility of this implementation will be understood in the future laboratories, when you will implement the pipeline version of your 16-bit MIPS processor!

In this laboratory you will design, VHDL description, implement and test the Instruction Execute Unit of your own single cycle MIPS 16 processor.

The data-path of the processor (32-bit version), including the control unit and the necessary control signals, is presented in the next figure. In order to reduce the complexity of the data-path, the control signals were not explicitly connected, but rather they can be easily identified by their names.

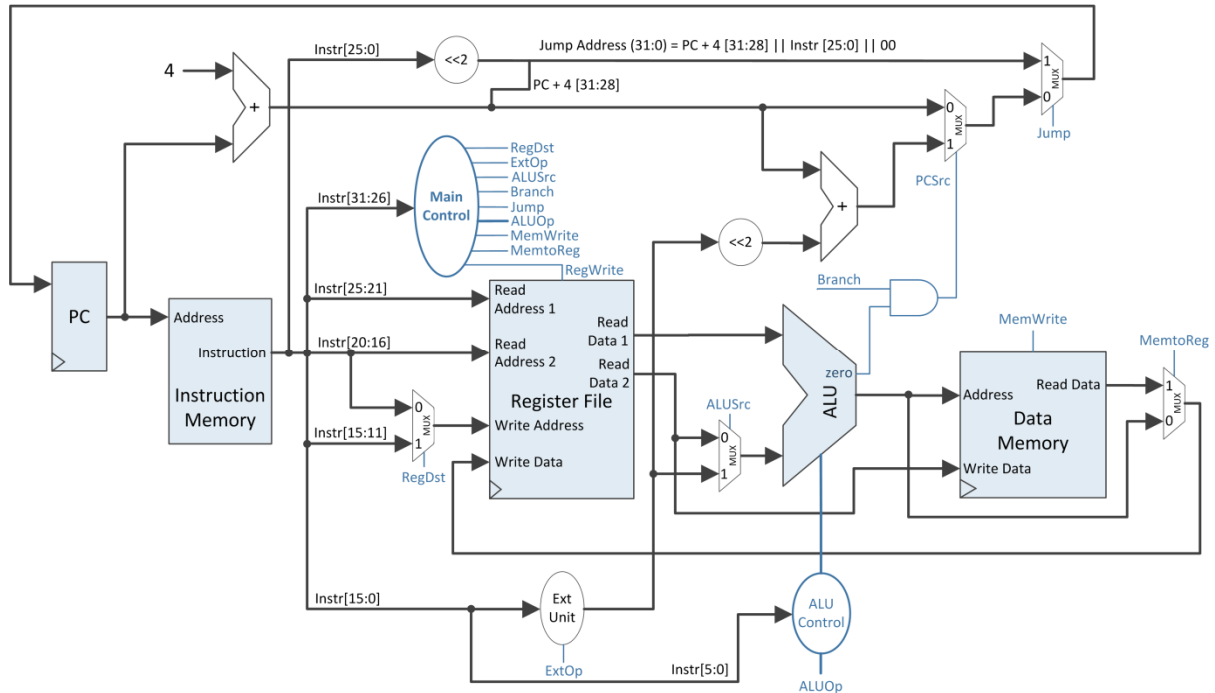


Figure 7-1: MIPS 32 Single-Cycle Data-Path + Control

As a reminder, the instruction formats for your MIPS 16 processor are presented below:

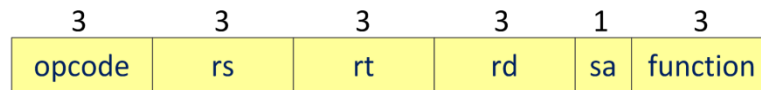


Figure 7-2: R-type Instruction format

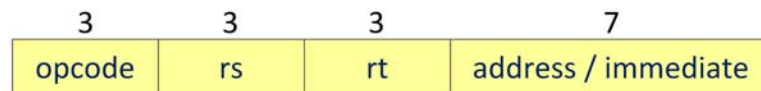


Figure 7-3: I-type Instruction format



Figure 7-4: J-type Instruction format

The Execute Unit (Ex) consists in the following components:

- Arithmetic Logic Unit (ALU)
- ALU Control
- Multiplexer
- Shift Left 2 and adder for branch target address computation

Please refer to laboratories 2 and 4 for the characteristics of these components for your Single-Cycle MIPS 16 processor.

The data-path of the Instruction Execute Unit is presented in Figure 7-5.

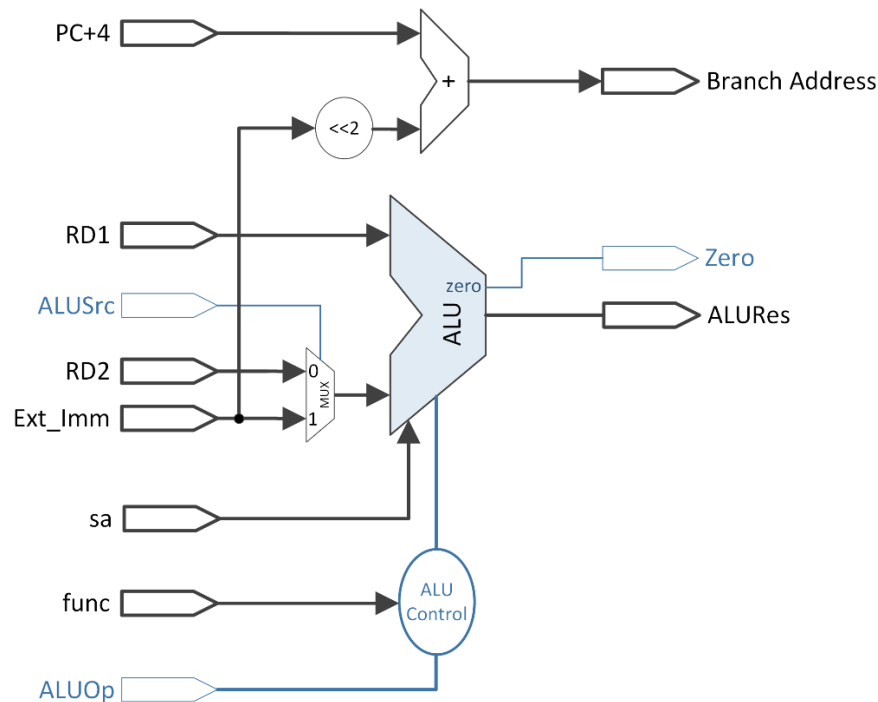


Figure 7-5: Instruction Execute Data-Path for MIPS 32

The EX unit provides, as output, the ALU Result used for writing the result of arithmetical/ logical instructions in the Register File or used as the address for the Data Memory in the case of lw and sw instructions. In addition, the ALU provides another output, the Zero Signal, which indicates whether the result of the ALU is equal to zero or not (if the result is equal to zero the signal will have as value 1, otherwise 0). For simplicity of your future, first version, pipeline implementation, the EX unit also includes the branch target address computation.

The inputs of the Ex unit are:

- Next Sequential Instruction Address (PC+4)
- 32-bit Read Data 1 (RD1)
- 32-bit Read Data 2 (RD2)
- 32-bit Extended Immediate (Ext\_Imm)

- 6-bit function field (func)
- 5-bit shift amount (sa)
- Control Signals:
  - ALUSrc – selects between Read Data 2 and Extended Immediate as input to the second port of the ALU
  - ALUOp – ALU operation code provided by the Main Control Unit

The outputs of the Ex unit are:

- 32-bit Branch target address
- 32-bit ALU result (ALURes)
- 1-bit Zero signal

The meaning of the control signals:

- ALUSrc = 0 → the Read Data 2 signal is the second input for the ALU
- ALUSrc = 1 → the Extended Immediate signal is the second input for the ALU
- ALUOp → is defined by the Main Control Unit according to the operations implemented in the ALU.

The branch target address is computed with the following formula:

Branch Address  $\leftarrow$  PC + 4 + S\_Ext(Imm)  $\ll$  2;

The Zero signal together with the Branch Control Signal is used in order to select between the normal sequential execution of the program (PC + 4) or the branch target address.

ALU Control Unit defines the ALU operations encoded in the ALUCtrl control signal. For I-type instructions, the encoding of the ALUCtrl is simply defined by the ALUOp signal. For R-type instructions, the value of ALUCtrl is defined by the fixed value ALUOp and the function field.

### 7.3. Laboratory Assignments

Read carefully and completely each activity before you begin!

Prerequisites:

- All the assignments from the laboratories 4, 5, 6 completed
- The instruction fetch unit implemented and tested on the Digilent Development Board.
- The instruction decode unit implemented and tested on the Digilent Development Board.
- Xilinx project with “test\_env” including the IF and ID units (Laboratory 6 Assignment 6.3.3)

Attention: If the homework from the previous laboratories is not completed, you will receive a 1 for this and all future laboratories until the homework is done without the possibility of any corrections to the mark!

### 7.3.1. Instruction Execute Unit design

Taking into account the instruction execution data-path from Figure 7-5 design a new component (new entity) in the “test\_env” project for your own single-cycle MIPS 16. All the data fields are 16-bits wide.

The Ex entity will contain the hardware components described in Figure 7-5 that will not be implemented with other components (use behavioral VHDL description)

Use one line of code for the branch target address computation (no shift required only addition).

Use a process (with a case statement) for the ALU implementation as in laboratory 2. The 1-bit shift amount is used only for the logical and / or arithmetic shift operations.

Use a process (with a case statement) for the implementation of the ALU Control. The encoding of the ALU Ctrl control signal is dependent on your own 15 instructions and defines the arithmetic-logical operations implemented by the ALU.

**Attention!** Be careful when transforming the data fields from Figure 7-5 (MIPS 32) into your own single-cycle MIPS 16 implementation.

### 7.3.2. Testing of the Instruction Execute Unit

Instantiate the Execution Unit in the “test\_env” project. At this moment, you will connect the output of the Execution Unit (AluRes) to the Write Data port of the ID Unit (WD input)

You have to test all your arithmetical / logical instructions on the Digilent Development Board: Add, Sub, Shift left, Shift Right, And, Or, Addi, etc. Make sure that all the instructions perform correctly.

All the signals present on the data-path must be connecting to the SSD, i.e. the outputs of the IF, ID and EX units. Use switches in order to control the display on the SSD (multiplexor):

- $sw(7:5) = 000 \rightarrow$  display the instruction on the SSD
- $sw(7:5) = 001 \rightarrow$  display the next sequential PC ( $PC + 1$  output) on the SSD
- $sw(7:5) = 010 \rightarrow$  display the RD1 signal on the SSD
- $sw(7:5) = 011 \rightarrow$  display the RD2 signal on the SSD
- $sw(7:5) = 100 \rightarrow$  display the Ext\_Imm signal on the SSD

- $sw(7:5) = 101 \rightarrow$  display the ALURes signal on the SSD
- $sw(7:5) = 111 \rightarrow$  display the WD signal on the SSD

On the LEDs, you will display the control signals from the Main Control Unit. You have 8 x 1-bit control signals and ALUOp. Use another switch to control the display on the LEDs:

- $sw(0) = 0 \rightarrow$  Display the 1-bit control signals on the LEDs. The order of the control signals is your own choice.
- $sw(0) = 1 \rightarrow$  Display the n-bit ALUOp signal on the LEDs (the rest of LEDs will have the value '0' for now)

Trace your program (without any memory operation or conditional /unconditional jump) on the Digilent Development Board instruction by instruction. Be sure that all the control signals / data fields are correct.

## 7.4. References

- [1] Computer Architecture Lectures 3 & 4 slides.
- [2] D. A. Patterson, J. L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface", 5<sup>th</sup> edition, ed. Morgan-Kaufmann, 2013.
- [3] D. A. Patterson and J. L. Hennessy, "Computer Organization and Design: A Quantitative Approach", 5<sup>th</sup> edition, ed. Morgan-Kaufmann, 2011.
- [4] MIPS® Architecture for Programmers, Volume I-A: Introduction to the MIPS32® Architecture, Document Number: MD00082, Revision 5.01, December 15, 2012
- [5] MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual, Revision 6.02
- [6] MIPS32® Architecture for Programmers Volume IV-a: The MIPS16e™ Application-Specific Extension to the MIPS32™ Architecture, Revision 2.62.
  - Chapter 3: The MIPS16e™ Application-Specific Extension to the MIPS32® Architecture.