Laboratory 07

Single-Cycle MIPS CPU Design – smaller: 16-bits version
One clock cycle per instruction

1. Objectives

Study, design, implement and test
- **Instruction Execute Unit for the 16-bit Single-Cycle MIPS CPU**
- **Memory Unit for the 16-bit Single-Cycle MIPS CPU**
- **Write Back Unit for the 16-bit Single-Cycle MIPS CPU**
- **Other necessary connections for jump address computation**

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**

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, Memory Unit, Write Back Unit and the rest of the connections 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.

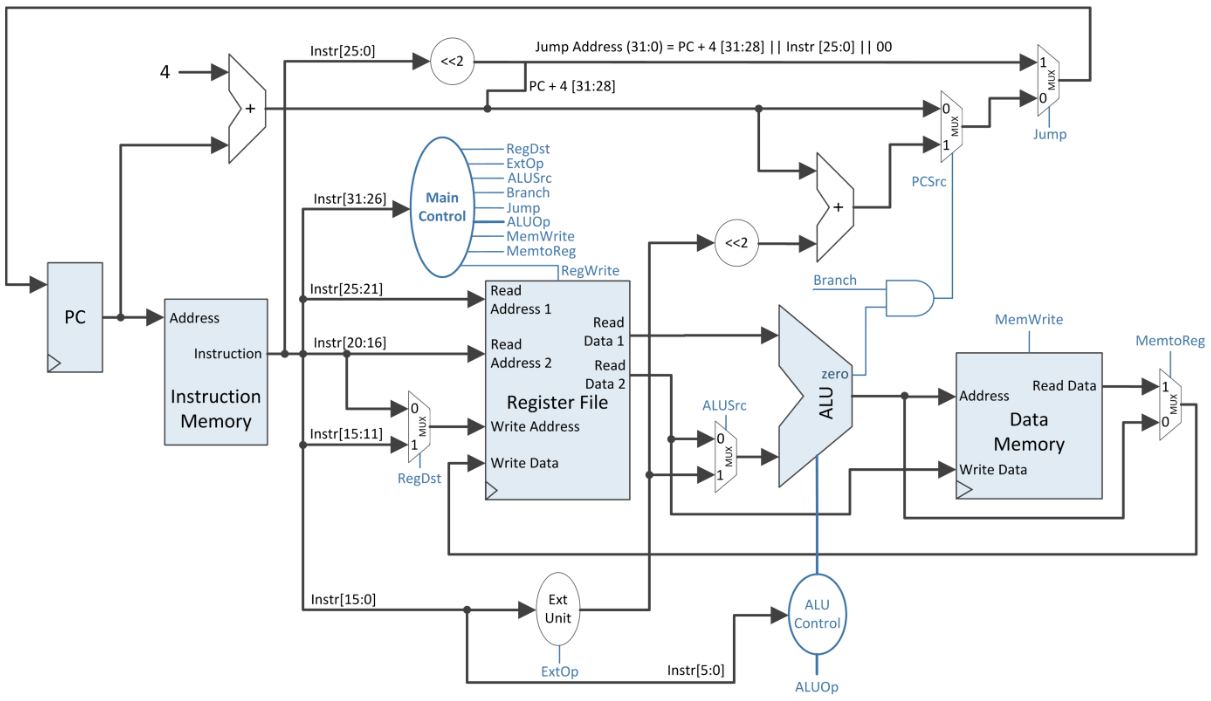

Figure 1: MIPS 32 Single-Cycle Data-Path + Control

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

3	3	3	3	1	3
opcode	rs	rt	rd	sa	function

Figure 2: R-type Instruction format

3	3	3	7
opcode	rs	rt	address / immediate

Figure 3: I-type Instruction format

3	13
opcode	target address

Figure 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 02 and 04 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 5.

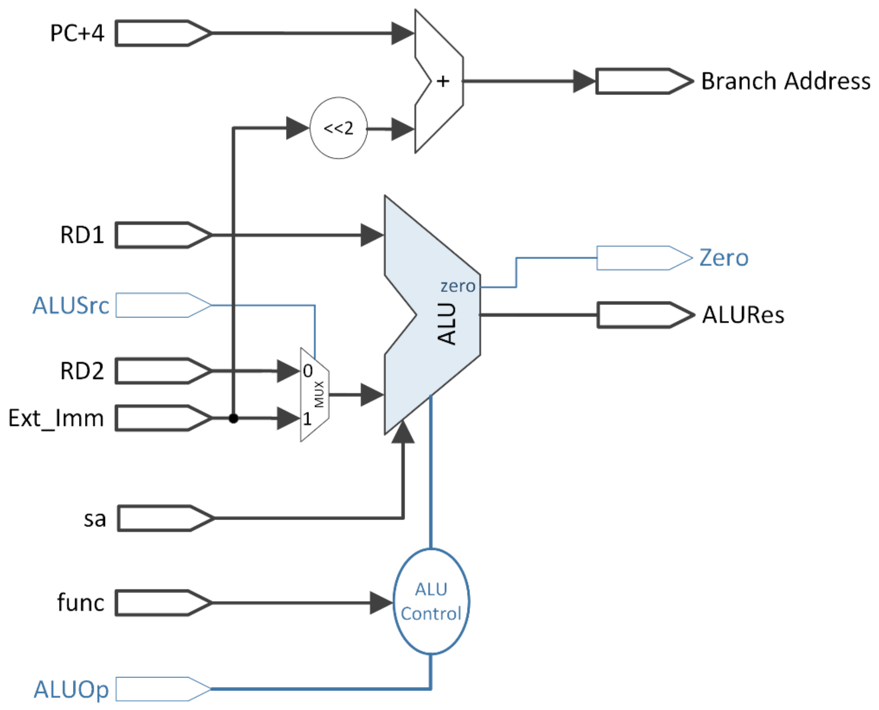

Figure 5: Instruction Execute Data-Path for MIPS 32

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 ← PC + 4 + S_Ext(Imm) << 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.

The Memory Unit consist in the following component

- Data Memory

The data-path of the Memory Unit is presented in Figure 6.

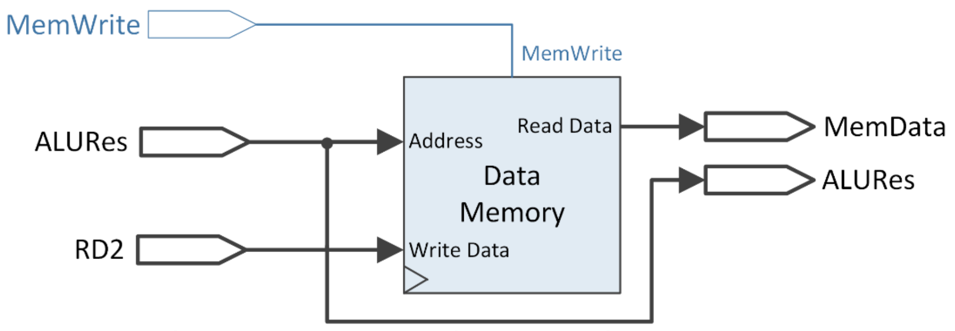

Figure 6: Memory Unit Data-Path for MIPS 32

The Data Memory is a RAM with asynchronous read and synchronous write operations. A similar RAM implementation (with synchronous read) was done in laboratory 03.

The inputs of the Memory Unit:

- The clock signal (for the Data Memory Writes)
- 32-bit ALURes signal – consists in the address for the Data Memory
- 32-bit RD2 signal – the second output of the Register File (used only for store word instructions) is the Write Data field for the Data Memory
- MemWrite control signal

The outputs of the Memory Unit:

- 32-bit MemData, the Read Data from the Data Memory (used only for load word instructions).
- 32-bit ALURes, this signal is also the result of the arithmetic-logical instructions that must be stored in Register File, so it is also fed as output for the Memory Unit and input to the Write Back Unit.

The only control signal present in this stage is the MemWrite Control Signal.

- MemWrite = 0 → Nothing is written in the Data Memory.
- MemWrite = 1 → The RD2 signal is written in the Data Memory at the address indicated by the ALURes signal.

The Write Back unit is simply the last multiplexor from Figure **1** and the rest of the components are: the AND gate for generating the PCSrc control signal, the jump address computation.

The control signal for the multiplexor is MemtoReg and identifies what value is fed to the Write Data port of the Register File in the ID state:

- MemtoReg = 0 → the ALURes signal is the input to the Write Data port of the Register File.
- MemtoReg = 1 → the MemData is the input to the Write Data port of the Register File.

PCSrc ← Branch and Zero;

3. Laboratory Assignments

Read carefully and completely each activity before you begin!

Prerequisites:

- All the assignments from the laboratories 04, 05, 06 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 (Lab 06 Assignment 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!

3.1. Instruction Execute Unit design

Taking into account the instruction execution data-path from Figure 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 **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 02. 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 ALUCtrl 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 5 (MIPS 32) into your own single-cycle MIPS 16 implementation.

3.2. Memory Unit Design

Describe a new component (new entity) for the Memory Unit. Use the RAM implementation from laboratory 03 and change the read operation to an asynchronous one. Write only with processes inside the Memory Unit.

3.3. Adding the rest of the logic necessary for the processor

Instantiate the Execution Unit and the Memory Unit in the "test_env" project. Connect all the signals from the Ex Unit and Memory Unit in the data-path. The MemWrite signal should be validated with an output of the MPG component as it was previously implemented for the writing in the Register File (RegWrite signal).

Add the write back multiplexor for your own MIPS processor. Use only one line of code to implement this multiplexor.

Complete your own MIPS processor implementation with the jump address computation and the PCSrc signal computation. Complete all the necessary connections for the data-path as in Figure **1** (jump address, branch target address, write back in the register file, etc.).

3.4. Testing it ALL: You own Single-Cycle MIPS 16 processor

At this moment you should have all the components form the data-path implemented in the “test_env” project.

For connecting to the SSD use all the signals present on the data-path, i.e. the outputs of the IF, ID, Ex, M and WB units. Use switches in order to control the display on the SSD (multiplexor):

- sw(7:5) = 000 → display the instruction on the SSD
- sw(7:5) = 001 → display the next sequential PC (PC + 1 output) on the SSD
- sw(7:5) = 010 → display the RD1 signal on the SSD
- sw(7:5) = 011 → display the RD2 signal on the SSD
- sw(7:5) = 100 → display the Ext_Imm signal on the SSD
- sw(7:5) = 101 → display the ALURes signal on the SSD
- sw(7:5) = 110 → display the MemData signal on the SSD
- sw(7:5) = 111 → 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 → Display the 1-bit control signals on the LEDs. The order of the control signals is your own choice.
- sw(0) = 1 → Display the n-bit ALUOp signal on the LEDs (the rest of LEDs will have the value ‘0’ for now)

If needed for debugging the processor on the Digilent Development Board you can additionally display other signals on the SSD / LEDs: branch target address, jump address, ALUCtrl, etc.

Now trace your program on the Digilent Development Board instruction by instruction. Be sure that all the control signals / data fields are correct.

4. References

- Computer Architecture Lectures 3 & 4 slides.
- MIPS® Architecture For Programmers, Volume I-A: Introduction to the MIPS32® Architecture, Document Number: MD00082, Revision 5.01, December 15, 2012
- MIPS® Architecture For Programmers Volume II-A: The MIPS32® Instruction Set Manual, Revision 6.02
- 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.