HO CHI MINH UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND COMPUTER ENGINEERING

----------o0o----------

REPORT ASSIGNMENT

Name of students:

- Nguyen Tien Anh: 1752076
- Nguyen Duc Khoi: 1752302

Major: Computer Engineering

Lecturer: Pham Quoc Cuong

Table of Contents

1 Introduction

1.1 Verilog

Verilog, standardized as IEEE 1364, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits, as well as in the design of genetic circuits. In 2009, the Verilog standard (IEEE 1364-2005) was merged into the SystemVerilog standard, creating IEEE Standard 1800-2009.

1.2 MIPS

MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA): A-1:19 developed by MIPS Computer Systems (an American company that is now called MIPS Technologies). There are multiple versions of MIPS: including MIPS I, II, III, IV, and V; as well as five releases of MIPS32/64 (for 32- and 64-bit implementations, respectively). The early MIPS architectures were 32-bit only; 64-bit versions were developed later.

In this assignment, we will try to implement a simple version of 32-bit MIPS (32 bits long instructions and 32 registers in the register file) on a De2i-150 development kit using verilog description language. Final result we be tested on the board showing different signal on built-in 7 segment LEDs. Our design is able to perform some simple R-type instructions as well as a couple of instructions from other types. These instructions include: addi, add, substract, and, or, set on less than, conditional branch and unconditional jump. The design can only perform one instruction for every clock cycle.

2 Problem I

Algorithm

```
procedure bubbleSort (list: array of items)
   loop = list.count;
   for i = 0 to loop-1 do:
      swapped = false
```

```
        for j = 0 to loop-1 do:
           if (list[j] < list[j+1]) then
              swap (list[j], list[j+1])
              swapped = true
           end if
        end for
        if (not swapped) then
           break
        end if
     end for
  end procedure return list
```

Description

Bubble sort is sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. The flag variable *swapped* which will help us see if any swap has happened or not. If no swap has occurred, i.e. the array requires no more processing to be sorted, it will come out of the loop.

Input	Number of instructions
1,2,3,4,5,6,7,8,9,10	575
1,2,3,4,5,10,9,8,7,6	471
10,9,8,7,6,5,4,3,2,1	86

3 Problem II

3.1 Design

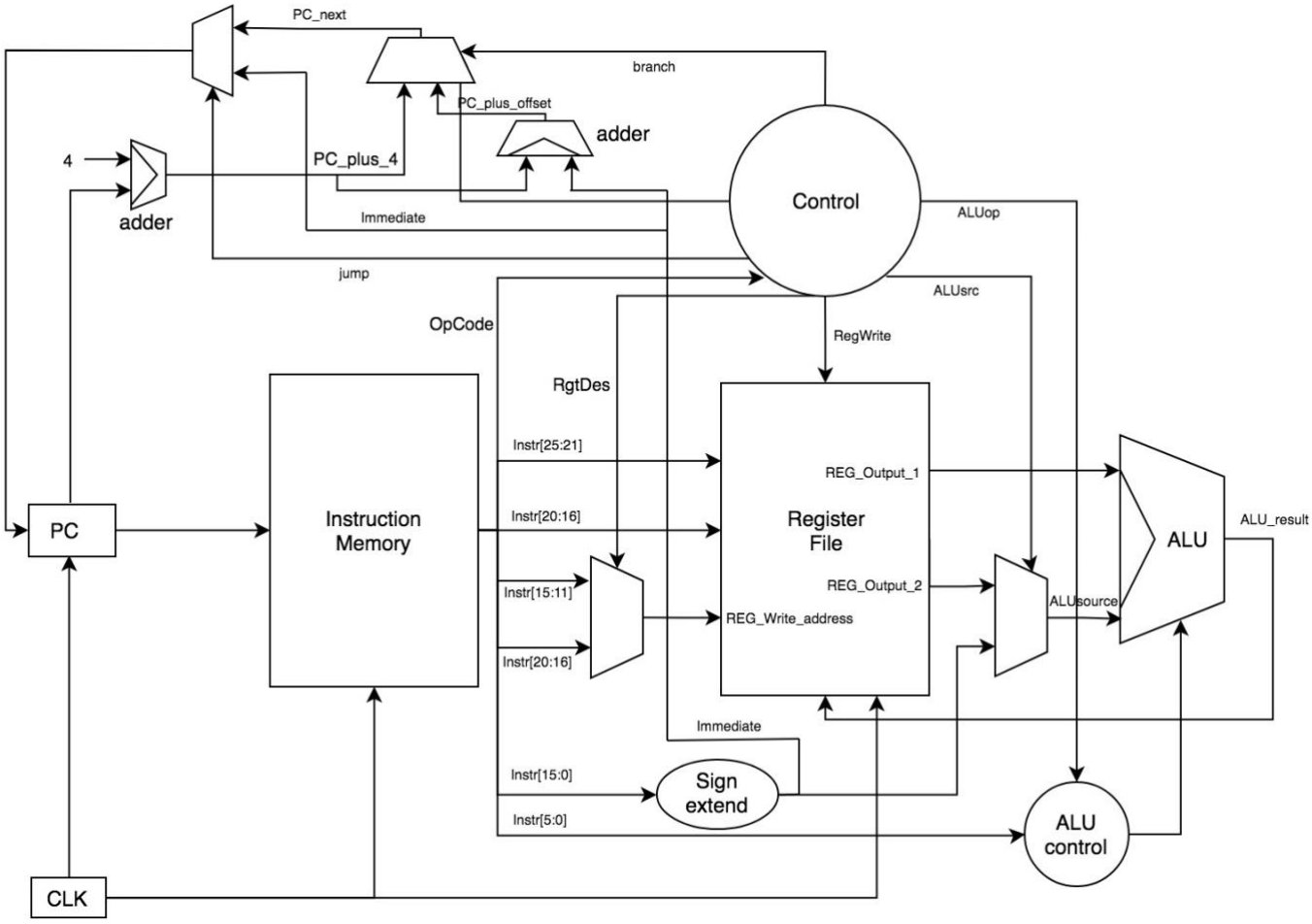

Figure 1: Schematic

Description

The above figure shows the organization of our simple MIPS design with multiplexors and control unit. Although the figure shows all the modules of the data path, some of them are not explicitly built in the actual implementation. For example, the multiplexor module to select the next data for PC is constructed in the final module by using one “always” block without any explicit module instantiation. We also omit the shift-2 module in branch and jump instructions since we can do it with a simple wiring.

3.2 Implement

3.2.1 Module Instruction Memory

Interface

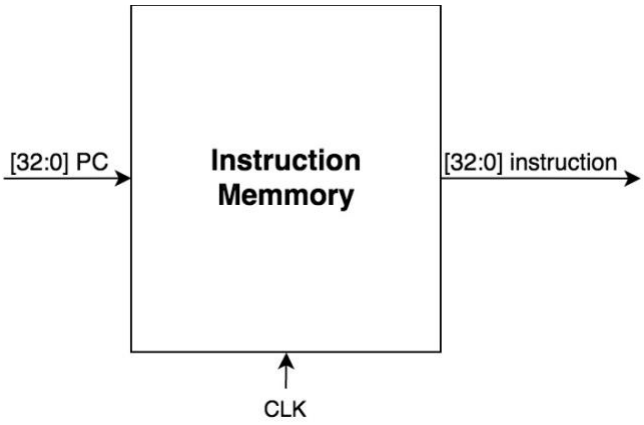

Figure 2: Module Instruction Memory

Description

Input of this module is an address of an instruction get from the PC and output is the instruction with the corresponding address.

3.2.2 Module Register File

Interface

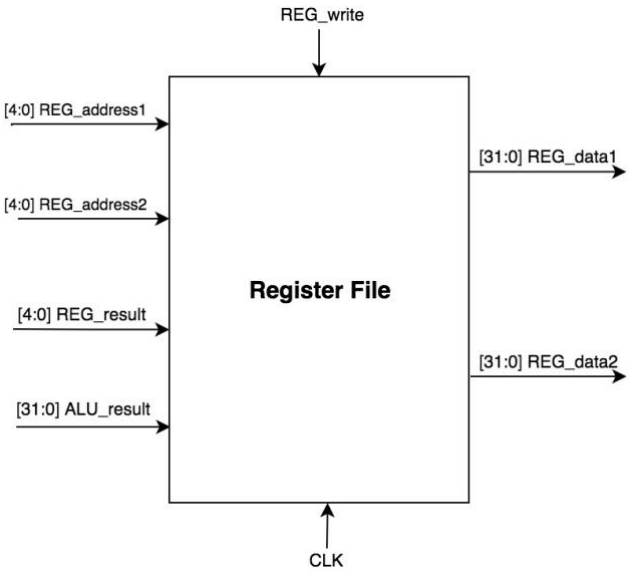

Figure 3: Module Register Files

Description

Input of this module are register numbers from an instruction as well as the result from the module ALU, and the output are data which is stored in these registers. If the signal *REG_write* is active, the *ALU_result* will be written to the register result.

3.2.3 Module ALU

Interface

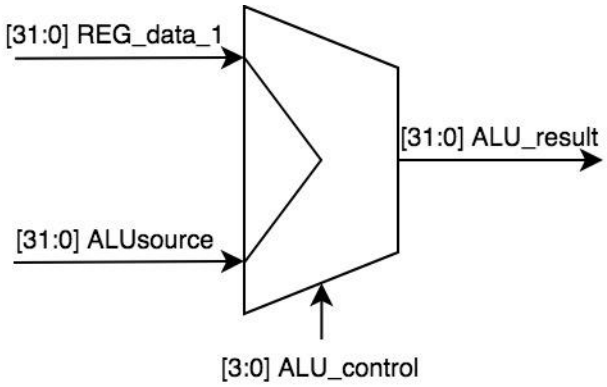

Figure 4: Module ALU

Description

This module is the arithmetic logic unit which performs these operations: *add, sub, and, or, slt, branch, jump* based on the control signal. Output is the result of the operation, which is decided by control signal, between two received input values. This module also has the output *Zero*, indicating whether the result is equal to zero. Each operation requires one distinct control signal:

and	4'b0000
or	4'b0001
add	4'b0010
subtract	4'b0011
set on less than	4'b0111
beq	4'0110

and and *or* operations are constructed within a built in operation in Verilog, while *add* and *subtract* are built from Full-adder and Full_subtractor model. For *set on less than* and *beq* operations, we just need to perform the subtraction between 2 values and output the necessary signal. Above step can be built within one always block.

3.2.4 Module Control Unit

Interface

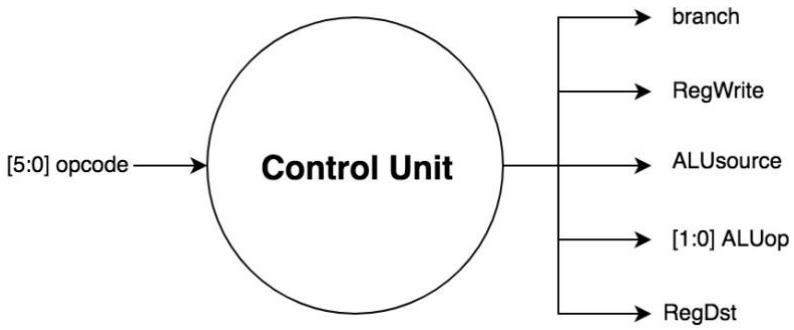

Figure 5: Module Control Unit

Description

The Control Unit will generate all the control signals for other module in the design including (*register file*, *ALU*, and all almost all the multiplexors). It receives a portion of 5 bits signal (*opcope*) from instructions fetched from *instruction memory* as an input and decides the type of instructions. *ALU* also performs on different input sources (from register and from immediate field). The *Control Unit* handles the fields, input and outputs for modules in the design with different multiplexors using select signal. *Register File* is a sequential element for storing and reading operations. Control signal will control when register file can write on a specific address. Generating process are built combinationally within an always block. Below are the specific functions of different control signal.

RegDes	decide the address for write operation of register file
RegWrite	decide when the register file can write
ALUScr	select the second source for *ALU* (from register or from immediate field)
ALUOp	input for *ALUcontrol*, a submodule used for controlling the ALU operation
Branch	enable branch operation
Jump	enable jump operation

3.3 Result

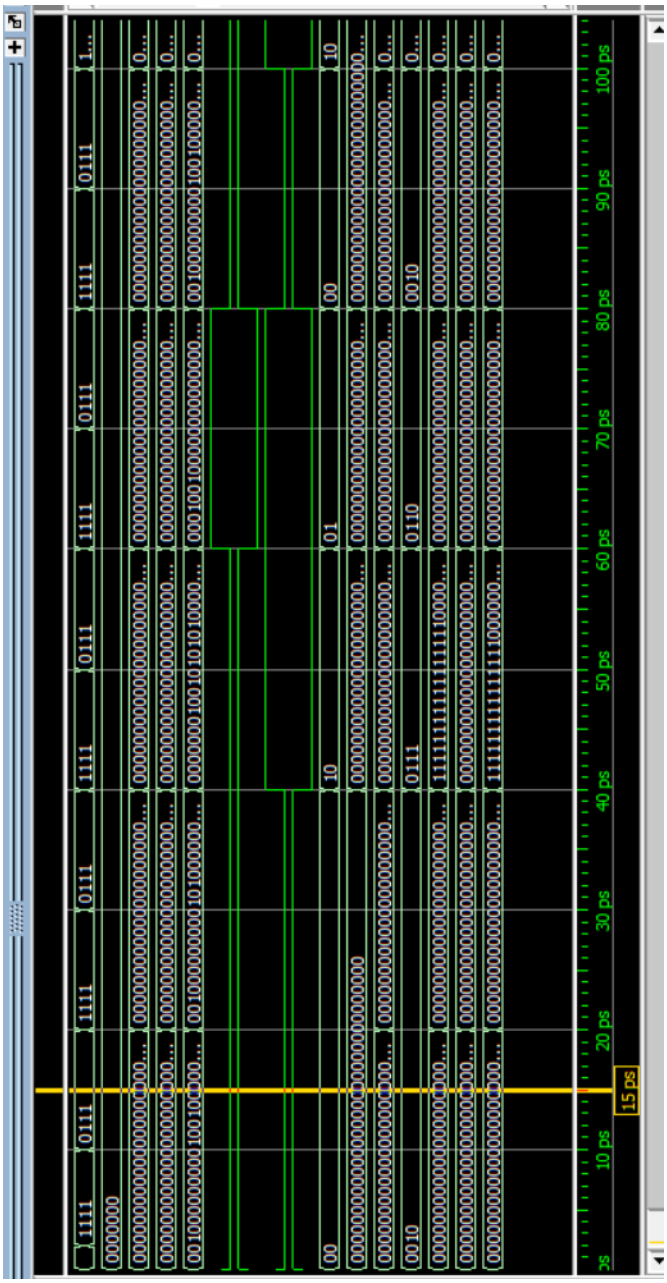

Figure 6: Stimulation on ModelSim

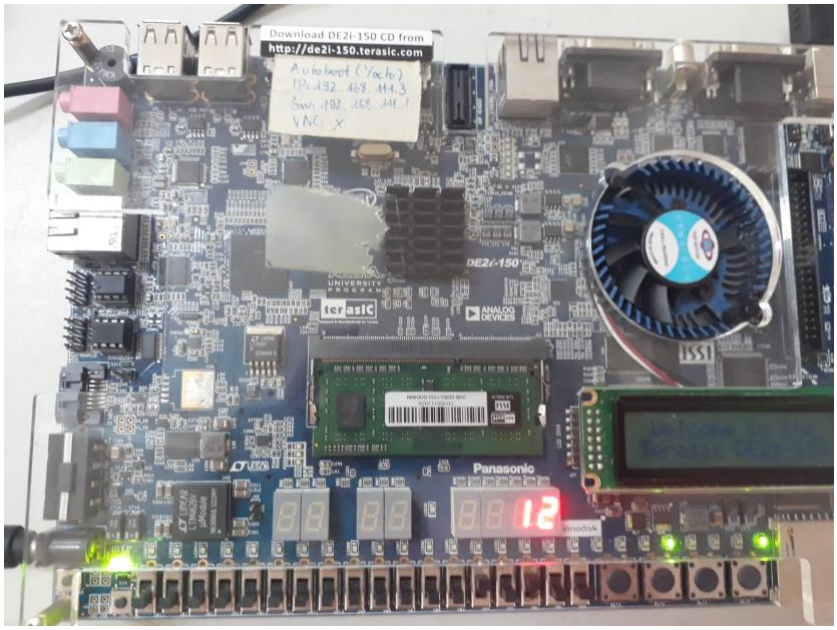

Figure 7: Stimulation on De2i kit

3.4 Conclusion

The assignment requires knowledge about MIPS, the data path and Verilog. In particular, we have used branch, conditional, and memory instructions within a loop to solve the first question. Whereas for the second question, each module is built by Verilog and connect together by the data path model. Moreover, we also implement the design on De2i-150 FPGA development kit via the Quartus software.