ESKISEHIR TECHNICAL UNIVERSITY
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

LAB-5
SINGLE-CYCLE CPU

1 Introduction

In this lab, you will implement a single-cycle MIPS CPU. You can adapt some of the components you implemented on previous labs for this CPU architecture. You will implement it step by step, beginning a CPU that executes a few basic instructions.

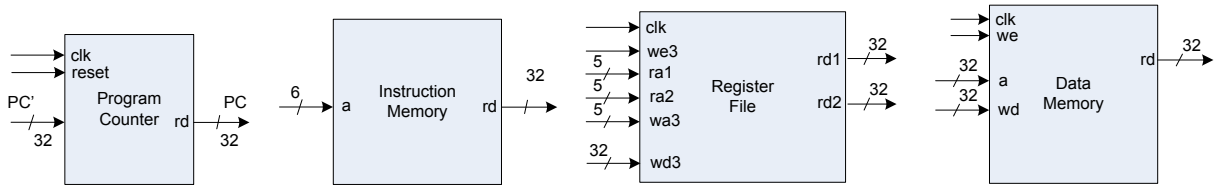
2 Single-cycle CPU Description

The single-cycle microarchitecture executes an entire instruction in one cycle. So, the cycle time is limited by the slowest instruction.

We will divide our microarchitecture into two interacting parts: the datapath and the control. The datapath operates on words of data. It contains structures such as memories, registers, ALUs, and multiplexers. The state elements of the datapath are introduced in Figure 1. MIPS is a 32-bit architecture, so we will use a 32-bit datapath. The control unit receives the current instruction from the datapath and tells the datapath how to execute that instruction. Specifically, the control unit produces multiplexer select, register enable, and memory write signals to control the operation of the datapath.

We will start designing the CPU by connecting the state elements from Figure 1 with combinational logic that can execute the various instructions. Control signals determine which specific instruction is carried out by the datapath at any given time. The controller contains combinational logic that generates the required control signals based on the current instruction.

Figure 1: State Elements of the MIPS Processor



2.1 Program Counter

The program counter is an ordinary 32-bit register. Its output, PC, indicates to the current instruction. Its input, PC', indicates the address of the next instruction. The reset input signal provides us to reset the PC register asynchronously. If reset input is 1 then PC is equal to 0. Otherwise, on the rising edge of the clock, input is loaded to the output.

2.2 Instruction Memory

The instruction memory has a single read port.1 It takes a 6-bit instruction address input, a, and reads the 32-bit data, instruction, from that address onto the read data output, rd. It consists of array of std_logic_vector. If there is a change on the address, the data on this address is loaded to the rd output. You can define an array as following.

```
type ROM_Array is array (0 to 63) of std_logic_vector(31 downto 0);
```

2.3 Register File

The 32-element x 32-bit register file has two read ports and one write port. The read ports take 5-bit address inputs, ra1 and ra2, each specifying one of $2^5 = 32$ registers as source operands. They read the 32-bit register values onto read data outputs rd1 and rd2, respectively. The write port takes a 5-bit address input, wa3, a 32-bit write data input, wd3, a write enable input, we3 and lastly a clock. If the write enable is 1, the register file writes the data into the specified register on the rising edge of the clock. The register file is read combinationally. If there is a change on the address, the new data is written on the output rd buses. The read process does not wait the rising edge of the clock.

2.4 Data Memory

The data memory has a single read/write port. If the write enable, we, is 1, it writes data wd into address a on the rising edge of the clock. If the write enable is 0, it reads address a onto rd. Read operation is performed combinationally. You can easily define a memory array as following for the data memory.

```
type ram_type is array (0 to 255) of std_logic_vector(31 downto 0);
```

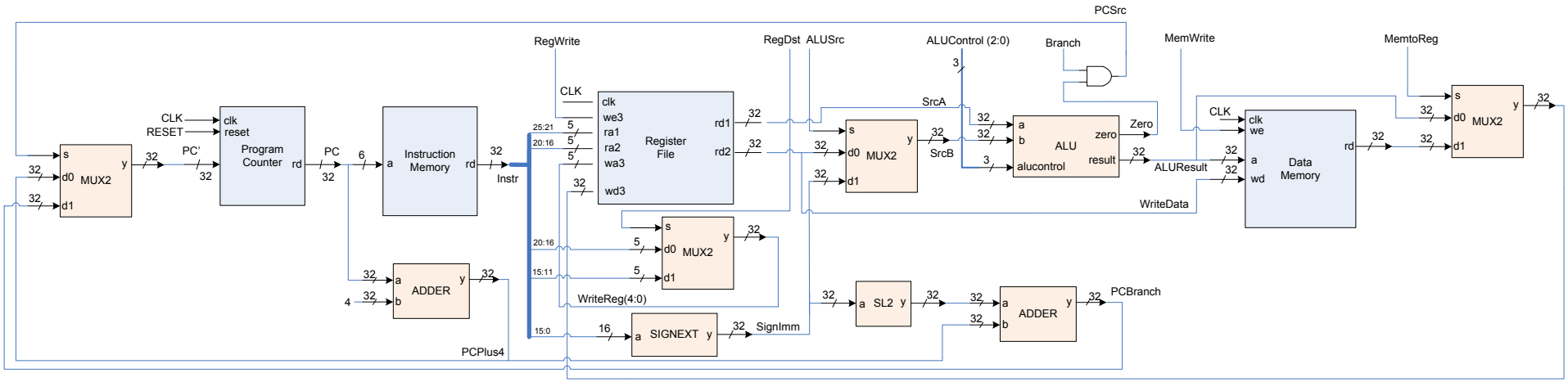
2.5 Read and Write Process

The instruction memory, register file, and data memory are all read as combinational. In other words, if the address changes, the new data appears at rd after some propagation delay. No clock is involved. They are written only on the rising edge of the clock. The state of the system is changed only at the clock edge. The address, data, and write enable must setup sometime before the clock edge and must remain stable until a hold time after the clock edge.

3 Single-Cycle Datapath

The figure given below shows the datapath of the MIPS processor. After you finish writing the components inside the datapath, you will connect them according to the figure. The datapath given here provides just a few instructions. If you extend your instruction set, you should change or add some components into this design.

Figure 2: Datapath of the MIPS Processor



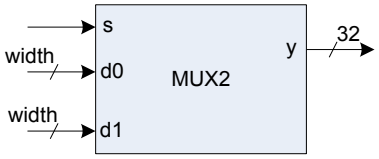
3.1 The Components of the Datapath

The pink colored components are the components of the datapath. They are all combinational systems.

3.1.1 Multiplexers

The datapath consists of 2x1 components. The multiplexers are combinational circuits that select binary information from one of the inputs according to the select input and direct it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

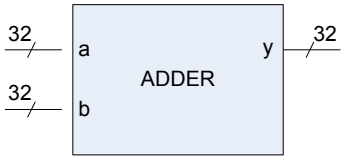
Figure 3: Entity of the 2x1 Multiplexer



3.1.2 ADDER

Adder performs addition operation on its operands combinationally. The figure given below shows the entity of the ADDER component.

Figure 4: Entity of the ADDER



3.1.3 Shift Left 2 Unit (SL2 Component)

The unit performs 2-bit shift left operation on its operand. The operation result is written on the output combinationally. The result means actually that input operand is multiplied by 4.

Figure 5: Entity of the SL2

3.1.4 Sign Extender Unit (SIGNEXT Component)

Sign Extender Unit extends the 16-bit signal to the 32-bit signal. It assigns the 16 most significant bits of the output signal to the 0 or 1 depends on the sign bit of the input. If the sign bit, most significant bit, of the input is 1, then It puts 1 to the 16 most significant bits of the output signal. If the sign bit, most significant bit, of the input is 0, then It puts 0 to the 16 most significant bits of the output signal.

Figure 6: Entity of the SIGNEXT

3.1.5 ALU

The ALU component performs some arithmetic operations on its input operands depends on the Table 1. It shows which operation is done according to the F(2:0). You are familiar to write the vhdl description of an ALU from previous labs. Zero input is necessary for branch instructions. It indicates whether result of the ALU is zero or not.

Figure 7: Entity of the ALU

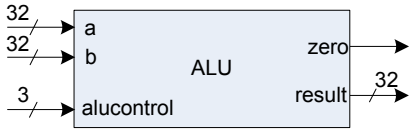
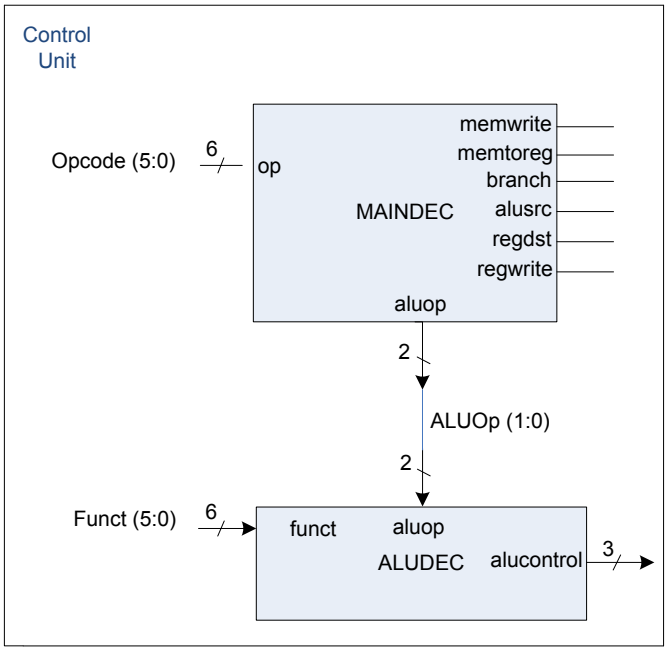


Table 1: ALU Operations

F(2:0)	Function
000	A AND B
001	A OR B
010	A + B
011	not applicable
100	A AND $\bar{B}$
101	A OR $\bar{B}$
110	A - B
111	SLT

4 Single-Cycle Control

Figure 8: Internal structure of the Control Unit



The control unit computes the control signals depends on the opcode and funct fields of the instruction, Instr (31:26) and Instr (5:0) You can see necessary control signals attached to the datapath on the Figure 2.

Most of the control information comes from the opcode, however for R-type instructions also the funct field is used to determine the ALU operation. Therefore, we will simplify our design by dividing our control unit design into two blocks of combinational logic, as shown in Figure 3. The main decoder generates most of the output signals from the opcode. It also determines a 2-bit ALUOp signal. The ALU decoder uses this ALUOp signal together with the funct field to generate ALUControl. The meaning of the ALUOp signal is given in Table 1 below. Table 2 is a truth table for the ALU decoder.

Table 2: ALUOp Encoding table

ALUOp	Meaning
00	Add
01	Subtract
10	Look at funct field of the instr
11	Not applicable

Table 3: Truth table for ALU Decoder

ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)

The meanings of the three ALUControl signals are given in Table 1. Because ALUOp is never 11, the truth table can use don't care's X1 and 1X instead of 01 and 10 to simplify the logic. When ALUOp is 00 or 01, the ALU should add or subtract, respectively. When ALUOp is 10, the decoder examines the funct field to determine the ALUControl. Note that, for the R-type instructions we implement, the first two bits of the funct field are always 10, so we may ignore them to simplify the decoder.

Table 4 is a truth table for the main decoder that summarizes the control signals as a function of the opcode. All R-type instructions use the same main decoder values; they differ only in the ALU decoder output. For instructions that do not write to the register file such as sw and beq nstructions, the RegDst and MemtoReg control signals are don't cares (X). The address and data to the register write port do not matter because RegWrite is not asserted. The logic for the decoder can be designed using your favorite techniques for combinational logic design.

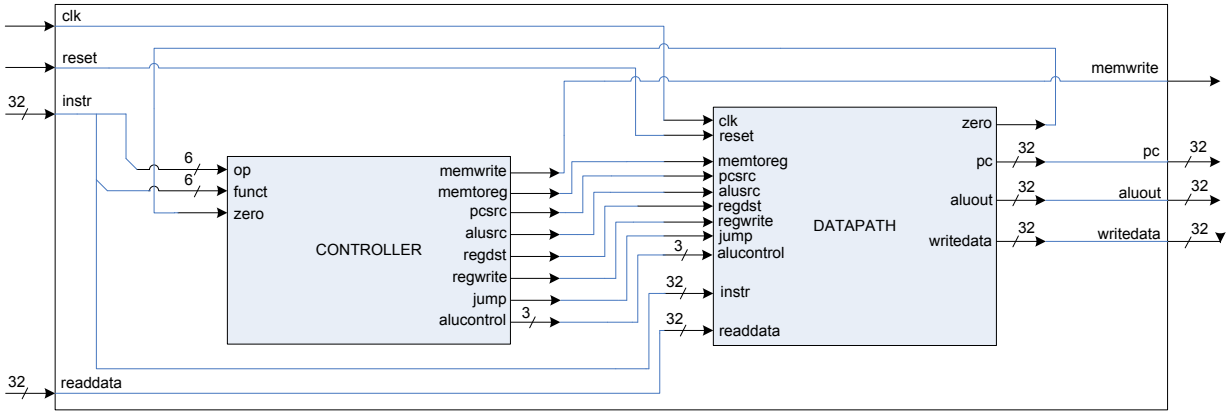
Table 4: The truth table of the Main Decoder

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
R-Type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01

5 MIPS Processor

The datapath and controller units of the MIPS processor are connected as shown in the Figure below.

Figure 9: MIPS Processor



6 Instruction Types

6.1 R-Type Instructions

The name R-type is short for register-type. R-type instructions use three registers as operands: two as sources, and one as a destination. Figure 10 shows the R-type machine instruction format.

Figure 10: R-type Machine instruction format

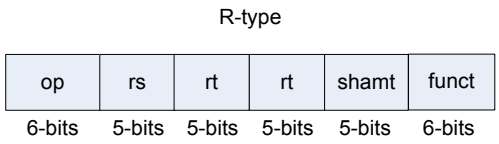


Table 5: R-Type instructions

Opcode	Name	Description	Operation
100000	add	add	[rd] = [rs] + [rt]
100010	sub	subtract	[rd] = [rs] - [rt]
100100	and	and	[rd] = [rs] & [rt]
100101	or	or	[rd] = [rs] | [rt]
101010	slt	set less than	[rs] = [rt] ? [rd] = 1 : [rd] = 0

Your CPU is responsible to perform the R-type instructions given in Table 5.

6.2 I-Type Instructions

The name I-type is short for immediate-type. I-type instructions use two register operands and one immediate operand. Figure 11 shows the I-type machine instruction format.

Figure 11: I-type Machine instruction format

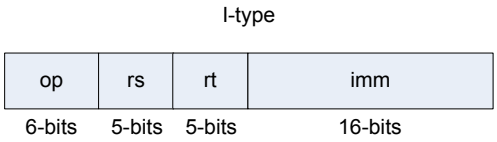


Table 6: I-Type instructions

Opcode	Name	Description	Operation
100011	lw	load word	[rt] = [Address]
101011	sw	store word	[Address] = [rt]

Your CPU is responsible to perform the I-type instructions given in Table 6.

6.3 J-Type Instructions

The name J-type is short for jump-type. This format is used only with jump instructions. Figure 12 shows the J-type machine instruction format.

Figure 12: J-type Machine instruction format

Table 7: J-Type instructions

Opcode	Name	Description	Operation
000100	beq	Branch if equal	if ([rs] == [rt]) PC = Branch Address

Your CPU is responsible to perform the J-type instruction given in Table 7.

7 Procedure

In this lab, you will design a 32-bit single-cycle MIPS Processor. It consists of combinational and sequential components.

- Firstly, you will design all of the components of the CPU architecture explained above.
- Connect them as described above.
- After completed the CPU, you will write an example program which includes the instructions that you are responsible.
- You should simulate your completed design using ISE or another VHDL simulator to prove the correctness of your design. Prepare a short report with the VHDL codes and the simulation results.
- **Bonus Grades** are available for those who add extra instructions into his/her CPU architecture.
  Extra instructions are;

Table 8: Extra instructions

Opcode	Name	Description	Operation	Type
000010	j	jump	PC=Jump Address	J-type
000101	bne	Branch if not equal	if ([rs]!=[rt]) PC = Branch Address	I-Type
001000	addi	Add immediate	[rt] = [rs] + SignImm	I-Type

If you want, you can add more instructions into your CPU architecture to build complete MIPS CPU.

8 Figures

9 Tables