VIETNAM NATIONLA UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING
---oOo---

DIGITAL SYSTEM DESIGN WITH HDL
CE213.O12.MTCL.2

LAB REPORT 7

VIETNAMESE NAME : THIẾT KẾ SIMPLE PROCESSOR
ENGLISH NAME: DESIGING A SIMPLE PROCESSOR

STUDENT NAME: Trương Duy Đức
STUDENT ID: 21521970
Lecturer: Tạ Trí Đức

Contents

Pictures

I. Content of Lab

1.1. Objectives

Using the Verilog HDL language, design a simple Processor.

1.2. Practice content

Base on DATAPATH design block and the Control Unit design block, along with the ALU Control design block, to design a simple Processor capable of executing the following instructions:

- add $1, $2, $3
- lw $1, 0($2)
- sw $1, 0($2)"

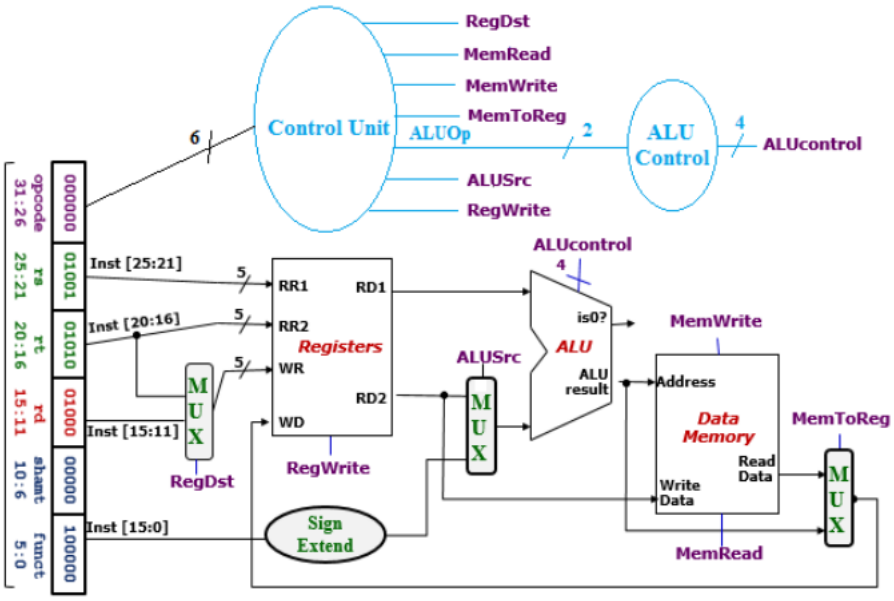

Figure 1: Data path and Control Unit according to the MIPS architecture.

II.THEORETICAL BASIS

1. Processor

"A Simple Processor" is a general concept to describe a straightforward processing unit, often designed to illustrate basic principles of computer architecture and processing. A Simple Processor typically includes a limited set of instructions and functionalities, aiding learners or beginners in gaining a better understanding of how computers operate.

A Simple Processor usually consists of fundamental components such as:

1. **Central Processing Unit (CPU):** The core of any processing system, executing instructions and managing data flow.
2. **Registers:** Quick and close-to-CPU temporary storage used for holding data temporarily.
3. **Memory:** Larger storage area where programs and data are stored.
4. **Control Unit:** Oversees CPU operations, controls instruction execution, and manages data flow.
5. **Instruction Set:** A set of instructions that the CPU can execute.

Simple Processors are often employed for educational purposes, simulation, or in applications where a complex processing unit is unnecessary. Grasping the workings of a Simple Processor helps in understanding how computers perform basic tasks such as storing, computing, and controlling instruction flow.

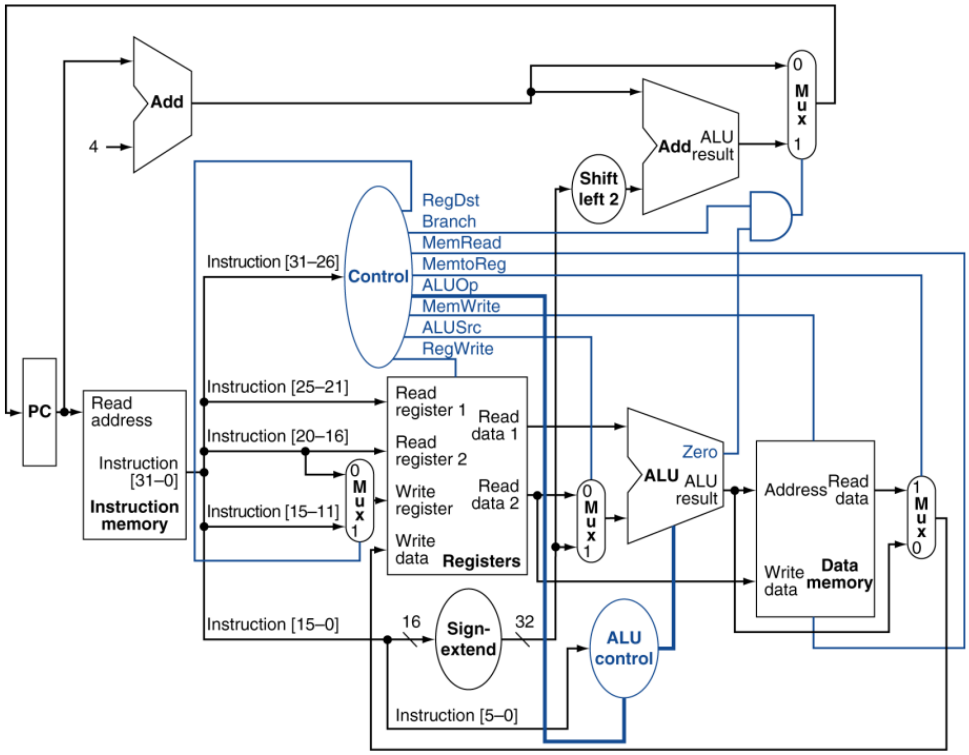

Figure 2: " Data Path" and "Control Unit" in the MIPS architecture

III.PRACTICAL

1. Overview

In this lab, we only need to complete the connections of the simple Processor using the simple DATAPATH (lab 4) and the simple Control Unit (lab 5).

There is a 32-bit INSTRUCTION as input, and each instruction group will have its own specific function, as shown in the Figure below.

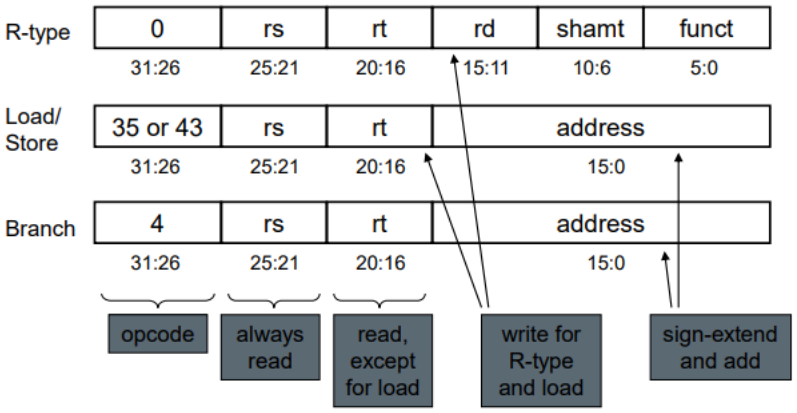

Figure 3: Instruction Formats

Signal paths of the R-type instruction group (ADD):

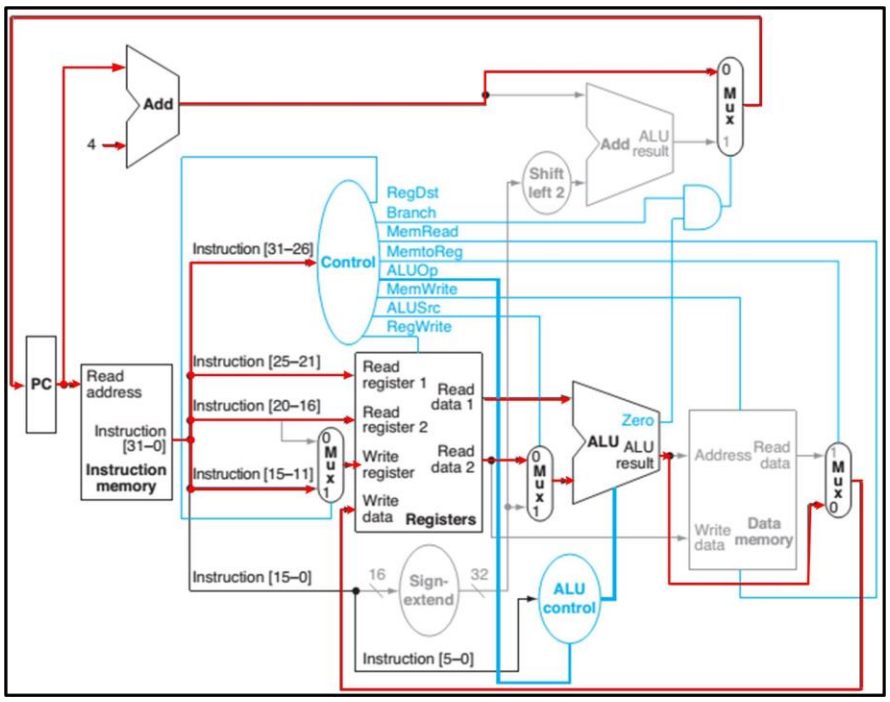

Figure 4:The bold (red) lines represent the active paths during the execution of the ADD instruction.

Signal paths of the lw instruction:

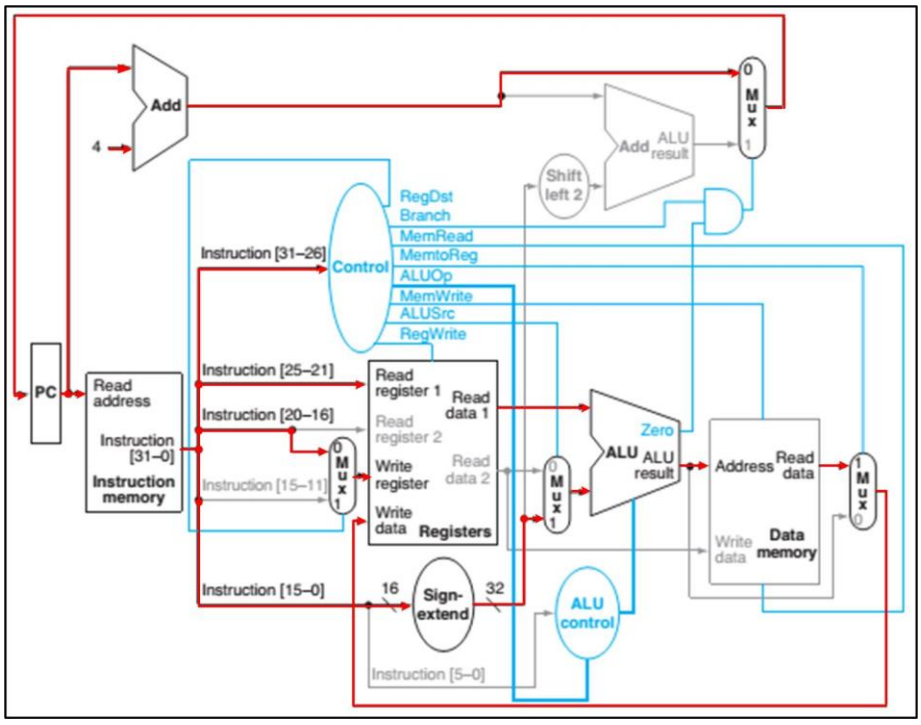

Figure 5:The bold (red) lines represent the active paths during the execution of the lw instruction.

Signal paths of the sw instruction:

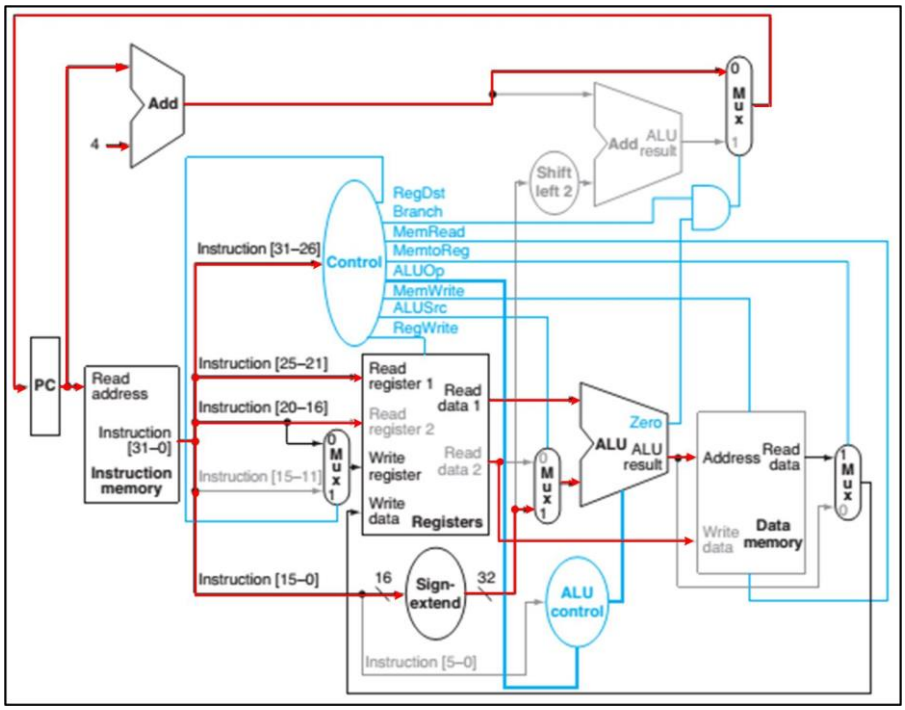

Figure 6:The bold (red) lines represent the active paths during the execution of the sw instruction.

2. CODE

2.1. CODE LAB07- MAIN

```
module
LAB07(INST,CLK,OUT);//,REGDST,ALUSRC,MEMWRITE,MEMREAD,MEMTOREG,R
EGWRITE);

input [31:0] INST;
input CLK;
output [31:0] OUT;

 wire REGDST,ALUSRC,MEMWRITE,MEMREAD,MEMTOREG,REGWRITE;
 wire [1:0] ALUOP;
 wire [2:0] ALUCONTROL;


DATAPATH
DATAPATH1(.CLK(CLK),.INST(INST[25:0]),.REGDST(REGDST),.ALUSRC(ALUSRC),.A
LUCONTROL(ALUCONTROL),.MEMWRITE(MEMWRITE),.MEMREAD(MEMREAD),.
MEMTOREG(MEMTOREG),.OUT(OUT),.REGWRITE(REGWRITE));

Control_Unit
Control_Unit1(.OP(INST[31:26]),.REGDST(REGDST),.MEMREAD(MEMREAD),.MEMW
RITE(MEMWRITE),.MEMTOREG(MEMTOREG),.ALUOP(ALUOP),.ALUSRC(ALUSRC)
,.REGWRITE(REGWRITE));
Alu_control Alu_control1(.ALUOP(ALUOP),.ALUcontrol(ALUCONTROL));

endmodule
```

2.2. CODE DATAPATH

```
module
DATAPATH(CLK,INST,REGDST,ALUSRC,ALUCONTROL,MEMWRITE,MEMREAD,ME
MTOREG,OUT,REGWRITE);

input [25:0] INST;
output [31:0] OUT;
input CLK;
input REGDST,ALUSRC,MEMWRITE,MEMREAD,MEMTOREG,REGWRITE;
input [2:0] ALUCONTROL;
wire [31:0] RD1,RD2,ALURESULT,O_SIGNEXTEND,READDATA,WD;
wire [4:0] WR;

MUX_32_bit #(.WIDTH(5))
mux1(.OP(REGDST),.A(INST[20:16]),.B(INST[15:11]),.O(WR));

REGFILE
REG1(.RWE(REGWRITE),.WA(WR),.WD(WD),.RA1(INST[25:21]),.RA2(INST[20:16]),.C
LK(CLK),.RD1(RD1),.RD2(RD2));

wire [31:0] OM2;
MUX_32_bit #(.WIDTH(32))
mux2(.OP(ALUSRC),.A(RD2),.B(O_SIGNEXTEND),.O(OM2));

ALU ALU1(.A(RD1),.B(OM2),.O(ALURESULT),.OP(ALUCONTROL),.OF());

SRAM
SRAM1(.CLK(CLK),.WE(MEMWRITE),.RE(MEMREAD),.WD(RD2),.RA(READDATA),.
ADDRESS(ALURESULT));

SIGN_EXTEND SignEXTEND(.I(INST[15:0]),.O(O_SIGNEXTEND));

MUX_32_bit #(.WIDTH(32))
mux3(.OP(MEMTOREG),.A(ALURESULT),.B(READDATA),.O(WD));

assign OUT=WD;
endmodule
```

2.3. CODE CONTROL UNIT

```
module
Control_Unit(OP,REGDST,MEMREAD,MEMWRITE,MEMTOREG,ALUOP,ALUSRC,REG
WRITE);

input [5:0] OP;
output reg REGDST,MEMREAD,MEMWRITE,MEMTOREG,ALUSRC,REGWRITE;
output reg [1:0] ALUOP;

always @(*)
		case(OP)
		1://add
		begin
			ALUOP=2'd2;
			ALUSRC=1'b0;
			REGDST=1'b1;
			REGWRITE=1'b1;
			MEMREAD=1'b0;
			MEMWRITE=1'b0;
			MEMTOREG=1'b0;
		end
		3://sub
		begin
			ALUOP=2'd3;
			ALUSRC=1'b0;
			REGDST=1'b1;
			REGWRITE=1'b1;
			MEMREAD=1'b0;
			MEMWRITE=1'b0;
			MEMTOREG=1'b0;
		end
		5://and
		begin
			ALUOP=2'd0;
			ALUSRC=1'b0;
			REGDST=1'b1;
			REGWRITE=1'b1;
			MEMREAD=1'b0;
			MEMWRITE=1'b0;
			MEMTOREG=1'b0;
		end
		7://or
		begin
```

```
                        ALUOP=2'd1;
                        ALUSRC=1'b0;
                        REGDST=1'b1;
                        REGWRITE=1'b1;
                        MEMREAD=1'b0;
                        MEMWRITE=1'b0;
                        MEMTOREG=1'b0;
                end
                4://lw
                begin
                        ALUOP=2'd2;
                        ALUSRC=1'b1;
                        REGDST=1'b0;
                        REGWRITE=1'b1;
                        MEMREAD=1'b1;
                        MEMWRITE=1'b0;
                        MEMTOREG=1'b1;
                end
                2://sw
                begin
                        ALUOP=2'd2;
                        ALUSRC=1'b1;
                        REGDST=1'b0;
                        REGWRITE=1'b1;
                        MEMREAD=1'b0;
                        MEMWRITE=1'b1;
                        MEMTOREG=1'b0;
                end
                default:
                begin
                        ALUOP=2'dx;
                        ALUSRC=1'bx;
                        REGDST=1'bx;
                        REGWRITE=1'bx;
                        MEMREAD=1'bx;
                        MEMWRITE=1'bx;
                        MEMTOREG=1'bx;
                end
        endcase
endmodule
```

2.4. CODE ALU CONTROL

```
module  Alu_control(ALUOP,ALUcontrol);

input [1:0] ALUOP;
output reg[2:0] ALUcontrol;

always @(*)
        case(ALUOP)
                0: ALUcontrol=3'd1;//AND
                1: ALUcontrol=3'd3;//OR
                2: ALUcontrol=3'd5;//ADD
                3: ALUcontrol=3'd6;//SUB
        endcase
endmodule
```

2.5. CODE TESTBENCH

```
`timescale 1ns/100ps
module tb;
reg [31:0]INST;
reg CLK;
wire [31:0] OUT;
LAB07 uut(.INST(INST),.CLK(CLK),.OUT(OUT));
initial begin
#0      CLK=0;
end
initial begin
#0      INST=32'b000001_00010_00001_00001_00000000000;//add $2,$2,$1
#1              $display("OUTPUT :%d",OUT);
#9      INST=32'b000001_00010_00011_00001_00000000000;//add $1,$2,$3
#10     INST=32'b000001_00100_00011_00010_00000000000;//add $2,$4,$3
#10     INST=32'b000011_00010_00011_00001_00000000000;//sub $1,$2,$3
#10     INST=32'b000100_00010_00011_0000000000000000;//lw $3,0($2);
#10     INST=32'b000010_00001_00010_0000000000000000;//sw $2,0($2);
end
always #5 begin
        #0 CLK=~CLK;

end
always #10 begin
#0 $display("OUTPUT :%d",OUT);
end
endmodule
```

3. Simulate

3.1. Waveform

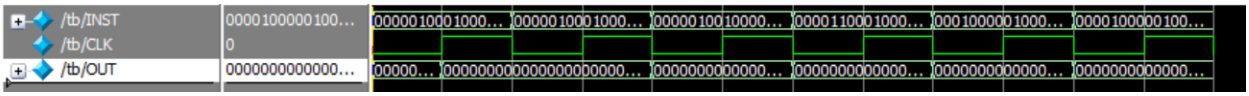

Figure 7: Waveform

3.2. Transcript

```
# OUTPUT :          3
# OUTPUT :          5
run
# OUTPUT :          7
run
# OUTPUT :          4
run
# OUTPUT :         15
run
# OUTPUT :          4
```

Figure 8: Transcript

3.3. Memory List

3.3.1 Register's Memory List

0000001f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a	00000000000000000000000000001010	00000000000000000000000000001001	00000000000000000000000000001000
00000007	00000000000000000000000000000111	00000000000000000000000000000110	00000000000000000000000000000101
00000004	00000000000000000000000000000100	00000000000000000000000000000011	00000000000000000000000000000010
00000001	00000000000000000000000000000001	00000000000000000000000000000001	

Figure 9: Initial Register's Memory List

0000001f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a	00000000000000000000000000001010	00000000000000000000000000001001	00000000000000000000000000001000
00000007	00000000000000000000000000000111	00000000000000000000000000000110	00000000000000000000000000000101
00000004	00000000000000000000000000000100	00000000000000000000000000000011	00000000000000000000000000000010
00000001	00000000000000000000000000000011	00000000000000000000000000000001	

Figure 10: Register's Memory List after add $1,$2,$1

Explanation: **add $1, $2, $1** where $1 stores the value 32'd1 and $2 is 32'd2, so when adding them, the result is 32'd3. Looking at the Register's Memory List, the value of $1 has changed (in the red highlighted area) from 32'd1 to 32'd3. The result is as expected.

Figure 11: Register's Memory List after add $1,$2,$3

Explanation: Similarly to the previous instruction, registers $2 and $3 contain the values 32'd2 and 32'd3 respectively. Therefore, the result of $1 will be updated to 32'd5.

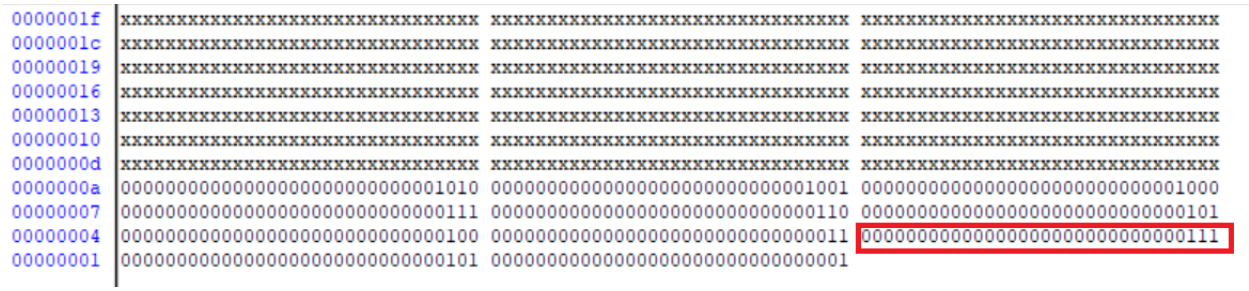

Figure 12: Register's Memory List after add $2,$4,$3

Figure 13: Register's Memory List after sub $1,$2,$3

Explanation: We have registers $2 and $3 containing the values 32'd7 and 32'd3 respectively. The result of subtracting these two registers will be 32'd4, and register $1 has been updated accordingly. The result matches the prediction.

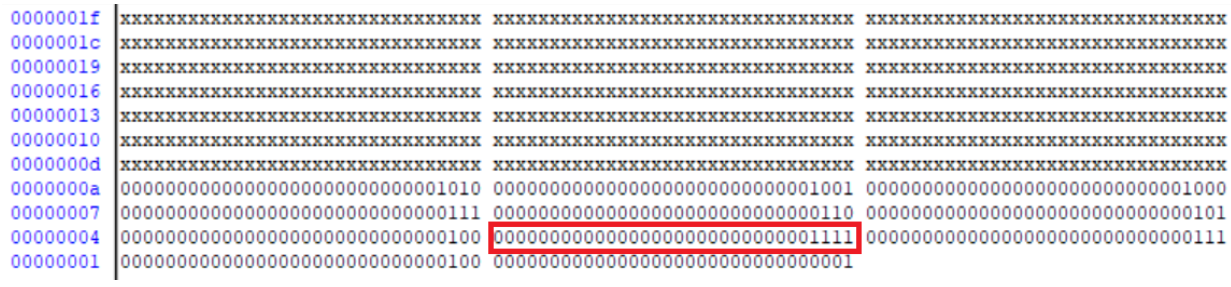

Figure 14: Register's Memory List after lw $3,0($2)

Explanation: We have $2 storing the value 32'd7, and the offset is 0. The sum of the offset and the value in $2 is 32'd7. In this case, we will store the value of register $3 into the address at the 4th location of the Data memory (the value at address 4 in data memory is 32'd15). The result matches the prediction.

3.3.1 Data-Memory's Memory List

0000003a	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000037	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000034	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000031	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000002e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000002b	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000028	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000025	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000022	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a	00000000000000000000000000000001	00000000000000000000000000010100	00000000000000000000000000010010
00000007	00000000000000000000000000001111	00000000000000000000000000001100	00000000000000000000000000000101
00000004	00000000000000000000000000000100	00000000000000000000000000001000	00000000000000000000000000000111
00000001	00000000000000000000000000000110	00000000000000000000000000000011	

Figure 15: Initial Data-memory's Memory List

0000003a	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000037	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000034	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000031	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000002e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000002b	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000028	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000025	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000022	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a	00000000000000000000000000000001	00000000000000000000000000010100	00000000000000000000000000010010
00000007	00000000000000000000000000000111	00000000000000000000000000001100	00000000000000000000000000000101
00000004	00000000000000000000000000000100	00000000000000000000000000001000	00000000000000000000000000000111
00000001	00000000000000000000000000000110	00000000000000000000000000000011	

Figure 10: Data-memory's Memory List after sw $2,0($2)

Explanation: Similarly to the previous **lw** instruction, we have offset + value in $2 = 0 + 7 = 7. Therefore, we will store the value of register $2 (32'd7) into the address at the 7th location of the Data Memory. The result matches the prediction.

4. Schematic

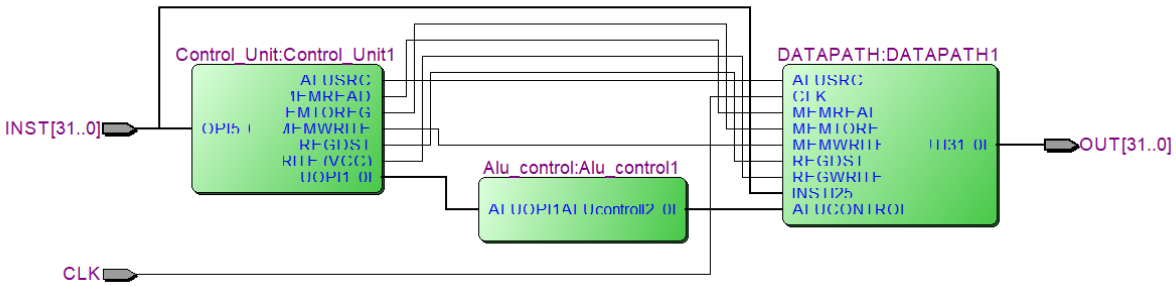

Figure 11: Lab7's Schematic

IV. EVALUATION

No errors were found, and the circuit runs as intended.