ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO THỰC HÀNH LAB05 - THIẾT KẾ DATAPATH ĐƠN GIẢN

HỌ VÀ TÊN: NGUYỄN GIA BẢO NGỌC – 21520366
NGUYỄN QUỐC TRƯỜNG AN – 21521810
NGUYỄN TRƯƠNG TIẾN ĐẠT – 21521946

LỚP: CE213.O11.2

GIẢNG VIÊN HƯỚNG DẪN:

HỒ NGỌC DIỄM

TP. HỒ CHÍ MINH – Tháng 12 năm 2023

MỤC LỤC

I. Mục tiêu.....2

II. Chuẩn bị thực hành. 2

III. Nội dung thực hành. 2

1. Thiết kế mạch: 3

2. Testbench: 6

Danh mục bảng và hình ảnh

Bảng 1 - Code Verilog thiết kế DATAPATH.....5
Bảng 2 - Code Verilog Testbench cho DATAPATH 7

Hình 1 - Datapath theo kiến trúc MIPS 2
Hình 2 - Mạch RTL DATAPATH.....5
Hình 3 - Mô phỏng Testbench DATAPATH.....8
Hình 4 – Kết quả các thanh ghi Register File sau khi chạy testbench.....9
Hình 5 - Kết quả các vùng nhớ Data Memory sau khi chạy testbench.....10

I. Mục tiêu.

Sinh viên sử dụng ngôn ngữ Verilog HDL, thiết kế một DATAPATH đơn giản theo kiến trúc MIPS.

II. Chuẩn bị thực hành.

- Kiểm tra lại các module MUX, Register File, ALU, Data Memory đã thiết kế trong các bài Lab trước.

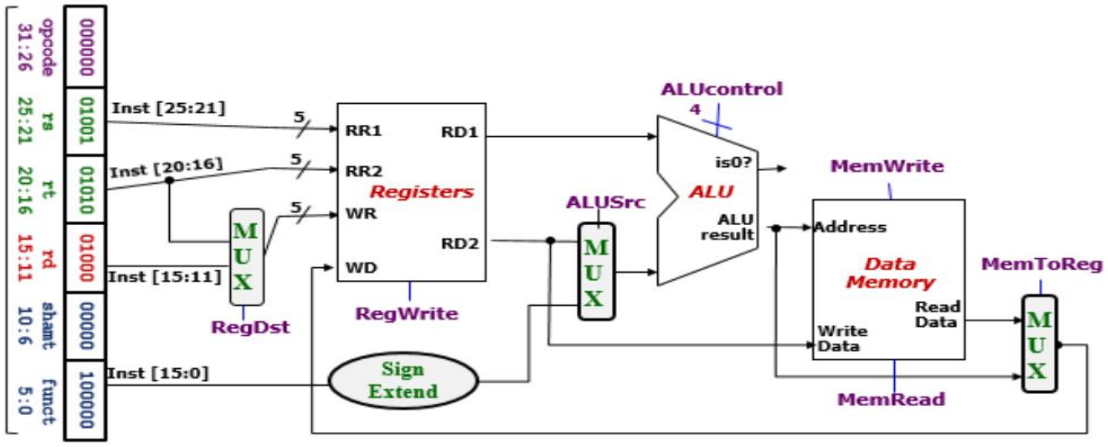
- Tìm hiểu lại kiến trúc và cách thức hoạt động của DATAPATH trong kiến trúc MIPS.

- Phân tích hoạt động DATAPATH trong Hình 1.

III. Nội dung thực hành.

Sinh viên dựa vào lý thuyết về DATA PATH của kiến trúc MIPS đã học trong môn Kiến trúc máy tính như Hình 1 và sử dụng lại các module đã thiết kế trong các Lab trước để thiết kế một DATAPATH thực hiện các lệnh sau dùng ngôn ngữ Verilog HDL:

- add $1, $2, $3
- lw $1, 0($2)
- sw $5, 0($2)



Hình 1 - Datapath theo kiến trúc MIPS

Yêu cầu thực hiện:

- Đưa thiết kế DATAPATH đã chuẩn bị ở nhà vào project.

- Viết testbench kiểm tra thiết kế trên phần mềm mô phỏng ModelSim ứng với lệnh add $1, $2 $3.

- Viết testbench kiểm tra thiết kế trên phần mềm mô phỏng ModelSim ứng với lệnh lw $1, 0($2).

- Viết testbench kiểm tra thiết kế trên phần mềm mô phỏng ModelSim ứng với lệnh sw $5, 0($2).

1. Thiết kế mạch:

- Code verilog:

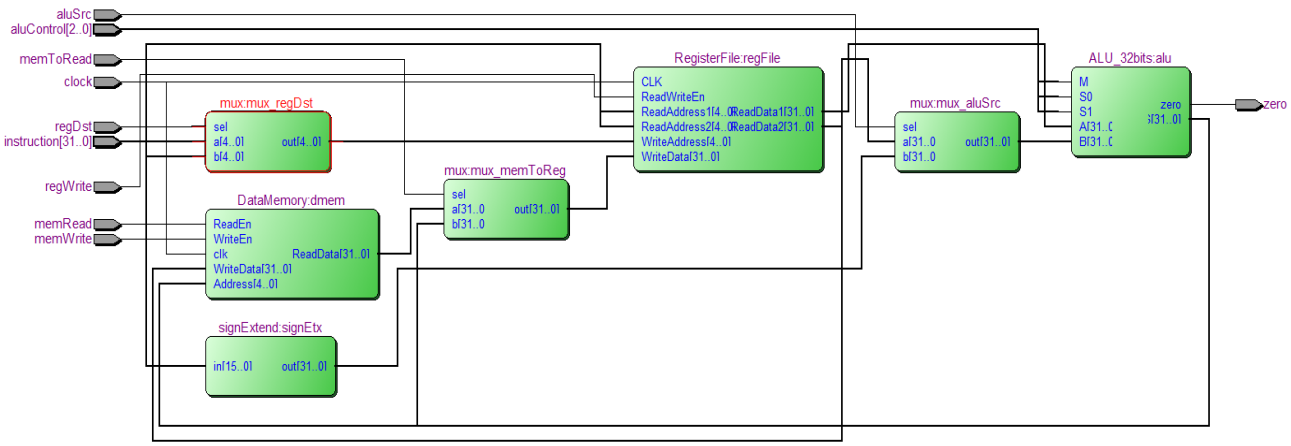
```
module datapath ( zero,
                  clock,
                  instruction,
                  regDst,
                  regWrite,
                  aluSrc,
                  aluControl,
                  memWrite,
                  memRead,
                  memToRead
                  );

    output zero;

    input clock;
    input [31:0] instruction;
    input regDst;
    input regWrite;
    input aluSrc;
    input [2:0]aluControl;
    input memWrite;
    input memRead;
    input memToRead;

    wire [4:0] writeReg;  // out mux_regDst
```

```
wire [31:0] readData1;  // from registerFile
wire [31:0] readData2;  // from registerFile
wire [31:0] writeData;  // out mux memToReg
wire [31:0] imm;                // out signExtend
wire [31:0] aluSrcB;            // out mux aluSrc
wire [31:0] aluOut;             // from ALU
wire [31:0] readData;    // readData from data memory

mux #(
        .DATA_WIDTH(5)
        )
        mux_regDst (.out(writeReg),
                        .sel(regDst),
                        .a(instruction[20:16]),
                        .b(instruction[15:11])
                        );

RegisterFile regFile(
                                .ReadData1(readData1),
                                .ReadData2(readData2),
                                .CLK(clock),
                                .ReadAddress1(instruction[25:21]),
                                .ReadAddress2(instruction[20:16]),
                                .WriteAddress(writeReg),
                                .WriteData(writeData),
                                .ReadWriteEn(regWrite)
                                );

signExtend signEtx ( .out(imm),
                          .in(instruction[15:0])
                        );

mux mux_aluSrc(.out(aluSrcB),
                        .sel(aluSrc),
                        .a(readData2),
                        .b(imm)
                        );
```

```
        ALU_32bits alu ( .zero(zero),
                         .S(aluOut),
                         .A(readData1),
                         .B(aluSrcB),
                         .M(aluControl[2]),
                         .S1(aluControl[1]),
                         .S0(aluControl[0])
                         );

        DataMemory dmem(.ReadData(readData),
                        .clk(clock),
                        .WriteEn(memWrite),
                        .ReadEn(memRead),
                        .WriteData(readData2),
                        .Address(aluOut[9:0])
                        );

        mux mux_memToReg (.out(writeData),
                          .sel(memToRead),
                          .a(readData),
                          .b(aluOut)
                          );

endmodule
```

Bảng 1 - Code Verilog thiết kế DATAPATH

- Mạch RTL:



Hình 2 - Mạch RTL DATAPATH

2. Testbench:

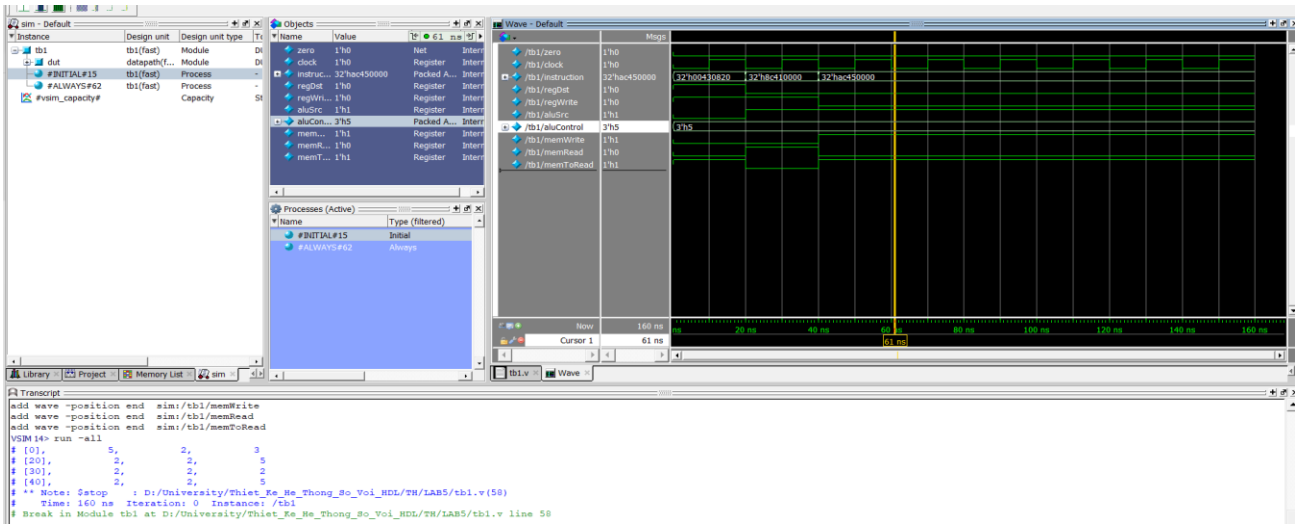
- Code Verilog:

```
`timescale 1ns/1ps
module tb1();

	wire zero;

	reg clock;
	reg [31:0] instruction;
	reg regDst;
	reg regWrite;
	reg aluSrc;
	reg [2:0]aluControl;
	reg memWrite;
	reg memRead;
	reg memToRead;

	initial begin
		// add $1, $2, $3
		clock = 0;
		instruction = 32'b00000000010000110000100000100000;
		regDst = 1;
		regWrite = 1;
		aluSrc = 0;
		aluControl = 3'b101;
		memWrite = 0;
		memRead = 0;
		memToRead = 1;
		#20

		// lw $1, 0($2)
		instruction = 32'b10001100010000010000000000000000;
		regDst = 0;
		regWrite = 1;
		aluSrc = 1;
		aluControl = 3'b101;
```

```
            memWrite = 0;
            memRead = 1;
            memToRead = 0;
            #20

            // sw $5, 0($2)
            instruction = 32'b10101100010001010000000000000000;
            regDst = 0;
            regWrite = 0;
            aluSrc = 1;
            aluControl = 3'b101;
            memWrite = 1;
            memRead = 0;
            memToRead = 1;
            #20

            #100 $stop;

      end

      always #10 clock = ~clock;

      datapath   dut (    .zero(zero),
                          .clock(clock),
                          .instruction(instruction),
                          .regDst(regDst),
                          .regWrite(regWrite),
                          .aluSrc(aluSrc),
                          .aluControl(aluControl),
                          .memWrite(memWrite),
                          .memRead(memRead),
                          .memToRead(memToRead)
                          );

endmodule
```

Bảng 2 - Code Verilog Testbench cho DATAPATH

- Chạy mô phỏng trên ModelSim:



Hình 3 - Mô phỏng Testbench DATAPATH

- Đọc kiểm tra kết quả và giải thích testbench:

Để thuận tiện cho việc kiểm tra kết quả thực thi 3 lệnh add $1, $2 $3, lw $1, 0($2) và sw $5, 0($2), ta khởi tạo giá trị của các thanh ghi trong Register File và các word trong Data Memory có giá trị bằng với địa chỉ của nó. Ví dụ:

+ Thanh ghi $0: value = 0

+ Thanh ghi $1: value = 1

+ Thanh ghi $2: value = 2

…

+ DMEM[0]: value = 0

+ DMEM[1]: value = 1

+ DMEM[2]: value = 2

…

Như vậy, ta lần được thực hiện các lệnh sau:

+ add $1, $2 $3 => $1 = $2 + $3 = 2 + 3 = 5

+ lw $1, 0($2) => $1 = DMEM[0 + $2] = DMEM[2] = 2

+ sw $5, 0($2) => DMEM[0 + $2] = $5 => DMEM[2] = 5

Vậy sau khi thực hiện 3 lệnh trên, ta có:

+ Thanh ghi $1: value = 2

+ Thanh ghi $2: value = 2

+ Thanh ghi $3: value = 3

và

+ DMEM[2]: value = 5

- Sau khi biết được kết quả mong muốn có được, ta thực hiện chuyển đổi 3 lệnh trên sang mã máy dựa trên kiến trúc tập lệnh MIPS:

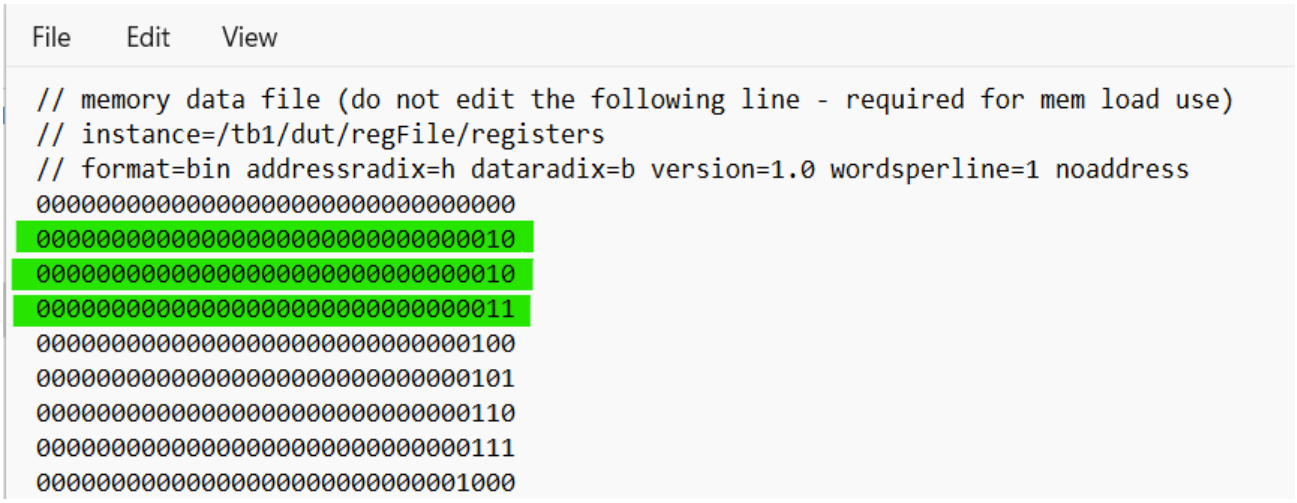
+ add $1, $2 $3 : 00000000010000110000100000100000

+ lw $1, 0($2) : 10001100010000010000000000000000

+ sw $5, 0($2) : 10101100010001010000000000000000

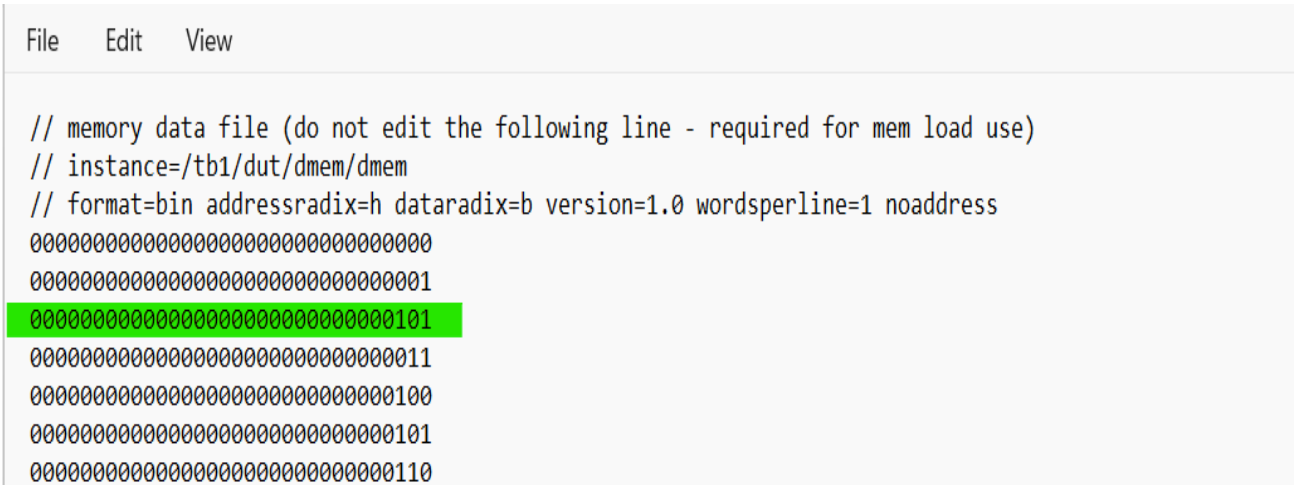
- Testbench được thực hiện chạy trên ModelSim và kết quả các thanh ghi và word của Register File và Data Memory được lưu trong file text định dạng .bin:

+ Kết quả $1, $2, $3:



Hình 4 – Kết quả các thanh ghi Register File sau khi chạy testbench

+ Kết quả DMEM[2]:



Hình 5 - Kết quả các vùng nhớ Data Memory sau khi chạy testbench