|  |
| --- |
| **DIGITAL SYSTEM DESIGN LABORATORY** |
| **LAB 4** |

**BASIC BUILDING BLOCKS OF SINGLE CYCLE MICROPROCESSOR**

### I. LAB OBJECTIVES

### This Lab experiments are intended to implement basic building blocks of Single Cycle Microprocessor

### II. DESCRIPTION

### Single Cycle Microprocessor datapath to be implemented is in figure 2.1.

### 

### Figure 2.1: Single Cycle Microprocessor DataPath

### III. LAB PROCEDURE

### III.1 EXPERIMENT NO. 1

##### III.1.1 AIM: To implement Program Counter

##### 

**III.1.2 CODE**

module Program\_Counter (clk, reset, PC\_in, PC\_out);

input clk, reset;

input [7:0] PC\_in;

output [7:0] PC\_out;

reg [7:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=8’b0;

else

PC\_out<=PC\_in;

end

endmodule

**III.1.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

module test\_Program\_Counter;

reg t\_clk, t\_reset;

reg [7:0] t\_PC\_in;

wire [7:0] t\_PC\_out;

Program\_Counter DUT(.clk(t\_clk), .reset(t\_reset), .PC\_in(t\_PC\_in), .PC\_out(t\_PC\_out));

initial begin

$dumpfile("test\_Program\_Counter.vcd");

$dumpvars(1);

$monitor($time,,"reset=%b |clk=%b | PC\_in=%b | PC\_out=%b |", t\_reset,t\_clk,, t\_PC\_in, t\_PC\_out);

t\_reset=1; t\_clk=0; t\_PC\_in=8'b00000000;

repeat(100) begin

#5 t\_clk=0;

#5 t\_PC\_in = $random;

#5 t\_clk=1;

end

#5 $finish;

end

endmodule

A screenshot of a computer

Description automatically generated

2. Write the Top level module to implement this module in FPGA KIT

module lab4\_task1(SW,LEDR,LEDG,KEY);

input [17:0] SW;

input [3:0] KEY;

output [17:0] LEDR;

output [7:0] LEDG;

assign LEDR=SW;

Program\_Counter (.clk(KEY[0]), .reset(SW[17]), .PC\_in(SW[7:0]), .PC\_out(LEDG[7:0]));

endmodule

### III.2 EXPERIMENT NO. 2

##### III.2.1 AIM: To implement 32 bit Adder

##### Diagram Description automatically generated

**III.2.2 CODE**

module Adder32Bit(input1, input2, out);

input [7:0] input1, input2;

output [7:0] out;

reg [7:0]out;

always@( input1 or input2)

begin

out <= input1 + input2;

end

endmodule

**III.2.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

module test\_Adder32Bit;

reg [7:0] t\_input1, t\_input2;

wire [7:0] t\_out;

Adder32Bit DUT(.input1(t\_input1), .input2(t\_input2), .out(t\_out));

initial begin

$dumpfile("test\_Adder32Bit.vcd");

$dumpvars(1);

$monitor($time,,"input1=%b | input2=%b ||| out=%b", t\_input1,t\_input2,t\_out);

t\_input1=8'b00011110; t\_input2=8'b11011010;

repeat(10) begin

#5 t\_input1=$random; t\_input2=$random;

end

#5 $finish;

end

endmodule

A screenshot of a computer

Description automatically generated

2. Write the Top level module to implement this module in FPGA KIT

module lab4\_task2(SW,LEDR,LEDG,KEY);

input [17:0] SW;

input [3:0] KEY;

output [17:0] LEDR;

output [7:0] LEDG;

assign LEDR=SW;

Adder32Bit (.input1(SW[15:8]), .input2(SW[7:0]), .out(LEDG[7:0]));

endmodule

### III.3 EXPERIMENT NO. 3

##### III.3.1 AIM: To implement the datapath for PC = PC + 1

##### Diagram Description automatically generated

**III.3.2 CODE**

module PC\_ADD\_1(clk, reset, instruction\_address);

input clk, reset;

output [7:0] instruction\_address;

wire [7:0] W\_PC\_add\_1;

Program\_Counter C1(.clk(clk), .reset(reset), .PC\_in(W\_PC\_add\_1), .PC\_out(instruction\_address));

Adder32Bit C2(.input1(8'b1), .input2(instruction\_address), .out(W\_PC\_add\_1));

endmodule

**III.3.3 LAB ASSIGNMENT**

1. Write testbenches to verify above module and attach waveforms.

module testbench\_PC\_add\_1;

reg clk, reset;

wire [7:0] instruction\_address;

PC\_ADD\_1 DUT(.clk(clk), .reset(reset), .instruction\_address(instruction\_address));

always #5 clk = ~clk;

initial begin

$dumpfile("testbench\_PC\_add\_1.vcd");

$dumpvars(0,testbench\_PC\_add\_1);

$monitor($time,,"reset=%b,clk=%b,instruction\_address=%b", reset,clk,instruction\_address);

reset=1; clk=0;

#5 reset=0;

#100 $finish;

end

endmodule

A screenshot of a video game

Description automatically generated

2. Write the Top level module to implement this module in FPGA KIT

module lab4\_task3(SW,LEDR,LEDG,KEY);

input [17:0] SW;

input [3:0] KEY;

output [17:0] LEDR;

output [7:0] LEDG;

assign LEDR=SW;

PC\_ADD\_1 (.clk(KEY[0]), .reset(SW[17]), .instruction\_address(SW[7:0]));

endmodule

### III.4 EXPERIMENT NO. 4

##### III.4.1 AIM: To implement the Instruction memory

##### Diagram Description automatically generated

**III.4.2 CODE**

module Instruction\_Memory (read\_address, instruction, reset);

input reset;

input [7:0] read\_address;

output [15:0] instruction;

reg [15:0] Imemory [7:0];

integer k;

// I-MEM in this case is addressed by word, not by byte

assign instruction = Imemory[read\_address];

always @(posedge reset)

begin

for (k=0; k<16; k=k+1)

begin

// here Out changes k=0 to k=16

Imemory[k] = 16'b0;

end

Imemory[0] = 32'b00100000000010000000000000100000;

//addi $t0, $zero, 32

Imemory[1] = 32'b00100000000010010000000000110111;

//addi $t1, $zero, 55

Imemory[2] = 32'b00000001000010011000000000100100;

//and $s0, $t0, $t1

Imemory[3] = 32'b00000001000010011000000000100101;

//or $s0, $t0, $t1

Imemory[4] = 32'b10101100000100000000000000000100;

//sw $s0, 4($zero)

Imemory[5] = 32'b10101100000010000000000000001000;

//sw $t0, 8($zero)

Imemory[6] = 32'b00000001000010011000100000100000;

//add $s1, $t0, $t1

Imemory[7] = 32'b00000001000010011001000000100010;

//sub $s2, $t0, $t1

Imemory[8] = 32'b00010010001100100000000000001001;

//beq $s1, $s2, error0

Imemory[9] = 32'b10001100000100010000000000000100;

//lw $s1, 4($zero)

Imemory[10]= 32'b00110010001100100000000001001000;

//andi $s2, $s1, 48

Imemory[11] =32'b00010010001100100000000000001001;

//beq $s1, $s2, error1

Imemory[12] =32'b10001100000100110000000000001000;

//lw $s3, 8($zero)

Imemory[13] =32'b00010010000100110000000000001010;

//beq $s0, $s3, error2

Imemory[14] =32'b00000010010100011010000000101010;

//slt $s4, $s2, $s1 (Last)

Imemory[15] =32'b00010010100000000000000000001111;

//beq $s4, $0, EXIT

Imemory[16] =32'b00000010001000001001000000100000;

//add $s2, $s1, $0

Imemory[17] =32'b00001000000000000000000000001110;

//j Last

Imemory[18] =32'b00100000000010000000000000000000;

//addi $t0, $0, 0(error0)

Imemory[19] =32'b00100000000010010000000000000000;

//addi $t1, $0, 0

Imemory[20] =32'b00001000000000000000000000011111;

//j EXIT

Imemory[21] =32'b00100000000010000000000000000001;

//addi $t0, $0, 1(error1)

Imemory[22] =32'b00100000000010010000000000000001;

//addi $t1, $0, 1

Imemory[23] =32'b00001000000000000000000000011111;

//j EXIT

Imemory[24] =32'b00100000000010000000000000000010;

//addi $t0, $0, 2(error2)

Imemory[25] =32'b00100000000010010000000000000010;

//addi $t1, $0, 2

Imemory[26] =32'b00001000000000000000000000011111;

//j EXIT

Imemory[27] =32'b00100000000010000000000000000011;

//addi $t0, $0, 3(error3)

Imemory[28] =32'b00100000000010010000000000000011;

//addi $t1, $0, 3

Imemory[29] =32'b00001000000000000000000000011111;

//j EXIT

end

endmodule

|  |  |  |
| --- | --- | --- |
| reset | Read\_address | PC\_out |
| 1 | 32’b0 | 32'b00100000000010000000000000100000 |
| 0 | 32’b0 | 32'b00100000000010000000000000100000 |
| 0 | 32’b0000….. 0011 | 32'b00000001000010011000000000100101 |
| 0 | 32’b0000 …….0111 | 32'b00000001000010011001000000100010 |

**III.4.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

module test\_Instruction\_Memory;

reg t\_reset;

reg [7:0] t\_read\_address;

wire [15:0] t\_instruction;

Instruction\_Memory DUT(.read\_address(t\_read\_address), .reset(t\_reset),.instruction(t\_instruction));

initial begin

$dumpfile("test\_Instruction\_Memory.vcd");

$dumpvars(0,test\_Instruction\_Memory);

$monitor($time,,"reset=%b | read address=%b ||| instruction=%b", t\_reset,t\_read\_address,t\_instruction);

t\_reset=1;

#5 t\_reset=0;t\_read\_address=0;

repeat(7) begin

#5 t\_read\_address+=1;

end

#200 $finish;

end

endmoduleA screenshot of a computer

Description automatically generated

2. Write the Top level module to implement this module in FPGA KIT

module lab4\_task4(SW,LEDR,LEDG,KEY);

input [17:0] SW;

input [3:0] KEY;

output [17:0] LEDR;

output [7:0] LEDG;

assign LEDR=SW;

Instruction\_Memory (.read\_address(SW[16:9]), .reset(SW[17]),.instruction(LEDG[7:0]));

endmodule

### III.5 EXPERIMENT NO. 5

##### III.5.1 AIM: To implement the Instruction memory datapath

##### Diagram Description automatically generated

**III.5.2 CODE**

module Instruction\_Datapath (clk, reset, Instruction);

input clk, reset;

output [15:0] Instruction;

endmodule

**III.5.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

2. Write the Top level module to implement this module in FPGA KIT

### III.6 EXPERIMENT NO. 6

##### III.6.1 AIM: To implement the Register File

##### Diagram, schematic Description automatically generated

**III.6.2 CODE**

module Register\_File (read\_addr\_1, read\_addr\_2, write\_addr, read\_data\_1, read\_data\_2, write\_data, RegWrite, clk, reset);

input [2:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [7:0] write\_data;

input clk, reset, RegWrite;

output [7:0] read\_data\_1, read\_data\_2;

reg [7:0] Regfile [7:0];

integer k;

//assign read\_data\_1 = Regfile[read\_addr\_1];

always @(read\_data\_1 or Regfile[read\_data\_1])

begin

if (read\_data\_1 == 0) read\_data\_1 = 0;

else read\_data\_1 = Regfile[read\_addr\_1];

end

//assign read\_data\_2 = Regfile[read\_addr\_2];

always @(read\_data\_2 or Regfile[read\_data\_2])

begin

if (read\_data\_2 == 0) read\_data\_2 = 0;

else read\_data\_2 = Regfile[read\_addr\_2];

end

always @(posedge clk or posedge reset) // Ou combines the block of reset into the block of posedge clk

begin

if (reset==1'b1)

begin

for (k=0; k<8; k=k+1)

begin

Regfile[k] = 8'b0;

end

end

else if (RegWrite == 1'b1) Regfile[write\_addr] = write\_data;

end

endmodule

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| reset | clk | read\_addr\_1 | read\_data\_1 | read\_addr\_2 | read\_data\_2 | RegWrite | write\_addr | write\_data |
| 1 | x | x | 32’b0 | x | 32’b0 | x | x | x |
| 0 | ↑ | 32’d3 | 32’b0 | 32’d4 | 32’b0 | 1 | 32’b001 | 32’b0111 |
| 0 | ↑ | 32’d3 | 32’b0 | 32’d4 | 32’b0 | 1 | 32’b010 | 32’b1000 |
| 0 | ↑ | 32’d3 | 32’b0 | 32’d4 | 32’b0 | 1 | 32’b011 | 32’b1001 |
| 0 | ↑ | 32’d3 | 32’b0 | 32’d4 | 32’b0 | 1 | 32’b100 | 32’b1010 |
| 0 | ↑ | 32’d3 | 32’b0 | 32’b00 | 32’b0 | 1 | 32’b101 | 32’b1011 |
| 0 | ↑ | 32’b001 | 32’b0111 | 32’b101 | 32’b1011 | 0 | x | x |
| 0 | ↑ | 32’b011 | 32’b1001 | 32’b100 | 32’b1010 | 0 | x | x |
| 0 | ↑ | 32’b101 | 32’b1011 | 32’b101 | 32’b1011 | 0 | x | x |
| 0 | ↑ | 32’b100 | 32’b1010 | 32’b100 | 32’b1010 | 0 | x | x |

**III.6.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

2. Write the Top level module to implement this module in FPGA KIT

### III.7 EXPERIMENT NO. 7

##### III.7.1 AIM: To implement the ALU

##### Diagram, schematic Description automatically generated

**III.7.2 CODE**

module alu(

input [2:0] alufn,

input [7:0] ra,

input [7:0] rb\_or\_imm,

output reg [7:0] aluout,

output reg zero);

parameter ALU\_OP\_ADD = 3'b000,

ALU\_OP\_SUB = 3'b001,

ALU\_OP\_AND = 3'b010,

ALU\_OP\_OR = 3'b011,

ALU\_OP\_NOT\_A = 3'b100,

ALU\_OP\_LW = 3'b101,

ALU\_OP\_SW = 3'b110,

ALU\_OP\_BEQ = 3'b111;

always @(\*)

begin

case(alufn)

ALU\_OP\_ADD : aluout = ra + rb\_or\_imm;

ALU\_OP\_SUB : aluout = ra - rb\_or\_imm;

ALU\_OP\_AND : aluout = ra & rb\_or\_imm;

ALU\_OP\_OR : aluout = ra | rb\_or\_imm;

ALU\_OP\_NOT\_A : aluout = ~ ra;

ALU\_OP\_LW : aluout = ra + rb\_or\_imm;

ALU\_OP\_SW : aluout = ra + rb\_or\_imm;

ALU\_OP\_BEQ : begin

zero = (ra==rb\_or\_imm)?1'b1:1'b0;

aluout = ra - rb\_or\_imm;

end

endcase

end

endmodule

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| alufn | ra | rb\_or\_imm | aluout | zero |
| 3'b000 | 32’d8 | 32’d2 | 32’d10 | 0 |
| 3'b001 | 32’d8 | 32’d2 | 32’d6 | 0 |
| 3'b010 | 32’b1000 | 32’b0010 | 32’b0000 | 0 |
| 3'b011 | 32’b1000 | 32’b0010 | 32’b1010 | 0 |
| 3'b100 | 32’b1000 | 32’d2 | 32’b0111 | 0 |
| 3'b101 | 32’d8 | 32’d2 | 32’d10 | 0 |
| 3'b110 | 32’d8 | 32’d2 | 32’d10 | 0 |
| 3'b111 | 32’d8 | 32’d2 | 32’d6 | 0 |
| 3'b111 | 32’d8 | 32’d8 | 32’d0 | 1 |

**III.7.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

2. Write the Top level module to implement this module in FPGA KIT

### III.8 EXPERIMENT NO. 8

##### III.8.1 AIM: To implement the 32 bit RAM

##### Diagram Description automatically generated

**III.8.2 CODE**

module Data\_Memory (addr, write\_data, read\_data, clk, reset, MemRead, MemWrite);

input [7:0] addr;

input [7:0] write\_data;

output [7:0] read\_data;

input clk, reset, MemRead, MemWrite;

reg [7:0] DMemory [7:0];

integer k;

assign read\_data = (MemRead) ? DMemory[addr] : 8'bx;

always @(posedge clk or posedge reset)

begin

if (reset == 1'b1)

begin

for (k=0; k<8; k=k+1)

begin

DMemory[k] = 8'b0;

end

end

else

if (MemWrite) DMemory[addr] = write\_data;

end

endmodule

**III.8.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

2. Write the Top level module to implement this module in FPGA KIT

### III.9 EXPERIMENT NO. 9

##### III.9.1 AIM: To implement Multiplexer

##### Chart, diagram Description automatically generated

**III.9.2 CODE**

module Mux\_N\_bit (in0, in1, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1;

output [N-1:0] mux\_out;

input control;

assign mux\_out = select? in1: in0 ;

endmodule

**III.9.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

2. Write the Top level module to implement this module in FPGA KIT

### III.10 EXPERIMENT NO. 10

##### III.10.1 AIM: To implement Sign\_Extension

##### Diagram Description automatically generated

**III.10.2 CODE**

module Sign\_Extension (sign\_in, sign\_out);

input [5:0] sign\_in;

output [7:0] sign\_out;

assign sign\_out[5:0]=sign\_in[5:0];

assign sign\_out[7:6]=sign\_in[5]?2'b11:2'b0;

endmodule

**III.10.3 LAB ASSIGNMENT**1. Write testbenches to verify above module and attach waveforms.

2. Write the Top level module to implement this module in FPGA KIT

**III. LAB REPORT GUIDELINES**

Students write up a report on the Verilog HDL implementation experiment projects created in this lab. The lab report should include Verilog code for the module under test, Verilog test bench code and a truth table results, and example data input and output to validate the experiment. Simulation Result in form of Simulation Capture Screen.