|  |
| --- |
| **DIGITAL SYSTEM DESIGN LABORATORY** |
| **LAB 6** |

**MULTI-CYCLE MICROPROCESSOR DESIGN**

SUBMITED BY

NGUYEN LE QUOC ANH

Date Submitted: January 3, 2022

Date Performed: January 7, 2022

Lab Section: 6

Course Instructor: Dr. Vo Minh Thanh

### I. LAB OBJECTIVES

### This Lab experiments are intended to design and test a Multi-Cycle Microprocessor

### II. DESCRIPTION

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

### 

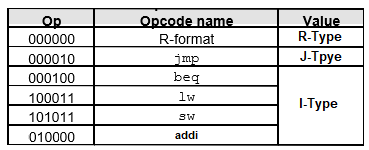
### Figure 2.1: Multi-cyclye Cycle Microprocessor DataPath

### III. LAB PROCEDURE

### III.1 EXPERIMENT NO. 1

##### III.1.1 AIM: To understand and write the assembly codes using MIPS Instruction

##### Instruction Operation codes:

****

Instruction Formats:

**Timeline

Description automatically generated**

**Table

Description automatically generated**

Register names and orders:



Assume the Assembly code code start from address PC=0x00000000, one instruction is store in one memory location.

**Testing Assembly Program 1:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22   
sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8  
beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

**Testing Assembly Program 2:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22   
sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

**III.1.2 LAB ASSIGNMENT**1) Compile the Assembly **Testing Assembly Program 1** into machine code (decimal code and binary code)

2) What is the value of Register $s8 after running **Testing Assembly Program 1**  program

3) Compile the Assembly **Testing Assembly Program 2** into machine code (decimal code and binary code)

4) What is the value of Register $s8 after running **Testing Assembly Program 2**  program

### III.2 EXPERIMENT NO. 2

##### III.2.1 AIM: To implement Verilog code to test ALL the Components of Multi-Cycle processor

##### 

**III.2.2 LAB ASSIGNMENT  
Write the Verilog modules for all required components in Datapath of the Multi-Cycle Processor Design**

* 1. Program Counter

• Code

Text

Description automatically generated

• Testbench

Text

Description automatically generated

* Simulation

Table

Description automatically generated

* 1. Data memory

• Testbench

Text

Description automatically generated

• Code

Text

Description automatically generated

Text

Description automatically generated

* Simulation

Graphical user interface, text, application

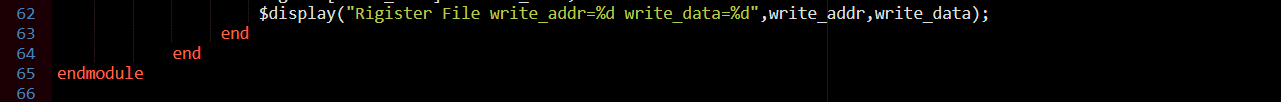
Description automatically generated

* 1. Register file

• Code

Text

Description automatically generated



• Testbench

Text

Description automatically generated

* Simulation

Table

Description automatically generated

* 1. ALU

• Code

A picture containing text

Description automatically generated

• Testbench

Text

Description automatically generated

* Simulation

Table

Description automatically generated

* 1. Holding register

• Testbench

Text

Description automatically generated

• Code

Text

Description automatically generated

* Simulation

Table

Description automatically generated

* 1. Sign extends

• Code

Text

Description automatically generated

• Testbench

A screenshot of a computer

Description automatically generated with medium confidence

* Simulation

Graphical user interface

Description automatically generated with medium confidence

* 1. Shift left by 2

• Code

Text

Description automatically generated

• Testbench

Text

Description automatically generated

* Simulation

Graphical user interface, application

Description automatically generated

* 1. 32-bit MUX 2

• Code

Text

Description automatically generated

• Testbench

A screenshot of a computer

Description automatically generated with medium confidence

* Simulation

Graphical user interface

Description automatically generated with medium confidence

* 1. MUX 4

• Code

Text

Description automatically generated

• Testbench

A screenshot of a computer

Description automatically generated with medium confidence

* Simulation

Table

Description automatically generated with medium confidence

* 1. CONCAT

• Code

Text

Description automatically generated

• Testbench

Text

Description automatically generated

* Simulation

Graphical user interface, application

Description automatically generated with medium confidence

### III.3 EXPERIMENT NO. 3

##### III.3.1 AIM: To Write Verilog code to implement the complete Datapath of Multi-Cycle processor

##### 

**III.3.2 LAB ASSIGNMENT**1) Write Verilog code to implement complete Datapath Multi-Cycle Processor module

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

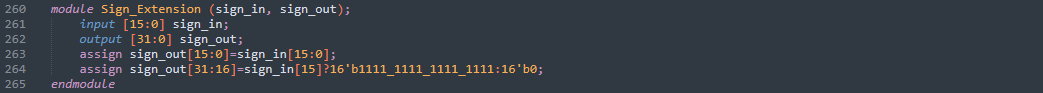
Text

Description automatically generated

Text

Description automatically generated





Text

Description automatically generated



Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

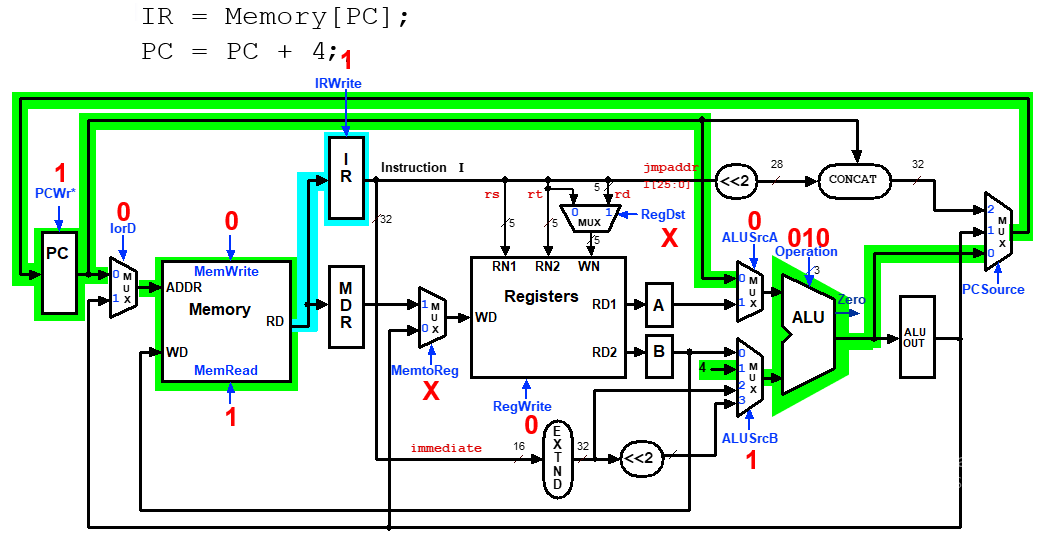
Text

Description automatically generated

Text

Description automatically generated

2) Write testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step 1 (Instruction Fetch : **IF**)



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|PC\_in=%h|PC\_out=%h|mux\_1\_out=%h|Mem\_Read\_data=%b|instruction=%b|ALU\_in\_A=%h|ALU\_in=%h|ALU\_out=%h",clk,reset,PC\_in[3:0],PC\_out[3:0],mux\_1\_out[3:0],Mem\_Read\_data[31:26],instruction[31:26],ALU\_in\_A[3:0],ALU\_in\_B[3:0],ALU\_out[3:0]);

Iord=0;

MemRead=1;

MemWrite=0;

IRwrite=1;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

Operation\_ALU=3'b000;

PCWr=1;

reset=1;clk=0;

#15 reset=0;

#15 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold;

output [27:0] jump\_28\_bit;

wire [31:0] W\_RD1, W\_RD2,Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, A\_data, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=0;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

content <= 0;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

* **Simulation**

Table

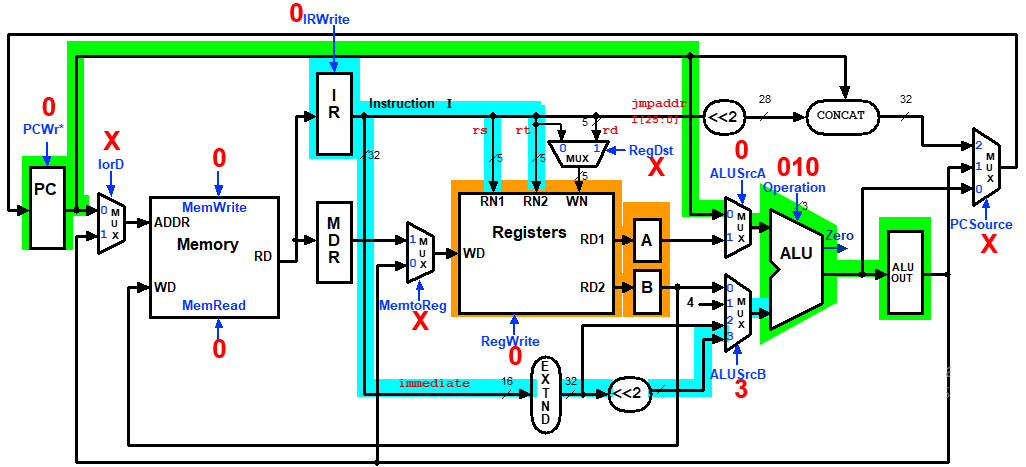
Description automatically generated

3) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step-2 (Instruction Decode & Register Fetch : **ID**)

A = Reg[IR[25-21]]; (A = Reg[rs])

B = Reg[IR[20-15]]; (B = Reg[rt])

ALUOut = (PC + sign-extend(IR[15-0]) << 2);



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|IMM=%h|PC\_out=%d|ALU\_in\_A=%h|ALU\_in\_B=%h|ALU\_out=%h|Rs=%d|Rt=%d|RD1=%d|RD2=%d",clk,reset,instruction,instruction[15:0],PC\_out,ALU\_in\_A[31:0],ALU\_in\_B[31:0],ALU\_out[31:0],instruction[25:21], instruction[20:16], W\_RD1, W\_RD2,);

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b11;

PCSource=2'b00;

Operation\_ALU=3'b101; // ID

PCWr=0;

reset=1;clk=0;

#15 reset=0;

#15 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, A\_data, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'h8;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

content <= 32'b00000010010100111010000000100000; // R-type

#20 content <= 32'b10001110010100010000000000000010; // I-type

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

* **Simulation**

In line 89, the executed address is loaded in PC\_out.

**Text

Description automatically generated**

In line 112, the instructions of R-type and I-type is loaded into the holding register module.

Text

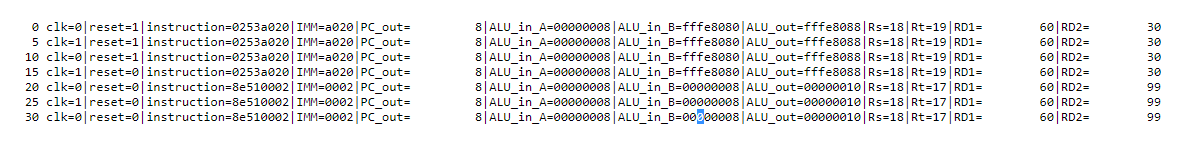
Description automatically generated

The following values are predefined in register file module

Text

Description automatically generated

The result shown in time unit 15 corresponding to the R-type instruction, while the one in time unit 25 corresponding to the I-type instruction



4) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step-3 (Memory Reference Instructions : **MEM**)



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|Rs=%d|RD1=%d|IMM=%h|ALU\_in\_A=%d|ALU\_in\_B=%h|ALU\_out=%d",clk,reset,instruction,instruction[20:16],W\_RD1,instruction[15:0],ALU\_in\_A[31:0],ALU\_in\_B[31:0],ALU\_out[31:0]);

Iord=1;

MemRead=1;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

Operation\_ALU=3'b101; // ID

PCWr=0;

reset=1;clk=0;

#15 reset=0;

#15 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, W\_RD1, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'h8;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

content = 32'b10001110010100010000000000000010;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

* **Simulation**

Load the following instruction into the holding register module.

Text

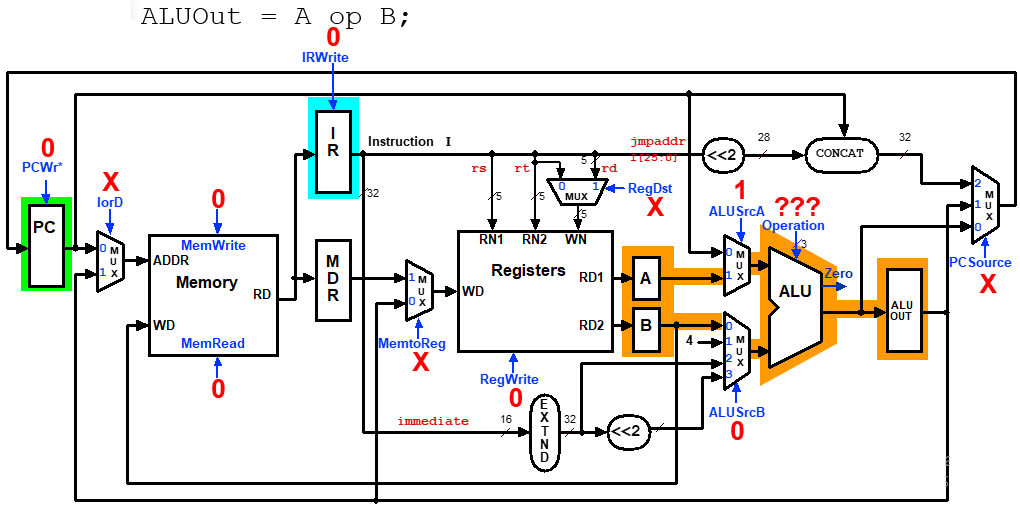
Description automatically generated

The memory address shown below consistent with the input instruction.

A screenshot of a computer

Description automatically generated with medium confidence

5) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step-6 (ALU Instruction (R-Type))



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|ALU\_in\_A=%d|ALU\_in\_B=%d|ALU\_out=%d|Rs=%d|Rt=%d|RD1=%d|RD2=%d",clk,reset,instruction,ALU\_in\_A[31:0],ALU\_in\_B[31:0],ALU\_out[31:0],instruction[25:21], instruction[20:16], W\_RD1, W\_RD2,);

// state 6 R-type

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

Operation\_ALU=3'b000; // ID

PCWr=0;

reset=1;clk=0;

#15 reset=0;

#15 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, A\_data, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'h8;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

content = 32'b00000010010100111010000000100000;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

* **Simulation**

Load the following instruction into holding register module.

Graphical user interface, text, website

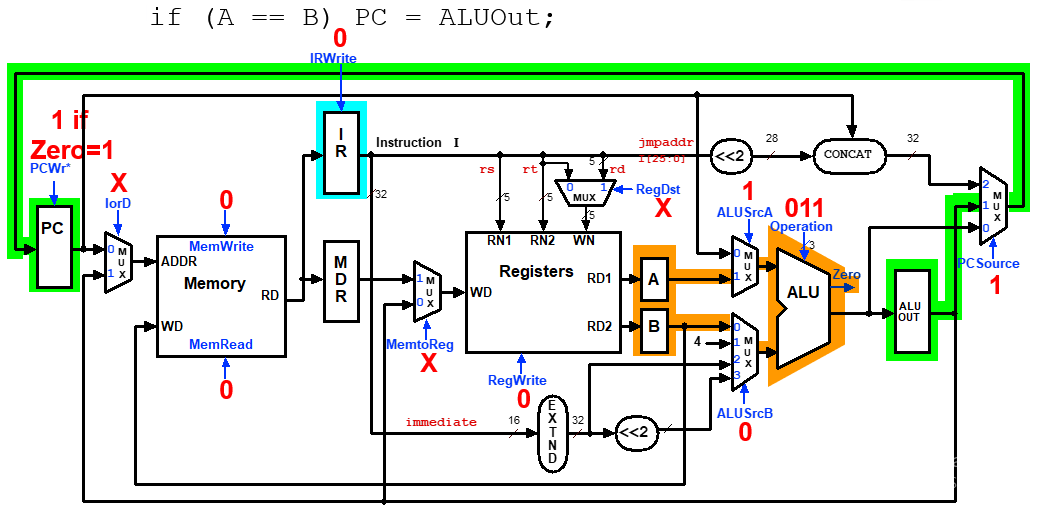
Description automatically generated

The result is shown below.

Diagram

Description automatically generated with low confidence

6) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step 8- Branch Instructions



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

reg [31:0] instruction,ALU\_out\_hold;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,MDR\_out,W\_RD1, W\_RD2;

wire zero;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2,zero);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|ALU\_in\_A=%d|ALU\_in\_B=%d|ALU\_out=%d|ZERO=%b|ALUout=%h|PC\_in=%h",clk,reset,instruction,ALU\_in\_A[31:0],ALU\_in\_B[31:0],ALU\_out[31:0],zero,ALU\_out\_hold,PC\_in);

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b1;

Operation\_ALU=3'b111; // ID

PCWr=0;

ALU\_out\_hold=32'h8;

reset=1;clk=0;

#15 reset=0;

// beq $t2, $t2, 0x02

#1 instruction=32'b00010001010010100000000000000010;

// state 8 I-type | Branch completion

#1 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2,zero);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

input [31:0] instruction,ALU\_out\_hold;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,MDR\_out,W\_RD1, W\_RD2;

output zero;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data,wire\_blank;

wire [4:0] mux\_3\_out;

wire PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(wire\_blank, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, W\_RD1, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(W\_RD2, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(wire\_blank, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'b0;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//load branching address

content = 32'h8;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

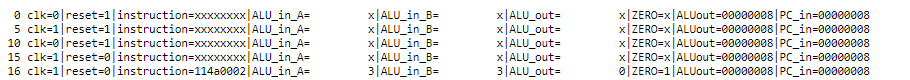
* **Simulation**

Initialized the next instruction address and the instruction for the branch equal in the testbench

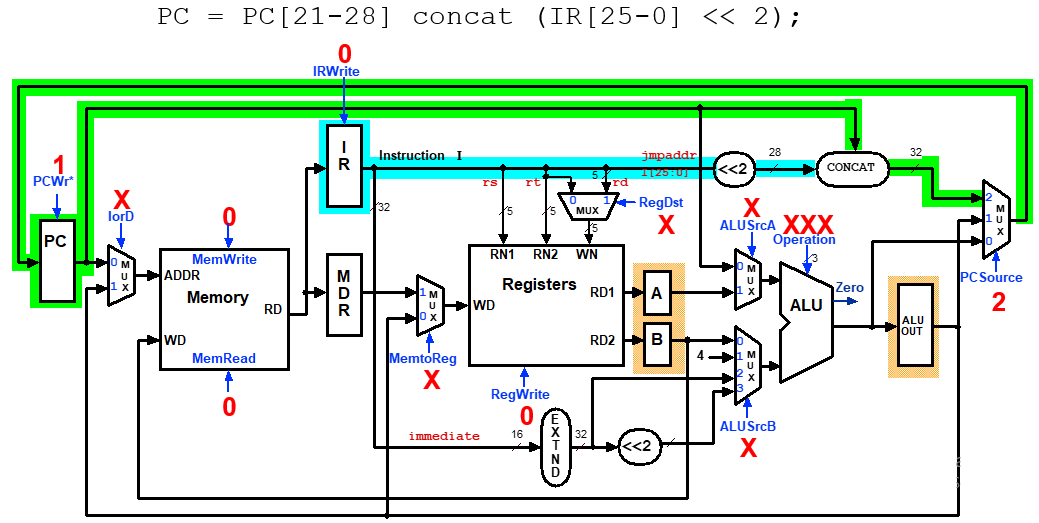
Text

Description automatically generated

Obtain the result



7) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Execution Step 9 - Jump Instruction



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

reg [31:0] instruction,ALU\_out\_hold;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,MDR\_out,W\_RD1, W\_RD2;

wire zero;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2,zero);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|Jump\_addr=%h|PC\_in=%h",clk,reset,instruction,Jump\_addr,PC\_in);

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b10;

Operation\_ALU=3'b010; // ID

PCWr=1;

ALU\_out\_hold=32'h8;

reset=1;clk=0;

#15 reset=0;

// J 0x03

#1 instruction=32'b00001000000000000000000000000011;

// J 0x7

#2 instruction=32'b00001000000000000000000000000111;

// state 9 J-type | Jump completion

#20 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2,zero);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

input [31:0] instruction,ALU\_out\_hold;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,MDR\_out,W\_RD1, W\_RD2;

output zero;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data,wire\_blank;

wire [4:0] mux\_3\_out;

wire PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(wire\_blank, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, W\_RD1, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(W\_RD2, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(wire\_blank, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'b0;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//load branching address

content = 32'h8;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

* **Simulation**

Initialize the following Jump instruction in the testbench.

Text

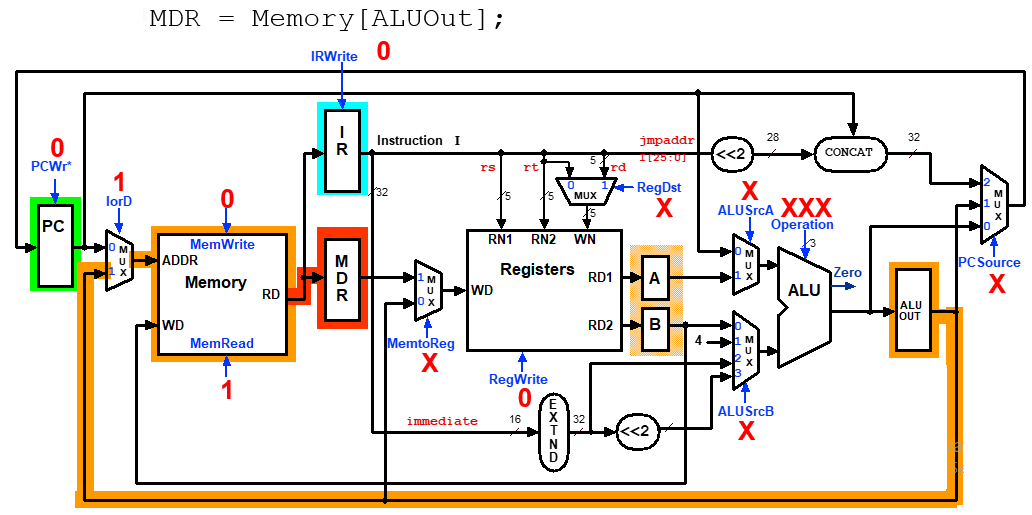
Description automatically generated with medium confidence

Obtain result.

Text

Description automatically generated

8) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step 3’- Memory Access - Read (lw instruction)



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|ALU\_out=%d|mux\_1\_out=%d|Mem\_Read\_data=%d|",clk,reset,instruction,ALU\_out[31:0],mux\_1\_out,Mem\_Read\_data);

Iord=1;

MemRead=1;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

Operation\_ALU=3'b101; // ID

PCWr=0;

reset=1;clk=0;

#15 reset=0;

#15 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, W\_RD1, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'h8;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

content = 32'b10001110010100010000000000000010;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

DMemory[62] = 32'd666;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

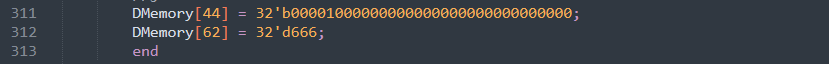
end

end

endmodule

* **Simulation**

Initialize the value at memory location 62.

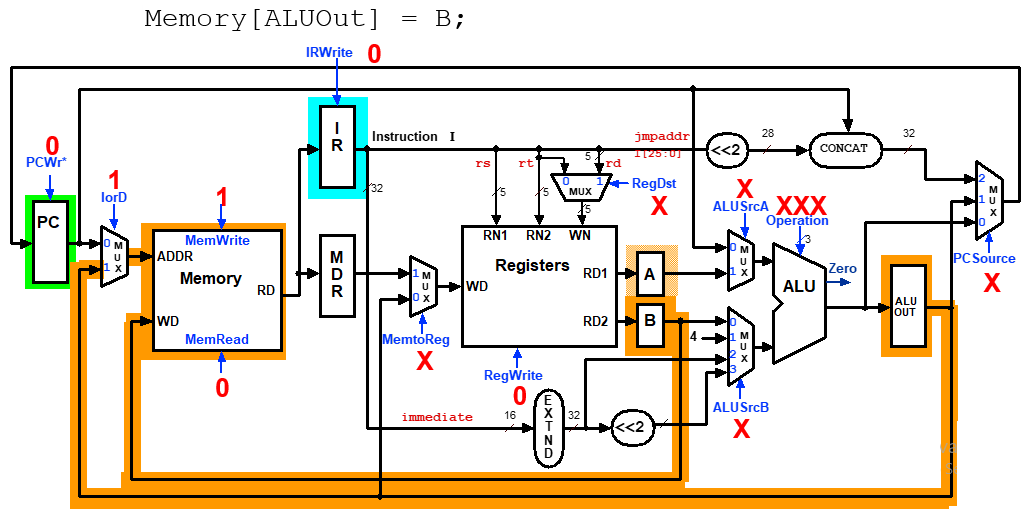


Obtain result

A picture containing text

Description automatically generated

9) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Execution Steps 5 Memory Access - Write (sw)



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|Rs=%d|RD1=%d|IMM=%h|ALU\_in\_A=%d|ALU\_in\_B=%h|ALU\_out=%d|Write\_data=%d",clk,reset,instruction,instruction[20:16],W\_RD1,instruction[15:0],ALU\_in\_A[31:0],ALU\_in\_B[31:0],ALU\_out[31:0],B\_data);

Iord=1;

MemRead=0;

MemWrite=1;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

Operation\_ALU=3'b101; // ID

PCWr=0;

reset=1;clk=0;

#15 reset=0;

#15 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, W\_RD1, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'h8;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

content = 32'b10101110010100010000000000000010;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

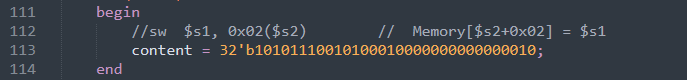
end

end

endmodule

* **Simulation**

Initialize the sw instruction in the holding register module.

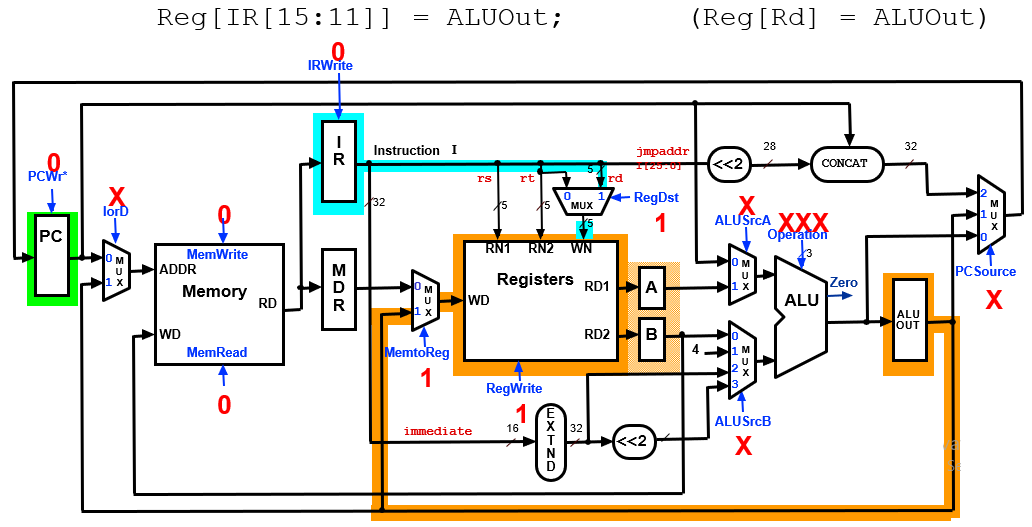


Obtain the result

Diagram

Description automatically generated with low confidence

10) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step 7: ALU Instruction (R-Type)



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|ALU\_in\_A=%d|ALU\_in\_B=%d|ALU\_out=%d|Rs=%d|Rt=%d|RD1=%d|RD2=%d|write\_data=%d",clk,reset,instruction,ALU\_in\_A[31:0],ALU\_in\_B[31:0],ALU\_out[31:0],instruction[25:21], instruction[20:16], W\_RD1, W\_RD2,mux\_2\_out);

// state 6 R-type

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=1;

RegWrite=1;

RegDst=1;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

Operation\_ALU=3'b000; // ID

PCWr=0;

reset=1;clk=0;

#15 reset=0;

#25 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, A\_data, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'h8;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

content = 32'b00000010010100111010000000100000;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

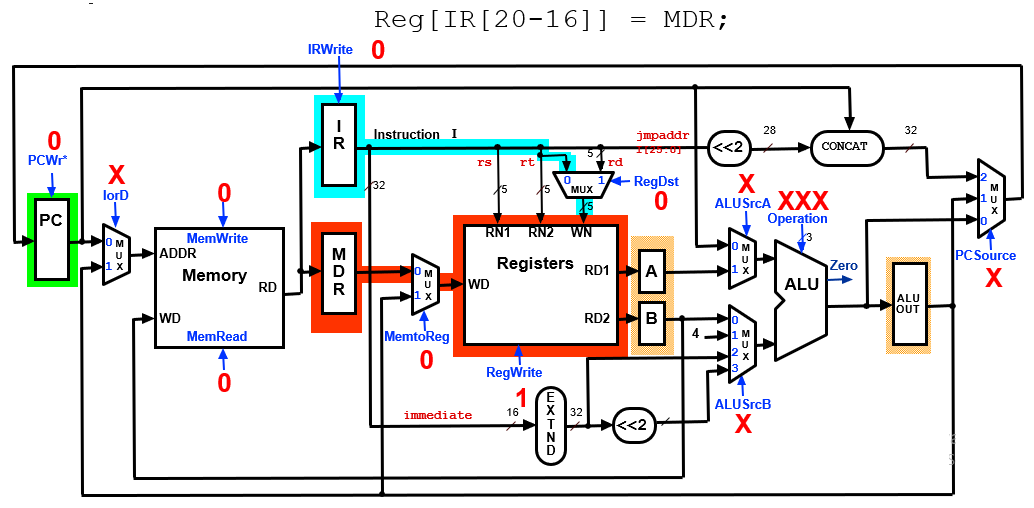
* **Simulation**

The result from step 6 is stored in register file as shown.

Graphical user interface

Description automatically generated with low confidence

11) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor inMulticycle Execution Steps 4 Memory Read Completion (lw)



* **Code**

module testbench;

reg clk, reset;

reg PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

reg [2:0]ALUop;

reg [1:0] ALUSrcB,PCSource;

reg [2:0] Operation\_ALU;

wire [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

wire [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

wire [27:0] jump\_28\_bit;

Datapath\_Multi\_cycle\_Processor dut(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

always #5 clk=~clk;

initial begin

$monitor($time,,"clk=%b|reset=%b|instruction=%h|ALU\_out=%d|mux\_1\_out=%d|Mem\_Read\_data=%d|",clk,reset,instruction,ALU\_out[31:0],mux\_1\_out,Mem\_Read\_data);

Iord=1;

MemRead=1;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=1;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

Operation\_ALU=3'b101; // ID

PCWr=0;

reset=1;clk=0;

#15 reset=0;

#15 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk, reset,PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead,ALUop,ALUSrcB,PCSource,Operation\_ALU,ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,jump\_28\_bit,mux\_1\_out,W\_RD1, W\_RD2);

input clk, reset;

input PCWrite,Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA,PCWr,IRwrite,MemRead;

input [2:0]ALUop;

input [1:0] ALUSrcB,PCSource;

input [2:0] Operation\_ALU;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr,mux\_1\_out;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold,W\_RD1, W\_RD2;

output [27:0] jump\_28\_bit;

wire [31:0] Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire zero,PCWrcond,and\_out;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(Mem\_Read\_data,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, W\_RD1, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=32'h8;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]

content = 32'b10001110010100010000000000000010;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;//$t0

Regfile[9]=32'd2;//$t1

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;//$s1

Regfile[18]=32'd60;//$s2

Regfile[19]=32'd30;//$s3

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1 //r[21]=t0+t1=1+2=3

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22=d30

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1 = d30 //memory[62]=d30

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

//R[17]=memory[62]=d30

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x16 //R[23]=0x16

DMemory[28] = 32'b00100000000101110000000000010000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[32] = 32'b00100000000100100000000000110111;

//addi $s3, $zero, 0x34 // load immediate value 0x22 to register $s3

DMemory[36] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x119 // load immediate value 0x77 to register $s5

DMemory[40] = 32'b00100000000101010000000001110111;

//j 0x00

DMemory[44] = 32'b00001000000000000000000000000000;

DMemory[62] = 32'd666;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

* **Simulation**

The result from step 3’ will be stored in the register file of location 17 as shown.

Graphical user interface

Description automatically generated with medium confidence

### III.4 EXPERIMENT NO. 4

1) Write the Verilog code to implement the Microprocessor Control module using FSM with the following State graph:

Diagram

Description automatically generated

module controller(in\_reset,opcode, reset,clk,PCWrite,Iord,MemRead,MemWrite

,IRwrite,MemtoReg,RegWrite,RegDst,ALUSrcA,ALUSrcB,PCSource,ALUop,PCWrcond);

// ~~~~~~~~~~~~~~~~~~~ PORTS ~~~~~~~~~~~~~~~~~~~ //

// opcode, clock, and reset inputs

input [5:0] opcode; // from instruction register

input clk,in\_reset;

// control signal outputs

output reg PCWrite,Iord,MemRead,MemWrite,IRwrite,MemtoReg,RegWrite,RegDst,ALUSrcA;

output reg [1:0] ALUSrcB,PCSource;

output reg [2:0] ALUop;

output reg PCWrcond;

output reg reset;

// ~~~~~~~~~~~~~~~~~~~ REGISTER ~~~~~~~~~~~~~~~~~~~ //

// 4-bit state register

reg [3:0] state;

// ~~~~~~~~~~~~~~~~~~~ PARAMETERS ~~~~~~~~~~~~~~~~~~~ //

// state parameters

parameter s0 = 4'd0;

parameter s1 = 4'd1;

parameter s2 = 4'd2;

parameter s3 = 4'd3;

parameter s4 = 4'd4;

parameter s5 = 4'd5;

parameter s6 = 4'd6;

parameter s7 = 4'd7;

parameter s8 = 4'd8;

parameter s9 = 4'd9;

parameter s10 = 4'd10;

parameter s\_Reset = 4'd11; // reset

// opcode[5:4] parameters

parameter J = 6'b000010; // Jump or NOP

parameter R = 6'b000000; // R-type

parameter BEQ = 6'b000100; // Branch

parameter BNE = 6'b000101; // Branch

parameter SW = 6'b101011; // I-type

parameter LW = 6'b100011; // I-type

parameter ADDI = 6'b001000; // I-type

// OP code control for ALU

parameter OP\_R\_TYPE = 3'b000;

parameter OP\_I\_TYPE = 3'b001;

parameter OP\_J\_TYPE = 3'b010;

parameter OP\_BR\_TYPE = 3'b111;

parameter OP\_IF\_TYPE = 3'b100;

parameter OP\_ID\_TYPE = 1;

parameter OP\_RS\_TYPE = 3'b110;

// ~~~~~~~~~~~~~~~~~~~ STATE MACHINE ~~~~~~~~~~~~~~~~~~~ //

// control state machine

always @(posedge clk or posedge in\_reset)

begin

// check for reset signal. If set, write zero to PC and switch to Reset State on next CC.

if (in\_reset) begin

PCWrite=0;

Iord=0;

MemRead=1;

MemWrite=0;

IRwrite=1;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_RS\_TYPE;

PCWrcond=0;

reset =1;

state <= s\_Reset;

end

else

begin // if reset signal is not set, check state at pos edge

case (state)

s\_Reset:

begin

PCWrite=0;

Iord=0;

MemRead=1;

MemWrite=0;

IRwrite=1;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_RS\_TYPE;

PCWrcond=0;

reset =0;

state <= s0;

$display("state Reset");

end

s0:

begin

Iord=0;

MemRead=1;

MemWrite=0;

IRwrite=1;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_IF\_TYPE;

PCWrcond=0;

state <= s1;

PCWrite=1;

$display("state 0");

end

s1:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b11;

PCSource=2'b00;

ALUop=OP\_ID\_TYPE;

PCWrcond=0;

$display("state 1");

case(opcode[5:0])

J: state <= s9;

R: state <= s6;

SW: state <= s2;

LW: state <= s2;

ADDI: state <= s2;

BEQ: state <= s8;

endcase

end

s2:

begin

PCWrite=0;

Iord=1;

MemRead=1;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

$display("state 2");

if(opcode[5:0]== ADDI)

begin

state <= s10;

$display("ADDI state");

end

else if(opcode[5:0]== SW)

begin

state <= s5;

$display("SW state");

end

else

begin

state <= s3;

$display("LW state");

end

$display("state 2");

end

s3:

begin

PCWrite=0;

Iord=1;

MemRead=1;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

state <= s4;

$display("state 3");

end

s4:

begin

PCWrite=0;

Iord=1;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=1;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 4");

end

s5:

begin

PCWrite=0;

Iord=1;

MemRead=0;

MemWrite=1;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 5");

end

s6:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

ALUop=OP\_R\_TYPE;

PCWrcond=0;

state <= s7;

$display("state 6");

end

s7:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=1;

RegWrite=1;

RegDst=1;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

ALUop=OP\_R\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 7");

end

s8:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b01;

ALUop=OP\_BR\_TYPE;

PCWrcond=1;

state <= s0;

$display("state 8");

end

s9:

begin

PCWrite=1;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b10;

ALUop=OP\_J\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 9");

end

s10:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=1;

RegWrite=1;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

ALUop=OP\_R\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 7");

end

default: begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_RS\_TYPE;

PCWrcond=0;

$display("state default control");

state <= s\_Reset;

end

endcase

end

end

endmodule

2) Write the ALU control module with the following input and output truth table:

**Table

Description automatically generated**

module ALU\_Control(Op\_intstruct,ints\_function,ALUOp);

input [5:0] ints\_function;

input [2:0] Op\_intstruct;

output reg [2:0] ALUOp;

// OP code control for ALU

parameter OP\_R\_TYPE = 3'b000;

parameter OP\_I\_TYPE = 3'b001;

parameter OP\_J\_TYPE = 3'b010;

parameter OP\_BR\_TYPE = 3'b011;

parameter OP\_IF\_TYPE = 3'b100;

parameter OP\_ID\_TYPE = 3'b101;

parameter OP\_RS\_TYPE = 3'b110;

always @(\*)

begin

case(Op\_intstruct)

OP\_R\_TYPE: // R -Type Instruction look at fuction

begin

ALUOp =3'b000;

if(ints\_function==6'b100000) // add

begin

ALUOp =3'b000;

$display("fuction Add");

end

if(ints\_function==6'b100010) // sub

begin

ALUOp =3'b001;

$display("fuction sub");

end

if(ints\_function==6'b100100) // and

begin

ALUOp =3'b010;

$display("fuction and");

end

if(ints\_function==6'b100101) // or

begin

ALUOp =3'b010;

$display("fuction or ");

end

end

OP\_I\_TYPE:

begin

ALUOp =3'b000;

$display("LW or SW");

end

OP\_J\_TYPE:

begin

ALUOp =3'b000;

$display("Jump");

end

OP\_BR\_TYPE: // beq instruction

begin

ALUOp =3'b111;

$display("BEQ or BNE");

end

OP\_IF\_TYPE: // Store Instruction

begin

ALUOp =3'b000;

$display("Add IF");

end

OP\_ID\_TYPE: // addi Instruction

begin

ALUOp =3'b000;

$display("add ID");

end

OP\_RS\_TYPE: //bne

begin

ALUOp =3'b111;

$display("Reset OP");

end

default :

begin

ALUOp =3'b000;

$display("ALU default");

end

endcase

end

endmodule

3) Write Verilog code to implement complete Multi-Cycle Processor module

Diagram, schematic

Description automatically generated

module testbench;

reg clk,reset;

wire [31:0] ALU\_out\_hold,ALU\_out,PC\_in,PC\_out,Mem\_Read\_data;

wire [31:0] instruction,MDR\_out,mux\_2\_out,B\_data,ALU\_in\_B,ALU\_in\_A,Jump\_addr;

wire [27:0] jump\_28\_bit;

wire PCWr,IRwrite,MemRead;

Datapath\_Multi\_cycle\_Processor DUT(clk,reset,PCWr,IRwrite,MemRead,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,mux\_2\_out,B\_data,ALU\_in\_B,ALU\_in\_A,ALU\_out,ALU\_out\_hold,Jump\_addr,jump\_28\_bit);

always #5 clk=~clk;

initial begin

$monitor($time," clk=%b|reset=%b|PCWr=%b,MemRead=%b,PC\_in=%d, PCout=%d|Mem\_Read\_data=%b,IRwrite=%b,opcode=%b,rs=%d,rt=%d,rd=%d,imm=%d,MDR\_out=%b,mux\_2\_out=%d,B\_data=%d, ALU\_in\_A=%d , ALU\_in\_B=%d ,ALU\_out=%d, ALU\_out\_hold=%d,Jump\_addr=%d,jump\_28\_bit=%d",clk,reset,PCWr,MemRead,PC\_in,PC\_out,Mem\_Read\_data[31:26],IRwrite,instruction[31:26],instruction[25:21],instruction[20:16],instruction[15:11], instruction[15:0],MDR\_out[31:26],mux\_2\_out,B\_data,ALU\_in\_A,ALU\_in\_B,ALU\_out,ALU\_out\_hold,Jump\_addr,instruction[25:0]);

reset=1;clk=0;

#10 reset=0;

#800 $finish;

end

endmodule

module Datapath\_Multi\_cycle\_Processor(clk,in\_reset,PCWr,IRwrite,MemRead,PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,mux\_2\_out,B\_data,ALU\_in\_B,ALU\_in\_A,ALU\_out,ALU\_out\_hold,Jump\_addr,jump\_28\_bit);

input clk, in\_reset;

output [31:0] ALU\_in\_B,ALU\_in\_A,ALU\_out,B\_data,mux\_2\_out,Jump\_addr;

output [31:0] PC\_in,PC\_out,Mem\_Read\_data,instruction,MDR\_out,ALU\_out\_hold;

output PCWr,IRwrite,MemRead;

output [27:0] jump\_28\_bit;

wire [2:0]ALUop;

wire [31:0] mux\_1\_out,W\_RD1, W\_RD2,Extend\_out,Branch\_addr,A\_data;

wire [4:0] mux\_3\_out;

wire PCWrite,zero,PCWrcond,and\_out,reset;

wire Iord,MemWrite,MemtoReg,RegWrite,RegDst,ALUSrcA;

wire [1:0] ALUSrcB,PCSource;

wire [2:0] Operation\_ALU;

Program\_Counter comp1(clk, reset,PCWr,PC\_in, PC\_out);

Mux\_32\_bit comp2(PC\_out, ALU\_out\_hold, mux\_1\_out, Iord);

Data\_Memory comp3(clk,mux\_1\_out, B\_data, Mem\_Read\_data, MemRead, MemWrite);

holding\_reg comp4(instruction, Mem\_Read\_data, IRwrite, clk, reset);

holding\_reg comp5(MDR\_out, Mem\_Read\_data, 1'b1, clk, reset);

Mux\_32\_bit comp6(MDR\_out,ALU\_out\_hold, mux\_2\_out, MemtoReg);

Register\_File comp7(clk,instruction[25:21], instruction[20:16], mux\_3\_out, W\_RD1, W\_RD2, mux\_2\_out, RegWrite);

Mux\_5\_bit comp8(instruction[20:16], instruction[15:11], mux\_3\_out, RegDst);

Sign\_Extension comp9(instruction[15:0], Extend\_out);

shift\_left\_2 comp10(Extend\_out, Branch\_addr);

holding\_reg comp11(A\_data, W\_RD1, 1'b1, clk, reset);

holding\_reg comp12(B\_data, W\_RD2, 1'b1, clk, reset);

Mux\_32\_bit comp13(PC\_out, A\_data, ALU\_in\_A, ALUSrcA);

Mux4\_32\_bit comp14(B\_data, 32'd4,Extend\_out,Branch\_addr , ALU\_in\_B, ALUSrcB);

alu comp15(Operation\_ALU, ALU\_in\_A, ALU\_in\_B, ALU\_out,zero);

holding\_reg comp16(ALU\_out\_hold, ALU\_out , 1'b1, clk, reset);

shift\_left\_2\_28bit comp17(instruction[25:0], jump\_28\_bit);

concate comp18(PC\_out[31:28],jump\_28\_bit,Jump\_addr);

Mux4\_32\_bit comp19(ALU\_out, ALU\_out\_hold,Jump\_addr, 32'b0, PC\_in, PCSource);

controller comp20(in\_reset,instruction[31:26], reset,clk,PCWrite,Iord,MemRead,MemWrite,IRwrite,MemtoReg,RegWrite,RegDst,ALUSrcA,ALUSrcB,PCSource,ALUop,PCWrcond);

ALU\_Control comp21(ALUop,instruction[5:0],Operation\_ALU);

and comp22(and\_out,zero,PCWrcond);

or comp23(PCWr,and\_out,PCWrite);

endmodule

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

input clk, reset,PC\_write;

input [31:0] PC\_in;

output reg [31:0] PC\_out;

always @ (posedge clk or posedge reset)

begin

if(reset==1'b1)

PC\_out<=0;

else if (PC\_write==1'b1)

PC\_out<=PC\_in;

end

endmodule

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

input [31:0] addr;

input [31:0] write\_data;

output [31:0] read\_data;

input MemRead, MemWrite,clk;

reg [31:0] DMemory [63:0];

integer k;

initial begin

for (k=0; k<64; k=k+1)

begin

DMemory[k] = 32'b0;

end

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[0] = 32'b10101110010100010000000000000010;

//add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x90

DMemory[4] = 32'b00000010010100111010000000100000;

//add $s5 $t0 $t1

DMemory[8] = 32'b00000001000010011010100000100000;

//sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22

DMemory[12] = 32'b00000010010100111000100000100010;

//sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

DMemory[16] = 32'b10101110010100010000000000000010;

//lw $s1, 0x02($s2) // $s1 = Memory[$s2+0x02]

DMemory[20] = 32'b10001110010100010000000000000010;

//beq $t2,$t3, End //beq $t2,$t3, 0x03

DMemory[24] = 32'b00010001010010110000000000000011;

//addi $s7, $zero, 0x10

DMemory[28] = 32'b00100000000101110000000000010000;

//j 0x00

DMemory[32] = 32'b00001000000000000000000000000000;

//addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

DMemory[36] = 32'b00100000000100100000000001010101;

//addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3

DMemory[40] = 32'b00100000000100110000000000100010;

//addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5

DMemory[44] = 32'b00100000000101010000000001110111;

end

assign read\_data = (MemRead) ? DMemory[addr] : 32'bx;

always @(posedge clk)

begin

if (MemWrite)

begin

DMemory[addr] = write\_data;

$display("Data memory write\_addr=%d write\_data=%d",addr,write\_data);

end

end

endmodule

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

input [4:0] read\_addr\_1, read\_addr\_2, write\_addr;

input [31:0] write\_data;

input clk,RegWrite;

reg checkRegWrite;

output reg [31:0] read\_data\_1, read\_data\_2;

reg [31:0] Regfile [31:0];

integer k;

initial

begin

for (k=0; k<32; k=k+1)

begin

Regfile[k] = 32'd10;

end

Regfile[8]=32'd1;

Regfile[9]=32'd2;

Regfile[10]=32'd3; //$t2

Regfile[11]=32'd4; //$t3

Regfile[17]=32'd99;

Regfile[18]=32'd60;

Regfile[19]=32'd30;

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_1=%d,read\_data\_1=%h",read\_addr\_1,read\_data\_1);

end

end

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

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

begin

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

else

begin

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

//$display("read\_addr\_2=%d,read\_data\_2=%h",read\_addr\_2,read\_data\_2);

end

end

always @(posedge clk)

begin

if (RegWrite == 1'b1)

begin

Regfile[write\_addr] = write\_data;

$display("Rigister File write\_addr=%d write\_data=%d",write\_addr,write\_data);

end

end

endmodule

module alu(

input [2:0] alufn,

input [31:0] ra,

input [31:0] rb\_or\_imm,

output reg [31:0] 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\_XOR = 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\_XOR : aluout = ra ^ rb\_or\_imm;

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

module holding\_reg(output\_data, input\_data, write, clk, reset);

// data size

parameter N = 32;

// inputs

input [N-1:0] input\_data;

input write, clk, reset;

// outputs

output [N-1:0] output\_data;

// Register content and output assignment

reg [N-1:0] content;

// update regisiter contents

always @(posedge clk or write)

begin

if (reset)

begin

content <= 0;

end

else if (write)

begin

content <= input\_data;

end

end

assign output\_data = content;

endmodule

module concate(PC\_in,IR\_in,PC\_out);

input [3:0] PC\_in;

input [27:0] IR\_in;

output[31:0] PC\_out;

assign PC\_out={PC\_in, IR\_in};

endmodule

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

input [15:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[15:0]=sign\_in[15:0];

assign sign\_out[31:16]=sign\_in[15]?16'b1111\_1111\_1111\_1111:16'b0;

endmodule

module shift\_left\_2 (sign\_in, sign\_out);

input [31:0] sign\_in;

output [31:0] sign\_out;

assign sign\_out[31:2]=sign\_in[29:0];

assign sign\_out[1:0]=2'b00;

endmodule

module shift\_left\_2\_28bit (sign\_in, sign\_out);

input [25:0] sign\_in;

output [27:0] sign\_out;

assign sign\_out={2'b00,sign\_in};

endmodule

module Mux\_5\_bit (in0, in1, mux\_out, select);

parameter N = 5;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module Mux\_32\_bit (in0, in1, mux\_out, select);

parameter N = 32;

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

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

input select;

assign mux\_out = select? in1: in0 ;

endmodule

module Mux4\_32\_bit (in0, in1,in2, in3, mux\_out, select);

parameter N = 32;

input [N-1:0] in0, in1,in2,in3;

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

input [1:0]select;

assign mux\_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);

endmodule

//////////////////////////

module controller(in\_reset,opcode, reset,clk,PCWrite,Iord,MemRead,MemWrite

,IRwrite,MemtoReg,RegWrite,RegDst,ALUSrcA,ALUSrcB,PCSource,ALUop,PCWrcond);

// ~~~~~~~~~~~~~~~~~~~ PORTS ~~~~~~~~~~~~~~~~~~~ //

// opcode, clock, and reset inputs

input [5:0] opcode; // from instruction register

input clk,in\_reset;

// control signal outputs

output reg PCWrite,Iord,MemRead,MemWrite,IRwrite,MemtoReg,RegWrite,RegDst,ALUSrcA;

output reg [1:0] ALUSrcB,PCSource;

output reg [2:0] ALUop;

output reg PCWrcond;

output reg reset;

// ~~~~~~~~~~~~~~~~~~~ REGISTER ~~~~~~~~~~~~~~~~~~~ //

// 4-bit state register

reg [3:0] state;

// ~~~~~~~~~~~~~~~~~~~ PARAMETERS ~~~~~~~~~~~~~~~~~~~ //

// state parameters

parameter s0 = 4'd0;

parameter s1 = 4'd1;

parameter s2 = 4'd2;

parameter s3 = 4'd3;

parameter s4 = 4'd4;

parameter s5 = 4'd5;

parameter s6 = 4'd6;

parameter s7 = 4'd7;

parameter s8 = 4'd8;

parameter s9 = 4'd9;

parameter s10 = 4'd10;

parameter s\_Reset = 4'd11; // reset

// opcode[5:4] parameters

parameter J = 6'b000010; // Jump or NOP

parameter R = 6'b000000; // R-type

parameter BEQ = 6'b000100; // Branch

parameter BNE = 6'b000101; // Branch

parameter SW = 6'b101011; // I-type

parameter LW = 6'b100011; // I-type

parameter ADDI = 6'b001000; // I-type

// OP code control for ALU

parameter OP\_R\_TYPE = 3'b000;

parameter OP\_I\_TYPE = 3'b001;

parameter OP\_J\_TYPE = 3'b010;

parameter OP\_BR\_TYPE = 3'b111;

parameter OP\_IF\_TYPE = 3'b100;

parameter OP\_ID\_TYPE = 1;

parameter OP\_RS\_TYPE = 3'b110;

// ~~~~~~~~~~~~~~~~~~~ STATE MACHINE ~~~~~~~~~~~~~~~~~~~ //

// control state machine

always @(posedge clk or posedge in\_reset)

begin

// check for reset signal. If set, write zero to PC and switch to Reset State on next CC.

if (in\_reset) begin

PCWrite=0;

Iord=0;

MemRead=1;

MemWrite=0;

IRwrite=1;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_RS\_TYPE;

PCWrcond=0;

reset =1;

state <= s\_Reset;

end

else

begin // if reset signal is not set, check state at pos edge

case (state)

s\_Reset:

begin

PCWrite=0;

Iord=0;

MemRead=1;

MemWrite=0;

IRwrite=1;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_RS\_TYPE;

PCWrcond=0;

reset =0;

state <= s0;

$display("state Reset");

end

s0:

begin

Iord=0;

MemRead=1;

MemWrite=0;

IRwrite=1;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_IF\_TYPE;

PCWrcond=0;

state <= s1;

PCWrite=1;

$display("state 0");

end

s1:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b11;

PCSource=2'b00;

ALUop=OP\_ID\_TYPE;

PCWrcond=0;

$display("state 1");

case(opcode[5:0])

J: state <= s9;

R: state <= s6;

SW: state <= s2;

LW: state <= s2;

ADDI: state <= s2;

BEQ: state <= s8;

endcase

end

s2:

begin

PCWrite=0;

Iord=1;

MemRead=1;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

$display("state 2");

if(opcode[5:0]== ADDI)

begin

state <= s10;

$display("ADDI state");

end

else if(opcode[5:0]== SW)

begin

state <= s5;

$display("SW state");

end

else

begin

state <= s3;

$display("LW state");

end

$display("state 2");

end

s3:

begin

PCWrite=0;

Iord=1;

MemRead=1;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

state <= s4;

$display("state 3");

end

s4:

begin

PCWrite=0;

Iord=1;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=1;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 4");

end

s5:

begin

PCWrite=0;

Iord=1;

MemRead=0;

MemWrite=1;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b10;

PCSource=2'b00;

ALUop=OP\_I\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 5");

end

s6:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

ALUop=OP\_R\_TYPE;

PCWrcond=0;

state <= s7;

$display("state 6");

end

s7:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=1;

RegWrite=1;

RegDst=1;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

ALUop=OP\_R\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 7");

end

s8:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b01;

ALUop=OP\_BR\_TYPE;

PCWrcond=1;

state <= s0;

$display("state 8");

end

s9:

begin

PCWrite=1;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b10;

ALUop=OP\_J\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 9");

end

s10:

begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=1;

RegWrite=1;

RegDst=0;

ALUSrcA=1;

ALUSrcB=2'b00;

PCSource=2'b00;

ALUop=OP\_R\_TYPE;

PCWrcond=0;

state <= s0;

$display("state 7");

end

default: begin

PCWrite=0;

Iord=0;

MemRead=0;

MemWrite=0;

IRwrite=0;

MemtoReg=0;

RegWrite=0;

RegDst=0;

ALUSrcA=0;

ALUSrcB=2'b01;

PCSource=2'b00;

ALUop=OP\_RS\_TYPE;

PCWrcond=0;

$display("state default control");

state <= s\_Reset;

end

endcase

end

end

endmodule

module ALU\_Control(Op\_intstruct,ints\_function,ALUOp);

input [5:0] ints\_function;

input [2:0] Op\_intstruct;

output reg [2:0] ALUOp;

// OP code control for ALU

parameter OP\_R\_TYPE = 3'b000;

parameter OP\_I\_TYPE = 3'b001;

parameter OP\_J\_TYPE = 3'b010;

parameter OP\_BR\_TYPE = 3'b011;

parameter OP\_IF\_TYPE = 3'b100;

parameter OP\_ID\_TYPE = 3'b101;

parameter OP\_RS\_TYPE = 3'b110;

always @(\*)

begin

case(Op\_intstruct)

OP\_R\_TYPE: // R -Type Instruction look at fuction

begin

ALUOp =3'b000;

if(ints\_function==6'b100000) // add

begin

ALUOp =3'b000;

$display("fuction Add");

end

if(ints\_function==6'b100010) // sub

begin

ALUOp =3'b001;

$display("fuction sub");

end

if(ints\_function==6'b100100) // and

begin

ALUOp =3'b010;

$display("fuction and");

end

if(ints\_function==6'b100101) // or

begin

ALUOp =3'b010;

$display("fuction or ");

end

end

OP\_I\_TYPE:

begin

ALUOp =3'b000;

$display("LW or SW");

end

OP\_J\_TYPE:

begin

ALUOp =3'b000;

$display("Jump");

end

OP\_BR\_TYPE: // beq instruction

begin

ALUOp =3'b111;

$display("BEQ or BNE");

end

OP\_IF\_TYPE: // Store Instruction

begin

ALUOp =3'b000;

$display("Add IF");

end

OP\_ID\_TYPE: // addi Instruction

begin

ALUOp =3'b000;

$display("add ID");

end

OP\_RS\_TYPE: //bne

begin

ALUOp =3'b111;

$display("Reset OP");

end

default :

begin

ALUOp =3'b000;

$display("ALU default");

end

endcase

end

endmodule

4) Write following Assembly code and compile into binary machine code to verify R-Type Instruction execution and Datapath operation of the complete Multi-Cycle Processor, write the testbench simulate and check the simulation output data.

add $s4, $s2, $s3 // $s4 = $s2 + $s3   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3

and $s4, $s2, $s3 // $s4 = $s2 + $s3   
or $s1, $s2, $s3 // $s1 = $s2 – $s3

Load the following instruction into the data\_memory module in the code above to obtain result

Text

Description automatically generated

After executed instruction add

Table

Description automatically generated

After executed instruction sub

A picture containing table

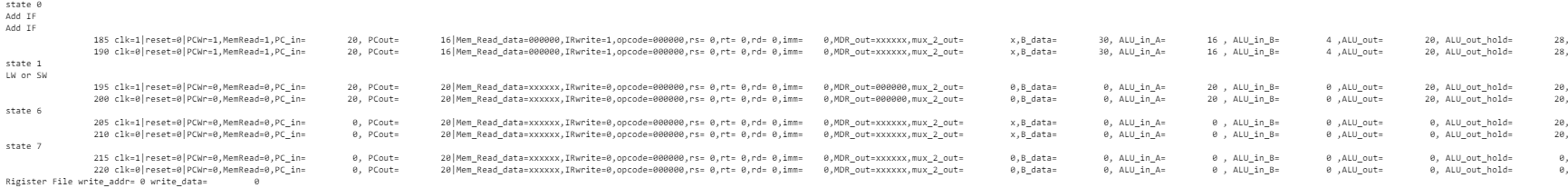
Description automatically generated

After executed instruction and

A picture containing table

Description automatically generated

After executed instruction or



5) Write following Assembly code and compile into binary machine code to verify SW and LW Instruction execution and Datapath operation of the complete Multi-Cycle Processor, write the testbench simulate and check the simulation output data.

sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

Load the following instruction sets.

Text

Description automatically generated

After executed instruction sw

Table

Description automatically generated with medium confidence

After executed instruction lw

Text

Description automatically generated

The result is correct since after store 99 to memory location 62, we load it to register location 22.

6) Write following Assembly code and compile into binary machine code to verify beq and bne Instruction execution and Datapath operation of the complete Multi-Cycle Processor, write the testbench simulate and check the simulation output data.

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

add $s4, $s2, $s3//   
beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

add $s4, $s2, $s3

End:

Load instruction set as follow

Text

Description automatically generated

Value of $s5 and $s4 are not equal.



The result jump to END after executing the first instruction as expected at 35 time unit.

##### A picture containing text Description automatically generated

7) Compile the following code into binary machine code and store in Instruction memory to test the complete Multi-Cycle Complete Processor.

**Testing Assembly Program 1:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2 R18

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3 R19

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5 R21

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=R18+R19 = 0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=R18-R19=0x33   
sw $s1, 0x02($s2) // Memory[0x55+0x02] = $s1 0x33 => RAM[0x57]

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8  
beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

Insert the following instruction set.

Text

Description automatically generated

Result after executing instruction 0 and 4

Table

Description automatically generated with low confidence

Result after executing instruction 8,12, and 16

Text

Description automatically generated with medium confidence

Result after executing instruction 20,24, and 28

Table

Description automatically generated with low confidence

8) Compile the following code into binary machine code and store in Instruction memory to test the Complete Multi-Cycle Processor.

**Testing Assembly Program 2:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22   
sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

Insert the following instruction set

Text

Description automatically generated

The result is the same as above except for at instruction 28, since s5 =s4, the branch is triggered, and next instruction is waiting at END in 325 time unit as shown.

Table

Description automatically generated

9) Write the assembly code to carry following calculation formula

Sum = 1+2+3+ …..9;

Compile the following code into binary machine code and store in Instruction memory to test the Complete Multi-cycle Processor.

The assembly code is shown

A screenshot of a computer

Description automatically generated with medium confidence

The testbench result is

A picture containing text

Description automatically generated