MIPS in One Cycle Project

Using Verilog Implementation with Vivado

Mahmoud Ghamry

Introduction

This report presents the design and implementation of a single-cycle MIPS (Microprocessor without Interlocked Pipeline Stages) processor using Verilog HDL and synthesized in Xilinx Vivado. The MIPS architecture, developed by MIPS Computer Systems, represents one of the pioneering examples of Reduced Instruction Set Computing (RISC) architecture, which emphasizes simplicity and efficiency in processor design.

System Architecture and Components: The processor implements a 32-bit data path with the following key architectural features:

- 32-bit instruction format
- 32 x 32-bit Register File
- Word-addressable memory
- Big-endian memory organization
- Von Neumann architecture with separate instruction and data memories

Core Components Implementation:

1. Arithmetic Logic Unit (ALU)
    - Supports operations: ADD, SUB, AND, OR, SLT
    - 32-bit operands with flag generation
    - Zero flag for branch decision making
2. Control Unit
    - Generates control signals based on opcode and function fields
    - Manages data path control including:
        - ALU operation selection
        - Register write enable
        - Memory read/write signals
        - Branch control
3. Memory System
    - Instruction Memory (ROM-style)
    - Data Memory (RAM-style)
    - Word-aligned addressing
    - Synchronous read/write operations

4. Register File
    - 32 general-purpose registers (R0-R31)
    - R0 hardwired to zero
    - Dual read ports and single write port
    - Synchronous write, asynchronous read

Instruction Set Support: The processor implements fundamental MIPS instructions including:

- Arithmetic: ADD, SUB, ADDI
- Logical: AND, OR
- Memory: LW, SW
- Branch: BNE
- Comparison: SLT,

Design Methodology: The implementation follows a modular design approach with:

1. Behavioral Verilog modeling
2. Hierarchical module organization
3. Synchronous design principles
4. Testbench-driven verification
5. Synthesis optimization for FPGA implementation

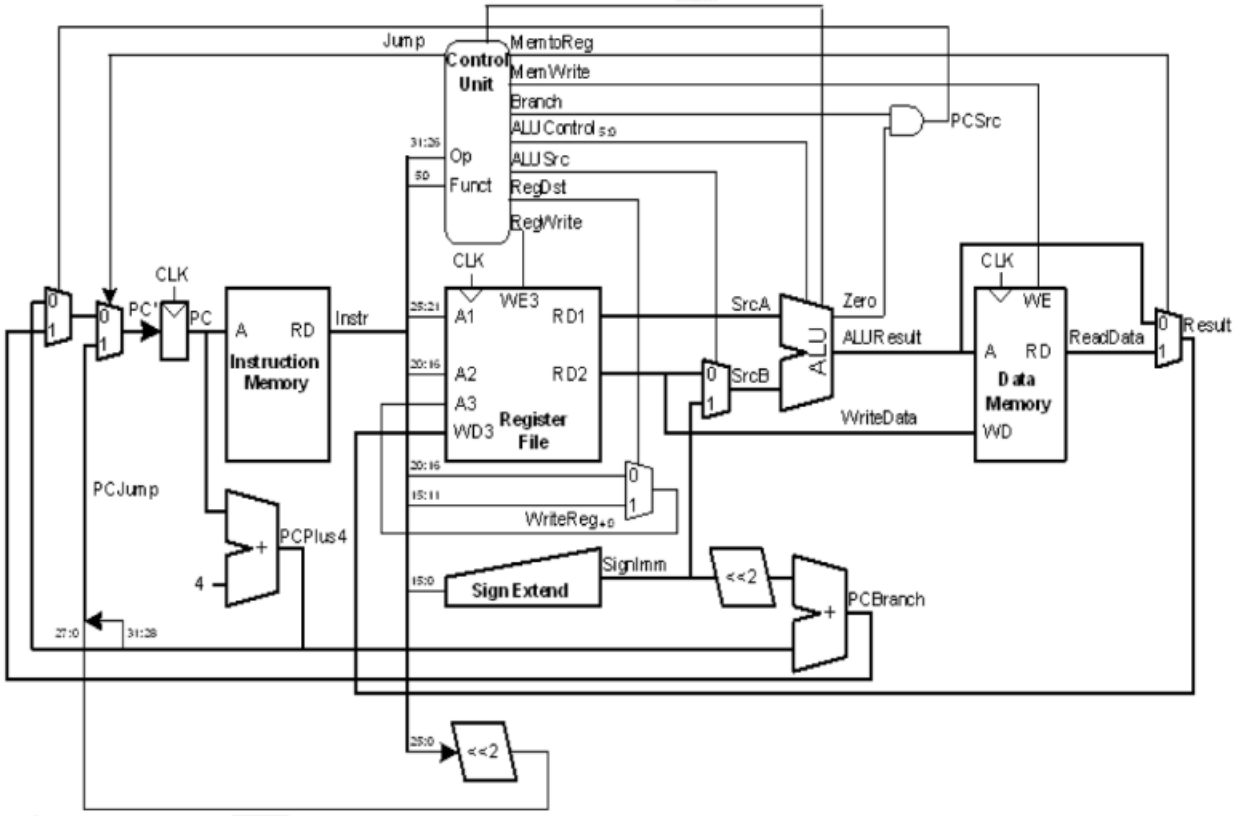
Performance Considerations:

- Single-cycle execution with fixed latency
- Critical path analysis through ALU and memory access
- Resource utilization optimization
- Timing constraints management

For more details visit my repo : MIPS

Architecture

Main Structure



Formats

R-Type

6	5	5	5	5	6
opcode	rt	rs	rd	shift	func

I-Type

6	5	16
opcode	rt	immediate/offset

J-Type

6	26
opcode	offset

Files

Data Memory

```
module Data_Memory(
    A , WD , clk , WE , RD
);

parameter WIDTH = 32 ;
input clk , WE ;
input [WIDTH-1:0]WD,A;
output [WIDTH-1:0]RD;

reg [WIDTH-1:0]mem[1023:0];

assign  RD = mem[A] ;

always @(posedge clk) begin

    if(WE)begin
        mem[WD] <= A ;
    end
    else begin

    end

end

endmodule
```

Instruction Memory

```
module Instruction_Memory(
    A , RD
);

    parameter WIDTH = 32 ;
    input [WIDTH-1:0]A;
    output [WIDTH-1:0]RD;

    reg [WIDTH-1:0]mem[1023:0];
    initial
     $readmemh("mem.dat",mem);



    assign RD = mem[A/4];       You, 11 hours ago • Edits

endmodule
```

Register File

```
module Register_File (
    input [4:0] A1, A2, A3,
    input [31:0] WD3,
    input clk,rst, WE3,
    output reg [31:0] RD1, RD2
);

    reg [31:0] Register_File [31:0];

    always @(*) begin
        RD1 = Register_File[A1];
        RD2 = Register_File[A2];
    end

    integer i = 0;
    always @(posedge clk) begin
        if(!rst)begin
            for ( i=0 ; i < 32 ; i = i + 1 ) begin
                Register_File[i] <= 0;
            end
        end
        else if (WE3) begin
            Register_File[A3] <= WD3;
        end
    end

endmodule
```

Adder

```
module Adder(       You, 4 days ago • First Push
    in1,in2,out
);

    parameter WIDTH = 32 ;
    input [WIDTH-1:0]in1,in2;
    output [WIDTH-1:0]out;
    assign out = in1 +in2 ;

endmodule
```

ALU

```
module ALU(
    A,B,SELECT,OUT,ZERO
);

    parameter WIDTH = 32 ;

    input [WIDTH-1:0]A,B;
    input [2:0]SELECT;
    output reg[WIDTH-1:0]OUT;
    output reg ZERO;

    always @(*) begin
        case (SELECT)
            3'b000:begin
                OUT = A & B ;
            end
            3'b001:begin
                OUT = A | B ;
            end
            3'b010:begin
                OUT = A + B ;
            end
            3'b110:begin
                OUT = A - B ;
            end
            3'b111:begin
                OUT = ( A < B ) ? 1 : 0 ;       You, 4 days ago • Edit Files
            end
        endcase
        if(!OUT)begin
            ZERO = 1 ;
        end
        else begin
            ZERO = 0 ;
        end
    end
endmodule
```

ALU Decoder

```
module ALUop(       You, 4 days ago • First Push
    input [5:0]func,
    input [1:0]aluop,
    output reg [2:0]op
);

    always @(*) begin
        if(aluop == 2'b00)begin
            op = 3'b010 ;
        end
        else if (aluop == 2'b01) begin
            op = 3'b110 ;
        end
        else if (aluop == 2'b10 ) begin
            if (func == 6'b100_000) begin
                op = 3'b010;
            end
            else if (func == 6'b100_010) begin
                op = 3'b110 ;
            end
            else if (func == 6'b100_100) begin
                op = 3'b000 ;
            end
            else if (func == 6'b100_101) begin
                op = 3'b001 ;
            end
            else if (func == 6'b101_010) begin
                op = 3'b111 ;
            end
            else begin

            end
        end
        else begin

        end
    end

endmodule
```

Control Unit

```
module Control_Unit(
    op,funct,MemtoReg,MemWrite,Branch,Jump,ALUControl,ALUSrc,RegDst,RegWrite
);

    input [5:0]op,funct;
    output reg MemtoReg,MemWrite,Branch,ALUSrc,RegDst,Jump,RegWrite;
    output  [2:0]ALUControl;

    reg [1:0]ALUop;


     ALUop aludecode(funct,ALUop,ALUControl);

    always @(*) begin
        case(op)       You, yesterday • Edits

            6'b000_000:begin
                {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,ALUop,Jump} = 9'b1100_00100;
            end
            6'b100_011:begin
                {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,ALUop,Jump} = 9'b1010_01000;
            end
            6'b101_011:begin
                {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,ALUop,Jump} = 9'b0010_10000;
            end
            6'b000_100:begin
                {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,ALUop,Jump} = 9'b0001_00010;
            end
            6'b001_000:begin
                {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,ALUop,Jump} = 9'b1010_00000;
            end
            6'b000_010:begin
                {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,ALUop,Jump} = 9'b0000_00001;
            end

                default: begin
                    {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,ALUop,Jump} = 0;
                end

        endcase
    end

endmodule
```

PC (Program Counter)

```
module DFF(
    D,Q,clk,rst
);
parameter WIDTH = 32 ;          You, yesterday • Add Files
input [WIDTH-1:0]D;
input clk,rst;
output reg[WIDTH-1:0]Q;

always @(posedge clk) begin
    if (!rst) begin
        Q <= 0 ;
    end
    else begin
        Q <= D ;
    end
end
endmodule
```

MUX 2 X 1

```
module mux2_1(
    A,B,SELECT,OUT
);

  parameter WIDTH = 1 ;
        You, 1 second ago • Uncommitted changes

  input [WIDTH-1:0]A,B;
  input SELECT;
  output [WIDTH-1:0]OUT;

  assign OUT = (SELECT) ? B : A;
endmodule
```

Shifter

```
module shift2(      You, 4 days ago • First Push
    datain,dataout
);
    parameter WIDTH = 32 ;

    input [WIDTH-1:0]datain;
    output [WIDTH-1:0]dataout;

    assign dataout = datain << 2 ;
endmodule
```

Sign extend

```
module signextend(      You, 4 days ago • First Push
    datain,dataout
);
    input [15:0]datain;
    output [31:0]dataout;

    assign dataout = {{16{datain[15]}},datain};
endmodule
```

MIPS Top

```
module datapath(
    clk,rst,Result
);
    input clk,rst;
    output [31:0]Result;
    parameter WIDTH = 32 ;      You, yesterday • Add Files

    wire [WIDTH-1:0]input1_mux1,out_mux1;

    wire [WIDTH-1:0]PC_plus4;
    wire [WIDTH-1:0]instr;
    wire MemtoReg,MemWrite,Branch,Jump,ALUSrc,RegDst,RegWrite;
    wire [2:0]ALUControl;
    Control_Unit CPU(instr[31:26],instr[5:0],MemtoReg,MemWrite,Branch,Jump,ALUControl,ALUSrc,RegDst,RegWrite);

    wire [31:0]addr;
    mux2_1 #(.WIDTH(WIDTH))mux1(input1_mux1,addr,Jump,out_mux1);

   assign addr = {PC_plus4[31:28],instr[25:0],2'b00};
    wire [WIDTH-1:0]PC_OUT;
    DFF #(.WIDTH(WIDTH))PC(out_mux1,PC_OUT,clk,rst);


    Adder #(.WIDTH(WIDTH))PCPLUS4(PC_OUT,4,PC_plus4);


    Instruction_Memory #(.WIDTH(WIDTH))INSTR_MEM(PC_OUT,instr);



    wire [4:0]WriteReg;
    mux2_1 #(.WIDTH(5))mux2(instr[20:16],instr[15:11],RegDst,WriteReg);

    wire [WIDTH-1:0]SignImm,SrcB,wiretomux3;
    mux2_1 #(.WIDTH(WIDTH))mux3(wiretomux3,SignImm,ALUSrc,SrcB);
```

```
    signextend SIGNEXTENDED(instr[15:0],SignImm);

    wire [WIDTH-1:0]OUT_shifter2;
    shift2 #(.WIDTH(WIDTH))SHIFTER_BRANCH(SignImm,OUT_shifter2);

    wire [WIDTH-1:0]PCbranch;
    Adder #(.WIDTH(WIDTH))PCBRANCH(OUT_shifter2,PC_plus4,PCbranch);

    wire [WIDTH-1:0]SrcA;
    Register_File REg_FILE(instr[25:21],instr[20:16],WriteReg,Result,clk,rst,RegWrite,SrcA,wiretomux3);


    wire [WIDTH-1:0]OUT_ALU;
    ALU #(.WIDTH(WIDTH))alu(SrcA,SrcB,ALUControl,OUT_ALU,ZERO);

    wire PCSrc;
    assign PCSrc = Branch & ZERO ;
    mux2_1 #(.WIDTH(WIDTH))mux4(PC_plus4,PCbranch,PCSrc,input1_mux1);

    wire [WIDTH-1:0]ReadData;
    Data_Memory #(.WIDTH(WIDTH))DataMEM(OUT_ALU,wiretomux3,clk,MemWrite,ReadData);

    mux2_1 #(.WIDTH(WIDTH))mux5(OUT_ALU,ReadData,MemtoReg,Result);

endmodule
```

Testbench

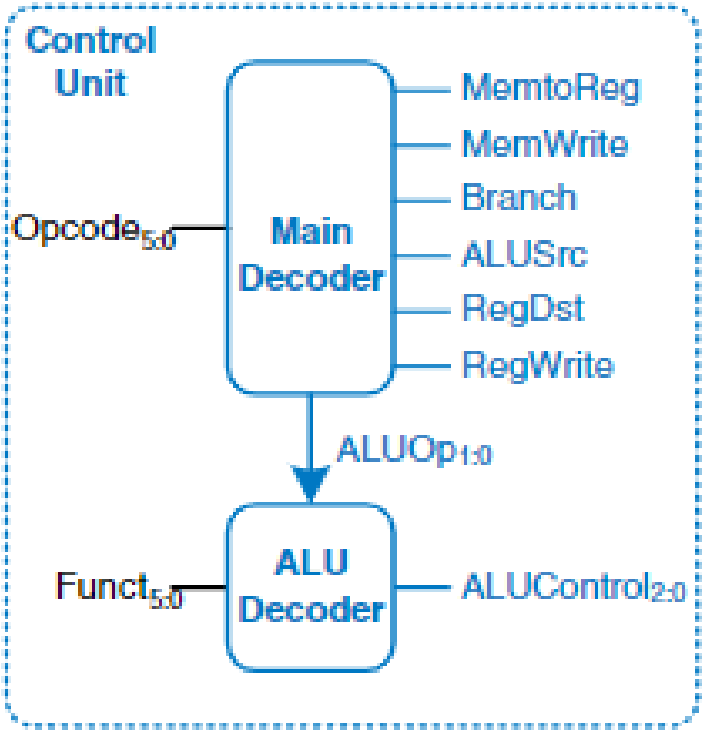
```
module MIPS_tb;

  logic clk ,rst;
  logic [31:0]Result;

  datapath MIPS(clk,rst,Result);

  initial begin
    clk = 0 ;
    forever begin
        #10 clk = ~clk;
    end
  end

    initial begin

    rst = 0 ;
    @(negedge clk);
    rst = 1 ;
    #300;       You, 13 hours ago • Add work & Files
    $stop;
    end



endmodule
```

DO File

```
vlib work
vlog -f lists.list
vsim -voptargs=+acc work.MIPS_tb
#add wave *       You, 13 hours ago • Add work & Files
add wave -position insertpoint sim:/MIPS_tb/MIPS/*
run -all
#quit -sim
```

```
Adder.v
ALU.v
ALUOP.v
Control_Unit.v
Data_Memory.v
datapath.v
DFF.v
Instruction_Memory.v
mux2_1.v
Register_File.v
shift2.v
signextend.v
MIPS_tb.sv        You,
```

Important Tables



ALUOp	Meaning
00	add
01	subtract
10	look at `funct` field
11	n/a

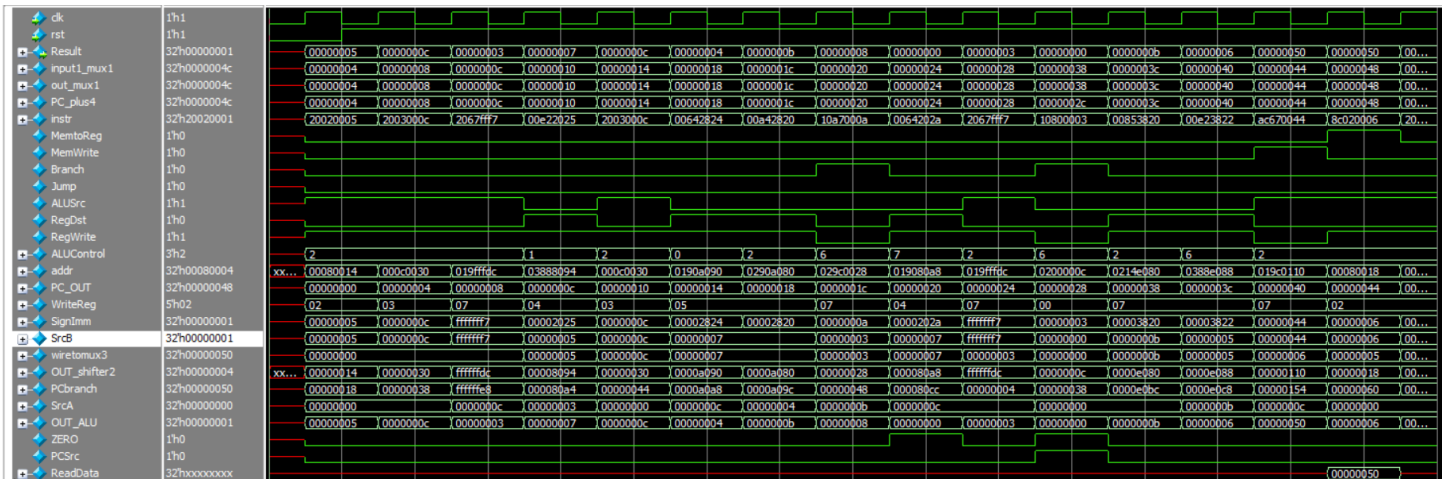
ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (subtract)
1X	100000 (`add`)	010 (add)
1X	100010 (`sub`)	110 (subtract)
1X	100100 (`and`)	000 (and)
1X	100101 (`or`)	001 (or)
1X	101010 (`slt`)	111 (set less than)

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump
R-type	000000	1	1	0	0	0	0	10	0
`lw`	100011	1	0	1	0	0	1	00	0
`sw`	101011	0	X	1	0	1	X	00	0
`beq`	000100	0	X	0	1	0	X	01	0
`addi`	001000	1	0	1	0	0	0	00	0
`j`	000010	0	X	X	X	0	X	XX	1

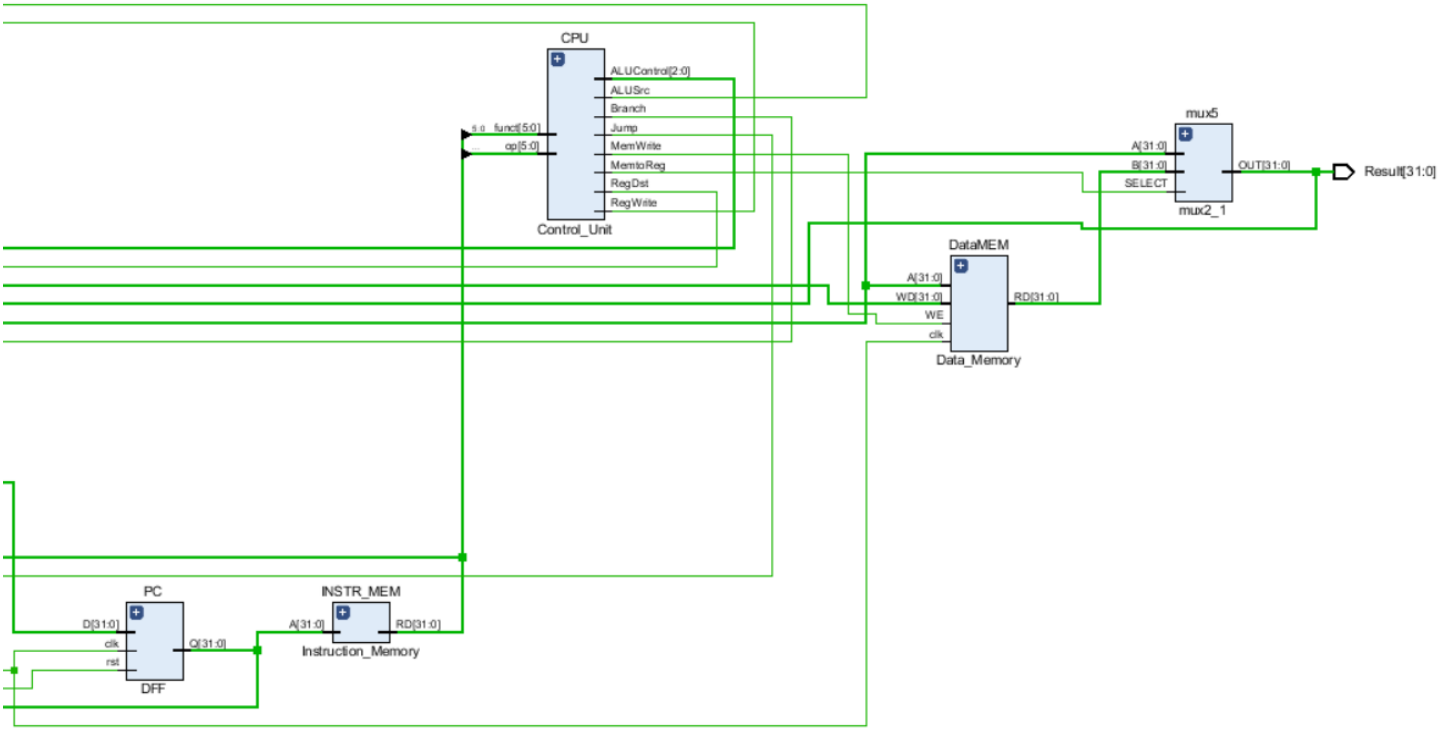
Element	Parameter	Delay (ps)
register clk-to-Q	t_{pcq}	30
register setup	t_{setup}	20
multiplexer	t_{mux}	25
ALU	t_{ALU}	200
memory read	t_{mem}	250
register file read	$t_{RF\text{read}}$	150
register file setup	$t_{RF\text{setup}}$	20

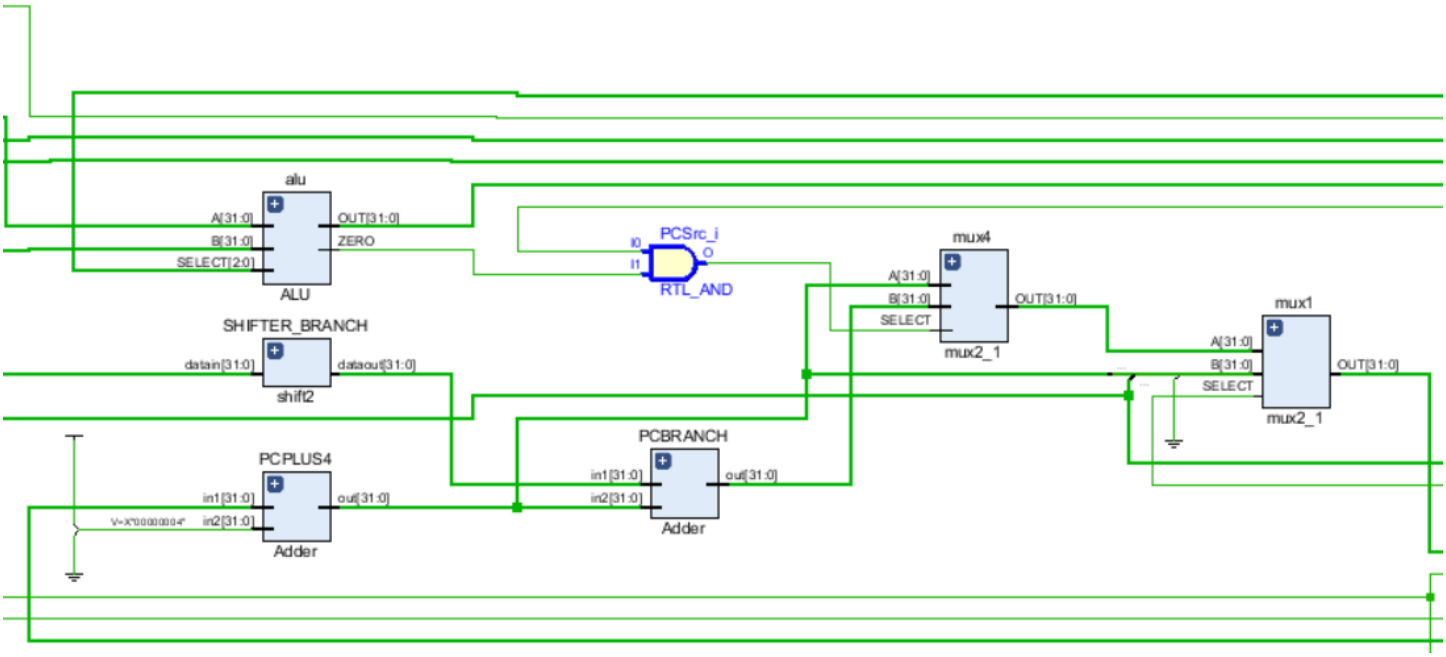
Results

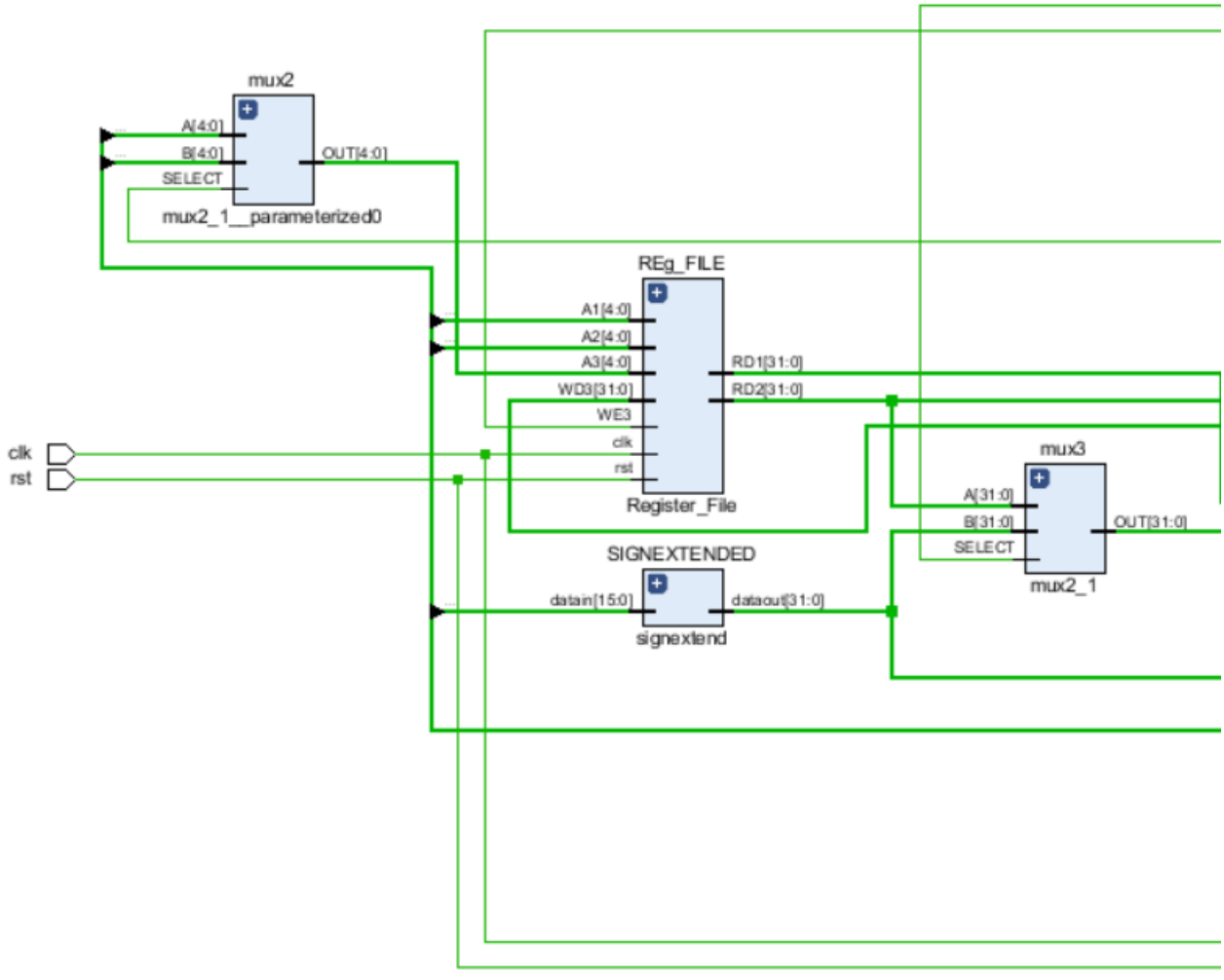
Waveform



Vivado




alu
A[31:0]
B[31:0]
SELECT[2:0]
OUT[31:0]
ZERO
ALU
PCSrc_i
I0
I1
O
RTL_AND
mux4
A[31:0]
B[31:0]
SELECT
OUT[31:0]
mux2_1
mux1
A[31:0]
B[31:0]
SELECT
OUT[31:0]
mux2_1
SHIFTER_BRANCH
datain[31:0]
dataout[31:0]
shift2
PCPLUS4
in1[31:0]
in2[31:0]
out[31:0]
Adder
PCBRANCH
in1[31:0]
in2[31:0]
out[31:0]
Adder


mux2
A[4:0]
B[4:0]
SELECT
OUT[4:0]
mux2_1__parameterized0
REg_FILE
A1[4:0]
A2[4:0]
A3[4:0]
WD3[31:0]
WE3
clk
rst
RD1[31:0]
RD2[31:0]
Register_File
clk
rst
mux3
A[31:0]
B[31:0]
SELECT
OUT[31:0]
mux2_1
SIGNEXTENDED
datain[15:0]
dataout[31:0]
signextend

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.182 ns	Worst Hold Slack (WHS):	0.720 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	520	Total Number of Endpoints:	520	Total Number of Endpoints:	265

Name 1	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
N datapath	358	299	48	154	358	25	34	2
alu (ALU)	8	32	0	25	8	0	0	0
CPU (Control_Unit)	67	3	0	32	67	1	0	0
PC (DFF)	182	8	16	67	182	7	0	0
PCBRANCH (Adder)	0	0	0	2	0	0	0	0
PCPLUS4 (Adder_0)	0	0	0	2	0	0	0	0
REg_FILE (Register_F...	145	256	32	117	145	0	0	0

```
## Clock signal
#set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

References

Harris, D. M. (2007). *Digital Design and Computer Architecture.*

https://cs.stackexchange.com/questions/90485/mips-cpu-single-cycle-mips-processor-r-type-instruction-aluop-code-confusion