CPU Module:

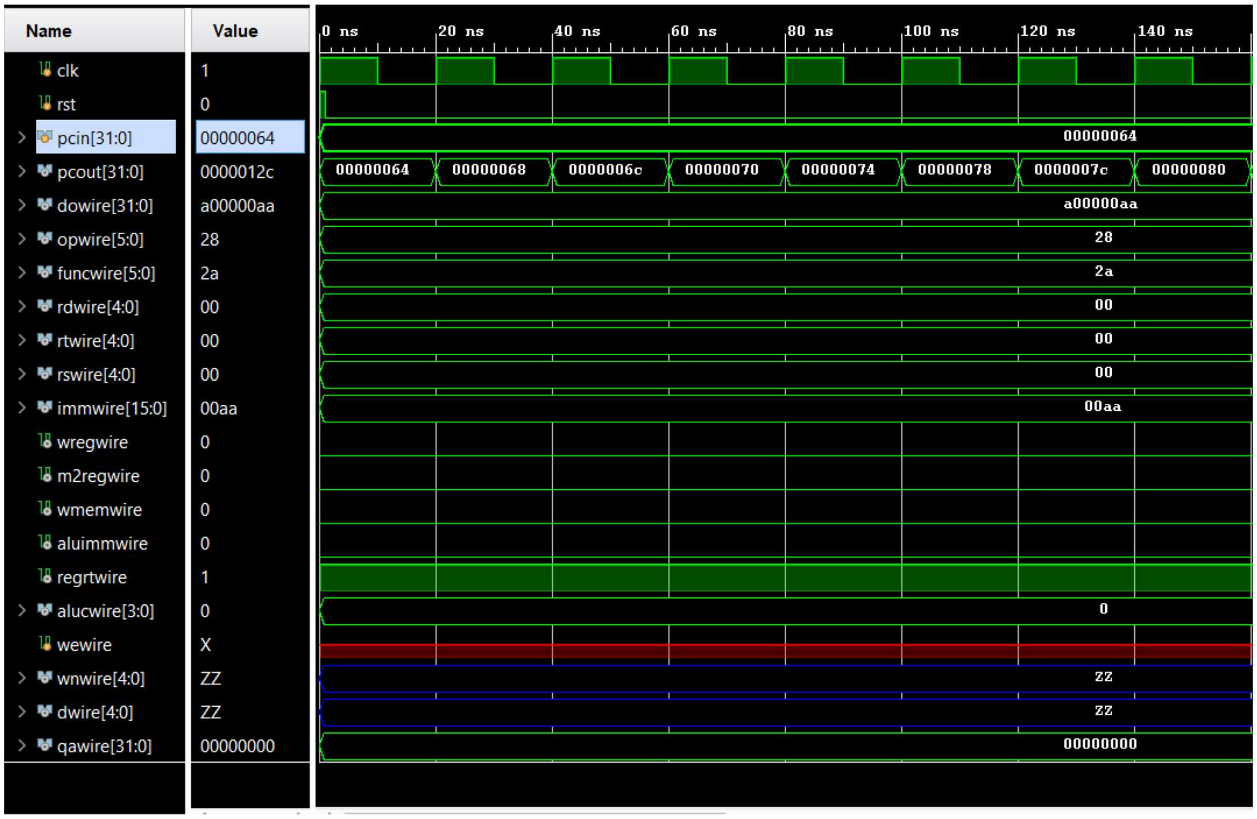
```
`timescale 1ns / 1ps
module PC(
    input clk,rst,
    input [31:0] PCAddress,
    output reg[31:0] oldPC
    );
    reg [31:0] newPC;
    always@(posedge clk) begin
        if(rst) begin
            newPC = PCAddress;
            oldPC = newPC;
        end
        else begin
            newPC = newPC + 4;//PC Adder
            oldPC = newPC;
        end
    end
endmodule

module InstructMem(
    input [31:0] if_idin,
    output [31:0] if_idout
    );
    reg [31:0] instr_mem[0:127];
    initial begin
        //initialize the first 10 words of the Data memory with the following HEX values.
        instr_mem[100]=32'hA00000AA;
        instr_mem[104]=32'h10000011;
        instr_mem[108]=32'h20000022;

        instr_mem[112]=32'h30000033;

        instr_mem[116]=32'h40000044;

        instr_mem[120]=32'h50000055;

        instr_mem[124]=32'h60000066;

        instr_mem[128]=32'h70000077;

        instr_mem[132]=32'h80000088;

        instr_mem[136]=32'h90000099;

    end

    assign if_idout = instr_mem[if_idin];

endmodule


module IFID(

    input clk,

    input [31:0] do, //output from Instruction memory

    //setup R Type

    output reg [5:0] op, func, //output into Control Unit

    output reg [4:0] rd, rs, rt, // (rd,rt) into mux. (rs,rt) into regfile

    output reg [15:0] imm //immediate value going into extender

    );

    always@(posedge clk) begin

        op = do[31:26]; // R-Type opcode

        func = do[5:0]; // R-Type func

        rs = do[25:21]; // R-Type first source

        rt = do[20:16]; // R-Type target source

        rd = do[15:11]; // R-Type destination register

        imm = do[15:0]; // Address/Immediate Value

    end

endmodule
```

```
module ControlUnit(
    input [5:0] op,
    input [5:0] func,
    output reg wreg,
    output reg m2reg,
    output reg wmem,
    output reg aluimm,
    output reg regrt,
    output reg [3:0] aluc
    );
    always@(op,func) begin
        case(op)
            6'b100011: begin
                aluimm = 1;
                aluc = 4'b0010;
                wreg = 1;
                m2reg = 1;
                wmem = 0;
                regrt = 1;
            end
            default: begin
                aluc = 4'b0000;
                aluimm = 0;
                wreg = 0;
                m2reg = 0;
                wmem = 0;
                regrt = 1;
            end
        endcase
    end
endmodule

module ControlMux(
    input regrt,
    input [4:0] rd, rt,
    output [4:0] wn
    );
    assign wn = regrt?rt:rd; //Multiplexer operation
endmodule

module RegFile(
    input we,
    input [4:0] rna,
    input [4:0] rnb,
    input [4:0] wn,
    input [4:0] d,
    output reg [31:0] qa,
    output reg [31:0] qb
    );
    //initialize all registers to 0
    reg [31:0] registers [0:127];
    initial begin
        registers[0] = 0;
        registers[1] = 0;
        registers[2] = 0;
        registers[3] = 0;
        registers[4] = 0;
        registers[5] = 0;
        registers[6] = 0;
        registers[7] = 0;
        registers[8] = 0;
        registers[9] = 0;
        registers[10] = 0;
        registers[11] = 0;
        registers[12] = 0;
        registers[13] = 0;
        registers[14] = 0;
        registers[15] = 0;
    end
    always@(rna, rnb) begin
        qa = registers[rna];
        qb = registers[rnb];
    end
endmodule

module SignExtend(//input immediate value, extend to 32 bit value
    input [15:0] imm,
    output reg [31:0] long //extended value
    );
    always@(imm) begin
        if(imm[15] == 1) begin
            long[31:16] = 16'hffff;
            long[15:0] = imm;
        end
        else begin
            long[31:16] = 16'h0000;
            long[15:0] = imm;
        end
    end
endmodule

module IDEXE(
    input clk, wreg, m2reg, wmem, aluimm, //input to IDEXE
    input [3:0] aluc, //output from control unit
    input [31:0] qa, qb, //output from regfile
    input [4:0] mux, //output from mux into IDEXE
    input [31:0] extend, //output from signextender
    output reg ewreg, em2reg, ewmem, ealuimm, //outputs from control unit
    output reg [3:0] ealuc, //out from control unit, into ALU
    output reg [31:0] eqa, eqb, //outputs from regfile
    output reg [4:0] emux, //output from multiplexor
    output reg [31:0] eextend //output from sign extender
    );
    always@(posedge clk) begin
        ewreg = wreg;
        em2reg = m2reg;
        ewmem = wmem;
        ealuimm = aluimm;
        emux = mux;
        ealuc = aluc;
        eqa = qa;
        eqb = qb;
        eextend = extend;
    end
endmodule
```

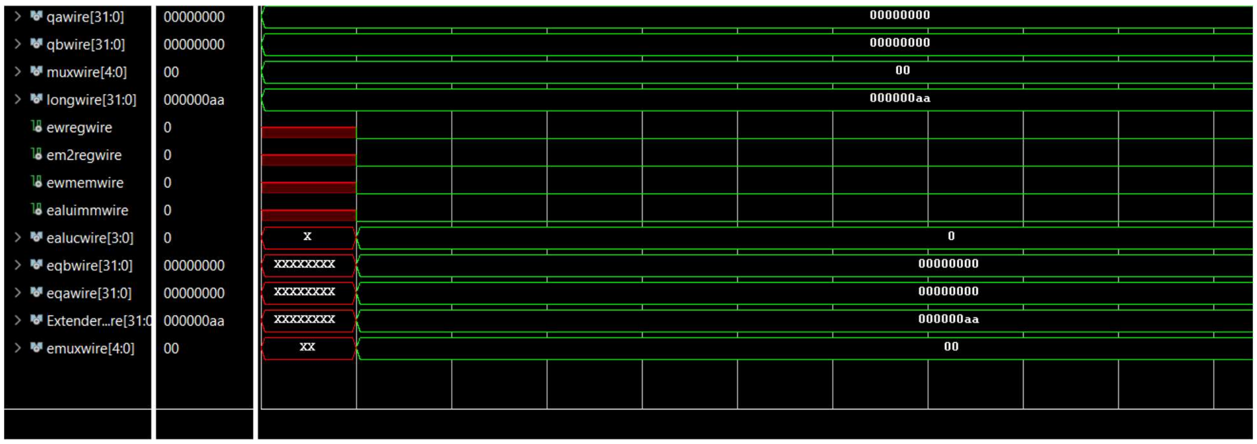
Testbench Module:

```
`timescale 1ns / 1ps

module CPU_Test();

   reg clk, rst;

   reg [31:0] pcin;

   wire [31:0] pcout;

   wire [31:0] dowire;

   wire [5:0] opwire;

   wire [5:0] funcwire;

   wire [4:0] rdwire, rtwire, rswire;

   wire [15:0] immwire;

   wire wregwire, m2regwire, wmemwire, aluimmwire;

   wire regrtwire;

   wire [3:0] alucwire;

   reg wewire;

   wire [4:0] wnwire;

   wire [4:0] dwire;

   wire [31:0] qawire, qbwire;

   wire [4:0] muxwire;

   wire [31:0] longwire;

   wire ewregwire, em2regwire, ewmemwire, ealuimmwire;

   wire [3:0] ealucwire;

   wire [31:0] eqbwire, eqawire, Extenderwire;

   wire [4:0] emuxwire;


PC pc(clk, rst, pcin, pcout);

InstructMem instmem(pcin, dowire);

IFID ifid(clk, dowire, opwire, funcwire, rdwire, rswire, rtwire, immwire);

ControlUnit ctrunit(opwire, funcwire, wregwire, m2regwire, wmemwire,
                        aluimmwire, regrtwire, alucwire);
RegFile regfile(wewire, rswire, rtwire, wnwire, dwire, qawire, qbwire);
ControlMux ctrmux(regrtwire, rdwire, rtwire,muxwire);
SignExtend extender(immwire, longwire);
IDEXE idexe(clk, wregwire, m2regwire, wmemwire, aluimmwire, alucwire,
                qawire, qbwire, muxwire, longwire, ewregwire, em2regwire,
                ewmemwire, ealuimmwire, ealucwire, eqawire, eqbwire,
                emuxwire, Extenderwire);

    initial begin
        clk = 1;
        pcin = 100;
        rst = 1;
        #1 rst = 0;
    end

    always begin
        #10 clk = !clk;
    end
endmodule
```

Waveforms:



Name	Value
clk	1
rst	0
pcin[31:0]	00000064
pcout[31:0]	0000012c
dowire[31:0]	a00000aa
opwire[5:0]	28
funcwire[5:0]	2a
rdwire[4:0]	00
rtwire[4:0]	00
rswire[4:0]	00
immwire[15:0]	00aa
wregwire	0
m2regwire	0
wmemwire	0
aluimmwire	0
regrtwire	1
alucwire[3:0]	0
wewire	X
wnwire[4:0]	ZZ
dwire[4:0]	ZZ
qawire[31:0]	00000000



Name	Value
qawire[31:0]	00000000
qbwire[31:0]	00000000
muxwire[4:0]	00
longwire[31:0]	000000aa
ewregwire	0
em2regwire	0
ewmemwire	0
ealuimmwire	0
ealucwire[3:0]	0
eqbwire[31:0]	00000000
eqawire[31:0]	00000000
Extender...re[31:0	000000aa
emuxwire[4:0]	00