Design and implement Single Cycle Implementation with following specifications.

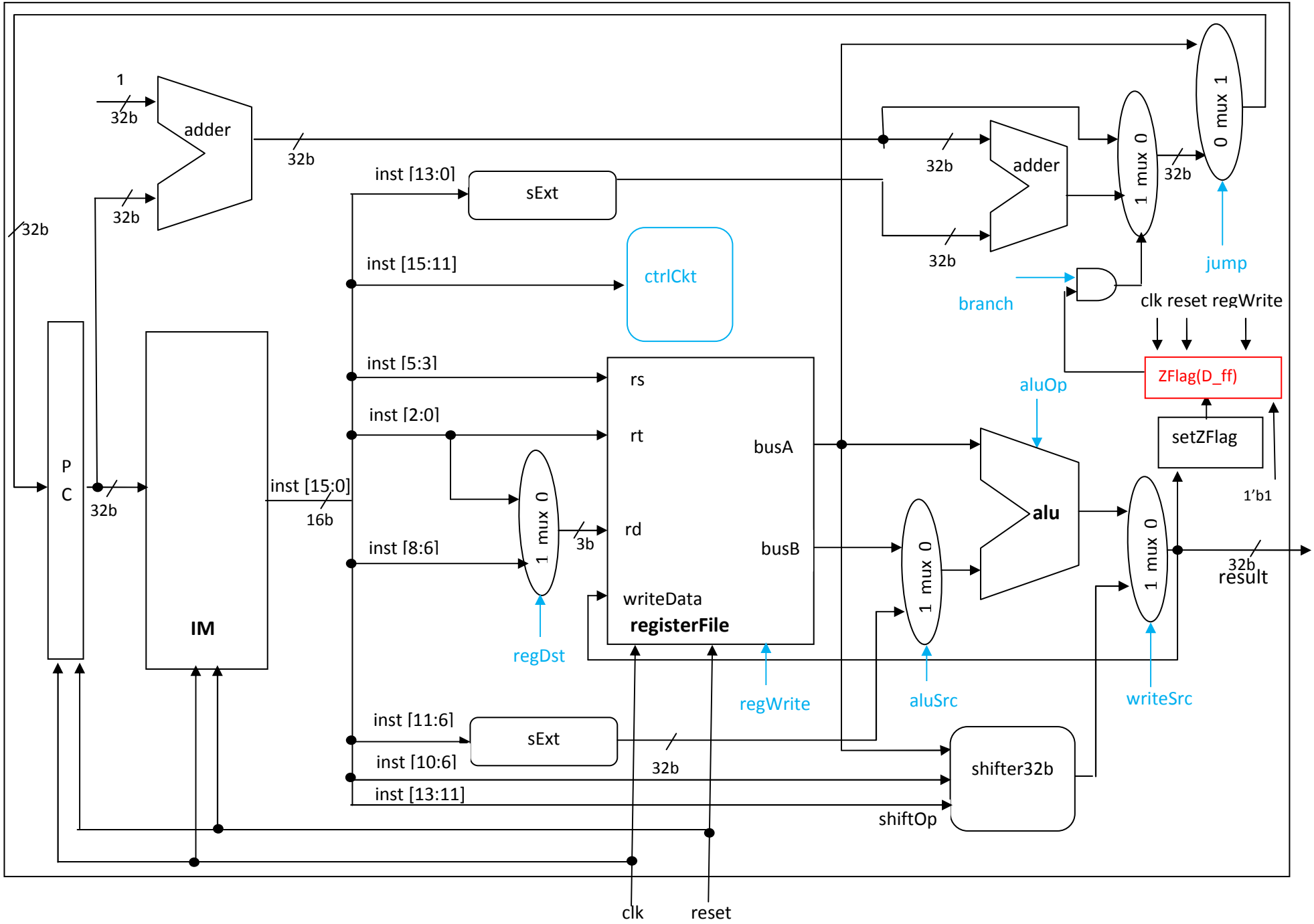
Instruction Format

Instruction			Instruction Format															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shift	Arithmetic	Right	0	0	0	0	1	Offset (5bits)					Rs			Rd		
Shift	Logical	Right	0	0	0	1	0	Offset (5bits)					Rs			Rd		
Shift	Circular	Right	0	0	0	1	1	Offset (5bits)					Rs			Rd		
Shift	Logical	Left	0	0	1	0	0	Offset (5bits)					Rs			Rd		
Shift	Circular	Left	0	0	1	0	1	Offset (5bits)					Rs			Rd		
Add/Sub	Add	Reg	0	1	0	0	X	X	X	Rd			Rs			Rn		
Add/Sub	Add	Imm	0	1	0	1	Offset (6bits)						Rs			Rd		
Add/Sub	Sub	Reg	0	1	1	0	X	X	X	Rd			Rs			Rn		
Add/Sub	Sub	Imm	0	1	1	1	Offset (6bits)						Rs			Rd		
BZ			1	0	Offset (14bits)													
JR			1	1	X	X	X	X	X	X	X	X	Rs			X	X	X

Control circuit

Instruction	Opcode[15:14]	Opcode[13:11]	regDst	regWrite	aluSrc	aluOp	writeSrc	branch	jump
Shift Arithmetic Right	00	001	0	1	1	0	1	0	0
Shift Logical Right	00	010	0	1	1	0	1	0	0
Shift Circular Right	00	011	0	1	1	0	1	0	0
Shift Logical Left	00	100	0	1	1	0	1	0	0
Shift Circular Left	00	101	0	1	1	0	1	0	0
Add Register	01	00x	1	1	0	0	0	0	0
Add Immediate	01	01x	0	1	1	0	0	0	0
Subtract Register	01	10x	1	1	0	1	0	0	0
Subtract Immediate	01	11x	0	1	1	1	0	0	0
BZ	10	xxx	0	0	1	0	0	1	0
JR	11	xxx	0	0	1	0	0	0	1

module ctrlCkt(input [4:0] opcode,output reg regDst, output reg regWrite, output reg aluSrc, output reg aluOp, output reg writeSrc, output reg branch, output reg jump);


1
32b
adder
32b
32b
32b
inst [13:0]
sExt
32b
adder
32b
1 mux 0
32b
0 mux 1
jump
branch
clk reset regWrite
ZFlag(D_ff)
setZFlag
1'b1
inst [15:11]
ctrlCkt
inst [5:3]
inst [2:0]
rs
rt
busA
aluOp
alu
P
C
32b
IM
inst [15:0]
16b
1 mux 0
3b
rd
busB
inst [8:6]
writeData
registerFile
regDst
regWrite
1 mux 0
aluSrc
1 mux 0
writeSrc
32b
result
inst [11:6]
sExt
32b
inst [10:6]
inst [13:11]
shifter32b
shiftOp
clk
reset

Modules

Instruction Memory (IM)

```
moduleD_ff_IM(input clk, input reset, input d, output reg q);
        always@(reset or posedgeclk)
        if(reset)q=d;
endmodule

moduleregister_IM(input clk, input reset, input [15:0] d_in,output [15:0] q_out);
        D_ff_IM dIM0 (clk, reset, d_in[0], q_out[0]);
        D_ff_IM dIM1 (clk, reset, d_in[1], q_out[1]);
        D_ff_IM dIM2 (clk, reset, d_in[2], q_out[2]);
        D_ff_IM dIM3 (clk, reset, d_in[3], q_out[3]);
        D_ff_IM dIM4 (clk, reset, d_in[4], q_out[4]);
        D_ff_IM dIM5 (clk, reset, d_in[5], q_out[5]);
        D_ff_IM dIM6 (clk, reset, d_in[6], q_out[6]);
        D_ff_IM dIM7 (clk, reset, d_in[7], q_out[7]);
        D_ff_IM dIM8 (clk, reset, d_in[8], q_out[8]);
        D_ff_IM dIM9 (clk, reset, d_in[9], q_out[9]);
        D_ff_IM dIM10 (clk, reset, d_in[10], q_out[10]);
        D_ff_IM dIM11 (clk, reset, d_in[11], q_out[11]);
        D_ff_IM dIM12 (clk, reset, d_in[12], q_out[12]);
        D_ff_IM dIM13 (clk, reset, d_in[13], q_out[13]);
        D_ff_IM dIM14 (clk, reset, d_in[14], q_out[14]);
        D_ff_IM dIM15 (clk, reset, d_in[15], q_out[15]);
endmodule

module mux32to1( input [15:0] outR0,outR1,outR2,outR3,outR4,outR5,outR6,outR7,outR8,outR9,outR10,outR11,outR12,outR13,
outR14,outR15,outR16,outR17,outR18,outR19,outR20,outR21,outR22,outR23,outR24,outR25,outR26,outR27,outR28,outR29,
outR30,outR31,input [4:0] Sel,output reg [15:0] outBus );
        always@(outR0 or outR1 or outR2 or outR3 or outR4 or outR5 or outR6 or outR7 or outR8 or outR9 or outR10 or outR11 or outR12
or outR13 or outR14 or outR15 or outR16 or outR17 or outR18 or outR19 or outR20 or outR21 or outR22 or outR23 or outR24 or outR25 or
outR26 or outR27 or outR28 or outR29 or outR30 or outR31 or Sel)
        case (Sel)
                5'b00000: outBus=outR0;     5'b00001: outBus=outR1;     5'b00010: outBus=outR2;     5'b00011: outBus=outR3;
                5'b00100: outBus=outR4;     5'b00101: outBus=outR5;     5'b00110: outBus=outR6;     5'b00111: outBus=outR7;
                5'b01000: outBus=outR8;     5'b01001: outBus=outR9;     5'b01010: outBus=outR10;    5'b01011: outBus=outR11;
                5'b01100: outBus=outR12;    5'b01101: outBus=outR13;    5'b01110: outBus=outR14;    5'b01111: outBus=outR15;
                5'b10000: outBus=outR16;    5'b10001: outBus=outR17;    5'b10010: outBus=outR18;    5'b10011: outBus=outR19;
                5'b10100: outBus=outR20;    5'b10101: outBus=outR21;    5'b10110: outBus=outR22;    5'b10111: outBus=outR23;
                5'b11000: outBus=outR24;    5'b11001: outBus=outR25;    5'b11010: outBus=outR26;    5'b11011: outBus=outR27;
                5'b11100: outBus=outR28;    5'b11101: outBus=outR29;    5'b11110: outBus=outR30;    5'b11111: outBus=outR31;
        endcase
endmodule
```

```
module IM(  input clk, input reset, input [31:0] pc, output [15:0] instr );

    wire [15:0] Qout0, Qout1, Qout2, Qout3, Qout4, Qout5, Qout6, Qout7, Qout8, Qout9, Qout10, Qout11, Qout12, Qout13, Qout14, Qout15,
    Qout16, Qout17, Qout18, Qout19, Qout20, Qout21, Qout22, Qout23, Qout24, Qout25, Qout26, Qout27, Qout28, Qout29, Qout30, Qout31;

    register_IM rIM0 (clk, reset, 16'b01_01_010000_000_001, Qout0);       //addi $r1, $r0, 16
    register_IM rIM1 (clk, reset, 16'b01_01_000110_000_111, Qout1);       //addi $r7, $r0, 6
    register_IM rIM2 (clk, reset, 16'b00_011_00101_001_001, Qout2);       //csr $r1, $r1, 5
    register_IM rIM3 (clk, reset, 16'b00_010_11111_001_010, Qout3);       //lsr $r2, $r1, 31
    register_IM rIM4 (clk, reset, 16'b00_001_00001_001_001, Qout4);       //asr $r1, $r1, 1
    register_IM rIM5 (clk, reset, 16'b00_101_00010_001_001, Qout5);       //csl $r1, $r1, 2
    register_IM rIM6 (clk, reset, 16'b00_100_00001_010_010, Qout6);       //lsl $r2, $r2, 1
    register_IM rIM7 (clk, reset, 16'b01_00_000_011_011_010, Qout7);      //add $r3, $r3, $r2
    register_IM rIM8 (clk, reset, 16'b01_11_000001_001_001, Qout8);       //subi $r1, $r1, 1
    register_IM rIM9 (clk, reset, 16'b10_0000_0000_0000_01, Qout9);       //bz 1
    register_IM rIM10 (clk, reset, 16'b11_0000_0000_111_000, Qout10);     //jr $r7
    register_IM rIM11 (clk, reset, 16'b01_00_000_101_011_010, Qout11);    //add $r5, $r3, $r2
    register_IM rIM12 (clk, reset, 16'b01_10_000_100_011_010, Qout12);    //sub $r4, $r3, $r2
    register_IM rIM13 (clk, reset, 16'b01_11_000010_101_101, Qout13);     //subi $r5, $r5, 2
    register_IM rIM14 (clk, reset, 16'b00_010_00001_101_110, Qout14);     //lsr $r6, $r5, 1
    register_IM rIM15 (clk, reset, 16'b01_00_000_100_100_110, Qout15);    //add $r4, $r4, $r6
    register_IM rIM16 (clk, reset, 16'b00_011_00001_010_001, Qout16);     //csr $r1, $r2, 1
    register_IM rIM17 (clk, reset, 16'b01_01_111110_011_011, Qout17);     //add $r3, $r3, -2
    register_IM rIM18 (clk, reset, 16'b00_100_00001_011_110, Qout18);     //lsl $r0, $r3, 1
    register_IM rIM19 (clk, reset, 16'b01_00_000_111_110_001, Qout19);    //add $r7, $r6, $r1
    register_IM rIM20 (clk, reset, 16'b01_00_000_000_000_000, Qout20);    //add $r0, $r0, $r0
    register_IM rIM21 (clk, reset, 16'b0000000000000000, Qout21);
    register_IM rIM22 (clk, reset, 16'b0000000000000000, Qout22);
    register_IM rIM23 (clk, reset, 16'b0000000000000000, Qout23);
    register_IM rIM24 (clk, reset, 16'b0000000000000000, Qout24);
    register_IM rIM25 (clk, reset, 16'b0000000000000000, Qout25);
    register_IM rIM26 (clk, reset, 16'b0000000000000000, Qout26);
    register_IM rIM27 (clk, reset, 16'b0000000000000000, Qout27);
    register_IM rIM28 (clk, reset, 16'b0000000000000000, Qout28);
    register_IM rIM29 (clk, reset, 16'b0000000000000000, Qout29);
    register_IM rIM30 (clk, reset, 16'b0000000000000000, Qout30);
    register_IM rIM31 (clk, reset, 16'b0000000000000000, Qout31);
    mux32to1 mIM
(Qout0,Qout1,Qout2,Qout3,Qout4,Qout5,Qout6,Qout7,Qout8,Qout9,Qout10,Qout11,Qout12,Qout13,Qout14,Qout15,Qout16,Qout17,Qout18,Qout19,Qo
ut20,Qout21,Qout22,Qout23,Qout24,Qout25,Qout26,Qout27,Qout28,Qout29,Qout30,Qout31,pc[4:0],instr);

endmodule
```

Register File

```
module D_ff (input clk, input reset, input regWrite, input decOut1b, input d,output reg q);
        always @ (negedgeclk)
        begin
                if(reset==1)    q=0;
                else
                        if(regWrite == 1 && decOut1b==1)  q=d;
        end
endmodule

module register32bit( input clk, input reset, input regWrite, input decOut1b, input [31:0] writeData,output  [31:0] outR );

module registerSet( input clk, input reset, input regWrite, input [7:0] decOut, input [31:0] writeData,
                        output [31:0] outR0,outR1,outR2,outR3,outR4,outR5,outR6,outR7 );

module decoder3to8( input [2:0] destReg,output reg [7:0] decOut);

module mux8to1( input [31:0] outR0,outR1,outR2,outR3,outR4,outR5,outR6,outR7, input [2:0] Sel, output reg [31:0] outBus );

module registerFile(input clk, input reset, input regWrite, input [2:0]srcRegA, input [2:0] srcRegB,  input [2:0] destReg,
                        input [31:0] writeData,output [31:0] outBusA, output [31:0] outBusB );
```

Multiplexers

```
module mux2to1_3bits(input [2:0] in1, input [2:0] in2, input sel,output reg [2:0] muxout);
module mux2to1_32bits(input [31:0] in1, input [31:0] in2, input sel,output reg [31:0] muxout);
```

Sign Extension

```
module signExt6to32( input [5:0] offset,output reg [31:0] signExtOffset);
module signExt14to32( input [13:0] offset,output reg [31:0] signExtOffset);
```

ALU, Dedicated Adder and setZeroFlag

```
module alu(input [31:0] aluIn1, input [31:0] aluIn2, input aluOp,output reg [31:0] aluResult);
module adder(input [31:0] in1, input [31:0] in2,output reg [31:0] adder_out);
module shifter32b(input [31:0] rs, input [4:0] shiftAmt, input [2:0] opcode, output reg [31:0] shiftOut);
module setZflag( input [31:0] Result, output reg zero);
```

Top Module

modulesingleCycle(input clk, input reset,output [31:0] Result);

Testbench

```
moduletestBench;
        regclk;
        reg reset;
        wire [31:0] Result;
        singleCycleuut (.clk(clk), .reset(reset), .Result(Result));
        always
        #5 clk=~clk;
        initial
        begin
                clk=0; reset=1;
                #5  reset=0;
                #300 $finish;
        end
endmodule
```

Simulation Results