```
PROBLEM 1 - Barrel Shifter
// 2:1 MUX
module mux2_1(InA,InB,S,out);
    // define inputs and outputs
    input InA,InB,S;
    output out;
    // wires local to the module
    wire a1,a2,n1;

    // logic starts
    not1 n5(S,n1);
    nand2 n4(InA,n1,a1);
    nand2 n2(InB,S,a2);
    nand2 n3(a1,a2,out);

endmodule
// end of module

// 4:1 MUX using 2:1 MUX
module mux4_1 (out,InA,InB,InC,InD,S);
    // define inputs and outputs
    input InA,InB,InC,InD;
    input [1:0] S;
    output out;
    // wires local to the module
    wire s1,s2;

    // logic starts
    mux2_1 m1 (.InA(InA),.InB(InB),.S(S[0]),.out(s1));
    mux2_1 m2 (.InA(InC),.InB(InD),.S(S[0]),.out(s2));
    mux2_1 m3 (.InA(s1),.InB(s2),.S(S[1]),.out(out));

endmodule
// end of module

// This module is an array of 16 2:1 MUXes used in shift logic
module stage (a,b,sel,out);
    // define inputs and outputs
    input [15:0]a,b;
    input sel;
    output [15:0]out;

    // logic starts
    mux2_1 k1 [15:0] (.InA(a), .InB(b), .S(sel), .out(out));

endmodule
// end of module

// Code for 16-bit barrel shifter
module shift_16 (In, Cnt, Op, Out);
   // define inputs and outputs
   input [15:0] In;
   input [3:0]  Cnt;
   input [1:0]  Op;
   output [15:0] Out;
```

```
    // wires local to the module
    wire [15:0] S0,S1,S2;
    wire [15:0] L0,L1,L2,L3;

    // level 1 shift
    mux2_1 M1 [13:0](.InA(In[13:0]), .InB(In[15:2]), .out(L0[14:1]), .S(Op[1]));
    mux4_1 n5       (.out(L0[0]), .InA(In[15]), .InB(1'b0), .InC(In[1]), .InD(In[1]), .S(Op));
    mux4_1 n6       (.out(L0[15]), .InA(In[14]), .InB(In[14]), .InC(In[0]), .InD(In[15]), .S(Op));
    stage s1        (.a(In), .b(L0), .sel(Cnt[0]), .out(S0));

    //level 2 shift
    mux2_1 M2 [11:0](.InA(S0[11:0]), .InB(S0[15:4]), .out(L1[13:2]), .S(Op[1]));
    mux4_1 N2 [3:0] (.out({L1[15],L1[14],L1[1],L1[0]}), .InA({S0[13:12],S0[15:14]}), .InB({S0[13:
    12],2'b00}), .InC({S0[1:0],S0[3:2]}), .InD({S0[15],S0[15],S0[3:2]}), .S(Op));
    stage s2        (.a(S0), .b(L1), .sel(Cnt[1]), .out(S1));

    //level 4 shift
    mux2_1 M3 [7:0] (.InA(S1[7:0]), .InB(S1[15:8]), .out(L2[11:4]), .S(Op[1]));
    mux4_1 N3 [7:0] (.out({L2[15:12],L2[3:0]}), .InA({S1[11:8],S1[15:12]}), .InB({S1[11:8],
    4'b0000}), .InC({S1[3:0],S1[7:4]}), .InD({{{4{S1[15]}},S1[7:4]}), .S(Op));
    stage s3        (.a(S1), .b(L2), .sel(Cnt[2]), .out(S2));

    //level 8 shift
    mux4_1 N4 [15:0](.out(L3), .InA({S2[7:0],S2[15:8]}), .InB({S2[7:0],8'd0}), .InC({S2[7:0],S2[
    15:8]}), .InD({{8{S2[15]}},S2[15:8]}), .S(Op));
    stage s4        (.a(S2), .b(L3), .sel(Cnt[3]), .out(Out));

endmodule
// end of module
```

PROBLEM 2 - ALU Design

```
// CLA logic
module cla (g,p,c,C0);
    // define inputs and outputs
    input [3:0]g,p;
    input C0;
    output [4:1]c;

    // logic starts
    assign c[1] = (g[0] | (p[0]&C0));
    assign c[2] = (g[1] | (p[1]&g[0]) | (p[1]&p[0]&C0));
    assign c[3] = (g[2] | (p[2]&g[1]) | (p[2]&p[1]&g[0]) | (p[2]&p[1]&p[0]&C0));
    assign c[4] = (g[3] | (p[3]&g[2]) | (p[3]&p[2]&g[1]) | (p[3]&p[2]&p[1]&g[0]) | (p[3]&p[2]&p[
    1]&p[0]&C0));

endmodule
// end of module

// CLA4 logic
module cla4 (a,b,C0,S,gg,pg,cout);
    // define inputs and outputs
    input [3:0] a,b;
    input C0;
    output cout;
    output [3:0] S;
    output gg,pg;
    // wires that are local to the module
    wire [3:1]C;
    wire [3:0]g,p;

    // Generate and Propogate logic
    assign g = a&b;
    assign p = a^b;

    // Generating Carry for other adders
    assign C[1] = (g[0] | (p[0]&C0));
    assign C[2] = (g[1] | (p[1]&g[0]) | (p[1]&p[0]&C0));
    assign C[3] = (g[2] | (p[2]&g[1]) | (p[2]&p[1]&g[0]) | (p[2]&p[1]&p[0]&C0));
    assign cout = (g[3] | (p[3]&g[2]) | (p[3]&p[2]&g[1]) | (p[3]&p[2]&p[1]&g[0]) | (p[3]&p[2]&p[
    1]&p[0]&C0));

    // Calculate the Sum
    assign S=p^{C[3:1], C0};

    // Group generate and group propogate logic
    assign pg = &p;
    assign gg = g[3] | (p[3] & g[2]) | (p[3]&p[2]&g[1]) | (p[3]&p[2]&p[1]&g[0]);

endmodule
//end of module

// CLA16 from CLA4
module cla16 (A,B,Cin,Out,Cout);
    // define inputs and outputs
    input [15:0]A,B;
    input Cin;
```

```
    output Cout;
    output [15:0]Out;
    // wires local to the module
    wire [3:0]gg,pg;
    wire [3:1]carry;

    // Instantiating CLA4 here
    cla4 k4 (.a(A[3:0]),.b(B[3:0]),.C0(Cin),.S(Out[3:0]),.gg(gg[0]),.pg(pg[0]),.cout(carry[1]));

    cla4 k3 (.a(A[7:4]),.b(B[7:4]),.C0(carry[1]),.S(Out[7:4]),.gg(gg[1]),.pg(pg[1]),.cout(carry[
    2]));
    cla4 k2 (.a(A[11:8]),.b(B[11:8]),.C0(carry[2]),.S(Out[11:8]),.gg(gg[2]),.pg(pg[2]),.cout(
    carry[3]));
    cla4 k1 (.a(A[15:12]),.b(B[15:12]),.C0(carry[3]),.S(Out[15:12]),.gg(gg[3]),.pg(pg[3]),.cout(
    Cout));

    CLA locic to calculate final carry
    cla h1 (.g(gg),.p(pg),.c({Cout,carry[3:1]}),.C0(Cin));

endmodule
// end of module

// ALU module
module ALU (A, B, Cin, Op, invA, invB, sign, Out, OFL, Zero);
    // define inputs and outputs
    input [15:0] A;
    input [15:0] B;
    input   Cin;
    input [2:0] Op;
    input   invA;
    input   invB;
    input   sign;
    output [15:0] Out;
    output   OFL;
    output   Zero;
    // wires local to the module
    wire cout;
    wire [15:0] c1;
    reg OFL, Zero;
    reg C;
    reg [1:0]Opcode;
    wire [15:0]a1,b1;
    wire [15:0]out;
    reg [15:0]Out;
    wire [15:0]O;

    assign a1 = invA ? (~A) : A;
    assign b1 = invB ? (~B) : B;

    // Instantiate CLA here
    cla16 C1 (.A(a1),.B(b1),.Cin(C),.Out(out),.Cout(cout));
    // Instantiate shifter from problem 1 here
    shift_16 S1(.In(a1),.Cnt(b1[3:0]),.Op(Opcode), .Out(O));


    always@(*)begin
```

```
        C= Cin|invA|invB;
        case(Op)
            // Rotate Left
            3'd0: begin Opcode = 2'b00; Out=O; OFL = 1'b0; end
            // Shift left logical
            3'd1: begin Opcode = 2'b01; Out=O; OFL = 1'b0; end
            // Rotate Right
            3'd2: begin Opcode = 2'b10; Out=O; OFL = 1'b0; end
            // Shift Right Arithmetic
            3'd3: begin Opcode = 2'b11; Out=O; OFL = 1'b0; end
            // Addition
            3'd4: begin
                    Out = out;
                    // Logic for OFL
                    OFL = ((cout==1'b1)&(sign==1'b0))|((sign==1'b1)&(cout^(C1.k1.C[3])));
                  end
            // OR
            3'd5: begin Out = a1 | b1; OFL = 1'b0; end
            // XOR
            3'd6: begin Out = a1 ^ b1; OFL = 1'b0; end
            // AND
            3'd7: begin Out = a1 & b1; OFL = 1'b0; end
        endcase
        // Logic for Zero detection
        Zero = !(|Out);
    end

endmodule
// end of module
```

PROBLEM 3

For this problem, it is sufficient if you provided instructions (3), (4) and (5). The other instructions are just for illustrative purposes

```
(1)   load   $t1, addrA
(2)   load   $t2, addrB
(3)   add    $t3, $t1, $t2
(4)   sub    $t4, $t3, $t2
(5)   add    $t1, $t3, $t4
(6)   sw     $t3, addrC
(7)   sw     $t4, addrD
(8)   sw     $t1, addrA
```

PROBLEM 4

Many possible solutions exist for this problem. All correct solutions have been given credit

```
clear   $t0
addi    $s1, $zero, 10

loop:   lw      $t1, 0($b0)      # $t1 = b[i]
        sub     $t1, $t1, $s0    # $t1 = b[i] – c
        sw      $t1, 0($a0)      # store $t1 to address a[i]
        addi    $a0, $a0, 4      # $a0 = address of a[i+1]
        addi    $b0, $b0, 4      # $b0 = address of b[i+1]
        addi    $t0, $t0, 1      # $t0 = $t0 + 1
        beq     $t0, $s1, finish # $if ($t0 == 10) finish
        j       loop

finish:
```

(a) Total number of instructions = (2*1) + (7 * 11) + (1 *10) = **89**

(b) Memory data references = (2 * 11) = **22**

PROBLEM 5

(a)

Total number of loads	=	40
Total number of stores	=	40
Total number of branches	=	40
Total number of arithmetic	=	50
Total number of shifts	=	14
Total number of multiplies	=	16

CPI = [(40 * 2) + (40 * 1) + (40 * 4) + (50 * 1) + (14 * 1) + (16 * 10)] / 200 = 504 / 200

CPI = 2.52

(b)

- Old number of multiplies (8% of 200) = 16

Now 50% of 16 multiplies are replaced by shift-add sequences

- New number of multiplies (16 * 50%) = 8
- Additional number of ALU instructions (16 * 50% * 3.5) = 28
- Total number of instructions = 200 – 16 + 8 + 28 = 220

New CPI = [(40 * 2) + (40 * 1) + (40 * 4) + (50 *1) + (14 * 1) + (8 * 10) + (28 * 1)] / 220 = 452 / 220

CPI = 2.05

PROBLEM 6

(a) DATASET A

P1

An ideal instruction sequence is one that contains instructions entirely from Class A with CPI 1
IPC = 1
Peak performance = IPC * Clock rate = **4000 MIPS (Million Instructions Per Second)**

P2

An ideal instruction sequence is one that contains instructions entirely from Class A or class B or class C with CPI 3
IPC = 1/3
Peak performance = IPC * Clock rate = **2000 MIPS (Million Instructions Per Second)**

DATASET B

P1

An ideal instruction sequence is one that contains instructions entirely from Class A with CPI 1
IPC = 1
Peak performance = IPC * Clock rate = **4000 MIPS (Million Instructions Per Second)**

P2

An ideal instruction sequence is one that contains instructions entirely from Class A or class B or class C or class D with CPI 2
IPC = 1/2
Peak performance = IPC * Clock rate = **3000 MIPS (Million Instructions Per Second)**

(b) DATASET A

Average CPI of P1 = (2A + B + C + D + E) / (2+1+1+1+1)
= [(2 * 1) + 2 + 3 + 4 + 5] / 6
= **2.667**

Average CPI of P2 = (2A + B + C + D + E) / (2+1+1+1+1)
= [(2 * 3) + 3 + 3 + 5 + 5] / 6
= **3.667**

Average time to execute 1 instruction in P1 = (16 / 6) / 4 GHz
Average time to execute 1 instruction in P2 = (22 / 6) / 6 GHz

Speed of P2 / Speed of P1
= (Average time per instruction on P1) / (Average time per instruction on P2)

= **1.09**

So, P2 is 1.09 times faster than P1

DATASET B
Average CPI of P1 = (2A + B + C + D + E) / (2+1+1+1+1)
= [(2 * 1) + 2 + 3 + 4 + 5] / 6
= **2.667**

Average CPI of P2 = (2A + B + C + D + E) / (2+1+1+1+1)
= [(2 * 2) + 2 + 2 + 2 + 6] / 6
= **2.667**

Average time to execute 1 instruction in P1 = (16 / 6) / 4 GHz
Average time to execute 1 instruction in P2 = (16 / 6) / 6 GHz

Speed of P2 / Speed of P1
= (Average time per instruction on P1) / (Average time per instruction on P2)
= **1.50**

So, P2 is 1.50 times faster than P1

(c) DATASET A
We need to find the frequency of P1 at which the speedup ratio is 1. Assume frequency is F GHz

[(16 / 6) / F GHz ÷ (22 / 6) / 6 GHz] = 1
F = 4.363 GHz

At 4.363 GHz, P1 has the same performance as P2

DATASET B
We need to find the frequency of P1 at which the speedup ratio is 1. Assume frequency is F GHz

[(16 / 6) / F GHz ÷ (16 / 6) / 6 GHz] = 1
F = 6 GHz

At 6 GHz, P1 has the same performance as P2

PROBLEM 7

(a) For 'bzip2'
New CPI = [CPU time * clock rate] / No. of instructions
= [700 * 4 * 10 ^9] / (0.85 * 2389 * 10 ^9)
= **1.378**

For 'go'
New CPI = [CPU time * clock rate] / No. of instructions
= [620 * 4 * 10 ^9] / (0.85 * 1658 * 10 ^9)

= **1.759**

(b) For 'bzip2'
CPI @ 3GHz = [750 * 3 * 10^9] / (2389 * 10^9)
= **0.941**

[CPI @ 4 GHz] / [CPI @ 3 GHz] = 1.378 / .941 = 1.464

For 'go'
CPI @ 3GHz = [700 * 3 * 10^9] / (1658 * 10^9)
= **1.266**

[CPI @ 4 GHz] / [CPI @ 3 GHz] = 1.759 / 1.266 = 1.389

Clock rate ratio = New clock / Old Clock = 4 GHz / 3 GHz = 1.333

The ratios are dissimilar. This is because, although the number of instructions have been reduced by 15%, CPU time has been reduced by a lower percentage

(c) For 'bzip2'
CPU time reduction = [1 – (New execution time / Old execution time)] * 100 %
= [1 – (700 / 750)] * 100 %
= **6.667 %**

For 'go'
CPU time reduction = [1 – (New execution time / Old execution time)] * 100 %
= [1 – (620 / 700)] * 100 %
= **11.43 %**