2.14 Write SRC code to implement the following C statements, assuming all variables are 32-bit integers:

a. if (a == 0) a = a + 1; else a = a - 1;

b. for (i = 0; i < 10; i++)
   ndigit[i] = 0; assuming a declaration of ndigit[10]  

2.15 Testing a difference against zero is not the same as comparing two numbers in finite precision arithmetic. Propose an encoding for an SRC branch instruction that specifies two registers to be compared, rather than one register to be compared against zero.

a. What potential problems might there be with implementing the modified instruction?

b. How would condition codes improve the situation?

c. Can you suggest a restructuring of the SRC branch that would help without using condition codes?  

2.16 Procedure-calling sequences are standard groups of instructions that transfer control from the main program to a procedure, supplying it with input arguments if necessary. Return sequences finish the procedure by setting up any output arguments and transferring control back to the point following the call. Write a call and return sequence for an SRC procedure with two input and one output arguments.  

2.19 Modify the SRC to include the swap (op = 7) instruction that exchanges the contents of two registers, ra and rb, by writing RTN for the new instruction.  

2.20 a. Modify the SRC RTN to include a conditional jump instruction, jpr (op = 29). It should use format 2 in Figure 2.9. The jpr instruction uses relative addressing, rel, instead of a branch target register. The jump should be taken only if ra = 0.

b. Change the meaning of jpr so that the jump is taken only if the register specified by the ra field has a nonzero value.  

2.23 Using the hardware in Figure 2.23, write the RTN description and the control sequence that implements the following:


2.24 Design data path logic that will allow any one of the register transfers below to be done in one step, and give the control signals required to do each transfer.  

A(m - 1..0) ← B(m - 1..0);
C(m - 1..0) ← B(m - 1..0) + 1;
A(m - 1..0) ← C(m - 1..0);
B(m - 1..0) ← C(m - 1..0);