Computer Architecture: Spring 2001 – Test 2 Solution

1. (10 pts) Draw the architecture for an 8-bit Carry select adder where the grouping size is 4.

   See Slide #8 on Supplemental notes, Chapter 4.

2. (15 pts) Look at FIGURE 1 in the attached reference material. Give the equations for the following Propagate, Generate terms in terms of the appropriate A, B bits.

   G4,5 = G5 + P5G4 = A5 B5 + (A5+B5)A4 B4

   P4,5 = P5 P4 = (A5+B5)(A4+B4)

3. (15 pts) In class, we discussed two array architectures for a N x N multiplier (2N product bits). The simple array multiplier used N-1 ripple adders to sum the partial products. An alternate architecture was called the CSA (Carry Save Adder) multiplier and it required an extra adder at the end for a total of N adders. What was the PRINCIPLE benefit of the CSA approach? Draw the architecture of the CSA multiplier to illustrate your point.

See slide #17 on Supplemental notes, Chapter 4. There is a well defined critical path that we speed up by using a FAST adder like a CLA at the end to add the sum and carry vectors to produce the most significant half of the product.

4. (10 pts) Convert the following number in single precision IEEE floating point format to its decimal value (*no exponents allowed in the final decimal value*).

a.
Sign bit: 0
Exponent field: 01111111 *= 7fh = 127*
Significand Field: 110 0000 0000

Exponent = 127 - bias = 127 - 127 = 0. Number is $1.11_2 = 1 + .5 + .25 = 1.75$

b. If I want to increase the RANGE of a floating point number, I add bits to the:

   SIGNIFICAND **EXPONENT**

c. Give me a 'special' number that does not follow the normal IEEE formatting rules for exponents, significands (you do not have to tell me what the format is, just the name of the special number).

Some special numbers: zero, signed infinities, NaN (not a Number).

5. (5 pts) In the early days of microprocessor development, what was the original motivation behind using a microprogrammed approach to control implementation rather than hardwired control? *Microprogrammed logic is easier to design and modify, and this was important at the time because there were no automated synthesis tools for random logic.*

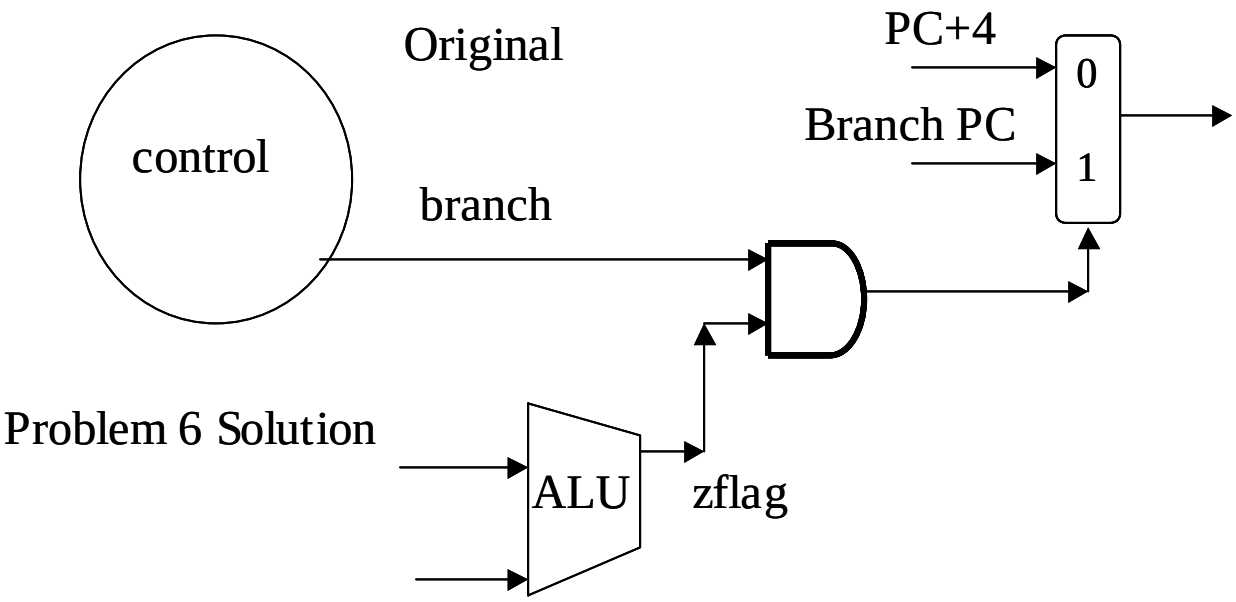
6. (15 pts) Refer to FIGURE 2, which shows the datapath for the SCI implementation.

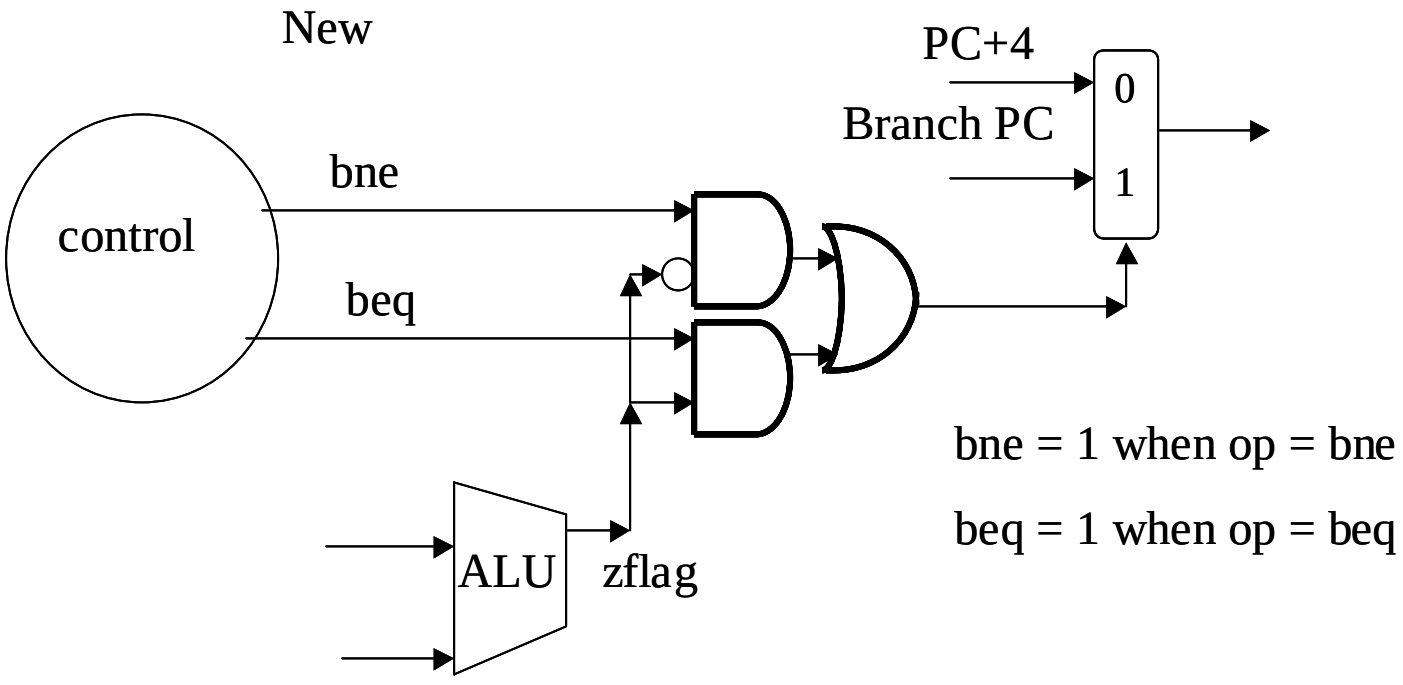
   The gating that is shown works fine for the BEQ instruction (BRANCH = 1 and ZERO =1 selects the branch address to be loaded into the PC). Modify this logic so that it will also accommodate the BNE instruction. Make your changes on FIGURE 2 (you can add extra control lines if you want, just make sure that you tell me what causes them to be asserted).

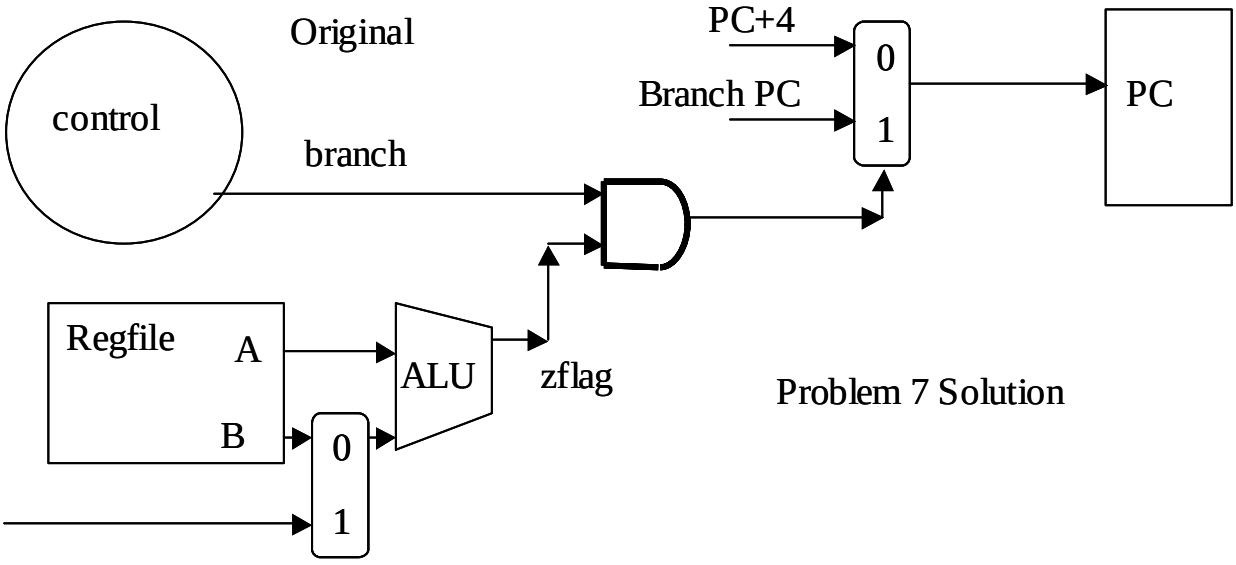
7. (15 pts) Refer to FIGURE 3, which shows the datapath for the SCI implementation. Make any necessary changes to the datapath to support the JR (jump register) instruction. Show any changes directly on Figure 3. If you add control lines, be sure to tell me what they are needed for.

8. (15 pts) Refer to FIGURE 4, which shows the datapath and state diagram for the MCI implementation. Using the EXISTING control lines, change the state diagram to support the JR (jump register) instruction. Show the extra state(s) that are needed to support this. For the ALUOP control lines, just use the names "ADD", "SUB", "AND", "OR", or "SLT" to indicate the appropriate operation.

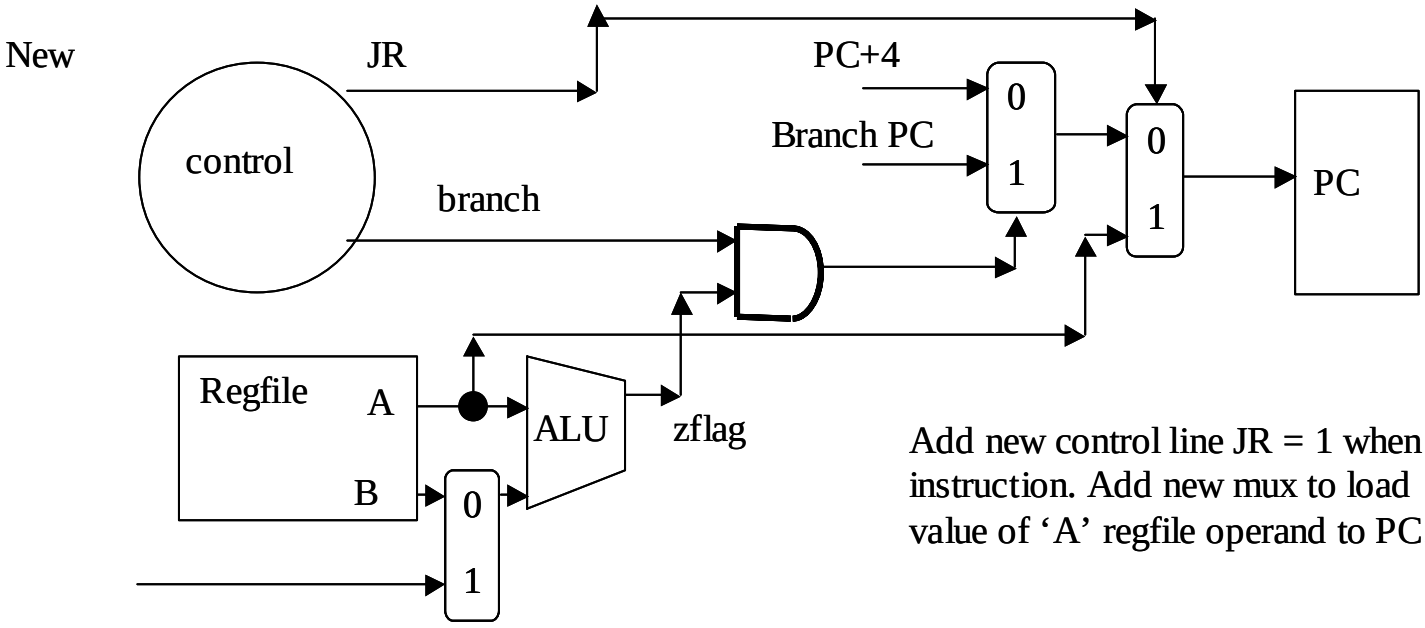
Problem 6 Solution





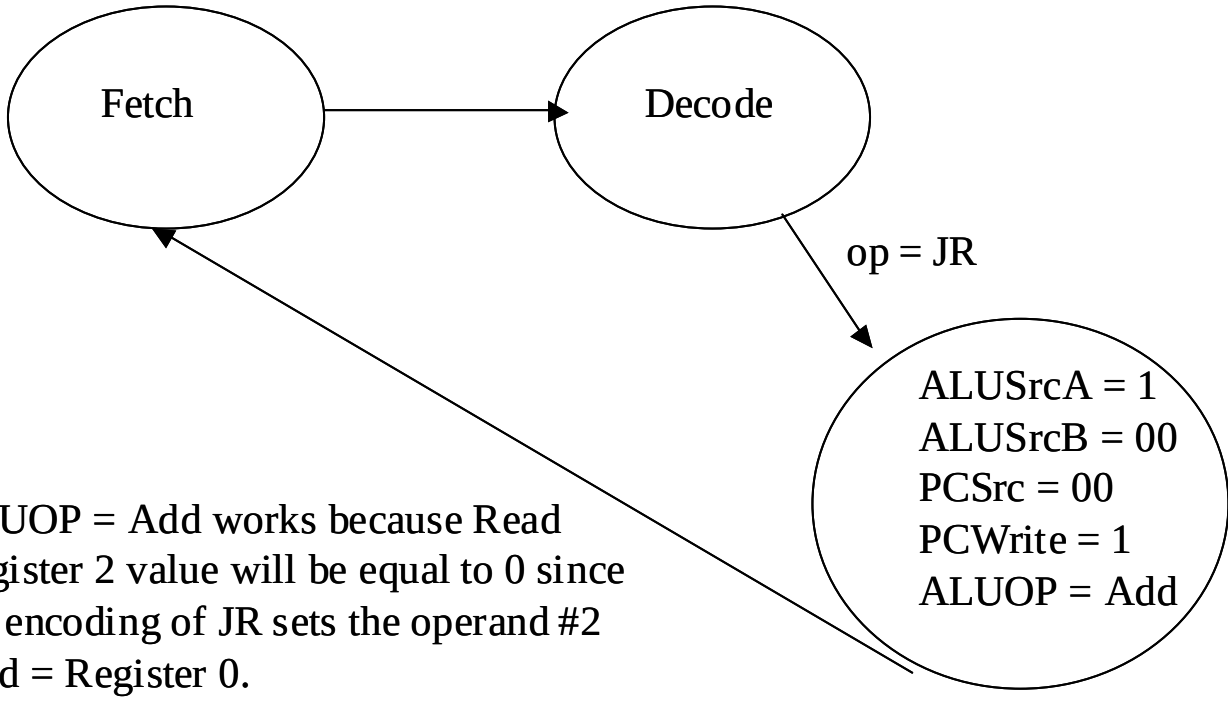


Problem 7 Solution



Add new control line JR = 1 when JR instruction. Add new mux to load value of ‘A’ regfile operand to PC.

Problem 8 Solution



ALUOP = Add works because Read Register 2 value will be equal to 0 since the encoding of JR sets the operand #2 field = Register 0.