Computer Architecture

You are to consider a hypothetical machine called SIC, for Single Instruction Computer. As its name implies, SIC has only one instruction: subtract and branch if negative, or sbn for short. The sbn instruction has three operands, each consisting of the address of a word in memory:

```
sbn a , b, c    # Mem[a] = Mem[a] - Mem[b] ; if  (Mem[a]<0)  go to c
```

The instruction will subtract the number in memory location b from the number in location a and place the result back in a, overwriting the previous value. If the result is greater than or equal to 0, the computer will take its next instruction from the memory location just after the current instruction. If the result is less than 0, the next instruction is taken from memory location c. SIC has no registers and no instructions other than sbn.

Although it has only one instruction, SIC can imitate many of the operations of more complex instruction sets by using clever sequences of sbn instructions. For example, here is a program to copy a number from location a to location b:

```
Start: sbn temp , temp , .+1   # Sets temp to zero
       sbn temp , a , .+1      # Sets temp to -a
       sbn b , b , .+1         # Sets b to  zero
       sbn b , temp , .+1      # Sets b to  - temp , which is a
```

In the program above, the notation .+1 means "the address after this one," so that each instruction in this program goes on to the next in sequence whether or not the result is negative. We assume temp to be the address of a spare memory word that can be used for temporary results.

1. (10%) **(a)** Write a SIC program to add a and b, leaving the result in a and leaving b unmodified. You need to write comments for your SIC program. **(b)** How many sbn instructions are necessary for your program?

2. (15%) **(a)** Write a SIC program to multiply a by b, putting the result in c. Assume that memory location one contains the number 1. Assume that a and b are greater than 0 and that it's OK to modify a or b. (Hint: What does the following program compute?) You need to write comments for your SIC program. **(b)** How many sbn instructions are necessary for your program?

```
c = 0 ; while (b > 0) {b = b - 1 ; c = c+ a ; }
```

3. (15%) **(a)** Someone suggests including another instruction **abn** into SIC, as defined in the following. So this new machine has two instructions: **abn** and **sbn**. Let's call this new machine as DIC (Double-Instruction Computer). Write a DIC program for the multiplication problem in Problem 2. Your objective is to minimize the number of total instructions needed in your program. You need to write comments for your DIC program. **(b)** How many **abn** instructions are necessary for your program? **(c)** How many **sbn** instructions are necessary for your program?

```
abn a , b, c    # Mem[a] = Mem[a] + Mem[b] ; if  (Mem[a]<0)  go to c
```

4. (10%) Compare SIC and DIC in terms of hardware and software aspects.

Data Structures

5 **[Sorting]** (20%)

The *median* of n numbers is the $((n+1)/2)$th largest if n is odd, and the average of the $(n/2)$th and $(n/2+1)$th if n is even. Show how to modify quicksort to find the median of an array of n numbers in $O(n)$ expected time.

6 **[Graphs]** (15%)

Tow graphs $G=(V,E)$ and $G'=(V',E')$ are *isomorphic* if there is a one-to –one and onto function $f: V \to V'$ such that $(v,w) \in E$ if and only if $(f(v), f(w)) \in E'$. In general it seems to be hard to decide whether two graphs are isomorphic without trying all possible isomorphism functions f. For some special cases, however , better methods are known.

two

Given ~~tow~~ trees, show how to test whether they are isomorphic.

7 **[Searching]** (15%)

Let B_n be the number of binary trees on n nodes. Explain the recurrence relation

$$B_n = \sum_{i=0}^{n-1} B_i B_{n-i-1}, \quad n > 1;$$

$$B_0 = B_1 = 1$$

Show that

$$B_n = \frac{(2n)!}{n!(n+1)!}$$