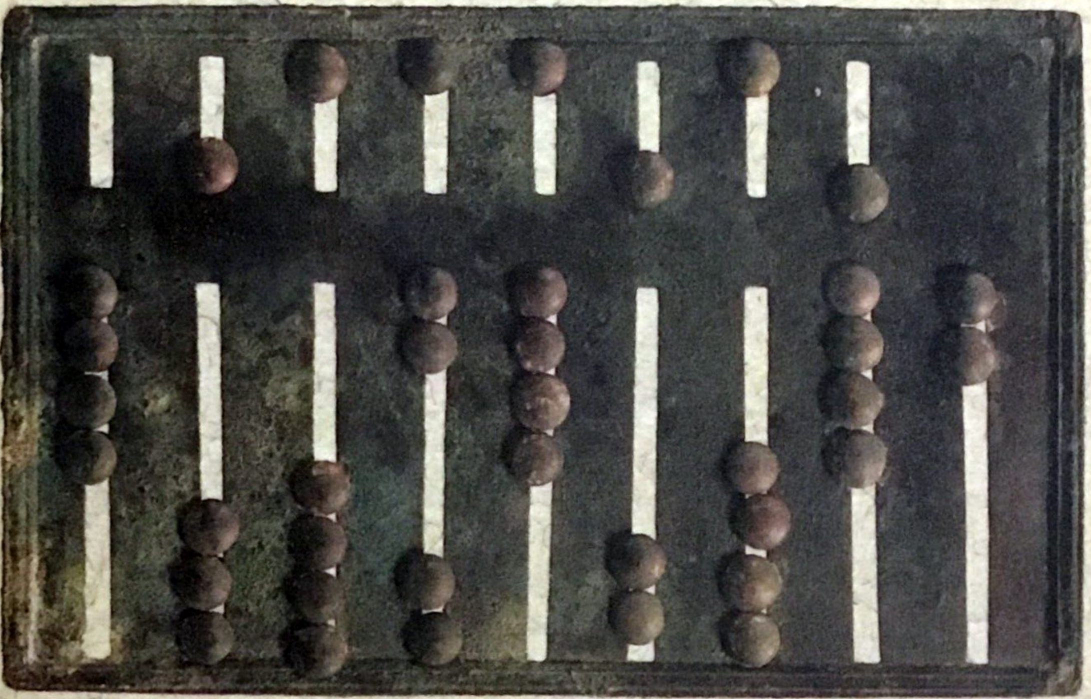
Second Edition

Computer Organization & Design

THE HARDWARE/SOFTWARE INTERFACE



David A. Patterson and John L. Hennessy

In More Depth

The Single Instruction Computer

The computer architecture used in this book, MIPS, has one of the simpler instruction sets in existence. However, it is possible to imagine even simpler instruction sets. In this assignment, 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.

3.29 [10] <§3.15> Write a SIC program to add `a` and `b`, leaving the result in `a` and leaving `b` unmodified.

3.30 [20] <§3.15> 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 this program compute?)

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