Nullary Computer

Edwin Andrés Niño Velásquez

Nullary Computer

Brian Huck has invented a new powersaving computer. With the current CMOS based processors, a certain amount of power is lost each time a bit is changed from 0 to 1 or back. To avoid this problem, Brian’s new Nullary Core stores only zeros. All numbers are stored in nullary form, as shown:

- 0
- 1 0
- 2 00
- 3 000
- 4 0000
- ...

As an examples we can give programs which calculate prime numbers. His initial 64-nit model has 26 registers, each of which may store up to 64 nits, and any attempt to store more than 64 nits will result in a run time error. There is also a flag register, which contains either a zero, or nothing.

Instruction set

The instruction set is as follows (include an explanation and how to simulate In C):

- A : Add a zero to the value in register A (similarly for all uppercase letters) : a++; in C.
- a : First, empty the flag register. Then, if possible, remove a zero from register A, and place it in the flag register (similarly for all lowercase letters) : flag = 0; if(a>0) flag=1; a--; in C.
- (: If the flag register is empty, jump past the matching). Otherwise, empty the flag register: while(flag) { flag=0; .... in C.
-) : Jump to the matching (: .... } in C.

Task

Your task will be to write a sorting program for Brian's Nullary Corebased Prototype Computer. The NCPC has limited memory, so your program must be no longer than 5432 instructions. Also, the running time of your program must be no more than $5*10^6$ steps for any possible input, where a step is considered to be the execution of one instruction. Important note: You must submit the nullary source code of this program, and not some Java, C or C++ source code.

- Input

  The numbers to be sorted will be given in the first 24 registers A-X; the remaining two registers (Y and Z) will be empty.

- Output

  The sorted numbers should be in registers A through X, in increasing order. Register Y and Z should be empty.

Sample programs

- **b(b)a(Ba)**

  Move register A to register B (by first emptying register B, then repeatedly pulling a single zero from register A and placing it into B).

- **XXXa(GIa)i(g(FYg)y(Gy)f(Zb(z)z(i(YBi)y(Iy))f)Zb(zb)z(xz)i)x**

  Set the flag register if the number of zeros in register A is prime.

Basic Commands

Reset A (A<-0)	a(a)
Move A B	<Reset B>a(Ba)
Assign A B {C} (B<-A)	<Reset B><Reset C>a(BCa)<Move C A>
A+=B {C}	<Assign B C>b(Ab)<Assign C B>
A-=B {C}	<Assign B C>b(ab)<Assign C B>
A++	A
A--	a

IF

IF_GT A B {C}	<Assign A D><Assign B E>e(de)d(…c(c))
IF_LT A B {C}	<Assign A D><Assign B E>d(ed)e(…c(c))

Loops

While_GT A B {D,E}	<Assign A D><Assign B E> e(de)d (… <Assign A D><Assign B E>e(de)d)
While_LT A B {D,E}	<Assign A D><Assign B E> d(ed)e (… <Assign A D><Assign B E>d(ed)e)
For0 B A {D,E} (For b=0 to A-1)	<Reset B><While_LT B A> (… B)
For1 B A {D,E} (For b=1 to A)	<Reset B><While_LT B A> (B …)

Using Loops

A*=B {C}	<Assign A D><FOR1 C B>(<A+=D>) <A-=D>
C = A*B {D}	<Reset C>C<FOR1 D B>(<C+=A>)
C = A/B	<Reset C><WHILE_LT A B>(C<A-=B>)A <IF_GT A B>(C)
C = A%B	<D=A/B><D*=B><Assign A C><C-=D>
A^=B {C}	<FOR1 C B>(<A*=A>)
C = A^B	<Reset C>C<FOR1 D B>(<C*=A>)