A 2–stack machine for multiplying natural numbers

Timothy Murphy
<tim@maths.tcd.ie>
School of Mathematics, Trinity College Dublin

April 22, 2003

Contents

We define a 2-stack machine which implements the function

$$[m][n] \mapsto [mn],$$

where $[m] = 1^m 0$. Our construction is expressed as a program in the tiny language $\mathcal{S}$, which we use for defining stack machines.

1 Strategy

We start by reading in the first number m and storing it (as 01^m) on the main stack (stack 0).

Then we enter the 'main loop', in which we read in the second number n bit-by-bit. As we read in each 1, we run through the main stack, writing out m 1's, at the same time storing these 1's on the auxiliary stack (stack 1). Then when the main stack is exhausted, we 'rewind' m from the auxiliary stack to the main stack, and return to the main loop.

1 ⟨`Multiplication.S` 1⟩ ≡
  ⟨Read first number onto main stack 2⟩
Loop: ⟨Read in bit of second number; if it is 0, write it out and halt 3⟩
  ⟨Pop 1's from main stack, write them and push them onto auxiliary stack 4⟩
  ⟨Rewind auxiliary stack onto main stack, and jumpto Loop 5⟩

2 Reading first number onto main stack

We start by pushing 0 onto stack 0, to mark the bottom of the stack. Then we read successive bits, pushing them onto stack 0 as long as they are 1's. When we meet a 0 we push it onto stack 1, to mark the bottom of that stack, and move on to the main loop.

At the end of this phase stack 0 holds 01^m and stack 1 holds 0.

2 ⟨Read first number onto main stack 2⟩ ≡
  put0; *push0*;
  read; *push0*; *jump* -2;
  pop0; *push1*;

This code is used in chunk 1.

3 Starting the cycle

We read in a bit from the second number.

If it is 0 then we are done; we write out 0 and halt.

If it is 1 then we enter the main cycle.

3 ⟨Read in bit of second number; if it is 0, write it out and halt 3⟩ ≡
  read; *jump3*; *write*; *halt*;

This code is used in chunk 1.

4 The main cycle

We go through the m 1's on stack 0, writing out a 1 for each 1, and also pushing a 1 onto stack 1.

When we meet a 0 (at the bottom of stack 0) we push it onto stack 1 to mark the bottom of that stack.

4 ⟨ Pop 1's from main stack, write them and push them onto auxiliary stack 4 ⟩ ≡

pop0 ; *jump4* ;
push0 ; *put1* ; *jump4* ;
push1 ; *write*; *jump* − 7;

This code is used in chunk 1.

5 Rewinding

Next we 'rewind' stack 1 onto stack 0.

5 ⟨ Rewind auxiliary stack onto main stack, and jumpto Loop 5 ⟩ ≡

pop1 ; *jump4* ;
push1 ; *put1* ; *jumpto Loop*;
push0 ; *jump* − 6;

This code is used in chunk 1.

6 The whole program

7 Appendix: Literate programming

This little program was written in `cweb`, Donald Knuth's implementation of his concept of 'literate programming'.

In brief, documentation and program are combined in a single 'web' file. This can then be processed in two ways: by `ctangle` to produce the program, or by `cweave` to produce the documentation.

This document is based on the web file `Multiplication.w`. The actual program (in the language $\mathcal{S}$) is produced by

```
% ctangle Multiplication.w
```

On the other hand, this document was produced by

```
% cweave Multiplication.w
```

producing the LaTeX file *TuringMachine.tex* which can be processed in the usual way

```
% latex Multiplication
% xdvi Multiplication
% dvips Multiplication
```

Index

List of Refinements