A 2–stack machine for multiplying natural numbers

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

April 16, 2003

Contents

WE DEFINE a 2-stack machine which implements the function

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

where $[m] = 1^m0$. 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 stack 1.

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 stack 1, writing out m 1's, at the same time storing them on stack 2. Then when stack 1 is exhausted, we 'rewind' m from stack 2 to stack 1, and return to the main loop.

1 ⟨Read first number onto main stack 2⟩
⟨Enter main loop 3⟩

2 Reading m onto stack 1

We start by pushing 0 onto stack 1, to mark the bottom of the stack. Then we read successive bits, pushing them onto stack 1 as long as they are 1's. When we meet a 0 we push it onto stack 2, to mark the bottom of that stack, and move on to the main loop.

At the end of this phase stack 1 holds 01^m and stack 2 holds 0.

2 ⟨Read first number onto main stack 2⟩ ≡
push1;
read; *push1*; *jump* -2;
pop1; *push2*;

This code is used in chunk 1.

3 The main loop

We read successive bits from the second number n, and go through the main cycle as long as the bit is 1.

3 ⟨Enter main loop 3⟩ ≡
⟨Read in bit of second number and halt if 0 4⟩
⟨Enter cycle if bit is 1 5⟩

This code is used in chunk 1.

4 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.

4 ⟨Read in bit of second number and halt if 0 4⟩ ≡
read; *jump3*; *write*; *halt*;

This code is used in chunk 3.

5 The main cycle

We go through the m 1's on stack 1, writing out a 1 for each 1, and pushing m onto stack 2. Then we 'rewind' stack 2 onto stack 1.

5 ⟨Enter cycle if bit is 1 5⟩ ≡
  ⟨Pop 1's from stack 1, write them and push them on stack 2 6⟩
  ⟨Rewind stack 2 onto stack 1 7⟩
This code is used in chunk 3.

6 First part of the cycle

6 ⟨Pop 1's from stack 1, write them and push them on stack 2 6⟩ ≡
  pop1; *jump4*;
  push2;
  put1;
  jump3;
  write;
  $jump - 6$;
This code is used in chunk 5.

7 Second part of the cycle

7 ⟨Rewind stack 2 onto stack 1 7⟩ ≡
  pop2; *jump3*;
  put1;
  $jump - 14$;
  push1;
  $jump - 5$;
This code is used in chunk 5.

8 The whole program

```
/*1:*/
/*2:*/
push 1;
read;push 1;jump -2;
pop 1;push 2;/*:2*/
/*3:*/
/*4:*/
read;jump 3;write;halt;/*:4*/
/*5:*/
/*6:*/
```

```
pop 1;jump 4;
push 2;put 1;jump 3;
write;jump -6;/*:6*/
/*7:*/
pop 2;jump 3;
put 1;jump -14;
push 1;jump -5;/*:7*//*:5*//*:3*//*:1*/
```

9 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