Pseudo-Code Reference Manual

1 Introduction

This document precisely defines the data types and input/output formats for the pseudo-code language described in section 1.2 of B. MacLennan's book *Principles of Programming Languages* (second edition, Holt, Rinehart & Winston, 1987).

This language manipulates data of three types: floating point numbers, array indexes (integers), and instructions. The instruction pointer *ip* is also manipulated by various instructions. Furthermore, most instructions affect the data memory, which contains 1000 words, each of which holds ten digits (plus) a sign. The notation $D[i]$ means the contents of data memory at address i.

2 Instructions Themselves

Abstractly, an instruction consists of a sign field s, an operation field *op*, and three address fields: *opn1*, *opn2*, *dest*. Each kind of instruction are described by its *operation code*, which is a combination of the s and *op* fields.

2.1 Input Representation (External Syntax)

The input representation of instructions consists of an sign, which is either `+` or `-` (ASCII codes 43 or 45), followed by a decimal digit (ASCII codes from 48 to 57, inclusive), followed by a space character (ASCII code 32), followed by three groups of three decimal digits each, with these groups separated by a single space. The sign encodes the s field, the first digit the *op* field, while the remaining three groups of decimal digits encode the *opn1*, *opn2*, and *dest* fields (in order). For example, the representation

```
+0 001 002 003
```

encodes the instruction whose s field is $+$, whose *op* field is 0, whose *opn1* field is 1, whose *opn2* field is 2, and whose *dest* field is 3.

3 Floating Point Numbers

Abstractly, each floating point number can be thought of as representing a rational number that can be written in the form $m \times 10^e$, where m (the mantissa) is an integer that ranges from -99999999 to 99999999 (inclusive) and e (the exponent) is an integer that ranges from -50 to 49 (inclusive). The exact range may be larger, and the precison may be more than 8 decimal digits, at the discretion of the implementor.

3.1 Input/Output Representation

The input/output representation of the floating point numbers is like that of the real literals in most modern programming languages. It consists of an optional sign, followed by a decimal fraction (consisting of a string of decimal digits, a decimal point (.), and another string of decimal digits, where either string may be empty, not not both), the letter “e” (in lower or upper case), an optional sign, and a string of one or more decimal digits.

The mantissa is given by the sign and the decimal fraction. The exponent follows the letter “e”. For example, the number 27×10^0 (i.e., 27) is represented by the following strings:

```
+2.70000000e+01
27.e0
0.27e2
```

3.2 Instructions for Manipulating Floating Point Numbers

The instructions for manipulating floating point numbers are adequately described in MacLennan’s book, so a few examples will suffice.

The effect of the +0 instruction (move) is

$$\mathrm{D[dest]} \leftarrow \mathrm{D[opn1]}$$

which says that the contents of the data memory addressed by the *opn1* field are stored in the word whose address is addressed by the *dest* field. For example, the instruction written

```
+0 000 999 027
```

moves the contents of location 0 to location 27.

The effect of the +1 instruction (add) is

$$\mathrm{D[dest]} \leftarrow \mathrm{D[opn1]} \oplus \mathrm{D[opn2]}$$

which says that the contents of data memory addressed by the *opn1* and *opn2* fields are added (using floating point addition), and stored in the word whose address is addressed by the *dest* field. For example, the instruction written

```
+0 011 022 033
```

adds the contents of locations 11 and 22, and stores the result in location 33.

3.3 Internal Representation

Floating point numbers must be stored so that numbers from 0 to 999 can be treated as indexes, which are described below. (The obvious mapping from the external representation into ten-digit words will serve this purpose, although other representations are fine.)

4 Indexes

Abstractly, (array) indexes are a subset of the mathematical integers. The exact subset is left to the implementor, but should include at least all the integers from 0 to 999 inclusive.

4.1 Input/Output Representation

As above, the input/output representation of the indexes consists of an explicit sign, followed by a decimal digit, followed by a space character, followed by three groups of three decimal digits each, with these groups separated by a single space.

These respresentations are interpreted as decimal notation for integers, in the usual way, after deleting the spaces. For example, the index 27 is represented by the following string:

```
+0 000 000 027
```

Notice that, the above is also the external representation for the floating point number 27×10^0. This coincidence is required to make MacLennan's figure 1.3 work correctly.

4.2 Instructions for Indexing

The instructions for manipulating indexes require some comment.

The pseudo-code instructions do not explicitly allow arithmetic on indexes. However, it is possible to treat indexes as floating point numbers, and hence manipulate them as floating point numbers. It is also possible to treat floating point numbers that represent the integers from 0 to 999 (at least) as indexes.

The instruction with operation code `+6` is an array fetch. It has the following effect

$$\text{D[dest]} \leftarrow \text{D[opn1 + D[opn2]]}$$

That is, it stores the contents of the location with address $opn1 + D[opn2]$ into the location addressed by the *dest* field. Thus the location addressed by *opn2* acts as an index. See the example on page 12 of MacLennan's book. Note however, that the expression $x_i \leftarrow z$ on page 12, should read "$x_i \rightarrow z$." It is an error for a program to try to address outside the data memory.

The instruction with operation code `-6` is an array store. It has the following effect

$$\text{D[opn2 + D[dest]]} \leftarrow \text{D[opn1]}$$

That is, it stores the contents of the location with address *opn1* into the location with address $opn2 + D[dest]$. Thus the location addressed by *dest* acts as an index. See the example on page 12 of MacLennan's book.

The instruction with operation code `+7` is an increment-and-test instruction. It has the following effect

D[opn1] $\leftarrow$D[opn1] + 1
if D[opn1] < D[opn2] **then** ip $\leftarrow$dest

5 Programs

Programs are as described on page 14 of MacLennan's book. However, we have different representations for the "flag values" and a separate representations for the flag value that ends the initial data section and the flag value that ends the instruction section.

The flag value that ends the intial data section is written

```
+9.9999999e99
```

The flag value that ends the instruction section is written with embedded spaces:

```
+9 999 999 999
```