LOCAL VARIABLES

John R. Hart
John Perona
HARTRONIX Inc.
1201 N. Stadem dr.
Tempe, Az. 85202

INTRODUCTION

There are some cases where the number of variables inside a procedure exceed the maximum that can be easily accessed with simple stack operators.

Local variables are proposed to free programs from stack thrashing and spaghetti like manipulation that can occur in programs with a large number of stack variables.

DESCRIPTION

Standard FORTH can easily access the top three elements on the stack. The Return stack can contain a fourth variable. Programs with more than four variables require a careful arrangement of the stack order and execution sequence. Sometimes there is no reasonable solution. These problems make it difficult to justify using FORTH for tasks that require a large number of variables.

There are several ways to implement local variables. One is to use a standard variable and save its contents at the start of the procedure and restore at the end. Other methods would be to assign space in memory or on one of the stacks and develope a means of addressing the space.

A data stack implementation has the following advantages:

1. Ease of allocation of temporary space.
   A simple adjustment to the stack and local frame pointer create Local variable space.
2. Ease of accessing the local variables.
   Addition of an offset to the local frame pointer gets the address of a Local.
3. Self initialization
   All stack input to a procedure is in the same space that is assigned to the Local input parameters.
4. Ease of deallocation of Local space.
   When a procedure is finished Local space is deallocated by moving the data on the stack up to the end of the Local frame pointer.
5. Overflow exceptions are detected by the stack and dictionary error checking procedures already in FORTH.

SYNTAX

The syntax developed to define the local variables is similar to the usual input output comment used in FORTH. Local variables are defined after the name of the procedure by enclosing them in braces "{}".

First the input parameters are listed followed by a dash. Next the temporary variables followed by a dash and finally the output parameters. To facilitate variable length output and recursion, the output labels are used only as a comment. When the semicolon is encountered, the local labels are forgotten from the dictionary.

```
: EXAMPLE   { IN1 IN2 - TMP1 TMP2 - OUT1 OUT2 }
  IN1 IN2 + -> TMP1
  IN1 IN2 - -> TMP2
  IN1 TMP1 *   IN2 TMP2 * \ leaves the outputs on the stack
```

When referenced, local variables are fetched to the stack. When the label is preceded by a goes to arrow " -> ", a number is taken from the stack and stored in the variable. If the address of the local is needed an " 'A " will put the address of the local on the stack.

This allows words like " +! " to be used.

Example:

```
      45 'A TMP2 +!@
```

The example adds 45 to TMP2 and fetches it to the stack.

Implementation can be divided into the execution phase and the compilation phase.

IMPLEMENTATION

At the start of a procedure containing local variables, the old local frame pointer is saved on the return stack. The new frame pointer is set up and any temporary variables are filled with zeros.

The initialization of the locals is accomplished by calling the EXECUTE.LOCALS program. The two initialization parameters which follow the call are:

1. The total number of locals.
2. The number of temporary variables.

After initialization, the EXECUTE.LOCALS program calls the body of the procedure which is located after the initialization parameters.

When the procedure that is using locals exits, it returns to the last part of the EXECUTE.LOCALS program where the local space is deallocated. The old frame pointer is restored and the output is copied to the current stack position.

References to locals inside of the body of the procedure will compile in one of three different programs. LOCAL returns the address of the variable. LOCAL@ returns the contents of the variable. LOCAL! stores a word from the stack into the variable.

These programs should be written in machine code so they will execute as fast as possible.

```
: EXECUTE.LOCALS          ( --- )
    R> FRAME @ >R >R              \ save old frame pointer
    R@ 1 + @ ?DUP                 \ get number of temporary vars
      IF 1 DO Ø LOOP THEN         \ initialize temporary vars
    SP@ FRAME !                   \ setup new frame pointer
    R@ 2 + EXECUTE                \ execute body of procedure
    FRAME @                       \ bottom of frame
    DUP R@ @ +                    \ top of frame
    OVER SP@ 3 - -                \ number of words to move
    MOVE>
    R> @ SP+!                     \ adjusts stack pointer
    R> FRAME !                    \ restore old frame pointer
;

: LOCAL                   ( --- ads )
    R> DUP 1+ >R @              \ gets the local offset
    FRAME @ +                   \ calculates the local address
;

: LOCAL@                  ( --- data )
    LOCAL @
;

: LOCAL!                  ( data --- )
    LOCAL !
;
```

Compilation of locals begins with a " { " following the name of a procedure. This program first compiles in the address of the EXECUTE.LOCALS program followed by two zeros. Then the current values of HERE and LATEST must be saved. HERE is set to zero and the error vector is set to the local compiling error program.

Input labels are parsed and created in a new dictionary whose space is located 100 locations from the top to the stack. The zero in HERE is a flag saying that no labels have been created yet. When a " - " is found, the parsing of input labels stops and the parsing of temporary labels starts.

When the second " - " is encountered, the create phase is finished and the program looks for the " } " skipping over the output labels. If the symbols " { - - } " are missing or in the wrong order the local compile error will be called. After correct creation of local labels, the error vector is changed to the forget locals program. HERE and LATEST are restored.

The remainder of the procedure compiles in the normal manner but the semicolon forgets the local labels.

NOTE: To obtain the remainder of the code contact HARTRONIX, Inc. The address is listed at the beginning of this article.