COMP40 Assignment: Assembly-Language Programming

Contents

1 Purpose and overview

The purpose of this assignment is to deliver on the second half of the course title: you get to do some assembly-language programming. You will consolidate and solidify your knowledge of machine-level programming by implementing a calculator that uses Reverse Polish Notation[1], like the immortal HP 15C[2].

2 An RPN calculator

The COMP 40 RPN calculator reads commands from standard input and prints results to standard output. Like all RPN calculators, it works with a *value stack*. In this case, a value on the stack is one Universal Machine word. The command set is shown in Figure 1; Figure 2 shows an example interaction. You will find a complete reference implementation in file `/comp/40/www/homework/calc.c`[3], and you can run a binary in `/comp/40/bin/calc40`.

Your assignment is to implement this calculator in Universal Machine Assembly language. *Your calculator must duplicate the output of the reference implementation exactly.*

The implementation of the calculator is mostly straightforward: the only persistent state is the value stack, and this value stack is manipulated by each command independently of the others, using purely local reasoning. There is one dirty trick, however: in order to make it possible to read the digits of a numeral one character at a time, the calculator uses a finite-state machine with two states: *waiting* and *entering*. The normal state, which is also the initial state, is *waiting*. The *entering* state is used only when the entry of a numeral is in progress.

- If the machine is *waiting* and it sees a digit, it treats that digit as the start of a numeral, pushes the *value* of that digit, then transitions to the *entering* state.
- If the machine is *entering* and it sees a digit, that digit *continues* a numeral that was already pushed. The machine therefore takes the number on the top of the stack, multiplies it by 10, and adds the value of the next digit.
- In either state, if the machine sees a nondigit, it performs the command associated with that nondigit (if any), then transitions to the *waiting* state.

Here are two examples:

- If the machine sees the string "42", it first pushes the number 4 (value of the digit '4'), then transitions into the *entering* state. It then sees the digit '2' while still in the *entering* state, so it pops 4 and pushes $10 \times 4 + 2$, that is, 42. The result is the single number 42 on the stack.
- If the machine sees the string "4 2", with a space between the digits, it first pushes the number 4 (value of the digit '4'), then transitions into the *entering* state. It then sees the space character while still in the *entering* state. Because the space character is not a digit, the machine performs the associated command (doing nothing) and transitions back to the *waiting* state. Finally, while in the *waiting state*, it sees the digit '2', so it pushes the number 2. The result is *two* numbers on the stack: 2 on top and 4 on the bottom.

[1] See URL `http://www8.hp.com/us/en/pdf/Why_RPN_1_tcm_245_1078603.pdf`.
[2] See URL `http://hp15c.org/hp15c.php`.
[3] See URL `http://www.cs.tufts.edu/comp/40/homework/calc.c.txt`.

Command	*Function*
n	Push n onto the value stack, where n is a numeral (sequence of digits).
space	Does nothing, but may be used to separate numerals, as in the command sequence "6 7*."
newline	Print the contents of the value stack
+	Pop y from the value stack, then pop x from the value stack, then push $x + y$.
−	Pop y from the value stack, then pop x from the value stack, then push $x - y$.
*	Pop y from the value stack, then pop x from the value stack, then push $x \times y$.
/	Pop y from the value stack, then pop x from the value stack, then push $x \div y$. If y is zero, print an error message and leave the stack unchanged.
|	Pop y from the value stack, then pop x from the value stack, then push $x \vee y$, where $\vee$ stands for bitwise or.
&	Pop y from the value stack, then pop x from the value stack, then push $x \wedge y$, where $\wedge$ stands for bitwise and.
c	(Change sign.) Pop x from the value stack, then push $-x$.
~	Pop x from the value stack, then push $\neg x$, where $\neg$ stands for bitwise complement.
s	Swap the two values on top of the value stack (exchange x and y).
d	Duplicate the value on the top of the stack. (The HP 15C uses the ENTER key.)
p	Pop a value off the value stack and discard it.
z	Remove all values from the value stack (zero stack).

Figure 1: Calculator commands

```
sunfire31{nr}403: calc40
6 7 *
>>> 42
2 +
>>> 44
11 /
>>> 4
c
>>> -4
p
466 319sd+240c807c    sd-
>>> 0
>>> -807
>>> 932
>>> 319
```

Figure 2: Interacting with the RPN calculator