CSCI 410
11 – Introduction to Stack Machines

Stacks: “Last in, first out”
Stacks are a LIFO (Last in, first out) data structure. Think of pancakes:

- This pancake was put on top last.
- Which one would you eat first?
- Which would you eat second?

Three Operations
- peek: Look at the top item on the stack.
- push: Add an item to the top of the stack.
- pop: Remove the top item from the stack.

Stack Machines
- Idea: all data is on a stack
- Basic operation:
  - Push operands
  - Apply operator: pops n operands, pushes answer
- Advantages (contrast with register machines)
  - Simple op codes (no need to specify to/from registers)
  - More local/fast storage (stack can be larger than # regs)
- Disadvantages
  - Need for load/store instructions to memory
  - More operations/some duplication of effort on repeated operations

Some Stack Machines
- RPN (postfix) calculator
- PostScript, Forth languages
- JVM (Java virtual machine), CLR (.NET virtual machine)
- Various hardware implementations (CPUs, etc.)

Example: RPN Calculator
Reverse Polish Notation
- Name from “Polish notation” – invented by Jan Lukasiewicz (1920s) – this was the prefix form
- “Reverse” (postfix) invented in 1954 and independently again in 1960’s (including by Dijkstra)
- Operators follow operands
  - 4 * (1 + 7) - normal “infix” notation
  - 4 1 7 * + - RPN “postfix” notation
  - 1 7 + * - an equivalent expression (due to commutativity)
- Note no parentheses needed, no operator precedence!

Used in many handheld calculators
Interpreting RPN

\[
2 6 1 - 8 \times + \\
2 (6 1 -) 8 \times + \\
2 5 8 \times + \\
2 (5 8 \times) + \\
2 40 + \\
(2 40 +) \\
42
\]

Stack Machine Interpretation

\[
\begin{align*}
&2 6 1 - 8 \times + \\
&2 (6 1 -) 8 \times + \\
&2 5 8 \times + \\
&2 (5 8 \times) + \\
&2 40 + \\
&(2 40 +) \\
&42
\end{align*}
\]

Example: Forth

\[
\text{Example: Forth}
\]

\[
: \text{fib} \ ( \ +n \ -- \ +n2 \ ) \ ( \text{compute the } n\text{-th Fibonacci number} )
\]

dup \theta \ if
\theta \ 1 \ begin
rot \ dup \ 1 \ > \ while
1 -
rot rot
dup rot +
repeat
drop nip
then ;
\]

More about Forth

\[
\text{More about Forth}
\]

- In addition to above, language has:
  - Recursion
  - Local variables
  - A secondary stack
  - Global memory manipulation
  - ...

- History
  - Language invented \textasciitilde1968 by Charles Moore
  - Compact, highly portable – popular for small footprint
  - See e.g., OpenFirmware

Preview: Hack Virtual Machine

<table>
<thead>
<tr>
<th>Command</th>
<th>Return value (after popping top operand)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>(+) \ 2 \left(1\right) \ complemented</td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>(-) \ 2 \left(1\right) \ complemented</td>
<td></td>
</tr>
<tr>
<td>neg</td>
<td>\text{true} if (x = y) \ and \ false \ otherwise</td>
<td></td>
</tr>
<tr>
<td>eq</td>
<td>\text{true} if (x &gt; y) \ and \ false \ otherwise</td>
<td></td>
</tr>
<tr>
<td>gt</td>
<td>\text{true} if (x \leq y) \ and \ false \ otherwise</td>
<td></td>
</tr>
<tr>
<td>lt</td>
<td>\text{true} if (x &gt; y) \ and \ false \ otherwise</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>\text{true} if (x \text{ and } y) \ and \ false \ otherwise</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>\text{true} if (x \text{ or } y) \ and \ false \ otherwise</td>
<td></td>
</tr>
<tr>
<td>not</td>
<td>\text{true} if (x = y) \ and \ false \ otherwise</td>
<td></td>
</tr>
</tbody>
</table>

Preview: Hack Virtual Machine

<table>
<thead>
<tr>
<th>Arithmetic / Boolean commands</th>
<th>Program flow commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>label \ (declaration)</td>
</tr>
<tr>
<td>sub</td>
<td>goto \ (label)</td>
</tr>
<tr>
<td>neg</td>
<td>if-goto \ (label)</td>
</tr>
<tr>
<td>eq</td>
<td></td>
</tr>
<tr>
<td>gt</td>
<td></td>
</tr>
<tr>
<td>lt</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>not</td>
<td></td>
</tr>
</tbody>
</table>

Memory access commands

- pop \(x \) (pops \(x\), which is a variable)
- push \(y\) (y being a variable or a constant)