MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2004

Recitation 25
Stack and Procedures

Contracts

Input Register(s) whose value is read and used before it is written.

Output Register(s) designated as output.

Modifies Register(s) whose value after the code block *could* differ from their original value.

1. What is the contract for the following code:

```
expt
 (assign result (const 1))
expt-loop
 (test (op <=) (reg arg1) (const 0))
 (branch (reg continue))
 (assign result (op *) (reg result) (reg arg0))
 (assign arg1 (op -) (reg arg1) (const 1))
 (goto (label expt-loop))
```

Input:
Output:
Modifies:

2. What is the contract for the following code:

```
foo
 (assign y (reg x))
 (assign x (op cons) (reg x) (reg y))
 (test (op null?) (reg x))
 (branch (label yack))
 (assign val (const 2))
 (assign x (reg y))
 (goto (reg continue))
yack
 (assign foo (const 7))
 (assign val (op car) (reg x))
 (goto (reg continue))
```

Input:
Output:
Modifies:

Save and Restore

`(save` *reg)*

Place the value in register *reg* on top of the stack. This will also increment the stack pointer (`sp`). If the stack has no space left (by default, 1000 elements), a "stack overflow" error is signalled. To place the value in the register `result` on the stack:
`(save result)`

`(restore` *reg)*

Take the top value off the stack and put it in register *reg*. This will also decrement the stack pointer (`sp`). If the stack has nothing on it, a "stack underflow" error is signalled. To remove the top element of the stack and place it in the register `result`:
`(restore result)`

Procedure Call

1. `save` things you care about
2. `assign` values to the inputs, including `continue` to an appropriate label
3. `goto` the procedure's label
4. return label
5. `restore` things you cared about, in reverse order

Problems

3. Implement `aexpb`, which computes ae^b. You should call `expt` in your solution.

4. Translate `list-copy` into register machine code:

```
(define (list-copy lst)
  (if (null? lst)
      '()
      (cons (car lst)
            (list-copy (cdr lst)))))
```

Register machine code:

```
list-copy
 (test (op null?) (reg arg0))
 (branch (label list-copy-done))
 (save arg0)
 (save continue)
 (assign arg0 (op cdr) (reg arg0))
 (assign continue (label list-copy-after))
 (goto (label list-copy))
list-copy-after
```

```
list-copy-done
 (assign result (const ()))
 (goto (reg continue))
```

5. Translate `list-ref` into register code, verbatim:

```
(define (list-ref lst k)
  (if (= k 0)
      (car lst)
      (list-ref (cdr lst) (- k 1))))
```

Register machine code:

```
list-ref
 (test (op =) (reg arg1) (const 0))
 (branch (label list-ref-done))
```

```
list-ref-done
 (assign result (op car) (reg arg0))
 (goto (reg continue))
```

6. Translate `sum-of-exps` into register code:

```
(define (sum-of-exps lst)
  (if (null? lst)
      0
      (+ (expt 2.71828 (car lst))
         (sum-of-exps lst))))
```

   You should unroll the recursive call into a loop (don't translate the code verbatim).