

Exercise 5.11.

When we introduce `save` and `restore` in section 5.1.4, we didn't specify what would happen if you tried to restore a register that was not the last one saved, as in the sequence

```
(save y)
(save x)
(restore y)
```

There are several reasonable possibilities for the meaning of `restore`:

- `(restore y)` puts into `y` the last value saved on the stack, regardless of what register that value came from. This is the way our simulator behaves. Show how to take advantage of this behavior to eliminate one instruction from the Fibonacci machine of section 5.1.4 (figure 5.12).
- `(restore y)` puts into `y` the last value saved on the stack, but only if that value was saved from `y`; otherwise, it signals an error. Modify the simulator to behave this way. You will have to change `save` to put the register name on the stack along with the value.
- `(restore y)` puts into `y` the last value saved from `y` regardless of what other register were saved after `y` and not restored. Modify the simulator to behave this way. You will have to associate a separate stack with each register. You should make the `initialize-stack` operation initialize all the register stacks.

Answer.

- Observe that when the control reaches `afterfib-n-2` in the Fibonacci machine, the register `val` contains `Fib (n - 2)` and the last value saved on the stack is `Fib (n - 1)`. The machine then must add up `Fib (n - 1)` and `Fib (n - 2)` to compute their sum and return to their caller. As addition is commutative, `Fib (n - 1)` and `Fib (n - 2)` can be distributed arbitrarily between register `n` and `val`. This indicates that relocation on their values become unnecessary and the instructions

```
(assign n (reg val))
(restore val)
```


that move their values around can be replaced by a single instruction

```
(restore n)
```

- We change the stack instructions `save` and `restore` to use the stack with a pair that consists of the designated register name as well as the value.

```
(define (make-save inst machine stack pc)
  (let ((reg-name (stack-inst-reg-name inst)))
    (let ((reg (get-register machine reg-name)))
      (let ((record (list reg-name (get-contents reg))))
        (lambda ()
          (push stack record)
          (advance-pc pc))))))

(define (make-restore inst machine stack pc)
  (let ((reg-name (stack-inst-reg-name inst)))
    (record (pop stack)))
  (if (eq? reg-name (car record))
      (let ((reg (get-register machine reg-name)))
```

*. Creative Commons  2014, Lawrence X. A. Yan (颜世敏, aka 颜序).
Email address: informlarry@gmail.com

c. We can associate registers other than `pc` and `flag` each with a separate stack. The stack is named by appending that register's name with a suffix "`-stack`". To do this, we modify the constructor for stack to take a name as its argument.

The **make-new-machine** procedure, employs a stack table instead of a single **stack** variable so that all the registers can save and restore their value independly. Whenever a register is allocated, the machine immediately creates a stack associated with it.

2

```

'register-allocated)
(define (allocate-stack stack-name)
  (set! stack-table
    (cons (list stack-name (make-stack stack-name))
      stack-table)))
(define (lookup-register reg-name)
  (let ((val (assoc reg-name register-table)))
    (if val
      (cadr val)
      (error "Unknown register: " reg-name))))
(define (lookup-stack stack-name)
  (let ((val (assoc stack-name stack-table)))
    (if val
      (cadr val)
      (error "Unknown stack: " stack-name))))
(define (execute)
  (let ((insts (get-contents pc)))
    (if (null? insts)
      'done
      (begin
        ((instruction-execution-proc (car insts))
          (execute)))))
(define (dispatch message)
  (cond ((eq? message 'start)
    (set-contents! pc the-instruction-sequence)
    (execute))
    ((eq? message 'install-instruction-sequence)
    (lambda (seq) (set! the-instruction-sequence seq)))
    ((eq? message 'allocate-register) allocate-register)
    ((eq? message 'get-register) lookup-register)
    ((eq? message 'get-stack) lookup-stack)
    ((eq? message 'install-operations)
    (lambda (ops) (set! the-ops (append the-ops ops))))
    ((eq? message 'operations) the-ops)
    (else (error "Unknown request -- MACHINE" message))))
dispatch)))

```

Like the register, we use the `get-stack` procedure to look up the stack with a given name in a given machine:

```

(define (get-stack machine stack-name)
  ((machine 'get-stack) stack-name))

```

Since the `stack` variable in the register machine has been replaced by a stack table, the assembler now must extract the stack associated with the designated register from the `save` and `restore` instructions when it generate their execution procedures.

```

(define (update-insts! insts labels machines)
  (let ((pc (get-register machine 'pc))
    (flag (get-register machine 'flag))
    (ops (machine 'operations)))
    (for-each
      (lambda (inst)
        (set-instruction-execution-proc!
          inst
          (make-execution-procedure

```

```

        (instruction-text inst) labels machine
        pc flag ops)))
    insts)))

(define (make-execution-procedure inst labels machine
                                     pc flag ops)
  (cond ((eq? (car inst) 'assign)
        (make-assign inst machine labels ops pc))
        ((eq? (car inst) 'test)
        (make-test inst machine labels ops flags pc))
        ((eq? (car inst) 'branch)
        (make-branch inst machine labels flag pc))
        ((eq? (car inst) 'goto)
        (make-goto inst machine labels pc))
        ((eq? (car inst) 'save)
        (make-save inst machine pc))
        ((eq? (car inst) 'restore)
        (make-restore inst machine pc))
        ((eq? (car inst) 'perform)
        (make-perform inst machine labels ops pc))
        (else (error "Unknown instruction type -- ASSEMBLE"
                      inst))))

(define (make-save inst machine pc)
  (let ((reg (get-register machine
                          (stack-inst-reg-name inst)))
        (stack (get-stack machine
                          (stack-inst-stack-name inst))))
    (lambda ()
      (push stack (get-contents reg))
      (advance-pc pc))))

(define (make-restore inst machine pc)
  (let ((reg (get-register machine
                          (stack-inst-reg-name inst)))
        (stack (get-stack machine
                          (stack-inst-stack-name inst))))
    (lambda ()
      (set-contents! reg (pop stack))
      (advance-pc pc))))

(define (stack-inst-reg-name stack-instruction)
  (cadr stack-instruction))

(define (stack-inst-stack-name stack-instruction)
  (string->symbol
   (string-append (symbol->string
                     (cadr stack-instruction))
                  "-stack")))

```