5.2 A Register-Machine Simulator

In order to gain a good understanding of the design of register machines, we must test the machines we design to see if they perform as expected. One way to test a design is to hand-simulate the operation of the controller, as in exercise 5.5. But this is extremely tedious for all but the simplest machines. In this section we construct a simulator for machines described in the register-machine language. The simulator is a Scheme program with four interface procedures. The first uses a description of a register machine to construct a model of the machine (a data structure whose parts correspond to the parts of the machine to be simulated), and the other three allow us to simulate the machine by manipulating the model:

`(make-machine <register-names> <operations> <controller>)`
constructs and returns a model of the machine with the given registers, operations, and controller.

`(set-register-contents! <machine-model> <register-name> <value>)`
stores a value in a simulated register in the given machine.

`(get-register-contents <machine-model> <register-name>)`
returns the contents of a simulated register in the given machine.

`(start <machine-model>)`
simulates the execution of the given machine, starting from the beginning of the controller sequence and stopping when it reaches the end of the sequence.

As an example of how these procedures are used, we can define `gcd-machine` to be a model of the GCD machine of section 5.1.1 as follows:

```
(define gcd-machine
  (make-machine
   '(a b t)
   (list (list 'rem remainder) (list '= =))
   '(test-b
       (test (op =) (reg b) (const 0))
       (branch (label gcd-done))
       (assign t (op rem) (reg a) (reg b))
       (assign a (reg b))
       (assign b (reg t))
       (goto (label test-b))
     gcd-done)))
```

The first argument to `make-machine` is a list of register names. The next argument is a table (a list of two-element lists) that pairs each operation name with a Scheme procedure that implements the operation (that is, produces the same output value given the same input values). The last argument specifies the controller as a list of labels and machine instructions, as in section 5.1.

To compute GCDs with this machine, we set the input registers, start the machine, and examine the result when the simulation terminates:

```
(set-register-contents! gcd-machine 'a 206)
done
(set-register-contents! gcd-machine 'b 40)
```

To compute GCDs with this machine, we set the input registers, start the machine, and examine the result when the simulation terminates:

```
(set-register-contents! gcd-machine 'a 206)
done
(set-register-contents! gcd-machine 'b 40)
done
(start gcd-machine)
done
(get-register-contents gcd-machine 'a)
2
```

This computation will run much more slowly than a `gcd` procedure written in Scheme, because we will simulate low-level machine instructions, such as `assign`, by much more complex operations.

Exercise 5.7. Use the simulator to test the machines you designed in exercise 5.4.