Hand-Crafted Register Machines

1. What does the following register machine compute if we assume its input is in register $x$, and its output is in register $y$?

   (registers $x$ $y$ $t$)
   (operations $*$ $+$)
   (controller
     (assign $y$ (const 5))
     (assign $t$ (op $*$) (reg $x$) (const 7))
     (assign $y$ (op $+$) (reg $t$) (reg $y$))
     (assign $t$ (op $*$) (reg $x$) (reg $x$))
     (assign $t$ (op $*$) (reg $t$) (const 3))
     (assign $y$ (op $+$) (reg $t$) (reg $y$)))

2. Draw data path and control diagrams for this machine.
3. Suppose the machine above computes $f(x)$. Modify it so that it computes $f(1) + f(2) + \ldots + f(k)$, where $k$ is an additional input register and leaves its answer in $y$.

(registers x y t)
(operations * +)
(controller

(assign y (const 5))
(assign t (op *) (reg x) (const 7))
(assign y (op +) (reg t) (reg y))
(assign t (op *) (reg x) (reg x))
(assign t (op *) (reg t) (const 3))
(assign y (op +) (reg t) (reg y))

)
4. Given the machine which computes \( f(x) \), write a new machine which has inputs in a and b registers and computes \( f(a)/f(b) \), leaving the result in y. You may also use a continue register and a stack.

(registers x y t a b continue)
(operations * +)
(controller

(assign y (const 5))
(assign t (op *) (reg x) (const 7))
(assign y (op +) (reg t) (reg y))
(assign t (op *) (reg x) (reg x))
(assign t (op *) (reg t) (const 3))
(assign y (op +) (reg t) (reg y))

)

Register Machine Diagrams

Early in the semester, we saw some Scheme code to calculate square root using Newton’s method:

(define (sqrt x)
  (define (good-enough? ..))
  (define (improve ..))
  (define (sqrt-iter guess)
    (if (good-enough? guess)
        guess
        (sqrt-iter (improve guess)))))
  (sqrt-iter 1))
5. Define a register machine for this calculation. Assume for now that `good-enough?` and `improve` are primitives in the machine.

\[
\begin{align*}
&\text{(registers)} \\
&\text{(operations)} \\
&\text{(controller)}
\end{align*}
\]

6. Draw the data flow and control flow diagrams for your machine.

7. Update your machine given that `improve` is defined within `sqrt` as

\[
\begin{align*}
&\text{(define (improve guess)} \\
&\quad (\text{average guess} (/ x guess)))
\end{align*}
\]
Stacks, Iteration, Recursion, Calling Conventions

8. Specify a register machine to implement the following iterative numeric process:

```
(define (explode b n)
  (define (ex-iter base counter product)
    (if (= counter 0)
      product
      (ex-iter (* base base)
        (- count 1)
        (* base product)))))
  (ex-iter b n 1))
```
Implementation of recursive processes requires some way to “remember” two important things. First, we must keep track of how deep into the recursion we are and where we are supposed to return to each step of the way. Second, we must remember the intermediate state of a computation so that, upon return from the recursive call, the deferred operations can be completed. This leads us to one useful calling convention governing the use of a stack. We can use a continue register to indicate where to go when done, and use the stack to keep track of our recursion. In this convention, the calling routine saves away the registers it will need in order to complete the computation upon return.

9. Specify a register machine to implement the following recursive explode process:

```scheme
(define (explode b n)
  (if (= n 0)
      1
      (* b (explode (* b b)
                   (- n 1)))))
```

Trace the values in the registers and stack for `(explode 3 2)`.