Models of Computation II, Exercises 1: Register Machines

1. (a) The graphical representation looks like:

```
START → R₁⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻ нарко
```

It is very easy to forget to label the start state, but it is essential to do so — otherwise how would you know where to begin? **Always label the start state!**

(b) The computation is: (0,0,2,0,0), (1,0,1,0,0), (2,1,1,0,0), (5,1,1,0,0), (6,1,1,1,0), (0,1,1,1,0), (1,1,0,1,0), (2,2,0,1,0), (3,2,0,0,0), (4,2,0,0,1), (1,3,0,0,1), (2,4,0,0,1), (5,4,0,0,1), (6,4,0,1,1), (5,4,0,1,0), (6,4,0,2,0), (0,4,0,2,0), (7,4,0,2,0).

This register machine computes the sum of the first \( x \) odd numbers, this is equivalent to \( x^2 \), so:

\[
f(x) = \sum_{k=0}^{x-1} (1 + 2k) = x^2
\]

Register \( R_0 \) is used for the accumulator and final result, \( R_1 \) is the input and used for termination of the machine, \( R_2 \) is used for the loop that calculates \( 2k \), and \( R_3 \) stores a copy of \( R_2 \) whilst it is destructively used by the loop.

States \( L_1 \) to \( L_4 \) compute \( 1 + 2k \) and copies \( R_2 \) into \( R_3 \) whilst \( R_2 \) is decremented. \( L_5 \) and \( L_6 \) moves \( R_3 \) back into \( R_2 \) and increments the value of \( R_2 \) by 1 (this is equivalent to the \( \Sigma \) operation incrementing \( k \) for the next addition).

2. (a) i. The following register machine computes \( f \):

\[
L_0 : R_2^- \rightarrow L_1, L_2
L_1 : R_1^- \rightarrow L_0, L_4
L_2 : R_1^- \rightarrow L_3, L_4
L_3 : R_0^+ \rightarrow L_2
L_4 : HALT
\]

The machine first reduces \( R_1 \) (which has initial value \( x_1 \)) by the amount in \( R_2 \) (initially \( x_2 \)): this is the loop between instructions \( L_0 \) and \( L_1 \). If it cannot reduce
that far, it means $x_2 > x_1$, and so the machine halts at $L_4$, with $R_0$ still at its initial value of 0 (which is $f(x_1, x_2)$). If $R_1$ can be reduced that far, its contents is then $x_1 - x_2$, which is copied into $R_0$ by the loop between $L_2$ and $L_3$. When this loop exits, the machine will halt with $R_0 = x_1 - x_2 = f(x_1, x_2)$.

ii. Graphically, the register machine looks like:

(Variations are possible, but this is the simplest that always halts successfully.)

(b) i. Recall what it means for register machine $M$ to compute $g(x_1, x_2)$:

The computation of $M$ starting with $R_0 = 0$, $R_1 = x_1$, $R_2 = x_2$ and all other registers set to 0 halts with $R_0 = y$ if and only if $g(x_1, x_2) = y$.

Since $g(x_1, x_2)$ is undefined when $x_2 > x_1$, there is no $y$ with $g(x_1, x_2) = y$. Therefore the machine cannot halt. Instead, it must run forever.

ii. The simplest way to make the machine run forever if $x_2 > x_1$ is to have $L_1$ loop back on itself when $R_1 = 0$:

3. (a) One possible coding is the following:

Other codings are possible by renaming $L_1$, $L_2$ and $L_3$ consistently. $L_0$ cannot be renamed, because it is the start state.
(b) The machine computes the remainder of $x$ divided by 2. That is, the function

$$f(x) \equiv \begin{cases} 
0 & \text{if } x \text{ is even} \\
1 & \text{if } x \text{ is odd} 
\end{cases}$$

To see why, consider that whenever the machine is in state $L_0$, register $R_1$ has been decreased by an even amount from its initial value of $x$. (At the start, $R_1$ has been decreased by 0, which is even.) If $R_1 = 0$, it must be that $x$ was even, and so the machine halts with $R_0 = 0$. Whenever the machine is in state $L_1$, register $R_1$ has been decreased by an odd amount from its initial value. Therefore, if $R_1 = 0$ it must be that $x$ was odd, so by incrementing $R_0$ then halting the machine halts with $R_0 = 1$. If $R_1 > 0$ in state $L_0$, it can be decremented by 1 so that it has been decremented an odd number of times when the machine enters state $L_1$. Similarly, if $R_1 > 0$ in state $L_1$, it can be decremented by 1 so that it has been decremented an even number of times when the machine enters state $L_0$. Finally, since $R_1$ is decreased on every loop, we can conclude that the machine always halts eventually.
4. (a) add $R_1$ to $R_2$:

(b) It would be enough for us to add $R_1$ to $R_2$ if $R_2$ was set to 0. So let’s zero $R_2$ first! copy $R_1$ to $R_2$:

(c) We can implement multiplication by repeated addition. multiply $R_1$ by $R_2$ to $R_0$:

(d) test $R_1 < R_2$:
(e) The register machine computes the greatest value $f(x)$ such that $(f(x))^2 \leq x$. That is, it computes the floor of the positive square-root of $x$: $f(x) = \lfloor \sqrt{x} \rfloor$. The machine loops testing whether $(1 + R_0)^2$ is greater than $R_1$, incrementing $R_0$ until it is.