Exercise 1. [URM decidability]
Write a URM program that decides the predicate $M(x, y)$="x divides y".[1]

Example solution
We start with x and y in registers 1 and 2, respectively. The idea is to run a counter k in register 3, and simultaneously keep track of the corresponding multiple of x, namely $x \cdot k$, in register 4. So, after $k \in \mathbb{N}$ iterations of the main routine, the memory will be in the following configuration:

$$\begin{array}{cccc} R_1 & R_2 & R_3 & R_4 \\ x & y & k & k \cdot x \end{array}$$

Now, if x is a divisor of y, then $k \cdot x$ will hit the number y before k does, or at the same time (that happens if $x = 1$). On the other hand, if k hits the number y while $k \cdot x$ has not yet hit y, then we can be sure that y is not a multiple of x. This justifies the following procedure to determine whether x divides y:

1 $\quad R_2 \stackrel{?}{=} R_4 \xrightarrow{\text{yes}}$ output 1

$\downarrow$no

2 $\quad R_2 \stackrel{?}{=} R_3 \xrightarrow{\text{yes}}$ output 0

$\downarrow$no

3 $\quad R_3 := R_3 + 1$

4 $\quad R_4 := R_4 + R_1$

5 $\quad$ back to step 1

This diagram gives the plan for the program that we want to write. However, our repertoire of URM-instructions does not allow us to express the instruction $R_4 := R_4 + R_1$ in a direct way. Instead, to achieve this we need to add to the program a sub-routine that only makes use of increments by 1. With this addition, the plan for our procedure becomes as follows:

[1] Just to be precise: we say that x divides y if there is a number $k \in \mathbb{N}$ such that $x = y \cdot k$. In particular, every number divides 0, while 0 divides only itself.

1	$R_2 \stackrel{?}{=} R_4$	$\xrightarrow{\text{yes}}$	output 1
	$\downarrow$no		
2	$R_2 \stackrel{?}{=} R_3$	$\xrightarrow{\text{yes}}$	output 0
	$\downarrow$no		
3	$R_3 := R_3 + 1$		
4	$R_5 := 0$		
5	$R_1 \stackrel{?}{=} R_5$	$\xrightarrow{\text{yes}}$	back to step 1
	$\downarrow$no		
6	$R_4 := R_4 + 1$		
7	$R_5 := R_5 + 1$		
8	back to step 5		

It is now a small step to turn this plan into a URM program. The result is the following:

1. J(2,4,9)
2. J(2,3,12)
3. S(3)
4. Z(5)
5. J(1,5,1)
6. S(4)
7. S(5)
8. J(1,1,5)
9. Z(1)
10. S(1)
11. J(1,1,99)
12. Z(1)

Exercise 2. [Unlimited register machines]
Write a URM program that computes the Fibonacci function, defined as follows:

- $F(0) = 1$
- $F(1) = 1$
- $F(n+2) = F(n) + F(n+1)$

Solution. To compute the number $F(x)$, we can use the following procedure. For $k = 0, 1, 2, \ldots$, we run through the following memory configurations.

R_1	R_2	R_3	R_4	R_5
x	$F(k)$	$F(k+1)$	$F(k+1)$	k

The idea of the procedure is captured by the following diagram. Notice that the first three instructions just have the role of setting the initial content of R_2, R_3 and R_4 to 1, since in the initial state, when $k = 0$, we have $F(0) = F(1) = 1$.

$$
\begin{array}{lll}
1 & R_2 := R_2 + 1 & \\
2 & R_3 := R_3 + 1 & \\
3 & R_4 := R_4 + 1 & \\
4 & R_1 \stackrel{?}{=} R_5 \quad \xrightarrow{\text{yes}} & R_1 := R_2 \text{ and stop} \\
 & \downarrow\text{no} & \\
5 & R_5 := R_5 + 1 & \\
6 & R_4 := R_2 + R_3 & \\
7 & R_2 := R_3 & \\
8 & R_3 := R_4 & \\
9 & \text{back to step 4} &
\end{array}
$$

The reason that we keep $F(k+1)$ stored twice in the memory, is that when k is incremented to $k+1$, the value $F(k+1)$ in cell 5 is transformed into $F(k+2)$. However, we still need to keep track of the the value of $F(k+1)$ in the memory, since this value is needed later to compute $F(k+3) = F(k+1) + F(k+2)$.

As usual, since we don't have a direct instruction for computing binary sum in the URM language, step 3 of the above procedure needs to be implemented by means of a sub-routine which uses a counter h in register 6. Keeping this in mind, we can write the above procedure as the following URM program, where instructions 6-10 implement the subroutine for $R_4 := R_2 + R_3$.

1. S(2)
2. S(3)
3. S(4)
4. J(1,5, 14)
5. S(5)
6. Z(6)
7. J(2,6,11)
8. S(6)
9. S(4)
10. J(1,1,7)
11. T(3,2)
12. T(4,3)
13. J(1,1,4)
14. T(2,1)