

## Models of Computation II, Exercises 1: Register Machines

1. Consider the register machine given by the following code:

$$\begin{aligned}
 L_0 &: R_1^- \rightarrow L_1, L_7 \\
 L_1 &: R_0^+ \rightarrow L_2 \\
 L_2 &: R_2^- \rightarrow L_3, L_5 \\
 L_3 &: R_3^+ \rightarrow L_4 \\
 L_4 &: R_0^+ \rightarrow L_1 \\
 L_5 &: R_2^+ \rightarrow L_6 \\
 L_6 &: R_3^- \rightarrow L_5, L_0 \\
 L_7 &: HALT
 \end{aligned}$$

- (a) Give the graphical representation of the register machine.
- (b) Give the computation (that is, the sequence of configurations) when the register machine is run from the initial configuration  $(0, 0, 2, 0, 0)$ .

2. In this question, you will design register machines that implement subtraction.

- (a) Consider the function  $f(x_1, x_2)$  defined as

$$f(x_1, x_2) \stackrel{\text{def}}{=} \begin{cases} x_1 - x_2 & \text{if } x_1 \geq x_2 \\ 0 & \text{otherwise} \end{cases}$$

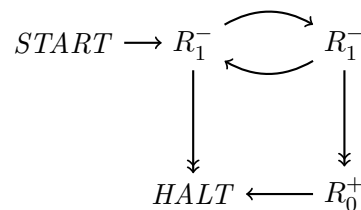
- i. Define a register machine that computes the function  $f$ .
- ii. Draw the graph corresponding to the register machine.

- (b) Consider the partial function  $g(x_1, x_2)$  defined as

$$g(x_1, x_2) \stackrel{\text{def}}{=} \begin{cases} x_1 - x_2 & \text{if } x_1 \geq x_2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- i. How would a register machine implementing  $g(x_1, x_2)$  behave when  $x_2 > x_1$ ? (Hint: consider the definition on Slide 14 of the notes carefully.)
- ii. By adapting your answer to part (a) (or otherwise), define a register machine that computes the partial function  $g$ .

3. Consider the register machine represented by the following graph:

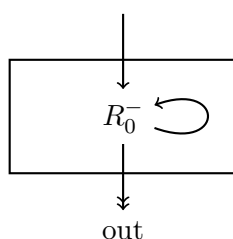


- (a) Give the code of the register machine. (Note, there is more than one way to do this, depending on how you label the states.)
- (b) Describe the function of one argument,  $f(x)$ , that is computed by the register machine.

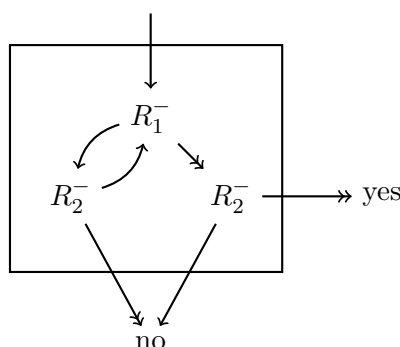
4. In order to construct register machines to perform complex operations, it is useful to build them from smaller components that we'll call *gadgets*, which perform specific operations.

A gadget will be defined by a (partial) register machine graph that has a designated initial label and one or more designated exit labels (which contain no instructions). The gadget will operate on registers that are specified in the gadget's name, and are used for input and output — we call these the input/output registers. The gadget may use other registers for temporary storage — we call these scratch registers. The gadget may assume the scratch registers are initially set to 0, and *must* ensure that they are set back to 0 when the gadget exits. (Ensuring that the scratch registers are reset to 0 is important so that the gadget may be safely used within loops.)

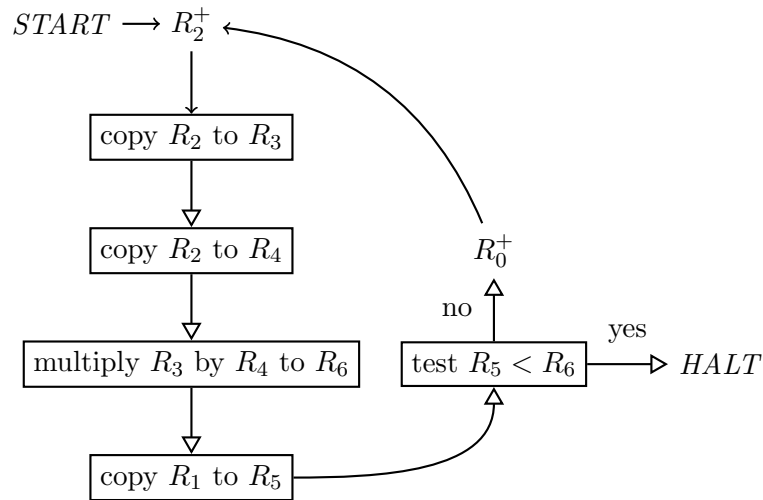
The gadget “zero  $R_0$ ” sets register  $R_0$  to be zero, whatever its initial value. It is defined by the graph:



A slightly more complicated example is the gadget “test  $R_1 = R_2$ ”, which determines whether the initial values of  $R_1$  and  $R_2$  are equal, possibly overwriting their values in the process. The gadget has two exit labels: “yes” and “no”. The gadget exits to “yes” if the values are equal and “no” if they are unequal. It is defined by the graph:



We can use gadgets to help us construct other gadgets and complete register machines. For instance we can construct the following register machine,  $M$ , using gadgets for copy, multiply and test for  $<$ :



When we use gadgets this way, we draw the links out of them like  $\rightarrow\triangleright$ . If there is more than one exit point from the gadget, we label these links accordingly. Each of these links can stand for any number of actual edges out of the gadget (for example, with the “test  $R_1 = R_2$ ”, the “no” link would correspond to both of the “no” edges).

In constructing such machines, we rename the registers used by each gadget: all of its scratch registers should be renamed to things that do not occur in the rest of the machine, and its input/output registers should be renamed to whichever registers the program requires. As an example, suppose we defined the gadget “copy  $R_1$  to  $R_2$ ” using a  $R_3$  as a scratch register. To construct the instance “copy  $R_2$  to  $R_3$ ” that is used in  $M$ , we would rename  $R_3$  in the gadget to something fresh, say  $R_7$ , and rename  $R_1$  and  $R_2$  to  $R_2$  and  $R_3$  respectively. We would then wire the output edge of the  $R_2^+$  instruction to the input location in the gadget instance, and all of the output edges of the gadget instance to the input instruction of a second instance of the copy gadget corresponding to “copy  $R_2$  to  $R_4$ ”.

- (a) Define a gadget “add  $R_1$  to  $R_2$ ” which adds the initial value of  $R_1$  to register  $R_2$ , storing the result in  $R_2$  but restoring  $R_1$  to its initial value. That is, if the initial state is  $R_1 = r_1$  and  $R_2 = r_2$  then the final state will be  $R_1 = r_1$  and  $R_2 = r_1 + r_2$ . (You will need to use a scratch register to restore  $R_1$  to its initial value. Remember: you can assume the scratch register initially has value 0, but must ensure that it also have value 0 when the gadget exits.)
- (b) Define a gadget “copy  $R_1$  to  $R_2$ ” which copies the value of  $R_1$  into register  $R_2$ , leaving  $R_1$  with its initial value. You may construct this gadget from gadgets that have already been defined.
- (c) Define a gadget “multiply  $R_1$  by  $R_2$  to  $R_0$ ” which multiplies  $R_1$  by  $R_2$  and stores the result in  $R_0$ , possibly overwriting the initial values of  $R_1$  and  $R_2$ . Again, you may use already-defined gadgets in your definition.
- (d) Define a gadget “test  $R_1 < R_2$ ” which determines whether the initial value of  $R_1$  is less than that of  $R_2$ , possibly overwriting their values in the process. The gadget should exit to “yes” if it is, and “no” otherwise.

- (e) Describe the function of one argument  $f(x)$  computed by the register machine  $M$  defined above.