Tutorial 7

Exercise 1: Simulate the computation of the following program for RAM that obtains a sequence (of length 1) consisting a single number 4 as an input, i.e., write down a sequence of configurations through which the RAM goes during this computation:

$$\begin{array}{rl}
0: & R_2 := \text{READ}() \\
1: & R_1 := 2 \\
2: & \textbf{goto } 5 \\
3: & R_1 := R_1 * R_1 \\
4: & R_2 := R_2 - 1 \\
5: & \textbf{if } (R_2 > 0) \textbf{ goto } 3 \\
6: & \text{WRITE}(R_1) \\
7: & \textbf{halt}
\end{array}$$

Determine what is computed by this program, i.e., what it will produce as an output when it obtains a number n as an input.

Exercise 2: Determine what the following program for RAM computes, i.e., describe in detail its behaviour for an arbitrary input and describe what will be on the output.

Remark: For clarity, addresses of instructions are not given explicitly, symbolic labels are used instead.

$$\begin{array}{rl}
 & R_4 := 4 \\
 & R_3 := \text{READ}() \\
 & R_1 := R_4 + R_3 \\
 & R_0 := 0 \\
L_1: & \textbf{if } (R_1 = R_4) \textbf{ goto } L_2 \\
 & [R_1] := R_0 \\
 & R_1 := R_1 - 1 \\
 & \textbf{goto } L_1 \\
L_2: & R_2 := \text{READ}() \\
 & \textbf{if } (R_2 \leq 0) \textbf{ goto } L_3 \\
 & \textbf{if } (R_2 > R_3) \textbf{ goto } L_3 \\
 & R_1 := R_4 + R_2
\end{array}
\qquad
\begin{array}{rl}
 & R_0 := [R_1] \\
 & R_0 := R_0 + 1 \\
 & [R_1] := R_0 \\
 & \textbf{goto } L_2 \\
L_3: & R_2 := 1 \\
L_4: & \textbf{if } (R_2 > R_3) \textbf{ goto } L_5 \\
 & R_1 := R_4 + R_2 \\
 & R_0 := [R_1] \\
 & \text{WRITE}(R_0) \\
 & R_2 := R_2 + 1 \\
 & \textbf{goto } L_4 \\
L_5: & \textbf{halt}
\end{array}$$

Exercise 3: For each of the following problems, design a program for a RAM solving the given problem.

Remark: It is not necessary to deal with wrong data on input that do not correspond to specifications of inputs.

a) INPUT: integers x, y (i.e., $x, y \in \mathbb{Z}$)
OUTPUT: value $x + y$

b) INPUT: integers x, y (i.e., $x, y \in \mathbb{Z}$)

OUTPUT: $\max\{x, y\}$

c) INPUT: natural number n (i.e., $n \in \mathbb{N}$)
OUTPUT: sequence of numbers $1, 2, \ldots, n$

Remark: The sequence on output will be empty for $n = 0$.

d) INPUT: sequence of numbers $a_1, a_2, \ldots, a_n, 0$, where $n \geq 0$ and $a_i \in \mathbb{Z} - \{0\}$ for $1 \leq i \leq n$
OUTPUT: $\prod_{i=1}^{n} a_i$

Remark: Notation $\prod_{i=1}^{n} a_i$ denotes the product $a_1 \cdot a_2 \cdot \ldots \cdot a_n$. For $n = 0$, the output will be 1.

e) INPUT: sequence of numbers $a_1, a_2, \ldots, a_n, 0$, where $n \geq 0$ and $a_i \in \mathbb{Z} - \{0\}$ for $1 \leq i \leq n$
OUTPUT: sequence of numbers $a_n, a_{n-1}, \ldots, a_1$

Exercise 4: Construct a program for a RAM that reads a number n from the input and writes the n-th Fibonacci number on the output. You can assume that the number n on the input is nonnegative (i.e., you don't have to consider the situation when $n < 0$). Recall that Fibonacci numbers $F_0, F_1, F_2, \ldots$ are defined by the following recurrence relation:

$$F_n = \begin{cases} 0 & \text{for } n = 0 \\ 1 & \text{for } n = 1 \\ F_{n-1} + F_{n-2} & \text{for } n > 1 \end{cases}$$

Exercise 5: Consider the following Algorithm 1. The input of this algorithm can be an arbitrary natural number n (i.e., values of variable n can be arbitrarily big natural numbers).

Algorithm 1:

```
PrintSeq(n):
    print n
    while n > 1 do
        if n mod 2 = 0 then
            n := n / 2
        else
            n := 3 * n + 1
        print n
```

a) Draw the control-flow graph of this algorithm.

b) Describe a computation performed by this algorithm when it gets number 5 as an input. Write down the sequence of configurations in this computation.

c) How many steps are performed by the algorithm when the input is number 7? What will be the output in this case?

d) Construct a program for RAM implementing this algorithm.

Remark: This program for RAM should read the value of value of variable n from the input.

Exercise 6: Consider Algorithm 2 described by a pseudocode.

Algorithm 2: Insertion sort

```
INSERTION-SORT ():
    n := READ ()
    for i := 0 to n − 1 do
        A[i] := READ ()
    for j := 1 to n − 1 do
        x := A[j]
        i := j − 1
        while i ≥ 0 and A[i] > x do
            A[i + 1] := A[i]
            i := i − 1
        A[i + 1] := x
    for i := 0 to n − 1 do
        WRITE (A[i])
```

a) Draw a control-flow graph representing this pseudocode.

b) Implement this algorithm as a program for RAM.

c) Describe how different parts of your program for RAM correspond to edges of the control-flow graph.

Exercise 7: Describe how to construct, for an arbitrary Turing machine $\mathcal{M}$, a program for RAM that performs the same algorithm as machine $\mathcal{M}$. Consider the following variants of Turing machines:

a) A Turing machine with one tape infinite on one side

b) A Turing machine with one tape infinite on both sides

c) A Turing machine with several tapes infinite on both sides

Remark: It is not necessary to explicitly construct these programs for RAM. It sufficient to describe informally the behaviour of these machines.

Exercise 8: Construct a program for a RAM that reads two numbers x and k from the input a writes the value of k-th bit of the number x (i.e., 0 or 1) on the output. The bits are numbered starting from 0 and 0-th bit is the least significant bit. You can assume that $x \geq 0$ and $k \geq 0$ (i.e., you don't have to consider the cases when $x < 0$ or $k < 0$).

Exercise 9: Construct a program for a RAM that reads two numbers x and y from the input (you can assume that $x \geq 0$ and $y \geq 0$) and writes their product $x \cdot y$ on the output. To make this job more difficult, you must conform to the following restrictions:

- In your program, you ***must not*** use arithmetic instructions for multiplication and division. However, you can use the following arithmetic instruction that implements a shift to the right by one bit:

  $$R_i := \mathit{rshift}(R_j)$$

  This instruction basically does exactly the same thing as the following intruction: $R_i := \lfloor R_j / 2 \rfloor$.

- The total number of instructions performed by your program must be polynomial with respect to the numbers of bits of numbers x and y.

- Do you have some idea how to solve this problem without instructions of the form $R_i := \mathit{rshift}(R_j)$, i.e., how to compute the value $x \cdot y$ on a RAM that can use only addition and subtracktion as arithmetic operations in such a way that the total number of steps is polynomial with respect the number of bits of numbers x and y?