Solution for 'registers'

1 Subtask 1

The key is to set up the following procedure:

- Given 3 registers r_a, r_b, r_c, assign $r_c = r_a$ if $r_a = r_b$, or 0 otherwise. (Here we slightly abuse the notation r_a, r_b, r_c to refer to either the register itself or the value stored inside).

To do so, take a new register r_d and set $r_d = r_a$ XOR r_b. Now, if $r_a = r_b$, r_d stores all 0, else at least one bit of r_d is 1. We take r_d OR (r_d shifted left) a total of 4 times, and then take r_d OR (r_d shifted right) a total of 4 times.

Once we do this, r_d has its 4 least significant bits all 0 if $r_a = r_b$, or all 1 otherwise. One we do this, we can set r_c = (NOT (r_d)) AND (r_a).

As there are only 16 possible inputs, we can encode all 16 cases.

2 Subtask 2

We start with the operations `and(1,0,k)`, `not(2,1)`, `add(3,0,2)`. After these instructions are executed, $r[3][k]$ will tell us which of $r[0]$ and $r[1]$ is larger.

From here, a similar branching approach will enable us to solve the problem.

3 Subtask 3

Proceed as with the earlier subtask. Repeatedly take the minimum between two numbers.

4 Subtask 4

Use a tournament structure and parallelize the operations.

5 Subtask 5

Use bubble sort idea to repeatedly bubble the smallest element to the bottom.

6 Subtask 6

Observe that bubble sort can be parallelized. Essentially, do the following:

Initially our register contains $a[5], a[4], a[3], a[2], a[1], a[0]$. Using the ideas from the above subtasks, we can modify the values to

$\max(a[5], a[4]), \min(a[5], a[4]), \max(a[3], a[2]), \min(a[3], a[2]), \max(a[1], a[0]), \min(a[1], a[0])$.

Afterwards, we reverse the parity (i.e. $a[4], a[3]$ now becomes $\max(a[4], a[3]), \min(a[4], a[3])$).

Repeat this for n times.