

# Challenges

## Scoring

Each challenge (bonuses included) final score is evaluated with the following table.

Work for all inputs	Resources used	Number of cycles	Total
bool	%	%	
80	10	10	100

That percentage is then multiplied to the question value to get a number of point. The final score is the sum of all those points.

Challenges will be tested with inputs different from the ones given as example in this document.

Objectives may be of different types:

- bool: You either get all the point or none at all.
- %: The best competitor for this challenge get all the points, the worst, none. Every one else score is given by a linear interpolation between the two extremes. You must succeed at boolean objectives before getting those points.

Definitions:

- Number of cycles: Number of cycles to get an output for all the test inputs.
- Resources used: (number of membanks) x (size of membanks) + cores

**Unless otherwise noted, programs must run in a loop.** That is, it must wait for the next input after one have been processed.

## 1. Value passing (10 pts)

### Restrictions

- Two inputs (a, b)
- Two outputs (A, B)

### Description

Move values from the two inputs to the two outputs (A=a, B=b).

### Example inputs

Inputs:

```
235 225
87 128
162 156
242 220
81 46
15 233
192 126
62 79
```

100 2  
140 65

Outputs:

235 225  
87 128  
162 156  
242 220  
81 46  
15 233  
192 126  
62 79  
100 2  
140 65

### Bonus (5 pts)

Move values from the two inputs to the two outputs but cross them so that  $A=b$ ,  $B=a$ .

## 2. Simple sum (10 pts)

### Restrictions

- Two inputs (a, b)
- One output (A)

### Description

Add values from the two inputs and return the result to the single output ( $A=a+b$ ). Output should wrap if it exceeds maximum.

### Example inputs

Input:

0 0  
0 1  
153 108  
250 224  
16 176  
109 18  
203 78  
143 19  
141 71  
228 145

Output:

0  
1  
5  
218

192  
127  
25  
162  
212  
117

### Bonus (10 pts)

Do the same but with 16 bits little endian numbers. You must therefore have four inputs (a, b, c, d) and 2 outputs such as:

```
result = b << 8 | a + d << 8 | c
A = result & 0xff
B = result >> 8
```

## 3. Conditional negation (20 pts)

### Restrictions

- Three inputs (a, b, c)
- One output (A)

### Description

Output A must be c when  $a < b$ , 0 when  $a = b$  and  $-c$  when  $a > b$ .

### Example inputs

Input:

232 4 35  
222 75 35  
48 48 35  
145 123 35  
71 248 35  
28 64 35  
86 251 35  
43 0 35  
252 196 35  
64 192 0

Output:

221  
221  
0  
221  
35  
35  
35  
221

221  
0

Note: Simulator does not know whether its output is a signed or unsigned number and therefore always display them as unsigned. For example, 221 is 0b11011101 which is -35 when interpreted as a signed 8 bits number.

#### Bonus (5 pts)

Output  $\max(a, b)$  when  $a = b$ .

### 4. Parallel base 2 logarithm (20 pts)

#### Restrictions

- Three inputs (a, b, c)
- Three outputs (A, B, C)

#### Description

Compute the base-2 logarithm for all inputs such as

```
A = floor(log2(a))  
B = floor(log2(b))  
C = floor(log2(c))
```

#### Example inputs

Input:

```
113 167 204  
21  57  26  
201 121 204
```

Output:

```
6 7 7  
4 5 4  
7 6 7
```

#### Bonus (5 pts)

Compute base-16 logarithms instead of base-2 logarithms.

### 5. Multiplication (30 pts)

#### Restrictions

- Two inputs (a, b)
- One output (A)

## Description

Perform  $A = a * b$ . Overflows must wrap around.

## Example inputs

Input:

```
0 3
0 0
6 8
3 32
79 39
66 56
```

Output:

```
0
0
48
96
21
126
```

## 6. Gaussian blur

### Restrictions

- 900 inputs (a)
- 900 outputs (A)

### Description

Given a 30x30 input image, output a 30x30 blurred image using the following kernel:

```
1/16 1/8 1/16
1/8 1/4 1/8
1/16 1/8 1/16
```

Such as

$$A_{i,j} = 1/16a_{i-1,j-1} + 1/8a_{i-1,j} + 1/16a_{i-1,j+1} + \\ 1/8a_{i,j-1} + 1/4a_{i,j} + 1/8a_{i,j+1} + \\ 1/16a_{i+1,j-1} + 1/8a_{i+1,j} + 1/16a_{i+1,j+1} +$$

This challenge is special, for two reasons:

1. It's the first challenge which you should probably solve using a 3D CPU. I suggest using a Zx30x30 CPU with all cores on the first Z-layer used as inputs and all cores on the last Z-layer used as outputs. Z is up to you. You may use a few intermediate z-layers to perform your calculations.
2. CPU configuration is tedious. Sadly, as today, the simulator does not support range input. Therefore, to use 900 inputs, you must list all of them. The same limitation applies to outputs and `core_to_mem`. Please, do not try to list everything by hand and instead use a tool like Python to generate those lists for you. Here are some examples to base your work on:

```
# Print a list of 900 elements in the format accepted by the LAVAL simulator
print(", ".join([str(i) for i in range(900)]))
```

```
# Print every cores ID part of the second column of the first layer
print(", ".join([str(i) for i in range(1, 30*30, 30)]))
```

### **Example inputs**

Too big to include here.