Report for the implementation of an ALU

Explanation for the control bits in the bitwise OR operation

According to figure 2 in the task description document, here are the control bits for the OR operation:

zx	nx	zy	ny	f	no
0	1	0	1	0	1

This means that the inputs x and y are negated so every bit is flipped. The function f = 0 means, out = x&y, or in this case NOT(x)&NOT(y). As the bit no is set, the output gets flipped once more. So, the output is NOT(NOT(x)&NOT(y)). Using De Morgans rule seen in formula 1 we already have the NOT-OR operation.

$$\overline{A} \cap \overline{B} = \overline{A \cup B} \tag{1}$$

Negated once more, the operation becomes an OR operation as can be seen in formula 2.

$$\overline{\overline{A} \cap \overline{B}} = A \cup B \tag{2}$$

This is how the OR operation in this ALU works.

Specification fulfilment

All the outputs requested in the task description were created using different instructions. For readability, all the control bits were packed to a parameter on the bus in the testbench as can be seen in figure 2.

For Testing, each test follows the same pattern. Setting inputs, set control bits, check output, display error if occurred, display detailed test result. As required, the finished test report also gets printed in the end. An example output is shown in figure 1.

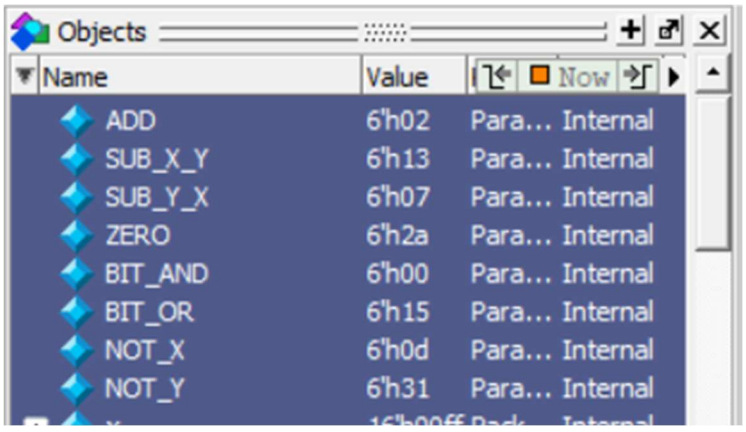

Figure 2: packed control bits

```
# Test nr:              8
# in x:               255
# in y:                 0
# out :              -256
# neg :                 1
# zer :                 0
# ----------------------
# Test nr:              9
# in x:               255
# in y:                 0
# out :              -255
# neg :                 1
# zer :                 0
# ----------------------
# tests :               9
# passed:               9
# error :               0
```

Figure 1 sample output of test procedure

As also requested, all the inputs and outputs are shown
as waveform. The waveform is shown in figure 3. The hex number on the control bus correlates with the local parameters defined earlier.

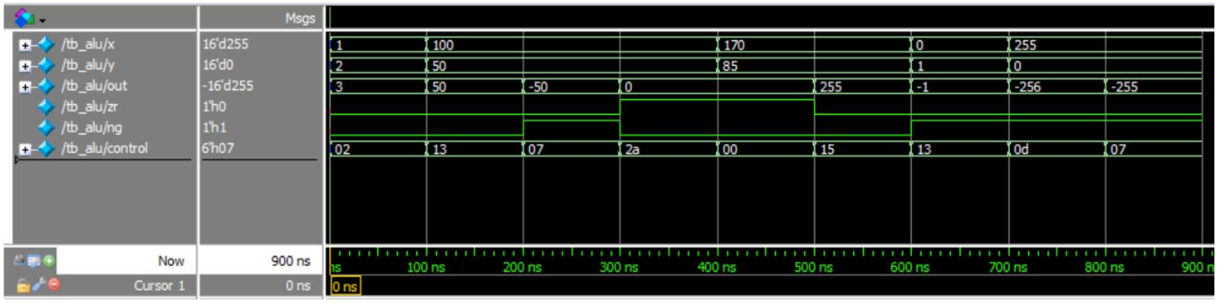

Figure 3: in- and outputs as waveform

Explanation for the implementation method used

The idea was to create multiple stages for the in- and outputs. The ALU consists of multiple multiplexers. Between those multiplexers there is always a new signal, whether implicit as multiple if/else commands or, as in this case, with multiple explicit stages for the in and outputs. The idea can be seen in figure 4.

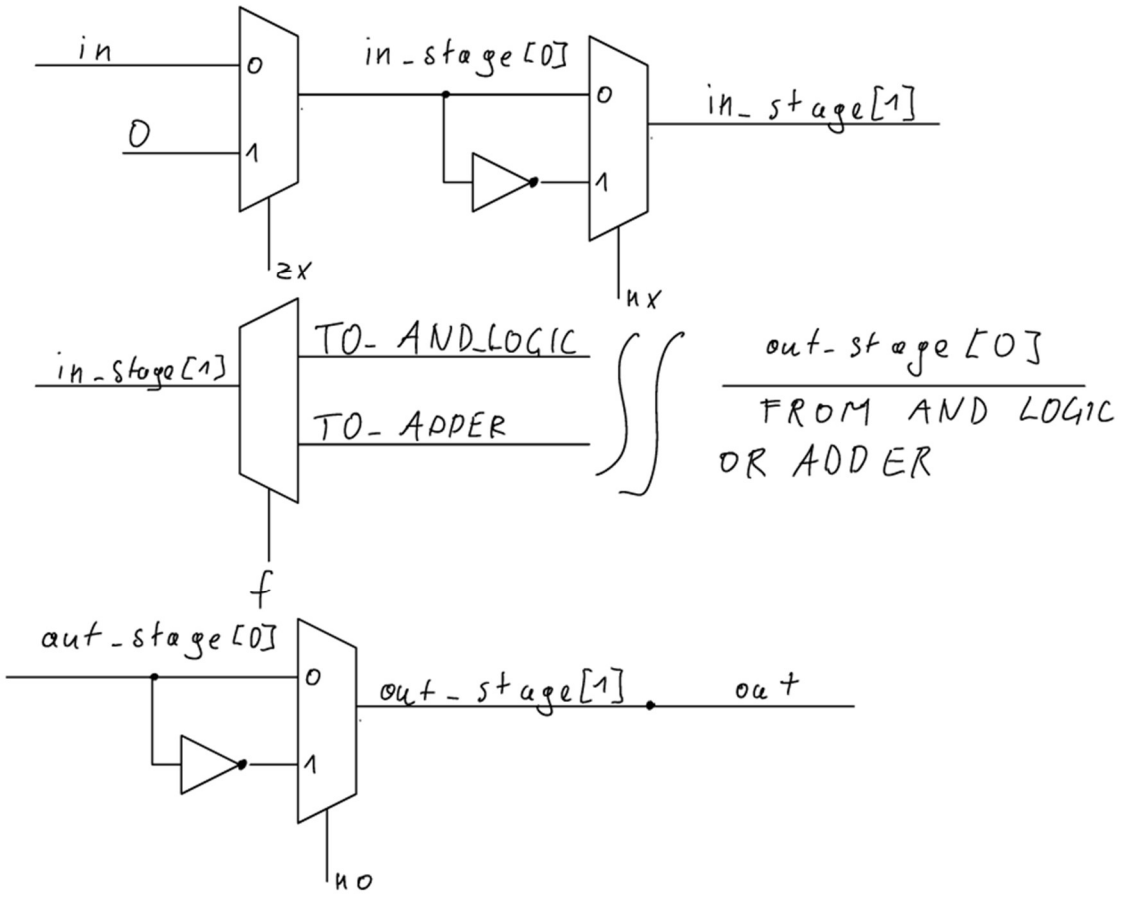

Figure 4: Implementation concept

Another, more complex method would be to create for every command a case expression and jump to the relevant action. As how this ALU was designed, the setting up the inputs and outputs is a much simpler way to implement the logic. Therefore, this way was chosen.