Objective:

The objective of this code was to compare a certain amount of 32-bit numbers to one 32-bit number and check which bit string has the most similar bit pattern.

Implementation:

Or truth table:

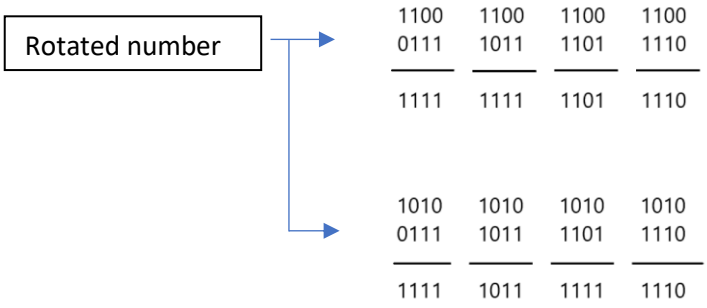
A	B	Output
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

01111111111111111111111111111111

The or functionality was used alongside a 32-bit string with mainly 1s and one 0 rotated around the string. When a number is or-ed with 1 the output is always going to be 1, when two opposing numbers are or-ed the answer will also be 1. However, if two 0s are or-ed then the answer will be 0. This understanding was used to check whether two bits were equal or not.

Eg. Compare two 4-bit numbers using the or method above.

The two numbers: 1100, 1010.
The string used to rotate a 0: 0111.



The purpose of rotating the 0 is because we only care about comparing 1 bit at a time. Using the 0 as a comparison makes it easy to identify whether the number being or-ed with it is a zero or a one. Since we do not care about the other bits then we or them with 1 and get back a value of 1 each time.
This is the main idea around the program.

As can be seen in the above example the rotating number initially starts with a 0 at the MSB position and ends at the LSB position. A comparison to a number placed in a register (0xfffffffe) was used to make sure that the bit does not keep rotating indefinitely.

As the program is only checking for consecutive bits that are the same when compared to another number a counter was used to keep track of how many consecutive bits were equal.

Check top and X: This loop was used to compare the top number with the base number X (base meaning the number all other numbers are being compared to). These two numbers were or-ed with the rotating bit string, then the results were compared with each other. If they were equal, then the program is sent to a loop "COUNTER1" which will rotate the 0 in the bit string by one. As rori is not available in the Nios II language a register was used to save a constant 1 so that it could emulate being rotated by an immediate. After the rotation is done a counter is incremented by 1 then there is an unconditional branch that goes back to the initial loop and the same procedure is repeated.
If the rotated bit string is equivalent to 0xfffffffe then it will also branch but not to the counter, it will branch to the next loop which will compare a different number.

This is repeated for the other two numbers being compared. However, in "CHECK_NUM2_AND_X" once it is done it branches to a method that will check the values in the registers.

To check which number had the most consecutive bits equal to number X cmpgeu was used. The destination register will either hold a 1 or a 0. 1 signifies that the operand on the left was either equal or greater than the operand on the right and 0 signifies the operand on the right was greater. Branches were used to check if the destination register held a 1 or 0, since r0 holds 0s only, then the values were compared to r0. Then the appropriate branch was selected and the registers in each branch would show which register had the larger counter. The counter was used to see which bit string had the largest number of consecutive bits.

Bugs:

A bug I encountered was the counter would add an extra one, I have not fully understood why yet.