Home Work 10. Branch Condition Evaluation Unit (BCE) and Shifter

For assignment 8 you are asked to write two separate modules. One for Branch Control Unit and the other for shifter.

Branch Condition Evaluation Unit Explanation

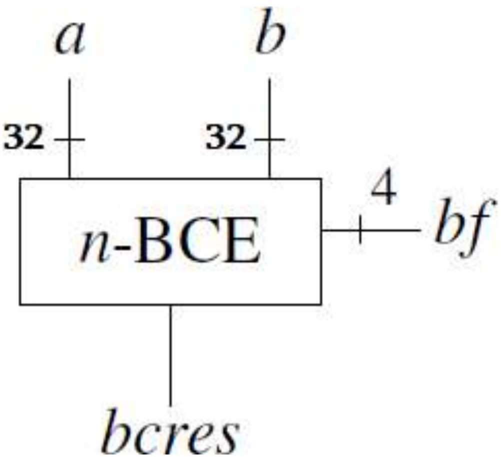


Figure 1

Figure 1 displays the block diagram of BCE. It evaluates if the branch condition is true. BCE tests for the following conditions: [a] < 0; [a] >= 0; a = b; a != b; [a] <= 0; [a] > 0. Take into account that [a] means two's complement representation. Whenever we compare a to 0, we should take the sign into account.

Inputs:

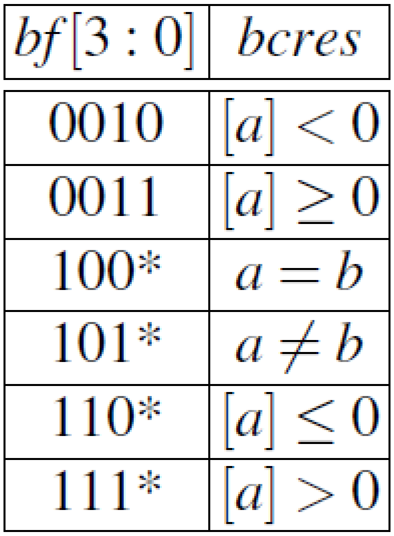
a – 32 bit long left test operand

b – 32 bit long right test operand

bf – 4 bit long input that determines the condition to be tested (See Table 1)

Outputs:

bcres – 1 bit long result. It is 1 if the condition is true, it is 0 if the condition is false.



$bf[3:0]$	$bcres$
0010	$[a] < 0$
0011	$[a] \geq 0$
100*	$a = b$
101*	$a \neq b$
110*	$[a] \leq 0$
111*	$[a] > 0$

Table 1

LAB TASK FOR BCE

1) **Implementation**: Write a Verilog Module For BCE
2) **Simulation & Verification**: Write testbench, generate waveforms and test BCE for every condition from table 1.

Shifter Explanation

There are 3 kinds of shift operation: Logical shift left, Logical shift right, and arithmetic shift left. The shifter module has 3 inputs and 1 output.

Inputs:

funct – 2 bit long input. Specifies type of shift (See Table 2)

a – 32 bit long input. Number to be shifted

N – 5 bit long input. Amount to shift

Output:

R – 32-bit long number obtained after the shift.

We use << operator for right shift and >> operator for left shift. However, both operators perform logical shifts. You may find online <<< and >>> operators for arithmetic shifts, but BEAR IN MIND that <<< and >>> are NOT SYNTHESIZABLE on the board. In order to implement Arithmetic shift right, we suggest using the following algorithm:

Algorithm for Arithmetic Right Shift:

Let's say we want to shift **a** by **N** times.

Step 1: Perform logical shift right to obtain the following:

$$\underline{0^{N}a[31:N]}$$

Step 2: Determine the sign bit of a:

$$S = a[31]$$

Step 3: Sign Extend S to obtain Sntd. Originally S is 1 bit long, extend it to 32 bit. (If you submitted Assignment 7, you already know how to do sign extension).

$$\text{Sntd}= \begin{cases} 2^{32}-1 & S = 1\ all\ 1s \\ 0 & S = 0\ \ all\ 0s \end{cases}$$

Step 4: Perform logical shift left on Sntd to obtain the following:

$$\text{Sntd}[N-1:0]0^{32-N}$$

Step 5: Perform bitwise OR operation on results obtained in Step 1 and Step 4

$$\underline{0^{N}a[31:N]\ \ V\ Sntd[N-1:0]0^{32-N}}$$

The result is the following:

$$R = \underline{Sntd[N-1:0]\ a[31:N]\ \ .}$$

Examples:

Let’s say number is 4’b1010 and N is 1.

In case of srl, we shift the number right once and fill the front with 0s:

Logical shift right (srl): R = 4’b0101

In case of sra, we shift the number right once and fill the front with sign bits (in our case sign bit is 1):

Arithmetic shift right (sra): R = 4’b1101

In case of sll, we shift the number left once and fill the tail with 0s:

number is 4’b1010

Logical shift left (sll): R = 4’b0100

Arithmetic shift left (sll): R = 4’b0100

There is no difference between arithmetic shift left and logical shift left.

LAB TASK FOR SHIFTER

1) **Implementation**: Write a Verilog Module For SHIFTER
2) **Simulation & Verification**: Write testbench, generate waveforms and test BCE for every condition from table 2.

Funct	Shift Type
00	shift left logical
10	shift right logical
11	shift right arithmetic

Table 2

We have seen several elegant modules of the previous assignments. If you would like to keep up ELEGANT work, our advice is to review Chapter 7.5 in Pr. Wolfgang's book (One that you used last semester). Especially, Pay close attention to Fig 45 and auxiliary signals in BCE.

It PLEASES the heart to see elegant modules. However, our hearts ache when we see reports written carelessly. Writing understandable reports is important for your future careers. That is why you are asked to USE THE TEMPLATE.

Good Luck