Extending The Count

This circuit gives us a set of operations to use. If we use mnemonics to represent those operations, they are:

- **LOAD X Y**: transfer Y’s value to X
- **ADD X Y Z**: calculate the sum of the values of X and Y, and store the result at Z.
- **NOT X Y**: negate X’s value, and store the result at Y
- **XOR X Y Z**: xor the values of X and Y, and store the result at Z
- **BRZ X**: conditional branch, branch to line X if the ALU result is zero
- **BRA X**: unconditional branch, branch to line X

We can then leverage the mnemonics to write an assembly program to get the sum of 5, 4, 3, 2, 1, and store the sum in RA.

- Initialize RB = 0 and RA = 5. Have a loop, in each iteration, the result of the sum of the values stored at RA and RB is assigned to RB, and then decrement RA’s value by 1. As long as the value in RA is not zero, the loop continues. Put the result to RA after the loop terminates.
If using pseudocode, we can have something like

```plaintext
0   RB <= 0
1   RA <= 5
2   while (RA != 0) {
3       RB <= RA + RB
4       RA <= RA - 1
5   }
6   RA <= RB
```

If using mnemonics to translate the above pseudocode into assembly program, we can have

```plaintext
0   LOAD RB 0 # initialize RB = 0
1   LOAD RA 5 # initialize RA = 5
2   ADD  RA RB RB # Add RA and RB and save the result to RB
3   ADD  RA -1 RA # RA <= RA - 1
4   BRZ  6 # If RA = 1, result of above line is 0. (ALU result is 0,
       CR=1) jump to line 6
5   BRA  2 # branch to line 2
6   LOAD  RA RB
```

We then can translate the above assembly program to machine instructions, which will be the program stored in Program Memory to calculate the sum of 5, 4, 3, 2, 1.

```plaintext
#  J1 J0  S1  S0  C1  C0  O0  D3  D2  D1  D0
0  0  0   1  0  0  1  1  d  d  d  d (way2: S1S0=01 D3-D0=0000)
1  0  0   0  1  0  0  1  0  1  0  1
2  0  0   0  0  0  0  1  d  d  d  d
3  0  0   0  1  0  0  0  1  1  1  1 (way2: S1S0=11 D3-D0=dddd)
4  1  0  d  d  d  d  d  0  1  1  0
5  1  1  d  d  d  d  d  0  0  1  0
6  0  0   0  0  1  0  d  d  d  d
```