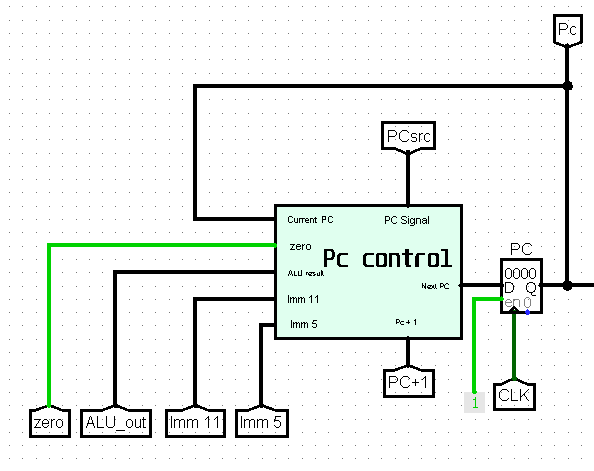
**Table of Contents:**

1. Introduction ………………………………………………..………. 1
2. Phase One Components ……………………………………….. 2
   1. PC Control ……………………………………………………….
      1. Branch Control ………………………………………
   2. Instruction Memory …………………………………………
   3. Register File……………………………………………………
   4. ALU ……………………………………………........................
   5. Extender …………………………………………………………..
   6. Data Memory …………………………………………………..
   7. Control Unit ……………………………………………………..
3. Testing ………………………………………………………………….
   1. Sum of an Array …………………………………………….
   2. Test Code 2024\_v2 ……………………………………….
   3. Test 0 …………………………………………………………….
   4. Test 1 ……………………………………………………………
   5. Multiplication and Applications Test …………….
      1. Multiplication Function …………………………
      2. Powers ………………………………………………….
      3. Product of an Array ………………………………

**2.1 PC Control:**

The Program Counter Control is the component which is responsible for generating the next PC value.



The PC Control Signal is called PC Src and it is used to decide how the next PC value will be calculated.

The Next PC Values:

1. PC + 1:

* This is the main value of the next PC.
* Used for most of the instruction.
* Calculated by adding 1 to the current PC address.
* Its PC Src signal is (xx00).

1. Reg (Rs):

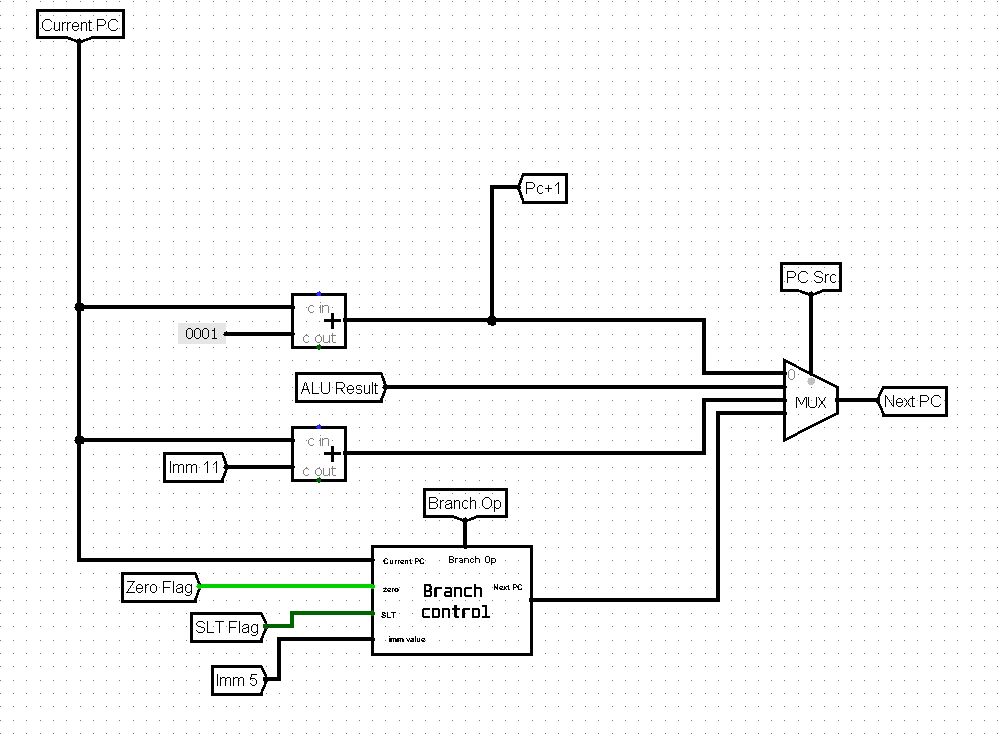
* Used for the Jump Register (JR) instruction.
* In JR instruction Rs is added to 0 and the Pc control takes the result as ALU result and stores that value as the new PC.

1. PC + sign-extend (Imm11):

* Used for the Jump/Jump and Link (J/JAL) instructions.
* The new pc value is the sign extended value of Imm(11) added to the current PC address.

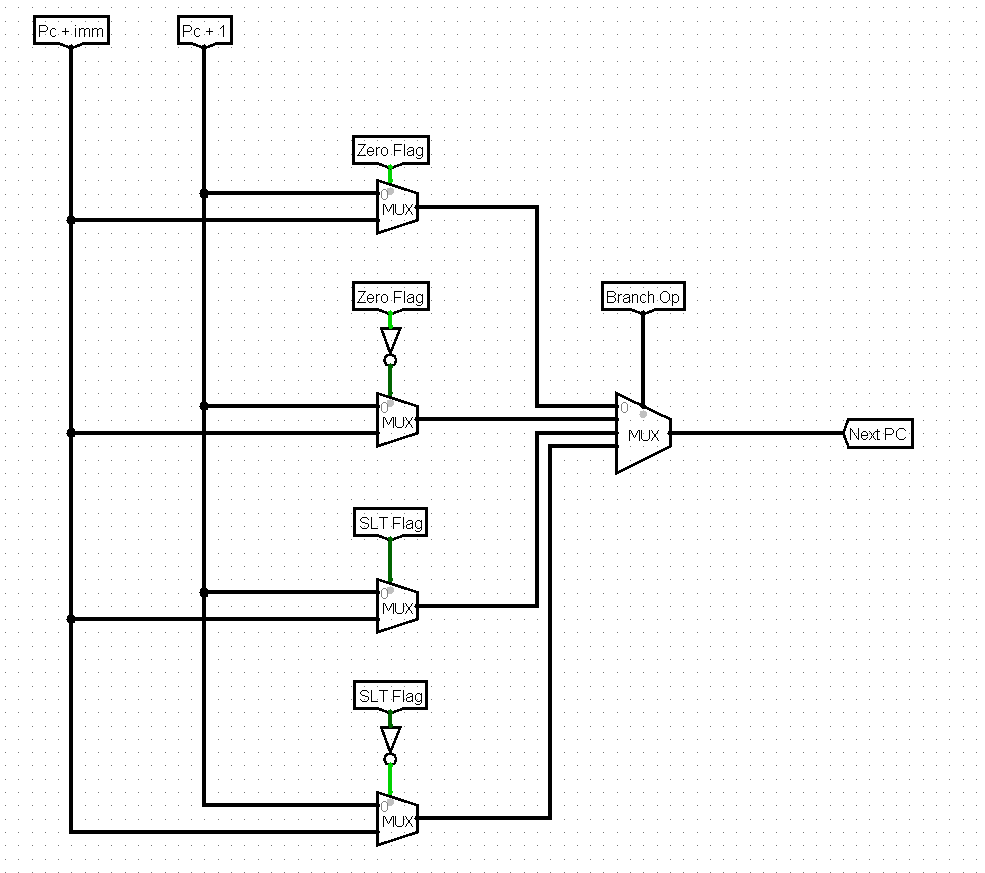
1. PC + sign-extend (Imm5):

* Used for the branch (BEQ/BNE/BLT/BGE) instructions.
* The new pc value is the sign extended value of Imm(5) added to the current PC address.



**2.1.1 Branch Control:**

In the case of a branch instruction the next PC value is either PC + 1 or PC + sign-extend (Imm5). The Branch Control unit uses the zero flag to check for equality and inequality and the SLT flag to check for less than and greater or equal conditions.



**3 Testing:**

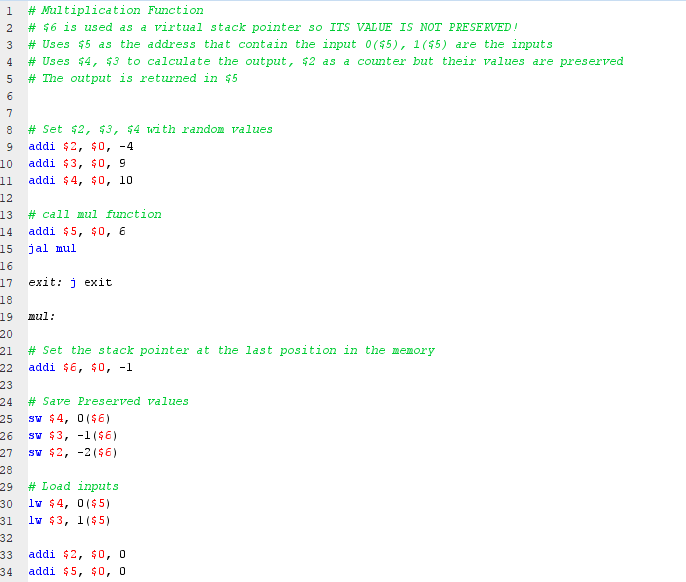
**3.5 Multiplication and Applications Test:**

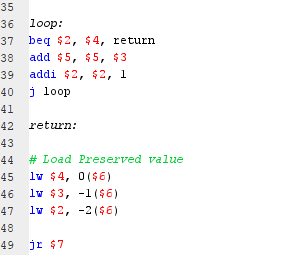
**3.5.1 Multiplication Function:**

The objective of the implementation of the multiplication function is to be able to use it in in larger applications.

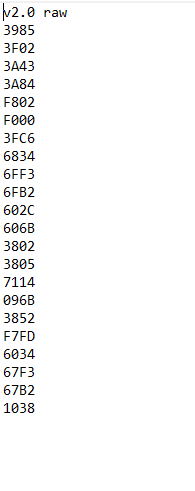
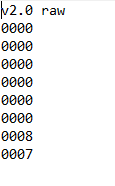
To achieve that we need to use a little of registers as possible. So we will need a virtual Stack.

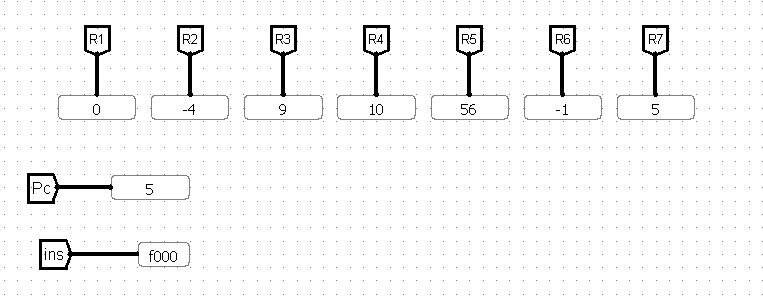
The input of the function is in $5 and it will contain the address that contain the inputs in memory 0($5), 1($5) are the inputs. Uses $4, $3 to calculate the output, $2 as a counter but their values are preserved at the last 3 places in the memory each time you call the function. $6 will be used as the stack pointer only in this function so ITS VALUE IS NOT PRESERVED!

**Assembly Code:**



**Converted Instructions:**  **Memory Input:**

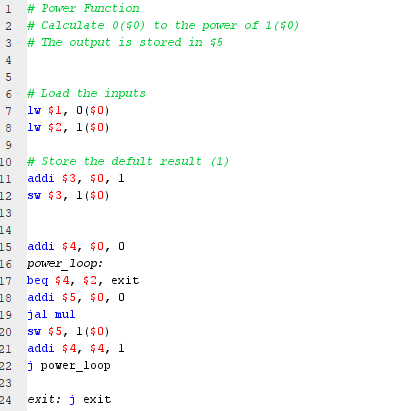


**Output:** (8 x 7 =56) stored in R5 and the values of R2, R3, and R4 are preserved.

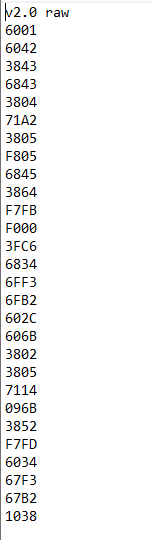
**3.5.2 Powers:**

In this test case we use the mul function (3.5.1) to calculate powers. The inputs are the first and second words in memory {0($0), 1($0)}. The output equals the first input raised to the power of the second input. The output is stored in R5

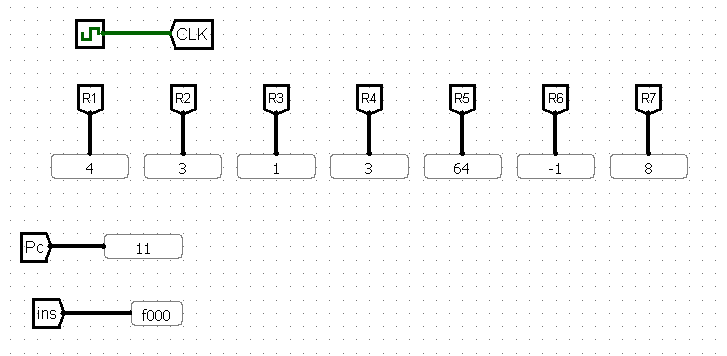
**Assembly Code:**

****

**Converted Instructions:**  **Memory Input:**

****

**Output:** (43 = 64) stored in R5.

 R2, R3, and R4 are used in the main program and preserved when calling mul function.

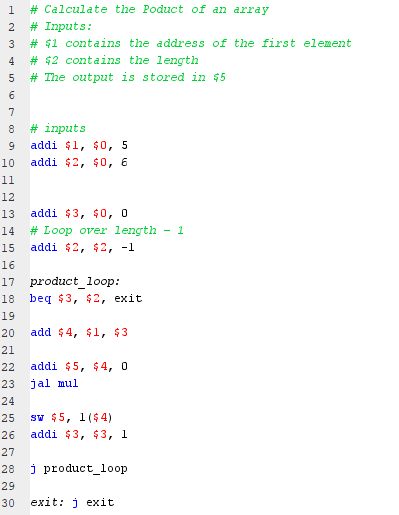
**3.5.3 Product of an Array:**

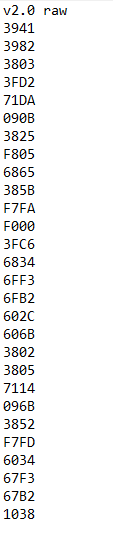
In this test case we use the mul function (3.5.1) to calculate the product of an array stored in memory.

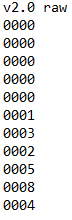
The inputs are R1 contains the address of the first element R2 contains the length of the array

The output is stored in R5

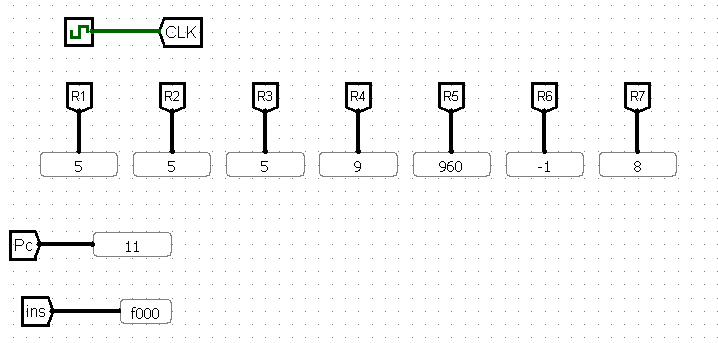
**Assembly Code:**



**Converted Instructions:**  **Memory Input:**



**Output:** (1 x 3 x 2 x 5x 8 x 4 = 960) stored in R5

R2, R3, and R4 are used in the main program and preserved when calling mul function.