

BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Computer Systems Engineering

COMPUTER ARCHITECTURE (ENCS4370)

Course Project #2

Designing and testing a simple single cycle RISC processor

Students name: Osama Rihami ID:1190560

Student name: Ferdos Ahmad ID: 1161937

Instructor: Ayman Hroub

Section: 3.

Date: 12/6/2022

# Abstract

The aim of our project is to design and test a simple single cycle RISC processor, and to build a data path ISA which has three instructions formats namely, R-type, I-type and J-type with 24-bits using the Logisim simulator tool.

Contents

[Abstract 2](#_Toc105953092)

[Theory and Procedure 4](#_Toc105953093)

[Arithmetic and Logical Unit (ALU) 5](#_Toc105953094)

[Register File 6](#_Toc105953095)

[Instruction Memory 7](#_Toc105953096)

[Data Memory 7](#_Toc105953097)

[Control Unit 8](#_Toc105953098)

[PC Unit 10](#_Toc105953099)

[Extender 11](#_Toc105953100)

[Full Datapath 12](#_Toc105953101)

[Simulation and Testing 13](#_Toc105953102)

[Before testing 14](#_Toc105953103)

[After testing 16](#_Toc105953104)

[Conclusion 17](#_Toc105953105)

# 

# Theory and Procedure

At this project we Design and implement a single cycle-Datapath and its control logic.

We build a five-stage single cycle: (Instruction Fetch (Instruction Memory), Instruction Decode (Register), Execute (ALU), Memory (Data Memory) and the Write Back (Register).

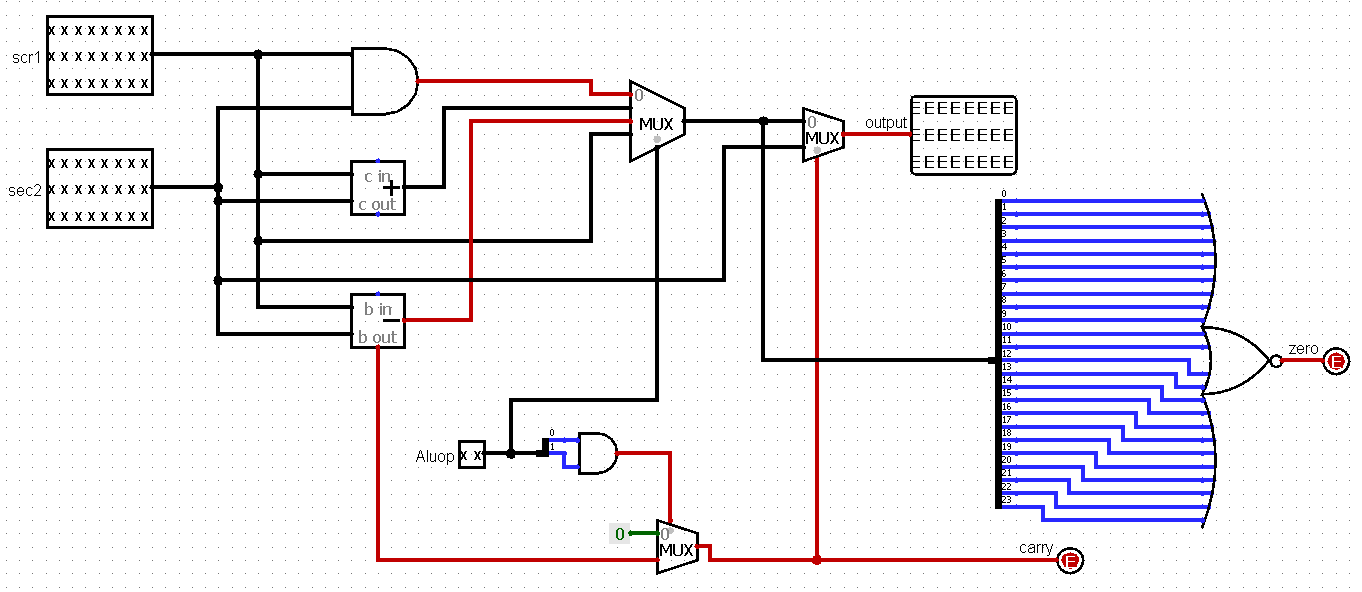
Our design consists of seven main blocks:

* Instruction Memory
* Register File
* Arithmetic and Logical Unit (ALU)
* Data Memory
* Control Unit
* Pc Unit
* Extender

Then we build the full Datapath using these components along with the required gates and registers.

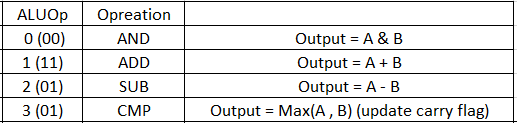
## Arithmetic and Logical Unit (ALU)

A 24-bit ALU component that takes two 24-bit inputs (A and B) then do one of these operations: AND, ADD, SUB and CMP, and the result well be sent to the 24-bit (output), It also has two output flags (zero and the carry). The operation one on the inputs is determined by a MUX with 2-bit selection input called (Aluop).



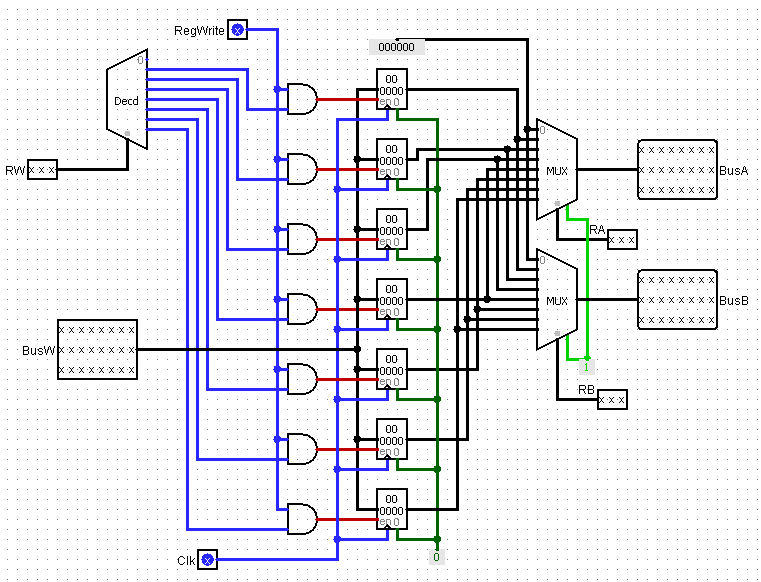
The zero flag only have a high value when all the output bits are (0), the carry flag will only be updated when the ALUOp = 11 (CMP and CAS instructions), and will result 1 when B input have a higher value than A and 0 otherwise.

The truth table for the ALU:



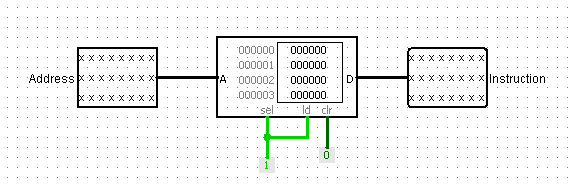
## Register File

We build the Register File using 7 (24-bit) registers (R1,R7) and 0 constant value for R0 ,and using two multiplexers for output ‘BusA’ and ‘BusB’ with the 3-bit selection input taken from (RA and RB) for read operations , and for the write operation we used a decoder with the 3-bit selection input (RW) which enables the write operation on the specified register writing the data taken from (BusW),also we have RegWrite input which enables or disable the write operation , this is a sequential circuit so the write operations only happens at the clock (Clk) trigger.



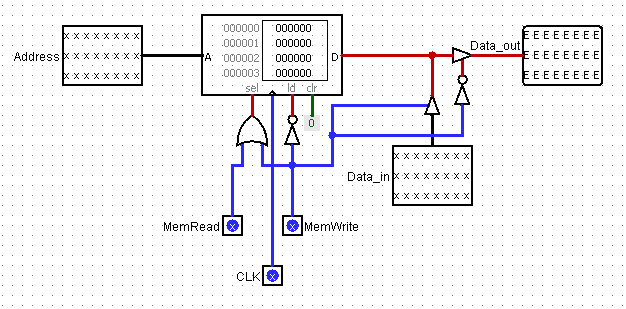
## Instruction Memory

We build the instruction memory using a (24\*24) bit RAM, which takes the address and load the instruction to the output, this is a combinational circuit so we don’t use clock.



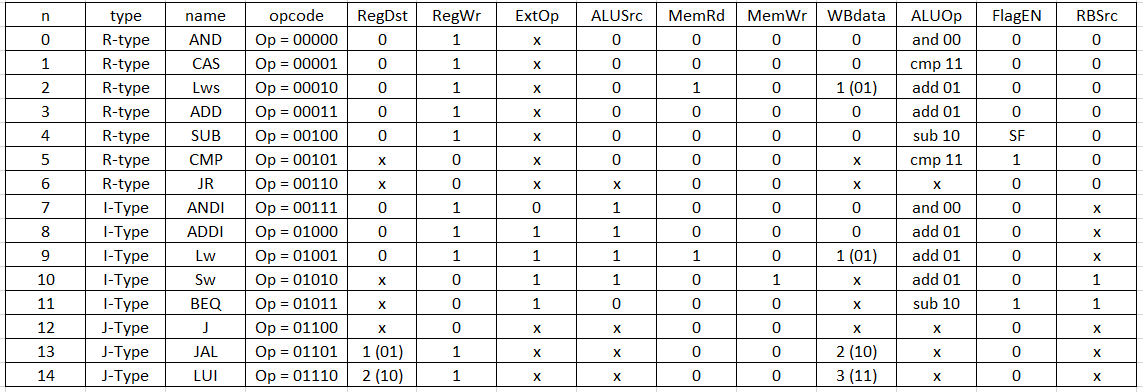
## Data Memory

Data Memory is a lot similar to the instruction memory, but we add two tri-state buffers, one for the load instruction which loads the data at a specified address to the output (Data\_out) only when the input (MemRead = 1), and the other one for the store instruction when (MemWrite =1) it writes the data from (Data\_in) on the memory at the specified address, this is a sequential circuit so the write operations only happen at the clock (Clk) trigger.



## Control Unit

For the control unit, at first, we build the Truth Table for all the signals in the Datapath, then we write the Boolean functions for the signals, the control unit takes the 5-bit opcode input ,2-bit Cond input, and 1- bit SF input, and have 10 outputs as its shown in the table:



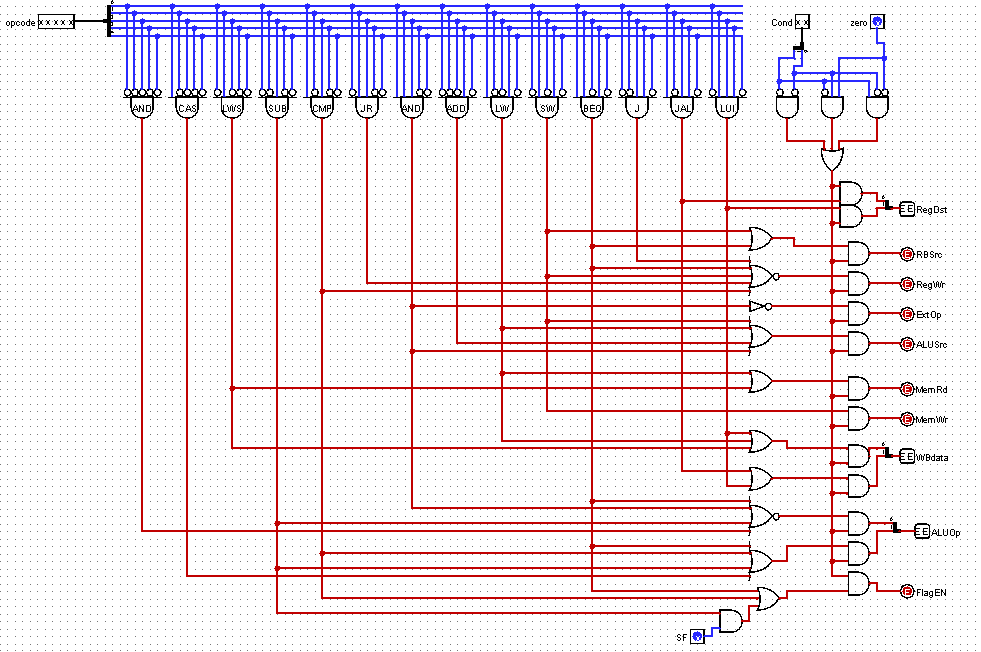
And the Boolean functions:

|  |  |  |  |
| --- | --- | --- | --- |
| RegDst0 = JAL |  |  |  |
| RegDst1 = LUI |  |  |  |
| RegWr = (CMP + JR + SW +BEQ +J)' | | | |
| ExtOp = ANDI' |  |  |  |
| ALUSrc = ANDI + ADDI + LW +SW | | |  |
| MemRd = LW + LWS | |  |  |
| MemWr = SW |  |  |  |
| WBdata0 = LWS + LW +LUI | | |  |
| WBdata1 = JAL + LUI | |  |  |
| ALUOp0 = (AND + SUB + ANDI + BEQ)' | | | |
| ALUOp1 = CAS + SUB + CMP + BEQ | | | |
| FlagEN = (SUB \* SF) + CMP + BEQ | | | |

Then we build the design using 5-bit AND gates.

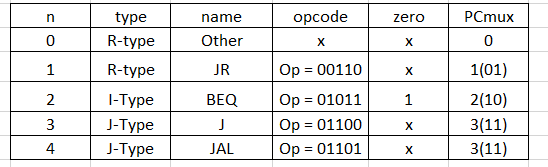
Also, the all output will always be 0 unless, the Cond =00, or Cond = 01 & zero = 1

, or Cond = 10 & zero = 0



## PC Unit

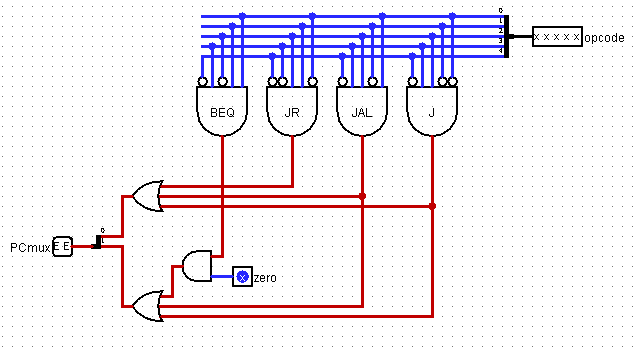
Its similar to the control unit, it takes the opcode and the zero flag, and determine the value of the PC mux as shown in the truth table:



And the Boolean functions:

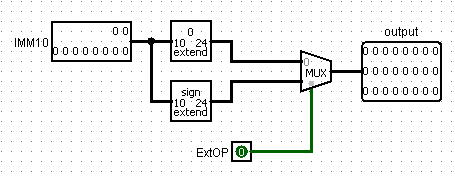
PCmux0 = JR + J +JAL

PCmux1 = BEQ\*ZERO + J + JAL



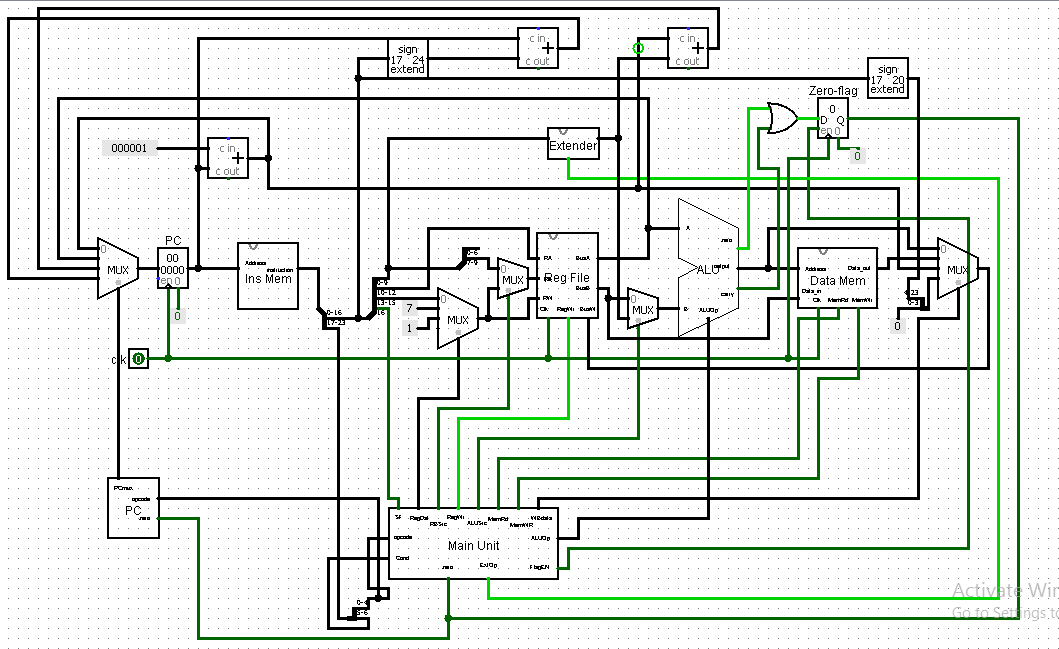
## Extender

It’s a simple circuit, which extends the 10-bit immediate value to 24 -bit, with a mux that determines if it’s a zero extender or a sign extender determined by the (ExtOP) input.



## Full Datapath

After building all the components we connect them together as shown in the following Datapath:



# Simulation and Testing

We tested the following code after we loaded it into the instruction memory:

lui 2

lw r2 , 1(r0)

addne r3 ,r2 ,r1

cas r6 ,r2 ,r1

andi r5 ,r2 , 3

sw r3 ,2(r0)

jal 100

Which is in binary is:

00 01110 0 0000000000000010

00 01001 0 010 000 0000000001

10 00011 0 011 010 001 0000000

00 00001 0 110 010 001 0000000

00 00111 0 101 010 0000000011

00 01010 0 011 000 0000000010

00 01101 0 0000000001100100

The program was loaded into the memory in hexa as follows:

1C0002

124001

866880

02C880

0EA803

146002

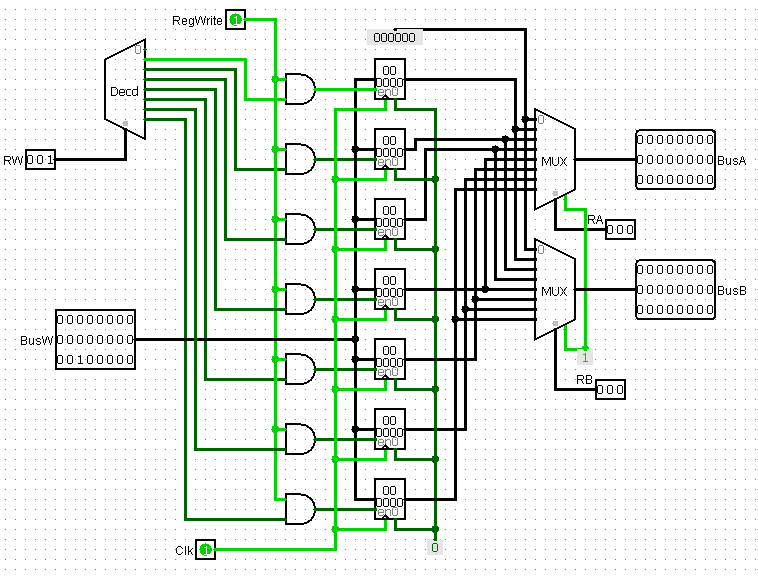
1A0064

This code sequence is supposed to load immediate number (2) in R1 register, and load value in address (0R0) to R2, then add R2 +R1 and store the output in R3 if the zero flag = 0. Then stores the maximum value between R2 and R1 in R6, then AND between immediate number (3) and R2 then store output in R5.

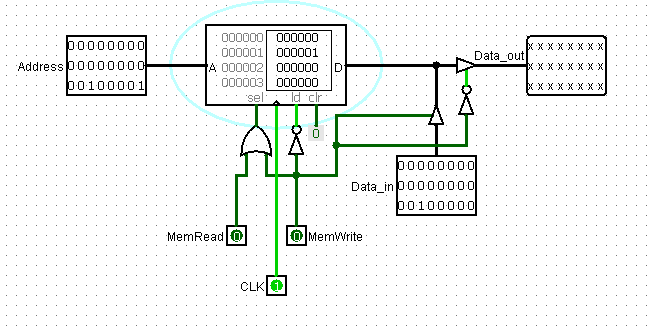
After that it stores the result R3 into the address 2(R0), then jump to 100 with R7 =PC+1.

## Before testing

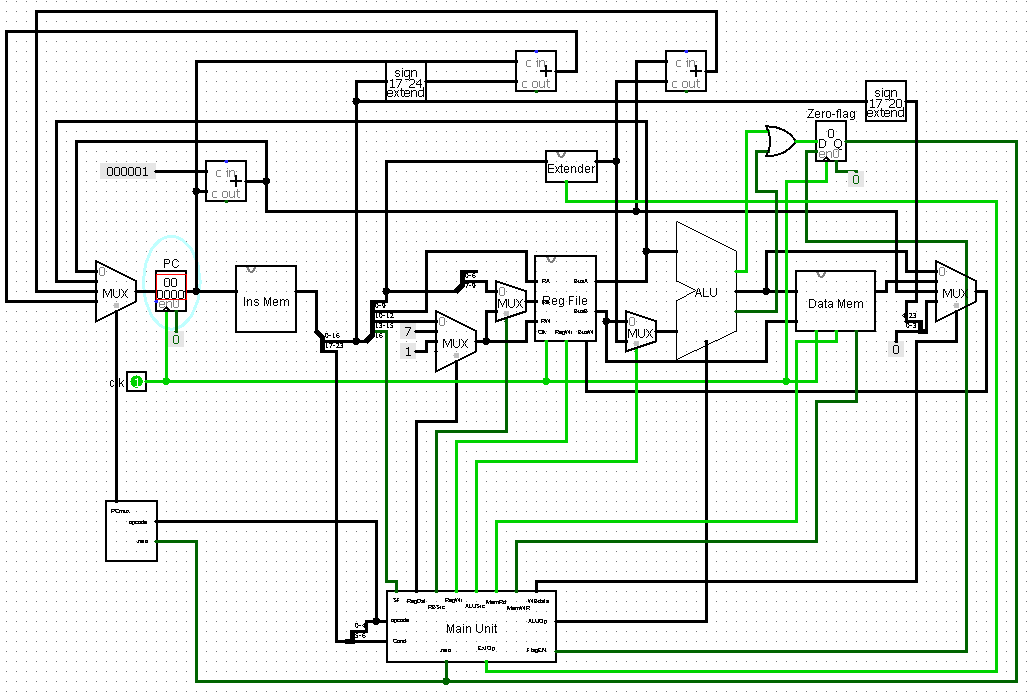
the register file is:



The data memory is:

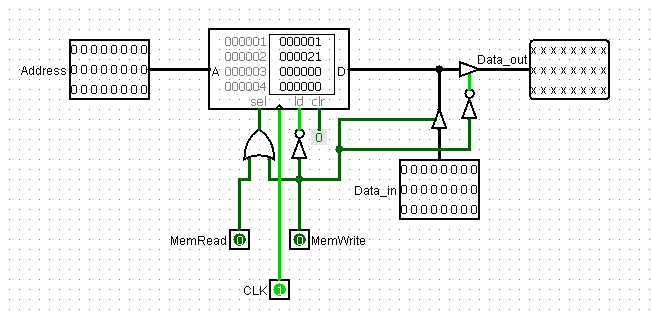


The full data path is:

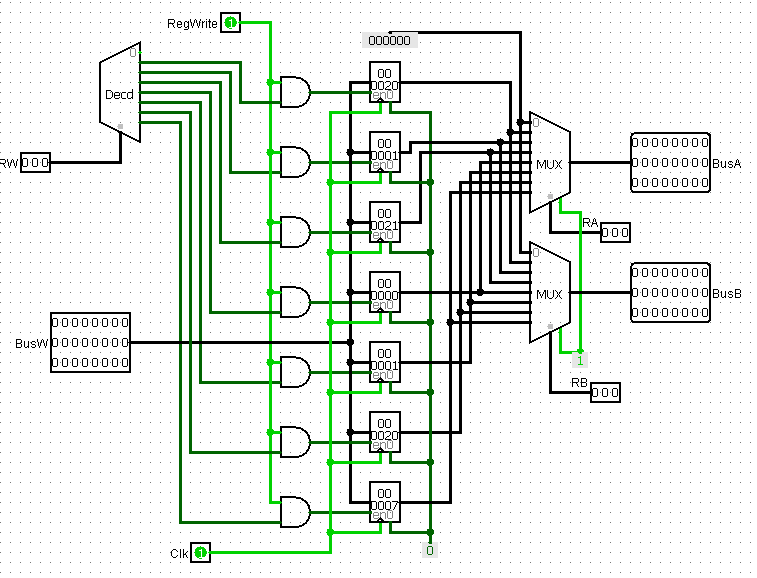


## After testing

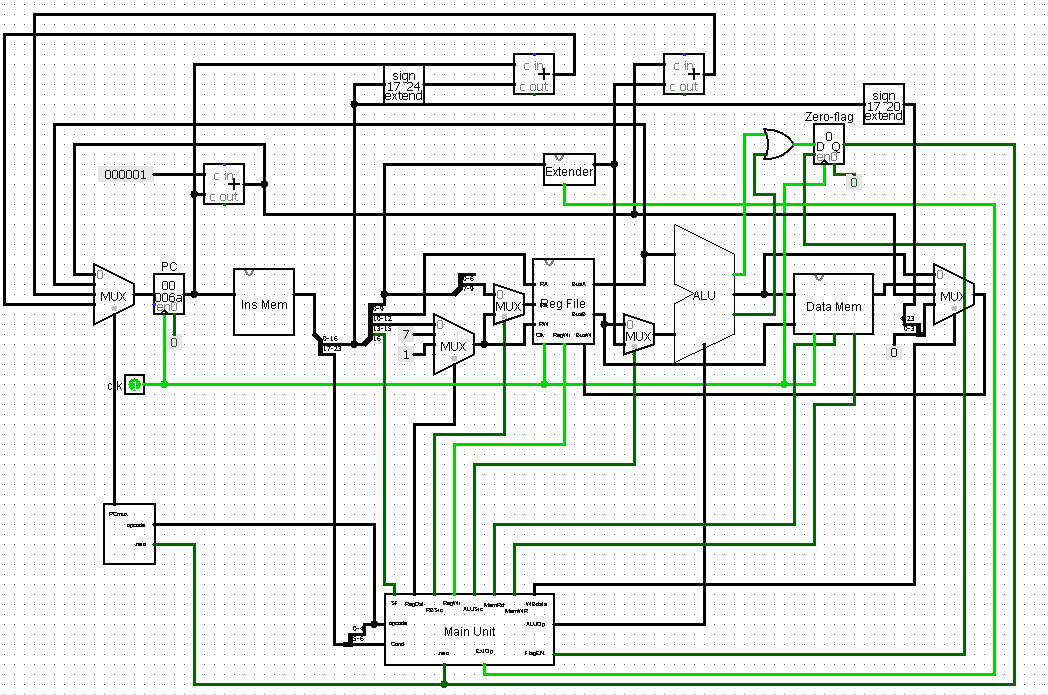
The data memory after testing is:



The register file after testing:



the full data path after testing is:



# Conclusion

The simulation worked fine and it outputs the correct data in the registers and right place in memory.