Maxwell Thimmig
Charlie Schafer
Jim Palomo

Project 3 Report | Group 7

1. Instruction & Instruction lists
    a. SIMP: Standardized Instructions for Mathematical Processing. The goal that SIMP strives to do is to accomplish the problem statement written in project 3. The approach used for this ISA was an ASIC (application-specific integrated circuit) style. Therefore, the only thing SIMP can do is to do the following:
        i. Given three initial values (A, B, C) the SIMP ISA will generate an array from A1 - A100 (DM[0] to DM[99]) that accomplishes the mathematical application of (A+B) XOR C.

A1	= (A + B)	XOR C
A2	= (A1 + B)	XOR C
A3	= (A2 + B)	XOR C
...		
A100	= (A99 + B)	XOR C

        ii. Generate a width array W1-W100 (DM[128] to DM[227]) for each A1-A100 that computes the width (given bit notation from A-array) from leftmost to rightmost one.

e.g. width(0110 1110) = 6
width(1000 0000) = 1

b. Instruction Table

Instr	A	B	C	D	Access	Format
init	0	0	0	0	xxx	00 xxx iii
	0	0	0	1		
	0	0	1	0		
	0	0	1	1		
*slt$_{R0}$	0	1	0	0	R0, R5 R4	0100 ----
inc$_{R5}$	0	1	0	1	R5, R5	0101 ---
sw$_{R5}$	0	1	1	0	R5, RA(R7)	0110 ----
	0	1	1	1		
fw$_{R7}$	1	0	0	0	R7 RA	1000 ----
ga$_{RA}$	1	0	0	1	RA, RB, RC	1001 ----
*bo$_{R0}$	1	0	1	0	R0	1010 ----
*sb$_{RA}$	1	0	1	1	RA (R5)	1011 ----
	1	1	0	0		
	1	1	0	1		
	1	1	1	0		
halt	1	1	1	1		1111 1111

* x = register access

e.g. xxx = 3 bit register access [0-7]

* - = don't care

e.g. fw$_{R7}$ = 1000 ---- = 1000 1111 = 1000 0101

* Access header:

Majority of instructions have registers hardwired to them. For instance, sb$_{RA}$ has registers RA and R5 hardwired. Therefore, when the instruction is called, sb$_{RA}$, DM[R5] = RA is enacted.

* RA = R1; RB = R2; RC = R3

* (R5) & (R7) are being used as a memory address holder and byte width holder respectively

Functional Descriptions

- slt_{R0}
  - If R5 < R4, R0 = 1
  - Else R0 = 0
- Inc_{R5}
  - R5 += 1
- Sw_{R5}
  - DM[128 + R5] = R7
- Fw_{R7}
  - R7 = width(RA)
- Ga_{RA}
  - RA = (RA + RB) XOR RC
- $B0_{R0}$
  - If R0 == 1, PC += immediate (-6 → working).
- Sb_{RA}
  - Store A to DM (DM[R5] = RA)
- Init
  - RX = [100,1,-1,15,73,51]
  - xxx iii = {0,1,2,3,4,5}: Mapping for iii

c. The most significant features that our ISA provides is the hardware implementation that allows for one instruction to conduct a series of computations. For instance, the find width instruction, fw_{R5}, would have required numerous shifts (sll/srl), additions (add), and subtractions (sub) to compute the width. However, this is all done in one single instruction: fw_{R5}. This allowed us to lower the instruction count dramatically.

Another example of this is the instruction ga_{RA}, which generates the A terms that are stored into the A-Array (A1-A100). Instead of having a general approach where the add and xor instructions are executed, ga_{RA} condenses these operations into one single instruction.

Finally, the majority of our instructions have been linked to specific registers for ASIC style implementations. Each instruction has set registers that they work with to execute their job. For example, store byte, sb_{RA}, uses registers R5 and RA (=R1) to perform its job. Sb_{RA} stores the value RA (after ga_{RA} is executed which performs (A+B) XOR C) into the data memory located at R5.

d. In regards to the instruction count, the hardware implementation has been modified to account for this. This was seen in the response from part c where we reduced the instruction count for finding a byte's width from four to . Similarly, ga_{RA} allowed us to reduce the number of instructions needed to fulfill its functionality from two to one. This ASIC style approach did not simplify the hardware, but maximized its efficiency in terms of reducing overall instruction count and achieving its required functionality. Hardware simplification was performed where possible and this will be shown below.

   This approach was used for almost every instruction in our instruction set. The idea of hardwiring each instruction with specific registers allowed for hardware simplification. Each instruction had specific registers accomplish its job. An example of this can be seen in the last paragraph of part c regarding sb_{RA}.

   The main limitations for this approach is that our hardware can only be used to perform this specific task of generating A-terms [(A+B) XOR C] and generating widths to be stored in arrays A and W. If any other problem statement was provided, our ASIC-style ISA would not be able to accommodate these changes.

e. The main compromises for our ISA is that almost every instruction has been hardwired. This allowed for a lower instruction count and allowed enough instructions to be used given an 8-bit limitation. At first, we decided to approach the project charter from a general-purpose point of view, similar to MIPS. However, with limited bits to represent instructions, a new outlook had to be drafted. Thus, we came up with the idea of hardwiring every register with specific instructions (note that some registers will accommodate for multiple instructions). Examples of hardwiring instructions can be seen above (e.g. in part c, sb_{RA} stores RA = R1 into the data memory address of R5).

2. ALU design

   The ALU compromises of several tasks:

   - Selection Result:
     - Determine which output will be generated depending on the opcode/instruction used (uses combinational logic).
   - A Term Generator:
     - Generate the A-terms (A1-100; [A+B] XOR C) that will be stored in the A-array
   - Width Determination:
     - Determine the width of an 8-bit binary number from the generated A-term stored in the A-array from leftmost bit to rightmost bit.
   - Init
     - Based on the init immediate bits (last 3 bits), a number is selected using a mux and constant outputs to initialize a register with a value. The following values implemented are [100, 1, -1, 15, 73, 51].
   - Slt
     - A single ALU block is used to perform the slt operation on two registers: SrcA and SrcB. This ALU block is given a constant opcode of 111 to reflect that it only performs the slt operation.
   - Write Enable
     - Depending on the opcode/instruction (sb_{RA}, sw_{R5}), write enable is determined with combinational logic.
   - Address
     - A single ALU block is used to determine the address that data will be stored when sb_{RA} and sw_{R5} are used.

   The SIMP ISA contains many special operations that are hardwired to register specific instructions. However, there are two specific instructions that our ISA performs which are distinguishable from other ISAs. The following instructions; generate A term, ga_{RA}, which computes A+B XOR C as well as find width, fw, allow for a total of six instructions between the two, to be performed in only two instructions. These circuits compute the A terms and their widths from leftmost to rightmost one using combinational logic.

3. RF design

   The SIMP ISA supports 8 registers but only uses 7 of them. The registers that are utilized are [0-5, 7], this leaves register 6 as available to future updates or implementations. All of the registers are linked to specific instructions but are not limited to reading one register at a time. The following shows how the registers are used:

   a. R0:
      i. Used for branching and slt result (bo_{R0} and slt_{R0}).
         For example, slt compares R5 with R4 and stores the result {0 = R5 >= R4; 1 = R5 < R4} into R0. Then, bo_{R0} is invoked and checks R0 to see if a branch is needed {0 = no branch; 1 = branch}.

   b. RA (=R1):
      i. Used to store the initial A value and subsequent A terms (A1-A100).
         For example, when ga_{RA} is called then the result (A+B) XOR C is stored into RA.

   c. RB (=R2):
      i. Stores the initial B term and is used when ga_{RA} is called for (A+B) XOR C.

   d. RC (=R3):
      i. Stores the initial C term and is used when ga_{RA} is called for (A+B) XOR C.

   e. R4:
      i. Used for slt comparison operations, slt_{R0}.
         For example, when slt_{R0} is called, R5 and R4 are compared (R5 < R4?) the result is then stored into R0.

   f. R5:
      i. The most versatile register which is used for slt_{R0}, inc_{R5}, sb_{RA}, and sw_{R5}. R5 is a versatile register because it is used for several instructions and represents the data memory address of the A-array and W-array (given +128 offset for a sw_{R5} instruction).

         For example, when storing and generating the A-array, R5 is used to hold the data memory address. R5 is then compared with R4 to see whether or not 100 data cells have been populated (slt_{R0} and $B0_{R0}$). R5 is then incremented (inc_{R5}) to continue storing the generated A-terms at subsequent memory addresses (sb_{RA}). The width of the A-term is then stored in the W-array (W1-W100) using R5's value (holding the address + offset of 128 | sw_{R5}).

g. R6:
    i. Currently not utilized but available if necessary (in the future).

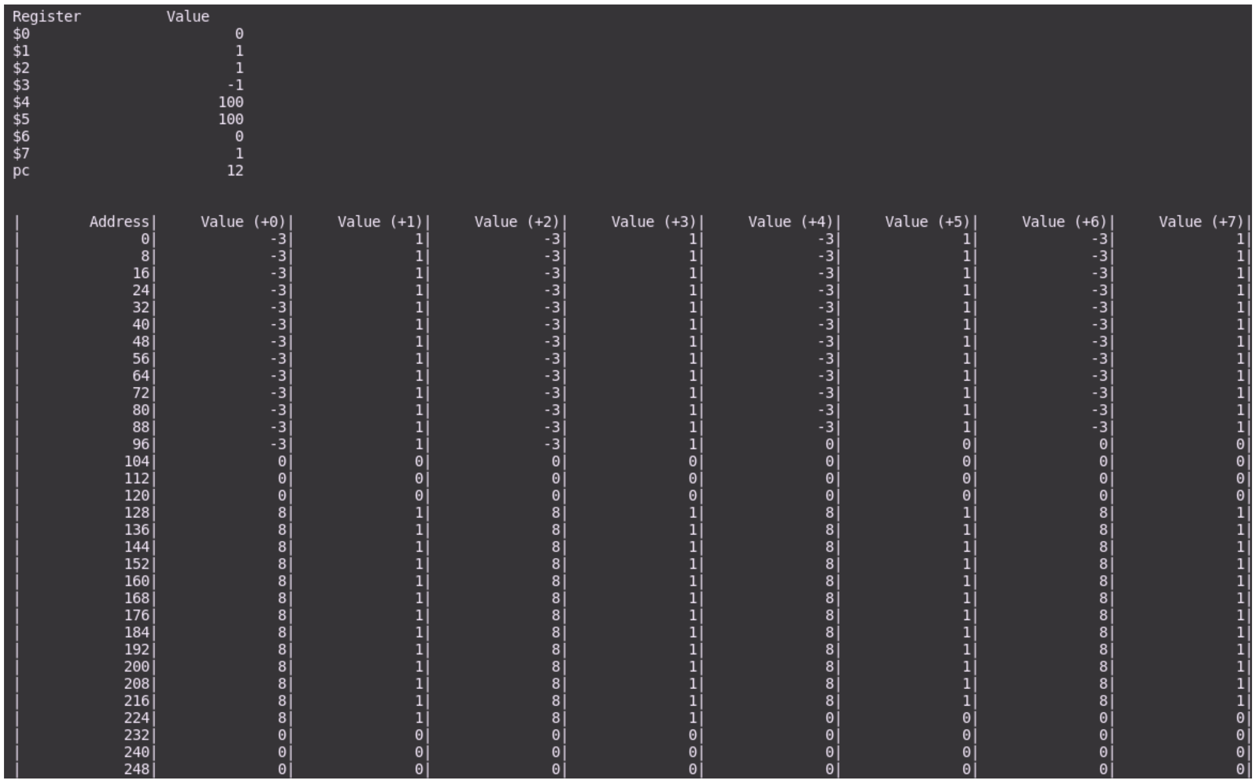
h. R7:
    i. Stores the computed width, fw_{R7}.
    For example, given an 8-bit binary number the instruction find width, fw_{R7}, will determine the width of that binary number from leftmost one to rightmost one and stores the value into R7.

4. Branch design
   The branch design only supports one branching instruction. The instruction, bo_{R0}, scans R0 to determine if a branch is necessary. If R0 is 0 when bo_{R0} is called then no branch will be taken. On the other hand, if R0 is 1 then a branch of PC += -6 will be executed. The value -6 was decided by locating the start of the next iteration for the next A-term computation (loop).

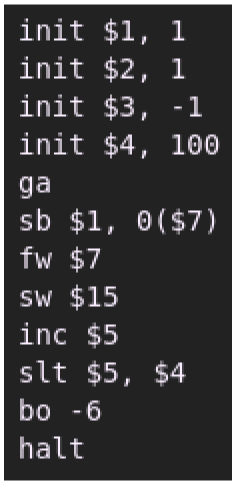
5. Data memory addressing modes
   SIMP provides two instructions that are used to access data memory. The two instructions are sw_{R5} and sb_{RA}. The range of addresses that are accessed through our design are from (DM[0] to DM[99]) for the A-array, and (DM[128] to DM[227]) for the W-array. The instructions can be differentiated as follows:
   - sb_{RA}: stores the A-term value within RA to the A-array address block sequentially.
   - sw_{R5}: stores the width from R7 that was calculated by fw_{R7} to an address of R5 plus a 128 offset.

6. Results for the Array-Width program

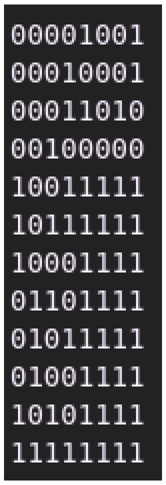
A = 1 | B = 1 | C = -1



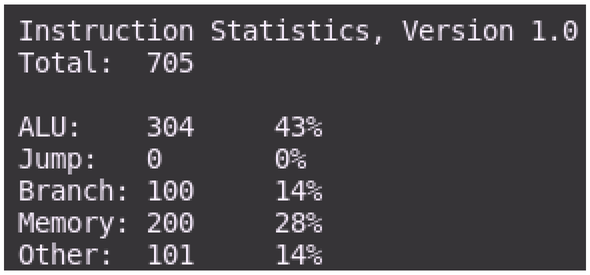
```
Register         Value
$0                   0
$1                   1
$2                   1
$3                  -1
$4                 100
$5                 100
$6                   0
$7                   1
pc                  12


|       Address|     Value (+0)|     Value (+1)|     Value (+2)|     Value (+3)|     Value (+4)|     Value (+5)|     Value (+6)|     Value (+7)|
|             0|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|             8|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            16|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            24|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            32|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            40|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            48|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            56|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            64|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            72|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            80|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            88|             -3|              1|             -3|              1|             -3|              1|             -3|              1|
|            96|             -3|              1|             -3|              1|              0|              0|              0|              0|
|           104|              0|              0|              0|              0|              0|              0|              0|              0|
|           112|              0|              0|              0|              0|              0|              0|              0|              0|
|           120|              0|              0|              0|              0|              0|              0|              0|              0|
|           128|              8|              1|              8|              1|              8|              1|              8|              1|
|           136|              8|              1|              8|              1|              8|              1|              8|              1|
|           144|              8|              1|              8|              1|              8|              1|              8|              1|
|           152|              8|              1|              8|              1|              8|              1|              8|              1|
|           160|              8|              1|              8|              1|              8|              1|              8|              1|
|           168|              8|              1|              8|              1|              8|              1|              8|              1|
|           176|              8|              1|              8|              1|              8|              1|              8|              1|
|           184|              8|              1|              8|              1|              8|              1|              8|              1|
|           192|              8|              1|              8|              1|              8|              1|              8|              1|
|           200|              8|              1|              8|              1|              8|              1|              8|              1|
|           208|              8|              1|              8|              1|              8|              1|              8|              1|
|           216|              8|              1|              8|              1|              8|              1|              8|              1|
|           224|              8|              1|              8|              1|              0|              0|              0|              0|
|           232|              0|              0|              0|              0|              0|              0|              0|              0|
|           240|              0|              0|              0|              0|              0|              0|              0|              0|
|           248|              0|              0|              0|              0|              0|              0|              0|              0|
```



```
init $1, 1
init $2, 1
init $3, -1
init $4, 100
ga
sb $1, 0($7)
fw $7
sw $15
inc $5
slt $5, $4
bo -6
halt
```

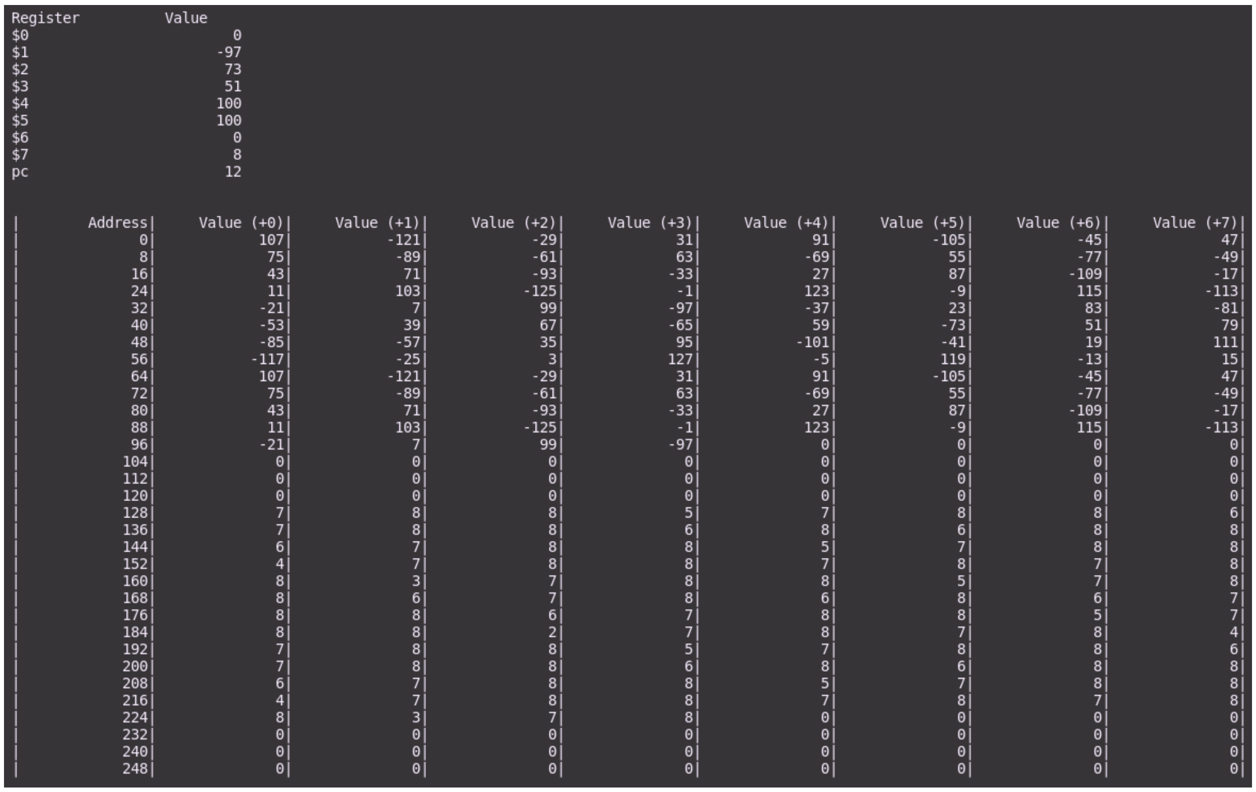


```
00001001
00010001
00011010
00100000
10011111
10111111
10001111
01101111
01011111
01001111
10101111
11111111
```

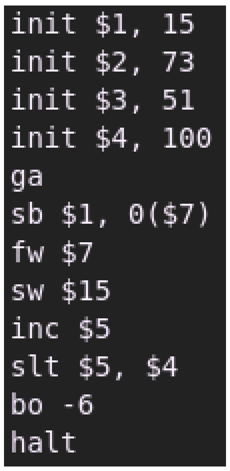


```
Instruction Statistics, Version 1.0
Total:   705

ALU:     304     43%
Jump:    0       0%
Branch:  100     14%
Memory:  200     28%
Other:   101     14%
```

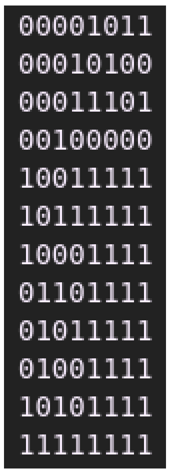
A = 15 | B = 73 | C = 51



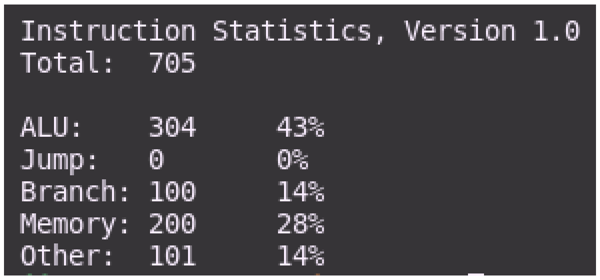
```
Register          Value
$0                    0
$1                  -97
$2                   73
$3                   51
$4                  100
$5                  100
$6                    0
$7                    8
pc                   12


|        Address|     Value (+0)|     Value (+1)|     Value (+2)|     Value (+3)|     Value (+4)|     Value (+5)|     Value (+6)|     Value (+7)|
|              0|            107|           -121|            -29|             31|             91|           -105|            -45|             47|
|              8|             75|            -89|            -61|             63|            -69|             55|            -77|            -49|
|             16|             43|             71|            -93|            -33|             27|             87|           -109|            -17|
|             24|             11|            103|           -125|             -1|            123|             -9|            115|           -113|
|             32|            -21|              7|             99|            -97|            -37|             23|             83|            -81|
|             40|            -53|             39|             67|            -65|             59|            -73|             51|             79|
|             48|            -85|            -57|             35|             95|           -101|            -41|             19|            111|
|             56|           -117|            -25|              3|            127|             -5|            119|            -13|             15|
|             64|            107|           -121|            -29|             31|             91|           -105|            -45|             47|
|             72|             75|            -89|            -61|             63|            -69|             55|            -77|            -49|
|             80|             43|             71|            -93|            -33|             27|             87|           -109|            -17|
|             88|             11|            103|           -125|             -1|            123|             -9|            115|           -113|
|             96|            -21|              7|             99|            -97|              0|              0|              0|              0|
|            104|              0|              0|              0|              0|              0|              0|              0|              0|
|            112|              0|              0|              0|              0|              0|              0|              0|              0|
|            120|              0|              0|              0|              0|              0|              0|              0|              0|
|            128|              7|              8|              8|              5|              7|              8|              8|              6|
|            136|              7|              8|              8|              6|              8|              6|              8|              8|
|            144|              6|              7|              8|              8|              5|              7|              8|              8|
|            152|              4|              7|              8|              8|              7|              8|              7|              8|
|            160|              8|              3|              7|              8|              8|              5|              7|              8|
|            168|              8|              6|              7|              8|              6|              8|              6|              7|
|            176|              8|              8|              6|              7|              8|              8|              5|              7|
|            184|              8|              8|              2|              7|              8|              7|              8|              4|
|            192|              7|              8|              8|              5|              7|              8|              8|              6|
|            200|              7|              8|              8|              6|              8|              6|              8|              8|
|            208|              6|              7|              8|              8|              5|              7|              8|              8|
|            216|              4|              7|              8|              8|              7|              8|              7|              8|
|            224|              8|              3|              7|              8|              0|              0|              0|              0|
|            232|              0|              0|              0|              0|              0|              0|              0|              0|
|            240|              0|              0|              0|              0|              0|              0|              0|              0|
|            248|              0|              0|              0|              0|              0|              0|              0|              0|
```



```
init $1, 15
init $2, 73
init $3, 51
init $4, 100
ga
sb $1, 0($7)
fw $7
sw $15
inc $5
slt $5, $4
bo -6
halt
```



```
00001011
00010100
00011101
00100000
10011111
10111111
10001111
01101111
01011111
01001111
10101111
11111111
```



```
Instruction Statistics, Version 1.0
Total:  705

ALU:     304     43%
Jump:    0       0%
Branch:  100     14%
Memory:  200     28%
Other:   101     14%
```

7. Individual / Group work reflection
    a.

	Day	Amount (minutes)
	10/15	20
	10/19	30
	10/20	45
	10/21	60
	10/27	180
	10/29	180
	10/31	240
Total		755

755 minutes of work were placed into this project which is equivalent to 12.58 hours.

In terms of content, Project 3 is currently the hardest project that has been released thus far. The reason for this is due to the number of different approaches possible for ISA implementation. As it happens, our group scrapped our ideas and started over three times before finalizing our approach.

In terms of approach, our Project 3 benefitted from a simultaneous simulator and hardware design effort. In previous projects, the approach has always been to tackle problems head on, one at a time. In this case, the group split off and worked on separate tasks, shifting back and forth when necessary.

In terms of group dynamics, the group worked well together compared to previous projects. The group worked on the tasks together during synchronous sessions. Each team member fully contributed to the analysis and design aspects of this project, which greatly accelerated troubleshooting and hardware implementation.

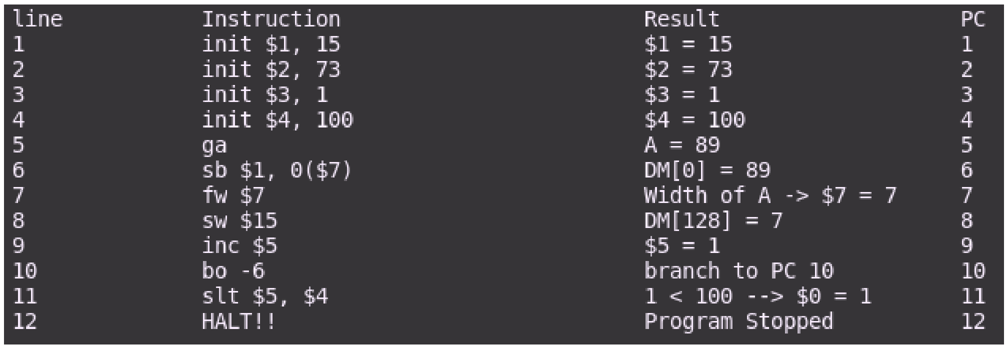
b. If you were restarting this project entirely, how would you have done it differently?
    i. After discussing this question, our group came to the conclusion that any other approach would have been unsatisfactory, due to lower performance and high similarity to MIPS. This would have detracted from our ability to design an original ISA tailored to the project charter. The challenge in creating our ASIC-style approach was an excellent learning opportunity for all members of the group.

c.

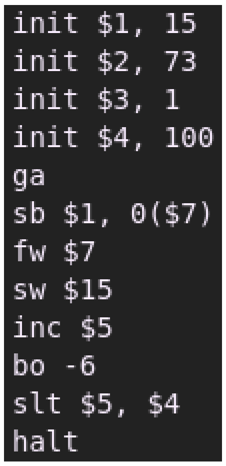
Members	Percentages
Maxwell Thimmig	33
Charlie Schafer	33
Jim Palomo	33

Part B) Software Package

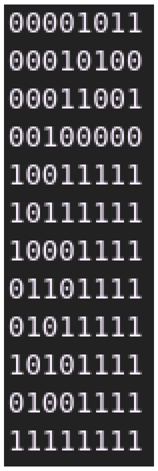
1. Provide a toy testcase program



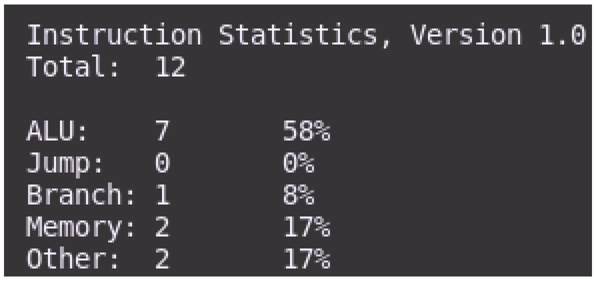
```
line          Instruction                        Result                   PC
1             init $1, 15                        $1 = 15                  1
2             init $2, 73                        $2 = 73                  2
3             init $3, 1                         $3 = 1                   3
4             init $4, 100                       $4 = 100                 4
5             ga                                 A = 89                   5
6             sb $1, 0($7)                       DM[0] = 89               6
7             fw $7                              Width of A -> $7 = 7     7
8             sw $15                             DM[128] = 7              8
9             inc $5                             $5 = 1                   9
10            bo -6                              branch to PC 10          10
11            slt $5, $4                         1 < 100 --> $0 = 1       11
12            HALT!!                             Program Stopped          12
```



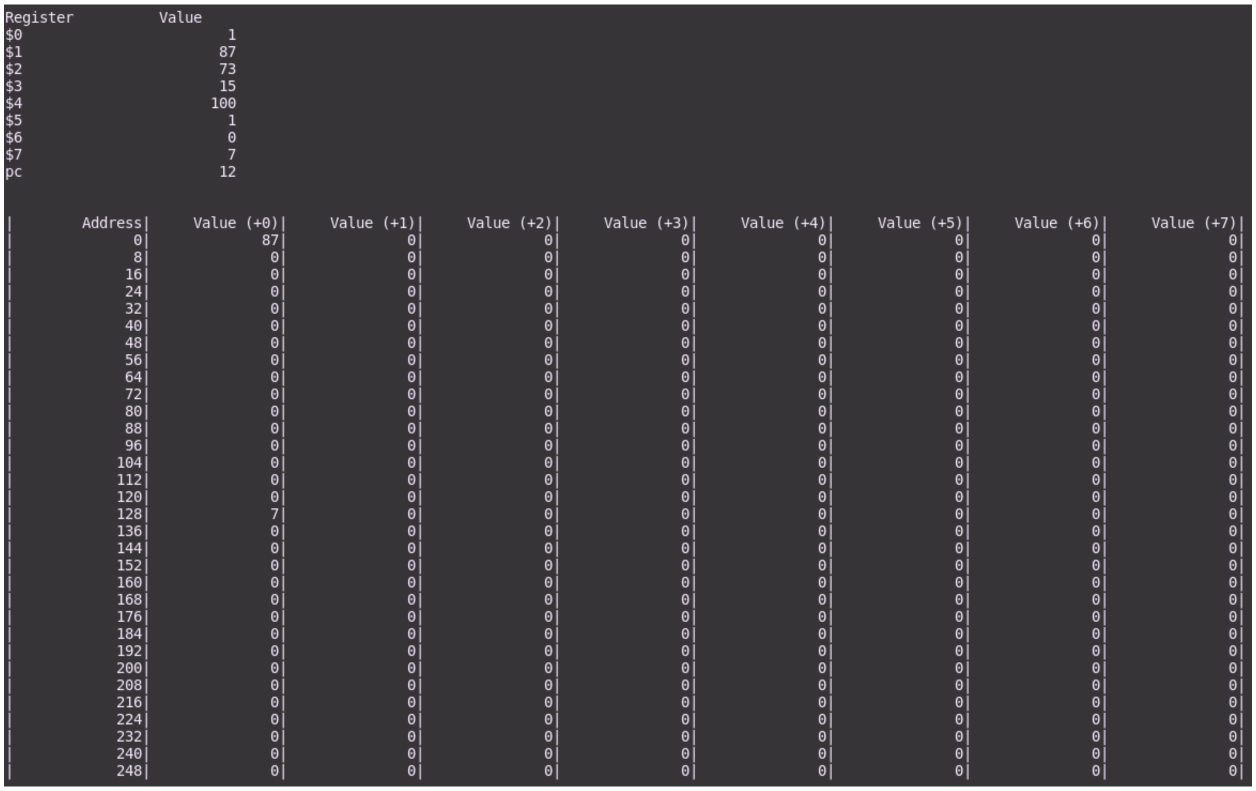
```
init $1, 15
init $2, 73
init $3, 1
init $4, 100
ga
sb $1, 0($7)
fw $7
sw $15
inc $5
bo -6
slt $5, $4
halt
```



```
00001011
00010100
00011001
00100000
10011111
10111111
10001111
01101111
01011111
10101111
01001111
11111111
```

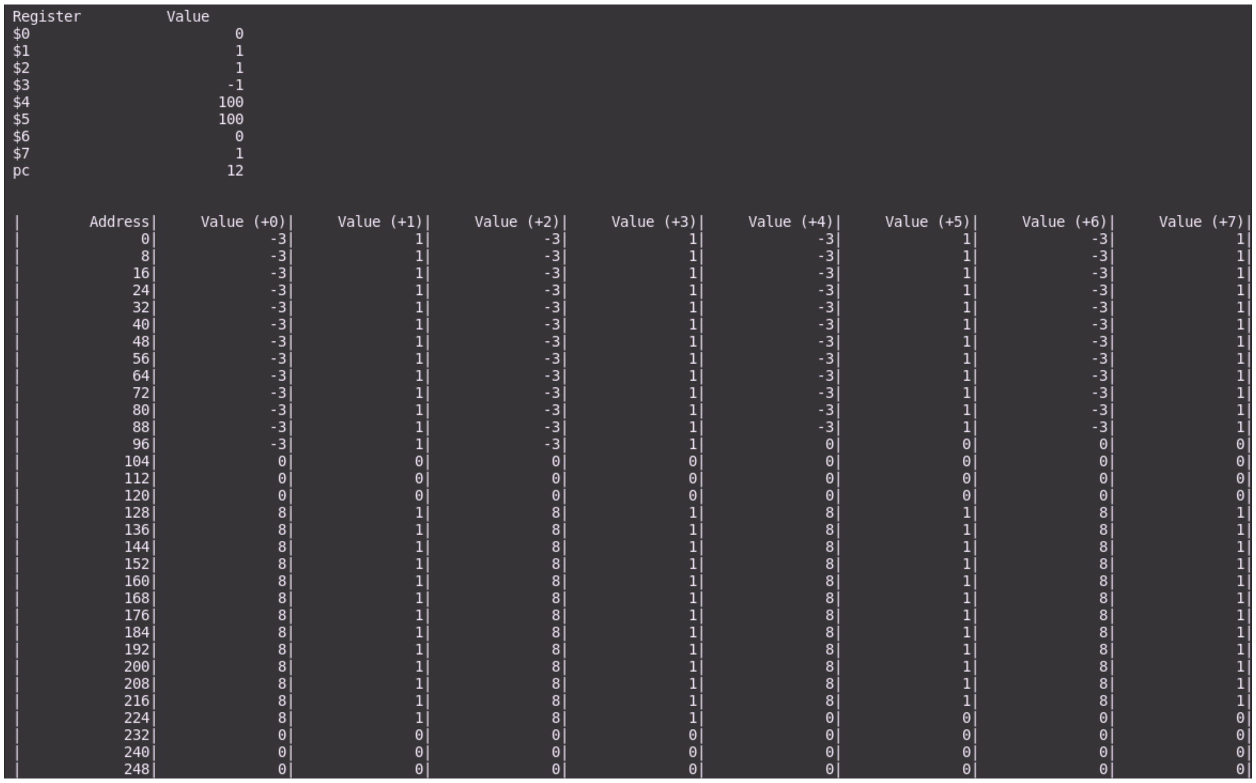


```
Instruction Statistics, Version 1.0
Total:  12

ALU:     7       58%
Jump:    0       0%
Branch:  1       8%
Memory:  2       17%
Other:   2       17%
```

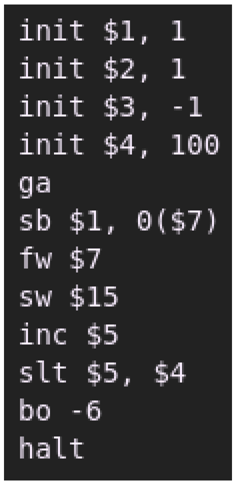


```
Register          Value
$0                    1
$1                   87
$2                   73
$3                   15
$4                  100
$5                    1
$6                    0
$7                    7
pc                   12

|       Address|    Value (+0)|    Value (+1)|    Value (+2)|    Value (+3)|    Value (+4)|    Value (+5)|    Value (+6)|    Value (+7)|
|             0|            87|             0|             0|             0|             0|             0|             0|             0|
|             8|             0|             0|             0|             0|             0|             0|             0|             0|
|            16|             0|             0|             0|             0|             0|             0|             0|             0|
|            24|             0|             0|             0|             0|             0|             0|             0|             0|
|            32|             0|             0|             0|             0|             0|             0|             0|             0|
|            40|             0|             0|             0|             0|             0|             0|             0|             0|
|            48|             0|             0|             0|             0|             0|             0|             0|             0|
|            56|             0|             0|             0|             0|             0|             0|             0|             0|
|            64|             0|             0|             0|             0|             0|             0|             0|             0|
|            72|             0|             0|             0|             0|             0|             0|             0|             0|
|            80|             0|             0|             0|             0|             0|             0|             0|             0|
|            88|             0|             0|             0|             0|             0|             0|             0|             0|
|            96|             0|             0|             0|             0|             0|             0|             0|             0|
|           104|             0|             0|             0|             0|             0|             0|             0|             0|
|           112|             0|             0|             0|             0|             0|             0|             0|             0|
|           120|             0|             0|             0|             0|             0|             0|             0|             0|
|           128|             7|             0|             0|             0|             0|             0|             0|             0|
|           136|             0|             0|             0|             0|             0|             0|             0|             0|
|           144|             0|             0|             0|             0|             0|             0|             0|             0|
|           152|             0|             0|             0|             0|             0|             0|             0|             0|
|           160|             0|             0|             0|             0|             0|             0|             0|             0|
|           168|             0|             0|             0|             0|             0|             0|             0|             0|
|           176|             0|             0|             0|             0|             0|             0|             0|             0|
|           184|             0|             0|             0|             0|             0|             0|             0|             0|
|           192|             0|             0|             0|             0|             0|             0|             0|             0|
|           200|             0|             0|             0|             0|             0|             0|             0|             0|
|           208|             0|             0|             0|             0|             0|             0|             0|             0|
|           216|             0|             0|             0|             0|             0|             0|             0|             0|
|           224|             0|             0|             0|             0|             0|             0|             0|             0|
|           232|             0|             0|             0|             0|             0|             0|             0|             0|
|           240|             0|             0|             0|             0|             0|             0|             0|             0|
|           248|             0|             0|             0|             0|             0|             0|             0|             0|
```

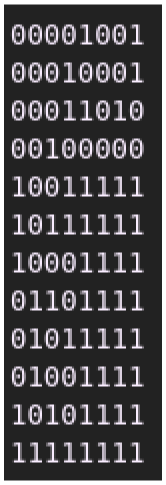
2. Provide two version of the array-width program
   A = 1 | B = 1 | C = -1



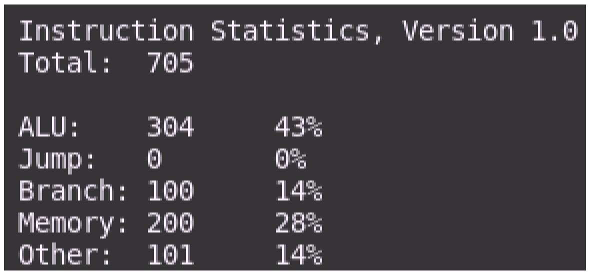
```
Register         Value
$0                   0
$1                   1
$2                   1
$3                  -1
$4                 100
$5                 100
$6                   0
$7                   1
pc                  12


|        Address|    Value (+0)|    Value (+1)|    Value (+2)|    Value (+3)|    Value (+4)|    Value (+5)|    Value (+6)|    Value (+7)|
|              0|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|              8|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             16|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             24|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             32|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             40|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             48|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             56|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             64|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             72|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             80|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             88|            -3|             1|            -3|             1|            -3|             1|            -3|             1|
|             96|            -3|             1|            -3|             1|             0|             0|             0|             0|
|            104|             0|             0|             0|             0|             0|             0|             0|             0|
|            112|             0|             0|             0|             0|             0|             0|             0|             0|
|            120|             0|             0|             0|             0|             0|             0|             0|             0|
|            128|             8|             1|             8|             1|             8|             1|             8|             1|
|            136|             8|             1|             8|             1|             8|             1|             8|             1|
|            144|             8|             1|             8|             1|             8|             1|             8|             1|
|            152|             8|             1|             8|             1|             8|             1|             8|             1|
|            160|             8|             1|             8|             1|             8|             1|             8|             1|
|            168|             8|             1|             8|             1|             8|             1|             8|             1|
|            176|             8|             1|             8|             1|             8|             1|             8|             1|
|            184|             8|             1|             8|             1|             8|             1|             8|             1|
|            192|             8|             1|             8|             1|             8|             1|             8|             1|
|            200|             8|             1|             8|             1|             8|             1|             8|             1|
|            208|             8|             1|             8|             1|             8|             1|             8|             1|
|            216|             8|             1|             8|             1|             8|             1|             8|             1|
|            224|             8|             1|             8|             1|             0|             0|             0|             0|
|            232|             0|             0|             0|             0|             0|             0|             0|             0|
|            240|             0|             0|             0|             0|             0|             0|             0|             0|
|            248|             0|             0|             0|             0|             0|             0|             0|             0|
```



```
init $1, 1
init $2, 1
init $3, -1
init $4, 100
ga
sb $1, 0($7)
fw $7
sw $15
inc $5
slt $5, $4
bo -6
halt
```

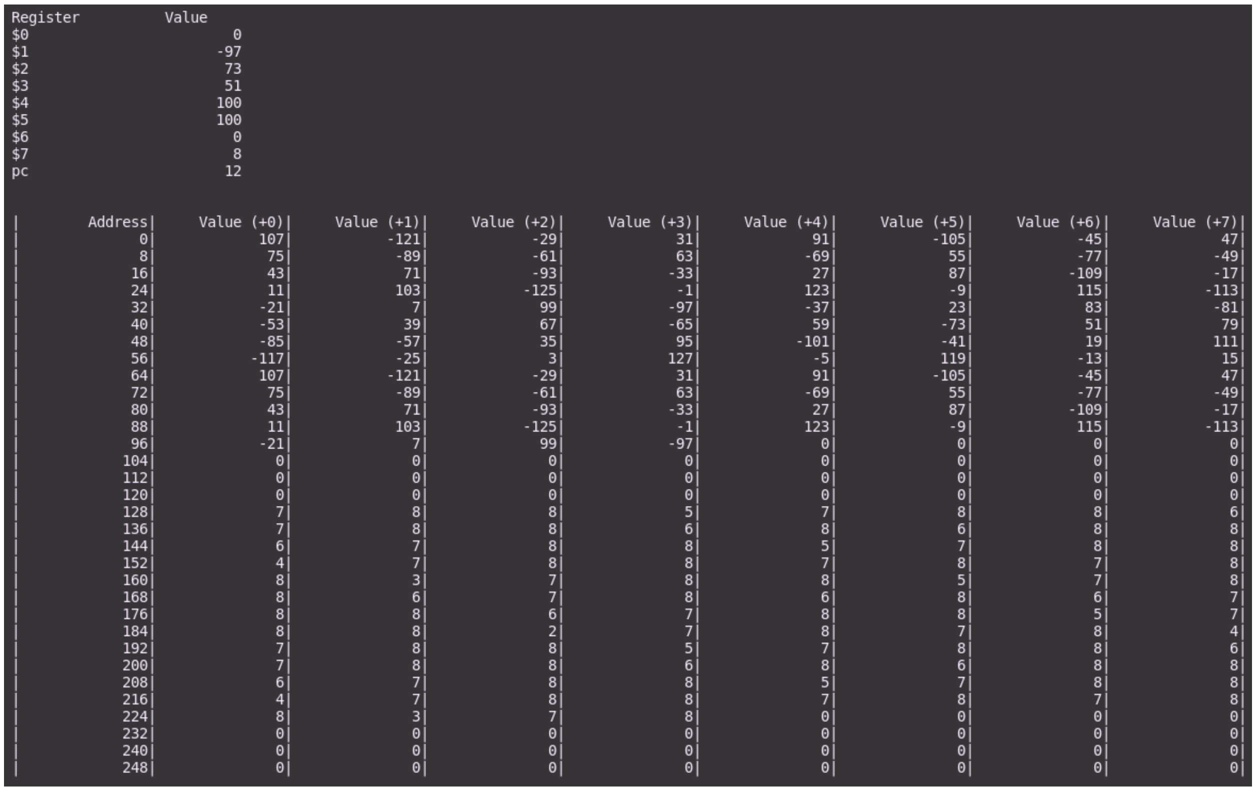


```
00001001
00010001
00011010
00100000
10011111
10111111
10001111
01101111
01011111
01001111
10101111
11111111
```

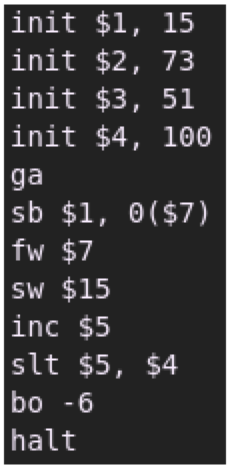


```
Instruction Statistics, Version 1.0
Total:   705

ALU:     304     43%
Jump:    0       0%
Branch:  100     14%
Memory:  200     28%
Other:   101     14%
```

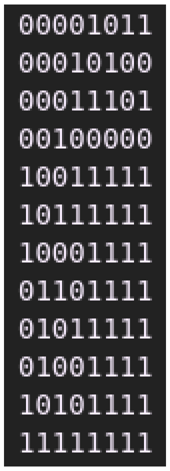
A = 15 | B = 73 | C = 51



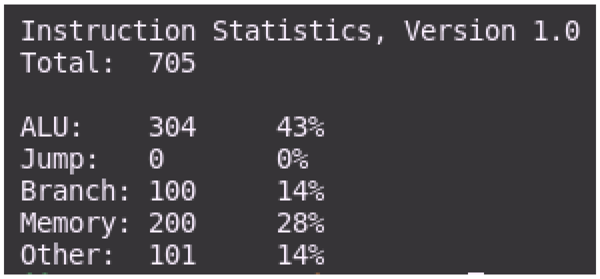
```
Register          Value
$0                    0
$1                  -97
$2                   73
$3                   51
$4                  100
$5                  100
$6                    0
$7                    8
pc                   12


|        Address|     Value (+0)|     Value (+1)|     Value (+2)|     Value (+3)|     Value (+4)|     Value (+5)|     Value (+6)|     Value (+7)|
|              0|            107|           -121|            -29|             31|             91|           -105|            -45|             47|
|              8|             75|            -89|            -61|             63|            -69|             55|            -77|            -49|
|             16|             43|             71|            -93|            -33|             27|             87|           -109|            -17|
|             24|             11|            103|           -125|             -1|            123|             -9|            115|           -113|
|             32|            -21|              7|             99|            -97|            -37|             23|             83|            -81|
|             40|            -53|             39|             67|            -65|             59|            -73|             51|             79|
|             48|            -85|            -57|             35|             95|           -101|            -41|             19|            111|
|             56|           -117|            -25|              3|            127|             -5|            119|            -13|             15|
|             64|            107|           -121|            -29|             31|             91|           -105|            -45|             47|
|             72|             75|            -89|            -61|             63|            -69|             55|            -77|            -49|
|             80|             43|             71|            -93|            -33|             27|             87|           -109|            -17|
|             88|             11|            103|           -125|             -1|            123|             -9|            115|           -113|
|             96|            -21|              7|             99|            -97|              0|              0|              0|              0|
|            104|              0|              0|              0|              0|              0|              0|              0|              0|
|            112|              0|              0|              0|              0|              0|              0|              0|              0|
|            120|              0|              0|              0|              0|              0|              0|              0|              0|
|            128|              7|              8|              8|              5|              7|              8|              8|              6|
|            136|              7|              8|              8|              6|              8|              6|              8|              8|
|            144|              6|              7|              8|              8|              5|              7|              8|              8|
|            152|              4|              7|              8|              8|              7|              8|              7|              8|
|            160|              8|              3|              7|              8|              8|              5|              7|              8|
|            168|              8|              6|              7|              8|              6|              8|              6|              7|
|            176|              8|              8|              6|              7|              8|              8|              5|              7|
|            184|              8|              8|              2|              7|              8|              7|              8|              4|
|            192|              7|              8|              8|              5|              7|              8|              8|              6|
|            200|              7|              8|              8|              6|              8|              6|              8|              8|
|            208|              6|              7|              8|              8|              5|              7|              8|              8|
|            216|              4|              7|              8|              8|              7|              8|              7|              8|
|            224|              8|              3|              7|              8|              0|              0|              0|              0|
|            232|              0|              0|              0|              0|              0|              0|              0|              0|
|            240|              0|              0|              0|              0|              0|              0|              0|              0|
|            248|              0|              0|              0|              0|              0|              0|              0|              0|
```



```
init $1, 15
init $2, 73
init $3, 51
init $4, 100
ga
sb $1, 0($7)
fw $7
sw $15
inc $5
slt $5, $4
bo -6
halt
```



```
00001011
00010100
00011101
00100000
10011111
10111111
10001111
01101111
01011111
01001111
10101111
11111111
```



```
Instruction Statistics, Version 1.0
Total:  705

ALU:    304     43%
Jump:   0       0%
Branch: 100     14%
Memory: 200     28%
Other:  101     14%
```

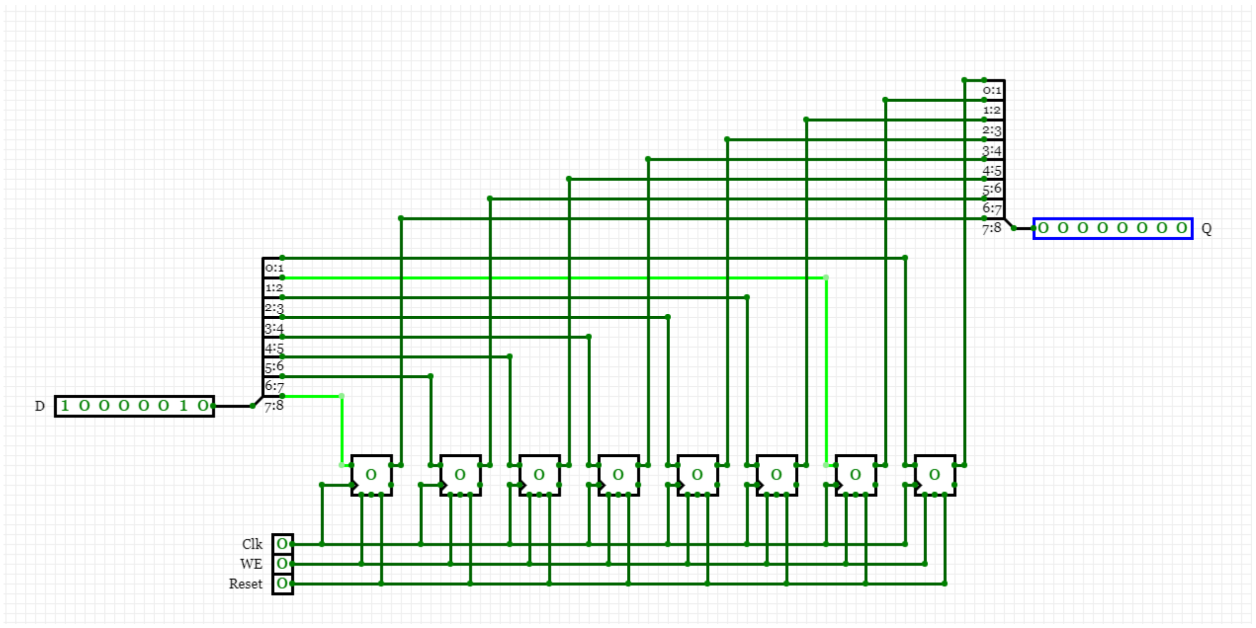
Part C) Hardware Package

1. Overall Schematic

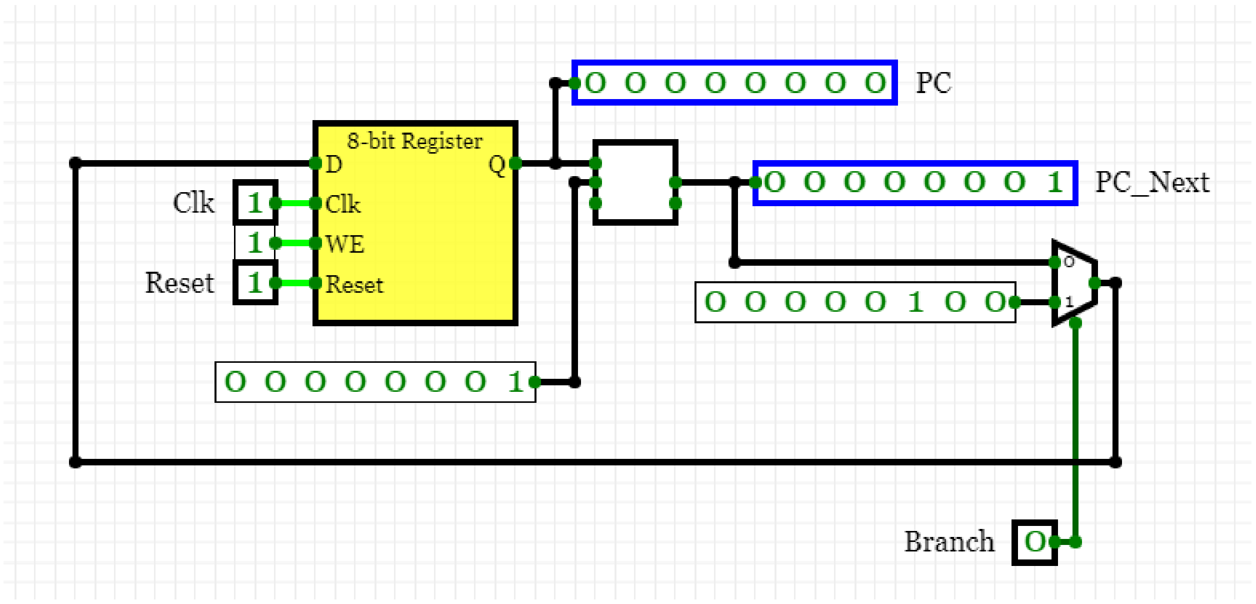
   Circuitverse link: [Project 3 ISA]()

   Components:

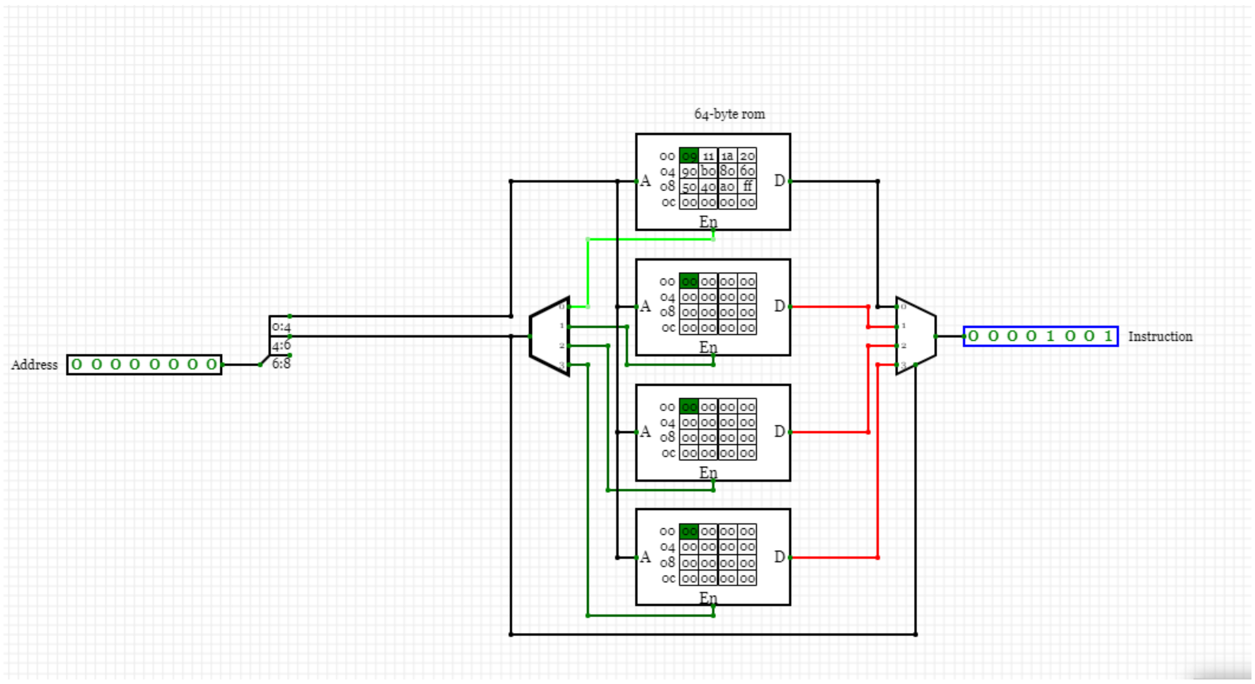
   8bit Register:



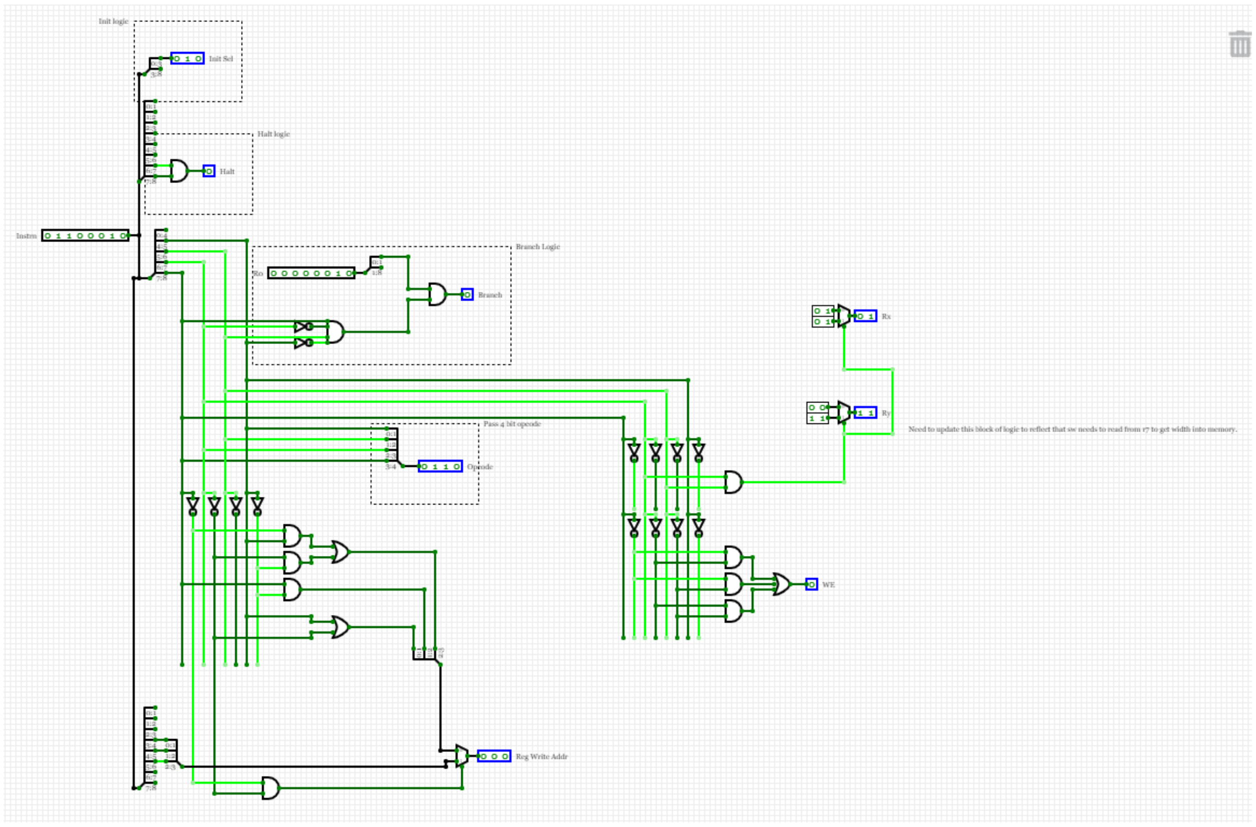
   PC-Logic:



Instruction Memory:

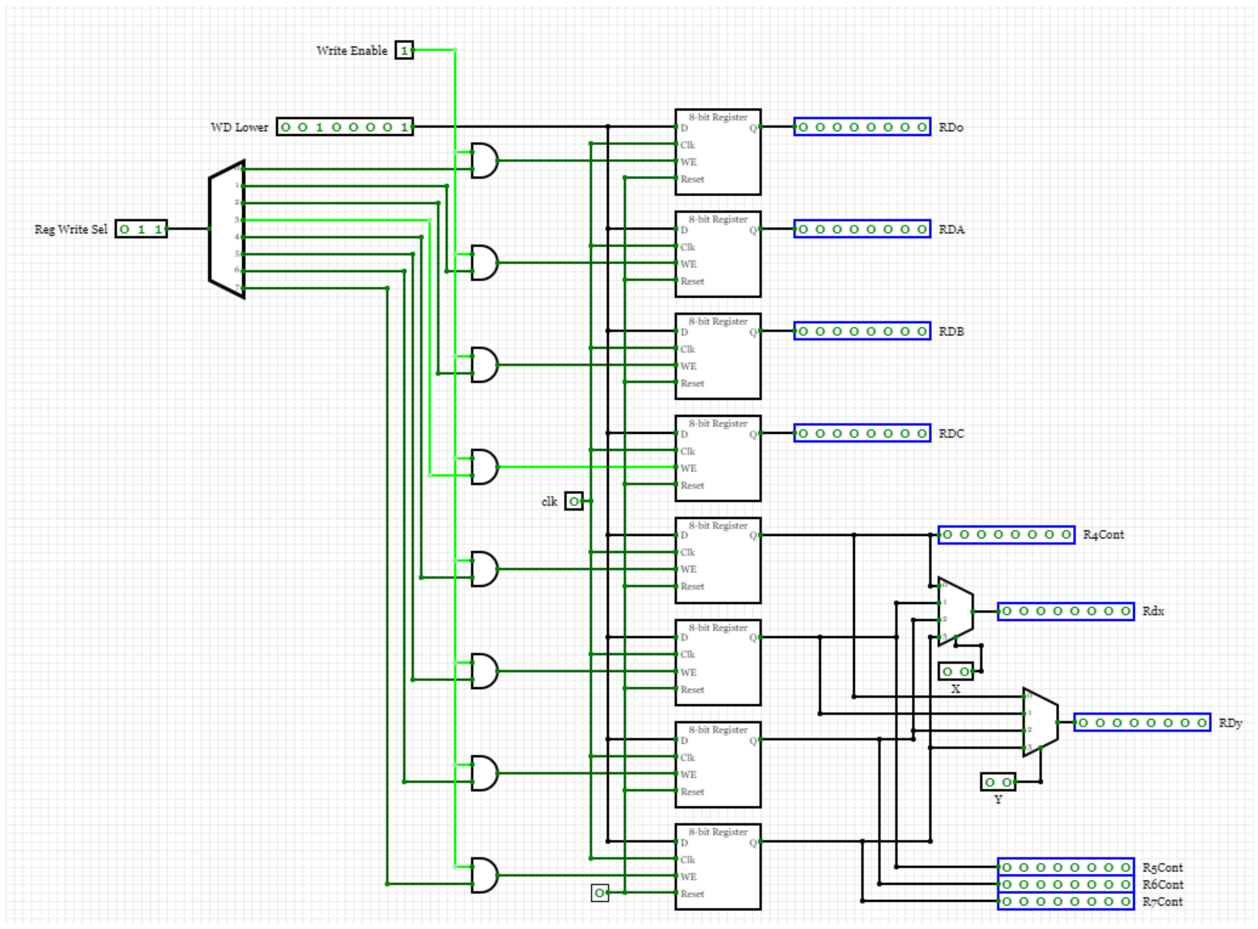


Control Unit:

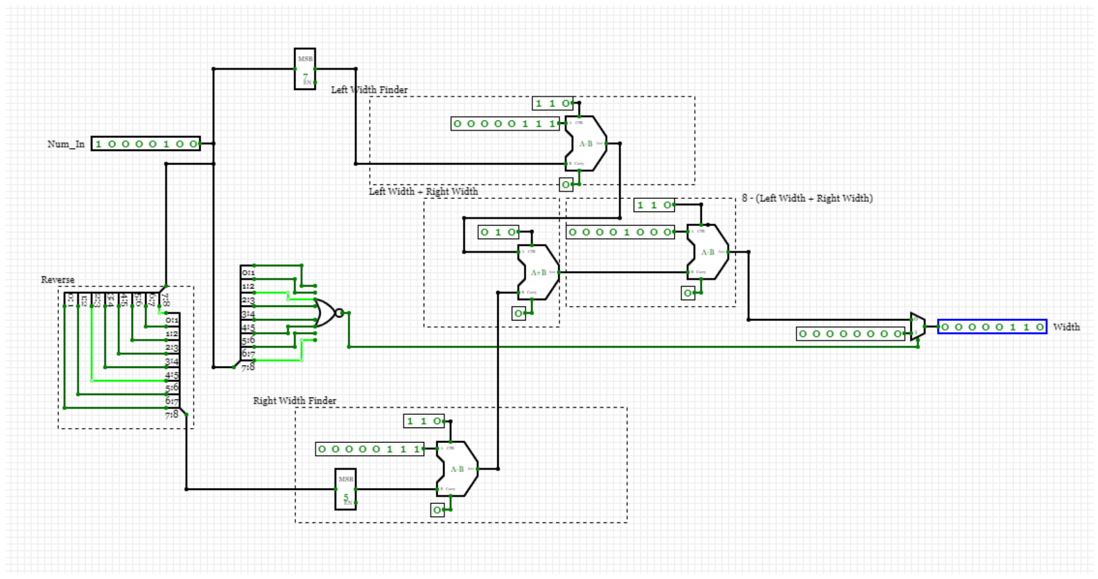


Need to update this block of logic to reflect that sw needs to read from r7 to get width into memory.

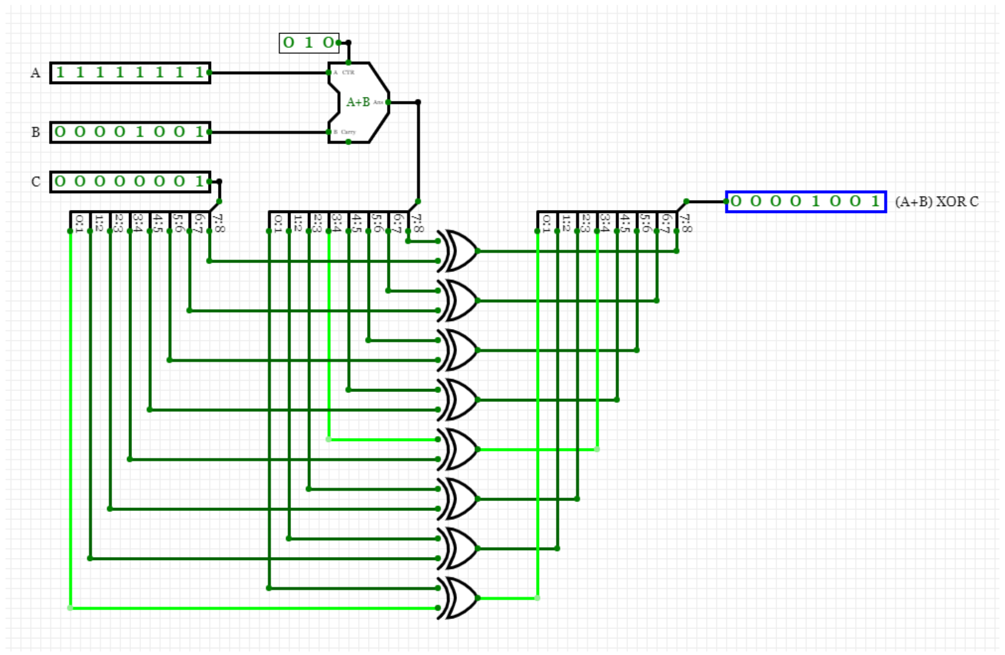
Register File:



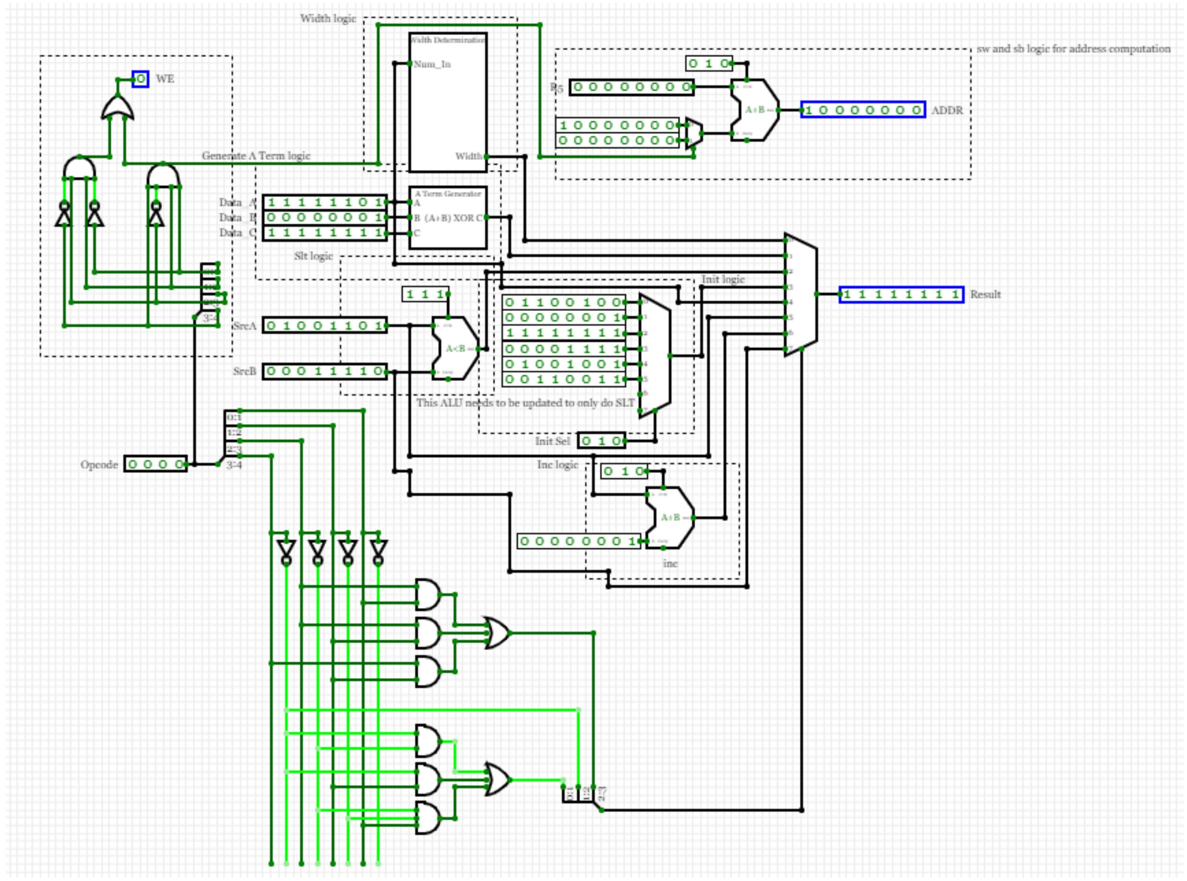
Width Determination:



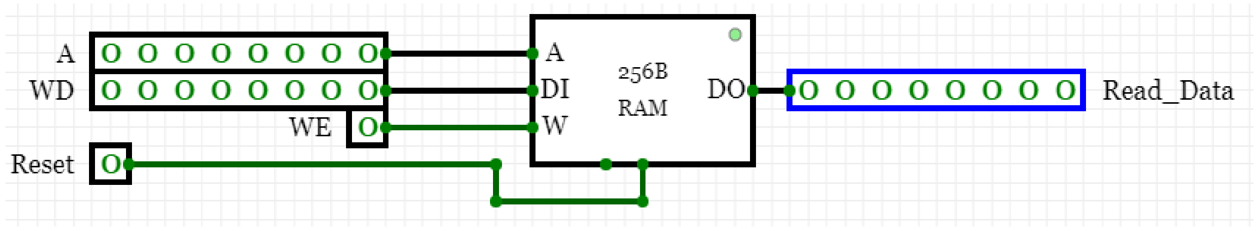
A-Term Generator:



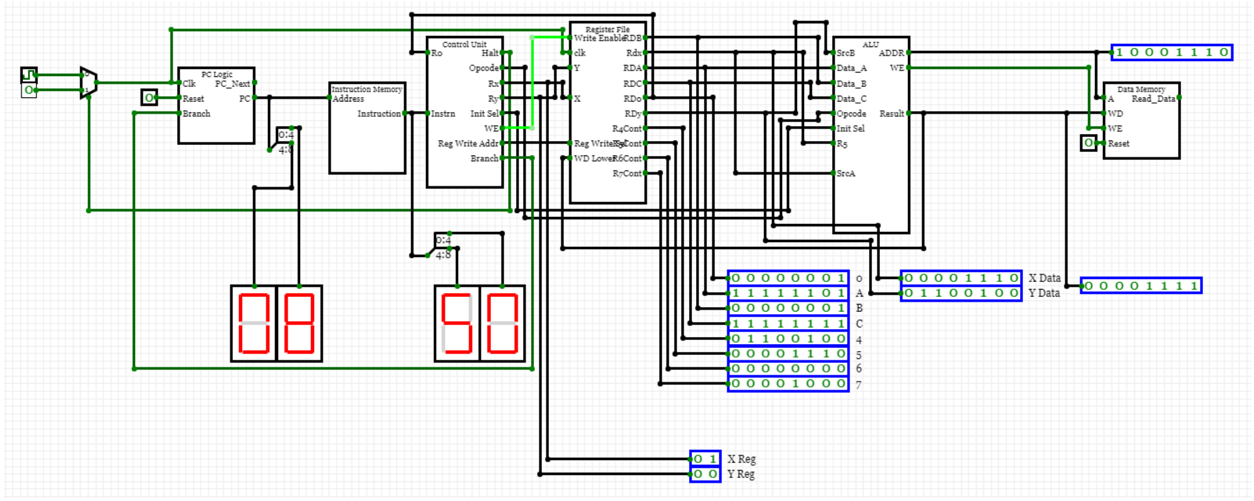
ALU:



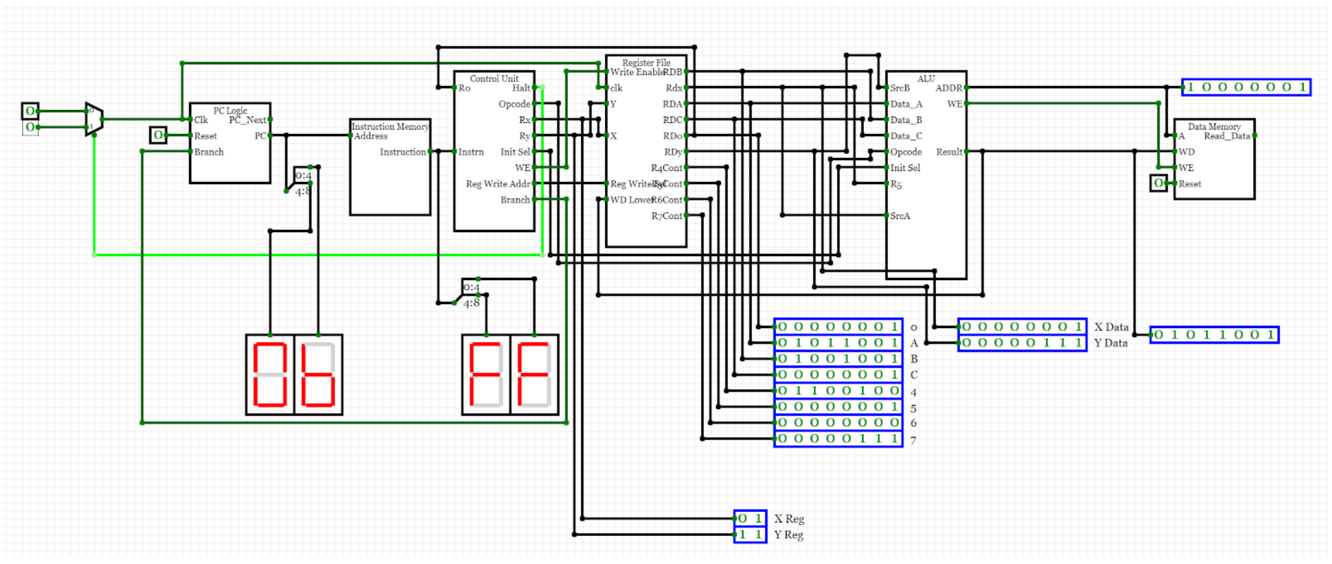
Data Memory:

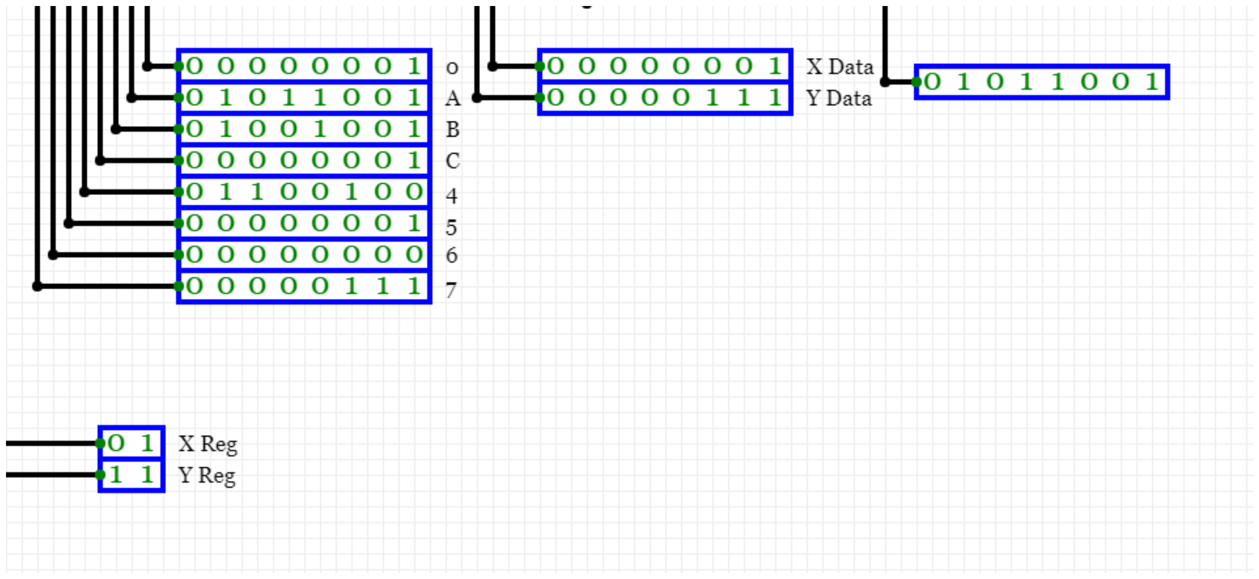


Main:

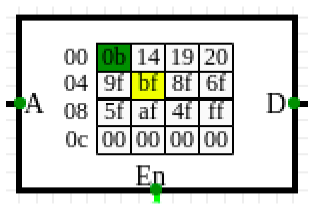


2. (10pts) Provide the schematics and result pictures of your toy testcase program to showcase your hardware implementation works:

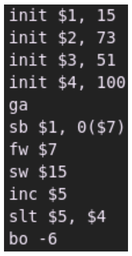




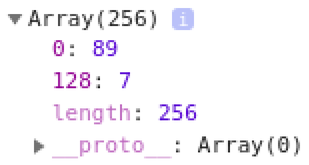
Input:



00	0b	14	19	20
04	9f	bf	8f	6f
08	5f	af	4f	ff
0c	00	00	00	00


```
init $1, 15
init $2, 73
init $3, 51
init $4, 100
ga
sb $1, 0($7)
fw $7
sw $15
inc $5
slt $5, $4
bo -6
```

Results for toy test case (data memory):



```
▼Array(256) i
   0: 89
   128: 7
   length: 256
  ▶__proto__: Array(0)
```

The following shows how only DM[0] = 89 & DM[128] = 7 are filled with non-zero values. The rest of the values are null (have not been touched therefore being zero).

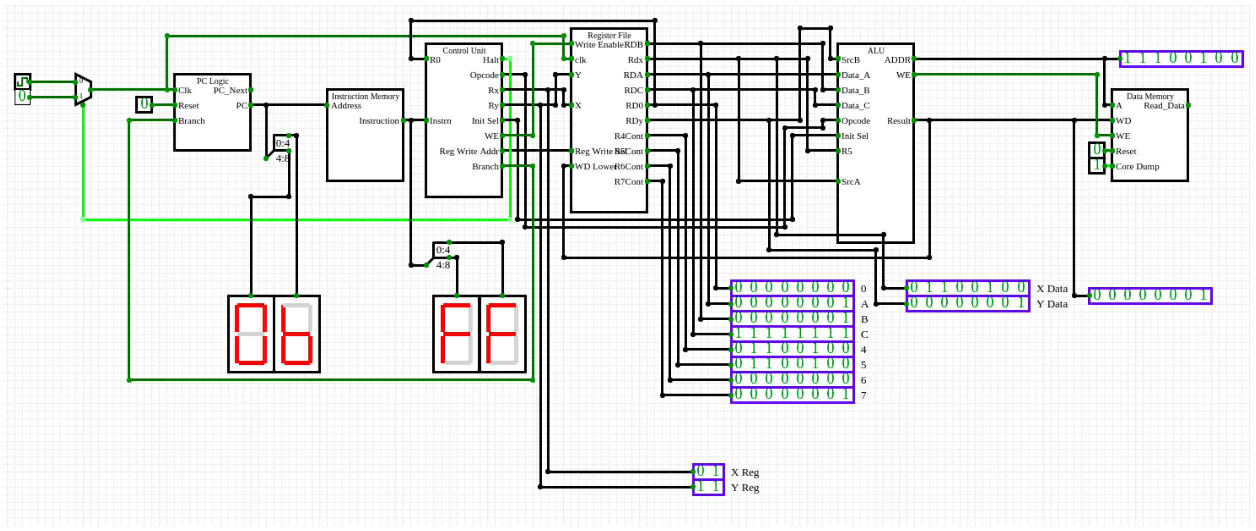
3. (Extra credit 10pts) Provide the schematics and result pictures of the two versions of Array-Width program in your ISA implementation to showcase your hardware works:

   Schematics above

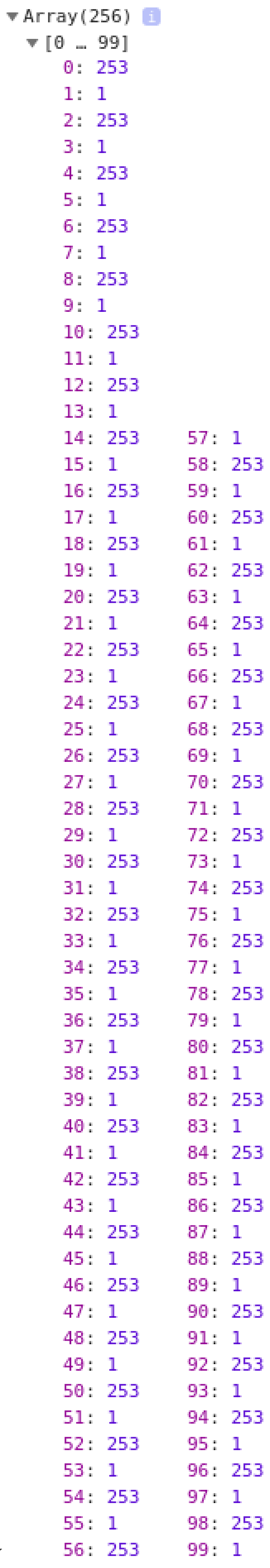
   A = 1 | B = 1 | C = -1

   Results for first 3 iterations:
   A1 = -3 → W1 = 8
   A1 = 1 → W1 = 1
   A1 = -3 → W1 = 8

   https://youtu.be/RC1vLbMcaoY

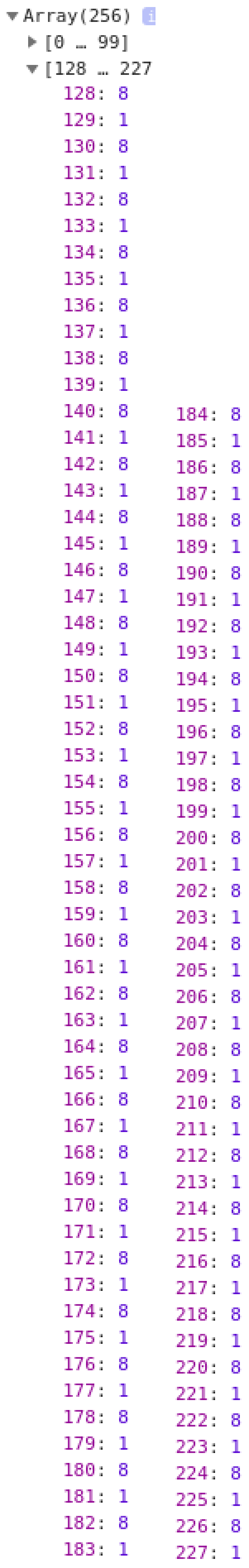


Core Dump V1 below:



```
▼Array(256) i
  ▼[0 … 99]
     0: 253
     1: 1
     2: 253
     3: 1
     4: 253
     5: 1
     6: 253
     7: 1
     8: 253
     9: 1
     10: 253
     11: 1
     12: 253
     13: 1
     14: 253    57: 1
     15: 1      58: 253
     16: 253    59: 1
     17: 1      60: 253
     18: 253    61: 1
     19: 1      62: 253
     20: 253    63: 1
     21: 1      64: 253
     22: 253    65: 1
     23: 1      66: 253
     24: 253    67: 1
     25: 1      68: 253
     26: 253    69: 1
     27: 1      70: 253
     28: 253    71: 1
     29: 1      72: 253
     30: 253    73: 1
     31: 1      74: 253
     32: 253    75: 1
     33: 1      76: 253
     34: 253    77: 1
     35: 1      78: 253
     36: 253    79: 1
     37: 1      80: 253
     38: 253    81: 1
     39: 1      82: 253
     40: 253    83: 1
     41: 1      84: 253
     42: 253    85: 1
     43: 1      86: 253
     44: 253    87: 1
     45: 1      88: 253
     46: 253    89: 1
     47: 1      90: 253
     48: 253    91: 1
     49: 1      92: 253
     50: 253    93: 1
     51: 1      94: 253
     52: 253    95: 1
     53: 1      96: 253
     54: 253    97: 1
     55: 1      98: 253
     56: 253    99: 1
```

A-Array



```
▼Array(256) i
  ▶[0 … 99]
  ▼[128 … 227
     128: 8
     129: 1
     130: 8
     131: 1
     132: 8
     133: 1
     134: 8
     135: 1
     136: 8
     137: 1
     138: 8
     139: 1
     140: 8     184: 8
     141: 1     185: 1
     142: 8     186: 8
     143: 1     187: 1
     144: 8     188: 8
     145: 1     189: 1
     146: 8     190: 8
     147: 1     191: 1
     148: 8     192: 8
     149: 1     193: 1
     150: 8     194: 8
     151: 1     195: 1
     152: 8     196: 8
     153: 1     197: 1
     154: 8     198: 8
     155: 1     199: 1
     156: 8     200: 8
     157: 1     201: 1
     158: 8     202: 8
     159: 1     203: 1
     160: 8     204: 8
     161: 1     205: 1
     162: 8     206: 8
     163: 1     207: 1
     164: 8     208: 8
     165: 1     209: 1
     166: 8     210: 8
     167: 1     211: 1
     168: 8     212: 8
     169: 1     213: 1
     170: 8     214: 8
     171: 1     215: 1
     172: 8     216: 8
     173: 1     217: 1
     174: 8     218: 8
     175: 1     219: 1
     176: 8     220: 8
     177: 1     221: 1
     178: 8     222: 8
     179: 1     223: 1
     180: 8     224: 8
     181: 1     225: 1
     182: 8     226: 8
     183: 1     227: 1
```

W-Array

A = 15 | B = 73 | C = 51

Results for first 3 iterations:

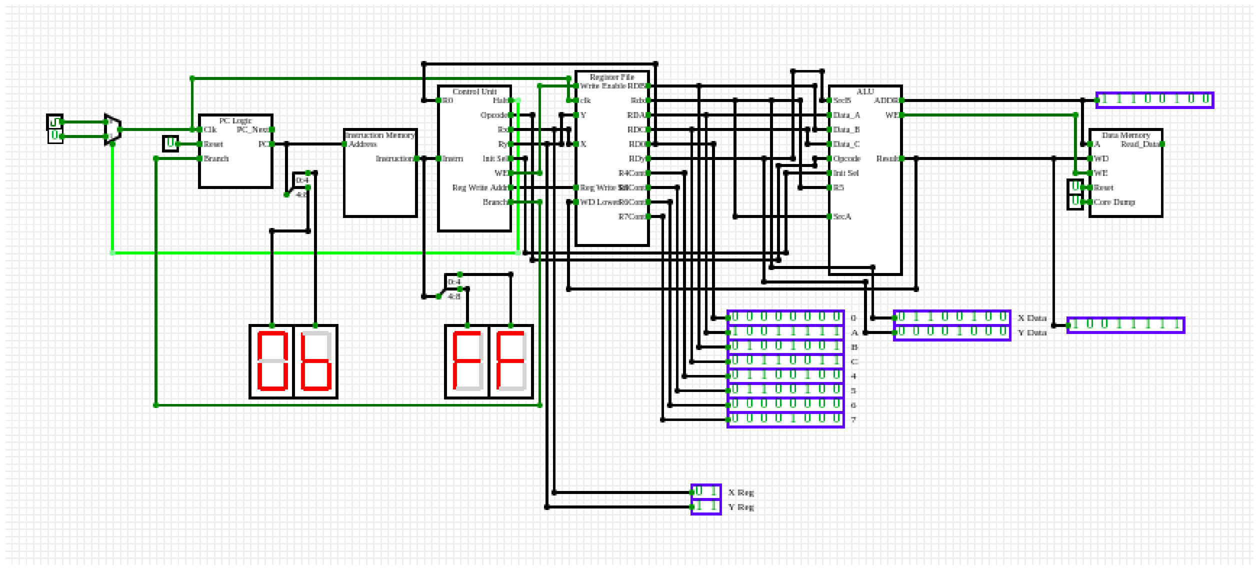
A1 = 107 → W1 = 7

A1 = -121 → W1 = 8

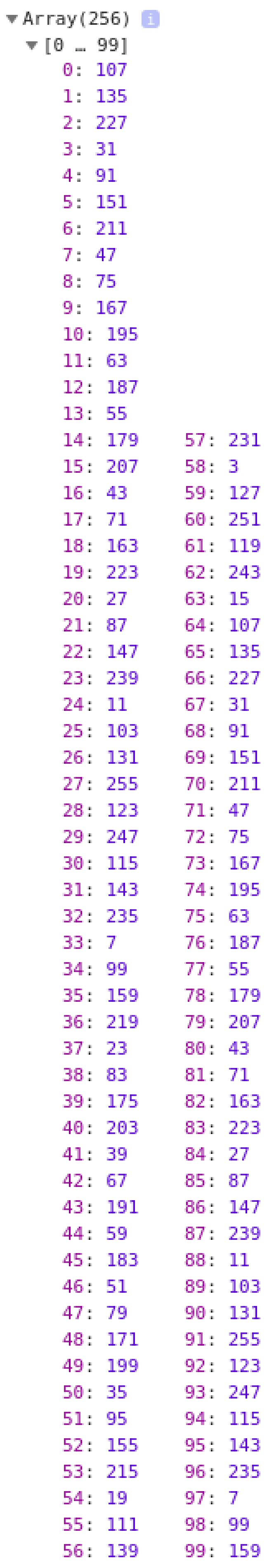
A1 = -29 → W1 = 8

https://youtu.be/nGNL8TWLnOY

At final iteration (stopped by halt)

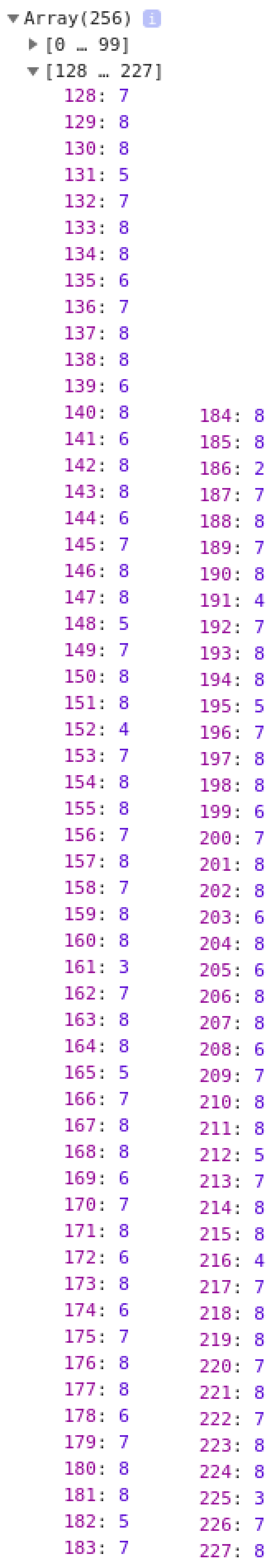


Core Dump V2 below:



```
▼Array(256) i
 ▼[0 … 99]
    0: 107
    1: 135
    2: 227
    3: 31
    4: 91
    5: 151
    6: 211
    7: 47
    8: 75
    9: 167
    10: 195
    11: 63
    12: 187
    13: 55
    14: 179      57: 231
    15: 207      58: 3
    16: 43       59: 127
    17: 71       60: 251
    18: 163      61: 119
    19: 223      62: 243
    20: 27       63: 15
    21: 87       64: 107
    22: 147      65: 135
    23: 239      66: 227
    24: 11       67: 31
    25: 103      68: 91
    26: 131      69: 151
    27: 255      70: 211
    28: 123      71: 47
    29: 247      72: 75
    30: 115      73: 167
    31: 143      74: 195
    32: 235      75: 63
    33: 7        76: 187
    34: 99       77: 55
    35: 159      78: 179
    36: 219      79: 207
    37: 23       80: 43
    38: 83       81: 71
    39: 175      82: 163
    40: 203      83: 223
    41: 39       84: 27
    42: 67       85: 87
    43: 191      86: 147
    44: 59       87: 239
    45: 183      88: 11
    46: 51       89: 103
    47: 79       90: 131
    48: 171      91: 255
    49: 199      92: 123
    50: 35       93: 247
    51: 95       94: 115
    52: 155      95: 143
    53: 215      96: 235
    54: 19       97: 7
    55: 111      98: 99
    56: 139      99: 159
```

A-array



```
▼Array(256) i
 ▶[0 … 99]
 ▼[128 … 227]
    128: 7
    129: 8
    130: 8
    131: 5
    132: 7
    133: 8
    134: 8
    135: 6
    136: 7
    137: 8
    138: 8
    139: 6
    140: 8        184: 8
    141: 6        185: 8
    142: 8        186: 2
    143: 8        187: 7
    144: 6        188: 8
    145: 7        189: 7
    146: 8        190: 8
    147: 8        191: 4
    148: 5        192: 7
    149: 7        193: 8
    150: 8        194: 8
    151: 8        195: 5
    152: 4        196: 7
    153: 7        197: 8
    154: 8        198: 8
    155: 8        199: 6
    156: 7        200: 7
    157: 8        201: 8
    158: 7        202: 8
    159: 8        203: 6
    160: 8        204: 8
    161: 3        205: 6
    162: 7        206: 8
    163: 8        207: 8
    164: 8        208: 6
    165: 5        209: 7
    166: 7        210: 8
    167: 8        211: 8
    168: 8        212: 5
    169: 6        213: 7
    170: 7        214: 8
    171: 8        215: 8
    172: 6        216: 4
    173: 8        217: 7
    174: 6        218: 8
    175: 7        219: 8
    176: 8        220: 7
    177: 8        221: 8
    178: 6        222: 7
    179: 7        223: 8
    180: 8        224: 8
    181: 8        225: 3
    182: 5        226: 7
    183: 7        227: 8
```

W-array

Appendix:

```
#################################################
# ECE 366 Project 3                             #
# Maxwell Thimmig, Charlie Shafer, Jim Palomo   #
#################################################

import os.path as Path

# twoscomp: 2's Complement [return binary (string)]    (string -> string)
# Input: s = binary (string)
# Return: 2's complement binary (string)
def twoscomp(s):
    for j in reversed(range(len(s))):
        if s[j] == '1':
            break

    t = ""
    for i in range(0, j, 1):         # flip everything
        t += str(1-int(s[i]))

    for i in range(j, len(s), 1):    # until the first 1 from the right
        t += s[i]

    return t                         # return 2's complement binary (string)

# twoscomp_dec: 2's Complement [return decimal (int)]     [use for sign
extend]
# Input: b = binary (string)
# Return: 2's complement decimal (int)
def twoscomp_dec(b):

    l = len(b)          # length of bit provided

    x = b[:1].zfill(l)  # save the first bit and fill with 0's until
original length
    x = x[::-1]         # flip binary

    x = int(x, 2) * -1  # value of binary (unsigned: 10000..0) * -1

    y = int(b[1:], 2)   # value of binary without the first bit
```

```python
   x += y              # add up differing values

   return x            # return 2's complement decimal (int)

# bin_to_dec: convert binary (string) to decimal (int)  [use for sign
extend]
# Input: binary (string)
# Return: Decimal (int)
def bin_to_dec(b):
   if(b[0]=="0"):
       return int(b, base=2)
   else:
       return twoscomp_dec(b)


# zero_extend: zero extend / unsigned operation (for specific operations)
# Input: binary (string)
# Return: decimal (int)
def zero_extend(b):
   return int(b, base=2)   # given a binary string, get unsigned decimal

# Integer to binary

# itosbin: convert integer (int) to signed binary (string)
# Input: i = integer (int) | n = # of bits of desired binary
# Return: returns signed binary (string)
def itosbin(i, n):
   s = ""
   if i >= 0:
       s = bin(i)[2:].zfill(n)
   else:
       s = bin(0-i)[2:].zfill(n)
       s = twoscomp(s)

   return s

# hex_to_bin: convert hex (string) to binary (string)
# Input: line = hex (string)
# Return: unsigned binary (string)
```

```python
def hex_to_bin(line):
   h = line.replace("\n", "")
   i = int(h, base=16)
   b = bin(i)
   b = b[2:].zfill(8)
   return b

# neg_int_to_hex:
# Input: x = input integer (int)
# Return: x = 2's complemented hexadecimal (string)
def neg_int_to_hex(x):
   x = bin(x & 0xffffffff)[2:]
   x = hex(int(x,2))[2:].zfill(2)
   x = "0x" + x

   return x

# int_to_hex: convert an decimal (int) to hex (string)
# Input: x = input integer (int)
# Return: hex (string)
def int_to_hex(x):
   if (x < 0):
       x = neg_int_to_hex(x)
   else:
       x = "0x" + str(hex(x))[2:].zfill(8)

   return x

# xor8: perform logic XOR on two 8 bit binary (strings)
# Input: x, y = 8-bit binary (strings)
# Return: XORed binary string
def xor8(x, y):
   s = ""
   for i in range(8):
       if x[i] == y[i]:
           s += '0'
       else:
           s += '1'

   return s
```

```
# findWidth: special instruction used to find the width of a 8-bit binary
code
# Input: s = decimal (string)
# Return: width of the specific decimal translated in 8-bit binary
def findWidth(s):               # take in decimal string
   b = int(s)                  # convert string decimal to int decimal
   b = itosbin(b, 8)          # convert integer to binary
   return len(b.strip("0"))    # strip surrounding zeros from 1...1 &
return length of remaining bits

# processR: process R-type instructions from machine code (string) to hex
(string)
# Input: b = 8 bit binary instruction
# Return: equivalent instruction in hex (string)
def processR(b):
   b_op = b[0:4]
   b_rx = b[4:6]
   b_ry = b[6:8]

   asm = ""

   if (b_op == '0100'):     # SLT (sets $R0 to 0 or 1)
       b_rx = b[4:8]
       rx = int((b_rx), base=2)

       rx = "$5, $4"

       asm = "slt " + rx

   elif (b_op == '1011'):     # SB
       b_rx = b[4:5]
       b_ry = b[5:8]
       rx = int((b_rx), base=2)
       ry = int((b_ry), base=2)

       rx = "$" + str(rx)
       ry = "$" + str(ry)

       asm = "sb " + rx + ", " + "0(" + ry + ")"
```

```python
    elif (b_op == '0110'):     # SW [store width]
        b_rx = b[4:8]
        rx = int((b_rx), base=2)

        rx = "$" + str(rx)

        asm = "sw " + rx

    elif (b_op == '0101'):    # inc
        # b_rx = b[4:8]               # inc R5
        rx  = int((b_rx), base=2)

        rx = "$5"

        asm = "inc "+ rx

    else:
        print (f'NO idea about op = {b_op}')

    return asm

# processB: process B-type instructions from machine code (string) to hex (string)
# Input: b = 8-bit binary instruction
# Return: equivalent instruction in hex (string)
def processB(b):
    b_op = b[0:4]
    imm  = b[4:8]

    asm = ""

    if (b_op == '1010'):        # BO ($R0 = 1 --> branch PC += -28; else no branch & PC += 1)
        imm = -6

        imm = str(imm)          # hardwired to -28

        asm = "bo " + imm       # compares R0 {if R0 == 1 --> branch; else no branch}
```

```
   else:
       print (f'NO idea about op = {b_op}')

   return asm

# processS: process S-type instructions from machine code (string) to hex
(string)
# Input: b = 8-bit binary instruction
# Return: equivalent instruction in hex (string)
def processS(b):
   b_op = b[0:4]
   b_rx = b[4:6]
   b_ry = b[6:8]

   asm = ""

   if (b_op == '1000'):          # FW [find width]
       rx = int((b_rx), base=2)

       rx = "$7"

       asm = "fw " + rx

   elif (b_op == '1001'):        # GA [generate A: (A + B) xor C]
       asm = "ga "               # stores into R1 [R1 = A]

   else:
       print (f'NO idea about op = {b_op}')

   return asm

# processInit: process B-type instructions from machine code (string) to
hex (string)
# Input: b = 8-bit binary instruction
# Return: equivalent instruction in hex (string)
def processInit(b):
   b_op = b[0:2]
   b_rx = b[2:5]
   imm  = b[5:8]

   asm = ""

   options = [100, 1, -1, 15, 73, 51]
   imm = options[int(imm, 2)]

   rx  = int((b_rx), base=2)
   rx = "$" + str(rx)

   imm = str(imm)

   asm = "init " + rx + ", " + imm

   return asm

# process: determine whether the process provided by machine code (string)
is B, R, I type.
# Input: b = 8 bit binary instruction | halt = halt variable used to stop
the program
# Return: MIPS equivalent instruction in hex (string) after determining
instruction type
def process(b, halt):

   if (halt != 1):
       b_op = b[0:4]

       if (b_op[:2] == '00'):                                         #
Init
           return processInit(b)
       elif (b_op == '1111'):                                         #
Halt
           halt = 1
           return "halt"
       elif (b_op == '1010'):                                         #
B-type (bo)
           return processB(b)
       elif (b_op == '1000' or b_op == '1001'):                       #
S-type (ga/fw)
           return processS(b)
```

```
        else:
            return processR(b)                                          #
R-type
    else:
        return # nothing since program stopped                          #
Program was halted

# disassemble: disassembles 8 bit instructions from input .txt file and
appends spliced instruction to a list
# Input: input_file = input .txt file (of 8-bit machine code) | asm_instr
= output .txt file (hex equivalent of machine code) | halt = stops program
# Return: list (instr) of all instructions from .txt file
def disassemble(input_file, asm_instr, halt):
    instr = []    # create empty list of user inputs

    ''' reasons for list:
            1. able to append at the end of list to KEEP ORDER
            2. mutable (change elements in list if necessary)
            3. creating a list data structure using string methods
(replace, split)
    '''

    line_count = 0
    if (halt != 1):
        # convert 8 bit machine code from input and write to output file
        for line in input_file:
            line_count += 1
            # bin_str = hex_to_bin(line)
            bin_str = line                              # currently only
testing binary
            asmline = process(bin_str, halt)
            output_file.write(asmline + '\n')

            # splice asmline using string methods and append spliced
instruction to list
            asm_instr.append(asmline)                   # save asmline into
asm_instr
            asmline = asmline.replace(",", " ")
            asmline = asmline.replace("  ", " ")
            asmline = asmline.replace("$", "")
```

```
           asmline = asmline.replace("128(", "")
           asmline = asmline.replace("(", "")
           asmline = asmline.replace(")", "")
           asmline = asmline.split(" ")
           instr.append(asmline)                       # append to another
list which results in a list-list data structure

   else:
       input_file.close()
       return instr

   output_file.write("\n") # newline in output file to show finished
   input_file.close()      # close input file since we no longer need it

   return instr            # return list of listed spliced instructions
(list-list)

# outputRegisters: output all 8 registers + pc
# Input: reg = array holding register data | pc = special registers
# Return: outputted registers via console & output file
def outputRegisters(reg, pc, hexValue):
   pReg = "Register"
   pVal = "Value"
   print(f"{pReg:<15}{pVal:^12}")

   # output header output file
   row_item = [pReg, pVal]
   output = '{:<15}{:^12}'.format(row_item[0], row_item[1])
   output_file.write(output + "\n")

   # output 32 registers from reg array
   for i in range(len(reg)):
       pReg = "$" + str(i)
       if (hexValue == 0):
           pVal = str(reg[i])
       else:
           pVal = int_to_hex(reg[i])
       print(f"{pReg:<15}{pVal:>12}")

       # output to txt file
```

```
        row_item = [pReg, pVal]
        output = '{:<15}{:>12}'.format(row_item[0], row_item[1])
        output_file.write(output + "\n")

    # output special registers
    pReg = "pc"
    if (hexValue == 0):
        pVal = str(pc)
    else:
        pVal = int_to_hex(pc)

    print(f"{pReg:<15}{pVal:>12}")

    row_item = [pReg, pVal] # output to txt file
    output = '{:<15}{:>12}'.format(row_item[0], row_item[1])
    output_file.write(output + "\n")

    print("\n")

# outputDataMem: output data memory array in similar format as MARS
# Input: mem = data memory array | hex_start = starting memory address |
hex_end = ending memory address
#         address = user selected hex or decimal address output |  value =
user selected hex or decimal value output
# Output: outputted data memory array in console and output file
def outputDataMem(mem, hex_start, hex_end, address, value):
    addr = v1 = v2 = v3 = v4 = v5 = v6 = v7 = v8 = ""

    # headers
    if (address == 0):       # [decimal address]
        addr = "Address"
        v1 = "Value (+0)"
        v2 = "Value (+1)"
        v3 = "Value (+2)"
        v4 = "Value (+3)"
        v5 = "Value (+4)"
        v6 = "Value (+5)"
        v7 = "Value (+6)"
        v8 = "Value (+7)"
        # output header to output .txt file [decimal]
        row_item = [addr, v1, v2, v3, v4, v5, v6, v7, v8]
        output =
'|{:>10}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|'.format(
row_item[0], row_item[1], row_item[2], row_item[3], row_item[4],
row_item[5], row_item[6], row_item[7], row_item[8])
        output_file.write(output + "\n")

    else:                   # [hexadecimal address]
        addr = "Address"
        v1 = "Value (+0)"
        v2 = "Value (+4)"
        v3 = "Value (+8)"
        v4 = "Value (+c)"
        v5 = "Value (+10)"
        v6 = "Value (+14)"
        v7 = "Value (+18)"
        v8 = "Value (+1c)"

        # output header to output .txt file [hex]
        row_item = [addr, v1, v2, v3, v4, v5, v6, v7, v8]
        output =
'|{:^10}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|'.format(
row_item[0], row_item[1], row_item[2], row_item[3], row_item[4],
row_item[5], row_item[6], row_item[7], row_item[8])
        output_file.write(output + "\n")


print(f"|{addr:>15}|{v1:>15}|{v2:>15}|{v3:>15}|{v4:>15}|{v5:>15}|{v6:>15}|
{v7:>15}|{v8:>15}|")

    j = 0

    if (value == 0):        # data memory [decimal values]
        for i in range(hex_start, hex_end, 1):
            if (j % 8 == 0):
                if (address == 1):
                    # addr = "0x" + str(hex(j*4 + 0x0))[2:].zfill(8)
                    addr = "0x" + str(hex(j + 0x0))[2:].zfill(8)
                else:
```

```
            # addr = str(j*4 + 0x0)
            addr = str(j + 0x0)

        if j < len(mem):
            v1 = str(mem[j])
        else:
            v1 = 0

        if j+1 < len(mem):
            v2 = str(mem[j+1])
        else:
            v2 = 0

        if j+2 < len(mem):
            v3 = str(mem[j+2])
        else:
            v3 = 0

        if j+3 < len(mem):
            v4 = str(mem[j+3])
        else:
            v4 = 0

        if j+4 < len(mem):
            v5 = str(mem[j+4])
        else:
            v5 = 0

        if j+5 < len(mem):
            v6 = str(mem[j+5])
        else:
            v6 = 0

        if j+6 < len(mem):
            v7 = str(mem[j+6])
        else:
            v7 = 0

        if j+7 < len(mem):
            v8 = str(mem[j+7])
```

```
               else:
                   v8 = 0


print(f"|{addr:>15}|{v1:>15}|{v2:>15}|{v3:>15}|{v4:>15}|{v5:>15}|{v6:>15}|
{v7:>15}|{v8:>15}|")
               row_item = [addr, v1, v2, v3, v4, v5, v6, v7, v8]
               output =
'|{:>10}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|'.format(
row_item[0], row_item[1], row_item[2], row_item[3], row_item[4],
row_item[5], row_item[6], row_item[7], row_item[8])
               output_file.write(output + "\n")

           j += 1

   else:        # data memory [hexadecimal values]
       for i in range(hex_start, hex_end, 1):
           if (j % 8 == 0):
               if (address == 1):
                   # addr = "0x" + str(hex(j*4 + 0x0))[2:].zfill(8)
                   addr = "0x" + str(hex(j + 0x0))[2:].zfill(8)
               else:
                   # addr = str(j*4 + 0x0)
                   addr = str(j + 0x0)

               if j < len(mem):
                   if (mem[j] < 0):
                       v1 = neg_int_to_hex(mem[j])
                   else:
                       v1 = "0x" + str(hex(mem[j]))[2:].zfill(8)
               else:
                   v1 = "0x" + "".zfill(8)

               if j+1 < len(mem):
                   if (mem[j+1] < 0):
                       v2 = neg_int_to_hex(mem[j+1])
                   else:
                       v2 = "0x" + str(hex(mem[j+1]))[2:].zfill(8)
               else:
                   v2 = "0x" + "".zfill(8)
```

```
            if j+2 < len(mem):
                if (mem[j+2] < 0):
                    v3 = neg_int_to_hex(mem[j+2])
                else:
                    v3 = "0x" + str(hex(mem[j+2]))[2:].zfill(8)
            else:
                v3 = "0x" + "".zfill(8)

            if j+3 < len(mem):
                if (mem[j+3] < 0):
                    v4 = neg_int_to_hex(mem[j+3])
                else:
                    v4 = "0x" + str(hex(mem[j+3]))[2:].zfill(8)
            else:
                v4 = "0x" + "".zfill(8)

            if j+4 < len(mem):
                if (mem[j+4] < 0):
                    v5 = neg_int_to_hex(mem[j+4])
                else:
                    v5 = "0x" + str(hex(mem[j+4]))[2:].zfill(8)
            else:
                v5 = "0x" + "".zfill(8)

            if j+5 < len(mem):
                if (mem[j+5] < 0):
                    v6 = neg_int_to_hex(mem[j+5])
                else:
                    v6 = "0x" + str(hex(mem[j+5]))[2:].zfill(8)
            else:
                v6 = "0x" + "".zfill(8)

            if j+6 < len(mem):
                if (mem[j+6] < 0):
                    v7 = neg_int_to_hex(mem[j+6])
                else:
                    v7 = "0x" + str(hex(mem[j+6]))[2:].zfill(8)
            else:
                v7 = "0x" + "".zfill(8)
                if j+7 < len(mem):
                    if (mem[j+7] < 0):
                        v8 = neg_int_to_hex(mem[j+7])
                    else:
                        v8 = "0x" + str(hex(mem[j+7]))[2:].zfill(8)
                else:
                    v8 = "0x" + "".zfill(8)


print(f"|{addr:>15}|{v1:>15}|{v2:>15}|{v3:>15}|{v4:>15}|{v5:>15}|{v6:>15}|
{v7:>15}|{v8:>15}|")
                row_item = [addr, v1, v2, v3, v4, v5, v6, v7, v8]
                output =
'|{:<10}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|{:>15}|'.format(
row_item[0], row_item[1], row_item[2], row_item[3], row_item[4],
row_item[5], row_item[6], row_item[7], row_item[8])
                output_file.write(output + "\n")




            j += 1
    print("\n")

# outputInstrStats: output instruction statistics (ALU, Jump, branch,
memory, other)
# Input: total, alu, jump, branch, memory, other = current values that are
held for each count variable
# Return: output instruction statistics on console and output .txt file
def outputInstrStats(total, alu, jump, branch, memory, other):
    print("Instruction Statistics, Version 1.0")
    output_file.write("\n\nInstruction Statistics, Version 1.0" + "\n")

    print(f"Total:\t{total}\n")
    output_file.write(f"Total:\t{total}\n\n")

    titles = ["ALU:", "Jump:", "Branch:", "Memory:", "Other:"]
    values = [alu, jump, branch, memory, other]
```

```
   percentages = [(alu/total)*100, (jump/total)*100, (branch/total)*100,
(memory/total)*100, (other/total)*100]
   i = 0
   while i < len(titles):
       print(f"{titles[i]:<8}{values[i]:<8}{percentages[i]:.0f}%")

output_file.write(f"{titles[i]:<8}{values[i]:<8}{percentages[i]:.0f}%" +
"\n")

       i += 1

# Main ----------------------------------------------------------------------

print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< Project 3
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n")

input_file = input("Enter input file> ")

file_exists = 0                                         # temp variable to
keep track of file exist state

# check if input file exists
while (file_exists != 1):
   if Path.isfile(input_file): # file exists
       print("File sucessfully loaded")
       input_file = open(input_file, "r")              # open input file
in read mode (r)
       file_exists = 1                                 # file exists so
set true
   else: # file does not exist, so ask for valid file
       print("File does not exist")
       file_exists = 0                                 # file does not
exists so set false
       input_file = input("Enter input file> ")

output_file = input("Enter desired output file> ")      # ask for user
output file name
output_file = open(output_file,"w")                     # create and open
output file in write mode (w)
print()
```

```
# hardcoded for testing purposes
# input_file = open("input.txt", "r")
# output_file = open("asm.txt", "w")

asm_instr = []

halt = 0

instr = disassemble(input_file, asm_instr, halt)

reg = [0] * 8         # four available register (00 = $0 | 01 = $1 | 10 =
$2 | 11 = $3)

mem = [0] * 256       # data memory

line = pc = 0

# instruction statistics
total = alu = jump = branch = memory = other = 0

# output header
pLine = "line"
pInstr = "Instruction"
pResult = "Result"
pPC = "PC"

print(f"{pLine:<15}{pInstr:<35}{pResult:<25}{pPC:<15}")
output_file.write(f"{pLine:<15}{pInstr:<35}{pResult:<25}{pPC:<15}" + "\n")

while (pc < len(instr)):
   cur = instr[pc]                        # give access to instr[] list-list
                                          # first list: separated machine code
instructions (access to opcode, rs, rt, rd, sa, func, imm)
                                          # second list: holds the first list
within itself and replicates "line numbers"

   line += 1                              # update for next instruction in instr
list

   if (cur[0] == "bo"):    # branch
```

```
            pInstr = asm_instr[pc]
            if (reg[0] == 1):         # $0 == 1?
                pc += -6              # pc = pc + imm
            else:
                pc += 1               # pc = pc + 1

            branch += 1

            pResult = "branch to PC " + str(pc)

    # must be fw/ga or sb/sw
    elif (cur[0] == "sb" or cur[0] == "sw" or cur[0] == "fw" or cur[0] ==
"ga" or cur[0] == "slt"):
        if (cur[0] == "sb"):        # SB
            mem[reg[5]] = reg[1]

            pResult = "DM[" + str(reg[int(cur[2])]) + "] = " +
str(reg[int(cur[1])])
            memory += 1

        elif (cur[0] == "sw"):       # SW
            mem[reg[5] + 128] = reg[7]

            pResult = "DM[" + str(reg[5] + 128) + "] = " + str(reg[7])
            memory += 1

        elif (cur[0] == "fw"):       # FW
            reg[7] = findWidth(reg[1])      # R1 = width of 8-bit

            pResult = "Width of A -> $7 = " + str(reg[7])
            alu += 1

        elif (cur[0] == "ga"):      # GA
            w = reg[1]
            x = reg[2]
            y = reg[3]
            z = (reg[1] + reg[2]) ^ reg[3]
            # reg[1] = (reg[1] + reg[2]) ^ reg[3]
            reg[1] = bin_to_dec(xor8(itosbin((reg[1] + reg[2]), 8),
itosbin(reg[3], 8)))
```

```
                pResult = "A = " + str(reg[1])
                alu += 1

            elif (cur[0] == "slt"):     # SLT
                if (reg[5] < reg[4]):  # if x < y
                    reg[0] = 1                             # R0 = 1
                else: # x > y
                    reg[0] = 0                             # R0 = 0

                pResult = str(reg[5]) + " < " + str(reg[4]) + " --> $0" + " = "
+ str(reg[0])
                other += 1

            pInstr = asm_instr[pc]
            pc += 1

        else:   # not fw/ga or sb/sw or branch
            if (cur[0] == "init"):      # INIT

                reg[int(cur[1])] = int(cur[2])

            elif (cur[0] == "inc"):    # INC
                reg[5] = reg[5] + 1

            elif (cur[0] == 'halt'):       # HALT
                print

            else:
                print("Instruction not implemented")

            pc += 1

            if (cur[0] == 'halt'):
                other += 1
            else:
                alu += 1

            if (cur[0] != 'halt'):  # HALT
                pResult = "$" + cur[1] + " = " + str(reg[int(cur[1])])
```

```
       else:
           pInstr = "HALT!!"
           pResult = "Program Stopped"
           pLine = line
           pPC = pc

           print(f"{pLine:<15}{pInstr:<35}{pResult:<25}{pPC:<15}")

output_file.write(f"{pLine:<15}{pInstr:<35}{pResult:<25}{pPC:<15}" + "\n")
           break

       pInstr = asm_instr[pc-1]

   pLine = line
   pPC = pc

   print(f"{pLine:<15}{pInstr:<35}{pResult:<25}{pPC:<15}")
   output_file.write(f"{pLine:<15}{pInstr:<35}{pResult:<25}{pPC:<15}" +
"\n")


print()
output_file.write("\n")

# output registers
outputRegisters(reg, pc, 0)

# output Data Memory
output_file.write("\n")
outputDataMem(mem, 0x0, 0x100, 0, 0)

# output instruction statistics
total = alu + jump + branch + memory + other
outputInstrStats(total, alu, jump, branch, memory, other)
```