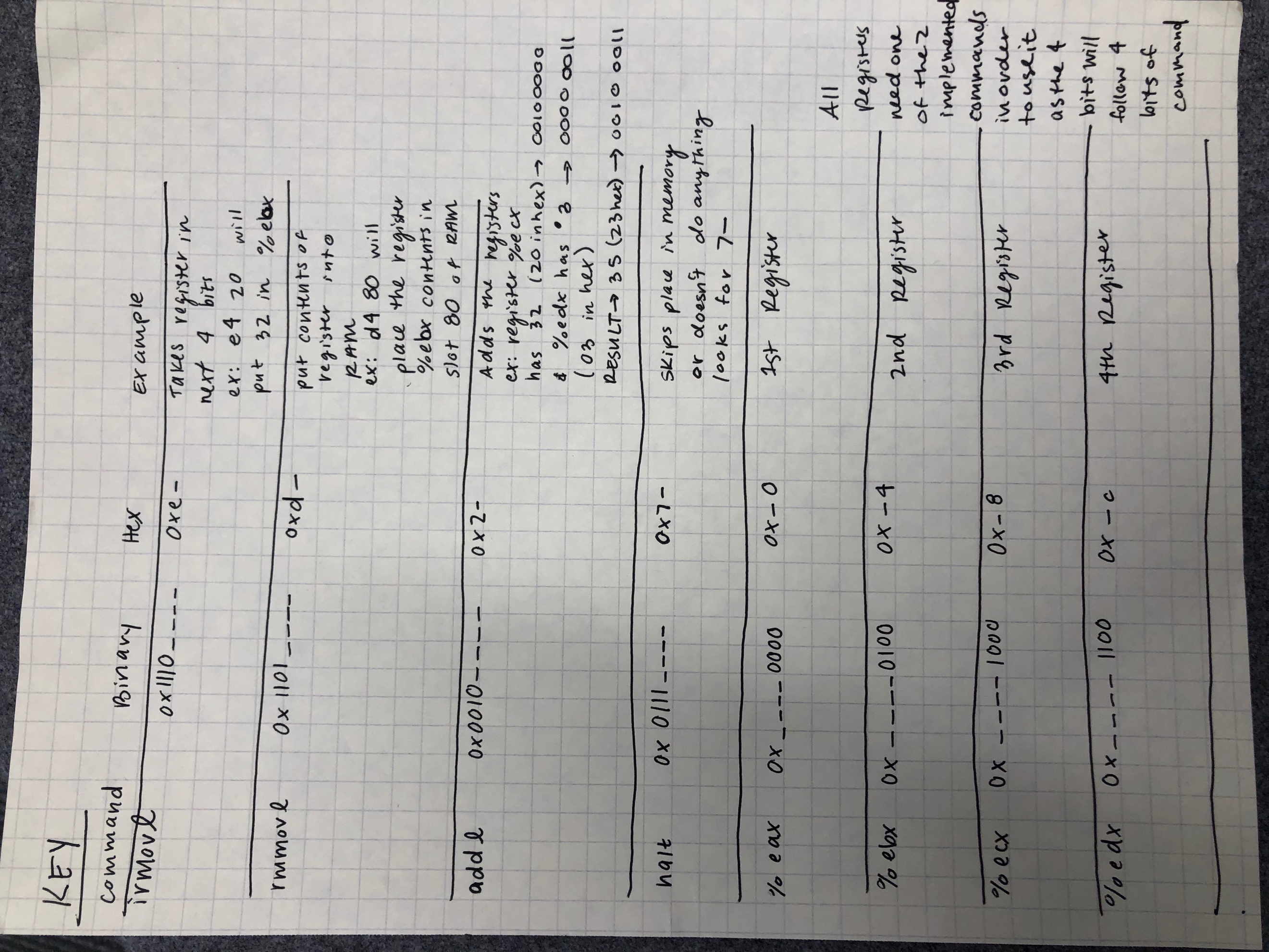
**Final Project Report**

*Steven Barringer, Lauren Ruffner, Turner Levey GROUP 22*

**Problem 1. Design a minimal instruction set architecture for an 8 bit load-store processor with appropriate binary encoding. Describe the ISA in detail in the report with all your design assumptions and rationale.**

*Ref) Detailed description of your Instruction Set Architecture covering the following points*

1. *A table showing the all the instructions you implemented, along with their op­codes.*
2. *For each instruction, explain what the instruction does along with the format of the instruction in terms of bit width – such as which bits represent the op­codes, which bits represent the values being read in (if applicable).*
   1. *The file that holds the hex code to save in the RAM is classed TEST ADD*

**Problem 2. Using the assembly language code that you designed in the previous step, create a program to perform the matrix addition.**

*Ref) Attach your Y86 assembly code, and corresponding captured output screen (generated by YIS). Clearly mention your assumption for matrix A, matrix B, and expected output matrix C.*

0x0000: | Main:

0x0000: 30f001000000 | irmovl $1, %eax

0x0006: 30f300000000 | irmovl $0, %ebx

0x000c: 400300000000 | rmmovl %eax, 0(%ebx)

0x0012: 30f002000000 | irmovl $2, %eax

0x0018: 400304000000 | rmmovl %eax, 4(%ebx)

0x001e: 30f003000000 | irmovl $3, %eax

0x0024: 400308000000 | rmmovl %eax, 8(%ebx)

0x002a: 30f004000000 | irmovl $4, %eax

0x0030: 40030c000000 | rmmovl %eax, 12(%ebx)

|

0x0036: 30f005000000 | irmovl $5, %eax

0x003c: 30f300000000 | irmovl $0, %ebx

0x0042: 400310000000 | rmmovl %eax, 16(%ebx)

0x0048: 30f006000000 | irmovl $6, %eax

0x004e: 400314000000 | rmmovl %eax, 20(%ebx)

0x0054: 30f007000000 | irmovl $7, %eax

0x005a: 400318000000 | rmmovl %eax, 24(%ebx)

0x0060: 30f008000000 | irmovl $8, %eax

0x0066: 40031c000000 | rmmovl %eax, 28(%ebx)

|

|

0x006c: 500300000000. | mrmovl 0(%ebx), %eax

0x0072: 501310000000 | mrmovl 16(%ebx), %ecx

0x0078: 6001 | addl %eax, %ecx

0x007a: 401320000000 | rmmovl %ecx, 32(%ebx)

|

0x0080: 500304000000 | mrmovl 4(%ebx), %eax

0x0086: 501314000000 | mrmovl 20(%ebx), %ecx

0x008c: 6001 | addl %eax, %ecx

0x008e: 401324000000 | rmmovl %ecx, 36(%ebx)

|

0x0094: 500308000000. | mrmovl 8(%ebx), %eax

0x009a: 501318000000 | mrmovl 24(%ebx), %ecx

0x00a0: 6001 | addl %eax, %ecx

0x00a2: 401328000000. | rmmovl %ecx, 40(%ebx)

|

0x00a8: 50030c000000. | mrmovl 12(%ebx), %eax

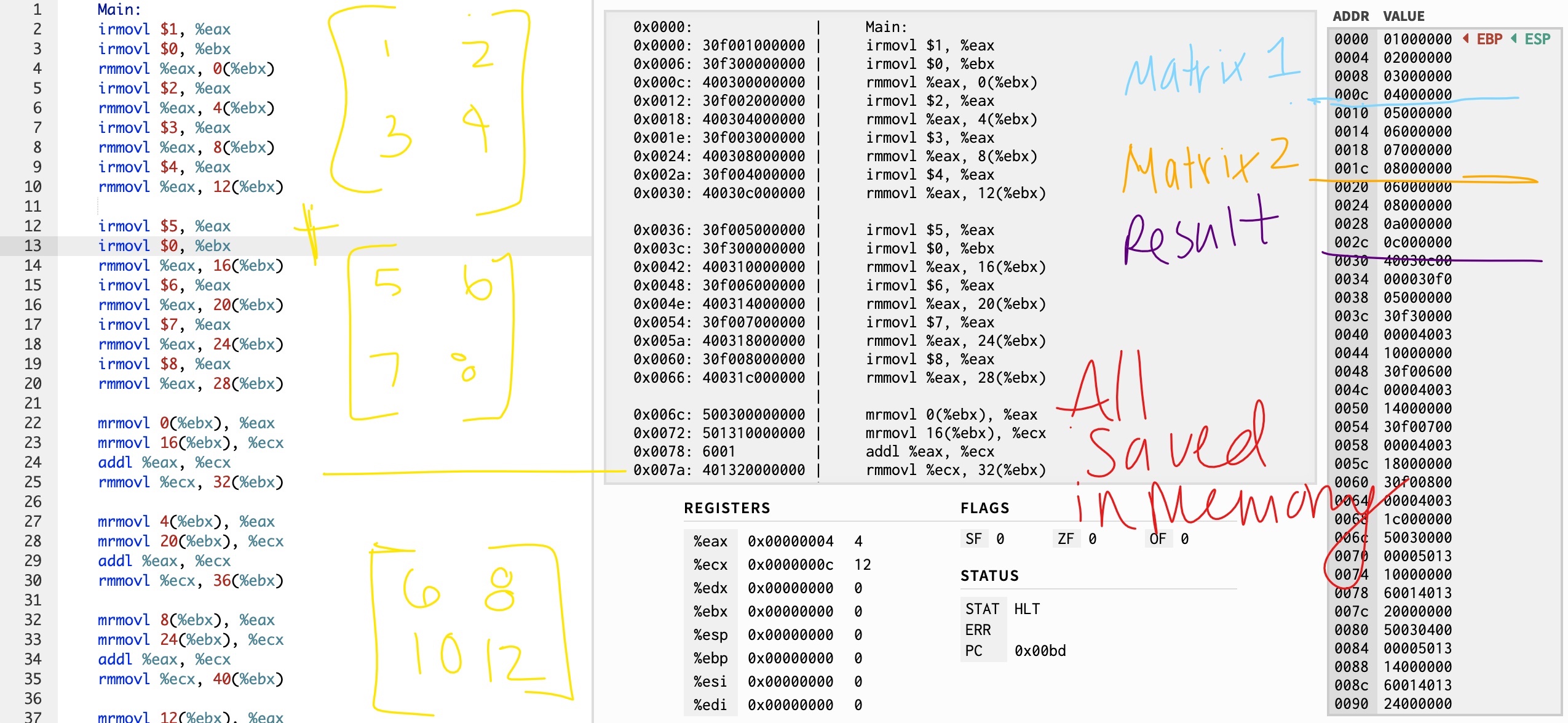
0x00ae: 50131c000000. | mrmovl 28(%ebx), %ecx

0x00b4: 6001 | addl %eax, %ecx

0x00b6: 40132c000000 | rmmovl %ecx, 44(%ebx)

0x00bc: 00 | halt

|



Assumptions for A and B: We assumed each matrix was of dimension 2 by 2.

The numbers in matrix A were {1,3,5,7} respectively. The numbers in matrix B were {2,4,6,8} respectively. These added together to form matrix C which is {3,7,11,15} respectively.

**Problem 3. Design and verify a minimal processor that can execute the ISA in activity # 1***.*

*Only design the bare minimum circuits which are absolutely essential to execute the program that you created in assembly, and design the peripheral input/output circuit(s) for this designed processor, which can interface with the RAM/ROM modules. You can do this by modifying the peripheral circuit which was you implemented in Lab 4. (Program/edit that ROM with the binary code that you developed in step 2, verify its correct execution on your processor.)*

*Ref) For each functional block of the processor.*

1. *Explain its operation. (You may want to insert screen shots from Logisim to help you in your explanation)* 
   1. Ram – Holds all of the hex code and stores the final output in a random location.
   2. Decoder – Takes in input from the hex code and returns something that can be run in registers.
   3. Registers – Hold data for use later. We have 4 registers: eax, ebx, ecx, and edx.
   4. ALU – Adds two 8-bit numbers. It is a 8-bit carry out. Returns valE which is then sent back into the register with an address where it should be saved in memory.
   5. PC Updater – The assembly code is run through the PC updater and the value is updated. Each line of code is run through it in this fashion.
   6. Buffer Registers – The buffers hold the initial instruction until it finds everything it needs.
   7. Register File – Holds the registers. Either valA and valB is one of the outputs of the register file depending on the instructions.
2. *If you split the work of creating these various sub­blocks, clearly demarcate which parts were contributed by each of the members.*

Most of the work was done in lab and divided evenly between the members. The functionality of each of the sub-blocks was contributed to by at least two members simultaneously.

**Problem 5. Final Project Demo in lab to TA, it will be part of your grade.**

*Show your TA how your processor run your program of activity #2.*

Done