HW 2

1.

A.

2^28 = 268435456 bytes of memory. You subtract one since you have 2^N -1 bytes of memory, so the range is 0 to (2^28)-1 separate address sizes including 0.

B.

$FFF because that is able to have the range of 0-4095, which means it can store a 40-bit word size.

2.

Address registers use unsigned arithmetic because they store the addresses in unsigned integers, and thus it would be easier to do arithmetic with the same data type. The address register's main role is to store and as such unsigned integers are just fine for its job. The accumulator's main job is to store arithmetic results and computations and for this reason, it used twos complement to do so. Twos complement is good for a number of reasons, one of them being the fact that it does arithmetic more reliably so, as both subtraction and addition have the same results compared to 1's complement and unsigned arithmetic. This also allows it to spot arithmetic overflow.

3.

The instruction register is used to store CUSP memory addresses, and for this reason, they are 12-bit sized. The IR is used to store the Data from the CUSP, and thus needs 24 bits.

4.

The fetch Decode Execute cycle starts with the instruction fetch, which gets the next instructions from memory. The second step is to increment the PC to the next step, which includes decoding the instructions. The third step is the actual execution of said instruction from step 3, which occurs in the IR. The fourth and final step is to go back to step 1 and start the process again by repeating it. The IR's role is to be loaded with the instructions from the memory of the PC. These instructions are values of contents of the memory at the location instructed by the contents of the PC. The PC's main role is to keep track of the instructions, which is always to increment, and then store the next instruction to be executed. The instructions the PC receives are contents of the PC plus 1.

5.

The CPU has a program counter that iterates through the instructions by PC + 1. The integers pointed out by the PC will be interpreted as instructions and the rest of it is the data. It reads data in this order, and thus is able to get the location or data word.

6.

The contents of the ACC which is $71A7EF, are placed into the location $045. After execution, the ACC still contains $71A7EF, except now the memory location $045 also contains $71A7EF.

HW 2

7.

a. Self-Modifying code: When an instruction cycle shows that the execution is to alter the instruction, and is able to do so while executing.
b. Instruction decoding: allows the CPU to derive what instruction is to be performed. This tells the CPU how many operands it needs.
c. Opcode: The portion of the instruction that identifies what operations need to be performed by the CPU.
d. Addressing mode: tells the CPU how to use addressing values to calculate necessary operands.
e. Flags: EQ, LV, OV flags are set after a result or execution of the instructions. OV happens when overflow occurs, EQ happens as a result is equal to 0, and LT occurs when the result is less than 0.

8.

a.

Direct addressing has the operand be in the memory location which is found by the addressing value. Immediate addressing has the operand be the addressing value at the same time, this means that the operand is right after the addressing instruction.

b.

Arithmetic instruction uses basic operations such as division, multiplication subtraction, addition and etc. This is used best for incrementing and is often used for counters in memory that maintain loop control. This method is a more efficient way to do these tasks compared to ACC. Logical instructions are used for logic-based operands such as and/or, as well as for bit manipulation functions. Each operation is performed individually instead of carrying previous results and interprets operands in a truth table style, making it more suited for this task.

9.

B.

LDA $123 = $000
STA $007 = $004 = $040550
INC $007 = #01B
CMA $008 = $200
JGT $006 = $04B = $000012
DEC $007 = $016
HLT = $FFFFFF

Final Values:

$006 = $000012
$006 = $040550

Sequence: $0D0 200 01B 004 04B 01C FFF FFF

HW 2

D.

LDA# $18E = $000

DIV# $002 = $019

N0P = $FFF038

NEGA = #FFF020

HLT = $FFFFFF

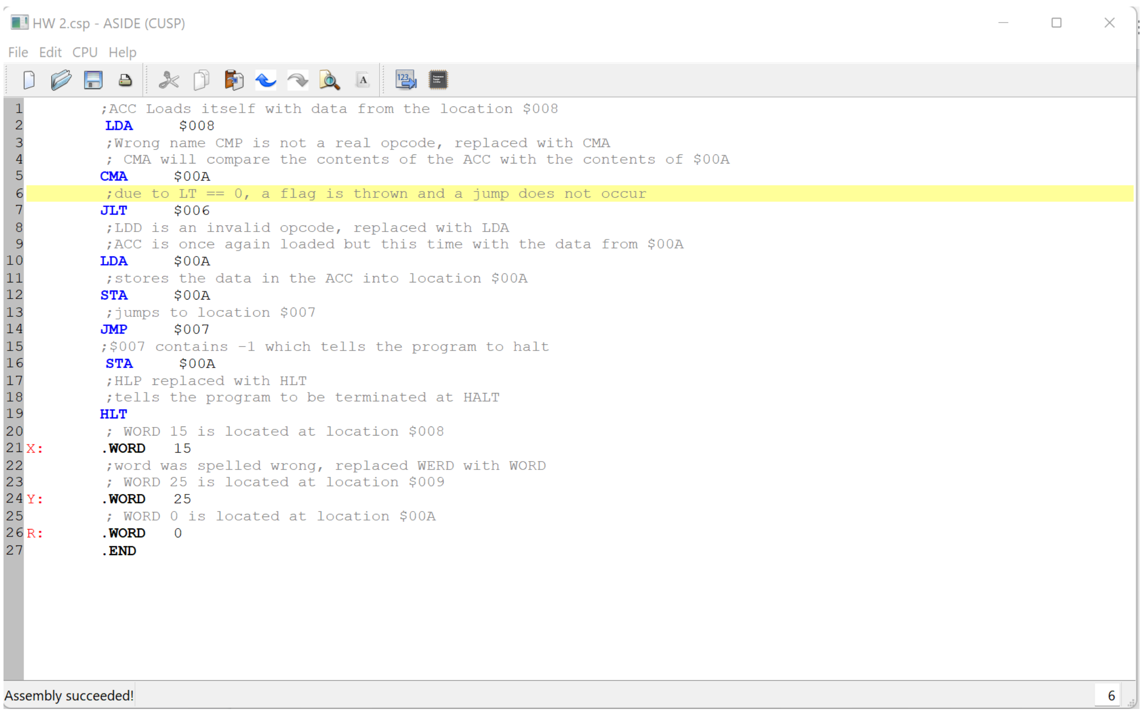
Final Values:

$18E = 256 decimal

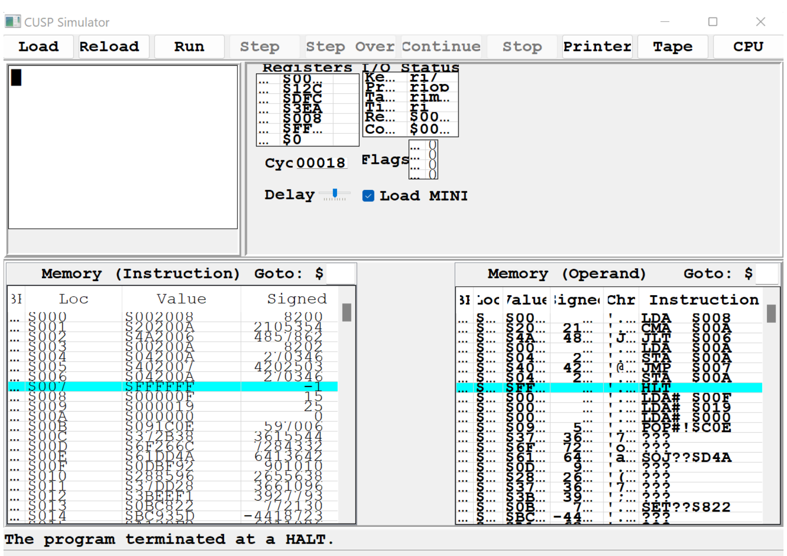
$002 = 18E

Sequence: $ 000 019 FFF 038 FFF 020 FFFFFF

10.



```
;ACC Loads itself with data from the location $008
 LDA     $008
 ;Wrong name CMP is not a real opcode, replaced with CMA
 ; CMA will compare the contents of the ACC with the contents of $00A
CMA     $00A
 ;due to LT == 0, a flag is thrown and a jump does not occur
JLT     $006
 ;LDD is an invalid opcode, replaced with LDA
 ;ACC is once again loaded but this time with the data from $00A
LDA     $00A
 ;stores the data in the ACC into location $00A
STA     $00A
 ;jumps to location $007
JMP     $007
;$007 contains -1 which tells the program to halt
 STA     $00A
 ;HLP replaced with HLT
 ;tells the program to be terminated at HALT
HLT
 ; WORD 15 is located at location $008
X: .WORD    15
 ;word was spelled wrong, replaced WERD with WORD
 ; WORD 25 is located at location $009
Y: .WORD    25
 ; WORD 0 is located at location $00A
R: .WORD    0
.END
```

HW 2