Jeff Ross
CSC 317
Program 3

IMPLEMENTATION
This program is written in C++ using only standard included libraries. The program runs inside of a continuous while(true) loop to infinitely run through instructions until it hits a HALT command to break out. The program first checks first increments the ic by one to indicate the next instruction. the addressing mode of the instruction to see what course of action is to be taken. At this segment, it will take the appropriate course of action take the address of the instruction and turn it into the EA. If an addressing mode which does not exist is called for, it will HALT and output a message.

Following this the Instructions will use the EA to perform their respective commands. The instructions have been split up into four different categories which are identified by bits 10 and 11. If the bits are 00, then it is either NOP, if the op is 1, or HALT if the op is 0. If it is HALT it prints out the contents of registers and accumulator and exits out of the while loop. If it is a NOP it goes to the bottom of the loop where it prints out the contents of the registers and the accumulator.

If bits 10 and 11 are 01 then it indicates that the instructions are memory instructions. The EA will be taken and used to find the place in memory to be interacted with. Special precautions have been taken to throw an illegal addressing mode if the mode is unsupported by the instruction. If bits 10 and 11 are 10 then it indicates mathematical operations. Depending on whether or not it is immediate it will operate with the contents at the location indicated by the address or by the address itself. After the operation is performed a mask is used to return the result to 24 bits if it has gone over. If the operation is a register operation, it masks with 12 bits instead.

If bits 10 and 11 are 11 then it indicates that the instruction is a branching instruction. If the addressing mode is immediate it throws out an illegal addressing mode, otherwise tests the conditions indicated by the instruction and returns either true or false. If true, the ic is updated to the address indicated in the instruction, if false nothing happens. At the end of the loop it prints out all the contents of the accumulator and the registers before returning back to the top.

TESTING
Testing was done using the following examples that have been included in the object files. These examples outline the requirements of testing LDX, STX, EMX, ADDX, SUBX and CLRX, testing LD, ST and EM using indexed and indirect addressing, and testing ADD, SUB, AND, OR and XOR using indexed and indirect addressing. These are respectively present in the included test.txt, test2.txt, and test3.txt files. An outline showing structure behind the examples is shown below, in order.

Jeff Ross
CSC 317
Program 3

```
020a05   Immediate addx 020 reg 0
aaaa01   Direct addx from aaa reg 0
010a45   Immediate subx 010 reg 0
aaba41   Direct subx from aab reg 0
050606   Immediate ldx 050 into reg 2
aac601   Direct ldx aac into reg 1
aac642   stx reg 2 into aac
aac681   emx reg 1 with aac
000a82   clrx reg2
aac602   Direct ldx aac into reg 2

store at aaa
030000
020000
123000
```

```
aaa605   ldx aaa into reg 1
bbb606   ldx bbb into reg 2
002409   ld from 3rd value in array at aaa (indexed)
aaa440   st into 1st value in array at aaa
bbb410   ld value from pointer from bbb to ccc (indirect)
00149a   em pointer to aaa from second value in array at bbb (indexed indirect)
aaa400   ld value from aaa to confirm swap

store at aaa
010000
bbb000
030000

store at bbb (array of pointers)
ccc000
aaa000

store at ccc
040000
```

```
aaa605   ldx aaa into reg 1
bbb606   ldx bbb into reg 2
aaa810   add value of 010 from pointer at aaa (indirect)
002919   and value of 030 from array of pointers (indexed indirect) 1 and 3 = 1
001819   add value of 020 from array of pointers (indexed indirect) 1 add 2 = 3
00194a   or value from second entry in array at bbb (indexed) 3 or 4 = 7
```

Jeff Ross
CSC 317
Program 3

003859 sub value of 050 from array of pointers (indexed indirect) 7 sub 5 = 2
004999 xor value of 060 from array of pointers (indexed indirect) 2 xor 6 = 4

store at aaa (array of pointers) each contains respective # (010 contains 1)
010000
020000
030000
050000
060000

store at bbb (an array containing value for 4)
000000
004000

COMPILATION AND USAGE

Compilation requires the g++ compiler. The program can be used by using “make” from the Makefile which compiles under “g++ -o main main.cpp functions.cpp”. This will create an executable called main.exe. The program takes one input file as a command line argument, so run as “main file.txt”. The files that the program runs on are structured such that the first value of every line is an address for the data to be stored. The next value is the number of pieces of data to be stored contiguously from that address, followed by all the data to be stored. Once all the data has been entered, the last line is just the address to set the program counter to. EVERY NUMBER IN THE FILE MUST BE WRITTEN IN HEXIDECIMAL FOR PROPER OPERATION.