DOCUMENTATION

Load and run a program from command line: *sandbox program.txt*
You can type instructions directly into the sandbox's console or type *load* or *LOAD* and inputting the filename of your program.

<u>Sandbox Instructions</u>

Load	- Asks user for name of program file to load and run
clear	- Clears the screen
clearmem	- Clears the sandbox memory
exit	- Exits the sandbox

ADD result var1 var2	- Addition:	result = var1 + var2
SUB result var1 var2	- Subtraction:	result = var1 - var2
DIV result var1 var2	- Division:	result = var1 / var2
MUL result var1 var2	- Multiplication:	result = var1 * var2
POW result var1 var2	- Raise to power:	result = var1 ^ var2
SQRT result var1	- Square root:	result = sqrt(var1)

*var1 and var2 can be a number or a variable

CP var1 var2	- Copy variable:	var1 = var2
SET var1 var2	- Sets var1:	var1 = var2/number

BEQ jump var1 var2	- If var1 == var2: currentLine += jump
BNE jump var1 var2	- If var1 != var2: currentLine += jump
BGT jump var1 var2	- If var1 > var2: currentLine += jump
BLT jump var1 var2	- If var1 < var2: currentLine += jump
BGE jump var1 var2	- If var1 >= var2: currentLine += jump
BLE jump var1 var2	- If var1 <= var2: currentLine += jump

*Empty lines in program file are counted

PRINT var	- Prints out the value of variable, var

<u>Writing Programs</u>
Write one instruction per line and empty lines are counted when branching.
Variables must have a value being used. For example:

ADD result var1 255

The above instruction will result in an error because var1 has no value.
The following code adds 1 + 255 and stores the value into the variable, result:

ADD result 1 255

If *result* was not initialized, it will be once the ADD instruction executes. If it already had a value, that value will be overwritten. You can then see the value of the *result* variable by inputting:

PRINT result

The following is a multiline example program showing how branching works in the sandbox (Line numbers are shown for easier reading and should not be in the actual program)

```
1 SET aTwo 2
2 SET aThree 3
3 SET aZero 0
4 BLT 3 aTwo aThree
5 PRINT aZero
6 PRINT aZero
7 PRINT aZero
8 PRINT aThree
```

First, 3 variables are created and initialized. Then, on line 4, there is a BLT (Branch-If-Less-Than) instruction that jumps down 3 lines if aTwo < aThree. After the branch instruction is executed in line 4, jumping down 3 lines brings the “program counter” to line 8 where it will execute the PRINT instruction and print out the value of aThree, which is 3. Lines 5,6, and 7 are never executed and the console will not display any 0’s when this program is run.

Example Programs

count.txt

```
1 SET n 10
2 PRINT n
3 ADD n n -1
4 BNE -3 n 0
```

Initially, the variable n is set to 10. In line 2, the value is n is printed. Then, in line 3 n is decremented by subtracting 1 from n’s value and storing it back into n. Line 4 causes the program to branch up 3 lines if n != 0. This repeats until n = 0 so the numbers 10 – 1 are displayed on the console.

fibonacci.txt

```
1 SET n 10
2 SET i 0
3 SQRT root5 5
4 ADD x 1 root5
5 POW x x i
6 SUB y 1 root5
7 POW y y i
8 SUB numerator x y
9 POW 2pow 2 i
10 MUL denominator 2pow root5
11 DIV result numerator denominator
12 PRINT result
13 ADD i i 1
14 BNE -12 i n
```

The variable i is used to keep track of how many times the routine in lines 3 – 11 is called. The routine in those lines is an algebraic method for calculating the ith Fibonacci number. Line 12 prints out the result of the current calculation. The value of i starts at 0 and is incremented in line 13. Line 14 checks if i == n, and if not it branches up 12 lines so it can run the calculation again for the incremented i. This program will print out the first n Fibonacci numbers. (In this case, it's the first 10) The amount of numbers calculated can be changed by altering the value of n in line 1.

Turing Complete

My sandbox is Turing complete because it implements all the basic operations that a simple CPU (Like MIPS) can do. The sandbox can read/write data (SET, CP, etc.), do conditional loops (BEQ, BNE, etc.), all the basic mathematical operations MIPS can do (ADD, SUB) as well as operations MIPS doesn't have an instruction for such as SQRT and POW. Unlike MIPS, my sandbox does not have built-in instructions for basic logic operations like AND and OR, but those can be simulated using the arithmetic operations that are already supported by the sandbox. Therefore, because my sandbox can do everything MIPS can, and since MIPS is proven to be Turing complete, my sandbox is also Turing complete.

Eugene Li
CS9163