Lab 2:: CPU Lab 1
Daquan Smith

CSC 137-02

Theory of Operation:

The overall goal of the FISC assembler is to make a program in Java that uses 5 or in this case 4 instructions in order to make a hex file that can be used in order to run our simulator. While thinking of a way to code the assembler I first found a way to take a file from the user and then read, and parse the file into hex numbers. This took several attempts but the way that I found was the easiest way to take each line and parse them separately into their own separate lists. With this specific assembler being a two-pass assembler I used two methods in order to find the labels and their line numbers first in order to make the branch when zero loops easier to run. In the second method, I used a for loop in order to go through the file until it ended. I then checked each of the inputs and saved them into a string. I used that string in order to check whether it was a specific instruction. If it was a specific instruction and not a comment or a label, the input would then go through the switch statement in order to run the specific code for that instruction. I would then use another string and add the specific bits that are from the table. I would check the whole line next and add the specific bits into the correct places. These bits will be used in order to make the specific hex values. In order to check if the program ran correctly I tested if there were comments that were in the wrong place, I then checked what type of files can be used as an input. After that, I decided to test whether or not my -l worked correctly with the right instructions. Ultimately, I decided to check if there were empty labels or instructions with the incorrect number of registers and see if they would display the correct errors.

For the simulator, we are basically going to be applying the opposite logic that we are using in the assembler. In this specific simulator, we are using a hex file, one from the assembler in this case, and then we are going to be acting out the instructions that are going to be performed on the four registers r0, r1, r2, r3. The way that I went about this specifically is by making an array of integers that hold the specific registers in their rightful positions such as rd, rn, and rm. When we grab the hex files from the file we will then change them into unsigned integers which is what I did in order to get the correct placement of the bits and the specific registers. Before this, I decided to make a series of if-else statements in my main method in order to check whether or not we will be displaying or if the amount of loops that will be done is actually the default, 20. Using the specific opcode bits, I go through and check which instruction is being used for the line that is being operated on. I then checked the other numbers in the list of integers and found out which registers will be used in order to perform the operation. In the end, I display the state of all of the registers and the program counter. I have an if statement in order to see if the actual line will be displayed as a string for disassembly. The way I tested this was by sending in different hex files in different combinations of both disassembly and no disassembly. I also tested bnz by adding in a small hex file and running the loops to some large number like 200 cycles.

Fibo1.s

Write an assembler code that continually computes and outputs Fibonacci numbers (in hex) starting with F(0) = 0, F(1) = 1. You must output both initial values of the sequence in your output. Note that by output you are simply relying on FISCs special ability for R3. This is not implemented in your C++ simulator as it isn't necessary as R3 is shown after every cycle. Do not check the validity of the numbers in the sequence, simply compute them as if the result is valid, and put the result in R3. Document your assembler program with comments. Save your file as fibo1.s

CODE:

```
Start: not r0 r1  ;making reg0 ff
       and r0 r0 r1
       not r1 r0
       add r1 r1 r1  ;multiplying by two
            not r1 r1      ; reg now equals one
end:  add r3 r0 r1  ;storing into reg three
            not r0 r1
            not r0 r0
            not r1 r3
            not r1 r1
            bnz end          ; looping fibo sequence
```

Assembler Output:

```
C:\Users\daqua\Fiscas\Fiscas\main>java Fiscas.java "C:\temp\fibo1.s" "C:\temp\output.hex" -l
***LABEL LIST***
Start 00
end 05
***MACHINE CODE***
00:90 not r0 r1
01:44 and r0 r0 r1
02:81 not r1 r0
03:15 add r1 r1 r1
04:91 not r1 r1
05:7 add r3 r0 r1
06:90 not r0 r1
07:80 not r0 r0
08:B1 not r1 r3
```

1. What is the last Fibonacci number output by your code?

The last Fibonacci number that is output in my code correctly on hex is E9 which in decimal is 233. After this value, we end up getting to values such as 79 in hex which is 121 and above numbers that are not displayed correctly or to the format of the Fibonacci Sequence.

2. How many cycles does it take to compute this number?

It takes my code 72 cycles to reach this value. The way that the assembler is made and the looping is done makes r3 hold that specific value for 6 cycles until it is changed to the next value in the Fibonacci sequence.

3. What value is the output for the first invalid Fibonacci number?

The output for the first invalid Fibonacci number is 79 in hex which is 121 in decimal representation. The numbers seemingly go down after this specific point which is at 78 cycles in the simulator output as shown above in the specific cycle screenshot.

4. Explain why this number is invalid, that is, what goes wrong with the arithmetic computation?

This number is invalid because it is adding the two numbers previous to it which are both stored in r0 and r1. The reason why this is occurring is due to the limitations of what can be represented as hex values. In this specific case we have an issue where we are adding the hexadecimal numbers of E9 and 90 which gives us a hex value that cannot be represented with 8 bits as that is our limitation with our assembler and our simulator. In order to fix an issue like this we would need to use more than one byte per register in order to store all of the specific values needed.

Fibo2.s

Write an assembler code that computes and outputs only the first 12 Fibonacci numbers (in hex) starting with F(0) = 0, F(1) = 1. You must output both initial values of the sequence in your output. Note that by output you are simply relying on FISC's special ability for R3. This is not implemented in your C++ simulator as it isn't necessary as R3 is shown after every cycle. Do not check the validity of the numbers in the sequence, simply compute them as if the result is valid, and put the result in R3. After the printing of the 12th number your program must effectively halt by looping to the address of the final BNZ instruction. Make your code as compact as possible. That is, use the fewest number of lines of code in your FISC assembler file. (Comment and symbol only lines do not count) Document your assembler program with comments. Save your file as fibo2.s

Fibo2.s:

```
Start: not r0 r1
        and r0 r0 r1
        not r1 r0
        add r1 r1 r1
        not r1 r1    ; needed to make reg1 1
        add r2 r1 r1
        add r2 r2 r2 ; contunuously multiplying by 2
        add r2 r2 r2
        add r2 r2 r2
        and r0 r2 r2 ; storing 16
        add r2 r2 r2
        add r2 r2 r2
        add r2 r0 r2
        add r2 r2 r0
        add r2 r2 r0
        and r0 r3 r3
        add r3 r1 r3 ; starting with 1
loop: and r1 r3 r3
        add r3 r0 r3
        and r0 r1 r1
        add r1 r2 r1 ; checking if r1 is last number
        bnz loop
end: not r2 r2 ; continuous loop
        bnz end
```

Assembler Output:

```
C:\Users\daqua\Fiscas\Fiscas\main>java Fiscas.java "C:\temp\fibo2.s" "C:\temp\output.hex" -l
***LABEL LIST***
Start 00
loop 16
end 21
***MACHINE CODE***
00:90 not r0 r1
01:44 and r0 r0 r1
02:81 not r1 r0
03:15 add r1 r1 r1
04:91 not r1 r1
05:16 add r2 r1 r1
06:2a add r2 r2 r2
07:2a add r2 r2 r2
08:2a add r2 r2 r2
09:68 and r0 r2 r2
10:2a add r2 r2 r2
11:2a add r2 r2 r2
12:a add r2 r0 r2
13:22 add r2 r2 r0
14:22 add r2 r2 r0
15:7c and r0 r3 r3
16:7d and r1 r3 r3
17:f add r3 r0 r3
18:54 and r0 r1 r1
19:25 add r1 r2 r1
20:d0 bnz loop
21:A2 not r2 r2
22:d5 bnz end
08:2a add r2 r2 r2
21:d1 bnz loop
22:A2 not r2 r2
23:d6 bnz end
```

1) *Outline your thought process for minimizing the code, what did you try first, and what problems did you encounter?*

When first attempting to make this code I tried simply using the regular Fibonacci code that I used in the first fibo1.s document. After finding out that it wouldn't be the way I wanted to

I began to understand that I would need to branch at least twice in order to get the correct output. Therefore I decided to have one loop that actually did the Fibonacci requirements and then I added a loop for the end. I assumed that the last number to be reached would be 144.

2) *Give at least one limitation of the FISC Instruction set that made this problem somewhat challenging.*

One limitation of the FISC instruction set is that you cannot simply store the number that you wanted and work on it. This made the program extremely hard to do because we only wanted the program to run to the 12th number in the Fibonacci sequence. You cannot simply just store the data and work on it directly. You have to and it with something else which easily leads to losing a value

3) *Think of one instruction that might make this problem easier to solve. Explain how this instruction would help*

One instruction that would extremely help with this would be a store instruction. This would help when it comes to making sure we do not lose a value

4) *How many arguments does your instruction take?*

This instruction would only need to have a read register and an address or another register. Therefore it would only need two arguments.

5) *Give an example of a line assembly using your new opcode and describe what the line is doing*

The instruction would look as follows:

```
str   10;
```

This instruction would simply hold the value that it gets from going to line 10 and grabbing that operation's final value.

6) *What is the opcode of your new instruction*

The opcode of this new instruction is going to be three bits instead of two and the address will take up the remaining 5 bits. The opcode for this new instruction will be 100.

7) *Give the machine code for your line of code in (2). Note this must be correct, your instruction must correctly fit into the opcode table using, as yet unused, extra bits. Think carefully about the FISC design.*

The machine code that would be displayed would look something like the following:

```
01: 8A  str 10
```

The above machine code gives the line at which the str instruction was used. The remaining 5 bits will be used to store the address that is going to be used to make the jump to the next line of code and store that value. Using this instruction the binary string for this whole argument would

be something like “10001010”. This would be changed to hex of course which would give us the complete hex representation as “8A”.

Conclusion:

In this lab I used my previous knowledge of assembly language and operations as well as new found knowledge of computer architecture and organization. Using my past knowledge of assembly language I was able to build an assembler with four instructions in order to display the given output that is needed for my simulator that I made. While making these programs I was able to better comprehend how a simple assembler may work as well as contemplate the storage of values inside of that assembler or simulator. While building the simulator in Logisim I was able to visually see the amount of wires and logical units used in order to simulate simple fibonacci numbers in hex. I also was able to visually see components of my simulator put to use, such as how the Z flag checks whether or not the past operation gave us a zero by checking the register that holds the value for our last operation.

My knowledge of circuits and architecture helped me with many things during this assignment such as the placement of bits in order for operation and the placement of the opcode to see what operation needed to be performed. Secondly, the logical knowledge on how each of the assembly instructions was used was also extremely helpful in order to help me visualize the instructions and the outputs that they may have given us. During this lab there were many times where knowledge learned throughout this specific class was needed in order to implement the given instructions without fail. Connecting both prior assembly knowledge and combinational logic to the assembler and simulator can help lead to minimized schematics for the simulator as well as easier to manage code for the assembler.

Fibo1.s Simulator Output:

```
C:\Users\daqua\Fiscas\Fiscsim\src\main>java Fiscsim.java "C:\temp\output.hex" 78 -d
Cycle:1 State:PC:1 Z: 0 R0:FF R1:00 R2: 00 R3:00
Disassemble:  not r0 r1
Cycle:2 State:PC:2 Z: 1 R0:00 R1:00 R2: 00 R3:00
Disassemble:  and r0 r0 r1
Cycle:3 State:PC:3 Z: 0 R0:00 R1:FF R2: 00 R3:00
Disassemble:  not r1 r0
Cycle:4 State:PC:4 Z: 0 R0:00 R1:FE R2: 00 R3:00
Disassemble:  add r1 r1 r1
Cycle:5 State:PC:5 Z: 0 R0:00 R1:01 R2: 00 R3:00
Disassemble:  not r1 r1
Cycle:6 State:PC:6 Z: 0 R0:00 R1:01 R2: 00 R3:01
Disassemble:  add r3 r0 r1
Cycle:7 State:PC:7 Z: 0 R0:FE R1:01 R2: 00 R3:01
Disassemble:  not r0 r1
Cycle:8 State:PC:8 Z: 0 R0:01 R1:01 R2: 00 R3:01
Disassemble:  not r0 r0
Cycle:9 State:PC:9 Z: 0 R0:01 R1:FE R2: 00 R3:01
Disassemble:  not r1 r3
Cycle:10 State:PC:10 Z: 0 R0:01 R1:01 R2: 00 R3:01
Disassemble:  not r1 r1
Cycle:11 State:PC:5 Z: 0 R0:01 R1:01 R2: 00 R3:01
Disassemble:  bnz 5
Cycle:12 State:PC:6 Z: 0 R0:01 R1:01 R2: 00 R3:02
Disassemble:  add r3 r0 r1
Cycle:13 State:PC:7 Z: 0 R0:FE R1:01 R2: 00 R3:02
Disassemble:  not r0 r1
Cycle:14 State:PC:8 Z: 0 R0:01 R1:01 R2: 00 R3:02
Disassemble:  not r0 r0
Cycle:15 State:PC:9 Z: 0 R0:01 R1:FD R2: 00 R3:02
Disassemble:  not r1 r3
Cycle:16 State:PC:10 Z: 0 R0:01 R1:02 R2: 00 R3:02
Disassemble:  not r1 r1
Cycle:17 State:PC:5 Z: 0 R0:01 R1:02 R2: 00 R3:02
Disassemble:  bnz 5
Cycle:18 State:PC:6 Z: 0 R0:01 R1:02 R2: 00 R3:03
Disassemble:  add r3 r0 r1
Cycle:19 State:PC:7 Z: 0 R0:FD R1:02 R2: 00 R3:03
Disassemble:  not r0 r1
Cycle:20 State:PC:8 Z: 0 R0:02 R1:02 R2: 00 R3:03
Disassemble:  not r0 r0
Cycle:21 State:PC:9 Z: 0 R0:02 R1:FC R2: 00 R3:03
Disassemble:  not r1 r3
Cycle:22 State:PC:10 Z: 0 R0:02 R1:03 R2: 00 R3:03
```

Disassemble: not r1 r1
Cycle:23 State:PC:5 Z: 0 R0:02 R1:03 R2: 00 R3:03
Disassemble: bnz 5
Cycle:24 State:PC:6 Z: 0 R0:02 R1:03 R2: 00 R3:05
Disassemble: add r3 r0 r1
Cycle:25 State:PC:7 Z: 0 R0:FC R1:03 R2: 00 R3:05
Disassemble: not r0 r1
Cycle:26 State:PC:8 Z: 0 R0:03 R1:03 R2: 00 R3:05
Disassemble: not r0 r0
Cycle:27 State:PC:9 Z: 0 R0:03 R1:FA R2: 00 R3:05
Disassemble: not r1 r3
Cycle:28 State:PC:10 Z: 0 R0:03 R1:05 R2: 00 R3:05
Disassemble: not r1 r1
Cycle:29 State:PC:5 Z: 0 R0:03 R1:05 R2: 00 R3:05
Disassemble: bnz 5
Cycle:30 State:PC:6 Z: 0 R0:03 R1:05 R2: 00 R3:08
Disassemble: add r3 r0 r1
Cycle:31 State:PC:7 Z: 0 R0:FA R1:05 R2: 00 R3:08
Disassemble: not r0 r1
Cycle:32 State:PC:8 Z: 0 R0:05 R1:05 R2: 00 R3:08
Disassemble: not r0 r0
Cycle:33 State:PC:9 Z: 0 R0:05 R1:F7 R2: 00 R3:08
Disassemble: not r1 r3
Cycle:34 State:PC:10 Z: 0 R0:05 R1:08 R2: 00 R3:08
Disassemble: not r1 r1
Cycle:35 State:PC:5 Z: 0 R0:05 R1:08 R2: 00 R3:08
Disassemble: bnz 5
Cycle:36 State:PC:6 Z: 0 R0:05 R1:08 R2: 00 R3:0D
Disassemble: add r3 r0 r1
Cycle:37 State:PC:7 Z: 0 R0:F7 R1:08 R2: 00 R3:0D
Disassemble: not r0 r1
Cycle:38 State:PC:8 Z: 0 R0:08 R1:08 R2: 00 R3:0D
Disassemble: not r0 r0
Cycle:39 State:PC:9 Z: 0 R0:08 R1:F2 R2: 00 R3:0D
Disassemble: not r1 r3
Cycle:40 State:PC:10 Z: 0 R0:08 R1:0D R2: 00 R3:0D
Disassemble: not r1 r1
Cycle:41 State:PC:5 Z: 0 R0:08 R1:0D R2: 00 R3:0D
Disassemble: bnz 5
Cycle:42 State:PC:6 Z: 0 R0:08 R1:0D R2: 00 R3:15
Disassemble: add r3 r0 r1
Cycle:43 State:PC:7 Z: 0 R0:F2 R1:0D R2: 00 R3:15
Disassemble: not r0 r1
Cycle:44 State:PC:8 Z: 0 R0:0D R1:0D R2: 00 R3:15
Disassemble: not r0 r0
Cycle:45 State:PC:9 Z: 0 R0:0D R1:EA R2: 00 R3:15

Disassemble: not r1 r3
Cycle:46 State:PC:10 Z: 0 R0:0D R1:15 R2: 00 R3:15
Disassemble: not r1 r1
Cycle:47 State:PC:5 Z: 0 R0:0D R1:15 R2: 00 R3:15
Disassemble: bnz 5
Cycle:48 State:PC:6 Z: 0 R0:0D R1:15 R2: 00 R3:22
Disassemble: add r3 r0 r1
Cycle:49 State:PC:7 Z: 0 R0:EA R1:15 R2: 00 R3:22
Disassemble: not r0 r1
Cycle:50 State:PC:8 Z: 0 R0:15 R1:15 R2: 00 R3:22
Disassemble: not r0 r0
Cycle:51 State:PC:9 Z: 0 R0:15 R1:DD R2: 00 R3:22
Disassemble: not r1 r3
Cycle:52 State:PC:10 Z: 0 R0:15 R1:22 R2: 00 R3:22
Disassemble: not r1 r1
Cycle:53 State:PC:5 Z: 0 R0:15 R1:22 R2: 00 R3:22
Disassemble: bnz 5
Cycle:54 State:PC:6 Z: 0 R0:15 R1:22 R2: 00 R3:37
Disassemble: add r3 r0 r1
Cycle:55 State:PC:7 Z: 0 R0:DD R1:22 R2: 00 R3:37
Disassemble: not r0 r1
Cycle:56 State:PC:8 Z: 0 R0:22 R1:22 R2: 00 R3:37
Disassemble: not r0 r0
Cycle:57 State:PC:9 Z: 0 R0:22 R1:C8 R2: 00 R3:37
Disassemble: not r1 r3
Cycle:58 State:PC:10 Z: 0 R0:22 R1:37 R2: 00 R3:37
Disassemble: not r1 r1
Cycle:59 State:PC:5 Z: 0 R0:22 R1:37 R2: 00 R3:37
Disassemble: bnz 5
Cycle:60 State:PC:6 Z: 0 R0:22 R1:37 R2: 00 R3:59
Disassemble: add r3 r0 r1
Cycle:61 State:PC:7 Z: 0 R0:C8 R1:37 R2: 00 R3:59
Disassemble: not r0 r1
Cycle:62 State:PC:8 Z: 0 R0:37 R1:37 R2: 00 R3:59
Disassemble: not r0 r0
Cycle:63 State:PC:9 Z: 0 R0:37 R1:A6 R2: 00 R3:59
Disassemble: not r1 r3
Cycle:64 State:PC:10 Z: 0 R0:37 R1:59 R2: 00 R3:59
Disassemble: not r1 r1
Cycle:65 State:PC:5 Z: 0 R0:37 R1:59 R2: 00 R3:59
Disassemble: bnz 5
Cycle:66 State:PC:6 Z: 0 R0:37 R1:59 R2: 00 R3:90
Disassemble: add r3 r0 r1
Cycle:67 State:PC:7 Z: 0 R0:A6 R1:59 R2: 00 R3:90
Disassemble: not r0 r1
Cycle:68 State:PC:8 Z: 0 R0:59 R1:59 R2: 00 R3:90

```
Disassemble:  not r0 r0
Cycle:69 State:PC:9 Z: 0 R0:59 R1:6F R2: 00 R3:90
Disassemble:  not r1 r3
Cycle:70 State:PC:10 Z: 0 R0:59 R1:90 R2: 00 R3:90
Disassemble:  not r1 r1
Cycle:71 State:PC:5 Z: 0 R0:59 R1:90 R2: 00 R3:90
Disassemble:  bnz 5
Cycle:72 State:PC:6 Z: 0 R0:59 R1:90 R2: 00 R3:E9
Disassemble:  add r3 r0 r1
Cycle:73 State:PC:7 Z: 0 R0:6F R1:90 R2: 00 R3:E9
Disassemble:  not r0 r1
Cycle:74 State:PC:8 Z: 0 R0:90 R1:90 R2: 00 R3:E9
Disassemble:  not r0 r0
Cycle:75 State:PC:9 Z: 0 R0:90 R1:16 R2: 00 R3:E9
Disassemble:  not r1 r3
Cycle:76 State:PC:10 Z: 0 R0:90 R1:E9 R2: 00 R3:E9
Disassemble:  not r1 r1
Cycle:77 State:PC:5 Z: 0 R0:90 R1:E9 R2: 00 R3:E9
Disassemble:  bnz 5
Cycle:78 State:PC:6 Z: 0 R0:90 R1:E9 R2: 00 R3:79
Disassemble:  add r3 r0 r1
```

Fibo2.s Simulator Output:

```
C:\Users\daqua\Fiscas\Fiscsim\src\main>java Fiscsim.java "C:\temp\output.hex" 200 -d
Cycle:1 State:PC:1 Z: 0 R0:FF R1:00 R2: 00 R3:00
Disassemble:  not r0 r1
Cycle:2 State:PC:2 Z: 1 R0:00 R1:00 R2: 00 R3:00
Disassemble:  and r0 r0 r1
Cycle:3 State:PC:3 Z: 0 R0:00 R1:FF R2: 00 R3:00
Disassemble:  not r1 r0
Cycle:4 State:PC:4 Z: 0 R0:00 R1:FE R2: 00 R3:00
Disassemble:  add r1 r1 r1
Cycle:5 State:PC:5 Z: 0 R0:00 R1:01 R2: 00 R3:00
Disassemble:  not r1 r1
Cycle:6 State:PC:6 Z: 0 R0:00 R1:01 R2: 02 R3:00
Disassemble:  add r2 r1 r1
Cycle:7 State:PC:7 Z: 0 R0:00 R1:01 R2: 04 R3:00
Disassemble:  add r2 r2 r2
Cycle:8 State:PC:8 Z: 0 R0:00 R1:01 R2: 08 R3:00
Disassemble:  add r2 r2 r2
```

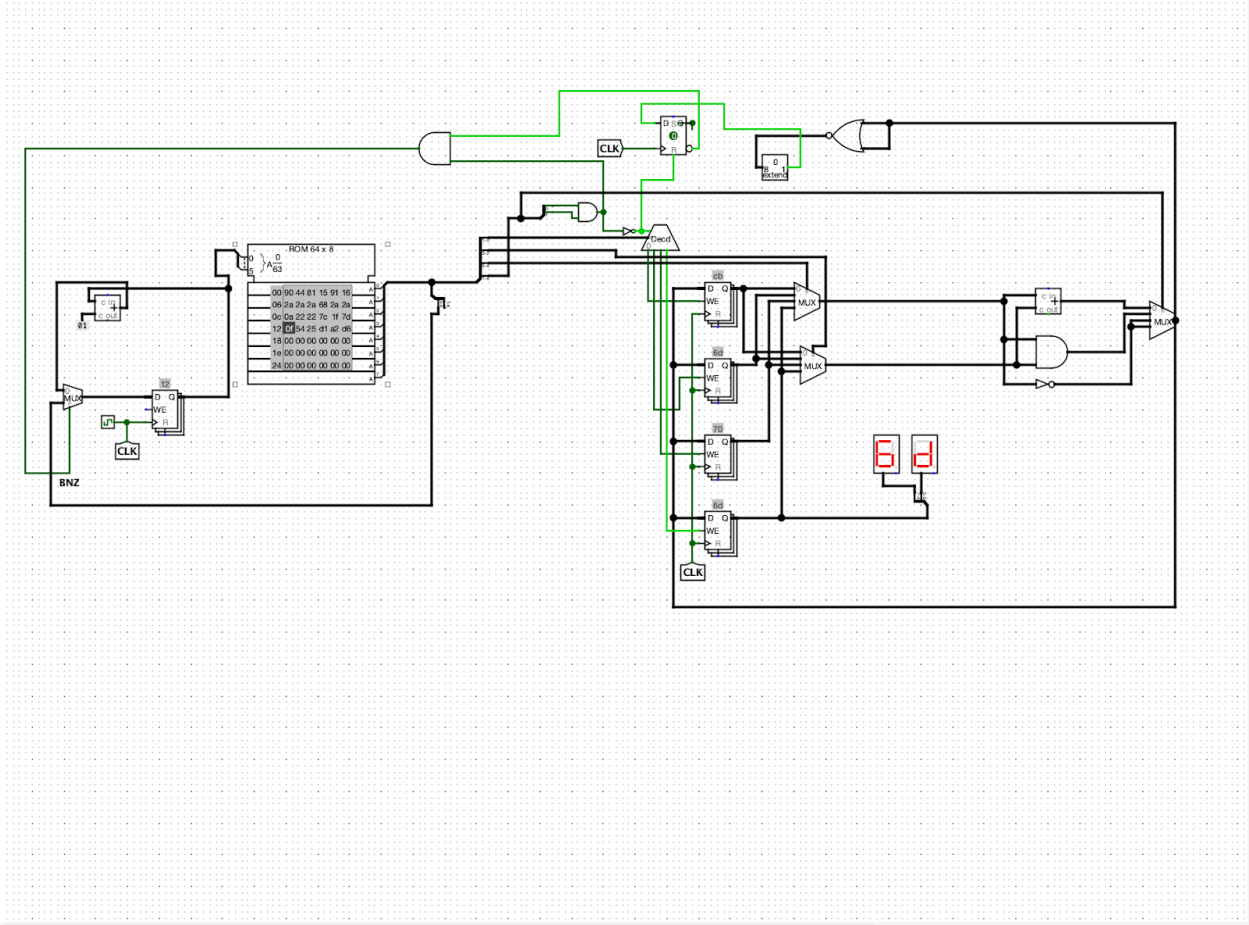
Cycle:9 State:PC:9 Z: 0 R0:00 R1:01 R2: 10 R3:00
Disassemble: add r2 r2 r2
Cycle:10 State:PC:10 Z: 0 R0:10 R1:01 R2: 10 R3:00
Disassemble: and r0 r2 r2
Cycle:11 State:PC:11 Z: 0 R0:10 R1:01 R2: 20 R3:00
Disassemble: add r2 r2 r2
Cycle:12 State:PC:12 Z: 0 R0:10 R1:01 R2: 40 R3:00
Disassemble: add r2 r2 r2
Cycle:13 State:PC:13 Z: 0 R0:10 R1:01 R2: 50 R3:00
Disassemble: add r2 r0 r2
Cycle:14 State:PC:14 Z: 0 R0:10 R1:01 R2: 60 R3:00
Disassemble: add r2 r2 r0
Cycle:15 State:PC:15 Z: 0 R0:10 R1:01 R2: 70 R3:00
Disassemble: add r2 r2 r0
Cycle:16 State:PC:16 Z: 1 R0:00 R1:01 R2: 70 R3:00
Disassemble: and r0 r3 r3
Cycle:17 State:PC:17 Z: 0 R0:00 R1:01 R2: 70 R3:01
Disassemble: add r3 r1 r3
Cycle:18 State:PC:18 Z: 0 R0:00 R1:01 R2: 70 R3:01
Disassemble: and r1 r3 r3
Cycle:19 State:PC:19 Z: 0 R0:00 R1:01 R2: 70 R3:01
Disassemble: add r3 r0 r3
Cycle:20 State:PC:20 Z: 0 R0:01 R1:01 R2: 70 R3:01
Disassemble: and r0 r1 r1
Cycle:21 State:PC:21 Z: 0 R0:01 R1:71 R2: 70 R3:01
Disassemble: add r1 r2 r1
Cycle:22 State:PC:17 Z: 0 R0:01 R1:71 R2: 70 R3:01
Disassemble: bnz r17
Cycle:23 State:PC:18 Z: 0 R0:01 R1:01 R2: 70 R3:01
Disassemble: and r1 r3 r3
Cycle:24 State:PC:19 Z: 0 R0:01 R1:01 R2: 70 R3:02
Disassemble: add r3 r0 r3
Cycle:25 State:PC:20 Z: 0 R0:01 R1:01 R2: 70 R3:02
Disassemble: and r0 r1 r1
Cycle:26 State:PC:21 Z: 0 R0:01 R1:71 R2: 70 R3:02
Disassemble: add r1 r2 r1
Cycle:27 State:PC:17 Z: 0 R0:01 R1:71 R2: 70 R3:02
Disassemble: bnz r17
Cycle:28 State:PC:18 Z: 0 R0:01 R1:02 R2: 70 R3:02
Disassemble: and r1 r3 r3
Cycle:29 State:PC:19 Z: 0 R0:01 R1:02 R2: 70 R3:03
Disassemble: add r3 r0 r3
Cycle:30 State:PC:20 Z: 0 R0:02 R1:02 R2: 70 R3:03
Disassemble: and r0 r1 r1
Cycle:31 State:PC:21 Z: 0 R0:02 R1:72 R2: 70 R3:03
Disassemble: add r1 r2 r1

Cycle:32 State:PC:17 Z: 0 R0:02 R1:72 R2: 70 R3:03
Disassemble: bnz r17
Cycle:33 State:PC:18 Z: 0 R0:02 R1:03 R2: 70 R3:03
Disassemble: and r1 r3 r3
Cycle:34 State:PC:19 Z: 0 R0:02 R1:03 R2: 70 R3:05
Disassemble: add r3 r0 r3
Cycle:35 State:PC:20 Z: 0 R0:03 R1:03 R2: 70 R3:05
Disassemble: and r0 r1 r1
Cycle:36 State:PC:21 Z: 0 R0:03 R1:73 R2: 70 R3:05
Disassemble: add r1 r2 r1
Cycle:37 State:PC:17 Z: 0 R0:03 R1:73 R2: 70 R3:05
Disassemble: bnz r17
Cycle:38 State:PC:18 Z: 0 R0:03 R1:05 R2: 70 R3:05
Disassemble: and r1 r3 r3
Cycle:39 State:PC:19 Z: 0 R0:03 R1:05 R2: 70 R3:08
Disassemble: add r3 r0 r3
Cycle:40 State:PC:20 Z: 0 R0:05 R1:05 R2: 70 R3:08
Disassemble: and r0 r1 r1
Cycle:41 State:PC:21 Z: 0 R0:05 R1:75 R2: 70 R3:08
Disassemble: add r1 r2 r1
Cycle:42 State:PC:17 Z: 0 R0:05 R1:75 R2: 70 R3:08
Disassemble: bnz r17
Cycle:43 State:PC:18 Z: 0 R0:05 R1:08 R2: 70 R3:08
Disassemble: and r1 r3 r3
Cycle:44 State:PC:19 Z: 0 R0:05 R1:08 R2: 70 R3:0D
Disassemble: add r3 r0 r3
Cycle:45 State:PC:20 Z: 0 R0:08 R1:08 R2: 70 R3:0D
Disassemble: and r0 r1 r1
Cycle:46 State:PC:21 Z: 0 R0:08 R1:78 R2: 70 R3:0D
Disassemble: add r1 r2 r1
Cycle:47 State:PC:17 Z: 0 R0:08 R1:78 R2: 70 R3:0D
Disassemble: bnz r17
Cycle:48 State:PC:18 Z: 0 R0:08 R1:0D R2: 70 R3:0D
Disassemble: and r1 r3 r3
Cycle:49 State:PC:19 Z: 0 R0:08 R1:0D R2: 70 R3:15
Disassemble: add r3 r0 r3
Cycle:50 State:PC:20 Z: 0 R0:0D R1:0D R2: 70 R3:15
Disassemble: and r0 r1 r1
Cycle:51 State:PC:21 Z: 0 R0:0D R1:7D R2: 70 R3:15
Disassemble: add r1 r2 r1
Cycle:52 State:PC:17 Z: 0 R0:0D R1:7D R2: 70 R3:15
Disassemble: bnz r17
Cycle:53 State:PC:18 Z: 0 R0:0D R1:15 R2: 70 R3:15
Disassemble: and r1 r3 r3
Cycle:54 State:PC:19 Z: 0 R0:0D R1:15 R2: 70 R3:22
Disassemble: add r3 r0 r3

Cycle:55 State:PC:20 Z: 0 R0:15 R1:15 R2: 70 R3:22
Disassemble: and r0 r1 r1
Cycle:56 State:PC:21 Z: 0 R0:15 R1:85 R2: 70 R3:22
Disassemble: add r1 r2 r1
Cycle:57 State:PC:17 Z: 0 R0:15 R1:85 R2: 70 R3:22
Disassemble: bnz r17
Cycle:58 State:PC:18 Z: 0 R0:15 R1:22 R2: 70 R3:22
Disassemble: and r1 r3 r3
Cycle:59 State:PC:19 Z: 0 R0:15 R1:22 R2: 70 R3:37
Disassemble: add r3 r0 r3
Cycle:60 State:PC:20 Z: 0 R0:22 R1:22 R2: 70 R3:37
Disassemble: and r0 r1 r1
Cycle:61 State:PC:21 Z: 0 R0:22 R1:92 R2: 70 R3:37
Disassemble: add r1 r2 r1
Cycle:62 State:PC:17 Z: 0 R0:22 R1:92 R2: 70 R3:37
Disassemble: bnz r17
Cycle:63 State:PC:18 Z: 0 R0:22 R1:37 R2: 70 R3:37
Disassemble: and r1 r3 r3
Cycle:64 State:PC:19 Z: 0 R0:22 R1:37 R2: 70 R3:59
Disassemble: add r3 r0 r3
Cycle:65 State:PC:20 Z: 0 R0:37 R1:37 R2: 70 R3:59
Disassemble: and r0 r1 r1
Cycle:66 State:PC:21 Z: 0 R0:37 R1:A7 R2: 70 R3:59
Disassemble: add r1 r2 r1
Cycle:67 State:PC:17 Z: 0 R0:37 R1:A7 R2: 70 R3:59
Disassemble: bnz r17
Cycle:68 State:PC:18 Z: 0 R0:37 R1:59 R2: 70 R3:59
Disassemble: and r1 r3 r3
Cycle:69 State:PC:19 Z: 0 R0:37 R1:59 R2: 70 R3:90
Disassemble: add r3 r0 r3
Cycle:70 State:PC:20 Z: 0 R0:59 R1:59 R2: 70 R3:90
Disassemble: and r0 r1 r1
Cycle:71 State:PC:21 Z: 0 R0:59 R1:C9 R2: 70 R3:90
Disassemble: add r1 r2 r1
Cycle:72 State:PC:17 Z: 0 R0:59 R1:C9 R2: 70 R3:90
Disassemble: bnz r17
Cycle:73 State:PC:18 Z: 0 R0:59 R1:90 R2: 70 R3:90
Disassemble: and r1 r3 r3
Cycle:74 State:PC:19 Z: 0 R0:59 R1:90 R2: 70 R3:E9
Disassemble: add r3 r0 r3
Cycle:75 State:PC:20 Z: 0 R0:90 R1:90 R2: 70 R3:E9
Disassemble: and r0 r1 r1
Cycle:76 State:PC:21 Z: 1 R0:90 R1:00 R2: 70 R3:E9
Disassemble: add r1 r2 r1
Cycle:77 State:PC:22 Z: 1 R0:90 R1:00 R2: 70 R3:E9
Disassemble: bnz r17

Cycle:78 State:PC:23 Z: 0 R0:90 R1:00 R2: 8F R3:E9

Simulator FIBO2 :



Fiscas.java Code:

```
import java.util.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
```

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

 class Fiscas {
	public static void main(String[] args) throws IOException {
			//If not file is provided then we will have to print error
	 String inputFilePath=args[0];
	 String ouputFilePath= args[1];
	 //Created two different files using the one input file given
	 FileInputStream inputFileStream=new FileInputStream(inputFilePath);
	 FileInputStream inputFileStream2=new FileInputStream(inputFilePath);
	 //Scanners to look at files
	 Scanner input=new Scanner(inputFileStream);
	 Scanner input2=new Scanner(inputFileStream2);
	 //Grabbing labels from label list inputing
	 ArrayList<String> labels=labelFinder(input);
	 ArrayList<String> hex =fileReader(input2,labels,ouputFilePath);
	if(args.length==3) {
	 String list=args[2];
	 if(list.contains("-l")) {
	 //Exit the method
	  System.out.println("***LABEL LIST***");
		for(int j=0;j<labels.size();j++) {
			System.out.println(labels.get(j));
		}
		System.out.println("***MACHINE CODE***");
		 for(int j=0;j<hex.size();j++) {
			System.out.println(hex.get(j));
		 }
	   }
	}

	input.close();
	input2.close();
}

public static ArrayList<String> labelFinder(Scanner input) {
	//String to hold current value
String a=" ";
//Labels array
ArrayList<String> labels=new ArrayList<String>();
//Line counter/Program counter
int programCounter=-1;
while (input.hasNextLine()) {
  programCounter++;
```

```
  while (input.hasNext()) {
          a=(String)input.next();
           if(a.contains(":")) {
             a=a.substring(0,a.length()-1);
             labels.add(a+" "+String.format("%02d",programCounter));
          }else if (a.contains(";")) {
          }else {
            switch (a.toLowerCase()) {
                  case "add":
                        programCounter++;
                        break;
                  case "bnz":
                        programCounter++;
                        break;
                  case "not":
                        programCounter++;
                        break;
                  case "and":
                        programCounter++;
                        break;
                  default :
                        break;
                  }
          }
}
          return labels;
          }
          return labels;
          }

public static ArrayList<String> fileReader(Scanner input,
ArrayList<String> list,String outputFilePath) throws IOException {
//String to hold current value
 String a = " ";
 //Array of hex value and Line counter
 ArrayList<String> s= new ArrayList<String>();
 //Array that holds each line
 ArrayList<String> instruction=new ArrayList<String>();
 ArrayList<String> machineCode= new ArrayList<String>();
 ArrayList<String> hexadecimals= new ArrayList<String>();
 //Program counter
 int programCounter=-1;
 //This is for changing string to int
  int hexadecimal=0;
 //Looping through the input file and gathering lines
          while (input.hasNextLine()){
```

```
programCounter++;
String inst="";
String hold="";
while(input.hasNext()) {
  String hex="";
  a=(String)input.next();
  switch (a.toLowerCase()) {
        case "add":
          hex=hex+"00";
          inst="add";
                for(int i=0;i<3;i++) {
                        String f=(String)input.next();
                        if(i==0) {
                                hold=bits(f);
                                inst=inst+" "+f;
                        }else {
                                hex=hex+bits(f);
                                inst=inst+" "+f;
                        }
                }
                hex=hex+hold;
                instruction.add(inst);
                hexadecimal=Integer.valueOf(hex,2);
                s.add(String.format("%02d",programCounter)+
                        ":"+Integer.toHexString(hexadecimal));
                hexadecimals.add(Integer.toHexString(hexadecimal));
                programCounter++;
                break;
        case "bnz":
                hex=hex+"11";
                inst="bnz";
                String counter=(String)input.next();
                inst=inst+" "+counter;
                instruction.add(inst);
                for (int i=0;i<list.size();i++) {
                        if(list.get(i).contains(counter)){
                                String position=list.get(i).substring(list.get(i).length()-2);
                                String bin=
Integer.toBinaryString(Integer.parseInt(position));

                                bin=String.format("%6S", bin).replace(' ','0');

                                hex=hex+bin;

                        }
                }
```

```
		hexadecimal=Integer.valueOf(hex,2);
		s.add(String.format("%02d",programCounter)+
			":"+Integer.toHexString(hexadecimal));
		hexadecimals.add(Integer.toHexString(hexadecimal));
		programCounter++;
		break;
	case "not":
		hex=hex+"10";
		inst="not";
			String l=(String)input.next();
			hold=bits(l);
			inst=inst+" "+l;
			l=(String)input.next();
			hex=hex+bits(l);
			inst=inst+" "+l;
			hex+="00";
		hex= hex+hold;
		instruction.add(inst);
		hexadecimal=Integer.valueOf(hex,2);
		s.add(String.format("%02d",programCounter)
				+":"+String.format("%02X",hexadecimal));
		hexadecimals.add(Integer.toHexString(hexadecimal));
		programCounter++;
		break;
	case "and":
		hex=hex+"01";
		inst="and";
		for(int i=0;i<3;i++) {
			String x=(String)input.next();
				if(i==0) {
					hold=bits(x);
					inst=inst+" "+x;
				}else {
					hex=hex+bits(x);
					inst=inst+" "+x;
				}
			}
		hex=hex+hold;
		hexadecimal=Integer.valueOf(hex,2);
		s.add(String.format("%02d",programCounter)+
				":"+Integer.toHexString(hexadecimal));
		instruction.add(inst);
		hexadecimals.add(Integer.toHexString(hexadecimal));
		programCounter++;
		break;
	default :
```

```
                        break;
                    }
            }
break;
}
for(int j=0;j<s.size();j++) {

            machineCode.add(s.get(j)+" "+instruction.get(j));
            }
            ouputFile(hexadecimals,outputFilePath);
            return machineCode;

            }

public static String bits(String f) {
            String r0= "00";
String r1= "01";
String r2= "10";
String r3= "11";
if(f.toLowerCase().contains("r0")) {
            f=r0;
            return f;
}else if(f.toLowerCase().contains("r1")) {
            f=r1;
            return f;
}else if(f.toLowerCase().contains("r2")) {
            f=r2;
            return f;
}else if(f.toLowerCase().contains("r3")) {
            f=r3;
            return f;
}else {
            System.out.println("Error not correct input!!");
            }
            return null;
}
public static void ouputFile(ArrayList<String> s,String o) throws IOException{
            Path path=Paths.get(o);
            if(Files.exists(path)) {
                        FileWriter writer=new FileWriter(o);
                        writer.write("v2.0 raw"+System.lineSeparator());
            for(String str:s) {
                        writer.write(str+System.lineSeparator());
            }
            writer.close();
}else {
```

```
        File ouputFile=new File(o);
        ouputFile.createNewFile();
        FileWriter writer=new FileWriter(o);
        writer.write("v2.0 raw"+System.lineSeparator());
                        for(String str:s) {
                                writer.write(str+System.lineSeparator());
                        }
                        writer.close();

                }

        }
}
```

Fiscsim.java Code:

```
import java.util.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.math.BigInteger;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

class Fiscsim{

public static void main(String[] args) throws FileNotFoundException {
        String inputFilePath=args[0];
        FileInputStream inputFileStream=new FileInputStream(inputFilePath);
        Scanner input=new Scanner(inputFileStream);
        //ArrayList<String> hexToInt= new ArrayList<String>();
        int[] registers= new int[4];
        String display=" ";
        registers[0]=0;
        registers[1]=0;
        registers[2]=0;
        registers[3]=0;
        int loops=20;

        if(args.length==1) {
                hexToInt(input,loops,registers,display);
```

```
	}

	if(args.length==2 ) {
		if(args[1].equals("-d")){
			display=args[1];
			hexToInt(input,loops,registers,display);
		}

		if(isInteger(args[1])){
			//changes the second one into loops
			loops=Integer.parseInt(args[1]);
			hexToInt(input,loops,registers,display);

		}

	}

	if (args.length==3 && isInteger(args[1]) && args[2].equals("-d")){
		display=args[2];
		loops=Integer.parseInt(args[1]);
		hexToInt(input,loops,registers,display);

	}


	input.close();
}


public static void hexToInt(Scanner input,int loops, int[] registers,String display){
	//Integers that were converted to bits
	int z=0;
	int runs=0;
	int branch=0;

	input.nextLine();
	ArrayList<String> instructions= new ArrayList<String>();
	ArrayList<String> wrdInst= new ArrayList<String>();


	while(input.hasNextLine()){
		instructions.add(input.nextLine());
	}
```

```
	for (int j=0;runs<loops;j++) {
		String test=" ";
		runs++;
		int value=0;
		String inst= instructions.get(j);
		int hex=Integer.parseInt(inst,16);

		//Storing the op and rn and rm correct positions
		int op= hex>>6;
		int rn = hex & 0b00111111;
		branch=rn;
		rn = rn>>4;
		int rm=hex & 0b00001111;
		rm=rm>>2;
		int rd= hex & 0b00000011;

//case statement for each op
	if(op==0) {
		String numString=" ";
		test+="add";
		value=(registers[rn]+registers[rm]);
		registers[rd]=value;
		test+=" "+rd+" "+rn+" "+rm;
		numString=String.format("%02x", value & 0xFF);
		if (numString.equals("0") || numString.equals("00")) {
			z=1;
		}else {
			z=0;
		}

	}
	if (op==1) {
		test+="and";
		value= (registers[rm] & registers[rn]);
		registers[rd]= value;

		if (value!=0 || value!=00) {
			z=0;

		}else {
			z=1;
		}

		test+=" "+rd+" "+rn+" "+rm;
	}
```

```
		if (op==2) {
			test+="not";
			value= ~(registers[rn]);
			registers[rd]= value;

			if (value !=0) {
				z=0;

			}else {
				z=1;
			}
			test+=" "+rd+" "+rn;
		}
		//branch when not zero
		if (op==3) {

			test+="bnz";
			if(z==0) {
				 j=branch-1;
			}
			test+=" "+branch;

		}
		//adding full instruction to
		wrdInst.add(test);
		//Outputting
		System.out.println("Cycle:"+runs+" State:PC:"+(j+1)+" Z: "+z+" R0:"+
					String.format("%02X", registers[0] & 0xFF)+ " R1:"
					+String.format("%02X", registers[1]& 0xFF)+ " R2: "+
					String.format("%02X", registers[2] & 0xFF)+" R3:"
					+String.format("%02X", registers[3]& 0xFF));

		if(display.contains("-d")) {
			disassemble(test);
		}



		}


}

private static void disassemble(String inst) {
```

```
        inst=inst.replace("0","r0");
        inst=inst.replace("1","r1");
        inst=inst.replace("2","r2");
        inst=inst.replace("3","r3");

        System.out.println("Disassemble: "+inst);


}


//Checks if argument is integer

                public static boolean isInteger(String arg) {
                        try {
                                Integer.parseInt(arg);
                                return true;
                        }catch(NumberFormatException e) {
                                return false;
                        }

                }



}
```

Simulator Schematic: