Subneg Summary

Assembly language, although tedious, has a way of really working the mind. For my implementation, I would like to give clarification and insight that may be helpful in understanding it all. First off, I have two main output locations in memory depending on whether or not the program is a Fibonacci implementation. This can be seen on **page 3** under Subneg line 31 & 32. In addition, some implementations consider the output futile and I will point out when that's the case. For the appendix, I put side by side the original memory logic and the command prompt runtime output. I also included the desired result below the memory dump for when a result is necessary. As a side note, I combined the functions CMP and JGT as well as ST and LD because I feel they are essentially the same, given our implementation conditions. To be consistent, I will include page numbers for each functionality reference at the end of their respective paragraphs.

I understand that the ADD functionality is one of the core essentials. I used ADD as well as it's general logic to implement most everything else. For ADD, I subtracted 2 from 0, and then subtracted -2 from 5 to get a result of 7, which I happened to place in the last memory location, memory[11]. After really getting the hang of how Subneg works and playing around with ADD alone, other implementations came more quickly (**page 4**).

For CMP/JGT, I wanted to jump if 8 was greater than 4. I did this by comparing the memory locations [7] and [6] respectively. I subtracted 8 from 4 to see if it was negative. Because it is, I jumped to memory[9] and then exited the program. The output result is futile (**page 4**).

For JMP, the unconditional jump, I subtracted 1 from 0 to get -1 and then jumped to memory[6] before exiting. The output result is futile (**page 4**).

For MVX, I moved 1 from memory[9] to memory[14] by converting 1 to a -1 and then subtracting that from 0. The result can be seen in memory[14] (**page 4**).

For ST/LD, I wasn't exactly sure how to implement this because the assignment only wants a memory array. For this reason, I garnered a value of 8 and then placed it in the last location of the memory, memory[17]. I did both by storing 8 into that location and loading 8 into that location. This is why I combined these two functionalities (**page 5**).

Now for the implementation of Example 8.2. I looked the example over and determined the algorithm summed up all the values in an array of size 8. I decided to simulate a for loop by having a counter, incrementing the counter by one, and adding up eight numbers. To simplify the process, each "array element" will be in adjacent memory locations. I then summed the eight integers starting from the number 1: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36. Notice lines 3 and 4 of Sum8. I increment the pointers of

element indexes by 1 every time I loop. This way, I will sum every pair of two numbers from left to right. With this, I successfully get a result of 36 (**page 5**).

Fibonacci is where most of my struggles occurred. I spent quite a bit of time designing the algorithm, but my largest headache came from the most simple of mistakes. Without double checking, I believed fib(10) equaled 34 when it actually equals 55. To explain my algorithm, I store my fibonacci numbers on the very first line. Memory[0] is my temporary number holder, memory[1] is the even fibonacci index, and memory[2] is the odd fibonacci index, fib[**2**] vs fib[**3**] for example. I used this even/odd system to my advantage. For the requirements, I need to find fib(10), fib(30), and the maximum fibonacci number possible using Java integers. I looked up the actual maximum number and that happened to reside on fib(46). The even numbers 10, 30, and 46 played to my advantage. I only need to output memory[1] and I can increment my counter by 2 before checking the counter to obtain results more hastily. In short, my algorithm adds memory[1] and memory[2] together and places the result in one of those locations every addition. For example, I add them and place them in memory[1]. The next time I add them, I place them in memory[2]. This goes back and forth until the counter has reached the sufficient number to where I exit the loop on a negative comparison. For fibMax, I strayed away from a counter. My first prototype hard-coded an integer value right below fib(46) so I could compare and quit. However, I found a way to be more flexible. I used the even/odd idea to subtract memory[2], the odd fibonacci index, from 0. Since fib(46) is the largest fibonacci integer in Java, fib(47) would be to large and wrap around to the negative numbers. From there, it was simple. Test memory[2] to see if it's negative. The fibonacci results for each test can be found in memory[1] (**page 6-7**).

Overall, this project was quite challenging but turned out to be fun and engaging. I hope I explained myself well enough and didn't cause confusion. Surprisingly, I managed to implement functionality in one way or another. I can say for sure that I have learned a lot.

Appendix

Subneg

Subneg.java

```java
import java.io.File;

public class Subneg {
    public static void main(String[] args) {
        int programCounter = 0;
        int a = 0;
        int b = 0;
        int c = 0;
        int[] memory = readFile(args[0]); //args[0]

        while (programCounter >= 0) {
            a = memory[programCounter];
            b = memory[programCounter + 1];
            c = memory[programCounter + 2];
            if (a < 0 || b < 0) {
                programCounter = -1;
            } else {
                memory[b] = memory[b] - memory[a];
                if (memory[b] >= 0) {
                    programCounter += 3;
                } else {
                    programCounter = c;
                }
            }
        }
        for (int i = 0; i < memory.length-2; i += 3) {
            System.out.println(memory[i] + " " + memory[i+1] + " " + memory[i+2]);
        }
        System.out.println();
//      System.out.println(memory[memory.length-1]); // Default print location
//      System.out.println(memory[1]); // Fibo print location
    }

    public static int[] readFile(String file) {
        try {
            File f = new File(file);
            Scanner s = new Scanner(f);
            int ctr = 0;
            while (s.hasNextInt()) {
                ctr++;
                s.nextInt();
            }
            s.close();
            int[] arr = new int[ctr];
            s = new Scanner(f);

            for (int i = 0; i < arr.length; i++) {
                arr[i] = s.nextInt();
            }
            s.close();
            return arr;
        }
        catch (Exception e) {
            return null;
        }
    }
}
```

ADD

```
10 9 3
9 11 6
10 9 -1
0 2 5
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg add.txt
10 9 3
9 11 6
10 9 -1
-4 2 7

7
```

CMP / JGT

```
7 6 9
0 1 0
4 8 0
4 3 -1
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg cmp_jgt.txt
7 6 9
-1 1 0
-4 8 0
4 3 -1

6
```

JMP

```
4 3 6
0 1 6
4 3 -1
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg jmp.txt
4 3 6
-2 1 6
4 3 -1

-1
```

MVX (Mov)

```
9 10 3
10 14 6
9 11 -1
1 0 0
0 0 0
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg mov.txt
9 10 3
10 14 6
9 11 -1
1 -1 -1
0 0 1

1
```

ST / LD

```
13 14 3
14 16 6
16 15 9
15 17 12
0 2 -1
0 5 0
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg st.txt
13 14 -10
14 16 6
16 15 9
15 17 12
0 2 -3
-8 8 8

8
```

Sum8

```
40 38 3
38 41 6
39 0 9
39 4 12
33 35 15
33 34 -1
35 34 21
35 35 24
38 38 27
39 33 30
36 37 0
2 7 0
1 0 0
-1 1 2
3 4 5
6 7 8
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg sum8.txt
47 38 3
38 48 6
39 0 9
39 4 12
33 35 15
33 34 -1
35 34 21
35 35 24
38 38 27
39 33 30
36 37 0
8 -1 -8
1 -6 -28
-1 1 3
6 10 15
21 28 36

36
```

Fibo10

```
0  0  1
2  0  6
0  1  9
0  0  12
54 50 15
49 48 18
48 50 21
48 48 24
1  0  27
0  2  30
0  0  33
49 48 36
48 50 39
48 48 42
53 50 3
52 51 -1
0  1  -9
0  1  10
-10 0 0
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg fibo10.txt
0 55 89
2 0 6
0 1 9
0 0 12
54 50 15
49 48 18
48 50 21
48 48 24
1 0 27
0 2 30
0 0 33
49 48 36
48 50 39
48 48 42
53 50 3
52 51 -1
0 1 1
-1 1 10
-10 0 0

55
```

Fibo30

```
0  0  1
2  0  6
0  1  9
0  0  12
54 50 15
49 48 18
48 50 21
48 48 24
1  0  27
0  2  30
0  0  33
49 48 36
48 50 39
48 48 42
53 50 3
52 51 -1
0  1  -29
0  1  30
-30 0 0
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg fibo30.txt
0 832040 1346269
2 0 6
0 1 9
0 0 12
54 50 15
49 48 18
48 50 21
48 48 24
1 0 27
0 2 30
0 0 33
49 48 36
48 50 39
48 48 42
53 50 3
52 51 -1
0 1 1
-1 1 30
-30 0 0

832040
```

FiboMax

```
0  0  1
2  0  6
0  1  9
0  0  12
1  0  15
0  2  18
0  0  21
2 36 24
36 37 -1
36 36 30
37 37 33
36 26 3
0 0 0
```

```
F:\Eclipse\eclipse-workspace\Subneg\bin>java Subneg fiboMax.txt
0 1836311903 -1323752223
2 0 6
0 1 9
0 0 12
1 0 15
0 2 18
0 0 21
2 36 24
36 37 -1
36 36 30
37 37 33
36 26 3
1323752223 -1323752223 0

1836311903
```