COMPUTER SYSTEMS ASSIGMENT 2

By 184510

MAY 9, 2019

UNIVERSITY OF SUSSEX
Computer Systems

Question 1)

a) Algorithm

```
Function SKIP:
        Input: number of empties
        while(empties > 0)
                memory[pointer] = 0
                pointer++
                empties—
```

Description of algorithm: Algorithm above will pass zeros into the memory as many times as you tell it.

```
Function PRINT:
        Input: number of fills
        while(fills > 0)
                memory[pointer] = 1
                pointer++
                memory[pointer] = 0
                pointer++
                fills—
```

Description of the algorithm: Algorithm above will pass ones into the memory as many times as you tell it. Every time print is run it prints two characters at a time.

```
Function MAIN:
        Loop
                PRINT 8
                SKIP 1, PRINT 3, SKIP 2, PRINT 3, SKIP 1
                SKIP 2, PRINT 2, SKIP 4, PRINT 2, SKIP 2
                SKIP 3, PRINT 1, SKIP 6, PRINT 1, SKIP 3
                SKIP 16
```

Description of the algorithm: Algorithm above will use PRINT and SKIP algorithms in order to print the correct pattern within the loop until the end of the memory.

Testing:

First test:

```
7
8   /**
9    *
10   * @author Kacper
11   */
12  public class Pattern {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          print(8);
19          System.out.println();
20          skip(1);print(3);skip(2);print(3);skip(1);
21          System.out.println();
22          skip(2);print(2);skip(4);print(2);skip(2);
23          System.out.println();
24          skip(3);print(1);skip(6);print(1);skip(3);
25          System.out.println();
26          skip(16);
27      }
28
29      public static void skip(int empties){
30          while(empties > 0){
31              System.out.print(0);
32              empties--;
33          }
34      }
35
36      public static void print(int fills){
37          while(fills > 0){
38              System.out.print(1);
39              System.out.print(0);
40              fills--;
41          }
42      }
43
44  }
```

```
run:
1010101010101010
0101010001010100
0010100000101000
0001000000010000
0000000000000000
```

I have tested the algorithms first in java. Then I have realised that the pattern is upside down.

Second test:

```
6   package pattern;
7
8   /**
9    *
10   * @author Kacper
11   */
12  public class Pattern {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          skip(3);print(1);skip(6);print(1);skip(3);
19          System.out.println();
20          skip(2);print(2);skip(4);print(2);skip(2);
21          System.out.println();
22          skip(1);print(3);skip(2);print(3);skip(1);
23          System.out.println();
24          print(8);
25          System.out.println();
26          skip(16);
27      }
28
29      public static void skip(int empties){
30          while(empties > 0){
31              System.out.print(0);
32              empties--;
33          }
34      }
35
36      public static void print(int fills){
37          while(fills > 0){
38              System.out.print(1);
39              System.out.print(0);
40              fills--;
41          }
42      }
43
```

pattern.Pattern > main >

Test Results | Output - pattern (run) | Usages

```
run:
0001000000010000
0010100000101000
0101010001010100
1010101010101010
0000000000000000BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
0001000000010000
0010100000101000
0101010001010100
1010101010101010
0000000000000000
```

I have decided to adjust the code in main in order to get the pattern in correct order.

I have successfully managed to do it as shown on the photo on the left hand side.

Adjusted MAIN algorithm with fixed bugs:

```
Function MAIN:
        Loop
                SKIP 3, PRINT 1, SKIP 6, PRINT 1, SKIP 3
                SKIP 2, PRINT 2, SKIP 4, PRINT 2, SKIP 2
                SKIP 1, PRINT 3, SKIP 2, PRINT 3, SKIP 1
                PRINT 8
                SKIP 16
```

Description of the algorithm: This algorithm does exactly as the previous main but I have adjusted the order of the lines so the pattern is no longer upside down and it is fully correct.

Pseudo code after debugging in part C of this coursework:

```
Loop
        CHECK IF AT END OF MEMORY
                IF NOT JUMP LOOP
        HALT
        PRINT PATTERN FOR FIRST LINE
        PRINT PATTERN FOR SECOND LINE
        PRINT PATTERN FOR THIRD LINE
        PRINT PATTERN FOR FOURTH LINE
        INCREMENT A POINTER TO SKIP A LINE
```

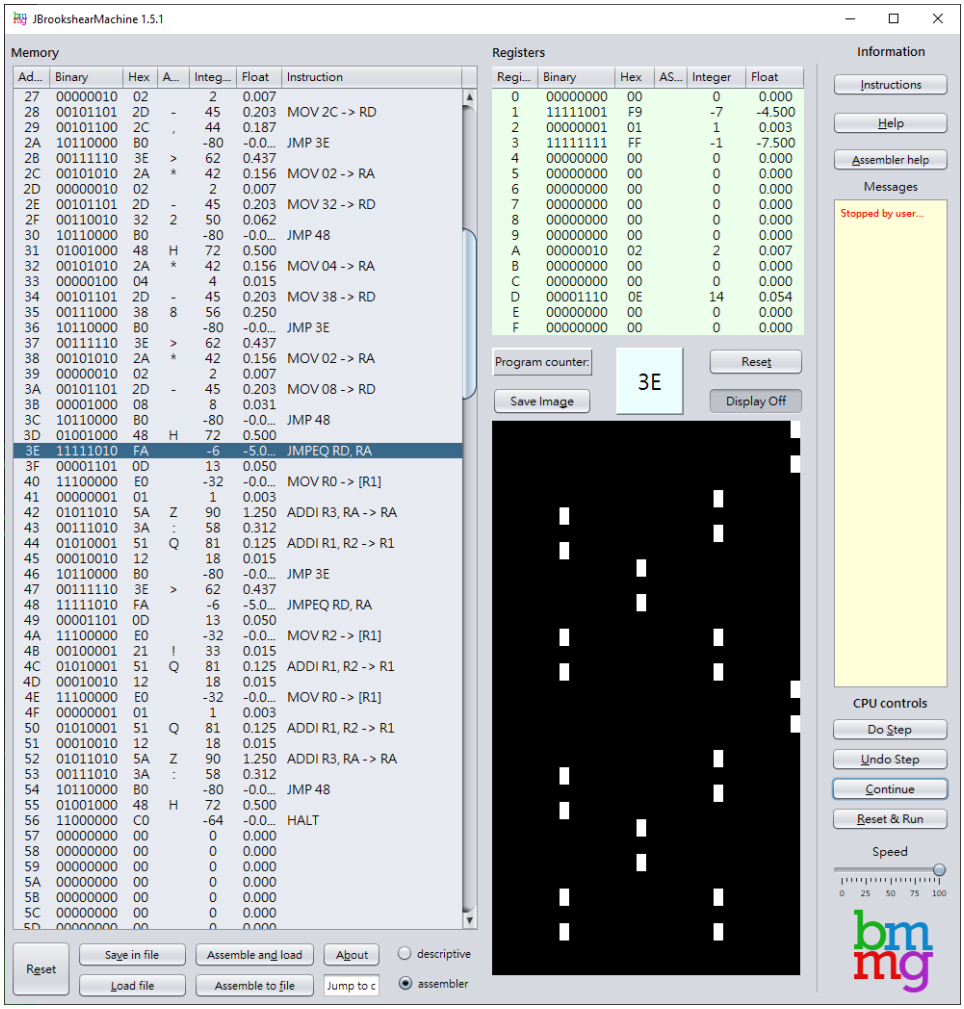
When I was writing the program, I realised I needed to print the pattern multiple times on the same line. So, I made an inner loop and now my Pseudocode is:

```
PATTERNS = [Array of Patterns]
PATTERNS ON LINE = 4
I=0
J=0
Loop
        CHECK IF AT END OF MEMORY
                IF NOT JUMP LOOP
        HALT
        Loop while I < PATTERNS ON LINE
                PRINT PATTERNS[J]
                INCREMENT I
        INCREMENT J
        CHECK IF AT END OF MEMORY
                IF NOT JUMP LOOP
        HALT
```

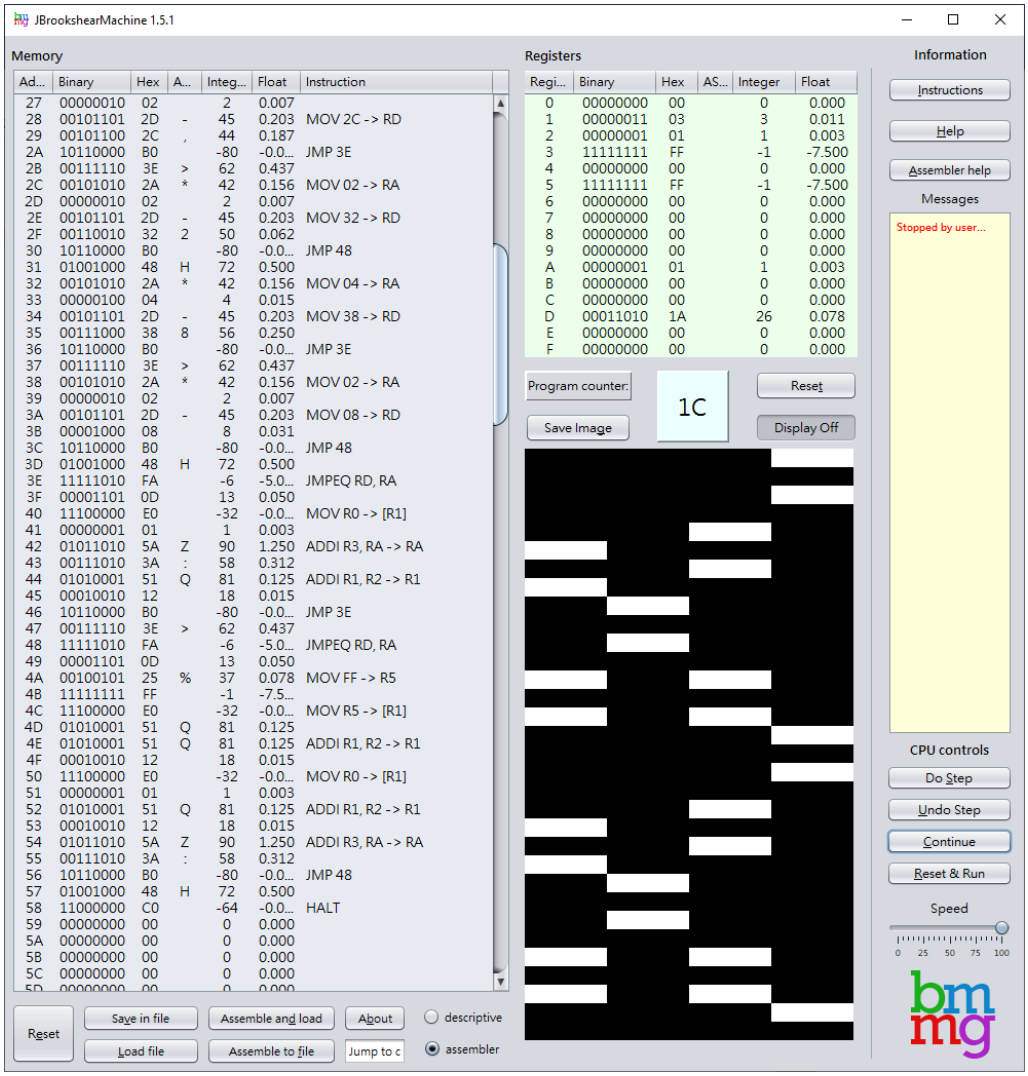
b) TEXT FILE THAT CAN BE FOUND IN ZIP FILE

c)

When I first run the program, I have realised there is a bug. The pixels seemed to be shifted.



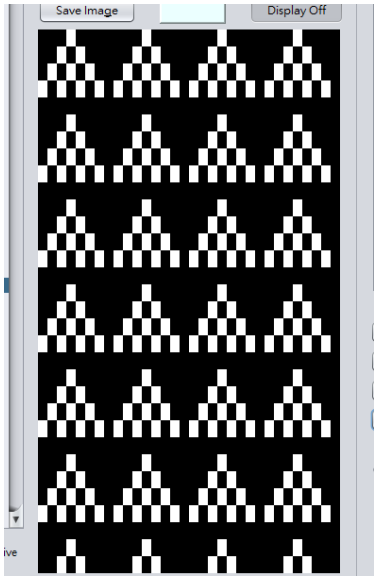
I have tried to debug this by writing the value FF instead of 1 into each memory location.



This is when I realised that the pixels of the display show the bits rather then the bytes. I have decided to rewrite my code to use hard coded bit patterns as I realised that it would take too long to write a program that generates it.

Perfect code description:

After looking at my pattern in the photo below, I have realised that the pattern is not exactly the one I was asked for:



The pattern presented in the coursework was overall much shorter. Then I have realised that each line should be printed twice. Luckily it was a very easy fix as all I had to do is change 4 to an 8 in my code shown below:

Pattern4_0.txt - Notepad

File Edit Format View Help

```
//This program will display a hard coded pattern

                MOV 1 -> R1                     //storing 1 for later
                MOV inner -> RB                 //storing inner loop for printing one line
                MOV [start_cell] -> RC          //load start of display memory

//I=0
//J=0
next_pattern:   MOV 0 -> RA                     //storing offset for pattern
start_line:     MOV 0 -> R4                     //storing offset for line
                MOV line_data -> RF             //load start of array
                ADDI RA, RF -> RF               //add index
                MOV [RF] -> RD                  //load pattern

//loop while I < patterns_on_line
inner:          MOV RD -> [RC]                  //write to display
//check if at end of memory
                MOV [end_cell] -> R0            //load end of display memory
                JMPEQ quit, RC                  //if at end of memory stop running
//increment I
                ADDI R1, RC -> RC               //increasing the pointer
                ADDI R1, R4 -> R4               //count number of printed patterns on line
                MOV [patterns_on_line] -> R0    //load max number of patterns on line
                JMPLT RB, R4                    //print again same pattern
//increment J
                ADDI R1, RA -> RA               //move pointer to next pattern
                MOV 5 -> R0                     //load last pattern number
                JMPEQ next_pattern, RA          //if last pattern reset to first
                JMP start_line                  //loop same pattern
quit:           HALT                            //stop program

start_cell:             DATA    80
end_cell:               DATA    FF
patterns_on_line:       DATA    4

//line_1:               DATA    00010000        //byte
//line_2:               DATA    00101000        //byte
//line_3:               DATA    01010100        //byte
//line_4:               DATA    10101010        //byte

line_data:              DATA    00010000,       00101000,       01010100,       10101010,       0000
//                              0               1               2               3               4
```

My pattern is made of 4 elements per line. By changing the number of “pattern_on_line” from 4 to 8, I have doubled the amount of times each element is being printed before going to the next pattern in the array.

Short description of the final program:

My program will display a hard-coded pattern. First, my program will load necessary information into memory. Then it will store both offsets, one for pattern and one for line. After this, the “inner” loop will be run. This loop will write 8 pixels to memory at a time, it will load the last cell into memory and if the last cell has been written the program will “halt”. However, if the last cell in memory has not been written then the pointer will be increased by one. Then the program will count how many patterns have been printed in a single line and check if the limit of them has been reached. If not, it will print pattern on the same line else it will start printing pattern on the new line.

d)

“R1” will store 1 for later use such as incrementing offsets. “RB” represents the inner loop that will be run to display pattern in a single line. “RC” represents the current cell that the program will write the pattern to. “RA” represents a number that represents which element of the pattern from the array is being used. “R4” represents the position in the line that is currently filled with patter. “RF” stores the first element of the pattern. “RD” stores the bit of the pattern (element of the array) that is about to be written. “R0” is used to check if the program is at the end of the memory. I have used the data instructions to store the start and the end of writable memory, and the array that stores the different patterns for each line. “JMPEQ quit” represents the part of the pseudo code that terminates the program. “JMPLT RB” represents the inner loop that displays the pattern 4 times twice in the same line in the pseudo code. “JMPEQ next_pattern” and “JMP start_line” represents the outer loop. “JMPEQ next_line” resets the pattern to the first one and “JMP start_line” starts a new line.

Question 2)

a) Interpreter

An interpreter is a program that as the name suggests interprets a high-level programming code into code that a computer or machine can understand [1]. Some integrated development environment provides both a complier and an interpreter as even that they do a similar job they have different benefits.

Compiler for example translates the entire program as a whole straight into machine code whereas the interpreter translates one statement at a time. The benefit of the compiler because of this is that it might take larger amount of time to analyse the source code, however overall program execution is much faster. However, the interpreter benefit is that it takes less amount of time to analyse the source code however the overall excitation takes longer. Interpreter is also more memory efficient as the object code is not generated immediately. Interpreter by executing a line at a time continues translating until the program find the first error which causes it to stop. This makes the debugging much faster as it shows which line caused the problem [2]. Another difference between a compiler and interpreter is that compiler convers the code into machine code which make the operating system run it directly whereas interpreter executes the code directly [3]. Interpreters are used for running scripts and smaller programs. This means that interpreters are very often installed within web servers. This allows the developer to execute scripts within webpages. Then these scripts can be easily edited and saved without the need to recompile the code [3].

IDEs like Python provide both, compiler and interpreter. This is most likely to give the program as much as help as he can need. For example, he might compile the program and if there is an error he can then use the interpreter to find where exactly the error is.

b) Disassembler

Disassembler is a program that translates machine language instructions into assembly language instructions. Disassembler allows reverse engineering. This means that cybersecurity expert might use one to investigate a computer virus. This will be done by decompiling malware to discover what it is doing. This means that virus machine language code can be translated back to assembly language that such an expert can understand [4].

He might want to do that in order to understand exactly what the virus is doing. By understanding the virus, he can then develop a piece of software that will be capable off detecting it and protecting the operating system against it or figuring out what the virus is trying to do.

One of the disadvantages could be that criminals that create viruses use techniques that make the process of reverse engineering a virus much harder [4]. This leads to another disadvantage, using a disassembler to investigate a virus is that it is very time consuming [5]. The biggest disadvantage of using an assembler to perform an inverse engineering process on virus is that the translation of machine code into high level language like C or C++ might not be fully accurate [6]. This is such a big problem as if from that translation software will be developed to protect from the virus, one wrong translation might cause trouble and the anti-virus software to fail and let the virus in. This of course would be tested before using this anti-virus in everyday life however this will make the development time of such software even longer. However, as the creator of the virus is able to make it hack the program translating it the developer of anti-virus cannot fully trust what disassembler displays.

References

1. Beal V. (Unknown) *interpreter.* Available at: https://www.webopedia.com/TERM/I/interpreter.html (Accessed: 25/04/2019)
2. Unknown (unknown) *Interpreter Vs Compiler: Difference Between Interpreter and Compiler.* Available at: https://www.programiz.com/article/difference-compiler-interpreter (Accessed: 25/04/2019)
3. Unknown (2010) *Interpreter.* Available at: https://techterms.com/definition/interpreter (Accessed: 26/04/2019)
4. Darril (Unknown) *Reverse Engineering in Malware.* Available at: https://blogs.getcertifiedgetahead.com/reverse-engineering-in-malware/ (Accessed: 02/05/2019)
5. Brew K. (2016) *Reverse Engineering Malware.* Available at: https://www.alienvault.com/blogs/labs-research/reverse-engineering-malware (Accessed: 02/05/2019)
6. Lysne O. (2018) *Reverse Engineering of Code.* Available at: https://link.springer.com/chapter/10.1007/978-3-319-74950-1_6 (Accessed at: 02/05/2019)