Contents

1 Basic Test Results

```
********** TESTING FOLDER STRUCTURE START **********
Running test4.sh:
File mult/Mult exists
File fill/Fill exists
File sort/Sort exists
File divide/Divide exists
Your logins are: muaz.abdeen, is that ok?

********** TESTING FOLDER STRUCTURE END **********

********** PROJECT TEST START **********

There are no presubmission tests for project 4!
Check on your own by using the simulators, it's more fun that way! :)
Don't forget to read the notes/FAQ in the moodle, they contain valuable hints!

********** PROJECT TEST END **********
```

2 README

```
muaz.abdeen
================================================================================
Muaz Abdeen, ID 300575297, muaz.abdeen@mail.huji.ac.il
================================================================================

                           Project 4 - Machine Language
                           ----------------------------


Submitted Files
---------------
(1)  README         - This file.
(2)  Mult.asm        - The Mult.asm implementation.
(3)  Fill.asm        - The Fill.asm implementation.
(4)  Sort.asm        - The Sort.asm implementation.
(5)  Divide.asm        - The Divide.asm implementation.


Remarks
-------
* In Mult.asm I used an algorithm that does not change the value stored
  in R1 (I prefered not to use R0 nor R1 as loop counter).
```

3 ./divide/Divide.asm

```
1  // File name: ~/divide/Divide.asm
2  // author: Muaz Abdeen
3  // Usage: set the values in R13 (dividend)
4  //        and R14 (divisor)
5  //          * both are strictly positive *
6
7
8  //**
9  // This program divides R13 by R14 and stores
10 // the quotient (R13/R14) in R15.
11 // This is an integer division (the remainder
12 // is discarded).
13 //
14 //       * Don't change the input registers *
15 //*
16
17 // *-(Long Division)-*
18 // k <- 0
19 // while (0 < divisor <= dividend):
20 //      divisor <<= 1
21 //      k <- k+1
22 // while (k > 0):
23 //      k <- k-1
24 //      divisor >>= 1
25 //     if (divisor <= dividend):
26 //        dividend -= divisor
27 //        quotient = (quotient << 1) + 1
28 //     else:
29 //        quotient <<= 1
30
31
32     // set variables: dividend & divisor
33     @R13
34     D=M
35     @dividend
36     M=D     // dividend = R13
37     @R14
38     D=M
39     @divisor
40     M=D     // divisor = R14
41     @R15
42     M=0     // R15 <- 0
43
44     // calculate k
45     @k
46     M=0
47   (KLOOP)
48       // check 0 < divisor
49       @divisor
50       D=M
51       @DIVISIONLOOP
52       D,JLE
53
54       // check divisor <= dividend
55       @divisor
56       D=M
57       @dividend
58       D=D-M
59       @DIVISIONLOOP
```

```
60      D,JGT
61
62      // shiftleft divisor
63      @divisor
64      D=M
65      M=M<<     // divisor <<= 1
66      // stores the divisor value before the last shift,
67      // to use it later in case the last shift caused
68      // the divisor to become negative.
69      @posDivisor
70      M=D
71
72      // increment k
73      @k
74      M=M+1     // k <- k+1
75
76      @KLOOP
77      0,JMP
78
79  (DIVISIONLOOP)
80      // decrement k then check (k >= 0)
81      @k
82      M=M-1
83      D=M
84      @END
85      D,JLT
86
87      // shiftright divisor
88       @divisor
89       D=M
90       @POSITIVE
91       D,JLT         // if the divisor is negative
92
93       @divisor
94      M=M>>         // divisor >>= 1
95      @ADJUST
96      0,JMP
97
98  (POSITIVE)
99      @posDivisor
100     D=M
101     @divisor
102     M=D
103
104 (ADJUST)
105     // check if (divisor <= dividend)
106     @divisor
107     D=M
108     @dividend
109     D=M-D         // dividend - divisor
110     @QUOTIENT
111     D,JLT         // if (divisor > dividend) goto QUOTIENT
112
113     @dividend
114     M=D         // dividend = dividend - divisor
115     @R15
116     M=M<<
117     M=M+1         // quotient = (quotient << 1) + 1
118
119     @DIVISIONLOOP
120     0,JMP
121
122 (QUOTIENT)
123     @R15
124     M=M<<        // quotient <<= 1
125
126     @DIVISIONLOOP
127     0,JMP
```

```
128
129     (END)
```

4 ./fill/Fill.asm

```
1   // This file is part of www.nand2tetris.org
2   // and the book "The Elements of Computing Systems"
3   // by Nisan and Schocken, MIT Press.
4   // File name: projects/04/Fill.asm
5
6   // Runs an infinite loop that listens to the keyboard input.
7   // When a key is pressed (any key), the program blackens the screen,
8   // i.e. writes "black" in every pixel;
9   // the screen should remain fully black as long as the key is pressed.
10  // When no key is pressed, the program clears the screen, i.e. writes
11  // "white" in every pixel;
12  // the screen should remain fully clear as long as no key is pressed.
13
14      // *-( FILL algorithm )-*
15      // while True:
16      //      address <- SCREEN
17      //      fill <- (KBD > 0) ? -1 : 0
18      //         while (address < KBD):
19      //            RAM[address] <- fill
20      //            address <- address + 1
21
22
23    (MAINLOOP)
24        // set initial variables
25      @SCREEN
26      D=A
27      @address
28      M=D             // address = SCREEN (M[address]=SCREEN)
29
30      // (KBD > 0) ? FILL : CLEAR
31      @KBD
32      D=M
33      @FILL
34      D,JGT         // if (KBD > 0) goto FILL
35      @CLEAR
36      D,JEQ         // if (KBD = 0) goto CLEAR
37
38    (FILL)
39        @fill
40        M=-1        // fill screen words with -1
41        @SCREENLOOP
42        0,JMP
43
44    (CLEAR)
45        @fill
46        M=0           // fill screen words with 0 (i.e clear it)
47        @SCREENLOOP
48        0,JMP
49
50    (SCREENLOOP)    // loop over all screen pixels
51        @fill
52        D=M         // fill (or clear) according to M[fill] value
53
54        @address
55        A=M         // set A-reg value to M[address] value
56        M=D           // fill (or clear) the word in address
57
58        @address
59        M=M+1         // increment the address
```

```
60
61      // check if reach end of the screen (address = KBD)
62      @KBD
63      D=A
64      @address
65      D=D-M         // D = KBD - address
66      @SCREENLOOP
67      D,JGT        // if (KBD > address) goto the SCREENLOOP
68
69      @MAINLOOP
70      0,JMP         // else: goto the MAINLOOP
71
72  (END)
```

5 ./mult/Mult.asm

```
1   // This file is part of www.nand2tetris.org
2   // and the book "The Elements of Computing Systems"
3   // by Nisan and Schocken, MIT Press.
4   // File name: projects/04/Mult.asm
5
6   // Multiplies R0 and R1 and stores the result in R2.
7   // (R0, R1, R2 refer to RAM[0], RAM[1], and RAM[2], respectively.)
8
9
10      // I used the algorithm (A) because it does not change the
11      // value stored in R1 (I prefered not to use R0 nor R1 as
12      // loop counter).
13      //
14      // algorithm (A)                // algorithm (B)
15      // R2 <- 0                        // R2 <- 0
16      // i <- 0
17      // while (i < R1):                // while (R1 > 0):
18      //        R2 <- R2 + R0            //        R2 <- R2 + R0
19      //        i = i + 1                //        R1 = R1 - 1
20
21
22      // R2 <- 0
23      @R2
24      M=0
25
26      // if R0=0 got to END
27      @R0
28      D=M
29      @END
30      D,JEQ
31
32      // i <- 0
33      @i
34      M=0
35
36    (LOOP)        // while i < R1:  R2 <- R2 + R0
37        @R1
38        D=M
39        @i
40        D=M-D    // D = i - R1
41        @END
42       D,JEQ    // if i = R1 goto END
43
44        @R0
45        D=M
46        @R2
47        M=M+D    // R2 = R2 + R0
48
49        @i
50        M=M+1    // i = i + 1
51
52       @LOOP
53       0,JMP
54
55    (END)
56        //@END
57        //0,JMP
```

6 ./sort/Sort.asm

```
1  // File name: ~/sort/Sort.asm
2  // author: Muaz Abdeen
3  // Usage: set the values in R14 (base address)
4  //        and R15 (length)
5
6  //**
7  // This program sorts in descending order the array
8  // starting at the address in R14 with length as
9  // specified in R15.
10 //*
11
12         // USED SORTING ALGORITHM:
13         //  *-(BUBBLE SORT)-*
14         //
15         // i <- 0
16         // while (i < R15-1):
17         //     j <- 0
18         //     while (j < R14-i-1):
19         //        if (arr[j] < arr[j+1]):
20         //            swap(arr[j], arr[j+1])
21         //        j <- j + 1
22         //     i <- i + 1
23
24
25
26      // set i index to 0
27      @i
28      M=0
29
30   (OUTERLOOP)
31       // check if i index out of bound (n-1)
32       @R15
33       D=M-1
34       @i
35       D=M-D
36       @END
37       D,JGE    // if i >= n-1 goto END
38
39       // set j index to 0
40      @j
41      M=0
42
43   (INNERLOOP)
44       // check if j index out of bound (n-i-1)
45       @i
46       D=M+1    // D = -(-i-1)
47       @R15
48       D=M-D    // D = n--(-i-1) = n-i-1
49       @j
50       D=M-D
51       @INCREMENTI
52       D,JGE    // if j >= n-i-1 goto INCREMENTI
53
54   (COMPARE)        // Compare arr[j] and arr[j+1]
55       @R14
56       D=M
57       @j
58       D=D+M     // D = arr + j
59
```

```
60        @currentIndex
61        M=D     // currentIndex = arr+j
62
63        A=D
64        D=M     // D = M[A] = M[arr + j]
65        @currentValue
66        M=D     // currentValue = arr[j]
67
68        @currentIndex
69         A=M+1     // A = arr+j+1
70         D=M     // D = M[A+1] = M[arr + j + 1]
71         @nextValue
72         M=D     // nextValue = arr[j+1]
73
74         // if arr[j+1] > arr[j] then swap arr[j+1] and arr[j]
75         @currentValue
76         D=D-M     // D = arr[j+1] - arr[j]
77        @SWAP
78        D,JGT
79
80        @INCREMENTJ
81        0,JMP
82
83    (SWAP)
84       // set arr[j+1] = currentValue
85        @currentIndex
86        A=M     // A = arr+j
87        D=M     // D = arr[j]
88        A=A+1     // A = arr+j+1
89        M=D     // arr[j+1] = arr[j]
90
91        // set arr[j] = nextValue
92        @nextValue
93        D=M     // D = arr[j+1]
94        @currentIndex
95        A=M     // A = arr+j
96        M=D     // arr[j] = arr[j+1]
97
98    (INCREMENTJ)
99        @j
100       M=M+1
101       @INNERLOOP
102       0,JMP
103
104   (INCREMENTI)
105       @i
106       M=M+1
107       @OUTERLOOP
108       0,JMP
109
110   (END)
```