FIT1008 – Intro to Computer Science
Solutions for Tutorial 3

Semester 1, 2018

Exercise 1

```
.data
prompt: .asciiz "Enter integer: "
nline:  .asciiz "\n"
n:      .word   0

.text


# read n
la      $a0, prompt     # load address of prompt into $a0
addi    $v0, $0, 4      # set syscall to 4
syscall                 # print prompt


addi    $v0, $0, 5      # set syscall to 5
syscall                 # read integer
sw      $v0, n          # let n = the integer read in

# while n > 1
loop:

lw      $t0, n          # $t0 = n
addi    $t1, $0, 1      # $t1 = 1
slt     $t2, $t1, $t0   # if n > 1 $t2 = 1 else $t2 = 0
beq     $t2, $0, end    # if $t2 <> 0 goto end

# print n
lw      $a0, n          # $a0 = n
addi    $v0, $0, 1      # set syscall 1
syscall                 # print n

# print nline
la      $a0, nline      # load address of nline into $a0
addi    $v0, $0, 4      # set syscall to 4
syscall                 # print newline

lw $t0, n
li $t1, 2
div $t0, $t1
mfhi $t2 # t2 = n % 2

bne $t2, $0, else # is odd
```

```
43
44  #even
45  li $t3, 3
46  mult $t0, $t3
47  mflo $t0
48  addi $t0, $t0, 1
49  sw $t0, n
50  j loop
51
52  else:
53  div $t0, $t1
54  mflo $t0
55  sw $t0, n # n = n//2
56  j loop
57
58  end:
59
60  lw      $a0, n          # $a0 = n
61  addi    $v0, $0, 1      # set syscall 1
62  syscall                 # print n
63
64
65  li $v0, 10 # set syscall to 10
66  syscall # exit
```

Exercise 2

Python implementation

```
1  size = int(input("Enter size: "))
2
3  list = [None] * size
4  i = 0
5  while i < size:
6      list[i] = int(input("Enter num: "))
7      i += 1
8
9  i = 0
10 product = 1
11 while i < size:
12     if i % 2 == 0:
13         product *= list[i]
14     i += 1
15
16 if size > 0:
17     print("Product: " + str(product))
```

MIPS implementation. Note: to simplify the solution, we've stored the list in the data segment, rather than dynamically creating it and reading in each of the values.

```
        .data
list:   .word 1,2,3,4,5,6,7,8,9,10
i:      .word 0
product: .word 0
size:   .word 10
out:    .asciiz "Product: "
nl:     .asciiz "\n"

        .text
main:   add $fp, $sp, $0

        # i = 0, product = 1
        sw $0, i
        addi $t0, $0, 1
        sw $t0, product

        # while i < size

while:  lw $t0, i
        lw $t1, size
        slt $t0, $t0, $t1
        beq $t0, $0, endloop

        # if i % 2 != 0
        lw $t0, i
        addi $t1, $0, 2
        div $t0, $t1
        mfhi $t0
        beq $t0, $0, endif

        # product *= list[i]
        la $t0, list
        lw $t1, i
        sll $t1, $t1, 2
        add $t0, $t0, $t1
        lw $t1, ($t0)
        lw $t2, product
        mult $t1, $t2
        mflo $t1
        sw $t1, product

endif:  # i += 1
        lw $t0, i
        addi $t0, $t0, 1
        sw $t0, i

        j while

endloop: # if size > 0:
        lw $t0, size
```

```
        sgt $t0, $t0, $0
        beq $t0, $0, exit

        # print("Product: " + str(product))
        addi $v0, $0, 4
        la $a0, out
        syscall
        addi $v0, $0, 1
        lw $a0, product
        syscall
        addi $v0, $0, 4
        la $a0, nl
        syscall

exit:   # Exit
        addi $v0, $0, 10
        syscall
```

Exercise 3

MIPS implementation. Note: to simplify the solution, we've stored the list in the data segment, rather than dynamically creating it and reading in each of the values.

```
        .data
list:                   .word 5         #contains the length
list_0:                 .word 1
list_1:                 .word 4
list_2:                 .word 8
list_3:                 .word 2
list_4:                 .word 1

i:                      .word 0
j:                      .word 0
is_palindrome:          .word 1

is_palindrome_prompt:   .asciiz "The␣list␣is␣a␣palindrome\n"
not_palindrome_prompt:  .asciiz "The␣list␣is␣not␣a␣palindrome\n"

        .text
#set j to len(list) -1
lw      $t0 list
subi    $t0 $t0 1
sw      $t0 j

#compare i to j
start_loop:
lw $t0 i
lw $t1 j

```

```
slt $t0 $t0 $t1          #if i < j, $t0 contains a 1

beq $t0 $0 end_loop      #if i >= j, we exit the while

#compare list[i] and list[j]

#get list[i]
lw $t0 i
li $t1 4
mult $t0 $t1             #compute offset for list[i]
mflo $t1
addi $t1 $t1 4

la $t0 list
add $t0 $t0 $t1          #compute the address of list[i]
lw $t2 ($t0)             #load list[i] into $t2

#get list[j]
lw $t0 j
li $t1 4
mult $t0 $t1             #compute offset for list[i]
mflo $t1
addi $t1 $t1 4

la $t0 list
add $t0 $t0 $t1          #compute the address of list[i]
lw $t3 ($t0)             #load list[j] into  $t3

#compare list[i] and list[j]
beq $t2 $t3 still_palindrome

#list[i] was different from list[j]

#print "List is not a palindrome"
la $a0 not_palindrome_prompt
li $v0 4
syscall

#set is_palindrome to False
sw $0 is_palindrome

still_palindrome:
#i = i + 1
lw $t0 i
addi $t0 $t0 1
sw $t0 i

#j = j - 1
lw $t0 j
subi $t0 $t0 1
sw $t0 j
```

```
78
79  j start_loop
80
81  end_loop:
82
83  #check if is_palindrome is still True
84  lw $t0 is_palindrome
85  beq $t0 $0 exit
86
87  #print "The list is a palindrome"
88  la $a0 is_palindrome_prompt
89  li $v0 4
90  syscall
91
92  exit:
93  li $v0 10
94  syscall
```

Exercise 4

(i) **sll** can be used to multiple integers by powers of 2, and **sra** can be used to divide integers by powers of 2.

(ii) 1

```
1          addi    $t0, $0, 6
2          addi    $t1, $0, 8
3          mult    $t0, $t1
4          mflo    $t1
```

2

```
1          addi    $t0, $0, 6
2          sll     $t1, $t0, 3
```