FIT1008 – Intro to Computer Science
Solutions for Tutorial 4

Semester 1, 2018

Exercise 1

```
function:
        addi $sp, $sp, -8       # allocate space for $ra and $fp
        sw $ra, 4($sp)          # save $ra on the stack
        sw $fp, 0($sp)          # save $fp on the stack
        addi $fp, $sp, 0        # copy $sp to $fp
        addi $sp, $sp, -4       # allocate 1 local variable 4bytes, say temp
        lw $t0, 8($fp)          # get argument a
        lw $t1, 12($fp)         # get argument b

        blt $t1, $t2, one        # if (b<a) goto one
        lw $t0, 8($fp)          # a
        sw $t0, -4($fp)         # temp = a
        j end                   # jump to end

one:    lw $t0, 12($fp)         # b
        sw $t0, -4($fp)         # temp = b

end:    lw $v0, -4($fp)         # return temp

        addi $sp, $sp, 4        # remove local variable temp
        lw $fp, 0($sp)          # restore $fp
        lw $ra, 4($sp)          # restore $ra
        addi $sp, $sp, 8        # remove $fp and $ra from stack
        jr $ra                  # return to address pointed to by $ra
```

The above code returns the maximum between the two arguments.
Associated Python code could have been:

```
def function(int a, int b):

   if b >= a:
      temp = a
   else:
      temp = b

   return temp
```

Exercise 2

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

        .text

        # setup frame
        addi    $fp, $sp, 0     # copy $sp to $fp

        # 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

        # n = collatz(n)
        addi    $sp, $sp, -4    # make space for 1 argument
        lw      $t0, n          # get value of n
        sw      $t0, 0($sp)     # pass n as arg1

        jal     collatz         # call collatz

        sw      $v0, n          # set n to the return value

        addi    $sp, $sp, 4     # remove space for arg on stack

        j loop
end:
        # exit
```

```
50          addi    $v0, $0, 10     # set syscall to 10
51          syscall                 # exit
52
53  collatz:
54          addi    $sp, $sp, -8    # make space for $fp and $ra
55          sw      $fp, 0($sp)     # store $fp on stack
56          sw      $ra, 4($sp)     # store $ra on stack
57          addi    $fp, $sp, 0     # copy $sp to $fp
58
59          lw      $t0, 8($fp)     # $t0 = arg1
60          addi    $t1, $0, 2      # $t1 = 2
61          div     $t0, $t1        # HI = arg1 % 2
62          mfhi    $t0             # $t0 = arg1 % 2
63          bne     $t0, $0, odd    # If $t0 <> 0 goto odd
64
65          lw      $t0, 8($fp)     # $t0 = arg1
66          addi    $t1, $0, 2      # $t1 = 2
67          div     $t0, $t1        # LO = $t0/$t1
68          mflo    $v0             # $v0 = LO
69
70          lw      $fp, 0($sp)     # restore $fp
71          lw      $ra, 4($sp)     # restore $ra
72          addi    $sp, $sp, 8     # remove space on stack for $fp and $ra
73
74          jr      $ra             # return to address pointed to by $ra
75
76  odd:    lw      $t0, 8($fp)     # $t0 = arg1
77          addi    $t1, $0, 3      # $t1 = 3
78          mult    $t0, $t1        # LO = 3*arg1
79          mflo    $t0             # $t0 = 3*arg1
80          addi    $v0, $t0, 1     # $v0 = 3*arg1 + 1
81
82          lw      $fp, 0($sp)     # restore $fp
83          lw      $ra, 4($sp)     # restore $ra
84          addi    $sp, $sp, 8     # remove space on stack for $fp and $ra
85
86          jr      $ra             # return to address pointed to by $ra
```

Exercise 3

Note: this translation assumes the list has already created in the heap and the reference is passed as a parameter to the function.

```
1
2                   .text
3
4   odd_product:    # save $fp and $ra
5                   addi    $sp, $sp, -8
6                   sw      $ra, 4($sp)
7                   sw      $fp, 0($sp)
8
9                   # update $fp
10                  addi    $fp, $sp, 0
11
12                  # allocate local variables
13                  addi    $sp, $sp, -8
14
15                  # setup product
16                  li      $t0, 1
17                  sw      $t0, -4($fp)    # save product
18
19                  # setup i
20                  li      $t0, 0
21                  sw      $t0, -8($fp)    # save i
22
23  prodloop:       lw      $t0, -8($fp)    # $t0 = i
24                  lw      $t1, 8($fp)     # $t1 = list
25                  lw      $t2, 0($t1)     # $t2 = len(list)
26                  bge     $t0, $t2, endprod       # check if i < len(list)
27
28                  # restore x
29                  lw      $t0, -8($fp)    # $t0 = i
30
31                  mul     $t3, $t0, 4     # $t3 = 4*i
32                  addi    $t3, $t3, 4     # $t3 = 4*i + 4
33                  add     $t3, $t3, $t1   # $t3 = address of list[i]
34                  lw      $t3, 0($t3)     # $t3 = list[i] = x
35
36                  # if x % 2 != 0
37                  li      $t4, 2
38                  div     $t3, $t4
39                  mfhi    $t4
40                  beq     $t4, $0,  else
41
42                  # product = product*x
43                  lw      $t4, -4($fp)
44                  mul     $t4, $t4, $t3
45                  sw      $t4, -4($fp)
46
```

```
47 else:            lw      $t0, -8($fp)
48                  addi    $t0, $t0, 1
49                  sw      $t0, -8($fp)
50
51                  j       prodloop
52
53
54 endprod:         # set return
55                  lw      $v0, -4($fp)
56
57                  # deallocate variables
58                  addi    $sp, $sp, 8
59
60                  # restore $fp and $ra
61                  lw      $fp, 0($sp)
62                  lw      $ra, 4($sp)
63                  addi    $sp, $sp, 8
64                  jr      $ra
```

Exercise 4

(i) The order is simply a convention; and it could happily be reversed, intertwined, etc, as long as everyone followed the same convention throughout function calls.

What happens when you have a variable number of arguments like you can have in Python?

(ii) Functions only need to access the memory at address `4($fp)` to restore the $ra to its original value right after jal was executed.