Assembly MIPS

Γεώργιος Κυριακόπουλος – el18153

Part A

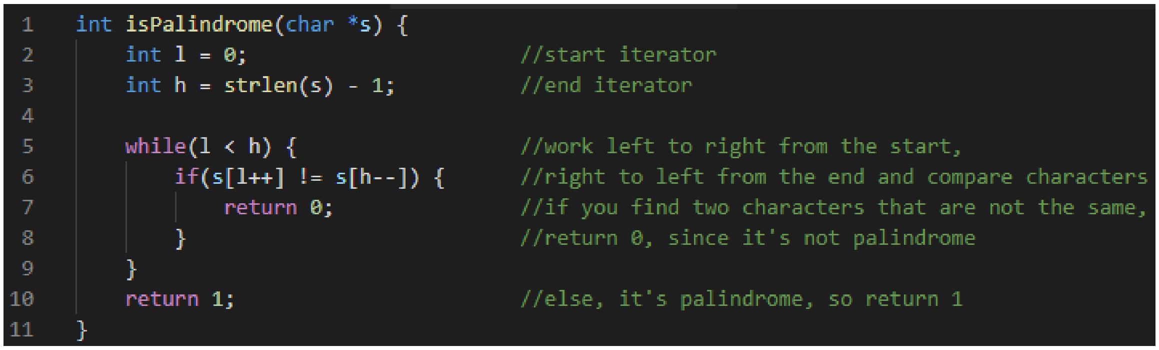
```
        addi    $sp,    $sp,    -8
        sw      $s1,    4($sp)                  #storing s1, s2, so we can restore them
        sw      $s2,    0($sp)                  #after we are done using them
        add     $t1,    $zero,  $zero           #t1 used as res and initialization res=0
        add     $t2,    $zero,  $zero           #t2 used as i and initialization i=0
FOR:    lw      $s1,    0($a0)                  #s1=v[i] (a0)
        lw      $s2,    0($a1)                  #s2=u[i] (a1)
        mul     $s1,    $s1,    $s2             #s1=v[i]*u[i]
        add     $t1,    $t1,    $s1             #res+=v[i]*u[i]
        addi    $a0,    $a0,    4               #next number of a0
        addi    $a1,    $a1,    4               #next number of a1
        addi    $t2,    $t2,    1               #i++
        slt     $t3,    $t2,    $a2             #t3=1, if i<n, meaning FOR loop goes on
        bne     $t3,    $zero,  FOR             #if t3!=0 (t3=1) then loop FOR
        add     $v0,    $t1,    $zero           #v0=res
        lw      $s2,    0($sp)                  #restoring s1 and s2 since we are done
        lw      $s1,    4($sp)
        addi    $sp,    $sp,    8
        jr      $ra
```

Part B – (check the C++ code snips included after the solution)

```
        addi  $sp,  $sp,   -4
        sw    $ra,  0($sp)              #saving $ra, because we are using jal later
        mov   $t0,  $a0                 #copy a0 to t0, because we need a0 as is later
        lbu   $t1,  0($t0)              #load first bit of a0 (t0) to t1 (a0=t0=*p=s)
        beq   $t1,  $zero, SUCC         #if we have an empty string, it’s palindrome
        jal   CNT                       #else we want to count the string length
        add   $t3,  $zero, $zero        #t3 used as l and initialization l=0
        addi  $t2,  $t2,   -1           #t2 is strlen and used as h=strlen(s)-1
        j     LOOP
LOOP:   slt   $t4,  $t3,   $t2          #t4=1, if l<h, meaning the LOOP should go on
        beq   $t4,  $zero, SUCC         #if t4=0, l>=h, so we are done with success
        add   $t5,  $t3,   $a0          #t5=s[l] (a0[l])
        add   $t6,  $t2,   $a0          #t6=s[h] (a0[h])
        bne   $t5,  $t6,   FAIL         #if s[l]!=s[h], the string is not palindrome
        addi  $t3,  $t3,   1            #l++, get to the next char from the start
        addi  $t2,  $t2,   -1           #h--, get to the previous char from the end
        j     LOOP
CNT:    addi  $t0,  $t0,   1            #next char of a0 (p++)
        lbu   $t1,  0($t0)              #current byte of a0 (t0) is loaded to t1
        bne   $t1,  $zero, CNT          #if it is not ‘\0’ loop CNT
        sub   $t2,  $t0,   $a0          #when loop is done, t2=strlen(s)=p-s
        jr    $ra
SUCC:   addi  $v0,  $zero, 1            #if all characters compared are equal, the
        j     EXIT                      #the string is palindrome, therefore v0=1
FAIL:   add   $v0,  $zero, $zero        #if there are unequal characters, the string
        j     EXIT                      #is not palindrome, therefore v0=0
```

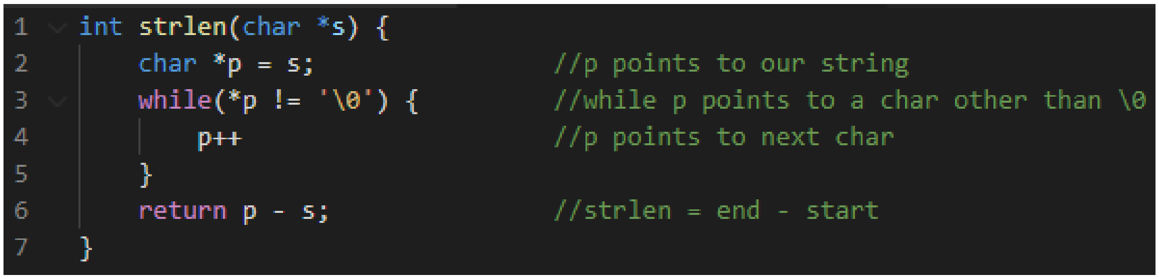
```
EXIT:   lw      $ra,    0($sp)                  #restore $ra, since jal may be used
        addi    $sp,    $sp,    4
        jr      $ra
```

The solution is based on the 2 following C++ code snips. The first one is an algorithm that checks if a string is palindrome, while the second one is an algorithm that counts the length of a string.



```
int isPalindrome(char *s) {
    int l = 0;                      //start iterator
    int h = strlen(s) - 1;          //end iterator

    while(l < h) {                  //work left to right from the start,
        if(s[l++] != s[h--]) {      //right to left from the end and compare characters
            return 0;               //if you find two characters that are not the same,
        }                           //return 0, since it's not palindrome
    }
    return 1;                       //else, it's palindrome, so return 1
}
```



```
int strlen(char *s) {
    char *p = s;                //p points to our string
    while(*p != '\0') {         //while p points to a char other than \0
        p++                     //p points to next char
    }
    return p - s;               //strlen = end - start
}
```

Part C

```
        li      $t0,    36                      #36 is ASCII code for ‘$’
        li      $t1,    43                      #43 is ASCII code for ‘+’
        li      $t2,    45                      #45 is ASCII code for ‘-‘
        li      $t3,    42                      #42 is ASCII code for ‘*’
        li      $t4,    47                      #47 is ASCII code for ‘/’
        j       LOOP
LOOP:   lb      $t5,    0($a0)                  #load current byte (ASCII code) of a0 to t5
        beq     $t5,    $t0,    EXIT            #check if t5 is equal to t0 (char = $, eof)
        slti    $t6,    $t5,    48              #check if t5 is an operator (operators<48=‘0’)
        beq     $t6,    $zero,  PUSH            #if t6=0, we push the number to the stack
        beq     $t5,    $t1,    ADDN            #if t5=’+’ (addition) go to ADDN
        beq     $t5,    $t2,    SUBN            #if t5=’-‘ (subtraction) go to SUBN
        beq     $t5,    $t3,    MULN            #if t5=’*’ (multiplication) go to MULN
        beq     $t5,    $t4,    DIVN            #if t5=’/’ (division) go to DIVN
EXIT:   lw      $v0,    0($sp)                  #v0 = result (top of stack, last number left)
        addi    $sp,    $sp,    4               #release its stack memory
        jr      $ra
PUSH:   addi    $sp,    $sp,    -4              #allocate space for word (4 bytes)
        addi    $t5,    $t5,    -48             #convert ASCII code to integer
        sw      $t5,    0($sp)                  #store the integer on the stack
        addi    $a0,    $a0,    1               #move a0 to its next byte (ASCII code)
        j       LOOP
ADDN:   lw      $t7,    0($sp)                  #retrieve second operand (top of stack)
        addi    $sp,    $sp,    4               #release its stack memory
        lw      $t8,    0($sp)                  #retrieve first operand (top of stack)
        add     $t8,    $t8,    $t7             #t8+=t7, addition of integers
        sw      $t8,    0($sp)                  #renew t8 with the answer on stack
        addi  $a0,  $a0,  1          #move a0 to its next byte (ASCII code)
        j     LOOP
SUBN:   lw    $t7,  0($sp)           #retrieve second operand (top of stack)
        addi  $sp,  $sp,  4          #release its stack memory
        lw    $t8,  0($sp)           #retrieve first operand (top of stack)
        sub   $t8,  $t8,  $t7        #t8-=t7, subtraction of integers
        sw    $t8,  0($sp)           #renew t8 with the answer on stack
        addi  $a0,  $a0,  1          #move a0 to its next byte (ASCII code)
        j     LOOP
MULN:   lw    $t7,  0($sp)           #retrieve second operand (top of stack)
        addi  $sp,  $sp,  4          #release its stack memory
        lw    $t8,  0($sp)           #retrieve first operand (top of stack)
        mul   $t8,  $t8,  $t7        #t8*=t7, multiplication of integers
        sw    $t8,   0($sp)          #renew t8 with the answer on stack
        addi  $a0,  $a0,  1          #move a0 to its next byte (ASCII code)
        j     LOOP
DIVN:   lw    $t7,  0($sp)           #retrieve second operand (top of stack)
        addi  $sp,  $sp,  4          #release its stack memory
        lw    $t8,  0($sp)           #retrieve first operand (top of stack)
        div   $t8,  $t8,  $t7        #t8/=t7, division of integers
        sw    $t8,  0($sp)           #renew t8 with the answer on stack
        addi  $a0,  $a0,  1          #move a0 to its next byte (ASCII code)
        j     LOOP
```