1η Σειρά Ασκήσεων Αρχιτεκτονικής Υπολογιστών

Ονοματεπώνυμο: Κουστένης Χρίστος
A.M: el20227
Ακαδημαϊκό Έτος: 2022-2023

ΜΕΡΟΣ Α

Οι το μέρη του κωδικά που έπρεπε να συμπληρωθούν είναι σημειωμένα :

```
        addi $s1, $s0, 12
LOOP: lw $t0, 0($s1)
        beq $t0, $zero, END
        div $t0, $s2
        slt $t1, $t0, 50
        beq $t1, $zero, ELSE
        mfhi $t0
        jmp NEXT
ELSE:   mflo  $t0
NEXT:  sw $t0, 0($s1)
        addi $s1, $s1, 4
        jmp LOOP
END:
```

ΜΕΡΟΣ Β

Θεωρούμε ότι τα arguments των ρουτίνων που υλοποιήσαμε είναι αποθηκευμένα στους $a0,$a1.

```
# $a0 -> address of first element of array A
# $a1 -> number of elements in the array
```

Bubble Sort Ρουτίνα

```
#############################################################
BubbleSort: addi $sp,$sp,-16 # Κάνουμε χώρο στη στοίβα για να υποδεχτεί τα δεδομένα
            sw $ra,0($sp)    # που θέλουμε να σώσουμε
            sw $a0,4($sp)
            sw $a1,8($sp)
            sw $s0,12($sp)

            move $s0,$a0 #Αποθηκεύουμε στον $s0 την διεύθυνση της αφετηρίας του πίνακα
            addi $t0,$zero,-1# Αρχικοποιούμε στον $t0 την μεταβλητή i = -1 του εξωτερ. βρόχου
            addi $t1,$a1,-1  # Αποθηκεύουμε στον $t1 το μέγεθος του πίνακα μειωμένο κατά -1
            addi $t2,$zero,1 # Αντιστοιχίζουμε την μεταβλητή swapped του C προγράμματος με τον
                             # τον $t2 καταχωρητή και την αρχικοποιούμε με 0(false)
            OuterLoop:addi $t0,$t0,1 # ++i (Αυξάνουμε το i κατά 1)
                      beq $t2,$zero,ExitBubble # if(!swapped) break;(Αντίστοιχη του C προγρ.)
                      move $t2,$zero # bool swapped = false = 0
                      slt $t3,$t0,$t1 # Is i < N-1 ?
                      beq $t3,$zero,ExitBubble # if not go to ExitBubble
                      move $t3,$zero # $t3 -> j = 0(Αρχικοποιήση του j για το εσωτερικό loop)
                      InnerLoop: sll $t4,$t3,2# j = 4*j αφού όλες οι λέξεις στη μνήμη είναι ευθυγραμμισμένες
                                              # ώστε να ξεκινούν από διεύθυνση που είναι πολλαπλάσιο του 4

                                 sub $t5,$t1,$t0 # $t5 <- N-i-1
                                 slt $t5,$t3,$t5 # Is j < N-i-1 ?
                                 beq $t5,$zero,OuterLoop # if not return to OuterLoop with bigger i
                                 add $t5,$t4,$s0 # $t5 <- address A[j]
                                 move $a0,$t5 # $a0 <- address A[j]
                                 addi $t5,$t5,4 # address A[j+1]
                                 move $a1,$t5 # $a1 <- address A[j+1]
                                 lw  $t7,0($a0) # $t7 <- A[j]
                                 lw  $t8,0($a1) # $t8 <- A[j+1]
                                 slt  $t6,$t8,$t7 # Is A[j] > A[j+1] ?
                                 beq $t6,$zero,InnerLoop # if not repeat Inner loop  for bigger j
                                 addi $t3,$t3,1 # ++j for the next repetition of the loop
                                 jal Swap # else Swap
                                 addi $t2,$zero,1 # swapped = true
                                 j InnerLoop # repeat InnerLoop
ExitBubble:  lw $s0,12($sp) # Επαναφορά από τη στοίβα στους καταχωρητές
             lw $a1,8($sp)  # των τιμών που είχαμε σώσει στην αρχή της Bubblesort
             lw $a0,4($sp)
             lw $ra,0($sp)
             addi $sp,$sp,16 # pop()
             jr $ra # Επιστροφή στη διεύθυνση PC+4
```

Cocktail Sort ρουτίνα

```
#####################################################################
CocktailSort: addi $sp,$sp,-16 # Κάνουμε χώρο στη στοίβα και σώζουμε τις τιμές των καταχωρητών
              sw $ra,0($sp)    # που θα χρειαστούμε
              sw $a0,4($sp)
              sw $a1,8($sp)
              sw $s0,12($sp)

              move $s0,$a0     # start address of array
              addi $t3,$a1,-1  # $t3 <- end = N-1
              addi $t0,$zero,1 # swapped = true
              move $t1,$zero   # $t1 <- (start = 0)
              While: beq $t0,$zero,ExitWhile # If no swapping took place go to ExitWhile
                     move $t0,$zero # swapped = false
                     move $t2,$zero # index of first loop i = 0
                     L1: sll $t4,$t2,2 # $t4 <- 4*i (Ευθυγραμμισμένη λέξεις στην μνήμη)
                         slt $t5,$t2,$t3 # Is i < end ?
                         beq $t5,$zero,ExitL1 #if not exit L1
                         add $t4,$t4,$s0 # $t4 <- start of array address + 4*i
                         la $a0,0($t4)    # $a0 <- address A[i]
                         la $a1,4($t4)    # $a0 <- address A[i+1]
                         lw $t6,0($a0)    # $t6 <- A[i]
                         lw $t7,0($a1)    # $t7 <- A[i+1]
                         slt $t4,$t7,$t6  # Is A[i+1] < A[i] ?
                         beq $t4,$zero,L1 # if not repeat L1
                         addi $t2,$t2,1   # ++i
                         jal Swap         # else Swap
                         addi $t0,$t0,1   # swapped = true
                         j L1             # repeat L1
              ExitL1:  beq $t0,$zero,ExitWhile  # if(!swapped) break;
                       move $t0,$zero #if not set it false again
                       addi $t3,$t3,-1 # end--
                       move $t2,$t3 # index of second loop j = end-1
                      L2: sll $t4,$t2,2 #Is j = 4*j ?
                         slt $t5,$t1,$t2 # Is start < j ?
                         beq $t5,$zero,ExitL2 # if yes then exit L2
                         addi $t2,$t2,-1 # --j (Έτσι κι αλλίως δε θα ξαναχρησιμοποιηθεί μέχρι την )
                                         # επόμενη επανάληψη)
                         add $t4,$t4,$s0  # $t4 <- start of array + 4*j
                         la $a0,0($t4)    # address A[j]
                         la $a1,4($t4)    # address A[j+1]
                         lw $t6,0($a0)    # $t6 <- A[j]
                         lw $t7,0($a1)    # $t7 <- A[j+1]
                         slt $t4,$t7,$t6  # Is A[j+1] < A[j] ?
                         beq $t4,$zero,L2 # if not repeat L2
                         jal Swap         # else Swap
```

```
                        addi $t0,$t0,1   # swapped = true
                        j L2             # repeat L2
              ExitL2: addi $t1,$t1,1 # start++
                      j While       # repeat While
ExitWhile: lw $ra,0($sp) # Επαναφορά στοιχείων που είχαν σωθεί στη στοίβα και pop() (Επιστροφή του stack
           lw $a0,4($sp) # pointer στη θέση που ήταν πριν την κλήση της CocktailSort
           lw $a1,8($sp)
           lw $s0,12($sp)
           addi $sp,$sp,16
           jr $ra        #
```

Insertion Sort Ρουτίνα

```
#########################################################################
InsertionSort:addi $sp,$sp,-4  # Στην InsertionSort δε χρησιμοποιήθηκε η swap και άρα ούτε οι καταχωρητές
              sw $ra,0($sp)    # $a0 και $a1 από άλλη υπορουτίνα άρα δε χρειάστηκε να τους σώσουμε στη στοίβα
              addi $t0,$zero,1 # i = 1
              LooP: sll $t1,$t0,2 # $t1<- 4*i για να προσπελάσουμε διευθύνσεις μνήμης ευθυγραμμισμένων λέξεων
                    slt $t2,$t0,$a1 # Is i < N?
                    beq $t2,$zero,ExitInsertion #if not Exit
                    add $t1,$t1,$a0 # start of array + 4*i
                    lw $t3,0($t1) # key
                    addi $t4,$t0,-1 # j
                    WhilE: slt $t5,$t4,$zero #  is j < 0?
                           bne $t5,$zero,ExitWhilE # if yes exit while
                           sll $t5,$t4,2 # 4*j
                           add $t5,$t5,$a0 # $t5 <- start of array address + 4*i
                           lw $t6,0($t5) # $t6 <- A[j]
                           slt $t7,$t3,$t6 # Is key < A[j]?
                           beq $t7,$zero,ExitWhilE # if not go to ExitWhilE
                           sw $t6,4($t5) # A[j+1] = key
                           addi $t4,$t4,-1 # --j
                           j WhilE # repeat While loop
              ExitWhilE:  sll $t5,$t4,2
                          add $t5,$t5,$a0
                          sw $t3,4($t5) # A[j+1] = key
                          addi $t0,$t0,1 # ++i
                          j LooP
ExitInsertion:lw $ra,0($sp)
              addi $sp,$sp,4
              jr $ra
```

Η **ρουτίνα Swap** που κλήθηκε από τις παραπάνω :

```
#############################################################################################
Swap: addi $sp,$sp,-8
      sw $t1, 0($sp)
      sw $t2, 4($sp)
      #pushed to the stack

      #swap
```

```
      lw $t1, 0($a0)
      lw $t2, 0($a1)
      sw $t1, 0($a1)
      sw $t2, 0($a0)

      #  all back to normal and pop
      lw $t2,4($sp)
      lw $t1,0($sp)
      addi $sp,$sp,8
      jr $ra
#############################################################################
```

ΜΕΡΟΣ Γ

Ο κώδικας σε C για την αναδρομική Insertion Sort στον οποίον στηρίξαμε τον κώδικα Assembly :

```
void RecursiveInsertionSort(int *A, int N){
   if (N <= 1) // end of recursions
      return;
   RecursiveInsertionSort(A, N-1 );
   int nth = A[N-1];
   int j = N-2;
   while (j >= 0 && A[j] > nth){
      A[j+1] = A[j];
      j--;
   }
   A[j+1] = nth;
}
```

```
RecursiveInsertionSort: addi $sp,$sp,-16 # Κάνω χώρο στη στοίβα και σώζω τις απαραίτητες μεταβλητές
                        sw $ra,0($sp)
                        sw $a0,4($sp)
                        sw $a1,8($sp)
                        sw $s1,12($sp)

                        slti $t0,$a1,1 # Is 1 > N ?
                        bne $t0,$zero,ExitAndRestore # if yes go to ExitAndRestore
                        move $s1,$a1 # $s1 <- N

                        addi $a1,$a1,-1 # $a1 <- n-1
                        jal RecursiveInsertionSort #RecursiveInsertionSort(A, N-1 )

                        sll $t0,$a1,2 # 4*(N-1)
                        add $t0,$a0,$t0 # address of A[N-1]
                        lw  $t3,0($t0) # nth = A[N-1]
                        addi $t0,$a1,-1 # j = N-2
```

```
                        WHILE: slt $t4,$t0,$zero # Is j<0?
                               bne $t4,$zero,Exit_2 # if yes then go to Exit_2

                               sll $t4,$t0,2 # 4*j
                               add $t4,$t4,$a0 # start + 4*j = [j]
                               lw $t5,0($t4) # $t5 <- A[j]
                               slt $t6,$t3,$t5 # nth < A[j]?
                               beq $t6,$zero,Exit_2 # if not leave while

                               sw $t5,4($t4) # A[j+1] = A[j]
                               subi $t0,$t0,1 # --j
                               j WHILE # repeat the loop
                        Exit_2: sll $t4,$t0,2 # 4*j
                                add $t4,$a0,$t4 # $t4 <- &A[j](address of A[j])
                                sw $t3,4($t4) # nth = A[j+1]

ExitAndRestore:lw $ra,0($sp) # Αποκατάσταση στοίβας και αρχικών τιμών στους καταχωρητές
               lw $a0,4($sp)
               lw $a1,8($sp)
               lw $s1,12($sp)
               addi $sp,$sp,16
               jr $ra
```