ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΩΝ ΣΥΣΤΗΜΑΤΩΝ

1η ΑΣΚΗΣΗ ΣΤΗΝ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. Έτος 2021-2022, 5ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

(τμήμα ΚΑΤ-ΠΑΠΑΓ)

ΟΝΟΜΑ : ********

ΕΠΩΝΥΜΟ : ********

Α.Μ. : 031****

ΜΕΡΟΣ Α

Για την συμπλήρωση του κώδικα assembly του MIPS απαιτείται πρώτα να αντιστοιχήσουμε τις μεταβλητές του κώδικα από την γλώσσα C σε καταχωρητές του επεξεργαστή MIPS.
Από τα συμπληρωμένα τμήματα του κώδικα καθώς και από την εκφώνηση της άσκησης προκύπτουν οι παρακάτω αντιστοιχίες:

```
int array1[30], array2[30], count;
int *x, *y;

count = 0;
x = &array1[5];
y = &array2[10];

for (count = 0; count < 10; count++) {
    if (*x > * y)
        *x += *y;
    else
        *y -= *x;
    x++;
    y++;
}
```

```
        add  $t0, $zero, $zero
        addi $s2, $s0  , 20
        addi $s3, $s1  , 40
LOOP:   lw   $t1, 0($s2)
        lw   $t2, 0($s3)
        slt  $t3, $t2, $t1
        beq  $t3, $zero, else
        add  $t1, $t1 , $t2
        sw   $t1, 0($s2)
        jmp  next
else:   sub  $t2, $t2, $t1
        sw   $t2, 0($s3)
next:   addi  $s2,  $s2, 4
        addi  $s3,  $s3, 4
        addi  $t0,  $t0, 1
        subi  $t5, $t0, 10
        bne   $t5, $zero, LOOP
```

Ωστόσο να διευκρινίσουμε την λειτουργία των καταχωρητών που χρησιμοποιούνται και εν τέλει την λειτουργία του τι επιτελείται στην κάθε γραμμή του κώδικα σε assembly MIPS. Έτσι :

- $zero : Μηδενικός καταχωρητής
- $s0 : Διεύθυνση πρώτου στοιχείου array1
- $s1 : Διεύθυνση δεύτερου στοιχείου array2
- $s2 : Αποθήκευση μεταβλητής x
- $s3 : Αποθήκευση μεταβλητής y
- $t0 : Προσωρινή αποθήκευση μετρητή

- $t1 : Προσωρινή αποθήκευση της τιμής *x
- $t2 : Προσωρινή αποθήκευση της τιμής *y
- $t3 : Προσωρινός καταχωρητής ελέγχου
- $t5 : Προσωρινός καταχωρητής αφαίρεσης

```
        add $t0,$zero,$zero   # αρχικοποιώ με count=0

        addi $s2,$s0,20   # ανάθεση x=&array1[5] (5*4=20 bytes)

        addi $s3,$s1,40 # ανάθεση x=&array2[10] (10*4=40 bytes)

LOOP: lw $t1, 0($s2) # ανάθεση στο $t1 της τιμής *x. Ανάθεση ετικέτας

        lw $t2,0($s3)   # ανάθεση στο $t2 της τιμής *y

        slt $t3,$t2,$t1    # έλεχος *y < *x .Αν ισχύει $t3=1 αλλιώς  $t3=1

        beg $t3,$zero,else # έλεγχος αν ο καταχωρητής $t3 ισούται με
$zero

        add $t1,$t1,$t2     # ισοδυναμία με *x+=*y;

        sw $t1,0($s2)     # ενημέρωση στην αντίστοιχη λέξη στην μνήμη με
την νέα τιμή *x

        jmp next # διακλάδωση χωρίς συνθήκη στην εντολή με ετικέτα next

else:   sub $t2,$t2,$t1  # ισοδυναμία με *y-=*x; Ανάθεση ετικέτας

        sw $t2,0($s3)  # ενημέρωση στην αντίστοιχη λέξη στην μνήμη με
την νέα τιμή *y

next:       addi $s2, $s2, 4 # x++ . Ανάθεση ετικέτας (1*4=4 bytes)

            addi: $s3,$s3,4 # y++ (1*4=4 bytes)

            addi: $t0,$t0,1 # count++
```

subi $t5,$t0,10 # Ανάθεση σε προσωρινό καταχωρητή το αποτέλεσμα της αφαίρεσης του μετρητή με το 10 για έλεγχο ορίων

bne $t5,$zero,LOOP # Αν το αποτέλεσμα δεν ισούται με μηδέν ($t5 !=0) τότε γίνεται επιστροφή στον βρόχο

ΜΕΡΟΣ Β

```
int binary_search(int *A, int N, int key) {
      return binary_search_rec(A, 0, N-1, key);
}
```

! $zero : Μηδενικός καταχωρητής
! $s0 : left
! $s1 : right
! $s2 : mid
! $s3 : key
! $s4 : A
! $s5 : N

```
addi $sp,$sp,-12          # ρύθμιση της στοίβας για 3 αντικείμενα
sw $ra,8($sp)             # αποθήκευση επιστροφής
sw $s0,4($sp)             # αποθήκευση ορίσματος left
sw $s1,0($sp)             # αποθήκευση ορίσματος right
addi $s0,$zero,0          # left = 0
addi $s1,$s5,-1           # right = - 1
jal binary_search_rec     # κλήση της  binary_search_rec
lw $ra,8($sp)             # επαναφορά της διεύθυνσης επιστροφής
addi $sp,$sp,12           # ρύθμιση δείκτη στοίβας για εξαγωγή 2 αντικειμένων
jr $ra                    # επιστροφή στον καλούντα
```

```
int binary_search_rec(int *A, int left, int right, int key) {
      int mid;

      if (right < left) return -1;

      mid = left + (right - left) / 2;
      if (A[mid] == key) return mid;

      else if (A[mid] > key)
            return binary_search_rec(A, left, mid-1, key);
      else
            return binary_search_rec(A, mid+1, right, key);
}
```

```
slt $t0,$s0,$s1                  # έλεγχος αν left < right
bne $t0,$zero,LOOP_1             # εαν ισχύει left < right μετάβαση στο  LOOP_1
LOOP_1:  addi $t6,$zero,-1       # επιστρέφει -1
          jr $ra                 # επιστροφή στον καλούντα
sub $t1,$s1,$s0                  # $t1=right-left
addi $t3,$zero,2                 # $t3=2
div $t2,$t1,$t3                  # $t2= (right-left) / 2
add $s2,$s0,$t2                  # $s2 = left + (right-left) / 2
sll $s2,$s2,2                    # προσωρινός καταχωρητής $s2 = 4 * $s2
add $s2,$s2,$s4                  # $s2= διεύθυνση του A[mid]
lw $t4,0($s2)                    # προσωρινός καταχωρητής $t4= A[mid]
beg $t4,$s3, LOOP_2              # αν A[mid] = key μετάβαση στο LOOP_2
bne $t4,$s3, LOOP_3              # αν A[mid] != key μετάβαση στο LOOP_3
LOOP_2: addi $v0,$t7,0           # $vo=mid
        jr $ra                   # επιστροφή στον καλούντα
LOOP_3: slt $t5, $t4,$s3         #  έλεγχος αν A[mid] < key
  bne $t5,$zero,LOOP_4  # εαν ισχύει A[mid] < key μετάβαση στο   LOOP_4
  beq $t5,$zero,LOOP_5 #  εαν ισχύει A[mid] > key μετάβαση στο   LOOP_5
LOOP_4:  addi $t7,$t7,-1         # $t7=mid-1
    jal binary_search_rec        # κλήση της  binary_search_rec
LOOP_5: addi $t7,$t7,1           # $t7=mid+1
    jal binary_search_rec        # κλήση της  binary_search_rec
```

```
int exponential_search(int *A, int N, int key) {
      int bound = 1;

      while(bound < N && A[bound] < key) {
            bound *=2;
      }

      if (bound < N-1)
            return binary_search_rec(A, bound/2, bound, key);
      else
            return binary_search_rec(A, bound/2, N-1, key);
}
```

```
!    $s6 : bound

addi $s6,$zero,1      # bound =1
addi $t0,$s5,-1       # $t0 = N-1

LOOP: slt $t1,$s6,$t0  # έλεγχος αν bound < N-1
      beq $t1,$zero, LOOP_6  # εαν ισχύει bound > N - 1 μετάβαση στο  LOOP_6
       bne $t1,$zero,LOOP_7 # εαν ισχύει bound < N -1 μετάβαση στο  LOOP_7

LOOP_7: addi $t2,$s6,$zero      # $t2 =bound
            sll $s6,$s6,2       # bound = bound * 2
            add $s6,$s6,$s4     # $s6= διεύθυνση του A[bound]
            lw $t3,0($s6)       # προσωρινός καταχωρητής $t3= A[bound]
            slt $t4,$t3,$s3     # έλεγχος αν A[bound]<key
       beq $t4,$zero,LOOP_6 # εαν ισχύει A[bound]>key μετάβαση στο  LOOP_6
       bne $t4,$zero,LOOP_8 # εαν ισχύει A[bound]<key μετάβαση στο  LOOP_8

LOOP_8: addi $ $t5,$zero,2     # $t5 = 2
           mult $s6,$s6,$t5    # $s6 = bound * 2
           j LOOP              # μετάβαση στο LOOP

LOOP_6: slt $t6,$s6,$t0        # έλεγχος αν bound < right
        bne $t6,$zero,LOOP_9 # εαν ισχύει bound < N -1 μετάβαση στο  LOOP_9
     beg $t6,$zero,LOOP_10 # εαν ισχύει bound < N -1 μετάβαση στο  LOOP_10

LOOP_9:  addi $t7,$zero,2          # $t7 = 2
            div $s6,$s6,$t7        # $s6 = bound / 2
            jal binary_search_rec  # κλήση της  binary_search_rec

LOOP_10: div $s6,$s6,$t7          # $s6 = bound / 2
             addi $s5,$s5,-1      # $s5 = N-1
             jal binary_search_rec # κλήση της  binary_search_rec
```

```
int interpolation_search(int *A, int N, int key) {
      int low = 0, high = N-1, pos;

      while (low <= up) {
            if ((key < A[low]) || (key > A[up]))
                  return -1;

            pos = low + (up-low) * (key-A[low])/(A[up]-A[low]);

            if (A[pos] == key)
                  return pos;
            else if (A[pos] > key)
                  up = pos - 1;
            else
                  low = pos+1;
      }
      return -1;
```

```
!   $s7 : low
!   $s8 : high
!   $s9 : pos
!   $t0 : up

addi $s7,$zero,0                    # low = 0
addi $s8,$s5,-1                     # high = N-1

LOOP: beg $t0,$s7,LOOP_11           # εαν up = low μετάβαση στο  LOOP_11
        slt $t0,$t0,$s7             # έλεγχος αν left < right
      bne $t0,$zero,LOOP_11         # εαν ισχύει up =1 μετάβαση στο  LOOP_11
        beq $t0,$zero,NEXT          # εαν ισχύει up = 0 μετάβαση στο  NEXT

LOOP_11:  addi $t1,$s7,0            #  $t1 = low
          sll $s7,$s7,2             # προσωρινός καταχωρητής $s7 = 4 * $s7
          add $s7,$s7,$s4           # $s7= διεύθυνση του A[low]
          lw $t2,0($s7)             # προσωρινός καταχωρητής $t2= A[low]
          slt $t1,$s3,$t2           # έλεγχος για key < A[low]
          bne $t1,$s3,LOOP_12       # εαν low != key μετάβαση στο LOOP_12

LOOP_12: addi $t3,$t0,0             # $t3=up
          sll $s8,$s8,2             # προσωρινός καταχωρητής $s8 = 4 * $s8
          addu $s8,$s8,$t0          # $s8= διεύθυνση του A[up]
          lw $t4,0($s8)             # προσωρινός καταχωρητής $t4= A[up]
          slt $t5,$t4,$s3           # έλεγχος αν A [up] < key
           bne $t5,$zero,LOOP_13    # εαν ισχύει A [up] < key μετάβαση στο
LOOP_13

LOOP_13: addi $t6,$zero,-1          # $t6 = -1
```

```
            addi $v0,$t6,0          # $v0 = -1
            jr $ra                  # επιστροφή στον καλούντα


sub $t3,$t3,$t1         # $t3 = up - low
sub $t7,$s3,$t2         # $t7 = key – A[low]
sub $t8,$t4,$t2         # $t8 = A[up] - A[low]
div $t7,$t7,$t8         # $t7 = (key-A [low]) / (A [up] – A [low])
mult $t7,$t3,$t7        # $t7 = (up – low) * (key-A [low]) / (A [up] – A [low])
add $s9,$s7,$t7         #s9 = low + (up – low) * (key-A [low]) / (A [up] – A [low])
addi $t6,$zero,0        # $t6=0
addi $t6,$s9,0          # pos =0
sll $s9,$s9,2           # προσωρινός καταχωρητής $s9 = 4 * $s9
add $s9,$s9,$s4         # $s9= διεύθυνση του A[pos]
lw $t9,0($s9)           # προσωρινός καταχωρητής $t9= A[pos]
beq $t9,$s3,LOOP_14     # εαν A[pos] = key μετάβαση στο  LOOP_14
bne $t9,$s3,LOOP_15     # εαν A[pos] != key μετάβαση στο  LOOP_15

LOOP_14: addi $vo,$t6,0     # $v0 = pos
         jr $ra             # επιστροφή στον καλούντα

LOOP_15: addi $t0,$zero,0       # $t0=0
         slt $t0,$t9,$s3        # έλεγχος για A[pos] < key
         bne $t0,$zero,LOOP_16  # εαν A[pos] < key μετάβαση στο  LOOP_16
         beq $t0,$zero,LOOP_17  # εαν A[pos] > key μετάβαση στο  LOOP_17

LOOP_16: addi $s7,$t6,1     # low = pos + 1
         j LOOP             # μετάβαση στο  LOOP

LOOP_17: addi $t3,$t6,-1    # up = pos -1
         j LOOP             # μετάβαση στο  LOOP

NEXT : addi $t5,$zero,0     # $t5 = 0 (αρχικοποίηση)
       addi $t5,$zero,-1    # $t5 = -1
       addi $vo,$t5,0       # $v0 = -1
       jr $ra               # επιστροφή στον καλούντα
```

ΜΕΡΟΣ Γ

Σε αυτό το μέρος θα πρέπει να γίνει η αναζήτηση με παρεμβολή με χρήση της αναδρομής. Η υλοποίηση αυτής σε γλώσσα C θα είναι :

```
int InterpolationSearch (int A[ ] ,int N, int key)
{
    int low =0, high = (N-1) ;
    while ( low<= high && key >= A[low] && key<= A[high])
   {
       if ( A[low] = key) return low;
       return -1;
    }
        int pos = low + (( (high-low) / ( A[high] – A[low] )) * (key- A[low])) ;
        if (A[pos] ==key)
        return pos;
        if (A[pos] < key)
        low = pos +1;
        else
        high = pos -1;
        }
    return -1;
}
```

Η υλοποίηση του ανωτέρω κώδικα σε ASSEMBLY MIPS θα είναι :

```
!   $a0 = A
!   $a1 = N
!   $a2= key
!   $s0 = low
!   $s1 = high
!   $s2 = pos

InterpolationSearch:
addi $s0,$zero,0                    #  low =0
addi $s1,$a1,-1                     #  high = N-1

WHILE: beq $s0,$s1,LOOP_1           # εαν low = high μετάβαση στο  LOOP_1
          slt $t2,$s0,$s1           # έλεγχος εαν low < high
          bne $t2,$zero, LOOP_1     # εαν low < high μετάβαση στο  LOOP_1
          beq $t2,$zero,NEXT        # εαν low > high μετάβαση στο  LOOP_1
LOOP_1:  addi $t0,$zero,$s0         # $t0= low
          sll $t0,$t0,2             # $t0 = $t0 *4
          add $t0,$t0,$a0           # $t0= διεύθυνση του A[low]
          lw $t0,0($t0)             # προσωρινός καταχωρητής $t0= A[low]
          beq $a2,$t0,LOOP_2        # εαν A[low]=key μετάβαση στο LOOP_2
```

```
            slt $t3,$a2,$t0              # έλεγχος για key < A[low]
            bne $t3,$zero,NEXT           # εαν  key < A[low] μετάβαση στο NEXT
            beq $t3,$zero,LOOP_2         # εαν  key > A[low] μετάβαση στο LOOP_2

LOOP_2: addi $t1,$zero,$s1               # $t1 = high
            sll $t1,$t1,2                # $t1 = $t1*4
            add $t1,$t1,$a0              # $t1= διεύθυνση του A[high]
            lw $t1,0($t1)                # προσωρινός καταχωρητής $t0= A[high]
            beq $a1,$t1,LOOP_4           # εαν A[high]=key μετάβαση στο LOOP_4
            slt $t4,$a1,$t1              # έλεγχος για key < A[high]
            bne $t4,$zero,LOOP_4         # εαν  key < A[high] μετάβαση στο LOOP_4
            beq $t3,$zero,NEXT           # εαν  key > A[high] μετάβαση στο NEXT

LOOP_3: addi $t5,$zero,-1          # $t5=-1
            addi $v0,$t5,$zero     # $v0=-1
            jr $ra                 # επιστροφή στον καλούντα

LOOP_4: beg $s0,$s1,LOOP_5     # εαν low = high μετάβαση στο  LOOP_5
            bne $s0,$s1,LOOP_6     # εαν low != high μετάβαση στο  LOOP_6

LOOP_5: beg $t0,$a2,LOOP_7     # εαν A[low]= key μετάβαση στο LOOP_7
            bne $t0,$a2,LOOP_3     # εαν A[low] != key μετάβαση στο LOOP_3

LOOP_6: sub $t6,$s1,$s0        # $t6 = high - low
            sub $t7,$t1,$t0          # $t7=A[high]- A[low]
            div $t6,$t6,$t7          # $t6= (high-low) / (A[high]-A[low])
            sub $t8,$a2,$t0          # $t8= key-A[low]
            mult $t6,$t6,$t8       # $t6 = ((high-low) / (A[high]-A[low])) * (key-A[low])
          add $s2,$t6,$s0  # pos = low+((high-low) / (A[high]-A[low])) * (key-A[low])
            addi $t9,$zero,$s2       # $t9 = pos
            sll $t9,$t9,2              # $t9 = $t9 *4
            add $t9,$t9,$a0            # $t9= διεύθυνση του A[pos]
            lw $t9,0($t9)              # προσωρινός καταχωρητής $t9= A[pos]
            beg $t9,$a2,LOOP_8     # εαν A[pos] = key μετάβαση στο  LOOP_8
            addi $t6,$zero,$zero       # $t6 = 0
            slt $t6,$t9,$a2            # έλεγχος για A[pos] < key
            bne $t6,$zero,LOOP_9   # αν A[pos] < key μετάβαση στο LOOP_9
            beq $t6,$zero,ELSE         # αν A[pos] > key μετάβαση στο ELSE

LOOP_7: addi $vo,$s0,$zero         # $v0 = low
            jr $ra                 # επιστροφή στον καλούντα

LOOP_8: addi $v0,$s2,$zero         # $v0 = pos
            jr $ra                 # επιστροφή στον καλούντα
```

```
LOOP_9: addi $s0,$s2,1      # low = pos +1
        j WHILE             # μετάβαση στο WHILE

ELSE:  addi $s1,$s2,-1      # high = pos -1
        j WHILE             # μετάβαση στο WHILE

NEXT: addi $t6,$zero,0      # $t6=0
      addi $t6,$zero,-1     # $t6 = -1
      addi $vo,$t6,0        # $v0 = -1
      jr $ra                # επιστροφή στον καλούντα
```