21/12/2015

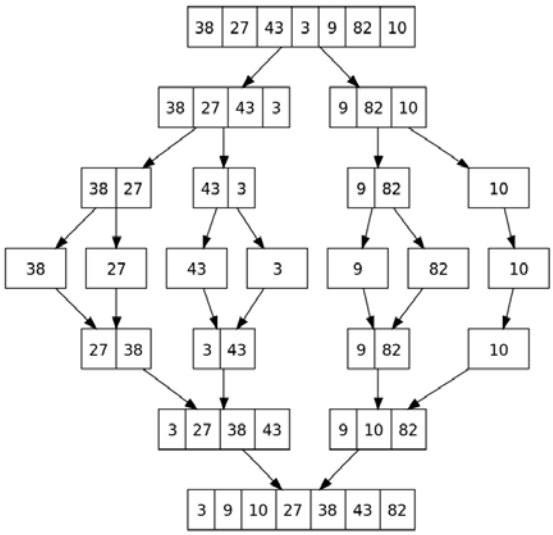
Αναφορά Εργαστηρίου 5

Ομάδα

Κολομβάκης Χρήστος (Α.Μ.: 2013030103)
Ζαχαριουδάκης Χρήστος (Α.Μ.: 2014030056)

Προεργασία

Κατά την προεργασία του εργαστηρίου μας ζητήθηκε η προσθήκη μιας νέας λειτουργίας ταξινόμησης στον κώδικα της Assembly , με την χρήση του αλγορίθμου ταξινόμησης merge sort . Ο αλγόριθμος merge sort λειτουργεί ως εξής :



Όπως φαίνεται και στην διπλανή εικόνα , ο πίνακας n στοιχείων διαχωρίζεται σε μικρότερους πίνακες , μέχρι ώσπου να καταλήξουμε σε n πίνακες με ένα μόνο στοιχείο . Έπειτα οι πίνακες αυτοί επανασυνδέονται σταδιακά , αλλά αυτή την φορά ταξινομημένοι . Αυτή η διαδικασία επαναλαμβάνεται μέχρι ωσότου να αναδημιουργηθεί ο αρχικός πίνακας , αλλά με τα στοιχεία του ταξινομημένα . Η ταξινόμηση γίνεται με τον έλεγχο κάθε στοιχείου του ενός πίνακα με κάθε στοιχείο του άλλου πίνακα. Το στοιχείο που θα προκύψει μικρότερο τοποθετείται στον επόμενο πίνακα στην κατάλληλη θέση . Δεν υπάρχει λόγος να γίνει έλεγχος μεταξύ των στοιχείων του ίδιου πίνακα καθώς έχει γίνει ήδη από προηγούμενη ταξινόμηση , όταν δημιουργήθηκε ο πίνακας .

Χρησιμοποιώντας τον αλγόριθμο merge sort , μας ζητήθηκε η ταξινόμηση σε αύξουσα σειρά με βάση το value . Επιπλέον μας ζητήθηκε η αντιμετώπιση του value ως short int , δηλαδή η αποθήκευση και η φόρτωση αυτού από την μνήμη με χρήση εντολών sb/lb και εντολών ολίσθησης (shift) . Οι εντολές που χρησιμοποιήσαμε για την διαχείριση του value μας δόθηκαν από τους υπεύθυνους του εργαστηρίου και η ακριβής λειτουργία τους αναλύεται παρακάτω .

Περιγραφή Ζητούμενων

Σκοπός του 5ου εργαστηρίου ήταν η περαιτέρω εξοικείωση με την γλώσσα προγραμματισμού Assembly , ιδιαίτερα στην χρήση αναδρομής , στην διαχείριση της στοίβας και την τήρηση των συμβάσεων που προβλέπει η Assembly , στην κλήση συναρτήσεων . Επίσης μας ζητήθηκε ως επιπρόσθετο ερώτημα η κλήση συνάρτηση μέσα σε συνάρτηση . Σε αυτή την περίπτωση , χρειάστηκε εκτός από την τήρηση των συμβάσεων

, η αποθήκευση (push) στην στοίβα (stack) του περιεχομένου του καταχωρητή $ra , των $s_ καταχωρητών και των καταχωρητών $t_ και η επαναφορά τους (pop) μετά την κλήση της συνάρτησης .

Ο καταχωρητής $ra αλλάζει τιμή κατά την εκτέλεση της εντολής **jal** και περιέχει την διεύθυνση της εντολής μετά την jal ,όπου θα πρέπει να επιστρέψει ο PC μετά την ολοκλήρωση της συνάρτησης . Έτσι είναι απαραίτητη η αποθήκευση του όταν καλούμε συνάρτηση μέσα σε συνάρτηση , ώστε η αρχική συνάρτηση να επιστρέψει στο σωστό σημείο της main (ή της συνάρτησης που την κάλεσε) όταν εκτελεστεί η εντολή **jr $ra .** Οι καταχωρητές $s_ αποθηκεύονται λόγω συμβάσεων και οι καταχωρητές $t_ αποθηκεύονται αυτοί των οποίων οι προηγούμενες τιμές χρειάζονται και μετά την εκτέλεση της συνάρτησης .

Επίσης απαραίτητη για την λειτουργία της αναδρομική συνάρτησης ήταν αποθήκευση και επαναφορά των ορισμάτων της , δηλαδή των τιμών που περνούσαν στους καταχωρητές $a_.

Περιγραφή της Εκτέλεσης

Εκτελέσαμε τον κώδικα της Assembly στο πρόγραμμα PCSPIM , όπου εξετάστηκε η ορθή λειτουργία του και η γνώση μας στην εντολές τις Assembly . Συγκεκριμένα η λειτουργία του προγράμματος εξετάστηκε με την δημιουργία 5 κόμβων με αρνητικές και θετικές τιμές , με την εκτέλεση της επιλογής ταξινόμησης από το μενού του προγράμματος και η εξακρίβωση της σωστής ταξινόμησης με σειριακή εκτύπωση των τιμών των κόμβων . Αφού ολοκληρώθηκε η σωστή λειτουργία του , εξεταστήκαμε στην κατανόηση του τρόπου λειτουργίας του κώδικα .

Συγκεκριμένα εξεταστήκαμε :

1. Στην κατανόηση λειτουργίας του κώδικα για **αποθήκευση ενός short int** από καταχωρητή στην μνήμη (store) οποίος μας δόθηκε από τους υπεύθυνους του εργαστηρίου :

Ο καταχωρητής *$s0 περιέχει τον short int και ο* καταχωρητής *$t4 την διεύθυνση μνήμης που θέλουμε να αποθηκεύσουμε*					
Εντολές Assembly	Εξήγηση	Χάρτης Μνήμης ($s0)			
		0	0	1	2
srl $s1, $s0, 4	Γίνεται ολίσθηση του αριθμού 4 θέσεις (bit) δεξιά , έτσι απομονώνεται το top byte , το οποίο αποθηκεύεται στον s1	0	0	**1**	2

andi $s0, $s0, 0x000F #s0	Απομονώνεται το bottom byte , με την λογική πράξη **and** του $s0 με τον αριθμό 15 (F), του όποιου τα 4 LSBs είναι 1 , αλλά τα υπόλοιπα είναι 0 . Άρα η πράξη αφήνει ανέγγιχτα τα 4 τελευταία bits του αριθμού .	0	0	0	**2**

Εντολές Assembly	Εξήγηση	Χάρτης Μνήμης ($t4)			
		0($t4)	1($t4)	2($t4)	3($t4)
sb $s1,0($t4)	Θεωρώντας ότι η αρχή της μνήμης από αριστερά στα δεξιά είναι η διεύθυνση 0($t4) , αποθηκεύεται πρώτα το top byte	**1**	0	0	0
sb $s1,0($t4)	Έπειτα ακριβώς διπλά στην διεύθυνση 1($t4) , το bottom byte .Έτσι αποθηκεύτηκε ο αριθμός στην μνήμη .	1	**2**	0	0

2. Στον κατανόηση λειτουργίας του αλγόριθμου merge soft , όπως περιγράφεται και στην προεργασία.
3. Στην κατανόηση των συμβάσεων των καταχωρητών callee – save και caller – save. Ειδικά για την αναδρομική συνάρτηση, πριν καλέσει τον εαυτό της, αποθηκεύουμε τις μεταβλητές που θα μεταβληθούν στο επόμενο κάλεσμά της, στην στοίβα. Όταν επιστρέψει ο έλεγχος στην συνάρτηση που την κάλεσε, τις επαναφέρουμε στην αρχική τους τιμή, διότι θα ξαναχρησιμοποιηθούν μέσα στην συνάρτηση και πρέπει να έχουν τις σωστές τιμές (Caller-Save). Στην συνάρτηση mergeSort, οι $t0, $t1, $t2, $t3 είναι Caller-Save, ενώ ο $ra, ο οποίος σώζεται και επαναφέρεται μόνο στον πρόλογο και στον επίλογο της συνάρτησης αντίστοιχα, είναι Callee-Save.
4. Στην κατανόηση της λειτουργίας των εντολών που κάνουν PUSH και POP στην στοίβα (stack) :
    A. PUSH :

```
addi $sp, $sp, -4
sw   $ra, ($sp)
```

Ο stack pointer ($sp) μειώνεται κατά 4 (4 bytes = 1 word), και έπειτα γίνεται εγγραφή σε αυτή την θέση μνήμης που άδειασε. Στην πραγματικότητα δεν γίνεται διαγραφή δεδομένων ,αλλά πανωγράφουμε σε δεδομένα που είναι πλέον άχρηστα σε εμάς .

B. POP :

```
lw $ra, ($sp)
addi $sp, $sp, 4
```

Εδώ φορτώνουμε τα δεδομένα από την μνήμη σε ένα καταχωρητή και τα 'διαγράφουμε' από την μνήμη αυξάνοντας τον $sp κατά 4 . Στην πραγματικότητα τα δεδομένα παραμένουν στην μνήμη , αλλά με την αύξηση του stack pointer δηλώνουμε ότι δεν τα χρειαζόμαστε πλέον , και το επόμενο push μπορεί να αποθηκεύσει σε αυτή την θέση μνήμης .

Παράδειγμα Εκτέλεσης:

Welcome. Enter a number from 1 to 9.

Press 1 to create a list with a specific number of values.
Press 2 to insert a value at the end of the list.
Press 3 to delete the last value of the list.
Press 4 to print a value of the list.
Press 5 to print the total number of nodes.
Press 6 to print the address of a node.
Press 7 to print the address of the list.
Press 8 to print the minimum value of the list.
Press 9 to sort the list.
Press 10 to exit the software.

1
Enter the number of nodes you want the list to have.
5
5 values will be inserted.
Enter the value to be inserted.
7
Enter the value to be inserted.
3

Enter the value to be inserted.
-45
Enter the value to be inserted.
-49
Enter the value to be inserted.
0
MENU
9
MENU
4
Enter which element you want printed. (1 for the first and so on...)
1
Id is : 1
Value is : -49

MENU
4
Enter which element you want printed. (1 for the first and so on...)
2
Id is : 2
Value is : -45

MENU
4
Enter which element you want printed. (1 for the first and so on...)
3
Id is : 3
Value is : 0

MENU
4
Enter which element you want printed. (1 for the first and so on...)
4
Id is : 4
Value is : 3

MENU
4
Enter which element you want printed. (1 for the first and so on...)
5
Id is : 5

Value is : 7

MENU

5

The total number of values of the list is: 5

MENU

3

MENU

4

Enter which element you want printed. (1 for the first and so on...)

5

The requested node does not exist.

MENU

5

The total number of values of the list is: 4

MENU

10

Συμπεράσματα

Αποκτήσαμε μεγαλύτερη εμπειρία στην υλοποίηση προγραμμάτων , χρησιμοποιώντας τις εντολές και τις συμβάσεις της γλώσσας Assembly για την υλοποίηση ενός δομημένου προγράμματος . Ιδιαίτερη σημαντική ήταν η εμπειρία που αποκτήσαμε στην κλήση συνάρτησης με των κατάλληλων εντολών και καταχωρητών . Θεωρήσαμε ιδιαίτερα ενδιαφέρουσα την λειτουργία της αναδρομικής συνάρτησης σε Assembly . Σε γλώσσες προγραμματισμού υψηλού επιπέδου όπως η C , διδαχθήκαμε οτι η αναδρομική λειτουργεί καλώντας τον εαυτό της συνεχώς μέχρι να πραγματοποιηθεί μια συνθήκη (non recursion statement) όπου και επιστρέφει στην συνάρτηση που την κάλεσε μέχρι τελικά να επιστέψει στην main. Σε κάθε εκτέλεση διατηρεί ένα ξεχωριστό αντίγραφο των τοπικών μεταβλητών της στην μνήμη . Με την χρήση της Assembly και της στοίβας , κατανοήσαμε βαθύτερα τον τρόπο με τον οποίο πραγματοποιείται η διαδικασία αυτή .

Παράρτημα - Κώδικας

A) Υλοποίηση σε Assembly

```
.data
.globl array
.globl Input
.globl Inst1
.globl Inst2
.globl Inst3
.globl Inst4
```

```
.globl Inst5
.globl Inst6
.globl Inst7
.globl Inst8
.globl Inst9
.globl err1
.globl err2
.globl err3
.globl ttl
.globl addr
.globl crMsg
.globl val
.globl id
.globl nline
.globl addfunc
.globl headadd
.globl err_notF
.globl errPos
.globl min1
.globl min2
.globl min3

min1: .asciiz "The address of the lowest value is:"
min2: .asciiz " The id of the lowest value is:"
min3: .asciiz " The lowest value is: "
headadd: .asciiz "The address of the list is: "
addfunc: .asciiz "The address of the variable is: "
nline: .asciiz "\n"
id: .asciiz "Id is : "
val: .asciiz "Value is : "
crMsg: .asciiz " values will be inserted.\n"
addr: .asciiz "Enter which value's address you want printed. (1 for the first and so on...)\n"
Input: .asciiz "Welcome. Enter a number from 1 to 9.\n"
Inst1: .asciiz "\nPress 1 to create a list with a specific number of values.\n"
Inst2: .asciiz "Press 2 to insert a value at the end of the list.\n"
Inst3: .asciiz "Press 3 to delete the last value of the list.\n"
Inst4: .asciiz "Press 4 to print a value of the list.\n"
Inst5: .asciiz "Press 5 to print the total number of nodes.\n"
Inst6: .asciiz "Press 6 to print the address of a node.\n"
Inst7: .asciiz "Press 7 to print the address of the list.\n"
Inst8: .asciiz "Press 8 to print the minimum value of the list.\n"
Inst9: .asciiz "Press 9 to sort the list.\n"
Inst10: .asciiz "Press 10 to exit the software.\n\n"
err1: .asciiz "Invalid input. Choose among 1 and 9.\n"
crList: .asciiz "Enter the number of nodes you want the list to have.\n"
insval: .asciiz "Enter the value to be inserted.\n"
print1: .asciiz "Enter which element you want printed. (1 for the first and so on...)\n"
ttl: .asciiz "The total number of values of the list is: "
err2: .asciiz "A list has already been created.\n"
err3: .asciiz "You need to create a list first.\n"
err_notF :  .asciiz "The requested node does not exist. \n"
```

```
errPos :  .asciiz "Please enter a positive integer.\n"

array: .align 2
      .space 800

temp: .align 2
.space 800

.text
.globl main

main:

la $s1, array
li $s2, 0 # the counter.
li $s3, 1 # id

move $t1, $s1

menu:

li $v0, 4
la $a0, Input
syscall

li $v0, 4
la $a0, Inst1
syscall

li $v0, 4
la $a0, Inst2
syscall

li $v0, 4
la $a0, Inst3
syscall

li $v0, 4
la $a0, Inst4
syscall

li $v0, 4
la $a0, Inst5
syscall

li $v0, 4
la $a0, Inst6
syscall

li $v0, 4
la $a0, Inst7
```

```
syscall

li $v0, 4
la $a0, Inst8
syscall

li $v0, 4
la $a0, Inst9
syscall

li $v0, 4
la $a0, Inst10
syscall

li $v0, 5
syscall

move $s0, $v0

blez $s0, err_label # The program checks if the user has given an input less than one...
bgt $s0, 10, err_label # ...or more than 9.

################################################################################
############

bne $s0, 1, case2 #the program will skip the code below if the user has not chosen one.
bne $s2, 0, err_l2

li $v0, 4
la $a0, crList
syscall

li $v0, 5
syscall

blez $v0 , menu # Ελεγχος για εισαγωγη μη θετικου αριθμού

move $s0, $v0
add $s2, $s2, $s0

move $a0, $s0 # the number of values to be inserted.
move $a1, $s1 # address of head.
move $a2, $s2 # counter.
move $a3, $s3 # id.

jal CreateList

move  $s3 , $v0
addi $s3 , $s3 , -1

b menu
```

```
#############################################################################
############

case2:

bne $s0, 2, case3
beq $s2, 0, err_l3

move $s0, $v0
addi $s2, $s2, 1

move $a1, $s1 # Address of head.
move $a2, $s2 # The counter.
move $a3, $s3 # The id.

jal insertNode

move $s3 , $v0

b menu

#############################################################################
############

case3:

bne $s0, 3, case4
beq $s2, 0, err_l3

move $a1, $s1 #Address of head.
move $a2, $s2 #The counter.

jal deleteLastNode

move $s2, $v0

b menu

#############################################################################
#############

case4:

bne $s0, 4, case5
beq $s2, 0, err_l3

li $v0, 4
la $a0, print1
syscall
```

```
li $v0, 5
syscall

move $s0, $v0

blez $s0, errPositive # The program checks if the user has given an input less than one...
bgt $s0, $s2, err_notFound # ...or a value bigger than the length of the list.

move $a0, $s0 # the value (its id) that we want printed.
move $a1, $s1 # Address of Head.
move $a2, $s2 # The counter.

jal PrintValue

b menu

################################################################################
#############

case5:

bne $s0, 5, case6
beq $s2, 0, err_l3

move  $a2 , $s2

jal NumberOfNodes

b menu

################################################################################
#############

case6:

bne $s0, 6, case7
beq $s2, 0, err_l3

li $v0, 4
la $a0, addr
syscall

li $v0, 5
syscall

blez $v0, errPositive # The program checks if the user has given an input less than one...
bgt $v0, $s2, err_notFound # ...or a value bigger than the length of the list.

move $a0, $v0

jal PrintValueAddress
```

```
b menu

################################################################################
#############

case7:

bne $s0, 7, case8
beq $s2, 0, err_l3

jal PrintListAddress

li $v0, 4
la $a0, nline
syscall


b menu

################################################################################
#############

case8:

bne $s0, 8, case9
beq $s2, 0, err_l3

move $a1, $s1 #Address of Head.
move $a2, $s2 #Counter.

jal FindMinValue

b menu

################################################################################
#############

case9:

bne $s0, 9, case10
beq $s2, 0, err_l3

move $a0 , $s1  # head of the array
move $a1 , $zero          # start of the array i
addi $s2 , $s2 , -1
move $a2 , $s2 # end of the array (num of elements) n-1
addi $s2 , $s2 , 1

jal mergeSort
```

```
b menu

################################################################################
#############

case10:

li $v0, 10
syscall

################################################################################
#############

err_label:

li $v0, 4
la $a0, err1
syscall

b menu

################################################################################
################

err_l2:

li $v0, 4
la $a0, err2
syscall

b menu

################################################################################
################

err_l3:

li $v0, 4
la $a0, err3
syscall

b menu

################################################################################
################

err_notFound:

li $v0, 4
la $a0, err_notF
syscall
```

```
b menu

################################################################################
################
errPositive:

li $v0, 4
la $a0, errPos
syscall

b menu

################################################################################
################

#main has ended

################################################################################
################

CreateList:

move $t0, $a0 #number of elements to be added.
move $t1, $a1 #address of the array.
move $t2, $a2 #counter
move $t3, $a3 #id.

li $v0, 1
move $a0, $t0
syscall

# The 3 lines above print the number of values that are going to be inserted.

li $v0, 4
la $a0, crMsg
syscall

# The message itself

inputloop:

sw $t3, 0($t1) # Enter the id in the array
addi $t1, $t1, 4

li $v0, 4
la $a0, insval
syscall

#Print string
```

```
############ Call second function

#Push

addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0 , $t1

jal StoreShortInt

move $t1 ,$v0

#Pop
lw $ra, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

#############

addi $t0, $t0, -1

addi $t3, $t3, 1

move $v0 , $t3

beq $t0, 0, exitfunc

b inputloop

exitfunc:

jr $ra

################################################################################
################

insertNode:

move $t0, $a0 #value to be inserted
move $t1, $a1 #address of the array
move $t2, $a2 #counter
move $t4, $a3 #id

addi $t4, $t4, 1 #increase id.

li $t3, 1 #variable to tranverse the list.

ins_loop:

addi $t1, $t1, 8
addi $t3, $t3, 1
```

```
bne $t3, $t2, ins_loop # branch

sw $t4, 0($t1)
addi $t1, $t1, 4

li $v0, 4
la $a0, insval
syscall

############ Call second function

#Push
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0 , $t1

jal StoreShortInt

#Pop
lw $ra, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

#############

move $v0 , $t4

jr $ra

################################################################################
################

deleteLastNode:

move $t1, $a1 #address of the array
move $t2, $a2 #counter


addi $t2, $t2, -1
move $v0, $t2

jr $ra

################################################################################
################

PrintValue:

move $t0, $a0 #value to be printed
move $t1, $a1 #address of the array
```

```
move $t2, $a2 #counter

############

#Push
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0, $t0 # the value (its id) that we want printed.
move $a1, $t1 # Address of Head.
move $a2, $t2 # The counter.

jal SearchNode

move $t1 ,$v0

#Pop
lw $ra, ($sp)                   # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

lw $t4, 0($t1)
addi $t1, $t1, 4

###########
#Push
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0 , $t1

jal ReadShortInt

move $t6,$v0

#Pop
lw $ra, ($sp)                   # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
##############

li $v0, 4
la $a0, id
syscall

li $v0, 1
move $a0, $t4
syscall

li $v0, 4
la $a0, nline
syscall
```

```
li $v0, 4
la $a0, val
syscall

li $v0, 1
move $a0,$t6
syscall

li $v0, 4
la $a0, nline
syscall

jr $ra

#############################################################################
###################

PrintValueAddress:

move $t0, $a0 #value to be printed
move $t1, $a1 #address of the array
move $t2, $a2 #counter

############

#Push
addi $sp, $sp, -4            # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0, $t0 # the value (its id) that we want printed.
move $a1, $t1 # Address of Head.
move $a2, $t2 # The counter.

jal SearchNode

move $t1 ,$v0

#Pop
lw $ra, ($sp)                        # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

###########

la $t4, 0($t1)

li $v0, 4
la $a0, addfunc
syscall

li $v0, 1
move $a0, $t4
```

```
syscall

li $v0, 4
la $a0, nline
syscall

jr $ra

################################################################################
###################

PrintListAddress:

move $t1, $a1 #address of the array

li $v0, 4
la $a0, headadd
syscall

li $v0, 1
move $a0, $t1
syscall

jr $ra

################################################################################
###################

FindMinValue:

move $t1, $a1 #Address of Head.
move $t2, $a2 #Counter (number of values).

addi $t2 , $t2 , -1 # The comparisons that must happen are counter - 1
lw $t3, 0($t1) # Holds the id of the lowest Value.
move $t4, $t1 # Holds the lowest value's address.

addi $t1, $t1, 4

###########
#Push
addi $sp, $sp, -4            # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0 , $t1

jal ReadShortInt

move $t6,$v0

#Pop
```

```
lw $ra, ($sp)                    # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
##############

beq $t2 , 0 , OneElement # The case where there is only one element

addi $t1, $t1, 8

MinLoop:

###########
#Push
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0 , $t1

jal ReadShortInt

move $t5,$v0

#Pop
lw $ra, ($sp)                    # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
##############

blt $t5, $t6, newMin
addi $t1, $t1, 8
addi $t2, $t2, -1

bne $t2, $zero, MinLoop
b endfunc

newMin:

move $t6, $t5 # Insert the new min value.
addi $t1, $t1, -4

lw $t7, 0($t1)
move $t3, $t7

move $t4, $t1 # Insert the new address.
addi $t2, $t2, -1
addi $t1, $t1, 12

bne $t2, $zero, MinLoop

endfunc:

li $v0, 4
la $a0, min1
```

```
syscall

li $v0, 1
move $a0, $t4
syscall

li $v0, 4
la $a0, nline
syscall

li $v0, 4
la $a0, min2
syscall

li $v0, 1
move $a0, $t3
syscall

li $v0, 4
la $a0, nline
syscall

li $v0, 4
la $a0, min3
syscall

li $v0, 1
move $a0, $t6
syscall

li $v0, 4
la $a0, nline
syscall

b endfunc2

#################

OneElement :

li $v0, 4
la $a0, min1
syscall

li $v0, 1
move $a0, $t4
syscall

li $v0, 4
la $a0, nline
syscall
```

```
li $v0, 4
la $a0, min2
syscall

li $v0, 1
move $a0, $t3
syscall

li $v0, 4
la $a0, nline
syscall

li $v0, 4
la $a0, min3
syscall

li $v0, 1
move $a0, $t6
syscall

li $v0, 4
la $a0, nline
syscall

endfunc2 :

jr $ra

################################################################################
################

NumberOfNodes :

move $t1,$a2

li $v0, 4
la $a0, ttl
syscall

li $v0, 1
move $a0, $t1
syscall

li $v0, 4
la $a0, nline
syscall

jr $ra
```

```
################################################################################
###############

mergeSort :

move $t0 , $a0 # a
move $t1 , $a1 # i
move $t2 , $a2 # j

bge $t1 , $t2 , after_iteration

add  $t3 , $t1 , $t2
srl $t3 , $t3 , 1 # mid

#Push
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $ra, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t0, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t1, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t2, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t3, ($sp)

move $a0 , $t0
move $a1 , $t1
move $a2 , $t3

jal mergeSort

#Pop
lw $t3, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t2, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t1, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t0, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $ra, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

#Push
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $ra, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t0, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t1, ($sp)
```

```
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t2, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t3, ($sp)

move $a0 , $t0
addi $t3 , $t3 , 1
move $a1 , $t3
addi $t3 , $t3 ,-1
move $a2 , $t2

jal mergeSort

#Pop
lw $t3, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t2, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t1, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t0, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $ra, ($sp)                  # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

#Push
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $ra, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t0, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t1, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t3, ($sp)
addi $sp, $sp, -4          # swse ton $ra sth stack
sw   $t2, ($sp) # j


move $a0 , $t0  # a
move $a1 , $t1  # i1
move $a2 , $t3  # j1
addi $t3 , $t3 , 1
move $a3 , $t3  # i2
addi $t3 , $t3 ,-1
# $t2 j2

jal merge

#Pop

lw $t2, ($sp)                  # fortwnw ton $ra apo th stack
```

```
addi $sp, $sp, 4
lw $t3, ($sp)                   # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t1, ($sp)                   # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $t0, ($sp)                   # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $ra, ($sp)                   # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

after_iteration :

jr $ra
################################################################################
################

merge :

addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $s4, ($sp) #
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $s5, ($sp) #
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $s6, ($sp) #
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $s0, ($sp) #
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $s1, ($sp) #

move $t6 , $a0 # a[0|i]
move $t7 , $a0 # a[0|j]

la $t4 , temp # temp[0|k]
la $t5 , temp # temp[0|j]

move  $s4 , $a1 # i | i = i1 #Parameters
move  $s5  , $a3 # j | j = i2

sll  $s0 , $s4 , 3 # i*8
add $t6 , $t6 , $s0 # a[i] | id
addi $t6 , $t6 , 4 # a[i] | value

sll  $s0 , $s5 , 3 # j*8
add $t7 , $t7 , $s0 # a[j] | id
addi $t7 , $t7 , 4 # a[j] | value

while1 :

        bgt $s4, $a2, exitLoop1
        bgt $s5, $t2, exitLoop1
```

```
##########
#Push
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0 , $t6

jal ReadShortInt

move $s0,$v0

#Pop
lw $ra, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
##############

##########
#Push
addi $sp, $sp, -4           # swse ton $ra sth stack
sw   $ra, ($sp)

move $a0 , $t7

jal ReadShortInt

move $s1,$v0

#Pop
lw $ra, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
##############

bge $s0 , $s1 , else1

###### Store Short Int
srl $t9, $s0, 4     #t9 holds the top byte
andi $s0, $s0, 0x000F #t8 hold the bottom byte

sb $t9,0($t4)      #Store the bytes in the memory in the correct order
sb $s0,1($t4)
######

addi $s4 ,$s4 , 1 # i++
addi $t6, $t6 , 8  # a[i++]
addi $t4, $t4 , 4  # temp[k++]

b while1

else1 :

          ###### Store Short Int
```

```
                srl $t9, $s1, 4      #t9 holds the top byte
                andi $s1, $s1, 0x000F #t8 hold the bottom byte

                sb $t9,0($t4)      #Store the bytes in the memory in the correct order
                sb $s1,1($t4)
                ######

                addi $s5 , $s5 , 1 # j++
                addi $t7, $t7 , 8  # a[j++]
                addi $t4, $t4 , 4  # temp[k++]
                b while1

exitLoop1 :

while2 :

        bgt $s4, $a2, exitLoop2

        ###########
        #Push
        addi $sp, $sp, -4           # swse ton $ra sth stack
        sw   $ra, ($sp)

        move $a0 , $t6

        jal ReadShortInt

        move $s0,$v0

        #Pop
        lw $ra, ($sp)                       # fortwnw ton $ra apo th stack
        addi $sp, $sp, 4
        ##############

        ###### Store Short Int
        srl $t9, $s0, 4      #t9 holds the top byte
        andi $s0, $s0, 0x000F #t8 hold the bottom byte

        sb $t9,0($t4)      #Store the bytes in the memory in the correct order
        sb $s0,1($t4)
        ######

        addi $s4 , $s4 , 1 # i++
        addi $t6, $t6 , 8  # a[i++]
        addi $t4, $t4 , 4  # temp[k++]
        b while2

exitLoop2 :

while3 :
```

```
        bgt $s5, $t2, exitLoop3

        ##########
        #Push
        addi $sp, $sp, -4           # swse ton $ra sth stack
        sw   $ra, ($sp)

        move $a0 , $t7

        jal ReadShortInt

        move $s1,$v0

        #Pop
        lw $ra, ($sp)                       # fortwnw ton $ra apo th stack
        addi $sp, $sp, 4
        ##############

        ###### Store Short Int
        srl $t9, $s1, 4     #t9 holds the top byte
        andi $s1, $s1, 0x000F #t8 hold the bottom byte

        sb $t9,0($t4)      #Store the bytes in the memory in the correct order
        sb $s1,1($t4)
        ######

        addi $s5 , $s5 , 1 # j++
        addi $t7, $t7 , 8  # a[j++]
        addi $t4, $t4 , 4  # temp[k++]
        b while3

exitLoop3 :

#Initialization

move  $s4 , $a1 # i | i = i1 #Parameters
move $s5 , $zero  #  j = 0;

la $t5 , temp # temp[0|j]
la $t6 , array # a[0|i]

sll  $s0 , $s4 , 3 # i*8
add $t6 , $t6 , $s0 # a[i] | id
addi $t6 , $t6 , 4 # a[i] | value

while4 :

        bgt $s4, $t2, exitLoop4    # !(i<=j2)

        ##########
        #Push
```

```
        addi $sp, $sp, -4           # swse ton $ra sth stack
        sw   $ra, ($sp)

        move $a0 , $t5

        jal ReadShortInt

        move $s0,$v0

        #Pop
        lw $ra, ($sp)                       # fortwnw ton $ra apo th stack
        addi $sp, $sp, 4
        ##############

        ###### Store Short Int
        srl $t9, $s0, 4     #t9 holds the top byte
        andi $s0, $s0, 0x000F #t8 hold the bottom byte

        sb $t9,0($t6)      #Store the bytes in the memory in the correct order
        sb $s0,1($t6)
        ######


        addi $s5 , $s5 , 1 # j++
        addi $s4 , $s4 , 1 # i++
        addi $t6, $t6 , 8  # a[i++]
        addi $t5, $t5 , 4  # temp[j++]
        b while4

exitLoop4 :

lw $s1, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $s0, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $s6, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $s5, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4
lw $s4, ($sp)                       # fortwnw ton $ra apo th stack
addi $sp, $sp, 4

jr $ra
################################################################################
################

SearchNode :

move $t0, $a0 #value to be printed
move $t8, $a1 #address of the array
move $t2, $a2 #counter
```

```
li $t9, 1 # variable to tranverse the list.

trav:

beq $t9, $t0, PrintVal

addi $t9, $t9, 1
addi $t8, $t8, 8

b trav

PrintVal :

move $v0 , $t8
move $v1 , $t0

jr $ra

################################################################################
################

StoreShortInt:

move $t1 , $a0 #Parameter

li $v0, 5            #Read input
syscall

move $t8, $v0    #t8 is the input

srl $t9, $t8, 4      #t9 holds the top byte
andi $t8, $t8, 0x000F #t8 hold the bottom byte

sb $t9,0($t1)      #Store the bytes in the memory in the correct order
sb $t8,1($t1)

addi $t1, $t1, 4

move $v0 , $t1

jr $ra

################################################################################
################
ReadShortInt:

move  $t1 , $a0

lb $t8,0($t1)       #t8 holds the top byte
lb $t9,1($t1)       #t6 holds the bottom byte
```

```
sll $t8, $t8, 4
or $t9, $t8, $t9

move $v0 , $t9

jr $ra

################################################################################
################
```