11/12/2015

Αναφορά 5ου εργαστηρίου

Προεργασία

Σ' αυτό το εργαστήριο κληθήκαμε να υλοποιήσουμε ,σε γλωσσά Assembly, την ταξινόμηση των στοιχείων της λίστας που δημιουργήσαμε σε προηγούμενα εργαστήρια , με χρήση στοίβας. Αρχικά, έπρεπε να μετατρέψουμε το value από int σε short σε όλη την έκταση του προγράμματος ,εκτός από τα σημεία που εμφανίζαμε αυτόν τον αριθμό(value). Έπειτα, χρειάστηκε να κατανοήσουμε τον τρόπο με τον οποίο ταξινομούνται τα στοιχειά συμφώνα με την merge sort και να την υλοποιήσουμε σε γλωσσά C,έτσι ώστε να μεταβούμε ομαλά στην Assembly. Τέλος, για να υλοποιήσουμε την συνάρτηση ταξινόμησης , κατανοήσαμε τη λειτουργιά της στοίβας , την ανάδρομη με χρήση στοίβας καθώς την ορθή χρήση caller-save ή called-save καταχωρητών, ώστε να σώζονται τα δεδομένα μετά την αναδρομική κλήση.

Περιγραφή ζητούμενων

Ο σκοπός αυτής της άσκησης είναι η περεταίρω εξοικείωση με την Assembly, μέσω της επαφής με την ανάδρομη στην γλωσσά αυτή, καθώς και με τη λειτουργιά της στοίβας. Για την υλοποίηση του κώδικα της αναδρομικής ταξινόμησης των στοιχείων , ήταν απαραίτητο να γνωρίζουμε ότι στην στοίβα πρέπει να σώσουμε τη διεύθυνση επιστροφής($ra), καθώς και τις παραμέτρους της συνάρτησης που θα μας χρειαστούν αργότερα($t). Γι' αυτό το λόγο διαχωρίσαμε ποιους καταχώρησες θέλουμε να χρησιμοποιήσουμε μετά τις κλήσεις , και βασιζόμενοι στις συμβάσεις, αποθηκεύαμε την τιμή τους και μετά την επαναφέραμε. Επιπλέον, για να μετατρέψουμε το value από integer σε short, αποθηκεύσαμε τον αριθμό με τέτοιο τρόπο έτσι ώστε να καταλαμβάνει αντί για 4 , 2 bytes.

Περιγραφή εκτέλεσης

Το πρώτο βήμα για την εκτέλεση της άσκησης ήταν να κάνουμε την μετατροπή του value σε short. Για να συμβεί αυτό , σπάσαμε τα bits που καταλαμβάνει το value, χωρίς να χάσουμε την πληροφορία , και την αποθηκεύσαμε στον ίδιο καταχωρείτε χωρίς τα λιγότερο σημαντικά bits.Μ' αυτό τον τρόπο δεν χάσαμε την πληροφορία του αριθμού. Το τμήμα του κώδικα μας, στο οποίο έγινε η μετατροπή ,είναι το παρακάτω:

```
lid $v0,5            # read value
syscall

move $t9, $v0

srl $t8, $t9, 4                # $t8 holds the top byte
andi $t9, $t9, 0x000F  # $t9 hold the bottom byte
sb $t8, 0($t1)       # Store the bytes in the memory in the correct order
sb $t9, 1($t1)
lb $t4,0($t1)          # $t4 holds the top byte
lb $t2,1($t1)          # $t2 holds the bottom byte
sll $t4, $t4, 4
or $t2, $t4, $t2


sh $t2,0($t1)        # move value to t2
```

Στην συνεχεία, έχοντας τον παρακάτω κώδικα στην C , ξεκινήσαμε να κάνουμε βήμα βήμα την μετατροπή σε Assembly.Η ταξινόμηση merge sort αποτελείται από δυο συναρτησεις,την mergeSort που είναι η αναδρομική ,και την merge που ταξινομεί κατάλληλα τα στοιχεία:

H MergeSort στην C	H merge στην C

void MergeSort(int *array, int left, int right)
{
int mid = (left+right)/2;
if(left<right)
{
/* Sort the left part */
MergeSort(array,left,mid);
/* Sort the right part */
MergeSort(array,mid+1,right);
/* Merge the two sorted parts */
Merge(array,left,mid,right);
}
	void Merge(int *array, int left, int mid, int right)
	{
	/*We need a Temporary array to store the new sorted part*/
	int tempArray[right-left+1];
	int pos=0,lpos = left,rpos = mid + 1;
	while(lpos <= mid && rpos <= right)
	{
	if(array[lpos] < array[rpos])
	{
	tempArray[pos++] = array[lpos++];
	}
	else
	{
	tempArray[pos++] = array[rpos++];
	}
	}
	while(lpos <= mid) tempArray[pos++] = array[lpos++];
	while(rpos <= right)tempArray[pos++] = array[rpos++];
	int iter;
	/* Copy back the sorted array to the original array */

```
        for(iter = 0;iter < pos; iter++)
        {
                array[iter+left] = tempArray[iter];
        }
        return;
}
```

Αρχικα ,στην Assembly, δημιουργήσαμε τον πρόλογο της συνάρτησης , δημιουργώντας χώρο για τις μεταβλητές που επιθυμούσαμε να κρατήσουμε σε caller-save καταχωρητές:

```
MergeSort:

addui $sp, $sp, -20        # swse ton $s0 sth stack

sw   $a0,0($sp)
sw   $a1,4($sp)              #left
sw   $a2,8($sp)              #number of elements
sw   $ra,16($sp)


move $t5,$a0
move $t1,$a1
move $t2,$a2

li $t4,2
div $t2, $t4
mfhi $t3                                  # $t3 holds mid , [mid = (right+left/2)]
move $a2,$t3

sw   $a3,12($sp)
```

Έπειτα , υλοποιήσαμε το «σώμα» της αναδρομικής συνάρτησης merge Sort ,προσέχοντας πριν και μετά από κάθε κλήση να αποθηκεύουμε τα δεδομένα που χρειαζόμασταν. Επιπλέον, κρατήσαμε κάθε κλήση της συνάρτησης στην στοίβα. Παρακάτω φαίνεται ένα κομμάτι από τον κώδικα της mergeSort , με τις δυο αναδρομικές κλήσεις της:

```
jal MergeSort

lw $t3, 8($sp)


addi $t3,$t3,1
move $a3,$t3

sw $a3,8($sp)
addi $sp, $sp, -4
```

```
sw $a2,($sp)
jal MergeSort
```

Στον επίλογο της συνάρτησης επαναφέραμε τον $sp , τον $rag και γενικά οποίο στοιχείο κινάμε «save»:

```
after_cond_sort:
lw $a0, 0($sp)       # restore $a0 (head of array) from stack
lw $a1, 4($sp)       # restore $a1 (num of elements) from stack
lw $ra, 16($sp)              # restore $ra from stack
lw   $a2,8($sp)   #num
lw   $a3,12($sp)

addui $sp, $sp, 20
jr $ra
```

<u>Συμπέρασμα</u>

Σ’ αυτό το εργαστήριο κατανοήσαμε την αναδρομή με χρήση στοίβας στην Assembly, τον ρόλο των καταχωρητών calle-save και caller-save μέσα στον πρόγραμμα, καθώς πώς να τους χρησιμοποιούμε σωστά . Μ’ αυτόν τον τρόπο, δεν θα χάνονται δεδομένα που επιθυμούμε να χρησιμοποιήσουμε και μετά τις κλήσεις των συναρτήσεων. Τέλος , μάθαμε ότι για να εκτελεστεί σωστά η αναδρομή , χρειάζεται ιδιαίτερη προσοχή στα ορίσματα.

<u>Παράρτημα</u>

```
.data

.globl main

.globl printmenu

.globl createArray

.globl insertArray

.globl deleteLast

.globl printItem

.globl numOfItems

.globl printAddressItem

.globl printArrayAddress
```

```
.globl printMinValue


                # Messages

menu_print:   .asciiz "\n\nType (1) to create the array\nType (2) to insert an item to array\nType (3) to delete the last item of array\nType (4) to print special item\nType (5) to print the number of items\nType (6) to print the address of special item\nType (7) to print the address of array\nType (8) to print the minimum value of all items of array\nType (9) to sort the array\nType (10) to exit\n"

askint:       .asciiz "\nPlease type your choice: "

askid:        .asciiz "\nGive id: "

askvalue:     .asciiz "\nGive value: "

bye:          .asciiz "\nProgram terminated. Bye!"

emptyarr:     .asciiz "\n----Empty Array----\n"

size:         .asciiz "\nGive the number of items: "

printinfo:    .asciiz "\nNode info: \n"

valueinfo:    .asciiz "\nValue: "

idinfo:       .asciiz "\nID: "

errornode:    .asciiz "\nThis node does not exist!\n"

idaddress:    .asciiz "\nID address: "

valueaddress: .asciiz "\nValue address: "

arrayaddress: .asciiz "\nAddress of array: "

minvalue:     .asciiz "\nMinimum value: "

number:       .asciiz "\nThe number of items is: "

node:         .asciiz "\nGive the item that you want: "

newline:      .asciiz "\n"


array: .align 2

            .space 800      # space for 100 nodes
```

```
mergeArray: .align 2
                                .space 800 # mergeArray is the array which we have the sorted
elements

.text

main:

la $s1, array          # $s1 is head of the array
li $s5,10              # $s5 is exit register
li $s6, 0              # initialization of $s6 which is the counter of nodes
move $t1, $s1

loop:

li $s4,1

jal printmenu

move $s0, $v0

case_1:                # case if choice is 1

bne $s0,$s4,case_2

li $v0, 4              # print how items want
la $a0, size
syscall
```

```
li $v0,5
syscall

move $s6, $v0                # $s6 has the number of items
move $a3, $s6                # counter as argument to $a3
move $a0, $s1                # head array as argument to $a1

jal createArray

move $s6, $v0
move $a0, $s1

jal printall

j loop

case_2:              # case if choice is  2

addi $s4,$s4,1

bne $s0,$s4,case_3

li $v0, 4            # message to read value
la $a0, askvalue
syscall
```

```
li $v0, 5           # read value
syscall
move $s2, $v0       # move value to $s2

li $v0, 4           # message to read id
la $a0, askid
syscall

li $v0, 5           # read id
syscall
move $s3, $v0               # move id to $s3

move $a0, $s1       # head array as argument to $a0
move $a1, $s2               # value as argument to $a1
move $a2, $s3       # ID as argument to $a2
move $a3, $s6               # counter as argument to $a3

jal insertArray

move $s6, $v0

li  $v0, 1          # print integer
move $a0, $s6
syscall

li  $v0, 4          # print new line
la  $a0, newline
```

```
syscall

move $a0, $s1         # head array as argument to $a0
move $a3, $s6                 # counter as argument to $a3
jal printall

li  $v0, 4            # print new line
la  $a0, newline
syscall

j loop

case_3:               # case if choice is 3

addi $s4,$s4,1

bne $s0,$s4,case_4

move $a0, $s1

move $a3, $s6

jal deleteLast

move $s6,$v0

move $a0, $s1         # head array as argument to $a0
```

```
move $a3, $s6                # counter as argument to $a3

jal printall

j loop

case_4:              # case if choice is 4

addi $s4,$s4,1

bne $s0,$s4,case_5

move $a0,$s1

jal printItem

j loop

case_5:              # case if choice is 5

addi $s4,$s4,1

bne $s0,$s4,case_6

move $a3, $s6                # counter as argument to $a3

jal numOfItems
```

```
j loop

case_6:             # case if choice is 6

addi $s4,$s4,1

bne $s0,$s4,case_7

move $a0,$s1

jal printAddressItem

j loop

case_7:             # case if choice is 7

addi $s4,$s4,1

bne $s0,$s4,case_8

move $a0, $s1

jal printArrayAddress

j loop

case_8:             # case if choice is 8
```

```
addi $s4,$s4,1
bne $s0,$s4,case_9

move $a0, $s1
move $a3, $s6

jal printMinValue


j loop

case_9:             # case if choice is 9

addi $s4,$s4,1
bne $s0,$s4,case_10

move $a0, $s1       # head array as argument to $a0
move $a1, $zero                 # 0 as argument to $a1
addi $t6, $s6, -1           # add -1 to counter
move $a2, $t6               # counter as argument to $a2


jal MergeSort

case_10:            # case 10 is the jump to loop label
```

```
li  $v0, 4           # print new line
la  $a0, newline
syscall
bne $s0, $s5,loop      # loop condition

li  $v0, 4           # print new line
la  $a0, newline
syscall


li $v0, 4            # print string of terminated program
la $a0, bye
syscall

li  $v0, 10          # syscall to exit of program
syscall

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

printmenu:           # menu function

li $v0, 4            # print menu string
la $a0, menu_print
syscall

li  $v0, 4           # print new line
la  $a0, newline
```

```
syscall

li $v0, 4            # print string
la $a0, askint
syscall

li  $v0, 5           # read choice
syscall


move $a0,$v0
move $v0,$a0
jr $ra                                  # return to main

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

createArray:

move $t6, $a3        # choice
move $t1, $a0                # head
li $t5,0             # counter


loop_1:

bge $t5,$t6,after_loop_1
li $v0,4
```

```
la $a0,askvalue       # print message  to ask value
syscall

li $v0,5              # read value
syscall

move $t9, $v0

srl $t8, $t9, 4                  # $t8 holds the top byte
andi $t9, $t9, 0x000F  # $t9 hold the bottom byte
sb $t8, 0($t1)        # Store the bytes in the memory in the correct order
sb $t9, 1($t1)
lb $t4,0($t1)            # $t4 holds the top byte
lb $t2,1($t1)            # $t2 holds the bottom byte
sll $t4, $t4, 4
or $t2, $t4, $t2


sh $t2,0($t1)         # move value to t2

li $v0,4
la $a0,askid          # print message ask id
syscall

li $v0,5              # read id
syscall
```

```
move $t3, $v0         # move id to $t3
sw $t3,4($t1)

addi $t5,$t5,1        # counter of nodes
addi $t1,$t1,8        # move head address

bne $t5,$t6,loop_1

after_loop_1:



move $v0, $t6

jr $ra

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

insertArray:

move $t1, $a0                 # $t1 has the head of array
move $t2, $a1                 # $t2 has the value
move $t3, $a2                 # $t3 has the ID
move $t6, $a3                 # $t7 has the number of elements


li $t4,0                      # $t4 is a help counter to the next while loop
```

```
while_label:          # the perpose of this loop is to go the head to the last element

bge $t4, $t6, after_loop

add $t1, $t1, 8                # move head to the next cellar
addi $t4, $t4, 1            # add 1 to counter

j while_label

after_loop:

sh $t2, 0($t1)                        # store the value and the ID
sw $t3, 4($t1)

addi $t6, $t6, 1            # add 1 to the number of nodes

move $v0, $t6                        # return value is the number of nodes

jr $ra

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

numOfItems:

move $t6,$a3
```

```
li $v0, 4            # print the arrat's size
la $a0, number
syscall

li  $v0, 1           # print integer
move $a0, $t6
syscall

jr $ra

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

printItem:

move $t1,$a0         # head

li $t5,1             # counter
li $t8,0             # boolean flag

li $v0,4
la $a0,node          # print message ask node
syscall

li $v0,5             # read id
syscall

move $t4, $v0
```

```
loop_2:

beq $t5,$t4,after_loop_2

addi $t1,$t1,8

addi $t5,$t5,1

j loop_2

after_loop_2:

lw $t2,0($t1)
lw $t3,4($t1)

addi $t8,$t8,1

li $v0,4
la $a0,valueinfo       # print message
syscall

li  $v0, 1           # print integer
move $a0, $t2
syscall

li $v0,4
```

```
la $a0,idinfo         # print message
syscall

li  $v0, 1            # print integer
move $a0, $t3
syscall

bne $t8,$zero,after_if

li $v0,4
la $a0,errornode      # print message
syscall

after_if:
jr $ra


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

printAddressItem:

move $t1,$a0          # head

li $t5,1              # counter
li $t8,0              # boolean flag

li $v0,4
```

```
la $a0,node             # print message ask node
syscall

li $v0,5                # read id
syscall

move $t4, $v0

loop_3:

beq $t5,$t4,after_loop_3

addi $t1,$t1,8
addi $t5,$t5,1

j loop_3

after_loop_3:

addi $t8,$t8,1

li $v0,4
la $a0,valueaddress    # print message
syscall

li  $v0, 1            # print integer
move $a0,$t1
```

```
syscall

addi $t1,$t1,4

li $v0,4
la $a0,idaddress       # print message
syscall

li  $v0, 1           # print integer
move $a0, $t1
syscall

bne $t8,$zero,after_if_3

li $v0,4
la $a0,errornode       # print message
syscall


after_if_3:
jr $ra

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

deleteLast:

move $t1,$a0
```

```
move $t6,$a3

li $t5,0              # help counter

bne $t6, $zero, after_cond_del     # case if array is empty

li  $v0, 4            # print new line
la  $a0, newline
syscall

li $v0, 4             # print string when array is empty
la $a0, emptyarr
syscall

move $v0, $t6

jr $ra

after_cond_del:

loop_4:

beq $t5,$t6,after_loop_4  # the purpose of this loop is to reach to the next of the last item

addi $t1,$t1,8
addi $t5,$t5,1
```

```
j loop_4

after_loop_4:
addi $t6,$t6,-1                    # add -1 to the number of items
move $s6,$t6

lw $t2,-8($t1)
lw $t3,-4($t1)

move $t2,$zero
move $t3,$zero

sw $t2,-8($t1)
sw $t3,-4($t1)

move $v0, $t6

jr $ra

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

printArrayAddress:

move $t1, $a0

li  $v0, 4          # print new line
```

```
la  $a0, newline
syscall

li $v0, 4            # print message
la $a0, arrayaddress
syscall

li  $v0, 1           # print array address
move $a0, $t1
syscall

li  $v0, 4           # print new line
la  $a0, newline
syscall

jr $ra

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

printMinValue:

move $t1, $a0
move $t6, $a3

move $t5,$zero       # help counter

li $t4, 100          # $t4 holds the minimum value
```

```
while_label_min:

bge $t5, $t6, after_loop_min

lw $t2, 0($t1)         # load the value

li  $v0, 4             # print new line
la  $a0, newline
syscall

bge $t2,$t4,after_cond_min

move $t4, $t2          # update minimum value

after_cond_min:

addi $t5, $t5, 1       # add 1 to help counter
addi $t1, $t1, 8       # add 8 to head address

j while_label_min

after_loop_min:

li $v0, 4              # print message
la $a0, minvalue
syscall
```

```
li $v0, 1
move $a0, $t4          # print minimum value
syscall

jr $ra

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

MergeSort:

addui $sp, $sp, -20         # swse ton $s0 sth stack

sw   $a0,0($sp)
sw   $a1,4($sp)        #left
sw   $a2,8($sp)        #num
sw   $ra,16($sp)



move $t5,$a0
move $t1,$a1
move $t2,$a2

li $t4,2
div $t2, $t4
mfhi $t3                                    # $t3 holds mid , [mid = (right+left/2)]
move $a2,$t3
```

```
sw   $a3,12($sp)



bge $t1 , $t2, after_cond_sort



jal MergeSort

lw $t3, 8($sp)



addi $t3,$t3,1

move $a3,$t3

sw $a3,8($sp)

addi $sp, $sp, -4

sw $a2,($sp)

jal MergeSort

lw   $a0,0($sp)

lw   $a1,4($sp)  #left

lw   $a2,8($sp)   #num

lw   $a3,12($sp)

jal mergee
```

```
after_cond_sort:
lw $a0, 0($sp)        # restore $a0 (head of array) from stack
lw $a1, 4($sp)        # restore $a1 (num of elements) from stack
lw $ra, 16($sp)               # restore $ra from stack
lw   $a2,8($sp)   #num
lw   $a3,12($sp)

addui $sp, $sp, 20
jr $ra



################################################################################
##
mergee:

loop0:
la $t8,mergeArray

move $t5,$a0
move $t1,$a1
move $t2,$a2
move $t4,$a3



bgt $t1,$t3,after_loop0
bgt $t4,$t2,after_loop0

la $t5,array              # hold array's head
```

```
la $t6,array                # hold array's head

lw $t9,0($t5)               # load the first element of the array

li $t8 , 8
mult $t3,$t8
mfhi $t7

la $t8,mergeArray

add $t6,$t6,$t7             # proxwraw to head ews to mid+1 stoixeio

lw $t7,0($t6)               # exei to stoixeio mid+1

la $t8,mergeArray           # hold mergeArray's head


bgt $t5,$t6,else_cond       # if(array[lpos] < array[rpos])
addi $t8,$t8,8              # tempArray[pos++] = array[lpos++];
addi $t5,$t5,8
lw $t9,0($t5)
move $t0,$t9
sw $t0,0($t8)

else_cond:
addi $t8,$t8,8              # tempArray[pos++] = array[rpos++];
addi $t6,$t6,8
```

```
lw $t7,0($t6)
move $t0,$t7
sw $t0,0($t8)
j loop0

after_loop0:

loop_11:
addi $t8,$t8,8          # tempArray[pos++] = array[lpos++];
addi $t5,$t5,8
lw $t9,0($t5)
move $t0,$t9
sw $t0,0($t8)
j loop_11

after_loop_11:

loop_22:
bgt $t4,$t2,after_loop_22
addi $t8,$t8,8          # tempArray[pos++] = array[rpos++];
addi $t6,$t6,8
lw $t7,0($t6)
move $t0,$t7
sw $t0,0($t8)
j loop_22

after_loop_22:
```

```
addi $t2,$t2,1              # size
loop_33:


beq $t2,$zero,after_loop_33  # print the temporary array
la $t8,mergeArray           # hold mergeArray's head

lw $t0,0($t8)

li  $v0, 1                  # print integer
move $a0, $t0
syscall

addi $t8,$t8,8              # move mergeArray's head
addi $t2,$t2,-1
j loop_33

after_loop_33:

lw  $a0,0($sp)
lw  $a1,4($sp)
lw  $a2,8($sp)
lw  $a3,12($sp)
lw  $ra,16($sp)
addi $sp, $sp, 20
```

```
jr $ra

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

printall:
move $t1,$a0
move $t6,$a3
li $t7,0

loop_5:
beq $t7,$t6,after_loop_5
lw $t2,0($t1)
lw $t3,4($t1)

li $v0,4
la $a0,valueinfo       # print message
syscall

li  $v0, 1           # print integer
move $a0, $t2
syscall

li $v0,4
```

```
la $a0,idinfo          # print message
syscall

li  $v0, 1             # print integer
move $a0, $t3
syscall

add $t1,$t1,8
addi $t7,$t7,1

j loop_5

after_loop_5:

jr $ra
```