ΗΡΥ 201– Ψηφιακοί Υπολογιστές

Φροντιστήριο για την 4η Εργαστηριακή Άσκηση.

Τι θα καλύψει το φροντιστήριο.

- Ανάλυση κεντρικών ζητημάτων που θα καλείστε να αντιμετωπίσετε:
  - Συμβάσεις καταχωρητών και χρήση τους
  - Πρόσβαση στην μνήμη (ανάγνωση/εγγραφή)
  - Ευθυγράμμιση μνήμης
- Παράδειγμα κλήσης συναρτήσεων σε Assembly
- Παράδειγμα με υπολογισμό μέσου όρου σε Assembly

Ζητούμενα του 4ου Εργαστηρίου (MIPS Assembly)

- Δέσμευση χώρου στην περιοχή .data για 100 χαρακτήρες
  - Τι τύπου χώρο δεσμεύουμε; Πόσος είναι ο πραγματικός χώρος που χρειαζόμαστε;
- Κλήση συναρτήσεων με χρήση jal και jr αφού πρώτα διαβαστούν τα δεδομένα από τον χρήστη
  - Αρα τα δεδομένα που εισάγει ο χρήστης είναι ορίσματα της συνάρτησης
- Συμβάσεις καταχωρητών για την κλήση συναρτήσεων.

Δέσμευση μνήμης

- Η δέσμευση μνήμης γίνεται στο κομμάτι .data του κώδικα Assembly.
- Η μνήμη που θα δεσμεύσουμε είναι για 100 χαρακτήρες.
- Για να δεσμεύσουμε μνήμη χρησιμοποιούμε την οδηγία συμβολομεταφραστή .space x.

Ευθυγράμμιση μνήμης

- Η μνήμη πρέπει να είναι ευθυγραμμισμένη σε τετράδες.
- Μπορούμε να είμαστε όμως σίγουροι που τελειώνει ένα string;
- Βολεύει να μετράμε τους χαρακτήρες ενός string για να δεσμεύσουμε κατάλληλα την μνήμη;
- Η οδηγία .align n μας βοηθάει στα παραπάνω.

Παραδειγμα χρήσης .align

Assembly

```
data
.globl array
.globl HW
.globl Name
.globl Sorry

HW: .asciiz “Hello World!\n"
Name: .ascii “My name is George! \n"
Sorry: .asciiz “Forgot the terminating character… Sorry…"

array: .align 2
                .space 200
```

Συμβάσεις Καταχωρητών (1/2)

- Οι συμβάσεις χρήσης καταχωρητών απαιτούν τα εξής:
  - Οι καταχωρητές $v0 χρησιμοποιούνται για την επιστροφή τιμών από συναρτήσεις (functions)
  - Οι καταχωρητές $a0..$a3 χρησιμοποιούνται για το πέρασμα παραμέτρων σε διαδικασίες και συναρτήσεις (procedures & functions)
  - Ο καταχωρητής $ra (return address) χρησιμοποιείται για την αποθήκευση της διεύθυνσης επιστροφής από υπορουτίνα

Συμβάσεις Καταχωρητών (2/2)

- Οι συμβάσεις χρήσης καταχωρητών απαιτούν τα εξής:
  - Οι καταχωρητές $t0..$t9 ονομάζονται «προσωρινοί» (temporary) και ενδύκνειται για την αποθήκευση προσωρινών τιμών οι οποίες δεν απαιτείται να διατηρηθούν και μετά από μια κλήση διαδικασίας ή συνάρτησης.
  - Οι καταχωρητές $s0..$s7 ονομάζονται saved και ενδείκνυται να χρησιμοποιούνται για αποθήκευση τιμών που διατηρούνται για περισσότερο χρόνο
- Οι καταχωρητές $s0-$s7 βολεύουν για αποθήκευση global μεταβλητών (μετρητής στοιχείων string, διευθύνσεις πινάκων κλπ).

Κλήση και επιστροφή συναρτήσεων

- Η κληση των συναρτήσεων γίνεται με την εντολή **jal**, aka jump and link.
- **Πριν καλέσουμε την συνάρτηση φροντίζουμε τα ορίσματά μας να είναι στους καταχωρητές ορισμάτων.**
- Η επιστροφή των συναρτήσεων γίνεται με την εντολή
  jr $ra, aka jump register.
- Προσοχή να μην πανωγράψουμε τον $ra! Τι θα γίνει αν μέσα σε συνάρτηση καλέσουμε άλλη συνάρτηση;

Πρόσβαση στην μνήμη

- Το κύριο σημείο στην πρόσβαση στην μνήμη είναι να γνωρίζουμε την διεύθυνση της λίστας Α.Α. (Array Address).
- Επίσης πρέπει να γνωρίζουμε ποιον χαρακτήρα του string θέλουμε να βρούμε.
- Πχ αν θέλω να βρώ το value του 4^{ου} χαρακτήρα τότε: address = A.A. + i (όπου i=3).
- Έπειτα lw $t0, 0($s0), όπου στον $s0 περιέχεται η παραπάνω διεύθυνση.

Παραδειγμα πανωγραψίματος του $ra

Assembly		

main:
li $a0, 4
li $a1, 5
jal foo1
addi $s0, $v0, 1
…
 	
foo1:
add $a0, $a0, $a1
jal foo1
addι $s0, $v1, 1
jr $ra
 	
foo2:
add $v1, $a0, $a1
jr $ra
	
	PC = 40
	$ra = PC + 4 = 48
	
	PC = 124
	$ra = PC + 4 = 132
	PC = $ra -> PC = 132
	
	PC = $ra -> PC = 132
		
		Επιστροφή πάντα στο ίδιο σημείο

Παραδειγμα προγράμματος κλήσης συνάρτησης

Assembly

```
.data
.globl variable
.globl Input
.globl mess1
.globl mess2
.globl mess3

Input: .asciiz "Give input:"
mess1: .asciiz "Calling a function.\n"
mess2: .asciiz "Inside function.\n"
mess3: .asciiz "Result returned from function:\n"
variable: .word 1
```

```
.text
.globl main
main:

li $v0, 4
la $a0, mess1
syscall

li $v0, 5
syscall

move $s0, $v0
la $s1, variable
sw $s0, 0($s1)

li $v0, 4
la $a0, mess1
syscall
```

```
move $a0, $s0
jal foo
move $s0, $v0

li $v0, 4
la $a0, mess3
syscall

li $v0, 1
move $a0, $s0
syscall

li $v0, 10
syscall
```

```
foo:
move $t0, $a0

li $v0, 4
la $a0, mess2
syscall

add $t0, $t0, 4
move $v0, $t0

jr $ra
```