Αρχιτεκτονική Υπολογιστών

Εργαστηριακές Ασκήσεις

Οκτώβριος 2013

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων
Πανεπιστήμιο Αιγαίου

Εμμανουήλ Καλλίγερος

Περιεχόμενα

Πρόλογος

Στις σελίδες που ακολουθούν θα βρείτε το εγχειρίδιο του εργαστηρίου του μαθήματος της Αρχιτεκτονικής Υπολογιστών. Στόχος του εργαστηρίου είναι να εξοικειωθείτε με τα βασικά της συγγραφής προγραμμάτων σε γλώσσα assembly, η οποία είναι η γλώσσα προγραμματισμού που βρίσκεται «πλησιέστερα» στο υλικό κάθε επεξεργαστή. Αυτό έχει σαν αποτέλεσμα τα προγράμματα που γράφονται σε assembly να είναι σημαντικά μεγαλύτερα από τα αντίστοιχα προγράμματα μίας γλώσσας υψηλού επιπέδου (π.χ. της C). Από την άλλη, ο προγραμματιστής έχει «άμεσο έλεγχο» του υλικού του επεξεργαστή, πράγμα που σημαίνει ότι μικρά και «κρίσιμα» κομμάτια κώδικα μπορούν να γραφτούν αποδοτικότερα σε assembly, ενώ ταυτόχρονα ο ίδιος (ο προγραμματιστής) αποκτά πολύ καλή γνώση του υλικού. Επειδή ο κάθε επεξεργαστής έχει διαφορετική assembly, για το εργαστήριο επιλέχθηκε η assembly του MIPS (Microprocessor without Interlocked Pipeline Stages). Ο συγκεκριμένος επεξεργαστής, λόγω της RISC (Reduced Instruction Set Computer) αρχιτεκτονικής του, έχει ένα πολύ απλό σύνολο εντολών, το οποίο θα σας επιτρέψει να ασχοληθείτε πολύ περισσότερο με τις τεχνικές προγραμματισμού σε assembly, παρά με το πώς θα χρησιμοποιήσετε στα προγράμματά σας τις διάφορες εντολές του. Για την εκτέλεση προγραμμάτων γραμμένων σε assembly του MIPS θα χρησιμοποιήσετε τον εξομοιωτή QtSpim, τον οποίο μπορείτε να κατεβάσετε ελεύθερα από το Internet. Περισσότερες λεπτομέρειες σχετικά με την assembly του MIPS μπορείτε να βρείτε στο 1ο Κεφάλαιο του εγχειριδίου αυτού, ενώ σχετικά με τον QtSpim μπορείτε να διαβάσετε στο 2ο Κεφάλαιο. Στο τέλος θα βρείτε τις ασκήσεις που πρέπει να υλοποιήσετε.

Εισαγωγή στην αρχιτεκτονική και στη γλώσσα assembly του MIPS

Στο κεφάλαιο αυτό θα βρείτε μία συνοπτική περιγραφή ορισμένων εντολών και τρόπων διευθυνσιοδότησης, οι οποίοι χρησιμοποιούνται συχνά κατά τη συγγραφή προγραμμάτων στη γλώσσα assembly του επεξεργαστή MIPS. Συγκεκριμένα, οι εντολές που θα περιγραφούν ανήκουν στην αρχιτεκτονική MIPS R2000, η οποία είναι υποσύνολο της αρχιτεκτονικής MIPS32. Αρχικά συζητούνται κάποια βασικά σημεία της αρχιτεκτονικής του MIPS, των οποίων η γνώση είναι απαραίτητη για τη συγγραφή προγραμμάτων assembly. Στη συνέχεια εξηγείται η δομή ενός έγκυρου προγράμματος assembly του MIPS, και τέλος περιγράφονται «στοχευμένα» συγκεκριμένες εντολές και τρόποι διευθυνσιοδότησης. Κατά αυτόν τον τρόπο καλύπτεται το μεγαλύτερο μέρος των «προγραμματιστικών αναγκών» των ασκήσεων του εργαστηρίου Αρχιτεκτονικής Υπολογιστών.

Χρειάζεται να διαβάσετε κάτι άλλο;

Το κεφάλαιο αυτό αποτελεί μία πολύ συνοπτική εισαγωγή στην αρχιτεκτονική και assembly του MIPS. Είναι λοιπόν ***απαραίτητο*** να ανατρέχετε και σε κάποιο πιο αναλυτικό εγχειρίδιο. Ένα πολύ καλό κείμενο αναφοράς αποτελεί το Παράρτημα Α του βιβλίου *Computer Organization and Design: The Hardware/Software Interface*, 3rd Ed., με τίτλο: ***Assemblers, Linkers and the SPIM Simulator***. Το παράρτημα αυτό μπορείτε να το βρείτε σε ηλεκτρονική μορφή στην ακόλουθη διεύθυνση:
http://pages.cs.wisc.edu/~larus/HP_AppA.pdf

Από το συγκεκριμένο παράρτημα θα πρέπει να διαβάσετε τις παραγράφους Α.1 – Α.5 και Α.9, ενώ στην παράγραφο Α.10 υπάρχει ομαδοποιημένη ολόκληρη η assembly της αρχιτεκτονικής MIPS R2000. Την παράγραφο αυτή θα πρέπει να συμβουλεύεστε κάθε φορά που έχετε απορία σχετικά με τη χρήση κάποιας εντολής, ενώ στις ασκήσεις μπορείτε, αν σας φανεί χρήσιμο, να χρησιμοποιήσετε οποιαδήποτε από τις εντολές της αρχιτεκτονικής MIPS R2000.

Αρχιτεκτονική MIPS R2000

Η βασική δομή της αρχιτεκτονικής MIPS R2000, η οποία εξομοιώνεται από το εργαλείο που χρησιμοποιείται στο εργαστήριο (*QtSpim* – βλ. επόμενο κεφάλαιο), φαίνεται στο Σχήμα 1.1. Αποτελείται από τη CPU (Central Processing Unit) και δύο συνεπεξεργαστές (Coprocessor 0 και 1).

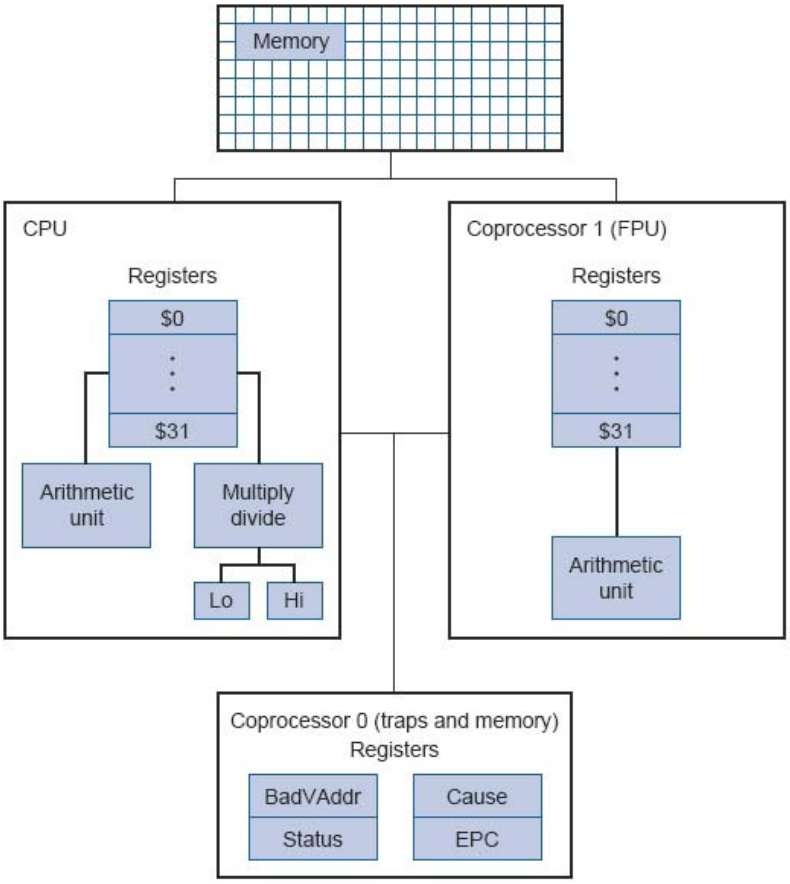
- Η CPU[1] περιλαμβάνει 32 καταχωρητές γενικού σκοπού, την αριθμητική-λογική μονάδα (ονομάζεται απλά Arithmetic unit στον MIPS) για την εκτέλεση ακέραιας πρόσθεσης και αφαίρεσης αλλά και λογικών πράξεων, καθώς και τη μονάδα που εκτελεί πολλαπλασιασμό και διαίρεση ακεραίων (Multiply / divide). Όσο αφορά τους καταχωρητές, με τον όρο «γενικού σκοπού» εννοούμε ότι τους καταχωρητές αυτούς μπορεί να τους χρησιμοποιήσει ο προγραμματιστής για όποιο σκοπό επιθυμεί στα προγράμματά του. Η τιμή τους μπορεί να μεταβληθεί μέσω εντολών assembly (πλην μίας εξαίρεσης στον MIPS), κάτι το οποίο δεν ισχύει για όλους τους καταχωρητές του MIPS, αλλά και των υπόλοιπων επεξεργαστών γενικότερα.
- Ο Coprocessor 1 είναι ουσιαστικά η μονάδα εκτέλεσης πράξεων κινητής υποδιαστολής (Floating Point Unit – FPU) και περιλαμβάνει 32 καταχωρητές για την αποθήκευση

[1] Η μονάδα που στον MIPS ονομάζεται CPU, καλείται συνήθως "Integer Unit" ή "Fixed Point Unit" (μονάδα επεξεργασίας ακεραίων ή μονάδα σταθερής υποδιαστολής) στην ορολογία της Αρχιτεκτονικής Υπολογιστών. Με τον όρο CPU (κεντρική μονάδα επεξεργασίας) εννοούμε συνήθως ολόκληρο τον επεξεργαστή ή το τμήμα του επεξεργαστή στο οποίο γίνεται η επεξεργασία όλων των ειδών των δεδομένων (ακεραίων και κινητής υποδιαστολής), συμπεριλαμβανομένης και της μονάδας ελέγχου.

τέτοιων αριθμών (κινητής υποδιαστολής). *Προσοχή: οι καταχωρητές αυτοί είναι διαφορετικοί από τους 32 καταχωρητές γενικού σκοπού της CPU*.

- Ο Coprocessor 0 διαχειρίζεται τις διακοπές (interrupts) και τις εξαιρέσεις (exceptions) του επεξεργαστή.
- Η μνήμη που φαίνεται στο Σχήμα 1.1 βρίσκεται εξωτερικά του MIPS και εκεί αποθηκεύονται τα προς εκτέλεση προγράμματα και τα δεδομένα τους (η μνήμη εξομοιώνεται και αυτή από τον QtSpim).

Σημειώνεται ότι η μεγάλη πλειοψηφία των εργαστηριακών ασκήσεων απαιτεί τη χρήση μόνο της μονάδας επεξεργασίας ακεραίων (CPU) του MIPS. Θα πρέπει επίσης να επισημανθεί ότι ένας επεξεργαστής MIPS υλοποιημένος σε υλικό (chip) μπορεί να περιλαμβάνει περισσότερους συνεπεξεργαστές από τους δύο που εξομοιώνει ο QtSpim.



Σχήμα 1.1. MIPS R2000 CPU, FPU και Coprocessor 0

Καταχωρητές του MIPS

- Όπως προαναφέρθηκε, ο MIPS περιλαμβάνει 32 καταχωρητές γενικού σκοπού για την αποθήκευση ακεραίων και επιπλέον 32 καταχωρητές για την αποθήκευση αριθμών κινητής υποδιαστολής. ***Όλοι οι καταχωρητές έχουν εύρος 32 bit***.
- Κατά τη συγγραφή προγραμμάτων assembly, μπροστά από το νούμερο ή το όνομα των καταχωρητών μπαίνει ο χαρακτήρας $. Άρα, υπάρχουν δύο τρόποι για να αναφερθείτε σε έναν καταχωρητή, μέσα σε μία εντολή:
  - είτε χρησιμοποιώντας τον αριθμό του, δηλ. $0 έως $31 *(ο τρόπος αυτός χρησιμοποιείται **μόνο** για τους καταχωρητές γενικού σκοπού)*,
  - είτε χρησιμοποιώντας το όνομά του, π.χ. $t1, $sp, κ.τ.λ.

  Συνήθως προτιμάται ο δεύτερος τρόπος.

- Οι καταχωρητές Lo και Hi (βλ. Σχήμα 1.1) χρησιμοποιούνται για την αποθήκευση των αποτελεσμάτων των πράξεων πολλαπλασιασμού και διαίρεσης. Τα περιεχόμενά τους δεν μπορούν να προσπελαστούν με χρήση των εντολών που χρησιμοποιούνται για τους καταχωρητές γενικού σκοπού, αλλά απαιτείται η χρήση ειδικών εντολών (εντολή mfhi – "move from Hi" και εντολή mflo – "move from Lo").

Πίνακας 1.1. Καταχωρητές γενικού σκοπού και κινητής υποδιαστολής του MIPS

Αριθμός	Όνομα	Χρήση
0	$zero	Περιέχει πάντα το 0, *δεν μπορεί να αλλάξει η τιμή του*
1	$at	*(assembler temporary)* Δεσμευμένος από τον assembler, δεν επιτρέπεται να χρησιμοποιείται από τον προγραμματιστή
2, 3	$v0, $v1	*(values)* Για επιστροφή τιμών από κλήση συναρτήσεων
4 – 7	$a0 – $a3	*(arguments)* Για πέρασμα τιμών σε συναρτήσεις
8 – 15	$t0 – $t7	*(temporaries)* Προσωρινοί, επιτρέπεται να μεταβληθεί η τιμή τους από συναρτήσεις
16 – 23	$s0 – $s7	*(saved values)* Προσωρινοί, δεν επιτρέπεται να μεταβληθεί η τιμή τους από συναρτήσεις – μία συνάρτηση, για να τους χρησιμοποιήσει, πρέπει να αποθηκεύσει αρχικά τις τιμές τους και να τις επαναφέρει πριν την ολοκλήρωση της εκτέλεσής της
24, 25	$t8, $t9	*(temporaries)* Προσωρινοί, επιτρέπεται να μεταβάλλονται από τις συναρτήσεις (σαν τους $t0 – $t7 παραπάνω)
26, 27	$k0, $k1	*(kernel)* Δεσμευμένοι για χρήση από το λειτουργικό σύστημα
28	$gp	*(global pointer)* Δείκτης στο μέσον της, μεγέθους 64Kbytes, static data περιοχής (segment) στη μνήμη
29	$sp	*(stack pointer)* Δείχνει στην τελευταία θέση της στοίβας (stack) του MIPS που χρησιμοποιήθηκε
30	$fp	*(frame pointer)* Δείχνει στην πρώτη θέση ενός frame που (συνήθως) δημιουργείται στη στοίβα κατά την κλήση μίας συνάρτησης
31	$ra	*(return address)* Αποθηκεύει τη διεύθυνση επιστροφής κατά την κλήση συναρτήσεων
	$f0 – $f31	*(floating point)* Για αποθήκευση αριθμών κινητής υποδιαστολής (βρίσκονται στον Coprocessor 1)

Στις ασκήσεις του εργαστηρίου θα χρησιμοποιήσετε κυρίως τους καταχωρητές $t και $s (τηρώντας μόνο κάποιους πολύ απλούς κανόνες στις ασκήσεις που θα σας ζητηθεί υλοποίηση συναρτήσεων), καθώς και τους $v και $a για την πραγματοποίηση κλήσεων συστήματος (system calls) του εξομοιωτή QtSpim.

Τύποι δεδομένων και λεκτικά σύμβολα της assembly του MIPS

Τύποι δεδομένων:

- Όλες οι εντολές καταλαμβάνουν 32 bits (4 bytes) στη μνήμη.
- Τύποι δεδομένων που μπορούν να διαχειριστούν οι εντολές της αρχιτεκτονικής MIPS R2000: byte (χαρακτήρας - 8 bits), half (μισή λέξη - 2 bytes), word (λέξη - 4 bytes). Επιπλέον αυτών, υπάρχουν και δύο τύποι για αριθμούς κινητής υποδιαστολής.
- Ένας χαρακτήρας απαιτεί, όπως δηλώνει και το όνομά του, 1 byte για την αποθήκευσή του.
- Ένας ακέραιος απαιτεί μία λέξη (word = 4 bytes) για την αποθήκευσή του.
- Ένας καταχωρητής γενικού σκοπού «χωράει» να αποθηκεύσει ακριβώς έναν ακέραιο.

Λεκτικά σύμβολα:

- Οι αριθμοί εισάγονται ως έχουν, στο δεκαδικό, π.χ. 12.
- Οι χαρακτήρες τοποθετούνται μεταξύ απλών εισαγωγικών, π.χ. 'b'.
- Οι συμβολοσειρές (string) τοποθετούνται μεταξύ διπλών εισαγωγικών, π.χ. "A string".

Δομή ενός προγράμματος assembly του MIPS

- Το πρόγραμμα αποθηκεύεται σε μορφή απλού κειμένου και περιλαμβάνει τις δηλώσεις των μεταβλητών και τον κώδικα. Το αρχείο απλού κειμένου με το πρόγραμμα πρέπει να σωθεί με επέκταση .s για να μπορεί να χρησιμοποιηθεί άμεσα από τον QtSpim.
- Σε κάθε πρόγραμμα οι δηλώσεις των μεταβλητών προηγούνται του κώδικα του προγράμματος

Δηλώσεις μεταβλητών

- Οι δηλώσεις των μεταβλητών τοποθετούνται μετά από την directive του assembler **.data**. Directive (οδηγία ή ντιρεκτίβα) ονομάζεται μία δήλωση σε ένα πρόγραμμα, η οποία *δεν μεταφράζεται η ίδια σε εκτελέσιμο κώδικα*, αλλά καθοδηγεί τον assembler ώστε να μεταφράσει σωστά το πρόγραμμα που ακολουθεί. Δηλαδή, γράφοντας .data σε κάποιο σημείο του προγράμματός μας, «λέμε» στον assembler ότι στον κώδικα που ακολουθεί γίνονται δηλώσεις μεταβλητών. Στην assembly MIPS, όλες οι directives ξεκινούν με τελεία.
- Στο τμήμα αυτό του προγράμματος δηλώνονται τα ονόματα των μεταβλητών που θα χρησιμοποιηθούν στο πρόγραμμα. Για κάθε μεταβλητή που δηλώνεται δεσμεύεται ο αντίστοιχος χώρος στην κύρια μνήμη (RAM).

Κώδικας του προγράμματος

- Τοποθετείται μετά από την directive του assembler **.text** και περιλαμβάνει τις εντολές του προγράμματος.
- Η εκτέλεση ξεκινάει πάντα από το label **main:** , το οποίο πρέπει οπωσδήποτε να χρησιμοποιείται σε οποιοδήποτε πρόγραμμα γράφεται σε assembly του MIPS για να τρέξει στον QtSpim.
- Στο τέλος κάθε προγράμματος πρέπει να χρησιμοποιείται η κλήση συστήματος exit (δείτε την παράγραφο *Κλήσεις συστήματος και Είσοδος/Έξοδος (εξομοιωτής QtSpim)* παρακάτω).

Σχόλια

- Σαν σχόλιο θεωρείται οτιδήποτε μετά το χαρακτήρα #, π.χ.
  # This is a comment.

Πρότυπο δομής προγράμματος

```
# Comment giving name of program and description of function
# Template.s
# Bare-bones outline of MIPS assembly language program

    .data       # Variable declarations follow this line
                # ...

    .text       # Instructions follow this line
main:           # Indicates start of code (first instruction to execute)
                # ...

# End of program
```

Δηλώσεις μεταβλητών

Η μορφή των δηλώσεων μεταβλητών είναι η εξής:

```
όνομα μεταβλητής:     .τύπος δεδομένων      τιμή
```

- Μία τέτοια δήλωση δεσμεύει τον κατάλληλο χώρο στη μνήμη για μία μεταβλητή του τύπου δεδομένων που ορίζει ο προγραμματιστής, και στο χώρο αυτό αποθηκεύει την τιμή που περιλαμβάνεται στη δήλωση
- Το πεδίο `τιμή` προσδιορίζει συνήθως την τιμή στην οποία αρχικοποιείται η συγκεκριμένη μεταβλητή. Στις δηλώσεις πινάκων, το πεδίο αυτό ορίζει και το μέγεθος του πίνακα (δείτε το παράδειγμα του πίνακα array2 παρακάτω).

Προσέξτε ότι σε μία δήλωση, το όνομα της μεταβλητής θα πρέπει πάντα να ακολουθείται από άνω-κάτω τελεία (:), ενώ και ο τύπος δεδομένων θα πρέπει να έχει μπροστά του μία τελεία (πρόκειται ουσιαστικά για μία directive προς τον assembler). Στη συνέχεια, θα ασχοληθούμε με μεταβλητές τύπου .word και .byte, οι οποίες χρησιμοποιούνται συχνότερα στα διάφορα προγράμματα.

Παραδείγματα:

```
var1:     .word    3                  # Creates a single integer variable with initial value 3
array1:   .byte    'a', 'b'           # Creates a 2-element character array with elements
                                      # initialized to a and b
array2:   .word    0:10               # Allocates an array of 10 integers, all initialized to 0
str:      .asciiz  "My first string \n" # Creates a string of ASCII characters (it allocates the
                                      # necessary space and initializes the characters
                                      # according to the given string). The .asciiz directive
                                      # terminates each string with the null ('\0') character
                                      # so as to be ready to use with QtSpim's print_string
                                      # system call
```

Εντολές φόρτωσης/αποθήκευσης (load/store)

- Η κύρια μνήμη του MIPS έχει οργάνωση 1 byte/θέση μνήμης. Άρα μία ακέραια μεταβλητή (.word) αποθηκεύεται σε 4 συνεχόμενες θέσεις μνήμης, ενώ μία μεταβλητή τύπου χαρακτήρα (.byte) χρειάζεται μία θέση μνήμης για την αποθήκευσή της.
- Μόνο οι εντολές φόρτωσης (load) και αποθήκευσης (store) μπορούν να προσπελάσουν την κύρια μνήμη για διάβασμα ή γράψιμο, αντίστοιχα.
- Όλες οι υπόλοιπές εντολές επιδρούν σε καταχωρητές.

<u>Βασικές εντολές φόρτωσης καταχωρητή από τη μνήμη:</u>

- `lw register_destination, word_var_in_RAM`
  Εντολή *Load Word (lw):* φορτώνει τον καταχωρητή `register_destination` με την τιμή μίας ακέραιας μεταβλητής που είναι αποθηκευμένη στην κύρια μνήμη `(word_var_in_RAM)`.
- `lbu register_destination, byte_var_in_RAM`
  Εντολή *Load Byte Unsigned (lbu):* φορτώνει το λιγότερο σημαντικό byte του καταχωρητή `register_destination` με την τιμή μίας μεταβλητής χαρακτήρα που είναι αποθηκευμένη στην κύρια μνήμη `(byte_var_in_RAM)`. Τα 24 πιο σημαντικά bits του καταχωρητή μηδενίζονται κατά την εκτέλεση της συγκεκριμένης εντολής.

Βασικές εντολές αποθήκευσης καταχωρητή στη μνήμη:

- `sw register_source, word_var_in_RAM`
  Εντολή *Store Word (sw):* αποθηκεύει την τιμή του καταχωρητή `register_source` στη θέση μίας ακέραιας μεταβλητής στην κύρια μνήμη `(word_var_in_RAM)`.
- `sb register_source, byte_var_in_RAM`
  Εντολή *Store Byte (sb):* αποθηκεύει το λιγότερο σημαντικό byte του καταχωρητή `register_source` στη θέση μίας μεταβλητής χαρακτήρα στην κύρια μνήμη `(byte_var_in_RAM)`.

Άμεση φόρτωση καταχωρητή:

- `li register_destination, value`
  Εντολή *Load Immediate (li):* φορτώνει την τιμή `value` στον καταχωρητή `register_destination`.

Αντιγραφή τιμής καταχωρητή:

- `move register_destination, register_source`
  Εντολή *Move (move):* αντιγράφει την τιμή του καταχωρητή `register_source` στον καταχωρητή `register_destination`.

Φόρτωση καταχωρητή με διεύθυνση μεταβλητής:

- `la register_destination, variable`
  Εντολή *Load Address (la):* φορτώνει τον καταχωρητή `register_destination` με τη διεύθυνση κύριας μνήμης της μεταβλητής `variable`. Η εντολή αυτή είναι χρήσιμη για την εκτύπωση μηνυμάτων στην κονσόλα του QtSpim (δείτε την παράγραφο *Κλήσεις συστήματος και Είσοδος/Έξοδος (εξομοιωτής QtSpim)* παρακάτω).

Σημειώνεται ότι οι *li*, *move* και *la* δεν είναι εντολές που υλοποιούνται από το υλικό του MIPS (δεν ανήκουν δηλαδή στο σύνολο εντολών του), αλλά προσφέρονται από τον assembler του επεξεργαστή. Τέτοιες εντολές ονομάζονται *ψευδοεντολές* (*pseudoinstructions*), ο προγραμματιστής μπορεί να τις χρησιμοποιεί κανονικά και ο assembler αναλαμβάνει, κατά τη μεταγλώττιση του προγράμματος, να τις «μεταφράσει» σε (περισσότερες από μία) εντολές του συνόλου εντολών του MIPS.

Παράδειγμα:

```
    .data
var1: .word  23          # Declare storage for var1; initial value is 23
str1:  .asciiz "Hello\n" # Declare storage for str1 and initialize string. Due to the use of .asciiz
                         # str1 is null-terminated ('\0'  is inserted by .asciiz as its last character)

    .text
main:
    lw $t0, var1         # Load register $t0 with the value of var1 (stored in RAM):  $t0 = var1
    li $t1, 5            # $t1 = 5 (load value 5 into register $t1)
    sw $t1, var1         # Store the value of $t1 into (RAM location of) var1:  var1 = $t1
    move $t2, $t0        # $t2 = $t0 ($t0 has the old value of var1, i.e. 23)
    la $t0, str1         # Copy RAM address of str1 into register $t0
    . . .
```

Τρόποι διευθυνσιοδότησης της κύριας μνήμης

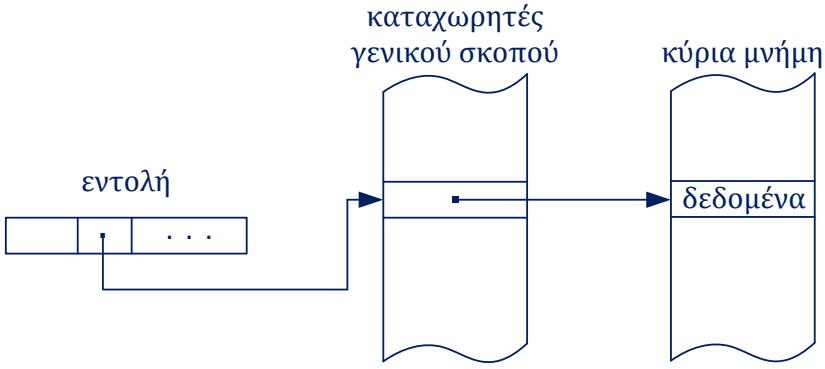
- Τρόποι διευθυνσιοδότησης ονομάζονται οι διαφορετικοί τρόποι με τους οποίους μπορεί κανείς να περιγράψει τη διεύθυνση μίας μεταβλητής στην κύρια μνήμη.
- Οι τρόποι διευθυνσιοδότησης που θα περιγραφούν στην παράγραφο αυτή *(έμμεση αναφορά στη μνήμη με χρήση καταχωρητή, αναφορά με χρήση καταχωρητή βάσης, αναφορά με χρήση βάσης και καταχωρητή δείκτη)* δεν είναι οι μόνοι που ενσωματώνονται στα σύνολα εντολών των μικροεπεξεργαστών, είναι όμως από τους πιο χρήσιμους. Ήδη, από την προηγούμενη παράγραφο, έχουν συζητηθεί κάποιοι επιπλέον τρόποι όπως ο άμεσος (εντολή *li*) ή ο τρόπος με κατ' ευθείαν αναφορά στους εσωτερικούς καταχωρητές (π.χ., εντολή *move*).
- Στην assembly του MIPS, οι διάφοροι διαθέσιμοι τρόποι διευθυνσιοδότησης χρησιμοποιούνται μόνο με τις εντολές φόρτωσης/αποθήκευσης (load/store), καθώς αυτές είναι οι μόνες που προσπελαύνουν την κύρια μνήμη.

Έμμεση αναφορά στη μνήμη με χρήση καταχωρητή (register indirect addressing):

- `lw $t2, ($t0)`

  Ο $t2 φορτώνεται με την τιμή της ακέραιας μεταβλητής, η οποία βρίσκεται στη διεύθυνση κύριας μνήμης $t0 (και στις επόμενες τρεις). Δηλαδή, το περιεχόμενο του $t0 χρησιμοποιείται σαν διεύθυνση για τη φόρτωση του $t2 από τη μνήμη.
- `sw $t2, ($t0)`

  Η τιμή του $t2 αποθηκεύεται στη θέση της ακέραιας μεταβλητής, η οποία βρίσκεται στη διεύθυνση κύριας μνήμης $t0 (και στις επόμενες τρεις). Όπως και στο προηγούμενο παράδειγμα, το περιεχόμενο του $t0 χρησιμοποιείται σαν διεύθυνση για την αποθήκευση του $t2 στη μνήμη.



Σχήμα 1.2. Προσπέλαση κύριας μνήμης με έμμεση αναφορά με χρήση καταχωρητή

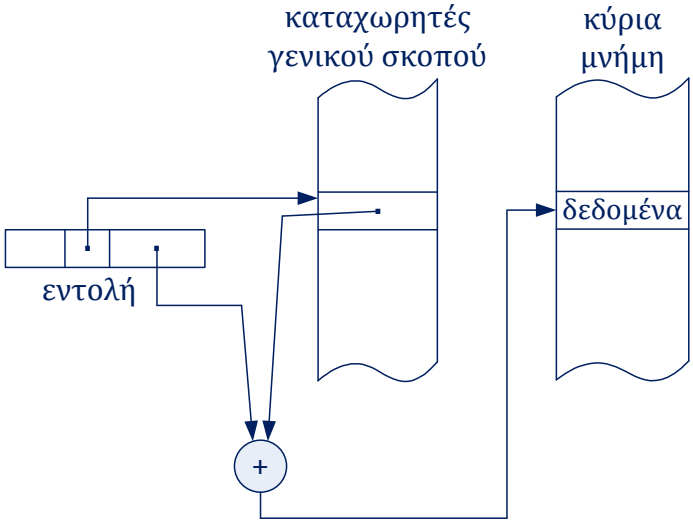
Αναφορά στη μνήμη με χρήση καταχωρητή βάσης (base addressing):

- `lw $t2, 8($t0)`

  Ο $t2 φορτώνεται με την τιμή της ακέραιας μεταβλητής, η οποία βρίσκεται στη διεύθυνση κύριας μνήμης $t0 + 8 (και στις επόμενες τρεις). Σε αυτή την περίπτωση, η διεύθυνση για τη φόρτωση του $t2 από τη μνήμη υπολογίζεται σαν το άθροισμα του περιεχομένου του $t0 με το 8. Το '8' ονομάζεται *μετατόπιση (offset)* από τη διεύθυνση που περιέχεται στον $t0 (ο $t0 είναι, στο παράδειγμα αυτό, ο καταχωρητής βάσης). *Σαν μετατόπιση μπορεί να χρησιμοποιηθεί οποιοσδήποτε αριθμός.*
- `sw $t2, -12($t0)`

  Η τιμή του $t2 αποθηκεύεται στη θέση της ακέραιας μεταβλητής, η οποία βρίσκεται στη διεύθυνση κύριας μνήμης $t0 – 12 (και στις επόμενες τρεις). Όπως και προηγουμένως, η διεύθυνση για την αποθήκευση του $t2 στη μνήμη προκύπτει από το άθροισμα του

περιεχομένου του $t0 με το –12 (δηλ., εκτελείται η πράξη $t0 + (– 12) = $t0 – 12). Είναι προφανές ότι και αρνητικοί αριθμοί μπορούν να χρησιμοποιηθούν σαν μετατοπίσεις.



Σχήμα 1.3. Προσπέλαση κύριας μνήμης με χρήση καταχωρητή βάσης

Αναφορά στη μνήμη με χρήση βάσης και καταχωρητή δείκτη (base-index addressing):

- `lw $t2, array1 + 8($t0)`
  Ο $t2 φορτώνεται με την τιμή της ακέραιας μεταβλητής, η οποία βρίσκεται στη διεύθυνση κύριας μνήμης array1 + ($t0 + 8) [και στις επόμενες τρεις]. Το array1 είναι μια μεταβλητή πίνακα ακεραίων, ο $t0 χρησιμοποιείται στο συγκεκριμένο παράδειγμα σαν καταχωρητής δείκτη και το '8' είναι η μετατόπιση. Η διεύθυνση για τη φόρτωση του $t2 από τη μνήμη υπολογίζεται σαν το άθροισμα της διεύθυνσης του array1 (δηλαδή, της διεύθυνσης του ***πρώτου στοιχείου του***) με το περιεχόμενο του $t0 και το 8.
- `sw $t2, array1 - 12($t0)`
  Η τιμή του $t2 αποθηκεύεται στη θέση της ακέραιας μεταβλητής, η οποία βρίσκεται στη διεύθυνση κύριας μνήμης array1 – ($t0 + 12) [και στις επόμενες τρεις]. Παρατηρήστε πως, σε αυτή την περίπτωση, το "– 12($t0)" σημαίνει ότι το **άθροισμα** του περιεχομένου του $t0 και του 12 **αφαιρείται** από τη διεύθυνση του array1.

Παρατηρήσεις:

- Οι διάφοροι τρόποι διευθυνσιοδότησης στις εντολές φόρτωσης/αποθήκευσης του MIPS μπορούν να χρησιμοποιηθούν με οποιουσδήποτε καταχωρητές γενικού σκοπού επιθυμείτε. Στα παραπάνω παραδείγματα, χρησιμοποιήθηκαν οι καταχωρητές $t0 και $t2, ώστε να γίνει, με πρακτικό τρόπο, κατανοητή η χρήση των τρόπων διευθυνσιοδότησης που παρουσιάστηκαν.
- Στους τρόπους διευθυνσιοδότησης με χρήση καταχωρήτη βάσης ή με χρήση βάσης και καταχωρητή δείκτη, '+' ή '–' μπροστά από τη μετατόπιση μπορεί να χρησιμοποιηθεί ***τόσο στις εντολές φόρτωσης, όσο και στις εντολές αποθήκευσης***.
- Όλοι οι τρόποι διευθυνσιοδότησης που αναφέρθηκαν σε αυτή την παράγραφο μπορούν να χρησιμοποιηθούν και στις εντολές *lbu* και *sb*, αλλά και σε όλες τις υπόλοιπες εντολές φόρτωσης/αποθήκευσης του MIPS.
- Οι συγκεκριμένοι τρόποι διευθυνσιοδότησης είναι ιδιαιτέρως χρήσιμοι για την προσπέλαση στοιχείων σε πίνακες, όπως θα φανεί από το παράδειγμα που ακολουθεί.

Παράδειγμα:

```
    .data
array1: .word  0:3          # Declare an array of 3 integers, all initialized to 0

    .text
main:
    la $t0, array1          # Load address of array (i.e., of its first element) into register $t0
    li $t1, 5               # $t1 = 5 (this is an example of immediate addressing)
    sw $t1, ($t0)           # First array element set to 5; register indirect addressing
    li $t1, 13              # $t1 = 13
    sw $t1, 4($t0)          # Second array element set to 13; base addressing
    li $t1, –7              # $t1 = –7
    li $t2, 8               # $t2 = 8
    sw $t1, array1 + 0($t2) # Third array element set to –7; base-index addressing
    . . .
```

Σημειώνεται ότι στην περίπτωση που ένα loop προσπελαύνει τα στοιχεία ενός πίνακα (κάτι που συμβαίνει στη μεγάλη πλειοψηφία των περιπτώσεων), προτιμάται συνήθως η διευθυνσιοδότηση με χρήση βάσης και καταχωρητή δείκτη, όπου ο καταχωρητής δείκτη αυξάνεται ή μειώνεται κατάλληλα ***(κατά 4 αν πρόκειται για πίνακα ακεραίων, κατά 1 αν πρόκειται για πίνακα χαρακτήρων, δηλ. string)***, ώστε να προσπελαύνεται το επόμενο στοιχείο του πίνακα σε κάθε επανάληψη του loop.

Αριθμητικές πράξεις

- Οι περισσότερες από τις εντολές αριθμητικών πράξεων συντάσσονται με τρία ορίσματα (το αποτέλεσμα και τα δύο τελούμενα της πράξης).
- Συνήθως, τα ορίσματα των εντολών αυτών είναι **μόνο** καταχωρητές. Σε μερικές εντολές μπορούν να χρησιμοποιηθούν και άμεσα τελούμενα (απευθείας δηλαδή κάποιοι αριθμοί που συμμετέχουν στην πράξη). Δεν είναι όμως δυνατή η χρήση μεταβλητών αποθηκευμένων στην κύρια μνήμη, σε εντολές αριθμητικών πράξεων.
- Οι πράξεις που εκτελούνται έχουν εύρος 32 bit (όσο δηλαδή και αυτό των καταχωρητών).

Εντολές πρόσθεσης (με παραδείγματα):

- `add $t0, $t1, $t2      # $t0 = $t1 + $t2`
- `addi $t2, $t3, 5       # $t2 = $t3 + 5`

  Add Immediate (addi): Προσθέτει το άμεσο τελούμενο (5) στον καταχωρητή $t3 και αποθηκεύει το αποτέλεσμα στον $t2. Προσέξτε ότι δεν υπάρχει αντίστοιχη εντολή αφαίρεσης. Παρόλα αυτά το άμεσο τελούμενο μπορεί να είναι αρνητικό, οπότε τελικά αφαιρείται από το περιεχόμενο του καταχωρητή (π.χ. η εντολή `addi $t9, $t5, –6` εκτελεί την πράξη $t9 = $t5 – 6).

Εντολή αφαίρεσης (με παράδειγμα):

- `sub $t2, $t3, $t4      # $t2 = $t3 – $t4`

Εντολές πολλαπλασιασμού (με παραδείγματα):

- `mul $t6, $t5, $t4      # $t6 = $t5 * $t4`

Multiply (mul): Πολλαπλασιάζει το περιεχόμενο των καταχωρητών $t5 και $t4 και αποθηκεύει το αποτέλεσμα στον $t6. Όμως, ο πολλαπλασιασμός δύο αριθμών των 32 bit δίνει αποτέλεσμα εύρους 64 bit. Η εντολή *mul* αποθηκεύει στο πρώτο από τα ορίσματά της (στον $t6 στο παράδειγμα) τα 32 λιγότερο σημαντικά ψηφία της πράξης που εκτελεί. Αυτό επαρκεί για τους πολλαπλασιασμούς που πρέπει να εκτελεστούν στα περισσότερα προγράμματα. Αν παρόλα αυτά χρειαστεί να πολλαπλασιάσετε πολύ μεγάλους αριθμούς και το αποτέλεσμα απαιτεί περισσότερα από 32 bit για την αναπαράσταση του, τότε θα πρέπει να χρησιμοποιήσετε την εντολή που ακολουθεί.

- `mult $t5, $t4 # (Hi, Lo) = $t5 * $t4`

  Η εντολή *mult* πολλαπλασιάζει τα περιεχόμενα των **δύο** καταχωρητών που δέχεται σαν ορίσματα και αποθηκεύει το αποτέλεσμα στους καταχωρητές Hi και Lo του MIPS (βλ. Σχήμα 1.1). Συγκεκριμένα, τα 32 λιγότερο σημαντικά bit του αποτελέσματος αποθηκεύονται στον καταχωρητή Lo, ενώ τα 32 περισσότερο σημαντικά στον καταχωρητή Hi. Τα περιεχόμενα των καταχωρητών αυτών μπορούν στη συνέχεια να μεταφερθούν στους καταχωρητές γενικού σκοπού του MIPS για περαιτέρω επεξεργασία χρησιμοποιώντας τις εντολές *mflo* και *mfhi* (δείτε παρακάτω).

Εντολές διαίρεσης (με παραδείγματα):

- `div $t9, $t8, $t7 # $t9 = $t8 / $t7`

  Divide (div): Διαιρεί το περιεχόμενο του καταχωρητή $t8 με το περιεχόμενο του $t7 και αποθηκεύει το ***ακέραιο πηλίκο*** στον $t9. Όταν η εντολή *div* συντάσσεται με τρία ορίσματα τότε πρόκειται για ψευδοεντολή. Αν θέλετε να χρησιμοποιήσετε στο πρόγραμμά σας και το *υπόλοιπο* της διαίρεσης, τότε μπορείτε να συντάξετε την εντολή *div* με δύο ορίσματα, όπως φαίνεται παρακάτω:

- `div $t8, $t7 # Lo = $t8 / $t7 (integer quotient)`
  `# Hi = $t8 mod $t7 (remainder)`

  Όταν η *div* ακολουθείται από δύο ορίσματα τότε εκτελείται η εντολή που περιλαμβάνεται στο σύνολο εντολών του MIPS (και όχι η προαναφερθείσα ψευδοεντολή με το ίδιο όνομα), η οποία αποθηκεύει το *ακέραιο πηλίκο* της διαίρεσης στον καταχωρητή Lo και το *υπόλοιπο* στον καταχωρητή Hi.

Εντολές μετακίνησης των περιεχομένων των καταχωρητών Lo και Hi (με παραδείγματα):

```
o mflo $s0          # Move From Lo instruction; it moves the
                    # contents of register Lo to $s0: $s0 = Lo
o mfhi $s1          # Move From Hi instruction; it moves the
                    # contents of register Hi to $s1: $s1 = Hi
```

Παρατηρήσεις:

- Όπως και στους τρόπους διευθυνσιοδότησης, έτσι και στις εντολές αριθμητικών πράξεων του MIPS μπορείτε να χρησιμοποιήσετε σαν ορίσματα όποιους καταχωρητές γενικού σκοπού επιθυμείτε.
- Εκτός από τις εντολές που αναφέρθηκαν, υπάρχει μία πληθώρα επιπλέον αριθμητικών και λογικών (*not*, *and*, *or*, *xor*, κ.τ.λ.) εντολών και ψευδοεντολών που μπορούν να χρησιμοποιηθούν κατά τη συγγραφή προγραμμάτων. Ανατρέξτε στο παράρτημα *Assemblers, Linkers and the SPIM Simulator* για περισσότερες λεπτομέρειες.
- Οι εντολές που παρουσιάστηκαν εκτελούν πράξεις μεταξύ ακεραίων. Προφανώς, στο σύνολο εντολών του MIPS υπάρχουν και εντολές για πράξεις αριθμών κινητής υποδιαστολής (π.χ. *add.s*, *sub.s*, *mul.s* και *div.s* για αριθμούς απλής ακρίβειας). Στις

ασκήσεις του εργαστηρίου θα χρειαστεί να εκτελέσετε τέτοια πράξη μία μόνο φορά, οπότε περισσότερες λεπτομέρειες θα δοθούν στην εκφώνηση της συγκεκριμένης άσκησης.

Εντολές ελέγχου της ροής του προγράμματος

Οι εντολές που θα περιγραφούν σε αυτή την παράγραφο πραγματοποιούν αλλαγή της ροής του προγράμματος (κάποιες από αυτές υπό συνθήκη). Με τον όρο «αλλαγή της ροής» εννοούμε ότι μία τέτοια εντολή αναγκάζει τον επεξεργαστή να «πάει σε κάποιο άλλο σημείο του προγράμματος», αντί να εκτελέσει σειριακά την εντολή που ακολουθεί. Κάτι τέτοιο είναι απαραίτητο για τον προγραμματισμό loop, για τον έλεγχο λογικών συνθηκών στο πρόγραμμα αλλά και για την υλοποίηση συναρτήσεων. Γίνεται λοιπόν εύκολα αντιληπτό ότι οι εντολές αυτές είναι πάρα πολύ σημαντικές για τη συγγραφή προγραμμάτων.

Οι περισσότερες από τις εντολές που θα παρουσιαστούν δέχονται σαν όρισμα και ένα label (ετικέτα). Με ένα label «μαρκάρουμε» ένα σημείο του προγράμματος, στο οποίο μας ενδιαφέρει να μεταβεί κάποια στιγμή η ροή του. Με label για παράδειγμα μαρκάρουμε την πρώτη εντολή ενός loop ή την πρώτη εντολή μίας συνάρτησης. Ένα label είναι ένα όνομα ακολουθούμενο από άνω-κάτω τελεία, π.χ. `first_loop:` Προσπαθήστε γενικά να χρησιμοποιείτε ονόματα ενδεικτικά της λειτουργίας του κώδικα που ακολουθεί το label.

Εντολές διακλάδωσης υπό συνθήκη (branches):

```
o  beq $t0, $t1, label     # Branch on equal (beq);
                           # go to label if $t0 = $t1
o  blt $t0, $t1, label     # Branch on less than (blt);
                           # go to label if $t0 < $t1
o  ble $t0, $t1, label     # Branch on less than or equal (ble);
                           # go to label if $t0 <= $t1
o  bgt $t0, $t1, label     # Branch on greater than (bgt);
                           # go to label if $t0 > $t1
o  bge $t0, $t1, label     # Branch on greater than or equal (bge);
                           # go to label if $t0 >= $t1
o  bne $t0, $t1, label     # Branch on not equal (bne);
                           # go to label if $t0 != $t1
```

Εντολές άλματος (jumps):

```
o  j label                 # Unconditional jump to label
o  jr $t3                  # Jump to register (jr) instruction;
                           # go to address contained in $t3
```

Στη συνέχεια δίνεται ένα παράδειγμα υλοποίησης ενός loop στη γλώσσα assembly του MIPS. Πρέπει να γίνει σαφές ότι, κατά τη συγγραφή προγραμμάτων assembly ***(για οποιονδήποτε επεξεργαστή)***, ο μόνος τρόπος υλοποίησης των loop είναι μέσω εντολών υπό συνθήκη διακλάδωσης ή/και άλματος. Το loop του παραδείγματος υπολογίζει το άθροισμα 8+7+6+5+4+3+2+1 και αποθηκεύει το αποτέλεσμα (36) στη μεταβλητή sum. Χρησιμοποιεί τον καταχωρητή $s0 σαν μετρητή των επαναλήψεων του loop και τον $s1 για τον υπολογισμό του αθροίσματος.

```
    .data
sum: .word 0                # Integer variable for storing the sum

    .text
```

```
main:
    li $s0, 8                   # $s0 is the loop counter and also contains the term
                                # that is summed every time; it is initialized to 8
    li $s1, 0                   # $s1 keeps the sum; it is initialized to 0

sum_loop:                       # The label "sum_loop:" marks the first instruction of the loop,
                                # which is the add instruction that follows the label
    add $s1, $s1, $s0           # $s1 = $s1 + $s0; add the current term ($s0) to the sum ($s1)
    addi $s0, $s0, -1           # $s0 = $s0 - 1; decrease loop counter
    bgt $s0, $zero, sum_loop    # If $s0 > $zero (which always contains the 0 value)
                                # go to label sum_loop
    sw $s1, sum                 # When the loop ends (after 8 repetitions) this instruction is
                                # executed; it stores the sum in $s1 to the variable sum in RAM
    . . .
```

Κλήση συναρτήσεων:

Στα προγράμματα assembly, η πρώτη εντολή μίας συνάρτησης «μαρκάρεται» με ένα label. Για την κλήση της συνάρτησης, στην assembly του MIPS, χρησιμοποιείται η εντολή *jal (Jump And Link)* με όρισμα το label της συνάρτησης:

- `jal funct_label  # Calls the function that starts in funct_label`

  Η εντολή *jal* εκτελεί δύο λειτουργίες:

  1. Αποθηκεύει το μετρητή προγράμματος (Program Counter – PC)[2] στον καταχωρητή $ra (return address). Έτσι, στον καταχωρητή $ra φυλάσσεται η διεύθυνση της εντολής που βρίσκεται μετά την εντολή κλήσης μίας συνάρτησης. Η διεύθυνση αυτή είναι απαραίτητη για τη συνέχιση της εκτέλεσης του προγράμματος μετά την επιστροφή από τη συνάρτηση.
  2. Μεταφέρει την εκτέλεση του προγράμματος στο label που έχει σαν όρισμα.

  Με απλά λόγια, η *jal* κάνει ό,τι και η *j (jump)*, με τη διαφορά ότι, πριν το άλμα, σώζει στον $ra τη διεύθυνση που θα επιστρέψει η εκτέλεση μετά την ολοκλήρωση της συνάρτησης.

Η επιστροφή από τη συνάρτηση γίνεται με τη βοήθεια της εντολής *jr* που αναφέρθηκε παραπάνω. Για το σκοπό αυτό, η *jr* θα πρέπει να λάβει σαν όρισμα τον καταχωρητή $ra, στον οποίο έχει προηγουμένως αποθηκευθεί (από τη *jal*) η διεύθυνση επιστροφής από τη συνάρτηση. Συνεπώς, η εντολή *jr $ra* θα πρέπει να είναι η τελευταία εντολή κάθε συνάρτησης.

- `jr $ra            # Return from function call (jump to register $ra)`

[2] Ο μετρητής προγράμματος είναι ένας ειδικός καταχωρητής στους επεξεργαστές, στον οποίο αποθηκεύεται η διεύθυνση της επόμενης προς εκτέλεση εντολής. Με άλλα λόγια, κάθε επεξεργαστής διαβάζει την τιμή του μετρητή προγράμματος για να βρει πού είναι, στη μνήμη, η επόμενη εντολή που πρέπει να εκτελέσει. Ο προγραμματιστής δεν μπορεί να διαβάσει ή να μεταβάλλει με άμεσο τρόπο την τιμή του συγκεκριμένου καταχωρητή (χρησιμοποιώντας για παράδειγμα εντολές όπως η *li* ή η *move*). Και αυτό γιατί ό λόγος ύπαρξης του μετρητή προγράμματος είναι συγκεκριμένος και άρα δεν επιτρέπεται η χρήση του για άλλες λειτουργίες. Τέτοιου είδους καταχωρητές σε έναν επεξεργαστή ονομάζονται *καταχωρητές ειδικού σκοπού*. Φυσικά, η τιμή του μετρητή προγράμματος αυξάνεται αυτόματα κατά την εκτέλεση κάθε εντολής (ώστε να δείχνει στη διεύθυνση της επόμενης), ενώ αλλάζει και κατά την εκτέλεση εντολών υπό συνθήκη διακλάδωσης και άλματος.

Κλήσεις συστήματος και Είσοδος/Έξοδος (εξομοιωτής QtSpim)

Ο εξομοιωτής που θα χρησιμοποιήσετε για την εκτέλεση των προγραμμάτων σας, προσφέρει και ένα μικρό σύνολο λειτουργιών, βασικός στόχος των οποίων είναι η προσθήκη δυνατοτήτων εισόδου/εξόδου στα προγράμματά σας (π.χ. η εκτύπωση ενός μηνύματος στο παράθυρο εισόδου/εξόδου –κονσόλα– του QtSpim, ή το διάβασμα από αυτή ενός ακεραίου ώστε να χρησιμοποιηθεί από το πρόγραμμα). Τέτοιες λειτουργίες προσφέρονται κανονικά από το λειτουργικό σύστημα ενός υπολογιστή μέσω μίας βιβλιοθήκης ειδικών συναρτήσεων, οι οποίες ονομάζονται *κλήσεις συστήματος*. Έτσι ακριβώς ονομάζονται και οι συγκεκριμένες λειτουργίες που προσφέρει ο QtSpim και μπορούν να ενεργοποιηθούν χρησιμοποιώντας την εντολή *syscall (System Call)* του MIPS. Τα βήματα για την πραγματοποίηση μίας κλήσης συστήματος στο περιβάλλον του QtSpim είναι τα εξής:

- Ο κωδικός της κλήσης συστήματος που θέλετε να πραγματοποιήσετε πρέπει να φορτωθεί στον καταχωρητή $v0, ενώ τα σχετικά ορίσματα στους καταχωρητές $a0–$a3 (ή στον $f12 αν πρόκειται για αριθμούς κινητής υποδιαστολής).
- Χρήση της εντολής *syscall*.
- Οι κλήσεις συστήματος που επιστρέφουν τιμές (π.χ. διάβασμα ακεραίου από την κονσόλα του QtSpim), τοποθετούν τις τιμές αυτές στον καταχωρητή $v0 (ή στον $f0 αν είναι κινητής υποδιαστολής).

Οι συνηθέστερα χρησιμοποιούμενες κλήσεις συστήματος που προσφέρει ο QtSpim φαίνονται στον πίνακα που ακολουθεί:

Πίνακας 1.2. Οι πιο συχνά χρησιμοποιούμενες κλήσεις συστήματος του QtSpim

Λειτουργία	Κωδικός Κλήσης Συστήματος	Ορίσματα	Αποτελέσματα
print_int	1	$a0 = ακέραιος για εκτύπωση	
print_float	2	$f12 = αριθμός κινητής υποδιαστολής (απλής ακρίβειας) για εκτύπωση	
print_string	4	$a0 = διεύθυνση κύριας μνήμης του προς εκτύπωση string	
read_int	5		ακέραιος στον $v0
read_float	6		αριθμός κινητής υποδιαστολής (απλής ακρίβειας) στον $f0
read_string	8	$a0 = διεύθυνση κύριας μνήμης της μεταβλητής τύπου string, όπου θα αποθηκευθεί το string που θα διαβαστεί από την κονσόλα $a1 = μέγεθος (πλήθος χαρακτήρων) της μεταβλητής αυτής	
exit	10		
print_char	11	$a0 = χαρακτήρας για εκτύπωση	
read_char	12		χαρακτήρας στον $v0

Παρατηρήσεις:

- Η κλήση συστήματος print_string χρειάζεται τη διεύθυνση ενός string, το οποίο τερματίζεται με το χαρακτήρα '\0' (όπως ακριβώς και στη γλώσσα προγραμματισμού

C). Κατά τη δήλωση μεταβλητών τύπου string, αυτό μπορεί να γίνει με χρήση της directive .asciiz.

- Οι κλήσεις συστήματος read_int και read_float περιμένουν από το χρήστη να πληκτρολογήσει τον αριθμό που επιθυμεί και να πατήσει το πλήκτρο Enter. Αν εσφαλμένα, κατά την πληκτρολόγηση του αριθμού, εισαχθούν χαρακτήρες, τότε οτιδήποτε μετά τον πρώτο χαρακτήρα αγνοείται.
- Η κλήση συστήματος read_string κάνει ό,τι ακριβώς και η συνάρτηση *fgets* στη γλώσσα προγραμματισμού C. Δηλαδή, αν έχει κληθεί με όρισμα *n* (μέγεθος string στον $a1), τότε διαβάζει μέχρι *n*–1 χαρακτήρες και τερματίζει το string με τον χαρακτήρα '\0'. Οι χαρακτήρες αυτοί (μαζί με το '\0') αποθηκεύονται στη μεταβλητή τύπου string, η διεύθυνση της οποίας έχει φορτωθεί στον $a0. Αν ο χρήστης πληκτρολογήσει λιγότερους από *n*–1 χαρακτήρες και πατήσει Enter, τότε η read_string θα αποθηκεύσει τους χαρακτήρες αυτούς, μαζί με το Enter (δηλαδή το χαρακτήρα '\n' – αλλαγή γραμμής) αλλά και το '\0'.
- Η κλήση συστήματος exit τερματίζει το εκτελούμενο πρόγραμμα. Όλα σας τα προγράμματα θα πρέπει να τελειώνουν με τη συγκεκριμένη κλήση συστήματος.

Σημειώνεται ότι ο QtSpim διαθέτει και επιπλέον κλήσεις συστήματος, πέρα από αυτές που αναφέρονται στον Πίνακα 1.2. Ανατρέξτε στο παράρτημα *Assemblers, Linkers and the SPIM Simulator* για περισσότερες λεπτομέρειες.

Παραδείγματα:

Εκτύπωση της τιμής του καταχωρητή $t2:

```
    li $v0, 1           # Load appropriate system call code into register $v0;
                        # code for printing integer is 1
    move $a0, $t2       # Move integer to be printed into $a0:  $a0 = $t2
    syscall             # Use syscall instruction to perform operation
```

Διάβασμα ακεραίου από την κονσόλα και αποθήκευσή του σε μία μεταβλητή στην κύρια μνήμη:

```
    .data
int_var:  .word  0      # Declare integer variable int_var and initialize to 0

    .text
main:
    li $v0, 5           # Load appropriate system call code into register $v0;
                        # code for reading integer is 5
    syscall             # Use syscall instruction to perform operation
    sw $v0, int_var     # Value read from keyboard returned in register $v0;
                        # store this in desired location
    . . .
```

Εκτύπωση string:

```
    .data
str1:  .asciiz  "Print this.\n"   # Declaration for string variable;
                                  # .asciiz directive makes string null terminated
    .text
main:
    li $v0, 4           # Load appropriate system call code into register $v0;
                        # code for printing string is 4
```

```
la $a0, str1            # Load address of string to be printed into $a0
syscall                 # Use syscall instruction to perform operation
```

Τερματισμός προγράμματος με χρήση της κλήσης συστήματος exit:

```
li $v0, 10              # System call code for exit = 10
syscall                 # Use syscall instruction to perform operation
```

Εισαγωγή στον QtSpim

Ο QtSpim είναι ένα, πολύ απλό στη χρήση, πρόγραμμα εξομοίωσης, το οποίο θα σας βοηθήσει να εκτελέσετε προγράμματα που έχουν γραφτεί στη γλώσσα assembly του επεξεργαστή MIPS (και συγκεκριμένα της αρχιτεκτονικής MIPS32). Κατά την φόρτωση ενός προγράμματος στον QtSpim πραγματοποιείται συντακτικός έλεγχος, ενώ κατά την εκτέλεση των εντολών του προγράμματος ανανεώνεται αυτόματα το περιεχόμενο των καταχωρητών και της μνήμης. Το κεφάλαιο αυτό εξηγεί τα βασικά σημεία που πρέπει να γνωρίζετε ώστε να χρησιμοποιήσετε τον QtSpim.

Από πού θα κατεβάσετε τον QtSpim

Μπορείτε να κατεβάσετε το πρόγραμμα εγκατάστασης του QtSpim από το SourceForge.org, ακολουθώντας το link:
http://pages.cs.wisc.edu/~larus/spim.html (δείτε το "New versions of spim" με τα κόκκινα γράμματα στο πάνω μέρος της σελίδας).

Εναλλακτικά μπορείτε να πάτε κατευθείαν στο:
http://sourceforge.net/projects/spimsimulator/files/

Σημειώνεται ότι υπάρχουν εκδόσεις του QtSpim για Windows, για Linux αλλά και για Mac.

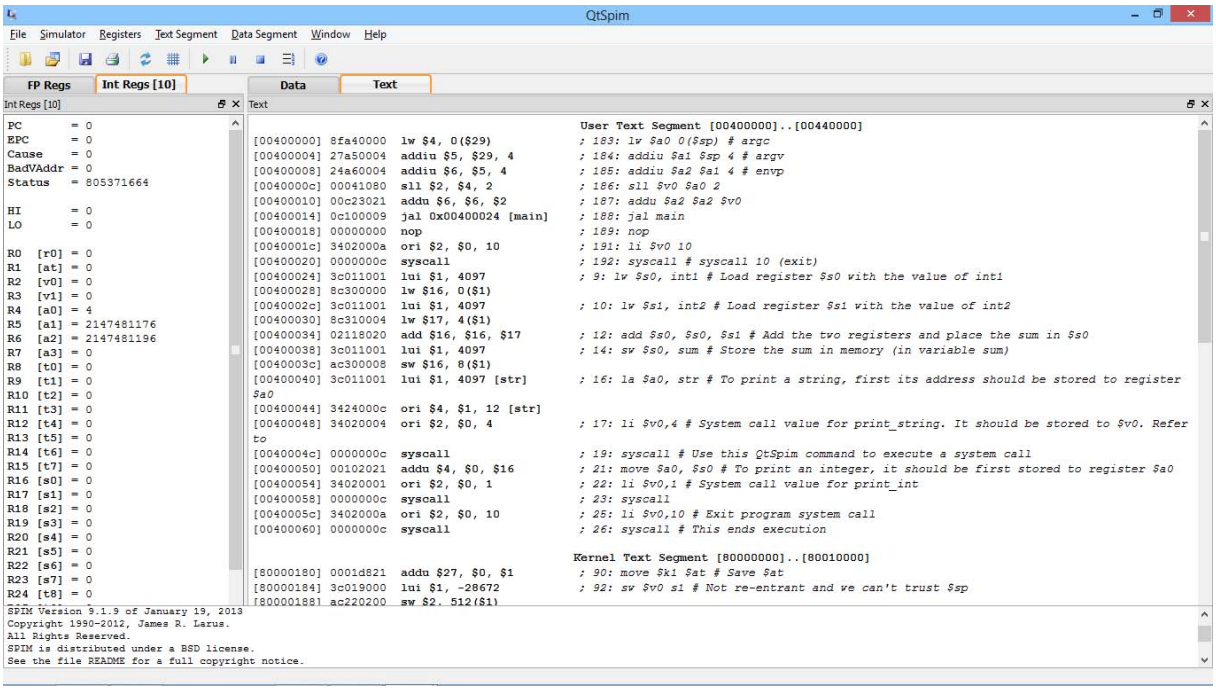
Τι να διαβάσετε σχετικά με τον QtSpim

Αφού εγκαταστήσετε τον QtSpim, από το μενού "Help" επιλέξτε "View Help" και διαβάστε το QtSpim Manual (είναι αρκετά μικρό).

Χρησιμοποιώντας τον QtSpim

Αφού εκτελέσετε τον QtSpim, θα ανοίξει ένα παράθυρο όπως αυτό που φαίνεται στο Σχήμα 2.1. Το παράθυρο αυτό αποτελείται από διάφορα τμήματα:

1. Τα κουμπιά στο επάνω μέρος μπορούν να χρησιμοποιηθούν για τη φόρτωση ενός προγράμματος, την εκτέλεσή του καθώς και για διάφορες άλλες λειτουργίες που αφορούν τον εξομοιωτή.
2. Οι καρτέλες *Regs* (Registers) δείχνουν τα περιεχόμενα όλων των καταχωρητών του MIPS (ακεραίων – *Int Regs* και κινητής υποδιαστολής – *FP Regs*).
3. Η καρτέλα *Text* δείχνει τις εντολές που είναι φορτωμένες στη μνήμη του MIPS και πρόκειται να εκτελεστούν. Συγκεκριμένα, από αριστερά προς τα δεξιά εμφανίζονται η διεύθυνση κύριας μνήμης από όπου ξεκινάει μία εντολή **(κάθε εντολή καταλαμβάνει 4 bytes)**, τα περιεχόμενα της διεύθυνσης στο δεκαεξαδικό (γλώσσα μηχανής), η εντολή σε μορφή assembly, όπου ο κάθε καταχωρητής εμφανίζεται με τον αριθμό που του αντιστοιχεί, ενώ μετά το ελληνικό ερωτηματικό φαίνονται η γραμμή του αρχείου πηγαίου κώδικα που έχετε φορτώσει μαζί με την αντίστοιχη εντολή που έχετε γράψει και τα όποια σχόλια έχετε εισάγει. Προσέξτε ότι επειδή κάποιες από τις εντολές που έχετε χρησιμοποιήσει ενδέχεται να είναι ***ψευδοεντολές***, αυτές αναλύονται σε περισσότερες από μία (πραγματικές) εντολές της assembly του MIPS.
4. Η καρτέλα *Data* δείχνει τις διευθύνσεις και τα περιεχόμενα των data και stack segments της μνήμης.
5. Στο κάτω μέρος του παραθύρου του QtSpim (*Information pane*) εμφανίζονται τα διάφορα μηνύματα του εξομοιωτή.



Σχήμα 2.1. Ο QtSpim

Πώς θα τρέξετε ένα πρόγραμμα στον QtSpim

1. Χρησιμοποιήστε έναν text editor (π.χ. Notepad) για να γράψετε το πρόγραμμά σας και σώστε το αρχείο με επέκταση .s (π.χ. yyyyyy.s).
2. Κάντε κλικ στο κουμπί “Load File” (η αντίστοιχη επιλογή υπάρχει κάτω από το μενού “File”) και ανοίξτε (“Open”) το αρχείο yyyyyy.s.
3. Αν δεν υπάρχουν συντακτικά λάθη και το πρόγραμμα φορτωθεί στη μνήμη του QtSpim, μπορείτε να το τρέξετε πατώντας το κουμπί “Run/Continue” (play) – η αντίστοιχη επιλογή βρίσκεται στο μενού “Simulator”. Όλες οι εντολές θα εκτελεστούν και τα τελικά περιεχόμενα της μνήμης και των καταχωρητών θα φαίνονται στις σχετικές καρτέλες του παραθύρου του εξομοιωτή.
4. Αν η εκτέλεση του προγράμματός σας ολοκληρωθεί σωστά και θέλετε να πραγματοποιήσετε μία νέα, πριν πατήσετε εκ νέου το “Run/Continue”, χρησιμοποιήστε το κουμπί ***“Clear Registers”*** του QtSpim (η σχετική λειτουργία βρίσκεται και στο μενού “Simulator”).

Διόρθωση σφαλμάτων (debugging)

Ο QtSpim έχει δύο λειτουργίες που μπορούν να σας βοηθήσουν να διορθώσετε τα σφάλματα σε ένα πρόγραμμα που δεν τρέχει όπως θα περιμένατε.

Η πρώτη, και πιθανόν και πιο χρήσιμη, είναι η δυνατότητα να τρέξετε ένα πρόγραμμα βηματικά, δηλ. εντολή-εντολή (λειτουργία **“Single Step”**). Κουμπί για τη βηματική εκτέλεση υπάρχει στη γραμμή εργαλείων στο επάνω μέρος του παραθύρου του QtSpim, ενώ η αντίστοιχη επιλογή εμφανίζεται και στο μενού “Simulator”. Κάθε φορά που κάνετε βηματική εκτέλεση, ο QtSpim εκτελεί μία μόνο εντολή και ενημερώνει τα περιεχόμενα όλων των καρτελών, ώστε να μπορέσετε να δείτε πώς τροποποίησε η συγκεκριμένη εντολή τους καταχωρητές ή τη μνήμη.

Στην περίπτωση που ένα πρόγραμμα τρέχει αρκετή ώρα πριν προκύψει ένα σφάλμα, η βηματική εκτέλεση μπορεί να αποδειχθεί κουραστική μέχρι να φτάσετε στο σημείο που βρίσκεται το σφάλμα. Σε τέτοιες περιπτώσεις είναι προτιμότερο να χρησιμοποιήσετε ένα

breakpoint, το οποίο αναγκάζει τον QtSpim να σταματήσει το τρέξιμο του προγράμματος πριν εκτελέσει την εντολή, στην οποία έχει τεθεί το breakpoint. Για να θέσετε ένα breakpoint σε μία εντολή πρέπει να εντοπίσετε την εντολή στην καρτέλα *Text* του QtSpim, να κάνετε δεξί κλικ επάνω της και να επιλέξετε "Set Breakpoint". Με τον ίδιο τρόπο και την επιλογή "Clear Breakpoint" μπορείτε να αφαιρέσετε ένα breakpoint από μία εντολή. Αφού έχετε θέσει τα breakpoints που επιθυμείτε στο πρόγραμμά σας, επιλέξτε κανονική εκτέλεση του προγράμματος (κουμπί "Run/Continue"). Το τρέξιμο του προγράμματος σταματάει πριν την εκτέλεση μίας εντολής στην οποία έχει τεθεί breakpoint και ο QtSpim σας ρωτάει αν θέλετε να συνεχίσετε την εκτέλεση ("Continue"), αν θέλετε να κάνετε βηματική εκτέλεση ("Single Step") ή αν θέλετε να σταματήσετε την εκτέλεση ("Abort"). Σε αυτό το σημείο μπορείτε να παρατηρήσετε τα περιεχόμενα των καταχωρητών και της μνήμης, τα οποία έχουν ανανεωθεί σύμφωνα με τα αποτελέσματα των εντολών που έχουν εκτελεστεί. Ακόμα και αν επιλέξετε "Abort", η εκτέλεση σταματάει στην εντολή με το breakpoint και μπορεί να συνεχιστεί αργότερα από το ίδιο σημείο, είτε με την επιλογή "Run/Continue", είτε με τη "Single Step".

Οι λειτουργίες της βηματικής εκτέλεσης και της τοποθέτησης breakpoint στο πρόγραμμά σας, θα σας βοηθήσουν να εντοπίσετε σφάλματα στον κώδικά σας. Για να τα διορθώσετε, κάντε τις απαραίτητες τροποποιήσεις στο αρχείο με τον πηγαίο κώδικα χρησιμοποιώντας και πάλι τον text editor, με τον οποίο το φτιάξατε. Στη συνέχεια, στο μενού "File" του QtSpim επιλέξτε "Reinitialize and Load File" και φορτώστε το διορθωμένο αρχείο.

Στο σημείο αυτό θα πρέπει να σημειωθεί ότι η επαναρχικοποίηση του εξομοιωτή ***("Reinitialize Simulator")*** *είναι απαραίτητη κάθε φορά που τροποποιείτε το πρόγραμμά σας* ***(δεν αρκεί δηλαδή απλώς η φόρτωση του νέου αρχείου)****. Για το λόγο αυτό εμφανίζεται και σαν ξεχωριστή επιλογή στο μενού "Simulator", ενώ στη γραμμή εργαλείων του QtSpim υπάρχει και ξεχωριστό κουμπί για την εκτέλεση της λειτουργίας επαναρχικοποίησης.*

Χρήσιμες πληροφορίες για τη χρήση του QtSpim

- Μπορείτε να εκτελέσετε όλες τις λειτουργίες του εξομοιωτή και μέσω των μενού "File" και "Simulator".
- Τα περιεχόμενα των καταχωρητών ή της μνήμης δεδομένων μπορούν να εμφανίζονται στο δυαδικό, στο δεκαεξαδικό ή στο δεκαδικό. Η σχετική επιλογή για τους καταχωρητές βρίσκεται στο μενού "Registers", ενώ για τη μνήμη δεδομένων στο μενού "Data Segment".
- Αν κάποιες φορές δεν χρειάζεται να τα βλέπετε, μπορείτε να απενεργοποιήσετε την εμφάνιση του "Kernel Text" (κώδικας πυρήνα – ο βασικός κώδικας εκκίνησης του εξομοιωτή) και των "Kernel Data" (δεδομένα πυρήνα – τα αντίστοιχα δεδομένα), από τις σχετικές επιλογές των μενού "Text Segment" και "Data Segment".
- Για να δείτε τα περιεχόμενα της μνήμης δεδομένων, κάντε απλά κλικ στην καρτέλα "Data".
- Κάνοντας δεξί κλικ στα περιεχόμενα ενός καταχωρητή ή μίας διεύθυνσης μνήμης, σας δίνεται η επιλογή να τα αλλάξετε δυναμικά.

Παράδειγμα προγράμματος

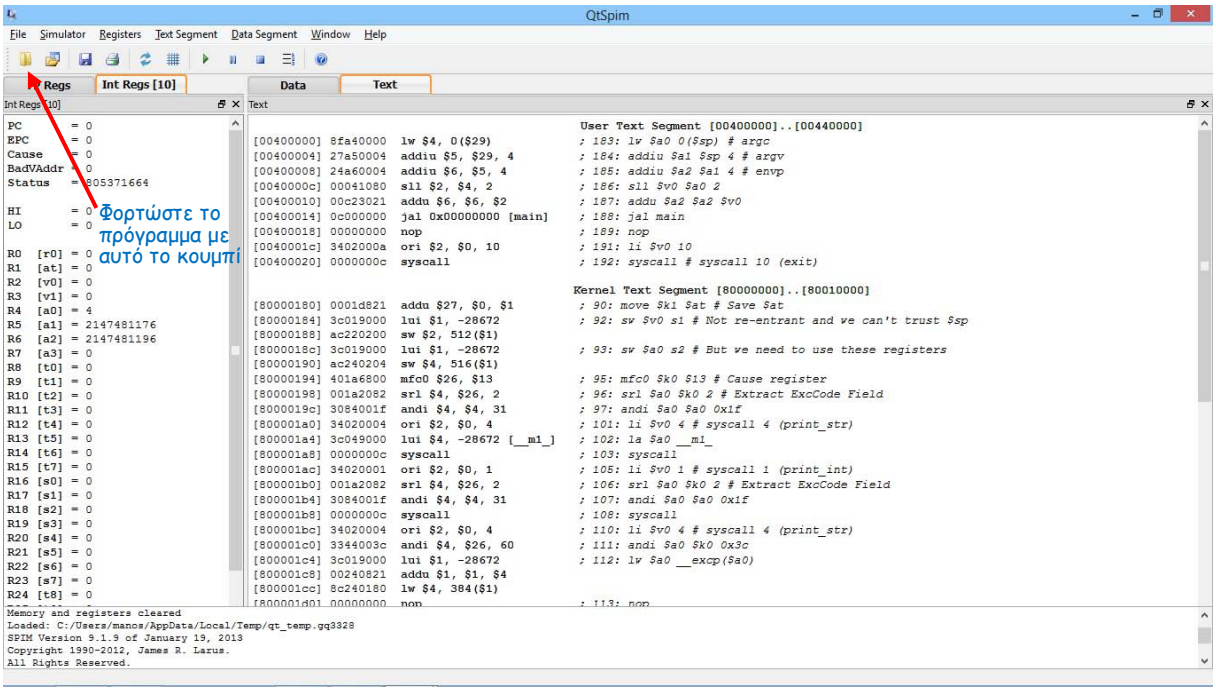
Παρακάτω δίνεται ένα παράδειγμα προγράμματος σε assembly του MIPS. Το πρόγραμμα αυτό υπολογίζει το άθροισμα δύο ακέραιων μεταβλητών (int1, int2), το αποθηκεύει σε μία τρίτη (sum) και το εκτυπώνει στην κονσόλα του QtSpim. Η κονσόλα είναι ένα ξεχωριστό παράθυρο του QtSpim, στο οποίο γίνεται όλη η είσοδος/ έξοδος των προγραμμάτων που εκτελούνται σε αυτόν. Μπορείτε να αντιγράψετε το πρόγραμμα σε έναν text editor, να σώσετε το αρχείο με

επέκταση .s και να το φορτώσετε στον QtSpim. Στη συνέχεια, μπορείτε να κάνετε κανονική ή βηματική εκτέλεση και να δείτε πως μεταβάλλονται οι τιμές των καταχωρητών που χρησιμοποιούνται από το πρόγραμμα (π.χ., του $s0 και του $s1). Επίσης, μπορείτε να δείτε τις τιμές των δύο μεταβλητών που προστίθενται (7 και 22), του τελικού αποτελέσματος (29) καθώς και της συμβολοσειράς (str) που ορίζεται στο πρόγραμμα, στο *User data segment* της καρτέλας *Data* του QtSpim (***προσοχή:*** *αν δεν αλλάξετε την αντίστοιχη επιλογή, οι τιμές αυτές θα εμφανίζονται στο δεκαεξαδικό).* ***Το πρόγραμμα είναι πλήρως σχολιασμένο, όπως θα πρέπει να είναι και οι ασκήσεις που θα παραδίδετε.***

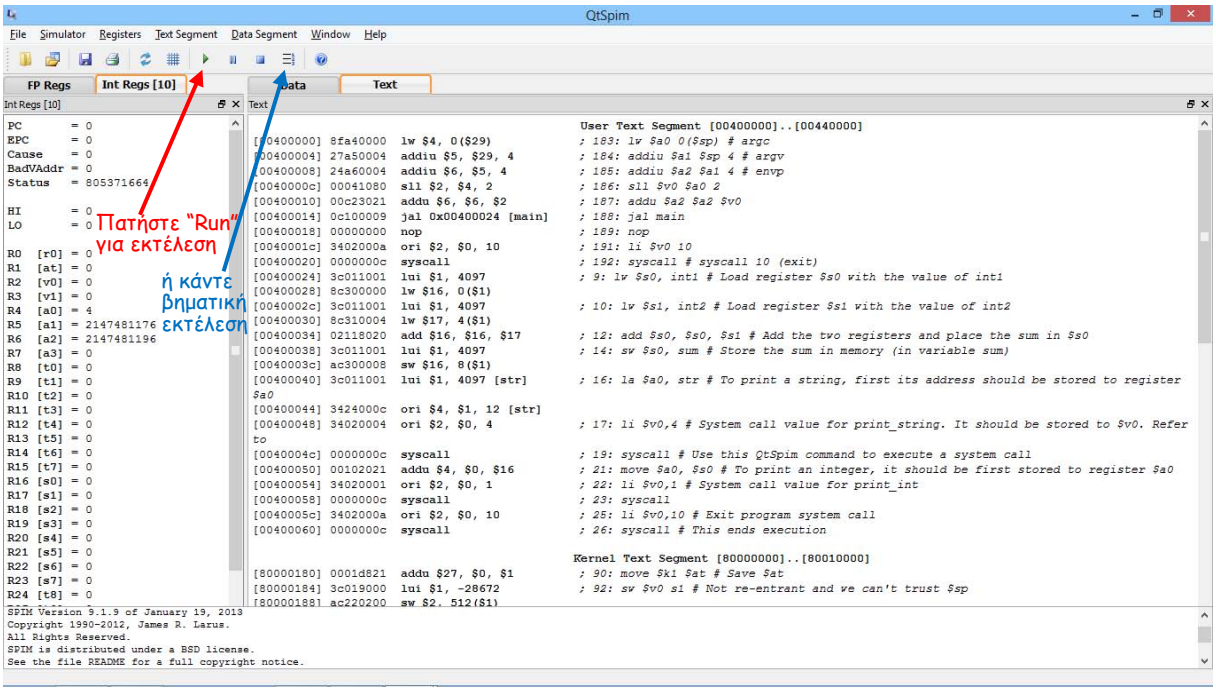
```
  .data                 # Program data are placed below the .data directive
int1: .word 7           # First integer variable, initialized to 7
int2: .word 22          # Second integer variable, initialized to 22
sum:  .word 0           # Variable for storing the sum of the two integers
str:  .asciiz "The sum of the two integers is: "        # String for printing the final result

  .text                 # Program is placed under the .text directive
main:                   # Standard label in QtSpim for the main program. It should be always used
  lw $s0, int1          # Load register $s0 with the value of int1
  lw $s1, int2          # Load register $s1 with the value of int2

  add $s0, $s0, $s1     # Add the two registers and place the sum in $s0

  sw $s0, sum           # Store the sum in memory (in variable sum)

  la $a0, str           # To print a string, first its address should be stored to register $a0
  li $v0, 4             # System call value for print_string. It should be stored to $v0. Refer to
                        # Figure A.9.1 of Assemblers, Linkers and the SPIM Simulator for more details
  syscall               # Use this MIPS command to execute a system call

  move $a0, $s0         # To print an integer, it should be first stored to register $a0
  li $v0, 1             # System call value for print_int
  syscall

  li $v0,10             # Exit program system call
  syscall               # This ends execution
```

Βήματα εκτέλεσης

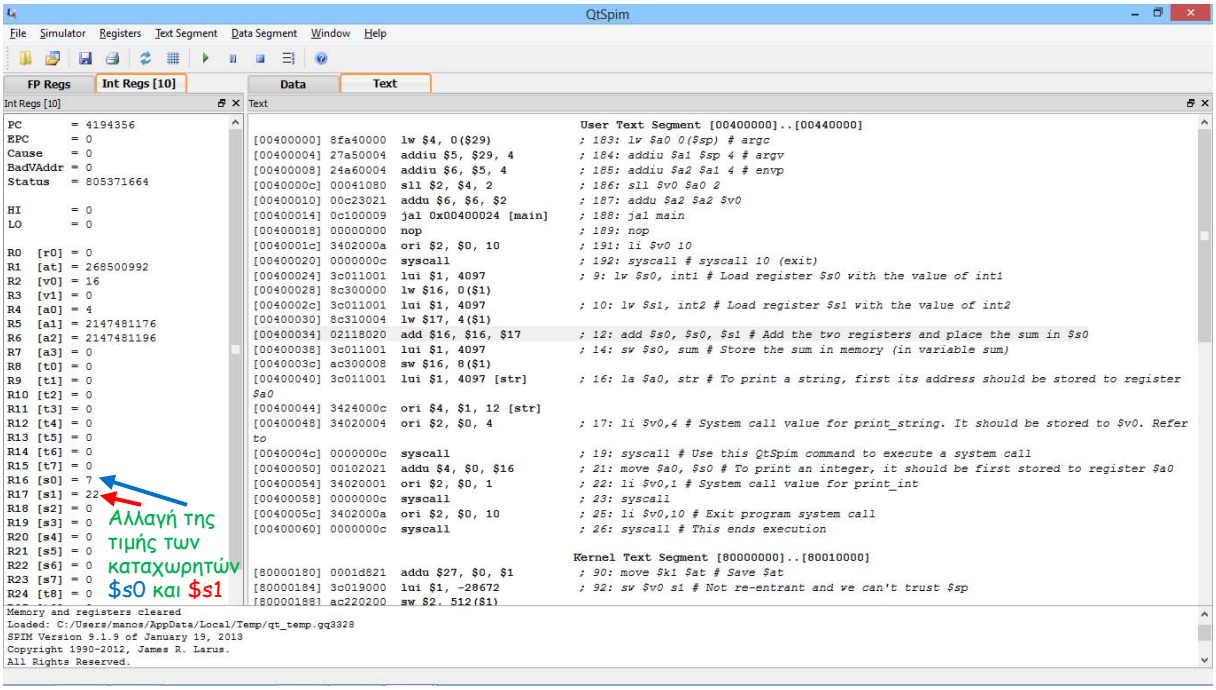
1. Φόρτωση του προγράμματος στον QtSpim

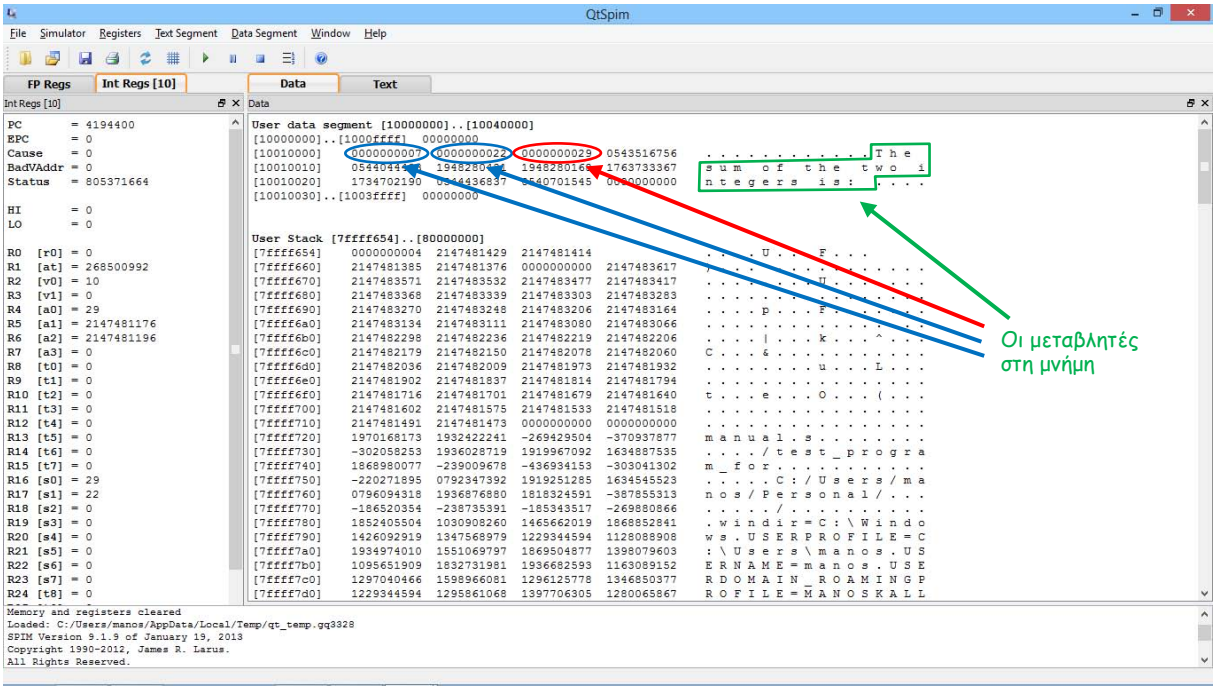


2. Εκτέλεση του προγράμματος

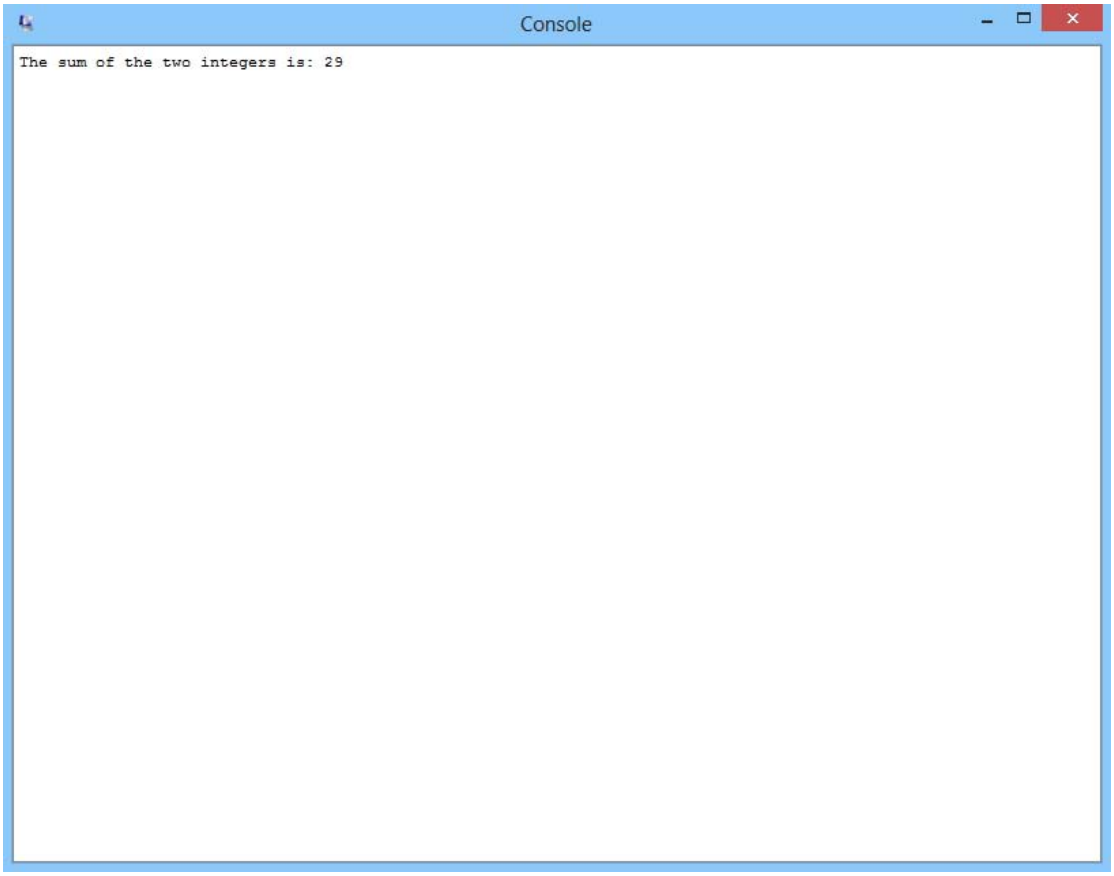


3. Παρατήρηση των αποτελεσμάτων στο περιβάλλον του QtSpim...





... και στην κονσόλα



Εργαστηριακές Ασκήσεις

Άσκηση 1

Ερωτήματα

1. Να γράψετε ένα πρόγραμμα, το οποίο να εμφανίζει το όνομά σας και τον αριθμό μητρώου σας, ***σε δύο συνεχόμενες γραμμές*** στην κονσόλα του QtSpim.

2. Να γράψετε ένα πρόγραμμα, το οποίο θα ζητάει από το χρήστη να εισάγει δύο ακέραιους αριθμούς, *εμφανίζοντας τα κατάλληλα μηνύματα στην κονσόλα του QtSpim*. Στη συνέχεια, θα εκτελεί την πράξη της **αφαίρεσης** του δεύτερου αριθμού από τον πρώτο και θα εμφανίζει το αποτέλεσμα, συνοδευόμενο από κατάλληλο μήνυμα (π.χ. "The result of the subtraction is "), στην κονσόλα.
   Για την αποθήκευση των δύο αριθμών και του αποτελέσματος της αφαίρεσης να χρησιμοποιήσετε συνολικά δύο καταχωρητές του MIPS, τους $t0 και $t1.

Παραδοτέα

Αρχεία με τον κώδικα assembly των προγραμμάτων σας **πλήρως σχολιασμένα**.

Παρατηρήσεις

1. Παραδίδετε μία άσκηση (2 αρχεία) ανά ομάδα.
2. Ο ελλιπής σχολιασμός του κώδικά σας θα οδηγήσει σε μείωση της βαθμολογίας της άσκησης.

Άσκηση 2

Ερώτημα

Η ακολουθία Fibonacci είναι μία αριθμητική σειρά, ο κάθε όρος της οποίας προκύπτει από το άθροισμα των δύο προηγούμενων, ενώ οι δύο πρώτοι όροι είναι οι 0 και 1 (δηλαδή, $fib_{n+2} = fib_{n+1} + fib_n$, με $fib_0 = 0$ και $fib_1 = 1$). Συνεπώς, οι πρώτοι όροι της ακολουθίας είναι οι: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ... Να γράψετε ένα πρόγραμμα, το οποίο θα ζητάει αρχικά από το χρήστη να εισάγει το πλήθος των όρων της ακολουθίας Fibonacci που επιθυμεί να εκτυπωθούν και, στη συνέχεια, θα εκτυπώνει τους όρους αυτούς στην κονσόλα του QtSpim, τον έναν δίπλα στον άλλο, χωρισμένους μεταξύ τους με κενά (δηλαδή, για είσοδο 5, θα πρέπει να εμφανίζεται: 0 1 1 2 3).

Διευκρινίσεις – Υποδείξεις

1. Είναι προφανές ότι για τον υπολογισμό και την εκτύπωση των όρων της ακολουθίας που ζητάει ο χρήστης, θα πρέπει να χρησιμοποιήσετε ένα loop. Κάθε επανάληψη του loop θα διαχειρίζεται δύο όρους της ακολουθίας, έστω τους i και i+1, οι οποίοι θα βρίσκονται αποθηκευμένοι σε δύο καταχωρητές. Συγκεκριμένα, σε κάθε επανάληψη θα εκτυπώνεται ο όρος i της ακολουθίας, θα υπολογίζεται ο όρος i+2 (από τους i και i+1) και θα αποθηκεύονται ***(στους ίδιους καταχωρητές που βρίσκονταν αποθηκευμένοι οι όροι i και i+1)*** οι όροι i+1 και i+2, ώστε να είναι έτοιμοι για την επόμενη επανάληψη του loop.
2. Χρησιμοποιήστε στον κώδικά σας το μικρότερο δυνατό αριθμό καταχωρητών του MIPS. Συγκεκριμένα, για τον υπολογισμό των όρων της ακολουθίας Fibonacci, χρειάζεστε τρεις καταχωρητές (δύο για την αποθήκευση των όρων της ακολουθίας που διαχειρίζεται κάθε επανάληψη του loop, καθώς και έναν προσωρινό για τις εναλλαγές των όρων). Στους τρεις προαναφερθέντες καταχωρητές δεν συνυπολογίζεται ο μετρητής των επαναλήψεων του loop.
3. Μην ξεχάσετε να χρησιμοποιήσετε την κλήση συστήματος exit στον κώδικά σας για το σωστό τερματισμό του προγράμματος.
4. Επαναλάβατε την εκτέλεση του προγράμματος, πραγματοποιώντας τη λειτουργία ***"Clear Registers"*** του QtSpim πριν πατήσετε εκ νέου το κουμπί "Run".

Παραδοτέο

Αρχείο με τον κώδικα assembly του προγράμματός σας **πλήρως σχολιασμένο**.

Άσκηση 3

Ερώτημα

Να γράψετε ένα πρόγραμμα assembly, το οποίο θα ζητάει από το χρήστη να εισάγει δύο ακέραιους αριθμούς και θα υπολογίζει το Μέγιστο Κοινό Διαιρέτη (ΜΚΔ) τους χρησιμοποιώντας τον αλγόριθμο του Ευκλείδη. Συγκεκριμένα:

1. Όταν το πρόγραμμα ξεκινάει θα πρέπει να τυπώνει στην κονσόλα του QtSpim την ακόλουθη φράση: "Calculation of the Greatest Common Divisor of two integers using Euclid's algorithm".
2. Στη συνέχεια θα πρέπει να ζητάει από το χρήστη να εισάγει τον πρώτο ακέραιο αριθμό και κατόπιν, μετά την εισαγωγή του πρώτου ακεραίου, θα πρέπει να ζητείται από το χρήστη και ο δεύτερος ακέραιος.
3. Μετά τους υπολογισμούς, το πρόγραμμα θα πρέπει να τυπώνει το μήνυμα: "The Greatest Common Divisor is: " και δίπλα σε αυτό τον ΜΚΔ των δύο αριθμών.
4. Σε περίπτωση που και οι δύο αριθμοί που εισήγαγε ο χρήστης είναι ίσοι με το 0, το πρόγραμμα θα πρέπει να τυπώνει το μήνυμα: "Both numbers are 0s!!!".
5. Μετά τον υπολογισμό του ΜΚΔ δύο αριθμών, το πρόγραμμα πρέπει να επιστρέφει στην αρχή (Βήμα 2) και να ζητάει από το χρήστη να εισάγει και πάλι δύο αριθμούς (το αρχικό μήνυμα του Βήματος 1 όμως δεν πρέπει να ξανατυπώνεται).

Διευκρινίσεις – Υποδείξεις

1. Ο αλγόριθμος του Ευκλείδη είναι από τους πιο γνωστούς αλγόριθμους υπολογισμού του ΜΚΔ και μπορείτε πολύ εύκολα να βρείτε σχετικές πληροφορίες. Η λειτουργία του βασίζεται σε διαδοχικές διαιρέσεις, από τις οποίες κάθε φορά απαιτείται το υπόλοιπο. Ένα παράδειγμα υπολογισμού του ΜΚΔ των αριθμών 1071 και 1029 φαίνεται στον πίνακα που ακολουθεί:

Πίνακας 3.3.1. Παράδειγμα εφαρμογής του αλγορίθμου του Ευκλείδη

Ακέραιος 1	Ακέραιος 2	Επεξήγηση
1071	1029	Αρχικές τιμές
1029	42	*Επανάληψη 1:* Το 42 είναι το υπόλοιπο της διαίρεσης 1071/1029 το οποίο τοποθετείται δεξιά, ενώ ο διαιρέτης 1029 τοποθετείται αριστερά
42	21	*Επανάληψη 2:* Επαναλαμβάνουμε το loop διαιρώντας το 1029 με το 42 (παίρνουμε υπόλοιπο 21)
21	0	*Επανάληψη 3:* 42/21=2 και υπόλοιπο 0. Ο αλγόριθμος τερματίζεται όταν ο αριθμός στα δεξιά είναι 0, ενώ ο μέγιστος κοινός διαιρέτης είναι ο αριθμός στα αριστερά (το 21 σε αυτή την περίπτωση)

Στο παραπάνω παράδειγμα ο Ακέραιος 1 είναι πάντα μεγαλύτερος του Ακεραίου 2. Εάν αρχικά ο Ακέραιος 2 ήταν μεγαλύτερος από τον Ακέραιο 1 δεν θα υπήρχε πρόβλημα, *αφού η πρώτη επανάληψη του αλγορίθμου θα αντέστρεφε τη σειρά των αριθμών.*

2. Ο χρήστης μπορεί να εισάγει και αρνητικούς αριθμούς. Επειδή όταν σε μία διαίρεση συμμετέχει ένας αρνητικός αριθμός το υπόλοιπο μπορεί να προκύψει αρνητικό, είναι προτιμότερο, πριν εφαρμόσετε τον αλγόριθμο του Ευκλείδη, να μετατρέψετε τους ακέραιους που εισήγαγε ο χρήστης σε **θετικούς** (αν φυσικά κάποιος από αυτούς είναι αρνητικός). Η

μετατροπή μπορεί να γίνει πολύ εύκολα αφαιρώντας κάθε έναν από τους αρνητικούς ακέραιους από το 0.

3. Προσέξτε ώστε κατά την εφαρμογή του αλγορίθμου του Ευκλείδη να αποφεύγονται οι διαιρέσεις με το 0. Ούτως ή άλλως, αν μετά από κάποια επανάληψη ο μικρότερος ακέραιος γίνει 0, ο αλγόριθμος πρέπει να τερματιστεί. Στην περίπτωση που ο ένας από τους δύο **αρχικούς** ακέραιους είναι 0 και ο άλλος όχι, τότε ο ΜΚΔ τους ισούται με τον μη μηδενικό ακέραιο (ή με την απόλυτη τιμή του αν αυτός είναι αρνητικός).
4. Για τον υπολογισμό του υπολοίπου μίας διαίρεσης χρησιμοποιήστε την εντολή div με **δύο** ορίσματα, η οποία αποθηκεύει το πηλίκο της διαίρεσης στον καταχωρητή Lo και το υπόλοιπο στον καταχωρητή Hi. Στη συνέχεια, για να μεταφέρετε το υπόλοιπο από τον καταχωρητή Hi στον καταχωρητή που επιθυμείτε, χρησιμοποιείστε την εντολή mfhi.

Παραδοτέο

Αρχείο με τον κώδικα assembly του προγράμματός σας **πλήρως σχολιασμένο**.

Άσκηση 4

Ερώτημα

Να γράψετε ένα πρόγραμμα assembly, το οποίο θα υπολογίζει και θα αποθηκεύει σε έναν πίνακα τους 20 πρώτους όρους μίας αριθμητικής προόδου, θα τους εκτυπώνει στην κονσόλα του QtSpim με **αντίστροφη** σειρά και στο τέλος θα εκτυπώνει το άθροισμά τους. Υπενθυμίζεται ότι κάθε όρος μίας αριθμητικής προόδου (α_{i+1}) υπολογίζεται από τον προηγούμενό του (α_i) από τη σχέση: $\alpha_{i+1} = \alpha_i + \omega$, όπου ω είναι το βήμα της αριθμητικής προόδου (σταθερός αριθμός). Συγκεκριμένα, το πρόγραμμά σας θα πρέπει:

1. Να δεσμεύει έναν πίνακα 20 ακεραίων (.word). Αυτό πρέπει να γίνεται στο τμήμα δηλώσεων του προγράμματος.
2. Να ζητάει από το χρήστη να εισάγει τον πρώτο όρο (α_0) καθώς και το βήμα ω της αριθμητικής προόδου. Αυτό θα γίνεται με δύο διαδοχικές ερωταποκρίσεις στην αρχή της εκτέλεσης του προγράμματος.
3. Να υπολογίζει 20 όρους της αριθμητικής προόδου ξεκινώντας από τον α_0 (ο α_0 θεωρείται ο πρώτος από τους 20 όρους), και να τους αποθηκεύει έναν προς ένα στον πίνακα που έχει δηλωθεί προηγουμένως.
4. Να προσπελαύνει τον πίνακα με **αντίστροφη** σειρά (από το τελευταίο προς το πρώτο στοιχείο) και να εκτυπώνει τους όρους της ακολουθίας τον έναν δίπλα στον άλλο, χωρισμένους με κενά.
5. Να εκτυπώνει, **σε επόμενη γραμμή της κονσόλας**, το άθροισμα των 20 όρων της ακολουθίας συνοδευόμενο από κατάλληλο μήνυμα (π.χ. "The sum of the 20 first terms of the sequence is "). *Προσοχή: το άθροισμα αυτό θα πρέπει να αποθηκευθεί, μετά τον υπολογισμό του, και σε μία **μεταβλητή στην κύρια μνήμη**, η οποία θα έχει δηλωθεί κατάλληλα μέσα στο πρόγραμμα.*

Διευκρινίσεις – Υποδείξεις

1. Είναι προφανές ότι για την υλοποίηση του ζητούμενου προγράμματος πρέπει να χρησιμοποιήσετε **δύο loop:** ένα για τον υπολογισμό των όρων της αριθμητικής προόδου και την αποθήκευση τους στον πίνακα, και ένα για την εκτύπωση των όρων (με αντίστροφη σειρά) στην κονσόλα του QtSpim.
2. Ο υπολογισμός του αθροίσματος των όρων της προόδου μπορεί να γίνει σε οποιοδήποτε από τα δύο προαναφερθέντα loop.
3. Για την προσπέλαση των διάφορων θέσεων του πίνακα χρησιμοποιήστε την *αναφορά στη μνήμη με χρήση βάσης και καταχωρητή δείκτη* (σελ. 11 του εγχειριδίου). Προσέξτε ότι επειδή πρόκειται για πίνακα ακεραίων, **κάθε θέση του καταλαμβάνει 4 byte** και άρα ο καταχωρητής δείκτη θα πρέπει να αυξηθεί κατά 4 για να προσπελάσετε το επόμενο στοιχείο του πίνακα (ή να μειωθεί κατά 4 για να προσπελάσετε το προηγούμενο).

Παραδοτέο

Αρχείο με τον κώδικα assembly του προγράμματός σας **πλήρως σχολιασμένο**.

Άσκηση 5

Ερώτημα

Να γράψετε ένα πρόγραμμα assembly, το οποίο θα ταξινομεί 10 ακέραιους αριθμούς χρησιμοποιώντας τον αλγόριθμο bubble sort και στη συνέχεια θα τους εκτυπώνει. Συγκεκριμένα:

1. Το πρόγραμμα θα ζητάει 10 ακέραιους αριθμούς από το χρήστη (με διαδοχικές ερωταποκρίσεις της μορφής "Give integer 1: ", "Give integer 2: ", κ.τ.λ.) και θα τους αποθηκεύει σε έναν πίνακα 10 θέσεων που θα έχει δεσμευθεί κατάλληλα στο τμήμα δηλώσεων του προγράμματος.
2. Στη συνέχεια θα εμφανίζει στην κονσόλα του QtSpim το μήνυμα "Unsorted array:" και στην επόμενη γραμμή θα εκτυπώνει τους ακεραίους που αποθηκεύθηκαν προηγουμένως στον πίνακα, τον έναν δίπλα στον άλλο, χωρισμένους με κενά.
3. Ακολούθως θα ταξινομεί τους ακεραίους του πίνακα κατά αύξουσα σειρά (από το μικρότερο στο μεγαλύτερο) χρησιμοποιώντας bubble sort.
4. Τέλος, θα εμφανίζει το μήνυμα "Sorted array:" και στην επόμενη γραμμή θα εκτυπώνει εκ νέου τα στοιχεία του πίνακα, τα οποία θα είναι πλέον ταξινομημένα.

Διευκρινίσεις – Υποδείξεις

1. Ο bubble sort είναι ο πιο απλός αλγόριθμος ταξινόμησης. Η λειτουργία του έχει ως εξής: το πρώτο στοιχείο του πίνακα συγκρίνεται με το δεύτερο. Εάν το πρώτο στοιχείο είναι μεγαλύτερο τότε ανταλλάσουν θέσεις. Κατόπιν, το δεύτερο στοιχείο συγκρίνεται με το τρίτο και γίνεται ανταλλαγή θέσεων αν το δεύτερο στοιχείο είναι μεγαλύτερο του τρίτου. Ακολούθως, το τρίτο στοιχείο συγκρίνεται με το τέταρτο κ.τ.λ. Η διαδικασία αυτή ονομάζεται *«πέρασμα» (pass)*. Συνεπώς, μετά το πρώτο πέρασμα ο μεγαλύτερος ακέραιος θα έχει τοποθετηθεί στη σωστή του θέση, δηλαδή στη θέση του πίνακα με το μεγαλύτερο δείκτη (στη 10η θέση στο ζητούμενο πρόγραμμα). Με διαδοχικά περάσματα ταξινομούνται και τα υπόλοιπα στοιχεία του πίνακα. Η διαδικασία της ταξινόμησης έχει ολοκληρωθεί όταν σε κάποιο πέρασμα δεν γίνει καμία ανταλλαγή θέσεων στοιχείων. Για το σκοπό αυτό απαιτείται η χρήση ενός flag (καταχωρητή), του οποίου η τιμή θα δηλώνει αν έχει γίνει ή όχι ανταλλαγή θέσεων σε ένα πέρασμα.

   Υπάρχουν διάφορες βελτιστοποιήσεις του αλγορίθμου bubble sort. Για παράδειγμα, είναι προφανές ότι μετά το *i*-οστό πέρασμα τα *i* τελευταία στοιχεία του πίνακα είναι ταξινομημένα, οπότε δεν χρειάζεται να ασχοληθούμε καθόλου μαζί τους στο επόμενο πέρασμα. Με πιο προσεκτική παρατήρηση μπορεί να δει κανείς ότι σε κάθε πέρασμα αρκεί να κάνουμε συγκρίσεις μέχρι το σημείο όπου έχει γίνει η τελευταία ανταλλαγή θέσεων στο προηγούμενο πέρασμα, αφού όλα τα στοιχεία από το σημείο αυτό και μετά είναι ταξινομημένα. Στο πρόγραμμά σας μπορείτε να υλοποιήσετε οποιαδήποτε παραλλαγή του αλγορίθμου επιθυμείτε.

   Για περισσότερες λεπτομέρειες αλλά και παραδείγματα εφαρμογής του bubble sort ανατρέξτε στις σημειώσεις και στο βιβλίο του μαθήματος Δομές Δεδομένων, αλλά και στην ηλεκτρονική διεύθυνση: http://en.wikipedia.org/wiki/Bubble_sort.

2. ***Προσοχή:*** Οι δύο εκτυπώσεις του πίνακα (πριν και μετά την ταξινόμηση) θα πρέπει να γίνονται από το **ίδιο τμήμα κώδικα**. Κάθε φορά που τελειώνει η εκτέλεση του τμήματος αυτού, το πρόγραμμα θα πρέπει να συνεχίζει σε διαφορετικό σημείο, ανάλογα με το αν είναι η πρώτη φορά (εκτύπωση του μη ταξινομημένου πίνακα) ή η δεύτερη (εκτύπωση του

ταξινομημένου πίνακα) που εκτελείται το συγκεκριμένο τμήμα κώδικα. Και εδώ θα χρειαστεί να χρησιμοποιήσετε έναν καταχωρητή, του οποίου η τιμή θα σας βοηθήσει να διακρίνετε μεταξύ των δύο διαφορετικών περιπτώσεων εκτύπωσης.

Παραδοτέο

Αρχείο με τον κώδικα assembly του προγράμματός σας **πλήρως σχολιασμένο**.

Άσκηση 6

Ερώτημα

Να γράψετε ένα πρόγραμμα assembly, το οποίο θα υπολογίζει τους *κινούμενους μέσους* 10 θερμοκρασιών και θα τους τυπώνει. Συγκεκριμένα:

1. Το πρόγραμμα θα ζητάει 10 θερμοκρασίες από το χρήστη (με διαδοχικές ερωταποκρίσεις της μορφής "Give temperature 1: ", "Give temperature 2: ", κ.τ.λ.) και θα τους αποθηκεύει σε έναν πίνακα 10 θέσεων που θα έχει δεσμευθεί κατάλληλα στο τμήμα δηλώσεων του προγράμματος. Οι θερμοκρασίες που θα εισάγονται θα είναι **ακέραιοι** αριθμοί (και άρα το ίδιο και ο πίνακας που θα αποθηκεύονται). Υποθέστε ότι αντιστοιχούν στη θερμοκρασία μίας περιοχής, για την ίδια εποχή του χρόνου, τα δέκα τελευταία χρόνια.
2. Στη συνέχεια, το πρόγραμμα σας θα εμφανίζει στην κονσόλα του QtSpim ένα μήνυμα, το οποίο θα ζητάει από το χρήστη το πλήθος των θερμοκρασιών, έστω ***k***, επί των οποίων θα υπολογίζονται οι κινούμενοι μέσοι. Σαν κινούμενους μέσους ορίζουμε τους μέσους όρους *k* θερμοκρασιών (με $k < 10$), οι οποίοι υπολογίζονται ως εξής: ο πρώτος μέσος όρος υπολογίζεται χρησιμοποιώντας τις *k* πρώτες θερμοκρασίες, ο επόμενος χρησιμοποιώντας τις θερμοκρασίες από τη 2η έως την *k*+1, κ.ο.κ., μέχρι να χρησιμοποιηθεί και η τελευταία θερμοκρασία. Για παράδειγμα αν οι δέκα θερμοκρασίες είναι οι θ_1, θ_2, ..., θ_{10} και $k = 7$, τότε προκύπτουν τέσσερις κινούμενοι μέσοι, οι οποίοι είναι οι: $(\theta_1+\theta_2+\theta_3+\theta_4+\theta_5+\theta_6+\theta_7)/7$, $(\theta_2+\theta_3+\theta_4+\theta_5+\theta_6+\theta_7+\theta_8)/7$, $(\theta_3+\theta_4+\theta_5+\theta_6+\theta_7+\theta_8+\theta_9)/7$ και $(\theta_4+\theta_5+\theta_6+\theta_7+\theta_8+\theta_9+\theta_{10})/7$.
3. Τέλος, το πρόγραμμα θα εμφανίζει ένα μήνυμα της μορφής "Average temperatures per *k* years:", **όπου *k* θα είναι η τιμή που έχει εισάγει ο χρήστης προηγουμένως**, θα υπολογίζει τους κινούμενους μέσους και θα τους εμφανίζει στην επόμενη γραμμή της κονσόλας του QtSpim, τον έναν δίπλα στο άλλο, χωρισμένους με κενά. Σημειώνεται ότι **δεν** χρειάζεται οι κινούμενοι μέσοι να αποθηκεύονται σε κάποιον πίνακα. Απλώς, ο καθένας από αυτούς θα εκτυπώνεται στην κονσόλα αμέσως μετά τον υπολογισμό του.

Διευκρινίσεις – Υποδείξεις

1. Ο υπολογισμός των κινούμενων μέσων πρέπει να γίνεται με κλήση μίας ***συνάρτησης***. Η ζητούμενη συνάρτηση θα υπολογίζει **έναν** κινούμενο μέσο σε κάθε κλήση της και θα παίρνει *δύο (2) παραμέτρους*: τη θέση του πίνακα θερμοκρασιών από όπου πρέπει να ξεκινήσει τους υπολογισμούς **(στον καταχωρητή $a0)**, καθώς και την τιμή *k* που έχει εισάγει ο χρήστης **(στον καταχωρητή $a1)**. Έτσι, για τον υπολογισμό του τρίτου κινούμενου μέσου στο παράδειγμα παραπάνω, $(\theta_3+\theta_4+\theta_5+\theta_6+\theta_7+\theta_8+\theta_9)/7$, η συνάρτηση θα πρέπει να κληθεί με την τιμή 2 στον καταχωρητή $a0 *(θεωρώντας ότι η αρίθμηση των θέσεων του πίνακα ξεκινάει από το 0 και άρα η 3η θερμοκρασία βρίσκεται στη θέση 2 του πίνακα)*, και με την τιμή 7 στον καταχωρητή $a1.
2. Οι συναρτήσεις στην assembly δεν είναι τίποτα άλλο από ένα τμήμα κώδικα που ξεκινάει από κάποιο label. Με άλλα λόγια, **δεν** υπάρχει κάποια συγκεκριμένη δήλωση με την οποία να ορίζουμε συναρτήσεις στην assembly. Απλά, βάζουμε ένα label κάπου μέσα στο τμήμα κώδικα του προγράμματός μας και κάτω από το label αυτό παραθέτουμε τον κώδικα της συνάρτησης.
3. Όταν θέλουμε να καλέσουμε μία συνάρτηση εκτελούμε την εντολή *jal FUNC_LABEL*, οπού *FUNC_LABEL* είναι το label από το οποίο ξεκινάει ο κώδικας της συνάρτησης. Πριν την εντολή *jal* θα πρέπει να έχουμε θέσει τους καταχωρητές που χρησιμοποιούνται για πέρασμα παραμέτρων σε συνάρτηση (τους $a0 και $a1 στην παρούσα άσκηση) στις

κατάλληλες τιμές. Οι τιμές των καταχωρητών αυτών χρησιμοποιούνται από τον κώδικα της συνάρτησης για τον υπολογισμό του επιθυμητού αποτελέσματος.

4. Η τελευταία εντολή κάθε συνάρτησης πρέπει να είναι πάντα η *jr $ra* (επιστροφή από κλήση συνάρησης).
5. Η συνάρτηση που σας ζητείται να υλοποιήσετε είναι μία απλή συνάρτηση, καθώς δεν χρειάζεται ούτε να καλέσει άλλες συναρτήσεις, αλλά ούτε και να διαχειριστεί κάποια ποσότητα μνήμης. Το μόνο που θα πρέπει να προσέξετε για να μην παραβείτε τους κανόνες χρήσης των καταχωρητών του MIPS (βλ. Πίνακα 1.1 του εγχειριδίου) είναι να χρησιμοποιήσετε στο κυρίως πρόγραμμα τους καταχωρητές $s0–$s7 και στη συνάρτηση τους καταχωρητές $t0–$t9. [3]
6. Οι κινούμενοι μέσοι που πρέπει να υπολογίσετε είναι πραγματικοί (και όχι ακέραιοι) αριθμοί. Για το λόγο αυτό, η διαίρεση με την τιμή *k* που έχει εισάγει ο χρήστης πρέπει να είναι **κινητής υποδιαστολής και όχι ακέραια** (σε αντίθεση με την άθροιση των θερμοκρασιών – οι θερμοκρασίες είναι ακέραιες και άρα και το άθροισμα τους είναι ακέραιο). Για να εκτελέσετε διαίρεση κινητής υποδιαστολής θα πρέπει τα τελούμενα της πράξης να βρίσκονται σε καταχωρητές κινητής υποδιαστολής αλλά και να έχουν μετατραπεί ώστε να έχουν την κατάλληλη μορφή (κινητής υποδιαστολής). Αυτό μπορεί να γίνει εκτελώντας διαδοχικά τις εντολές ***mtc1*** και ***cvt.s.w***. Η *mtc1* μεταφέρει το περιεχόμενο ενός ακέραιου καταχωρητή σε έναν καταχωρητή κινητής υποδιαστολής. Για παράδειγμα η *mtc1 $t0, $f0* μεταφέρει το περιεχόμενο του ακέραιου καταχωρητή $t0 στον καταχωρητή κινητής υποδιαστολής $f0 *(προσοχή, η σειρά των καταχωρητών κατά τη σύνταξη της mtc1 είναι αντίστροφη από αυτή των περισσότερων εντολών της assembly του MIPS)*. Στη συνέχεια, εκτελώντας *cvt.s.w $f1, $f0* ο ακέραιος αριθμός που αποθηκεύτηκε στον $f0 μεταφέρεται στον $f1 και μετατρέπεται σε κινητής υποδιαστολής (απλής ακρίβειας). Αρά πλέον ο $f1 περιέχει τον επιθυμητό αριθμό (αυτόν που είναι αποθηκευμένος στον $t0) σε μορφή κινητής υποδιαστολής. Η *cvt.s.w* μπορεί προφανώς να συνταχθεί χρησιμοποιώντας τον ίδιο καταχωρητή σαν source και destination, π.χ. *cvt.s.w $f0, $f0*.
7. Η προαναφερθείσα διαδικασία με τις εντολές *mtc1* και *cvt.s.w* πρέπει να πραγματοποιηθεί δύο φορές, μία για το άθροισμα των θερμοκρασιών και μία για την τιμή *k*.
8. Η διαίρεση των καταχωρητών κινητής υποδιαστολής, στους οποίους έχουν μεταφερθεί και μετατραπεί κατάλληλα το άθροισμα των θερμοκρασιών και το *k*, θα γίνει με χρήση της εντολής *div.s*.
9. Το αποτέλεσμα της διαίρεσης είναι η τιμή που πρέπει να επιστρέψει η συνάρτηση. Ο καταχωρητής για επιστροφή αποτελεσμάτων κινητής υποδιαστολής είναι ο $f0. Για το λόγο αυτό, η συνάρτηση, πριν την εντολή επιστροφής *jr $ra* θα πρέπει να αποθηκεύσει το αποτέλεσμα της διαίρεσης στον καταχωρητή $f0 (αυτό μπορεί να γίνει απευθείας, συντάσσοντας κατάλληλα την εντολή *div.s*).
10. Η εκτύπωση του υπολογισθέντα κινούμενου μέσου (η οποία γίνεται από το κυρίως πρόγραμμα, έξω από τη συνάρτηση), θα πραγματοποιηθεί με χρήση της κλήσης συστήματος print_float (με κωδικό 2). Η εντολή για την αντιγραφή της τιμής ενός

[3] Για την υλοποίηση πιο περίπλοκων συναρτήσεων, δηλ. συναρτήσεων που απαιτούν τη χρήση τόσο των $t όσο και των $s καταχωρητών, που διαχειρίζονται εσωτερικά κάποια ποσότητα μνήμης ή που καλούν άλλες συναρτήσεις, απαιτείται η διαχείριση της στοίβας του MIPS (τη διαχείριση αυτή την «κρύβει» εντελώς ο compiler όταν γράφουμε κώδικα σε γλώσσες υψηλού επιπέδου, π.χ. σε C). Η διαχείριση της στοίβας κάνει τον κώδικα της συνάρτησης ελαφρώς πιο περίπλοκο. Όποιος ενδιαφέρεται για περισσότερες λεπτομέρειες μπορεί να διαβάσει την παράγραφο Α.6 του παραρτήματος *Assemblers, Linkers and the SPIM Simulator* καθώς και τα http://cs.ucsb.edu/~franklin/154/Fall07/lectures/Lec5.ppt, http://uscspring2010.googlecode.com/svn/trunk/EE%20352/MIPSXCallingXConventionsXSummary.pdf.

καταχωρητή κινητής υποδιαστολής σε έναν άλλο είναι η *mov.s* (σε αντιστοιχία με τη move για τους καταχωρητές γενικού σκοπού).

Παραδοτέο

Αρχείο με τον κώδικα assembly του προγράμματός σας **πλήρως σχολιασμένο**.