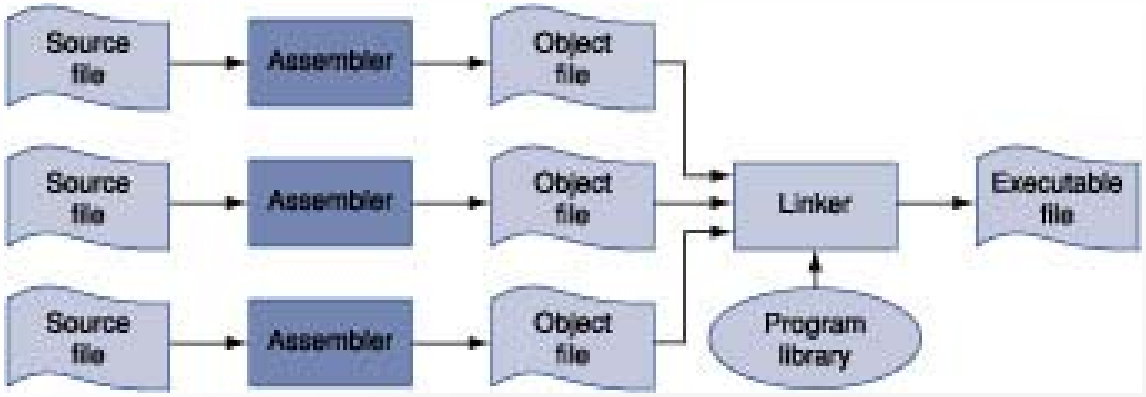
Εργαστήριο Αρ. 11

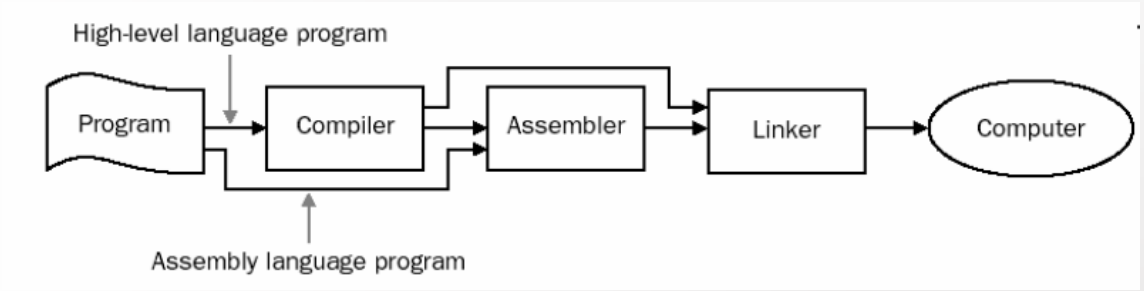
Εισαγωγή στην Αρχιτεκτονική MIPS

Πέτρος Παναγή, PhD

References:
http://pages.cs.wisc.edu/~larus/HP_AppA.pdf
http://spimsimulator.sourceforge.net/

Πώς Δημιουργείτε ένα Executable Αρχείο





Source File: Ένα TEXT αρχείο με τον πυγαίο κώδικα.

Assemble: Μεταφράζει τις assembly εντολές σε γλώσσα μηχανής

Linker: Ο Linker παίρνει διάφορα objects files και program library files για να δημιουργήσει το εκτελέσιμο αρχείο (executable file)

Executable File: Είναι το Binary File που αντιγράφεται στην μνήμη του υπολογιστή και εκτελείται.

Assembly with Labels → Assembly → Machine Language

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra

        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

```
addiu   $29, $29, -32
sw      $31, 20($29)
sw      $4,  32($29)
sw      $5,  36($29)
sw      $0,  24($29)
sw      $0,  28($29)
lw      $14, 28($29)
lw      $24, 24($29)
multu   $14, $14
addiu   $8,  $14, 1
slti    $1,  $8, 101
sw      $8,  28($29)
mflo    $15
addu    $25, $24, $15
bne     $1,  $0, -9
sw      $25, 24($29)
lui     $4,  4096
lw      $5,  24($29)
jal     1048812
addiu   $4,  $4, 1072
lw      $31, 20($29)
addiu   $29, $29, 32
jr      $31
move    $2,  $0
```

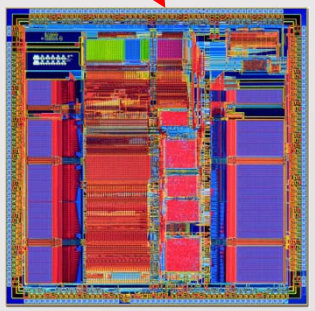
```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

```
   #include <stdio.h>

int
main (int argc, char *argv[])
{
        int i;
        int sum = 0;

        for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
        printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

C ή C++



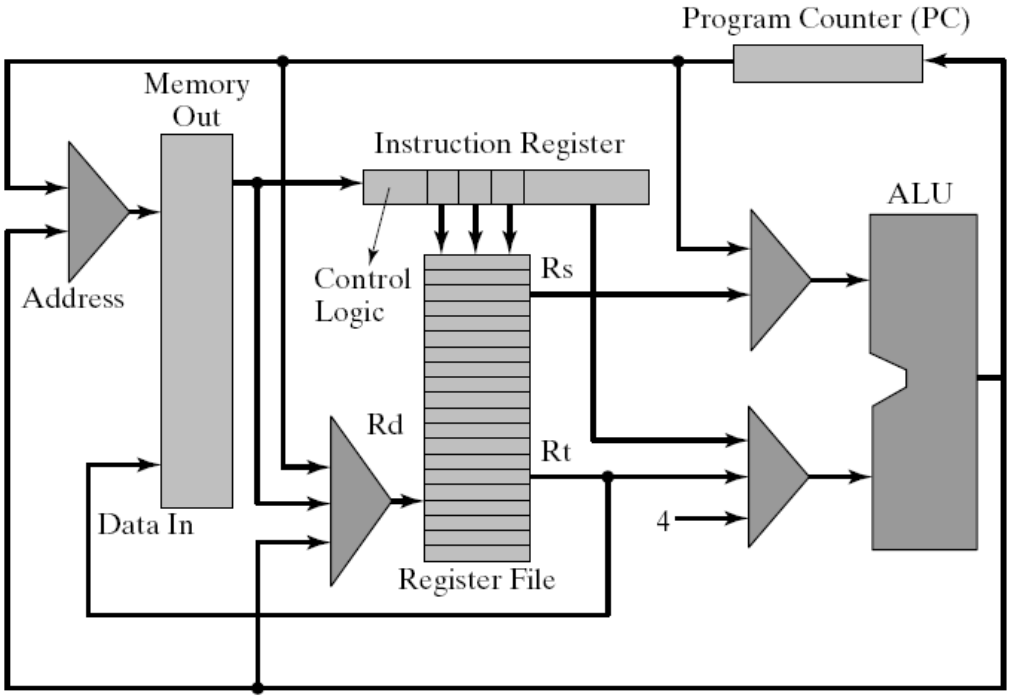
Assembly Language/Η Γλώσσα Μηχανής

Η Συμβολική γλώσσα (Assembly) είναι η συμβολική αντιπροσώπευση της δυαδικής γλώσσας μηχανών (Machine Language). Η Assembly είναι πιο αναγνώσιμη από τη γλώσσα μηχανών επειδή χρησιμοποιεί σύμβολα. Επιπλέον, η Assembly επιτρέπει στους προγραμματιστές να χρησιμοποιούν τις ετικέτες (Labels) για να προσδιορίζουν και να ονομάσουν τις μεταβλητές και τοποθεσίες μνήμης που φυλάσσουν τις οδηγίες ή τα στοιχεία.

Πλεονεκτήματα της Assembly: Το μικρό μέγεθος και η ***ταχύτητα*** των Assembly προγραμμάτων είναι τα κύρια πλεονεκτήματα σε σχέση με τις High Level Languages όπως C++ και Java. Αυτό είναι ιδιαίτερα σημαντικά για ενσωματωμένους υπολογιστής (embedded computer) όπου η υπολογιστική ισχύς και η μνήμη είναι περιορισμένη. Μπορούμε να ***εκμεταλλευτούμε τις αναβαθμίσεις των επεξεργαστών*** αρκετά πριν το κάνουν οι μεταγλωττιστές.

Μειονεκτήματα της Assembly: Ίσως το σημαντικό μειονέκτημά του είναι ότι τα προγράμματα που γράφονται στη Assembly ***είναι συγκεκριμένα για μια μηχανή*** και πρέπει να ξαναγραφούν συνολικά για να τρέξουν σε μια άλλη αρχιτεκτονική υπολογιστών. Ένα άλλο μειονέκτημα είναι ότι τα προγράμματα Assembly είναι πιο ***χρονοβόρα στο να γραφτούν*** από τα ισοδύναμα προγράμματα που γράφονται σε μια υψηλού επιπέδου γλώσσα.

Αρχιτεκτονική MIPS



- Control Unit
- Register File
- Arithmetic and Logic Unit (ALU)
- Program Counter (PC)
- Memory
- Instruction Register (IR)

MIPS simplified datapath diagram.

* Robert Britton, Professor Emeritus, **MIPS Assembly Language Programming**, Prentice Hall, 2004, ISBN-10:0131420445

Η αρχιτεκτονική MIPS έχει 32 καταχωρητές

Register name	Number	Usage
$zero	0	constant 0
$at	1	reserved for assembler
$v0	2	expression evaluation and results of a function
$v1	3	expression evaluation and results of a function
$a0	4	argument 1
$a1	5	argument 2
$a2	6	argument 3
$a3	7	argument 4
$t0	8	temporary (not preserved across call)
$t1	9	temporary (not preserved across call)
$t2	10	temporary (not preserved across call)
$t3	11	temporary (not preserved across call)
$t4	12	temporary (not preserved across call)
$t5	13	temporary (not preserved across call)
$t6	14	temporary (not preserved across call)
$t7	15	temporary (not preserved across call)
$s0	16	saved temporary (preserved across call)
$s1	17	saved temporary (preserved across call)
$s2	18	saved temporary (preserved across call)
$s3	19	saved temporary (preserved across call)
$s4	20	saved temporary (preserved across call)
$s5	21	saved temporary (preserved across call)
$s6	22	saved temporary (preserved across call)
$s7	23	saved temporary (preserved across call)
$t8	24	temporary (not preserved across call)
$t9	25	temporary (not preserved across call)
$k0	26	reserved for OS kernel
$k1	27	reserved for OS kernel
$gp	28	pointer to global area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address (used by function call)

- **Register $0** πάντα περιέχει την τιμή 0 (hardwired value 0)
- **Registers $at (1), $k0 (26),** and **$k1 (27)** χρησιμοποιούνται από τον assembler και το λειτουργικό σύστημα και δεν πρέπει να χρησιμοποιηθούν από τα προγράμματα χρηστών ή τους μεταγλωττιστές.
- **Registers $a0–$a3 (4–7)** χρησιμοποιούνται για να περάσουν τις πρώτες τέσσερις παραμέτρους (arguments) στις συναρτήσεις

Registers $v0 and $v1 (2, 3) χρησιμοποιούνται για να τιμές που επιστρέφουν οι συναρτήσεις .

Registers $t0–$t9 (8–15, 24, 25) είναι **caller-saved registers** που χρησιμοποιείται για να κρατήσει τις προσωρινές ποσότητες που δεν χρειάζονται να συντηρηθούν στις κλήσεις.

- **Registers $s0–$s7 (16–23)** είναι **callee-saved registers** μακρόβιες τιμές που πρέπει να συντηρηθούν στις κλήσεις.
- **Register $gp (28)** είναι **global pointer** που κρατά το μέσο του 64K block μνήμης στο static data segment.
- **Register $sp (29)** κρατά την τιμή stack pointer, που είναι και η τελευταία θέση στο stack.
- **Register $fp (30)** είναι το frame pointer.
- **Register $ra (31),** είναι **return address** σε procedure call.

Δομή ενός προγράμματος σε Assembly

```
##################################################################
#Program Name     : Basic program to print "The Answer = " and a number
#Programmer       : Petros Panayi  Stud. ID:000000
#Date Last Modif.: 10 Sep 2006
##################################################################
# Comments: A simple program that prints a string message and the results
#        Two syscall one for the string and the other for the integer
##################################################################
    .data           # data segment
str:
    .asciiz "the answer = "

    .text           # text segment
    .globl main

main:
    li $v0, 4       # 4 = print str syscall
    la $a0, str     # load address of string
    syscall         # execute the system call

    li $v0, 1       # 1 = print int syscall
    li $a0, 5       # load int value 5
    syscall         # execute the system call
```

Σύνταξη για τις οδηγίες Συμβολικής γλώσσας MIPS

```
[label:] Op-Code [operand], [operand], [operand] [#comment]
```

- Το Πεδίο **`label:`** όπως και τα **`[operand], [operand], [operand] [#comment]`** είναι προαιρετικά.
- Τα **κόμματα** είναι επίσης προαιρετικά.
- Ο διαχωρισμός των πεδίων γίνεται με **SPACES** ή **TABS**
- **Στη Συμβολική γλώσσα (Assembly) MIPS, οτιδήποτε σε μια γραμμή μετά από το σημάδι (#) είναι σχόλιο.**
- **S.O.S**. Όταν εκτυπώνουμε τα reports ο κώδικας και τα σχόλια δεν πρέπει να ξεπερνούν την μία γραμμή.

Instructions Format

Όλες οι MIPS εντολές έχουν **32bit** μέγεθος. Έχουμε τρία είδε εντολών.

R-Type (Register Type)

Op-Code	Rs	Rt	Rd		Function Code
000000	sssss	ttttt	ddddd	00000	ffffff

Op-Code: η λειτουργία της εντολής (operation of the instruction)
Rs: ο πρώτος καταχωρητής εισόδου (the first register, source operand)
Rt: ο δεύτερος καταχωρητής εισόδου (the second register, source operand)
Rd: ο καταχωρητής που θα πάρει το αποτέλεσμα (the register destination operand)
shamt: ποσότητα μετακίνησης
Function Code: συνάρτηση (function). Αυτό το πεδίο αποτελεί διαφοροποίηση της εντολής από το πεδίο Op-Code.

I-Type (Immediate Type)

Op-Code	Rs	Rt	Immediate
ffffff	sssss	ttttt	iiiiiiiiiiiiiiii

J-Type (Jump Type)

Op-Code	Target
00001f	tttttttttttttttttttttttttt

Βασικές Εντολές (Παράδειγμα)

Assembly instruction	Instr. Format	*op* / *funct*	Meaning	Comments
add $rd, $rs, $rt	R	0/32	$rd = $rs + $rt	Add contents of two registers
sub $rd, $rs, $rt	R	0/34	$rd = $rs - $rt	Subtract contents of two registers
addi $rt, $rs, imm	I	8	$rt = $rs + imm	Add signed constant
addu $rd, $rs, $rt	R	0/33	$rd = $rs + $rt	Unsigned, no overflow
subu $rd, $rs, $rt	R	0/35	$rd = $rs - $rt	Unsigned, no overflow
addiu $rt, $rs, imm	I	9	$rt = $rs + imm	Unsigned, no overflow

Άσκηση: Μετατρέψετε την πιο κάτω εντολή σε BIN και HEX

add $t2, $t0, $t1

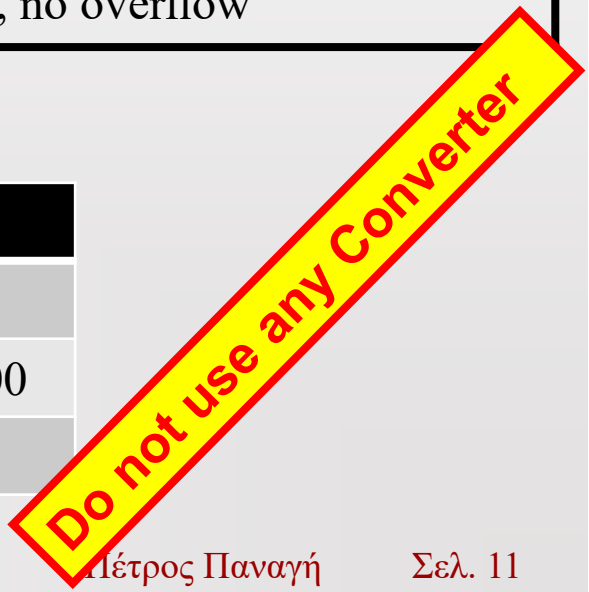
Βασικές Εντολές (Παράδειγμα)

Assembly instruction	Instr. Format	*op* / *funct*	Meaning	Comments
add $rd, $rs, $rt	R	0/32	$rd = $rs + $rt	Add contents of two registers
sub $rd, $rs, $rt	R	0/34	$rd = $rs - $rt	Subtract contents of two registers
addi $rt, $rs, imm	I	8	$rt = $rs + imm	Add signed constant
addu $rd, $rs, $rt	R	0/33	$rd = $rs + $rt	Unsigned, no overflow
subu $rd, $rs, $rt	R	0/35	$rd = $rs - $rt	Unsigned, no overflow
addiu $rt, $rs, imm	I	9	$rt = $rs + imm	Unsigned, no overflow

Άσκηση: Μετατρέψετε την πιο κάτω εντολή σε BIN και HEX

add $t2, $t0, $t1

Op Code	Rs	Rt	Rd	Shamt	Fun.
0	8	9	10	x	32
0000-00	01-000	0-1001-	0101-0	000-00	10-0000
0x01095020					

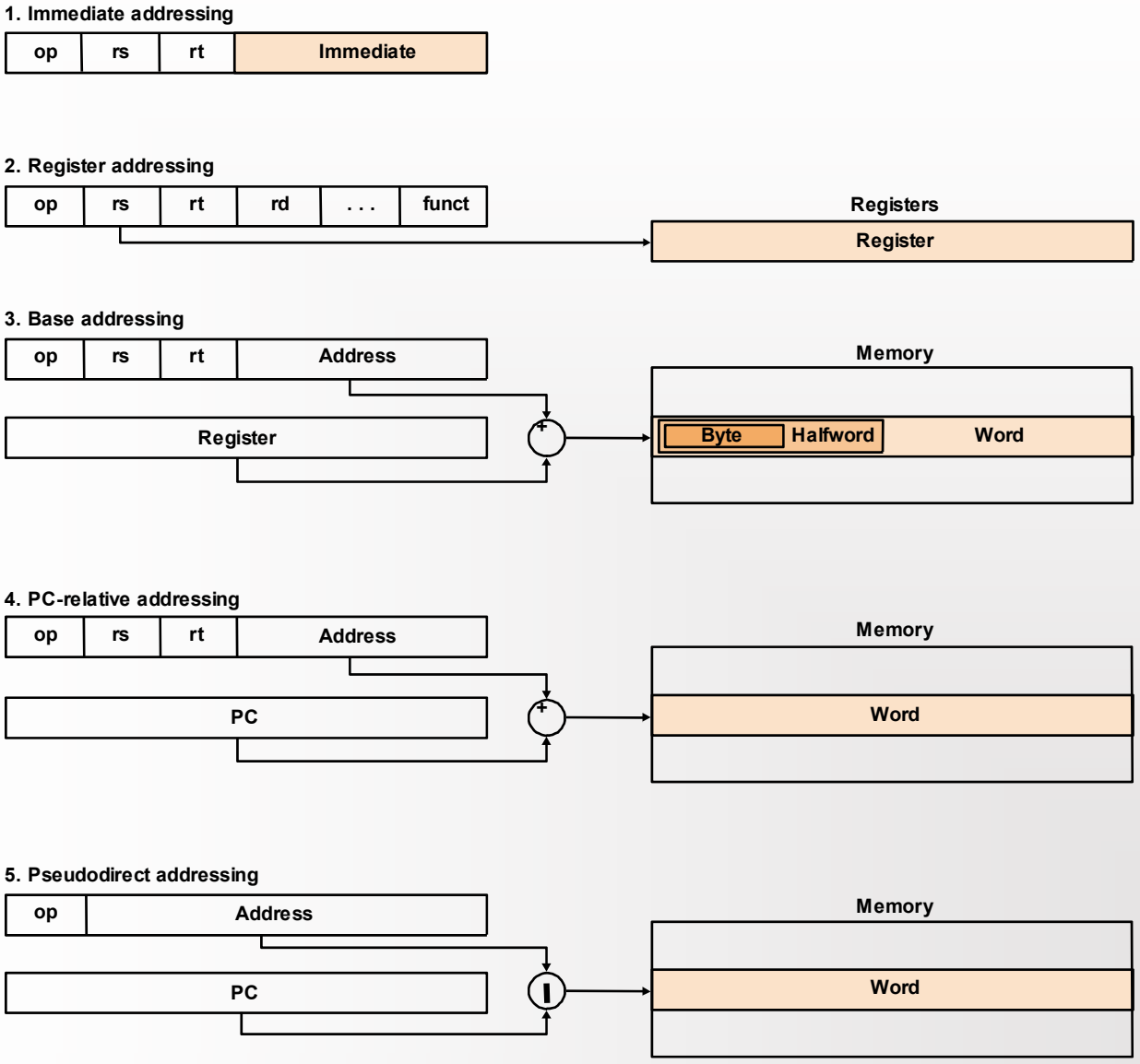


Βασικές Εντολές (Παράδειγμα)

lw $rt, imm($rs)	I	35	$rt = M[$rs + imm]	Load word from memory
sw $rt, imm($rs)	I	43	M[$rs + imm] = $rt	Store word in memory

beq $rs, $rt, imm	I	4	if($rs==$rt) PC = PC + imm (PC always points to next instruction)
bne $rs, $rt, imm	I	5	if($rs!=$rt) PC = PC + imm (PC always points to next instruction)

Διευθυνσιοδότηση/Addressing



Ένα Απλό Πρόγραμμα

```
#####################################################################
#Program Name    : Basic program to print "The Answer = " and a number
#Programmer      : Petros Panayi  Stud. ID:000000
#Date Last Modif.: 10 Sep 2006
#####################################################################
# Comments: A simple program that prints a string message and the results
#       Two syscall one for the string and the other for the integer
#####################################################################
    .data           # data segment
str:
    .asciiz "the answer = "

    .text           # text segment
    .globl main

main:
    li $v0, 4       # 4 = print str syscall
    la $a0, str     # load address of string
    syscall         # execute the system call

    li $v0, 1       # 1 = print int syscall
    li $a0, 5       # load int value 5
    syscall         # execute the system call
```

Never Use GREEK characters

MIPS assembler directives

Appendix A Assemblers, Linkers, and the SPIM Simulator Page:A-48

SPIM supports a subset of the MIPS assembler directives:

`.data <addr>` Subsequent items are stored in the data segment. If the optional argument addr is present, subsequent items are stored starting at address addr.

`.text <addr>` Subsequent items are put in the user text segment. In SPIM, these items may only be instructions or words (see the .word directive below). If the optional argument addr is present, subsequent items are stored starting at address addr.

`.ascii str` Store the string str in memory, but do not nullterminate it.

`.asciiz str` Store the string str in memory and null-terminate it.

`.align n` Align the next datum on a 2n byte boundary. For example, .align 2 aligns the next value on a word boundary. .align 0 turns off automatic alignment of .half, .word, .float, and .double directives until the next .data or .kdata directive.

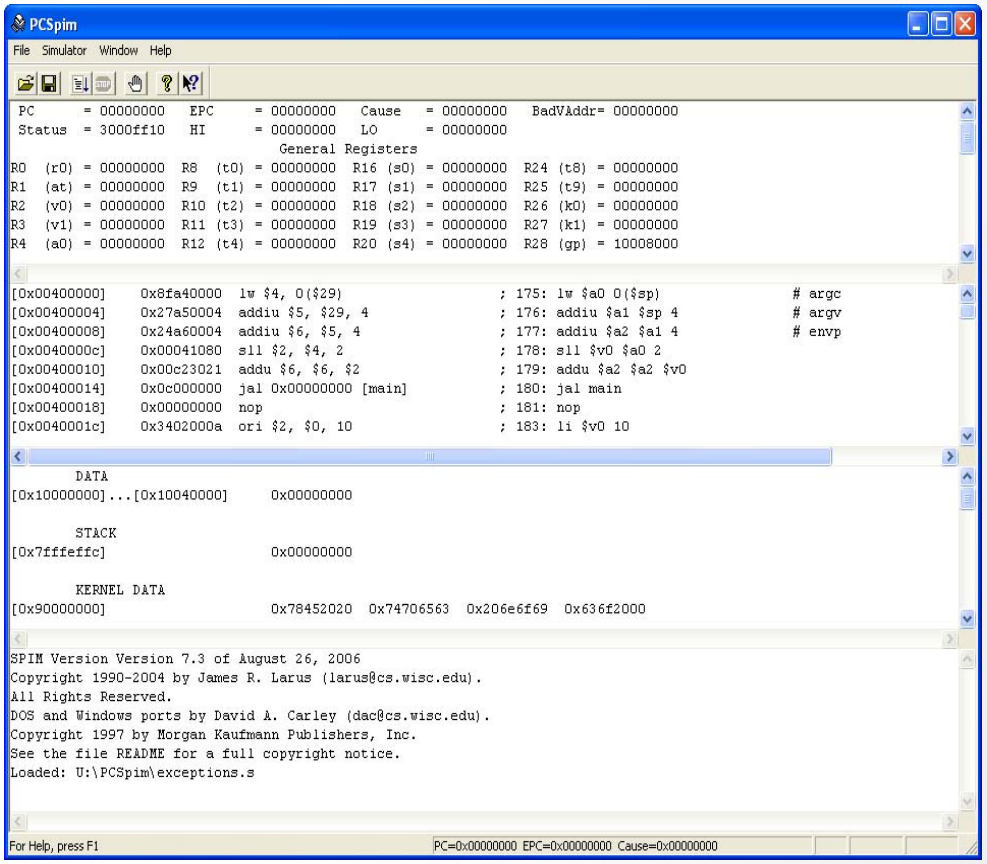
`.byte b1,..., bn` Store the n values in successive bytes of memory.

`.space n` Allocate n bytes of space in the current segment (which must be the data segment in SPIM).

PCSPIM

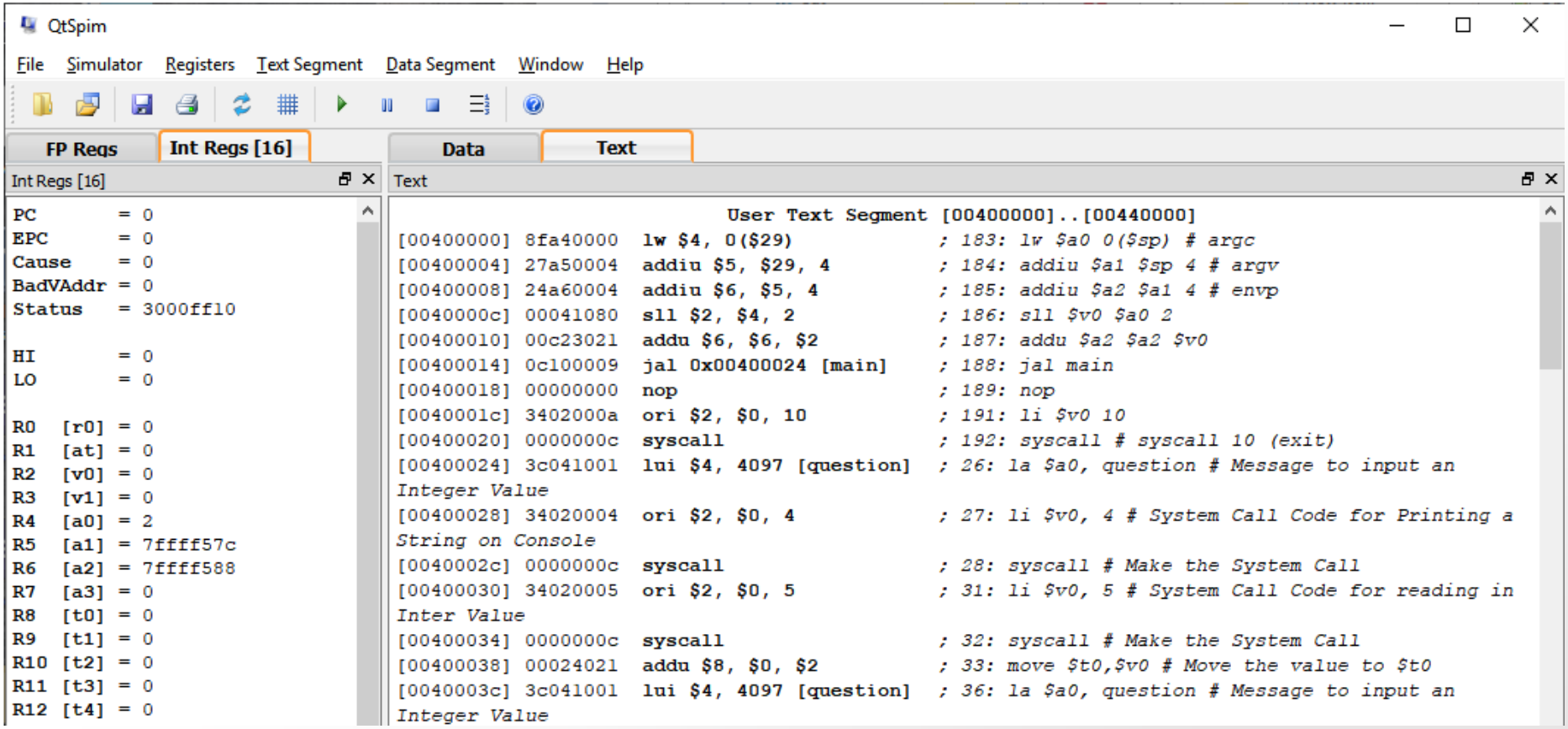
"**spim** is a self-contained simulator that will run MIPS32 assembly language programs."

OLD Version



- Το πρώτο παράθυρο παρουσιάζει τις τιμές όλων των Registers στο MIPS CPU και FPU. Ενημερώνεται όποτε το πρόγραμμά σας σταματά.
- Το δεύτερο παράθυρο παρουσιάζει τις εντολές από το πρόγραμμα σάς καθώς και εντολές από το σύστημα που αυτόματα έχει φορτωθεί όταν ξεκίνησε η εκτέλεση του προγράμματος σας. **
- Το τρίτο παράθυρο παρουσιάζει τα δεδομένα στην μνήμη και το Stack.
- Το τέταρτο παράθυρο παρουσιάζει διάφορα μηνύματα.

QtSpim



Never Use GREEK characters

```
      1          2          3          4
[0x00400000] 0x8fa40000 lw $4, 0($29)  ; 89: lw $a0, 0($sp)
```

1. Ο πρώτος αριθμός στη γραμμή, στα τετραγωνικά υποστηρίγματα, είναι η δεκαεξαδική διεύθυνση μνήμης της οδηγίας.

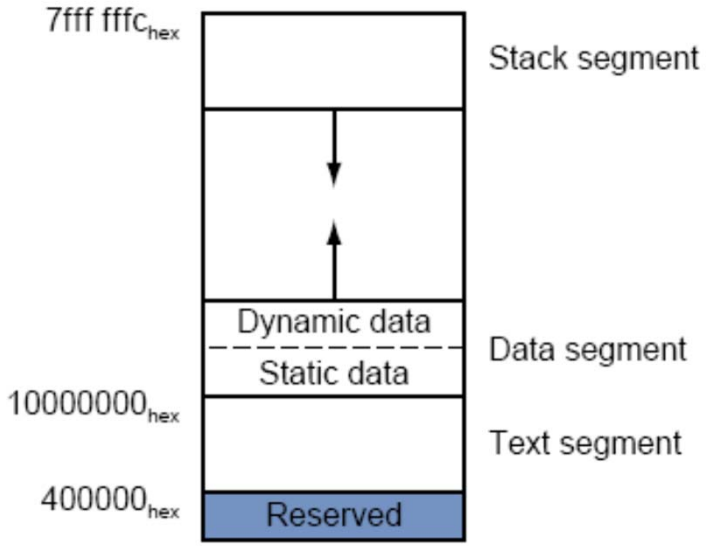
2. Ο δεύτερος αριθμός είναι η αριθμητική κωδικοποίηση της οδηγίας, ως δεκαεξαδικός αριθμός.

3. Το τρίτο στοιχείο είναι η μνημονική περιγραφή της οδηγίας. (Disassembly of 2)

Όλα μετά από την άνω τελεία είναι η πραγματική γραμμή από το αρχείο σας που παρήγαγε την οδηγία.

4. Ο αριθμός 89 είναι ο αριθμός γραμμών σε εκείνο το αρχείο. Μερικές φορές τίποτα δεν είναι στη γραμμή μετά από την άνω τελεία. Αυτό σημαίνει ότι η οδηγία παρήχθη από SPIM ως τμήμα της μετάφρασης ενός ***pseudoinstruction*** σε περισσότερους από ένα πραγματικό MIPS οδηγίας.

Σχεδιάγραμμα της μνήμης/Memory Layout

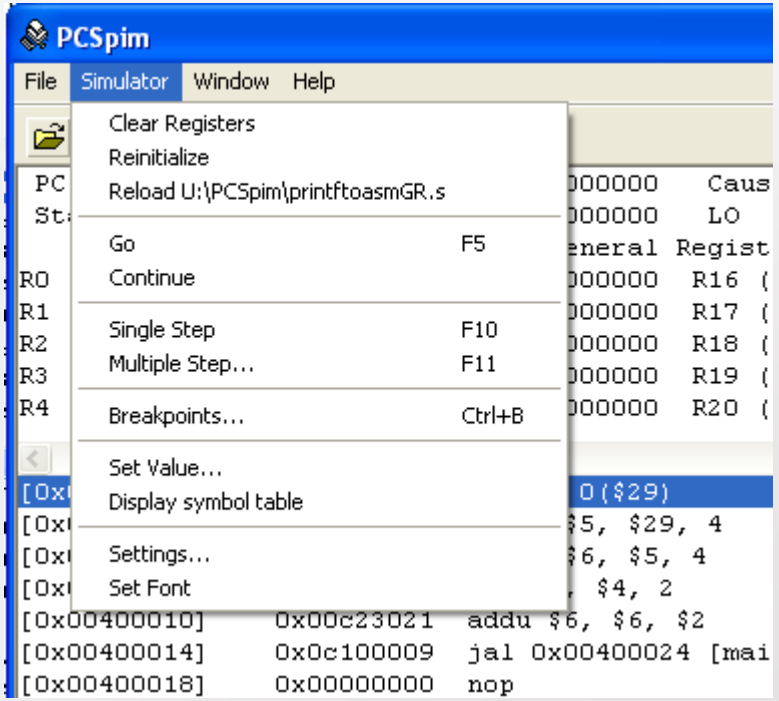


Η οργάνωση της μνήμης στα συστήματα MIPS είναι συμβατική. Το διάστημα διευθύνσεων ενός προγράμματος αποτελείται από τρία μέρη. Στο κατώτατο σημείο των διευθύνσεων του χρήστη (user address space) (**`0x400000`**) είναι το τμήμα κειμένων (***text segment***) το οποίο κρατά τις οδηγίες για ένα πρόγραμμα.

Επάνω από το text segment είναι το τμήμα στοιχείων (***data segment***) που αρχίζει από **`0x10000000`** και το όποιο διαιρείται σε δύο μέρη. Η στατική μερίδα στοιχείων (***static data portion***) περιέχει τα αντικείμενα το μέγεθος των οποίων και η διεύθυνση είναι γνωστή στον compiler και Linker. Αμέσως επάνω από αυτά τα αντικείμενα είναι δυναμικό στοιχείο (***dynamic data***). Δεδομένου ότι ένα πρόγραμμα διαθέτει το διάστημα δυναμικά (π.χ. από το malloc), η κλήση συστημάτων sbrk (sbrk system call) κινεί την κορυφή του τμήματος στοιχείων προς τα επάνω. Το ***program stack*** είναι στην κορυφή του διαστήματος διευθύνσεων (**`0x7fffffff`**). Αυξάνεται κάτω, προς τα κάτω δηλ. προς το τμήμα στοιχείων.

Εντολές PCSPIM

- Open: Επιλογή του πηγαίου κώδικα ενός προγράμματος
- Go (F5): Εκτέλεση του προγράμματος
- Single Step (F10):
- Breakpoints (Ctrl-B)
- Reload



PCSPIM System Calls

= Operating-System-like services

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	


```
move    $a0, $v0
li      $v0, 1          # print_int syscall
syscall
```

Άσκηση #1

Κατεβάστε το **`Homework1ENG.s`** από την ιστοσελίδα του μαθήματος και τρέξετε το στο περιβάλλον PCSPIM.

Κατανοήστε τι κάνει το πιο πάνω πρόγραμμα.

Να ετοιμάσετε ένα report βάση του παραδείγματος που υπάρχει στην ιστοσελίδα του μαθήματος που να περιγράφει την λειτουργία του προγράμματος.