Introduction into MIPS Assembly Language

Lab 12

See Appendix A of
Computer Organization and Design
http://pages.cs.wisc.edu/~larus/HP_AppA.pdf

MIPS Arithmetic - Registers

MIPS registers

Name	Number	Usage
`$zero`	0	constant 0
`$at`	1	reserved for assembler
`$v0 - $v1`	2-3	expression evaluation and function results
`$a0 - $a3`	4-7	arguments
`$t0 - $t7`	8-15	temporary, saved by caller
`$s0 - $s7`	16-23	temporary, saved by called function
`$t8 - $t9`	24-25	temporary, saved by caller
`$k0 - $k1`	26-27	reserved for kernel (OS)
`$gp`	28	points to middle of a 64K block in the data segment
`$sp`	29	stack pointer (top of stack)
`$fp`	30	frame pointer (beginning of current frame)
`$ra`	31	return address
`Hi, Lo`	-	store partial result of mult and div operations
`PC`	-	contains the address of the next instruction to be fetched (this is not a real MIPS register, and is only used to define instructions)
`status`	-	register 12 in coprocessor 0, stores interrupt mask and enable bits
`cause`	-	register 13 in coprocessor 0, stores exception type and pending interrupt bits
`epc`	-	register 14 in coprocessor 0, stores address of instruction causing exception

MIPS Arithmetic – Instruction Format

MIPS Instruction formats

Format	Bits 31-26	Bits 25-21	Bits 20-16	Bits 15-11	Bits 10-6	Bits 5-0
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	imm		
J	op	address				

pseudoinstructions, appear as real instructions in assembly language programs. The hardware, however, knows nothing about pseudoinstructions, so the assembler translates them into equivalent sequences of actual machine instructions.

Program sample with System Calls

```
#####################################################################
#Program Name     : Read three integers from the user and check there status
#Programmer       : Petros Panayi  Stud. ID:000000
#Date Last Modif.: 7 Sep 2006
#####################################################################
# Comments: This program requests three integers from the user and
#  prints on the console if they are positive, negative or zero
#####################################################################
    .data                   # data segment
question:          .asciiz "Please enter an Integer value:"
confimation_msg:   .asciiz "Your values are "
comma:             .asciiz ", "
#####################################################################
    .text                   # text segment
    .globl main
main:
    # διάβασε τρεις ακεραίους στο $t0, $t1 και $t2
    la    $a0, question     # Μήνυμα για είσοδο ακεραίου
    li    $v0, 4            # κώδικας συστήματος για τύπωση συμβολοσειράς
   syscall
.
.
    # 7.Exit
    li          $v0, 10      # exit system call
    syscall
```

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	

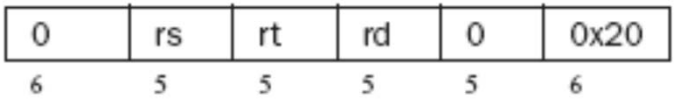
$v0

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

MIPS Arithmetic – Addition

Addition (with overflow)

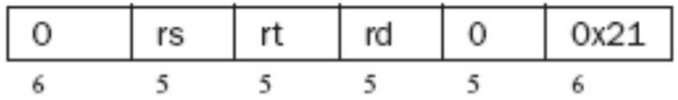
```
add rd, rs, rt
```




```
$rd = $rs + $rt
```

Addition (without overflow)

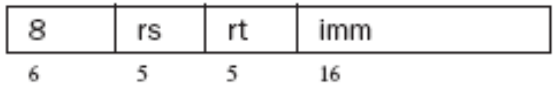
```
addu rd, rs, rt
```



Put the sum of registers `rs` and `rt` into register `rd`. Unsigned, no overflow

Addition immediate (with overflow)

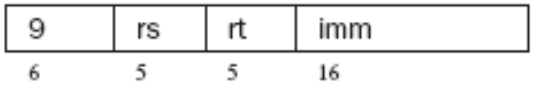
```
addi rt, rs, imm
```




```
$rt = $rs + imm
```

Addition immediate (without overflow)

```
addiu rt, rs, imm
```

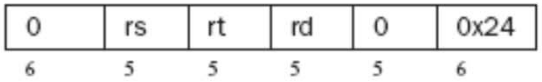


Put the sum of register `rs` and the sign-extended immediate into register `rt`.

MIPS Arithmetic – Logic Operations

AND

`and rd, rs, rt`

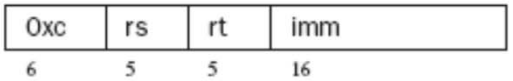


`$rd = $rs & $rt`

Put the logical AND of registers `rs` and `rt` into register `rd`.

AND immediate

`andi rt, rs, imm`

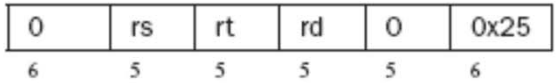


`$rt = $rs & imm`

Put the logical AND of register `rs` and the zero-extended immediate into register `rt`.

OR

`or rd, rs, rt`

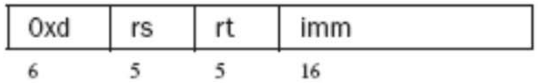


`$rd = $rs | $rt`

Put the logical OR of registers `rs` and `rt` into register `rd`.

OR immediate

`ori rt, rs, imm`



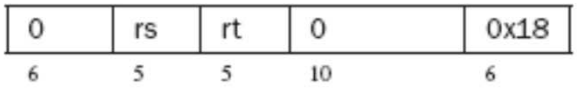
`$rt = $rs | imm`

Put the logical OR of register `rs` and the zero-extended immediate into register `rt`.

MIPS Arithmetic – Multiplication

Multiply

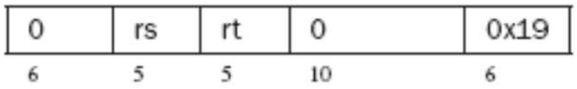
`mult rs, rt`



`Hi, Lo = $rs * $rt` 64 bit signed product in Hi and Lo

Unsigned multiply

`multu rs, rt`

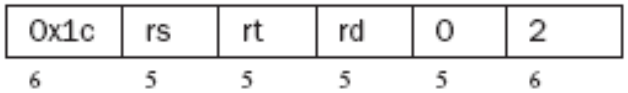


`Hi, Lo = $rs * $rt` 64 bit unsigned product in Hi and Lo

Multiply registers `rs` and `rt`. Leave the low-order word of the product in register `lo` and the high-order word in register `hi`.

Multiply (without overflow)

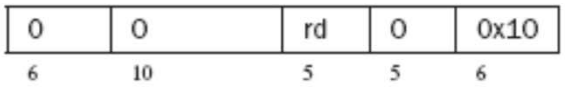
`mul rd, rs, rt`



Put the low-order 32 bits of the product of `rs` and `rt` into register `rd`.

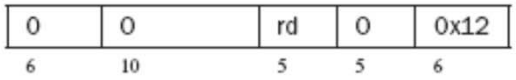
Move from hi

`mfhi rd`



Move from lo

`mflo rd`



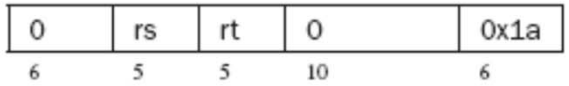
The multiply and divide unit produces its result in two additional registers, `hi` and `lo`. These instructions move values to and from these registers. The multiply, divide, and remainder pseudoinstructions that make this unit appear to operate on the general registers move the result after the computation finishes.

Move the `hi` (`lo`) register to register `rd`.

MIPS Arithmetic - Division

Divide (with overflow)

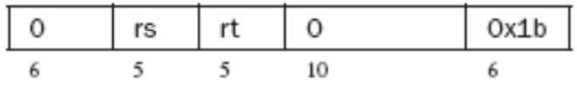
`div rs, rt`



Lo = $rs / $rt, Hi = $rs mod $rt

Divide (without overflow)

`divu rs, rt`



Lo = $rs / $rt, Hi = $rs mod $rt (unsigned)

Divide register `rs` by register `rt`. Leave the quotient in register `lo` and the remainder in register `hi`. Note that if an operand is negative, the remainder is unspecified by the MIPS architecture and depends on the convention of the machine on which SPIM is run.

Divide (with overflow)

`div rdest, rsrc1, src2` *pseudoinstruction*

Divide (without overflow)

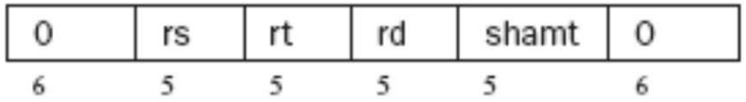
`divu rdest, rsrc1, src2` *pseudoinstruction*

Put the quotient of register `rsrc1` and `src2` into register `rdest`.

MIPS Shifting

Shift left logical

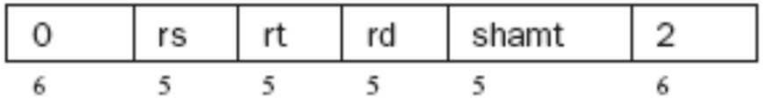
```
sll rd, rt, shamt
```



$rd = $rs << shamt

Shift right logical

```
srl rd, rt, shamt
```



$rd = $rs >> shamt

Rotate left

```
rol rdest, rsrc1, rsrc2        pseudoinstruction
```

Rotate right

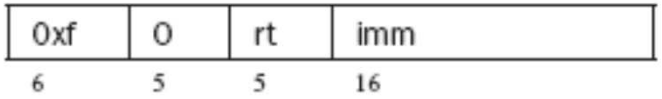
```
ror rdest, rsrc1, rsrc2        pseudoinstruction
```

Rotate register `rsrc1` left (right) by the distance indicated by `rsrc2` and put the result in register `rdest`.

MIPS Loading Data

Load upper immediate

`lui rt, imm`



$$\$rt = imm * 2^{16}$$

Load the lower halfword of the immediate `imm` into the upper halfword of register `rt`. The lower bits of the register are set to 0.

Load immediate

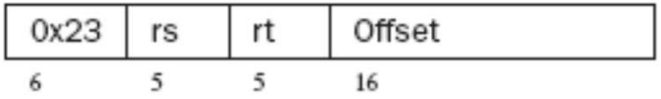
`li rdest, imm` *pseudoinstruction*

Move the immediate `imm` into register `rdest`.

MIPS Loading Data

Load word

```
lw rt, address
```



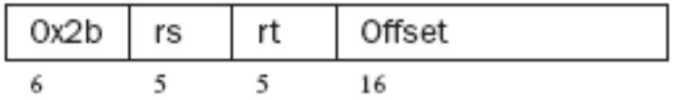

```
lw $rt, imm($rs)
```

```
$rt = M[$rs + imm]
```

Load the 32-bit quantity (word) at *address* into register `rt`.

Store word

```
sw rt, address
```




```
sw $rt, imm($rs)
```

```
M[$rs + imm] = $rt
```

Store the word from register `rt` at *address*.

MIPS Moving Data Between Registers

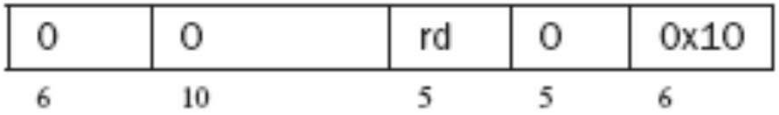
Move

```
move rdest, rsrc          pseudoinstruction
```

Move register `rsrc` to `rdest`.

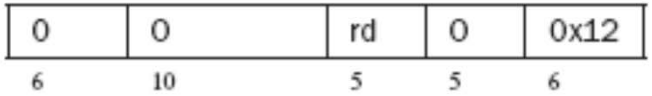
Move from hi

```
mfhi rd
```



Move from lo

```
mflo rd
```



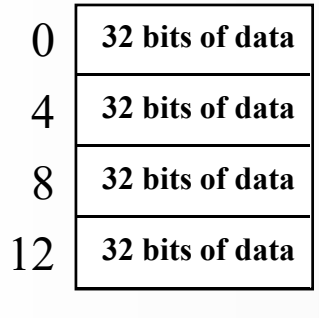
The multiply and divide unit produces its result in two additional registers, `hi` and `lo`. These instructions move values to and from these registers. The multiply, divide, and remainder pseudoinstructions that make this unit appear to operate on the general registers move the result after the computation finishes.

Move the `hi` (`lo`) register to register `rd`.

MIPS - Memory Organization

Bytes are nice, but most data items use larger "words"

For MIPS, a word is 32 bits or 4 bytes.



Registers hold 32 bits of data

2^{32} bytes with byte addresses from 0 to 2^{32}-1

2^{30} words with byte addresses 0, 4, 8, ... 2^{32}-4

Words are aligned

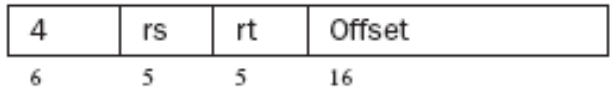
i.e., what are the least 2 significant bits of a word address?

MIPS Branching

Branch instructions use a signed 16-bit instruction *offset* field; hence they can jump $2^{15} - 1$ *instructions* (not bytes) forward or 2^{15} instructions backwards. The *jump* instruction contains a 26-bit address field.

Branch on equal

`beq rs, rt, label`

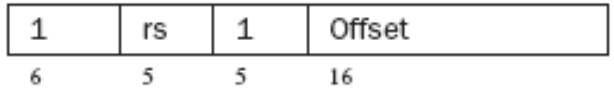


`if($rs==$rt)` `PC = PC +` *`imm`* (PC always points to next instruction)

Conditionally branch the number of instructions specified by the offset if register `rs` equals `rt`.

Branch on greater than equal zero

`bgez rs, label`

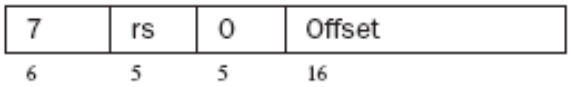


Conditionally branch the number of instructions specified by the offset if register `rs` is greater than or equal to 0.

MIPS Branching

Branch on greater than zero

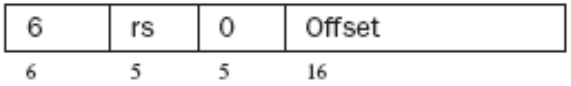
`bgtz rs, label`



Conditionally branch the number of instructions specified by the offset if register `rs` is greater than 0.

Branch on less than equal zero

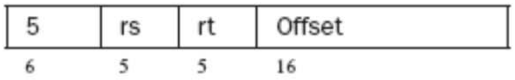
`blez rs, label`



Conditionally branch the number of instructions specified by the offset if register `rs` is less than or equal to 0.

Branch on not equal

`bne rs, rt, label`



Conditionally branch the number of instructions specified by the offset if register `rs` is not equal to `rt`.

`if($rs!=$rt) PC = PC + imm` (PC always points to next instruction)

MIPS Jump

Jump

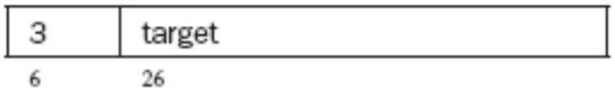
`j target`

PC = target*4 or target <<2

Unconditionally jump to the instruction at target.

Jump and link

`jal target`



`$ra = PC; PC = address*4` (Jump and link, *address = destination/4*)

Unconditionally jump to the instruction at target. Save the address of the next instruction in register `$ra`.

Jump register

`jr rs`

`PC = $rs`

Unconditionally jump to the instruction whose address is in register `rs`.

The last two are used mainly in procedures call

Άσκηση 1:

- Γράψετε ένα απλό πρόγραμμα που να διαβάζει από το πληκτρολόγιο ένα ακέραιο αριθμό Ν και να εκτυπώνει στην οθόνη το όνομα σας Ν φορές.

.data Segment

```
1  ###################################################################
2  #Program Name        : Print your name from 1 to n times
3  #Programmer          : Petros Panayi ID:000000
4  #Date Last Modif.    : 13 Sep 2016
5  ###################################################################
6  # Comments:          Print your name from 1 to n times
7  ###################################################################
8
9      .data        # data segment
10 questionStr:     .asciiz "Give an integer number between 1 and 100: "
11 myNameStr:       .asciiz "Petros Panayi\n"
12 ###################################################################
```

.text Segment

```
13      .text           # text segment
14      .globl main
15  main:
16      # Initiate the values used in the program
17      li $t0, 0       # set the counting value equal to 0
18      li $t1, 0       # Set the value for the sum
19
20
21      li $v0, 4               # 4 = print questionStr syscall
22      la $a0, questionStr     # load address of string
23      syscall                 # execute the system call
24
25
26      li  $v0, 5          # 5 = read_int
27      syscall             # syscall
28      move    $s0,$v0     # Save the number in s0
29  #---------------------------------------------------------------
30  loop:   addi $t0, $t0, 1    # $t0++
31          li $v0, 4           # 4 = print myNameStr syscall
32          la $a0, myNameStr # load address of string
33          syscall             # execute the system call
34          blt $t0, $s0, loop  # if ($t0 < $s0)
35                              # then goto loop
36  #---------------------------------------------------------------
37
38      # Exit
39      li          $v0, 10       # Call Exit system call)
40      syscall
```

Άσκηση 2:
MIPS – Loop Example - Άθροισμα Τετραγώνων

Σας δίνετε ένα πρόγραμμα το οποίο υπολογίζει το $Y = \sum_{\chi=0}^{\chi=N} \chi^2$

Αντιγράψτε το πρόγραμμα και τρέξτε το στο QtSPIM
Υπολογίστε τον αριθμών των εντολών που θα εκτελεστούν σε αυτό το πρόγραμμα.
~~Πιο το CPI του προγράμματος αν το mul χρειάζεται 5 κύκλους μηχανής για να εκτελεστεί;~~
~~Πιο το CPI του προγράμματος αν το mul χρειάζεται 1 κύκλους μηχανής για να εκτελεστεί;~~

MIPS – Loop Example 1/2

```
    .data        # data segment
questionStr:    .asciiz "Give an integer number between 1 and 100: "
answerStr:      .asciiz "The sum of the Squares is "
newLine:        .asciiz "\n"
#################################################################
    .text            # text segment
    .globl main
main:
    # Initiate the values used in the program
    li $t0, 0        # set the counting value equal to 0
    li $t1, 0        # Set the value for the sum


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


    li  $v0, 5       # 5 = read_int
    syscall          # syscall
    move    $s0,$v0      # Save the number in s0
```

MIPS – Loop Example 2/2

```
#-------------------------------------------------------------
loop:   addi $t0, $t0, 1     # $t0++
        mul $t2, $t0, $t0    # Multiply counter value by its self
                             # and save the square value in $t2
        add $t1, $t1, $t2    # $t1 += $t2
        blt $t0, $s0, loop   # if ($t0 <= $s0)
                             # then goto loop
#-------------------------------------------------------------
    li $v0, 4              # 4 = print answerStr syscall
    la $a0, answerStr      # load address of string
    syscall                # execute the system call

    move $a0, $t1          # Move the result to $a0
    li $v0, 1              # 1 = print_int
    syscall                # print int

    # Exit
    li          $v0, 10        # Call Exit system call)
    syscall
```

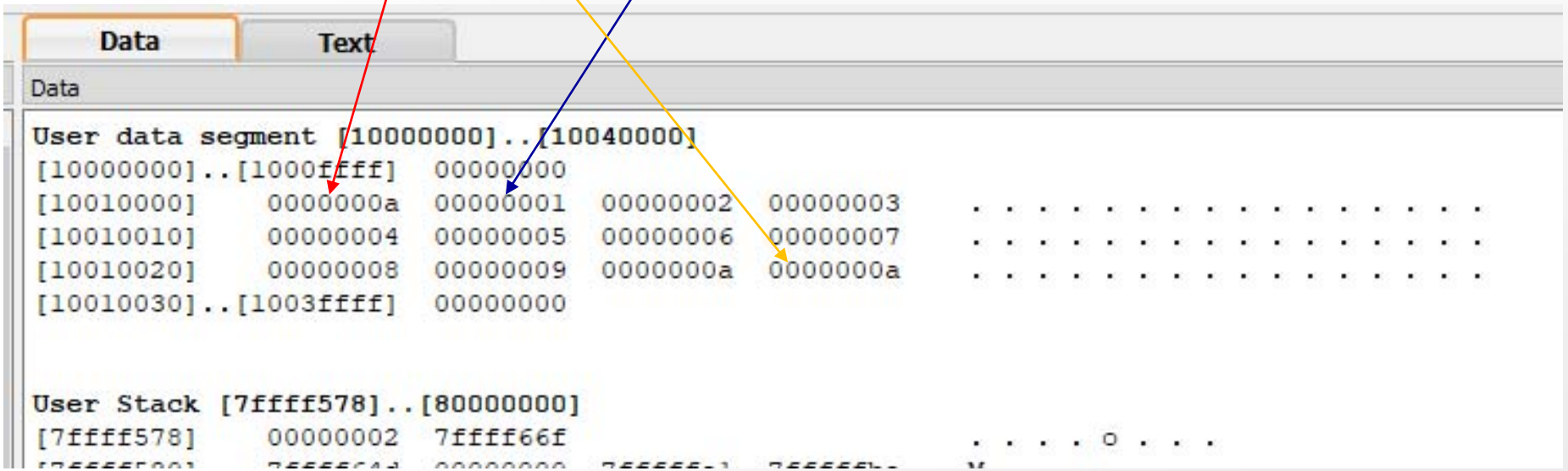
Άσκηση 3: MIPS – Loop Exercise

Συμπληρώστε τον κώδικα στο **Lab9Example3_std.s** ώστε να διπλασιάζονται οι τιμές του πίνακα **table** και να τυπώνονται στο console.

~~Πιο το CPI του προγράμματος αν το lw χρειάζεται 5 κύκλους μηχανής για να εκτελεστεί;~~
~~Πιο το CPI του προγράμματος αν το lw χρειάζεται 1 κύκλους μηχανής για να εκτελεστεί;~~

Άσκηση 3: MIPS – Loop Exercise

```
    .data          # data segment
size:       .byte   10                  #The size of the table
table:      .word   1 2 3 4 5 6 7 8 9 10   #The table of 10 words
newLine:    .asciiz "\n"                #New line
```



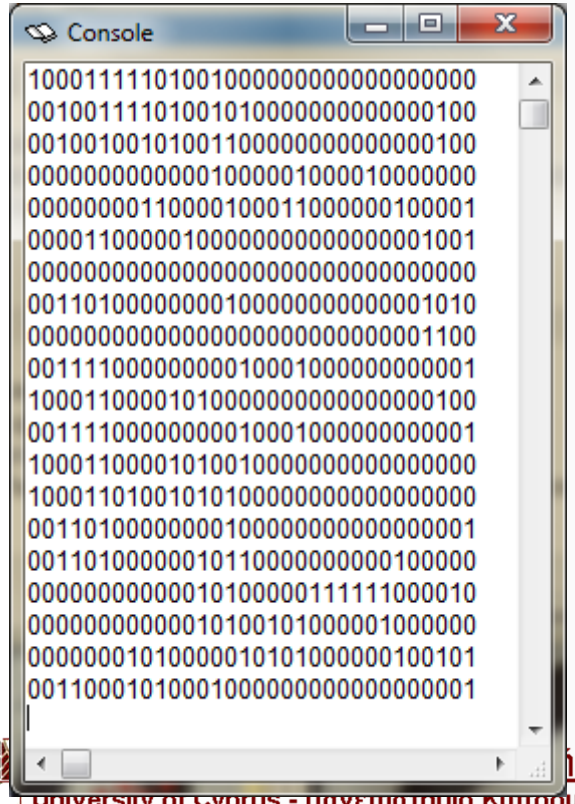
```
10      .data        # data segment
11  size:       .byte   10                          #The size of the table
12  table:      .word   1 2 3 4 5 6 7 8 9 10       #The table of 10 words
13  newLine:    .asciiz "\n"                        #New line
14  ##################################################################
15      .text            # text segment
16      .globl main
17  main:
18      # Initiate the values used in the program
19      li $t0, 0        # set the counting value equal to 0
20      lb $t1, size     #Load the size of the table
21      la $t2, table    #Load the start address of the table
22      li $v0, 1        #System Call print_int
23  #-----------------------------------------------------------------
24  loop:    #?#Load word from table
25           #?#Print on Console
26           #?#Double the value
27           #?#Print on Console
28           #?#Store new value back to the table
29           #?#Move the pointer to the next word
30           #?#Increase the counter
31           #?#Repeat until all words read
32  #-----------------------------------------------------------------
33
34
35      # Έξοδος
36      li  $v0, 10        #  Exit System Call
37      syscall
```

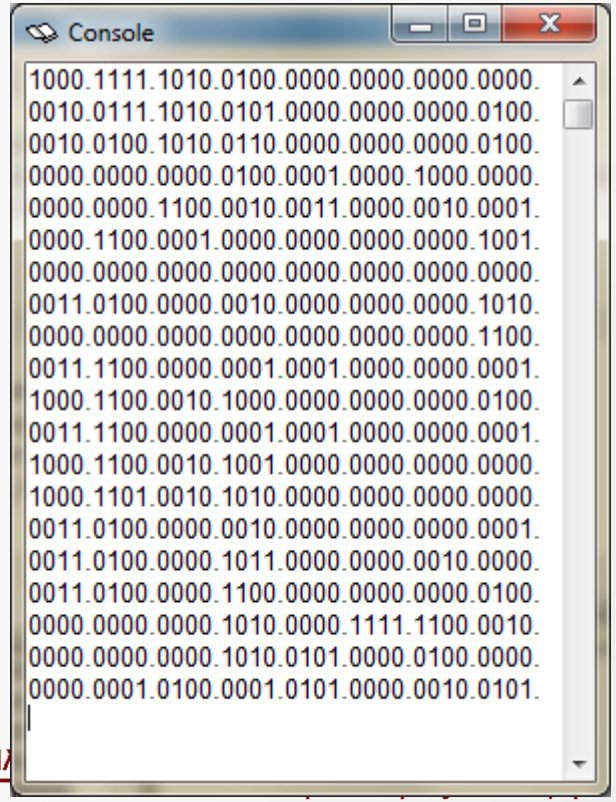
Load `word` from table

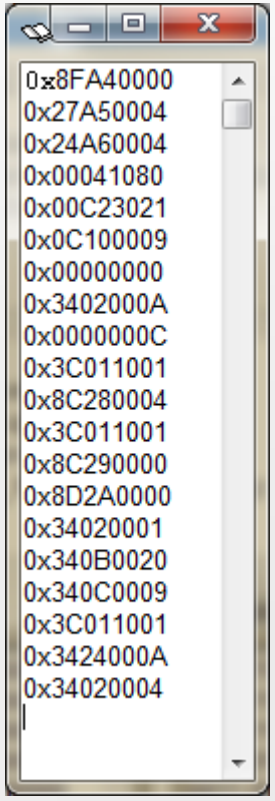
```
26  loop:   lw $a0, 0($t2)      #Load word from table
27          move $s0, $a0       #Store the value in $s0
28          li $v0, 1           #System Call print_int
29          syscall             #Print on Console
```

Άσκηση 4:
MIPS – Print .text Segment Example

1. Αντιγράψτε το πρόγραμμα που τυπώνει τις πρώτες 20 εντολές από το .text segment και τις τυπώνει σε δυαδική μορφή, τρέξτε το στο PCSPIM
2. Τροποποιήστε το πιο πάνω κώδικα ώστε να τυπώνεται μία τελεία για κάθε 4 δυαδικά ψηφία
3. Τροποποιήστε το πιο πάνω πρόγραμμα ώστε να τυπώνει τις εντολές του .text segment σε δεκαεξαρική μορφή.







Lab10Example4_1.s

```
1  ##################################################################
2  #Program Name        : A simple program that reads the 20 first lines from
3  #                       the .text and prints them on screen in binary format
4  #Programmer          : Petros Panayi Stud. ID:000000
5  #Date Last Modif.    : 20 March 2018
6  ##################################################################
7  # Comments: A simple program that reads the 20 first lines from
8  #           the .text and prints them on screen in binary format
9  ##################################################################
10     .data                    # data segment
11 codeAddress:
12     .word   0x00400000       # The start of the text segment
13 codeNumber:
14     .word   20               # the number of code to store
15 newLine:
16     .asciiz "\n"             # New line
17
18
19     .text                    # text segment
20     .align 2
21     .globl main
22 main:
23     lw   $t0, codeNumber     # Load the number of code into $t0  => $t0 = 20
24     lw   $t1, codeAddress
```

Lab10Example4_1.s

```
25
26  # Read the instructions from the .text segment
27  Loop1:
28      lw   $t2, 0($t1)              # load into $t2 the first address of .text <--------
29  # The following code will print the 32 bits on the console                            |
30      li $v0, 1                     # 1 = print int syscall                             |
31      li   $t3, 32                  # bitCounter = 32                                   |
32      Loop2:  rol $t2, $t2, 1       # rotate to LEFT by one position        <---|       |
33              and $a0, $t2, 1           # mask with 0x00000001 and store in $a0 |       |
34              syscall                   # execute the system call               ^       |
35              addi $t3, -1              # bitCounter--                          |       |
36              bgtz $t3, Loop2           # ----------->------------>-------------       |
37                                  #                                                   |
38      addi $t1, 4                 # $t1 = $t1 + 4     => $t1 = 0x00400004             ^
39      add  $t0, $t0, -1           # $t0 = $t0 - 1     => $t0 = 19                     |
40      la  $a0, newLine      # Print a new line                                        |
41      li  $v0, 4            #                                                         |
42      syscall               #                                                         |
43      bnez $t0, Loop1       # ------------>-----------------------------------------><-
44
45  # 7. Exit
46      li          $v0, 10         # Exit system call
47      syscall
```