Introduction into MIPS Assembly Language
Dr. Petros Panayi
(Functions and Stack)

Εργαστήριο 13

Πέτρος Παναγή, PhD

MIPS jal and jr

Jump

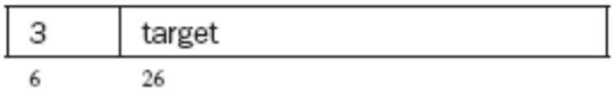
`j target`

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

MIPS – Procedures (ΟΧΙ ΓΙΑ ΤΟ EPL121)

register-use convention Also called **procedure call convention**. A software protocol governing the use of registers by procedures.

Registers `$a0`–`$a3` (4–7) are used to pass the first four arguments to routines (remaining arguments are passed on the stack). Registers `$v0` and `$v1` (2, 3) are used to return values from functions.

Registers `$t0`–`$t9` (8–15, 24, 25) are **caller-saved registers** that are used to hold temporary quantities that need not be preserved across calls

Registers `$s0`–`$s7` (16–23) are **callee-saved registers** that hold long-lived values that should be preserved across calls.

Register `$gp` (28) is a global pointer that points to the middle of a 64K block of memory in the static data segment.

Register `$sp` (29) is the stack pointer, which points to the last location on the stack. Register `$fp` (30) is the frame pointer. The `jal` instruction writes register `$ra` (31), the return address from a procedure call. These two registers are explained in the next section.

MIPS - Procedures

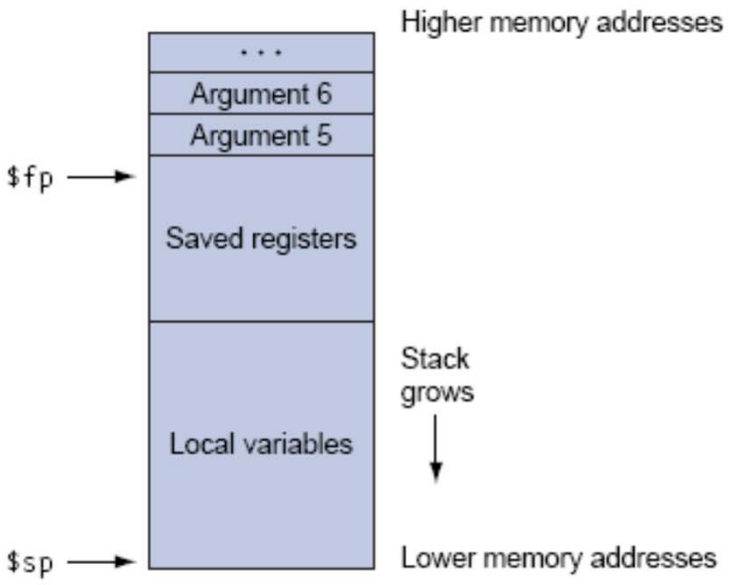
Before a called routine starts running, it must take the following steps to set up its stack frame:

1. Allocate memory for the frame by subtracting the frame's size from the stack pointer.
2. Save callee-saved registers in the frame. A callee must save the values in these registers (`$s0`–`$s7`, `$fp`, and `$ra`) before altering them since the caller expects to find these registers unchanged after the call. Register `$fp` is saved by every procedure that allocates a new stack frame. However, register `$ra` only needs to be saved if the callee itself makes a call. The other callee-saved registers that are used also must be saved.
3. Establish the frame pointer by adding the stack frame's size minus 4 to `$sp` and storing the sum in register `$fp`.

caller-saved register A register saved by the routine being called.

callee-saved register A register saved by the routine making a procedure call.

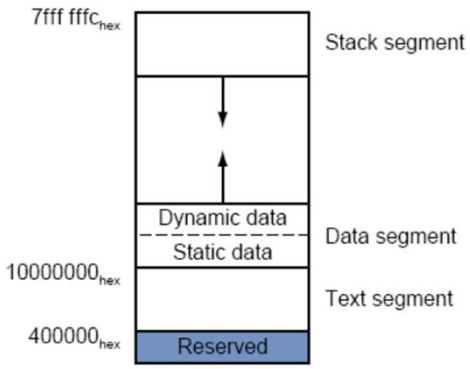
procedure call frame A block of memory that is used to hold values passed to a procedure as arguments, to save registers that a procedure may modify but that the procedure's caller does not want changed, and to provide space for variables local to a procedure.

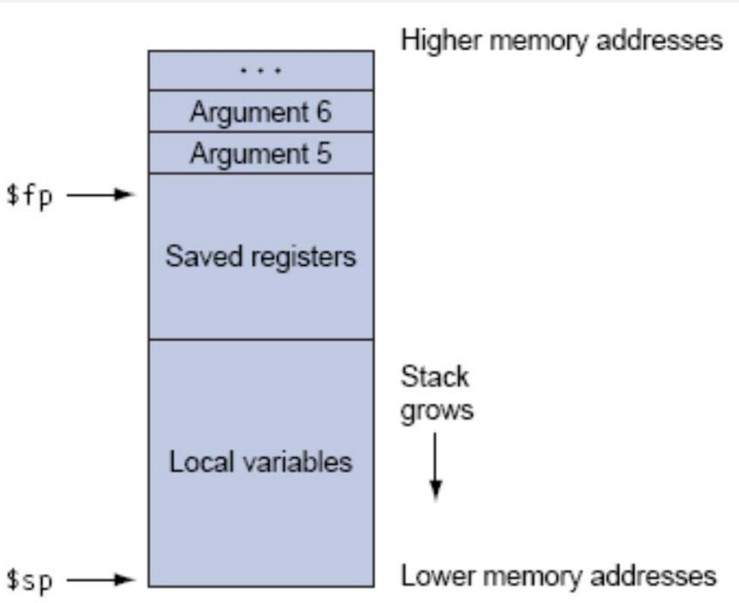


MIPS - Procedures

The **stack frame** consists of the memory between the frame pointer ($fp), which points to the first word of the frame, and the stack pointer ($sp), which points to the last word of the frame. The stack grows down from higher memory addresses, so the frame pointer points above the stack pointer.
The executing procedure uses the frame pointer to quickly access values in its stack frame.

procedure call frame A block of memory that is used to hold values passed to a procedure as arguments, to save registers that a procedure may modify but that the procedure's caller does not want changed, and to provide space for variables local to a procedure.





Κανόνας/Τυποποίηση για ΕΠΛ121

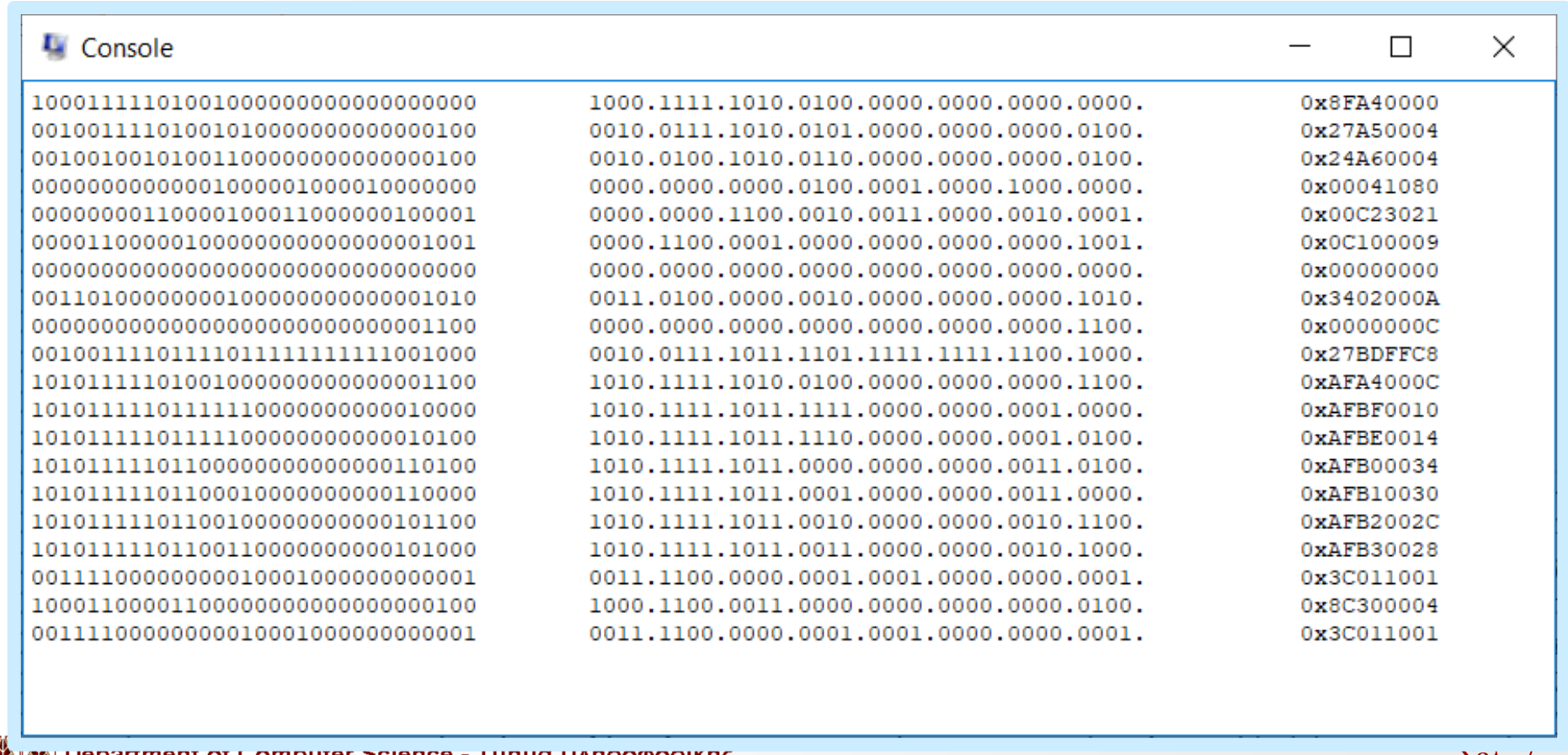
Όλοι οι καταχωρητές που χρησιμοποιούνται σε μία συνάρτηση θα πρέπει να αποθηκεύονται πρώτα στην στοίβα και με το πέρας της συνάρτησης οι τιμές τους να υποκαθιστούνται.

Εξαίρεση οι καταχωρητές $v0 και $v1

Άσκηση Εργαστηρίου

1. Τροποποιήστε το πρόγραμμα της περασμένης εβδομάδας ώστε να τυπώνει τις τρείς μορφές ενός word χρησιμοποιώντας συναρτήσεις όπως φαίνεται πιο κάτω.

(Θα πρέπει να υλοποιήσετε 3 διαφορετικές συναρτήσεις. Η κάθε συνάρτηση να τυπώνει ένα word ($a0).)



.text

```
10  ######## Stack Frame  Template #######
11  #Old $sp |-----------|
12  #        |   $s0     |
13  #$fp-->  |-----------|
14  #        |   $s1     |
15  #$sp+48->|-----------|
16  #        |   $s2     |
17  #$sp+44->|-----------|
18  #        |   $s3     |
19  #$sp+40->|-----------|
20  #        |   $s4     |
21  #$sp+36->|-----------|
22  #        |   $s5     |
23  #$sp+32->|-----------|
24  #        |   $s6     |
25  #$sp+28->|-----------|
26  #        |   $s7     |
27  #$sp+24->|-----------|
28  #        |   $fp     |
29  #$sp+20->|-----------|
30  #        |   $ra     |
31  #$sp+16->|-----------|
32  #        |   $a0     |
33  #$sp+12->|-----------|
34  #        |   $a1     |
35  #$sp+8-> |-----------|
36  #        |   $a2     |
37  #$sp+4-> |-----------|
38  #        |   $a3     |
39  #$sp --> |-----------|
40  #
41  ####S.O.S. we do not store V registers ###################################
42      .data                   # data segment
43  codeAddress:    .word   0x00400000      # The start of the text segment
44  codeNumber:     .word   20              # the number of code to store
45  newLine:        .asciiz "\n"            # New line
46  dot:            .asciiz "."             # The Dot
47  hexCode:        .asciiz " 0x"           # The Dot
48  tab:            .asciiz "\t"            # The Dot
    ------------------------------------------------------------------------
```

main

```
49  ###################################################################################
50      .text                       # text segment
51      .globl main
52  main:
53      # We have to save all the registers used into stack
54      subu $sp,$sp,56     # Stack frame is 56 bytes long
55      sw $a0, 12($sp)     # Save Argument 0 ($a0)
56      sw $ra, 16($sp)     # Save return address
57      sw $fp, 20($sp)     # Save frame pointer
58      sw $s0, 52($sp)     # Save $s0 in stack
59      sw $s1, 48($sp)     # Save $s1 in stack
60      sw $s2, 44($sp)     # Save $s2 in stack
61      sw $s3, 40($sp)     # Save $s3 in stack
62  #- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
63      lw   $s0, codeNumber    # Load the number of code into $s0  => $s0 = 20
64      lw   $s1, codeAddress
```

```
66  # Read the instructions from the .text segment
67  Loop1:
68      lw   $s3, 0($s1)         # load into $t2 the first address of .text  <-----------
69      move $a0, $s3            #                                                        |
70      jal PrintBinary          # Call Function PrintBinary                              |
71      la  $a0, tab             # Print a new line                                       |
72      li  $v0, 4               #                                                        |
73      syscall                  #                                                        ^
74                               #                                                        |
75      move $a0, $s3            #                                                        |
76      jal PrintBinaryDot       # Call Function PrintBinaryDot                           |
77      la  $a0, tab             # Print a new line                                       |
78      li  $v0, 4               #                                                        |
79      syscall                  #                                                        ^
80                               #                                                        |
81      move $a0, $s3            #                                                        |
82      jal PrintHex             # Call Function PrintHex                                 |
83      addi $s1,$s1, 4          # $s1 = $s1 + 4     => $s1 = 0x00400004                  ^
84      addi $s0, $s0, -1        # $s0 = $s0 - 1     => $s0 = 19                          |
85      la  $a0, newLine     # Print a new line                                           |
86      li  $v0, 4           #                                                            |
87      syscall              #                                                            |
88      bnez $s0, Loop1      # ------------>-----------------------------------------><---
89  #- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
90      # Exit: Restore all the registers used.
91      lw $a0, 12($sp)      # Restore old value of Argument 0 ($a0)
92      lw $ra, 16($sp)      # Restore old value of  return address
93      lw $fp, 20($sp)      # Restore old value of  frame pointer
94      lw $s0, 52($sp)      # Restore old value of  $s0 in stack
95      lw $s1, 48($sp)      # Restore old value of  $s1 in stack
96      lw $s2, 44($sp)      # Restore old value of  $s2 in stack
97      lw $s3, 40($sp)      # Restore old value of  $s3 in stack
98      addu $sp,$sp,56      # Pop stack 56 bytes long
99
100     jr $ra
```