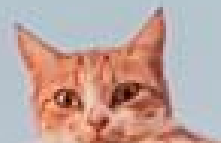


Computer Organization

Lab5 MIPS(4) - macro,procedure,memory

Topics

- **Macro** vs **Procedure**
- **Directive**
  - **.globl** vs **.extern**
  - **.globl main**
- **Memory**
  - **Local label** vs **Globl label**
  - **Static Storage** vs **Dynamic Storage**

Macro

Macros:

A **pattern-matching** and **replacement** facility that provide a simple mechanism to name a frequently used sequence of instructions.

- **Programmer invokes** the macro.
- **Assembler replaces** the macro call with the corresponding sequence of instructions.

Macros vs Subroutines:

- **Same:** permit a programmer to create and name a new abstraction for a common operation.
- **Difference:** Unlike subroutines, macros do not cause a subroutine call and return when the program runs since a macro call is replaced by the macro's body when the program is assembled.

Demo #1

```
.text
print_string:
        addi $sp,$sp,-4
        sw $v0,($sp)

        li $v0,4
        syscall

        lw $v0,($sp)
        addi $sp,$sp,4

        jr $ra
```

Assembler replaces the macro call with the corresponding sequence of instructions.

Q1: What's the difference between macro and procedue?

Q2: While save the macro's defination(on the right hand in this slides) in an asm file, and assamble it,
what's the assembly result?
Is the macro's defination file runable?

Q3: While save the procedure's defination(on the left hand in this slides) in an asm file, and assemble it,
what's the assembly result?
Is this file runable?

```
.macro print_string(%str)
.data
        pstr:   .asciiz   %str
.text
        addi $sp,$sp,-8
        sw $a0,4($sp)
        sw $v0,($sp)

        la $a0,pstr
        li $v0,4
        syscall

        lw $v0,($sp)
        lw $a0,4($sp)
        addi $sp,$sp,8
.end_macro
```

Procedure(1)

- In **CALLER**
  - **Before call the callee**
    - **Save caller-saved registers**
      - The called procedure can use these **registers(**$a0-$a3 **and** $t0-$t9**)** without first saving their value.
      - If the **caller** expects to use one of these registers after a call, it must **save** its value **before the call**.
    - **Pass arguments**
      - By convention, the first four arguments are passed in **registers** $a0-$a3. Any remaining arguments are pushed on the stack and appear at the beginning of the called procedure’ s stack **frame**.
  - **Execute a jal instruction**
    - **Saves** the return address in **register** $ra and **jumps** to **the callee’ s first instruction**.

Procedure(2)

- In **CALLEE(1)**
  - 1. **Allocate memory** for the stack by substracting the frame’ s size from the stack pointer(**$sp**).
  - 2. **Save callee-saved registers** in the frame.
    - A callee must **save** the values in these registers(**$s0-$s7 and $ra**) before altering them, since the caller expects to find these registers unchanged after the call.
    - **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.

Procedure(3)

In **CALLEE(2)**

- 3. **before returning to caller**
  - If the callee is a function that **returns value(s)**, place the returned value(s) in register **$v0~$v1**
  - **Restore all callee-saved registers** that were saved upon procedure entry
  - **Pop** the stack frame by adding the frame size to **$sp**
- 4. **Return** by jumping to the address in register **$ra**

Local label vs External label

- **Local label**
  - A label referring to an object that **can be used only within the file in which it is defined**.
- **External label**
  - A label referring to an object that **can be referenced from files other than the one in which it is defined**.

Find the usage of ".external" and ".globl" on page 10 and 11:
What's the relationship between globl main and the entrance of program?
What will happen if an external data have the same name with a local data?

Demo #2-1

```
it's in print callee.it's the default_str
it's in print caller.it's the default_str
```

Q1. Is the running result same as the sample snap?
Q2. How many "default_str" are defined in "lab5_print_callee.asm" ?
Q3. While executing the instruction "la $a0,default_str" in these two files, which "default_str" is used?

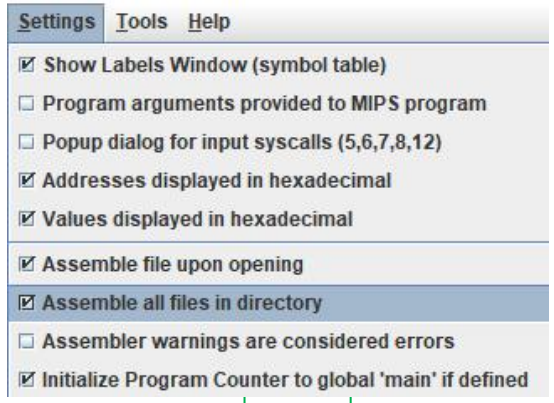
```
## "lab5_print_caller.asm" ##
.include "lab5_print_callee.asm"
.data
   str_caller:      .asciiz "it's in print caller."
.text
.globl main
main:
       jal print_callee

       addi $v0,$zero,4
       la  $a0,str_caller
       syscall
       la $a0,default_str   ###which one?
       syscall

       li $v0,10
       syscall
```

```
## "lab5_print_callee.asm" ##
.extern default_str 20
.data
       default_str:      .asciiz  "it's the default_str\n"
       str_callee:       .asciiz  "it's in print callee."
.text
print_callee:     addi $sp,$sp,-4
                  sw $v0,($sp)

                  addi $v0,$zero,4
                  la  $a0,str_callee
                  syscall
                  la $a0,default_str    ###which one?
                  syscall

                  lw $v0,($sp)
                  addi $sp,$sp,4
                  jr $ra
```

Demo #2-2

In Mars, set "Assemble all files in directory", put the following files in the same directory, then run it and answer the questions on page 10.

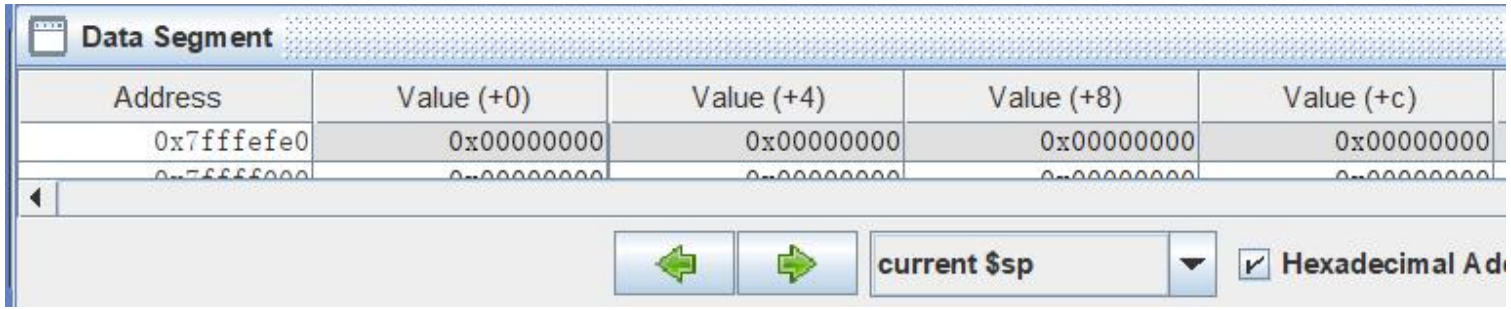
```
.data
   str_caller:      .asciiz      "it's in print caller."
.text
.globl  main
main:
        jal print_callee

        addi $v0,$0,0x0a636261
        sw $v0,defaulte_str

        addi $v0,$zero,4
        la  $a0,str_caller
        syscall
        la $a0,defaulte_str
        syscall

        li $v0,10
        syscall
```

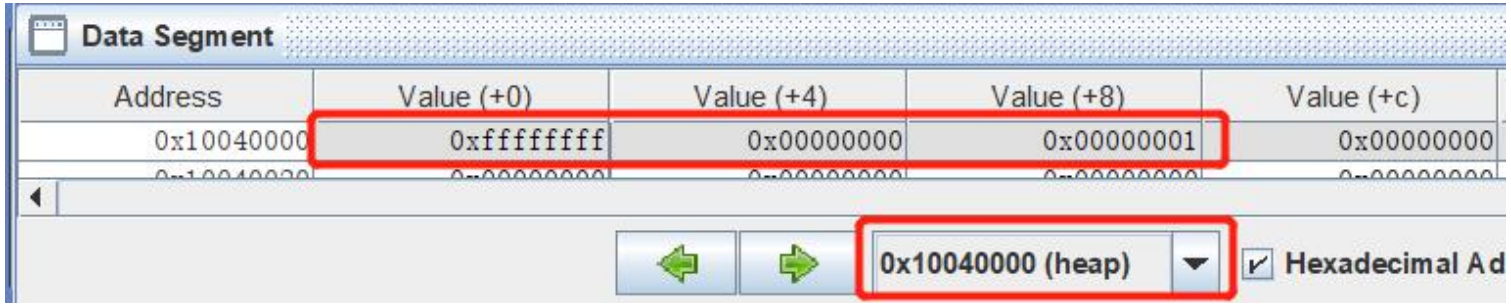


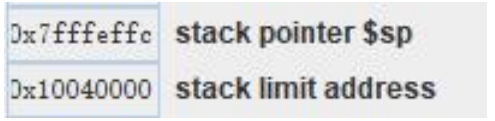

```
.data
        .extern      defaulte_str   20
        str_callee:          .asciiz       "it's in print callee."
        defaulte_str:      .asciiz       "ABC\n"
.text
.globl  print_callee
print_callee:     addi $sp,$sp,-4
                  sw $v0,($sp)

                  addi $v0,$zero,4
                  la  $a0,str_callee
                  syscall
                  la $a0,defaulte_str
                  syscall

                  lw $v0,($sp)
                  addi $sp,$sp,4
                  jr $ra
```

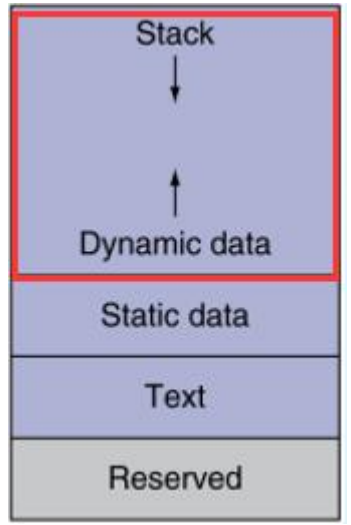
Stack vs Heap

- **Stack**: used to store the local variable, usually used in calle.
- **Heap**: The heap is reserved for **sbrk** and **break** system calls, and it not always present.









Demo #3-1 (1)

The following demo(composed of 4 pieces on page 12 and 13) is supposed to get and store the data from input device, get the minimal value among the data, the number of input data is determined by user.

```
.include "macro_print_str.asm"                    #piece 1/4
.data
        min_value: .word 0
.text
        print_string("please input the number:")

        li $v0,5            #read an integer
        syscall
        move $s0,$v0        #s0 is the number of integers

        sll $a0,$s0,2       #new a heap with 4*$s0
        li $v0,9
        syscall
        move $s1,$v0        #$s1 is the start of the heap
        move $s2,$v0        #$s2 is the point
```

```
print_string("please input the array\n")    #piece 2/4

add $t0,$0,$0

loop_read:

        li $v0,5        #read the array
        syscall
        sw $v0,($s2)

        addi $s2,$s2,4
        addi $t0,$t0,1
        bne $t0,$s0,loop_read
```

While the 1st input number is 0 or 1,what will happen? why?
Modify this demo to make it better

Demo #3-1 (2)

```
#piece 3/4
        lw $t0,($s1)        #initialize the min_value
        sw $t0,min_value
        li $t0,1
        addi $s2,$s1,4    #$s1 is the start of the heap

loop_find_min:
        lw $a0,min_value
        lw $a1,($s2)
        jal find_min
        sw $v0,min_value
        addi $s2,$s2,4                  #$s2 is the point
        addi $t0,$t0,1
        bne $t0,$s0 loop_find_min  #s0 is the number of integers

        print_string("the min value : ")
        li $v0,1
        lw $a0,min_value
        syscall

        end          #end is defined in the file is macro_print_str.asm
```

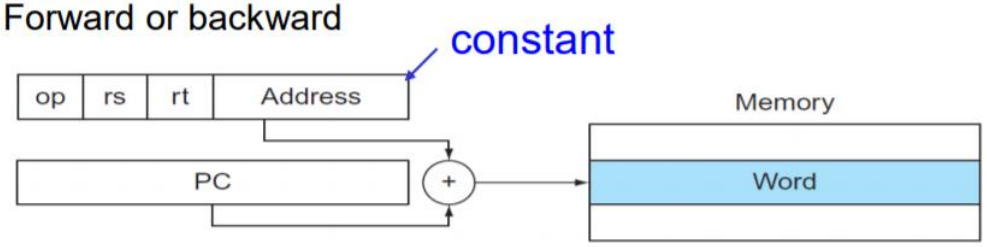
```
#piece 4/4
find_min:

        move $v0,$a0
        blt $a0,$a1,not_update
        move $v0,$a1

not_update:

        jr $ra
```

```
please input the number:3
please input the array
-1
0
1
the min value : -1
-- program is finished running --
```

Practice

- 1. Using Mars to find the value of ".text base address", ".data base address", ".extern base address", "heap base address" and "stack base address".
- 2. Find the value of globl lable "main", "print_callee" and the initial value of $PC of MIPS code on page 10.
- 3. Find the relationship between the binary part of the branch and jump instruction code and the address of the jumping destination according to the "Demo3" on page 12 and 13.

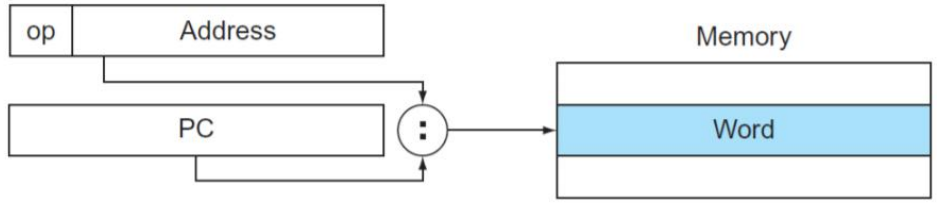
- Forward or backward



PC-relative addressing

- Target address = PC + constant × 4
- PC already incremented by 4 by this time

- Jump (`j` and `jal`) targets could be anywhere in text segment
  - Encode full address in instruction

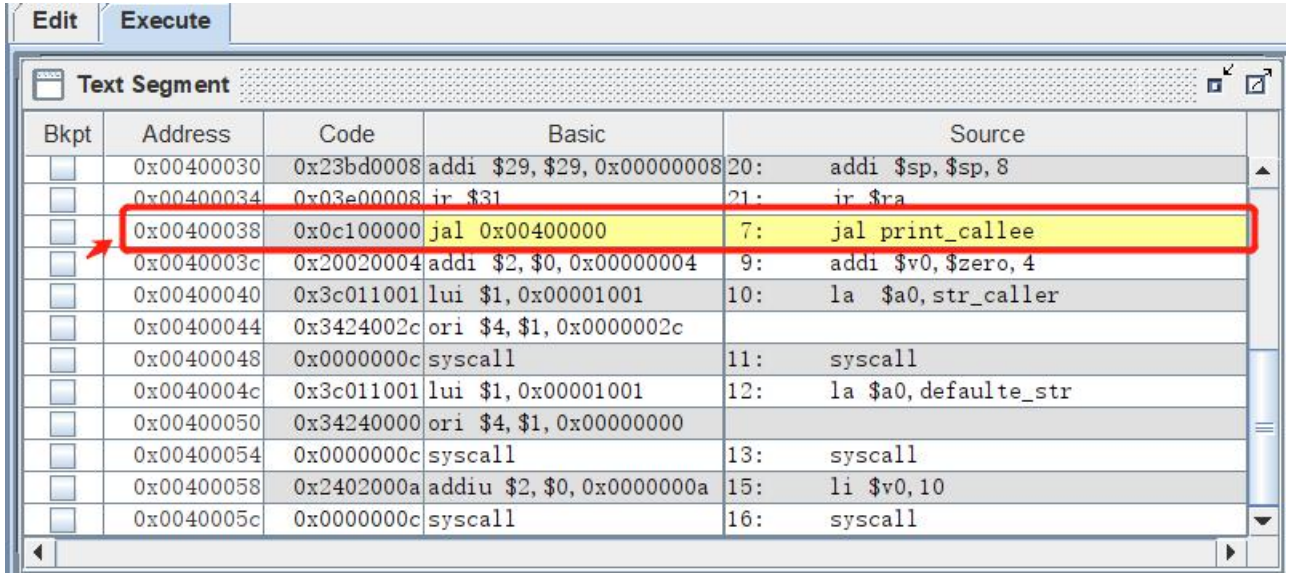


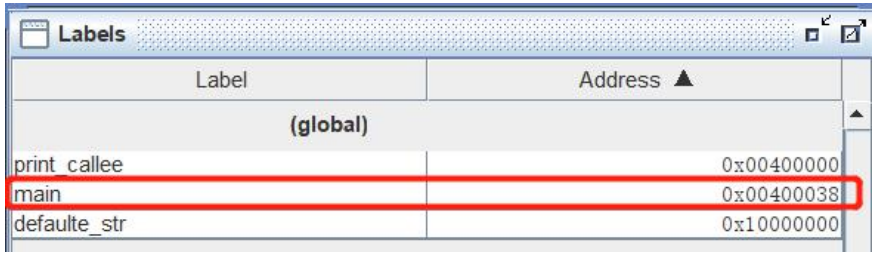
- (Pseudo)Direct jump addressing
  - Target address = PC31…28 : (address × 4)

Tips on Mars

To make the instruction labled by ‘global main’ as the 1st instruction to run, do the following settings.

In **Mars** ‘manual:
Settings -》Initialize Program Counter to global ‘main’ if defined





Tips : macro_print_str.asm

```
.macro print_string(%str)
	.data
	pstr: .asciiz %str
	.text
	la $a0,pstr
	li $v0,4
	syscall
.end_macro


.macro end
	li $v0,10
	syscall
.end_macro
```

Define and use macro, get help form help page of Mars