Computer organization

Lab5 MIPS(4) - macro,procedure,memory

2021 Spring term

wangw6@sustech.edu.cn

Topics

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

Macro

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

- Instead of repeatedly typing the same instructions every time they are used, a programmer invokes the macro and the assembler replaces the macro call with the corresponding sequence of instructions.
- Macros, like subroutines, permit a programmer to create and name a new abstraction for a common operation.
- Unlike subroutines, however, 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.
- After this replacement, the resulting assembly is indistinguishable from the equivalent program written without macros.

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 a asm file, and asamble it, what's the assembly result? Is this 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 :
  - **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.
  - **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.
  - **Execute a jal instruction**, which jumps to the callee's first instruction and saves the return address in register **$ra.**

Procedure(2)

While in **callee**

- 1. **Allocate memory for the frame** by substracting 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.

Procedure(3)

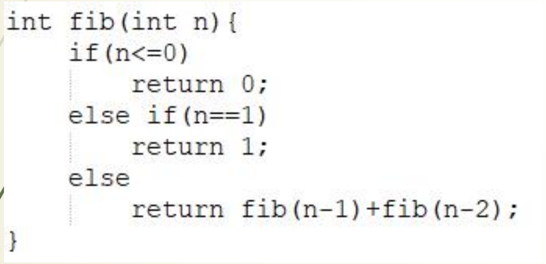
While in **callee**, **before returning to caller**

- If the callee is a function that returns a value, place the returned value in register **$v0**
- **Restore all callee-saved registers** that were saved upon procedure entry
- **Pop the stack frame** by adding the frame size to **$sp**
- **Return by jumping** to the address in register **$ra**

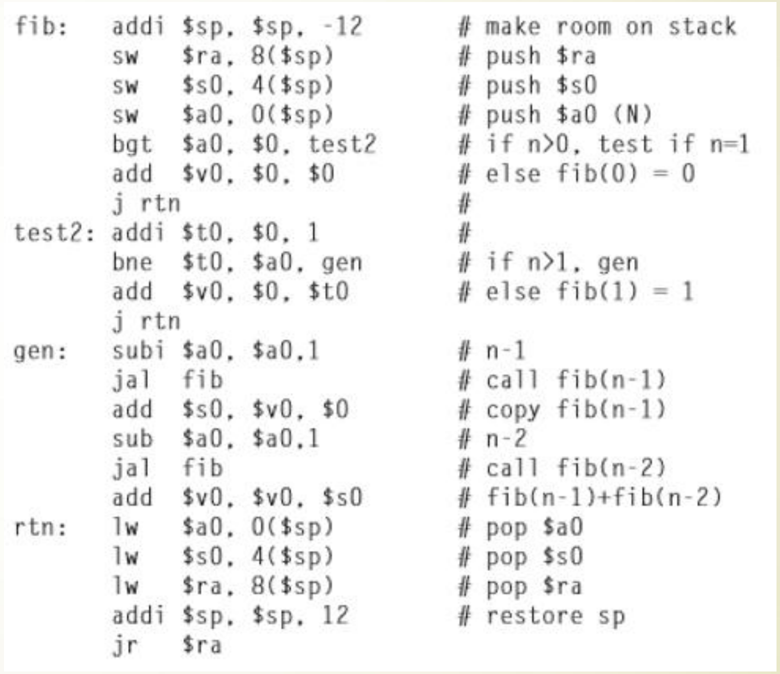
Demo #2

Implement the following C code in MIPS.

Q1. What is the total number of MIPS instructions needed to execute the procedure?



```
int fib(int n){
    if(n<=0)
        return 0;
    else if(n==1)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
```



```
fib:    addi $sp, $sp, -12      # make room on stack
        sw   $ra, 8($sp)        # push $ra
        sw   $s0, 4($sp)        # push $s0
        sw   $a0, 0($sp)        # push $a0 (N)
        bgt  $a0, $0, test2     # if n>0, test if n=1
        add  $v0, $0, $0        # else fib(0) = 0
        j rtn                   #
test2:  addi $t0, $0, 1         #
        bne  $t0, $a0, gen      # if n>1, gen
        add  $v0, $0, $t0       # else fib(1) = 1
        j rtn
gen:    subi $a0, $a0,1         # n-1
        jal  fib                # call fib(n-1)
        add  $s0, $v0, $0       # copy fib(n-1)
        sub  $a0, $a0,1         # n-2
        jal  fib                # call fib(n-2)
        add  $v0, $v0, $s0      # fib(n-1)+fib(n-2)
rtn:    lw   $a0, 0($sp)        # pop $a0
        lw   $s0, 4($sp)        # pop $s0
        lw   $ra, 8($sp)        # pop $ra
        addi $sp, $sp, 12       # restore sp
        jr   $ra
```

External label vs Local label

- **External** label
  - Also called **globl** label.
  - A label referring to an object that can be referenced from files other than the one in which it is defined.
  - example: .extern labelx 20
- **Local** label
  - A label referring to an object that can be used only within the file 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?

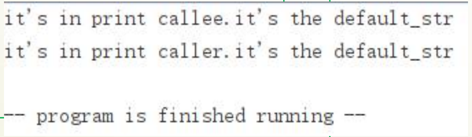
What will happen if an external data have the same name with a local data?

Demo #3-1

Q1. Is the running result same as the sample snap?
Q2. How many "default_str" are defined in "lab5_print_callee.asm"? 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
```



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

-- program is finished running --
```

- 1. Find the value of globl lable "main", "print_callee" and the initial value of $PC of MIPS code on page 11.

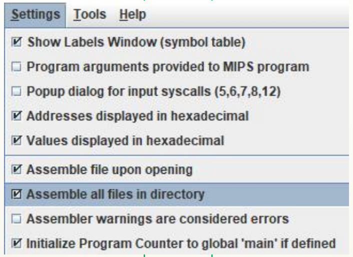
Demo #3-2

0X00400000 ↑ PC   0X 0040003C

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
```

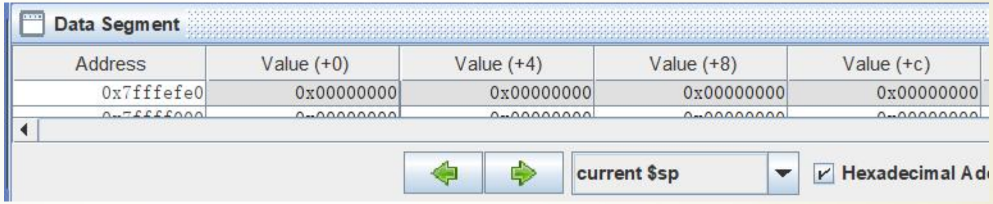
```
.data
        .extern      defaulte_str   20
        str_callee:   .asciiz  "it's in print callee."
        defaulte_str: .asciiz "ABC\n"
.text
.globl  print_callee
print_callee:   addi $sp,$sp,-4
                sw $v0,($sp)

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

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



- ☑ Show Labels Window (symbol table)
- ☐ Program arguments provided to MIPS program
- ☐ Popup dialog for input syscalls (5,6,7,8,12)
- ☑ Addresses displayed in hexadecimal
- ☑ Values displayed in hexadecimal
- ☑ Assemble file upon opening
- ☑ Assemble all files in directory
- ☐ Assembler warnings are considered errors
- ☑ Initialize Program Counter to global 'main' if defined

Stack vs Heap

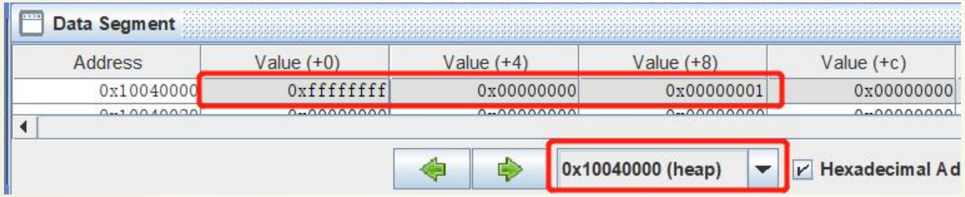
- **Stack**: used to store the local variable, usually used in callee
- **Heap**: The heap is reserved for sbrk and break system calls, and it not always present



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000

current $sp ✓ Hexadecimal Ad



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10040000	0xffffffff	0x00000000	0x00000001	0x00000000

0x10040000 (heap) ✓ Hexadecimal Ad

[illegible]

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 "Demo4" on page 13 and 14.

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")
        add $t0,$0,$0
```

```
loop_read:                      #piece 2/4
        li $v0,5        #read the array
        syscall
        sw $v0,($s2)

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

memory.

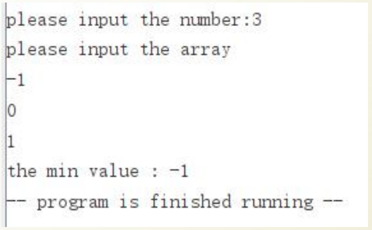
there's no new

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

Demo #4

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

loop_find_min:
        lw $a0,min_value
        lw $a1,($s2)
        jal find_min
        sw $v0,min_value
        addi $s2,$s2,4
        addi $t0,$t0,1
        bne $t0,$s0 loop_find_min

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

li $v0,10
syscall
```

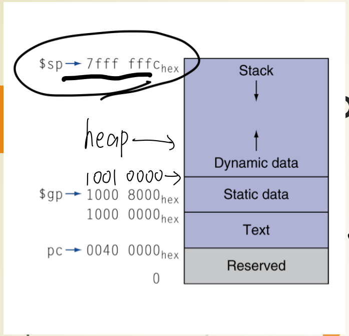
```
#piece 4/4
find_min:
        addi $sp,$sp,-4
        sw $ra,($sp)

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

        not_update:
        lw $ra,($sp)
        addi $sp,$sp,4

        jr $ra
```



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

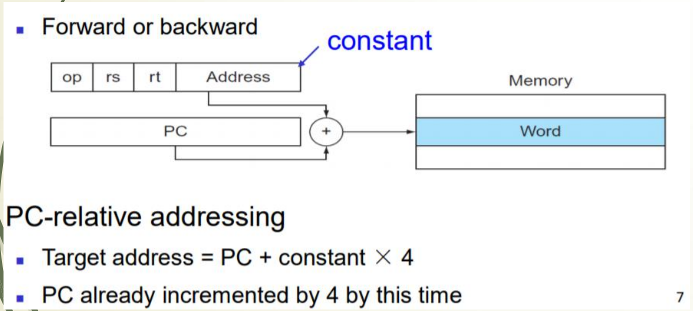
Mips 0x7fffefe0. (有一定保留 space)

0x7fffffC 在选取范围内从大向



ractice

- ✓ 1. Find the value of globl lable "main", "print_callee" and the initial value of $PC of MIPS code on page 11.
- ✓ 2. Using Mars to find the value of ".data base address" (0x10010000), ".extern base address" (0X10000000), "heap base address" (0x10040000) and "the stack base address".
- 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 "Demo4" on page 13 and 14.

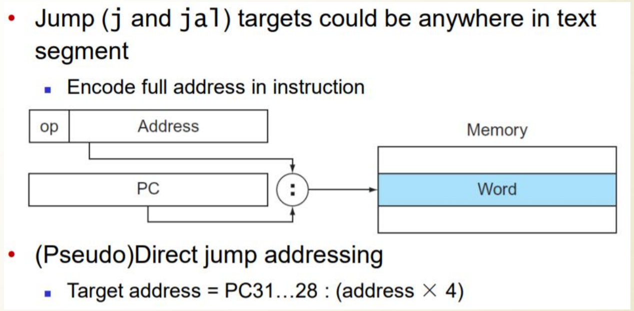


- Forward or backward

PC-relative addressing

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

7



- 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)

Print a multiplication table

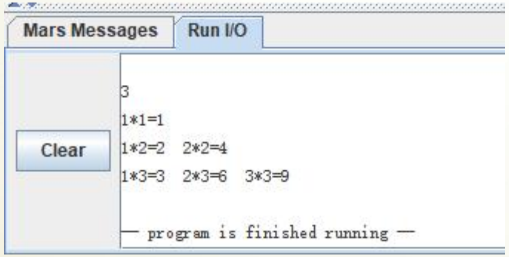
Print out a variable-length multiplication table whose number of rows is determined by the value of the input.

NOTICE:

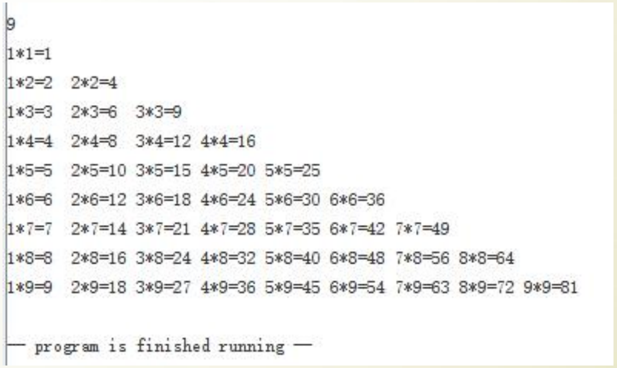
1) There Must be ONLY 3 syscalls in the code. The 1st one is to get the input value, the 2nd one is to print the multiplication table out, the final one is to terminate the execution.

2) TIPs: write the multiplication table as a string into a piece of memory, use "print string" syscall to print it out.

sample input and output are as flow:



```
3
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9

-- program is finished running --
```

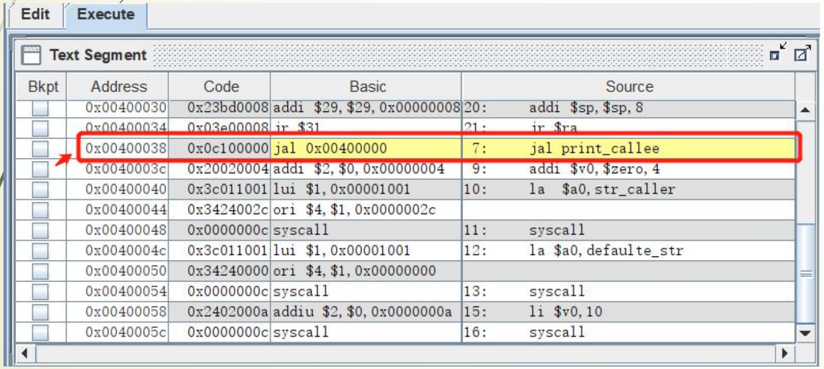


```
9
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12 4*4=16
1*5=5  2*5=10 3*5=15 4*5=20 5*5=25
1*6=6  2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7  2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8  2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9  2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

-- program is finished running --
```

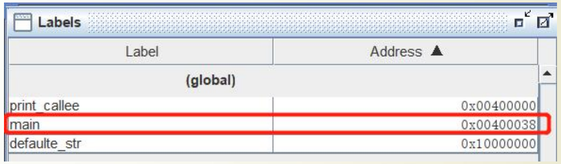
Tips on Mars

To make the global 'main' as the 1st instruction while running, do the following settings.

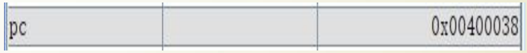
In **Mars**' manual：
Settings -》 Initialize Program Counter to global 'main' if defined



Bkpt	Address	Code	Basic	Source
	0x00400030	0x23bd0008	addi $29, $29, 0x00000008	20: addi $sp, $sp, 8
	0x00400034	0x03e00008	jr $31	21: jr $ra
	0x00400038	0x0c100000	jal 0x00400000	7: jal print_callee
	0x0040003c	0x20020004	addi $2, $0, 0x00000004	9: addi $v0, $zero, 4
	0x00400040	0x3c011001	lui $1, 0x00001001	10: la $a0, str_caller
	0x00400044	0x3424002c	ori $4, $1, 0x0000002c	
	0x00400048	0x0000000c	syscall	11: syscall
	0x0040004c	0x3c011001	lui $1, 0x00001001	12: la $a0, defaulte_str
	0x00400050	0x34240000	ori $4, $1, 0x00000000	
	0x00400054	0x0000000c	syscall	13: syscall
	0x00400058	0x2402000a	addiu $2, $0, 0x0000000a	15: li $v0, 10
	0x0040005c	0x0000000c	syscall	16: syscall



Label	Address ▲
(global)	
print_callee	0x00400000
main	0x00400038
defaulte_str	0x10000000



pc		0x00400038

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