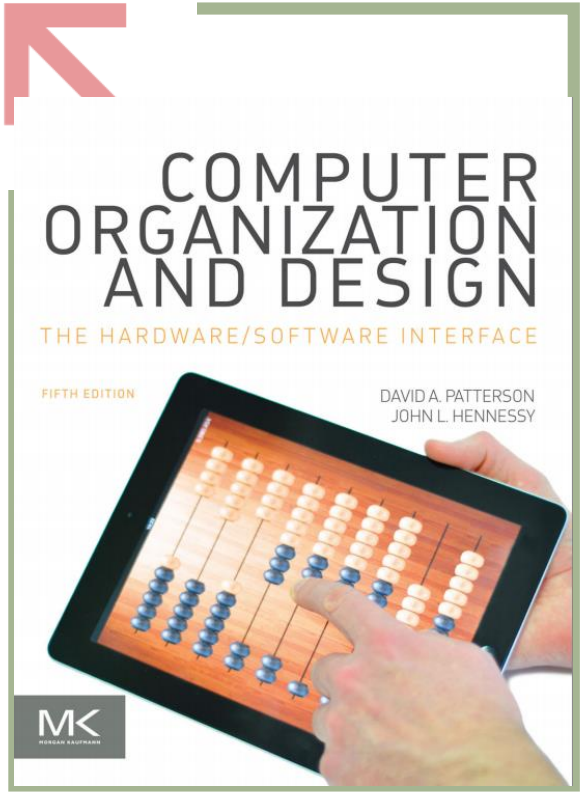
Assembly programming

function， memory

wangw6@sustc.edu.cn



Topics

- Macro vs Function
- Memory
  - Static data /global
  - Dynamic data
  - Stack vs Heap

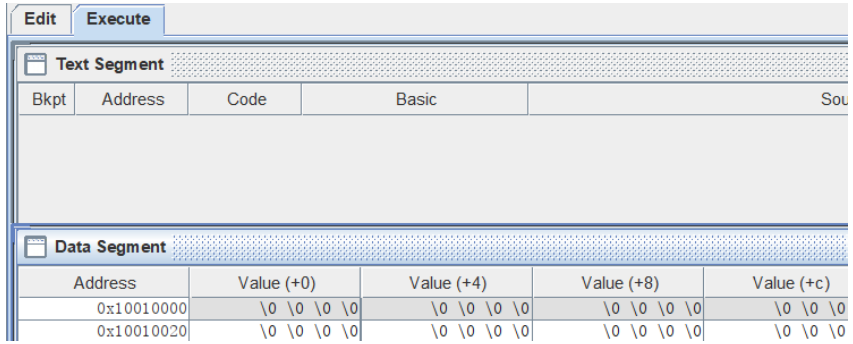
Macro

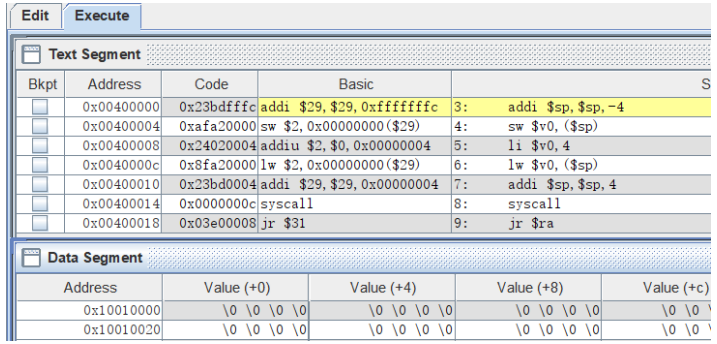
- Macros are a pattern-matching and replacement facility that provide a simple mechanism to name a frequently used sequence of instructions. Instead of repeatedly typing the same instructions every time they are used, a programmer invokes the macro and the assembler replaces the macro call with the corresponding 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

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

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

        li $v0,4
        syscall

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

        jr $ra
```





Procedure

- 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

- 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 velues in these registers($s0-$s7,$fp and $ra) before altering them,sinece 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

While in callee，before return 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 assembly. What is the total number of MIPS instructions needed to execute the function?

```
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 global label. A label referring to an object that can be referenced from files other than the one in which it is defined.
  - .extern labelx 20
- local label A label referring to an object that can be used only within the file in which it is defined.

Demo #3

There are two asm file, one is caller ,another is callee, assembly them and run, to find if the running result is same as the sample snap

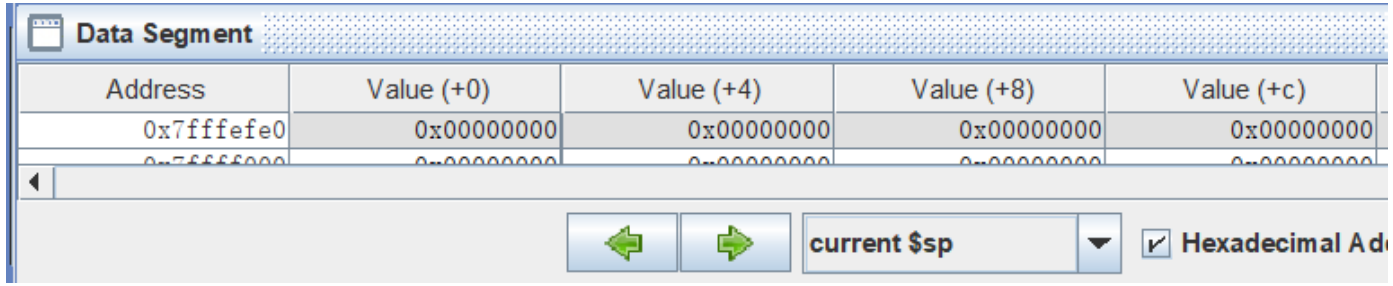
```
.include "lab5_print_callee.asm"
.data
str_caller: .asciiz "it's in print caller."
.text
.globl main
main:
        jal print_callee

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

        li $v0,10
        syscall
```

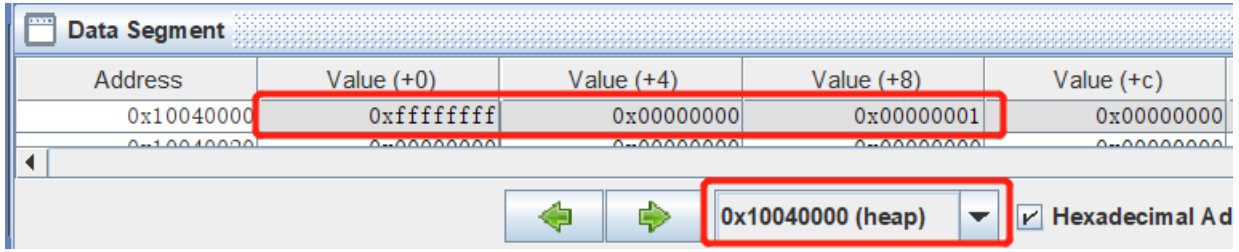
```
it's in print callee.it's the default_str
it's in print caller.it's the default_str

-- program is finished running --
```

```
.extern defaulte_str 20
.data
        defaulte_str: .asciiz    “it‘s the default_str\n”
        str_callee: .asciiz  "it's in print callee."
.text
print_callee:
        addi $sp,$sp,-8
        sw $ra,4($sp)
        sw $v0,($sp)

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

        lw $v0,($sp)
        lw $ra,4($sp)
        addi $sp,$sp,8
        jr $ra
```

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



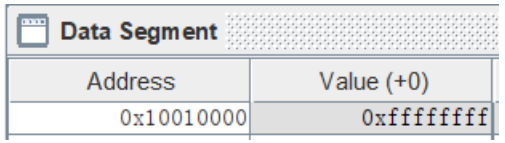


Demo #4

Demo #4 is to get the datas from input device, get the minimal value among the datas ,the number of input data is uncertain, determined by user

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

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

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

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



Demo #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
```

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

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

        li $v0,10
        syscall

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
```

Lab Assignment（23:55 March 26, Tuesday）

1. Implemented the Fibolacci number by loop and recursive methods respectively, count the the number of instructions required by fib(5).
2. write and test a program **that reads in a positive integer** using the SPIM system calls. **If** the integer is not positive, the program should **terminate with the message "Invalid Entry**"; **otherwise** the program should **print out the names of the digits of the integers**, delimited by exactly one space. For example, if the user entered "728," the output would be "Seven Two Eight."
3. Write and test a MIPS assembly language program to compute and **print the first 100 prime numbers.** A number n is prime if no numbers except 1 and n divide it evenly. You should implement two routines:
    1. **test_prime (n)** Return 1 if n is prime and 0 if n is not prime.
    2. **main () Iterate over the integers**, testing if each is prime. Print the first 100 numbers that are prime.

Tips on Mars

To make the global ‘main’ as the 1st instruction while running ,setting the initialization on PC
In Mars：
Settings -》Initialize Program Counter to global ‘main’ if defined