Computer organization

Lab4 MIPS(3) - switch , loop , function

2020 Spring term

wangw6@sustech.edu.cn

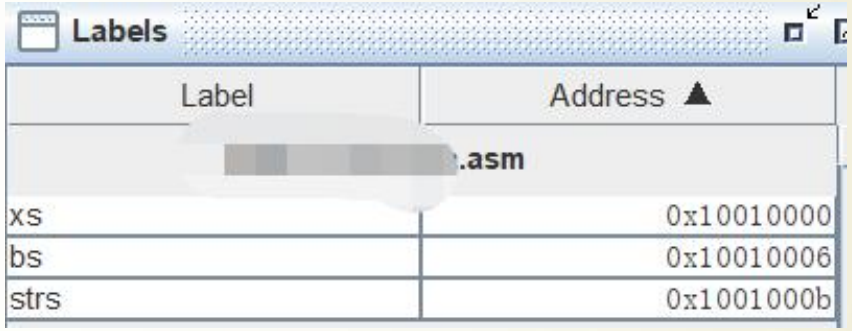
Topics

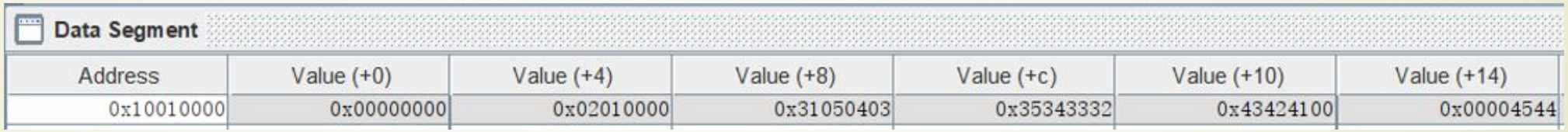
- branch
- loop
- function, call, return
- register PC

Array

```
.data
        xs:   .space    6
        bs:   .byte     1,2,3,4,5
        strs:.asciiz    "12345","ABCDE"
.text
main:
#insert code here to print the string
"ABCDE" of strs
        li $v0,4
        syscall

        li $v0,10
        syscall
```





Branch, Jump Instruction

- **Conditionally branch**
  - **beq** $t0,$t1,lable // branch to statement at label's address if $t1 and $t2 are equal
  - **bne** $t0,$t1,lable // branch to statement at label's address if $t1 and $t2 are NOT equal
  - **blt, ble, bltu, bleu, bgt, bge, bgtu, bgeu**
- **Unconditionally jump**

Jump (j)	Unconditionally jumps to a specified location. A symbolic address or a general register specifies the destination. The instruction j $31 returns from the a jal call instruction.
Jump And Link (jal)	Unconditionally jumps to a specified location and puts the return address in a general register. A symbolic address or a general register specifies the target location. By default, the return address is placed in register $31. If you specify a pair of registers, the first receives the return address and the second specifies the target. The instruction jal procname transfers to procname and saves the return address. For the two-register form of the instruction, the target register may not be the same as the return-address register. For the one-register form, the target may not be $31.

Branch

Are the results of two demos the same ?
*Modify it without changing the result by using **ble** or **blt** instead*

```
.include "../macro_print_str.asm"
.data
.text
        print_string("please input your score (0~100):")
        li $v0,5
        syscall
        move $t0,$v0
case1:
        bgt $t0,90,gt90_lable
case2:
        bge $t0,70,gt70lt90_lable
case3:
        print_string("\nNOT GOOD(less than 70)")
        j case_end

gt90_lable:
        print_string(  "\nEXCELLENT (exceed 90) ")
        j case_end
gt70lt90_lable:
        print_string("  \nGOOD(70~90)")
        j case_end
case_end:
        end
```

```
.include "../macro_print_str.asm"
.data
.text
        print_string("please input your score (0~100):")
        li $v0,5
        syscall
        move $t0,$v0
case1:
        bgt $t0,90,gt90_lable
        j case2
case2:
        bge $t0,70,gt70lt90_lable
        j case3
case3:
        print_string("\nNOT GOOD (less than 70) ")
        j case_end

gt90_lable:
        print_string(  "\nEXCELLENT (exceed 90) ")
        j case_end
gt70lt90_lable:
        print_string(  "\nGOOD(70~90)")
        j case_end
case_end:
        end
```

Loop

Compare the operations of loop in java and MIPS, when calculating the sum from 1 to 10.

Java:

```
public class CalculateSum{
    public static void main(String [] args){
        int i = 0;
        int sum = 0;
        for(i=0;i<=10;i++)
              sum = sum + i;
        System.out.print(" The sum from 1 to 10 : "   + sum );
    }
}
```

MIPS:

```
.include "macro_print_str.asm"
.data
      tdata: .word 0
.text
      add $t1,$zero,$zero
      addi $t0,$zero,0
      addi $t7,$zero,10
calcu:
      addi $t0,$t0,1
      add $t1,$t1,$t0
      bgt $t7,$t0,calcu

      print_string(  "The sum from 1 to 10 : ")
      move $a0,$t1
      li $v0,1
      syscall

      end
```

Demo #1

The code in the next page is expected to get 10 integers from the input device, and print it as the following sample

Will the code get desired result? If not, what happened ?

```
please input an array  (no more than 10 integer): 1
2
3
4
5
6
7
8
9
0

the arrayx is:1 2 3 4 5 6 7 8 9 0
-- program is finished running --
```

Demo #1

#piece 1/3

```
.include "../macro_print_str.asm"
.data
        arrayx:     .space      10
        str:        .asciiz     "\nthe arrayx is:"
.text
main:
        print_string("please input 10 integers: ")
        add $t0,$zero,$zero
        add $t1,$zero,10
        la $t2,arrayx
```

#piece 2/3

```
loop_r:
        li $v0,5
        syscall
        sw $v0,($t2)
        addi $t0,$t0,1
        addi $t2,$t2,4
        bne $t0,$t1,loop_r

        la $a0,str
        li $v0,4
        syscall
        addi $t0,$zero,0
        la $t2,arrayx
```

#piece 3/3

```
loop_w:
        lw $a0,($t2)
        li $v0,1
        syscall
        print_string(" ")
        addi $t2,$t2,4
        addi $t0,$t0,1
        bne $t0,$t1,loop_w
        end
```

The function of following code is to get 5 integers from input device, and find the min value and max value of them.
There are 4 pieces of code , write your code based on them,Can it find the real min and max?

```
#piece ?/4
.include "macro_print_str.asm"
.data
        min: .word 0
        max: .word 0
.text
        lw $t0,min
        lw $t1,max
        li $t7,5
        li $t6,0
        print_string("please input 5 integer:")
loop:
        li $v0,5
        syscall
        bgt $v0,$t1,get_max
        j get_min
```

```
#piece ?/4
get_max:
        move $t1,$v0
        j get_min
get_min:
        bgt $v0,$t0,judge_times
        move $t0,$v0
        j judge_times
```

```
#piece ?/4
judge_times:
        addi $t6,$t6,1
        bgt $t7,$t6,loop
```

```
#piece ?/4
        print_string("min : ")
        move $a0,$t0
        li $v0,1
        syscall
        print_string("max : ")
        move $a0,$t1
        li $v0,1
        syscall
        end
```

Function/procession

- **jal function_lable**
  - #Unconditionally jump to the instruction at function_lable. Save the address of the next instruction in register $ra
  - Used in caller while calling the function
- **jr $ra**
  - #Unconditionally jump to the instruction whose address is in register ra
  - Used in callee while return to the caller
- **lw / sw , $sp**
  - #protected register data by using stack in memory

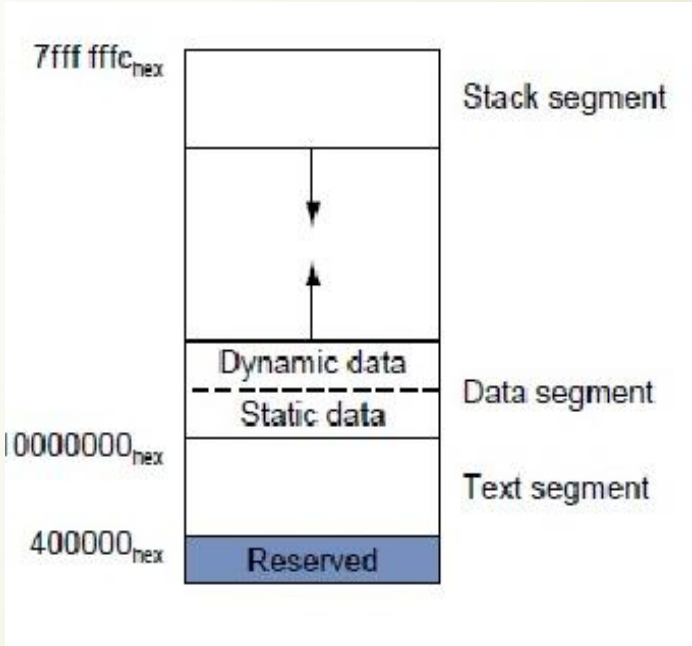
Stack segment

stack segment The portion of memory used by a program to hold procedure call frames.

The program ***stack segment***, resides **at the top of the virtual address space**(starting at address 7fffffff $_{hex}$).

Like dynamic data, the maximum size of a program's stack is not known in advance.

As the program **pushes values on the stack**, the operating system **expands** the stack segment **down, toward the data segment**.



Demo #2

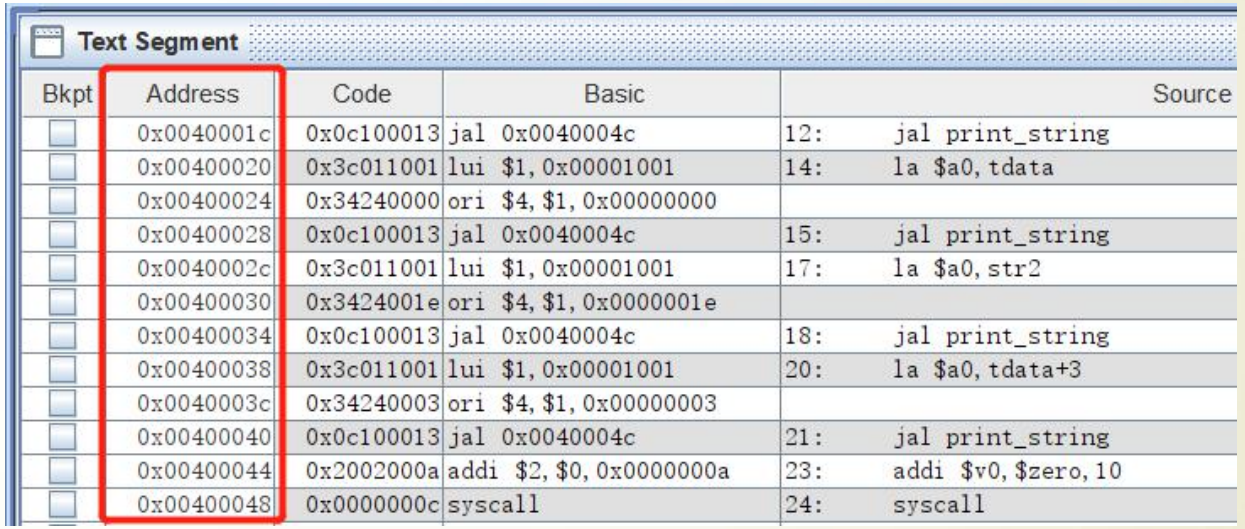
What's the value of $ra while jump and link to the print_string (at line 12,15,18,21)?

```
print_string:
        addi $sp,$sp,-8
        sw $a0,4($sp)
        sw $v0,0($sp)

        addi $v0,$zero,4
        syscall

        lw $v0,0($sp)
        lw $a0,4($sp)
        addi $sp,$sp,8

        jr $ra
```



Text Segment

Bkpt	Address	Code	Basic	Source
	0x0040001c	0x0c100013	jal 0x0040004c	12: jal print_string
	0x00400020	0x3c011001	lui $1, 0x00001001	14: la $a0, tdata
	0x00400024	0x34240000	ori $4, $1, 0x00000000	
	0x00400028	0x0c100013	jal 0x0040004c	15: jal print_string
	0x0040002c	0x3c011001	lui $1, 0x00001001	17: la $a0, str2
	0x00400030	0x3424001e	ori $4, $1, 0x0000001e	
	0x00400034	0x0c100013	jal 0x0040004c	18: jal print_string
	0x00400038	0x3c011001	lui $1, 0x00001001	20: la $a0, tdata+3
	0x0040003c	0x34240003	ori $4, $1, 0x00000003	
	0x00400040	0x0c100013	jal 0x0040004c	21: jal print_string
	0x00400044	0x2002000a	addi $2, $0, 0x0000000a	23: addi $v0, $zero, 10
	0x00400048	0x0000000c	syscall	24: syscall

pay attaction to the value of $pc

Demo #2

read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read

```
.data
        tdata: .space 6
        str1:  .asciiz "the orignal string is: "
        str2:  .asciiz "\nthe last two character of the string is: "
.text
        la $a0,tdata
        addi $a1,$zero,6
        addi $v0,$zero,8
        syscall
```

```
        la $a0,str1
        jal print_string

        la $a0,tdata
        jal print_string

        la $a0,str2
        jal print_string

        la $a0,tdata+3
        jal print_string

        addi $v0,$zero,10
        syscall
```

```
print_string:
        addi $sp,$sp,-8
        sw $a0,4($sp)
        sw $v0,0($sp)
        addi $v0,$zero,4
        syscall

        lw $v0,0($sp)
        lw $a0,4($sp)
        addi $sp,$sp,8
        jr $ra
```

IS it ok to remove the push and pop process on the stack in "print_string" ?

Recursion

fact is a function to calculate the Calculate the factorial.

C

```
int   fact(int n) {
      if(n<1)
            return 1;
      else
            return (n*fact(n-1));
}
```

IS it ok to remove the push and pop process on the stack in "fact" here ->

MIPS

```
fact:
      addi  $sp,$sp,-8        #adjust stack for 2 items
      sw    $ra, 4($sp)       #save the return address
      sw    $a0, 0($sp)       #save the argument n

      slti   $t0,$a0,1        #test for n<1
      beq    $t0,$zero,L1     #if n>=1,go to L1

      addi  $v0,$zero,1       #return 1
      addi  $sp,$sp,8         #pop 2 items off stack
      jr      $ra             #return to caller

L1:   addi  $a0,$a0,-1        #n>=1; argument gets(n-1)
      jal     fact            #call fact with(n-1)

lw    $a0,0($sp)              #return from jal: restore argument n
lw    $ra,4($sp)              #restore the return address
addi  $sp,$sp,8               #adjust stack pointer to pop 2 items

mul   $v0,$a0,$v0                   #return n*fact(n-1)

jr    $ra                           #return to the caller
```

practice

1. print out a 9*9 multiplication table.
    1. define a function to print a*b = c , the value of “a” is from parameter $a0,the value of “b” is from parameter $a1.
    2. less syscall is better
2. get a positive integer from input, calculate the sum from 1 to this value by using recursion, output the result
3. get a positive integer from input, output an integer in reverse order using loop and recursion seperately.

Tips

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

- ✓ Registers **$a0~$a3** are used to **pass the first four arguments to routines** (remaining arguments are passed on the stack).
- ✓ Registers **$v0~$v1** are used to **return values from functions**.
- ✓ Registers **$t0~ $t9** are ***caller-saved registers*** that are used to hold temporary quantities that need not be preserved across calls.
- ✓ Registers **$s0~$s7** are ***callee-saved registers*** that hold long-lived values that should be preserved across calls.
- ✓ Register **$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.

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 Mars