



Computer Organization

Lab4 MIPS(3)

Directive, Memory,
Machine Code



Topics

➤ Directives

- `.data`, `.text`
- `.macro`, `.endmacro` (optional)
- `.align` (0,1,2)
- `.globl` vs `.extern`, `.globl main`
 - practice 1-1,1-2 (page 9)

➤ Memory

- Static Storage vs Dynamic Storage(stack vs heap)
- practice 1-3 (page10)

➤ Practice 2

- Machine Code and Addressing



Directives in Mars

MARS 4.5 Help	
MIPS	MARS
License	Bugs/Comments
Acknowledgements	Instruction Set Song
Basic Instructions	Extended (pseudo) Instructions
Directives	Syscalls
Exceptions	Macros
.align	Align next data item on specified byte boundary (0=byte, 1=half, 2=word, 3=double)
.ascii	Store the string in the Data segment but do not add null terminator
.asciiz	Store the string in the Data segment and add null terminator
.byte	Store the listed value(s) as 8 bit bytes
.data	Subsequent items stored in Data segment at next available address
.double	Store the listed value(s) as double precision floating point
.end_macro	End macro definition. See .macro
.eqv	Substitute second operand for first. First operand is symbol, second operand is expression (like #define)
.extern	Declare the listed label and byte length to be a global data field
.float	Store the listed value(s) as single precision floating point
.globl	Declare the listed label(s) as global to enable referencing from other files
.half	Store the listed value(s) as 16 bit halfwords on halfword boundary
.include	Insert the contents of the specified file. Put filename in quotes.
.kdata	Subsequent items stored in Kernel Data segment at next available address
.ktext	Subsequent items (instructions) stored in Kernel Text segment at next available address
.macro	Begin macro definition. See .end_macro
.set	Set assembler variables. Currently ignored but included for SPIM compatibility
.space	Reserve the next specified number of bytes in Data segment
.text	Subsequent items (instructions) stored in Text segment at next available address
.word	Store the listed value(s) as 32 bit words on word boundary



Directive: `.macro`, `.endmacro`

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 procedure?

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

Is the macro's definition file runnable?

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

Is the procedure definition file runnable?

```
.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
```




Directive: .align(Demo #2-1)

```
.data          #A
str1: .ascii "Welcome"
str2: .ascii "to"
str3: .asciiz "MIPS32World"
.text
la $t0, str2
lb $t1, ($t0)
addi $t1, $t1, -32
sw $t1, ($t0)

la $a0, str1
li $v0, 4
syscall

li $v0, 10
syscall
```

```
.data          #B
str1: .ascii "Welcome"
str2: .ascii "to"
str3: .asciiz "MIPS32World"
.text
la $t0, str2
lw $t1, ($t0)
addi $t1, $t1, -32
sb $t1, ($t0)

la $a0, str1
li $v0, 4
syscall

li $v0, 10
syscall
```

Which demo(s) would invoke an exception "fetch address not aligned on word boundary 0x10010007" ?

Which instruction would invoke the exception? lb, sw, lw, sb?

Tips:

While transferring data, the address of data in memory is required to be aligned according to the bit width of data.



Directive: .align(Demo #2-2)

.align : align next data item on specified byte boundary(0=byte, 1=half, 2=word, 3=double)

Which demo(s) would run without exception?

Which demo(s) would get the output "WelcomeToMIPS32World" ?

```
.data           #A
str1: .ascii "Welcome"
str2: .ascii "to"
str3: .asciiz "MIPS32World"
```

```
.text
la $t0, str2
lh $t1, ($t0)
addi $t1, $t1, -32
sh $t1, ($t0)
```

```
la $a0, str1
li $v0, 4
syscall
li $v0, 10
syscall
```

```
.data           #B
str1: .ascii "Welcome"
.align 2
str2: .ascii "to"
str3: .asciiz "MIPS32World"
```

```
.text
la $t0, str2
lw $t1, ($t0)
addi $t1, $t1, -32
sw $t1, ($t0)
```

```
la $a0, str1
li $v0, 4
syscall
li $v0, 10
syscall
```

```
.data           #C
.align 2
str1: .ascii "Welcome"
str2: .ascii "to"
str3: .asciiz "MIPS32World"
```

```
.text
la $t0, str2
lw $t1, ($t0)
addi $t1, $t1, -32
sw $t1, ($t0)
```

```
la $a0, str1
li $v0, 4
syscall
li $v0, 10
syscall
```

```
.data           #D
str1: .ascii "Welcome"
str2: .ascii "to"
str3: .asciiz "MIPS32World"
```

```
.text
la $t0, str2
lb $t1, ($t0)
addi $t1, $t1, -32
sb $t1, ($t0)
```

```
la $a0, str1
li $v0, 4
syscall
li $v0, 10
syscall
```



Directive: `.globl` vs `.extern`

- **.include** : insert the contents of the specified **file**, put filename in quotes
- **.globl** : declare the listed **label(s)** as global to enable referencing from other files
- **.extern** : declare the listed **label** and byte length to be a global **data** field

➤ 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 FILE other than the one in which it is defined.**

*Find the usage of “`.extern`” and “`.globl`” on Demo 3-1 and 3-2
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

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 "print_callee.asm" ?

Q3. While executing the instruction "la \$a0,default_str" in these two files, which "default_str" is used?

```
## "print_caller.asm" ##  
.include "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
```

```
## "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 #3-2

In Mars, set "Assemble all files in directory" , put the following files in the same directory, then run it.

Practice 1-1 : answer the questions on last page again.

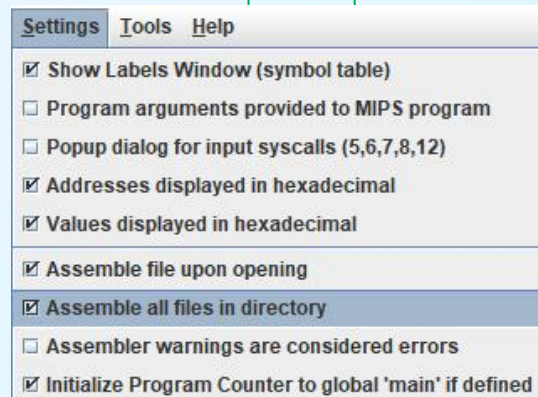
Practice 1-2: Find the value of globl lable "main" , "print_callee" and the initial value of \$PC

```
.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
```

Memory: 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.

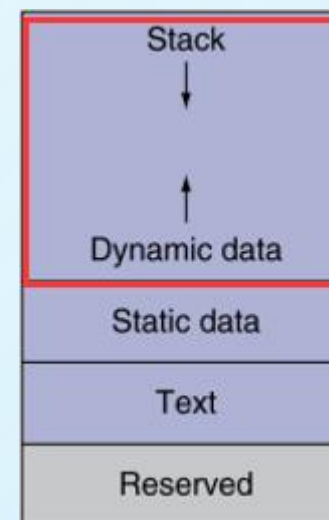
Data Segment				
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x7ffffefe0	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffffef00	0x00000000	0x00000000	0x00000000	0x00000000

current \$sp Hexadecimal Ad

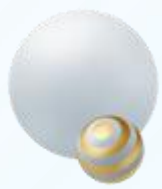
Data Segment				
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10040000	0xffffffff	0x00000000	0x00000001	0x00000000
0x10040004	0x00000000	0x00000000	0x00000000	0x00000000

0x10040000 (heap) Hexadecimal Ad

0x7ffffefc	stack pointer \$sp
0x10040000	stack limit address
0x10040000	heap base address



Pracitce 1-3: Using Mars to find the value of ".text base address" , ".data base address" , ".extern base address" , "heap base address" and "stack base address" .



Demo #4-1

The following demo(composed of 4 pieces on this page and the next) 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 #4-2

#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

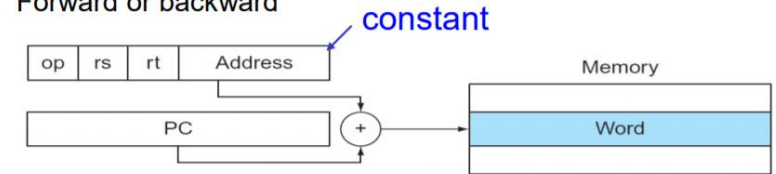
```
please input the number:3
please input the array
-1
0
1
the min value : -1
-- program is finished running --
```




Practice2

- Assemble the MIPS code of “Demo4” on page 12 and 13, answer the following question:
 - P2-1: Find three instructions in Demo4, which belong to R, I and J in the code of Demo 4,
 - what are the opcode of them?
 - what's the index of register in the R type instruction?
 - what's the immediate in the I type instruction?
 - what's the address of J type instruction?
 - P2-2: Find the relationship between the binary part of the branch and jump instruction code and the address of the jumping destination
 - step1: The value of two labels: **loop_find_min, find_min**
 - step2: What's the “address” in the machine code of instruction “jal find_min” and “bne \$t0,\$s0 loop_find_min”
 - Calculate the jumping destination based on the “address” got from step2, are the jumping destinations same with the value got from step1?

- Forward or backward



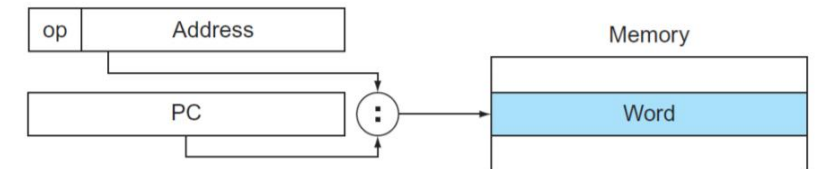
PC-relative addressing

- Target address = $PC + \text{constant} \times 4$
- 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 = $PC_{31...28} : (\text{address} \times 4)$



Tips: Machine Code of MIPS(1)

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

op(31:26)								
28-26	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
31-29								
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	set less than imm. unsigned	andi	ori	xori	load upper immediate
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	load byte unsigned	load half unsigned	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	load linked word	lwcl						
7(111)	store cond. word	swcl						



Tips: Machine Code of MIPS(2)

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

op(31:26)=000000 (R-format), funct(5:0)								
2-0 5-3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump register	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)			set l.t.	set l.t. unsigned				
6(110)								
7(111)								

Tips on Mars

To make the instruction labeled 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

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

Labels	
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 of Mars