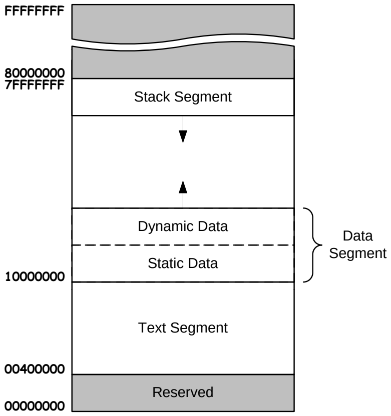
MIPS32 AL – Stack Segment & Functions Prelim
(data segment variables)

- 2 shortcomings of variables allocated in *data segment*:
  - Accessible from all of program's functions
    - *Global* variables!
  - Labels (names) must be distinct
- Can avoid above by allocating variables in *stack segment*
  - *Local* variables
    - Accessible only within function
    - *By rule*, a function must not access another function's memory unless such access is intended by design
- How?

```
         .data
int1:    .word 1234
int2:    .word 321
label:   .asciiz "\n1234 x 321 = "
         .text
         .globl main
main:
         la $a0, label
         li $v0, 4
         syscall
         la $a1, int1
         lw $t1, 0($a1)
         la $a2, int2
         lw $t2, 0($a2)
         mul $a0, $t1, $t2
         li $v0, 1
         syscall
         li $v0, 10
         syscall
```

MIPS32 AL – Stack Segment & Functions Prelim
(stack segment variables)

```
         .data
label:   .asciiz "\n1234 x 321 = "
         .text
         .globl main
main:
         li $t1, 1234
         li $t2, 321
         addiu $sp, $sp, -8
         sw $t1, 4($sp)
         sw $t2, 0($sp)
         la $a0, label
         li $v0, 4
         syscall
         lw $t3, 4($sp)
         lw $t4, 0($sp)
         mul $a0, $t3, $t4
         li $v0, 1
         syscall
         addiu $sp, $sp, 8
         li $v0, 10
         syscall
```

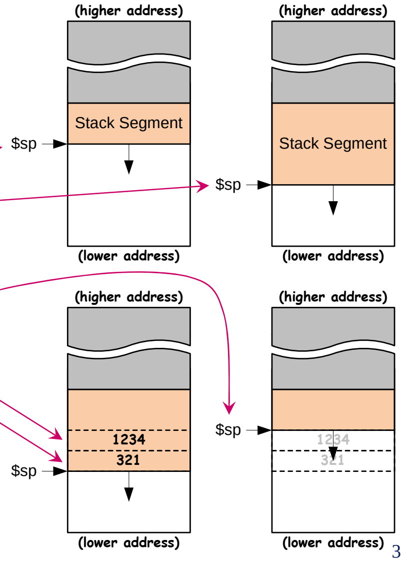


- Looks like much fuss for no good?
  - Meant to illustrate use of stack segment (not at all implying this code's better)
  - Need to know how to use stack segment *when doing functions*

MIPS32 AL – Stack Segment & Functions Prelim

(stack segment variables)

```
         .data
label:   .asciiz "\n1234 x 321 = "
         .text
         .globl main
main:
         li $t1, 1234
         li $t2, 321
         addiu $sp, $sp, -8
         sw $t1, 4($sp)
         sw $t2, 0($sp)
         la $a0, label
         li $v0, 4
         syscall
         lw $t3, 4($sp)
         lw $t4, 0($sp)
         mul $a0, $t3, $t4
         li $v0, 1
         syscall
         addiu $sp, $sp, 8
         li $v0, 10
         syscall
```



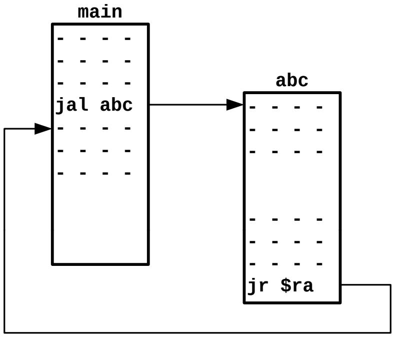
Exercise:
Modify program so that label is local variable.

MIPS32 AL – Stack Segment & Functions Prelim

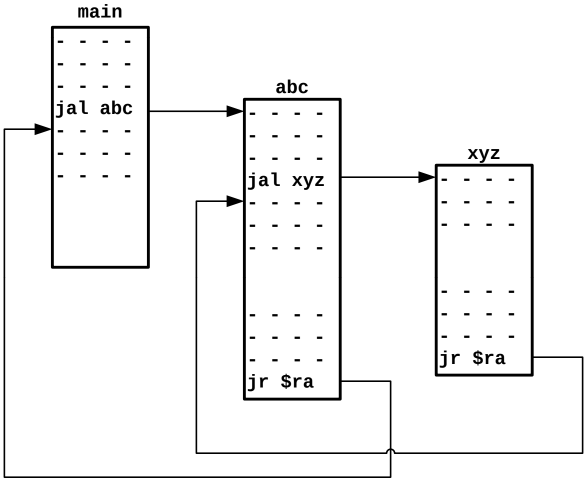
(why do functions, in case you still wonder)

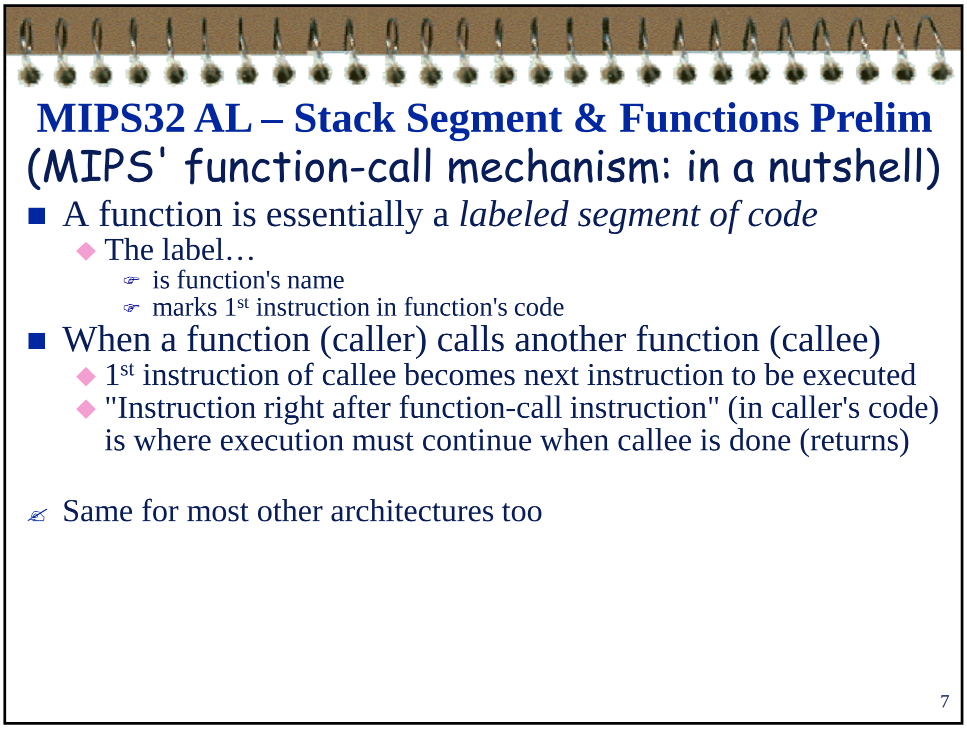
- Lecture note supplement
  - ***014 MIPS32AssemblyLanguageStackSegmentAndFunctionsPrelimSup01***
  - "Scoops about functions"
  - C++ specifically referred to, but most points apply generally
- Summary of potential benefits
  - Increased code reliability
  - Increased code reusability
  - Increased code updatability
  - Design and development facility
  - Organization and documentation facility

MIPS32 AL – Stack Segment & Functions Prelim
(invocation flow of control - simple)



MIPS32 AL – Stack Segment & Functions Prelim
(invocation flow of control - nested)

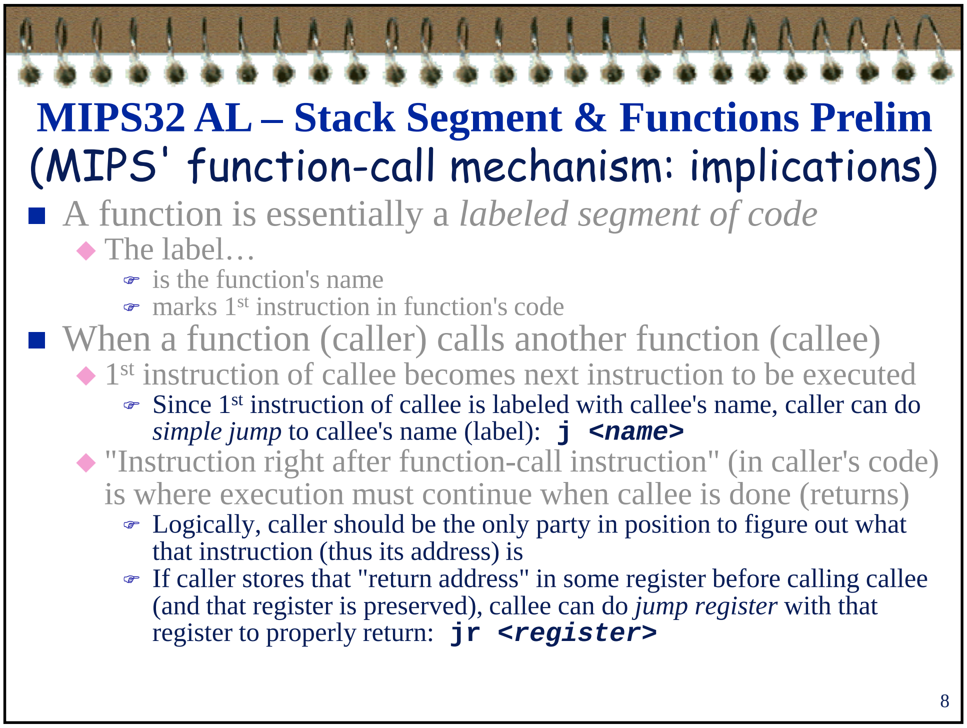




MIPS32 AL – Stack Segment & Functions Prelim
(MIPS' function-call mechanism: in a nutshell)

- A function is essentially a *labeled segment of code*
  - The label…
    - is function's name
    - marks 1st instruction in function's code
- When a function (caller) calls another function (callee)
  - 1st instruction of callee becomes next instruction to be executed
  - "Instruction right after function-call instruction" (in caller's code) is where execution must continue when callee is done (returns)

✍ Same for most other architectures too



MIPS32 AL – Stack Segment & Functions Prelim
(MIPS' function-call mechanism: implications)

- A function is essentially a *labeled segment of code*
  - The label…
    - is the function's name
    - marks 1st instruction in function's code
- When a function (caller) calls another function (callee)
  - 1st instruction of callee becomes next instruction to be executed
    - Since 1st instruction of callee is labeled with callee's name, caller can do *simple jump* to callee's name (label): `j <name>`
  - "Instruction right after function-call instruction" (in caller's code) is where execution must continue when callee is done (returns)
    - Logically, caller should be the only party in position to figure out what that instruction (thus its address) is
    - If caller stores that "return address" in some register before calling callee (and that register is preserved), callee can do *jump register* with that register to properly return: `jr <register>`

MIPS32 AL – Stack Segment & Functions Prelim

(MIPS' function-call mechanism: will this work?)

- A function is essentially a *labeled segment of code*
  - The label…
    - ☞ is the function's name
    - ☞ marks 1st instruction in function's code
- When a function (caller) calls another function (callee)
  - 1st instruction of callee becomes next instruction to be executed
    - ❷ ☞ To call calle, caller does

      ```
      j <label>
      ```

      where **`<label>`** is the label marking callee's 1st instruction
  - "Instruction right after function-call instruction" (in caller's code) is where execution must continue when callee is done (returns)
    - ❶ ☞ Caller figures out "return address" & stores it in **`$ra`** (per convention & to ensure preservation) before calling callee
    - ❸ ☞ To return to caller, callee does

      ```
      jr $ra
      ```

  - ✍ Of course, other conventions to be observed where applicable
    - ☞ Arguments in **`$a0`**, …, **`$a3`**, return value in **`$v0`** and **`$v1`**, *etc.*

MIPS32 AL – Stack Segment & Functions Prelim

(let's try doing it)

```
                        .data
int1:                   .word 123
int2:                   .word 234
out_msg:                .asciiz "\nSum: "
                        .text
                        .globl main
main:
                        la $a0, out_msg
                        li $v0, 4
                        syscall
                        la $t9, int1
                        lw $a0, 0($t9)
                        la $t9, int2
                        lw $a1, 0($t9)
                        la $ra, 0x12345
                        j sum2ints
                        move $a0, $v0
                        li $v0, 1
                        syscall
                        la $a0, out_msg
                        li $v0, 4
                        syscall
```

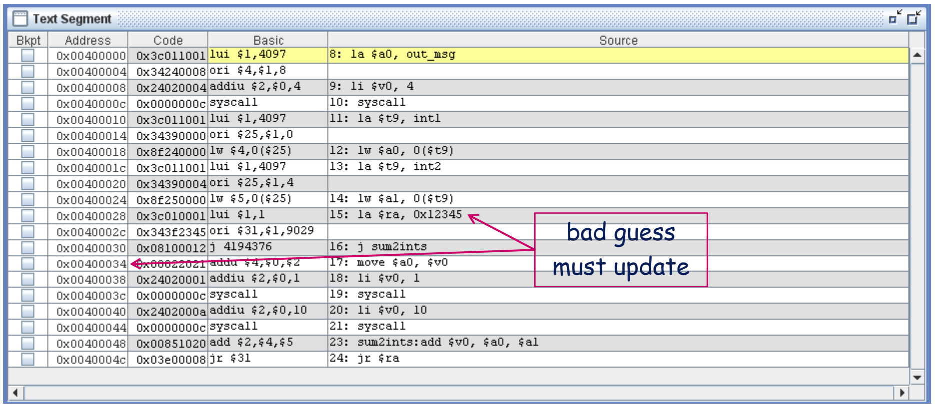
(continued on the right)

```
sum2ints:               add $v0, $a0, $a1
                        jr $ra
```

fill in some guess address so it will compile in MARS

MIPS32 AL – Stack Segment & Functions Prelim

(let's try doing it)



Bkpt	Address	Code	Basic	Source
□	0x00400000	0x3c011001	lui $1,4097	8: la $a0, out_msg
□	0x00400004	0x34240008	ori $4,$1,8	
□	0x00400008	0x24020004	addiu $2,$0,4	9: li $v0, 4
□	0x0040000c	0x0000000c	syscall	10: syscall
□	0x00400010	0x3c011001	lui $1,4097	11: la $t9, int1
□	0x00400014	0x34390000	ori $25,$1,0	
□	0x00400018	0x8f240000	lw $4,0($25)	12: lw $a0, 0($t9)
□	0x0040001c	0x3c011001	lui $1,4097	13: la $t9, int2
□	0x00400020	0x34390004	ori $25,$1,4	
□	0x00400024	0x8f250000	lw $5,0($25)	14: lw $a1, 0($t9)
□	0x00400028	0x3c010001	lui $1,1	15: la $ra, 0x12345
□	0x0040002c	0x343f2345	ori $31,$1,9029	
□	0x00400030	0x08100012	j 4194376	16: j sum2ints
□	0x00400034	0x00022021	addu $4,$0,$2	17: move $a0, $v0
□	0x00400038	0x24020001	addiu $2,$0,1	18: li $v0, 1
□	0x0040003c	0x0000000c	syscall	19: syscall
□	0x00400040	0x2402000a	addiu $2,$0,10	20: li $v0, 10
□	0x00400044	0x0000000c	syscall	21: syscall
□	0x00400048	0x00851020	add $2,$4,$5	23: sum2ints:add $v0, $a0, $a1
□	0x0040004c	0x03e00008	jr $31	24: jr $ra

bad guess
must update

MIPS32 AL – Stack Segment & Functions Prelim

(let's try doing it)

```
                  .data
int1:             .word 123
int2:             .word 234
out_msg:          .asciiz "\nSum: "
                  .text
                  .globl main
main:
                  la $a0, out_msg
                  li $v0, 4
                  syscall
                  la $t9, int1
                  lw $a0, 0($t9)
                  la $t9, int2
                  lw $a1, 0($t9)
                  la $ra, 0x00400034
                  j sum2ints
                  move $a0, $v0
                  li $v0, 1
                  syscall
                  la $a0, out_msg
                  li $v0, 4
                  syscall
```

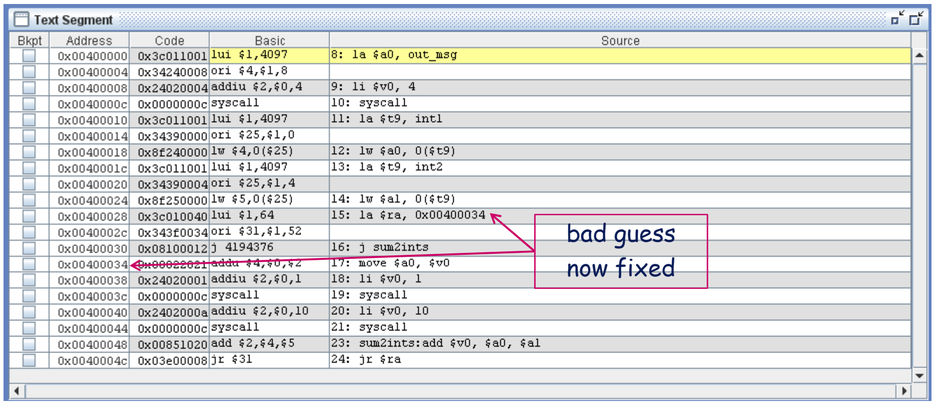
(continued on the right)

```
sum2ints:         add $v0, $a0, $a1
                  jr $ra
```

updated from
0x12345

MIPS32 AL – Stack Segment & Functions Prelim

(let's try doing it)



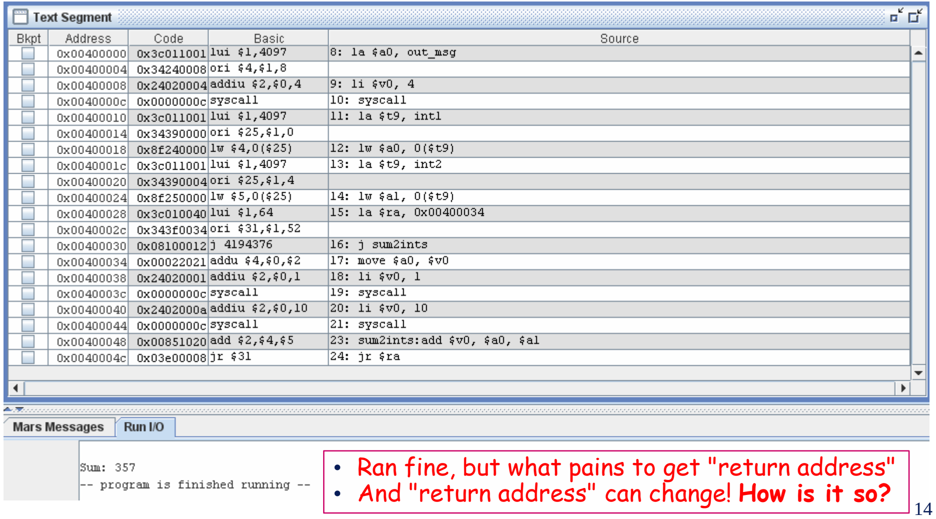
Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x3c011001	lui $1,4097	8: la $a0, out_msg
□	0x00400004	0x34240008	ori $4,$1,8	
□	0x00400008	0x24020004	addiu $2,$0,4	9: li $v0, 4
□	0x0040000c	0x0000000c	syscall	10: syscall
□	0x00400010	0x3c011001	lui $1,4097	11: la $t9, int1
□	0x00400014	0x34390000	ori $25,$1,0	
□	0x00400018	0x8f240000	lw $4,0($25)	12: lw $a0, 0($t9)
□	0x0040001c	0x3c011001	lui $1,4097	13: la $t9, int2
□	0x00400020	0x34390004	ori $25,$1,4	
□	0x00400024	0x8f250000	lw $5,0($25)	14: lw $a1, 0($t9)
□	0x00400028	0x3c010040	lui $1,64	15: la $ra, 0x00400034
□	0x0040002c	0x343f0034	ori $31,$1,52	
□	0x00400030	0x08100012	j 4194376	16: j sum2ints
□	0x00400034	0x00022021	addu $4,$0,$2	17: move $a0, $v0
□	0x00400038	0x24020001	addiu $2,$0,1	18: li $v0, 1
□	0x0040003c	0x0000000c	syscall	19: syscall
□	0x00400040	0x2402000a	addiu $2,$0,10	20: li $v0, 10
□	0x00400044	0x0000000c	syscall	21: syscall
□	0x00400048	0x00851020	add $2,$4,$5	23: sum2ints:add $v0, $a0, $a1
□	0x0040004c	0x03e00008	jr $31	24: jr $ra

bad guess
now fixed

MIPS32 AL – Stack Segment & Functions Prelim

(let's try doing it)



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x3c011001	lui $1,4097	8: la $a0, out_msg
□	0x00400004	0x34240008	ori $4,$1,8	
□	0x00400008	0x24020004	addiu $2,$0,4	9: li $v0, 4
□	0x0040000c	0x0000000c	syscall	10: syscall
□	0x00400010	0x3c011001	lui $1,4097	11: la $t9, int1
□	0x00400014	0x34390000	ori $25,$1,0	
□	0x00400018	0x8f240000	lw $4,0($25)	12: lw $a0, 0($t9)
□	0x0040001c	0x3c011001	lui $1,4097	13: la $t9, int2
□	0x00400020	0x34390004	ori $25,$1,4	
□	0x00400024	0x8f250000	lw $5,0($25)	14: lw $a1, 0($t9)
□	0x00400028	0x3c010040	lui $1,64	15: la $ra, 0x00400034
□	0x0040002c	0x343f0034	ori $31,$1,52	
□	0x00400030	0x08100012	j 4194376	16: j sum2ints
□	0x00400034	0x00022021	addu $4,$0,$2	17: move $a0, $v0
□	0x00400038	0x24020001	addiu $2,$0,1	18: li $v0, 1
□	0x0040003c	0x0000000c	syscall	19: syscall
□	0x00400040	0x2402000a	addiu $2,$0,10	20: li $v0, 10
□	0x00400044	0x0000000c	syscall	21: syscall
□	0x00400048	0x00851020	add $2,$4,$5	23: sum2ints:add $v0, $a0, $a1
□	0x0040004c	0x03e00008	jr $31	24: jr $ra

```
Sum: 357
-- program is finished running --
```

- Ran fine, but what pains to get "return address"
- And "return address" can change! **How is it so?**

MIPS32 AL – Stack Segment & Functions Prelim

(let's make some changes and see)

```
                    .data
int1:               .word 123
int2:               .word 234
int3:               .word 345
out_msg:            .asciiz "\nSum: "
                    .text
                    .globl main
main:               la $a0, out_msg
                    li $v0, 4
                    syscall
                    la $t9, int1
                    lw $a0, 0($t9)
                    la $t9, int2
                    lw $a1, 0($t9)
                    la $t9, int3
                    lw $a2, 0($t9)
                    la $ra, 0x12345
                    j sum2ints
                    move $a0, $v0
                    li $v0, 1
                    syscall
                    la $a0, out_msg
                    li $v0, 4
                    syscall
```

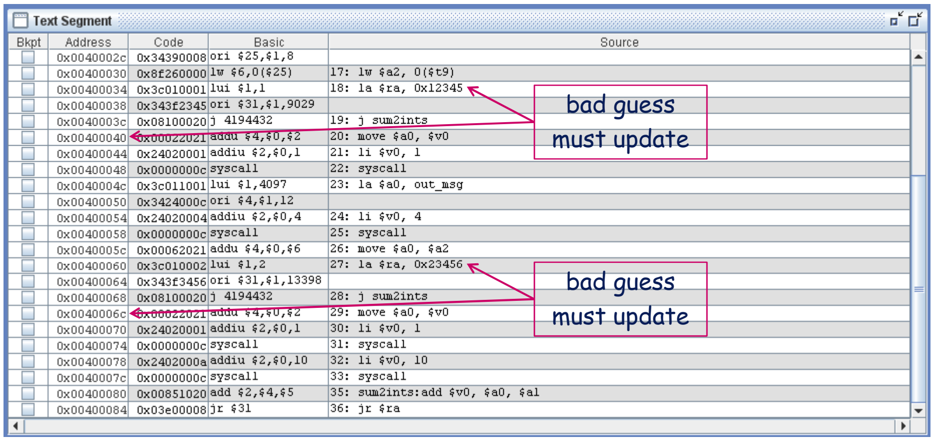
(continued on the right)

```
                    move $a0, $a2
                    la $ra, 0x23456
                    j sum2ints
                    move $a0, $v0
                    li $v0, 1
                    syscall
                    li $v0, 10
                    syscall

sum2ints:           add $v0, $a0, $a1
                    jr $ra
```

fill in some guess addresses so it will compile in MARS

MIPS32 AL – Stack Segment & Functions Prelim

(let's make some changes and see)



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x0040002c	0x34390008	ori $25,$1,8	
□	0x00400030	0x8f260000	lw $6,0($25)	17: lw $a2, 0($t9)
□	0x00400034	0x3c010001	lui $1,1	18: la $ra, 0x12345
□	0x00400038	0x343f2345	ori $31,$1,9029	
□	0x0040003c	0x08100020	j 4194432	19: j sum2ints
□	0x00400040	0x00022021	addu $4,$0,$2	20: move $a0, $v0
□	0x00400044	0x24020001	addiu $2,$0,1	21: li $v0, 1
□	0x00400048	0x0000000c	syscall	22: syscall
□	0x0040004c	0x3c011001	lui $1,4097	23: la $a0, out_msg
□	0x00400050	0x3424000c	ori $4,$1,12	
□	0x00400054	0x24020004	addiu $2,$0,4	24: li $v0, 4
□	0x00400058	0x0000000c	syscall	25: syscall
□	0x0040005c	0x00062021	addu $4,$0,$6	26: move $a0, $a2
□	0x00400060	0x3c010002	lui $1,2	27: la $ra, 0x23456
□	0x00400064	0x343f3456	ori $31,$1,13398	
□	0x00400068	0x08100020	j 4194432	28: j sum2ints
□	0x0040006c	0x00022021	addu $4,$0,$2	29: move $a0, $v0
□	0x00400070	0x24020001	addiu $2,$0,1	30: li $v0, 1
□	0x00400074	0x0000000c	syscall	31: syscall
□	0x00400078	0x2402000a	addiu $2,$0,10	32: li $v0, 10
□	0x0040007c	0x0000000c	syscall	33: syscall
□	0x00400080	0x00851020	add $2,$4,$5	35: sum2ints:add $v0, $a0, $a1
□	0x00400084	0x03e00008	jr $31	36: jr $ra

bad guess must update

bad guess must update

MIPS32 AL – Stack Segment & Functions Prelim

(let's make some changes and see)

```
                     .data
int1:                .word 123
int2:                .word 234
int3:                .word 345
out_msg:             .asciiz "\nSum: "
                     .text
                     .globl main
main:                la $a0, out_msg
                     li $v0, 4
                     syscall
                     la $t9, int1
                     lw $a0, 0($t9)
                     la $t9, int2
                     lw $a1, 0($t9)
                     la $t9, int3
                     lw $a2, 0($t9)
                     la $ra, 0x00400040
                     j sum2ints
                     move $a0, $v0
                     li $v0, 1
                     syscall
                     la $a0, out_msg
                     li $v0, 4
                     syscall
```

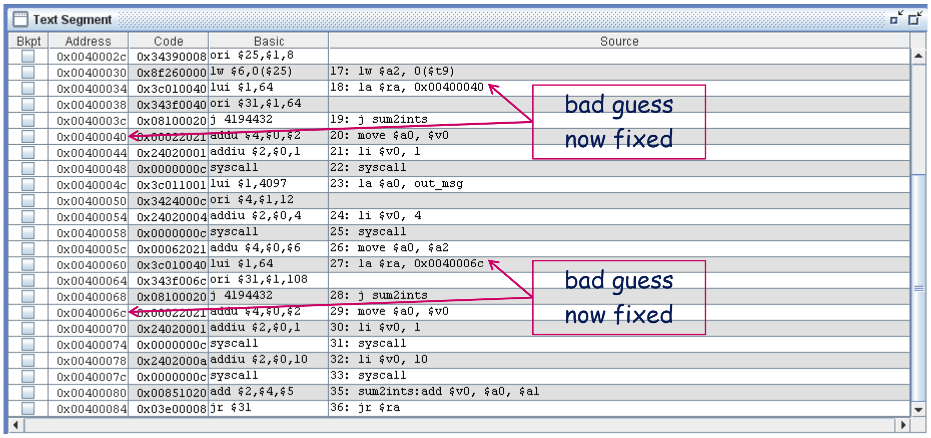
(continued on the right)

```
                     move $a0, $a2
                     la $ra, 0x0040006c
                     j sum2ints
                     move $a0, $v0
                     li $v0, 1
                     syscall
                     li $v0, 10
                     syscall

sum2ints:            add $v0, $a0, $a1
                     jr $ra
```

updated from
0x12345 and 0x23456

MIPS32 AL – Stack Segment & Functions Prelim

(let's make some changes and see)



Text Segment

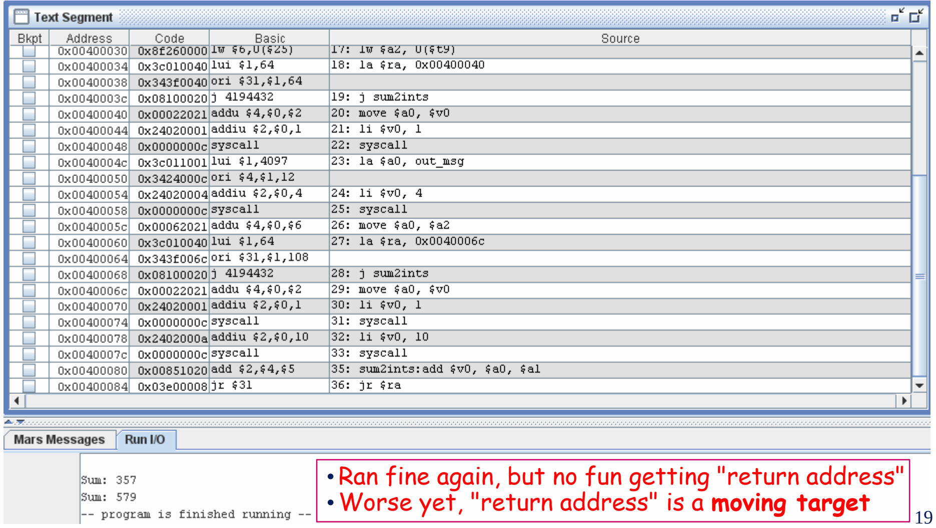
Bkpt	Address	Code	Basic	Source
□	0x0040002c	0x34390008	ori $25,$1,8	
□	0x00400030	0x8f260000	lw $6,0($25)	17: lw $a2, 0($t9)
□	0x00400034	0x3c010040	lui $1,64	18: la $ra, 0x00400040
□	0x00400038	0x343f0040	ori $31,$1,64	
□	0x0040003c	0x08100020	j 4194432	19: j sum2ints
□	0x00400040	0x00022021	addu $4,$0,$2	20: move $a0, $v0
□	0x00400044	0x24020001	addiu $2,$0,1	21: li $v0, 1
□	0x00400048	0x0000000c	syscall	22: syscall
□	0x0040004c	0x3c011001	lui $1,4097	23: la $a0, out_msg
□	0x00400050	0x3424000c	ori $4,$1,12	
□	0x00400054	0x24020004	addiu $2,$0,4	24: li $v0, 4
□	0x00400058	0x0000000c	syscall	25: syscall
□	0x0040005c	0x00062021	addu $4,$0,$6	26: move $a0, $a2
□	0x00400060	0x3c010040	lui $1,64	27: la $ra, 0x0040006c
□	0x00400064	0x343f006c	ori $31,$1,108	
□	0x00400068	0x08100020	j 4194432	28: j sum2ints
□	0x0040006c	0x00022021	addu $4,$0,$2	29: move $a0, $v0
□	0x00400070	0x24020001	addiu $2,$0,1	30: li $v0, 1
□	0x00400074	0x0000000c	syscall	31: syscall
□	0x00400078	0x2402000a	addiu $2,$0,10	32: li $v0, 10
□	0x0040007c	0x0000000c	syscall	33: syscall
□	0x00400080	0x00851020	add $2,$4,$5	35: sum2ints:add $v0, $a0, $a1
□	0x00400084	0x03e00008	jr $31	36: jr $ra

bad guess
now fixed

bad guess
now fixed

MIPS32 AL – Stack Segment & Functions Prelim

(let's make some changes and see)



Bkpt	Address	Code	Basic	Source
☐	0x00400030	0x8f260000	lw $6,0($25)	17: lw $a2, 0($t9)
☐	0x00400034	0x3c010040	lui $1,64	18: la $ra, 0x00400040
☐	0x00400038	0x343f0040	ori $31,$1,64	
☐	0x0040003c	0x08100020	j 4194432	19: j sum2ints
☐	0x00400040	0x00022021	addu $4,$0,$2	20: move $a0, $v0
☐	0x00400044	0x24020001	addiu $2,$0,1	21: li $v0, 1
☐	0x00400048	0x0000000c	syscall	22: syscall
☐	0x0040004c	0x3c011001	lui $1,4097	23: la $a0, out_msg
☐	0x00400050	0x3424000c	ori $4,$1,12	
☐	0x00400054	0x24020004	addiu $2,$0,4	24: li $v0, 4
☐	0x00400058	0x0000000c	syscall	25: syscall
☐	0x0040005c	0x00062021	addu $4,$0,$6	26: move $a0, $a2
☐	0x00400060	0x3c010040	lui $1,64	27: la $ra, 0x0040006c
☐	0x00400064	0x343f006c	ori $31,$1,108	
☐	0x00400068	0x08100020	j 4194432	28: j sum2ints
☐	0x0040006c	0x00022021	addu $4,$0,$2	29: move $a0, $v0
☐	0x00400070	0x24020001	addiu $2,$0,1	30: li $v0, 1
☐	0x00400074	0x0000000c	syscall	31: syscall
☐	0x00400078	0x2402000a	addiu $2,$0,10	32: li $v0, 10
☐	0x0040007c	0x0000000c	syscall	33: syscall
☐	0x00400080	0x00851020	add $2,$4,$5	35: sum2ints:add $v0, $a0, $a1
☐	0x00400084	0x03e00008	jr $31	36: jr $ra

```
Sum: 357
Sum: 579
-- program is finished running --
```

- Ran fine again, but no fun getting "return address"
- Worse yet, "return address" is a **moving target**

MIPS32 AL – Stack Segment & Functions Prelim

(**jal** to the rescue)

```
                        .data
int1:                   .word 123
int2:                   .word 234
int3:                   .word 345
out_msg:                .asciiz "\nSum: "
                        .text
                        .globl main
main:                   la $a0, out_msg
                        li $v0, 4
                        syscall
                        la $t9, int1
                        lw $a0, 0($t9)
                        la $t9, int2
                        lw $a1, 0($t9)
                        la $t9, int3
                        lw $a2, 0($t9)
                        jal sum2ints
                        move $a0, $v0
                        li $v0, 1
                        syscall
                        la $a0, out_msg
                        li $v0, 4
                        syscall
```

(continued on the right)

```
                        move $a0, $a2
                        jal sum2ints
                        move $a0, $v0
                        li $v0, 1
                        syscall
                        li $v0, 10
                        syscall

sum2ints:               add $v0, $a0, $a1
                        jr $ra
```

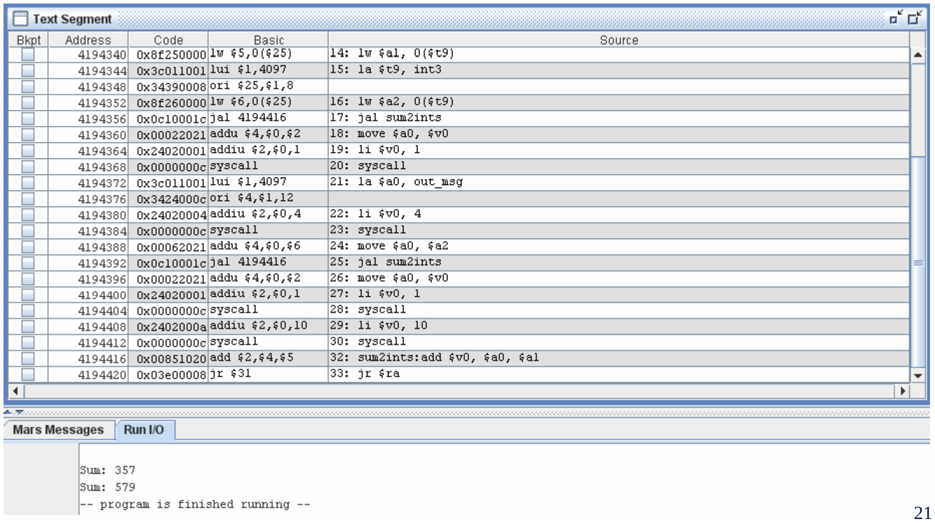
(previously)

```
la $ra, 0x0040006c
j sum2ints
```

(previously)

```
la $ra, 0x00400040
j sum2ints
```

MIPS32 AL – Stack Segment & Functions Prelim (jal to the rescue)

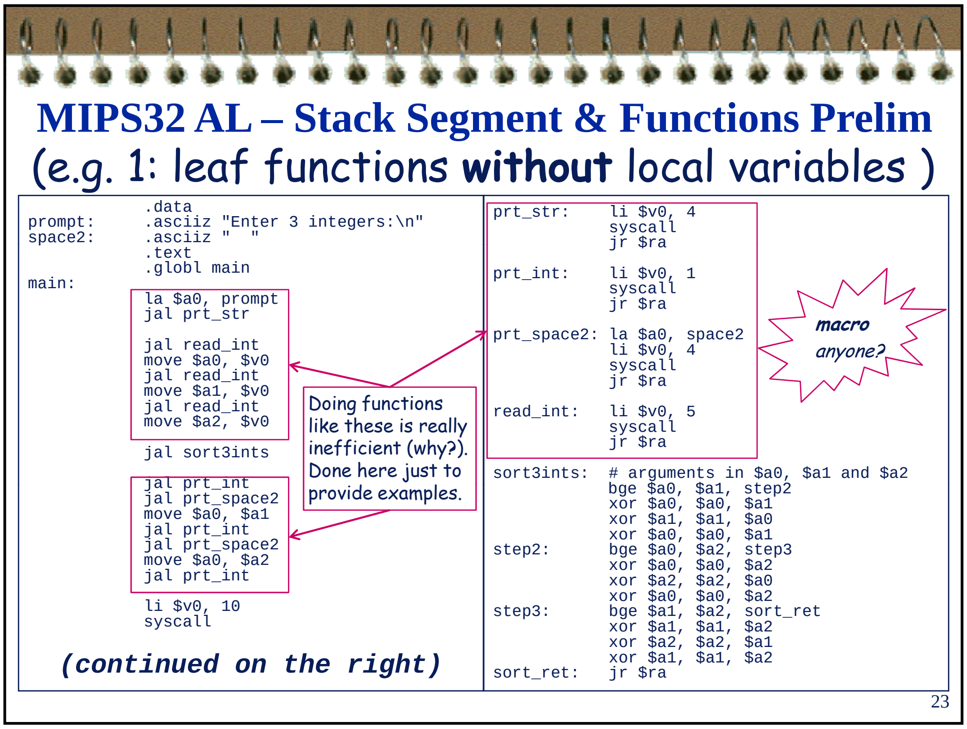


Bkpt	Address	Code	Basic	Source
□	4194340	0x8f250000	lw $5,0($25)	14: lw $a1, 0($t9)
□	4194344	0x3c011001	lui $1,4097	15: la $t9, int3
□	4194348	0x34390008	ori $25,$1,8	
□	4194352	0x8f260000	lw $6,0($25)	16: lw $a2, 0($t9)
□	4194356	0x0c10001c	jal 4194416	17: jal sum2ints
□	4194360	0x00022021	addu $4,$0,$2	18: move $a0, $v0
□	4194364	0x24020001	addiu $2,$0,1	19: li $v0, 1
□	4194368	0x0000000c	syscall	20: syscall
□	4194372	0x3c011001	lui $1,4097	21: la $a0, out_msg
□	4194376	0x3424000c	ori $4,$1,12	
□	4194380	0x24020004	addiu $2,$0,4	22: li $v0, 4
□	4194384	0x0000000c	syscall	23: syscall
□	4194388	0x00062021	addu $4,$0,$6	24: move $a0, $a2
□	4194392	0x0c10001c	jal 4194416	25: jal sum2ints
□	4194396	0x00022021	addu $4,$0,$2	26: move $a0, $v0
□	4194400	0x24020001	addiu $2,$0,1	27: li $v0, 1
□	4194404	0x0000000c	syscall	28: syscall
□	4194408	0x2402000a	addiu $2,$0,10	29: li $v0, 10
□	4194412	0x0000000c	syscall	30: syscall
□	4194416	0x00851020	add $2,$4,$5	32: sum2ints:add $v0, $a0, $a1
□	4194420	0x03e00008	jr $31	33: jr $ra

```
Sum: 357
Sum: 579
-- program is finished running --
```

MIPS32 AL – Stack Segment & Functions Prelim (MIPS instruction support for functions)

- Key:
  - Calling function (caller) does `jal <function_name>`
    - (`<function_name>` is label marking callee's 1st instruction)
    - Address (of instruction in caller) to return to *automatically* saved in `$ra`
  - Called function (callee) does `jr $ra` to return control to caller
- Effectively, `jal <function_name>` does 2 things:
  - ***Link*** part: ***saves address (of instruction) to return to into*** `$ra`
  - ***Jump*** part: `j <function_name>`
  - ✍ So, "link and jump" is perhaps a more appropriate name
- That should make doing functions in MIPS easy/fun
- (Or is it?)



MIPS32 AL – Stack Segment & Functions Prelim

(e.g. 1: leaf functions **without** local variables)

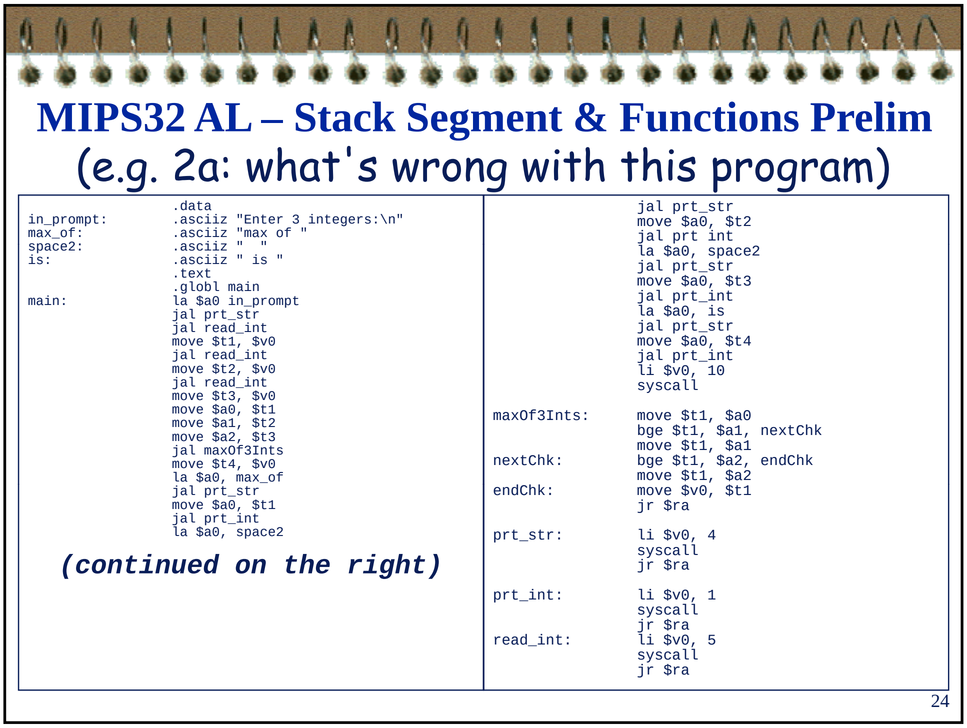
```
               .data
prompt:        .asciiz "Enter 3 integers:\n"
space2:        .asciiz "  "
               .text
               .globl main
main:
               la $a0, prompt
               jal prt_str

               jal read_int
               move $a0, $v0
               jal read_int
               move $a1, $v0
               jal read_int
               move $a2, $v0

               jal sort3ints

               jal prt_int
               jal prt_space2
               move $a0, $a1
               jal prt_int
               jal prt_space2
               move $a0, $a2
               jal prt_int

               li $v0, 10
               syscall
```

(continued on the right)

Doing functions like these is really inefficient (why?). Done here just to provide examples.

```
prt_str:       li $v0, 4
               syscall
               jr $ra

prt_int:       li $v0, 1
               syscall
               jr $ra

prt_space2:    la $a0, space2
               li $v0, 4
               syscall
               jr $ra

read_int:      li $v0, 5
               syscall
               jr $ra

sort3ints:     # arguments in $a0, $a1 and $a2
               bge $a0, $a1, step2
               xor $a0, $a0, $a1
               xor $a1, $a1, $a0
               xor $a0, $a0, $a1
step2:         bge $a0, $a2, step3
               xor $a0, $a0, $a2
               xor $a2, $a2, $a0
               xor $a0, $a0, $a2
step3:         bge $a1, $a2, sort_ret
               xor $a1, $a1, $a2
               xor $a2, $a2, $a1
               xor $a1, $a1, $a2
sort_ret:      jr $ra
```

macro anyone?



MIPS32 AL – Stack Segment & Functions Prelim

(e.g. 2a: what's wrong with this program)

```
                    .data
in_prompt:          .asciiz "Enter 3 integers:\n"
max_of:             .asciiz "max of "
space2:             .asciiz "  "
is:                 .asciiz " is "
                    .text
                    .globl main
main:               la $a0 in_prompt
                    jal prt_str
                    jal read_int
                    move $t1, $v0
                    jal read_int
                    move $t2, $v0
                    jal read_int
                    move $t3, $v0
                    move $a0, $t1
                    move $a1, $t2
                    move $a2, $t3
                    jal maxOf3Ints
                    move $t4, $v0
                    la $a0, max_of
                    jal prt_str
                    move $a0, $t1
                    jal prt_int
                    la $a0, space2
```

(continued on the right)

```
                    jal prt_str
                    move $a0, $t2
                    jal prt int
                    la $a0, space2
                    jal prt_str
                    move $a0, $t3
                    jal prt_int
                    la $a0, is
                    jal prt_str
                    move $a0, $t4
                    jal prt_int
                    li $v0, 10
                    syscall

maxOf3Ints:         move $t1, $a0
                    bge $t1, $a1, nextChk
                    move $t1, $a1
nextChk:            bge $t1, $a2, endChk
                    move $t1, $a2
endChk:             move $v0, $t1
                    jr $ra

prt_str:            li $v0, 4
                    syscall
                    jr $ra

prt_int:            li $v0, 1
                    syscall
                    jr $ra
read_int:           li $v0, 5
                    syscall
                    jr $ra
```

MIPS32 AL – Stack Segment & Functions Prelim

(e.g. 2b: is this good solution)

```
                  .data
in_prompt:        .asciiz "Enter 3 integers:\n"
max_of:           .asciiz "max of "
space2:           .asciiz "  "
is:               .asciiz " is "
                  .text
                  .globl main
main:             la $a0 in_prompt
                  jal prt_str
                  jal read_int
                  move $t1, $v0
                  jal read_int
                  move $t2, $v0
                  jal read_int
                  move $t3, $v0
                  move $a0, $t1
                  move $a1, $t2
                  move $a2, $t3
                  jal maxOf3Ints
                  move $t4, $v0
                  la $a0, max_of
                  jal prt_str
                  move $a0, $t1
                  jal prt_int
                  la $a0, space2
```

(continued on the right)

```
                  jal prt_str
                  move $a0, $t2
                  jal prt int
                  la $a0, space2
                  jal prt_str
                  move $a0, $t3
                  jal prt_int
                  la $a0, is
                  jal prt_str
                  move $a0, $t4
                  jal prt_int
                  li $v0, 10
                  syscall

maxOf3Ints:       move $t9, $a0
                  bge $t9, $a1, nextChk
                  move $t9, $a1
nextChk:          bge $t9, $a2, endChk
                  move $t9, $a2
endChk:           move $v0, $t9
                  jr $ra

prt_str:          li $v0, 4
                  syscall
                  jr $ra

prt_int:          li $v0, 1
                  syscall
                  jr $ra
read_int:         li $v0, 5
                  syscall
                  jr $ra
```

MIPS32 AL – Stack Segment & Functions Prelim

(e.g. 2c: to be safe, assume worst case)

```
                  .data
in_prompt:        .asciiz "Enter 3 integers:\n"
max_of:           .asciiz "max of "
space2:           .asciiz "  "
is:               .asciiz " is "
                  .text
                  .globl main
main:             la $a0 in_prompt
                  jal prt_str
                  jal read_int
                  move $t1, $v0
                  jal read_int
                  move $t2, $v0
                  jal read_int
                  move $t3, $v0
                  move $a0, $t1
                  move $a1, $t2
                  move $a2, $t3
                  addiu $sp, $sp, -12
                  sw $t1, 0($sp)
                  sw $t2, 4($sp)
                  sw $t3, 8($sp)
                  jal maxOf3Ints
                  move $t4, $v0
                  lw $t1, 0($sp)
                  lw $t2, 4($sp)
                  lw $t3, 8($sp)
                  addiu $sp, $sp, 12
                  la $a0, max_of
                  jal prt_str
                  move $a0, $t1
                  jal prt_int
                  la $a0, space2
```

(continued on the right)

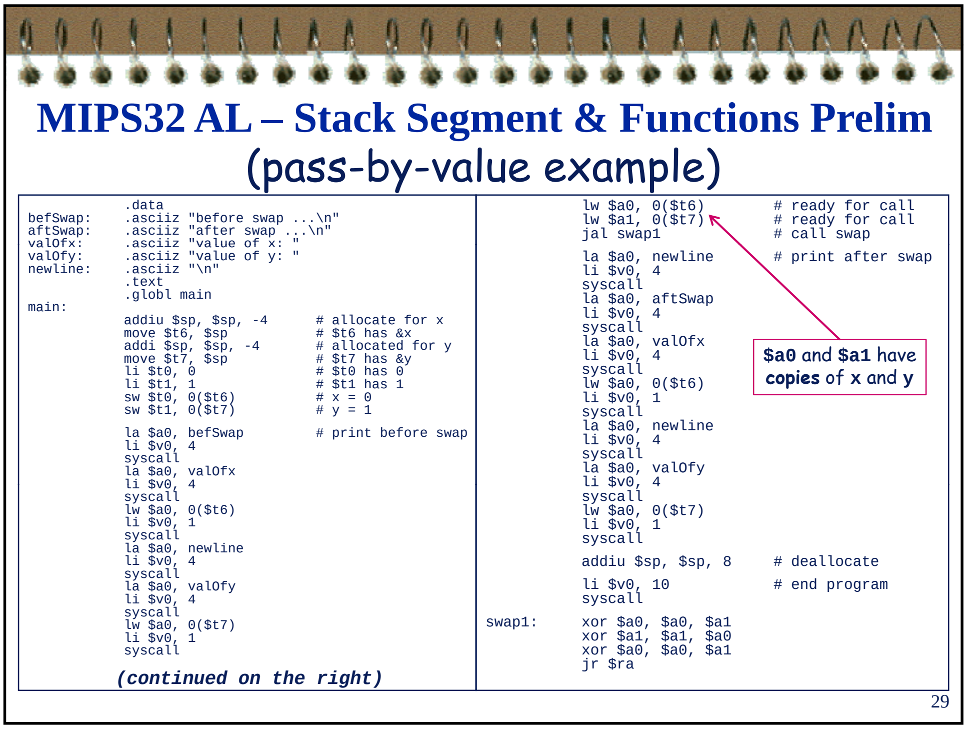
```
                  jal prt_str
                  move $a0, $t2
                  jal prt int
                  la $a0, space2
                  jal prt_str
                  move $a0, $t3
                  jal prt_int
                  la $a0, is
                  jal prt_str
                  move $a0, $t4
                  jal prt_int
                  li $v0, 10
                  syscall

maxOf3Ints:       move $t1, $a0
                  bge $t1, $a1, nextChk
                  move $t1, $a1
nextChk:          bge $t1, $a2, endChk
                  move $t1, $a2
endChk:           move $v0, $t1
                  jr $ra

prt_str:          li $v0, 4
                  syscall
                  jr $ra

prt_int:          li $v0, 1
                  syscall
                  jr $ra
read_int:         li $v0, 5
                  syscall
                  jr $ra
```

MIPS32 AL – Stack Segment & Functions Prelim
(pass by value vs pass by reference)

- Local variable in (***stack segment***) memory can be passed to function *by value* or *by reference*
  - To pass simple (non-array) local variable *by value*:
    - Load <u>*value*</u> of variable into relevant register (**`$a0`**, say) before calling
      - Function must be aware it's receiving (copy of) of variable's *value* and use it as such
  - To pass simple (non-array) local variable *by reference*:
    - Load <u>*address*</u> of variable into relevant register (**`$a0`**, say) before calling
      - Function must be aware it's receiving (copy of) variable's *address* and use it as such
  - Can pass *individual element* of local array by value or reference
    - Similar to passing simple (non-array) local variable described above
      - Load element's *value* or *address* (respectively) into relevant register before calling
  - *Entire local array* can be passed *by reference* through registers
    - By passing/providing (copy of) of array's starting *address* and array's *size*
- Food for thought:
  - How can we pass by value or reference *without using register?*
    - Related questions: How can we pass *entire local array by value*?
  - What about passing ***register*** *variable* & ***data segment*** *variable?*

MIPS32 AL – Stack Segment & Functions Prelim
(pass by value vs pass by reference)

- The determining factor
  - Whether function gets *copy of* or *address of* original variable
    - By value: function gets only a copy & has no way to change the original
    - By reference: function gets address & uses it to access the original
- Above always applies
  - Doesn't matter if variable is simple or composite (array)



MIPS32 AL – Stack Segment & Functions Prelim

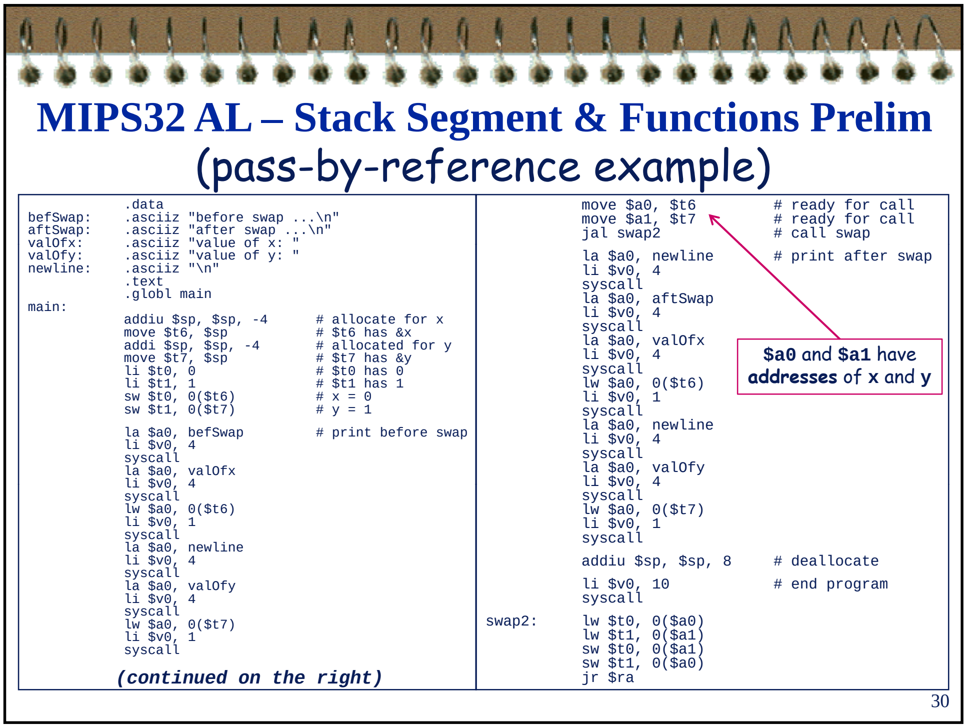
(pass-by-value example)

```
            .data
befSwap:    .asciiz "before swap ...\n"
aftSwap:    .asciiz "after swap ...\n"
valOfx:     .asciiz "value of x: "
valOfy:     .asciiz "value of y: "
newline:    .asciiz "\n"
            .text
            .globl main
main:
            addiu $sp, $sp, -4      # allocate for x
            move $t6, $sp           # $t6 has &x
            addi $sp, $sp, -4       # allocated for y
            move $t7, $sp           # $t7 has &y
            li $t0, 0               # $t0 has 0
            li $t1, 1               # $t1 has 1
            sw $t0, 0($t6)          # x = 0
            sw $t1, 0($t7)          # y = 1

            la $a0, befSwap         # print before swap
            li $v0, 4
            syscall
            la $a0, valOfx
            li $v0, 4
            syscall
            lw $a0, 0($t6)
            li $v0, 1
            syscall
            la $a0, newline
            li $v0, 4
            syscall
            la $a0, valOfy
            li $v0, 4
            syscall
            lw $a0, 0($t7)
            li $v0, 1
            syscall
```

(continued on the right)

```
            lw $a0, 0($t6)          # ready for call
            lw $a1, 0($t7)          # ready for call
            jal swap1               # call swap

            la $a0, newline         # print after swap
            li $v0, 4
            syscall
            la $a0, aftSwap
            li $v0, 4
            syscall
            la $a0, valOfx
            li $v0, 4
            syscall
            lw $a0, 0($t6)
            li $v0, 1
            syscall
            la $a0, newline
            li $v0, 4
            syscall
            la $a0, valOfy
            li $v0, 4
            syscall
            lw $a0, 0($t7)
            li $v0, 1
            syscall

            addiu $sp, $sp, 8       # deallocate

            li $v0, 10              # end program
            syscall

swap1:      xor $a0, $a0, $a1
            xor $a1, $a1, $a0
            xor $a0, $a0, $a1
            jr $ra
```

$a0 and **$a1** have **copies** of **x** and **y**



MIPS32 AL – Stack Segment & Functions Prelim

(pass-by-reference example)

```
            .data
befSwap:    .asciiz "before swap ...\n"
aftSwap:    .asciiz "after swap ...\n"
valOfx:     .asciiz "value of x: "
valOfy:     .asciiz "value of y: "
newline:    .asciiz "\n"
            .text
            .globl main
main:
            addiu $sp, $sp, -4      # allocate for x
            move $t6, $sp           # $t6 has &x
            addi $sp, $sp, -4       # allocated for y
            move $t7, $sp           # $t7 has &y
            li $t0, 0               # $t0 has 0
            li $t1, 1               # $t1 has 1
            sw $t0, 0($t6)          # x = 0
            sw $t1, 0($t7)          # y = 1

            la $a0, befSwap         # print before swap
            li $v0, 4
            syscall
            la $a0, valOfx
            li $v0, 4
            syscall
            lw $a0, 0($t6)
            li $v0, 1
            syscall
            la $a0, newline
            li $v0, 4
            syscall
            la $a0, valOfy
            li $v0, 4
            syscall
            lw $a0, 0($t7)
            li $v0, 1
            syscall
```

(continued on the right)

```
            move $a0, $t6           # ready for call
            move $a1, $t7           # ready for call
            jal swap2               # call swap

            la $a0, newline         # print after swap
            li $v0, 4
            syscall
            la $a0, aftSwap
            li $v0, 4
            syscall
            la $a0, valOfx
            li $v0, 4
            syscall
            lw $a0, 0($t6)
            li $v0, 1
            syscall
            la $a0, newline
            li $v0, 4
            syscall
            la $a0, valOfy
            li $v0, 4
            syscall
            lw $a0, 0($t7)
            li $v0, 1
            syscall

            addiu $sp, $sp, 8       # deallocate

            li $v0, 10              # end program
            syscall

swap2:      lw $t0, 0($a0)
            lw $t1, 0($a1)
            sw $t0, 0($a1)
            sw $t1, 0($a0)
            jr $ra
```

$a0 and **$a1** have **addresses** of **x** and **y**

MIPS32 AL – Stack Segment & Functions Prelim
(doing functions in MIPS: big picture)

- MIPS' provides limited function support in hardware
  - Programmers have to rely much on *function-call convention*
- No such thing as "The MIPS Calling Convention" ☹
  - Different conventions used by different programmers/assemblers
  - We'll study/use one convention similar to those
- Simple function (in light of convention we'll study/use):
  - *Leaf* <u>and</u> *has <=4 arguments* <u>and</u> *doesn't use local variables*
    - Leaf function → no **`jal`** instructions in its code
    - (what we have seen so far → no *stack-segment memory* needed)
- Complex function (in light of convention we'll study/use):
  - *Non-leaf* <u>and/or</u> *has >=5 arguments* <u>and/or</u> *uses local variables*
    - Non-leaf function → has **`jal`** instructions in its code
      - Includes *recursive* function
    - (will cover next → *stack-segment memory* needed)