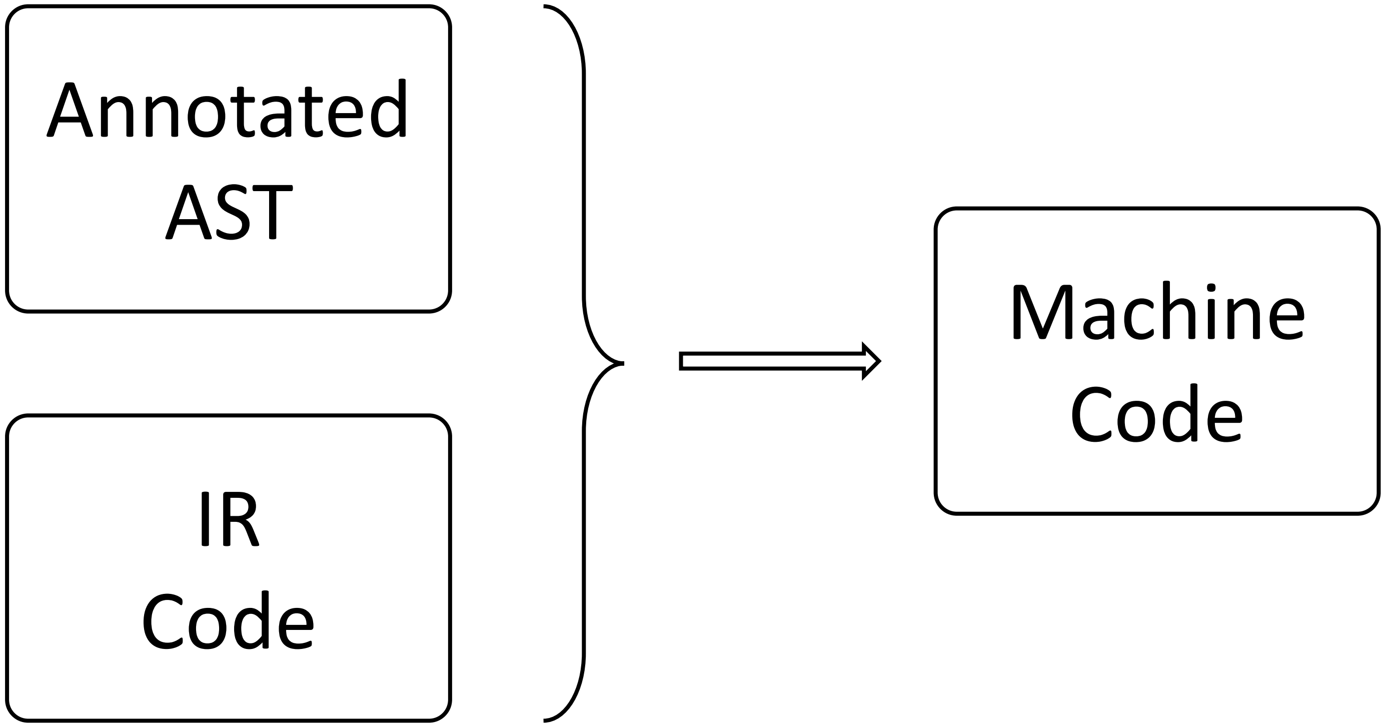
Compilation

0368-3133

Tutorial 10:

Code Generation

Code Generation



MIPS Architecture

- MIPS has 32 registers:
  - t0,..., t9 (general purpose)
  - a0, a1, a2, a3 (arguments)
  - v0 (return value, system calls)
  - sp (stack pointer)
  - fp (frame pointer)
  - ra (return address)
  - …

MIPS Architecture

- Labels

```
data_label_1: .word 17

data_label_2: .asciiz “abc”

code_label:
li $t0, 3
...
```

MIPS Architecture

- Basic assignments

```
li $t0, 3
move $t1, $t2
```

MIPS Architecture

- Arithmetic instructions
  - operate on registers and constants
  - add, sub, mul, div, and, or, xor, …

```
add $t2, $t0, $t1
mul $t3, t1, 7
```

MIPS Architecture

- Read from memory

```
lw $t0,4($t1)
lw $t0,label
lw $t0,label+4
lw $t0,label+8($t1)
```

MIPS Architecture

- Write to memory

```
sw $t0,2($t1)
sw $t0,label
sw $t0,label+4
sw $t0,label+8($t1)
```

MIPS Architecture

- Branches and Jumps

```
beq $t1, $t2, label
bne $t1, 7, label
j label
jal label
jalr $t1
```

MIPS Architecture

- System calls:
  - Syscall number passed via v0
  - Arguments are passed via a0, a1, a2, a3

- For example, calling **PrintInt(3)**:

```
li $v0, 1
li $a0, 3
syscall
```

SPIM: MIPS Simulator

```
.data
g_foo:  .word 17
g_str:  .asciiz “hello”
…
```

global data

```
.text
li $v0, 1
lw $a0, g_foo
syscall
li $v0, 4
la $a0, g_str
syscall
```

code

SPIM: MIPS Simulator

```
.data
g_foo:  .word 17
g_str:  .asciiz “hello”
…
```

global data

```
.text
li $v0, 1
lw $a0, g_foo
syscall
```

PrintInt(17)

```
li $v0, 4
la $a0, g_str
syscall
```

PrintStr(“hello”)

code

SPIM

- Running SPIM:

  ```
  > spim –f input_file
  ```

- Interactive debugging:

  ```
  > xspim
  ```

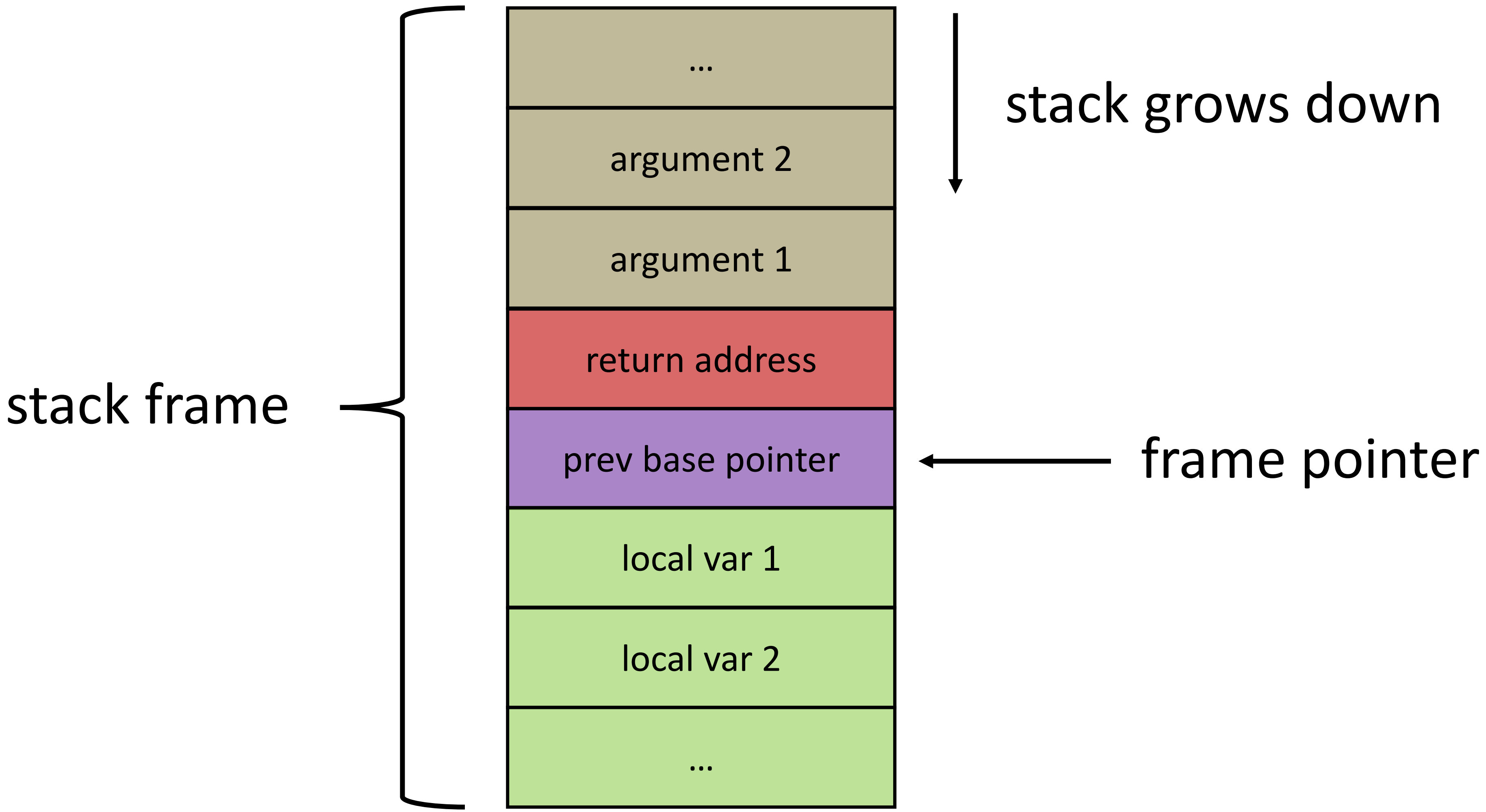
- Manual:
  - https://web.stanford.edu/class/cs143/materials/SPIM_Manual.pdf

Tutorial Roadmap

- **Today:**
  - Functions
  - Integers and global variables

- **Next week:** all about pointers
  - Strings
  - Arrays
  - Classes

Stack



…

argument 2

argument 1

return address

prev base pointer

local var 1

local var 2

…

stack frame

stack grows down

frame pointer

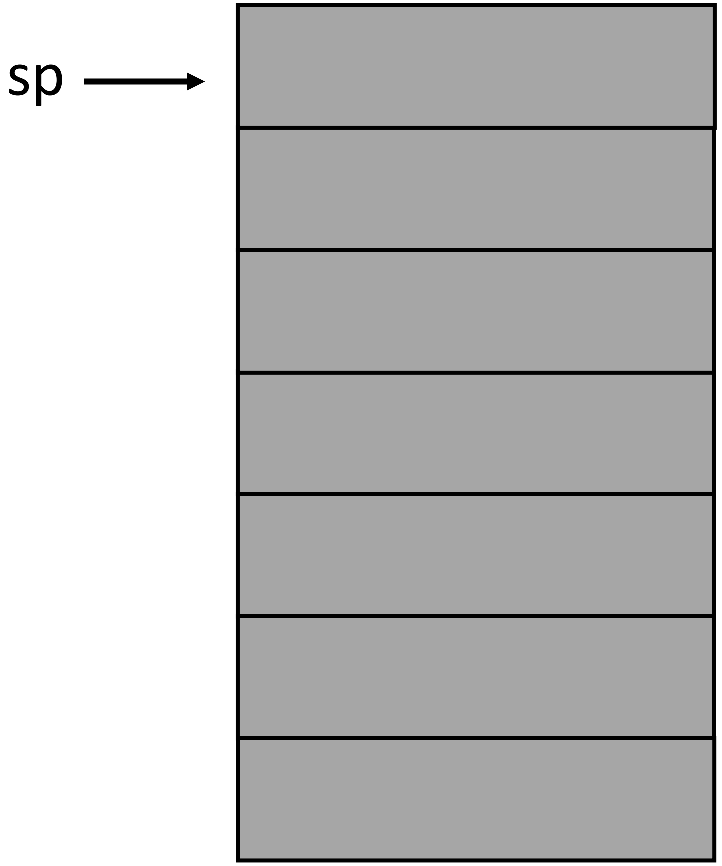
Stack

```
int f(int x, int y){
  int z = x + y;
  return z;
}
int g() {
  int x = f(10, 20)
}
```

```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

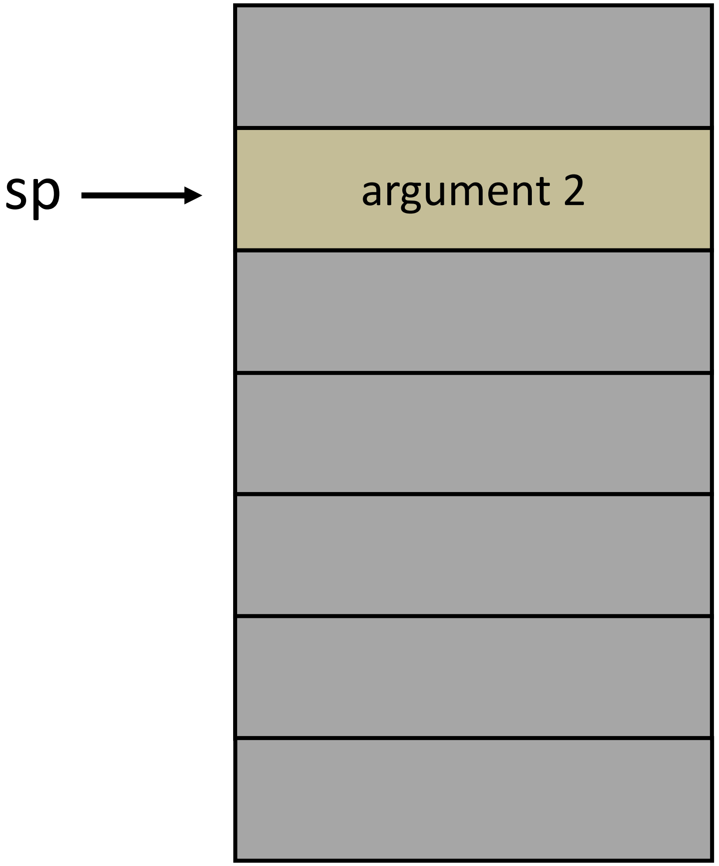
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

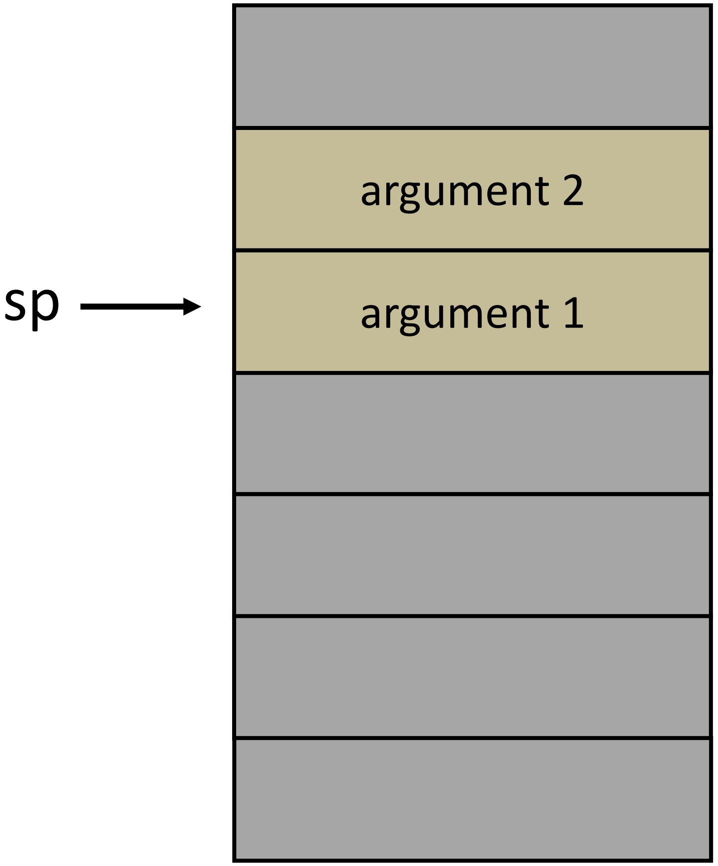
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

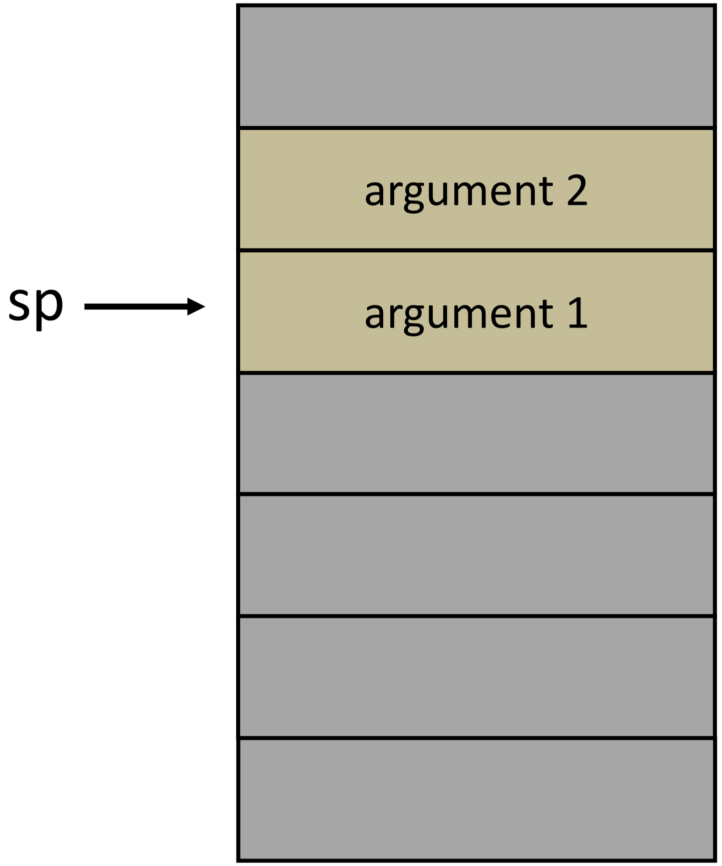
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

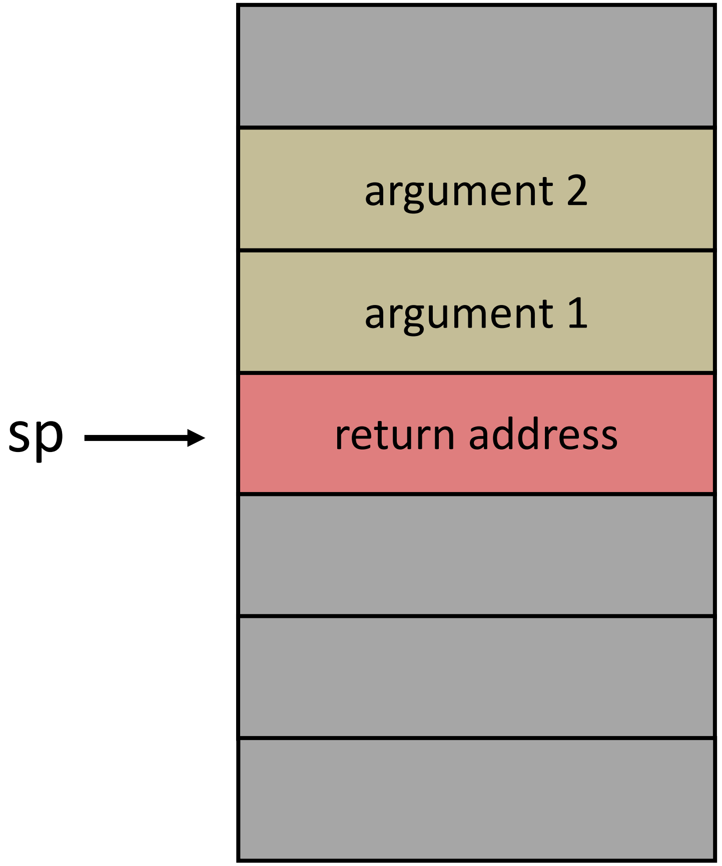
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

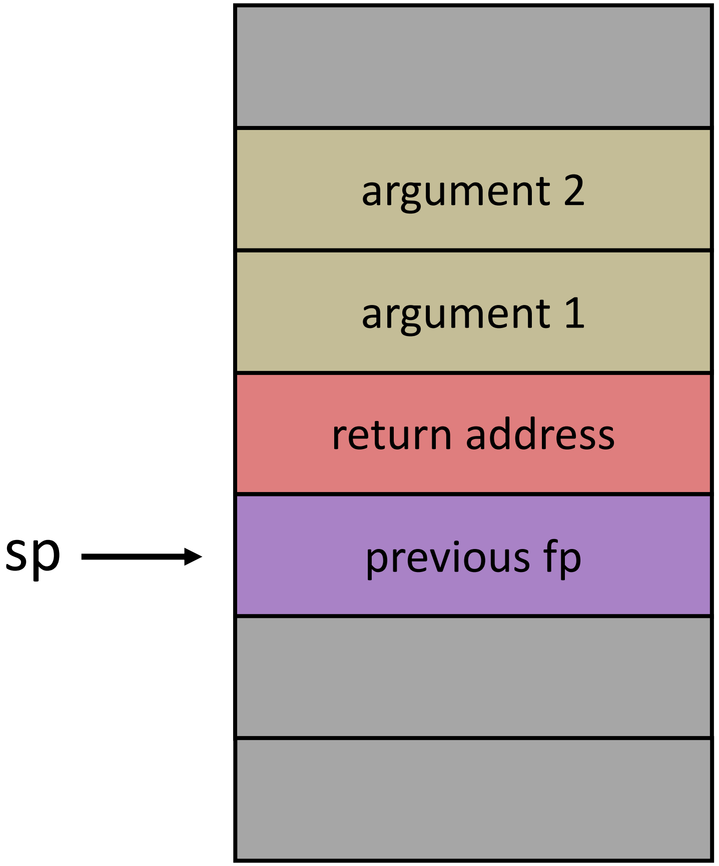
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

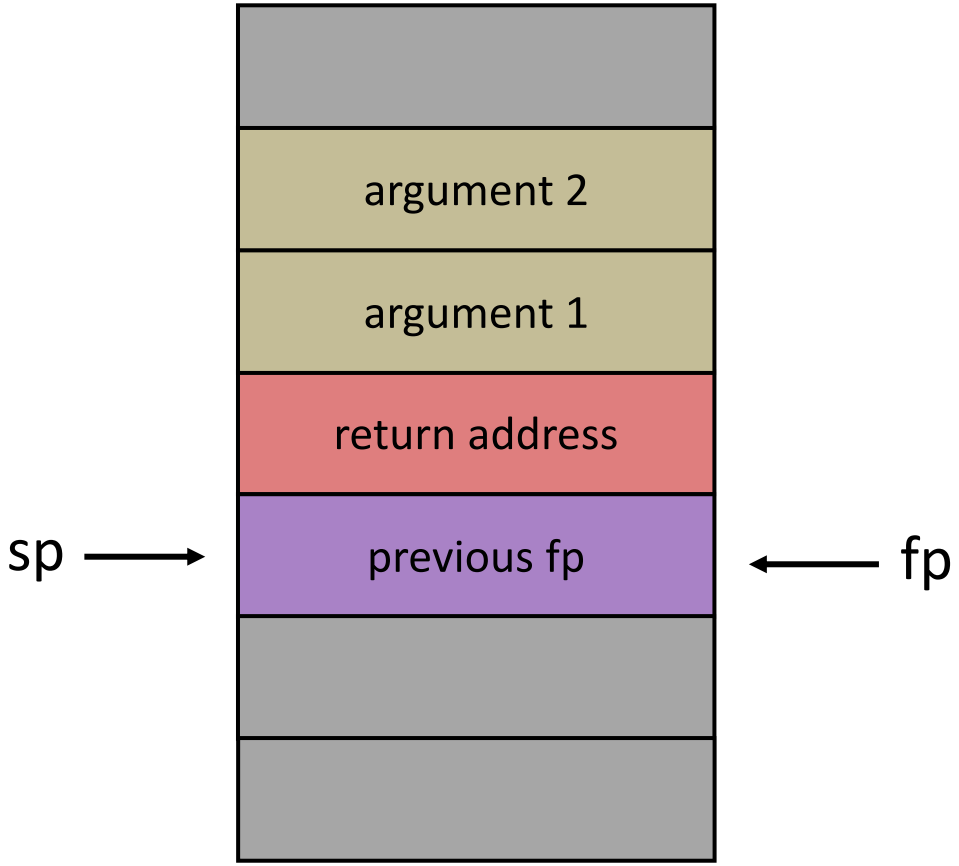
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

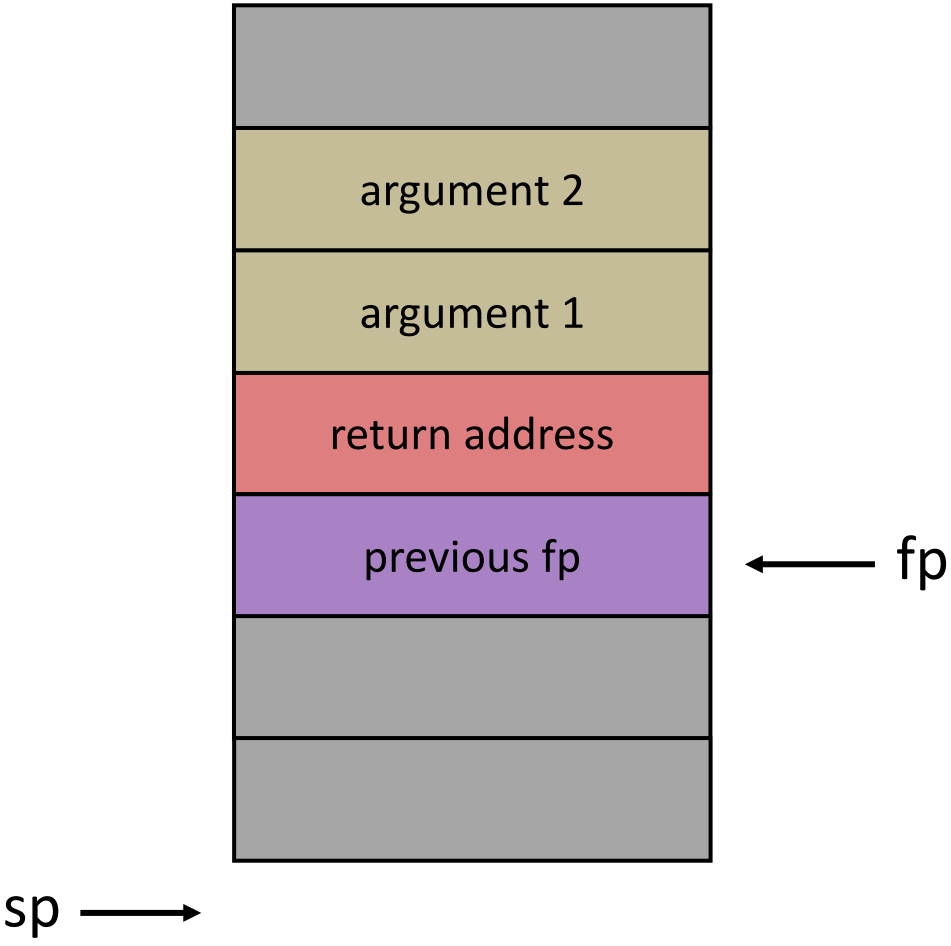
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

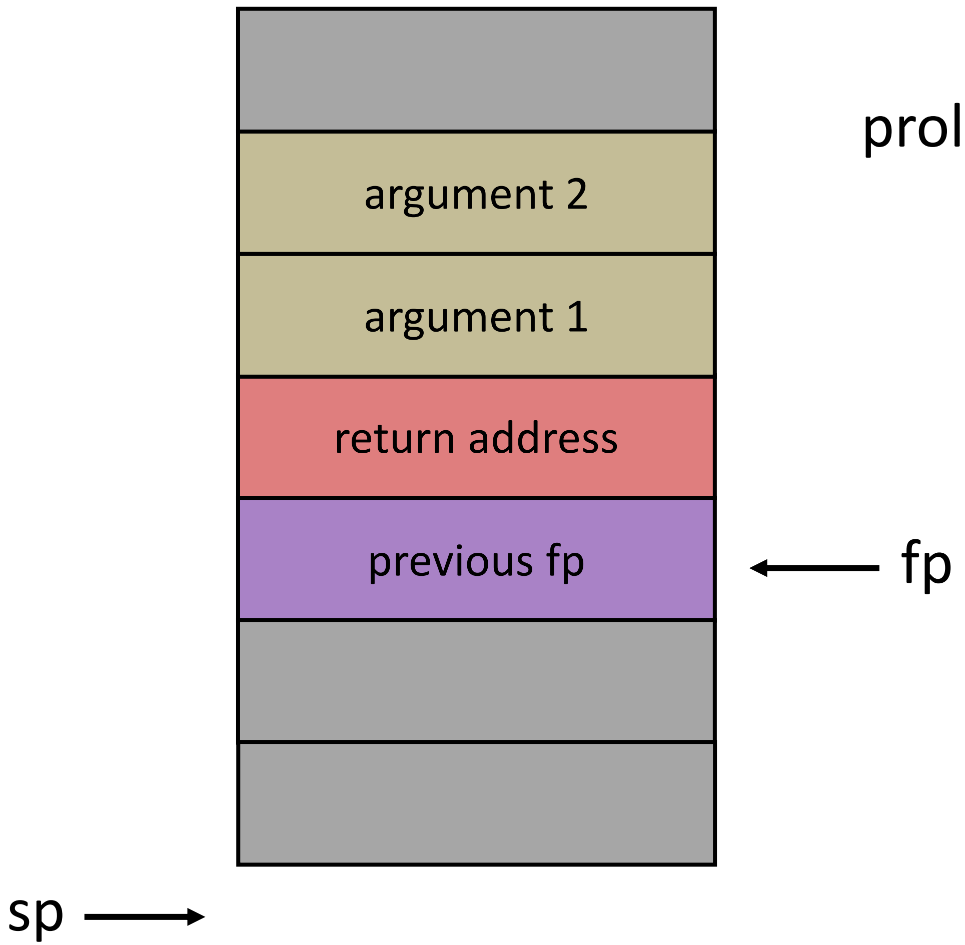
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

Stack

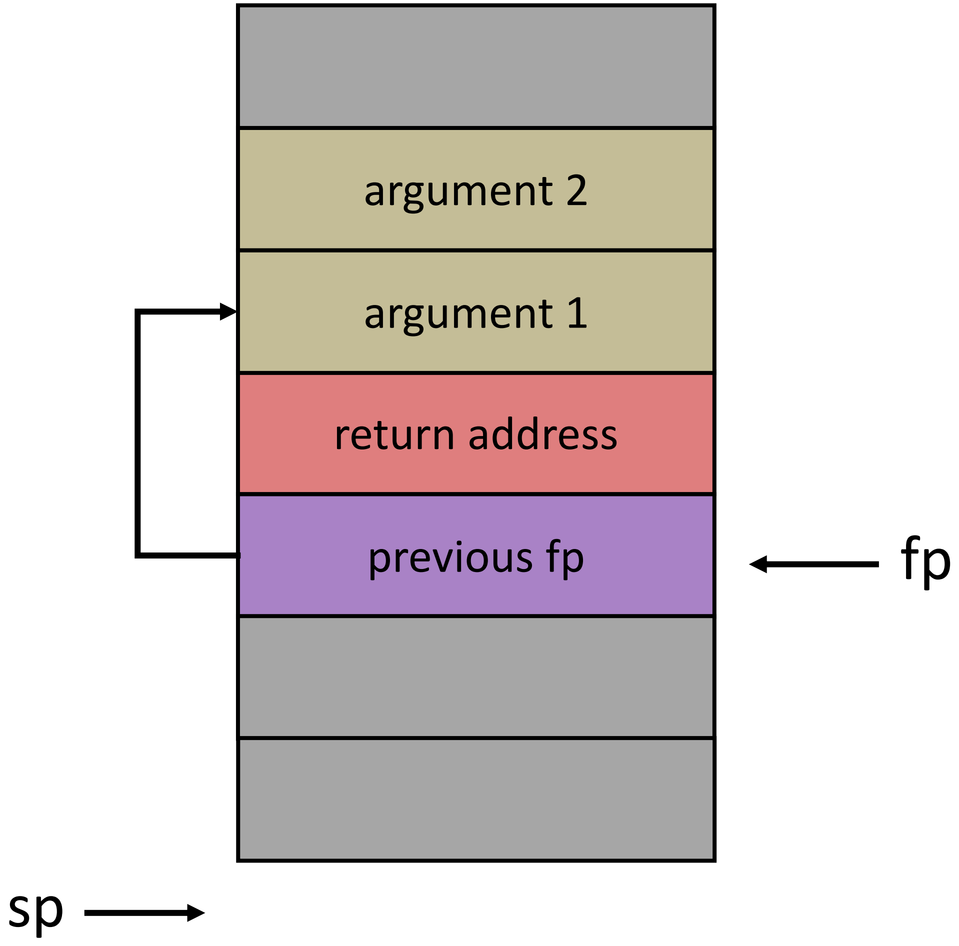


prologue

```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

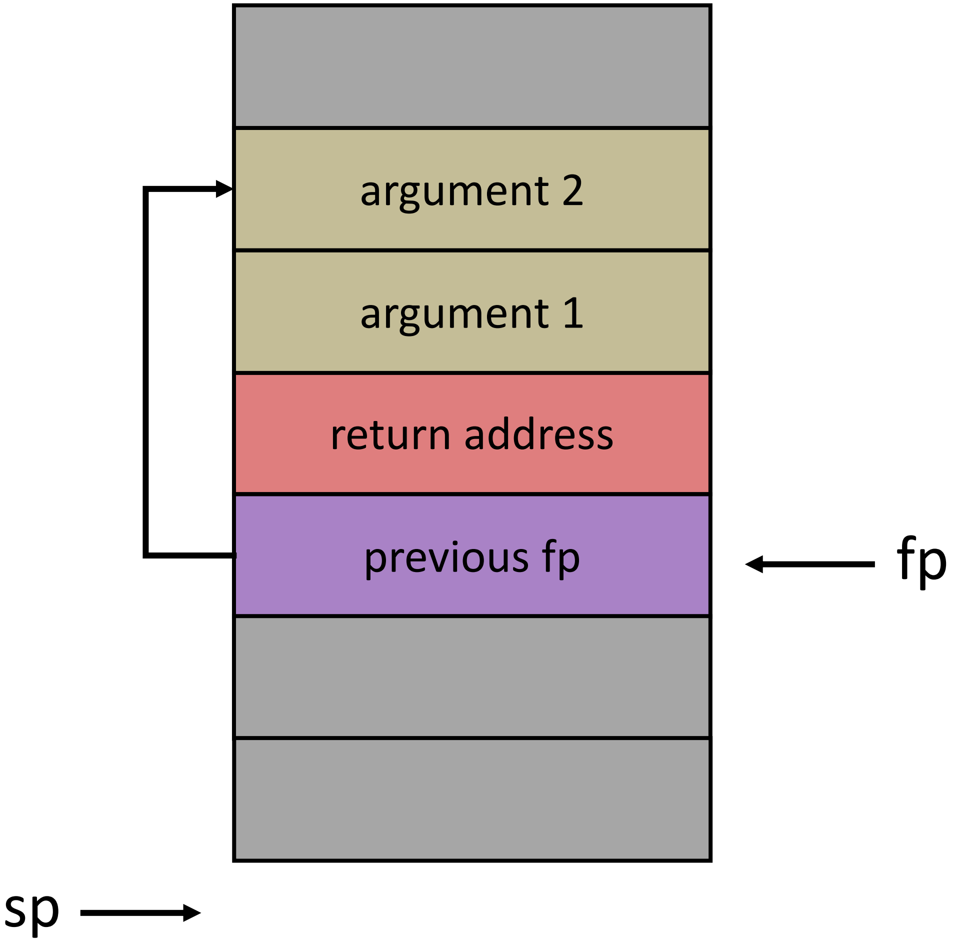
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

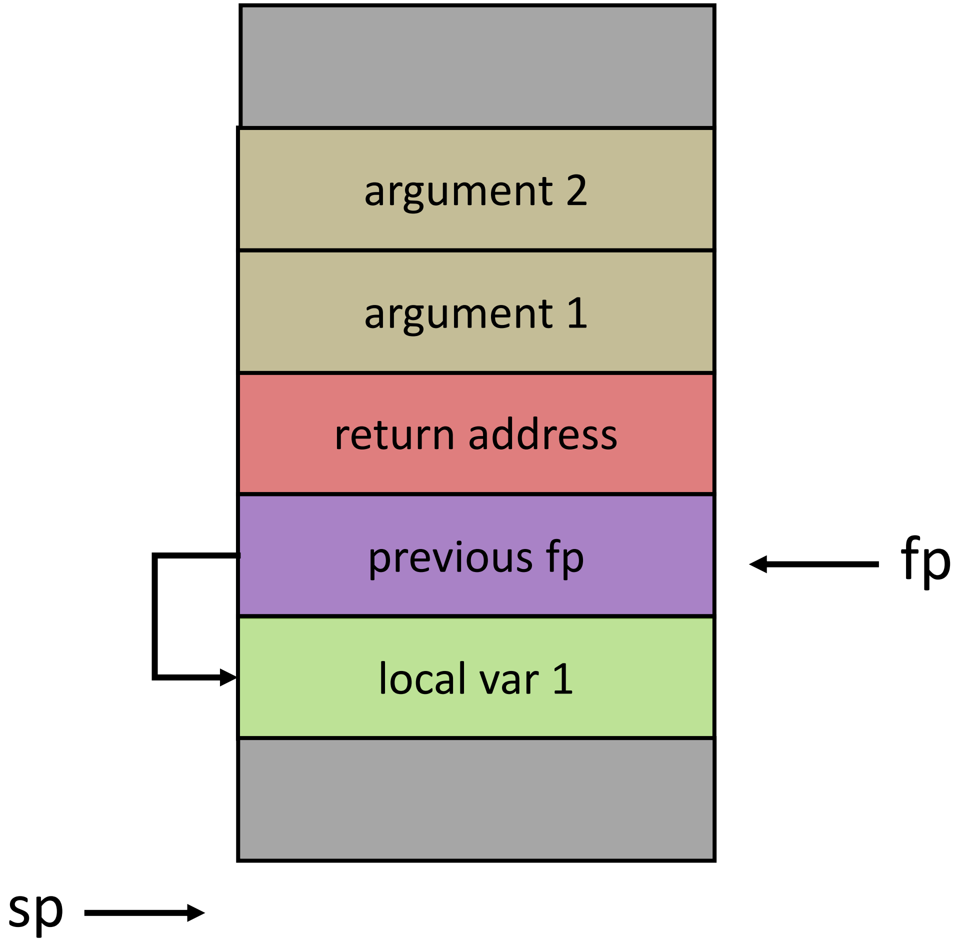
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

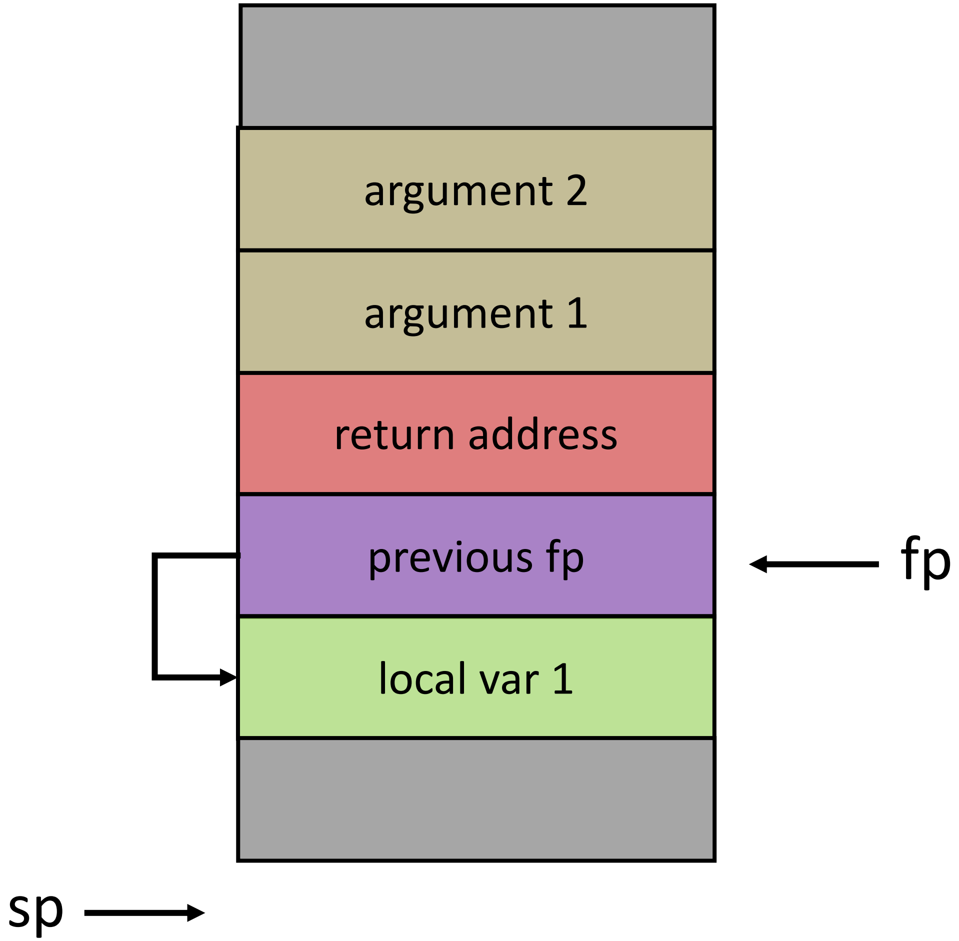
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

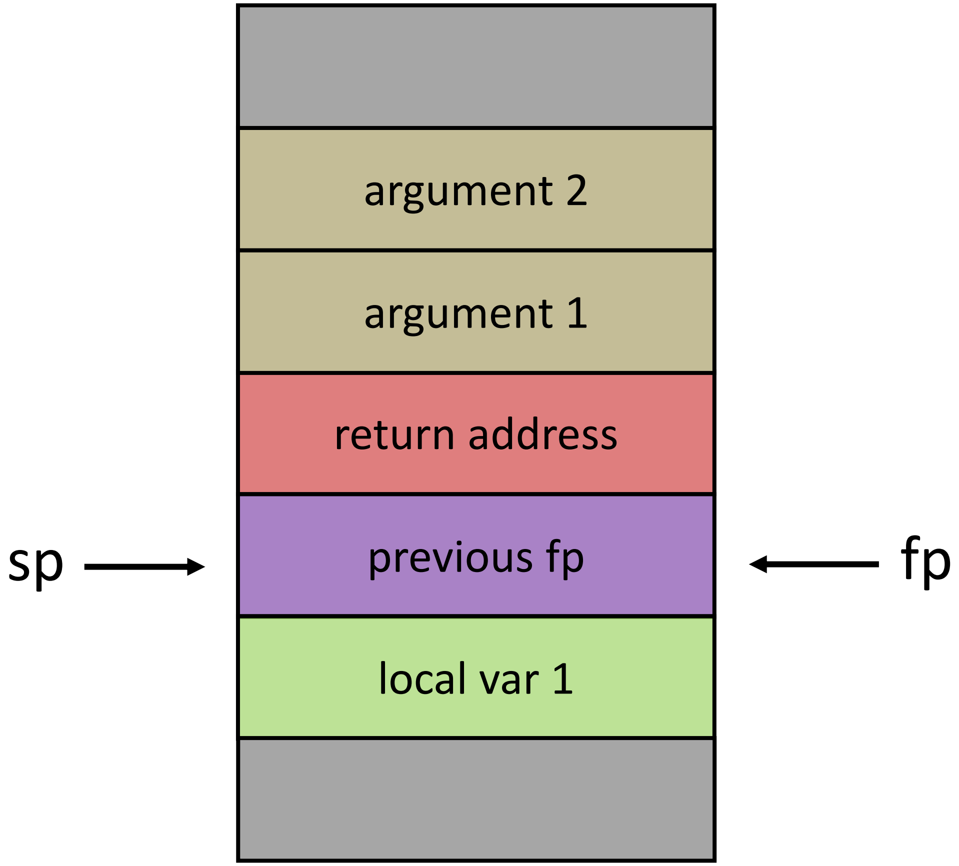
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

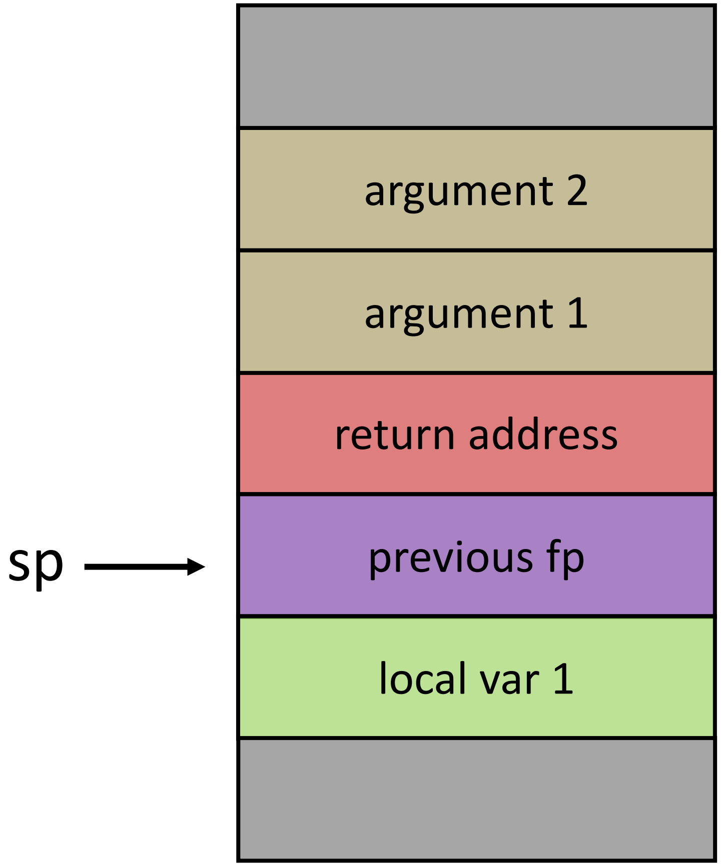
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

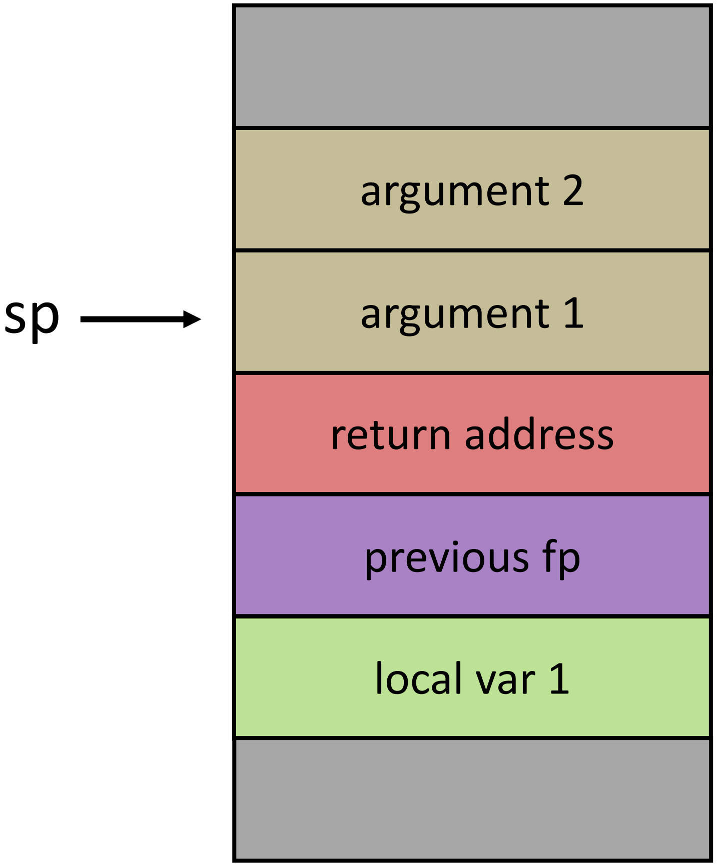
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

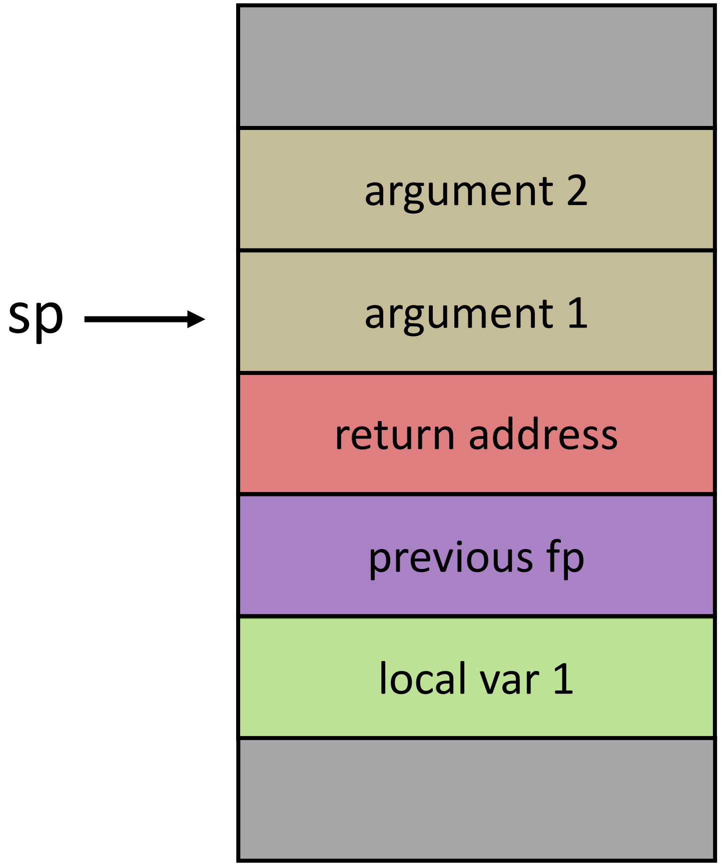
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

Stack



sp

argument 2

argument 1

return address

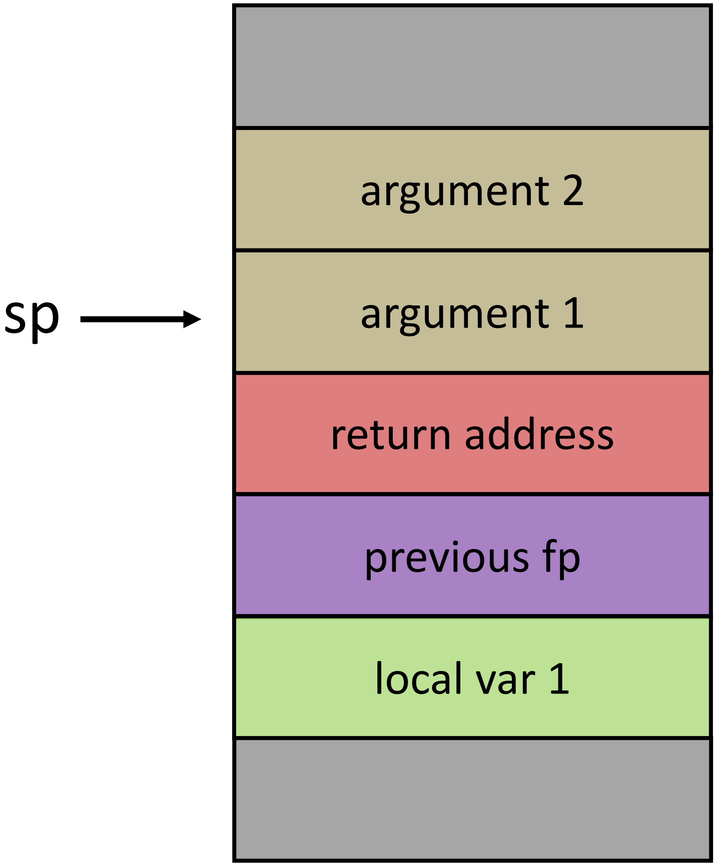
previous fp

local var 1

```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

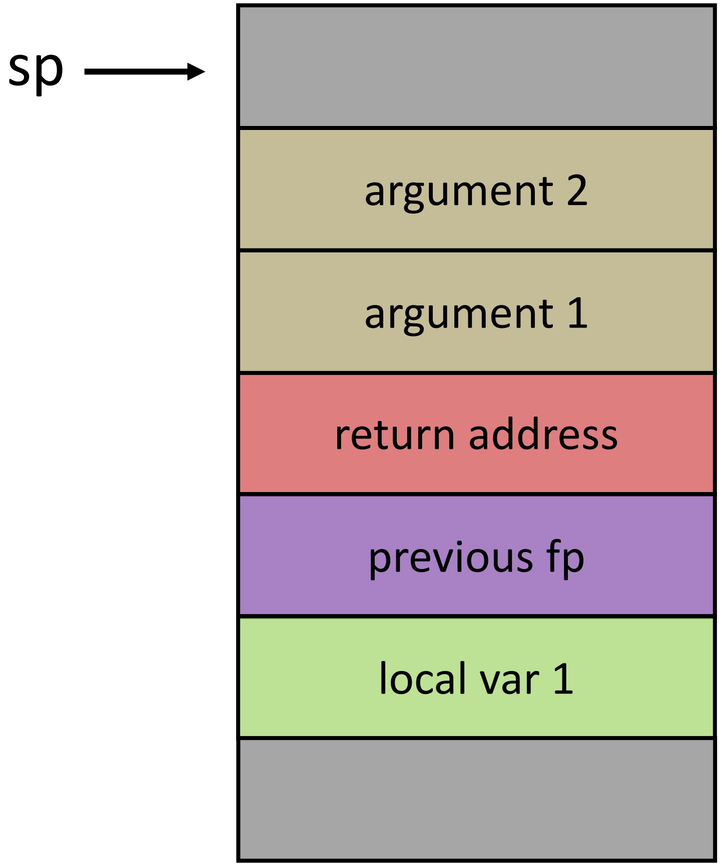
Stack



```
f:                          g:
subu $sp, $sp, 4            ...
sw $ra, 0($sp)              li $t0, 20
subu $sp, $sp, 4            subu $sp, $sp, 4
sw $fp, 0($sp)              sw $t0, 0($sp)
move $fp, $sp               li $t0, 10
subu $sp, $sp, 16           subu $sp, $sp, 4
lw $t0, 8($fp)              sw $t0, 0($sp)
lw $t1, 12($fp)             jal f
add $t2, $t0, $t1           addu $sp, $sp, 8
sw $t2, -4($fp)             move $t0, $v0
lw $v0, -4($fp)             ...
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

epilogue

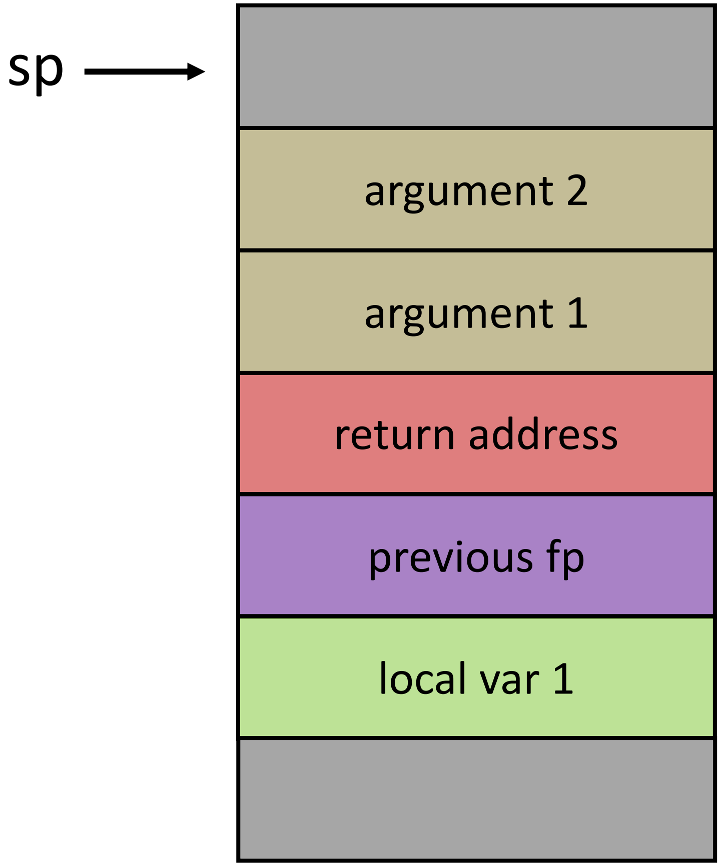
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

Stack

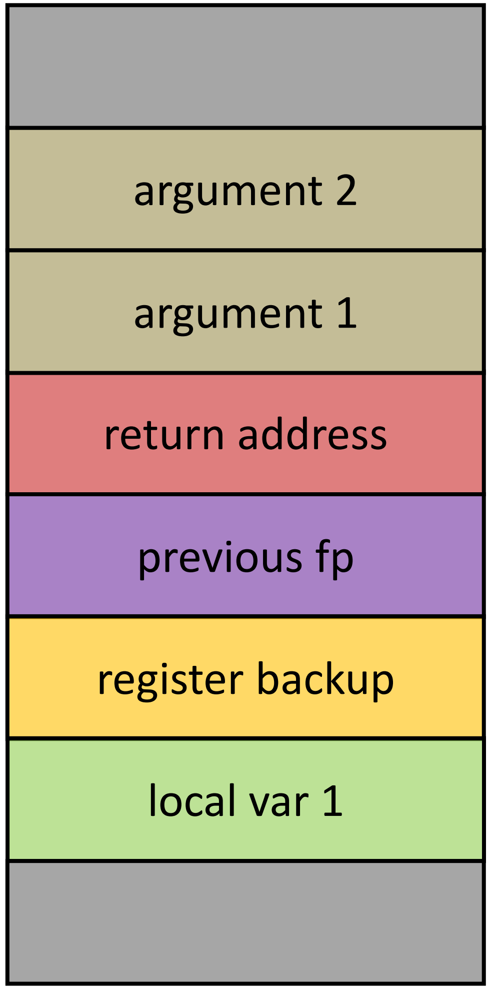


```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

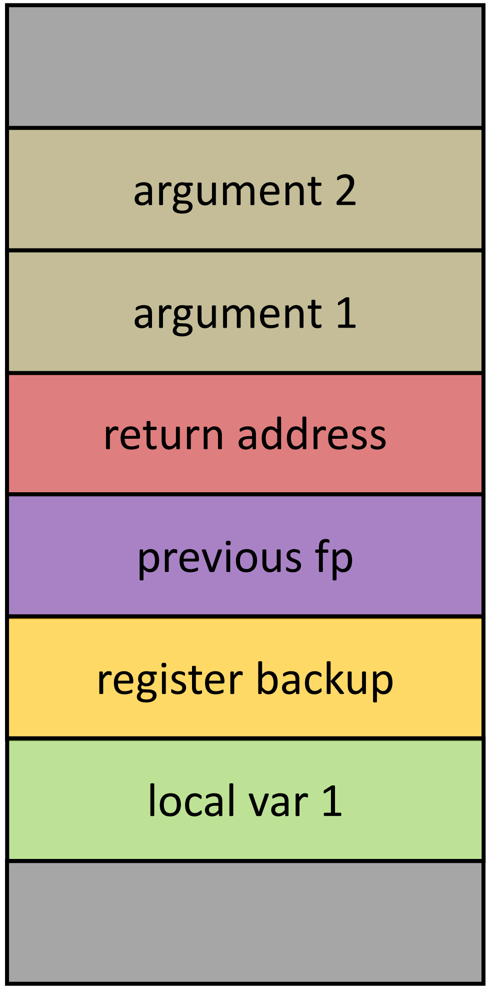
```
g:
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

Register Backup

- Called functions may modify registers
- Backup at the **prologue**
- Restore at the **epilogue**



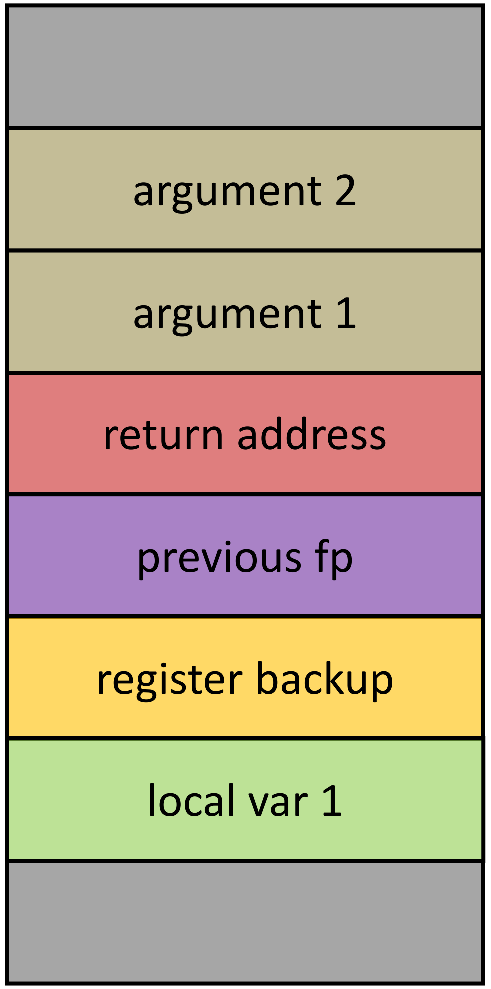
Register Backup: Prologue



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
subu $sp, $sp, 4
sw $t0, 0($sp)
…
subu $sp, $sp, 4
sw $t9, 0($sp)
subu $sp, $sp, 16
```

backup

Register Backup: Epilogue

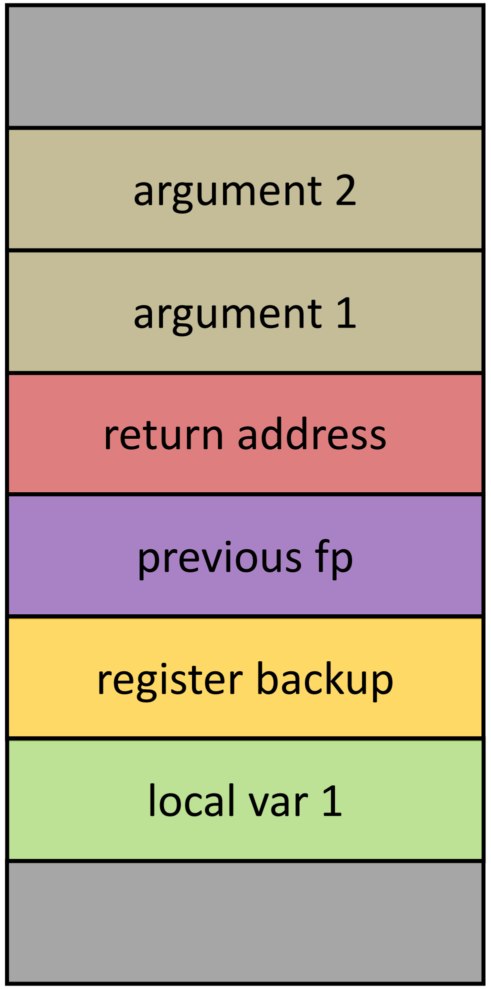


restore

```
f:
...

move $sp, $fp
lw $t0, -4($sp)
...
lw $t9, -40($sp)
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

Register Backup: Local Variables Offsets



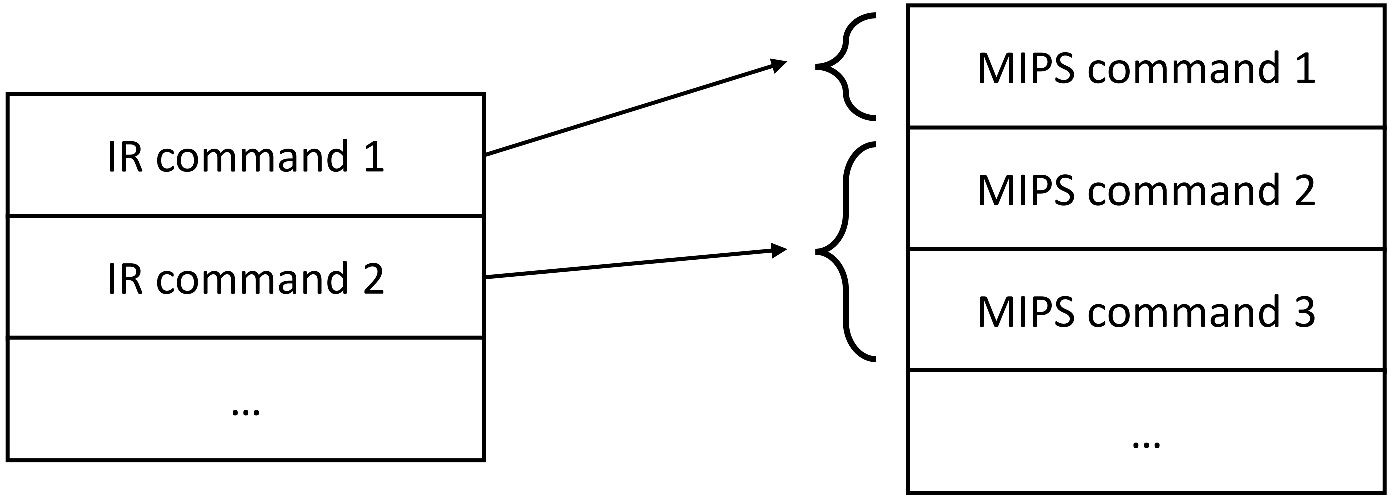
```
f:
...
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -44($fp)
lw $v0, -44($fp)
...
```

Translating IR to MIPS: Registers

- Our IR is likely to use too many registers
- Assume for now that the number of IR registers is reduced
  - Every IR register mapped to a CPU register (t0, … t9)
- Later in the course we will learn how to compute register allocation

Translating IR to MIPS

- Translate global variable initializations
- Translate the IR instructions for each function
  - Implement a translation function for each IR instruction
  - If the translation requires additional registers:
    - ➢Use registers s0, s1, …



Translating IR

- Global initializations (integers)

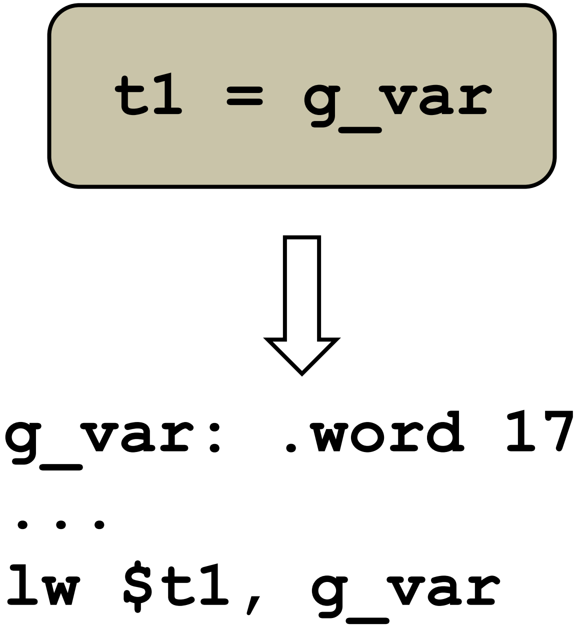
```
int g_1 = 7;
```

⇒

```
.data:
g_1:  .word 7
```

Translating IR

- Assignments (read from memory)
- For global variables:

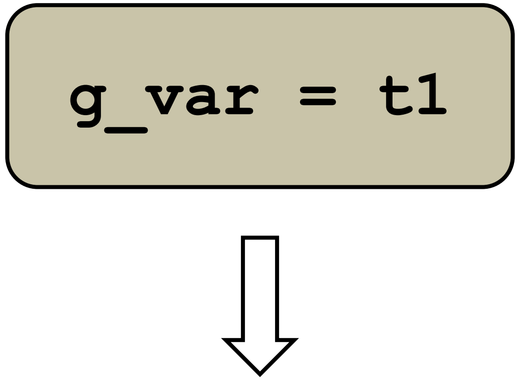


```
t1 = g_var
```

```
g_var: .word 17
...
lw $t1, g_var
```

Translating IR

- Assignments (write to memory)
- For global variables:

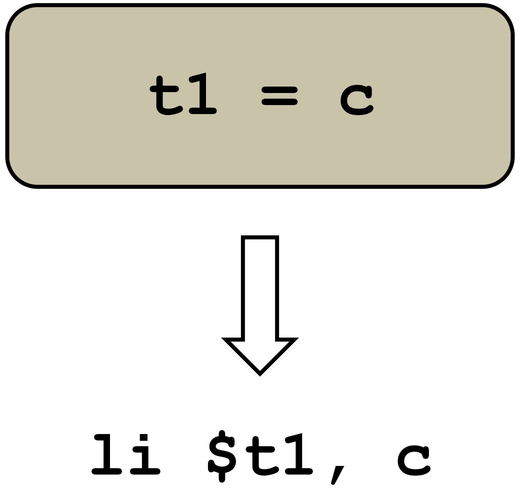


```
g_var = t1
```

```
g_var: .word 17
...
sw $t1, g_var
```

Translating IR

- Assignments (constant)

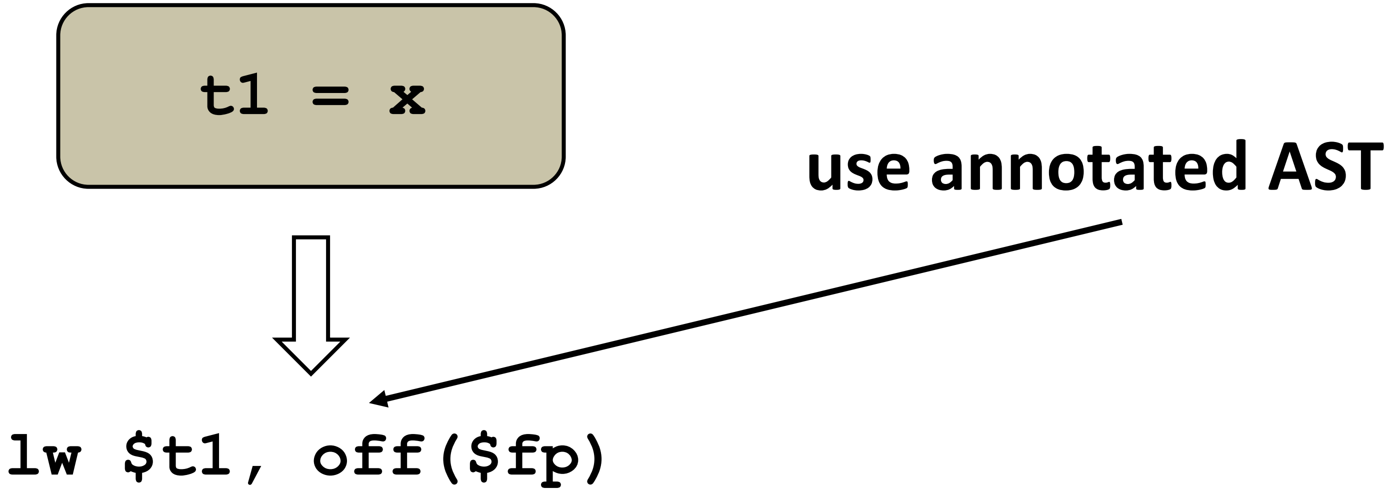


```
t1 = c
```

```
li $t1, c
```

Translating IR

- Assignments (read from memory)
- For local variables and parameters:



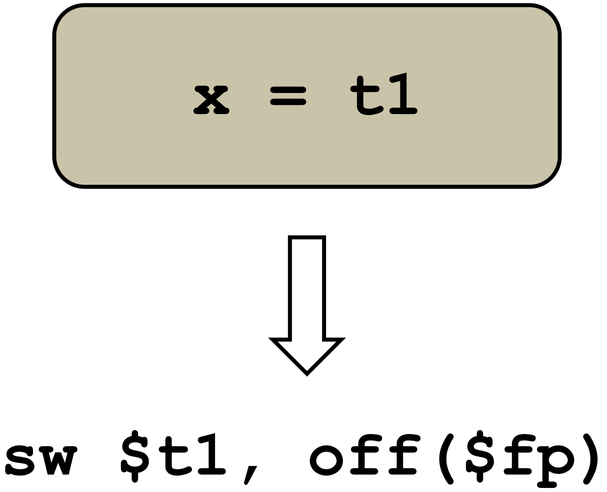
```
t1 = x
```

```
lw $t1, off($fp)
```

use annotated AST

Translating IR

- Assignments (write to memory)
- For local variables and parameters:

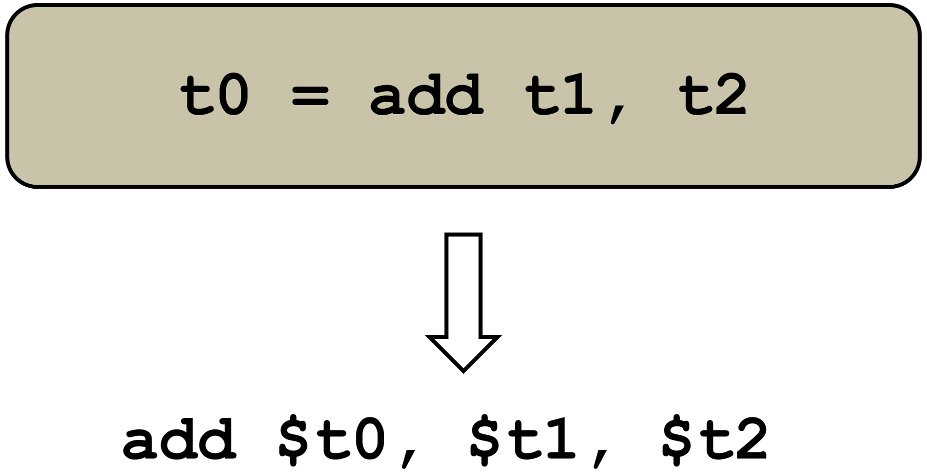


```
x = t1
```

```
sw $t1, off($fp)
```

Translating IR

- Arithmetic operation

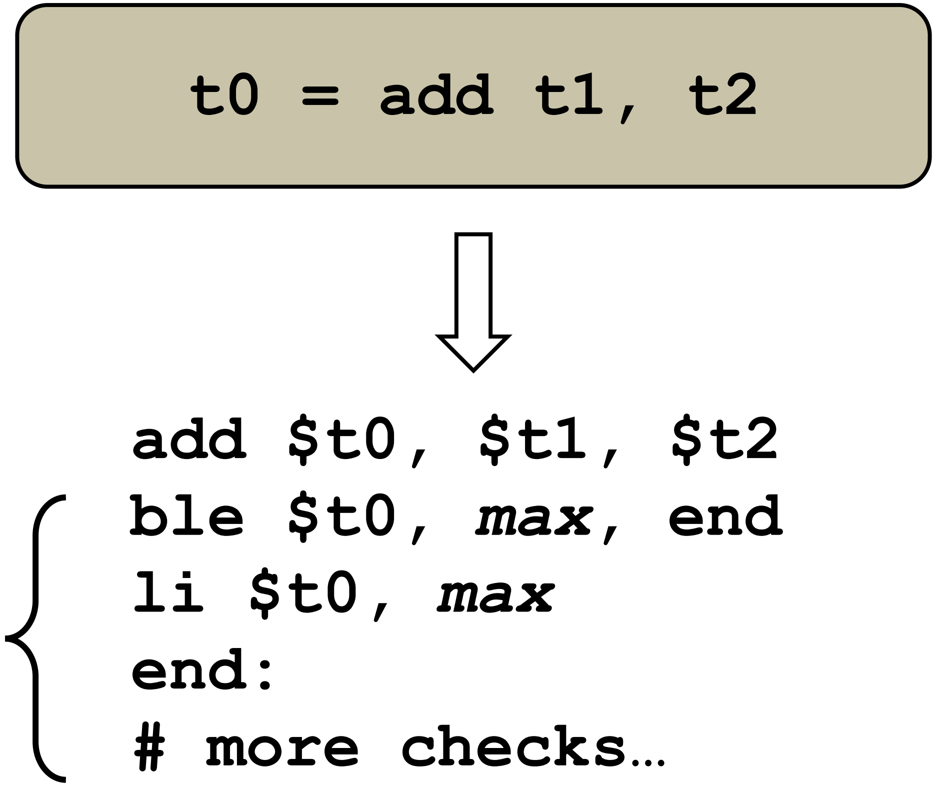


```
t0 = add t1, t2
```

```
add $t0, $t1, $t2
```

Translating IR

- Arithmetic operation



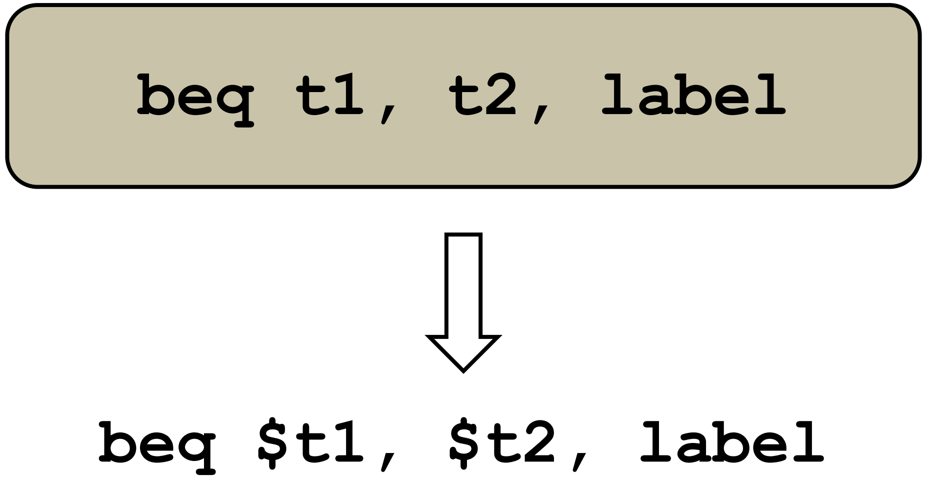
```
t0 = add t1, t2
```

```
add $t0, $t1, $t2
ble $t0, max, end
li $t0, max
end:
# more checks…
```

Bound integer values to be at most $2^{15} - 1$

Translating IR

- Branch



```
beq t1, t2, label
```

```
beq $t1, $t2, label
```

Translating IR

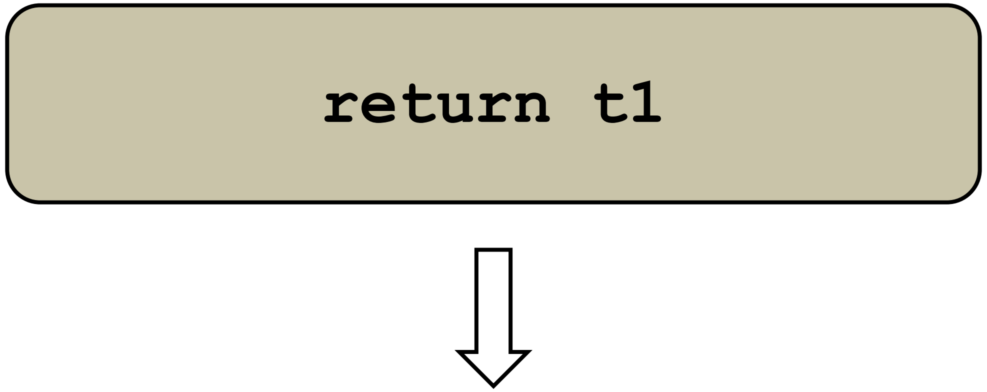
- Function call

```
t0 = call f(t1, t2)
```

```
subu $sp, $sp, 4
sw $t2, 0($sp)
subu $sp, $sp, 4
sw $t1, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
```

Translating IR

- Return (in a function f)
- Store result in **v0** and jump to f’s **epilogue label** (f_epilogue)



```
return t1
```

```
move $v0, $t1
j f_epilogue
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

Translating IR

```
int g = 70;
int f(int x){
   int z = x;
   if (z) {
      z = g
   }
   return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g:  .word 70
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g:  .word 70
.text
f:
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
```

Translating IR

```
int g = 70;
int f(int x){
   int z = x;
   if (z) {
      z = g
   }
   return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g:  .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
```

Translating IR

```
int g = 70;
int f(int x){
   int z = x;
   if (z) {
      z = g
   }
   return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
beq $t2, 0, end
```

Translating IR

```
int g = 70;
int f(int x){
   int z = x;
   if (z) {
      z = g
   }
   return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
beq $t2, 0, end
lw $t3, g
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
beq $t2, 0, end
lw $t3, g
sw $t3, -44($fp)
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
beq $t2, 0, end
lw $t3, g
sw $t3, -44($fp)
end:
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
beq $t2, 0, end
lw $t3, g
sw $t3, -44($fp)
end:
lw $t4, -44($fp)
```

Translating IR

```
int g = 70;
int f(int x){
   int z = x;
   if (z) {
      z = g
   }
   return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
beq $t2, 0, end
lw $t3, g
sw $t3, -44($fp)
end:
lw $t4, -44($fp)
move $v0, $t4
j f_epilogue
```

Translating IR

```
int g = 70;
int f(int x){
  int z = x;
  if (z) {
    z = g
  }
  return z;
}
```

```
f:
t1 = x
z = t1
t2 = z
beq t2, 0, end
t3 = g
z = t3
end:
t4 = z
return t4
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -44($fp)
lw $t2, -44($fp)
beq $t2, 0, end
lw $t3, g
sw $t3, -44($fp)
end:
lw $t4, -44($fp)
move $v0, $t4
j f_epilogue
f_epilogue:
# epilogue here
...
```

Execution Entry Point

- Execution of a MIPS program begins at the label main
- Execution of an input program begins at the function main
  - Main’s signature: void main()

Calling *main*

```
.data
...

.text
```

generated code for function **main**

```
user_main:
...
user_main_epilogue:
...
```

stub for calling **main**

```
main:
jal user_main
li $v0, 10
syscall
```

execution starts here