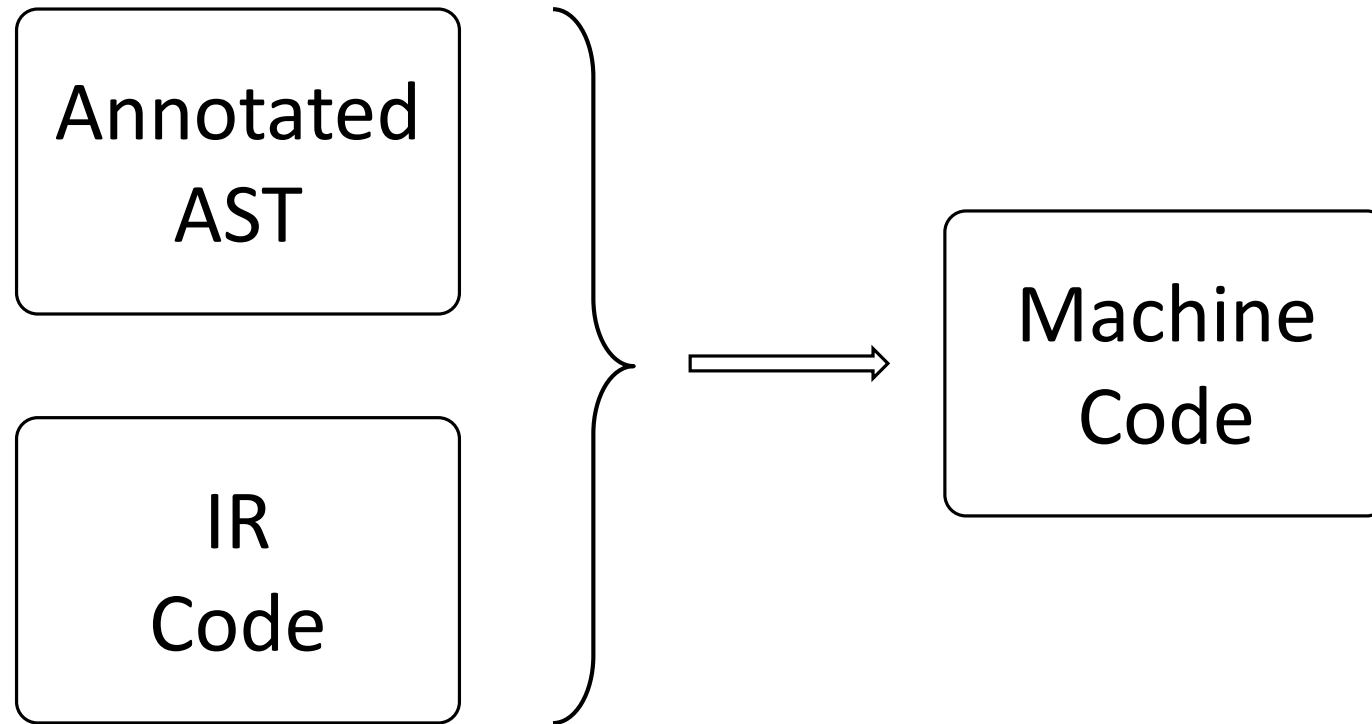


Code Generation

TEACHING ASSISTANT: DAVID TRABISH



Code Generation



MIPS Architecture

- MIPS has 32 registers:
 - t0, ..., t9
 - general purpose
 - a0, a1, a2, a3
 - arguments (in some compilers)
 - v0, v1
 - return value, system calls
 - sp, fp
 - stack pointer, frame pointer
 - ra
 - return address

MIPS Architecture

- Labels

```
data_label_1: .word 17
```

```
data_label_2: .ascii "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

```
.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

```
.data  
g_foo: .word 17  
g_str: .asciiz "hello"  
...
```

} global data

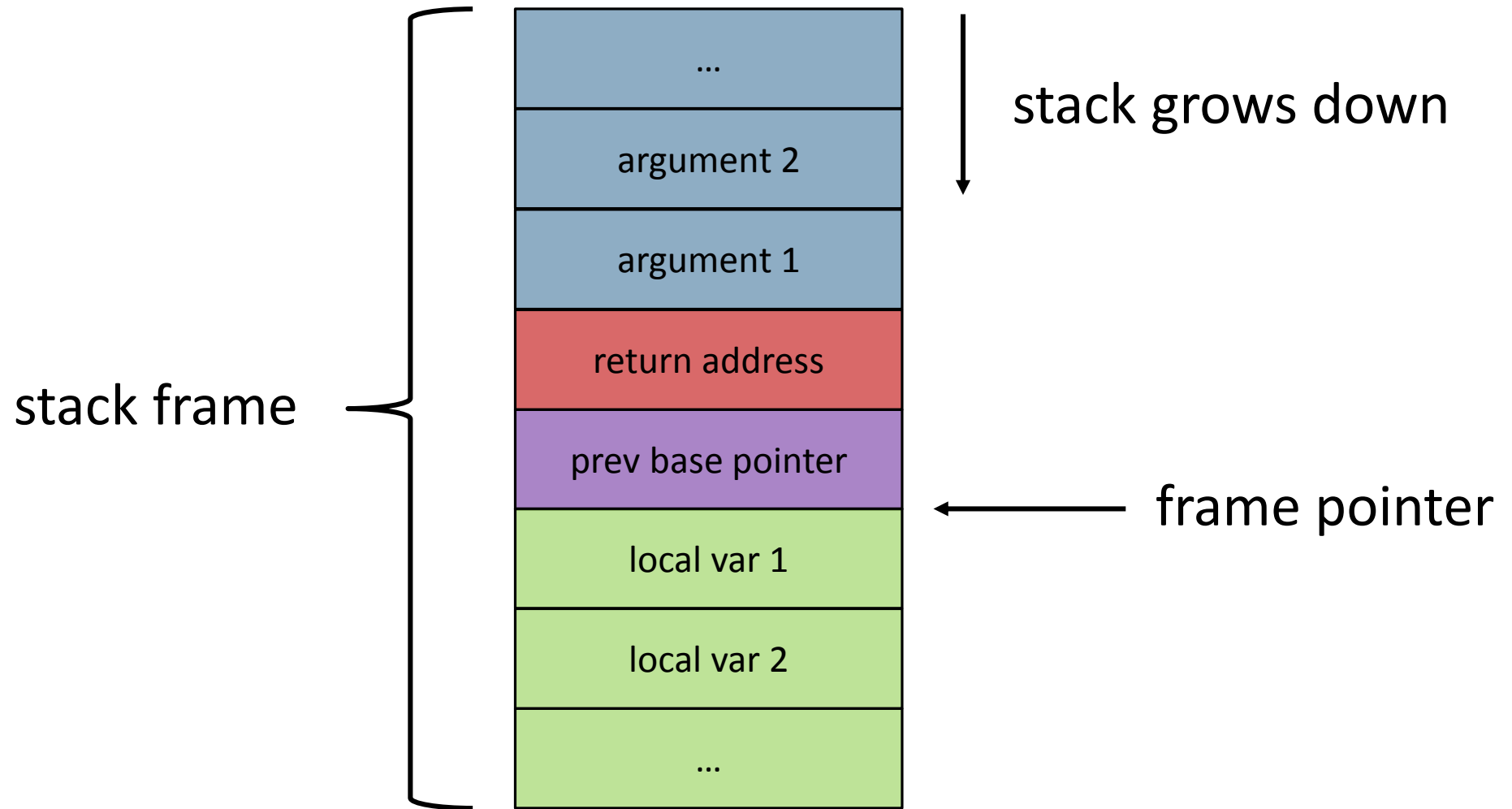
```
PrintInt(17) {  
    li $v0, 1  
    lw $a0, g_foo  
    syscall  
PrintStr("hello") {  
    li $v0, 4  
    la $a0, g_str  
    syscall
```

} code

SPIM

- Running SPIM:
 - `spim -f input_file`
- Interactive debugging:
 - `xspim`
- Tutorials:
 - <https://courses.cs.washington.edu/courses/cse410/08sp/notes/spim/SpimTutorial.pdf>
 - https://web.stanford.edu/class/cs143/materials/SPIM_Manual.pdf

Stack



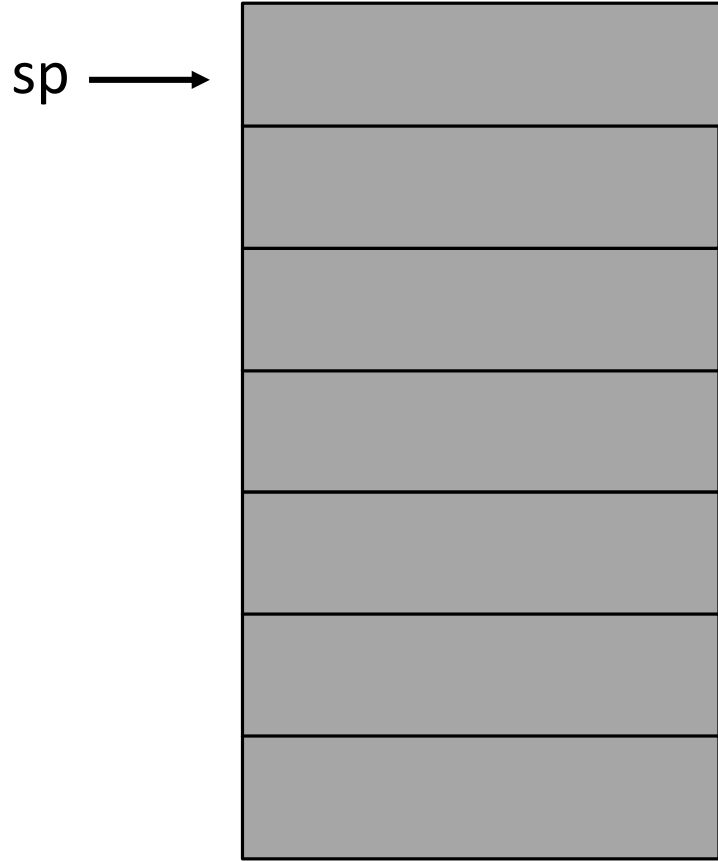
Stack

```
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  
sub $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



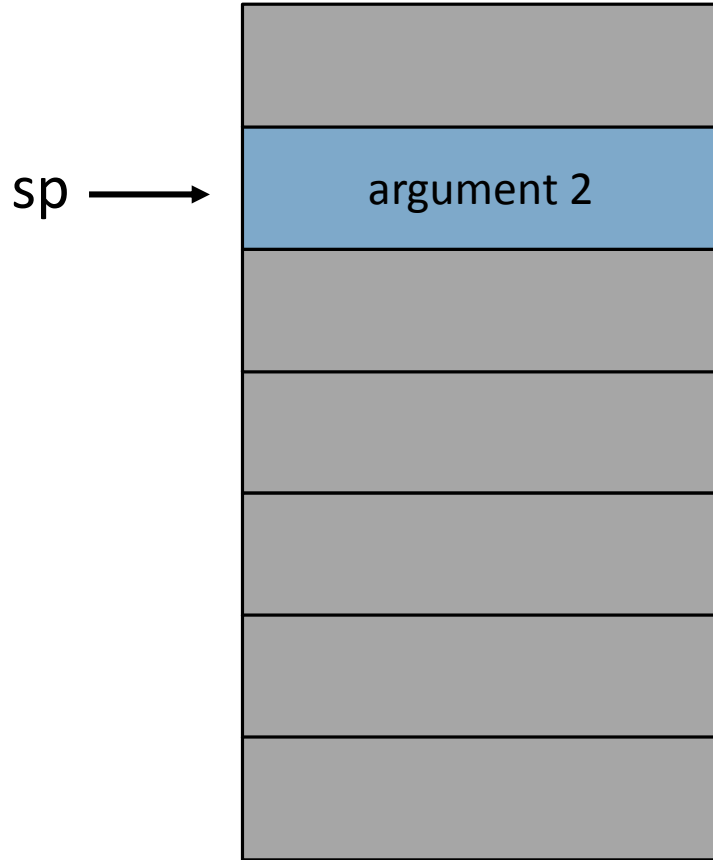
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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
...
```

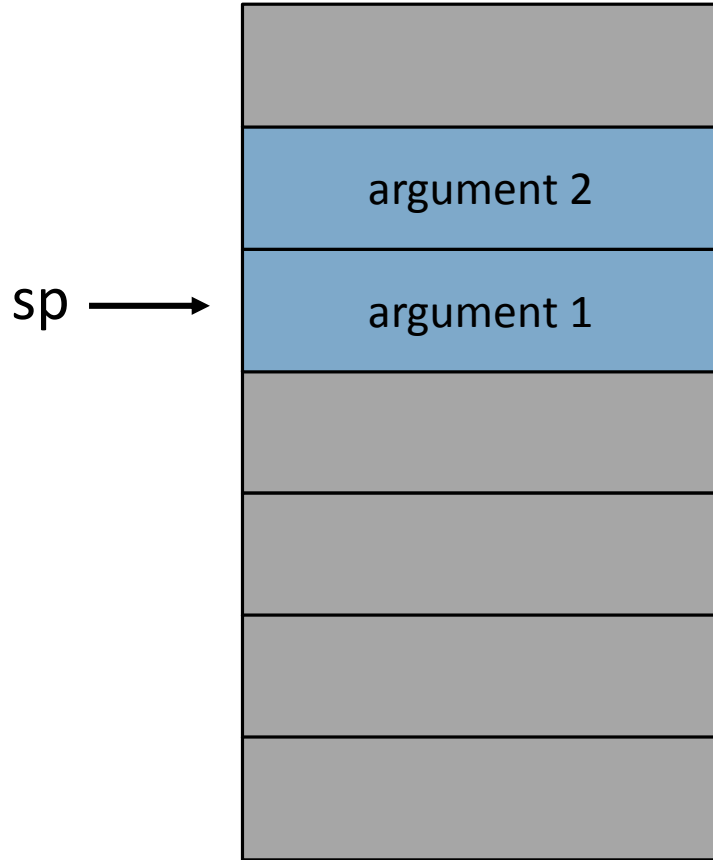

Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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



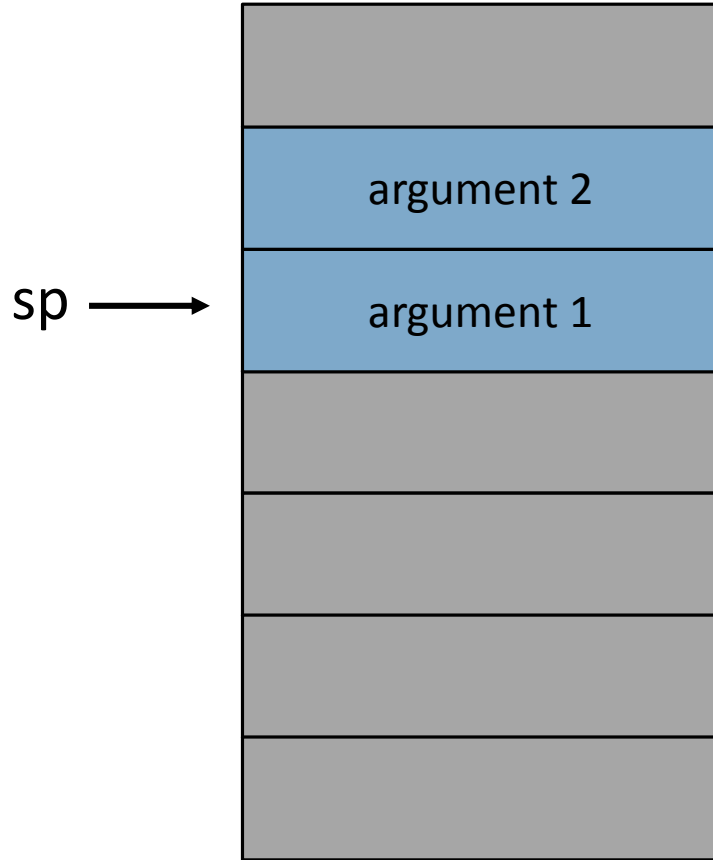
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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



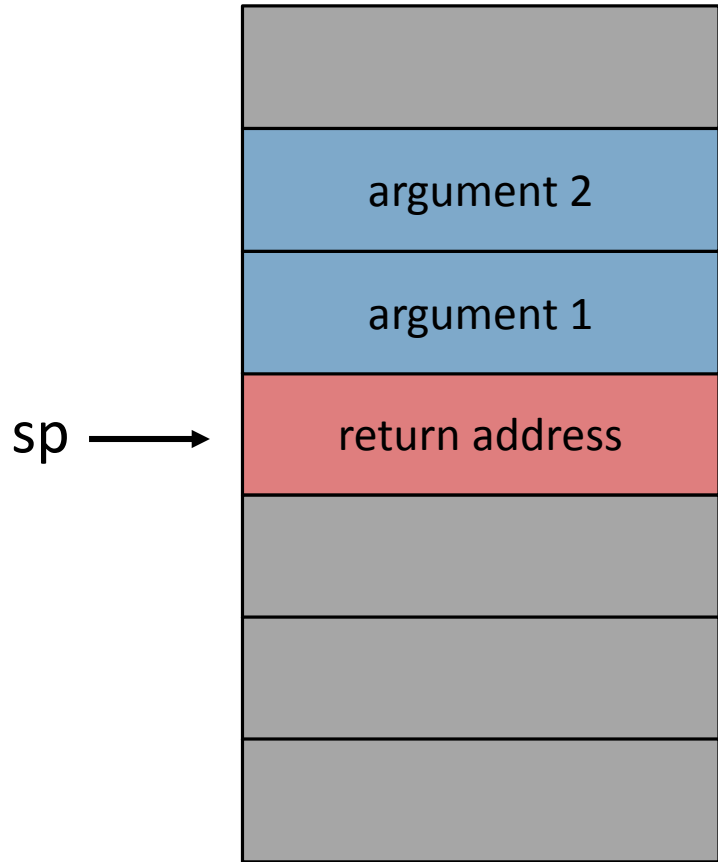
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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

sub \$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)

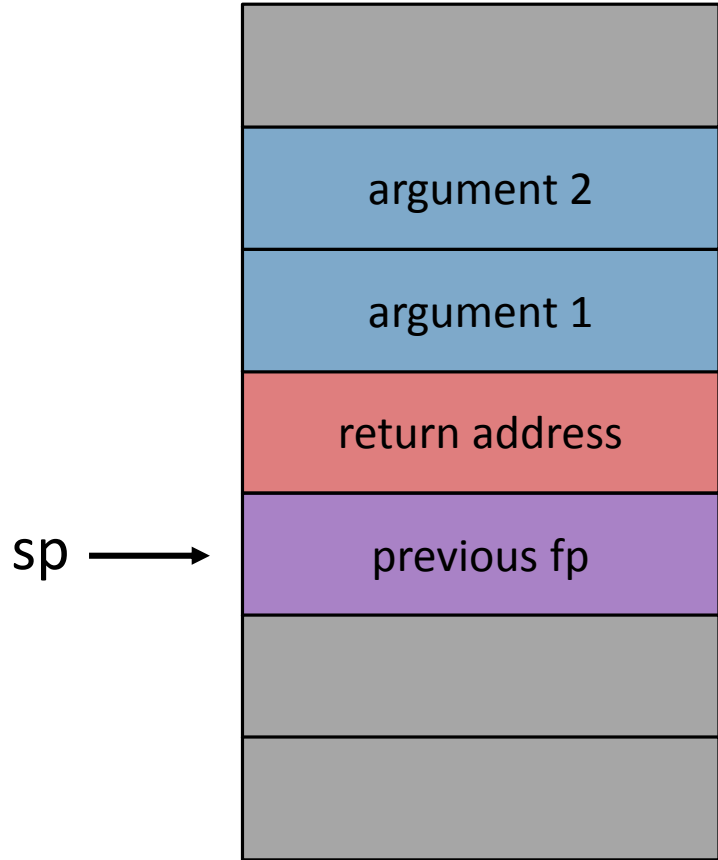
jal f

addu \$sp, \$sp, 8

move \$t0, \$v0

...

Stack



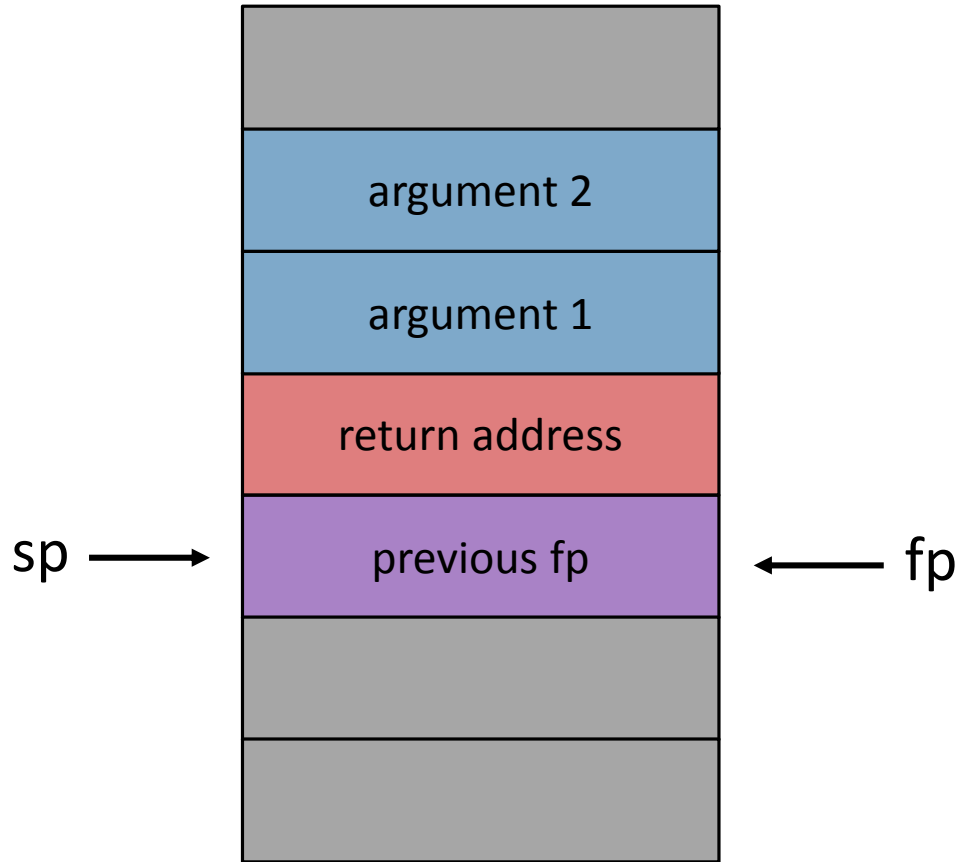
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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
...
```

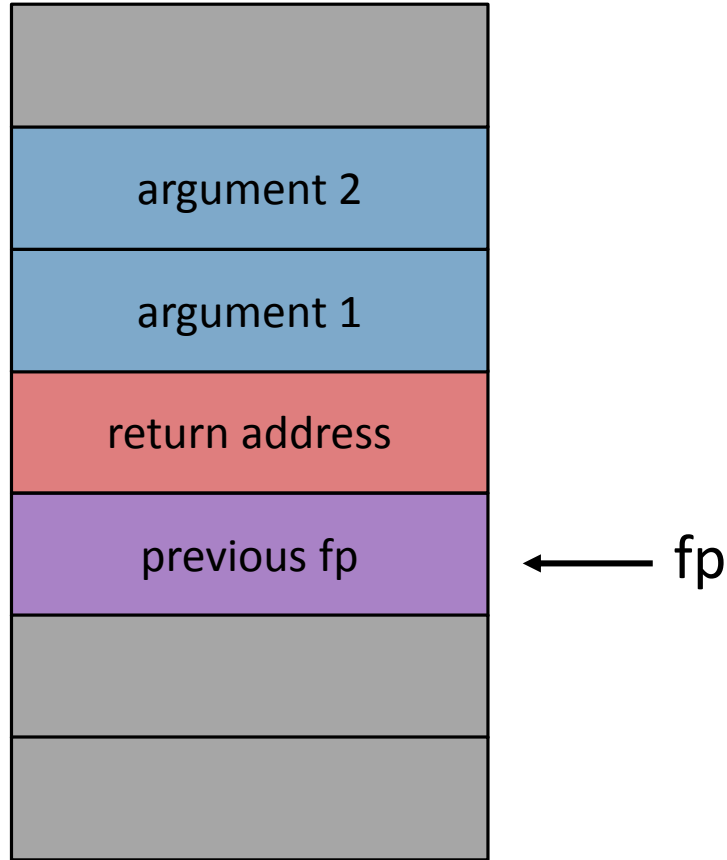
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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 →

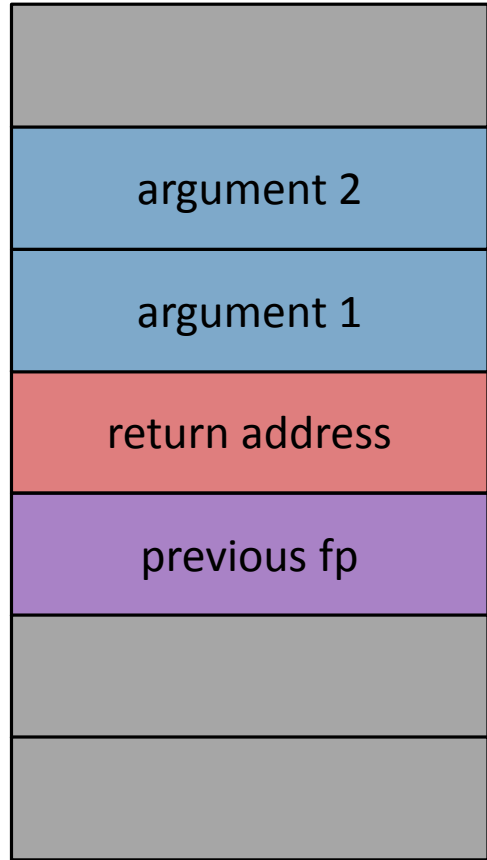
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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
sub $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
```

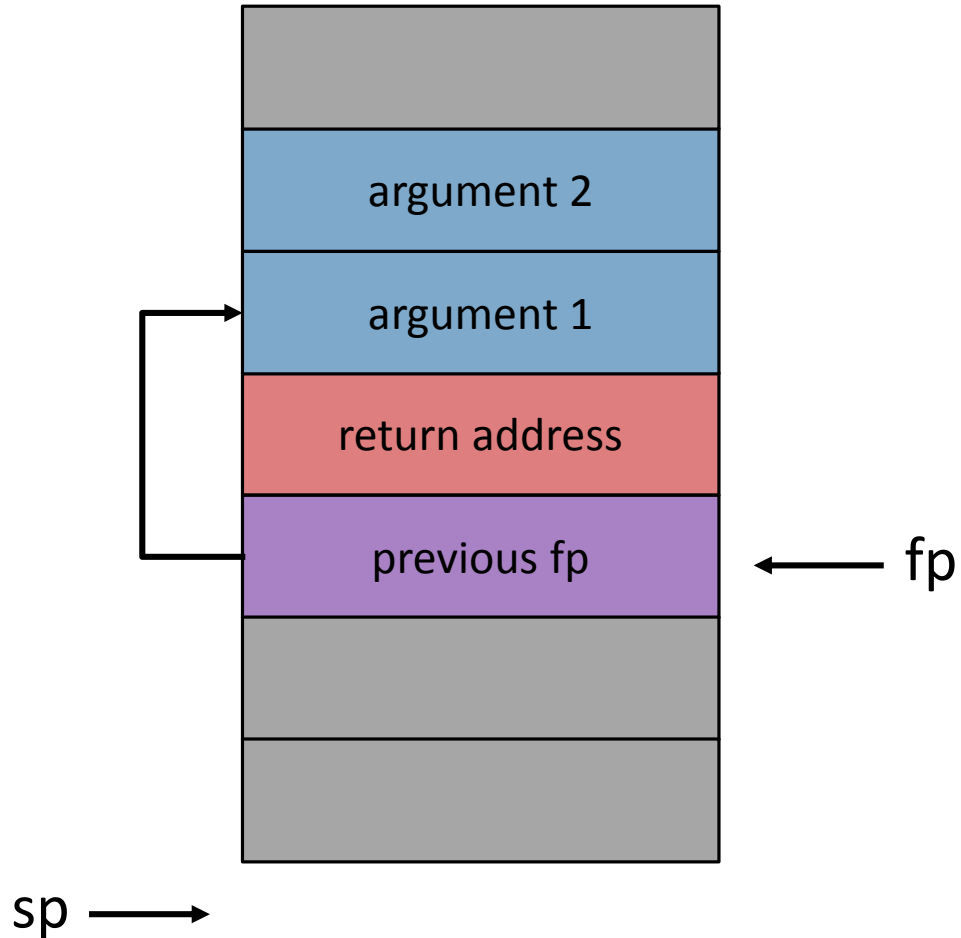
g:

```
...
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
...
```

sp →

← fp

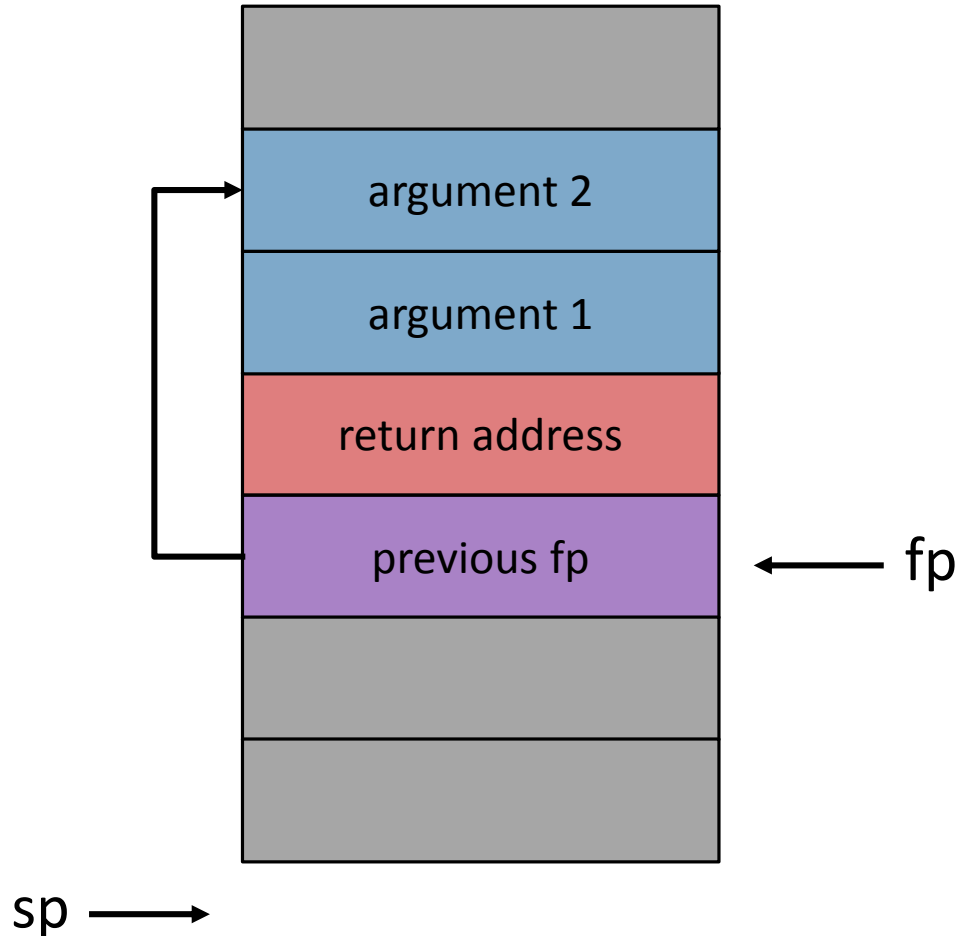
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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
...
```

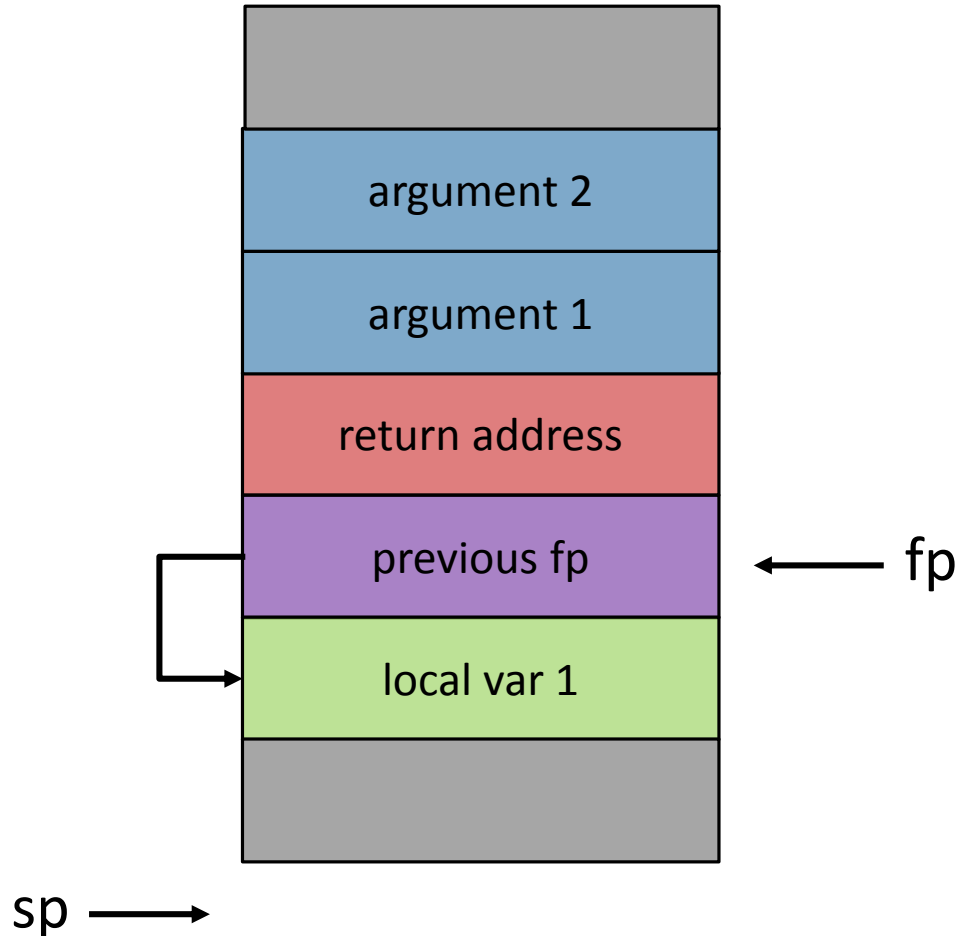
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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
...
```

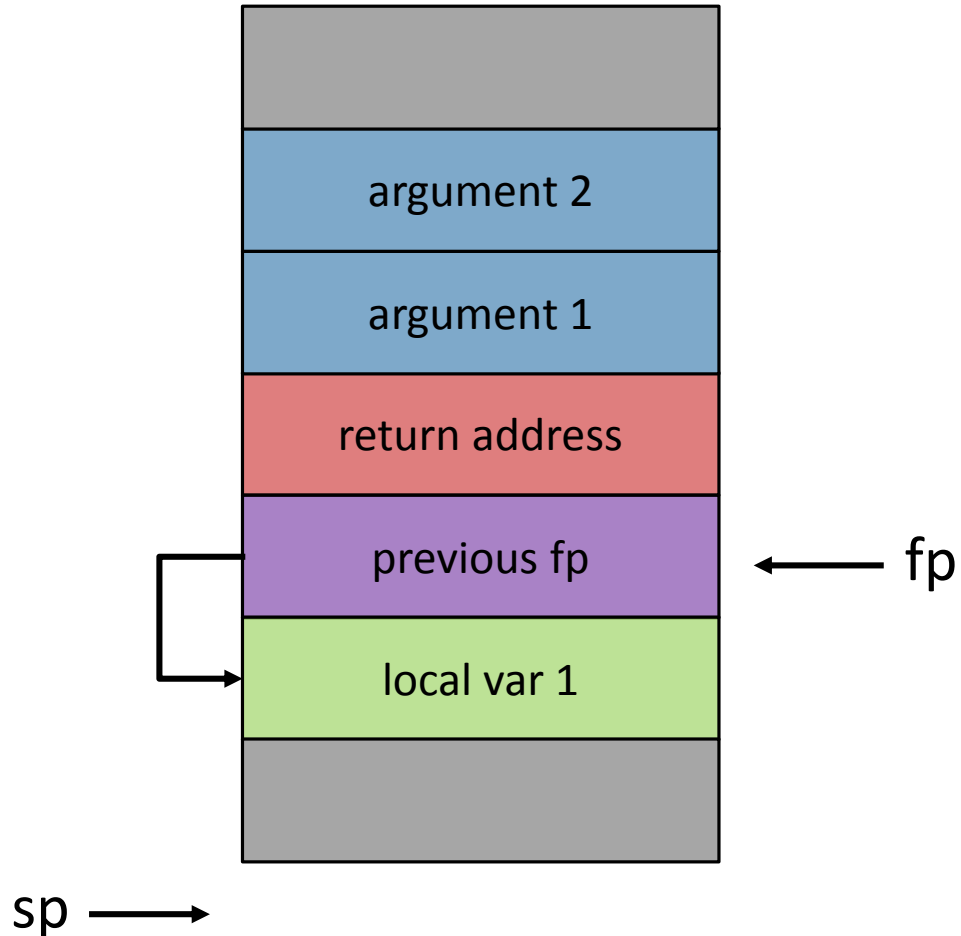
Stack



```
f:
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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



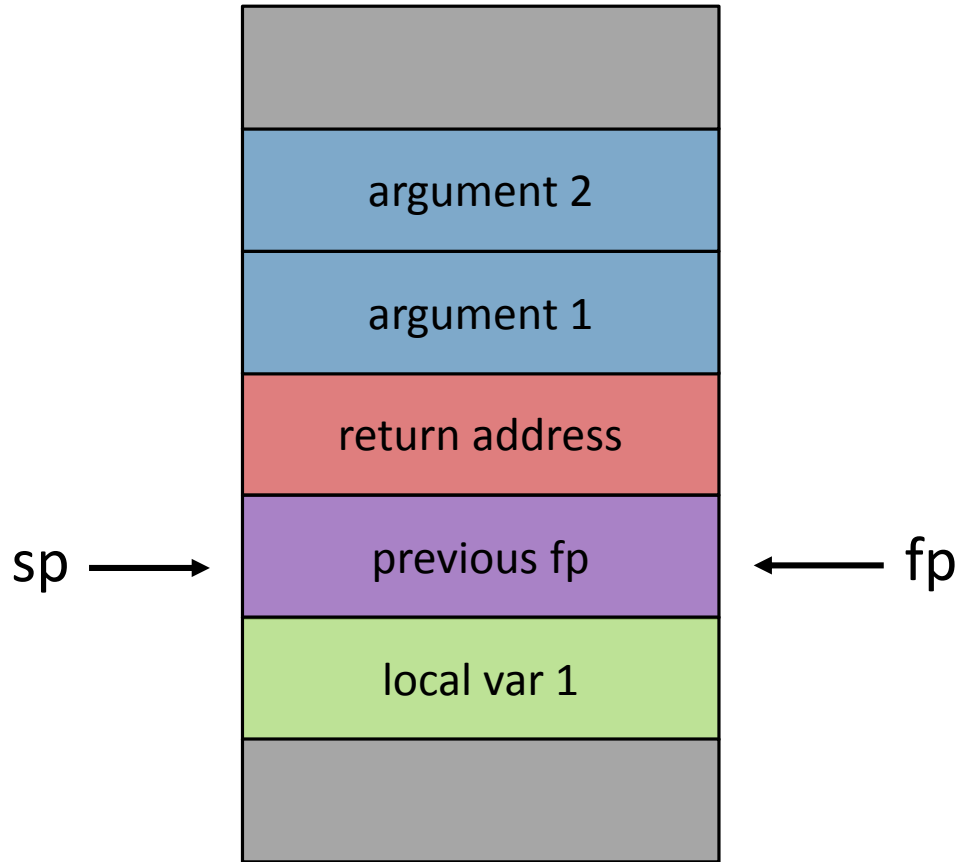
f:

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

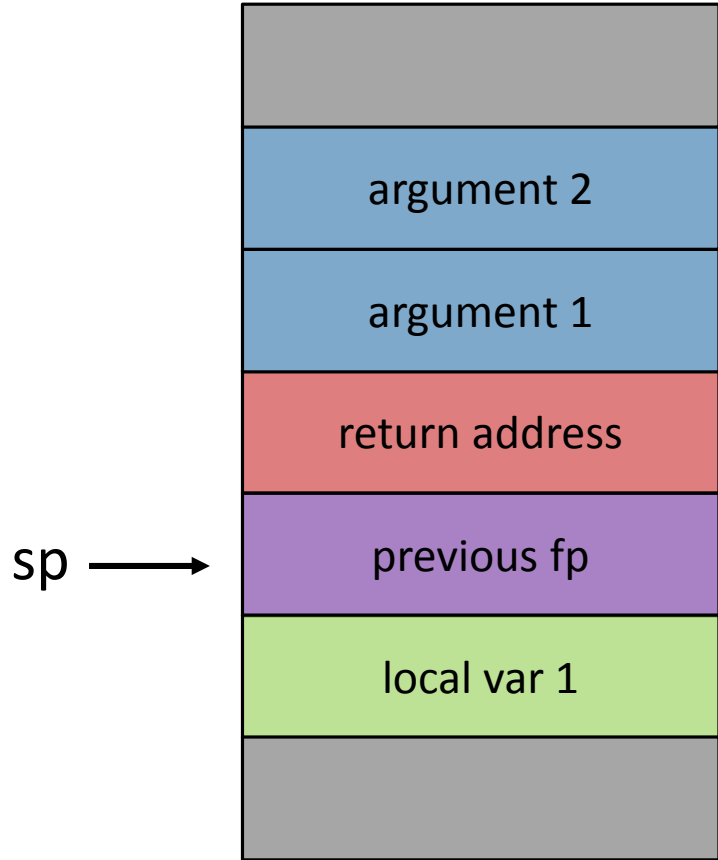
move \$sp, \$fp

```
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



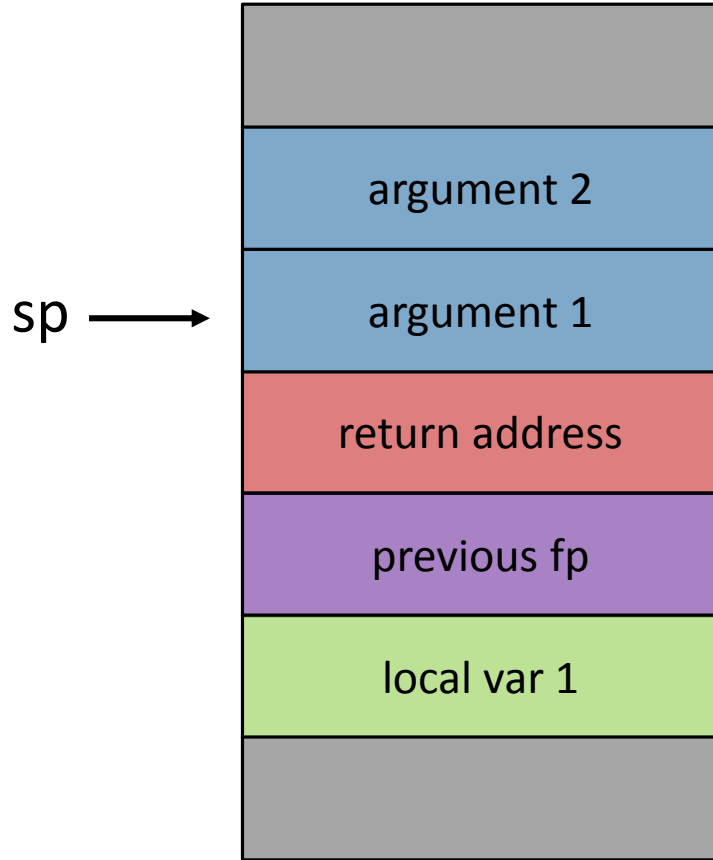
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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



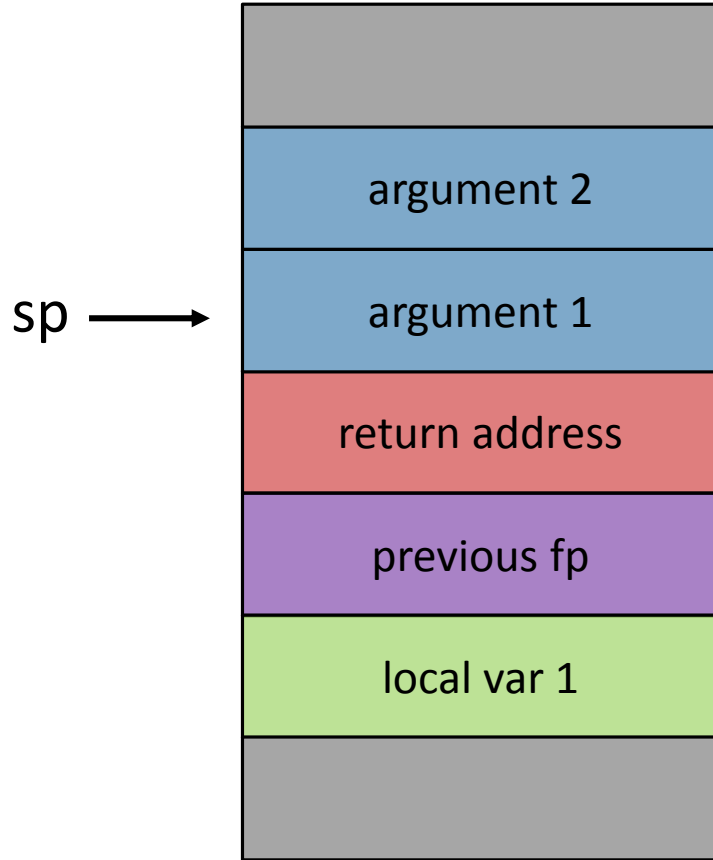
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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:

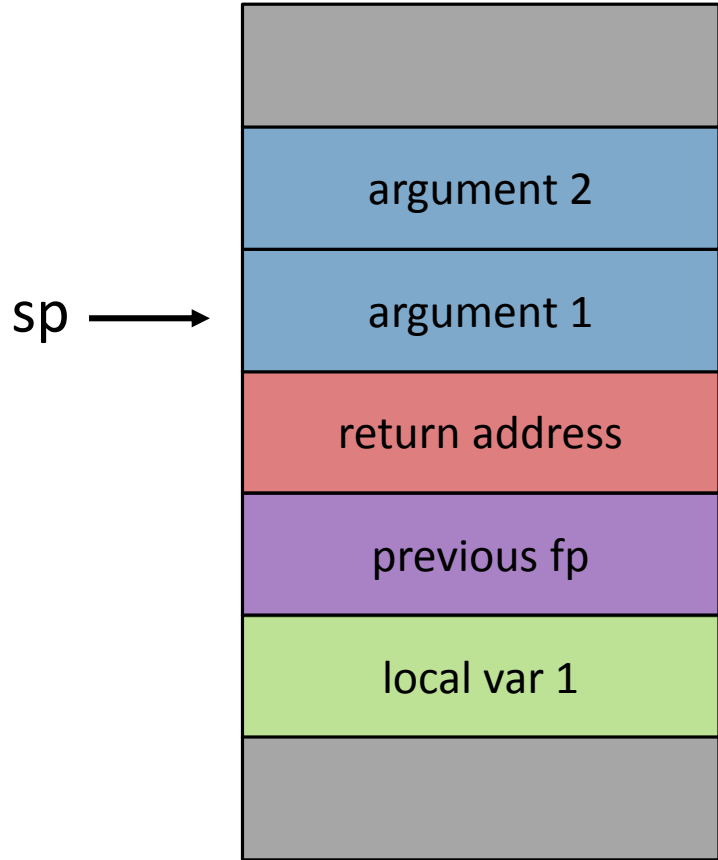
```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
```

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
sub $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
```

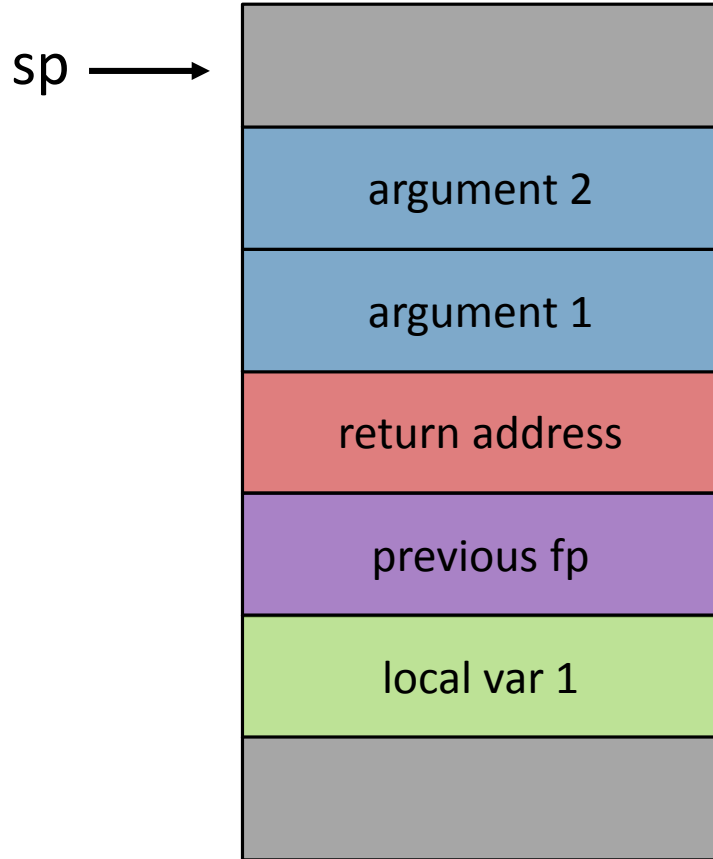
epilogue {

```
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



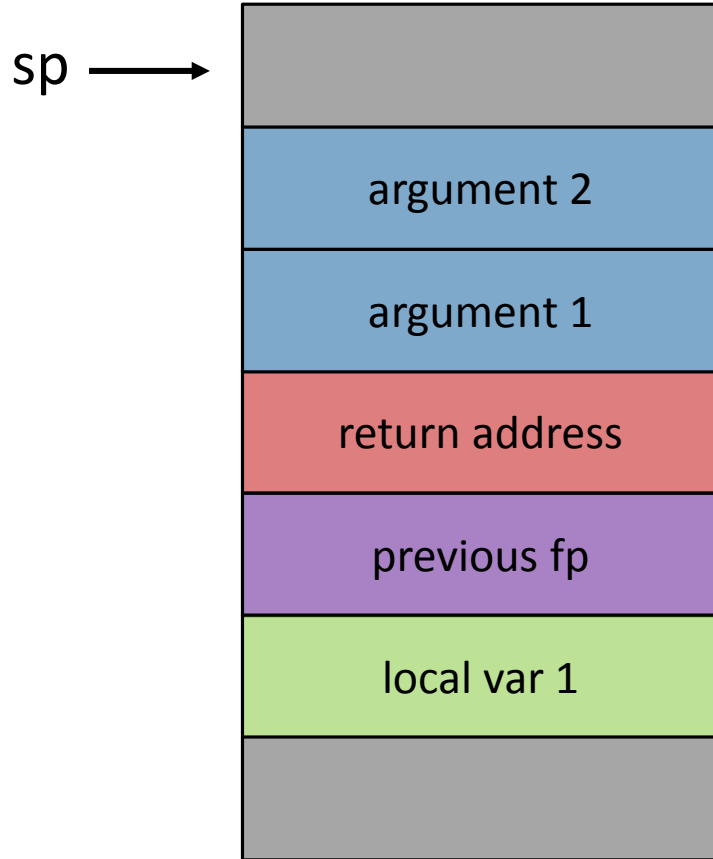
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $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:

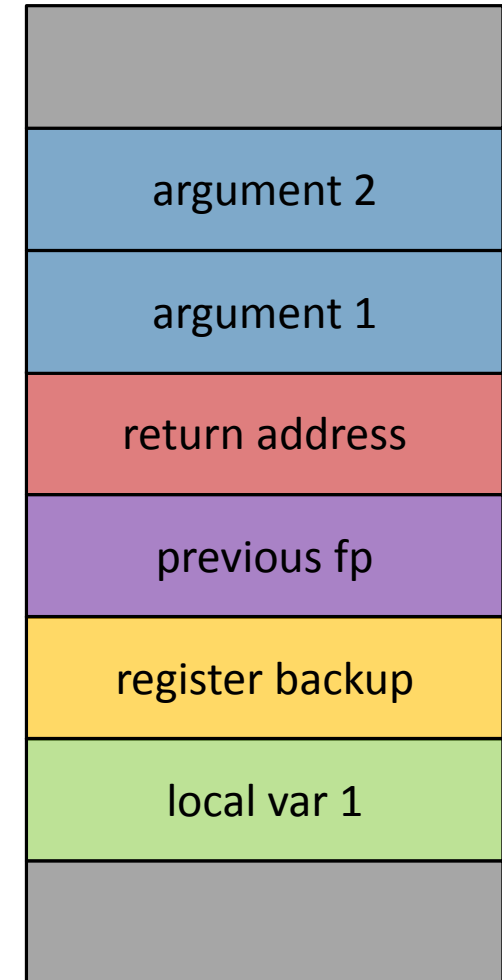
```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
sub $sp, $sp, 16
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -4($fp)
lw $v0, -4($fp)
move $sp, $fp
lw $fp, 0($sp)
lw $ra, 4($sp)
addu $sp, $sp, 8
jr $ra
```

g:

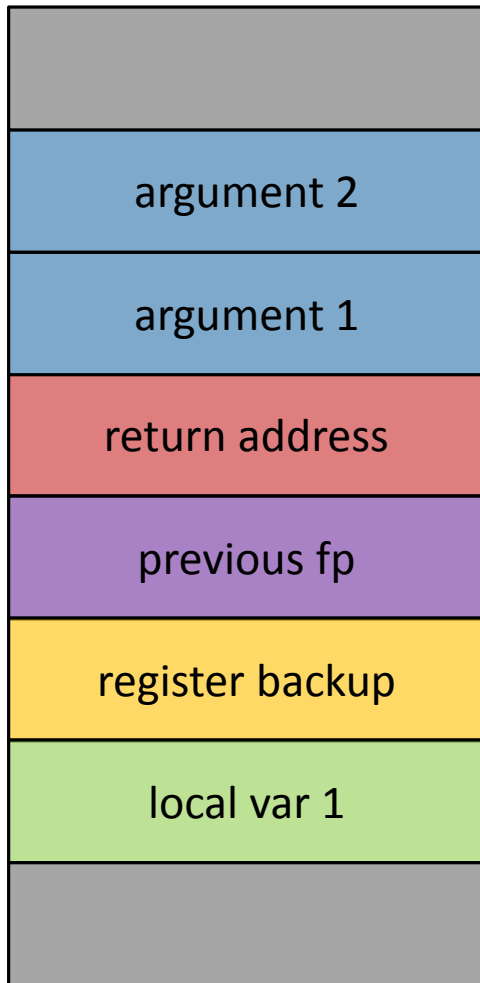
```
...
li $t0, 20
subu $sp, $sp, 4
sw $t0, 0($sp)
li $t0, 10
subu $sp, $sp, 4
sw $t0, 0($sp)
jal f
addu $sp, $sp, 8
move $t0, $v0
...
```

Register Backup

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



Register Backup: Prologue



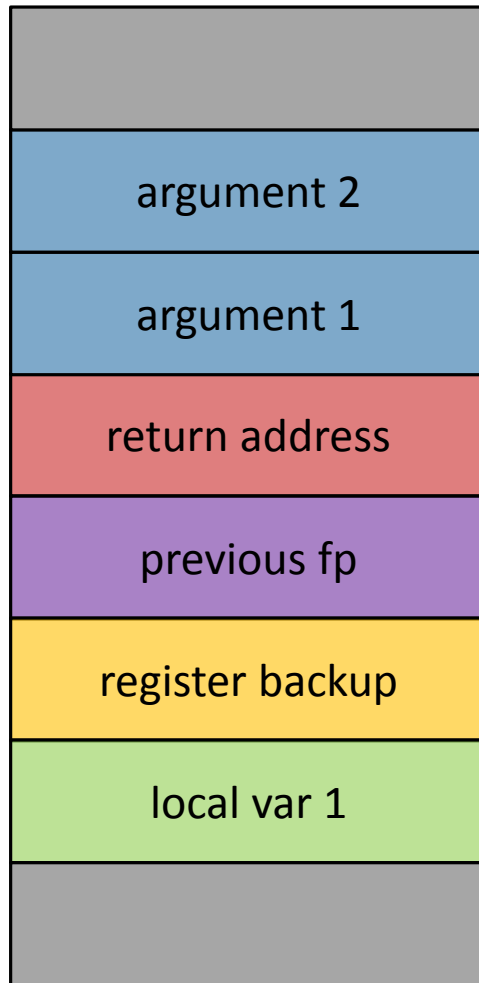
f:

```
subu $sp, $sp, 4
sw $ra, 0($sp)
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
```

backup {

```
subu $sp, $sp, 4
sw $t0, 0($sp)
...
subu $sp, $sp, 4
sw $t9, 0($sp)
sub $sp, $sp, 16
```

Register Backup: Epilogue

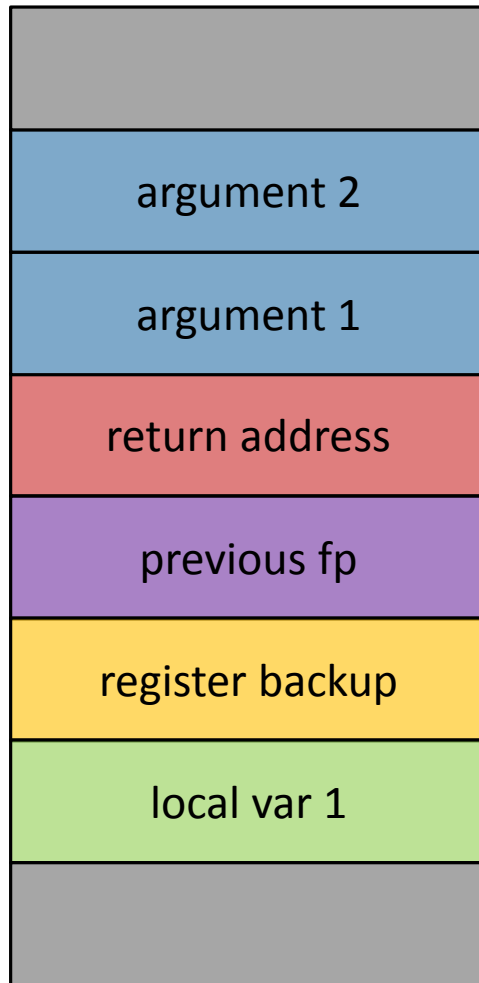


```
f:
...

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
```

restore {

Register Backup



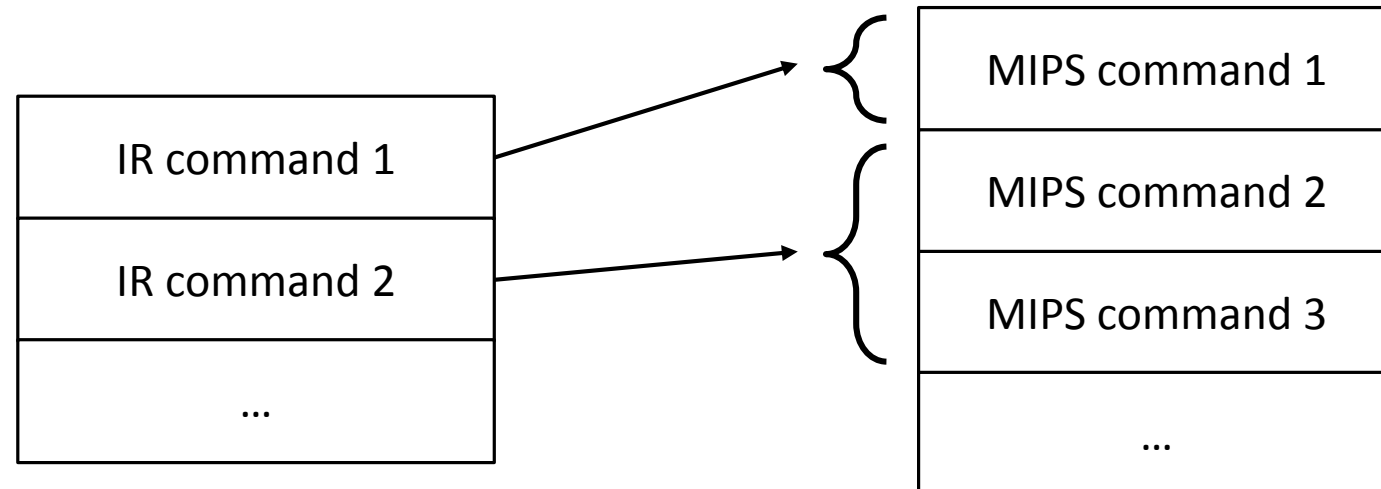
```
f:
...
lw $t0, 8($fp)
lw $t1, 12($fp)
add $t2, $t0, $t1
sw $t2, -44($fp)
lw $v0, -44($fp)
...
```

Translating IR to MIPS

- 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)
- We will see later how to compute this **register allocation**

Translating IR to MIPS

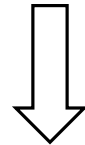
- Translate global variable initializations
- Translate the IR instructions for each function
 - Implement a translation function for each IR instruction
 - If the translation requires additional registers:
 - Use registers s0, s1, ...



Translating IR

- Assignments (constant)

`t1 = c`

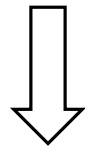


`li $t1, c`

Translating IR

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

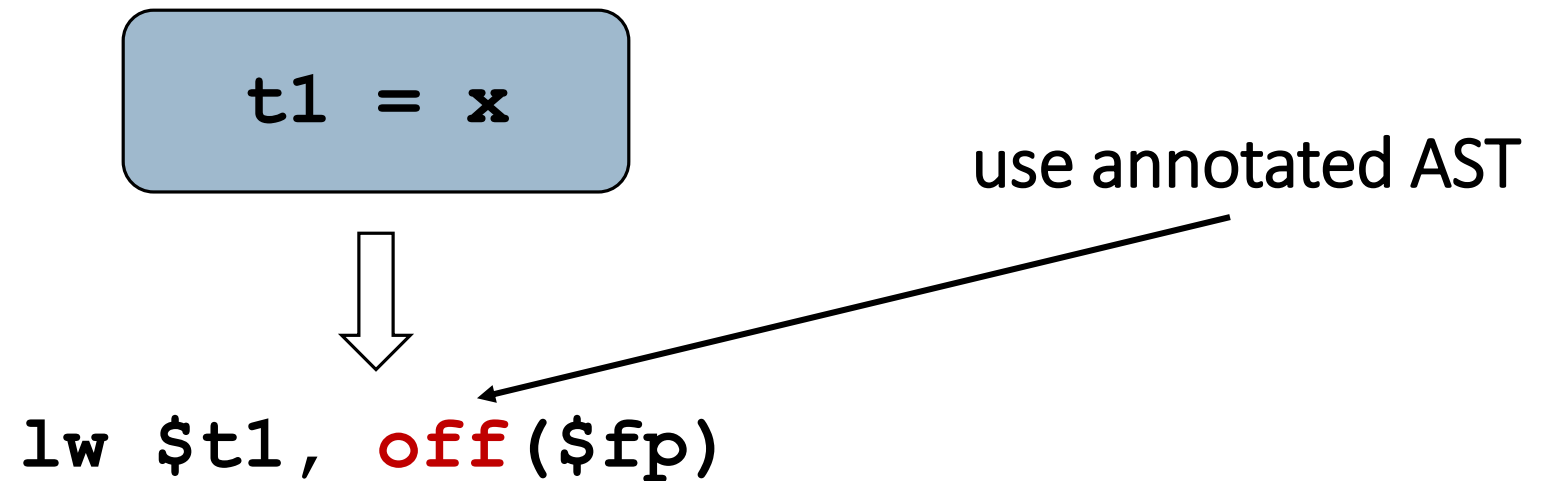
t1 = x



lw \$t1, off(\$fp)

Translating IR

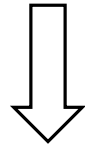
- Assignments (read from memory)
- For local variables and parameters:



Translating IR

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

x = t1

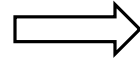


sw \$t1, off(\$fp)

Translating IR

- Global initializations

```
int g_1 = 7;
```

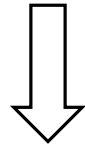


```
.data:  
g_1: .word 7
```

Translating IR

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

`t1 = g_var`

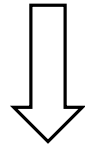


```
g_var: .word 17
...
lw $t1, g_var
```

Translating IR

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

`g_var = t1`

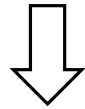


```
g_var: .word 17
...
sw $t1, g_var
```


Translating IR

- Arithmetic operation

`t0 = add t1, t2`

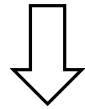


`add $t0, $t1, $t2`

Translating IR

- Arithmetic operation

`t0 = add t1, t2`

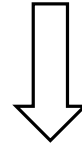


```
add $t0, $t1, $t2
ble $t0, max, end
li $t0, max
end:
# more checks...
```

Translating IR

- Branch

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



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

Translating IR

- Function call

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

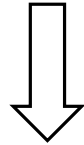


```
subu $sp, $sp, 4  
sw $t2, 0($sp)  
subu $sp, $sp, 4  
sw $t1, 0($sp)  
jal f  
addu $sp, $sp, 8  
move $t0, $v0
```

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  
t1 = z  
beq t1, 0, end  
t1 = g  
z = t1  
end:  
t1 = z  
return t1
```

Translating IR

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

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

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

Translating IR

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

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

```
.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
t1 = z
beq t1, 0, end
t1 = g
z = t1
end:
t1 = z
return t1
```

```
.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
t1 = z
beq t1, 0, end
t1 = g
z = t1
end:
t1 = z
return t1
```

```
.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
t1 = z
beq t1, 0, end
t1 = g
z = t1
end:
t1 = z
return t1
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
```

Translating IR

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

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

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
```

Translating IR

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

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

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
beq $t1, 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
t1 = z
beq t1, 0, end
t1 = g
z = t1
end:
t1 = z
return t1
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
beq $t1, 0, end
lw $t1, g
```

Translating IR

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

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

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
beq $t1, 0, end
lw $t1, g
sw $t1, -4($fp)
```

Translating IR

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

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

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
beq $t1, 0, end
lw $t1, g
sw $t1, -4($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
t1 = z
beq t1, 0, end
t1 = g
z = t1
end:
t1 = z
return t1
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
beq $t1, 0, end
lw $t1, g
sw $t1, -4($fp)
end:
lw $t1, -4($fp)
```

Translating IR

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

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

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
beq $t1, 0, end
lw $t1, g
sw $t1, -4($fp)
end:
lw $t1, -4($fp)
move $v0, $t1
j f_epilogue
```

Translating IR

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

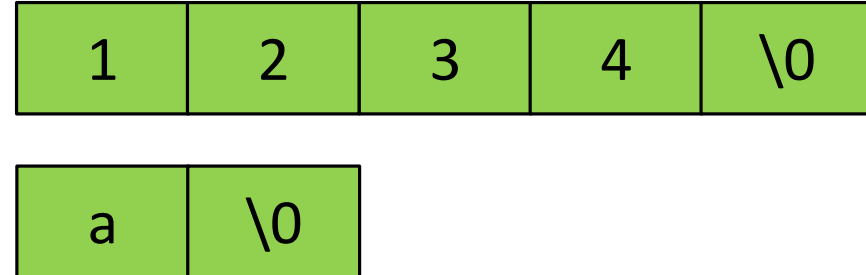
```
f:
t1 = x
z = t1
t1 = z
beq t1, 0, end
t1 = g
z = t1
end:
t1 = z
return t1
```

```
.data
g: .word 70
.text
f:
# prologue here
...
lw $t1, 8($fp)
sw $t1, -4($fp)
lw $t1, -4($fp)
beq $t1, 0, end
lw $t1, g
sw $t1, -4($fp)
end:
lw $t1, -4($fp)
move $v0, $t1
j f_epilogue
f_epilogue:
# epilogue here...
```

Strings

- We use null terminated strings
- Every character is one byte

```
string s1 = "1234";  
string s2 = "a";  
...  
...
```



Strings

- Assume that *s1* and *s2* are strings

```
if (s1 == s2) {  
  
}
```

```
t1 = s1;  
t2 = s2;  
t3 = str_eq t1, t2  
compare t3, 0  
...
```

Strings

- Inline string comparison

```
t1 = s1;  
t2 = s2;  
t3 = str_eq t1, t2  
compare t3, 0  
...
```

```
li $t3, 1 // result  
move $s0, $t1  
move $s1, $t2  
str_eq_loop:  
lb $s2, 0($s0)  
lb $s3, 0($s1)  
bne $s2, $s3, neq_label  
beq $s2, 0, str_eq_end  
addu $s0, $s0, 1  
addu $s1, $s1, 1  
j str_eq_loop  
neq_label:  
li $t3, 0  
str_eq_end:
```

Strings

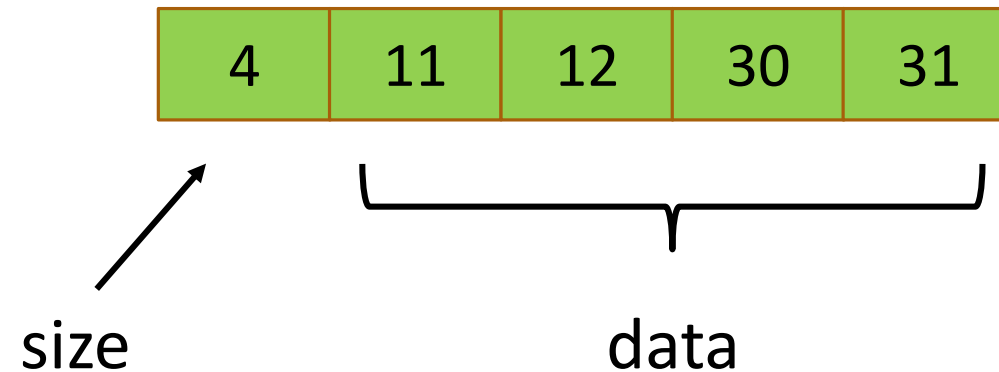
- Alternatively, create a function `str_eq`

```
t1 = s1;  
t2 = s2;  
t3 = str_eq t1, t2  
compare t3, 0  
...
```

```
subu $sp, $sp, 4  
sw $t2, 0($sp)  
subu $sp, $sp, 4  
sw $t1, 0($sp)  
jal str_eq  
addu $sp, $sp, 8  
move $t3, $v0
```

Arrays

- Each cell is 4 bytes (*int* or *pointer*)
- First cell is the **size** of the array
- The rest of the cells contain **data**



Arrays

- Creating arrays

```
t0 = new_array t1
```

```
li $v0, 9
move $a0, $t1
add $a0, $a0, 1
mul $a0, $a0, 4
syscall
move $t0, $v0
li $s0, size
sw $s0, 0($t0)
```

Arrays

- Array access

```
t0 = array_access t1, t2
```

```
move $s0, $t2  
add $s0, $s0, 1  
mult $s0, $s0, 4  
addu $s0, $t1, $s0  
lw $t0, 0($s0)
```

Runtime Errors

- Division by zero

```
t0 = div t1, t2
```

```
div $t0, $t1, $t2
```

Runtime Errors

- Division by zero

```
t0 = div t1, t2
```

```
beq $t2, 0, abort  
div $t0, $t1, $t2  
...  
abort:  
li $v0, 10  
syscall
```

Runtime Errors

- Out of bounds array access

```
t0 = array_access t1, t2
```

```
move $s0, $t2
add $s0, $s0, 1
mult $s0, $s0, 4
addu $s0, $t1, $s0
lw $t0, 0($s0)
```

Runtime Errors

- Out of bounds array access

```
t0 = array_access t1, t2
```

```
bltz $t2, abort
lw $s0, 0($t1)
bge $t2, $s0, abort
move $s0, $t2
add $s0, $s0, 1
mult $s0, $s0, 4
addu $s0, $t1, $s0
lw $t0, 0($s0)
...
abort:
li $v0, 10
syscall
```

Runtime Errors

- Null pointer dereference
 - Arrays
 - Field access
 - Method calls

Runtime Errors

- Null pointer dereference
- For example, in arrays:

```
t0 = array_access t1, t2
```

```
beq $t1, 0, abort
move $s0, $t2
add $s0, $s0, 1
mult $s0, $s0, 4
addu $s0, $t1, $s0
lw $t0, 0($s0)
...
abort:
li $v0, 10
syscall
```


Runtime Errors

- Null pointer dereference
- For example, in field accesses:

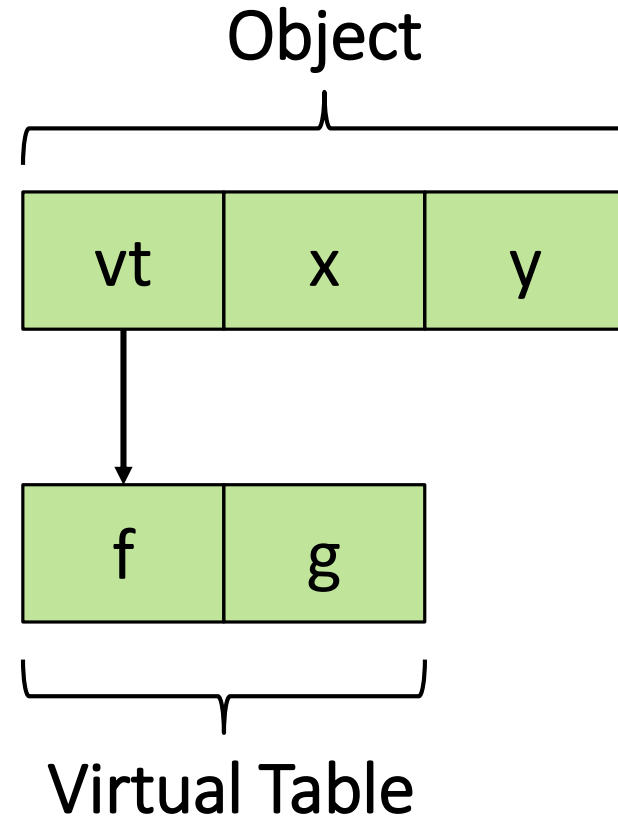
```
t0 = field_access t1, foo
```

```
beq $t1, 0, abort  
lw $t0, off($t1)  
...  
abort:  
li $v0, 10  
syscall
```

Classes

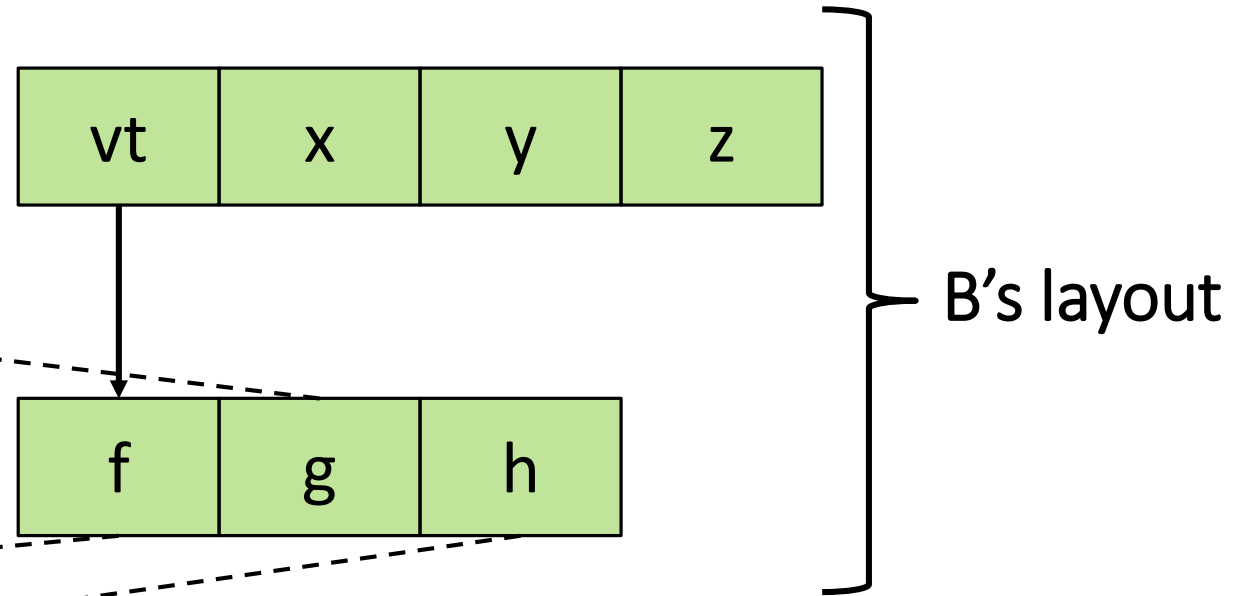
Class Layout

```
class A {  
    int x;  
    string y;  
    int f() { ...  
    int g() { ...  
}
```



Class Layout

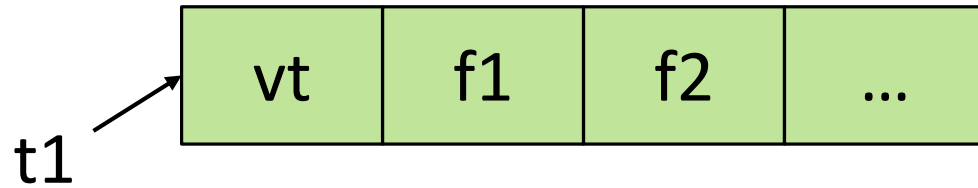
```
class A {  
    int x;  
    string y;  
    int f() { ...  
    int g() { ...  
}  
class B extends A {  
    int z;  
    int f() { ...  
    int h() { ...  
}
```



Field Access

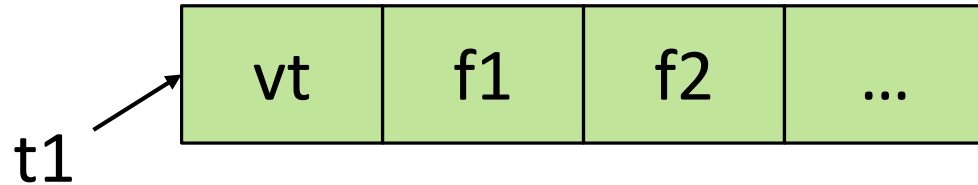
```
t0 = field_access t1, f2
```

```
lw $t0, 8($t1)
```



Field Access

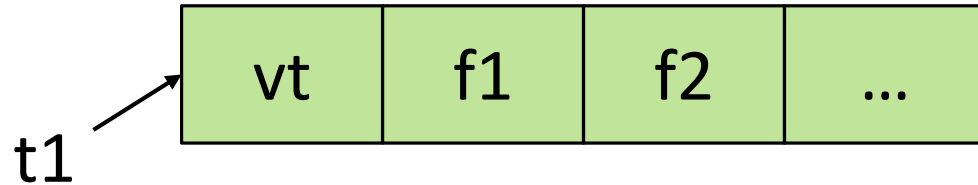
```
t0 = field_access t1, f2
```



```
beq $t1, 0, abort  
lw $t0, 8($t1)  
...  
abort:  
li $v0, 10  
syscall
```

Field Access

```
field_set t0, f2, t1
```

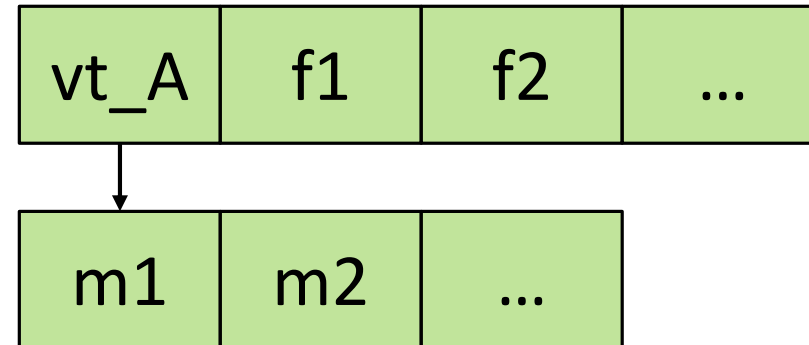


```
beq $t0, 0, abort  
sw $t1, 8($t0)  
...  
abort:  
li $v0, 10  
syscall
```

Creating Objects

```
class A {  
    int f1;  
    ...  
    int m1() { ...  
    ...  
}
```

```
A a = new A;
```



Creating Objects

```
class A {  
    int f1 = c;  
    ...  
    int m1() { ...  
    ...  
}
```

```
A a = new A;
```

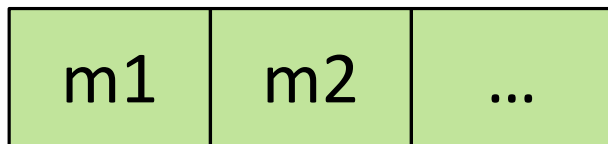
```
t0 = alloc (size)  
set_vt t0, A  
t1 = c  
field_set t0, f1, t1
```

IR

Creating Objects

```
t0 = alloc (size)
set_vt t0, A
t1 = c
field_set t0, f1, t1
...
```

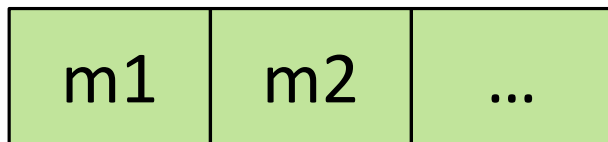
```
.data
vt_A: .word m1, m2, ...
```



Virtual Table

Creating Objects

```
t0 = alloc (size)
set_vt t0, A
t1 = c
field_set t0, f1, t1
...
```



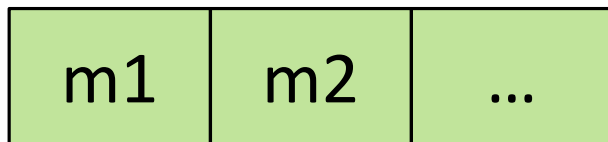
Virtual Table

```
.data
vt_A: .word m1, m2, ...
```

```
.text
li $v0, 9
li $a0, size
syscall
move $t0, $v0
```

Creating Objects

```
t0 = alloc (size)
set_vt t0, A
t1 = c
field_set t0, f1, t1
...
```



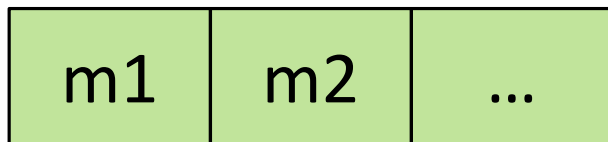
Virtual Table

```
.data
vt_A: .word m1, m2, ...

.text
li $v0, 9
li $a0, size
syscall
move $t0, $v0
la $s0, vt_A
sw $s0, 0($t0)
```

Creating Objects

```
t0 = alloc (size)
set_vt t0, A
t1 = c
field_set t0, f1, t1
...
```



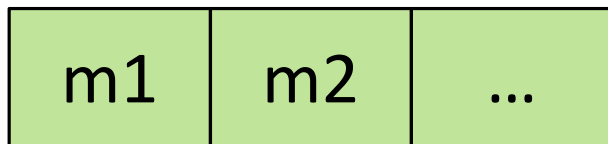
Virtual Table

```
.data
vt_A: .word m1, m2, ...

.text
li $v0, 9
li $a0, size
syscall
move $t0, $v0
la $s0, vt_A
sw $s0, 0($t0)
li $t1, c
```

Creating Objects

```
t0 = alloc (size)
set_vt t0, A
t1 = c
field_set t0, f1, t1
...
```



Virtual Table

```
.data
vt_A: .word m1, m2, ...
```

```
.text
li $v0, 9
li $a0, size
syscall
move $t0, $v0
la $s0, vt_A
sw $s0, 0($t0)
li $t1, c
sw $t1, 4($t0)
```

Method Calls

```
class A {  
    int f1 = c;  
    ...  
    int m1(int x) { ...  
    int m2(int x) { ...  
}
```

```
A a = new A;  
z = a.m2(7)
```

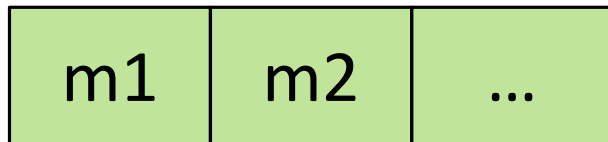
```
t0 = a  
t1 = 7  
t2 = virtual_call t0, m2, t1  
z = t2
```

IR

Method Calls

```
t0 = a  
t1 = 7  
t2 = virtual_call t0, m2, t1  
z = t2
```

```
subu $sp, $sp, 4  
sw $t1, 0($sp)
```

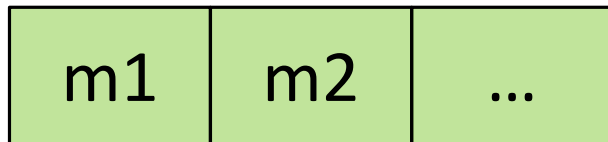


Virtual Table

Method Calls

```
t0 = a  
t1 = 7  
t2 = virtual_call t0, m2, t1  
z = t2
```

```
subu $sp, $sp, 4  
sw $t1, 0($sp)  
subu $sp, $sp, 4  
sw $t0, 0($sp)
```

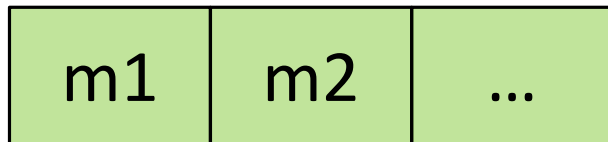


Virtual Table

Method Calls

```
t0 = a  
t1 = 7  
t2 = virtual_call t0, m2, t1  
z = t2
```

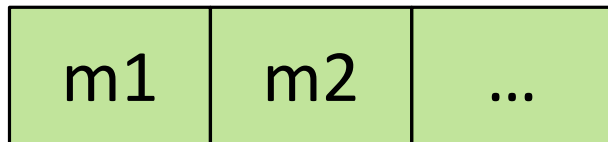
```
subu $sp, $sp, 4  
sw $t1, 0($sp)  
subu $sp, $sp, 4  
sw $t0, 0($sp)  
lw $s0, 0($t0)
```



Virtual Table

Method Calls

```
t0 = a  
t1 = 7  
t2 = virtual_call t0, m2, t1  
z = t2
```



Virtual Table

```
subu $sp, $sp, 4  
sw $t1, 0($sp)  
subu $sp, $sp, 4  
sw $t0, 0($sp)  
lw $s0, 0($t0)  
lw $s1, 4($s0)
```

Method Calls

```
t0 = a  
t1 = 7  
t2 = virtual_call t0, m2, t1  
z = t2
```

m1	m2	...
----	----	-----

Virtual Table

```
subu $sp, $sp, 4  
sw $t1, 0($sp)  
subu $sp, $sp, 4  
sw $t0, 0($sp)  
lw $s0, 0($t0)  
lw $s1, 4($s0)  
jalr $s1  
addu $sp, $sp, 8
```