



Tema 3 (III)

Fundamentos de la programación en ensamblador



Grupo ARCOS

Estructura de Computadores
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Contenidos

- I. Programación en ensamblador (III)
 1. Modos de direccionamiento
 2. Tipos de juegos de instrucciones
 3. Funciones: marco de pila

¡ATENCIÓN!

- ❑ Estas transparencias son un guión para la clase
- ❑ Los libros dados en la bibliografía junto con lo explicado en clase representa el material de estudio para el temario de la asignatura

Contenidos

- I. Programación en ensamblador (III)
 - 1. **Modos de direccionamiento**
 - 2. Tipos de juegos de instrucciones
 - 3. Funciones: marco de pila

Modos de direccionamiento

► Implícito

► Inmediato

► Directo {

- a registro
- a memoria

► Indirecto {

- a registro
- a memoria
- relativo

 {

- a registro índice
- a registro base
- a PC
- a Pila

Modos de direccionamiento

► Implícito

► Inmediato

► Directo

- a registro
- a memoria

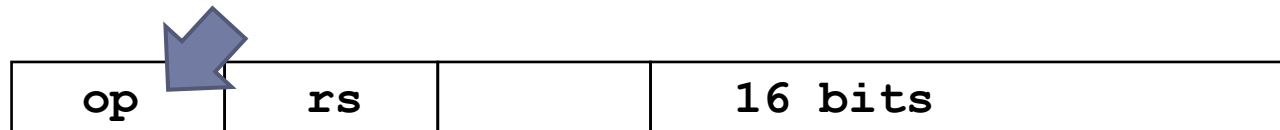
► Indirecto

- a registro
- a memoria
- relativo

- a registro índice
- a registro base
- a PC
- a Pila

Direccionamiento implícito

- ▶ El operando no está codificado en la instrucción, pero forma parte de esta.
- ▶ Ejemplo: **beqz \$a0 etiqueta**
 - ▶ Si registro \$a0 es cero, salta a etiqueta.
 - ▶ \$a0 es un operando, \$zero es el otro.



- ▶ V/I (Ventajas/Inconvenientes)
 - ✓ Es rápido: no es necesario acceder a memoria.
 - ✗ Pero solo es posible en unos pocos casos.

Modos de direccionamiento

► Implícito

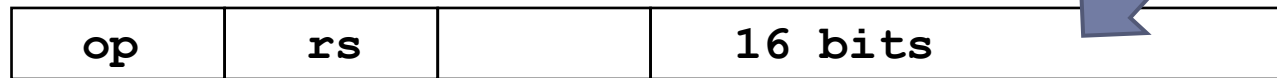
► Inmediato

► Directo { - a registro - a memoria

► Indirecto { - a registro - a memoria - relativo { - a registro índice - a registro base - a PC - a Pila

Direccionamiento inmediato

- ▶ El operando es parte de la instrucción.
- ▶ Ejemplo: `li $a0 0x00004f5`
 - ▶ Carga en el registro \$a0 el valor inmediato 0x00004f5.
 - ▶ El valor 0x00004f5 está codificado en la propia instrucción.



- ▶ V/I
 - ✓ Es rápido: no es necesario acceder a memoria.
 - ✗ No siempre cabe el valor:
 - ▶ Ej.: carga de valores de 32 bits de dos veces.

Modos de direccionamiento

► Implícito

► Inmediato

► Directo

- a registro
- a memoria

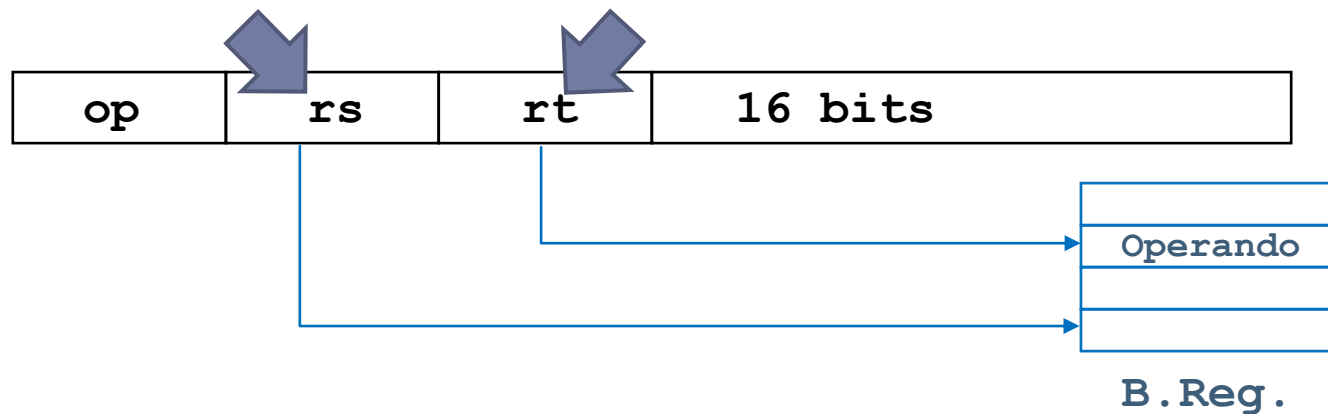
► Indirecto

- a registro
- a memoria
- relativo

- a registro índice
- a registro base
- a PC
- a Pila

Direccionamiento directo a registro

- ▶ El operando se encuentra en el registro.
- ▶ Ejemplo: `move $a0 $a1`
 - ▶ Copia en el registro \$a0 el valor que hay en el registro \$a1.
 - ▶ El identificador de \$a0 y \$a1 está codificado en la instrucción.



▶ V/I

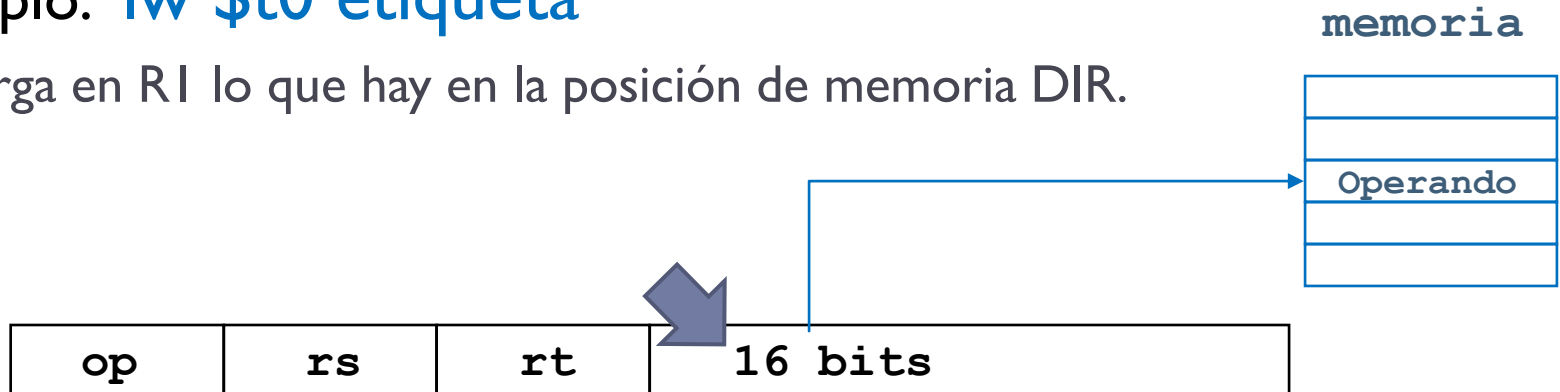
- ✗ El número de registros está limitado.
- ✓ Ejecución muy rápida (no hay acceso a memoria)

Direccionamiento directo a memoria

- ▶ El operando se encuentra en memoria, y la dirección está codificada en la instrucción.

- ▶ Ejemplo: **lw \$t0 etiqueta**

- ▶ Carga en R1 lo que hay en la posición de memoria DIR.



- ▶ V/I

- ✗ Acceso a memoria es más lento

- ✓ Acceso a un gran espacio de direcciones (capacidad > B.R.)

Modos de direccionamiento

► Implícito

► Inmediato

► Directo {

- a registro
- a memoria

► Indirecto {

- a registro
- a memoria
- relativo {
 - a registro índice
 - a registro base
 - a PC
 - a Pila

Direccionamiento directo vs. indirecto

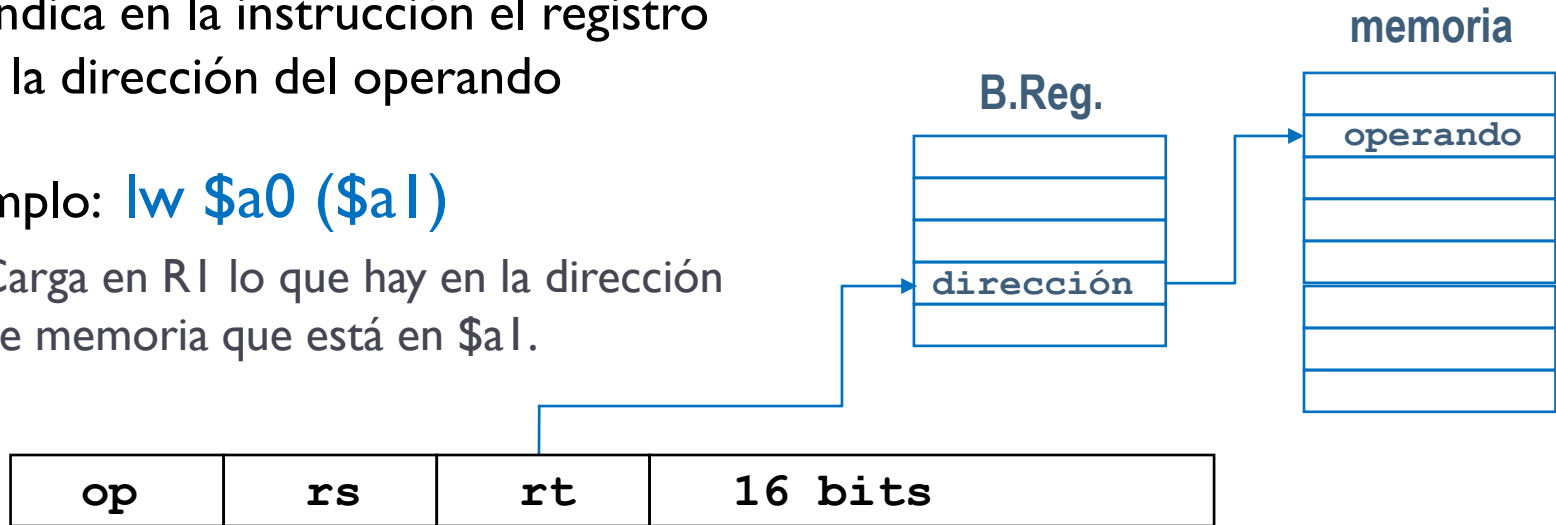
- ▶ En el direccionamiento directo se indica **dónde está el operando**:
 - ▶ En qué registro o en qué posición de memoria
- ▶ En el direccionamiento indirecto se indica **dónde está la dirección del operando**:
 - ▶ Hay que acceder a esa dirección en memoria
 - ▶ Se incorpora un nivel (o varios) de indireccionamiento

Direccionamiento indirecto a registro

- ▶ Se indica en la instrucción el registro con la dirección del operando

- ▶ Ejemplo: **lw \$a0 (\$a1)**

- ▶ Carga en R1 lo que hay en la dirección de memoria que está en \$a1.



- ▶ V/I

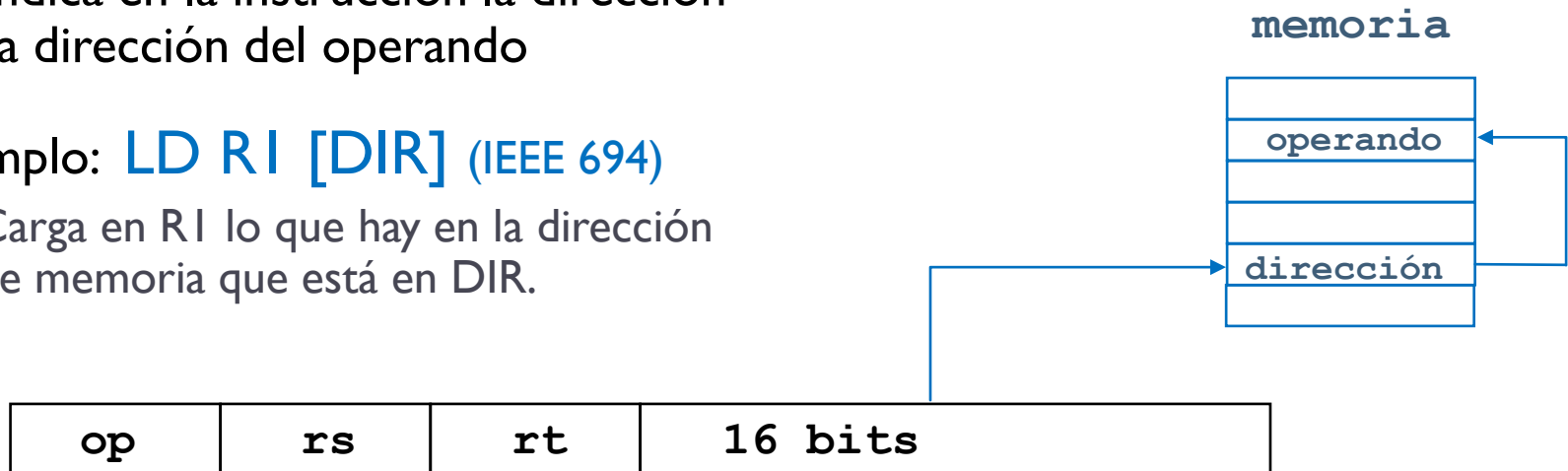
✓ Amplio espacio de direcciones

✗ Necesita un acceso menos a memoria que el indirecto a memoria

- ▶ La dirección de donde se encuentra el operando está en un registro

Direccionamiento indirecto a memoria

- ▶ Se indica en la instrucción la dirección de la dirección del operando
- ▶ Ejemplo: **LD RI [DIR]** (IEEE 694)
 - ▶ Carga en RI lo que hay en la dirección de memoria que está en DIR.



- ▶ V/I
 - ✓ Amplio espacio de direcciones
 - ✓ El direccionamiento puede ser anidado, multinivel o en cascada
 - ▶ Ejemplo: LD RI [[[.RI]]]
 - ✗ Puede requerir varios accesos memoria
 - ▶ Es más lento

Modos de direccionamiento

► Implícito

► Inmediato

► Directo {

- a registro
- a memoria

► Indirecto {

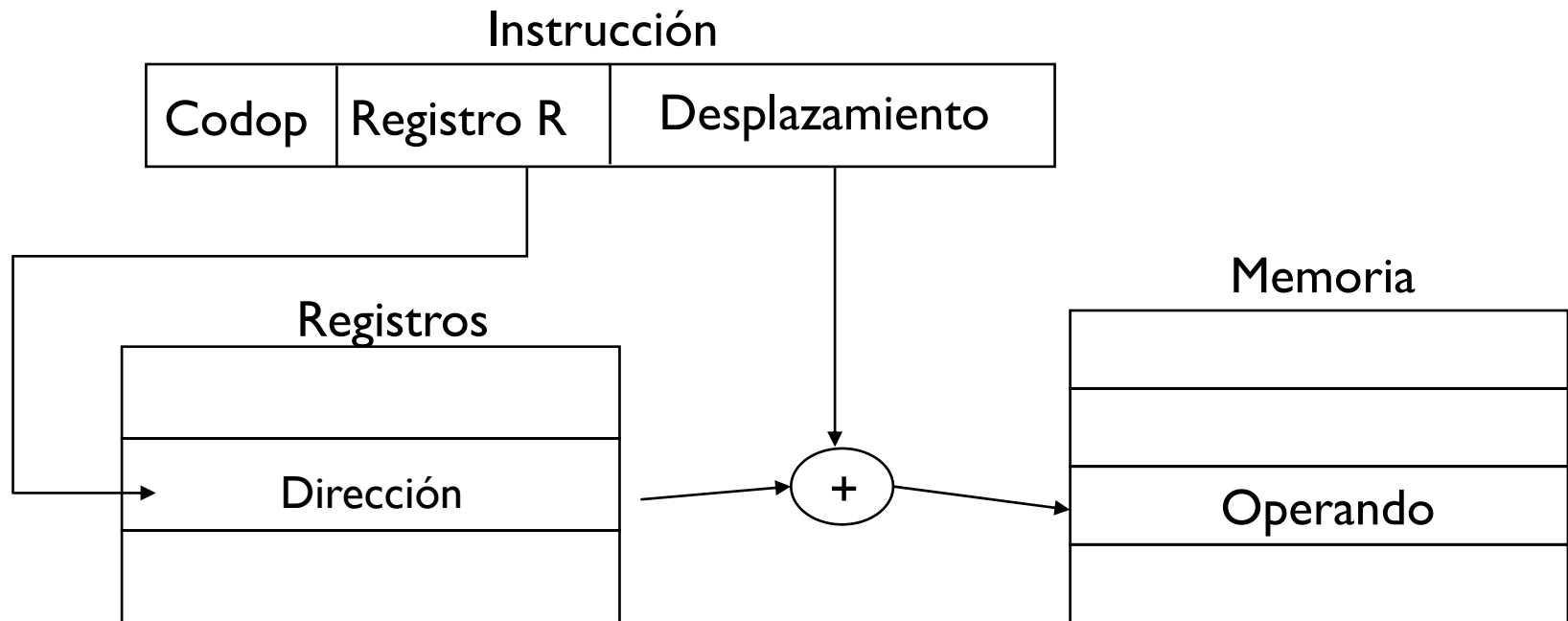
- a registro
- a memoria
- relativo

 {

- a registro índice
- a registro base
- a PC
- a Pila

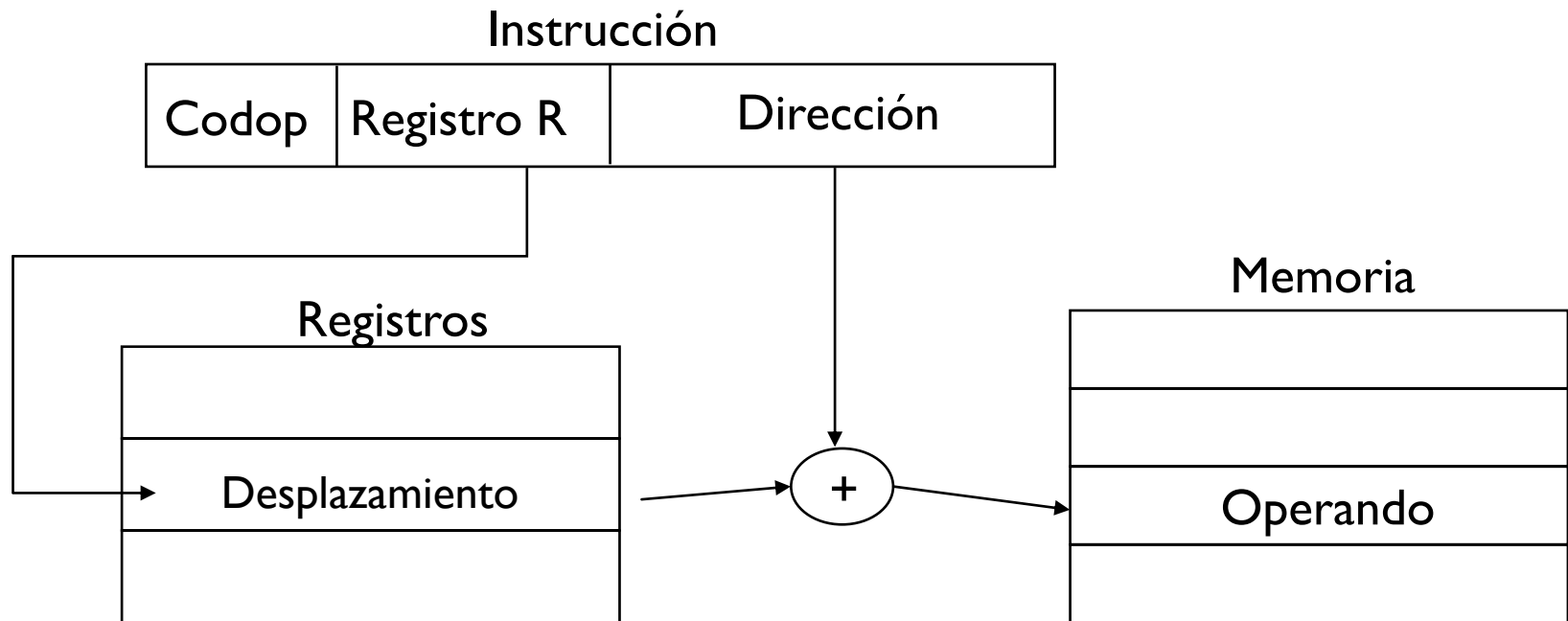
Direccionamiento indirecto relativo a registro base

- Ejemplo: `lw $a0 12($t1)`
 - Carga en \$a0 lo que hay en la posición de memoria dada por $\$t1 + 12$
 - $\$t1$ tiene la dirección base



Direccionamiento indirecto relativo a registro índice

- Ejemplo: `lw $a0 dir($t1)`
 - Carga en \$a0 lo que hay en la posición de memoria dada por `dir` + `$t1`
 - `$t1` tiene el desplazamiento (índice) respecto a la dirección `dir`



Ejemplo (base/índice)



```
int v[5] ;
```

```
main ( )
```

```
{
```

```
    v[3] = 5 ;
```

```
    v[4] = 8 ;
```

```
}
```

```
.data
```

```
v: .space 20    # 5int*4bytes/int
```

```
.text
```

```
.globl main
```

```
main:
```

```
    la $t0 v
```

```
    li $t1 5
```

```
    sw $t1 12($t0)
```

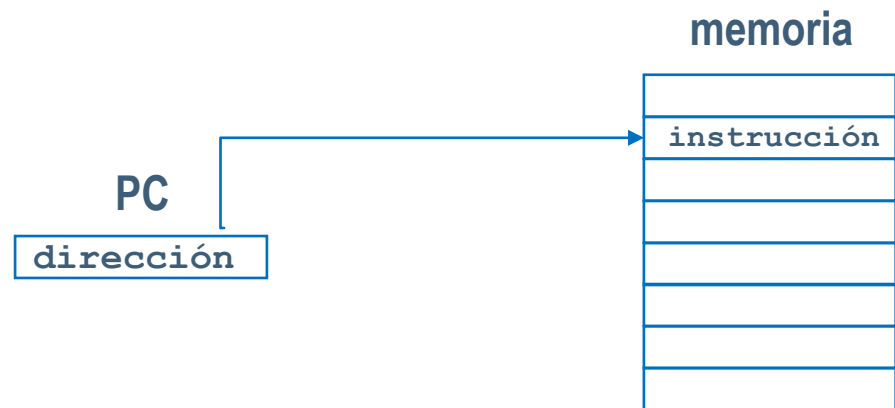
```
    la $t0 16
```

```
    li $t1 8
```

```
    sw $t1 v($t0)
```

Direccionamiento relativo al contador de programa

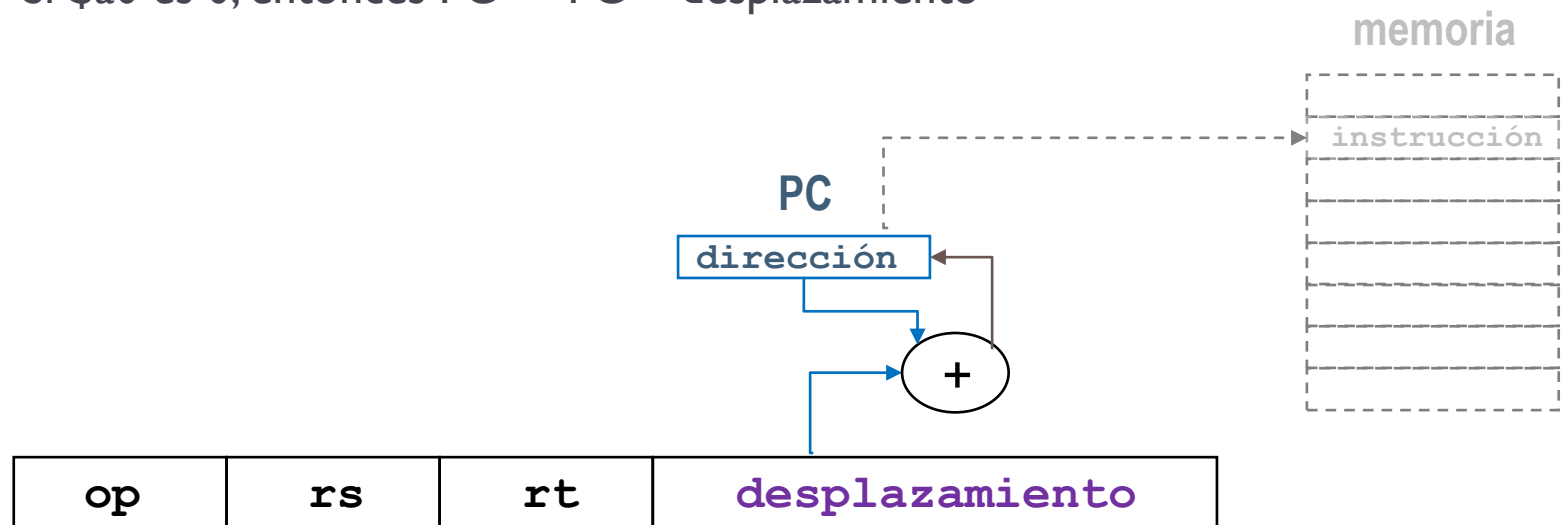
- ▶ El contador de programa PC:
 - ▶ Es un registro de 32 bits (4 bytes)
 - ▶ Almacena la dirección de la siguiente instrucción a ejecutar
 - ▶ Apunta a una palabra (4 bytes) con la instrucción a ejecutar



Direccionamiento relativo al contador de programa

► Ejemplo: `beqz $a0 etiqueta`

- La instrucción codifica etiqueta como el **desplazamiento** desde la dirección de memoria donde está esta instrucción, hasta la posición de memoria indicada en **etiqueta**.
- Si `$a0` es 0, entonces $PC \leftarrow PC + \text{desplazamiento}$



Ejercicio

2 minutos máx.

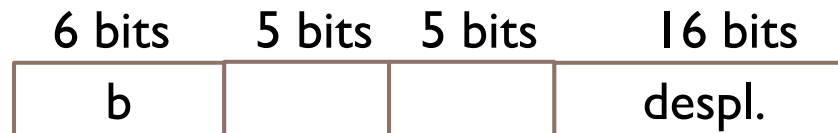


- ▶ Dadas estas 2 instrucciones para realizar un salto incondicional:

- ▶ 1) **j etiqueta1**



- ▶ 2) **b etiqueta2**



- ▶ Donde en la primera se carga la dirección en PC y en la segunda se suma el desplazamiento a PC (siendo este un número en complemento a dos)
- ▶ Se pide:
 - ▶ Indique razonadamente cual de las dos opciones es más apropiada para bucles pequeños.

Ejercicio (solución)



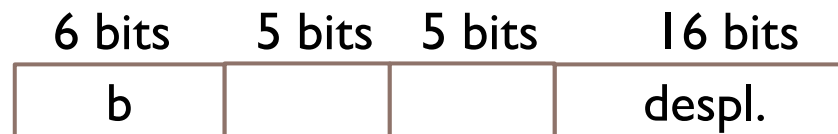
► Ventajas de la opción 1:

- El cálculo de la dirección es más rápido, solo cargar
- El rango de direcciones es mayor, mejor para bucles grandes



► Ventajas de la opción 2:

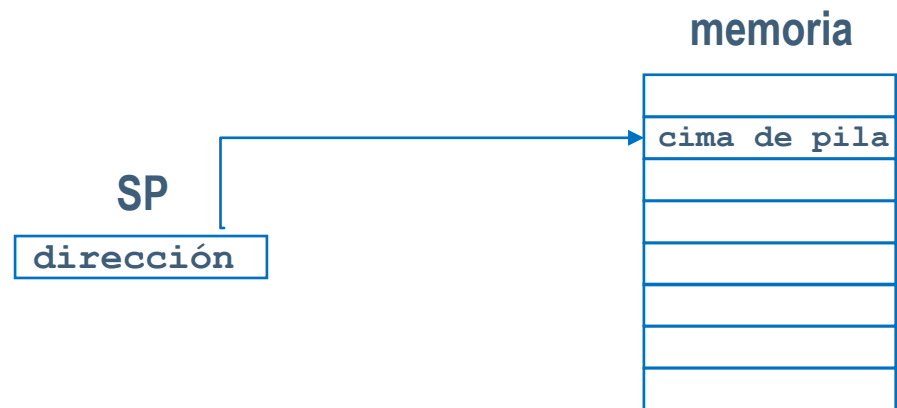
- El rango de direcciones a las que se puede saltar es menor (bucles pequeños)
- Permite un código relocizable



► La opción 2 sería más apropiada

Direccionamiento relativo a la pila

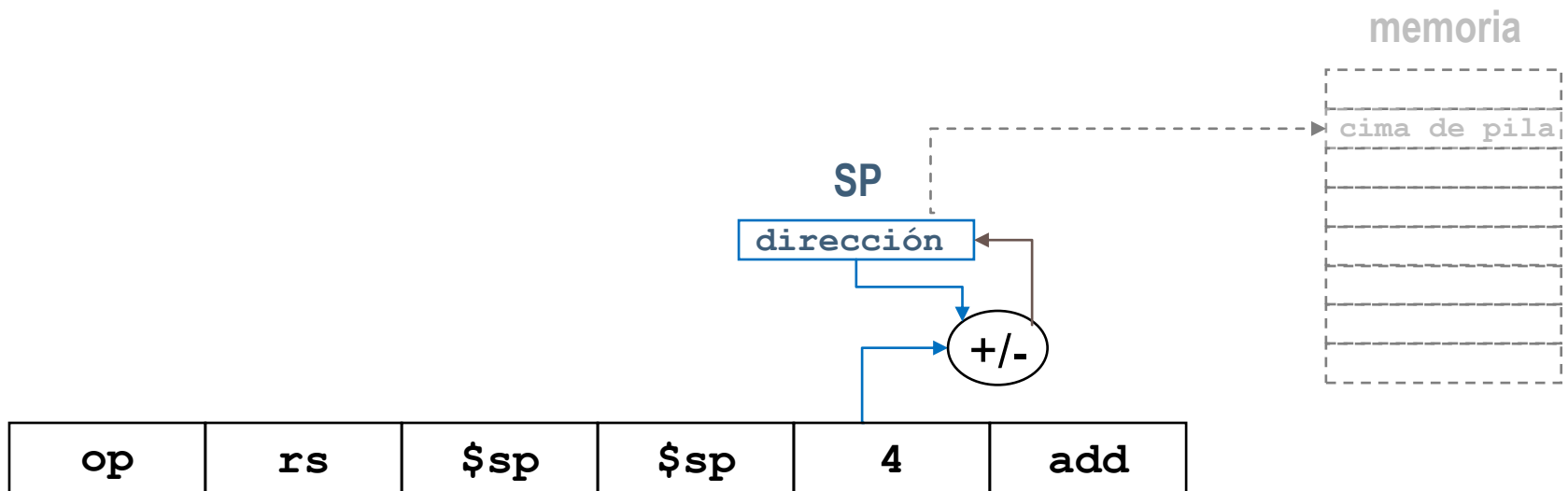
- ▶ El puntero de pila SP (*Stack Pointer*):
 - ▶ Es un registro de 32 bits (4 bytes)
 - ▶ Almacena la dirección de la cima de pila
 - ▶ Apunta a una palabra (4 bytes)
- ▶ Tres tipos de operaciones:
 - ▶ **push** \$registro
 - $\$sp = \$sp - 4$
 - $mem[\$sp] = \$registro$
 - ▶ **pop** \$registro
 - $\$registro = mem[\$sp]$
 - $\$sp = \$sp + 4$
 - ▶ **top** \$registro
 - $\$registro = mem[\$sp]$



Direccionamiento relativo a la pila

► Ejemplo: **push \$a0**

- `sub $sp $sp 4` # $\$SP = \$SP - 4$
- `sw $a0 ($sp)` # `memoria[$SP] = $a0`



Ejercicio

4 minutos máx.



- Indique el tipo de direccionamiento usado en las siguientes instrucciones MIPS:

1. `li $t1 4`
2. `lw $t0 4($a0)`
3. `bnez $a0 etiqueta`

Ejercicio (solución)



1. `li $t1 4`

- ▶ `$t1` -> directo a registro
- ▶ `4` -> inmediato



Qué es, y que se hace con él

2. `lw $t0 4($a0)`

- ▶ `$t0` -> directo a registro
- ▶ `4($a0)` -> indirecto relativo a registro base

3. `bnez $a0 etiqueta`

- ▶ `$a0` -> directo a registro
- ▶ `etiqueta` -> indirecto relativo a contador de programa

Ejemplos de tipos de direccionamiento

- ▶ **la \$t0 label inmediato**
 - ▶ El segundo operando de la instrucción es una dirección
 - ▶ PERO no se accede a esta dirección, la propia dirección es el operando
- ▶ **lw \$t0 label directo a memoria**
 - ▶ El segundo operando de la instrucción es una dirección
 - ▶ Hay que acceder a esta dirección para tener el valor con el que trabajar
- ▶ **bne \$t0 \$t1 label relativo a registro PC**
 - ▶ El tercer operando operando de la instrucción es una dirección
 - ▶ PERO no se accede a esta dirección, la propia dirección es PARTE del operando
 - ▶ En el formato de esta instrucción, label se codifica como un número en complemento a dos que representa el desplazamiento (como palabras) relativo al registro PC

Direccionamientos en MIPS

► Direccionamientos:

- Inmediato**

► Directo

- ▶ A memoria dir
- ▶ A registro \$r

▶ Indirecto

- A registro (dir)

► Relativo

- | | |
|-------------------------------------|----------------------|
| <input type="checkbox"/> A registro | desplazamiento(\$r) |
| <input type="checkbox"/> A pila | desplazamiento(\$sp) |
| <input type="checkbox"/> A PC | beq ... etiqueta l |

Contenidos

- I. Programación en ensamblador (III)
 1. Modos de direccionamiento
 2. **Tipos de juegos de instrucciones**
 3. Funciones: marco de pila

Juego de instrucciones

- ▶ Queda **definido** por:
 - ▶ Conjunto de instrucciones
 - ▶ Formato de la instrucciones
 - ▶ Registros
 - ▶ Modos de direccionamiento
 - ▶ Tipos de datos y formatos

Juego de instrucciones

- ▶ Distintas formas para la **clasificación** de un juego de instrucciones:
 - ▶ Complejidad del juego de instrucciones
 - ▶ CISC vs RISC
 - ▶ Modo de ejecución
 - ▶ Pila
 - ▶ Registro
 - ▶ Registro-Memoria, Memoria-Registro, ...

Juego de instrucciones

- ▶ Distintas formas para la **clasificación** de un juego de instrucciones:

- ▶ Complejidad del juego de instrucciones

- ▶ CISC vs RISC

- ▶ Modo de ejecución

- ▶ Pila
 - ▶ Registro
 - ▶ Registro-Memoria, Memoria-Registro, ...

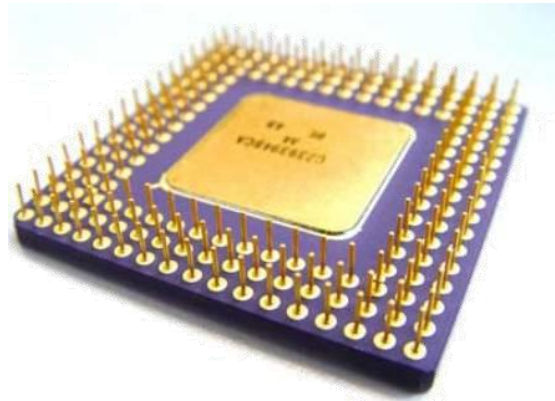
CISC

- ▶ *Complex Instruction Set Computer*
- ▶ Muchas instrucciones
- ▶ Complejidad variable
 - ▶ Instrucciones complejas
 - ▶ Más de una palabra
 - ▶ Unidad de control más compleja
 - ▶ Mayor tiempo de ejecución
- ▶ Diseño irregular

CISC vs RISC

► Observación:

- Alrededor del 20% de las instrucciones ocupa el 80% del tiempo total de ejecución de un programa
- El 80% de las instrucciones no se utilizan casi nunca
- 80% del silicio infrautilizado, complejo y costoso



RISC

- ▶ *Reduced Instruction Set Computer*
- ▶ Juegos de instrucciones reducidos
- ▶ Instrucciones simples y ortogonales
 - ▶ Ocupan una palabra
 - ▶ Instrucciones sobre registros
 - ▶ Uso de los mismos modos de direccionamiento para todas las instrucciones (alto grado de ortogonalidad)
- ▶ **Diseño más compacto:**
 - ▶ Unidad de control más sencilla y rápida
 - ▶ Espacio sobrante para más registros y memoria caché

Juego de instrucciones

- ▶ Distintas formas para la **clasificación** de un juego de instrucciones:
 - ▶ Complejidad del juego de instrucciones
 - ▶ CISC vs RISC
 - ▶ Modo de ejecución
 - ▶ Pila
 - ▶ Registro
 - ▶ Registro-Memoria, Memoria-Registro, ...

Modelo de ejecución

- ▶ Una máquina tiene un **modelo de ejecución** asociado.
 - ▶ Modelo de ejecución indica **el número de direcciones y tipo de operandos que se pueden especificar en una instrucción.**
- ▶ **Modelos de ejecución:**
 - ▶ 0 direcciones → Pila
 - ▶ 1 dirección → Registro acumulador
 - ▶ 2 direcciones → Registros, Registro-Memoria y Memoria-Memoria
 - ▶ 3 direcciones → Registros, Registro-Memoria y Memoria-Memoria

- ▶ 3 direcciones
- ▶ 2 direcciones
- ▶ 1 dirección
- ▶ 0 direcciones

Modelo de 3 direcciones

▶ Registro-Registro:

- ▶ Los 3 operandos son registros.
- ▶ Requiere operaciones de carga/almacenamiento.
- ▶ **ADD .R0, .R1, .R2**

▶ Memoria-Memoria:

- ▶ Los 3 operandos son direcciones de memoria.
- ▶ **ADD /DIR1, /DIR2, /DIR3**

▶ Registro-Memoria:

- ▶ Híbrido.
- ▶ **ADD .R0, /DIR1, /DIR2**
- ▶ **ADD .R0, .R1, /DIR1**

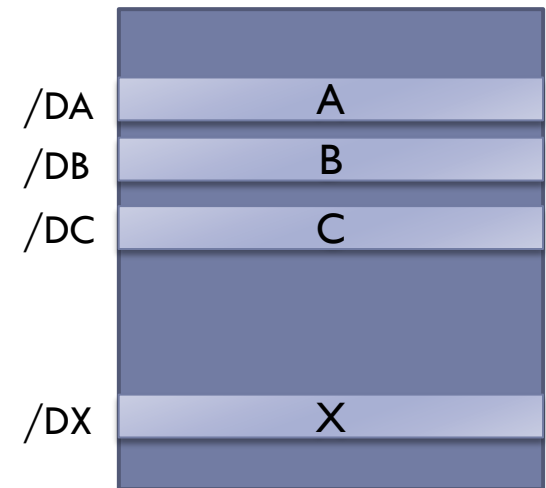
Ejercicio



► Sea la siguiente expresión matemática:

$$\text{► } X = A + B * C$$

Donde los operandos están en memoria tal y como se describe en la figura:

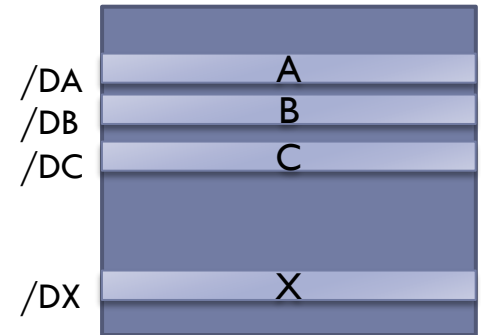


Para los modelos R-R y M-M, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

Ejercicio (solución)

$$X = A + B * C$$



► Memoria-Memoria:

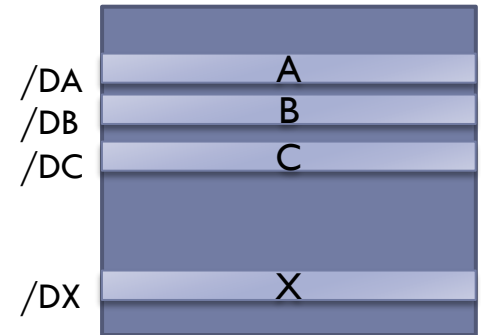
```
MUL /DX, /DB, /DC
ADD /DX, /DX, /DA
```

► Registro-Registro:

```
LOAD .R0, /DB
LOAD .R1, /DC
MUL .R0, .R0, .R1
LOAD .R2, /DA
ADD .R0, .R0, .R2
STORE .R0, /DX
```

Ejercicio (solución)

$$X = A + B * C$$



► Memoria-Memoria:

- 2 instrucciones
- 6 accesos a memoria
- 0 accesos a registros

```
MUL  /DX, /DB, /DC
ADD  /DX, /DX, /DA
```

► Registro-Registro:

- 6 instrucciones
- 4 accesos a memoria
- 10 accesos a registros

```
LOAD  .R0, /DB
LOAD  .R1, /DC
MUL    .R0, .R0, .R1
LOAD  .R2, /DA
ADD    .R0, .R0, .R2
STORE .R0, /DX
```

- ▶ 3 direcciones
- ▶ **2 direcciones**
- ▶ 1 dirección
- ▶ 0 direcciones

Modelo de 2 direcciones

▶ Registro-Registro:

- ▶ Los 2 operandos son registros.
- ▶ Requiere operaciones de carga/almacenamiento.
- ▶ **ADD .R0, .R1 (R0 <- R0 + R1)**

▶ Memoria-Memoria:

- ▶ Los 2 operandos son direcciones de memoria.
- ▶ **ADD /DIR1, /DIR2 (MP[DIR1] <- MP[DIR1] + MP[DIR2])**

▶ Registro-Memoria:

- ▶ Híbrido.
- ▶ **ADD .R0, /DIR1 (R0 <- R0 + MP[DIR1])**

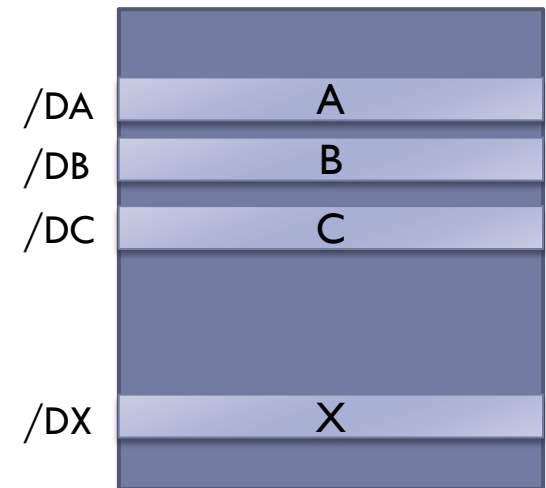
Ejercicio



► Sea la siguiente expresión matemática:

$$\text{X} = \text{A} + \text{B} * \text{C}$$

Donde los operandos están en memoria tal y como se describe en la figura:

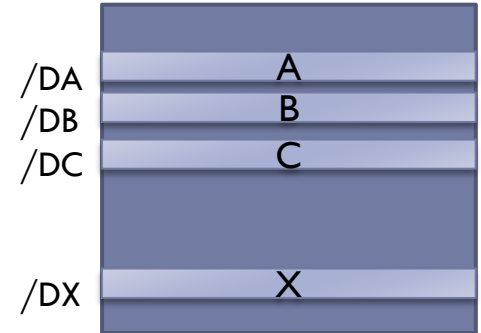


Para los modelos R-R y M-M, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

Ejercicio (solución)

$$X = A + B * C$$



► Memoria-Memoria:

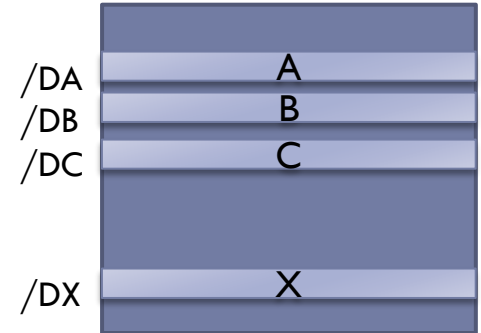
```
MOVE  /DX, /DB
MUL   /DX, /DC
ADD   /DX, /DA
```

► Registro-Registro:

```
LOAD  .R0, /DB
LOAD  .R1, /DC
MUL   .R0, .R1
LOAD  .R2, /DA
ADD   .R0, .R2
STORE .R0, /DX
```

Ejercicio (solución)

$$X = A + B * C$$



► Memoria-Memoria:

- 3 instrucciones
- 6 accesos a memoria
- 0 accesos a registros

```
MOVE  /DX, /DB
MUL   /DX, /DC
ADD   /DX, /DA
```

► Registro-Registro:

- 6 instrucciones
- 4 accesos a memoria
- 8 accesos a registros

```
LOAD  .R0, /DB
LOAD  .R1, /DC
MUL   .R0, .R1
LOAD  .R2, /DA
ADD   .R0, .R2
STORE .R0, /DX
```

- ▶ 3 direcciones
- ▶ 2 direcciones
- ▶ 1 dirección
- ▶ 0 direcciones

Modelo de 1 direcciones

- ▶ Todas las operaciones utilizan un operando implícito:
 - ▶ Registro acumulador
 - ▶ **ADD RI** $(AC \leftarrow AC + RI)$
- ▶ Operaciones de carga y almacenamiento siempre sobre el acumulador.
- ▶ Posibilidad de movimiento entre el registro acumulador y otros registros

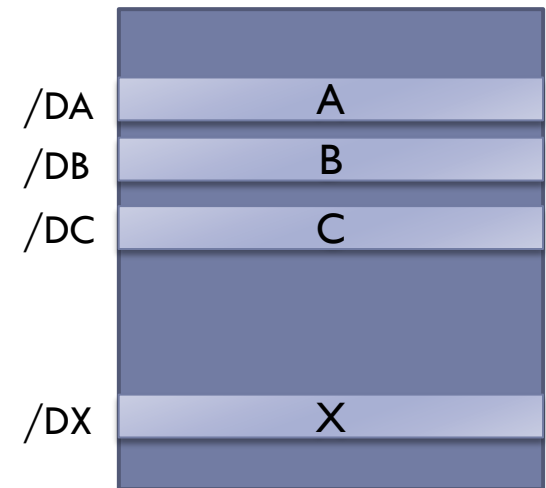
Ejercicio



► Sea la siguiente expresión matemática:

$$\text{X} = \text{A} + \text{B} * \text{C}$$

Donde los operandos están en memoria tal y como se describe en la figura:

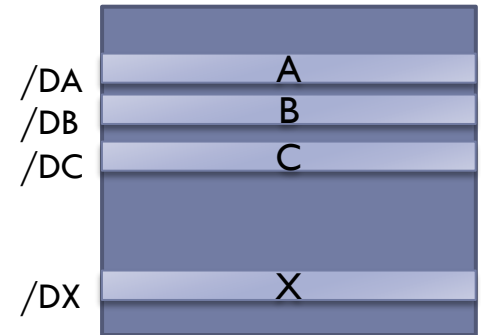


Para el modelo de 1 dirección, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

Ejercicio (solución)

$$X = A + B * C$$

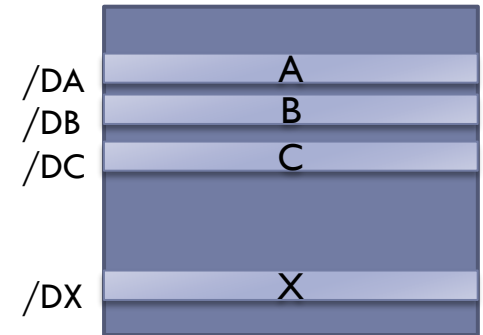


► Modelo de 1 sola dirección:

```
LOAD  /DB
MUL   /DC
ADD   /DA
STORE /DX
```

Ejercicio (solución)

$$X = A + B * C$$



► Modelo de 1 sola dirección:

- 4 instrucciones
- 4 accesos a memoria
- 0 accesos a registros

LOAD /DB
MUL /DC
ADD /DA
STORE /DX

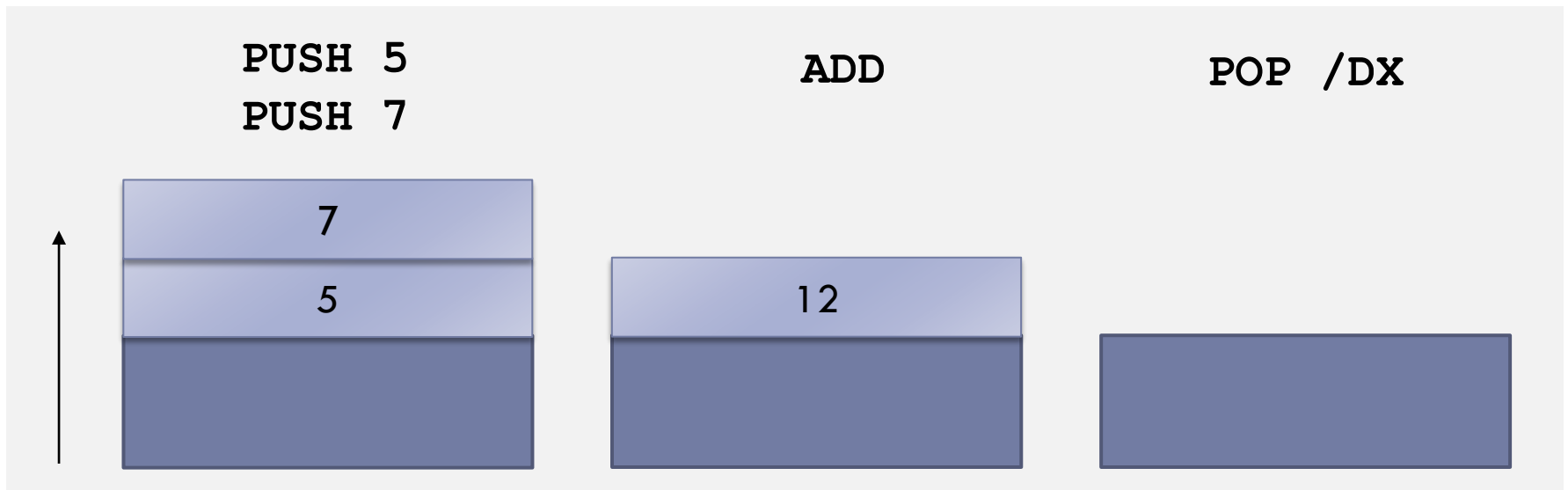
- ▶ 3 direcciones
- ▶ 2 direcciones
- ▶ 1 dirección
- ▶ 0 direcciones

Modelo de 0 direcciones

- ▶ Todas las operaciones referidas a la **pila**:
 - ▶ Los operandos están en la cima de la pila.
 - ▶ Al hacer la operación se retiran de la pila.
 - ▶ El resultado se coloca en la cima de la pila.
 - ▶ **ADD** $(pila[-1] = pila[-1] + pila[-2])$
- ▶ Dos operaciones especiales:
 - ▶ PUSH
 - ▶ POP

Ejemplo

```
push 5  
push 7  
add  
pop /dx
```



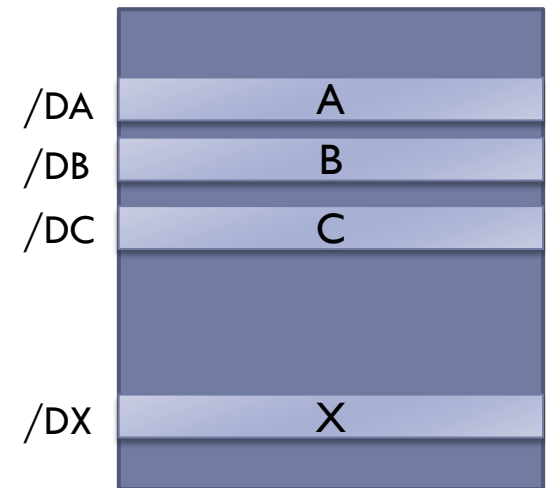
Ejercicio



► Sea la siguiente expresión matemática:

$$\text{X} = \text{A} + \text{B} * \text{C}$$

Donde los operandos están en memoria tal y como se describe en la figura:

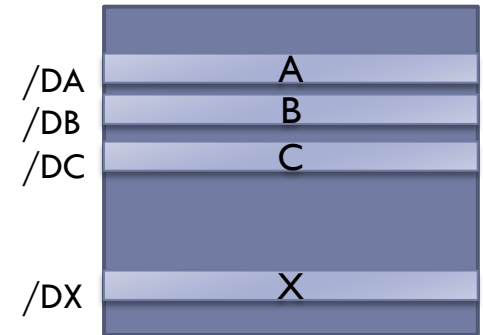


Para el modelo de 0 dirección, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

Ejercicio (solución)

$$X = A + B * C$$

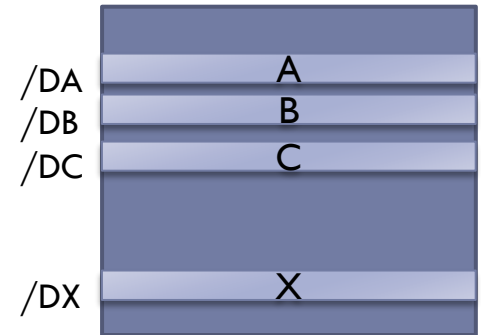


► Modelo de 0 direcciones:

```
PUSH /DB
PUSH /DC
MUL
PUSH /DA
ADD
POP /DX
```

Ejercicio (solución)

$$X = A + B * C$$



- ▶ Modelo de 0 direcciones:
 - ▶ 6 instrucciones
 - ▶ 4 accesos a memoria (datos)
 - ▶ 10 accesos a memoria (pila)
 - ▶ 0 accesos a registros

```
PUSH /DB
PUSH /DC
MUL
PUSH /DA
ADD
POP /DX
```

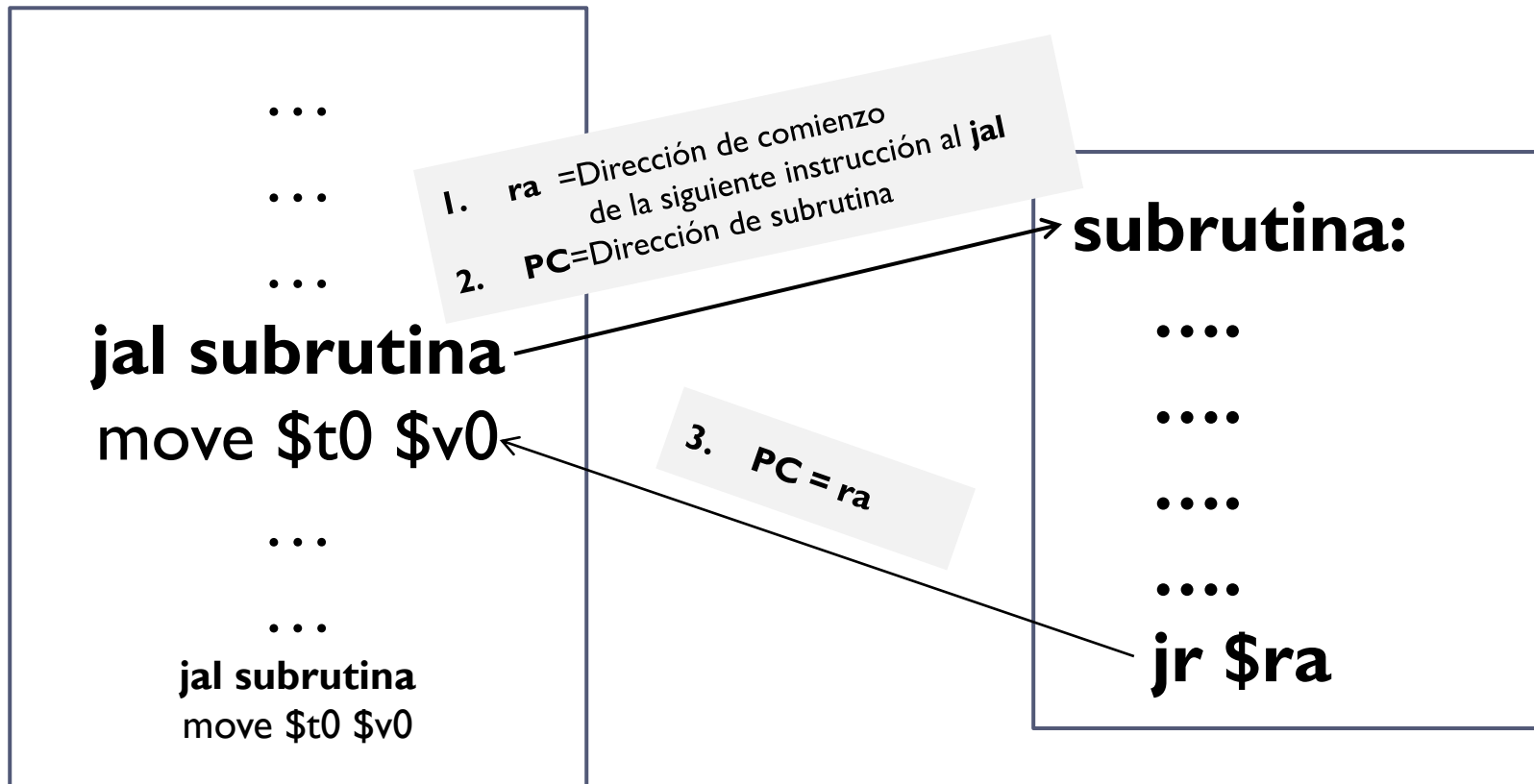

Contenidos

- I. Programación en ensamblador (III)
 1. Modos de direccionamiento
 2. Tipos de juegos de instrucciones
 3. **Funciones: marco de pila**

Funciones introducción

- ▶ Una **subrutina** es similar a un procedimiento, función o método en los lenguajes de alto nivel.
- ▶ Se precisa conocer tres aspectos:
 - ▶ Uso de la **instrucción jal/jr**
 - ▶ Uso de la **pila**
 - ▶ Uso de **marco de pila**
 - ▶ Protocolo de comunicación entre llamante y llamado
 - ▶ Convenio de organización de los datos internos

Instrucciones **jal** y **jr**

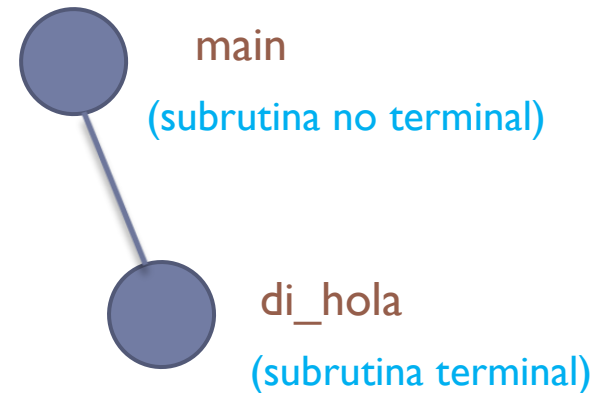


Ejemplo básico de jal+jr



```
void di_hola ( void )  
{  
    printf("hola\n") ;  
}
```

```
main ()  
{  
    di_hola() ;  
}
```



Ejemplo básico de jal+jr



```
void di_hola ( void )
{
    printf("hola\n") ;
}
```

```
main ()
{
    di_hola() ;
}
```

```
.data
msg: .asciiz "hola\n"
```

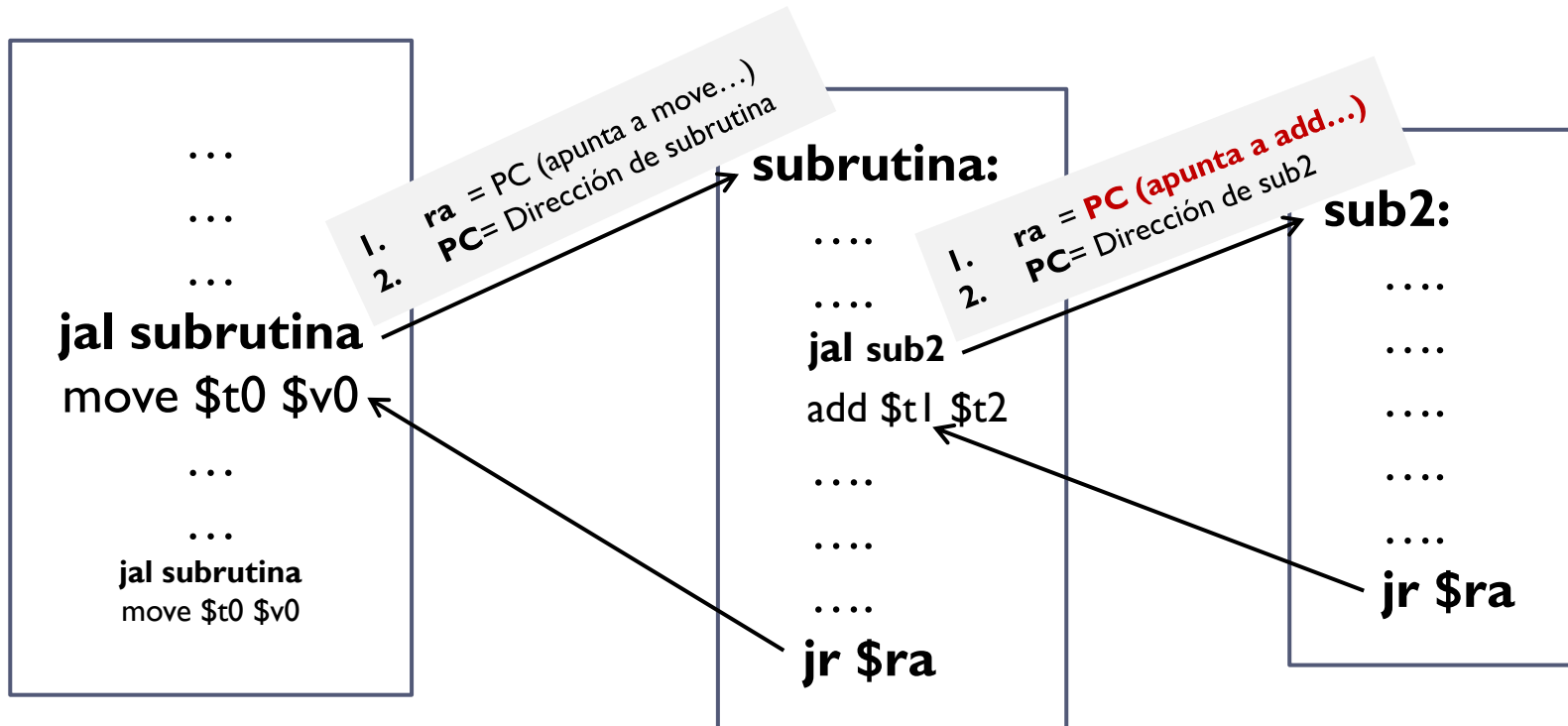
```
.text
.globl main
```

```
di_hola: la $a0 msg
         li $v0 4
         syscall
         jr $ra
```

```
main:   jal di_hola

        li $a0 10
        syscall
```

Pila: motivación



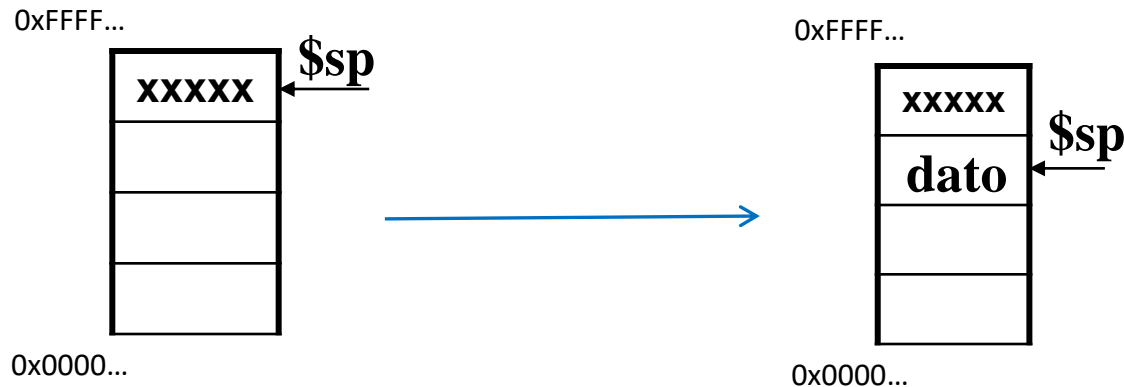
- Problema: si una subrutina llama a otra, se puede perder el valor de `$ra` (dirección de vuelta)
- Solución: usar la pila

Pila push

- ▶ jal/jr
- ▶ **Pila**
- ▶ Marco

push \$registro

subu \$sp \$sp 4
sw \$registro (\$sp)

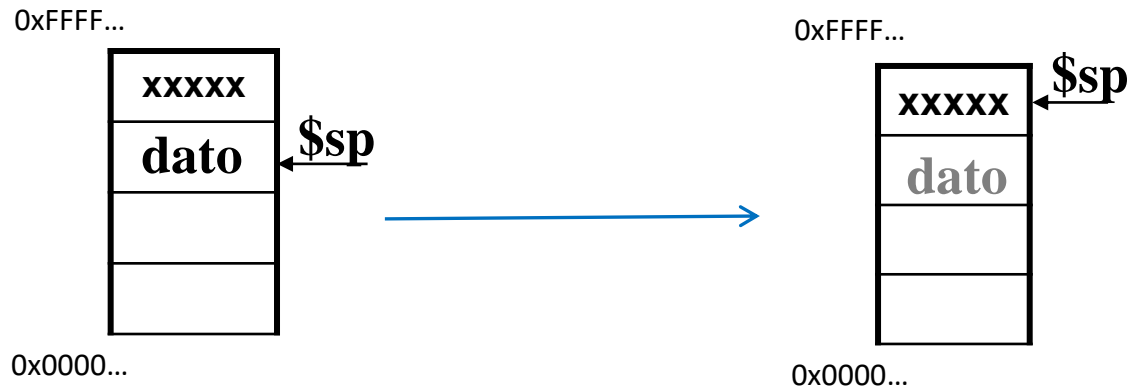


Pila pop

- ▶ jal/jr
- ▶ **Pila**
- ▶ Marco

pop \$registro

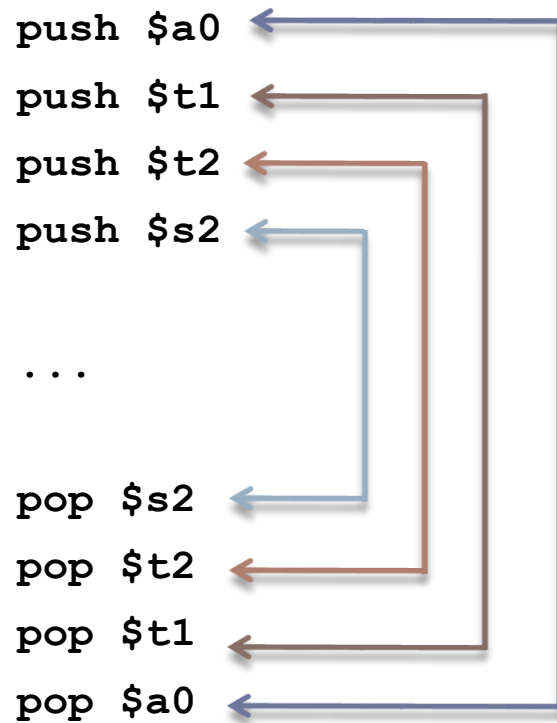
lw \$registro (\$sp)
addu \$sp \$sp 4



Pila

uso de push y pop consecutivos

- ▶ jal/jr
- ▶ **Pila**
- ▶ Marco



Pila

uso de push y pop consecutivos

- ▶ jal/jr
- ▶ **Pila**
- ▶ Marco

```
push $a0
push $t1
push $t2
push $s2
```

...

```
pop $s2
pop $t2
pop $t1
pop $a0
```

```
sub $sp $sp 4
sw $a0 ($sp)
sub $sp $sp 4
sw $t1 ($sp)
sub $sp $sp 4
sw $t2 ($sp)
sub $sp $sp 4
sw $s2 ($sp)
```

...

```
lw $s2 ($sp)
add $sp $sp 4
lw $s2 ($sp)
add $sp $sp 4
lw $s2 ($sp)
add $sp $sp 4
lw $s2 ($sp)
add $sp $sp 4
```

Marco de pila

uso de multiples push y pop agrupados

- ▶ jal/jr
- ▶ Pila
- ▶ **Marco**

```
push $a0
push $t1
push $t2
...
push $s2
```

...

```
pop $s2
...
pop $t2
pop $t1
pop $a0
```

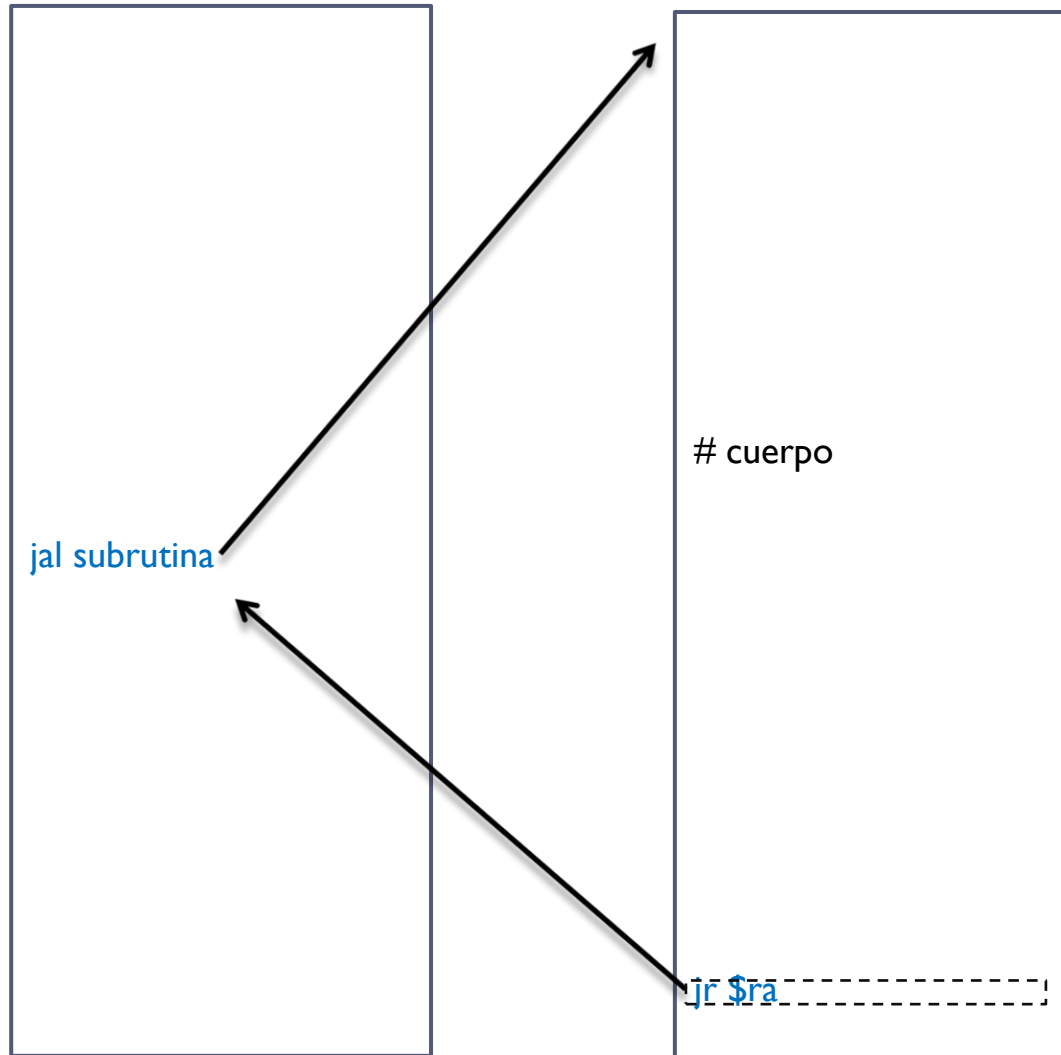
```
sub $sp $sp 16
sw $a0 16($sp)
sw $t1 12($sp)
sw $t2 8($sp)
...
sw $s2 4($sp)
```

...

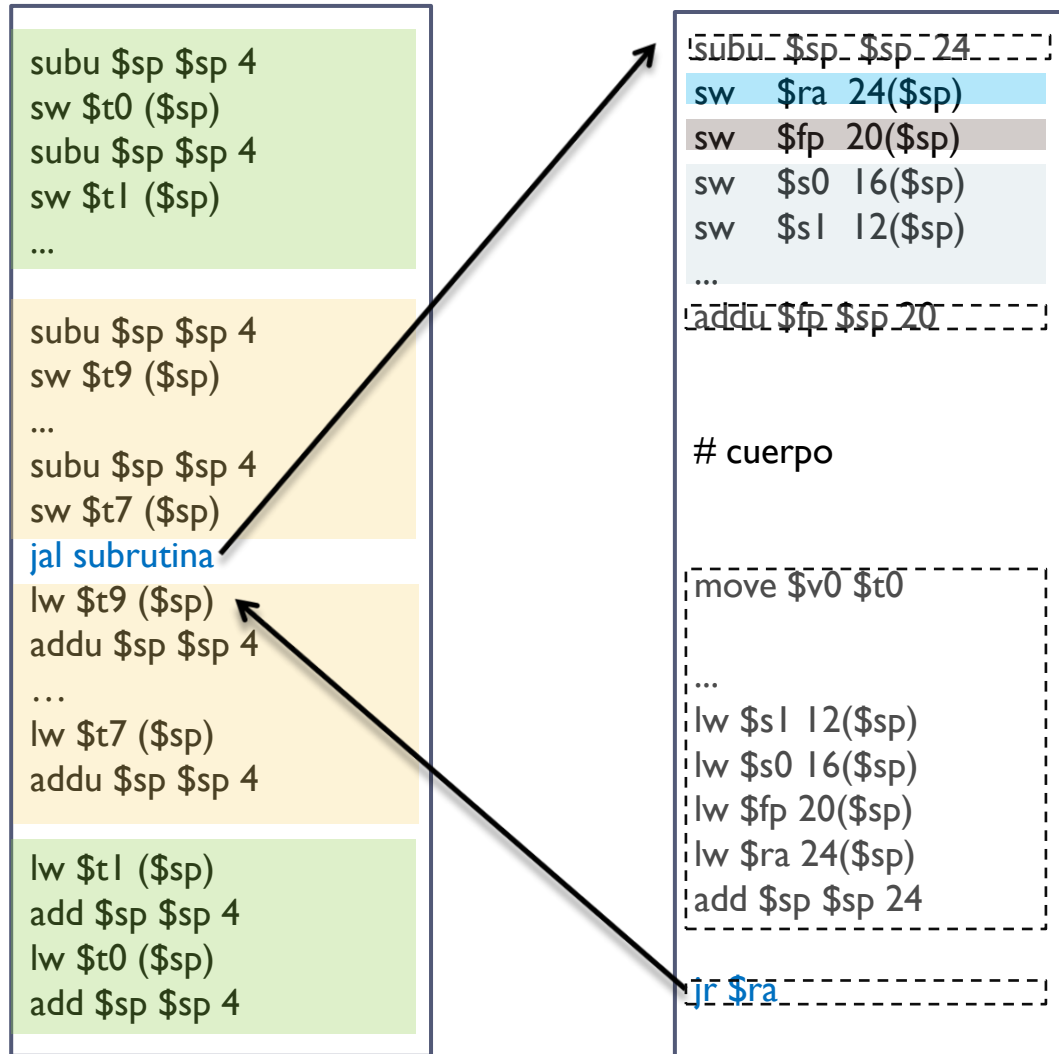
```
lw $s2 4($sp)
...
lw $t2 8($sp)
lw $t1 12($sp)
lw $a0 16($sp)
add $sp $sp 16
```

- ▶ jal/jr
- ▶ Pila
- ▶ **Marco**

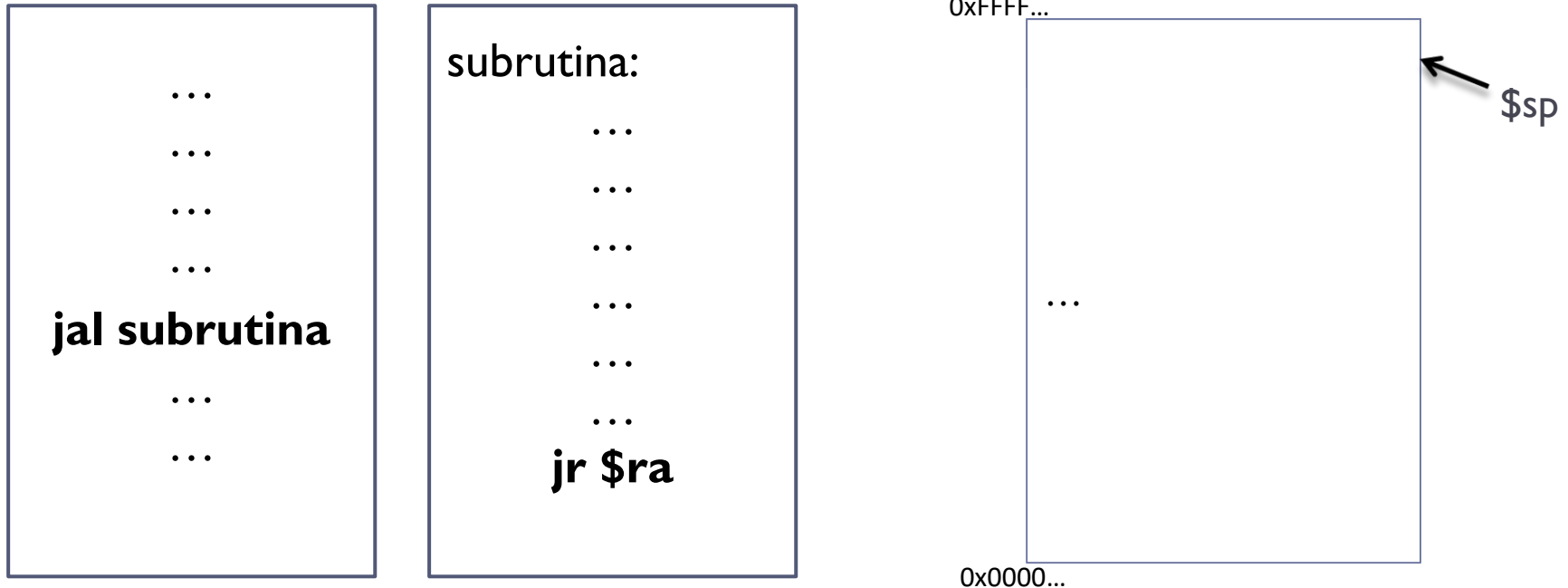
Marco de pila: resumen



Marco de pila: resumen

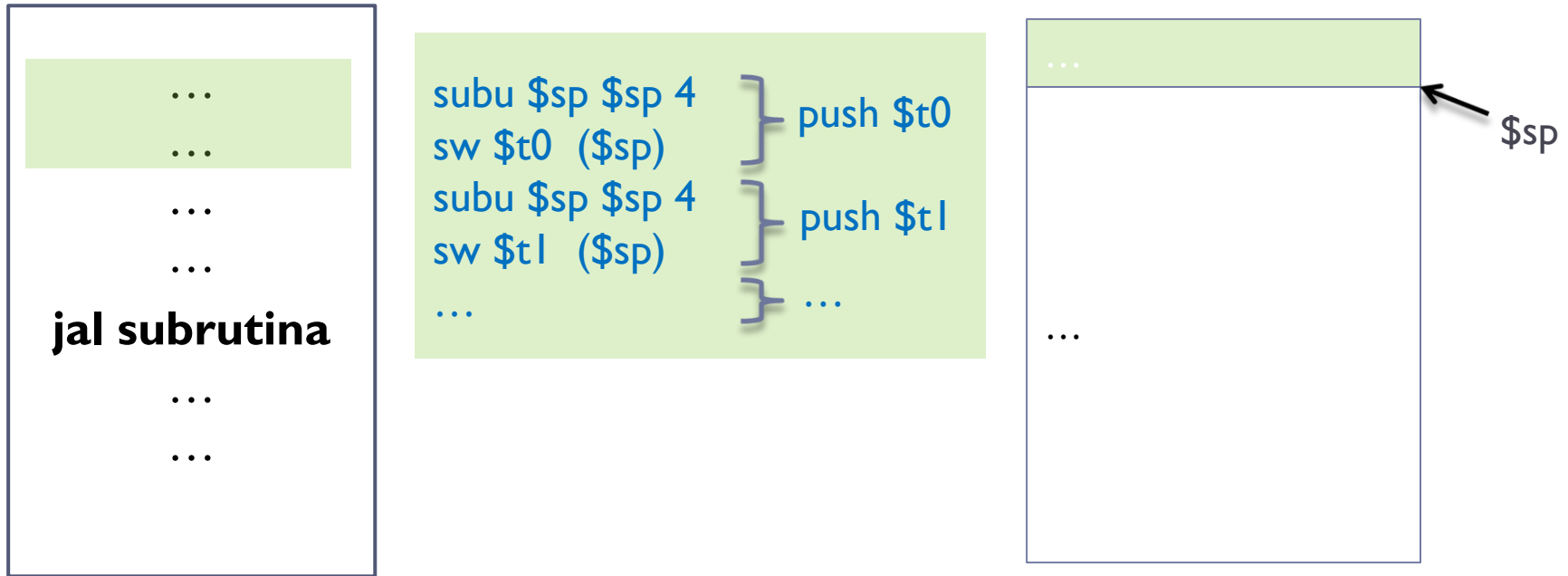


Marco de pila



- ▶ **Llamante:** quién realiza la llamada (jal/...)
- ▶ **Llamado:** quién es llamado (al final ejecuta jr \$ra)
 - ▶ Un llamado puede convertirse a su vez en llamante

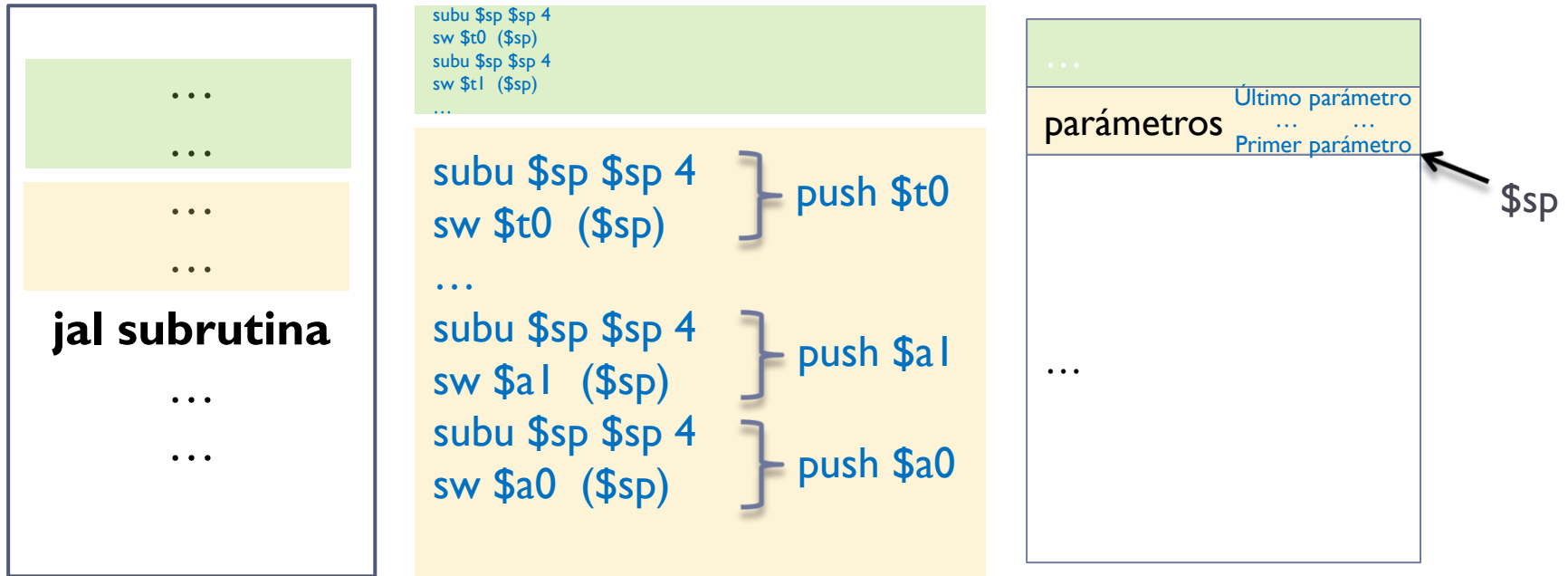
Marco de pila: llamante debe.... (1 / 3)



► Salvaguardar registros

- Una subrutina puede modificar cualquier registro **\$t0..\$t9**.
- Para preservar su valor, es necesario guardar en pila esos registros antes de la llamada a subrutina.
 - Se guardan en el registro de activación del llamante.

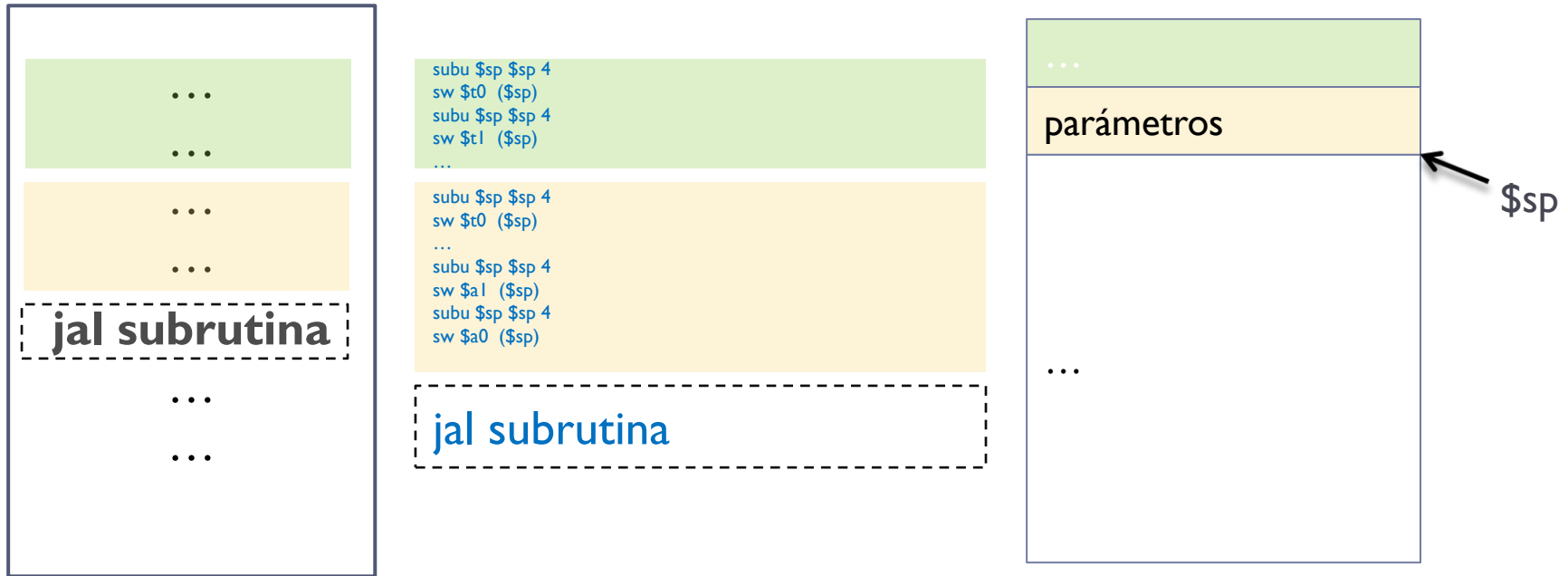
Marco de pila: llamante debe.... (2/3)



► Paso de argumentos

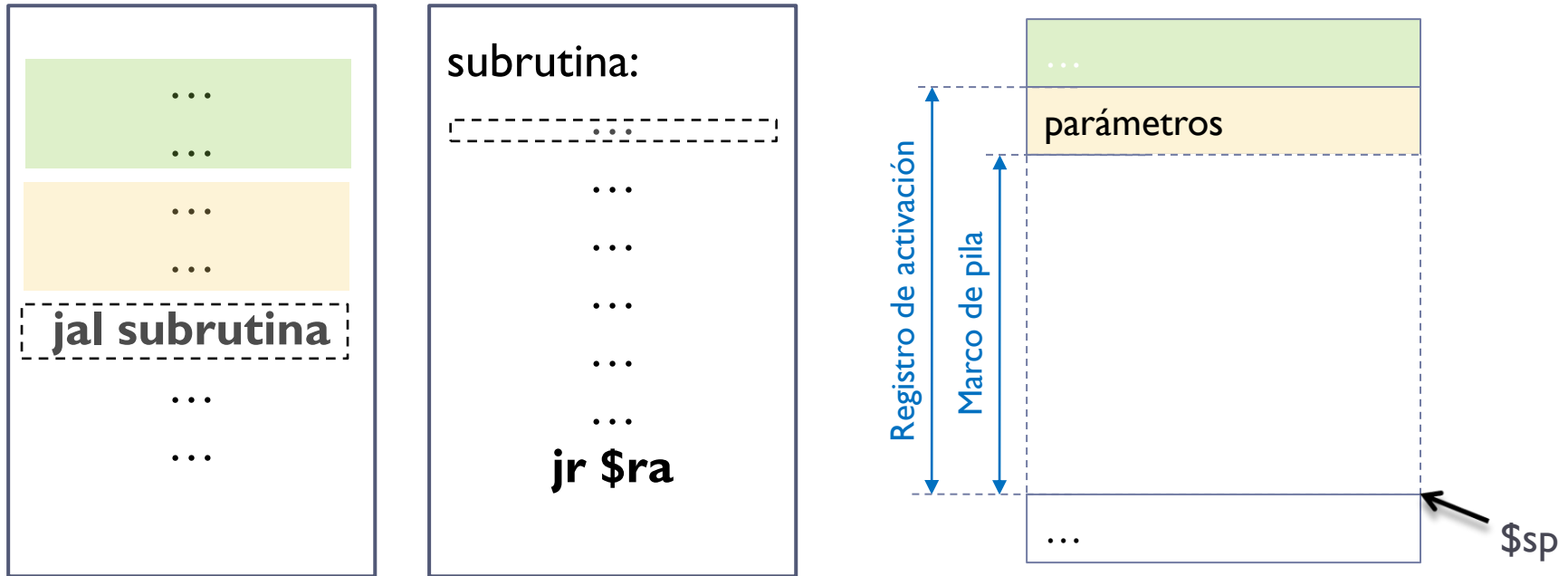
- Se utilizarán los registros generales **\$a0 .. \$a3** para los 4 primeros argumentos
- Se han de guardar los **\$a0 .. \$a3** en pila antes de modificarse
- Con más de 4 argumentos, deberá también usarse la pila para ellos
- Es parte del futuro **registro de activación** del llamado (**sección de parámetros**)

Marco de pila: llamante debe.... (3/3)



- ▶ Llamada a subrutina
 - ▶ Ejecutar la instrucción **jal subrutina**
 - ▶ Otras posibilidades:
 - ▶ **jal** etiqueta, **bal** etiqueta, **bltzal** \$reg, etiqueta, **bgezal** \$reg, etiqueta, **jalr** \$reg, **jalr** \$reg, \$reg

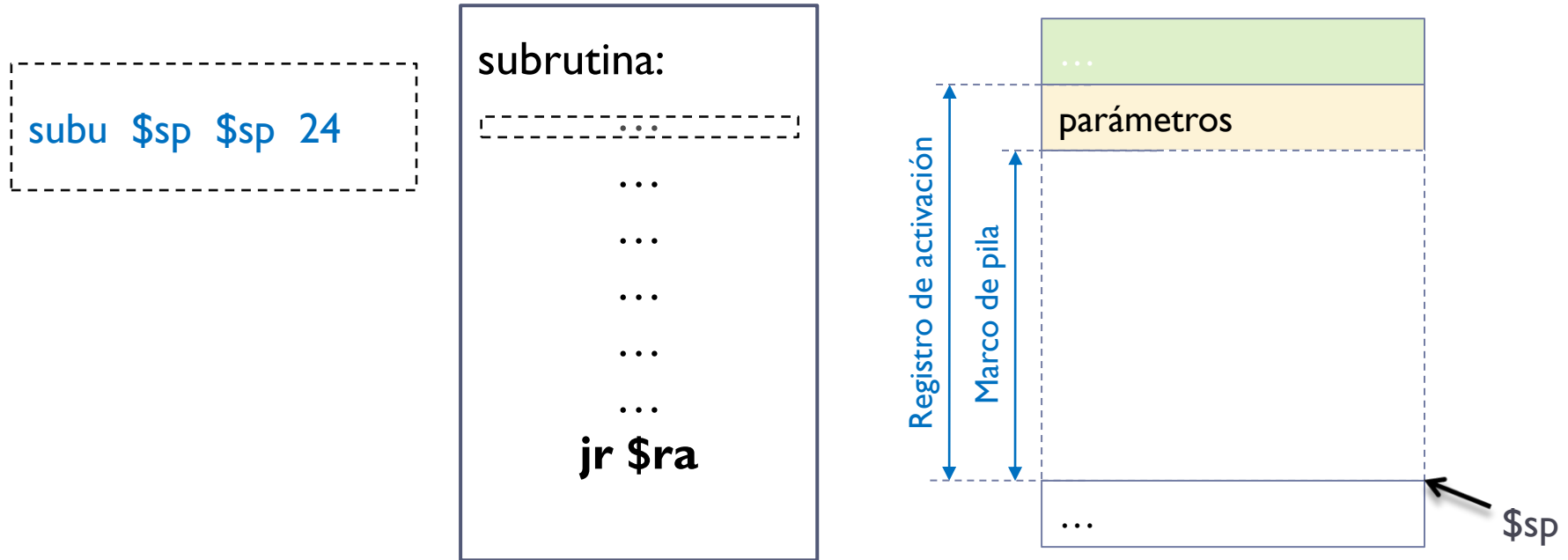
Marco de pila: llamado debe.... (1/4)



► Reserva del marco de pila

- $\$sp = \$sp - \langle \text{tamaño del marco de pila en bytes} \rangle$
- Se deberá reservar un espacio en la pila para almacenar los registros:
 - $\$ra$ y $\$fp$ si llama a otra rutina
 - $\$s0 \dots \$s9$ que se modifiquen dentro del llamado

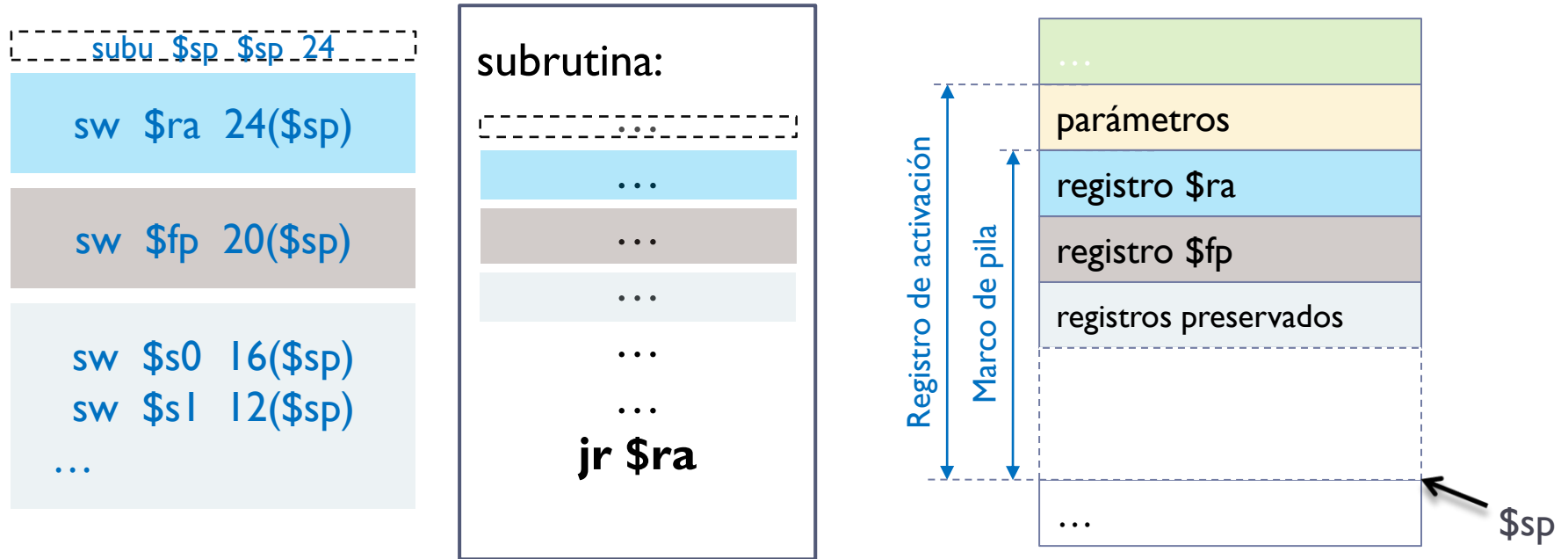
Marco de pila: llamado debe.... (1/4)



► Reserva del marco de pila

- $\$sp = \$sp - \text{<tamaño del marco de pila en bytes>}$
- Se deberá reservar un espacio en la pila para almacenar los registros:
 - `$ra` y `$fp` si llama a otra rutina
 - `$s0...$s9` que se modifiquen dentro del llamado

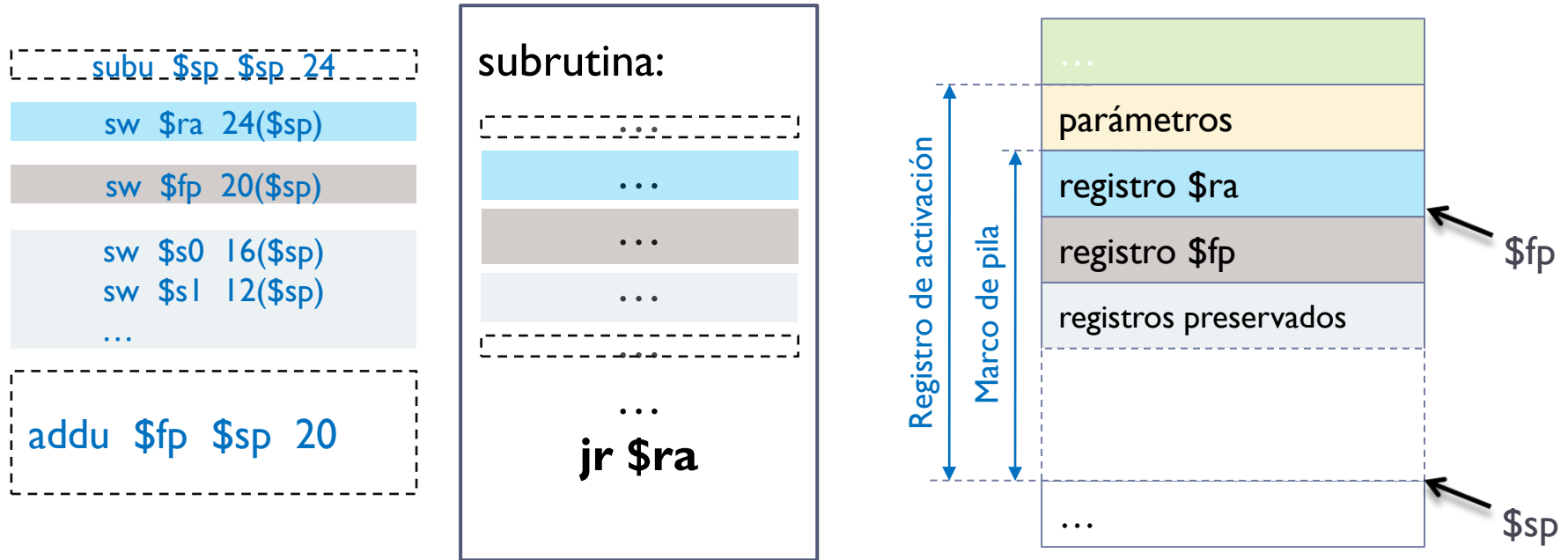
Marco de pila: llamado debe.... (2/4)



► Salvaguarda de registros

- Salvaguarda de `$ra` en la **región de retorno** (si llama a subrutina)
- Salvaguarda de `$fp` en la **región de marco de pila** (si llama a subrutina)
- Salvaguarda de los registros `$s0 .. $s9` en la **región de preservados**

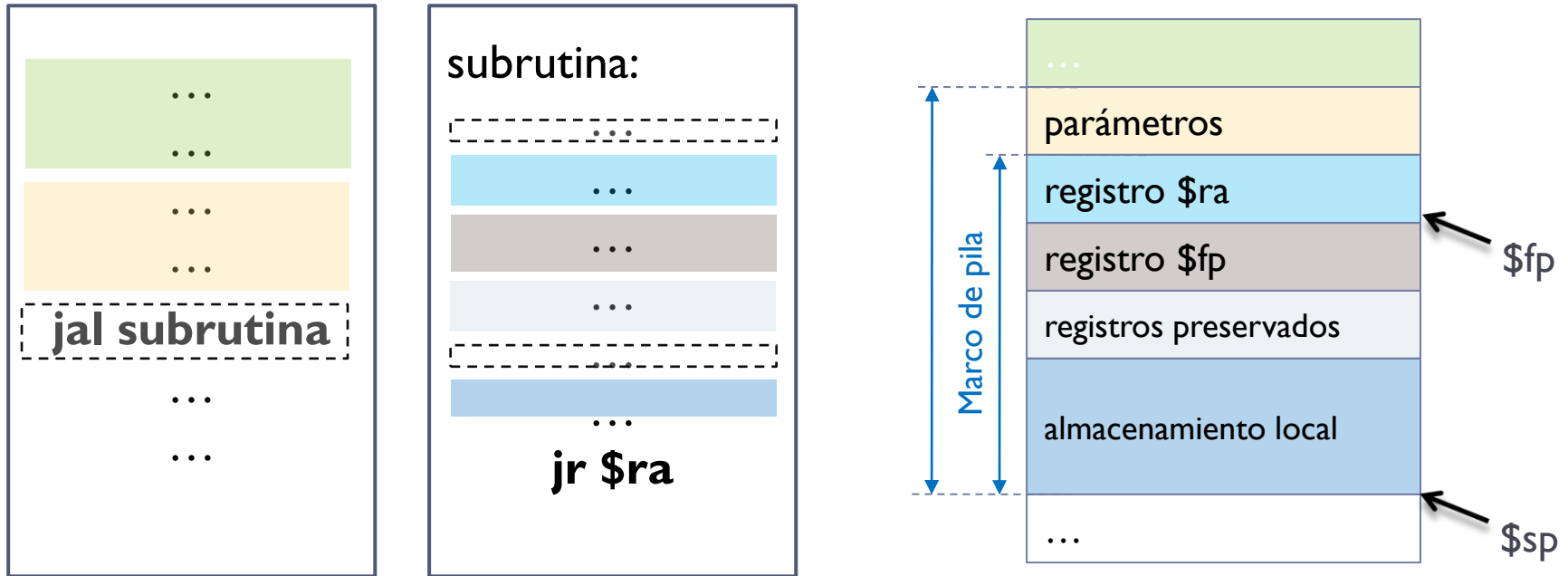
Marco de pila: llamado debe.... (3/4)



► Modificación de `$fp` (frame pointer)

- $\$fp = \$sp + \text{<tamaño del marco de pila en bytes>} - 4$
- `$fp` ha de apuntar al principio del marco

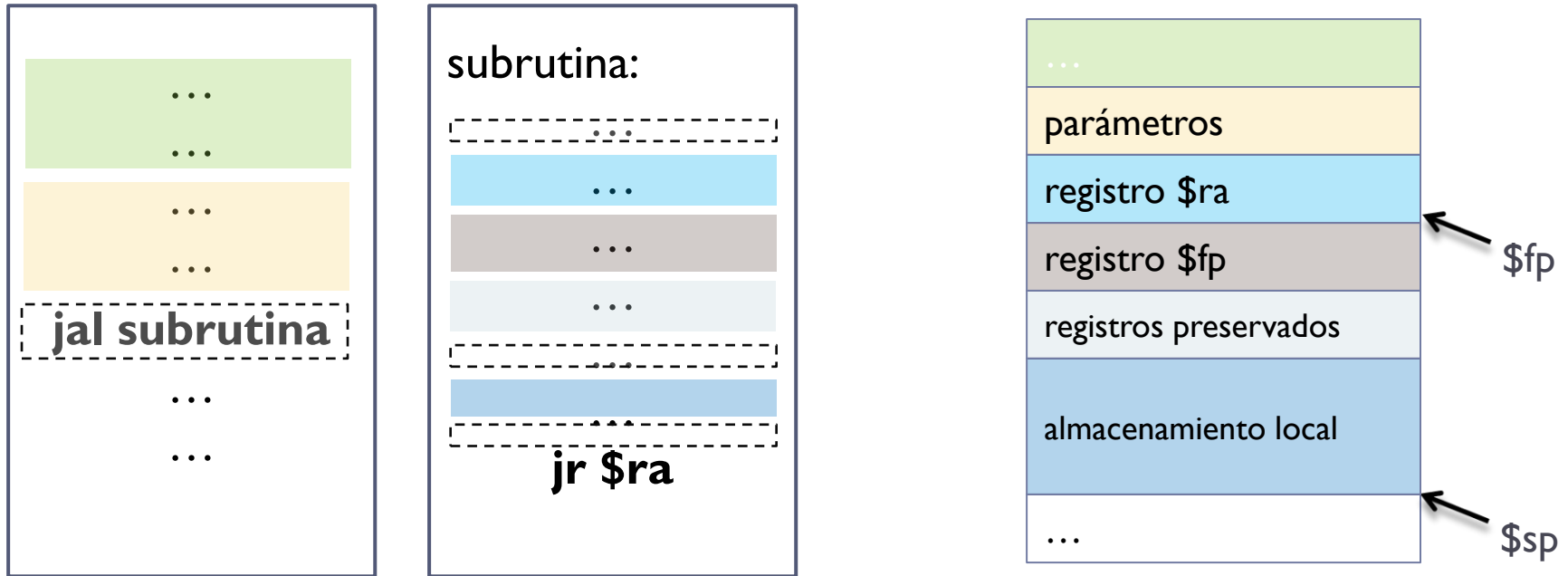
Marco de pila: llamado debe.... (4/4)



► Ejecución del cuerpo de la subrutina

- Ejecución de la subrutina
- Posible uso de **región de almacenamiento local**
 - Se guardarían los registros `$t0..$t9` si se llama a otra subrutina, se necesitan más registros, etc.

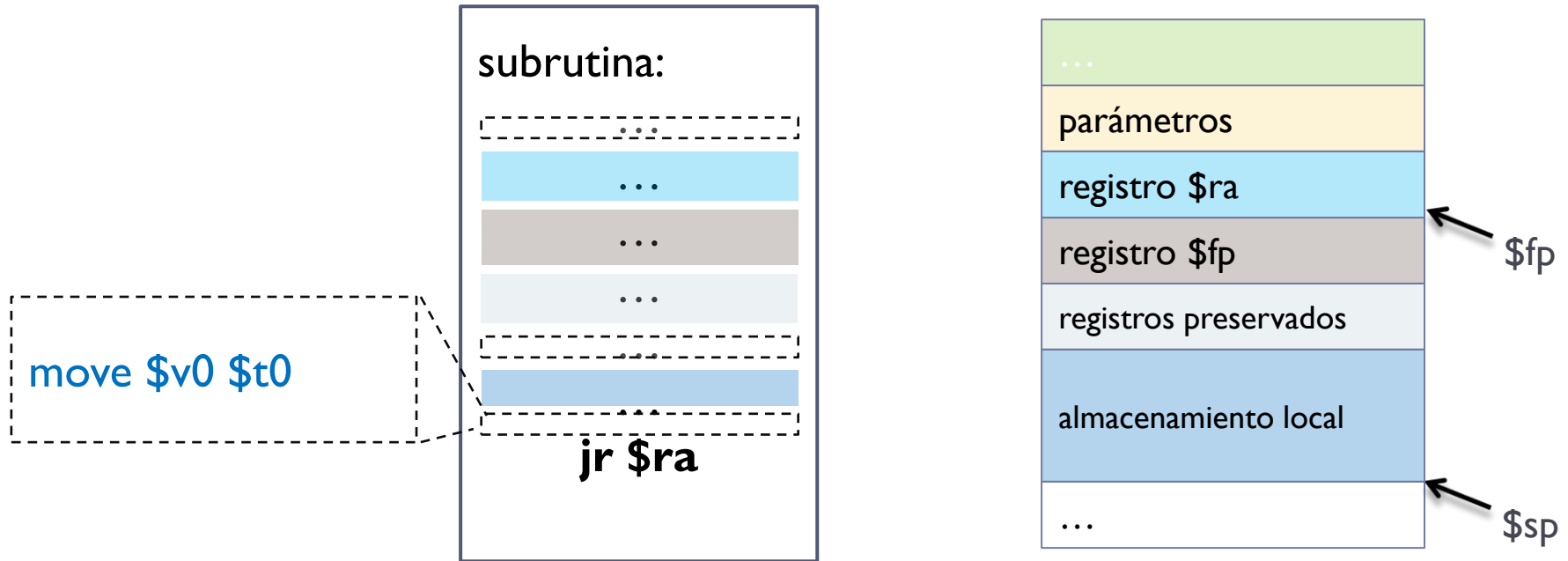
Marco de pila: llamado debe.... (1/2)



► Finalización de la subrutina

- Si hay resultados que dar al llamante, devolución de los argumentos de salida en **\$v0** y **\$v1** (o **\$f0** y **\$f2**)

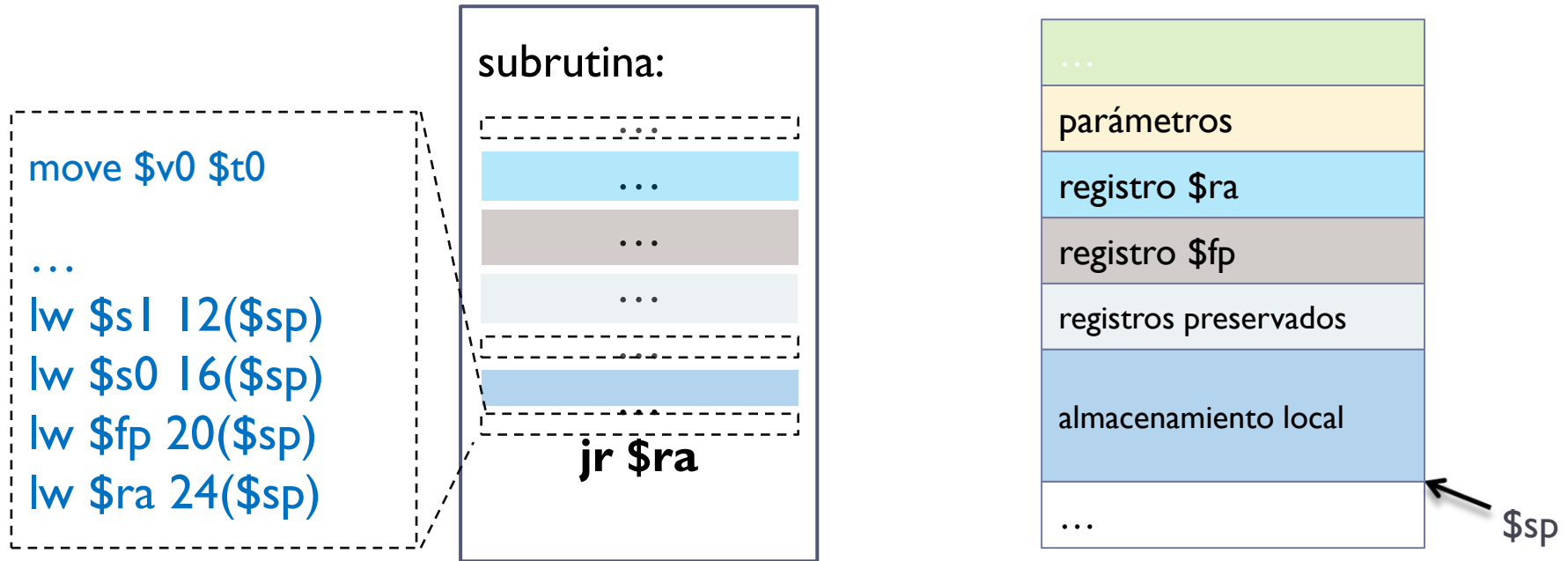
Marco de pila: llamado debe.... (1/2)



► Finalización de la subrutina

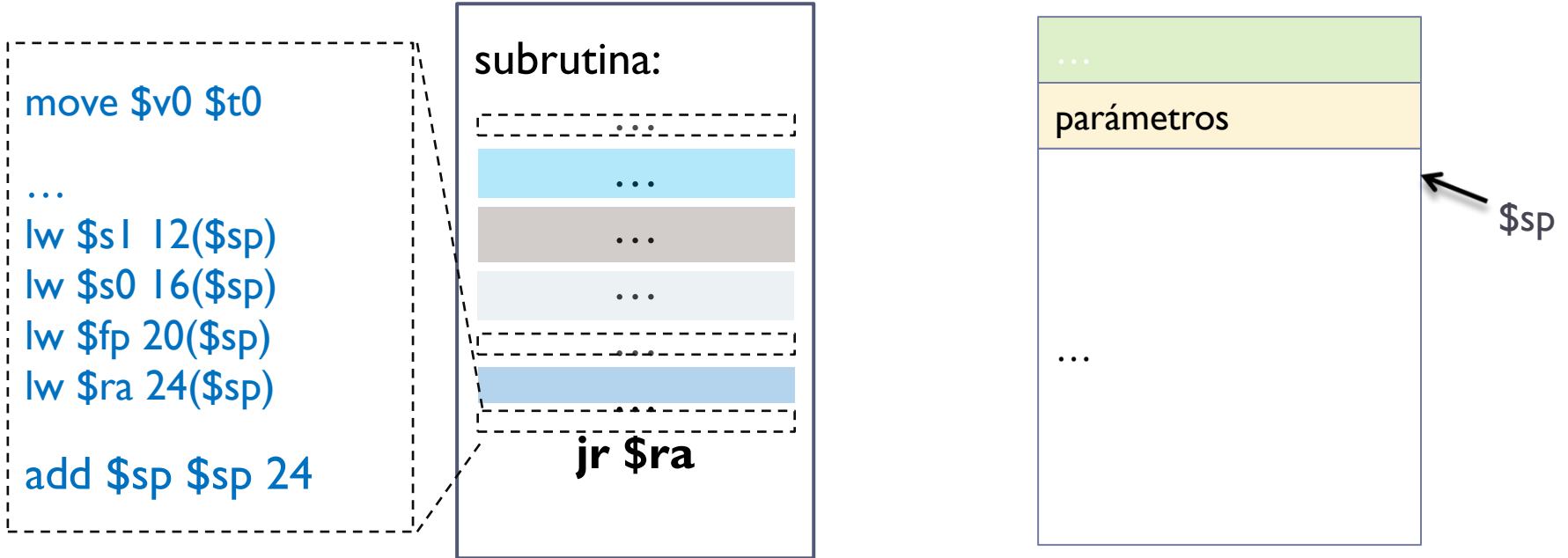
- Si hay resultados que dar al llamante, devolución de los argumentos de salida en \$v0 y \$v1 (o \$f0 y \$f2)

Marco de pila: llamado debe.... (1/2)



- Finalización de la subrutina
 - Restaurar registros preservados + `$fp` + `$ra`

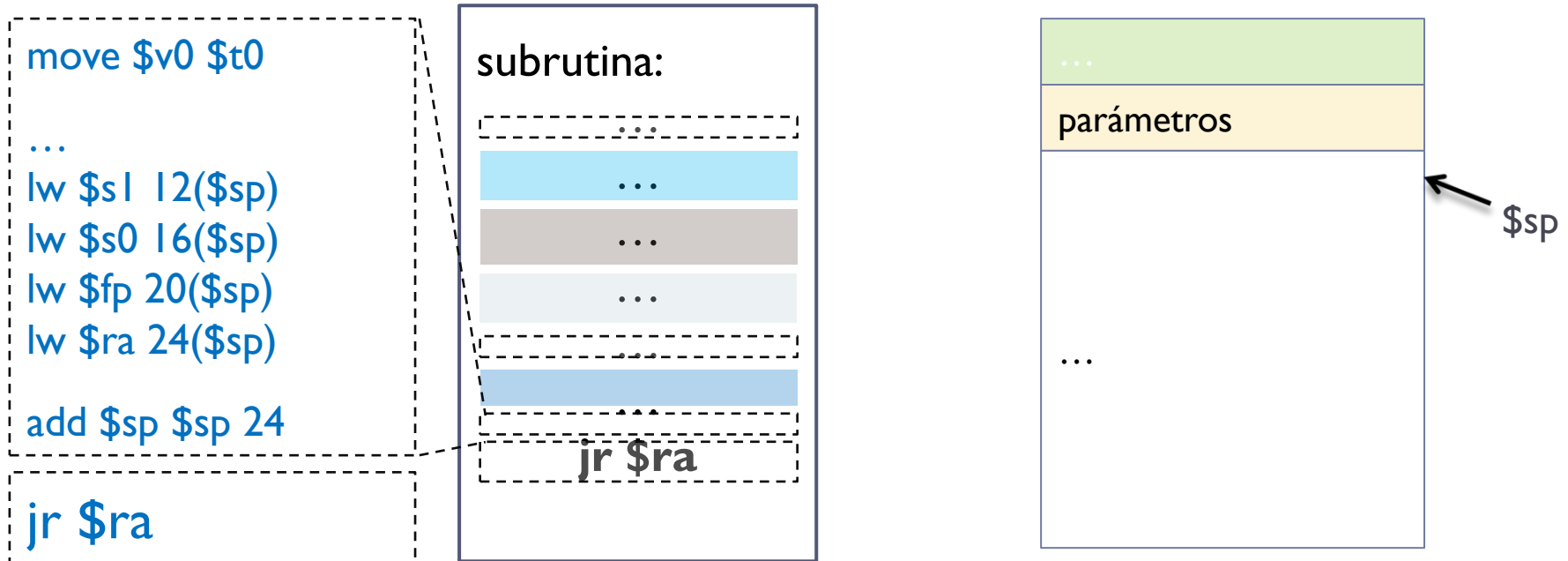
Marco de pila: llamado debe... (1/2)



► Finalización de la subrutina

- ▶ Liberar el espacio usado por el marco de pila:
 - ▶ $\$sp = \$sp + \text{tamaño del marco de pila}$

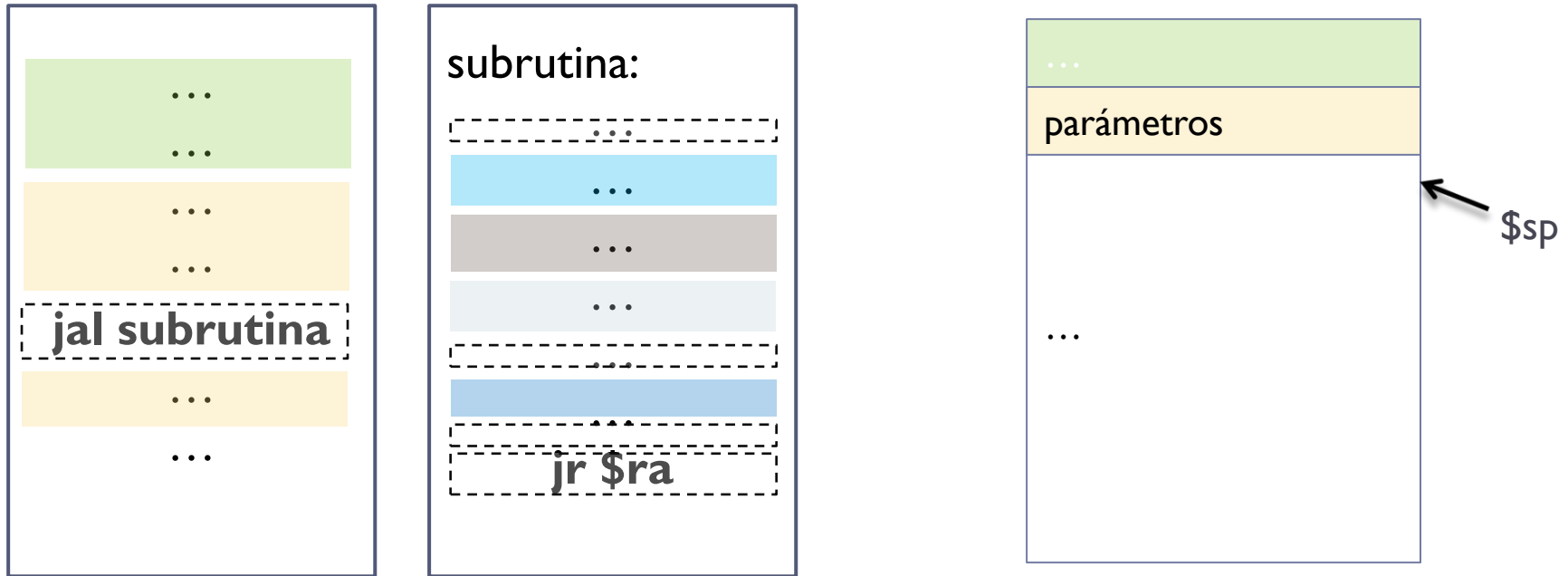
Marco de pila: llamado debe.... (2/2)



► Vuelta al llamante

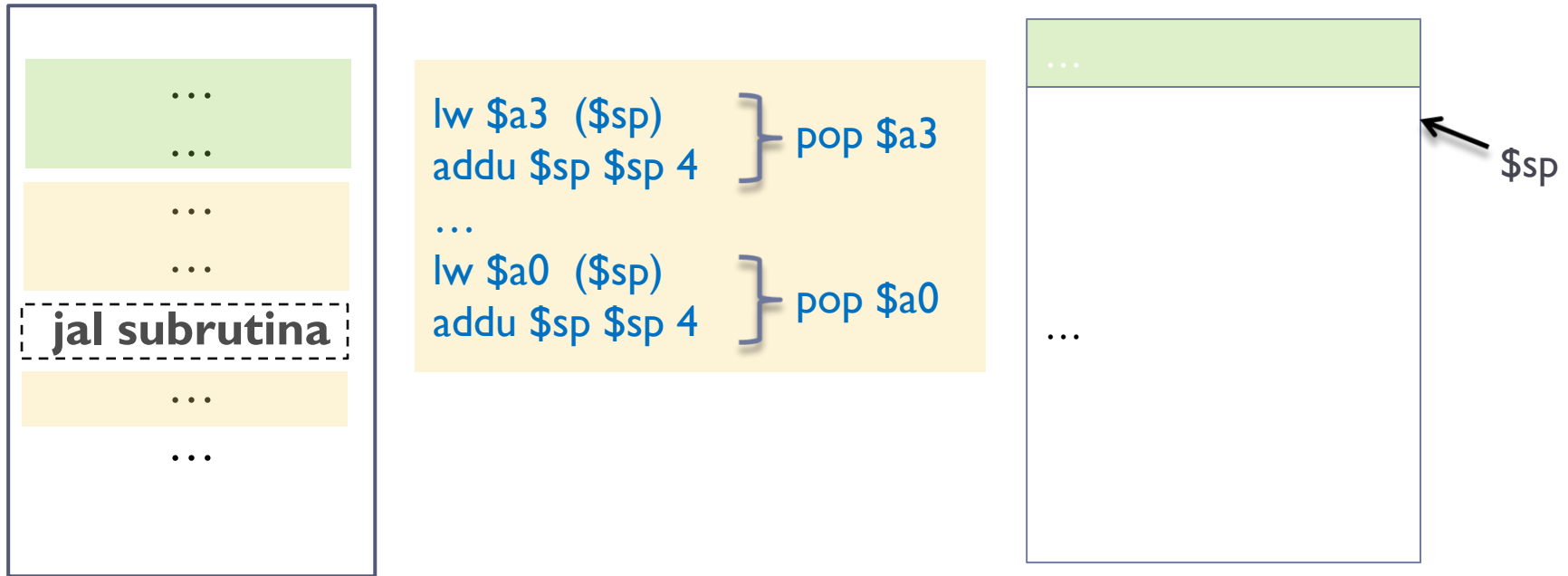
- jr \$ra

Marco de pila: llamante debe.... (1/2)



- ▶ **Quitar parámetros de pila**
 - ▶ Restaurar los registro `$a0 .. $a3` salvaguardados previamente
 - ▶ Es lo último que finaliza el registro de activación del llamado

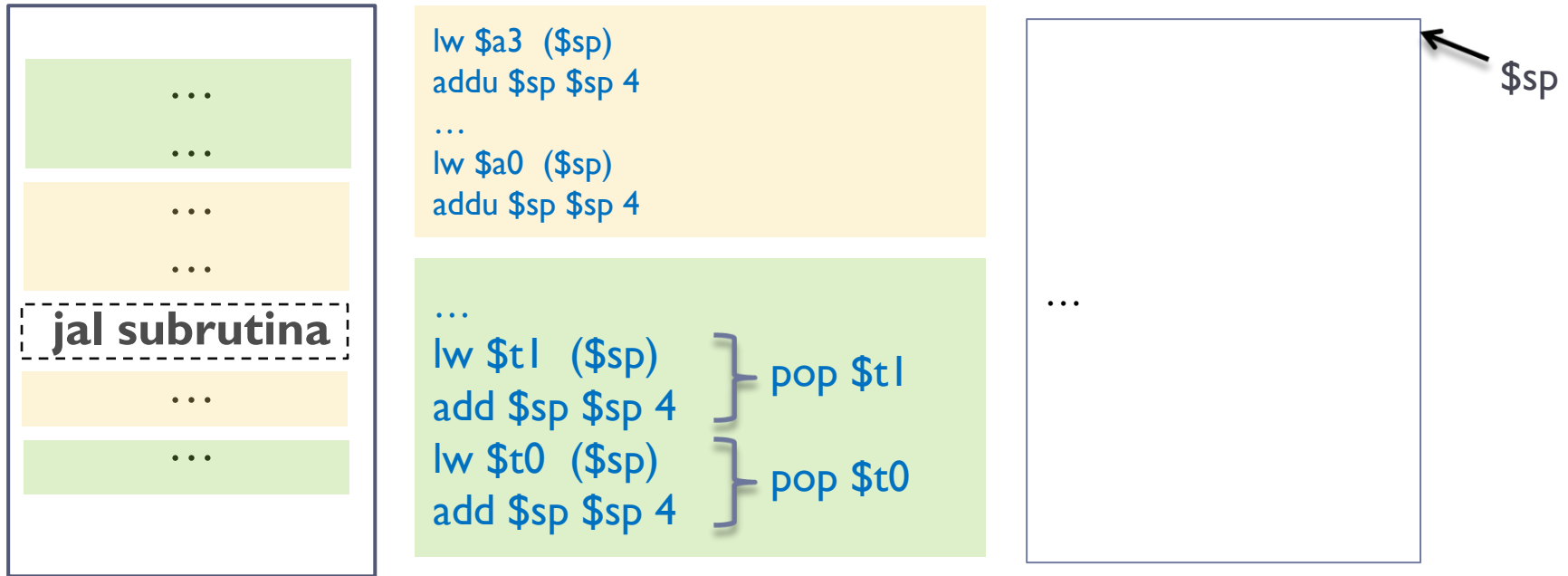
Marco de pila: llamante debe.... (1/2)



► Quitar parámetros de pila

- Restaurar los registro `$a0 .. $a3` salvaguardados previamente
- Es lo último que finaliza el registro de activación del llamado

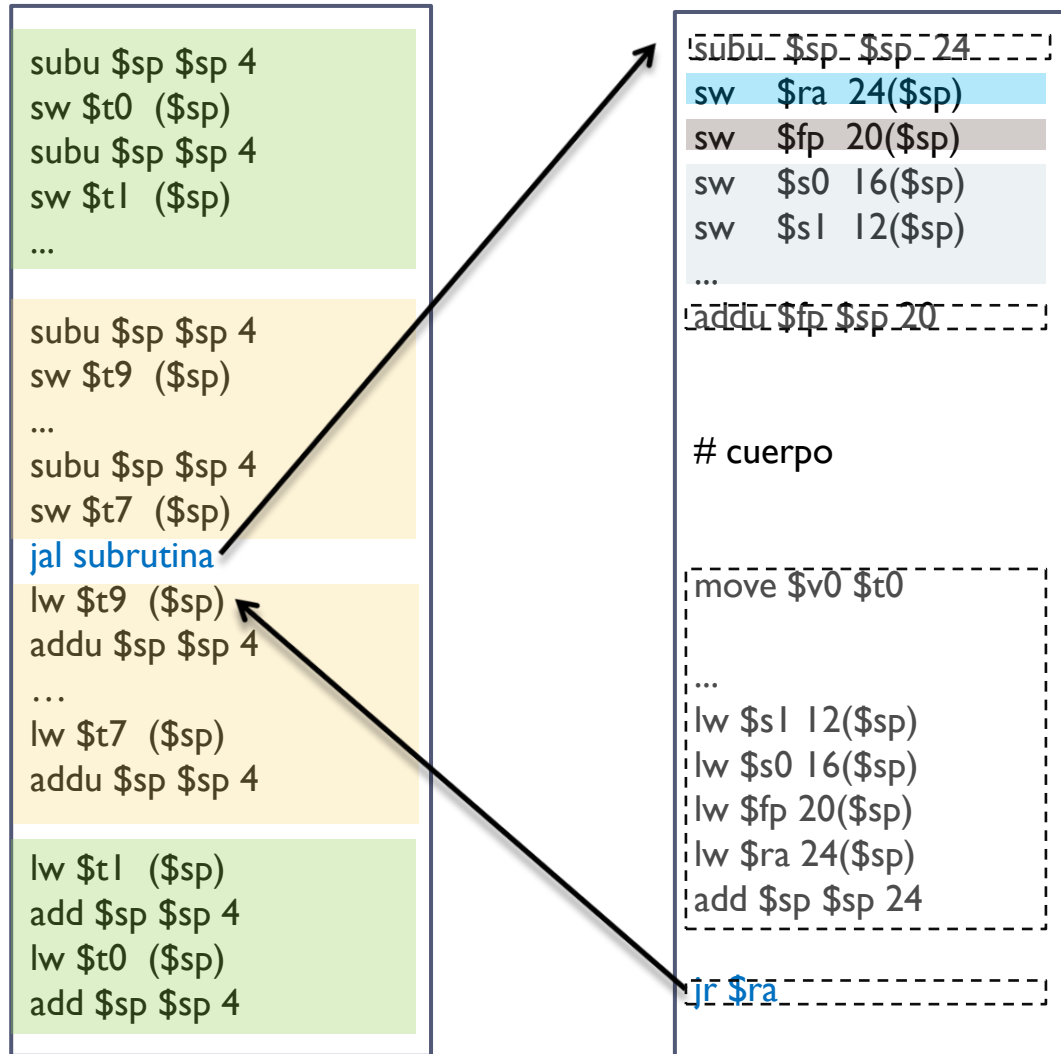
Marco de pila: llamante debe.... (2/2)



► Restaurar registros de pila

- Restaurar los registro `$t0..$t9` salvaguardados previamente
- Recordar que se guardan en zona del registro de activación del llamante

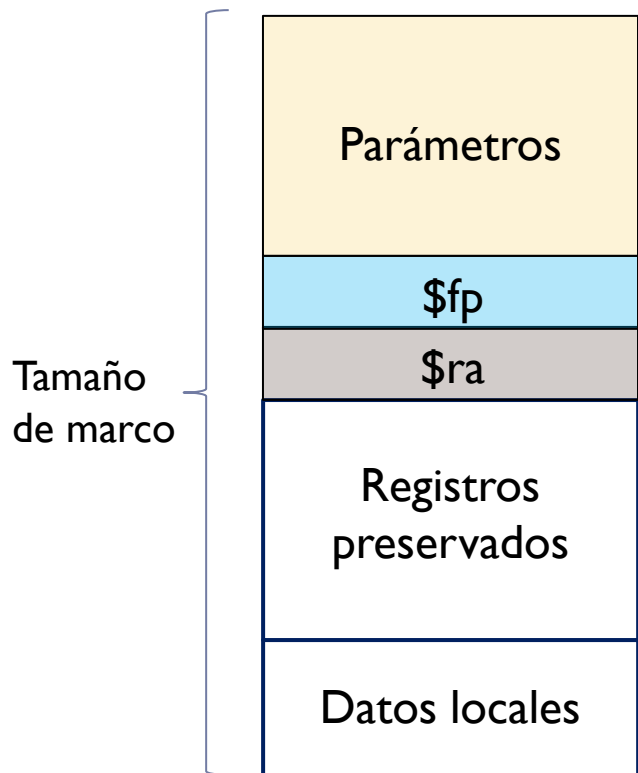
Marco de pila: resumen



Marco de pila: resumen

Subrutina llamante	Subrutina llamada
Salvaguarda de registros que no quiera que modifique la subrutina llamada	
Paso de parámetros	
Llamada a subrutina	
	Reserva del marco de pila
	Salvaguarda de registros
	Ejecución de subrutina
	Restauración de valores guardados
	Liberación de marco de pila
	Salida de subrutina
Restauración de registros guardados	

Marco de pila (convenio)



▶ Parámetros

- ▶ Si no terminal
- ▶ 4 palabras < Tamaño >= tamaño de los parámetros de la subrutina llamable con más parámetros
- ▶ Alineado a límite de palabra
- ▶ Se preservan parámetros (todos o del 4º en →)

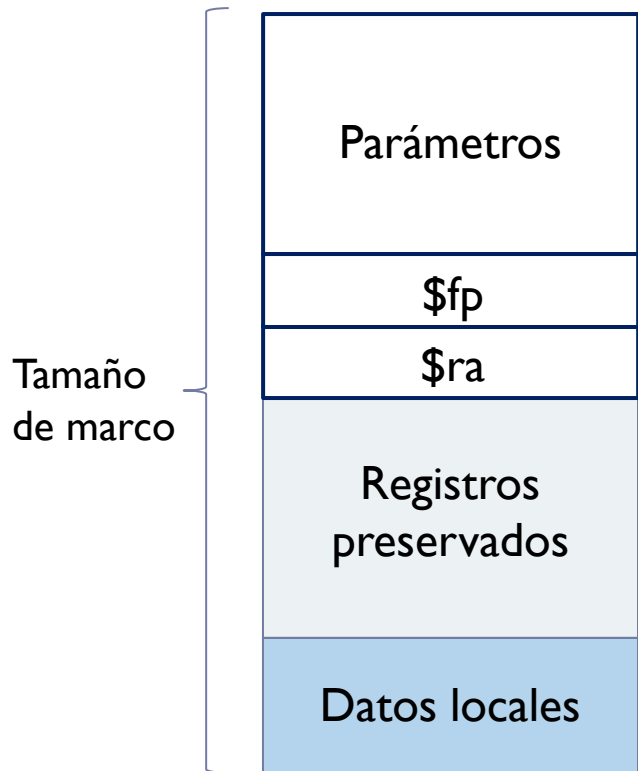
▶ Registro de retorno

- ▶ Si no terminal
- ▶ Tamaño == 1 palabra
- ▶ Alineado a límite de palabra
- ▶ Se guarda \$ra

▶ Registro de marco de pila

- ▶ Si no terminal
- ▶ Tamaño == 1 palabra
- ▶ Alineado a límite de palabra
- ▶ Se guarda \$fp

Marco de pila (convenio)



▶ Sección de registros preservados

- ▶ Si se necesita
- ▶ Tamaño == **numero de registros \$s*** a preservar
- ▶ Alineado a límite de **palabra**
- ▶ Se preservan los **registros \$s***

▶ Sección de relleno

- ▶ **Existe si** el número de bytes de la sección de parámetros, registros preservados, registro de retorno y registro de marco de pila **no** es **múltiplo** de **8 bytes**
- ▶ Tamaño == **1 palabra**

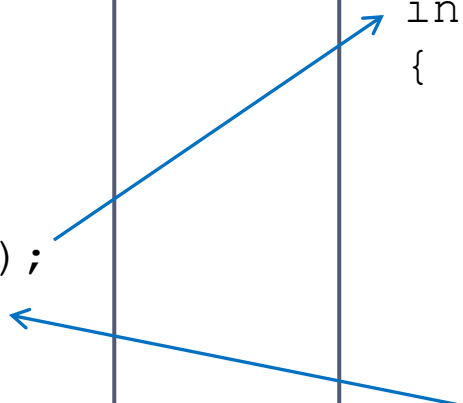
▶ Sección de almacenamiento local

- ▶ Si se necesita
- ▶ Tamaño == **numero de registros \$t*** a preservar
- ▶ Alineado a límite de **doble palabra**
- ▶ Se preservan los **registros \$t***

Ejemplo

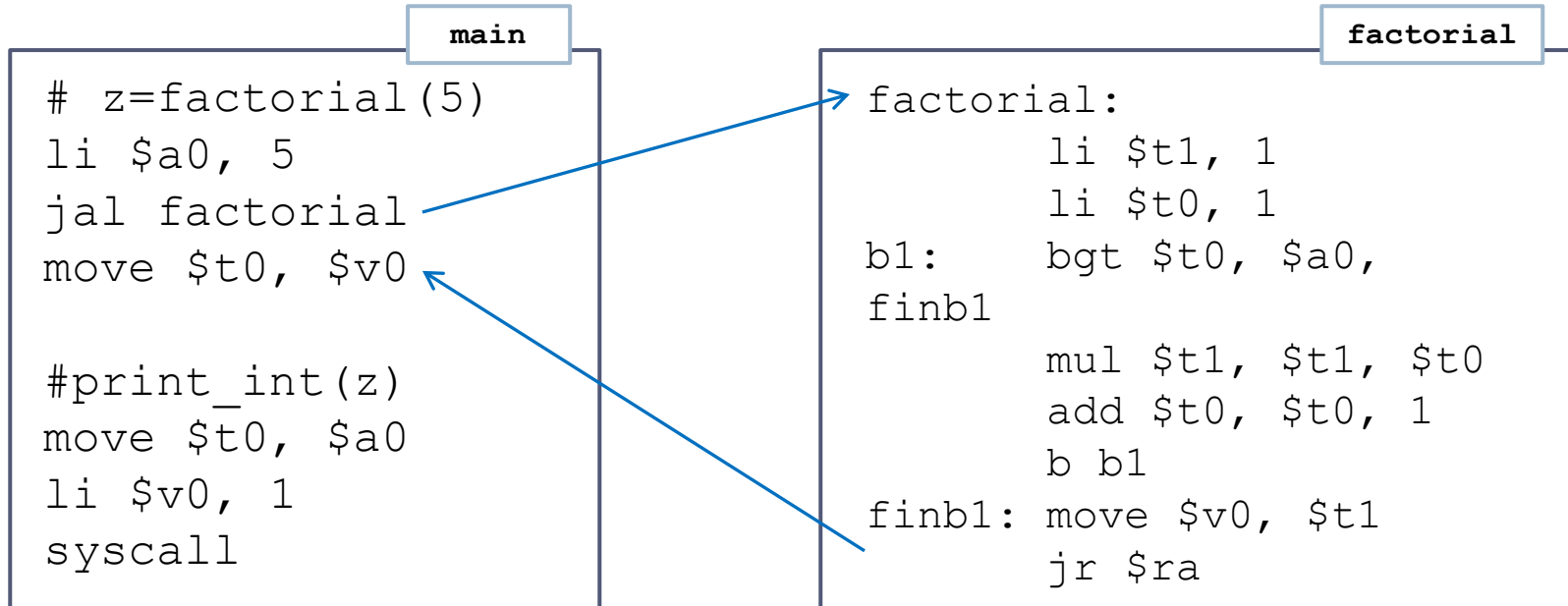
```
int main()
{
    int z;
    z=factorial(x);
    print_int(z);
}
```

```
int factorial(int x)
{
    int i;
    int r=1;
    for (i=1;i<=x;i++) {
        r*=i;
    }
    return r;
}
```



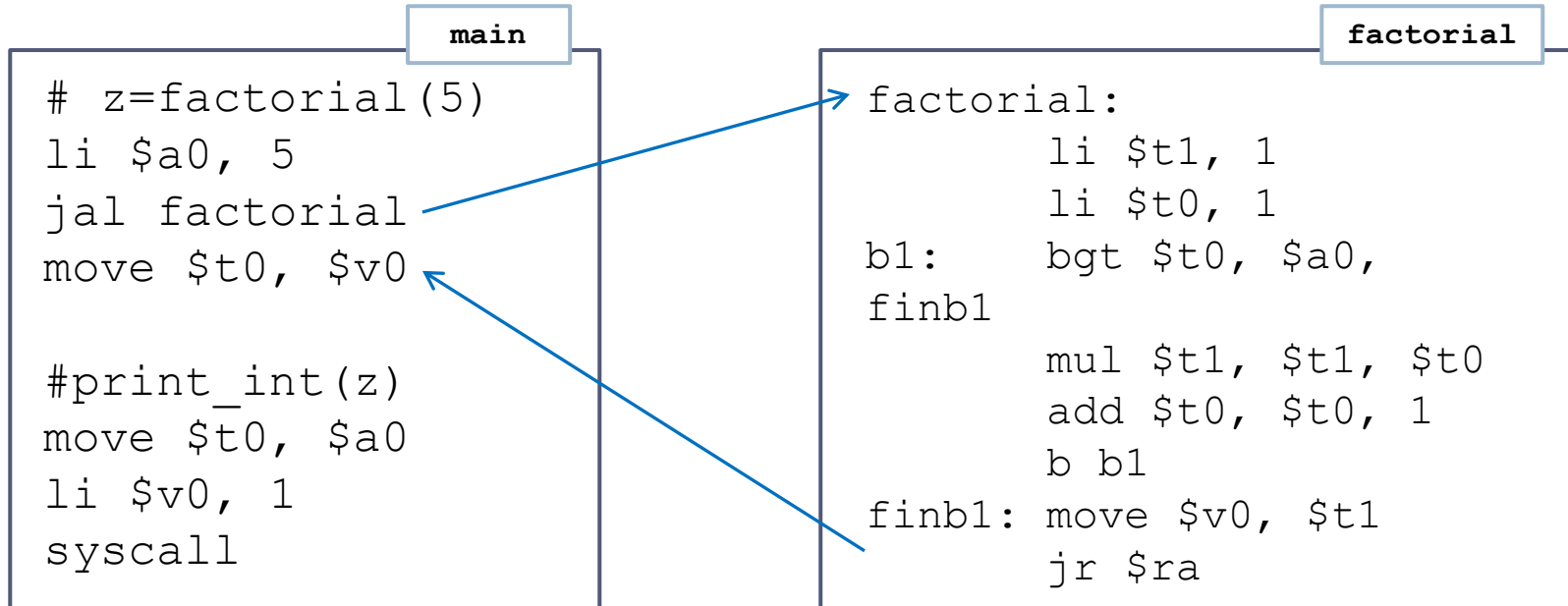
- Codificar la llamada de una función de alto nivel en ensamblador

Ejemplo



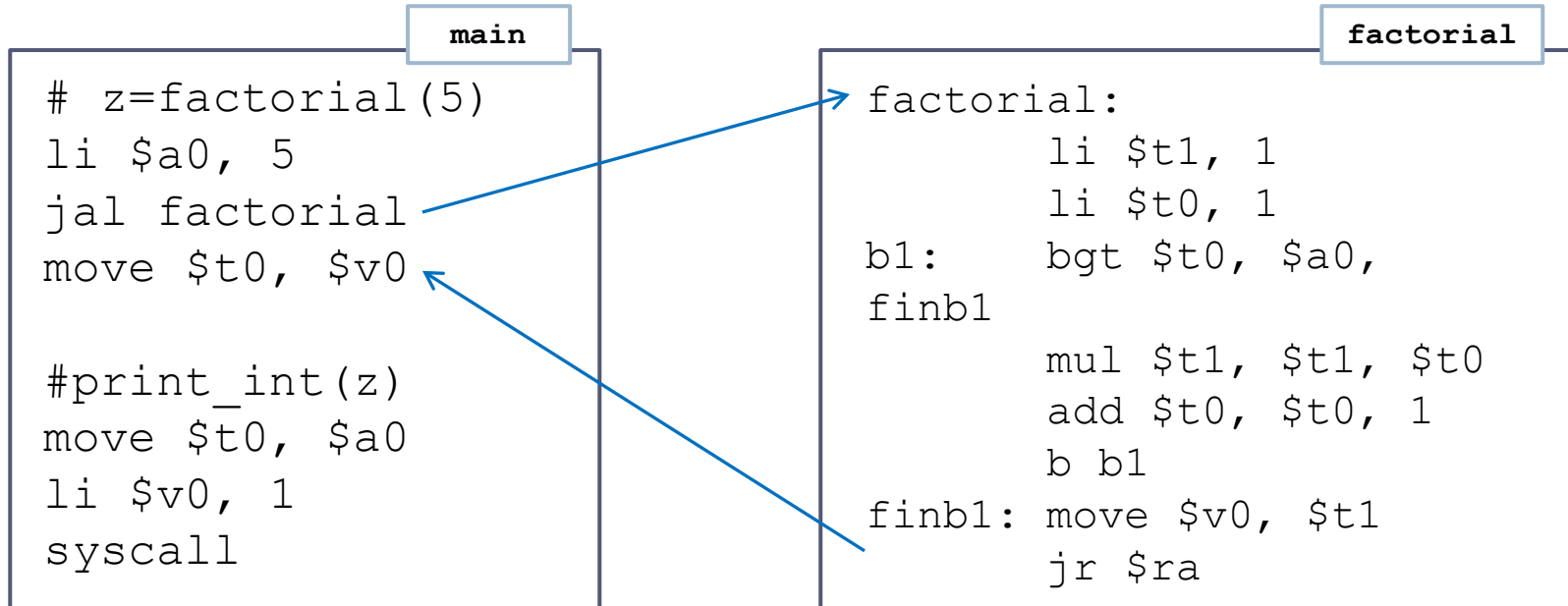
- Se codifican ambas funciones en ensamblador...

Ejemplo



- Se analizan ambas rutinas...

Ejemplo



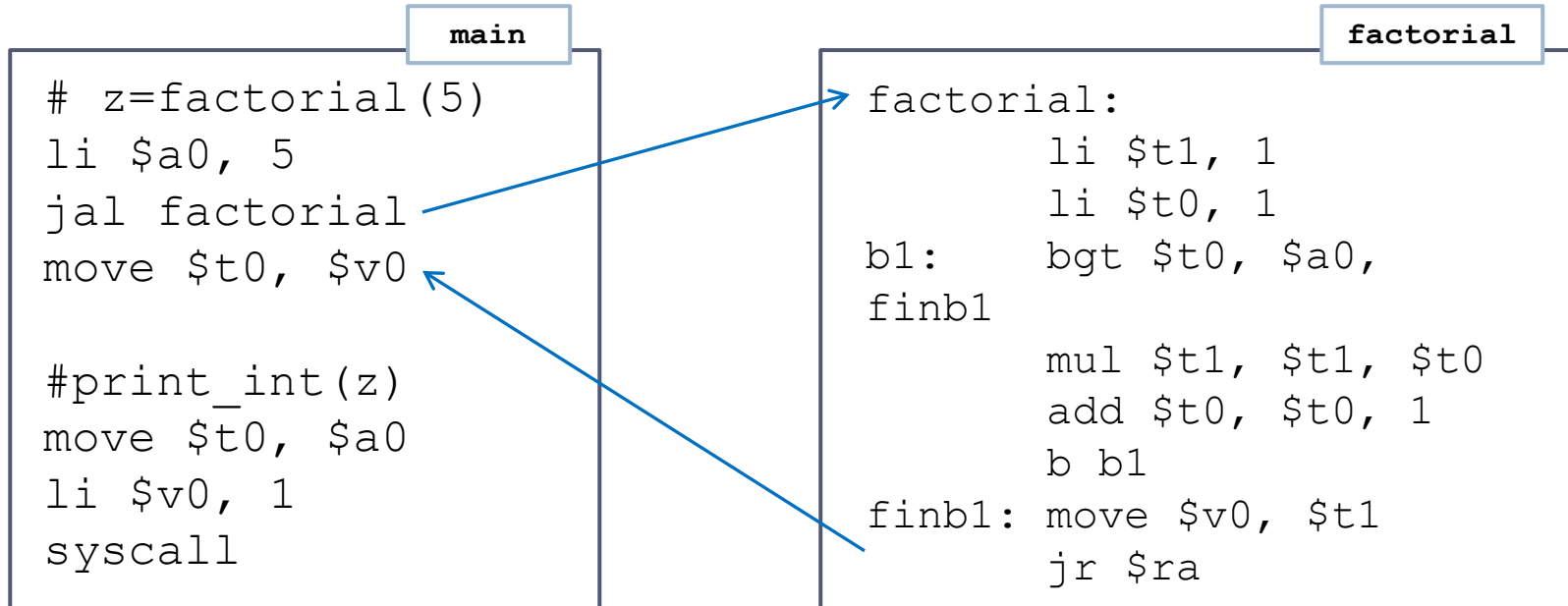
► Rutina no terminal (llama a otra)

- Sección de parámetros: 4
- Sección de registro de retorno: 1
- Sección de registro de marco de pila: 1
- Sección registros preservados: 0
- Sección de almacenamiento local: 0

► Rutina terminal

- Sección de parámetros: 0
- Sección de registro de retorno: 0
- Sección de registro de marco de pila: 0
- Sección registros preservados: 0
- Sección de almacenamiento local: 0

Ejemplo



- Se añade el prólogo y epílogo a las dos subrutinas (main y factorial) según lo calculado...

Ejemplo

main

```
sub $sp, $sp, 24
sw $fp, 24($sp)
sw $ra, 20($sp)
sw $a0, 4($sp)
add $fp, $sp, 20
```

```
# z=factorial(5)
li $a0, 5
jal factorial
move $t0, $v0
```

```
#print_int(z)
move $t0, $a0
li $v0, 1
syscall
```

```
lw $fp, 24($sp)
lw $ra, 20($sp)
lw $a0, 4($sp)
add $sp, $sp, 24
```

factorial

factorial:

```
li $t1, 1
li $t0, 1
b1: bgt $t0, $a0,
finb1
mul $t1, $t1, $t0
add $t0, $t0, 1
b b1
finb1: move $v0, $t1
jr $ra
```




Tema 3 (III)

Fundamentos de la programación en ensamblador



Grupo ARCOS

Estructura de Computadores
Grado en Ingeniería Informática
Universidad Carlos III de Madrid