



Tema 2: Instrucciones. Lenguaje del computador

Unidad 2: Lenguaje ensamblador

Rafael Casado González
Rosa María García Muñoz
María Teresa López Bonal
Universidad de Castilla–La Mancha



DEPARTAMENTO
DE SISTEMAS
INFORMÁTICOS



Instituto de Investigación
en Informática de Albacete

Estructura multinivel

■ Software del Sistema

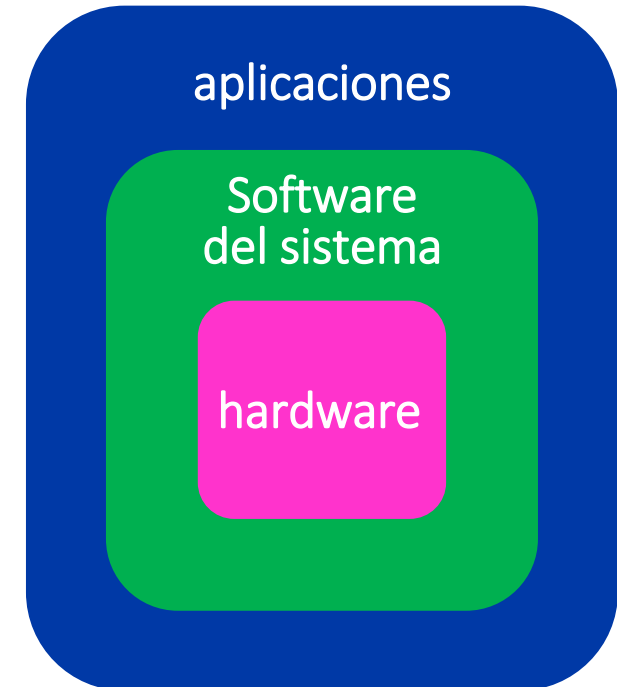
□ Sistemas operativos

(Windows, Linux, MacOS)

- Actúan como interfaz entre el hardware y los programas de usuario

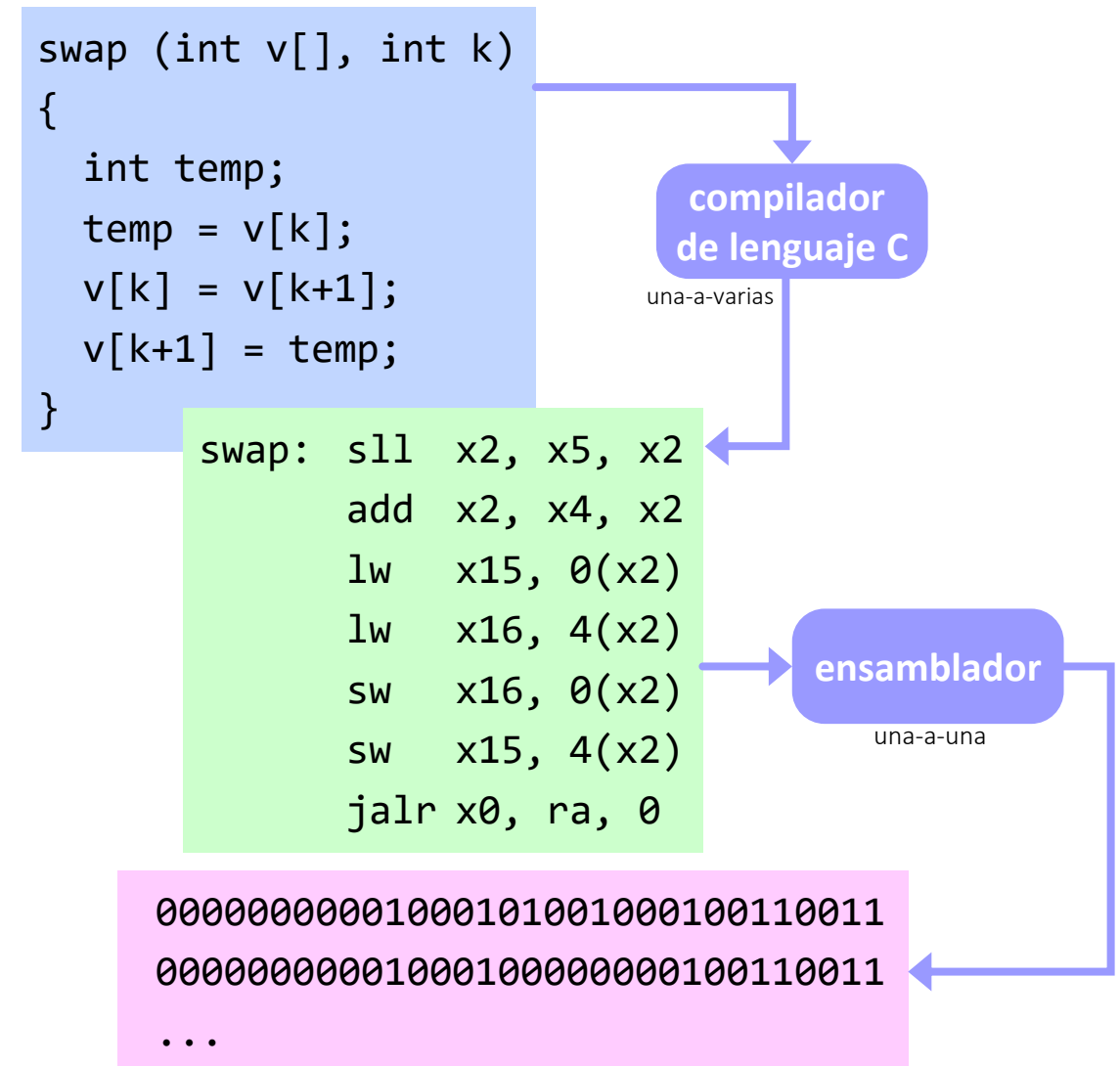
□ Compiladores

- Traducen programas escritos en lenguaje de alto nivel (C, Java,...) en instrucciones que el hardware pueda ejecutar



Estructura multinivel

- Programa en lenguaje de alto nivel (C)
- Programa en lenguaje ensamblador (para RISC-V)
- Código máquina (objeto, binario) (para RISC-V)



Lenguaje ensamblador

Lenguaje máquina

- Es un lenguaje muy restringido y de bajo nivel
- Está formado por instrucciones máquina
 - Representadas por ceros y unos
- Es el único lenguaje que entiende directamente el computador
 - El lenguaje máquina depende del procesador
 - Existe una incompatibilidad innata entre los distintos procesadores

Lenguaje ensamblador

Inconvenientes del lenguaje máquina

- Las instrucciones máquina son un conjunto de bits usualmente representado en binario o hexadecimal
 - La redacción de programas es compleja
 - El resultado es muy poco legible
- Además, el programador ha de realizar manualmente la asignación de memoria
 - Decidir en qué posiciones coloca el código y los datos
 - Calcular las direcciones de salto
- Para solucionar estos inconvenientes, se utiliza el lenguaje ensamblador

Lenguaje ensamblador

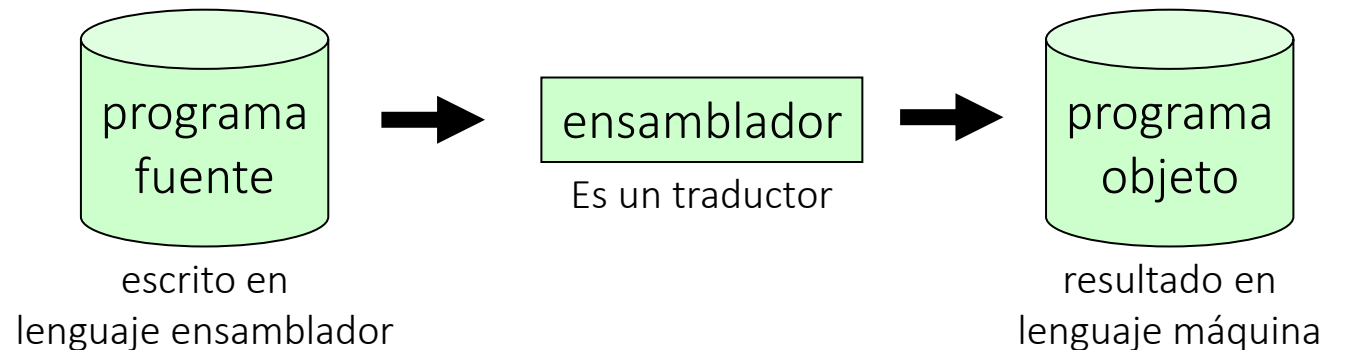
Lenguaje ensamblador

- Es un lenguaje de bajo nivel que facilita la tarea de programar en lenguaje máquina
- Permite escribir las instrucciones máquina en notación simbólica o nemotécnica
 - En vez de código hexadecimal o binario
- Permite utilizar direcciones de memoria simbólicas
 - En lugar de direcciones binarias absolutas
- Insertar comentarios

Lenguaje ensamblador

Programa ensamblador

- Es un traductor
 - En particular, un compilador
- Lee un programa fuente
 - Escrito en lenguaje ensamblador
- y lo traduce a un programa objeto
 - En lenguaje máquina



Lenguaje ensamblador

Programa ensamblador

- El desarrollo de los ensambladores fue un logro importante en la evolución de la tecnología de los computadores
 - Primer paso hacia los lenguajes de alto nivel
- Los lenguajes de alto nivel simplifican muchísimo la tarea del programador
- A cambio, el lenguaje ensamblador tiene acceso a todos los recursos e instrucciones de la máquina
 - Registros internos de la computadora
 - Flags de estado
 - Mapa de direcciones

Lenguaje ensamblador

Sentencias

- Las sentencias pueden ser

- Instrucciones

- Representaciones simbólicas de las instrucciones máquina, es decir, los nemotécnicos

- Pseudo-Instrucciones

- Alias de casos muy habituales de uso de ciertas instrucciones
 - Durante el ensamblado son traducidas a instrucciones reales de comportamiento equivalente
 - Permiten enriquecer el lenguaje sin añadir complejidad al HW

- Directivas

- Órdenes que se le dan al propio programa ensamblador
 - Parametrizan el proceso de traducción a lenguaje máquina

Lenguaje ensamblador

Sentencias

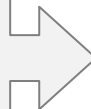
- Deben ocupar una línea del programa fuente
- Se descomponen en cuatro campos

`<sentencia> ::= [etiqueta:] <código> [operandos] [#comentario]`

Lenguaje ensamblador


Ejemplo

etiqueta:
Nombran la
siguiente
posición de
memoria.
Van al principio
de línea




```
#segmento de datos
.data
num1: .word 13           # defino un dato tamaño word
num2: .word 0x1500
resul: .word 0
# segmento de texto
.text
.globl main

main:
    la    t1, num1
    lw    t2, 0(t1)
    la    t3, num2
    lw    t4, 0(t3)
    add   t5, t2, t4
    addi  t5, t5, -7
    la    t6, resul
    sw    t5, 0(t6)
    addi  a7, zero, 10    # llamada al sistema operativo
    ecall                                # para salir del programa
```



comentarios
Comentario que se
extiende hasta el final
de la línea.
El ensamblador los
ignora.

Cada línea
contiene como
máximo una
sentencia



Lenguaje ensamblador

Ejemplo

.data

Sección de datos (opcional)
Contiene la declaración de variables necesarias. Se guardan en el segmento de datos de la memoria.

.text

Sección de texto (obligatoria)
Contiene las instrucciones. Se guardan en el segmento de texto de la memoria.

#segmento de datos

.data

```
num1:  .word 13
num2:  .word 0x1500
resul: .word 0
```

segmento de texto

.text

.globl main

main:

```
la    t1, num1
lw    t2, 0(t1)
la    t3, num2
lw    t4, 0(t3)
add   t5, t2, t4
addi  t5, t5, -7
la    t6, resul
sw    t5, 0(t6)
addi  a7, zero, 10
ecall
```

Lenguaje ensamblador

Sección .data

- Parte del programa dedicada a la declaración de variables inicializadas en tiempo de compilación

.data

etiqueta_nombre_var: <tipo> <valores> [#comentario]

■ Tipos

☐ .byte .half .word

☐ .asciz .string

☐ .zero

☐ .float .double

Lenguaje ensamblador

Sección .data

■ .byte

```
datos: .byte byte1, byte2, ...
```

- Almacena bytes de forma secuencial en la memoria de la máquina

■ .half

```
datos: .half halfword1, halfword2, ...
```

- Almacena medias palabras de forma secuencial en la memoria de la máquina

■ .word

```
datos: .word palabra1, palabra2, ...
```

- Almacena palabras de forma secuencial en la memoria de la máquina

Lenguaje ensamblador

Sección .data

■ .string / .asciz

```
texto: .string "cadena1", "cadena2", ...
```

```
texto: .asciz "cadena1", "cadena2", ...
```

- ☐ Almacena las cadenas como una lista de caracteres
- ☐ Cadenas terminadas implícitamente por el byte NULL

■ .zero

```
valores: .zero tamaño
```

- ☐ Pone a cero los bytes indicados en "tamaño"

Lenguaje ensamblador

Sección .data

■ .float

`datos: .float numero1, numero2, ...`

- Almacena la secuencia de números en la memoria
- Representados en coma flotante de simple precisión

■ .double

`datos: .double numero1, numero2, ...`

- Almacena la secuencia de números en la memoria
- Representados en coma flotante de doble precisión

Lenguaje ensamblador

Sección .bss (Block Started by Symbol)

- Parte del programa dedicada a la declaración de variables no inicializadas en tiempo de compilación

.bss

etiqueta_nombre_var: <tipo> <valores> [#comentario]

- Los valores se ponen solo para indicar cuantos son

Lenguaje ensamblador

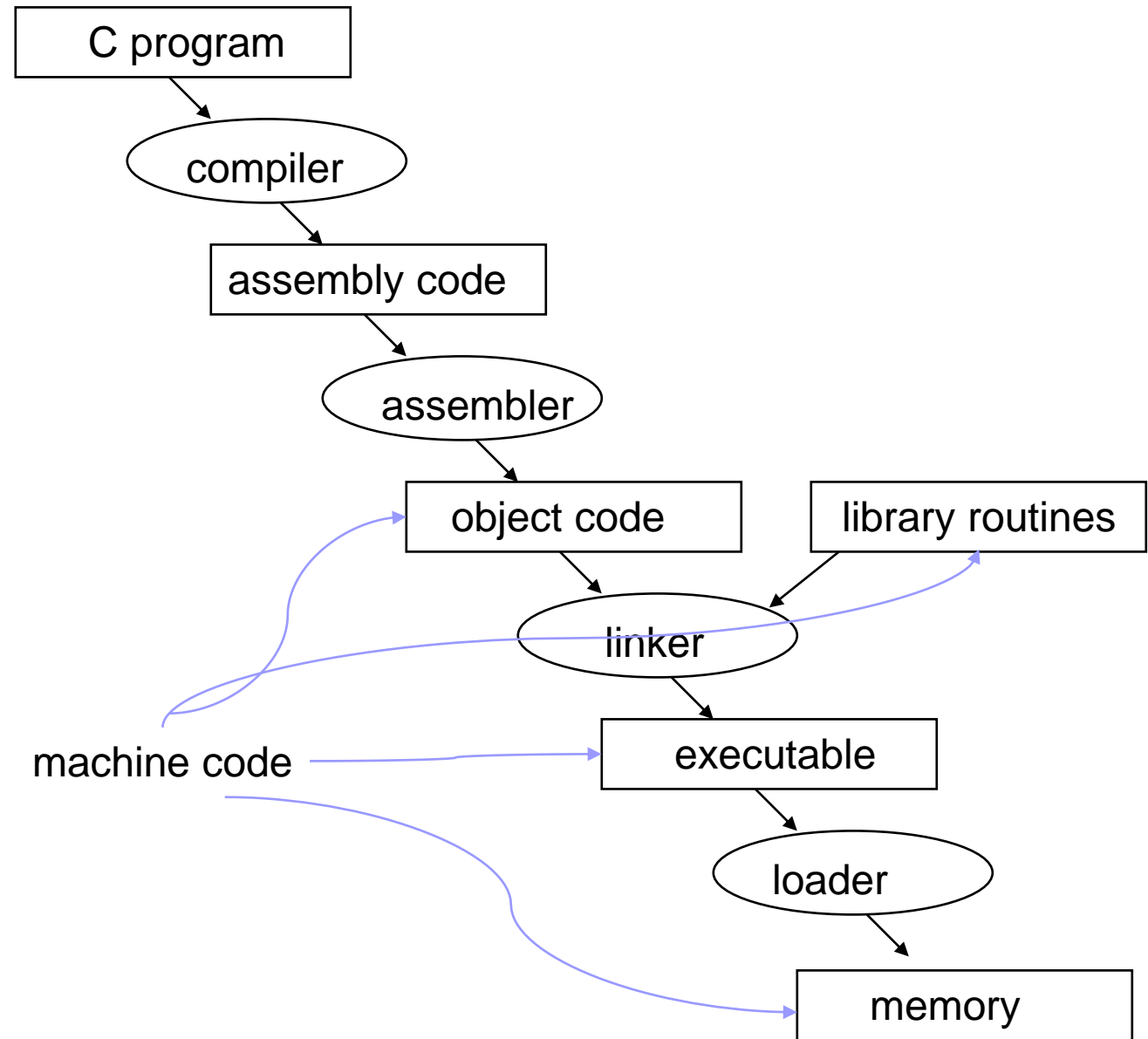
Sección .text

- Parte del programa dedicada al código ejecutable

.text

Lenguaje ensamblador

- Proceso completo de creación y ejecución de un programa



Lenguaje ensamblador

Programa enlazador: linker

- Un módulo puede tener referencias a subrutinas y datos definidos en otros módulos y librerías
 - El código de un módulo no puede ejecutarse si contiene referencias sin resolver
- Un programa enlazador (o linker)
 - Combina diversos módulos objeto en un único módulo ejecutable
 - Controla que las direcciones globales sean correctas

Lenguaje ensamblador

Programa cargador: loader

- La función del programa cargador es
 - Leer un fichero de un medio de almacenamiento externo
 - Cargarlo en la memoria principal
 - Proceder a su ejecución
 - Asignando al contador de programa la dirección de inicio del programa cargado

Procedimientos

Pasos de invocación y retorno

- El programa principal (caller) prepara los argumentos de llamada
- El programa principal transfiere el control al procedimiento
- El procedimiento (callee) adquiere los recursos necesarios
- El procedimiento realiza la tarea asignada
- El procedimiento almacena los resultados
- El procedimiento retorna el control al programa principal

Procedimientos

Ejemplo

subrutina que duplica el valor de las palabras de una zona de memoria dada
a0:
puntero al inicio de la zona de memoria
a1:
contador de palabras a duplicar
t0, t1:
otros registros afectados

```
.data
dirmem: .word 2,3,8,55,0xf,25,26
tam:    .word 3

.text
.globl main

main:
    la a0, dirmem
    la t0, tam
    lw a1, 0(t0)
    jal ra,duplicar

    addi a7,zero, 10
    ecall

duplicar:
    lw t0, 0(a0)
    add t1, t0,t0
    sw t1, 0(a0)
    addi a0, a0, 4
    addi a1, a1, -1
    bne a1, zero, duplicar
    jalr x0,ra,0
```

Prepara los argumentos de llamada a subrutina

llamada a subrutina

pc+4 → ra

retorno de subrutina

pc ← ra

Procedimientos

Criterio de uso de registros generales

Alias	Nombre	Descripción	¿Se espera que preserve su valor tras la ejecución de la subrutina?
zero	x0	cableado a 0x00000000	
ra	x1	dirección de retorno de subrutina	sí
sp	x2	puntero de pila (stack pointer)	sí
gp	x3	puntero global	sí
tp	x4	puntero de hebra (task pointer)	sí
t0-t2	x5-x7	registros temporales	no
s0/fp	x8	registro preservado / puntero de marco	sí
s1	x9	registro preservado	sí
a0-a1	x10-x11	argumentos / retorno de valores	no
a2-a7	x12-x17	argumentos	no
s2-s11	x18-x27	registro preservado	sí
t3-t6	x28-x31	registros temporales	no

Procedimientos

Paso de argumentos por pila

- ¿Y si el procedimiento invoca a su vez a otros procedimientos?
 - O se invoca a sí mismo recursivamente
- La dirección de retorno se gestiona en una pila (stack)
 - Donde el último elemento en entrar será el primero en salir

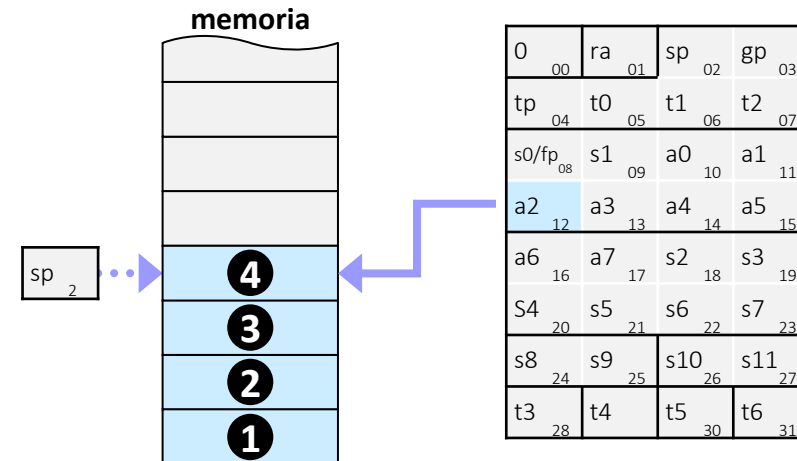
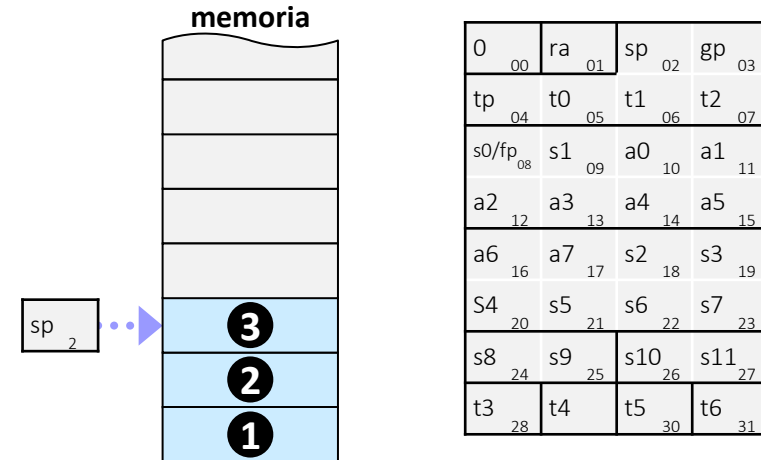
Procedimientos

Paso de argumentos por pila

■ Operación PUSH

- Introduce un elemento en la pila

```
addi sp, sp, -4  
sw    a2, 0(sp)
```



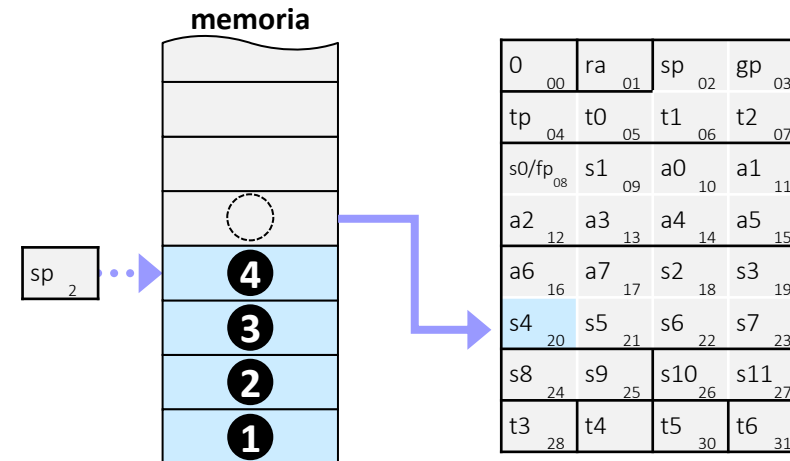
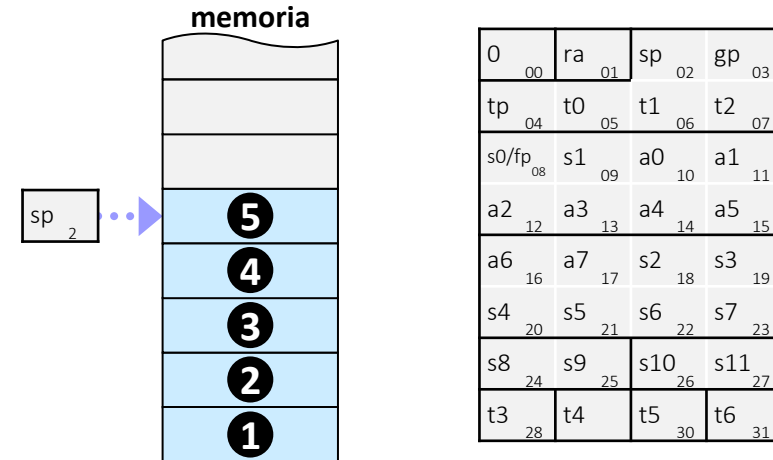
Procedimientos

Paso de argumentos por pila

■ Operación POP

- Extrae un elemento de la pila

```
lw s4, 0(sp)
addi sp, sp, +4
```



Procedimientos

Paso de argumentos por pila

- ¿Y si el procedimiento define más argumentos o resultados que los registros disponibles para ello?
- También se utiliza la pila
 - La pila mezcla argumentos de entrada / salida y direcciones de retorno

Procedimientos

Ejemplo

subrutina que duplica el valor de las palabras de una zona de memoria dada
a0:
puntero al inicio de la zona de memoria
pila:
contador de palabras a duplicar
t0, t1, t2:
otros registros afectados

```
.data
dirmem: .word 2,3,8,55,0xf,25,26

.text
.globl main
main:
    la    a0,dirmem
    addi  t6, zero, 5
    addi  sp, sp, -4
    sw    t6, 0(sp)

    jal   ra, duplicar

    addi  a7, zero, 10
    ecall

duplicar:
    lw    t0, 0(sp)
    addi  sp, sp, 4

bucle:
    lw    t1, 0(a0)
    add   t2, t1, t1
    sw    t2, 0(a0)
    addi  a0, a0, 4
    addi  t0, t0, -1
    bne   t0, zero, bucle

    jalr  zero, ra, 0
```

