

Subrutinas Procedimientos y Funciones

Material Elaborado por el Profesor Ricardo González
A partir de Materiales de las Profesoras
Angela Di Serio
María Blanca Ibañez

1

Como se pueden programar las aplicaciones

Programa
Monolítico

A
B
C
D
E
F
C
D
E
H
I
J
C
D
E
K
L

Programa
Principal

A
B
F
H
I
J
K
L

SubPrograma
Sub Rutina
Procedimiento

C
D
E

Llamado
Called

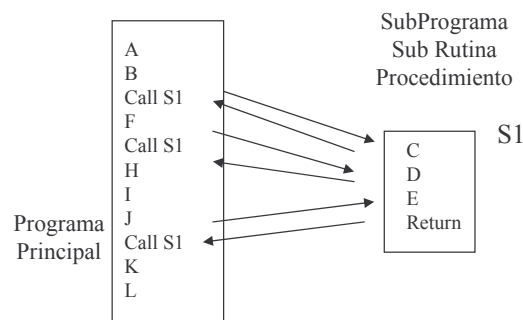
Llamador
Caller

2

Manejo de Subrutinas

¿Por qué son necesarias las subrutinas?

- Permiten reusar código.
- Simplifican la escritura de programas.
- Hacen que sea mas fácil modificar programas.
- Facilitan el trabajo en equipo.



3

Manejo de Subrutinas

```
main(){  
  ...  
  x = max(i,j);  
  ...  
}  
  
int max(int m, int n){  
  int mayor;  
  if (m > n)  
    mayor = m;  
  else  
    mayor = n;  
  return mayor;  
}
```

Diagrama de anotaciones:

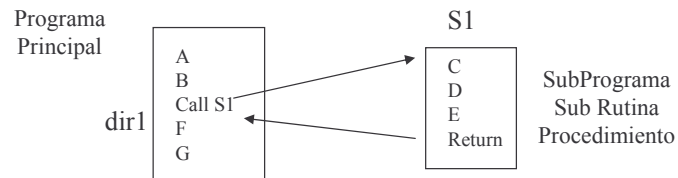
- Llamada a la subrutina: `x = max(i,j);`
- Parámetros actuales: `i, j`
- Encabezado de la subrutina: `int max(int m, int n){`
- Parámetros formales: `m, n`
- Variable local de la subrutina: `int mayor;`
- Cuerpo de la subrutina: `if (m > n) mayor = m; else mayor = n; return mayor;`

4

¿Qué ocurre cuando una subrutina sin parámetros se ejecuta?

Se debe:

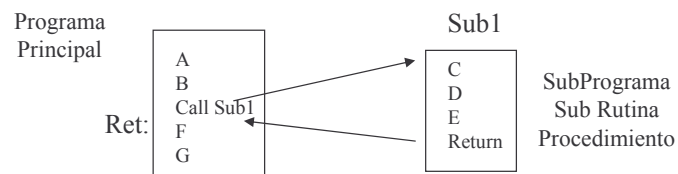
- Salvar la dirección de retorno.
- Saltar a la dirección donde comienza el código de la subrutina.
- Ejecutar la subrutina.
- Recuperar la dirección de retorno.
- Continuar la ejecución a partir de la dirección de retorno



5

Procedamos sin apoyo del lenguaje ensamblador

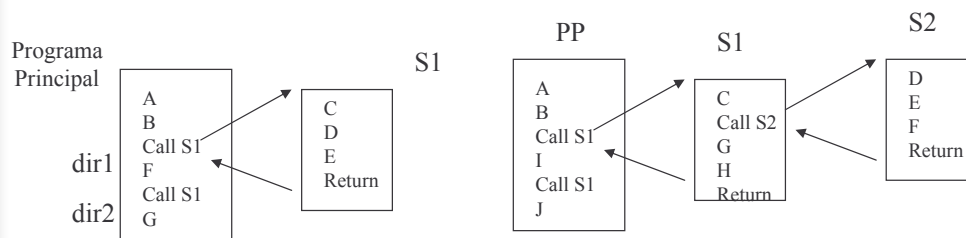
```
...                               Guarda en el registro 7
MOVE R7, RET ← la dirección de retorno
BR SUB1 ← Invocación a la subrutina
RET: <EL RESTO DEL PROGRAMA>
FIN
SUB1: <CUERPO SUBROUTINA>
BR R7 ← Regreso al punto siguiente de la invocación
```



6

Aspectos a considerar

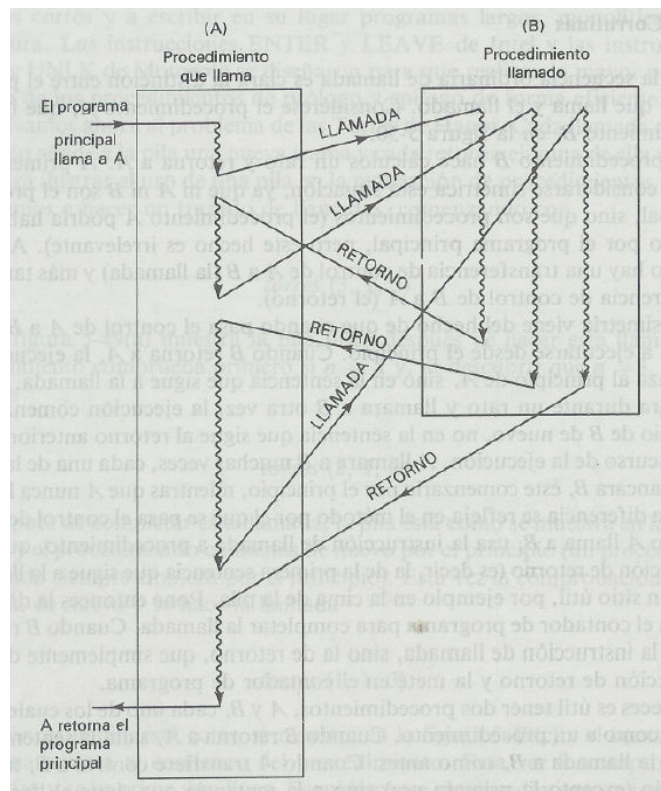
- Una subrutina puede llamarse desde distintas posiciones.
- Una subrutina puede contener llamadas a otras subrutinas. Esto posibilita el anidamiento de subrutinas hasta una profundidad arbitraria.
- Cada llamada a subrutina esta emparejada con una posición de retorno



7

Aspectos a considerar

Una subrutina puede llamarse desde distintas posiciones



8



¿Qué apoyo nos ofrecen los lenguajes ensambladores?

- Instrucción de llamada a subrutina
 - El programador indica cual es el nombre de la subrutina
 - El ensamblador guarda la dirección de retorno en:
 - a) Un registro predeterminado.
 - b) Al principio de la subrutina
 - c) En la pila de ejecución
- Instrucción que realiza el retorno

9

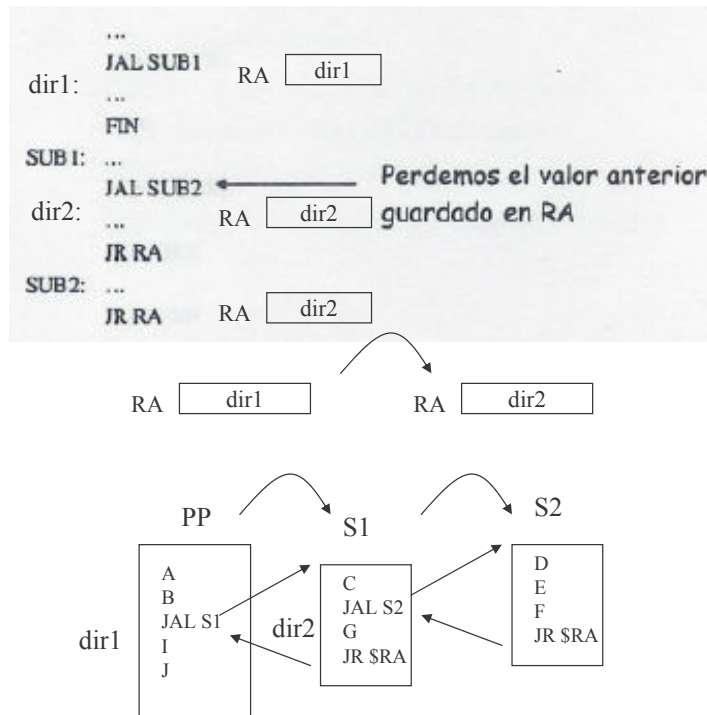


a) La posición de retorno en un registro predeterminado.

- Para invocar la subrutina
 - JAL <nombre subrutina>
JALR \$Registro con la direccion de la Subrutina
 - Coloca la dirección de la instrucción siguiente en el registro \$RA
 - Salta a la dirección indicada por <nombre subrutina> .
o por el contenido del Registro en el caso de JALR
- Para regresar de la subrutina
 - JR \$RA
 - Salta a la dirección indicada por \$RA

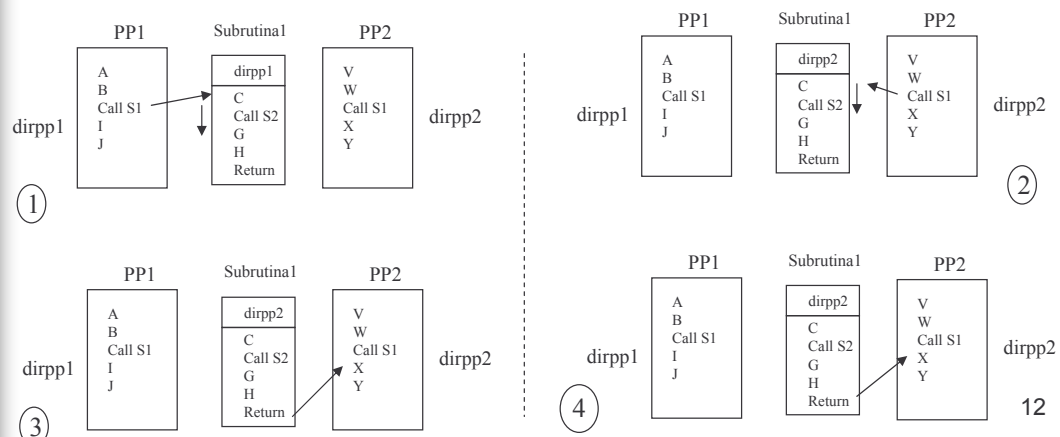
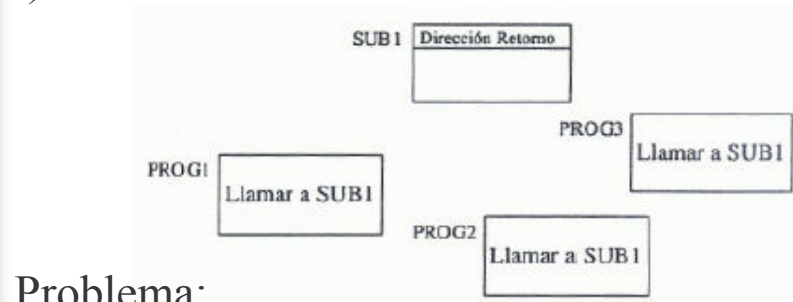
10

a) La posición de retorno en un registro predeterminado. Problema

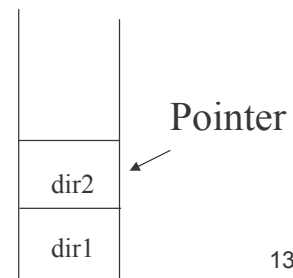
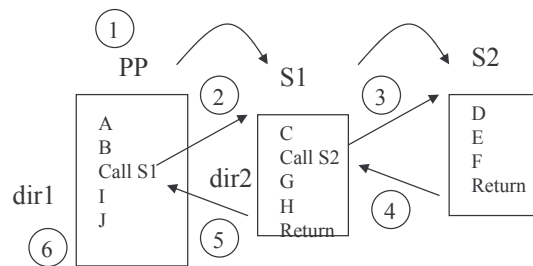
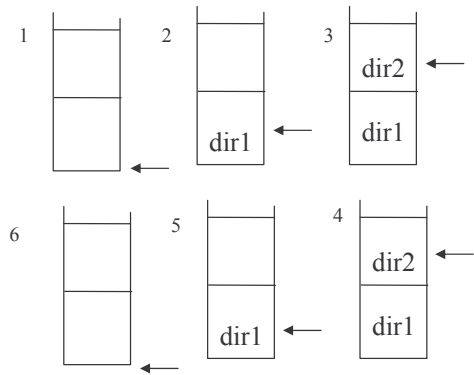
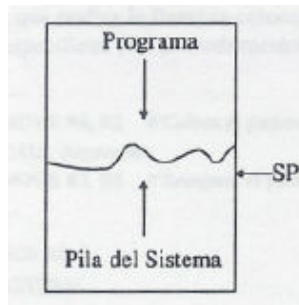


11

b) La dirección de retorno al comienzo de la subrutina



c) La dirección de retorno es guardada en una pila



13

Push y Pop de la Pila

PUSH

```
// palabras de 4 bytes
// SP sube una posición
SUB SP, 4

// Se guarda en la pila la dirección
de retorno
MOVE (SP), RA
```

SRA 100

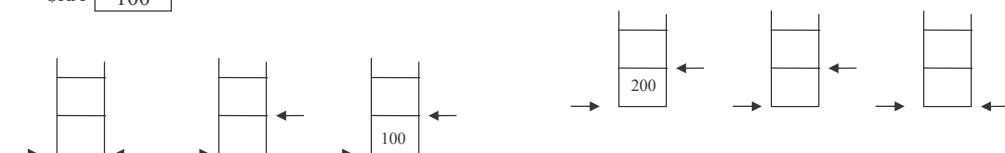
POP

```
// Se recupera la dirección de
retorno
MOVE RA, (SP)

// SP baja una palabra
ADD SP, 4
```

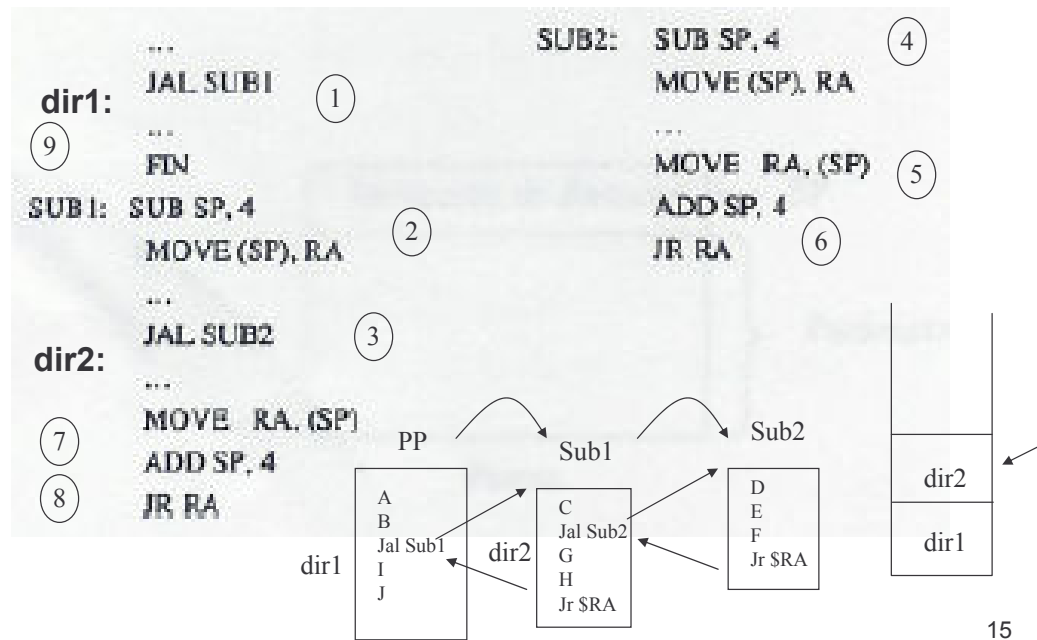
SRA 100

SRA 200



14

Uso de la pila para resolver el problema de las llamadas anidadas



15

Lenguaje ensamblador con instrucciones más elaboradas para el soporte de subrutinas

Manejo automático de la pila

- Para invocar a la subrutina
 - `CALL <nombre de la subrutina>`
 - Se apila la dirección de retorno en la pila de ejecución
 - Se salta al inicio de la subrutina
- Para regresar de la subrutina
 - `RETURN`
 - Se desapila la dirección de retorno en la pila de ejecución.
 - Se salta a la dirección de retorno

16



Manejo de los parámetros

- Se pueden almacenar
 - En los registros
 - En la pila de ejecución.
- Más adelante nos ocuparemos de diferentes clases de parametros
 - Por valor
 - Por referencia

17




Parámetros almacenados en registros

El subprograma, que realiza la llamada, coloca los parámetros en registros específicos y el procedimiento llamado los usa.

```
...  
MOVE R6, R2 // Coloca el parámetro en R6  
CALL decremento  
MOVE R2, R6 // Recupera el parámetro  
...  
decremento: SUB R6, 1  
RETURN
```

18



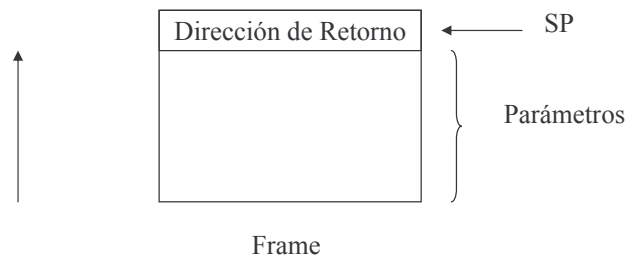
Evaluación de: Parámetros almacenados en registros

- Es una estrategia de fácil implantación.
- Hay un número limitado de registros, por tanto hay un número limitado de parámetros a utilizar.
- No funciona cuando hay recursión.
- Debe ser utilizado con gran cuidado cuando se utilizan subrutinas anidadas

19



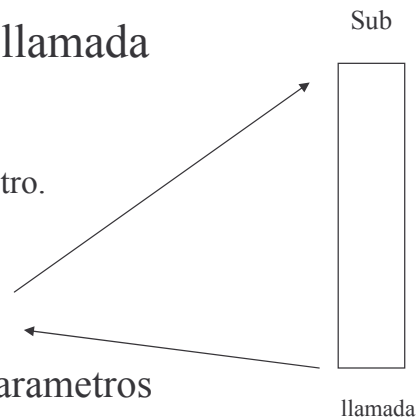
Parámetros almacenados en la pila de ejecución



20

Acciones a realizar para manejar los parámetros en la pila de ejecución

- En el lugar que se realiza la llamada
 - Para cada parámetro.
 - Guardar espacio para el parametro.
 - Colocar el parametro en la pila
 - Llamar al procedimiento
 - Recuperar el espacio de los parametros

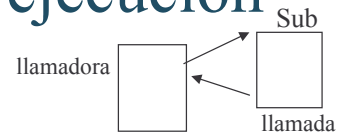


21

Acciones a realizar para manejar los parámetros en la pila de ejecución

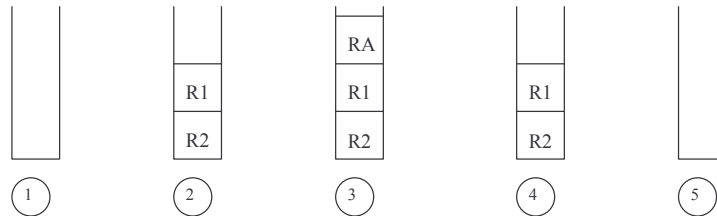
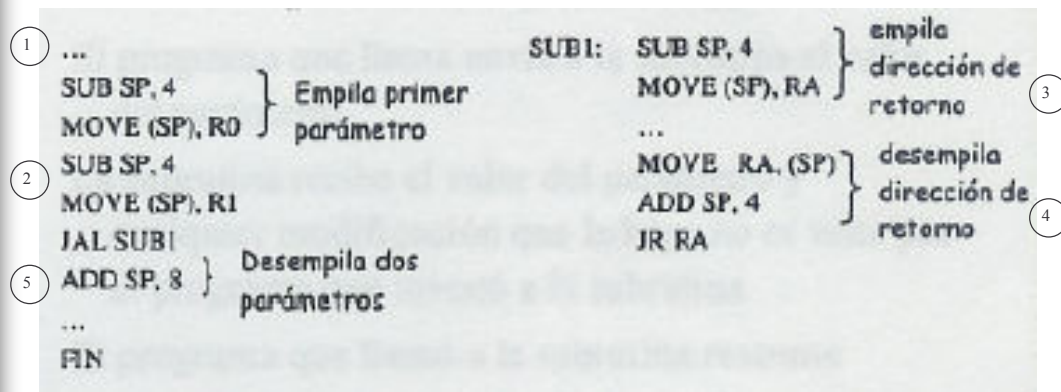
■ La subrutina llamada

- Guarda la dirección de retorno en la pila
- Extrae los parametros de la pila
- Realiza las acciones de la subrutina
- Deja los resultados en un registro o en la pila
- Recupera la dirección de retorno
- Regresa a la instrucción que está en la siguiente dirección a partir de la cual ocurrió la llamada (dirección de retorno).



22

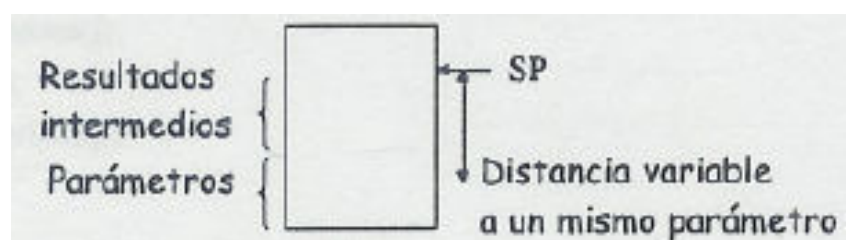
Parámetros almacenados en la pila de ejecución



23

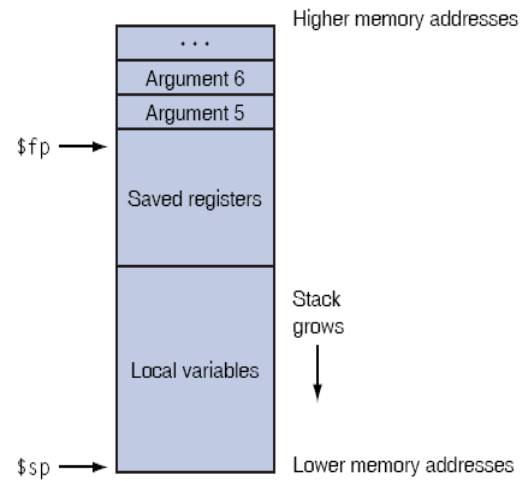
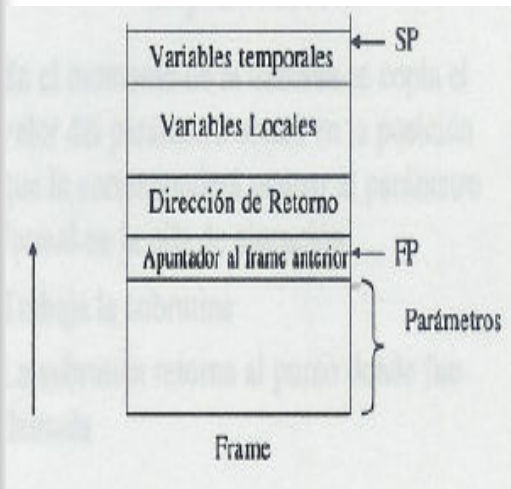
Parámetros almacenados en la pila de ejecución: Problema

- La pila puede usarse para guardar resultados intermedios de los cálculos a realizar
- Los parámetros entonces no estarán a una distancia fija del SP



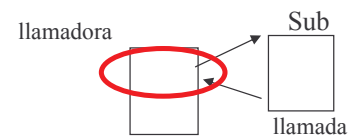
24

Reformulación del Frame



25

Acciones a realizar para manejar los frames

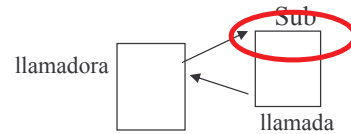


En la rutina que hace la llamada (llamadora)

- Para cada parámetro.
 - Guardar espacio para el parámetro.
 - Colocar el parámetro en la pila
- Llamar al procedimiento
- Recuperar el espacio de los parámetros actualizando el SP

26

Acciones a realizar para manejar los frames



Al entrar en la rutina llamada

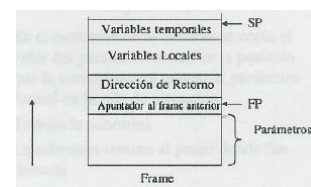
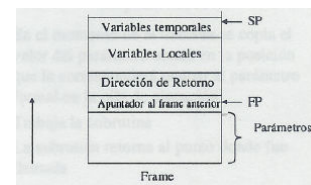
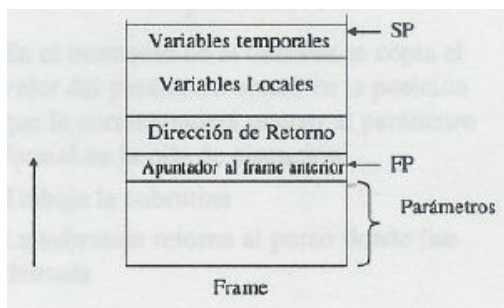
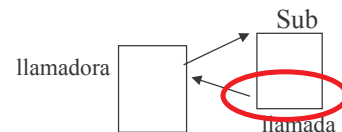
- Almacenar el valor de FP en la pila
- Almacenar la dirección de retorno en la pila
- Actualizar el valor del SP de manera que haya espacio para las variables locales de la subrutina

27

Acciones a realizar para manejar los frames

Al salir de la rutina llamada

- Actualizar el valor del RA: $RA \leftarrow FP - 4$
- Actualizar el valor del SP: $SP \leftarrow FP$
- Actualizar el valor del FP: $FP \leftarrow C(FP)$
- Regresar a la posición que indica el registro RA.



28



Paso de Parámetros

- ¿Cómo se implementan los diferentes tipos de pases de parámetros?
 - Valor
 - Referencia

29



Parámetros por Valor

- El programa que llama envía a la subrutina el valor del parámetro.
- La subrutina recibe el valor del parámetro y cualquier modificación que se le haga no es vista por el programa que invocó a la subrutina.
- Cuando el programa que llamó a la subrutina reasume el control, el parámetro mantiene el valor que tenía en el momento de la invocación.

30

Parámetros por Valor

```
procedure p (valor entero i) {  
    i = i + 1;  
}  
main(){  
    entero j = 3;  
    imprima(j);  
    p(j);  
    imprima(j);  
}
```

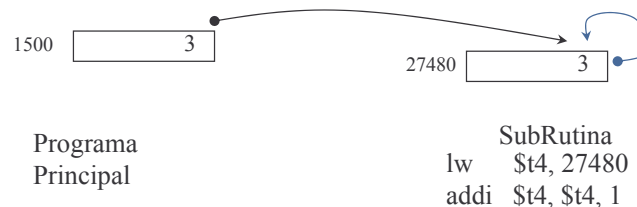
Resultados de la corrida

3
3

31

Parámetros por Valor (Implantación)

- En el momento de la llamada se copia el valor del parámetro actual en la posición que le corresponderá ocupar al parámetro formal en la pila de ejecución.
- Trabaja la subrutina.
- La subrutina retorna al punto donde fue llamada.



32

Parámetros por Referencia

- El programa que llama envía a la subrutina la dirección del parámetro.
- La subrutina recibe la dirección del parámetro y cualquier modificación que ésta le haga, será visible por el programa que invocó a la subrutina.
- Cuando el programa que llamó a la subrutina reasume el control, se observa que el valor del parámetro sufrió las modificaciones que sobre él realizó la subrutina.

33

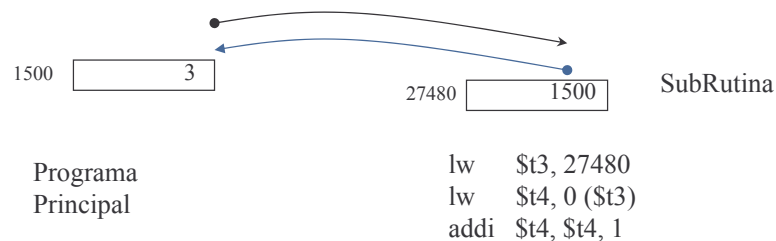
Parámetros por Referencia

```
procedure p (referencia entero i) {  
    i = i + 1;  
}  
main() {  
    entero j = 3;  
    imprima(j);  
    p(j);  
    imprima(j);  
}
```

Resultados de la corrida

3

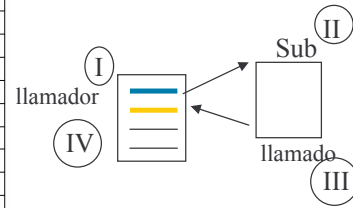
4



34

Uso de Registros en SPIM

Register name	Number	Usage
\$zero	0	constant 0
\$at	1	reserved for assembler
\$v0	2	expression evaluation and results of a function
\$v1	3	expression evaluation and results of a function
\$a0	4	argument 1
\$a1	5	argument 2
\$a2	6	argument 3
\$a3	7	argument 4
\$t0	8	temporary (not preserved across call)
\$t1	9	temporary (not preserved across call)
\$t2	10	temporary (not preserved across call)
\$t3	11	temporary (not preserved across call)
\$t4	12	temporary (not preserved across call)
\$t5	13	temporary (not preserved across call)
\$t6	14	temporary (not preserved across call)
\$t7	15	temporary (not preserved across call)
\$s0	16	saved temporary (preserved across call)
\$s1	17	saved temporary (preserved across call)
\$s2	18	saved temporary (preserved across call)
\$s3	19	saved temporary (preserved across call)
\$s4	20	saved temporary (preserved across call)
\$s5	21	saved temporary (preserved across call)
\$s6	22	saved temporary (preserved across call)
\$s7	23	saved temporary (preserved across call)
\$t8	24	temporary (not preserved across call)
\$t9	25	temporary (not preserved across call)
\$k0	26	reserved for OS kernel
\$k1	27	reserved for OS kernel
\$gp	28	pointer to global area
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address (used by function call)



35

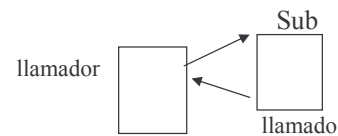
Convenciones a ser usadas en la invocación de subrutinas

Uso de la pila

- El stack pointer (\$sp) apunta a la primera posición libre de la pila
- La pila crece en el sentido de direcciones altas a direcciones bajas
- El frame pointer (\$fp) apunta a la primera dirección del marco

36

Convenciones a ser usadas en la invocación de subrutinas

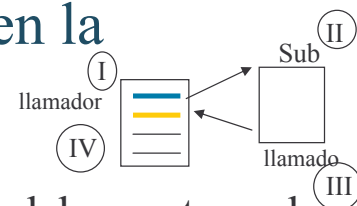


Convenciones con responsabilidad compartida

- El procedimiento (llamador) que invoca a otro procedimiento, es responsable de almacenar los registros \$a0, \$a1, \$a2, \$a3, \$v0, \$v1, \$t0, ..., \$t9
- El procedimiento invocado (llamado) es responsable de almacenar los registros \$s0, ..., \$s7, \$fp, \$ra

37

Convenciones a ser usadas en la invocación de subrutinas

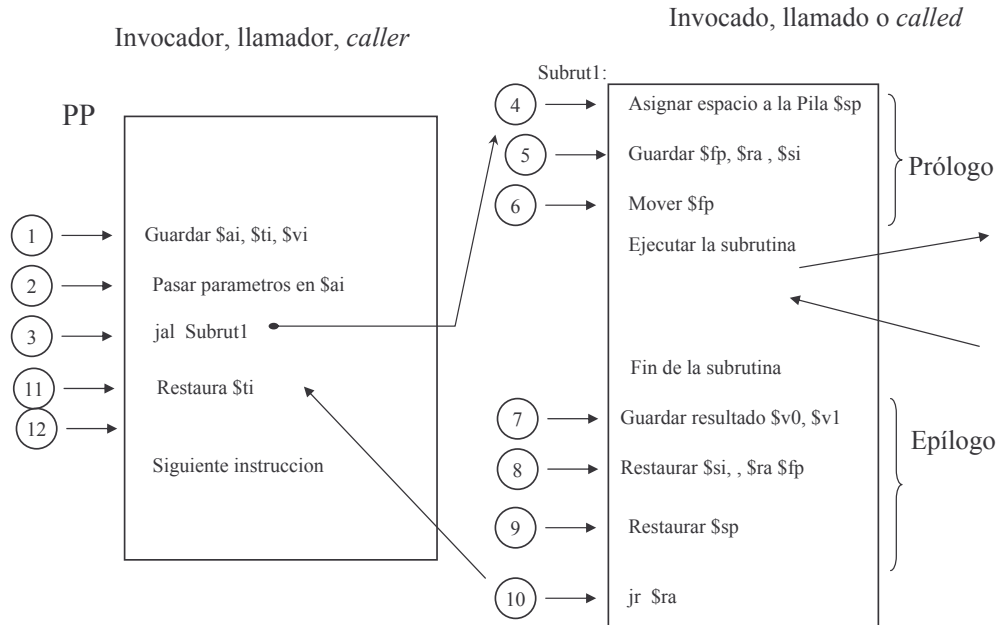


La convención que vamos a utilizar debe ser tomada en cuenta en cuatro puntos:

- I. Antes de que el "llamador" invoque al "llamado"
- II. Justo antes de que el "llamado" comience su ejecución
- III. Inmediatamente antes de que el "llamado" retorne al "llamador"
- IV. Antes de que el "llamador" reanude su ejecución luego de haber concluido el "llamado"

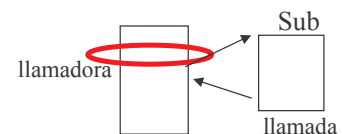
38

Convenciones a ser usadas en la invocación de subrutinas



39

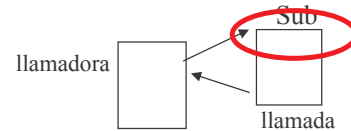
I) Antes de que el "llamador" invoque al "llamado"



- 1) Salva los registros que son responsabilidad del "llamador" si el contenido de los mismos es necesario después de la llamada (\$a0,...\$a3,\$t0,...\$t9)
- 2) Lleva a cabo el pase de argumentos. Los primeros cuatro argumentos son pasados en los registros (\$a0,...,\$a3). Cualquier argumento adicional se pasa usando la pila. Los argumentos son apilados quedando el quinto argumento en el tope de la pila.
- 3) Ejecuta la instrucción jal

40

II) Justo antes de que el "llamado" comience su ejecución

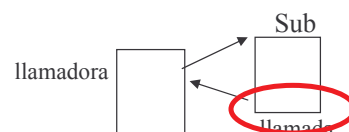


- 4) Asignar espacio a la Pila decrementando \$sp
- 5) Almacenar en la pila los registros que son responsabilidad del "llamado" (\$fp, \$ra, \$s0, ..., \$s7) y que son utilizados en la rutina. El registro \$ra será almacenado en la pila solamente si el "llamado" invoca a su vez a otro procedimiento. El registro \$fp siempre es salvado. Los otros registros se salvan si su contenido puede ser alterado dentro de la rutina.
- 6) Mueve el \$fp a la primera dirección del marco o bloque que le corresponde a la llamada en ejecución.

41

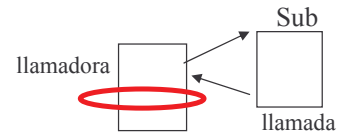
III) Inmediatamente antes de que el "llamado" retorne al "llamador"

- 7) Si el "llamado" es una función que retorna un valor, coloca dicho valor en \$v0 y \$v1. Si es necesario devolver más valores entonces a la rutina se le deben agregar parámetros adicionales por referencia para poder retornar, en estos parámetros, los resultados deseados.
- 8) Restaura todos los registros responsabilidad del "llamado" que fueron almacenados en la pila (\$fp, \$ra, \$s0, ..., \$s7). La pila debe quedar en el mismo estado en que fue recibida
- 9) Restaura \$sp al valor anterior a la llamada
- 10) Se retorna al llamador



42

IV) Después de que el "llamador" invoque al "llamado"

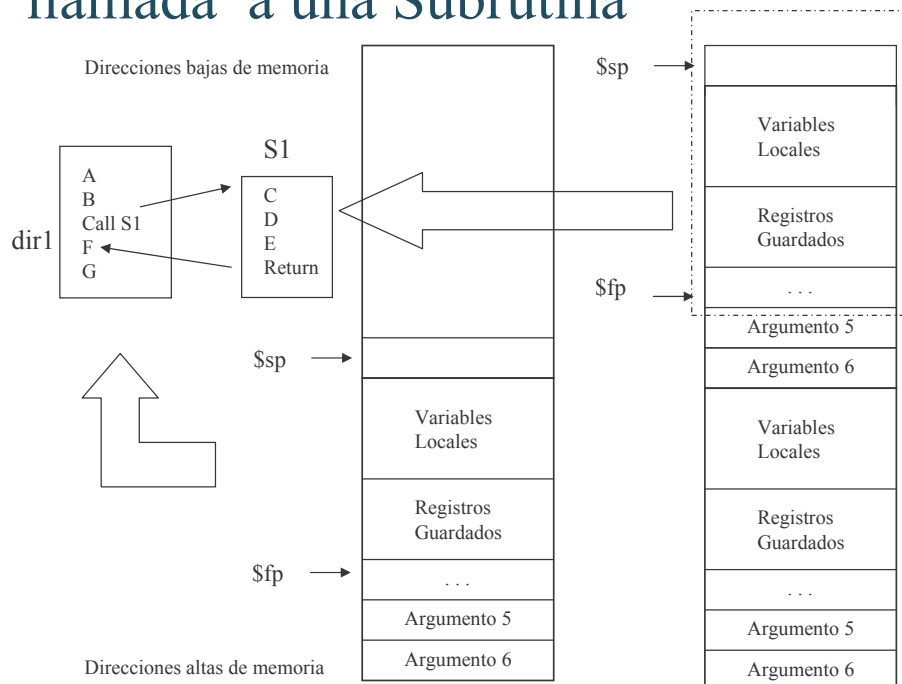


11) Restaurar los registros responsabilidad del "llamador" que fueron colocados en la pila (\$a0,...\$a3,\$t0,...\$t9)

12) En cuanto a los argumentos que fueron pasados por la pila serán desempilados por el "llamador", es decir, quien los colocó en la pila.

43

Como se maneja la pila (stack) durante la llamada a una Subrutina



44

Ejemplo de Subrutina

```
.data
head: .word 0
numnodos: .word 0
node1: .word 8 # Valor
next1: .word 0 # Apuntador al proximo elemento de la lista
node2: .word 2 # Valor
next2: .word 0 # Apuntador al proximo elemento de la lista
node3: .word 10 # Valor
next3: .word 0 # Apuntador al proximo elemento de la lista
node4: .word 4 # Valor
next4: .word 0 # Apuntador al proximo elemento de la lista
node5: .word 6 # Valor
next5: .word 0 # Apuntador al proximo elemento de la lista

minimo: .word 999
men1: .asciiz "Programa recursivo que calcula el minimo de una lista"
menmin: .asciiz "El valor del minimo es : "
mnodo1: .asciiz "El nodo tiene un valor de "
mnodo2: .asciiz " la direccion del proximo nodo es : "
linea: .asciiz "\n"
ban1: .asciiz "El minimo actual es : "
ban2: .asciiz "\n El valor que se esta procesando es el : "
.text
main:
    la $a0, men1
    li $v0, 4
    syscall

# Programa que construye la lista
    la $a0, head
    la $a1, numnodos
# Se agrega el primer nodo
    la $a2, node1
    jal insertar
# Se agrega el segundo nodo
    la $a0, head
    la $a1, numnodos
    la $a2, node2
    jal insertar
# Se agrega el tercer nodo
    .
    .
    .
    li $v0, 10
    syscall
```

Rutina para insertar un nodo en la lista

```
insertar:
# prologo
    addi $sp, $sp, -32
    sw $fp, 32($sp)
    addu $fp, $sp, 32
# fin prologo

    move $t1, $a0 # En $a0 se recibe la direccion del inicio de la lista
    move $t2, $a2 # En $a2 se recibe la direccion del nuevo nodo
    move $t3, $a1 # En $a1 se recibe el numero de nodos agregados a la lista

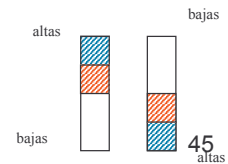
    lw $t4, 0($t1)

    beqz $t4, fininsert
    la $t1, 4($t4)
    lw $t4, 0($t1)

recorrido:
    beqz $t4, fininsert
    la $t1, 4($t1)
    lw $t4, 0($t1)
    j recorrido

fininsert:
    sw $t2, 0($t1)
    lw $t5, 0($t3)
    addi $t5, $t5, 1
    sw $t5, 0($t3)

# epilogo
    lw $fp, 32($sp)
    addu $sp, $sp, 32
    jr $ra
# fin epilogo
```



Ejemplo de Subrutina (sólo esquema)

```
.data
head: .word 0
numnodos: .word 0
node1: .word 8 # Valor
next1: .word 0 # Apuntador al proximo elemento de la lista
node2: .word 2 # Valor
next2: .word 0 # Apuntador al proximo elemento de la lista
node3: .word 10 # Valor
next3: .word 0 # Apuntador al proximo elemento de la lista
node4: .word 4 # Valor
next4: .word 0 # Apuntador al proximo elemento de la lista
node5: .word 6 # Valor
next5: .word 0 # Apuntador al proximo elemento de la lista

minimo: .word 999
men1: .asciiz "Programa recursivo que calcula el minimo de una lista"
menmin: .asciiz "El valor del minimo es : "
mnodo1: .asciiz "El nodo tiene un valor de "
mnodo2: .asciiz " la direccion del proximo nodo es : "
linea: .asciiz "\n"
ban1: .asciiz "El minimo actual es : "
ban2: .asciiz "\n El valor que se esta procesando es el : "
.text
main:
    la $a0, men1
    li $v0, 4
    syscall

# Programa que construye la lista
    la $a0, head
    la $a1, numnodos
# Se agrega el primer nodo
    la $a2, node1
    jal insertar
# Se agrega el segundo nodo
    la $a0, head
    la $a1, numnodos
    la $a2, node2
    jal insertar
# Se agrega el tercer nodo
    .
    .
    .
    li $v0, 10
    syscall
```

Rutina para insertar un nodo en la lista

```
insertar:
# prologo
    subu $sp, $sp, 32
    sw $fp, 32($sp)
    sw $ra, 28($sp)
    sw $si, 24($sp)
    addu $fp, $sp, 32
# fin prologo

    move $t1, $a0
    move $t2, $a2
    move $t3, $a1

    lw $t4, 0($t1)

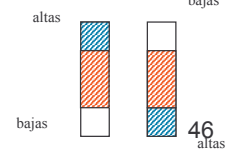
    beqz $t4, fininsert
    la $t1, 4($t1)
    lw $t4, 0($t1)

recorrido:
    beqz $t4, fininsert
    la $t1, 4($t1)
    lw $t4, 0($t1)
    j recorrido

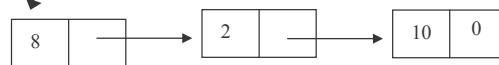
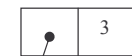
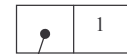
fininsert:
    jal otrasubrutina # eliminar esta linea

    sw $t2, 0($t1)
    lw $t5, 0($t3)
    addi $t5, $t5, 1
    sw $t5, 0($t3)

# epilogo
    lw $fp, 32($sp)
    lw $ra, 28($sp)
    lw $si, 24($sp)
    addu $sp, $sp, 32
    jr $ra
# fin epilogo
```



Ejemplo de Subrutina



47

Ejemplo de Subrutina

```

.data
head: .word 0
numnodos: .word 0
node1: .word 8 # Valor
next1: .word 0 # Apuntador al proximo elemento de la lista
node2: .word 2 # Valor
next2: .word 0 # Apuntador al proximo elemento de la lista
node3: .word 10 # Valor
next3: .word 0 # Apuntador al proximo elemento de la lista
node4: .word 4 # Valor
next4: .word 0 # Apuntador al proximo elemento de la lista
node5: .word 6 # Valor
next5: .word 0 # Apuntador al proximo elemento de la lista
  
```

```

minimo: .word 999
men1: .asciiz "Programa recursivo que calcula el minimo de una lista"
menmin: .asciiz "El valor del minimo es : "
mnodo1: .asciiz "El nodo tiene un valor de "
mnodo2: .asciiz " la direccion del proximo nodo es: "
linea: .asciiz "\n"
ban1: .asciiz "El minimo actual es: "
ban2: .asciiz "\n El valor que se esta procesando es el : "
.text
main:
  la $a0, men1
  li $v0, 4
  syscall
  
```

```

# Programa que construye la lista
la $a0, head
la $a1, numnodos
  
```

```

# Se agrega el primer nodo
la $a2, node1
jal insertar
  
```

```

# Se agrega el segundo nodo
la $a0, head
la $a1, numnodos
la $a2, node2
jal insertar
  
```

```

# Se agrega el tercer nodo
la $a0, head
la $a1, numnodos
  
```

Rutina para insertar un nodo en la lista

```

insertar:
# prologo
subu $sp, $sp, 24
sw $fp, 24($sp)
addu $fp, $sp, 24
# fin prologo
  
```

```

move $t1, $a0 # En $a0 se recibe la direccion del inicio de la
# lista
move $t2, $a2 # En $a2 se recibe la direccion del nuevo nodo
move $t3, $a1 # En $a1 se recibe el numero de nodos agregados
# a la lista
  
```

```

lw $t4, 0($t1)
  
```

```

beqz $t4, fininsert
la $t1, 4($t4)
lw $t4, 0($t1)
  
```

recorrido:

```

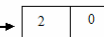
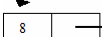
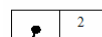
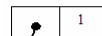
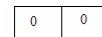
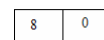
beqz $t4, fininsert
la $t1, 4($t1)
lw $t4, 0($t1)
j recorrido
  
```

```

fininsert: sw $t2, 0($t1)
lw $t5, 0($t3)
addi $t5, $t5, 1
sw $t5, 0($t3)
  
```

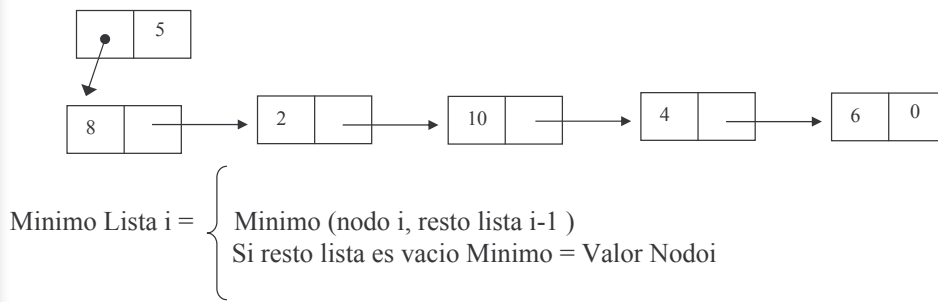
```

# epilogo
lw $fp, 24($sp)
addu $sp, $sp, 24
jr $ra
# fin epilogo
  
```

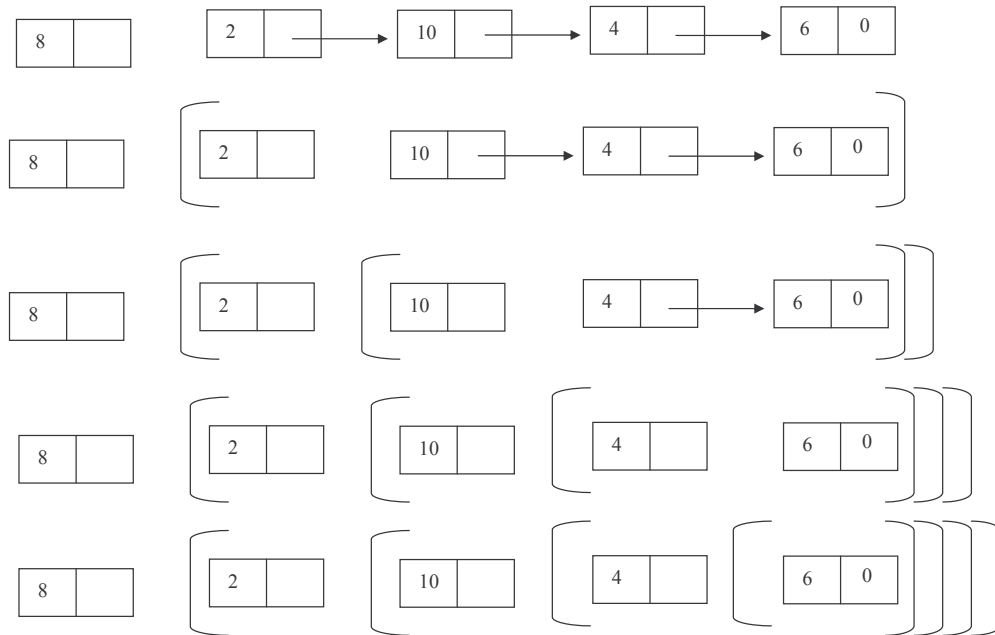


48

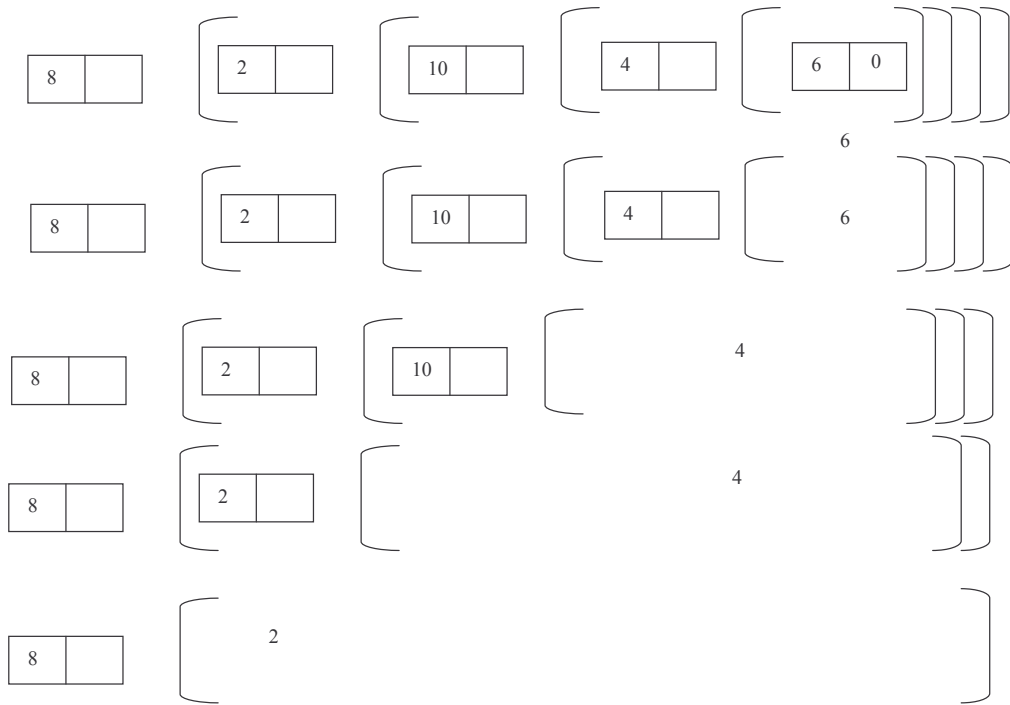
Otro Ejemplo de Subrutina



49



50



2

51

Otro Ejemplo de Subrutina

```
minimo: .word 999
men1:   .asciz "Programa recursivo que calcula el minimo de una lista\n"
menmin: .asciz "El valor del minimo es : "
mnode1: .asciz "El nodo tiene un valor de "
mnode2: .asciz " la direccion del proximo nodo es: "
linea:  .asciz "\n"
ban1:   .asciz "El minimo actual es: "
```

```
correp: lw $s3, head
        lw $a0, 0($s3) # Primer valor de la lista
        lw $a1, 4($s3) # El apuntador al resto de la lista
        jal bminimo
        sw $v0, minimo

        la $a0, menmin
        li $v0, 4
        syscall
        lw $a0, minimo
        li $v0, 1
        syscall
        li $v0, 10 # Fin del Programa
        syscall
```

```
bminimo:
# prologo
        subu $sp, $sp, 32
        sw $fp, 20($sp)
        sw $ra, 16($sp)
        sw $s3, 12($sp)
        sw $s2, 8($sp)
        sw $s1, 4($sp)
        addu $fp, $sp, 32
        addi $s5, $s5, 1 # No espere del prologo

# fin prologo
        # almaceno en mi frame mis variables locales
        move $s1, $a0
        move $s2, $a1
        bnez $s2, seguir # Si el resto de la lista no es igual a cero (hay mas elementos)
        move $v0, $s1 # devuelvo el primer valor de la lista.
        li $v1, 1 # van1

        # Para unificar los retornos
        sw $a0, 0($sp)
        subu $sp, $sp, 4
        jr $ra # Al llegar al final de la lista, el minimo es el elemento
                # que esta en el nodo o lista de tamaño 1

seguir: # debo salvar a0 antes de la llamada
        sw $a0, 0($sp)
        addi $sp, $sp, -4
        lw $a0, 0($s2)
        lw $a1, 4($s2)
        jal bminimo
        li $v1, 0

ret:     ble $v0, $a0, segret # Si el candidato a minimo es menor al minimo original
                # colocar entonces al nuevo candidato como minimo
        move $v0, $a0

segret:  # Restaura el valor de $a0
        lw $a0, 4($sp)
        addu $sp, $sp, 4

# epilogo
        lw $fp, 20($sp)
        lw $ra, 16($sp)
        lw $s3, 12($sp)
        lw $s2, 8($sp)
        lw $s1, 4($sp)
        addu $sp, $sp, 32
        addi $s6, $s6, 1 # No es parte del epilogo (bandera)
        jr $ra

# fin epilogo
```

52



FIN