Programación con Subrutinas

Objetivos

- Comprender las ventajas de una programación modular.
- Saber cuáles son los pasos a seguir en una llamada a subrutina.
- Entender cómo se lleva a cabo el paso de parámetros entre subrutinas.
- Aprender a implementar programas en lenguaje ensamblador mediante el uso de subrutinas.

Indice

- Llamadas a subrutinas
- Instrucciones involucradas
- Paso de parámetros
  - Por valor
    - en registro
    - en pila (stack)
  - Por referencia
- Técnicas de anidamiento
- Ejercicios: programación con subrutinas

Llamadas a subrutinas

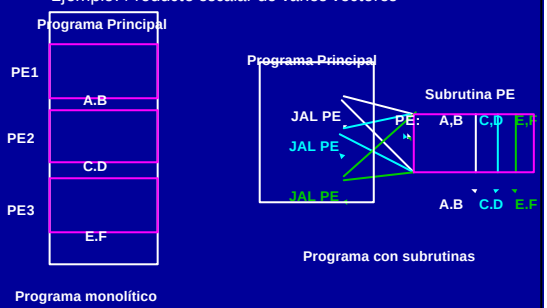
- Una subrutina implementa un algoritmo utilizado con mucha frecuencia.
- Programar con subrutinas proporciona :
  - Modularidad
  - Reutilización del código
    - librerías
  - Facilidad de depuración
  - Flexibilidad de ampliación
  - Claridad en el diseño

Opciones para invocar procedimientos o Subrutinas

- **Dirección de retorno debe guardarse (en registro especial o en un "GPR")**
- **Cambio de contexto: viejas arquitecturas tenía mecanismos para salvar registros, las nuevas requieren que el compilador se encargue**
- **"Caller saving": el que invoca el procedimiento guarda los registros que necesita luego del llamado**
- **"Callee saving": el procedimiento llamado salva los registros que necesita usar**

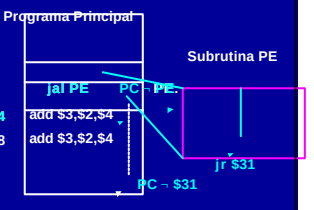
Llamadas a Subrutinas

- Ejemplo: Producto escalar de varios vectores



Llamadas a Subrutinas

- Instrucciones involucradas
  - **jal etiqueta (jump and link)**
  - **jr $31 (jump at register)**
- El programa principal ejecuta
  - **jal etiqueta**
    - **$31 ¬ PC+4**
    - **PC ¬ etiqueta**
- La subrutina ejecuta
  - **jr $31**
    - **PC ¬ $31**



Llamadas a Subrutinas

- Ejemplos de programación con subrutinas

Ejemplo 1

Programa principal

```
        .data 0x10000000
        .globl __start
        .text 0x400000
__start:
        ori $4,$0,5
        jal MA
        add $9,$5,$0
        .end
```

Subrutina MA

```
MA:
        add $5,$4,$4
        jr $31
```

Se pide:

1¿Cuál es la dirección de memoria que se almacena en el registro$31 y en el PC al realizarse la llamada a la subrutina?

2¿Qué valor se obtiene en los registros $4 y $9 tras la ejecución del programa anterior? ¿Qué función implementa la subrutina MA?

Llamadas a Subrutinas

Ejemplo 2

Programa principal

```
        .data 0x10000000
        .globl __start
        .text 0x400000
__start
        li $4,5
        li $5,0
        jal MA
        add $10,$0,$5
        ori $4,$0,6
        jal MA
        add $11,$0,$5
        add $12,$10,$11
        .end
```

Subrutina MA

```
        .text 0x400100
MA:
        add $5,$4,$4
        jr $31
```

Se pide:

1Cuál es el valor de PC y $31 en la 1ª y 2ª llamada.

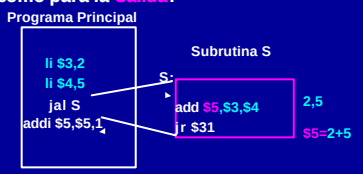
2 Indique cuál es el valor de $4 y $5 tras la 1ª llamada y tras la 2ª .

3 Determínese cuál es el valor de $12, después de ejecutar el programa anterior.

4 Indique los registros (de entrada y salida) utilizados para transferir parámetros

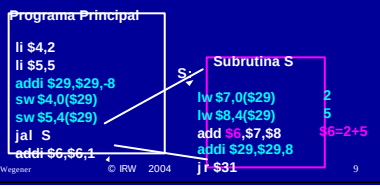
Paso de parámetros

- Parámetros
  - De entrada a la subrutina
  - De salida de la subrutina
- Mediante Registros
  - **Cualquiera de los registros se pueden utilizar excepto el $0, $1, $29, $31, Hi y LO . Tanto para la Entrada de parámetros como para la Salida.**



Paso de parámetros mediante la Pila

- **Programa Principal**
  - **Decrementar el apuntador de la pila (SP) en una palabra (4 bytes) por el número de registros a salvar.**
  - **Almacenar el dato en la posición actual apuntada por el SP.**
- **Subrutina**
  - **Cargar el dato desde la posición actual apuntada por SP en un registro.**
  - **Incrementar el SP en el número de palabras previamente decrementado(4 bytes * nº de registros).**



Paso de parámetros

Ejemplo 3 **Subrutina que suma dos enteros de 32 bits**

- **Versión 1: Paso de parámetros por registro**

```
## Area de datos
        .data 0x10000200
A:      .word 3
B:      .word 20
Suma:   .word
## Area de programa
        .text 0x400000
        .globl __start
## Programa Principal
__start:
        la $6,A
        la $7,B
        lw $4,0($6)
        lw $5,0($7)
        jal __Suma
        la $6, Suma
        sw $2,0($6)
        .end
## Subrutina __Suma
__Suma:
        add $2,$4,$5
        jr $31
```

Se pide

1 Indique los registros utilizados para el paso de parámetros (entrada y salida)

2 Indique la forma y la dirección de almacenamiento del resultado

3 Cambiar la instrucción sw $2,0($6) por sw $2, -4($6), ¿qué efectos tendría y qué problemas podría acarrear?

Paso de parámetros

Ejemplo 4 **Subrutina que suma dos enteros de 32 bits - Versión 2: Paso de parámetros por la pila**

```
## Area de datos
        .data 0x10000200
A:      .word 3
B:      .word 20
Suma:   .word
## Area de programa
        .text 0x400000
        .globl __start
## Programa Principal
__start:
        la $2,A
        la $3,B
        lw $4,0($2)
        lw $5,0($3)
        addi $29,$29,-8
        sw $4,0($29)
        sw $5,4($29)
        jal __Suma
        la $2,Suma
        sw $6,0($2)
.end
## Subrutina __Suma
__Suma:
        lw $7,0($29)
        lw $8,4($29)
        add $6,$7,$8
        addi $29,$29,8
        jr $31
```

Paso de parámetros

Ejemplo 4

Se pide

1 Indique los registros y las direcciones de memoria utilizadas para el paso de los parámetros(entrada y salida).

2 Indique el procedimiento y la dirección de almacenamiento del resultado.

3 Indique cuáles son las instrucciones encargadas de enviar los parámetros de entrada a la subrutina.

4 Indique cuáles son las instrucciones encargadas de recibir los parámetros de entrada a la subrutina.

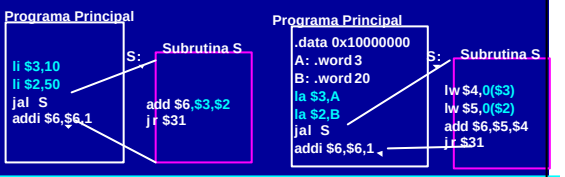
5 Indique si los parámetros transferidos son las direcciones de los datos (sus referencias) o son los datos propiamente (sus valores) .

Se propone como ejercicio:

1 Resolver el ejercicio anterior mediante el paso de parámetros por referencia, tanto en registros como en pila.

Paso de parámetros

- Paso de parámetros
  - **Por VALOR: el parámetro es el dato.**
  - **Por REFERENCIA : el parámetro es la dirección del dato**



Se propone como ejercicio

1 Razónese cuál es el tipo de parámetros (por valor o por referencia) que se pasan en los ejemplos:

Ejemplo1, Ejemplo2, Ejemplo3, Ejemplo4

Paso de parámetros

Subrutina para el cálculo del cuadrado de un entero de 32 bits

Ejemplo 5

- **Paso de parámetros por Valor**

```
Program ;
Var
  A,B : integer
begin
  A:=2
  Cuadrado(A)
  B:=A
End.
```

```
Function Cuadrado(A : integer) :
integer;
Begin
      Cuadrado: = A*A;
End;
```

Se pide

Traducir el código pascal a código ensamblador, realizando un paso de parámetros por valor

Paso de parámetros

Ejemplo 5 **Subrutina que calcula el cuadrado de un entero de 32 bits**

- **Paso de parámetros por Valor**

```
.data 0x10000200
.text 0x400000
.globl __start
__start:
  li $4, 2
  jal __Cuadrado
  add $6, $5, $0
.end
```

```
__Cuadrado:
        mult $4,$4
        mflo $5
        jr $31
```

Se pide

Obtengase una versión del programa en la que el pase de parámetros se realice por valor, pero a través de la pila

Paso de parámetros

Ejemplo 6 Subrutina que calcula el cuadrado de un entero de 32 bits

- Paso de parámetros por Referencia

```
      .data 0x10000200
A:    2
      .text 0x400000
      .globl __start
__start:
      la $3, A
      jal __Cuadrado
      add $6, $5, $0
      .end
```

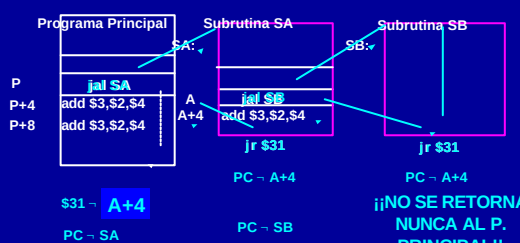
```
__Cuadrado:
        lw $4,0($3)
        mult $4,$4
        mflo $5
        jr $31
```

Se pide

Obtengase una versión del programa en la que el pase de parámetros se realice por referencia, pero a través de la pila

Técnicas de anidamiento

- Una subrutina llama (jal) a su vez a otra.
  - Ejemplo: P. Principal llama a SA y SA llama a SB
  - Si no se toman precauciones, existen problemas!

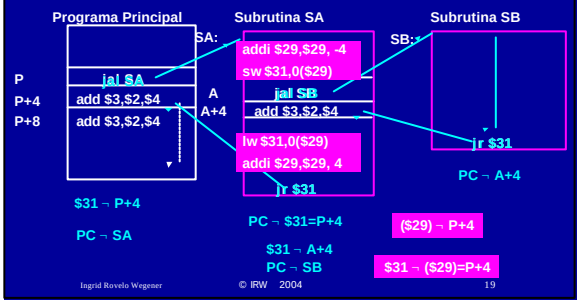


Técnicas de anidamiento

- ¿Cómo solucionar la perdida de la dirección de retorno ?
- Reglas
  - Si la subrutina llama a otras subrutinas (jal)
    - Al inicio de la subrutina se almacena en la pila la dirección de retorno($31).
    - Antes del retorno (jr) se recupera de la pila la dirección de retorno($31).

Técnicas de anidamiento

- Resolución del problema de la pérdida de dirección



Técnicas de anidamiento

- ¿Cómo evitar que el contenido de un registro sea modificado por una subrutina?
- **Procedimiento:**
  - **Se Almacena en la pila el contenido de los registros que la subrutina utilice para cálculos temporales**
    - **Método (Callee Save: Subrutina)**
      - **El invocado guarda los registros**

EJERCICIOS

- Ejercicio1
  - **Impleméntese** $\sum_{i=1}^{N} i \bullet a[i]$ **y que devuelva el resultado en $2**
  - **Paso de parámetros por referencia**
    - **$4=5, número de elementos del vector, por valor**
    - **$5= dirección de inicio del vector, por referencia**

```
ALGORITMO:
inicio
    i=1
    suma=0
    mientras i<=N hacer
            suma=suma+i* a[i]
            i=i+1
    fin mientras
Fin
```


EJERCICIOS - Solución

```
            .data 0x10000200
V:          .word 1,3,4,7,9
            .globl __start
__star:     la$5,V
            li $4,5
            jal proc
            .end

    .text 0x400300      #Dirección inicial del código de la
proc:                   subrutina
    addi $29,$29,-20    # Punto de entrada de la subrutina proc
    sw $5,0($29)
    sw $6,4($29)        # Se reserva en la pila 20 Bytes
    sw $8,8($29)        # Se guarda en la pila el contenido de
    sw $9,12($29)       #los registros $4, $6, $8, $9 y $11
    sw $11,16($29)      #
    li $6,1             #
                        # El $6 se emplea como contador
```


EJERCICIOS - Solución (cont.)

```
mientras:
    sle $11,$6,$4     # Se itera hasta que  $6 <= $4
    beq $11,$0,salida       #
    lw $8,0($5)       # se almacena en $8 el elemento del vector
    mult $6,$8              # $Hi $LO= $6*$8
    mflo $9           # $9 = $LO
    add $2,$2,$9      # se acumula la suma
    addi $5,$5,4      # $5 direcciona al siguiente elemento del vector

    addi $6,$6,1      # se incrementa al  contador
    j mientras              # se inicia una nueva iteración
```

EJERCICIOS - Solución (cont.)

Salida:

```
lw $5,0($29)        # Se restablece el valor original de los registros

lw $6,4($29)        # $4, $6, $8, $9 y $11
lw $8,8($29)        #
lw $9,12($29)       #
lw $11,16($29)      #
addi $29,$29,20     # Se restablece el valor del SP
jr $31              # retorno de la subrutina
```

Se pide

1 El registro $2 es modificado por la subrutina proc. A pesar de ello ¿porqué no es preservado en la pila ?

2 ¿Qué cambios se deben realizar si en vez la directiva .word se utiliza la .half y la .byte?

3 Impleméntese el programa realizando el paso de parámetros a través de la pila.

EJERCICIOS - Solución

- Ejercicio 2 $p = \sum_{i=1}^{n} a[i] \times b[i]$

```
data 0x10000000
va:     .word 54, 36, 32, -4    # Vector de enteros a
vb:     .word 11, -2, 8, -8     # Vector de enteros b
long:   .word 4                 # Tamaño de los vectores
prod:   .space 4                # Resultado del producto
        .text 0x400000          # Comienzo segmento de código
        .globl __start

__start:    la  $4, va          # $4 contiene dirección de a
        la  $5, vb              # $5 contiene dirección de b
        la  $3, long            # $3 apunta a long
        lw  $6, 0($3)           # $6 contiene la longitud
        la  $12, prod           # $12 contiene la dirección de prod
        jal p_esc               # salto a la subrutina
        j fin                   # salto al final del programa
```


EJERCICIOS - Solución (cont.)

```
p_esc:  addi $29, $29, -24      # Dejamos espacio en pila
        sw  $11,  0($29)        # Salvar registro
        sw  $12,  4($29)        # Salvar registro
        sw  $13,  8($29)        # Salvar registro
        sw  $4,  12($29)        # Salvar registro
        sw  $5,  16($29)        # Salvar registro
        sw  $6,  20($29)        # Salvar registro

        add  $2,  $0, $0        # inicializamos $2 a cero
bucle:  beq  $6,  $0, fins      # hemos llegado al final?
        lw   $11, 0($4)         # $11 contiene a[i]
        lw   $12, 0($5)         # $12 contiene b[i]
        mult $11, $12           # cá lculo de a[i]*b[i]
        mflo $13                # $13 contendrá el a[i]*b[i]
        add  $2, $2, $13        # actualizamos la suma
        addi $4, $4, 4          # siguiente de b
        addi $5, $5, 4          # siguiente de a
        addi $6, $6, -1         # nos queda un elemento menos
        j   bucle
```


EJERCICIOS - Solución 2 (cont.)

```
fins:   lw  $11,  0($29)        # Restaurar registro
        lw  $12,  4($29)        # Restaurar registro
        lw  $13,  8($29)        # Restaurar registro
        lw  $4,  12($29)        # Salvar registro
        lw  $5,  16($29)        # Salvar registro
        lw  $6,  20($29)        # Salvar registro

        addi $29, $29, 24       # Dejar el puntero de pila como
estaba
        jr $31

fin:    sw $2, 0($12)           # almacenamos el resultado en memoria
        nop
.end
```

Se pide

1 El registro $2 es modificado por la subrutina proc. A pesar de ello ¿porqué no es preservado en la pila ?

2 ¿Qué cambios se deben realizar si en vez la directiva .word (en va y v b sólo) se utiliza la .half y la .byte?

3 Impleméntese el programa realizando el paso de parámetros a tr avés de la pila

Llamadas al sistemas

Service	System Call Code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

Llamadas al sistemas

```
.data
     msg_hola: .asciiz "hola mundo\n"

.text
    .globl main
    main:
      # printf("hola mundo\n") ;
      li $2 4
      la $4 msg_hola
      syscall
```

Ventajas e inconvenientes del lenguaje ensamblador (I)

- **Ventajas del lenguaje ensamblador**
  - Útil cuando es crítico alguno de los siguientes factores:
    - Tiempo de ejecución del programa.
    - Tamaño del programa.
  - El programador puede seleccionar instrucciones específicas de la arquitectura para realizar una determinada operación.

Ventajas e inconvenientes del lenguaje ensamblador (II)

- **Inconvenientes del lenguaje ensamblador**
  - Los programas son específicamente inherentes a la máquina.
  - Son de mayor tamaño que los programas equivalentes escritos en lenguaje de alto nivel. Menor productividad del desarrollo software.
  - Los programas son difíciles de leer, escribir y pueden contener más errores.
- **Soluciones híbridas para aprovechar la fortaleza de cada lenguaje**
  - La mayor parte del programa se escribe en alto nivel.
  - Las secciones críticas en lenguaje ensamblador.

Funcionamiento del ensamblador (I)

- Un ensamblador traduce un archivo con sentencias en lenguaje ensamblador a un archivo de instrucciones máquina y datos binarios.
- Traducción en dos pasadas:
  - Primera pasada:
    - Calcula las posiciones de memoria que corresponden a los nombres simbólicos que aparecen en el programa para que sean conocidas cuando de traduzcan las instrucciones. Crea tabla de símbolos.
  - Segunda pasada:
    - Traduce cada sentencia del lenguaje ensamblador al combinar los equivalentes numéricos de los códigos de operación, especificadores de registros y rótulos de la tabla de símbolos en una instrucción legal.

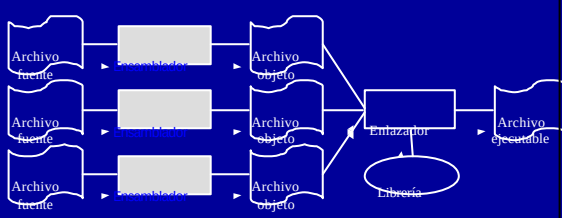
Funcionamiento del ensamblador (II)

- Rótulos o etiquetas
  - Externos o globales: Pueden ser referenciados desde otro archivo distinto de aquél en el que se define (hay que declararlos como tales).
  - Internos o locales: Sólo puede ser referenciado en el archivo en el que se define (locales por defecto).

El montador de enlaces (linker)

- El ensamblador procesa cada archivo de un programa de forma individual. Solamente se conocen las direcciones de los rótulos locales.
- Necesidad de otra herramienta: El montador de enlaces (o linker).
  - Combina una colección de archivos objetos y librerías (opcional) en otro archivo ejecutable al resolver los rótulos externos.
  - El ensamblador asiste al montador suministrándole una tabla de símbolos o rótulos y referencias no resueltas.

El montador de enlaces (linker)



Utilidades de los ensambladores

- Los ensambladores proporcionan diversas características (utilidades) que facilitan al programador la escritura de los programas.
- Utilidades:
  - Directivas para organizar datos en memoria.
    - Permite al programador describir los datos de una manera más concisa y natural que la representación binaria: decimal, ASCII, hexadecimal,...
  - Macros
    - Permiten nombrar una secuencia de instrucciones frecuentemente utilizada. No confundir con procedimiento o subrutina.

Definición de macroinstrucciones

- **Macroinstrucción**. Definición de una secuencia de sentencias de forma que puedan ser referenciadas mediante una única línea
  - Cuando en el código aparece una macroinstrucción, ésta se expande, se sustituye la llamada por la secuencia de sentencias que la forman
  - Puede aceptar un conjunto de parámetros que son sustituidos en cada llamada a la misma

Directivas relacionadas con macroinstrucciones:

Definición de macroinstrucción

- Ej. Etiqueta MACRO
- Fin de macroinstrucción ENDM
- Abortar la expansión de la macroinstrucción MEXIT

Utilidades de los ensambladores

- Pseudoinstrucciones o directivas
  - Son proporcionadas por algunos lenguajes ensambladores y no forman parte del repertorio de instrucciones del procesador. El ensamblador las sintetiza a partir de instrucciones puras del procesador.
- Utilización de símbolos
  - Muy útil en instrucciones de control de flujo o para hacer referencia a los datos.