Winmips – Subrutinas, Parámetros y Convenciones

Arquitectura de Computadoras

Winmips – Elementos de Subrutinas

- Mecanismo de llamado:
  - MSX88 → **CALL {dirección subrutina}**
  - WINMIPS → **JAL {dirección subrutina}**
- Mecanismo de retorno:
  - MSX88 → **RET**
  - WINMIPS → **JR R31 ; JAL deja la dir. de retorno en R31**
- Parámetros por registro:
  - MSX88 → **AX, BX, CX, DX**
  - WINMIPS → **R1 a R30**
- Parámetros por pila:
  - MSX88 → **PUSH {reg. 16 bits}, POP {reg. 16 bits}**
  - WINMIPS → **No hay pila, pero simulamos PUSH y POP**

Winmips – Elementos de Subrutinas

- Parámetros por valor:
  - MSX88 ➔ **MOV {reg. Destino}, {dir. memoria}**
    MOV {reg. Destino}, {valor literal}
  - WINMIPS ➔ **LD {reg. Destino}, {dir. memoria}(r0)**
    DADDI {reg. Destino}, R0, {valor literal}

  Ejemplos:

  LD r1, dato(r0) → **r1 = [dir. dato + r0]** (contenido / referencia)

  DADDI r1, r0 , 5 → **r1 = 5 + r0 = 5**

- Parámetros por referencia:
  - MSX88 ➔ **MOV {reg. Destino}, OFFSET {dir. memoria}**
  - WINMIPS ➔ **DADDI {reg. Destino}, r0, {dir. memoria}**

  Ejemplo:

  DADDI r1, r0, dato → **r1 = dir. dato + r0** (referencia)

Winmips – Ejemplo de subrutinas

```
.data
        result: .word 0
.text
        daddi r4, r0, 10     ; Primer parámetro en r4
        daddi r5, r0, 20     ; Segundo parámetro en r5
        jal sumar            ; Se invoca a la subrutina "sumar"
        sd r2, result(r0)    ; Resultado de "sumar" en r2
        halt

sumar:  dadd r2, r4, r5  ; subrutina "sumar" (r31 = dir. Retorno)
        jr r31           ; Retorna a la instr. sig. al "JAL sumar"
```

Winmips – Problemas de Subrutinas

Al utilizar subrutinas pueden surgir algunas dificultades y/o problemas que debemos solucionar. Estos son:

- **Caos en el uso de registros:** Podemos usar "todos" los registros para cualquier tarea. Es necesario mantener el "orden" reservando registros para funciones específicas. Ejemplo: parámetros, cálculos, uso de pila, etc.
- **Nombres de registros poco significativos:** al reservar registros para distintas funciones sería interesante darles nombres significativos. Si usamos r29 como puntero de pila podríamos denominarlo SP (Stack Pointer).
- **Anidamiento de subrutinas:** Cuando el programa principal invoca a una subrutina deja en r31 la dirección de retorno. Invocar una nueva subrutina desde esta deja la dirección de retorno en r31, sobrescribiendo la dirección anterior. Es necesario emplear un mecanismo para permitir los anidamientos.
- **Efectos laterales:** Cuando se invoca una subrutina es posible que esta utilice registros que estén en uso desde donde se la llama. Es necesario un mecanismo que preserve los valores de los registros.

Winmips – Convención

- Agrupar registros para funciones específicas y nombrarlos según esta función:
  - Usamos el mismo criterio y los nombres que usa el compilador de C para MIPS.
- Implementar una pila:
  - Destinar un registro como puntero de pila e implementar las operaciones de “PUSH” y “POP”.
- Utilizar la pila para:
  - Asegurar la dirección de retorno antes de invocar una subrutina.
  - Asegurar los valores de registros (según la convención de C) antes de invocar a una subrutina.

Winmips – Convención, registros

Registro	Nombre	Uso	Ref.
r0	$zero	Siempre cero, no se modifica	**1**
r1	$at	**A**ssembler **T**emporary – Reservado para el ensamblador	
r2 - r3	$v0 y $v1	**V**alores de retorno de la subrutina llamada	**1**
r4 - r7	$a0 - $a3	**A**rgumentos (parámetros) pasados a la subrutina llamada	**1**
r8 - r15 r24 - r25	$t0 - $t7 $t8 - $t9	Registros **T**emporales de propósito general. Cuando se retorna de una subrutina, no se garantiza la permanencia del valor anterior.	**1**
r16 - r23	$s0 - $s7	Registros **S**alvados. Si una subrutina los utiliza debe salvar los valores y reponerlos antes de retornar	**1, 2**
r26 - r27	$k0 - $k1	Para uso del **K**ernel del sistema operativo	
r28	$gp	**G**lobal **P**ointer: Puntero a zona de memoria estática del programa	**2**
r29	$sp	**S**tack **P**ointer: Puntero al tope de la pila	**1, 2**
r30	$fp	**F**rame **P**ointer: Puntero al marco de pila en una subrutina actual	**2**
r31	$ra	**R**eturn **A**ddress : Dirección de retorno de subrutina actual	**1, 2**

1 – registros que usamos en los ejercicios de la práctica.

2 – registros que deben salvarse antes de ser utilizados en una subrutina

Winmips – Convención, ejemplo

```
.data
    result: .word 0
.text
    daddi r4, r0, 10
    daddi r5, r0, 20
    jal sumar
    sd r2, result(r0)
    halt

sumar:  dadd r2, r4, r5
        jr r31
```

Convención programa equivalente

```
.data
    result: .word 0
.text
    daddi $a0, $zero, 10
    daddi $a1, $zero, 20
    jal sumar
    sd $v0, result($zero)
    halt

sumar:  dadd $v0, $a0, $a1
        jr $ra
```

¿Cuál es mas legible?

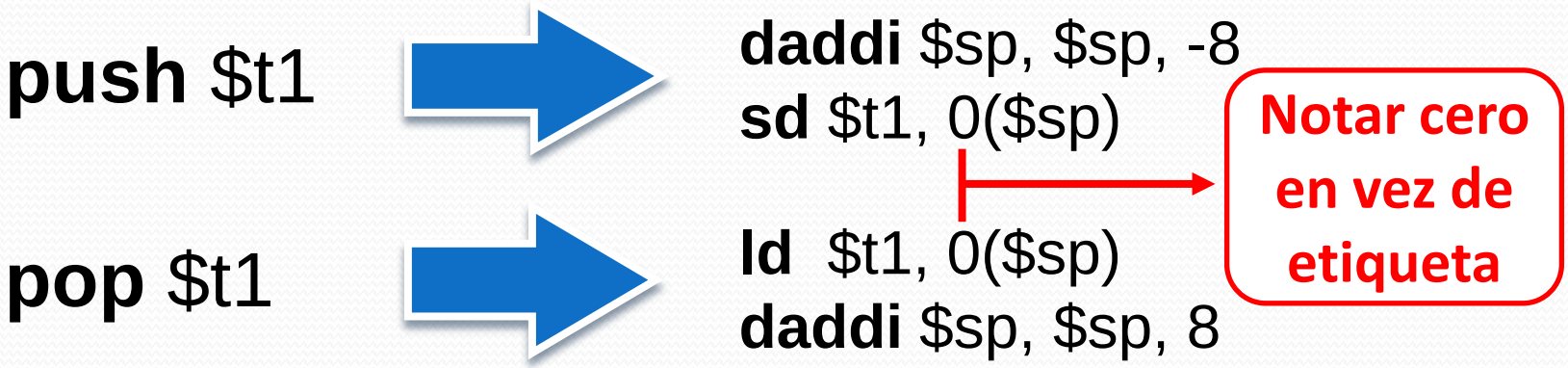
Convención – Registros

Preservación de registros:

- Una subrutina debe guardar los valores de los registros "garantizados" antes de modificarlos: $ra, $s0..$s7, $fp, $sp, $gp
- Si todas las subrutinas siguen la convención garantizamos la preservación de los valores luego de la invocación a otras subrutinas, evitando los efectos laterales.
- Cuando los registros que garantizan la preservación de los valores se acaban, es necesario un mecanismo adicional que permita salvar los valores. Necesitamos una pila.

Convención – pila

- No hay instrucciones específicas para manipular y mantener la pila.
- Todas las subrutinas usarán, por convención, el registro $sp (stack pointer - r29) como puntero de la pila.
- Por convención, se inicializa $sp con un valor alto de la memoria de programa y decrementa al apilar.
- Se definen las operaciones de “PUSH” y “POP” para apilar y desapilar registros en memoria de datos:



Convención – pila

"Apilado" consecutivo de varios (3) registros:

```
; Método 1
daddi   $sp, $sp, -8
sd      $s1, 0($sp)
daddi   $sp, $sp, -8
sd      $s2, 0($sp)
daddi   $sp, $sp, -8
sd      $s3, 0($sp)
```

```
; Método 2
daddi   $sp, $sp, -24 ; 8*3
sd      $s3, 0($sp)
sd      $s2, 8($sp)
sd      $s1, 16($sp)
```

```
; Método 3, acceder directamente en cualquier orden
daddi   $sp, $sp, -24 ; 8*3
sd      $s1, 0($sp)
sd      $s2, 8($sp)
sd      $s3, 16($sp)
```

Los mismos métodos pueden usarse para restaurar los registros

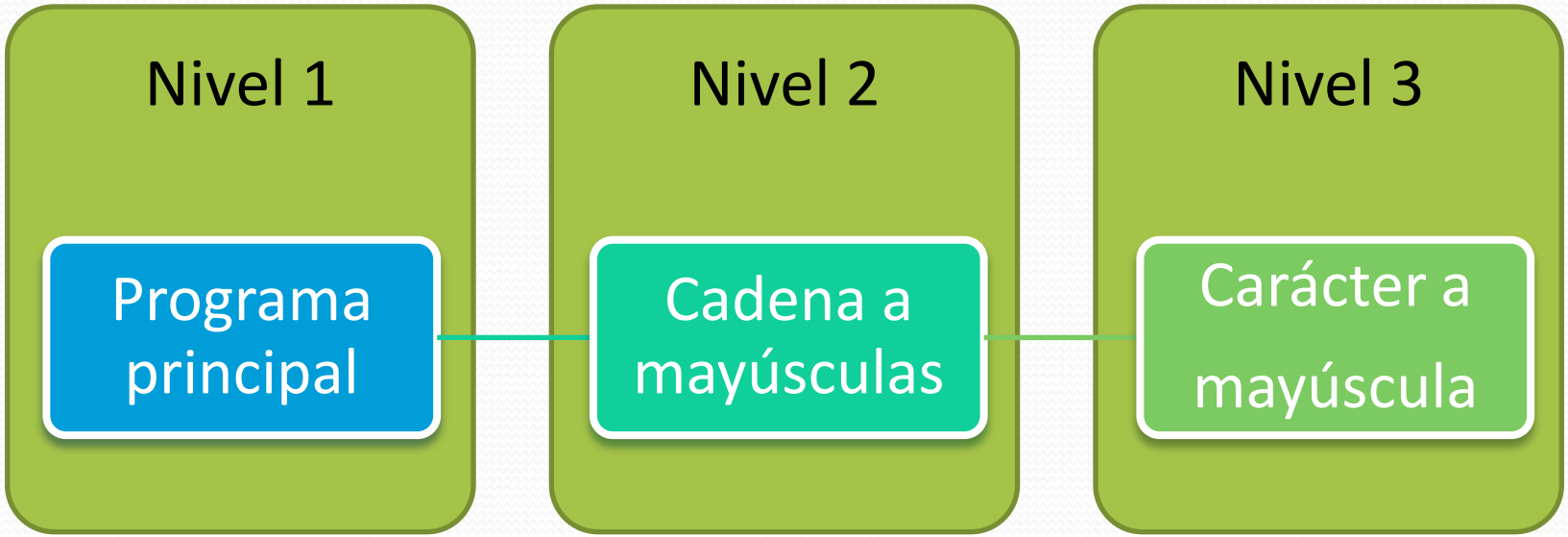
Convención – Subrutinas

Así, toda subrutina se dividirá en **tres partes**: el prólogo, el cuerpo y el epílogo.

```
subrut:   daddi $sp, $sp, -tamaño_frame   ; Reserva espacio en la pila
          sd $ra, 0($sp)                  ; Guarda la dirección de retorno
prólogo   sd $s0, 8($sp)                  ; Guarda el registro $s0
          sd $s1, 16($sp)                 ; Guarda el registro $s1
          :
          :
cuerpo    … instrucciones …               ; Cuerpo de la subrutina
          :
          :
epílogo   ld $ra, 0($sp)                  ; Recupera la dirección de retorno
          ld $s0, 8($sp)                  ; Recupera el registro $s0
          ld $s1, 16($sp)                 ; Recupera el registro $s1
          :
          :
          daddi $sp, $sp, tamaño_frame    ; Restaura el tope de la pila
          jr $ra                          ; Retorna
```

Subrutinas – ejemplo

Construir un programa que convierta una cadena de caracteres a mayúscula, utilizando una subrutina para convertir la cadena que a su vez utilice otra subrutina que convierta un carácter a mayúsculas.



Subrutinas – upcase

```
; Pasar un caracter a mayúscula.
; Parámetros:
; - $a0 -> caracter
; - $v0 -> caracter en mayúscula
; No se utiliza la pila porque no se usan registros que deban ser salvados
; Nota:  0x61 a 0x7A ➔ "a" a "z"          0x41 a 0x5A ➔ "A" a "Z"
;        0x61 ➔ 01100001₂ ➔ "a"           0x41 ➔ 01000001₂ ➔ "A"
upcase: DADD $v0, $a0, $zero   ; copia carácter en registro de salida
        SLTI $t0, $v0, 0x61    ; compara con 'a' minúscula
        BNEZ $t0, salir        ; no es un carácter en minúscula
        SLTI $t0, $v0, 0x7B    ; compara con el car sig a 'z' minúscula (z=7AH)
        BEQZ $t0, salir        ; no es un carácter en minúscula
        DADDI $v0, $v0, -0x20  ; pasa a minúscula (pone a '0' el 6to bit o bit5)
salir:  JR $ra                 ; retorna al programa principal
```

Subrutinas – upcaseStr

```
; Parametros: - $a0 -> inicio de cadena (puntero o referencia al primer carácter)
; Se utiliza la pila para guardar:
; - $ra -> porque se invoca a otra subrutina
; - $s0 -> para guardar la dirección de inicio de la cadena y recorrerla
```

```
upcaseStr:      DADDI $sp, $sp, -16     ; Reserva lugar en la pila -> 2 x 8
                SD $ra, 0($sp)
                SD $s0, 8($sp)
                DADD $s0, $a0, $zero    ; copia la dirección de la cadena
upcaseStrLOOP:  LBU $a0, 0($s0)         ; recupera el car actual como argumento para upcase
                BEQ $a0, $zero, upcaseStrFIN ; Si es el fin de la cadena, termina
                JAL upcase
                SB $v0, 0($s0)          ; Guarda el caracter procesado en la cadena
                DADDI $s0, $s0, 1       ; avanza al siguiente caracter
                J upcaseStrLOOP
upcaseStrFIN:   LD $ra, 0($sp)
                LD $s0, 8($sp)
                DADDI $sp, $sp, 16
                JR $ra
```

Subrutinas – Programa principal

El siguiente ejemplo sencillo convierte una cadena de caracteres a caracteres en mayúsculas:

```
.data
        cadena: .asciiz "Caza"
.text
        ; La pila comienza en el tope de la memoria de datos
        DADDI $sp, $0, 0x400
        ; Guarda como primer argumento para upcaseStr
        DADDI $a0, $0, cadena
        JAL upcaseStr
        …
        HALT
```