La utilización de la pila en MIPS

Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna

Marzo, 2016

Esquema de la lección

Introducción (I)

- Acabaremos este tema con una breve lección sobre la pila (stack) en MIPS.
- La pila es un método sistemático de preservar información.
- La analogía es la siguiente: cuando almacenamos un dato en una pila, lo amontonamos sobre el previo. Así, cada vez que extraemos algo de la pila, obtenemos la información en el orden inverso en la que fue almacenada (last input - first output).

Introducción (II)

- Muchos ensambladores tienen instrucciones específicas para manejar la pila:

ATENCIÓN ESTO NO ES MIPS:

```
1  push r0 #guarda el registro r0 en la pila
2  ....
3  pop r1 #obtengo el elemento en el top de la pila y lo guardo
       en el registro r1
```

- **Pero este no es el caso de MIPS. No hay instrucciones específicas para la pila.**

Introducción (III)

- La pila es útil para cualquier operación que necesite almacenar temporalmente datos.
- Pero hay algunos problemas relacionados con la programación de funciones para los que la pila se usa intensivamente:
  - **Problema:** Garantizar al usuario de la función que cierta información (por ejemplo, los registros salvados), no se van a modificar. La solución sería guardar en la pila esta información al principio de la función y restaurarla justo antes de volver al programa principal.
  - **Problema:** Se desea llamar desde una subrutina a otra subrutina. Si lo hacemos perderemos la dirección de retorno actual. Esto es un caso particular de lo anterior. La solución es guardar la dirección de retorno en la pila y restaurarla desde la pila antes de devolver el control al programa principal.

Introducción (IV). Ejemplo en código NO MIPS

ATENCIÓN ESTO NO ES MIPS:

```
4         jsr sub1 # salto a subrutina sub1. Se garantiza que los
              registros ra y rb no se modifican. La direccion de
              retorno en rs.
5         ...
6   sub1:
7         push rs
8         push rb
9         push ra  #hemos guardado la informacion a preservar
10        ...
11        jsr sub2 # otro salto a subrutina. Se asume que al volver
              de sub2 la pila esta en el mismo estado que tenia
              antes.
12        ... # tras volver de sub2 es seguro que rs ya ha cambiado
              de valor
13        pop ra
14        pop rb
15        pop rs #se restauran los valores. En particular la
              direccion de retorno en rs es restaurada
16        jr rs #saltamos a la direccion de retorno correcta
```

Elementos de la pila (I)

- Cuando un programa se carga en memoria, se crea un área de memoria denominada “pila”.
- La dirección **más alta** de la “pila” se denomina “el fondo de la pila”.
- Normalmente existe un registro, denominado “stack pointer” (SP) que al cargarse el programa en memoria, se inicializa al “fondo de la pila”.
- Cada vez que se hace una operación de push, el SP se decrementa para apuntar a la palabra inmediatamente por debajo en la memoria y entoces el valor se guarda en la posición apuntada por el SP. Es decir la pila “crece” de direcciones altas a direcciones bajas,
- La pila tiene un límite (stack limit). Si el SP llega a este límite, se produce una excepción denominada “stack overflow”.
- Cuando se hace una operación pop, se copia el valor apuntado por el SP en la memoria, y se incrementa SP para apuntar a la siguiente palabra.

Elementos de la pila (II)

Recuerda, en el significado de PUSH y POP es muy importante el orden de las operaciones:

- En un PUSH
  - Primero decrementar SP
  - Después guardar en la dirección SP.
- En un POP:
  - Primero cargar desde la dirección SP.
  - Después aumentar SP.

La pila en MIPS (I)

- MIPS proporciona el registro denominado "stack pointer" ($sp) (número 29) como única herramienta para controlar la pila.
- Supongamos que queremos hacer un push del registro $t0:

Ensamblador MIPS:

```
17  # Equivalente a PUSH en MIPS
18  addi $sp, $sp, -4 # Decrementamos el SP
19  sw $t0, 0($sp) # Guaramos valor en direccion apuntada por SP
```

- Supongamos que queremos hacer un pop sobre el registro $t1:

Ensamblador MIPS:

```
20  # Equivalente a POP en MIPS
21  lw $t1, 0($sp) # Cargamos desde direccion apuntada por SP
22  addi $sp, $sp, 4 # Incrementamos SP
```

La pila en MIPS (II)

- Supongamos que queremos hacer varios push: $t0, $t1 y $t2 en ese orden:

Ensamblador MIPS:

```
23  addi $sp, $sp, -12 # hacemos hueco a tres palabras
24  sw $s2, 0($sp)
25  sw $s1, 4($sp)
26  sw $s0, 8($sp)
```

- Ahora queremos restaurarlos los valores de los registros:

Ensamblador MIPS:

```
27  lw $s0, 8($sp) # Ojo, debemos cargarlos desde las
28  lw $s1, 4($sp) # direcciones correctas
29  lw $s2, 0($sp)
30  addi $sp, $sp, 12
```

Llamadas a subrutinas (I)

Uno de los usos más extendidos de la pila es guardar la dirección de retorno.

Ensamblador MIPS:

```
31        .....
32        jal et                # Llamada a la subrutina et
33        .....
34   et:  addi $sp, $sp, -4     # Entrada de la subrutina:
35        sw $ra, ($sp)         # Guardamos la dir. de retorno
36        .....
37        lw $ra, ($sp)         # Salida de subrutina:
38        addi $sp, $sp, 4      # Restauramos la dir. de retorno y la pila
39        jr $ra
```

Llamadas a subrutinas (II)

También sirve para respetar el convenio de los registros salvados.

Ensamblador MIPS:

```
40        .....
41        jal et              # Llamada a la subrutina et
42        .....
43   et:  addi $sp, $sp, -12  # Entrada de la subrutina:
44        sw $ra, ($sp)       # Guardamos la dir. de retorno
45        sw $s0, 4($sp)      # Guardamos los reg. salvados
46        sw $s1, 8($sp)      # que vamos a modificar en la subr.
47        .....
48        lw $ra, ($sp)       # Restauramos la direccion de retorno
49        lw $s0, 4($sp)      # Restauramos el valor de los reg. salvados
50        lw $s1, 8($sp)
51        addi $sp, $sp, 12    # Restauramos la pila
52        jr $ra               # Salimos de la subrutina
```

Llamadas a subrutinas (III)

Recuerda:

- Usa la pila para preservar sistemáticamente la dirección de retorno. De esta manera no perderás su valor si se hace otra llamada dentro de la subrutina.
- Usa la pila para preservar los registros salvados que vayas a modificar dentro de la subrutina. De esta manera, al restaurar sus valores, conseguirás cumplir con el convenio.
- Asegúrate de hacerlo sistemáticamente siempre justo al entrar y justo antes de salir de la subrutina (así el código es más legible y fácil de detectar errores).
- Revisa que la pila se mantiene equilibrada: lo que restas a $sp en la entrada, debe ser lo mismo que sumas a $sp en la salida.
- No te equivoques al restaurar cambiando las direcciones que asignaste a los registros.

El paso de argumentos y las variables locales

- Las subrutinas o funciones son una pieza clave para estructurar el código.
- En MIPS se establece un convenio para resolver algunos problemas más asociados al uso de las funciones:
  - Establecer un método estándar para comunicar argumentos a las funciones más allá de los 4 registros $a0, $a1, $a2 y $a3.
  - Usar la memoria como lugar de almacenamiento de variables locales, entendiendo por estas las que sólo son visibles en el cuerpo de la propia función y que son destruidas al salir de la función.

El marco de la pila (frame stack)

- Al principio del programa principal y de cada función se reserva una cantidad de memoria de la pila (stack frame) disminuyendo el valor del puntero de la pila.
- Los usos de esta memoria de pila reservada son:
  - Guardar las variables locales de la función.
  - Guardar los registros que es necesario almacenar: registros salvados, dirección de retorno y otros.
  - Guardar argumentos de funciones que se van a llamar desde esta función.
- Al final del programa principal o de la función que reserve un stack frame hay que restaurar el valor original al puntero de la pila, liberando así el stack frame.

Paso de argumentos por la pila

- Para entender el paso de argumentos de la pila hay que distinguir quién es la función **llamante** y cuál es la función **llamada**.
- En el paso de argumentos por la pila, la función llamante guarda en el top de la pila los argumentos que desea pasar, justo antes de la llamada.
- En el paso de argumentos por la pila, la función llamada lee los argumentos usando el puntero de la pila (se asume que al entrar en la función, los argumentos de la misma están ordenados en la parte superior de la pila).

Marco de pila en MIPS: reglas básicas (I)

- El llamado convenio de llamadas a funciones de MIPS establece exactamente lo que debe hacer el programa principal o cada función respecto a la pila, en particular como construir el **stack frame**.
- Al comenzar la función o el programa principal debe disminuirse el puntero de la pila en la cantidad suficiente para almacenar en la pila variables locales, dirección de retorno, registros salvados y todos los argumentos de cualquier función a la que se llame.
- Aunque el paso de argumentos se realiza mediante los registros $a0, $a1, $a2 y $a3 cuando no hay más de cuatro argumentos, hay que reservar en la pila espacio para estos argumentos también. La razón es que la función llamada podría usar estos huecos para almacenar temporalmente los registros de argumentos (se reserva para el mayor número de argumentos que una función llamada utilice).
- El área para variables locales debe comenzar siempre en una dirección múltiplo de 8. El tamaño del área para variables locales debe ser múltiplo de 8 (se pueden añadir a la pila palabras sin valores para rellenar y cumplir estas condiciones). Tal como se distribuye el stack frame esto permite que tras reservar el stack frame el puntero de la pila siempre esté en una dirección múltiplo de 8.

Marco de pila en MIPS: reglas básicas (II)

Distribución del marco de pila según el convenio (sólo se usará lo necesario):

Dirección	**Contenido**
$sp	Argumento 0
$sp + 4	Argumento 1
$sp + 8	Argumento 2
$sp + 12	Argumento 3
$sp + 16	Argumento 4
...	...
$sp + ...	Argumento n-1
$sp + ...	Registro salvado 0
...	...
$sp + ...	Registro salvado k-1
$sp + ...	Dirección de retorno
$sp + ...	Posible palabra de relleno (pad)
$sp + ... (debe ser múltiplo de 8)	Variable local 0
$sp + ...	Variable local 1
...	...
$sp + ...	Variable local m-1