Práctica 16. La pila

Objetivos

- Conocer cómo se utiliza la pila.

1. Introducción

La pila es una zona de memoria que se usa para almacenar información de nuestros programas de forma temporal. Un esquema de la memoria del RISC-V se muestra en la figura, aunque RISC-V no especifica un modelo de memoria 1.

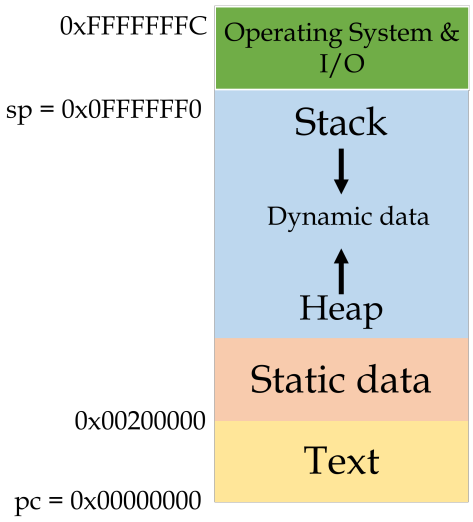

Figura 1: Estructura de la memoria

Una pila es una estructura de memoria de tipo LIFO (*Last In, First Out*), es decir, el último dato que se almacena es el primero que se obtiene al realizar una operación de extracción. Asociada a cualquier estructura de tipo pila existen dos operaciones básicas:

- **Apilar**: transferir datos hacia la pila (o si se prefiere, almacenar datos en la pila).
- **Desapilar**: transferir datos desde la pila (o "sacar" datos de la pila).

En la arquitectura RISC-V el segmento de pila es una zona de memoria que "crece" de direcciones superiores a direcciones inferiores, como se puede ver en la figura 1. En el simulador lo hace a partir de la dirección 0x0FFFFFFC. A medida que se van apilando datos se van ocupando direcciones más bajas. Esto contrasta, por ejemplo, con el funcionamiento del segmento de datos, donde cada dato que se almacena lo hace en direcciones crecientes.

Para gestionar el segmento de pila se dispone de un registro que no se ha utilizado hasta ahora. Se trata del registro `sp` (*stack pointer* o puntero de pila), que se encontrará en cada momento apuntando a la última dirección en uso dentro del segmento de la pila.

Las operaciones de apilar y desapilar se realizarán de la forma siguiente:

Apilar:

1. Se actualiza el puntero de pila restándole una cantidad igual al número de bytes del dato que se quiere apilar; es decir, "se reserva sitio" en la pila.
2. Se almacena el dato a partir de la dirección indicada por el puntero de pila mediante el modo de direccionamiento deseado.

Desapilar:

1. Se carga el dato a partir de la dirección indicada por el puntero de pila en el registro o registros deseados. Se tendrá en cuenta el tamaño del dato y se empleará un modo de direccionamiento adecuado.
2. Se actualiza el puntero de pila sumándole un entero igual al número de bytes del dato que ha sido leído, de modo que apunte así al dato siguiente.

Ejemplo de apilar y desapilar un byte, suponiendo que el dato a apilar está en `t0` y que cuando desapilamos lo guardamos en `t1`:

```
...
#Apilar
addi sp, sp, -1
sb t0, 0(sp)
...
#Desapilar
lb t1, 0(sp)
addi sp, sp, 1
...
```

Ejemplo de apilar y desapilar un `half`, suponiendo que el dato a apilar está en `t0` y que cuando desapilamos lo guardamos en `t1`:

```
...
#Apilar
addi sp, sp, -2
sh t0, 0(sp)
...
#Desapilar
lh t1, 0(sp)
addi sp, sp, 2
...
```

Ejemplo de apilar y desapilar un `word`, suponiendo que el dato a apilar está en `t0` y que cuando desapilamos lo guardamos en `t1`:

```
...
#Apilar
addi sp, sp, -4
sw t0, 0(sp)
...
#Desapilar
lw t1, 0(sp)
addi sp, sp, 4
```

La pila es un segmento de la memoria, por ello, a la hora de almacenar o extraer datos de la pila, al igual que ocurre al declarar datos en el segmento de datos, hay que tener en cuenta que los datos han de estar alineados.

Por tanto, para facilitar su uso, si trabajamos con datos de distintos tamaños apilaremos/desapilaremos siempre con el tamaño del más grande.

2. Las subrutinas y la pila

Cuando se utilizan subrutinas, en caso de necesitar más de ocho argumentos (los cuales irían por los registros `a0` al `a7`) o de devolver más de ocho resultados (los cuales irían en los mismos registros de `a0` al `a7`) hace falta utilizar la estructura de memoria conocida como pila.

El procedimiento será:

1. La función que llama a la subrutina, antes de la instrucción `jal` pasará los argumentos por `a0` al `a7` y el resto los apilará en la pila.
2. La subrutina, desapilará los argumentos de la pila (y el resto estará en los registros del `a0` al `a7`) y hará los cálculos.
3. La subrutina almacenará los resultados en los registros `a0` al `a7` y si hay más, los apilará en la pila.

4. La función que llamó a la subrutina (después de la instrucción `jal`) desapilará los resultados de la pila y utilizará los que vienen por los registros `a0` al `a7`.

2.1. Ejemplo de uso de la pila para pasar parámetros/resultados

El siguiente programa tiene una función que recibe 9 argumentos, 8 por registros y uno por la pila.

```
  .data

  .text
  .globl main
main:
  li a0, 1
  li a1, 2
  li a2, 3
  li a3, 4
  li a4, 5
  li a5, 6
  li a6, 7
  li a7, 8
  addi sp, sp, -1
  li t0, 9
  sb t0, 0(sp)
  jal ra, funcion

  li a7, 1
  ecall

  li a7, 10
  ecall

funcion:
  lb t0, 0(sp)
  addi sp, sp, 1
  add a0, a0, a1
  add a0, a0, a2
  add a0, a0, a3
  add a0, a0, a4
  add a0, a0, a5
  add a0, a0, a6
  add a0, a0, a7
  add a0, a0, t0
  ret
```

3. Subrutinas anidadas y recursivas

Se denominan subrutinas anidadas aquellas que contienen llamadas a otras subrutinas. Por ejemplo, una subrutina que indica si un número es primo que a su vez contiene una llamada a otra subrutina que sirve para indicar cuántos divisores tiene el número que se le pasa como parámetro.

Un caso especial de subrutinas anidadas son las subrutinas recursivas, que se llaman a sí mismas. Por ejemplo, para calcular el factorial de un número `n` se puede llamar a la propia subrutina con el parámetro `n-1`. Siempre tiene que haber un caso trivial que es el que garantiza que las llamadas no sean infinitas.

Si se quiere que una subrutina llame a otra (o bien que se llame a sí misma), mediante la instrucción `jal`, existe el problema de que automáticamente se modifica el contenido del registro `x1` (`ra`). Así, se pierde cualquier valor que este registro pudiera tener.

Si el programa principal llama a una subrutina A, que a su vez llama a otra subrutina B, se puede volver desde B a A mediante `jalr x0, 0(x1)`, pero al haberse sobreescrito el contenido de `x1` en la segunda llamada, será imposible volver desde A al programa principal.

La idea es que, antes de realizar una llamada desde una subrutina a otra se salve el contenido de `x1` en la pila. Así, las direcciones de retorno de las distintas llamadas anidadas/recursivas quedarán almacenadas a medida que se vayan produciendo. Para realizar los retornos, se desapilan las direcciones de retorno y se salta a éstas.

3.1. Ejemplo de uso de la pila para subrutinas anidadas

El siguiente programa tiene una función que calcula la suma del número de bits de un entero que valen 1 y una función que indica si dicho número es par o impar.

```
1   .data
2   cad: .string "Introduce un número: "
3   cad2: .string "El número de 1s es par"
4   cad3: .string "El número de 1s es impar"
5   .text
6   .globl main
7
8 main:
9   li a7, 4
10  la a0, cad
11  ecall
12
```

```
13   li a7, 5
14   ecall
15   # Argumento = a0 = número leído
16   jal ra, funcion
17   # Resultado = a0 = 0 si es par 1 si es impar
18   beq a0, zero, espar
19   la a0, cad3
20   j sigue
21 espar:
22   la a0, cad2
23   sigue:
24   li a7, 4
25   ecall
26
27   li a7, 10
28   ecall
29
30 funcion:
31   # Apilo ra porque hay una llamada anidada
32   addi sp, sp, -4
33   sw ra, 0(sp)
34   # Argumento = a0 = el mismo argumento que recibió funcion
35   jal ra, funcion2
36   # Resultado = a0 = suma de 1s
37   li t0, 2
38   rem a0, a0, t0 # Resto de dividir entre 2
39
40   # Desapilo ra para volver al main
41   lw ra, 0(sp)
42   addi sp, sp, 4
43   ret
44
45 funcion2:
46   li t0, 0  # Inicializo contador
47   bucle:
48   andi t2, a0, 1  # Saco el bit de menos peso (0 ó 1)
49   srli a0, a0, 1   # >> 1
50   add t0, t0, t2   # Sumo el 0 ó 1
51   bne a0, zero, bucle # Si el número tiene algún 1, sigo
52   mv a0, t0 #Resultado en a0
53   ret
```

4. Otros usos de la pila

Debido a su característica LIFO (*Last In, First Out*), la pila es útil, por ejemplo, para invertir el orden de los elementos de un vector o una cadena.

4.1. Ejemplo de uso de la pila para invertir una cadena

El siguiente programa lee una cadena y la invierte usando la pila.

```
  .data
cad: .string "Introduce una cadena: "
cad2: .zero 100
cad3: .zero 100
cad4: .string "La cadena introducida es: "

  .globl main
  .text
main:

  li a7, 4
  la a0, cad
  ecall

  li a7, 8  # Leo una cadena de 100 bytes y la guardo en cad2
  la a0, cad2
  li a1, 100
  ecall

  la t0, cad2 #Cadena origen (leída)
  la t1, cad3 #Cadena destino (invertida)
  addi sp, sp, -1 # Apilo el 0 en la pila para saber cuándo acabo
   de desapilar
  sb zero, 0(sp)

apilar:
  lb t2, 0(t0)  # Leo una letra de la cadean
  beq t2, zero, desapilar # Si es el \0 acabo de apilar
  addi sp, sp, -1 #Apilo la letra
  sb t2, 0(sp)
  addi t0, t0, 1  #Muevo el puntero de la cadena
  j apilar

desapilar:
  lb t2, 0(sp)    #Desapilo la letra
  addi sp, sp, 1
  sb t2, 0(t1)    #La guardo en la cadena de destino
  addi t1, t1, 1  #Actualizo el puntero de la cadena
  bne t2, zero, desapilar # Si no es el \0 sigo desapilando

  li a7, 4
  la a0, cad3
  ecall

```

```
44  li a7, 10
45  ecall
```

5. Ejercicio

5.1. Ejercicio

Realice un programa que pida un número por teclado (se debe controlar si el número insertado es positivo, en caso contrario, se mostrará un mensaje de error) e incluya una subrutina que indique si el número de divisores de un número es par o impar. Para calcular los divisores, dicha subrutina llamará a otra subrutina que calcula el número de divisores del número.

Aclaraciones:

- subrutina 1, devuelve 1 si es impar o 0 si es par
- subrutina 2, devuelve el número de divisores del número insertado
- la subrutina 1 llama a la subrutina 2 y luego según el valor devuelto, calcula qué tiene que devolver

5.2. Ejercicio

Realice un programa que lea 10 enteros y los almacene en memoria (reservar espacio con la directiva `space`) y a continuación, que guarde en memoria el vector al revés (en otra zona de memoria reservada con la directiva `space`). Utiliza la pila para cambiar el orden de los elementos del vector. NOTA: haz 3 versiones distintas del programa considerando enteros de tipo `byte`, `half` o `word`

5.3. Ejercicio

Considérese el siguiente fragmento de código que representa el segmento de datos:

```
1  .data
2  datos:
3  .byte 14, 23
4  .align 2
5  .word 47
6  .align 1
7  .half 9, 12, 15
```

```
8   .align 2
9   .zero 4
```

Escriba un programa que, utilizando los modos de direccionamiento realice las siguientes operaciones sin modificar el segmento de datos:

- Sumar el `word` con valor `47`, con el `half` de valor `15` y guardarlo en el `space` como dato de tipo `word`, utilizando la etiqueta `datos`.
- Leer el `byte` que vale 23, sumarle el `half` que vale 12 y guardarlo en el `half` que vale 15.