Clase 14 - RISC - V - Parte 4

IIC2343 - Arquitectura de Computadores

Profesor:
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl

Resumen de la clase pasada

Instrucción AND / OR / XOR

- Para las operaciones de AND, OR y XOR, al igual que la suma, se pueden realizar entre registros y literales.

AND:

and_entre_registros:

```
and t0, t1, t2
```

and_con_literal:

```
andi t0,t1,3
```

OR:

or_entre_registros:

```
or t0, t1, t2
```

or_con_literal:

```
ori t0,t1,5
```

XOR:

xor_entre_registros:

```
xor t0, t1, t2
```

xor_con_literal:

```
xori t0,t1,10
```

Instrucción SHL / SHR

- RISC-V deja realizar shifts lógicos y aritméticos
- El segundo operando indica la cantidad de shifts que se deben realizar

SHL lógico:

SHL_logico_con_regis:

```
sll t0, t1, t2
```

SHL_logico_con_literal:

```
slli t0,t1,3
```

SHR lógico:

SHR_logico_con_regis:

```
srl t0, t1, t2
```

SHR_logico_con_literal:

```
srli t0,t1,5
```

SHR aritmetico:

SHR_arit_con_regis:

```
sra t0, t1, t2
```

xor_con_literal:

```
srai t0,t1,10
```

Subrutinas: CALL dir

- Para realizar un CALL se debe hacer un “*jump and link*”.
- Esto corresponde a saltar a la dirección indicada almacenando el valor del registro PC en el registro especificado
- Por convención, la dirección de retorno debe ser almacenado en el registro **ra**
- La dirección de retorno se almacena en un registro, todavía no en el stack

call_por_label:

```
jal ra,dir
```

call_por_registro:

```
la t0,label
jalr ra,0(t0)
```

Subrutinas: RET

- Para realizar un RET se debe hacer un “*jump and link register*” guardando la dirección de retorno en zero
- Saltamos a la dirección almacenada en el registro **ra**

return:

```
jalr zero,0(ra)
```

Ejercicio: Duplicar los elementos de un arreglo

- **Objetivo**
- Simular la operación de duplicar cada valor de un arreglo almacenado en memoria
- **Idea Principal**
    - Duplicar un número equivale a sumar consigo mismo
- **Ejemplo:**

[1, 2, 3, 4, 5] ⇒ [2, 4, 6, 8, 10]

Ejercicio Multiplicación con Suma Sucesiva

```
# Duplicar los elementos de un arreglo
.data
arr: .word 1, 2, 3, 4, 5   # Arreglo original
n:   .word 5               # Tamaño del arreglo
.text
main:
    la t0, arr        # t0 = dirección del primer elemento del arreglo
    lw t1, n          # t1 = cantidad de elementos
    addi t2, zero, 0  # i = 0 (contador)
Loop: # Si i >= n, salir del bucle
    bge t2, t1, end
    lw t3, 0(t0) # t3 = arr[i] / Leer el valor actual del arreglo
    add t3, t3, t3    # t3 = arr[i] + arr[i] / Duplicar el valor
    sw t3, 0(t0) # Guardar el nuevo valor en el arreglo
    add a0, t3, zero # Damos el valor duplicado
    addi a7, zero, 1 # Código 1: PrintInt (imprime número entero)
    ecall
    addi a0, zero, 10 # Damos el valor 10 de la tabla de ASCII (\n)
    addi a7, zero, 11 # Código 11: Printchar (imprime un caracter ASCII)
    ecall
    # Avanzar al siguiente elemento
    addi t0, t0, 4    # mover puntero 4 bytes (siguiente entero)
    addi t2, t2, 1    # i = i + 1
    beq zero, zero, loop
end:# Terminar el programa
    addi a0, zero, 0  # Si el código de término es 0 es porque no hubo errores
    addi a7, zero, 10 # Termina el programa
    ecall
```

¿Dudas?

Instrucción PUSH?

- No tenemos esta instrucción en RISC-V
- Replicamos su funcionamiento, es decir disminuir el *stack pointer* y guardar en un registro

Formato:

push_de_un_registro:

```
addi sp, sp,-4
sw sp, 0(sp)
```

push_de_dos_registros:

```
addi sp, sp,-8
sw t0, 0(sp)
sw t1, 4(sp)
```

Instrucción POP?

- No tenemos esta instrucción en RISC-V
- Replicamos su funcionamiento, es decir aumentar el *stack pointer* y cargar el valor desde el *stack* hacia un registro

Formato:

pop_de_un_registro:

```
lw t0, 0(sp)
addi sp, sp,4
```

pop_de_dos_registros:

```
lw t0, 0(sp)
lw t1, 4(sp)
addi sp, sp,8
```

Instrucción POP?

- No tenemos esta instrucción en RISC-V
- Replicamos su funcionamiento, es decir aumentar el *stack pointer* y cargar el valor desde el *stack* hacia un registro

Formato:

pop_de_un_registro:

```
lw t0, 0(sp)
addi sp, sp,4
```

pop_de_dos_registros:

```
lw t0, 0(sp)
lw t1, 4(sp)
addi sp, sp,8
```

Ejemplo de subrutina

```
.text
main:
        beq zero, zero, _main

subrutina:
        add a0, a0, a1
        jalr zero, 0(ra)
_main:
        addi sp, sp, -4
        sw a0, 0(sp)
        jal ra, subrutina
        lw a0, 0(sp)
        addi sp, sp, 4
```

Subrutinas: Llamadas anidadas

- Necesitamos respaldar la dirección de retorno en el stack
- Antes de hacer una llamada recursiva, hacemos **PUSH** del registro **ra**
- Después del llamado, hacemos **POP** al registro **ra**

```
addi sp, sp, -4
sw ra, 0(sp)
jal ra, subrutina
lw ra, 0(sp)
addi sp, sp, 4
```

¿Dudas?

Convención de llamadas RISC-V

- Se define una convención de llamada para estandarizar el uso de registros y subrutinas
- Usar los nombres simbólicos (**zero, ra, sp, a0–a7, s0–s11, t0–t6)** en lugar de **x0, x1**, etc
- Uso principal de registros:
  - **ra**: dirección de retorno
  - **a2–a7**: argumentos de entrada
  - **a0–a1**: valores de retorno
  - **sp**: tope del stack

Convención de llamadas RISC-V - Preservación de registros

- Algunos registros deben mantener su valor después de una subrutina (**s0–s11**)
- Si una subrutina usa registros **s**, ella debe respaldarlos (**callee-save**).
- Si se usan registros **t** o **a**, quien llama debe respaldarlos (**caller-save**).

Callee-save: sp, s0–s11

Caller-save: ra, t0–t6, a0–a7

EJERCICIO !!

Suma recursiva de un arreglo con subrutinas anidadas

Objetivo

- Calcule la suma total de los elementos de un arreglo de **forma recursiva**

Se implementarán subrutinas para:

- Calcular la suma del arreglo **(suma)**
- Imprimir un entero **(imprimir)**
- Terminar el programa **(terminar)**

Suma recursiva de un arreglo con subrutinas anidadas

Idea Principal

- La suma de los n elementos de un arreglo puede expresarse recursivamente como:

suma(arr, n) =

0 **si n == 0**

arr[n-1] + suma(arr, n-1) **en otro caso**

- Además, separaremos las acciones de imprimir y terminar como subrutinas independientes, permitiendo ver cómo se anidan llamadas

  (main ➜ suma ➜ imprimir ➜ terminar).

Suma recursiva de un arreglo con subrutinas anidadas

Idea Principal

- La suma de los n elementos de un arreglo puede expresarse recursivamente como:

suma(arr, n) =

0 **si n == 0**

arr[n-1] + suma(arr, n-1) **en otro caso**

- Además, separaremos las acciones de imprimir y terminar como subrutinas independientes, permitiendo ver cómo se anidan llamadas

  (main ➜ suma ➜ imprimir ➜ terminar).

¿Dudas?

Clase 14 - RISC - V - Parte 4

IIC2343 - Arquitectura de Computadores

Profesor:
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl