Clase 12 - RISC - V - Parte 2

IIC2343 - Arquitectura de Computadores

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

Resumen de la clase pasada

RISC - V

- Tiene 32 registros de propósito general, cada uno de 32 bits.
- Direcciones de memoria de 32 bits. Las palabras de memoria son de 32 bits (4 bytes), pero l**as direcciones son por cada byte**
- Las instrucciones de operaciones aritméticas/lógicas tienen la estructura: palabra clave, registro de destino
- El r**egistro de destino** es donde se va a almacenar el valor de la operación.

Imagen Referencia: Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

RISC - V

Registro	Nombre	Uso
x0	zero	**Registro *zero***. Siempre almacena el valor 0, no se puede escribir.
x1	ra	***Return Address***. Se usa para guardar la dirección de retorno de una subrutina.
x2	sp	***Stack Pointer***. Almacena la dirección del último elemento del *stack*.
x3	gp	***Global Pointer***.
x4	tp	***Thread Pointer***.
x5-x7	t0 – t2	Registros temporales. Pueden perder su valor al llamar una subrutina.
x8-x9	s0 – s1	Registros guardados. Preservan su valor al llamar una subrutina.
x10-x11	a0 – a1	Argumentos de funciones / Valores de retorno.
x12-x17	a2 – a7	Argumentos de funciones.
x18-x27	s2 – s11	Registros guardados. Preservan su valor al llamar una subrutina.
x28-x31	t3 – t6	Registros temporales. Pueden perder su valor al llamar una subrutina.

Imagen Referencia: Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

¿Dudas?

La sección .data

- Aquí se definen variables y arreglos
- Solo usamos `.byte` y `.word`
- `.byte`:
    - 1 byte
    - valores pequeños (0–255)
    - `.byte` 10
- `.word`:
    - 4 byte
    - enteros normales
    - `.word` 123

```
1  a: .byte 10        # ocupa 1 byte
2  b: .word 5000      # ocupa 4 bytes
3  nums: .word 1,2,3,4
4
5
```

La sección .text

- Aquí va el código ejecutable
- Comienza con la etiqueta `main:`
- Cada línea es una instrucción del procesador
- Trabajamos con **registros**, no con variables directas

Instrucción ecall

- Sirve para llamar al sistema y hacer acciones como imprimir números o terminar el programa
- Lo que hace depende del valor en a7 (**esto por convención en RARS**)

```
1  lw a0, res        # valor a imprimir
2  addi a7, zero, 7 # código 7 = imprimir entero
3  ecall
4
5  addi a7, zero, 10 # código 10 = salir
6  ecall
7
```

¿Dudas?

Instrucción lw

- Carga un número desde memoria a un registro

Formato:

leer_memoria_directo:

```
lw t0, dir
```

leer_memoria_indirecto:

```
lw t0, (t1)
```

Leer_memoria_indirecto con offset:

```
lw t0, 0(t1)
```

```
1  lw t0, x       # carga el valor guardado en "x"
2  lw t1, 0(t0)   # carga desde la
3                 # dirección apuntada por t0
4
5
6
7
```

Instrucción sw

- Guarda un número desde un registro hacia memoria
- Formato:

escribir_memoria_directo:

```
sw t0, dir, t2
```

escribir_memoria_indirecto:

```
sw t0, (t1)
```

escribir_memoria_indirecto con offset:

```
sw t0, 0(t1)
```

```
# guardar e imprimir
end:
sw  t2,res,t3 # t2 = Mem[res] , t3 = res
sw  t2,(t3)   # t2 = Mem[t3]
sw  t2,0(t3)   # t2 = Mem[t3 + 0]
```

Instrucción beq (branch if equal)

- Salta a una etiqueta si dos registros son iguales.
- Se usa para crear bucles o condiciones.
- Formato:

```
beq reg1, reg2, etiqueta
```

```
1
2  beq t1, zero, fin   # si t1 == 0, salta a "fin"
3
4
5
```

¿Dudas?

EJERCICIO !!

Ejercicio Multiplicación con Suma Sucesiva

- **Objetivo**
- Simular la operación de multiplicación mediante sumas repetidas usando instrucciones básicas de RISC - V
- **Idea Principal**
    - Multiplicar x * y equivale a sumar x a sí mismo y veces.
- **Ejemplo:**
- $4 \times 3 = 4 + 4 + 4 = 12$

Ejercicio Multiplicación con Suma Sucesiva

```
# Multiplicación - sucesiva
.data
x: .word 4
y: .word 3
res: .word 0
.text
main:
# cargar los valores
lw t0, x
lw t1, y
lw t2, res
# bucle
loop:
beq t1, zero, end # condicion
add t2, t2, t0    # ADD es REG,REG,REG
addi t1, t1,-1
beq zero, zero, loop
# guardar e imprimir
end:
sw  t2,res,t3 # t2 = Mem[res] , t3 = res
sw  t2,(t3)   # t2 = Mem[t3]
sw  t2,0(t3)   # t2 = Mem[t3 + 0]
lw a0, res
addi a7, zero,1  # preparar el ECALL
ecall
```

¿Dudas?

Clase 12 - RISC - V - Parte 2

IIC2343 - Arquitectura de Computadores

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