Clase 13 - RISC - V - Parte 3

IIC2343 - Arquitectura de Computadores

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

Resumen de la clase pasada

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
```

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?

Instrucción SUB

- La resta solamente se puede realizar entre dos registros, a diferencia de la suma
- **No existe la resta con literal** en RISC-V, pero se puede hacer la suma con números negativos

Formato:

resta_entre_registros:

```
sub t0, t1, t2
```

resta_con_literal:

```
addi t0,t1,-2
```

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
```

Instrucción Bonus (Extensión M)

- RISC-V tiene diversas extensiones (más instrucciones)
- Una de ellas es la extensión M, que pueden usar en evaluaciones

multiplicacion:

```
mul t0, t1, t2
```

division

```
div t0, t1, t2
```

resto

```
rem t0, t1, t2
```

division_sin_signo:

```
divu t0, t1, t2
```

resto_sin_signo:

```
divu t0, t1, t2
```

¿Dudas?

Saltos condicionales

- Vamos a tener cuatro instrucciones de saltos condicionales
- Los saltos son encargados de realizar la comparación entre los registros
- No se tiene una instrucción de comparación anteriormente (CMP)

branch_if_equal:

```
beq t0,t1,dir
```

branch_if_less_than:

```
blt t0,t1,dir
```

branch_if_less_than_unsigned:

```
bltu t0,t1,dir
```

branch_if_not_equal:

```
bne t0,t1,dir
```

branch_if_greater_equal:

```
bge t0,t1,dir
```

branch_if_gt_equal_unsigned:

```
bgeu t0,t1,dir
```

RISC – V : Subrutinas y stack

- Vamos a tener cuatro instrucciones de saltos condicionales
- Los saltos son encargados de realizar la comparación entre los registros
- No se tiene una instrucción de comparación anteriormente (CMP)

branch_if_equal:

```
beq t0,t1,dir
```

branch_if_less_than:

```
blt t0,t1,dir
```

branch_if_less_than_unsigned:

```
bltu t0,t1,dir
```

branch_if_not_equal:

```
bne t0,t1,dir
```

branch_if_greater_equal:

```
bge t0,t1,dir
```

branch_if_gt_equal_unsigned:

```
bgeu t0,t1,dir
```

RISC – V : Subrutinas y stack

- El manejo del stack es algo más complicado que nuestro computador
- No existen instrucciones de PUSH y POP
- Los llamados a las subrutinas **no almacenan la dirección** de retorno automáticamente en el stack
- Nosotros somos los encargados de mover el stack pointer (SP) para no sobreescribir la memoria

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 !!

¿Dudas?

Clase 13 - RISC - V - Parte 3

IIC2343 - Arquitectura de Computadores

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