Auxiliar 7 - Assembler RISC-V

Otoño 2025

Profesores: Alexandra Ibarra
Luis Mateu
Rodrigo Urrea

Auxiliares: Luciano Márquez
Tomás Vergara

Resumen

Registros

Registro	Descripción
`zero`	`0`
`a0, ... , a7`	Argumentos de una función (`a0` almacena el valor que retorna una función)
`t0, ... , t6`	Valores temporales (no se preservan entre llamados)
`s0, ... , s11`	Registros resguardados (al retornar se deben dejar intactos)
`ra`	Dirección de retorno de una función
`sp`	Puntero a la pila de memoria
`gp`	Puntero a las variables globales
`tp`	Puntero a las variables locales de un thread

Instrucciones

- Operaciones

Instrucción		**Descripción**
`add`	`rd,r1,r2`	Suma los registros `r1` y `r2` dejando el resultado en `rd`
`addi`	`rd,d1,im`	Suma el registro `r1` con el valor inmediato `im` dejando el resultado en `rd`
`sub`	`rd,r1,r2`	Resta los registros `r1` y `r2` dejando el resultado en `rd`
`lui`	`rd,im`	Carga los 20 bits más significativos de `im` en el registro `rd`
`mul`	`rd,r1,r2`	Multiplica los registros `r1` y `r2` dejando el resultado en `rd`
`div`	`rd,r1,r2`	Divide los registros `r1` y `r2` dejando el resultado en `rd`
`rem`	`rd,r1,r2`	Divide los registros `r1` y `r2` dejando el resto de la operación en `rd`
`xor`	`rd,r1,r2`	«O exclusivo» lógico entre los bits de `r1` y `r2` dejando el resultado en `rd`
`xori`	`rd,r1,im`	«O exclusivo» lógico entre los bits de `r1` y el inmediato `im` dejando el resultado en `rd`
`or`	`rd,r1,r2`	«O» lógico entre los bits de `r1` y `r2` dejando el resultado en `rd`
`ori`	`rd,r1,im`	«O « lógico entre los bits de `r1` y el valor inmediato `im` dejando el resultado en `rd`
`and`	`rd,r1,r2`	«Y exclusivo» lógico entre los bits de `r1` y `r2` dejando el resultado en `rd`
`andi`	`rd,r1,im`	«Y» lógico entre los bits de `r1` y el valor inmediato `im` dejando el resultado en `rd`

`sll rd,r1,r2`	*Shift left* de `r1` en `r2` bits dejando el resultado en `rd`
`slli rd,r1,r2`	*Shift left* de `r1` en `im` (valor inmediato) bits dejando el resultado en `rd`
`srl rd,r1,r2`	*Shift right* de `r1` en `r2` bits dejando el resultado en `rd`
`srli rd,r1,r2`	*Shift right* de `r1` en `im` (valor inmediato) bits dejando el resultado en `rd`
`sra rd,r1,r2`	*Shift right* aritmético de `r1` en `r2` bits dejando el resultado en `rd`
`srai rd,r1,r2`	*Shift right* aritmético de `r1` en `im` (valor inmediato) bits dejando el resultado en `rd`

- Accesos a memoria

Instrucción	Descripción
`lb rd,im(r1)`	Carga un byte de la memoria ubicada en `r1` con offset `im` en el registro `rd`
`lh rd,im(r1)`	Carga la mitad de un word de la memoria ubicada en `r1` con offset `r1` en el registro `rd`
`lw rd, im(r1)`	Carga un word de la memoria ubicada en `r1` con ofset `im` en el registro `rd`
`lbu rd, im(r1))`	Carga un byte sin signo de la memoria ubicada en `r1` con offset `im` en el registro `rd`
`lhu rd,im(r1)`	Carga la mitad de un word sin signo de la memoria ubicada en `r1` con offset `im` en el registro `rd`
`sb rd,im(r1)`	Guarda un byte de la memoria ubicada en `r1` con offset `im` en el registro `rd`
`sh rd,im(r1)`	Guarda la mitad de un word de la memoria ubicada en `r1` con offset `im` en el registro rd
`sw rd,im(r1)`	Guarda un word de la memoria ubicada en `r1` con offset `im` en el registro `rd`

- Saltos

Instrucción	Descripción
`.label:`	Crea una etiqueta con el nombre `label` (no es una dirección en si misma)
`beq r1,r2,.label`	Salta a `label` si `r1` es igual a `r2`
`bne r1,r2,.label`	Salta a `label` si `r1` no es igual a `r2`
`blt r1,r2,.label`	Salta a `label` si `r1` es menor que `r2`
`bge r1,r2,.label`	Salta a `label` si `r1` es mayor o igual a `r2`
`bltu r1,r2,.label`	Salta a `label` si `r1` es menor que `r2`, considerando `r1` y `r2` sin signo
`bgeu r1,r2,.label`	Salta a `label` si `r1` es mayor o igual `r2`, considerando `r1` y `r2` sin signo
`j .label`	Salta a `label` si `r1` es igual a `r2`

- Llamadas a funciones

Instrucción	Descripción
`call label`	Llama a la función `label`, no es lo mismo que hacer un salto simple
`ret`	Retorna de la función volviendo a la instrucción siguiente, usando `ra` y devolviendo el valor almacenado en `a0`

Preguntas

P1. RISC-V a C

Traduzca el siguiente código en RISC-V a C:

RISC-V

```
1           .text
2           .align 2
3           .globl start
4           .type start, @function
5   start:
6           lui   a5, %hi(n)
7           lw    a5, %lo(n)(a5)
8           li    a4, 1
9           li    a0, 1
10          ble   a5, a4, .L1
11  L3:
12          mul   a0, a0, a5
13          addi  a5, a5, -1
14          bne   a5, zero, .L3
15  L1:
16          ret
17          .globl n
18          .section .sdata, "aw"
19          .align 2
20          .type n, @object
21          .size n, 4
22  n:
23          .word 4
```

P2 (T3 Otoño 2021) Ordenamiento lexicográfico

La funcion `sort` está programada en assembler RISC-V en el archivo `sort-rv.s`. Esta función ordena lexicográficamente un areglo de strings usando un algoritmo ridículamente ineficiente.

El archivo `sort-rv-apnom.s` es una copia de `sort-rv-s`. Modifique la función `sort` en `sort-rv-apnom.s` de modo que se ordene el arreglo primero por apellido, y si los apellidos son iguales, entonces por nombre. El programa de prueba invoca `sort` y muestra en pantalla el resultado del ordenamiento. La salida ordenada en su solución debe ser:

```
maria fernandez
monica fernandez
vero fernandez
ana gonzalez
diego gonzalez
pedro gonzalez
tatiana jerez
alberto perez
jose perez
juan perez
```

- El archivo `sort-c.c` es la versión de C de `sort`. Compile y ejecute esta versión (no pasa el test de prueba).

```
$ make sort-c
$ qemu-riscv32 sort-c
```

- Programe primero una versión en C de lo pedido en el archivo `sort-c-apnom.c`. Revise que funcione correctamente con:

```
$ make sort-c-apnom
$ qemu-riscv32 sort-c-apnom
```

- Reprograme en assembler la función `sort` en el archivo `sort-rv-apnom.s`. Compile y ejecute con:

```
$ make sort-rv-apnom
$ qemu-riscv32 sort-rv-apnom
```

- Use ddd para entender y depurar su tarea. Seleccione el menú `View` ⟶ `Machine code window` para ver el assembler.