Auxiliar 7

Assembler Risc-V

Profesor: Luis Mateu

Auxiliares: Gerard Cathalifaud
Vicente González
Joaquín López
Rodrigo Urrea

Semestre: Primavera 2023

1. Resumen

1.1. Registros

Registro	Descripción
`zero`	0
`a0, ... , a7`	Argumento de una función[1]
`t0, ... , t6`	Valores temporales (No se preservan entre llamados!)
`s0, ... , s11`	Registros resguardados (Se deben dejar intactos antes de retornar!)
`ra`	Dirección de retorno de una función, para saber "donde" continuar
`sp`	Puntero a la pila de memoria
`gp`	Puntero a las variables globales
`tp`	Puntero a las variables locales de un *thread*

1.2. Instrucciones

- Operaciones

Instrucción	Descripción
`add rd,r1,r2`	Suma los registros `r1` y `r2` dejando el resultado en `rd`
`addi rd,r1,im`	Suma el registro `r1` con el valor inmediato `imm` 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 los registros `r1` y `r2` dejando el resultado en `rd`
`xori rd,r1,im`	"O exclusivo" lógico entre los bits del registro `r1` y el inmediato `im` dejando el resultado en `rd`
`or rd,r1,r2`	"O" lógico entre los bits de los registros `r1` y `r2` dejando el resultado en `rd`
`ori rd,r1,im`	"O" lógico entre los bits del registro `r1` y el inmediato `im` dejando el resultado en `rd`
`and rd,r1,r2`	"Y" lógico entre los bits de los registros `r1` y `r2` dejando el resultado en `rd`
`andi rd,r1,im`	"Y" lógico entre los bits del registro `r1` y el inmediato `im` dejando el resultado en `rd`

[1] `a0` es la usado para guardar el valor retornado por una función

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

- Accesos a memoria[2]

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 `im` en el registro `rd`
`lw`	`rd,im(r1)`	Carga un word de la memoria ubicada en `r1` con offset `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 instrucción 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` es no 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 que `r2`
`bltu`	`r1,r2,.label`	Salta a `label` si `r1` es menor que `r2`, considerando que ambos no tienen signo
`bgeu`	`r1,r2,.label`	Salta a `label` si `r1` es mayor o igual que `r2`, considerando que ambos no tienen signo
`j`	`.label`	Salto incondicional a `label`

- Llamada 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`)

1.3. Herramientas

- Compilar (generalmente se pone en un Makefile)[3]:

```
path/riscv64-unknown-elf-gcc -march=rv32im -mabi=ilp32 archivo.c -o archivo
```

Para las tareas:

```
make archivo
```

- Ejecutar

```
qemu-riscv32 binario
```

[2]Un byte son 4 bits, una word son 32/64 bits dependiendoe de la arquitectura

- Generar assembler[3]:

  ```
  path/riscv64-unknown-elf-gcc -march=rv32im -mabi=ilp32 -S archivo.c -o archivo.s
  ```

  Para las tareas:

  ```
  make archivo.s
  ```

2. Preguntas

P1. Traduzca el siguiente programa en Assembly RISC-V a código C:

```
        .text
        .align 8
        .globl start
        .type start, @function
start:
        lui a5, %hi(n)
        lw a5, %lo(n)(a5)
        li a4, 1
        li a0, 1
        ble a5, a4, .L1

.L3:
        mul a0, a0, a5
        addi a5, a5, -1
        bne a5, zero, .L3
.L1:
        ret
        .globl n
        .section .sdata, "aw"
        .align 2
        .type n, @object
        .size n, 4
  n:
        .word 4
```

[3]Donde `path` es `/opt/riscv/bin`

P2. **(T3 Otoño 2021)**: Ordenamiento lexicográfico

La función `sort` está programada en assembler Risc-V en el archivo `sort-rv.s`. Esta función ordena lexicográficamente un arreglo 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 en C de `sort`. Compile y ejecute esa 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 fucione 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. Seleccion el menú View → Machine code window para ver el assembler.