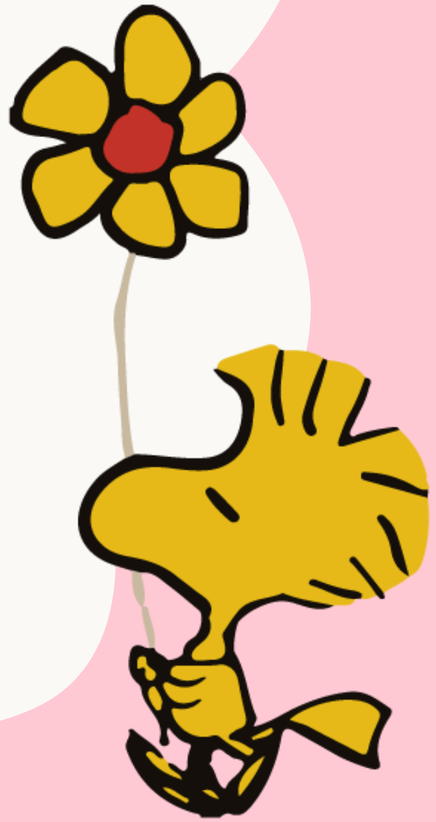
Ayudantía 6 :RISC-V

IIC2343

Daniela Ríos



REGISTROS

Mini Resumen: basado en material hecho por Dafne Arriagada (2023-2)

Name (código)	qué es?	qué hace?
zero		Almacena valor 0 y no cambia. Ignora las escrituras.
ra	return adress	Almacena la **dirección de retorno** de las subrutinas
sp		apunta al último elemento almacenado.
gp		apunta al segmento de memoria donde se almacenan las variables globales
tp	thead pointer	apunta al segmento de memoria donde se almacenan las variables de un thread para aplicaciones de múltiples threads
t0-t6	temporary registers	**Pierden** su valor entre llamados de subrutinas
s0-s11	saved registers	**Preservan** su valor entre llamados de subrutinas
a0-a7 **a0-a1**	(1)argument registers (2)return value registers	1. Registros para **argumentos de subrutinas**. 2. También se utilizan para almacenar **valores de retorno**

INSTRUCCIONES

Aritméticas

Mnemotecnia	Instrucción	Tipo	Descripción
`ADD rd, rs1, rs2`	**Adición**	**R**	**`rd ← rs1 + rs2`**
`SUB rd, rs1, rs2`	**Sustracción**	**R**	**`rd ← rs1 - rs2`**
`ADDI rd, rs1, imm12`	**Adición de literal**	**I**	**`rd ← rs1 + imm12`**
`SLT rd, rs1, rs2`	Configurar "menor a"	R	`rd ← rs1 < rs2 ? 1 : 0`
`SLTI rd, rs1, rs2`	Configurar "menor a" literal	I	`rd ← rs1 < imm12 ? 1 : 0`
`SLTU rd, rs1, rs2`	Configurar "menor a" sin signo	R	`rd ← rs1 < rs2 ? 1 : 0`
`SLTIU rd, rs1, imm12`	Configurar "menor a" literal sin signo	I	`rd ← rs1 < imm12 ? 1 : 0`
`LUI rd, imm20`	Cargar literal "superior" (20 bits)	U	`rd ← imm20 << 12 (SHL 12)`
`AUIP rd, imm20`	Sumar literal "superior" a PC (20 bits)	U	`rd ← PC + imm20 << 12 (SHL 12)`

*** Para restar un literal, usamos `ADDI` con un literal negativo.**

Lógicas

Mnemotecnia	Instrucción	Tipo	Descripción
`AND rd, rs1, rs2`	Operación **AND**	**R**	`rd ← rs1 & rs2`
`OR rd, rs1, rs2`	Operación **OR**	**R**	`rd ← rs1 | rs2`
`XOR rd, rs1, rs2`	Operación **XOR**	**R**	`rd ← rs1 ^ rs2`
`ANDI rd, rs1, imm12`	Operación **AND** con literal	**I**	`rd ← rs1 & imm12`
`ORI rd, rs1, imm12`	Operación **OR** con literal	**I**	`rd ← rs1 | imm12`
`XORI rd, rs1, imm12`	Operación **XOR** con literal	**I**	`rd ← rs1 ^ imm12`

Mnemotecnia	Instrucción	Tipo	Descripción
`SLL rd, rs1, rs2`	Operación ***shift left*** lógico	**R**	`rd ← rs1 << rs2`
`SRL rd, rs1, rs2`	Operación ***shift right*** lógico	**R**	`rd ← rs1 >> rs2`
`SRA rd, rs1, rs2`	Operación ***shift right*** aritmético	**R**	`rd ← rs1 >> rs2`
`SLLI rd, rs1, shamt`	Operación ***shift left*** lógico con literal	**I**	`rd ← rs1 << shamt`
`SRLI rd, rs1, shamt`	Operación ***shift right*** lógico con literal	**I**	`rd ← rs1 >> shamt`
`SRAI rd, rs1, shamt`	Operación ***shift right*** aritmético con literal	**I**	`rd ← rs1 >> shamt`

* `shamt` o ***shift amount*** es la cantidad de ***shifts*** a realizar y se codifica como un entero a partir de los 5 bits menos significativos del literal (`imm12[4:0]`).

Carga y Almacenamiento

Mnemotecnia	Instrucción	Tipo	Descripción
`LW rd, imm12(rs1)`	**Cargar *word* (32 bits)**	**I**	**`rd ← mem[rs1 + imm12]`**
`LH rd, imm12(rs1)`	Cargar *half word* (16 bits)	I	`rd ← mem[rs1 + imm12]`
`LB rd, imm12(rs1)`	Cargar byte (8 bits)	I	`rd ← mem[rs1 + imm12]`
`LWU rd, imm12(rs1)`	Cargar *word* sin signo (32 bits)	I	`rd ← mem[rs1 + imm12]`
`LHU rd, imm12(rs1)`	Cargar *half word* sin signo (16 bits)	I	`rd ← mem[rs1 + imm12]`
`LBU rd, imm12(rs1)`	Cargar byte sin signo (8 bits)	I	`rd ← mem[rs1 + imm12]`
`SW rs2, imm12(rs1)`	**Almacenar *word* (32 bits)**	**S**	**`rs2 → mem[rs1 + imm12]`**
`SH rs2, imm12(rs1)`	Almacenar *half word* (16 bits)	S	`rs2(15:0) → mem[rs1 + imm12]`
`SB rs2, imm12(rs1)`	Almacenar byte (8 bits)	S	`rs2(7:0) → mem[rs1 + imm12]`

* Para direccionar en `LW` y `SW`, se usa el formato `offset(x)`, donde la dirección se almacena en el registro `x`. Ejemplo: `4(sp)`

Saltos y Subrutinas

Mnemotecnia	Instrucción	Tipo	Descripción
`BEQ rs1, rs2, imm12`	Salto con condición “igual”	B	`if rs1 == rs2: PC ← PC + imm12`
`BNE rs1, rs2, imm12`	Salto con condición “distinto”	B	`if rs1 != rs2: PC ← PC + imm12`
`BGE rs1, rs2, imm12`	Salto con condición “mayor o igual”	B	`if rs1 >= rs2: PC ← PC + imm12`
`BGEU rs1, rs2, imm12`	Salto con condición “mayor o igual” sin signo	B	`if rs1 >= rs2: PC ← PC + imm12`
`BLT rs1, rs2, imm12`	Salto con condición “menor”	B	`if rs1 < rs2: PC ← PC + imm12`
`BLTU rs1, rs2, imm12`	Salto con condición “menor” sin signo	B	`if rs1 < rs2: PC ← PC + imm12`
`JAL rd, imm20`	Salto incondicional con “enlace”	J	`rd ← PC+4; PC ← PC + imm20`
`JALR rd, imm12(rs1)`	Salto incondicional con “enlace” a registro	I	`rd ← PC+4; PC ← rs1 + imm12`

* En estos casos, el literal representa el *offset* para llegar a la instrucción deseada desde el *Program Counter*. A nivel de código, se observa como el *label* de la dirección a saltar.

Pseudo-instrucciones

Mnemotecnia	Instrucción	Instrucción(es) base
`LI rd, imm12`	Cargar literal en registro que utiliza ≤ 12 bits	`ADDI rd, zero, imm12`
`LI rd, imm`	Cargar literal en registro que utiliza > 12 bits	`LUI rd, imm[31:12]; ADDI rd, rd, imm[11:0]`
`LA rd, sym`	Cargar dirección en registro	`AUIPC rd, sym[31:12]; ADDI rd, rd, sym[11:0]`
`MV rd, rs`	Copiar registro	`ADDI rd, rs, 0`
`NOT rd, rs`	Complemento de 1	`XORI rd, rs, -1`
`NEG rd, rs`	Complemento de 2	`SUB rd, zero, rs`
`BGT rs1, rs2, offset`	Salto si `rs1 > rs2`	`BLT rs2, rs1, offset`
`BLE rs1, rs2, offset`	Salto si `rs1 ≤ rs2`	`BGE rs2, rs1, offset`
`BGTU rs1, rs2, offset`	Salto si `rs1 > rs2` sin signo	`BLTU rs2, rs1, offset`
`BLEU rs1, rs2, offset`	Salto si `rs1 ≤ rs2` sin signo	`BGEU rs2, rs1, offset`

* Las pseudo-instrucciones mnemotécnicas se traducen a las instrucciones reales de RISC-V. Esto ayuda a tener instrucciones de operaciones útiles en un lenguaje más sencillo de entender.

Pseudo-instrucciones

Mnemotecnia	Instrucción	Instrucción(es) base
`BEQZ rs1, offset`	Salto si `rs1 = 0`	`BEQ rs1, zero, offset`
`BNEZ rs1, offset`	Salto si `rs1 ≠ 0`	`BNE rs1, zero, offset`
`BGEZ rs1, offset`	Salto si `rs1 ≥ 0`	`BGE rs1, zero, offset`
`BLEZ rs1, offset`	Salto si `rs1 ≤ 0`	`BGE zero, rs1, offset`
`BGTZ rs1, offset`	Salto si `rs1 > 0`	`BLT zero, rs1, offset`
`J offset`	Salto incondicional	`JAL zero, offset`
`CALL offset12`	Llamado a subrutina (dirección ≤ 12 bits)	`JALR ra, ra, offset12`
`CALL offset`	Llamado a subrutina (dirección > 12 bits)	`AUIPC ra, offset[31:12]; JALR ra, ra, offset[11:0]`
`RET`	Retorno de la subrutina	`JALR zero, 0(ra)`
`NOP`	No se realiza ninguna operación	`ADDI zero, zero, 0`

ECALL

Instrucción. que permite realizar una solicitud al entorno de ejecución (sistema operativo)

¿Cómo?

- a7 almacena 1(print) o 10(exit)
- a0 para almacenar lo que queremos printear

ECALL

Ejemplos:

CASO 1: QUIERO PRINTEAR

```
li a7, 1
li a0,2
ecall

out:

2
-- program is finished
running (0) --
```

CASO 2: QUIERO TERMINAR EL PROGRAMA

```
li a7, 10
ecall

out:

 -- program is finished running (0) --
```

Veamos la ayudantía :D