RISC-V Architecture

RISC-V

RISC-V architecture provides registers from 'x1 to x31' that are general purpose refister.
7 out if which are temporary register 't0 to t6'
a0 to a7 are used for function arguments
s0 to s11 are used as saved registers or within function definitions.

LI : Load inmediate hace esto si a0 = 0x00000000 (hace un LUI y un ADDI)

```
LI a0,0x111117FF
```

Vuelve a0 a 0x111117FF en dos pasos
LUI : Load Upper inmmediate si t1 = 0x0´20000000 , si tenemos

```
LUI t1,0x12345
```

Vuelve t1 a 0x12345000
ADDI : ADD Inmmediate si t1 = 123456000 , si tenemos

```
ADDI t1,0x678
```

Vuelve t1 a 0x12345678
ADD: Suma 2 registros el tercero es donde lo vas a guardar

```
ADD t2,t3,t2
```

ORI: La instrucción ori realiza una operación OR bit a bit entre un registro y una constante inmediata, y almacena el resultado en otro registro.

```
ORI rd, rs1, imm
```

- rd es el registro de destino donde se almacenará el resultado de la operación OR.
- rs1 es el registro fuente que se usará en la operación OR.
- imm es el valor inmediato (constante) que se usará en la operación OR.

```
# Cargamos un valor en el registro x1
LI x1, 123   # x1 = 123

# Realizamos la operación OR con la constante inmediata 0x00FF
ORI x3, x1, 0x00FF  # x3 = x1 OR 0x00FF
```

```
# x3 ahora tiene el resultado de la operación OR
```

LW (load word): Esta instrucción se utiliza para cargar un valor de 32 bits (una palabra) desde la memoria en un registro. Se usa para acceder a datos almacenados en la memoria y cargarlos en un registro para su posterior procesamiento.

```
LW t0, 0(a0)  # Carga un valor de memoria en el registro t0
```

ejemplo:

```
.data
valor: .word 0x12345678  # Se declara una palabra de datos en memoria

.text
LW t0, valor   # Carga el valor de memoria en el registro t0

# Aquí puedes hacer operaciones con el valor en t0 si es necesario
```

MV (move): Esta instrucción se utiliza para copiar el valor de un registro en otro registro. No implica acceso a la memoria ni operaciones aritméticas o lógicas. Simplemente realiza una copia directa del valor de un registro a otro. Por ejemplo:

```
MV t1, t0  # Copia el valor de t0 en t1
```

SUB (substract) : Es una resta

```
# Cargamos valores en registros
LI x1, 30    # x1 = 30
LI x2, 15    # x2 = 15

# Restamos los valores
SUB x3, x1, x2  # x3 = x1 - x2 (15)

# x3 ahora tiene el resultado de la resta
```

MUL (multiplication) : Es una multiplicacion

```
# Cargamos valores en registros
li x1, 5     # x1 = 5
li x2, 6     # x2 = 6

# Multiplicamos los valores
mul x3, x1, x2  # x3 = x1 * x2 (30)

# x3 ahora tiene el resultado de la multiplicación
```

DIV (division) : Es una division

```
# Cargamos valores en registros
LI x1, 20    # x1 = 20
LI x2, 4     # x2 = 4

# Dividimos los valores
DIV x3, x1, x2  # x3 = x1 / x2 (5)

# x3 ahora tiene el resultado de la división
```

REM (resto) : Obtener el resto de una division

```
REM a1, s3, t3 # a1 = s3 % t31
```

rem a1, s3, t3 Divides the value of t3 (denominator) from the value of s3 (numerator) and stores the remainder into the register a1.

Operaciones compuestas

```
# Supongamos que ya tenemos los valores de b, c, d y e en los registros correspondientes
# b + c
ADD x5, x1, x2    # x5 = b + c
# d / e
DIV x6, x3, x4    # x6 = d / e
# (b + c) * (d /5 e)
MUL x5, x5, x6    # x5 = (b + c) * (d / e)
# x5 ahora tiene el resultado de la operación (b + c) * (d / e)
```

Operaciones logicas

AND: and bit a bit entre x1 y x2

```
# Supongamos que x1 = 0b11001100 y x2 = 0b00110011
and x5, x1, x2   # x5 = x1 AND x2
# x5 ahora tiene el resultado de la operación AND
```

OR: or bit a bit entre x1 y x2

```
# Supongamos que x1 = 0b11001100 y x2 = 0b00110011
OR x5, x1, x2   # x5 = x1 OR x2
# x5 ahora tiene el resultado de la operación OR
```

XOR bit a bit entre x1 y x2

```
# Supongamos que x1 = 0b11001100 y x2 = 0b00110011

XOR x5, x1, x2   # x5 = x1 XOR x2

# x5 ahora tiene el resultado de la operación XOR
```

Y si queres poder meterle un "i" para comparar lo que viene siendo esta parte
0x12345|678| <- ESO

Barrel shifts

Tambien es posible hacer desplazamientos logicos en lenguaje ensamblador, estos son los operandos para hacer dicha accion

SLLI desplazamiento logico hacia la izquierda

```
SLLI rd, rs1, shamt
```

rd es el registro de destino donde se almacenará el resultado del desplazamiento.
rs1 es el registro fuente que contiene el valor que se va a desplazar a la izquierda.
shamt es la cantidad de bits que se desplazarán hacia la izquierda. Este valor debe ser un número inmediato (constante) de 5 bits.

SRAI desplazamiento logico hacia la derecha. en el conjunto de instrucciones RISC-V se utiliza para realizar una operación de desplazamiento aritmético a la derecha (shift right arithmetic) en un valor de registro. El formato de la instrucción SRAI es el siguiente:

```
SRAI rd, rs1, shamt
```

Operadores cuantificativos

Menor o igual ($\leq$) : a instrucción SLT (Set Less Than) en el conjunto de instrucciones RISC-V se utiliza para realizar una comparación con signo entre dos valores en registros y establecer un tercer registro en 1 si el primer valor es menor que el segundo, o en 0 si no lo es. El formato de la instrucción SLT es el siguiente:

```
SLT rd, rs1, rs2
```

- rd es el registro de destino donde se almacenará el resultado de la comparación.
- rs1 es el registro fuente que contiene el primer valor para la comparación.
- rs2 es el registro fuente que contiene el segundo valor para la comparación.

Mayor igual ($\geq$) : Para comparar si un valor en rs1 es mayor o igual que un valor en rs2, puedes usar SLT y luego realizar una instrucción de salto condicional en función del resultado de SLT, pero invirtiendo el sentido de la condición.

```
SGT x3, x2, x1    # Compara si x2 es menor que x1 (invierte los registros)
```

Saltos

J (Jump):

Formato: J offset
La instrucción J se utiliza para realizar un salto incondicional. Simplemente realiza un salto a una dirección de memoria absoluta especificada por offset. A diferencia de BEQ, no realiza una comparación entre registros; siempre ejecutará el salto.
Ejemplo: La ejecución se desviará directamente a la dirección de memoria indicada por el offset, sin importar las condiciones.

Saltar si Igual $=$: La instrucción BEQ se utiliza para realizar un salto condicional. Compara el contenido de dos registros, rs1 y rs2, y si son iguales, realiza un salto

```
BEQ rs1, rs2, etiqueta
```

- rs1 es el primer registro que se compara.
- rs2 es el segundo registro que se compara.
- etiqueta es el nombre de la etiqueta de destino a la que se saltará si rs1 y rs2 son iguales.

Saltar si Distinto de ($\neq$) : se representa generalmente con la instrucción de salto condicional "BNE" (Branch if Not Equal). La instrucción BNE se utiliza para comparar dos registros y saltar a una etiqueta específica si los registros no son iguales.

```
BNE rs1, rs2, etiqueta
```

- rs1 es el primer registro que se compara.
- rs2 es el segundo registro que se compara.
- etiqueta es el nombre de la etiqueta de destino a la que se saltará si rs1 y rs2 si rs1 es distinto que rs2.

Saltar si Mayor e igual que ($\geq$): en el conjunto de instrucciones RISC-V se utiliza para realizar una comparación de "mayor o igual que" (greater than or equal) y realizar un salto condicional basado en esa comparación. Su formato es el siguiente:

```
BGE rs1, rs2, etiqueta
```

- rs1 es el primer registro que se compara.
- rs2 es el segundo registro que se compara.
- etiqueta es el nombre de la etiqueta de destino a la que se saltará si rs1 y rs2 es mayor o iguales.

Saltar si Menor e igual que ($\leq$): en el conjunto de instrucciones RISC-V se utiliza para realizar una comparación de "menor o igual que" (lesser than or equal) y realizar un salto condicional basado en esa comparación. Su formato es el siguiente:

```
BLE rs1, rs2, etiqueta
```

- rs1 es el primer registro que se compara.
- rs2 es el segundo registro que se compara.
- etiqueta es el nombre de la etiqueta de destino a la que se saltará si rs1 y rs2 es mayor o iguales.