Instrucciones de RISC-V

li t0, imm
- Carga en inmediato imm en el registro t0

ld rd, offset(rs1)
- Carga un valor obtenido en memoria en el registro rd, offset es un inmediato que se suma al registro rs1 para obtener la dirección de memoria

sd rs1, offset(rs2)
- Carga un valor en registro a memoria, rs1 es el registro que contiene el valor que se va a almacenar, offset es un desplazamiento (un valor inmediato) que se suma a la dirección contenida en el registro rs2 para calcular la dirección efectiva de memoria donde se almacenará el dato.

bne rs1, rs2, label
- En RISC-V, la instrucción bne (branch if not equal) se utiliza para realizar un salto condicional basado en la comparación de dos registros. Si los registros no son iguales, la ejecución salta a una dirección especificada.

Ejercicio 1

El siguiente código de máquina y su desensamblado RISC-V **computa la suma prefijo en el mismo arreglo** (*in-place prefix sum*). El arreglo `a` está en el segmento ELF `.bss` y empieza en `0x2FC0` y termina en `0x3008` **exclusive**. Como sus elementos son `unsigned long`, cada uno ocupa 8 bytes y por lo tanto tiene 9 elementos.

```
0000000000000634 <main>:
 634: 0613                  li    a2,0x3008   # <__BSS_END__> &a[9]
 636: b206                  li    a5,0x2FC8   # <a+0x8> &a[1]
 638: 6398                  ld    a4,0(a5)    # a4 = a[i]
 63a: ff87b683              ld    a3,-8(a5)   # a3 = a[i-1]
 63e: 9736                  add   a4,a4,a3
 640: e398                  sd    a4,0(a5)    # a[i] = a4
 642: 07a1                  addi  a5,a5,8     # "i++"
 644: fec79ae3              bne   a5,a2,0x638 # <main+0x10>, "i<9"
 648: 8082                  ret
```

cargo en a2 el final del arreglo a[9]
cargo en a5 la dirección del primer elemento del arreglo a[1]

==========

cargo en a4 lo que tengo en memoria en a[1]
cargo en a3 lo que tengo en memoria en a[0]

guardar en a4 = a4 + a3 → a4 = a[1] + a[0]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[1] = a4 es la suma

aumentó la dire a5 ahora nos da a[2]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[2] = &a[9]

==========

cargo en a4 lo que tengo en memoria en a[2]
cargo en a3 lo que tengo en memoria en a[1]

guardar en a4 = a4 + a3 → a4 = a[2] + a[1]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[2] = a4 es la suma

aumentó la dire a5 ahora nos da a[3]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[3] = &a[9]

==========

cargo en a4 lo que tengo en memoria en a[3]
cargo en a3 lo que tengo en memoria en a[2]

guardar en a4 = a4 + a3 → a4 = a[3] + a[2]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[3] = a4 es la suma

aumentó la dire a5 ahora nos da a[4]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[4] = &a[9]

==========

cargo en a4 lo que tengo en memoria en a[4]
cargo en a3 lo que tengo en memoria en a[3]

guardar en a4 = a4 + a3 → a4 = a[4] + a[3]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[4] = a4 es la suma

aumentó la dire a5 ahora nos da a[5]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[5] = &a[9]

==========

cargo en a4 lo que tengo en memoria en a[5]
cargo en a3 lo que tengo en memoria en a[4]

guardar en a4 = a4 + a3 → a4 = a[5] + a[4]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[5] = a4 es la suma

aumentó la dire a5 ahora nos da a[6]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[6] = &a[9]

==========

cargo en a4 lo que tengo en memoria en a[6]
cargo en a3 lo que tengo en memoria en a[5]

guardar en a4 = a4 + a3 → a4 = a[6] + a[5]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[6] = a4 es la suma

aumentó la dire a5 ahora nos da a[7]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[7] = &a[9]

==========

cargo en a4 lo que tengo en memoria en a[7]
cargo en a3 lo que tengo en memoria en a[6]

guardar en a4 = a4 + a3 → a4 = a[7] + a[6]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[7] = a4 es la suma

aumentó la dire a5 ahora nos da a[8]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[8] = &a[9]

==========

cargo en a4 lo que tengo en memoria en a[8]
cargo en a3 lo que tengo en memoria en a[7]

guardar en a4 = a4 + a3 → a4 = a[8] + a[7]

cargo en memoria a5 + 0 lo que tenía en a4 es decir a[8] = a4 es la suma

aumentó la dire a5 ahora nos da a[9]
si a5 no es igual a a2 salto al bucle de nuevo es decir &a[9] = &a[9]

resultado final:
a[1] = a[1] + a[0]
a[2] = a[2] + a[1]
a[3] = a[3] + a[2]
a[4] = a[4] + a[3]
a[5] = a[5] + a[4]
a[6] = a[6] + a[5]
a[7] = a[7] + a[6]
a[8] = a[8] + a[7]

2FC8 = 0010 1111 1100 1000
1
2FC9 = 0010 1111 1100 1001
1
2FCA = 0010 1111 1100 1010
1
2FCB = 0010 1111 1100 1011
1
2FCB = 0010 1111 1100 1100

Ejercicio 3

Planificar con Round Robin Q=2 para los siguientes procesos que tienen mezcla entre cómputo CPU y espera IO. Ante situaciones de simultaneidad, ordenar alfabéticamente, por ejemplo ¿Cuál de los tres procesos inicia en tiempo 0?: el "A".

Proceso	Inicio	CPU	IO	CPU
A	0	1	4	3
B	0	1	1	1
C	0	8		

	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	C	B	C	C	A	A	C	C	A	C	C							
	A	A																
	B	C	A	A	C	C	A	A	C									
		B							A		C							

Supongamos que tenemos el registro de paginación apuntando al marco físico `satp=0x00000000FE0`.

0x00000000FE0 ---------- 0x1FF: 0x00000000000, ---- ⋮ 0x004: 0x00000000000, ---- 0x003: 0x00000000000, ---- 0x002: 0x00000000FEA, XWRV 0x001: 0x00000000FEA, XWRV 0x000: 0x00000000FEA, XWRV	0x00000000FEA ---------- 0x1FF: 0x00000000000, ---- ⋮ 0x004: 0x00000000000, ---- 0x003: 0x00000000000, ---- 0x002: 0x000000AD0BE, XWRV 0x001: 0x000000AD0BE, XWRV 0x000: 0x000000AD0BE, XWRV	0x000000AD0BE ---------- 0x1FF: 0x00000000000, ---- ⋮ 0x004: 0x00000000000, ---- 0x003: 0x00000D1AB10, XWR- 0x002: 0x00000DECADA, -WRV 0x001: 0x000CAFECAFE, ---- 0x000: 0x0000000ABAD, X--V

a) Traducir de **virtual a física** las direcciones:

`0x0000`

`0x1000`

`0x2000`

`0x3000`

Esquema (9,9,9,12)
0x0000 = 0x 000 0000 0000 0000 0000 0000 0000 0000 0000 0000
PD1 = 0 → (0x00000000FEA)
PD2 = 0 → (0x000000AD0BE)
PD3 = 0 → (0x0000000ABAD)
OFFSET = 0 → 0x0000000ABAD000

0x1000 = 0x 000 0000 0000 0000 0000 0000 1000 0000 0000 0000 PD1 = 0 →
(0x00000000FEA) PD2 = 0 → (0x000000AD0BE) PD3 = 8 → INVALIDA

0x40402980 = 000000001 000000010 000000010 100110000000
0x40202980 = 000000001 000000001 000000010 100110000000
0x40002980

0x402980
0x202980
0x2980 000000010 1001 1000 0000

0x100802980 = 000000100 000000100 000000010 100110000000

0x80402980 = 000000010 000000010 0000000010 100110000000

Ejercicio 5

0x634 = 000000000 000000000 000000000 011000110100

PD1 = 0x0

PD2 = 0x0

PD3 = 0x0

OFFSET = 0x634

0xABAD634

0x636 = 000000000 000000000 000000000 011000110110

0x2FC8 = 000000000 000000000 000000010 111111001000
→ 0xDECADA4040

0x2FC0 = 000000000 000000000 000000010 11111100 0000
→0xDECADA4030

0x2FD0 = 000000000 000000000 000000010 111111010000
→0xDECADA4048

0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D

1	1	1	0	E
1	1	1	1	F