



Tema 2:

# Arquitectura del procesador

Fundamentos de computadores II

**José Manuel Mendías Cuadros**

*Dpto. Arquitectura de Computadores y Automática  
Universidad Complutense de Madrid*





# Contenidos

- ✓ Arquitectura RISC-V.
- ✓ Instrucciones y datos.
- ✓ Modelo de memoria.
- ✓ Registros.
- ✓ Modos de direccionamiento.
- ✓ Repertorio de instrucciones.
- ✓ Extensiones.
- ✓ Arquitecturas RISC vs. CISC.

Transparencias basadas en los libros:

- S.L. Harris and D. Harris. *Digital Design and Computer Architecture. RISC-V Edition.*
- D.A. Patterson and J.L. Hennessy. *Computer Organization and Design. RISC-V Edition.*



# Arquitectura RISC-V

- **RISC-V ISA** (*Instruction Set Architecture*) es una arquitectura:
  - Abierta, no propietaria y en evolución.
  - Originariamente desarrollada en la Univ. de Berkeley en 2010.
  - Actualmente coordinada por el consorcio RISC-V International.
- Es de **tipo RISC** (*Reduced Instruction Set Computer*) por lo que:
  - Tiene un repertorio reducido de instrucciones simples.
  - Sólo las instrucciones de carga y almacenamiento acceden a memoria.
  - El resto de instrucciones trabajan con datos almacenados en registros.
  - Dispone de un gran número de registros de propósito general.
  - Tienen un conjunto reducido de modos de direccionamiento.
  - Instrucciones de tamaño fijo y con un número reducido de formatos.
- Estudiaremos el **repertorio base RV32I con la extensión RVM**.
  - Datos enteros de 32 bits e instrucciones de 32 bits (RV32I).
  - Con operaciones de multiplicación y división sobre enteros (RVM)



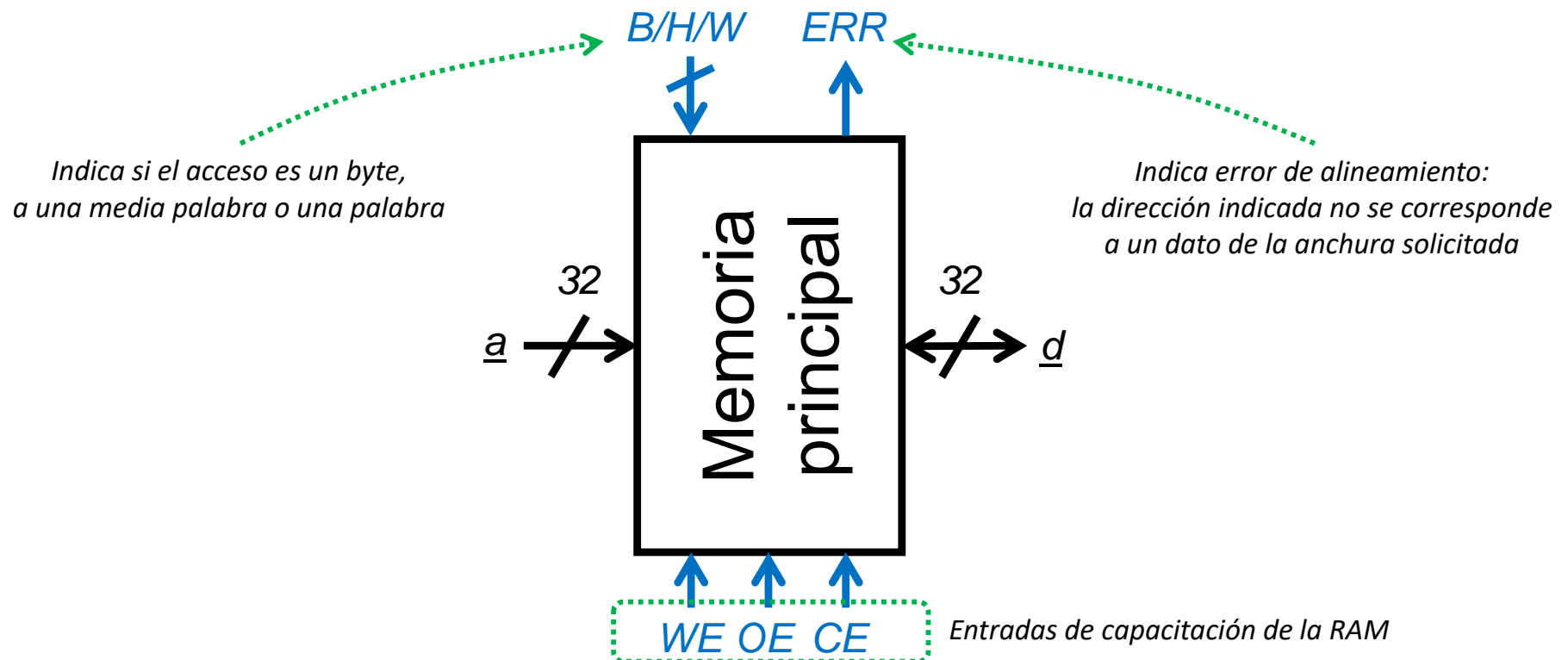
# Instrucciones y datos

- Toda **instrucción** en RISC-V ocupa **32 bits**.
- Las **instrucciones** en RISC-V operan con **datos** o **direcciones** de **32 bits**.
  - Los **datos son números enteros** (con signo) o números **naturales** (sin signo) codificados **en complemento a 2** o **binario puro** respectivamente.
  - Las **direcciones son números naturales** codificados en **binario puro**.
- No obstante, puede trabajar con **números de menor anchura**:
  - Típicamente son **extendidos a 32 bits** previamente a operar con ellos.
  - Según el caso, se **extenderá su signo** (sExt) o se **completarán con 0s** por la izquierda (zExt).
- Los tamaños de datos más comunes son:
  - **Palabra** (*word*): 32 bits.
  - **Media palabra** (*half word*): 16 bits.
  - **Byte**: 8 bits.



# Modelo de memoria

- Asume una **memoria principal RAM** de **4 GiB** ( $2^{32} \times 8b = 2^{30} \times 32b$ ):
  - Bus de datos y de direcciones **de 32 bits**.
  - **Direccionable por bytes** (cada byte tiene una dirección única).
  - Contiene datos de 8, 16 y 32 bits e instrucciones de 32 bits.
  - Todos ellos **alineados** y con **ordenación little-endian**.





# Modelo de memoria

## Alineamiento

- En la memoria de un RISC-V la **información está alineada**, es decir, existen restricciones de ubicación en función de su tamaño.
  - **Byte**: puede ubicarse **en cualquier dirección**.
  - **Media palabra**: puede ubicarse **solo en direcciones múltiplo de 2** (pares).
  - **Palabra**: puede ubicarse **solo en direcciones múltiplo de 4**.
    - Aplica a datos de 32 bits e instrucciones.
  - En general, datos de **N bytes deben ubicarse en direcciones múltiplo de N**.
  - Cuando se ubican consecutivamente varios datos de distinto tamaño en memoria, quedan huecos vacíos.

Tamaño	Dato
byte	0x24
palabra	0x3b257a02
½ palabra	0x3e27
palabra	0x01c6d823

Dir.	+0	+1	+2	+3
3c000000	24			
3c000004	3b257a02			
3c000008	3e27			
3c00000c	01c6d823			

RISC-V: Datos alineados

Dir.	+0	+1	+2	+3
3c000000	24	3b257a		
3c000004	02	3e27	01	
3c000008	6d823			
3c00000c				

Datos no alineados



# Modelo de memoria

## Ordenamiento

- En la memoria de un RISC-V los bytes de una palabra/media palabra tienen **ordenamiento little-endian**:
  - En la **dirección más baja** se ubica el **byte menos significativo**, es decir, la dirección del dato coincide con la de su byte menos significativo.
  - Los bits dentro del byte tienen el orden habitual.
- Otros procesadores tiene **ordenamiento big-endian**:
  - En la **dirección más baja** se ubica el **byte más significativo**, es decir, la dirección del dato coincide con la de su byte más significativo.

Tamaño	Dato
byte	0x24
palabra	0x3b257a02
½ palabra	0x3e27
palabra	0x01c6d823

Dir.	+0	+1	+2	+3
3c000000	24			
3c000004	02	7a	25	3b
3c000008	27	3e		
3c00000c	23	d8	c6	01

RISC-V: Little-Endian

Dir.	+0	+1	+2	+3
3c000000	24			
3c000004	3b	25	7a	02
3c000008	3e	27		
3c00000c	01	c6	d8	23

Big-Endian

# Registros



- Todos los datos de un programa residen en memoria, pero para ser operados por un RISC-V deben previamente cargarse en registros.
- Un RISC-V dispone de 32 registros de 32 bits de propósito general.
  - Pueden usarse indistintamente.
  - Se numeran del x0 al x31.
  - El registro x0, contiene la constante 0 y su escritura no tiene efecto.
  - No obstante, para facilitar la programación, cada registro tiene un alias que permite recordar su uso por convenio más habitual.
- Adicionalmente dispone del registro especial, PC (Program Counter)
  - Contiene la dirección de memoria que ocupa la instrucción a ejecutar.
  - Al terminar de ejecutarla se incrementa +4 (cada instrucción ocupa 4B)
    - Excepto en caso de que la instrucción ejecutada sea de salto.



# Registros



# Reg.	Alias	Descripción
x0	zero	zero – cero
x1	ra	return address – dirección de retorno
x2	sp	stack pointer – puntero de pila
x3	gp	global pointer – puntero global
x4	tp	task pointer – puntero de hebra
x5...x7	t0...t2	temporary register – registro temporal
x8	s0/fp	saved register / frame pointer registro preservado / puntero de marco
x9	s1	saved register – registro preservado
x10...x17	a0...a7	argument register – registro de argumento
x18...x27	s2...s11	saved register – registro preservado
x28...x31	t3...t6	temporary register – registro temporal

programando en ensamblador deben **usarse siempre los alias**

# Modos de direccionamiento



- Los **modos de direccionamiento** son el conjunto de mecanismos que permiten indicar **dónde se encuentran los operandos** de una instrucción
  - Indican al procesador la ubicación de los datos y la manera de obtenerlos.
- Los **operandos de una instrucción** pueden estar ubicados en:
  - La propia **instrucción**. <sup>4</sup>
  - Un **registro del procesador**, deberá indicarse cuál.
  - La **memoria del computador**, deberá indicarse la dirección que ocupa.  $a = b + 3$
- En RISC-V solo existen **4 modos** de direccionamiento:
  - **Inmediato**: el operando es una **constante** ubicada en la **propia instrucción**.
  - **Directo a registro**: el operando se encuentra en un **registro del procesador**.
  - **Relativo a registro-base**: el operando se encuentra en la **memoria**.
    - Su dirección se obtiene sumando el contenido de un registro y un inmediato.
  - **Relativo a PC**: el operando es una **dirección** (de salto).
    - Se obtiene sumando el contenido del PC y un inmediato.



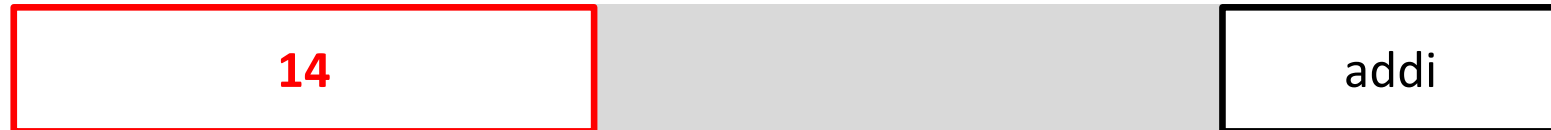
# Modos de direccionamiento

## Inmediato

- El operando es una **constante** contenida en la **propia instrucción**.
  - En **ensamblador** se indica explícitamente la **constante** con la que operar:

**addi** x1, x1, **14**

- La **instrucción máquina** contiene un campo que almacena la **constante**:



- Dado que las instrucciones son de 32b y los operandos inmediatos están contenidos en ellas, las constantes son de menor anchura:
  - **Inmediatos sin signo de 5 bits**: se usan sin extender.
  - **Inmediatos con signo de 12/13 bits**: se extienden a 32 bits antes de usarlos.
    - Si la constante es de 13 bits la instrucción solo almacena los 12 más significativos.
  - **Inmediatos de 20 bits**: se usan sin extender pero desplazados.
  - **Inmediatos de 21 bits**: se extienden a 32 bits antes de usarlos.
    - La instrucción solo almacena los 20 más significativos.



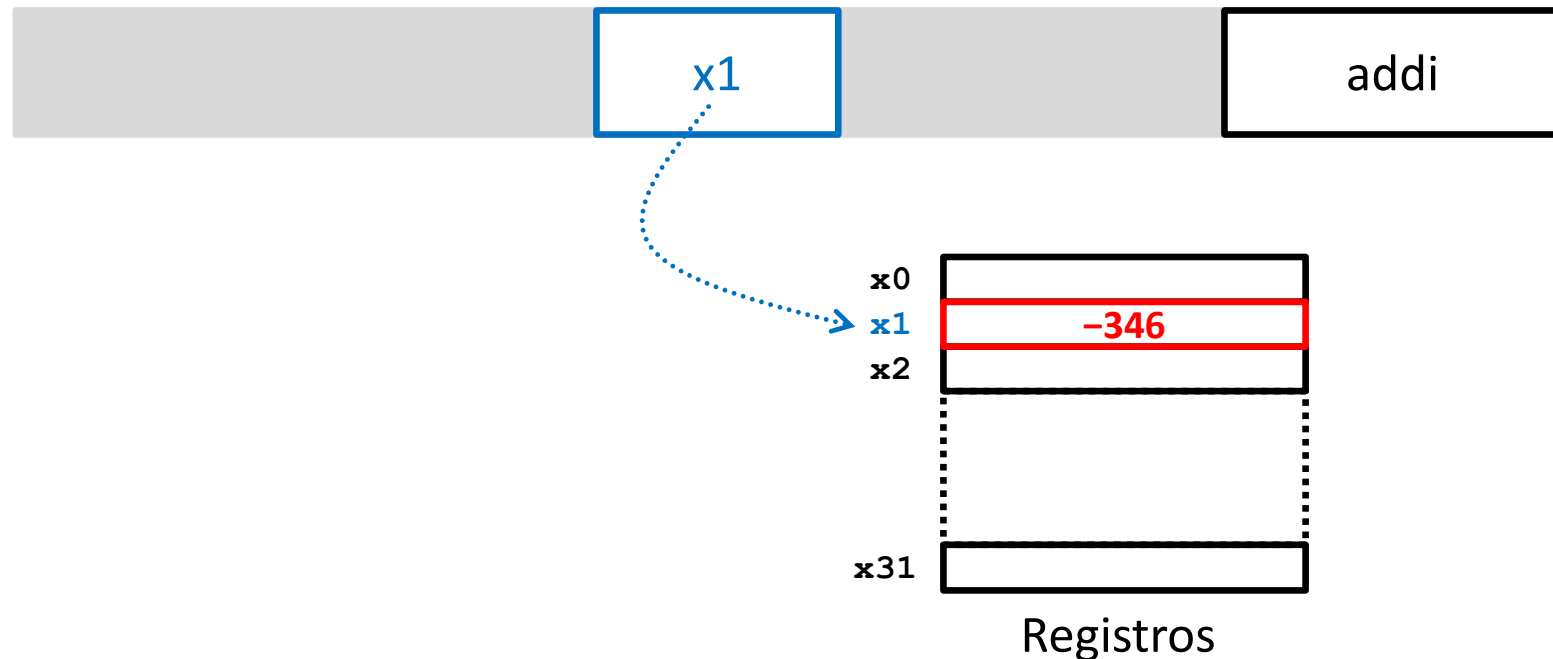
# Modos de direccionamiento

## Directo a registro

- El operando está almacenado en un **registro del procesador**.
  - En **ensamblador** se indica el nombre **registro** que contiene el dato con el que operar:

`addi x1, x1, 14`

- La **instrucción máquina** contiene un campo que indica el **número del registro**:





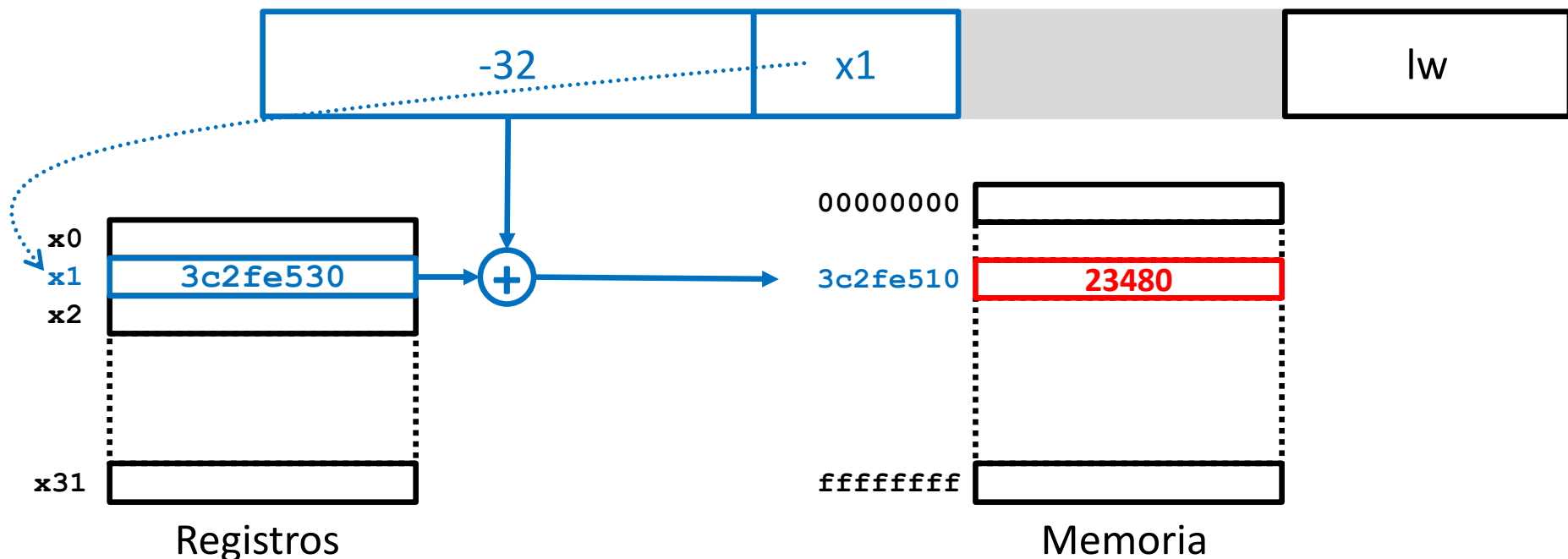
# Modos de direccionamiento

## Relativo a registro base

- El operando está en una posición de **memoria** cuya **dirección** se calcula:
  - Sumando el contenido de un **registro del procesador** (registro base) y un **desplazamiento constante** (*offset*) contenido en la propia instrucción.
  - En **ensamblador** se indica explícitamente el **desplazamiento** y el **registro**:

**lw** x3, **-32 (x1)**

- La **instrucción máquina** contiene campos para indicar **ambos elementos**:



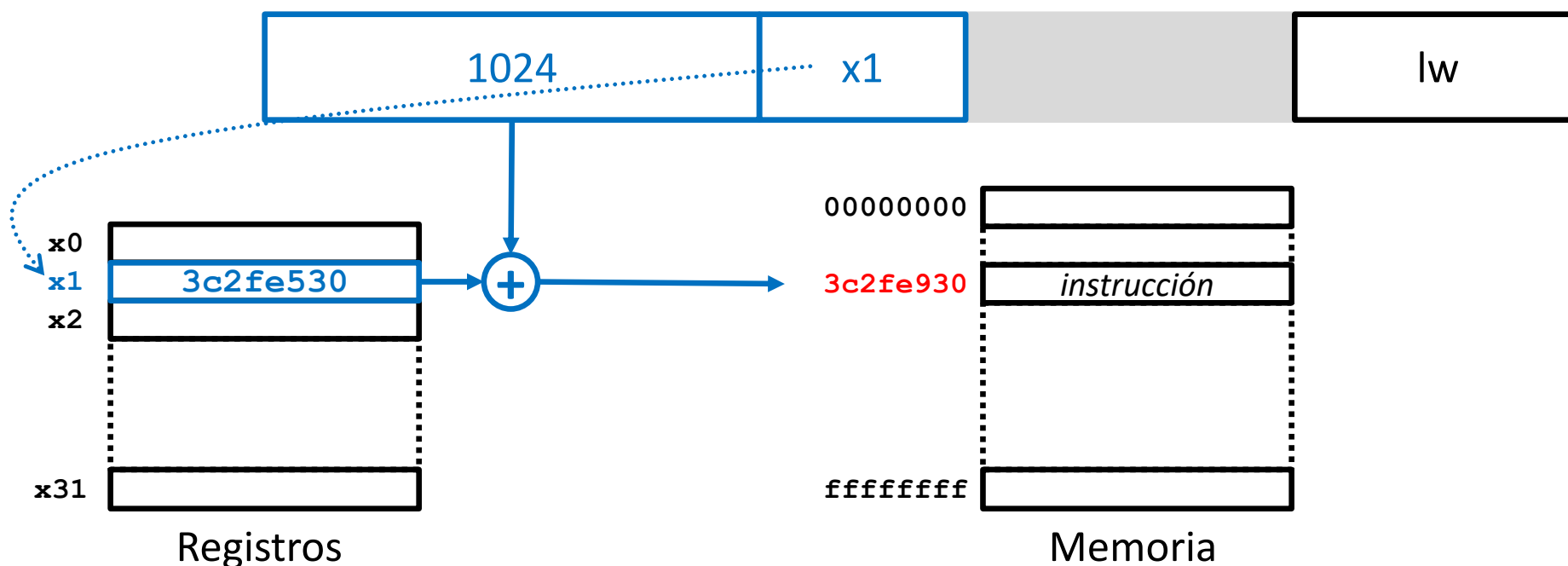


# Modos de direccionamiento

## Relativo a registro base

- Un **caso particular** de aplicación de este direccionamiento, es cuando el operando es la propia **dirección** calculada.
  - Se usa en instrucciones de salto.
  - La dirección (de salto) se calcula del mismo modo: **sumando** el contenido de un **registro** y un **desplazamiento constante** contenido en la instrucción.

`jalr x3, x1, 1024`





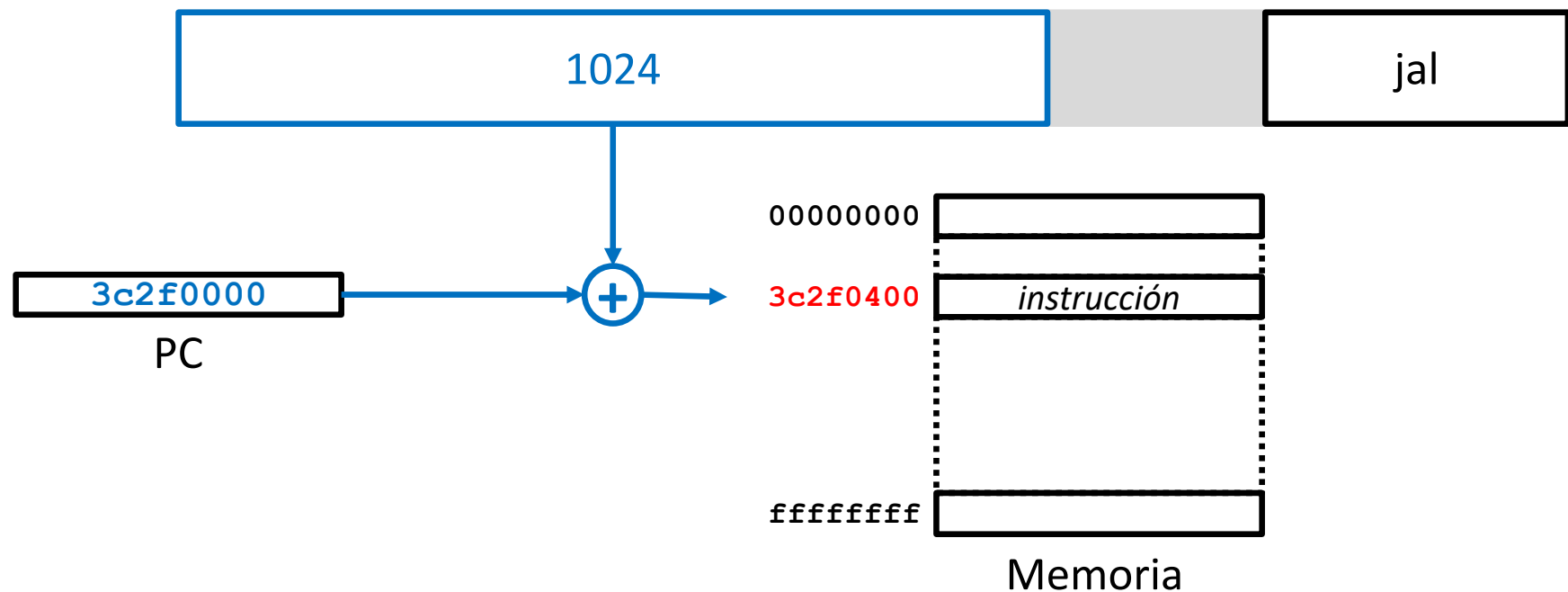
# Modos de direccionamiento

## Relativo a PC

- El operando es una **dirección** (de salto) que se calcula:
  - **Sumando** el contenido del **PC** (dirección de la instrucción en ejecución) y un **desplazamiento constante** (*offset*) contenido en la propia instrucción.
  - En **ensamblador** solo se indica explícitamente el **desplazamiento**:

**jal** x3, **1024**

- La **instrucción máquina** contiene un campo para indicar el **desplazamiento**:





# Modos de direccionamiento

## Sobre direccionamiento relativo

- En RISC-V **no existe** direccionamiento absoluto porque el relativo (a PC o a un registro base) es más conveniente en la mayoría de casos.
  - **Direccionamiento absoluto**: la instrucción se indica explícitamente la dirección de memoria en donde se ubica el dato/instrucción.
  - El direccionamiento absoluto requiere indicar los 32 bits de la dirección.
  - En un direccionamiento relativo solo se indica la diferencia existente entre 2 direcciones que suelen requerir menos bits para codificarse.
- Los desplazamientos pueden ser inmediatos cortos porque:
  - En el caso de instrucciones, lo habitual es saltar a direcciones cercanas.
    - Ubicadas, por tanto, a pequeños desplazamientos relativos al PC.
  - En el caso de datos, suelen concentrarse en una región contigua de memoria.
    - Si la dirección de comienzo de la región se almacena en un registro base, podrá accederse a todos los datos con pequeños desplazamientos relativos a la base.
- Además, el relativo al PC permite que el código sea reubicable:
  - El cálculo de direcciones efectivas de salto es siempre correcto con independencia de la dirección de memoria en donde se ubique el programa.





# Repertorio de instrucciones

## Concepto y tipos de instrucciones

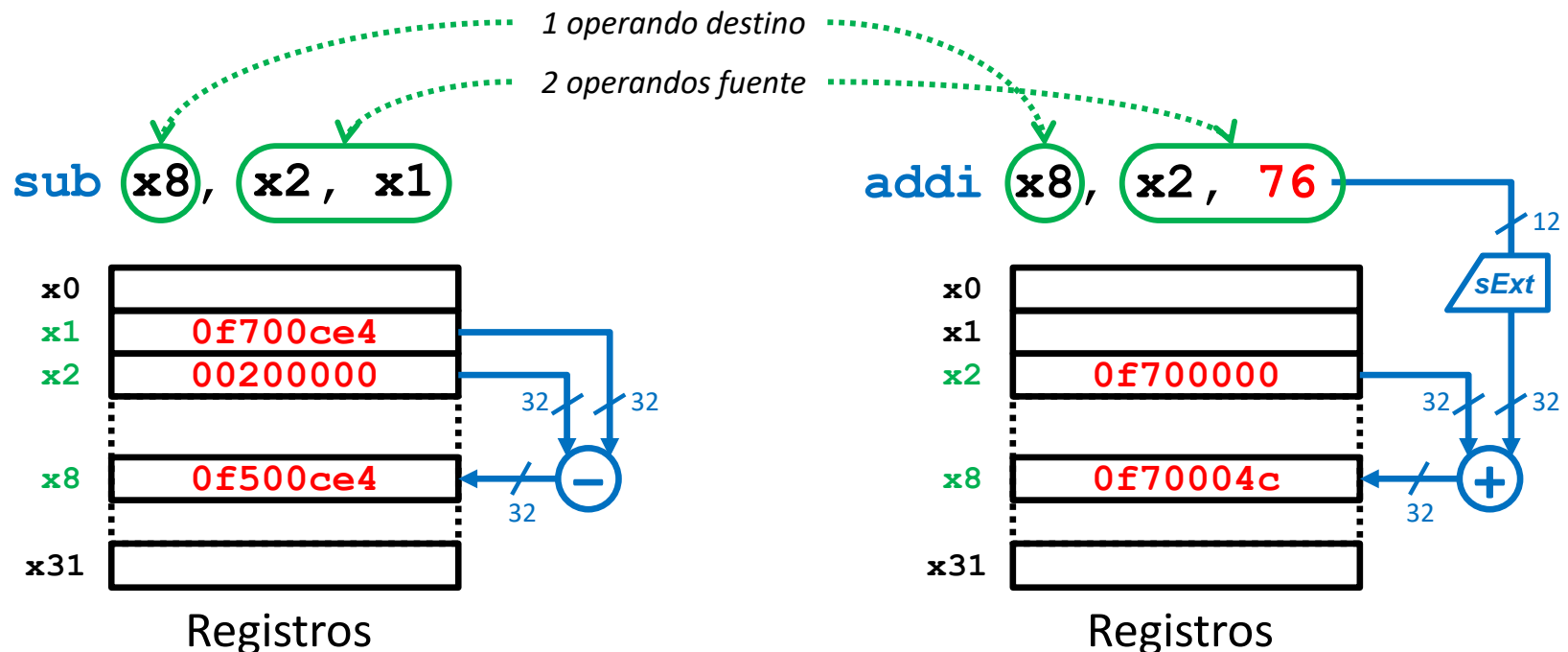
- El **repertorio de instrucciones** es el conjunto de todas las instrucciones que puede ejecutar un procesador.
  - Todos los programas que ejecuta un computador son secuencias de instrucciones pertenecientes a un mismo repertorio.
- Las instrucciones se pueden **clasificar** en diferentes tipos:
  - **Transferencia de datos**: copian datos entre registros y memoria.
  - **Aritméticas**: realizan operaciones de tipo aritmético.
  - **Lógicas**: realizan operaciones de tipo lógico bit a bit.
  - **Desplazamiento**: realizan operaciones de desplazamiento de bits.
  - **Salto**: rompen el orden implícito de ejecución modificando el PC.
  - **Privilegiadas**: permiten acceso a funcionalidades para control del sistema.
- El repertorio de instrucciones del RISC-V:
  - Es **extremadamente reducido**, evita cualquier duplicidad.
  - **Instrucciones y modos de direccionamiento** están **fuertemente acoplados**.



# Repertorio de instrucciones

## Aritméticas (i)

- Permiten realizar **operaciones de tipo aritmético** con 2 operandos fuente y 1 operando destino, todos de 32 bits.
  - El **operando izquierdo** se encuentra siempre en un **registro**.
  - El **operando derecho** se encuentra en un **registro** o es un **inmediato corto**.
    - La constante inmediata son **12b en C2** en el rango  $[-2048, +2047]$  pero su **signo se extiende hasta los 32b** antes de operar con ella.
  - El **resultado** se almacena siempre en un **registro**.





# Repertorio de instrucciones

## Aritméticas (ii)

Instrucción	Operación	Descripción
<b>add</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 + rs2$	<b>add</b> suma
<b>sub</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 - rs2$	<b>sub</b> tract resta
<b>slt</b> <i>rd, rs1, rs2</i>	$rd \leftarrow \text{if } (rs1 <_s rs2) \text{ then } (1) \text{ else } (0)$	set if <b>l</b> ess <b>t</b> han "menor que" con signo
<b>sltu</b> <i>rd, rs1, rs2</i>	$rd \leftarrow \text{if } (rs1 <_u rs2) \text{ then } (1) \text{ else } (0)$	set if <b>l</b> ess <b>t</b> han <b>u</b> nsigned "menor que" sin signo
<b>addi</b> <i>rd, rs1, imm<sub>12b</sub></i>	$rd \leftarrow rs1 + \text{sExt}(imm)$	<b>add i</b> mmediate suma una constante
<b>slti</b> <i>rd, rs1, imm<sub>12b</sub></i>	$rd \leftarrow \text{if } (rs1 <_s \text{sExt}(imm)) \text{ then } (1) \text{ else } (0)$	set if <b>l</b> ess <b>t</b> han <b>i</b> mmediate "menor que constante" con signo
<b>sltiu</b> <i>rd, rs1, imm<sub>12b</sub></i>	$rd \leftarrow \text{if } (rs1 <_u \text{sExt}(imm)) \text{ then } (1) \text{ else } (0)$	set if <b>l</b> ess <b>t</b> han <b>i</b> mmediate <b>u</b> nsigned "menor que constante" sin signo

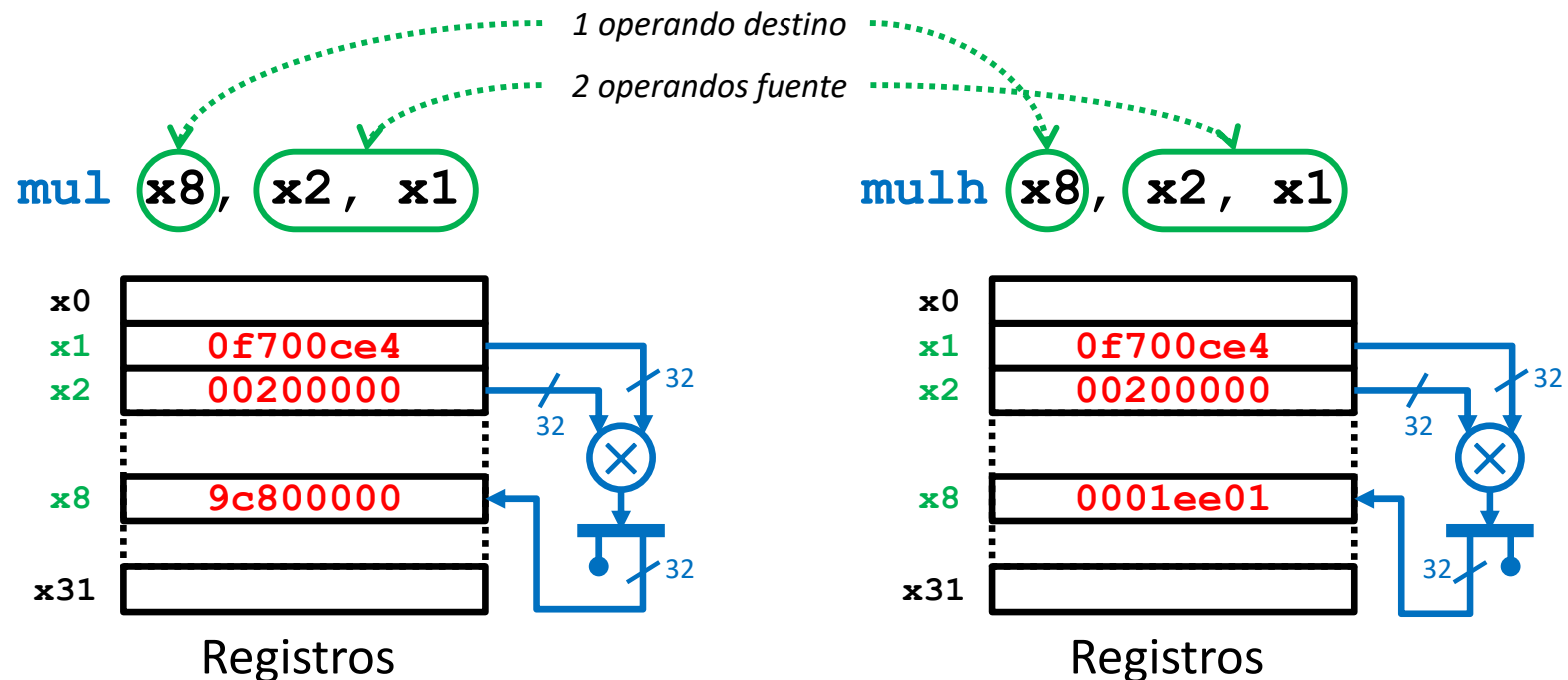
- Existen **instrucciones diferentes** de comparación para interpretar ambos operandos fuente como **datos con y sin signo**.
- **No existe** resta con operando inmediato, porque basta con sumar el opuesto.



# Repertorio de instrucciones

## Multiplicación y división (i)

- El resultado de multiplicar 2 datos de 32 bits requiere 64 bits.
  - Por ello existen 2 tipos de instrucciones de multiplicación distintas: una para calcular la parte alta del resultado y otra para calcular la parte baja.
  - Existen instrucciones diferentes que obtienen la parte alta de resultado según los operandos fuente sean datos con y sin signo.
  - Solo existe una instrucción para obtener la parte baja del resultado.
  - Todos los operandos de estas instrucciones se encuentran en registros.

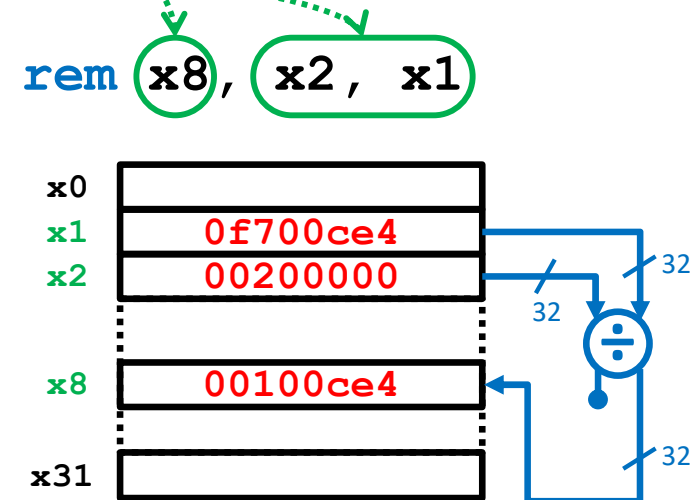
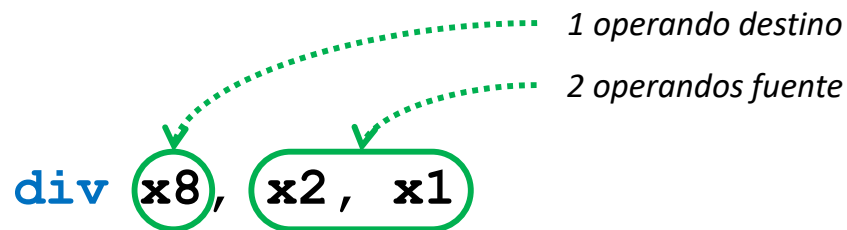




# Repertorio de instrucciones

## Multiplicación y división (ii)

- La **división entera** de 2 datos de 32 bits da lugar a dos resultados: el cociente y el resto, ambos de 32 bits.
  - Por ello existen **2 tipos de instrucciones distintas**: una para obtener el **cociente** y otra el **resto**.
  - Cada una con variantes para operar con **datos con y sin signo**.
  - Todos los **operandos** de estas instrucciones se encuentran en **registros**.



Registros

Registros



# Repertorio de instrucciones

## Multiplicación y división (iii)

Instrucción	Operación	Descripción
<b>mul</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1 * rs2)_{31:0}$	<b>m</b> ultiply multiplicación entera (32b menos significativos)
<b>mulh</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1_s * rs2_s)_{63:32}$	<b>m</b> ultiply <b>h</b> igh multiplicación entera con signo (32b más significativos)
<b>mulhsu</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1_s * rs2_u)_{63:32}$	<b>m</b> ultiply <b>h</b> igh <b>s</b> igned <b>u</b> nsigned multiplicación entera mixta (32b más significativos)
<b>mulhu</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1_u * rs2_u)_{63:32}$	<b>m</b> ultiply <b>h</b> igh <b>u</b> nsigned multiplicación entera sin signo (32b más significativos)
<hr/>		
<b>div</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1 /_s rs2)$	<b>d</b> ivide división entera con signo
<b>divu</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1 /_u rs2)$	<b>d</b> ivide <b>u</b> nsigned división entera sin signo
<b>rem</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1 \%_s rs2)$	<b>r</b> emainder resto entero con signo
<b>remu</b> <i>rd, rs1, rs2</i>	$rd \leftarrow (rs1 \%_u rs2)$	<b>r</b> emainder <b>u</b> nsigned resto entero sin signo

- Estas instrucciones **no forman parte del repertorio RV32I** pero sí de su extensión RVM.







# Repertorio de instrucciones

## Lógicas (ii)

- Las **operaciones de tipo lógico bit a bit**, se usan para **manipular los bits individuales** de un dato.
  - Un **operando** contiene el **dato a manipular**.
  - Otro **operando** contiene una **máscara** que indica los bits a cambiar.
  - Se utilizará una operación distinta según el cambio que se desee.

```
00001111010100000000101000110100 <----- dato
or  000000000000000000000000000011111111 <----- máscara
-----
00001111010100000000101011111111 <----- dato manipulado
```

la instrucción **or** pone a 1 aquellos bits del dato  
cuyos correspondientes en la **máscara** estén a 1

```
00001111010100000000101000110100
and 000000000000000000000000000011111111
-----
000000000000000000000000000000110100
```

la instrucción **and** pone a 0 aquellos bits del dato  
cuyos correspondientes en la **máscara** estén a 0

```
00001111010100000000101000110100
xor 000000000000000000000000000011111111
-----
00001111010100000000101011001011
```

la instrucción **xor** complementa aquellos bits del dato  
cuyos correspondientes en la **máscara** estén a 1





# Repertorio de instrucciones

## Lógicas (iii)

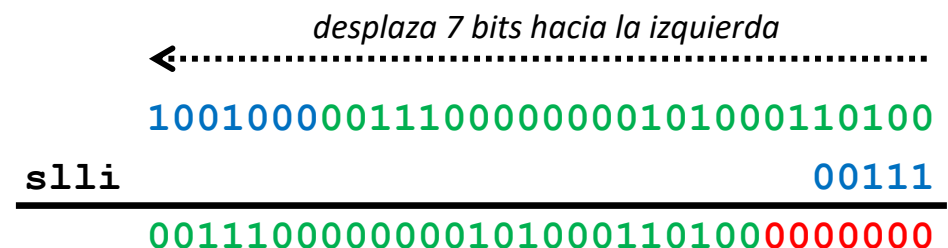
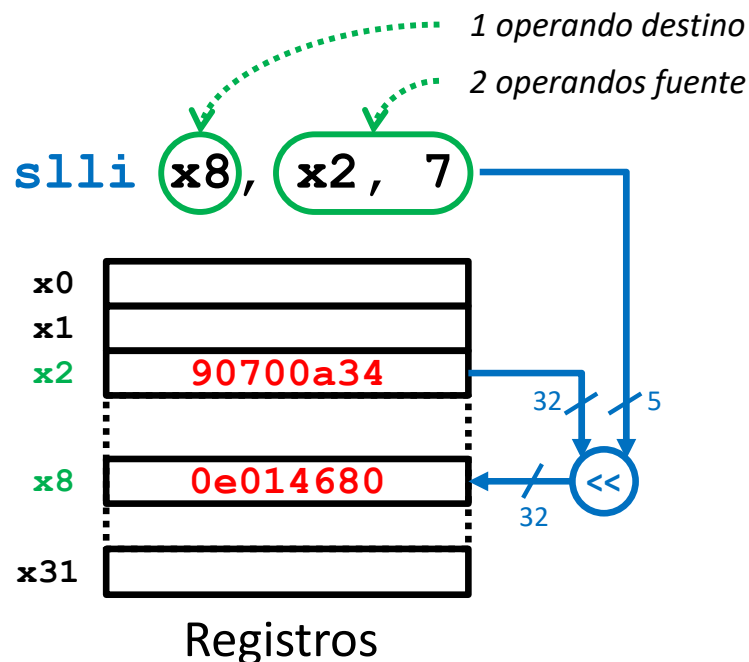
Instrucción	Operación	Descripción
<b>and</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \& rs2$	<b>and</b> "y lógica" bit a bit
<b>or</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \mid rs2$	<b>or</b> "o lógica" bit a bit
<b>xor</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \wedge rs2$	<b>xor</b> "o-exclusiva lógica" bit a bit
<b>andi</b> <i>rd, rs1, imm<sub>12b</sub></i>	$rd \leftarrow rs1 \& sExt(imm)$	<b>and</b> immediate "y lógica" bit a bit con constante
<b>ori</b> <i>rd, rs1, imm<sub>12b</sub></i>	$rd \leftarrow rs1 \mid sExt(imm)$	<b>or</b> immediate "o lógica" bit a bit con constante
<b>xori</b> <i>rd, rs1, imm<sub>12b</sub></i>	$rd \leftarrow rs1 \wedge sExt(imm)$	<b>xor</b> immediate "o-exclusiva lógica" bit a bit con constante



# Repertorio de instrucciones

## De desplazamiento (i)

- Permiten **desplazar los bits** de un operando fuente un número de posiciones indicadas por otro y almacenarlo en un operando destino.
  - El **operando izquierdo de 32b** se encuentra siempre en un **registro**.
  - El **operando derecho de 5b** se encuentra en un **registro** o es un **inmediato**.
    - Del **registro** se toman los **5b menos significativos**.
    - La **constante inmediata** son **5b en binario puro** que no se extienden.
  - El **resultado** se almacena siempre en un **registro**.

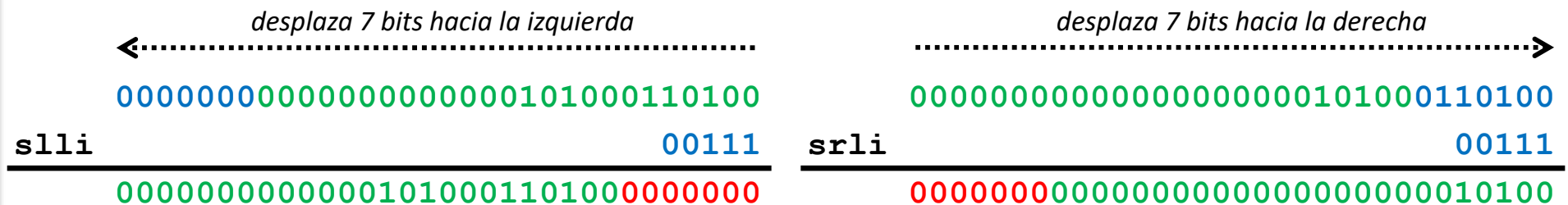




# Repertorio de instrucciones

## De desplazamiento (ii)

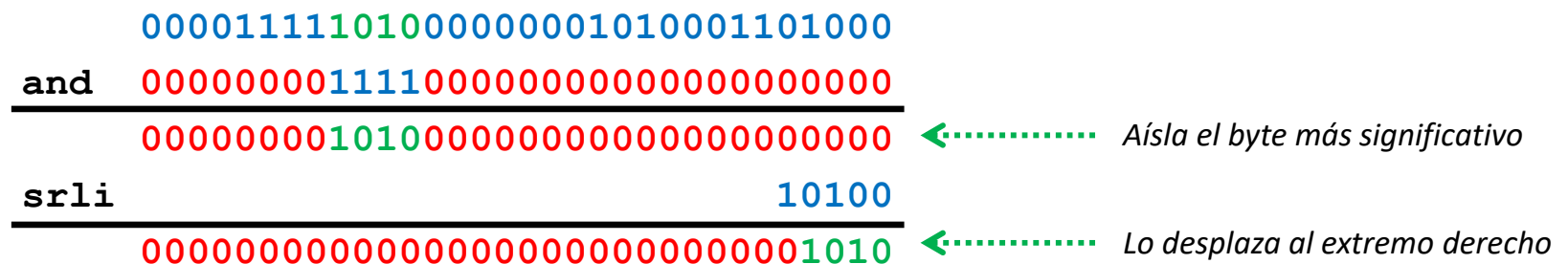
- Las instrucciones de **desplazamiento lógico** insertan 0 por un extremo del dato y descartan los bits que salen por el otro extremo.
  - Permiten el **reescalado** de datos **sin signo**:
    - Desplazar  $n$  bits a la izquierda equivale a **multiplicar por  $2^n$**
    - Desplazar  $n$  bits a la derecha equivale a **dividir por  $2^n$**



$$2612 \ll 7 = 2612 \times 2^7 = 334336$$

$$2612 \gg 7 = 2612 \div 2^7 = 20$$

- Aplicadas tras una lógica bit a bit permiten **extraer campos** de un dato:





# Repertorio de instrucciones

## De desplazamiento (iii)

- La instrucción de **desplazamiento aritmético a la derecha** propaga el bit de signo por la izquierda y descarta los bits que salen por la derecha.
  - Permite el **reescalado** de datos **con signo**:

.....desplaza 7 bits hacia la derecha.....>

```
11111111111111110001101000110100
srai                                00111
-----
1111111111111111111111111000110100
```

$-58828 \gg 7 = -58828 \div 2^7 = -460$

- No existe instrucción de **desplazamiento aritmético a izquierda**
  - Para resultados válidos (el dato reescalado con signo puede representarse con 32b), el desplazamiento lógico es equivalente.

.....desplaza 7 bits hacia la izquierda.....<

```
111111111111111110001101000110100
slli                                00111
-----
11111111100011010001101000000000
```

$-58828 \ll 7 = -58828 \times 2^7 = -7529984$



# Repertorio de instrucciones

## De desplazamiento (iv)

versión 15/01/23

tema 2:  
Arquitectura del procesador

FC-2

29

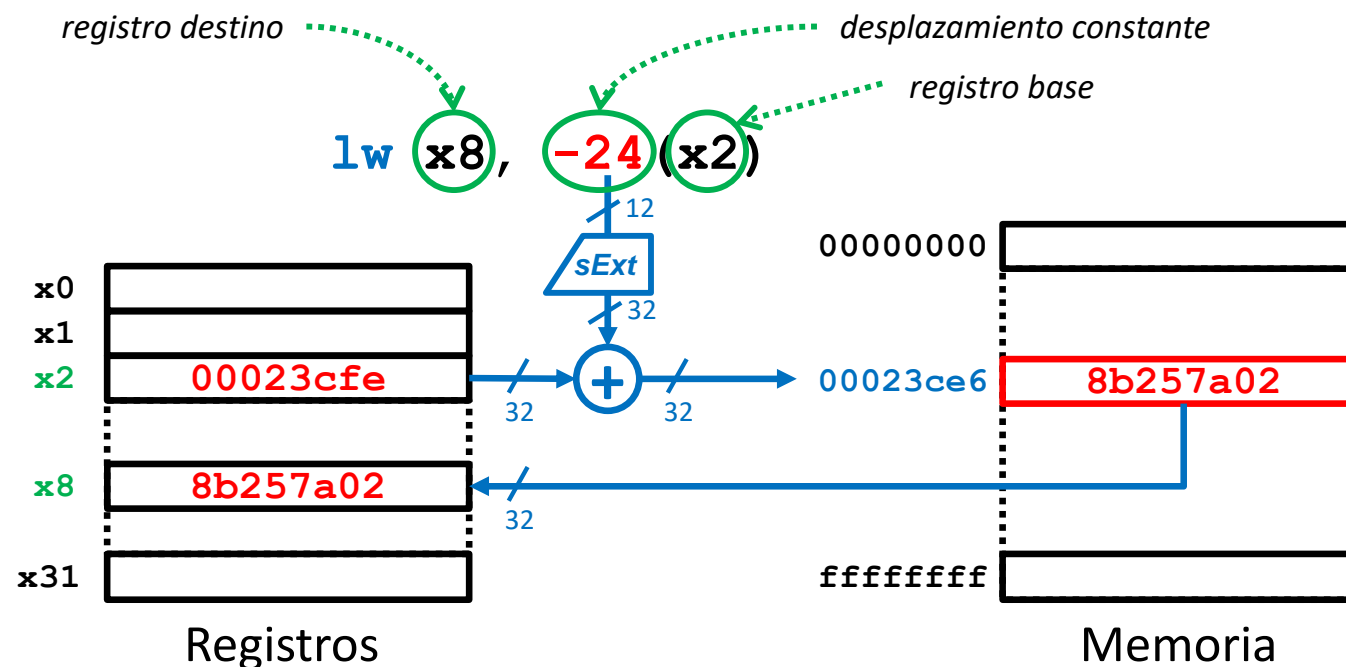
Instrucción	Operación	Descripción
<b>sll</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \ll rs2_{4:0}$	shift left logical desplazamiento lógico a izquierda
<b>srl</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \gg rs2_{4:0}$	shift right logical desplazamiento lógico a derecha
<b>sra</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \ggg rs2_{4:0}$	shift right arithmetical desplazamiento aritmético a derecha
<b>slli</b> <i>rd, rs1, imm<sub>5b</sub></i>	$rd \leftarrow rs1 \ll imm$	shift left logical immediate desplazamiento lógico a izquierda con constante
<b>srl</b> <i>rd, rs1, imm<sub>5b</sub></i>	$rd \leftarrow rs1 \gg imm$	shift right logical immediate desplazamiento lógico a derecha con constante
<b>srai</b> <i>rd, rs1, imm<sub>5b</sub></i>	$rd \leftarrow rs1 \ggg imm$	shift right arithmetical immediate desplazamiento aritmético a derecha con constante



# Repertorio de instrucciones

## De transferencia de datos: carga

- Permiten copiar un dato de memoria a un registro.
  - Utiliza direccionamiento indirecto con registro base para indicar la dirección de memoria que ocupa el dato.
    - Dicha dirección es la suma de una dirección base y un desplazamiento.
    - La dirección base se encuentra en un registro.
    - El desplazamiento es un inmediato de 12b en C2 cuyo signo se extiende a 32b.
  - El dato leído de memoria se carga en un registro.

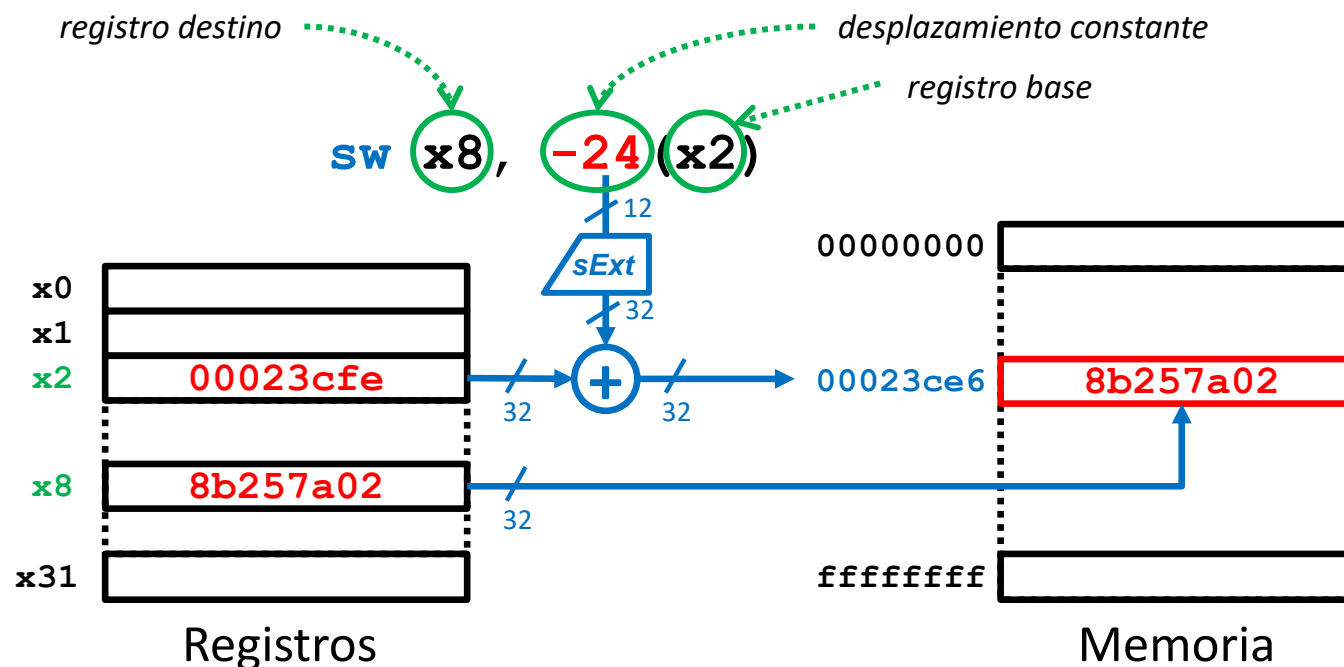




# Repertorio de instrucciones

## De transferencia de datos: almacenaje

- Permiten copiar un dato de un registro a memoria.
  - El dato se encuentra en un registro.
  - Utiliza direccionamiento indirecto con registro base para indicar la dirección de memoria en donde almacenar el dato.
    - Dicha dirección es la suma de una dirección base y un desplazamiento.
    - La dirección base se encuentra en un registro.
    - El desplazamiento es un inmediato de 12b en C2 cuyo signo se extiende a 32b.





# Repertorio de instrucciones

## De transferencia de datos (i)

Instrucción	Operación	Descripción
<b>lw</b> <i>rd, imm<sub>12b</sub>(rs1)</i>	$rd \leftarrow \text{Mem}[rs1 + \text{sExt}(imm)]$	load <b>w</b> ord carga palabra
<b>lh</b> <i>rd, imm<sub>12b</sub>(rs1)</i>	$rd \leftarrow \text{sExt}(\text{Mem}[rs1 + \text{sExt}(imm)]_{15:0})$	load <b>h</b> alf carga media palabra con signo
<b>lhu</b> <i>rd, imm<sub>12b</sub>(rs1)</i>	$rd \leftarrow \text{zExt}(\text{Mem}[rs1 + \text{sExt}(imm)]_{15:0})$	load <b>h</b> alf <b>u</b> nsigned carga media palabra sin signo
<b>lb</b> <i>rd, imm<sub>12b</sub>(rs1)</i>	$rd \leftarrow \text{sExt}(\text{Mem}[rs1 + \text{sExt}(imm)]_{7:0})$	load <b>b</b> yte carga byte con signo
<b>lbu</b> <i>rd, imm<sub>12b</sub>(rs1)</i>	$rd \leftarrow \text{zExt}(\text{Mem}[rs1 + \text{sExt}(imm)]_{7:0})$	load <b>b</b> yte <b>u</b> nsigned carga byte sin signo
<b>sw</b> <i>rs2, imm<sub>12b</sub>(rs1)</i>	$\text{Mem}[rs1 + \text{sExt}(imm)] \leftarrow rs2$	store <b>w</b> ord almacena palabra
<b>sh</b> <i>rs2, imm<sub>12b</sub>(rs1)</i>	$\text{Mem}[rs1 + \text{sExt}(imm)]_{15:0} \leftarrow rs2_{15:0}$	store <b>h</b> alf almacena media palabra
<b>sb</b> <i>rs2, imm<sub>12b</sub>(rs1)</i>	$\text{Mem}[rs1 + \text{sExt}(imm)]_{7:0} \leftarrow rs2_{7:0}$	store <b>b</b> yte almacena byte

- Existen **instrucciones diferentes** para copiar datos de **8b**, **16b** o **32b**
- Ídem para hacerlo extendiendo el signo o completando con ceros.

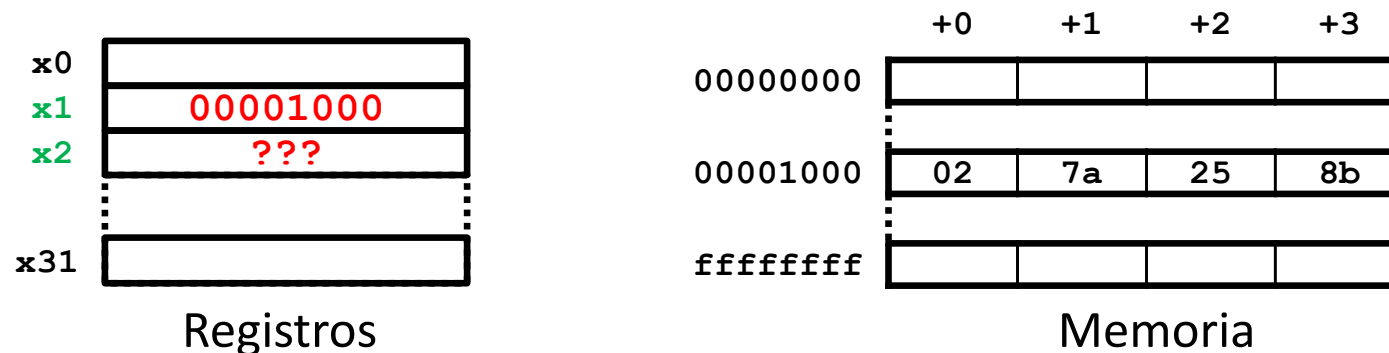




# Repertorio de instrucciones

## De transferencia de datos (ii)

- Los datos en memoria están alineados y con orden Little-Endian



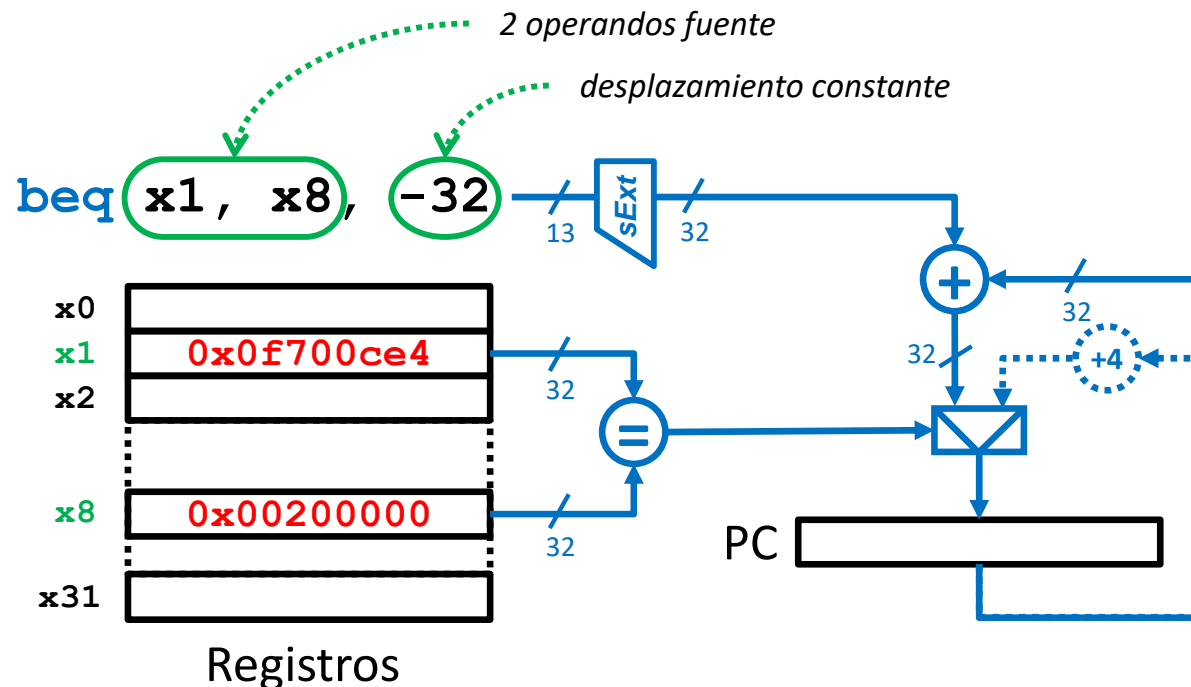
Instrucción	Operación	Resultado
<b>lw</b> x2, 0(x1)	$x2 \leftarrow \text{Mem}[x1 + \text{sExt}(0)]$	carga 8b257a02 en x2
<b>lhu</b> x2, 0(x1)	$x2 \leftarrow \text{zExt}(\text{Mem}[x1 + \text{sExt}(0)]_{15:0})$	carga 00007a02 en x2 ( $7a02 =_2 +31234$ )
<b>lhu</b> x2, 2(x1)	$x2 \leftarrow \text{zExt}(\text{Mem}[x1 + \text{sExt}(2)]_{15:0})$	carga 00008b25 en x2 ( $8b25 =_2 +35621$ )
<b>lh</b> x2, 0(x1)	$x2 \leftarrow \text{sExt}(\text{Mem}[x1 + \text{sExt}(0)]_{15:0})$	carga 00007a02 en x2 ( $7a02 =_{C2} +31234$ )
<b>lh</b> x2, 2(x1)	$x2 \leftarrow \text{sExt}(\text{Mem}[x1 + \text{sExt}(2)]_{15:0})$	carga ffff8b25 en x2 ( $8b25 =_{C2} -29915$ )
<b>lbu</b> x2, 3(x1)	$x2 \leftarrow \text{zExt}(\text{Mem}[x1 + \text{sExt}(3)]_{7:0})$	carga 0000008b en x2 ( $8b =_2 +139$ )
<b>lb</b> x2, 3(x1)	$x2 \leftarrow \text{sExt}(\text{Mem}[x1 + \text{sExt}(3)]_{7:0})$	carga ffffffff8b en x2 ( $8b =_{C2} -177$ )
<b>lh</b> x2, 3(x1)	$x2 \leftarrow \text{sExt}(\text{Mem}[x1 + \text{sExt}(3)]_{7:0})$	al ejecutarse provoca error de alineamiento



# Repertorio de instrucciones

## De salto condicional (i)

- Permiten romper la secuencia normal de ejecución **saltando a una dirección cercana** cuando **se cumple cierta condición**.
  - Compara **2 operandos fuente** que se encuentran en **registros**.
  - Utiliza **direccionamiento relativo al PC** para indicar la dirección con que actualizar el PC en caso de ser cierta la comparación.
    - Dicha dirección es la suma del **contenido del PC** y un desplazamiento corto.
    - El **desplazamiento** es una constante de **13b en C2** cuyo signo se extiende a 32b





# Repertorio de instrucciones

## De salto condicional (ii)

- Las **instrucciones de salto condicional** se usan para implementar en ensamblador **estructuras de control** de tipo *if, while, for...*

**C/C++**

```
...  
if( a != b )  
    a = a + 1;  
    c = c - b;  
...
```

*dir*  
*dir+4*  
*dir+8*

**ASM**

```
...  
beq    x5, x6, 8  
addi   x5, x5, 1  
sub     x7, x7, x6  
...
```

cuando se ejecuta **beq**  
el PC contiene su dirección,  
sumando 8 al PC se salta a **sub**  
(solo si la comparación es cierta)

$a \rightarrow x5$     $b \rightarrow x6$     $c \rightarrow x7$

Vinculación de variables C con registros del RSIC-V

- El **desplazamiento inmediato** (que se suma al PC para efectuar el salto):
  - Al **tener signo**, permite hacer **saltos hacia adelante o hacia atrás**.
  - Al ser de **13 bits** y cada instrucción ocupar 4B, **permite saltar hasta 1024 instrucciones hacia atrás y 1023 hacia adelante** de la instrucción de salto.
    - Una contante de 13b en C2 está en el rango  $[-4096, +4095]$
  - Sus **2 bits menos significativos son 0**, por estar las instrucciones alineadas
    - De hecho, el menos significativo no se almacena en la instrucción.



# Repertorio de instrucciones

## De salto condicional (iii)

Instrucción	Operación	Descripción
<b>beq</b> <i>rs1, rs2, imm<sub>13b</sub></i>	<i>if ( rs1 = rs2 ) then ( PC ← PC + sExt(imm<sub>12:1</sub> &lt;&lt; 1) )</i>	<b>b</b> branch if <b>e</b> qual salta si “igual que”
<b>bne</b> <i>rs1, rs2, imm<sub>13b</sub></i>	<i>if ( rs1 ≠ rs2 ) then ( PC ← PC + sExt(imm<sub>12:1</sub> &lt;&lt; 1) )</i>	<b>b</b> branch if <b>n</b> ot <b>e</b> qual salta si “distinto que”
<b>blt</b> <i>rs1, rs2, imm<sub>13b</sub></i>	<i>if ( rs1 &lt;<sub>s</sub> rs2 ) then ( PC ← PC + sExt(imm<sub>12:1</sub> &lt;&lt; 1) )</i>	<b>b</b> branch if <b>l</b> ess <b>t</b> han salta si “menor que” con signo
<b>bge</b> <i>rs1, rs2, imm<sub>13b</sub></i>	<i>if ( rs1 ≥<sub>s</sub> rs2 ) then ( PC ← PC + sExt(imm<sub>12:1</sub> &lt;&lt; 1) )</i>	<b>b</b> branch if <b>g</b> reater than or <b>e</b> qual salta si “mayor o igual que” con signo
<b>bltu</b> <i>rs1, rs2, imm<sub>13b</sub></i>	<i>if ( rs1 &lt;<sub>u</sub> rs2 ) then ( PC ← PC + sExt(imm<sub>12:1</sub> &lt;&lt; 1) )</i>	<b>b</b> branch if <b>l</b> ess <b>t</b> han <b>u</b> nsigned salta si “menor que” sin signo
<b>bgeu</b> <i>rs1, rs2, imm<sub>13b</sub></i>	<i>if ( rs1 ≥<sub>u</sub> rs2 ) then ( PC ← PC + sExt(imm<sub>12:1</sub> &lt;&lt; 1) )</i>	<b>b</b> branch if <b>g</b> reater than or <b>e</b> qual <b>u</b> nsigned salta si “mayor o igual que” sin signo

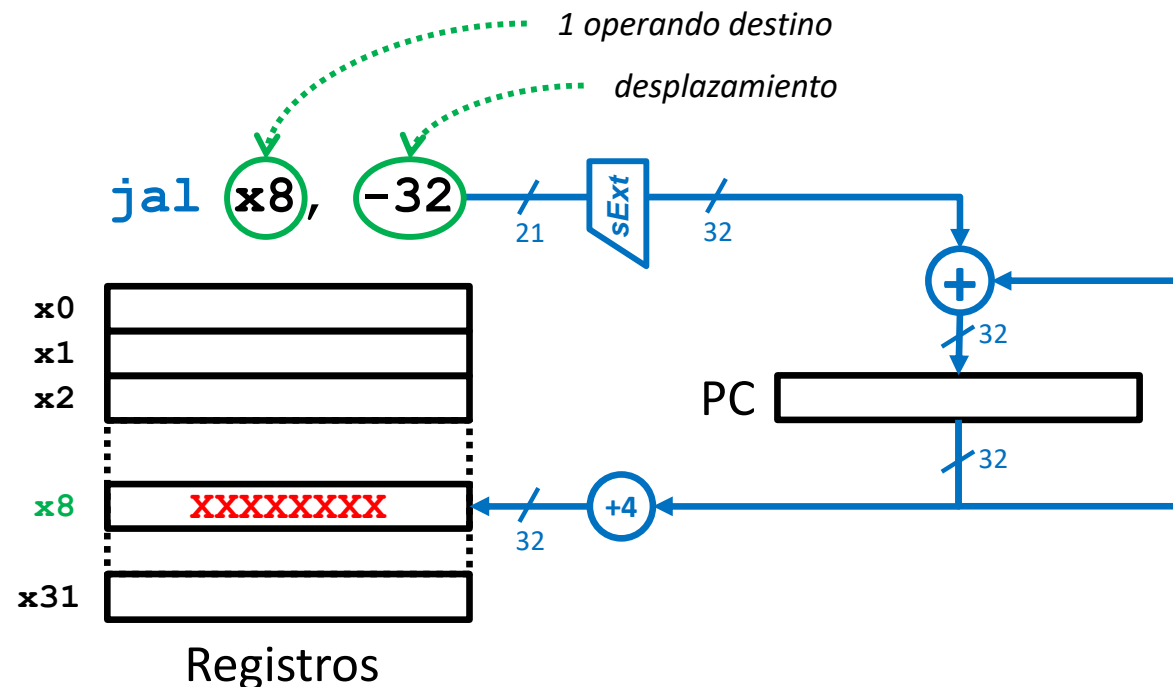
- Existen **instrucciones diferentes** para interpretar los operandos fuente que se comparan como **datos con y sin signo**.
- **No existen** instrucciones de **salto** con un **operando inmediato**.
- **No existen** instrucciones “**mayor que**” ni “**menor o igual que**”, porque basta con usar las existentes cambiando el orden de los operandos.



# Repertorio de instrucciones

## De salto a función: **jal**

- Permiten romper la secuencia normal de ejecución **saltando a una dirección lejana**, pero **guardando la dirección de retorno**.
  - Utiliza **direccionamiento relativo al PC** para indicar la dirección con que actualizar el PC.
    - Dicha dirección es la suma del **contenido del PC** y un desplazamiento largo.
    - El **desplazamiento** es una constante de **21b en C2** cuyo signo se extiende a 32b.
  - La **dirección de la siguiente instrucción** (retorno) se almacena en un **registro**.

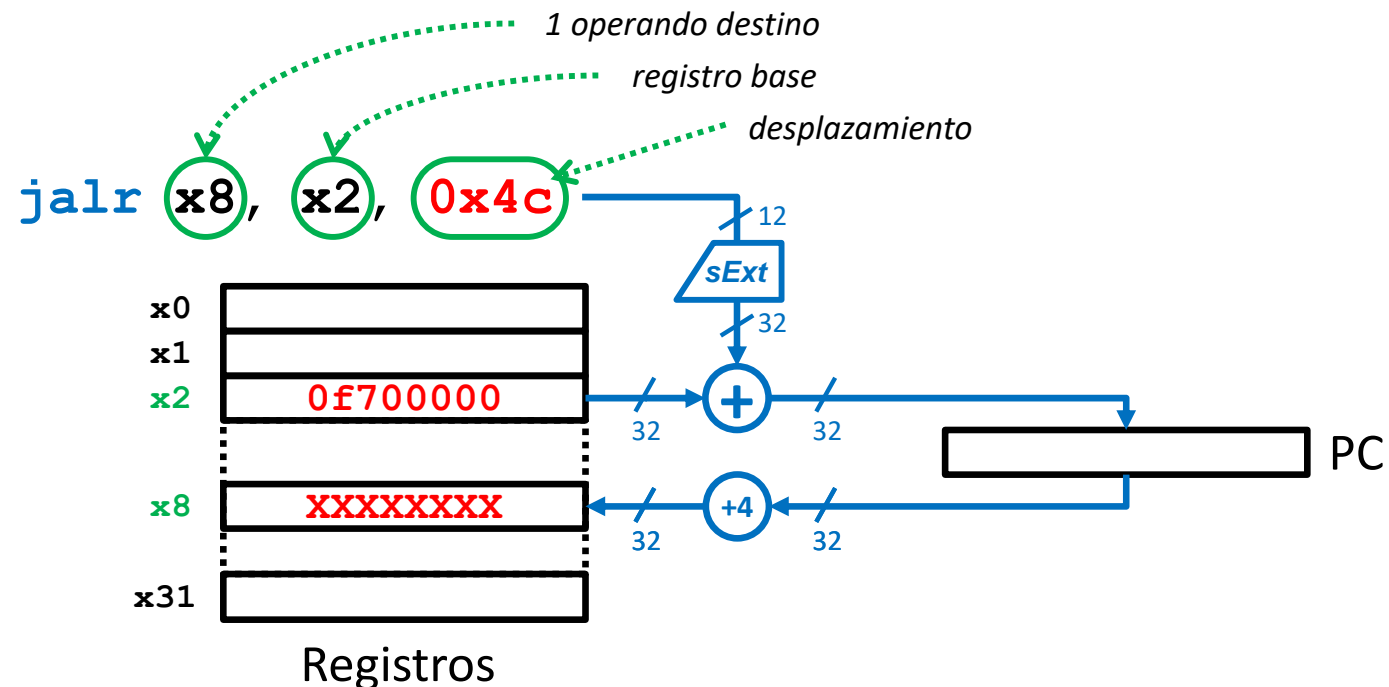




# Repertorio de instrucciones

## De salto a función: **jalr**

- Permiten romper la secuencia normal de ejecución **saltando a una dirección**, pero **guardando la dirección de retorno**.
  - Utiliza **direccionamiento con registro base** para indicar la dirección con que actualizar el PC.
    - Dicha dirección es la suma del **contenido del registro** y un desplazamiento corto.
    - El **desplazamiento** es una constante de **12b en C2** cuyo signo se extiende a 32b.
  - La **dirección de la siguiente instrucción** (retorno) se almacena en un **registro**.





# Repertorio de instrucciones

## De salto a función (i)

- Las **instrucciones de salto a función** se usan para implementar en ensamblador las **llamadas a funciones** y procedimientos.
  - Cada una de **las funciones que forman un programa** se encuentra ubicada a partir de una **dirección distinta** de memoria.
  - Llamar a una función supone **saltar a la dirección de su primera instrucción**.
  - Retornar de una función supone **saltar a la dirección de la instrucción siguiente** a la que hizo la llamada a la función.

### Lenguaje C

```
...  
a = b - c;  
foo();  
c = c + 1;  
...
```

$a \rightarrow x5$   $b \rightarrow x6$   $c \rightarrow x7$

### Ensamblador RISC-V

*dir-4*  
*dir*  
*dir+4*

```
...  
sub    x5, x6, x7  
jal    x1, 1000  
addi   x7, x7, 1  
...
```

cuando se ejecuta `jal` el PC contiene su dirección, se almacena PC+4 en `x1` y se suma 1000 al PC para saltar a la función.

Vinculación de variables C con registros del RISC-V





# Repertorio de instrucciones

## De salto a función (ii)

Instrucción	Operación	Descripción
<code>jalr rd, rs1, imm<sub>12b</sub></code>	$PC \leftarrow rs1 + sExt(imm), rd \leftarrow PC+4$	jump and link register salto a función con dirección relativa a registro base
<code>jal rd, imm<sub>21b</sub></code>	$PC \leftarrow PC + sExt(imm_{20:1} \ll 1), rd \leftarrow PC+4$	jump and link salto a función con dirección relativa a PC

- En el repertorio del RISC-V **no existen** instrucciones **de retorno desde función**, ni de **salto incondicional** a una instrucción, pero pueden usarse éstas para hacerlo:
  - Suponiendo que la dirección de retorno está almacenada en el registro **xn**, el retorno se hace con: `jalr x0, xn, 0`
  - El salto incondicional (relativo al PC) a la dirección que ocupa cierta instrucción se hace con: `jal x0, imm21b`

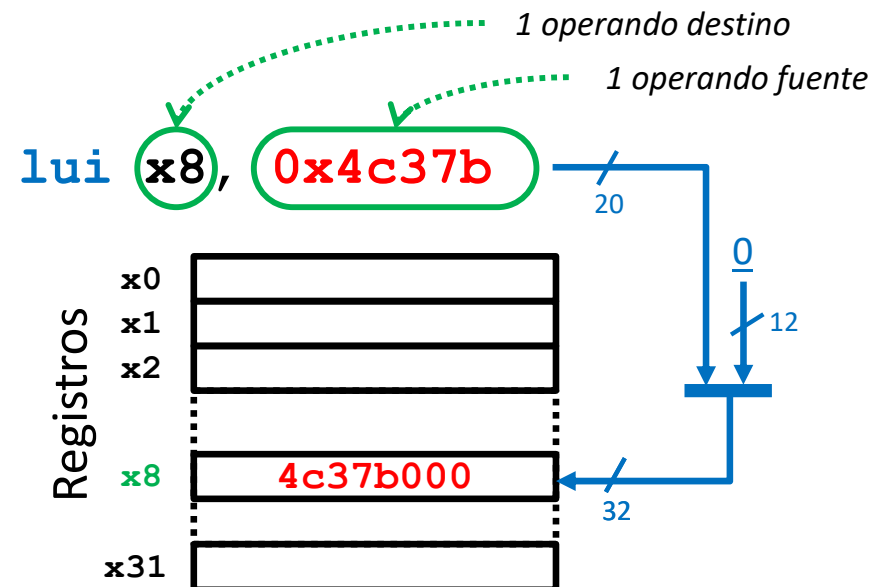




# Repertorio de instrucciones

## Instrucción **lui** (i)

- Permite **cargar una constante** en la parte alta de un registro.
  - El **operando fuente** es una constante inmediata de **20b**.
    - Se extiende hasta 32b rellenando con 12 ceros por la derecha.
  - El **resultado** se almacena en un **registro**.



Instrucción	Operación	Descripción
<b>lui</b> <i>rd</i> , <i>imm</i> <sub>20b</sub>	$rd \leftarrow imm \ll 12$	load <b>u</b> pper <b>i</b> mmediate copia constante desplazada



# Repertorio de instrucciones

## Instrucción `lui` (ii)

- La instrucción `lui` se usa para operar con constantes largas de 32b.
  - Los immediatos en instrucciones aritmético-lógicas son cortos (12b).

**C**

```
...  
a = b + 0xabcde123;  
...
```

*a* → *x5*    *b* → *x6*

Vinculación de variables C con registros

**ASM**

```
...  
lui    x7, 0xabcde  
addi   x7, x7, 0x123  
add     x5, x6, x7  
...
```

←..... x7 = abcde000  
←..... x7 = abcde000  
+ 00000123  
-----  
abcde123

- Dado que la instrucción `addi` extiende el signo del inmediato, si el bit 11 de la constante larga es 1, debe sumarse 1 a la constante usada en `lui`

**C**

```
...  
a = b + 0xabcde987;  
...
```

*a* → *x5*    *b* → *x6*

Vinculación de variables C con registros

**ASM**

```
...  
lui    x7, 0xabcdf  
addi   x7, x7, 0x987  
add     x5, x6, x7  
...
```

←..... x7 = abcdf000  
←..... x7 = abcdf000  
+ fffffff987  
-----  
abcde987



# Repertorio de instrucciones

## Instrucción **lui** (iii)

- Aunque no es muy común, la **instrucción lui** también puede usarse para **trabajar con direcciones absolutas de memoria** de 32b.
  - **Transfiriendo datos** ubicados en cualquier dirección absoluta de memoria.

ASM

```
...  
lui x6, 0x76543  
lw x5, 0x210(x6)  
...
```

*carga en x5 el dato ubicado en la dirección 0x76543210*

ASM

```
...  
lui x6, 0x76543  
sw x5, 0x210(x6)  
...
```

*almacena el dato contenido en x5 en la dirección 0x76543210*

- **Saltando a funciones** ubicadas en cualquier dirección absoluta de memoria.

ASM

```
...  
lui x6, 0x76543  
jalr x1, x6, 0x210  
...
```

*salta a la instrucción ubicada en la dirección 0x76543210*

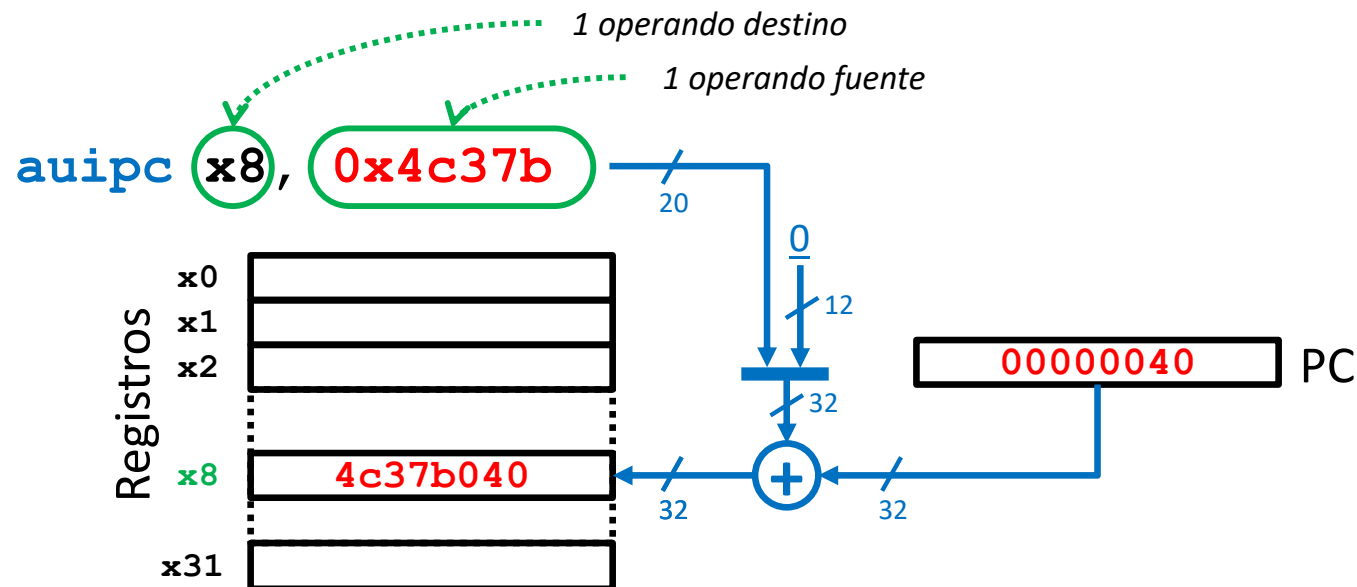
- Dado que las instrucciones **lw**, **sw** y **jalr** también extienden el signo del desplazamiento inmediato, cuando el **bit 11 de la dirección absoluta es 1**, debe **sumarse 1 a la constante usada en lui**



# Repertorio de instrucciones

## Instrucción **auipc** (i)

- Permite **sumar una constante** con la parte alta del PC.
  - El **operando fuente** es una constante inmediata de **20b**.
    - Se extiende hasta 32b rellenando con 12 ceros por la derecha.
  - El **resultado** se almacena en un **registro**.



Instrucción	Operación	Descripción
<b>auipc</b> <i>rd</i> , <i>imm</i> <sub>20b</sub>	$rd \leftarrow PC + (imm \ll 12)$	<b>add upper immediate to PC</b> suma constante desplazada y PC



# Repertorio de instrucciones

## Instrucción **auipc** (ii)

- La instrucción **jal** aún usando un desplazamiento largo solo permite saltar a funciones ubicadas en un ámbito de  $\pm 1\text{MiB}$ , a veces insuficiente.
  - Al ser el desplazamiento de 21b y cada instrucción ocupar 4B, solo permite saltar hasta 262144 instrucciones hacia atrás y 262143 hacia adelante.
- La instrucción **auipc** junto a **jalr** se utiliza para hacer saltos relativos al PC a funciones ubicadas en cualquier dirección de memoria.
  - Permitiendo abarcar el espacio de direccionamiento completo de  $\pm 4\text{GiB}$



- Como la instrucción **jalr** extiende el signo del inmediato, si el bit 11 del desplazamiento de 32b es 1, debe sumarse 1 a la constante usada en **auipc**



# Repertorio de instrucciones

## Instrucción **auipc** (iii)

- Idéntico caso sucede con las instrucciones de carga y almacenamiento, ya que usan un desplazamiento relativo al PC corto de 12b
  - Sólo pueden acceder a datos ubicados en el ámbito cercano de  $\pm 2\text{KiB}$ .
- Combinando estas instrucciones con **auipc**, se puede acceder a datos ubicados en cualquier dirección de memoria.

*dir* **auipc** x6, 0x76543  
**lw** x5, 0x210(x6)  
...

carga en x5 el dato ubicado en la dirección ( $\text{dir} + 0\text{x76543210}$ )

*dir* **auipc** x6, 0x76543  
**sw** x5, 0x210(x6)  
...

almacena el dato contenido en x5 en la dirección ( $\text{dir} + 0\text{x76543210}$ )

- Ídem que en casos anteriores, si el bit 11 del desplazamiento de 32b relativo al PC es 1, debe sumarse 1 a la constante usada en **auipc**



# Repertorio de instrucciones

## Instrucciones más populares

- Existen instrucciones que se ejecutan con más frecuencia que otras
  - Se comprueba contando las veces que se ejecuta cada instrucción al ejecutar un conjunto de programas estándar (SPEC CPU2006)

Instrucción	Descripción	Frecuencia	Acumulada
<b>lw</b>	<i>carga</i>	19.48%	19.48%
<b>addi</b>	<i>suma inmediata</i>	17.22%	36.70%
<b>sw</b>	<i>almacenaje</i>	8.05%	44.75%
<b>add</b>	<i>suma</i>	7.57%	52.32%
<b>bne</b>	<i>salta si “distinto que”</i>	4.14%	56.46%
<b>slli</b>	<i>desplazamiento lógico a izquierda</i>	3.65%	60.11%
<b>beq</b>	<i>salta si “igual que”</i>	3.27%	63.38%
<b>mul</b>	<i>multiplicación</i>	2.02 %	65.40%

fuelle (adaptación): D.A. Patterson and J.L. Hennessy. Computer Organization and Design. RISC-V Edition (2021)



# RISC-V: repertorios y extensiones

- RISC-V es una **arquitectura abierta y flexible**.
  - Define diferentes **repertorios base** de instrucciones y **extensiones**.
  - Todo **procesador RISC-V** debe soportar uno de los **repertorios base** y **opcionalmente algunas de las extensiones**.
- **Repertorios base:**
  - **RV32I**: instrucciones y datos/direcciones de 32 bits.
  - **RV32E**: versión del RV32I con solo 16 registros.
  - **RV64I**: instrucciones de 32 bits y datos/direcciones de 64 bits.
  - **RV128I**: instrucciones de 32 bits y datos/direcciones de 128 bits.
- **Extensiones:**
  - **RVM**: añade la multiplicación, división y resto enteros.
  - **RVF**: añade 32 registros para datos en punto flotante de 32 bits así como operaciones aritméticas, relacionales y de conversión en punto flotante.
  - **RVD**: versión para datos en punto flotante de 64 bits (doble precisión).
  - **RVQ**: versión para datos en punto flotante de 128 bits (cuádruple precisión).
  - **RVC**: extensión con instrucciones comprimidas de 16 bits.





# RISC-V: repertorios y extensiones

## Ejemplo: repertorio RV64I (i)

- Repertorio base RV64I.
  - Datos enteros de 64 bits e instrucciones de 32 bits.
  - Dispone de 32 registros de 64 bits.
  - Añade instrucciones de transferencias de 64b con memoria.
  - Redefine las instrucciones de transferencia de 32b.
  - Añade una nueva instrucción para carga de datos de 32b sin signo.

Instrucción	Operación	Descripción
<b>ld</b> <i>rd, imm(rs1)</i>	$rd \leftarrow \text{Mem}[rs1 + \text{sExt}(\text{imm}_{12b})]$	carga doble palabra
<b>sd</b> <i>rs2, imm(rs1)</i>	$\text{Mem}[rs1 + \text{sExt}(\text{imm}_{12b})] \leftarrow rs2$	almacena doble palabra

Instrucción	Operación	Descripción
<b>lw</b> <i>rd, imm(rs1)</i>	$rd \leftarrow \text{sExt}(\text{Mem}[rs1 + \text{sExt}(\text{imm}_{12b})]_{31:0})$	carga palabra con signo
<b>lwu</b> <i>rd, imm(rs1)</i>	$rd \leftarrow \text{zExt}(\text{Mem}[rs1 + \text{sExt}(\text{imm}_{12b})]_{32:0})$	carga palabra sin signo
<b>sw</b> <i>rs2, imm(rs1)</i>	$\text{Mem}[rs1 + \text{sExt}(\text{imm}_{12b})] \leftarrow rs_{31:0}$	almacena palabra



# RISC-V: repertorios y extensiones

## Ejemplo: repertorio RV64I (ii)

- Repertorio base RV64I.
  - Las instrucciones aritmético-lógicas trabajan con datos de 64b (los operandos inmediatos siguen siendo de 12b pero extendidos a 64b)
  - Redefine las instrucciones de desplazamiento para trabajar con datos de 64b (requieren 6b para indicar el número de bits a desplazar)

Instrucción	Operación	Descripción
<b>sll</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \ll rs2_{5:0}$	desplazamiento lógico a izquierda
<b>srl</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \gg rs2_{5:0}$	desplazamiento lógico a derecha
<b>sra</b> <i>rd, rs1, rs2</i>	$rd \leftarrow rs1 \ggg rs2_{5:0}$	desplazamiento aritmético a derecha
<b>slli</b> <i>rd, rs1, imm</i>	$rd \leftarrow rs1 \ll imm_{6b}$	desplazamiento lógico a izquierda
<b>srli</b> <i>rd, rs1, imm</i>	$rd \leftarrow rs1 \gg imm_{6b}$	desplazamiento lógico a derecha
<b>srai</b> <i>rd, rs1, imm</i>	$rd \leftarrow rs1 \ggg imm_{6b}$	desplazamiento aritmético a derecha



# RISC-V: repertorios y extensiones

## Ejemplo: repertorio RV64I (iii)

### ■ Repertorio base RV64I.

- Añade instrucciones aritmético-lógicas y de desplazamiento para trabajar con datos de 32b (sufijo w).

Instrucción	Operación	Descripción
<b>addw</b> <i>rd, rs1, rs2</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} + rs2_{31:0})$	suma palabra
<b>subw</b> <i>rd, rs1, rs2</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} - rs2_{31:0})$	resta palabra
<b>addiw</b> <i>rd, rs1, imm</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} + \text{sExt}(\text{imm}_{12b})_{31:0})$	suma palabra

Instrucción	Operación	Descripción
<b>sllw</b> <i>rd, rs1, rs2</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} \ll rs2_{4:0})$	desplazamiento lógico a izquierda
<b>srlw</b> <i>rd, rs1, rs2</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} \gg rs2_{4:0})$	desplazamiento lógico a derecha
<b>sraw</b> <i>rd, rs1, rs2</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} \ggg rs2_{4:0})$	desplazamiento aritmético a derecha
<b>slliw</b> <i>rd, rs1, imm</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} \ll \text{imm}_{5b})$	desplazamiento lógico a izquierda
<b>srliw</b> <i>rd, rs1, imm</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} \gg \text{imm}_{5b})$	desplazamiento lógico a derecha
<b>sraiw</b> <i>rd, rs1, imm</i>	$rd \leftarrow \text{sExt}(rs1_{31:0} \ggg \text{imm}_{5b})$	desplazamiento aritmético a derecha

# Arquitecturas RISC vs. CISC



- RISC-V es un claro representante de las **arquitecturas tipo RISC**.
  - Otras arquitecturas RISC son: PowerPC, DEC Alpha, MIPS, ARM, SPARC...
  - Es la **arquitectura dominante en dispositivos móviles**.
  - El 75% de los procesadores son de arquitectura ARM.
- Pero también existen arquitecturas con un paradigma completamente opuesto denominadas **tipo CISC** (Complex Instruction Set Computer).
  - Tienen un **repertorio amplio de instrucciones complejas**.
  - Las instrucciones pueden acceder tanto a **datos almacenados en memoria** como a datos ubicados en **registros internos**.
  - Dispone de un **número reducido de registros**, algunos de los cuales tienen un **propósito específico**.
  - Tiene un **amplio conjunto de modos de direccionamiento**.
  - Instrucciones de **tamaño variable** y **gran cantidad de formatos distintos**.
  - Son arquitecturas CISC: Motorola 68K, Intel x86, AMD x86-64...
  - Es la **arquitectura dominante en computadores personales**.



# Arquitecturas RISC vs. CISC

## Arquitectura x86 (i)

- La **arquitectura x86** es el **principal representante de arquitectura CISC**
  - Introducida por Intel en 1978 en sus microprocesadores 8086 y 8088 ha ido evolucionando en sucesivas generaciones.
  - Usada en computadores personales desde el lanzamiento del IBM-PC en 1981.

Característica	RISC-V (RV32I)	x86
Núm. de registros	32 de propósito general	8, algunos con restricciones de uso
Núm. de operandos	3 (2 fuente, 1 destino)	2 (1 fuente, 1 fuente/destino)
Localización de operandos	Registros o inmediatos	Registros, inmediatos o memoria
Tamaño de operando	32 bits	8, 16 o 32 bits
Flags de condición	No	Sí
Número de instrucciones	Reducido	Elevado
Tipos de instrucciones	Simples	Simples y complejas
Codificación de instrucciones	Fija: 4 bytes/instrucción	Variable: de 1 a 15 bytes/instrucción



# Arquitecturas RISC vs. CISC

## Arquitectura x86 (i)

Instrucción	Operación	Suma con...
<b>add</b> AH, BL	$AH \leftarrow AH + BL$	registros de 8b
<b>add</b> AX, -1	$AH \leftarrow AH + 0xffff$	inmediato de 16b
<b>add</b> EAX, EBX	$EAX \leftarrow EAX + EBX$	registros de 32b
<b>add</b> EAX, 42	$EAX \leftarrow EAX + 0x0000002a$	inmediato de 32b
<b>add</b> EAX, [20]	$EAX \leftarrow EAX + Mem[20]$	direccionamiento absoluto
<b>add</b> EAX, [ESP]	$EAX \leftarrow EAX + Mem[ESP]$	direccionamiento registro base
<b>add</b> EAX, [EDX+40]	$EAX \leftarrow EAX + Mem[EDX+40]$	direccionamiento registro base y desplazamiento
<b>add</b> EAX, [60+EDI*4]	$EAX \leftarrow EAX + Mem[60+EDI*40]$	desplazamiento y registro índice escalado
<b>add</b> EAX, [EDX+80+EDI*4]	$EAX \leftarrow EAX + Mem[EDX+80+EDI*40]$	registro base, desplazamiento, y registro índice escalado
<b>add</b> [20], EAX	$Mem[20] \leftarrow Mem[20] + EAX$	registro de 32b almacenada en memoria
<b>add</b> [20], 42	$Mem[20] \leftarrow Mem[20] + 42$	inmediato almacenada en memoria

# Acerca de *Creative Commons*



## ■ Licencia CC (*Creative Commons*)

- Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:



**Reconocimiento** (*Attribution*):

En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.



**No comercial** (*Non commercial*):

La explotación de la obra queda limitada a usos no comerciales.



**Compartir igual** (*Share alike*):

La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: <https://creativecommons.org/licenses/by-nc-sa/4.0/>