Principios del ISA
Lección 4

Prof.Ing. Fabián Zamora Ramírez

CE-4301 Arquitectura de Computadores I

Área de Ingeniería en Computadores

Instituto Tecnológico de Costa Rica

Agenda

ISAs a través de los años

- **CISC**
  - x86 (Intel)
  - PDP-x
  - IBM System/360
  - VAX
- **RISC (PC / Servers)**
  - Digital Alpha
  - MIPS, Inc.
  - Hewlett-Packard PA-RISC
  - Sun Microsystems SPARC
- **RISC (Embebidos)**
  - Advanced RISC Machines ARM
  - Advanced RISC Machines Thumb
  - Hitachi SuperH
  - Mitsubishi M32R
  - MIPS, Inc. MIPS16

Componentes de un set de instrucciones

- **Clase de ISA**: Acceso a memoria / uso de operandos.
  - Registro-Memoria: X86
  - Load / Store: ARM / MIPS
- **Direccionamiento** de memoria: endianness, alineamiento, etc
- **Modos de direccionamiento.**
- **Tipos y tamaño de operandos.**
- **Operaciones.**
- **Control de flujo**
- **Encodificación**

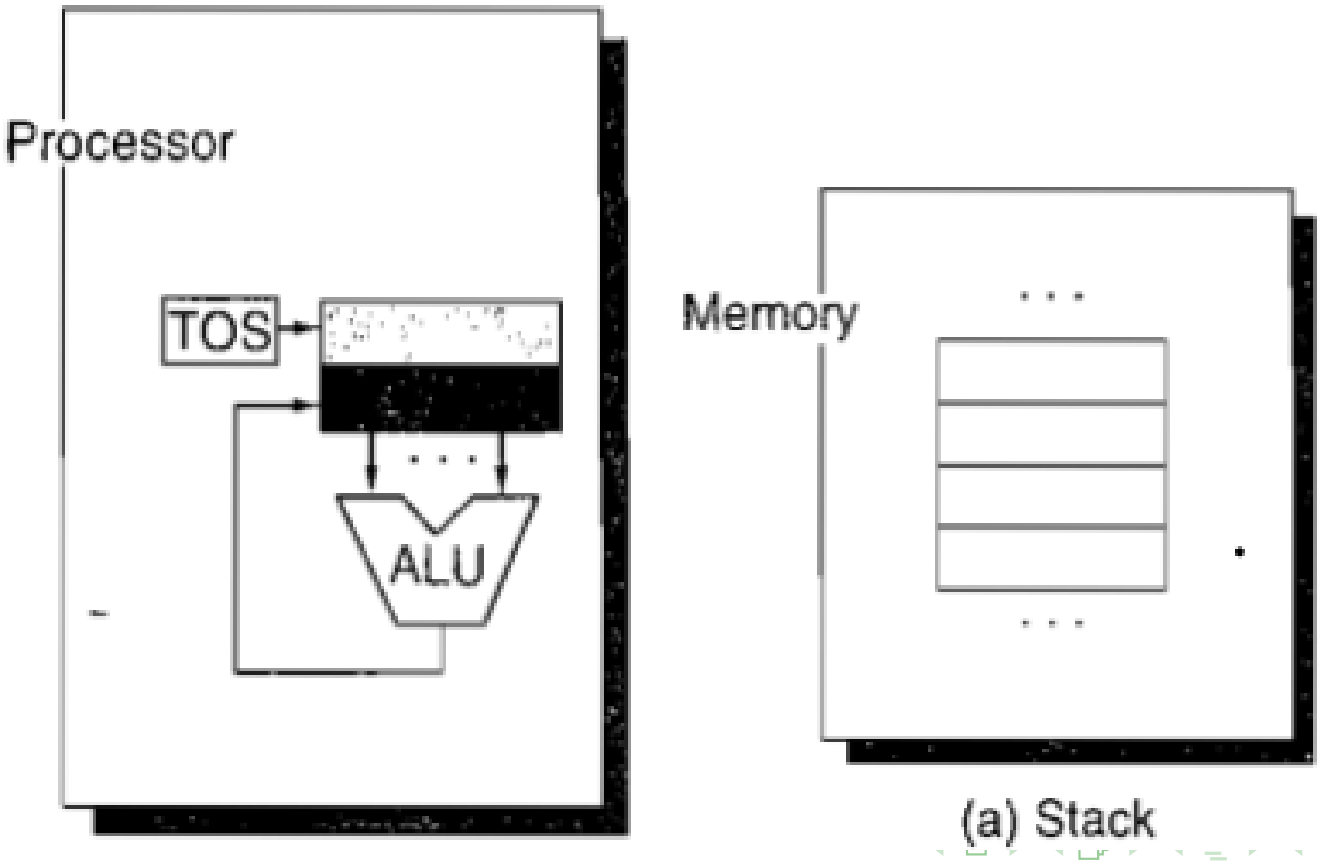
Clases del ISA

Un ISA puede incluirse en distintas clases según la forma en que se almacenan y referencian los operandos. Las 4 clases más importantes de ISA's son:

- Stack
- Acumulador
- Registro-Memoria
- Registro-Registro

Stack - Pila

En la clase Stack, los operandos se encuentran en una **única pila.** Para realizar una operación se debe insertar los operandos (push) y luego extraer el resutado (pop) de la misma.



Stack - operaciones

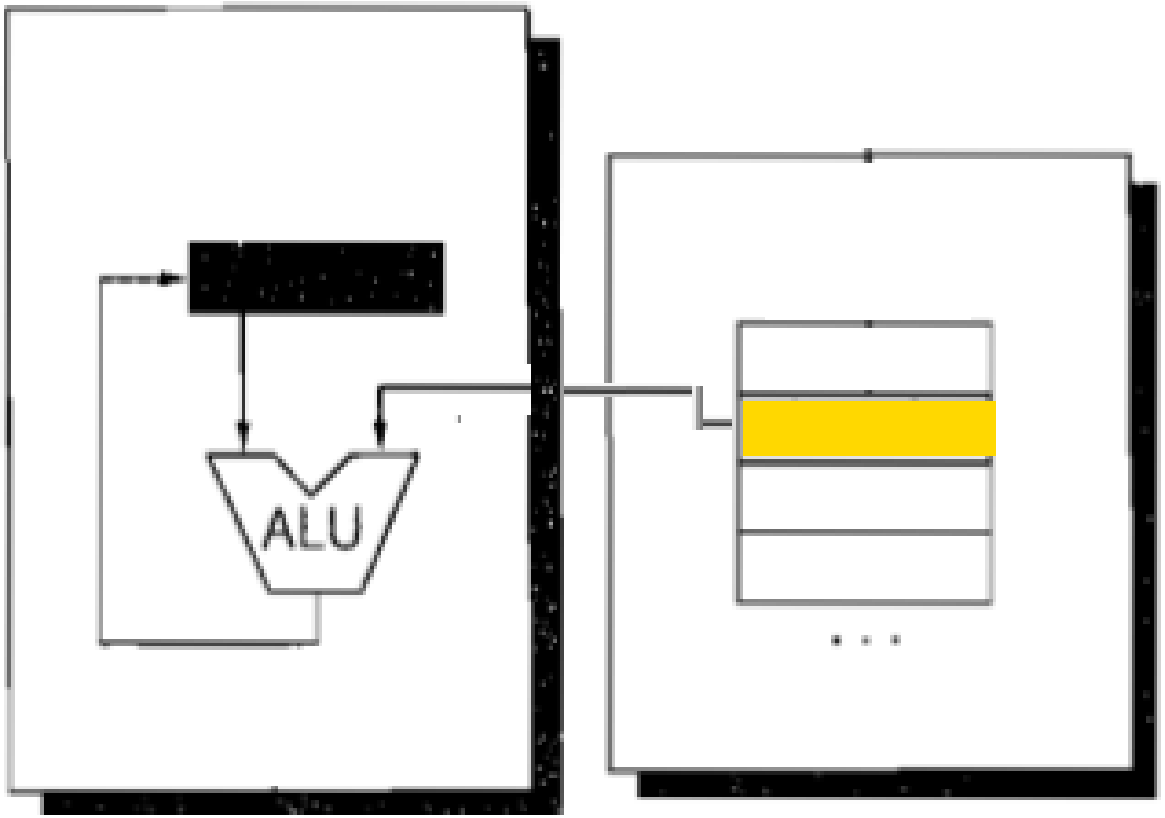
Supongamos operación: A=B+C

```
push B; // insertar operando 1  hacia latch ALU
push C; // insertar operando 2  hacia latch ALU
add;  //suma
pop A; /extraer resultado hacia el TOS
```

Los operandos están **implícitos** en el TOS (no se definen/nombran explícitamente). El **TOS** está se encuentra apuntando a memoria.

Acumulador

En las arquitecturas con acumulador, un operando se encuentra **implícito** en el acumulador y el otro se encuentra directamente en memoria.



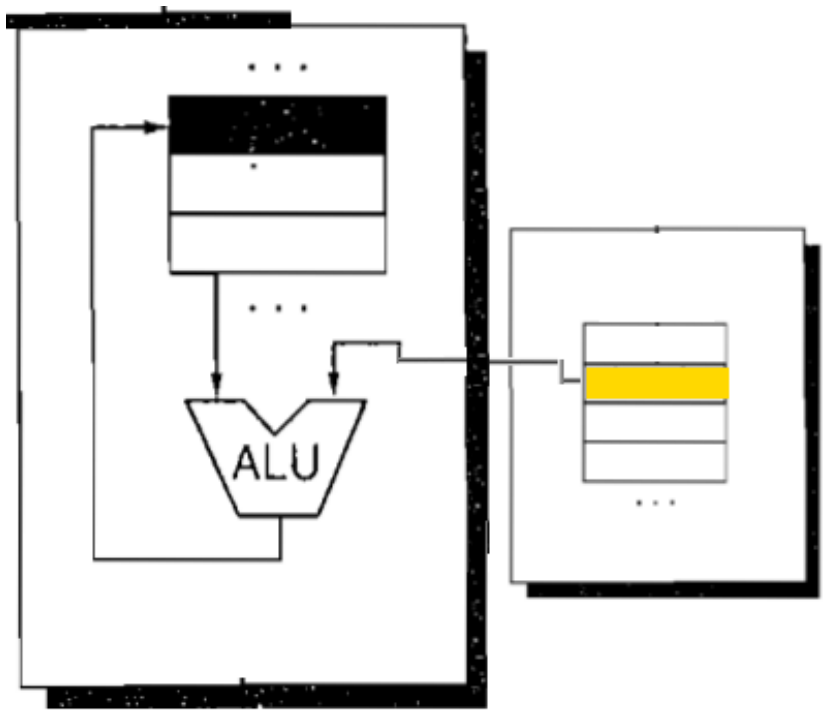
Acumulador - Operaciones

Supongamos operación A=B+C:

```
Load B; // Carga el operando B al acumulador
Add C; //  Suma C (directamente desde memoria) al operando implícito B
Store A ; / Almacena en memoria el resultado (desde el acumulador)
```

Registro (registro-memoria)

En esta arquitectura se cuenta con registros de propósito general (GPRs). Uno de los operados corresponde a un GPR y el otro proviene directamente de memoria.



Registro - operaciones

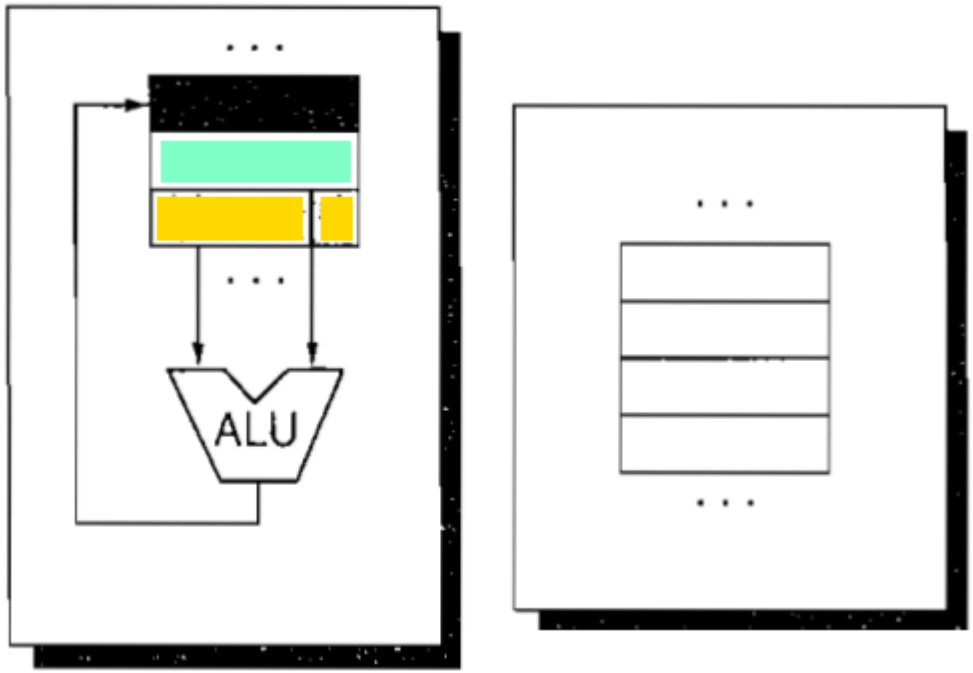
Supongamos operación: A=B+C

```
Load R1, B;// Cargar B en GPR R1
Add  R3, R1, C; //Sumar R1 con C (dir memoria)
Store R3, A; // Almacenar GPR resultado en mem.
```

Ambos operandos son explícitos
La mayoría de arquitecturas CISC utilizan esta clase.

Registro-Registro

Una arquitectura de la clase registro-registro no realiza operaciones directamente desde memoria, sino que las realiza enteramente con operandos en registros de propósito general.



Tipo y tamaño de operandos

El tipo de los operandos dentro de la palabra de instrucción se encuentra, en la mayoría de los casos, en el código de operación (**opcode**)

El tipo de operando, además, define el tamaño.

Tipos comunes

- byte (8bits)
- media palabra (16bits)
- palabra (32 bits) / punto flotante de precisión simple (SP-FP)
- doble palabra (64 bits) / punto flotante de precisión doble (DP-FP)

Enteros: típicamente en representación complemento a 2.
Flotantes: IEEE 754.
Caracteres: ASCII
Decimal: BCD**

Direccionamiento de memoria

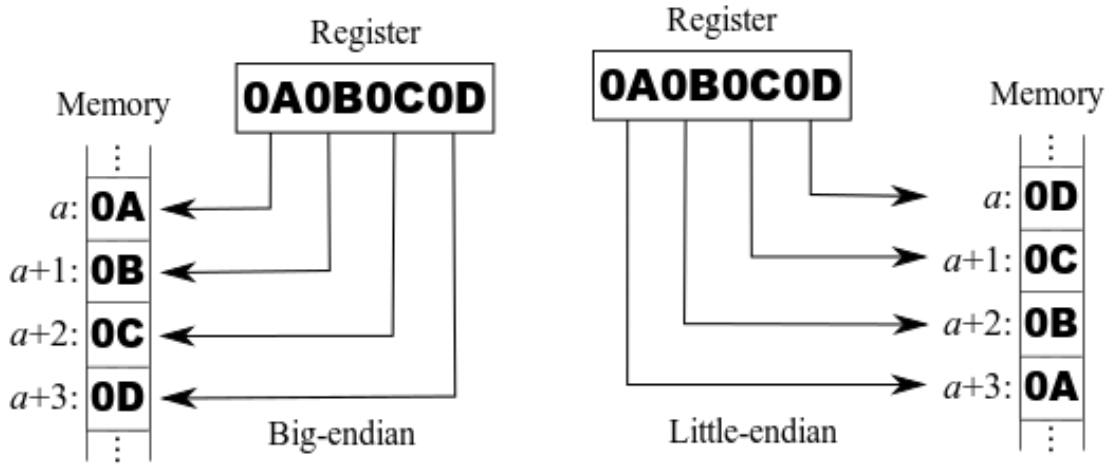
Independientemente de la arquitectura (reg-reg, reg-mem. etc) se debe tener una forma de interpretar y especificar las direcciones de memoria. Existen diferentes formas de acceder a una dirección:

- Nivel de byte (8 bits)
- Nivel de media palabra (16 bits)
- Nivel de palabra / word (32 bits)
- Nivel de doble palabra (64 bits)

Ordenamiento de bytes en memoria

Existen dos formas de ordenar los bytes en una dirección de memoria *(Endianness)*:

- **Little Endian**: Ordena el byte **menos significativo** en la dirección más pequeña de la palabra (doble palabra).
- **Big Endian**: Ordena el byte **más significativo** en la dirección más pequeña de la palabra (doble palabra).



Alineamiento

El acceso se encuentra alineado si $A \bmod s = 0$

	A Value of 3 low-order bits of byte address							
Width of object s	0	1	2	3	4	5	6	7
1 byte (byte)	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 bytes (half word)	Aligned		Aligned		Aligned		Aligned	
2 bytes (half word)		Misaligned		Misaligned		Misaligned		Misaligned
4 bytes (word)	Aligned				Aligned			
4 bytes (word)		Misaligned				Misaligned		
4 bytes (word)			Misaligned				Misaligned	
4 bytes (word)				Misaligned				Misaligned
8 bytes (double word)	Aligned							
8 bytes (double word)		Misaligned						
8 bytes (double word)			Misaligned					
8 bytes (double word)				Misaligned				
8 bytes (double word)					Misaligned			
8 bytes (double word)						Misaligned		
8 bytes (double word)							Misaligned	
8 bytes (double word)								Misaligned

Modos de direccionamiento

Los modos de direccionamiento se refiere a la forma en que las arquitecturas **especifican** la dirección de un objecto que van a acceder.

- **Dirección efectiva**:El valor final de dirección especificado por el modo de direccionamiento.

Modos de direccionamiento

Múltiples modos de direccionar datos dentro de una instrucción:

- Registro
- Inmediato
- Desplazamiento
- Indexado
- Directo
- Indirecto a memoria
- Autoincremento
- Autodecremento
- Escalado

Modo Registro

Cuando los operandos se encuentran en **registros**.

Ejemplo:

```
Add R4, R3
```

Lo que significa que:

```
Regs[R4] <- Regs[R4] + Regs[R3]
```

Modo Inmediato

Modo utilizado para referenciar operaciones con **constantes**.

Ejemplo:

```
Add R4, #3
```

Lo que significa que:

```
Regs[R4] <- Regs[R4] + 3
```

Desplazamiento

El dato referencia se encuentra en la posición de memoria dada por la suma de una base (un inmediato) + un registro.

Ejemplo:

```
Add R4, 200(R1)
```

Lo que significa que:

```
Regs[R4] <- Regs[R4] + Mem[200 + Regs[R1]]
```

Registro Indirecto

La dirección se determina por el contenido de un registro. Útil para utilizar punteros.

Ejemplo:

```
Add R4,(R1)
```

Lo que significa que:

```
Regs[R4] <- Regs[R4] + Mem[Regs[R1]]
```

Indexado : Base + índice

La dirección se calcula como la suma de un registro base + un registro índice. Utilizado para direccionar arreglos.
Ejemplo:

```
Add R4,(R1 + R2)
```

Lo que significa que:

```
Regs[R4] <- Regs[R4] + Mem[Regs[R1] + Regs[R2]]
```

Directo o Absoluto

La dirección efectiva se referencia directamente. Útil para acceder datos estáticos.
Ejemplo:

```
Add R4,(1001)
```

Lo que significa que:

```
Regs[R4] <- Regs[R4] + Mem[1001]
```

Indirecto a Memoria

La dirección efectiva se calcula como el contenido de otra dirección de memoria.
Ejemplo:

```
Add R4,@(R1)
```

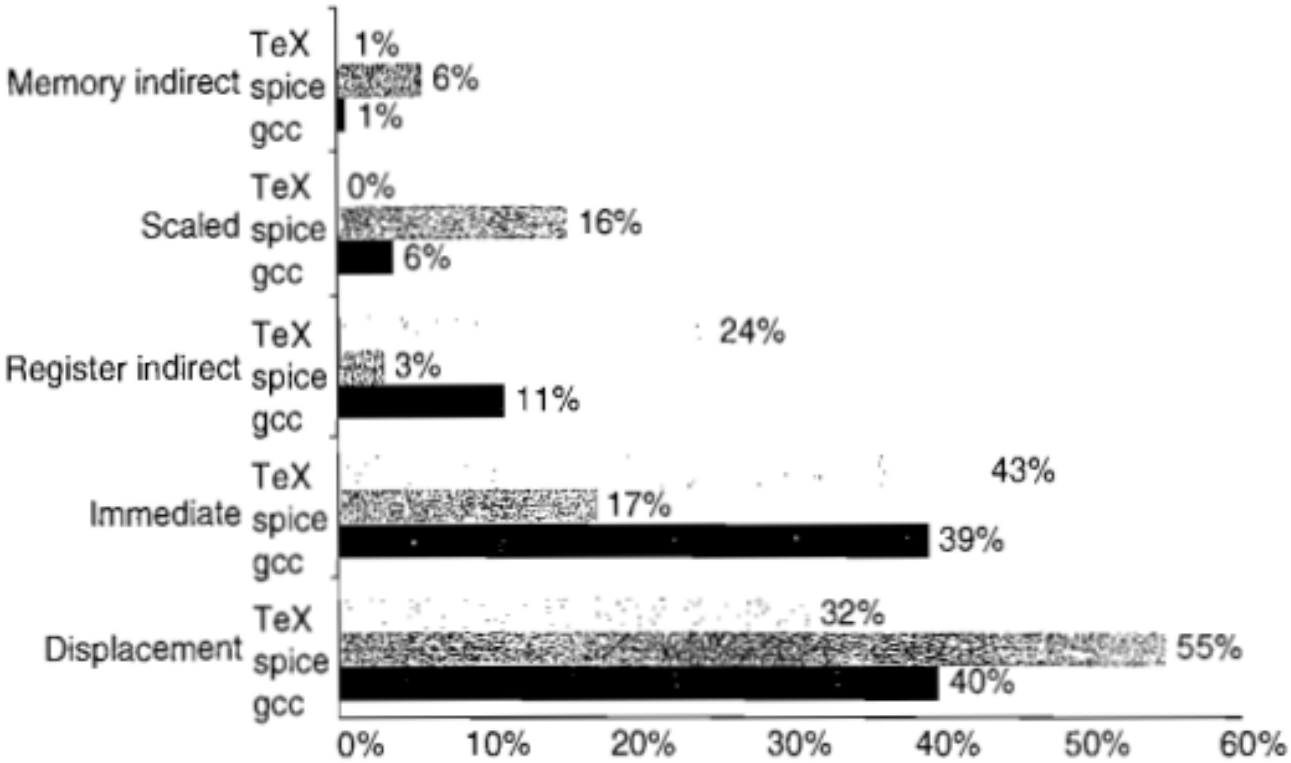
Lo que significa que:

```
Regs[R4] <- Regs[R4] + Mem[ Mem[Regs[R1]]]
```

Modos de direccionamiento - Resumen

Addressing mode	Example instruction	Meaning	When used
Register	`Add R4,R3`	`Regs[R4] ← Regs[R4] + Regs[R3]`	When a value is in a register.
Immediate	`Add R4,#3`	`Regs[R4] ← Regs[R4] + 3`	For constants.
Displacement	`Add R4,100(R1)`	`Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]`	Accessing local variables (+ simulates register indirect, direct addressing modes).
Register indirect	`Add R4,(R1)`	`Regs[R4] ← Regs[R4] + Mem[Regs[R1]]`	Accessing using a pointer or a computed address.
Indexed	`Add R3,(R1 + R2)`	`Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]`	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	`Add R1,(1001)`	`Regs[R1] ← Regs[R1] + Mem[1001]`	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	`Add R1,@(R3)`	`Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]`	If R3 is the address of a pointer p, then mode yields $*p$.
Autoincrement	`Add R1,(R2)+`	`Regs[R1] ← Regs[R1] + Mem[Regs[R2]]` `Regs[R2] ← Regs[R2] + d`	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d.
Autodecrement	`Add R1, -(R2)`	`Regs[R2] ← Regs[R2] - d` `Regs[R1] ← Regs[R1] + Mem[Regs[R2]]`	Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.
Scaled	`Add R1,100(R2)[R3]`	`Regs[R1] ← Regs[R1] + Mem[100 + Regs[R2] + Regs[R3] * d]`	Used to index arrays. May be applied to any indexed addressing mode in some computers.

Modos de direccionamiento



Referencias

J Hennesy and David Patterson (2012)

Computer Architecture: A Quantitative Approach. 5th Edition. Elsevier – Morgan Kaufmann. [Appendix A, Appendix K]

Jeferson González G. (2017)

Material de clase: Arquitectura de computadores I.

Sitios web recomendados

- Alineamiento de memoria:
  https://www.ibm.com/developerworks/library/pa-dalign/
- Apéndices Hennesey:
  https://booksite.elsevier.com/9780123838728/references.php
- Byte addressing:
  https://en.wikipedia.org/wiki/Byte_addressing