Nivel ISA – Parte 2

Arquitectura y Organización de Computadoras II

Ticiano J. Torres Peralta

REV2021

Las 7 Dimensiones del ISA

Que tiene que considerar un diseñador de ISA?

- La clase del ISA
- **El modelo de memoria**
- **Los modos de direccionamiento**
- Tipos y tamaños de datos (y de operandos)
- Tipos de operaciones
- Tipos de control de flujo
- Formato de instrucción
- *Registros Disponibles* (Arquitecturas GPR)

Un pequeño apartado

Lo que veremos es mas que nada desde el punto de vista del diseñador del compilador, o en otras palabras, desde el punto de vista de la salida del compilador.

Esta aclaración es significativa porque la mayoría de las maquinas corren en dos modos: modo Kernel y modo Usuario.

El modo Kernel es para el sistema operativo, donde esta permitido ejecutar TODAS las instrucciones disponibles.

Aquí nos concentraremos mas desde el punto de vista del modo Usuario, el modo para aplicaciones de usuario que restringe ciertas instrucciones peligrosas.

Modelo de Memoria

Independientemente del tipo de ISA que se decide utilizar, es crucial definir como una dirección de memoria será interpretada y como es especificada.

Interpretación de Direcciones

¿Que tipo de objeto acedemos desde el punto de vista de su dirección, y su tamaño?

Todas las memorias principales están divididas en celdas con direcciones consecutivas y hoy en día el tamaño de celda mas común es la de 8 bits (1 Byte). Pero, también es común que el procesador pueda/prefiera manipular los datos en palabras de 4 Byte o 8 Bytes (y posiblemente muy pronto 16 Bytes).

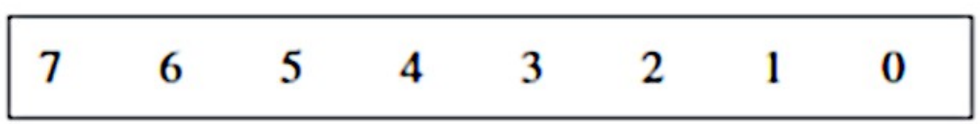
Todas ISAs recientes tienen varias instrucciones que proveen acceso a memoria a través de byte (8 bits), half word (media palabra – 16 bits), word (32 bits), y double word (doble palabra – 64 bits).

Interpretación de Direcciones

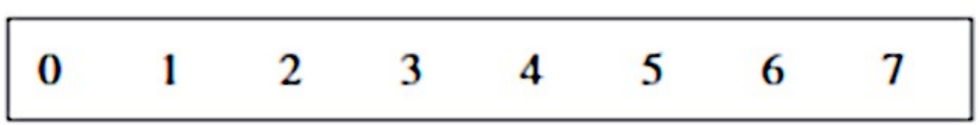
¿Como están los bytes de la palabra ordenados en memoria?

Hay dos convenciones de como es el ordenamiento de los bytes desde el punto de vista de un objeto mas grande.

Little Endian: Si tenemos una dirección, x...x000, donde comienza una palabra de 8 Bytes, esa dirección contienen el byte menos significativo de nuestra palabra.



Big Endian: Si tenemos una dirección, x...x000, donde comienza una palabra de 8 Bytes, esa dirección contienen el byte mas significativo de nuestra palabra.



Interpretación de Direcciones

¿La memoria requiere alineamiento?

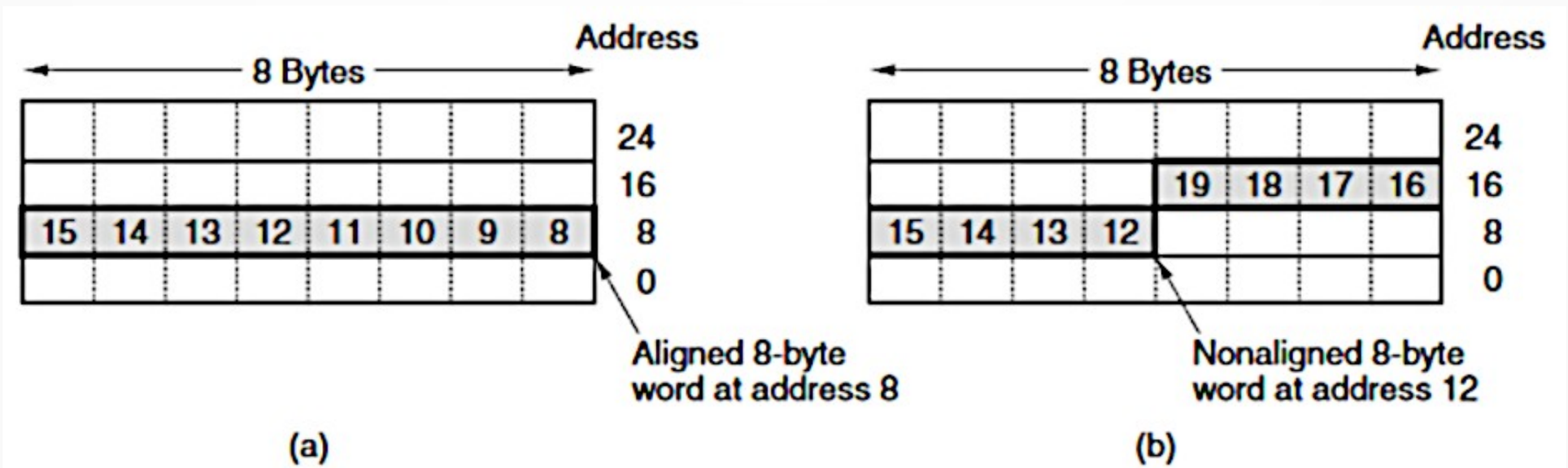
Un problema importante en muchas computadoras es que accesos a objetos mas grandes que un byte deben ser alineados.

Estas restricciones aveces están impuestas por razones de hardware porque es típico que alinean la memoria en múltiples de la palabra. (Requiere menos lógica, circuitos mas simples y mas baratos.)

El resultado puede ser dos cosas:

- No te permite referenciar una palabra de forma desalineada.
- Podes hacerlo pero, un acceso a una palabra desalineada debe ser resuelto por varias referencias alineadas. Flexibilidad a un costo alto de rendimiento.

Direccionamiento de Memoria



Direccionamiento de Memoria

	Value of 3 low-order bits of byte address							
Width of object	**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

Interpretación de Direcciones

¿Como es nuestro espacio de direcciones?

La mayoría de las computadoras tratan la memoria como un solo espacio desde el punto de vista del nivel ISA, extendiendo desde 0 a $2^{32} - 1$ (para 32 bits) y 0 a $2^{64} - 1$ (para 64 bits).

Hay otra posibilidad donde una computadora divide en dos espacios de direccionamiento, uno para instrucciones y otro para datos.

Es un esquema mas complejo, pero tiene ventajas:

- Podes direccionar tanto 2^{32} bytes de instrucciones como 2^{32} bytes de datos con una interfaz de direccionamiento de 32 bits.
- Porque escrituras van a espacio de datos, es imposible sobre escribir instrucciones de un programa.

Modos de Direccionamiento

La instrucciones de un ISA pueden tener operandos. Los operandos son efectivamente direcciones, osea, apuntan adonde esta el dato.

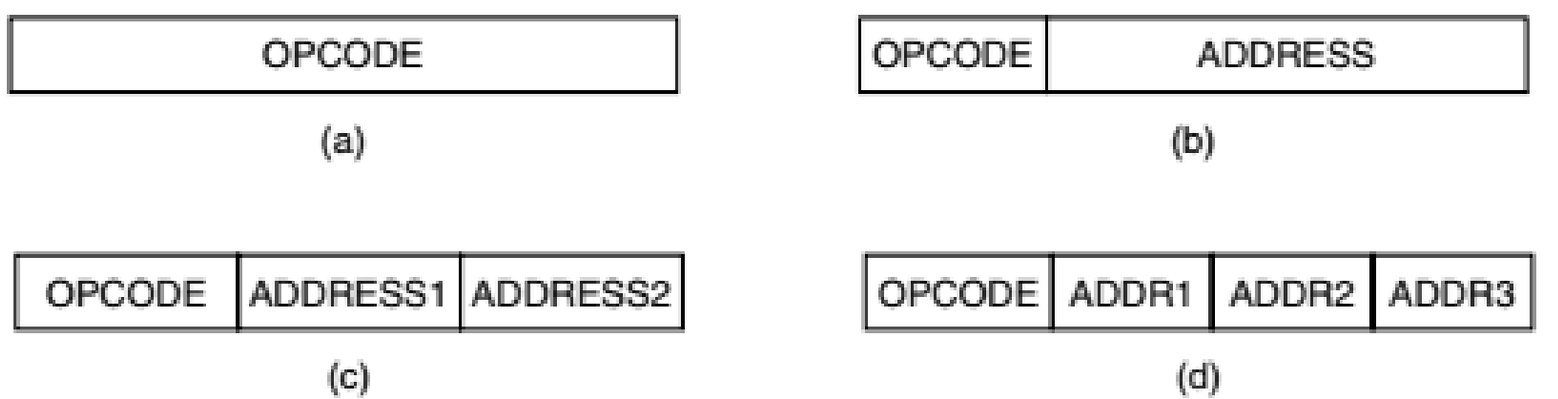


Figure 5-9. Four common instruction formats: (a) Zero-address instruction. (b) One-address instruction (c) Two-address instruction. (d) Three-address instruction.

Modos de Direccionamiento

La ISA tienen que especificar de que forma acceder al objeto en cuestión (indicado por el operando). Las mismas se conocen como modos de direccionamiento.

El modo de direccionamiento se extiende un poco mas allá que el modelo de memoria, porque el operando puede indicar una dirección de memoria, un registro o un constante.

Cuando se direcciona un lugar en memoria, la dirección especifica en memoria es llamada la **Dirección Efectiva**.

Modos de Direccionamiento

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

Registro (Registro-Registro):

Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.

Este modo usa una dirección para seleccionar los registros correctos.

Porque el registro es un componente rápido y tienen direcciones cortas, este modo de dirección es el mas utilizado.

Consideraciones:

En arquitecturas modernas de load/store, casi todas las instrucciones usan exclusivamente este modo de direccionamiento. Las únicas instrucciones que usan algún modo de direccionamiento que accede a memoria son las que leen un valor de memoria y lo colocan en un registro (load), y las que escriben un valor de un registro a memoria (store).

Compiladores tratan de maximizar la utilización de este momo de direccionamiento, y trabajan mucho para determinar cuales variables van a ser usados de forma frecuente para colocarlas en registros. (Un buen ejemplo son variables que controlan un bucle)

Modos de Direccionamiento

Immediate:

Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.

El modos mas simple de direccionar un operando es que la dirección sea el valor mismo del operando. Este tipo de operando se lo conoce como un **operando inmediato** porque la instrucción no tiene que hacer una referencia para buscar el valor, está disponible de forma inmediata el momento que la instrucción es buscada (fetched) de memoria.

Consideraciones:

Muy útil para definir un constante, pero con la limitación que solo se tiene la cantidad de bits del restante tamaño de la instrucción para representar ese constante. Se utiliza mucho para constantes pequeños de valores enteros.

Modos de Direccionamiento

Direct o Absolute:

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.

En este modo, un operando directamente contiene al dirección efectiva de memoria.

Consideraciones:

Muy útil para definir variables globales donde su dirección puede ser determinada en tiempo de complicación.

Parecido a direccionamiento inmediato, tienen la limitación en la cantidad de bits disponibles, en este caso para referenciar una dirección de memoria.

Modos de Direccionamiento

Register Indirect:

Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.

Aquí, a diferencia del modo anterior, la dirección efectiva de memoria se encuentra en un registro. Una dirección usada de esta forma es conocida como un **puntero**.

Una gran ventaja sobre la anterior es que puede referenciar memoria sin el costo de tener una dirección completa incrustada en la instrucción. También la ventaja de poder modificar esa dirección.

Consideraciones:

Útil para trabajar con arreglos.

Modos de Direccionamiento

Displacement:

Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes).

Aveces es útil poder direccionar a memoria desde un punto de referencia. Con este tipo de direccionamiento la dirección efectiva es calculada desde una dirección base presente en la instrucción y un registro que contiene un desplazamiento.

Consideraciones:

Muy útil para hacer operaciones entre elementos de dos arreglos de la misma longitud.

(Tanenbaum llama a este modo Indexed, cuidado!!!)

```
        MOV R1,#0          ; accumulate the OR in R1, initially 0
        MOV R2,#0          ; R2 = index, i, of current product: A[i] AND B[i]
        MOV R3,#4096       ; R3 = first index value not to use
LOOP:   MOV R4,A(R2)       ; R4 = A[i]
        AND R4,B(R2)       ; R4 = A[i] AND B[i]
        OR R1,R4           ; OR all the Boolean products into R1
        ADD R2,#4          ; i = i + 4 (step in units of 1 word = 4 bytes)
        CMP R2,R3          ; are we done yet?
        BLT LOOP           ; if R2 < R3, we are not done, so continue
```

Figure 5-18. A generic assembly program for computing the OR of A_i AND B_i for two 1024-element arrays.

Modos de Direccionamiento

Indexed (Based-Indexed):

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.

Igual que el modo anterior, solo que aquí, la dirección base es contenida en un registro.

Consideraciones:

Muy útil para operaciones que involucran matrices.

Modos de Direccionamiento

Stack:

En este modo no hay operandos, osea, son implícitos y sus direcciones siempre son dos primeros valores que salen de una pila estilo LIFO. El resultado es empujado de vuelta a la pila

a1 = a2 + a3;

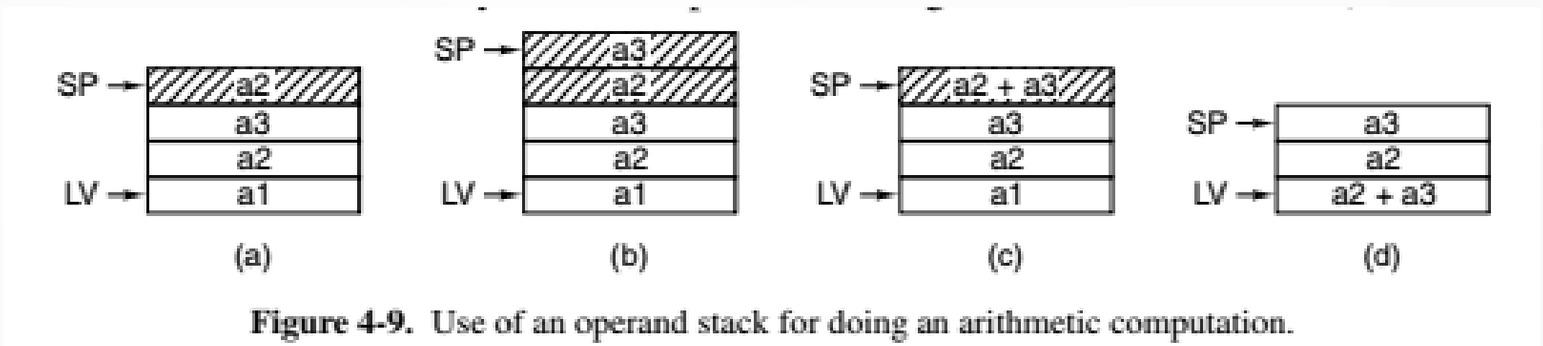
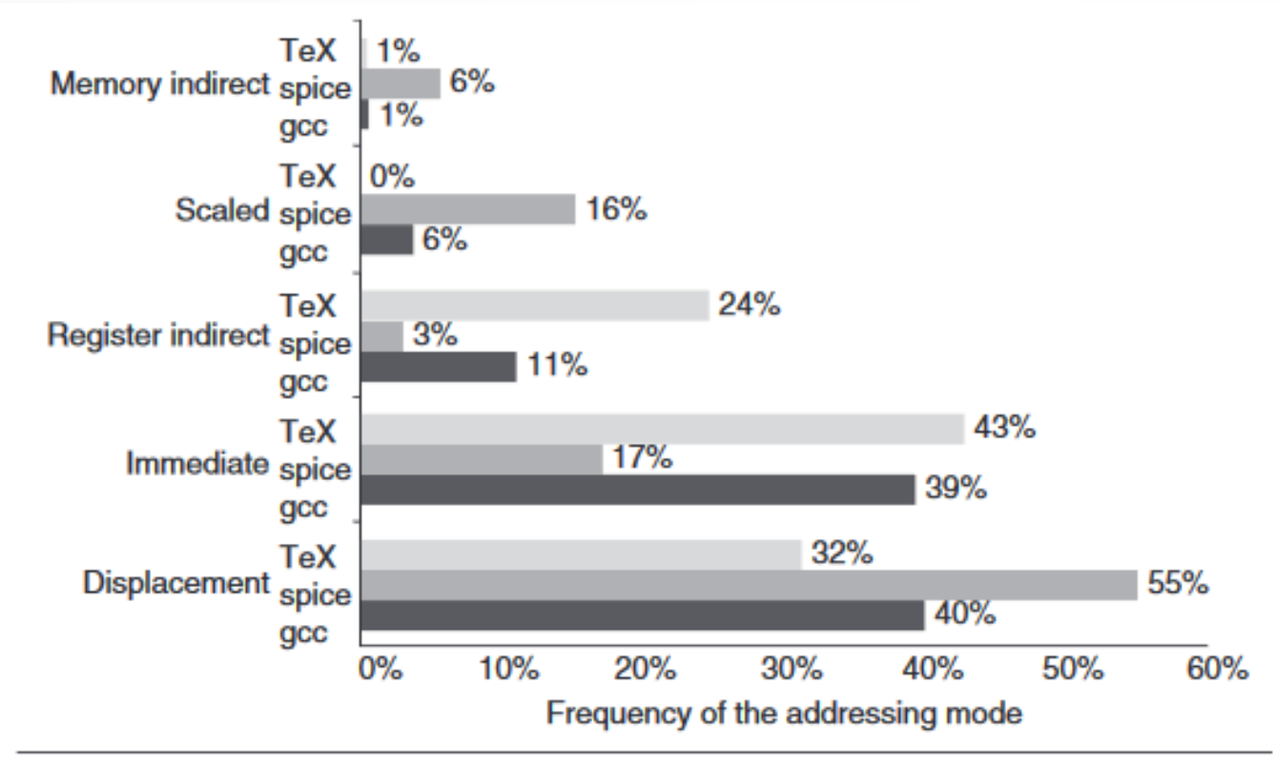


Figure 4-9. Use of an operand stack for doing an arithmetic computation.

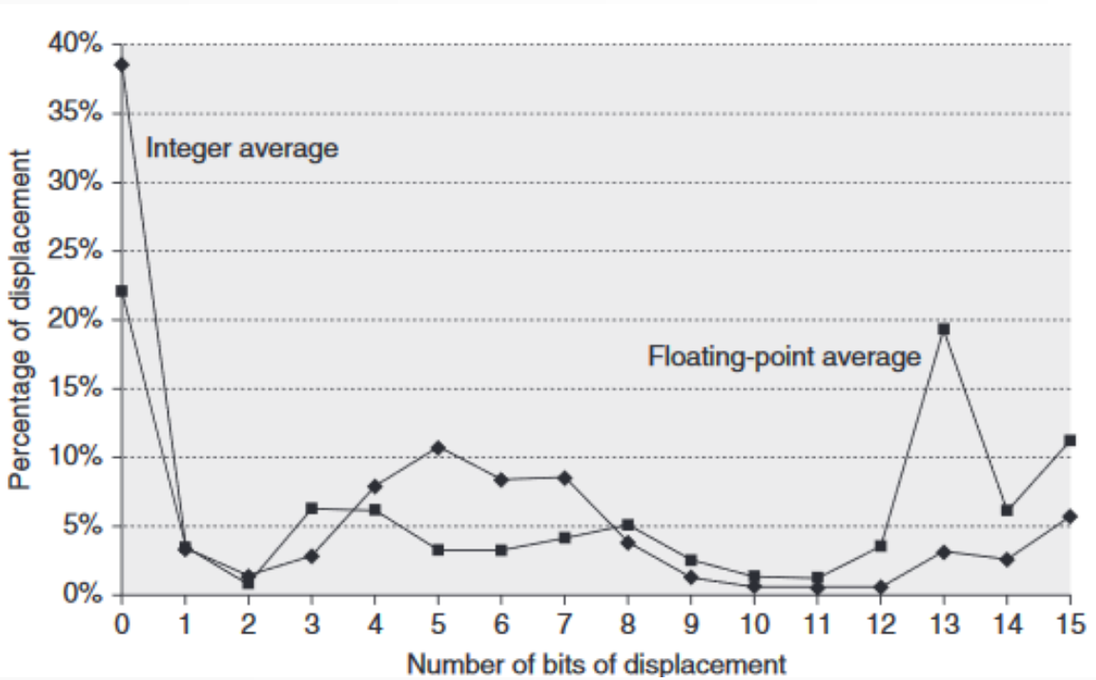
(Este modo de direccionamiento no se usa en arquitecturas GPRs, es exclusivo a la arquitectura de pila, la cual Tanenbaum usa para la MIC-1.)

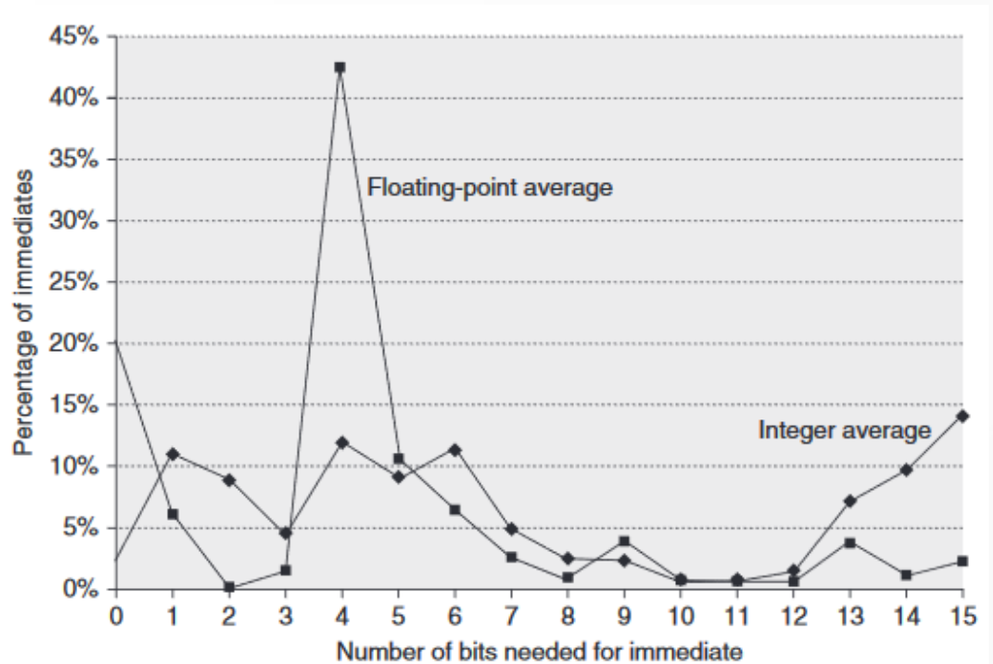
Gráficos Interesantes



Este grafico muestra que tan frecuente son los modos de direccionamiento en tres programas llamados: TeX, Spice, y GCC.

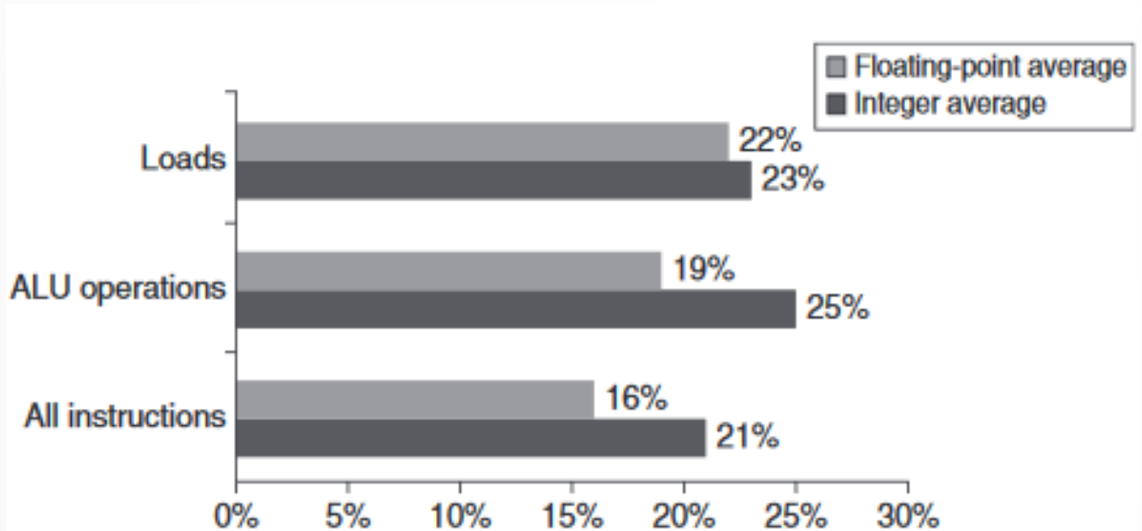
Gráficos Interesantes





Este grafico muestra que tan frecuente pueden ser presentados el valor de desplazamiento e inmediato con diferentes bits en SPEC2000.

Gráficos Interesantes



Este grafico muestra que tan frecuente se uso el modo de direccionamiento inmediato en el SPEC2000.