Tema 2: Arquitectura del procesador

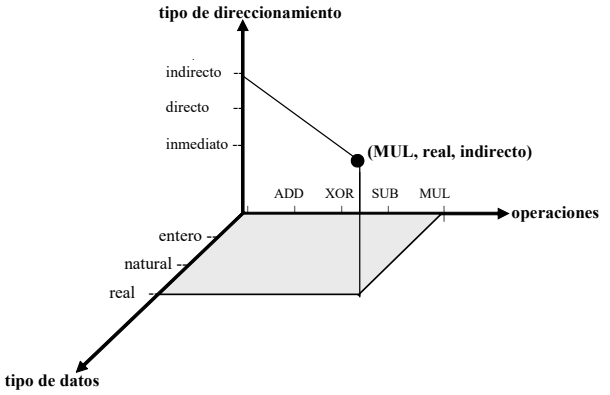
Índice

1. Introducción
2. Formato de las instrucciones
3. Direccionamiento
4. Instrucciones que operan sobre datos
5. Instrucciones de control del flujo de ejecución
6. Instrucciones paralelas SIMD
7. Rendimiento
8. Influencia en el rendimiento de las alternativas de diseño
9. Procesadores RISC y CISC

1. Introducción

Elementos que determinan una arquitectura ISA (*Instruction Set Architecture*)

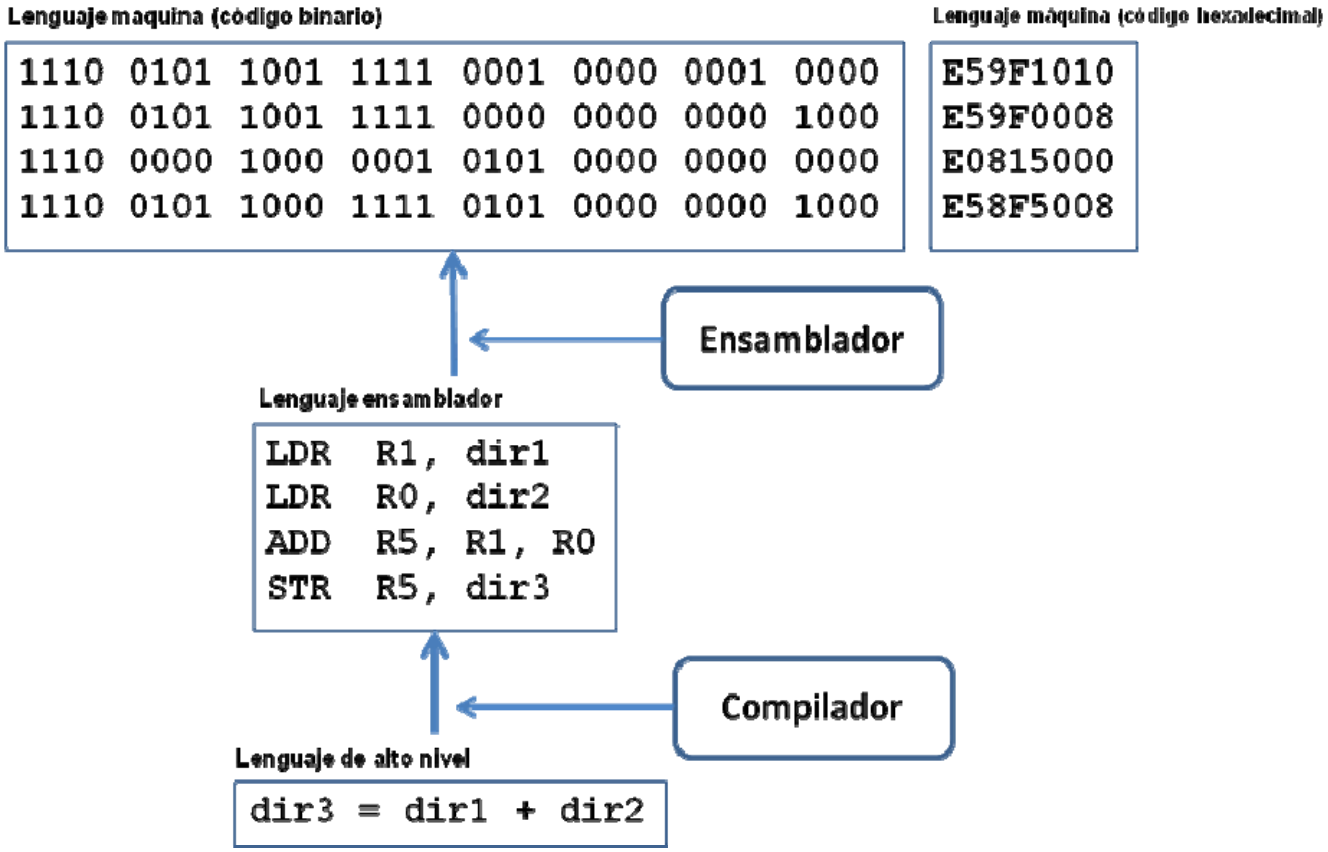
- Las instrucciones son las acciones elementales que puede ejecutar un computador.
- Una acción compleja deberá codificarse como una secuencia de instrucciones (programa).
- Una instrucción codifica los siguientes elementos:
  - una *operación básica*
  - que se realiza sobre unos *datos* ubicados en memoria o en registros
  - a los que se accede utilizando un *modo de direccionamiento*.
- Por tanto el repertorio de instrucciones vendrá definido por:
  - Conjunto básico de operaciones que se realizan sobre los datos: suma, resta, etc.
  - Tipos de datos y formatos que manejan las instrucciones: naturales, enteros, reales, caracteres, etc.
  - Modos de direccionamiento de los datos en la memoria: inmediato, directo, indirecto, etc.
- Propiedad de ortogonalidad.
  - Determina la posibilidad de combinar en una instrucción: operaciones, tipos de datos y modos de direccionamiento



1. Introducción

Modelo de ejecución de programas en una arquitectura (1)

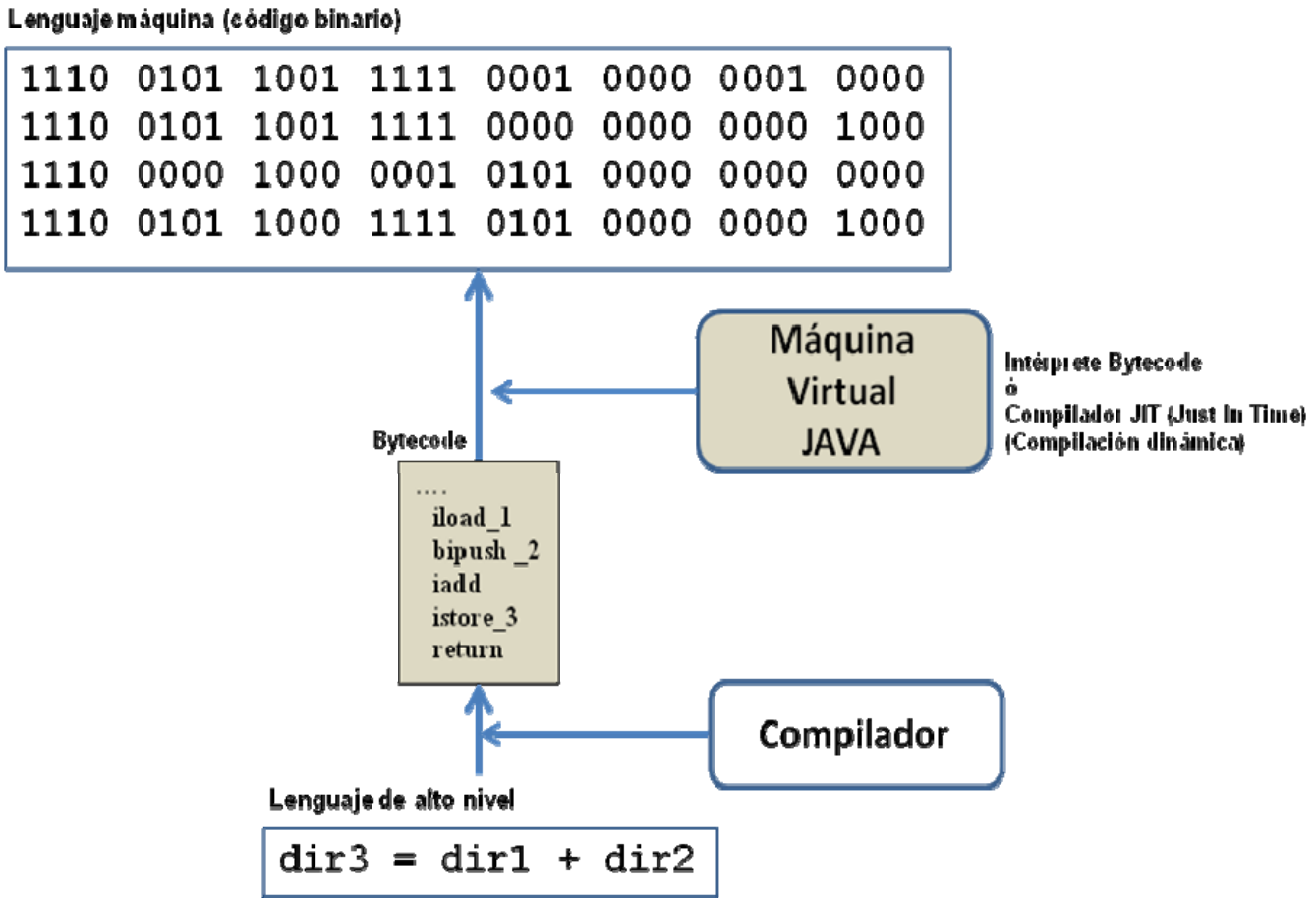
Modelo compilado



1. Introducción

Modelo de ejecución de programas en una arquitectura (2)

Modelo con máquina virtual



2. Formato de las instrucciones

Formato de las instrucciones

• Las informaciones de las instrucciones se codifican siguiendo un formato preestablecido.

• El formato determinará la longitud en bits de las instrucciones así como el significado y longitud de cada uno de sus campos.

• En general una instrucción se compone de los siguientes campos:

- Código de operación (CO)
- Operandos fuente (OP1, OP2,...)
- Operando destino o resultado (OPd)
- Dirección de la instrucción siguiente (IS)

CO	OP1	OP2	.....	OPd	IS

• Clasificación de los conjuntos de instrucciones:

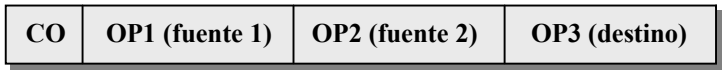
- Clasificación según el número de operandos explícitos por instrucción
- Clasificación según la forma de almacenar operandos en la CPU

2. Formato de las instrucciones

Diseño del repertorio de instrucciones (1)

Clasificación según el número de operandos explícitos por instrucción

- **3 operandos explícitos**



ejemplo: ADD B,C,A A ← B + C

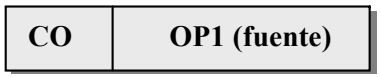
- Máxima flexibilidad
- Ocupa mucha memoria si los operandos no están en registros

- **2 operandos explícitos**

ejemplo: ADD B, C C ← B + C

- Reduce el tamaño de la instrucción
- Se pierde uno de los operandos

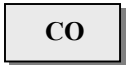
- **1 operando explícito**



ejemplo: ADD B Acumulador ← <Acumulador> + B

- Supone que fuente 1 y destino es un registro predeterminado (acumulador)
- Se pierde un operando fuente

- **0 operandos explícitos**



ejemplo: ADD cima de pila ← <cima de pila> + <cima de pila - 1>

- Se trata de computadores que trabajan sobre una pila

2. Formato de las instrucciones

Diseño del repertorio de instrucciones (2)

Ejemplo de codificación de una expresión en los 4 tipos de repertorios

E = (A – B)*(C + D)

3 operandos	2 operandos	1 operando	0 operandos

ADD C, D, C
SUB A, B, A
MUL A, C, E
	ADD C, D
	SUB A, B
	MUL B, D
	MOV D, E
		LOAD A
		SUB B
		STORE A
		LOAD C
		ADD D
		MUL A
		STORE E
			(PUSH) LOAD A
			(PUSH) LOAD B
			SUB
			(PUSH) LOAD D
			(PUSH) LOAD C
			ADD
			MUL
			(PULL) STORE E

2. Formato de las instrucciones

Diseño del repertorio de instrucciones (3)

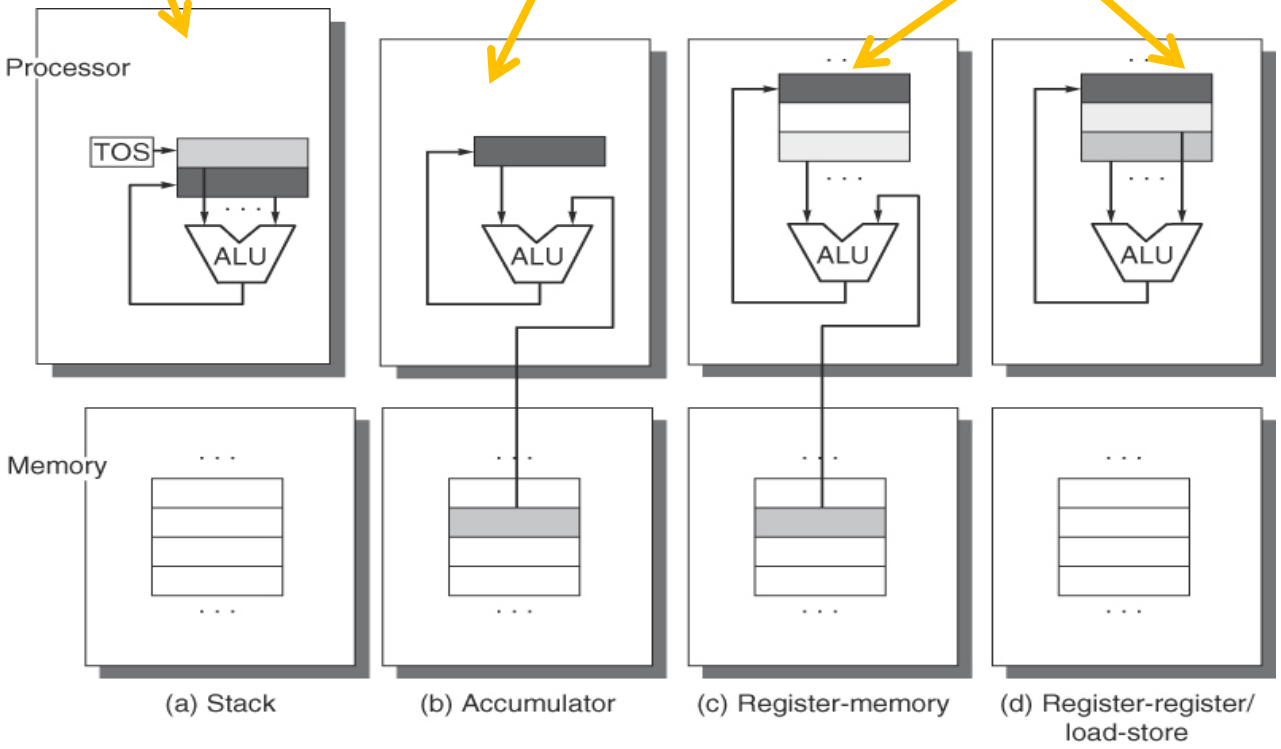
Clasificación según la forma de almacenar operandos en la CPU

- Arquitectura tipo pila (HP 3000/70)
- Arquitectura de acumulador (Motorola 6809)
- Arquitectura de **registros de propósito general** (IBM 360)

Arquitectura **tipo pila**
pila ← <cima pila> + <cima pila - 1>

Arquitectura **Acumulador**
Acumulador ←<Acumulador> + B

Arquitectura **Registros de propósito general**
C ← A + B

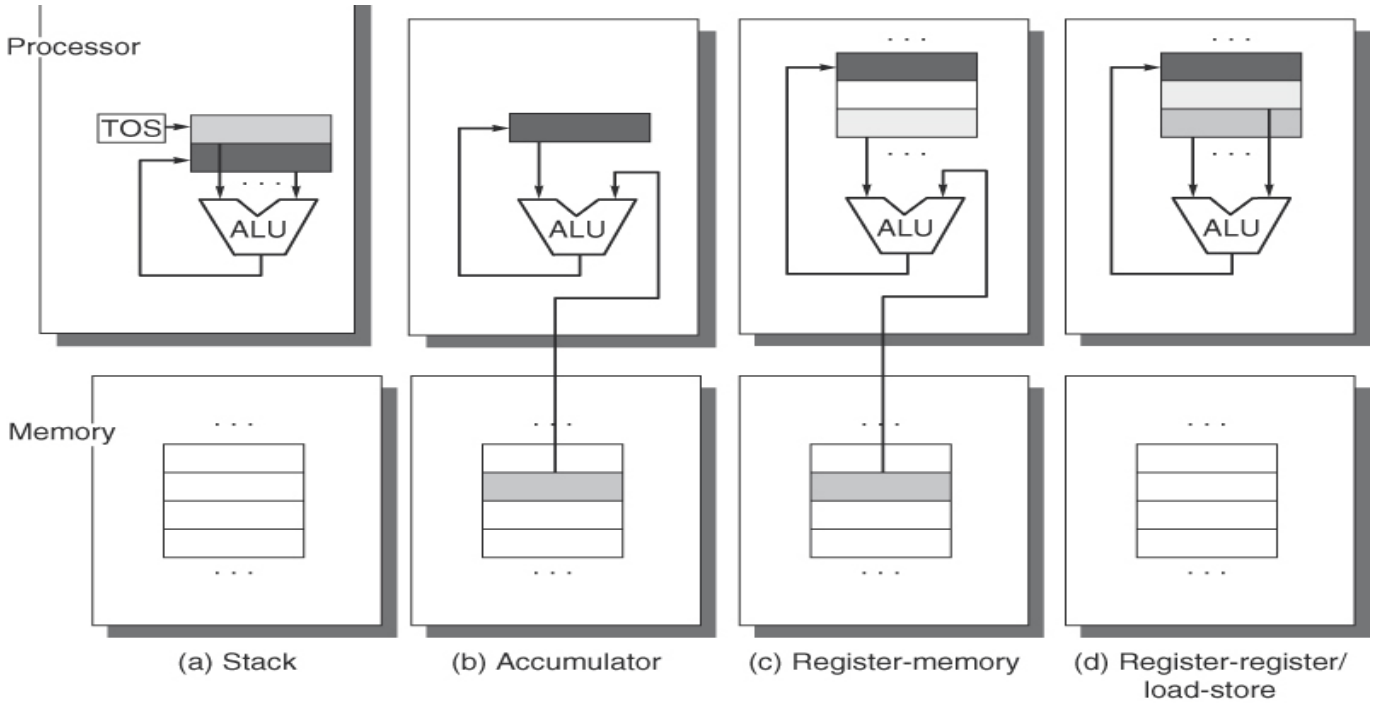


2. Formato de las instrucciones

Diseño del repertorio de instrucciones (4)

Código para la **operación C=A+B** en cada una de las arquitecturas

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C



2. Formato de las instrucciones

Diseño del repertorio de instrucciones (5)

Arquitecturas de registros de propósito general

Clasificación según el número máximo de operandos (2 ó 3) que pueden tener las instrucciones de la ALU y cuantos de ellos se pueden ubicar en memoria:

(**memoria - operandos**)

(**0** - **3**) Arquitectura registro-registro (carga-almacenamiento).
- Utilizan 3 operandos en total, y 0 en memoria.
- Formato de longitud fija y codificación simple.
- Las instrucciones se ejecutan en un número similar de ciclos.
- SPARC, MIPS, PowerPC

(**1** - **2**) Arquitectura registro-memoria.
- Utilizan dos operandos en total, uno ubicado en la memoria.
- Intel 80X86, Motorola 68000

(**3** - **3**) Arquitectura memoria-memoria.
- 3 operandos en total, pudiendo ser ubicados los 3 en memoria.
- VAX

2. Formato de las instrucciones

Diseño del repertorio de instrucciones (6)

Ventajas y desventajas de las arquitecturas de registros de propósito general

Tipo	Ventajas	Desventajas
Registro-registro (0/3)	Instrucciones simples, de longitud fija. Las instrucciones tardan un número de ciclos similar en ejecutarse	Hay que traer operandos de memoria a registros → Programas más largos
Registro-memoria (1/2)	Se puede acceder a los datos sin una carga (load) previa. El formato de instrucción suele ser fácil de codificar	Se pierde un operando fuente en la instrucción. Como la dirección de memoria es grande podemos estar limitando el campo de registro y con ello el número de registros. Los ciclos de reloj por instrucción dependen de la ubicación de los operandos
Memoria-memoria (2/2) ó (3/3)	Mas compacta . No gasta registros para datos temporales.	Gran variación en tamaños de instrucción. Además, gran variación en trabajo por instrucción. Los accesos a memoria crean cuello de botella en la misma (esta opción no se usa hoy en día)

2. Formato de las instrucciones

Tres variaciones básicas en la codificación de instrucciones

- **Longitud variable**: Soporta cualquier número de operandos y modos de direccionamiento. Menor tamaño medio de programas
- **Longitud fija**: Mínimo número de operandos y modos de direccionamiento (normalmente el modo de direccionamiento no se codifica, viene implícito en el OPCODE). Programas más largos
- **Híbrido:** El que tenga longitud fija o variable depende generalmente del OPCODE

Operation and no. of operands	Address specifier 1	Address field 1	...	Address specifier *n*	Address field *n*

(a) Variable (e.g., Intel 80x86, VAX)

Operation	Address field 1	Address field 2	Address field 3

(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)

Operation	Address specifier	Address field

Operation	Address specifier 1	Address specifier 2	Address field

Operation	Address specifier	Address field 1	Address field 2

(c) Hybrid (e.g., IBM 360/370, MIPS16, Thumb, TI TMS320C54x)

2. Formato de las instrucciones

Códigos de operación de longitud fija y variable

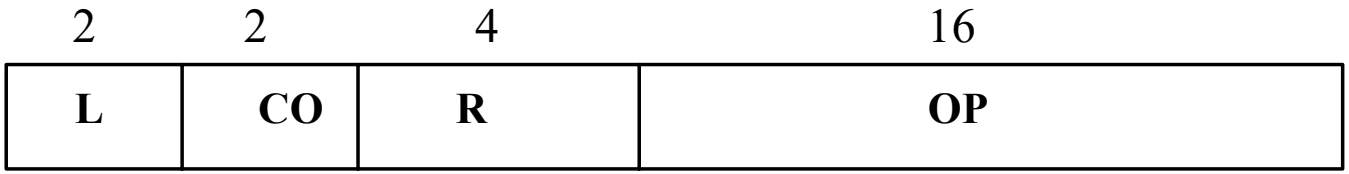
- La longitud del CO de un repertorio no tiene por que ser la misma para todas las instrucciones.
- Se puede ampliar el CO en instrucciones que necesiten menos bits en los operandos
- Ejemplo: Máquina con instrucciones de longitud fija de 24 bits y 16 registros generales

CO	R	OP	
0 0 0 0 0 0 0 1 . . . 1 1 1 0	R R . . . R	OP OP	15 instrucciones de 2 operandos (uno en registro y el otro en memoria) (CO de 4 bits)
1 1 1 1 1 1 1 1 . . 1 1 1 1	0 0 0 0 0 0 0 1 1 1 1 0	OP	15 instrucciones de 1 operando (en registro) (CO de 8 bits)
1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 . . 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2^{16} = 65.536 instrucciones de 0 operandos (CO de 24 bits)

2. Formato de las instrucciones

Otra alternativa

- Dedicar 2 bits para indicar si la instrucción tiene 0, 1 o 2 operandos:



En este caso podemos codificar los siguientes grupos de instrucciones:

L = 00 ➔ CO de 2 bits ➔ 4 instrucciones de 2 operandos

L = 01 ➔ CO de 6 bits ➔ 64 instrucciones de 1 operando

L = 10 ➔ CO de 22 bits ➔ 4.194.304 instrucciones de 0 operandos

2. Formato de las instrucciones

Optimización del CO variable en función de la frecuencia de las instrucciones

Dos alternativas a considerar:

- Frecuencia de aparición en el programa ➔ optimización de memoria
- Frecuencia de ejecución en el programa ➔ optimización del tráfico CPU-Memoria

La 2ª alternativa es más interesante en la actualidad, pues prima la velocidad de ejecución sobre la memoria .necesaria para almacenar el programa.

Para optimizar el CO se puede utilizar la **codificación de Huffman**:

1) Se escriben en una columna las instrucciones y a su derecha su frecuencia de ejecución. Cada elemento de la columna será un nodos terminal del **árbol de decodificación**.

2) Se modifica la columna actual uniendo las dos frecuencias menores de dicha columna con sendos arcos, obteniéndose un nuevo nodo cuyo valor será la suma de los nodos de procedencia.

3) Se repite el paso 2) hasta llegar a la raíz del árbol que tendrá valor 1

4) Comenzando en la raíz, asignamos 0 (1) al arco superior y 1 (0) al inferior hasta llegar a los nodos terminales

5) Se obtiene el código de cada instrucción recorriendo el árbol de la raíz a la instrucción y concatenando los valores de los arcos del camino

2. Formato de las instrucciones

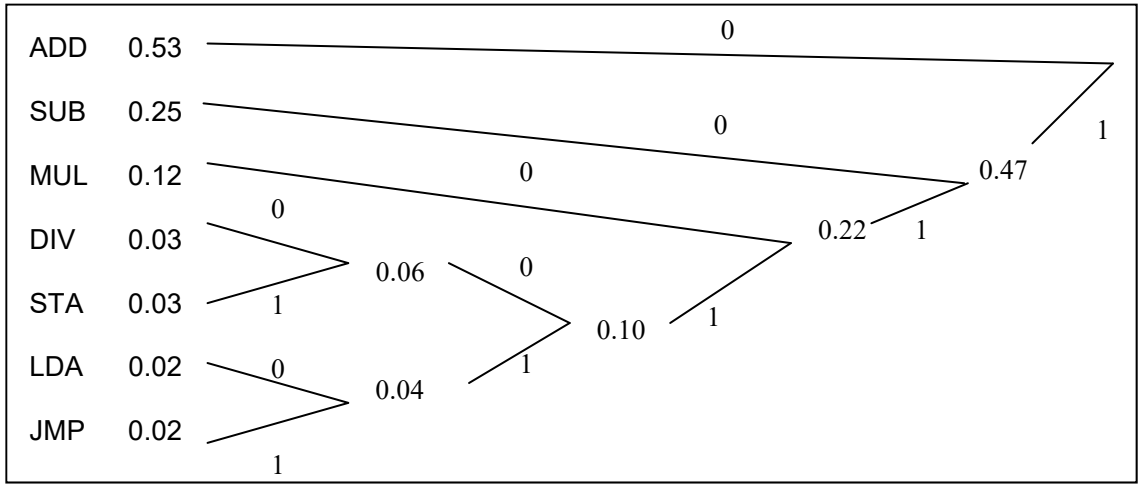
Ejemplo

Supongamos las siguientes frecuencias de ejecución de 7 tipos diferentes de instrucciones:

- CO de longitud fija: (se necesitarían 3 bits)

Tipo de instrucciones	Frecuencia de ejecución	Código de Huffman
ADD	0.53	0
SUB	0.25	10
MUL	0.12	110
DIV	0.03	11100
STA	0.03	11101
LDA	0.02	11110
JMP	0.02	11111

- CO de longitud variable: (se necesitan 1,89 bits)



$$l_m = \sum_i f_i * l_i = 0.53 \times 1 + 0.25 \times 2 + 0.12 \times 3 + 0.003 \times 5 + 0.003 \times 5 + 0.02 \times 5 + 0.02 \times 5 =$$

$$1.89\ \_\ bits\ < 3\ \_\ bits$$

3. Direccionamiento

➢ **Determina la forma de acceder a los operandos en memoria o registros.**

➢ **Propiedades generales del direccionamiento.**

- **Resolución**
- **Orden de los bytes en memoria**
- **Alineación**
- **Espacios de direcciones**
- **Modos de direccionamiento**

1. Resolución

- Es la menor cantidad de información direccionada por la arquitectura.
- El mínimo absoluto es un bit, aunque esta alternativa la utilizan pocos procesadores.
- Lo más frecuente en los procesadores actuales es utilizar resoluciones de 1 o 2 bytes.
- La resolución puede ser diferente para instrucciones y datos.

- Ejemplos:

Resolución	MC68020	VAX-11	IBM/370	B1700	B6700	iAPX432
Instrucciones Datos	16 8	8 8	16 8	1 1	48 48	1 8

3. Direccionamiento

Propiedades generales del direccionamiento.

2. Orden de los bytes en memoria

El concepto de *endian* lo introdujo Cohen para expresar la forma como se ordenan en memoria los bytes de un dato simple (escalar) de varios bytes.

- **Modo *big-endian***
  Almacena el byte más significativo del escalar en la dirección más baja de memoria

- **Modo *little-endian***
  Almacena el byte más significativo del escalar en la dirección más alta de memoria.

Ejemplo: el hexadecimal 12 34 56 78 almacenado en la dirección de memoria 184

Big-endian		**Little-endian**	
dirección	Byte(dato)	dirección	Dato(byte)
184	12	**184**	78
185	34	185	56
186	56	186	34
187	78	187	12

3. Direccionamiento

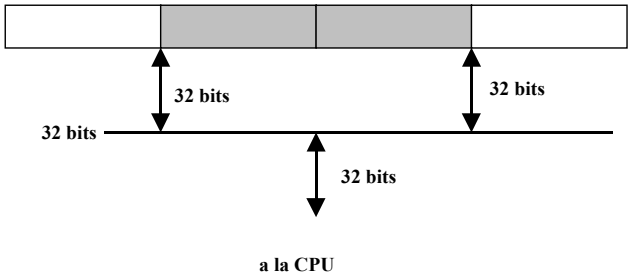
Propiedades generales del direccionamiento.

3. Alineación

Un objeto de datos de ***n* bytes** ubicado en la dirección de memoria **D** está alineado si ***D mod n = 0***

Objeto de datos direccionado (tamaño)	Alineaciones correctas
byte	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
media palabra (2 bytes)	0, 2, 4, 6, 8, 10
palabra (4 bytes)	0, 4, 8, 12
doble palabra (8 bytes)	0, 8, 16

- Determinadas máquinas sólo permiten accesos alineados
- La falta de alineación implica complicaciones hardware
- Los programas con accesos alineados se ejecutan más rápidamente
- Para alinear datos se utiliza una red de alineación.



3. Direccionamiento

Propiedades generales del direccionamiento.

4. Espacios de direcciones

Un mismo procesador pueden tener hasta 3 espacios de direcciones diferentes:

- Espacio de direcciones de registros
- Espacio de direcciones de memoria
- Espacio de direcciones de entrada/salida

Los espacios de direcciones de memoria y entrada/salida de algunos procesadores están unificados (un solo espacio) ➔

- Los puertos de E/S ocupan direcciones de memoria.

- No existen instrucciones específicas de E/S

- Se utilizan las de referencia a memoria (carga y almacenamiento) con las direcciones asignadas a los puertos.

3. Direccionamiento

Propiedades generales del direccionamiento.

5. Modos de direccionamiento

Determinan la forma como el operando (OPER) presente en las instrucciones especifica la dirección efectiva (DE) del dato operando (DO) sobre el que se realiza la operación indicada por CO.

Direccionamiento Inmediato.

DO = OPER

- El dato operando se ubica en la propia instrucción ==> no requiere accesos a memoria.
- Se suele utilizar para datos constantes del programa
- El tamaño está limitado por el número de bits de OPER

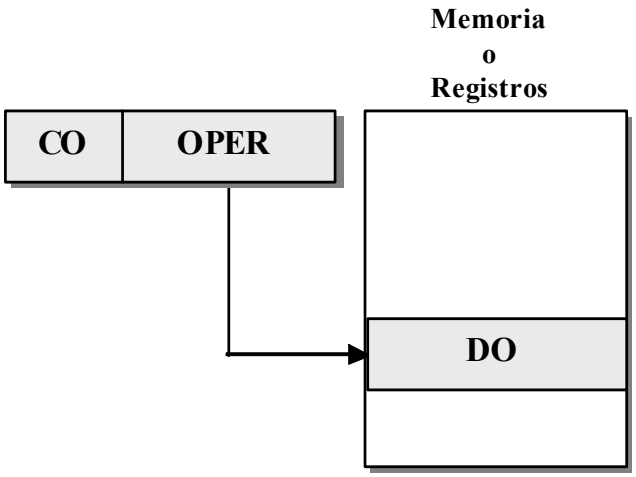
Direccionamiento Implícito

- El dato operando se supone ubicado en algún lugar específico de la máquina, por ejemplo, una pila

3. Direccionamiento

Direccionamiento Directo (memoria o registros)

OPER = Dirección de memoria o de un registro
DE = OPER
DO = <OPER>

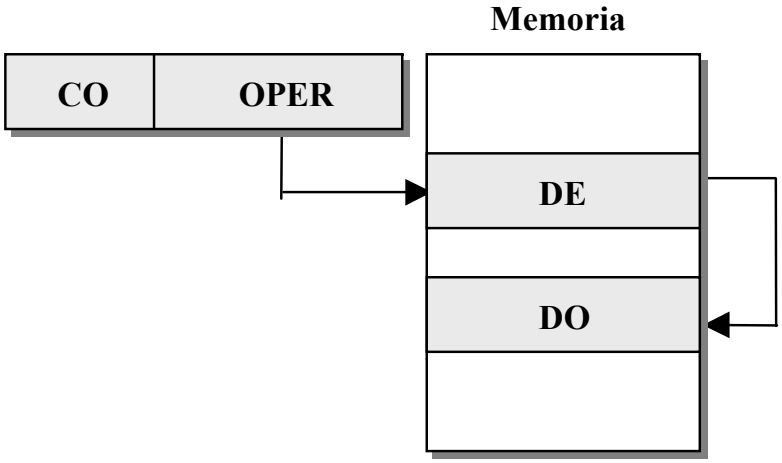


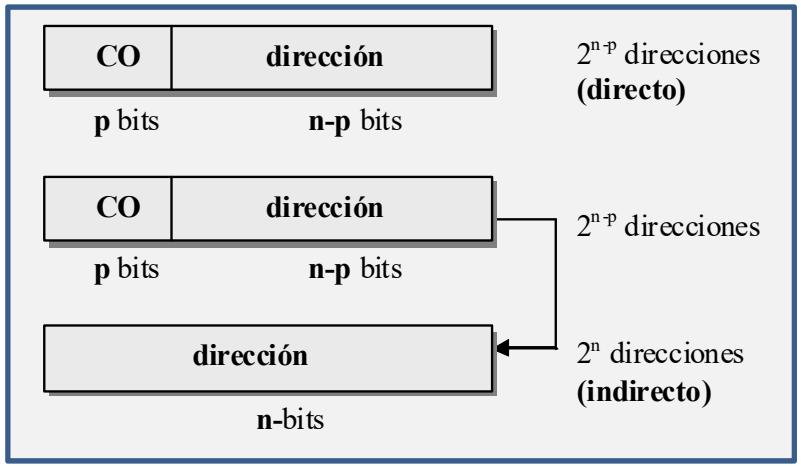
- La especificación de un **registro requiere menor número de bits** que la de una posición de memoria
- El acceso a los **registros es más rápido** que a Memoria
- El direccionamiento directo a memoria se conoce como **absoluto**
- Con frecuencia se limita el número de bits de OPER limitando el acceso a sólo una parte de la memoria (**página cero**)

3. Direccionamiento

Direccionamiento Indirecto (memoria)

OPER = Dirección de memoria
DE = <OPER>
DO = <<OPER>>

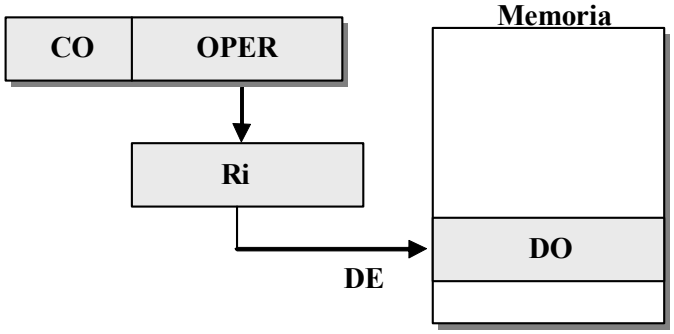




- Permite el tratamiento de una **dirección de memoria como un dato**
- Permite el **paso por referencia** de parámetros a subrutinas
- Permite **referenciar un espacio mayor** de direcciones

3. Direccionamiento

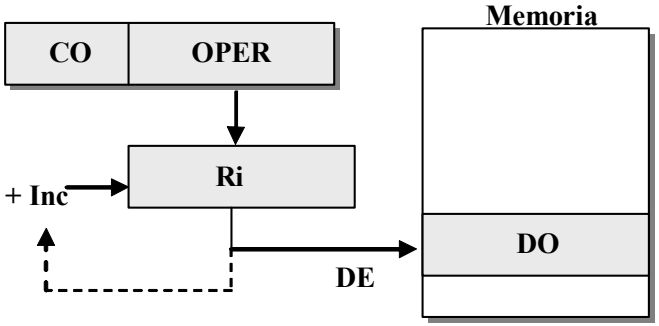
Indirecto registro



Puro

DE = <Ri>

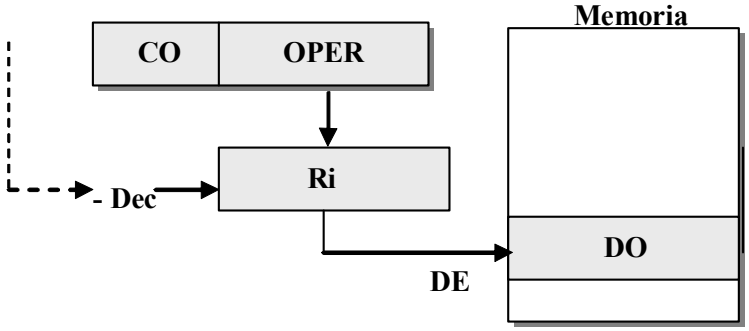
con Ri cualquier registro



Indirecto registro con postincremento

DE = <Ri>; Ri ← <Ri> + Inc

con Inc = 1, 2 ó 4 bytes



Indirecto registro con predecremento

Ri ← <Ri> - Dec, DE = <Ri>,

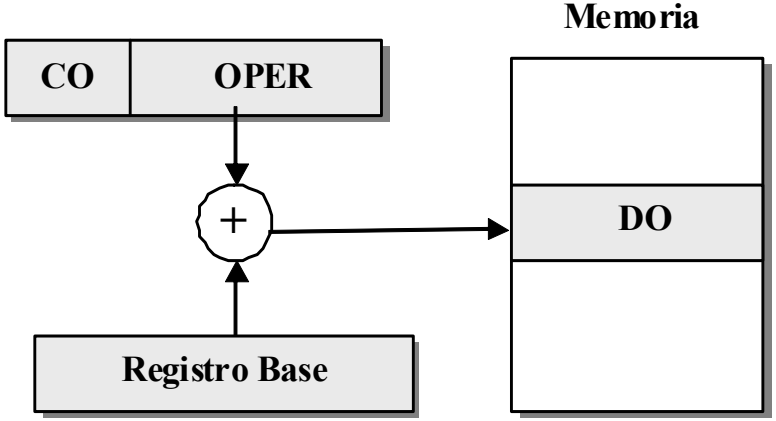
con Dec = 1, 2 ó 4 bytes

3. Direccionamiento

Modos con desplazamiento: direccionamiento base más desplazamiento

DE = <Registro base> + OPER; OPER = desplazamiento

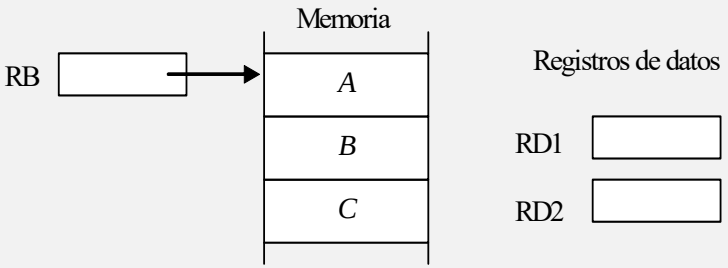
- Se utiliza para la reubicación de datos en memoria



Ejemplo: A = B + C

Programa

```
LOAD RB, 1; RD1  (RD1 <--  <RB>+1)          <RB> + 1 = B
LOAD RB, 2; RD2  (RD2 <-- <RB>+2)           <RB> + 2 = C
ADD RD1; RD2     (RD1 <-- <RD1> + <RD2>)
STORE RD1; RB,0  (<RB>+0 <-- <RD1>)         <RB> + 0 = A
```



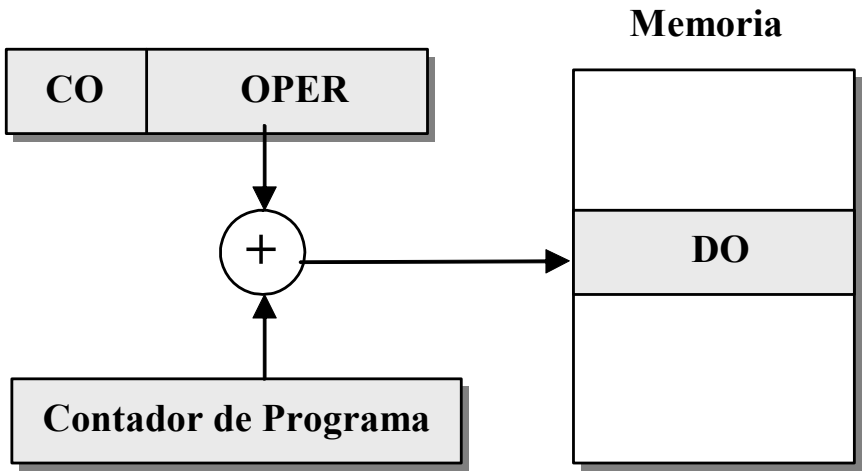
- Se pueden reubicar los datos A, B y C cambiando sólo el contenido de RB, sin alterar el programa.

3. Direccionamiento

Modos con desplazamiento: Direccionamiento relativo

DE = <Contador de programa> + OPER; OPER = desplazamiento

- Se utiliza en las instrucciones de salto para conseguir la reubicabilidad del código
- El desplazamiento en estas instrucciones tiene signo (c2) lo que significa que el salto se puede dar hacia posiciones anteriores o posteriores a la ocupada por la instrucción.

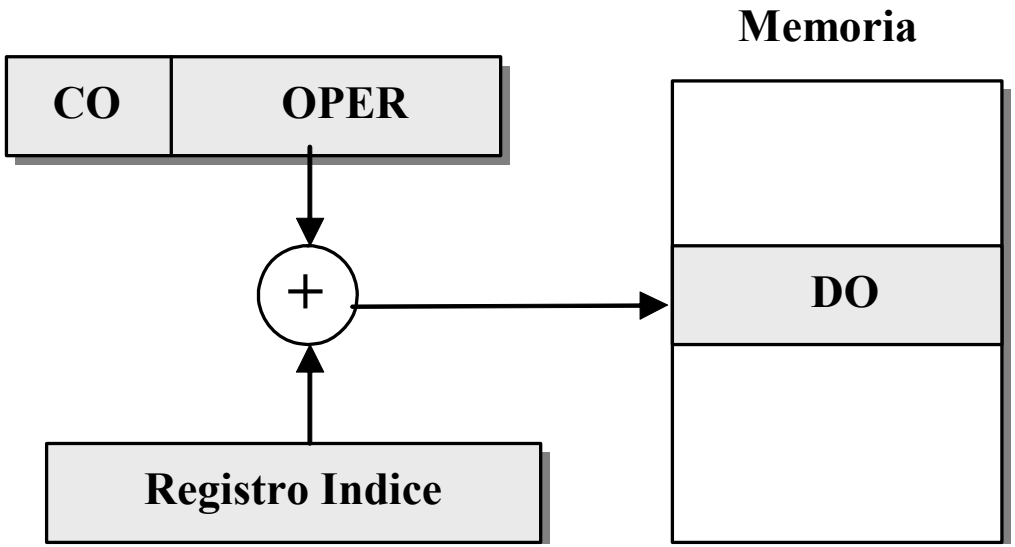


3. Direccionamiento

Modos con desplazamiento: Direccionamiento indexado

DE = <Registro índice> + OPER; OPER = desplazamiento

- Se utiliza para recorrer estructuras lineales como los *arrays*
- Par facilitar su uso se suele complementar con el pre o post incremento o decremento del registro índice

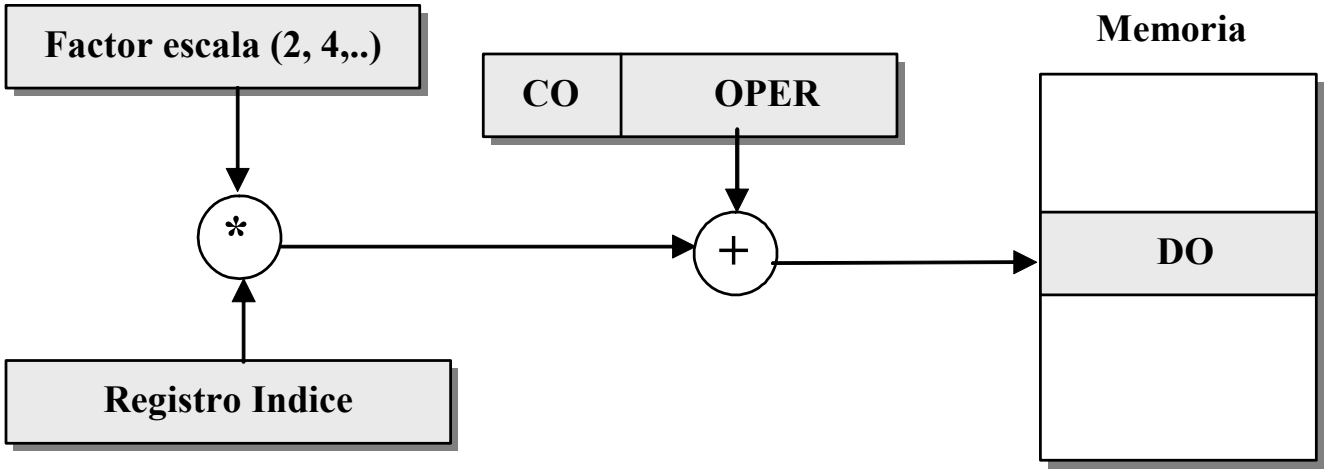


3. Direccionamiento

Modos con desplazamiento: Direccionamiento indexado con factor de escala

DE = <Registro índice> *<Factor de escala>+ OPER; OPER = desplazamiento

- Se utiliza para recorrer estructuras lineales con elementos de 2, 4,.. palabras
- También se puede utilizar el pre o post incremento o decremento del registro índice



3. Direccionamiento

Modos de direccionamiento del MIPS R-2000

- Inmediato

- Registro

  LA = R

- Relativo

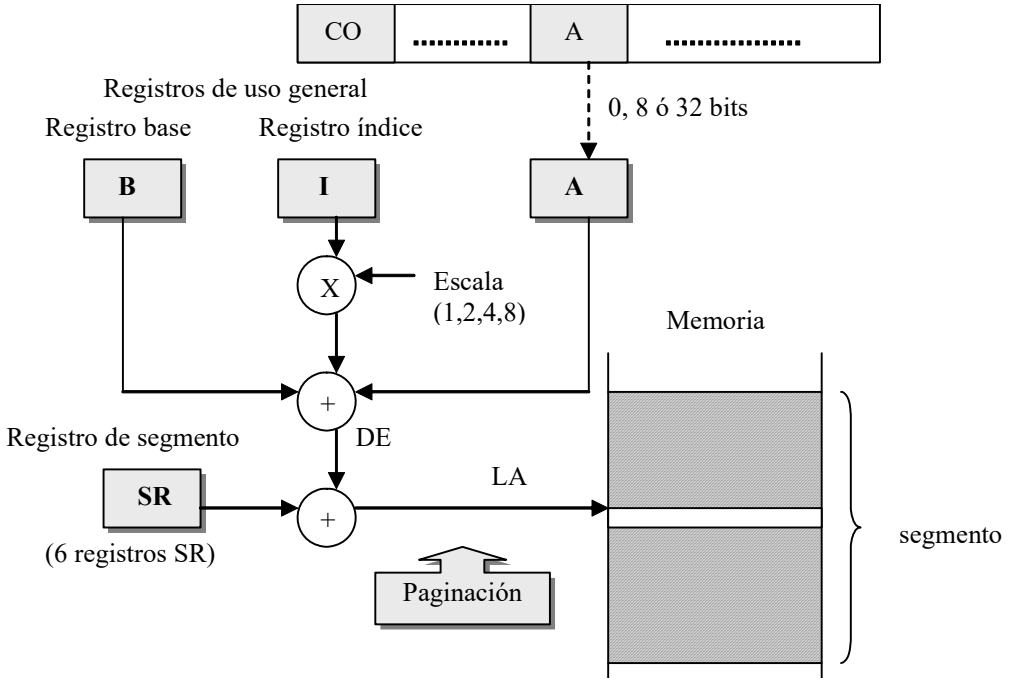
  DE = <PC> + Desplazamiento

- Indirecto registro con desplazamiento
  (= base + desplazamiento)

  DE = <Ri> + Despla

3. Direccionamiento

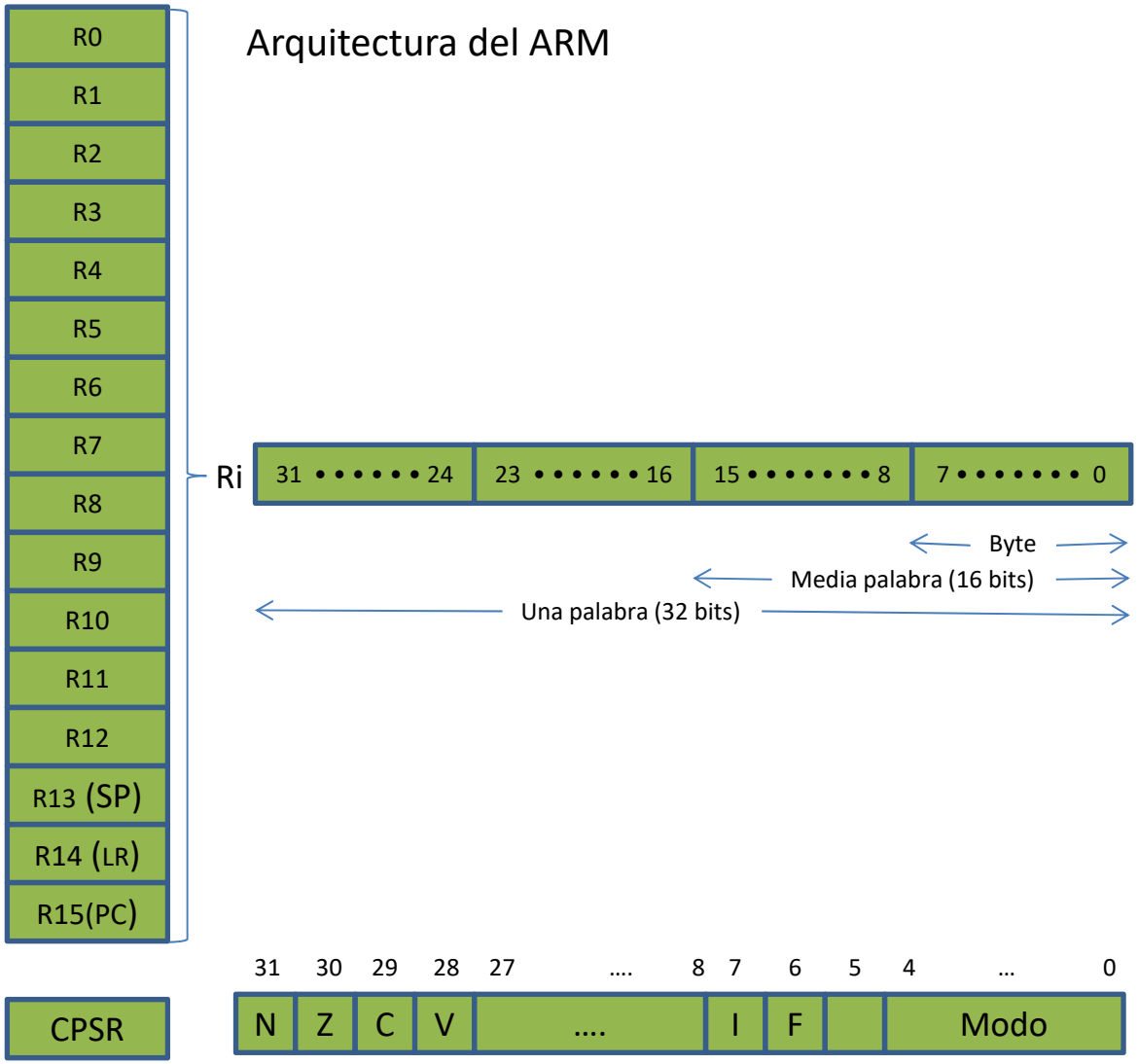
Modos de direccionamiento del Pentium II



• Inmediato	DO = A (1,2,4 bytes)
• Registro	LA = R, DO = <R> (LA = dirección lineal)
• Desplazamiento	LA = <SR> + A
• Base	LA = <SR> + <B>
• Base + desplazamiento	LA = <SR> + <B> + A
• Indexado	LA = <SR> + <I>xEscala + A
• Base + desplazamiento indexado	LA = <SR> + <B> + <I> + A
• Base + desplazamiento indexado escalado	LA = <SR> + <B> + <I>xEscala + A
• Relativo	LA = <PC> + A

3. Direccionamiento

Modos de direccionamiento del ARM (1)

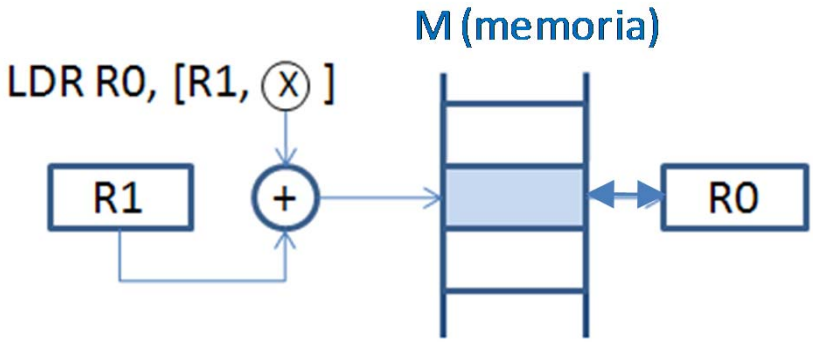


3. Direccionamiento

Modos de direccionamiento del ARM (2)

Operandos de acceso a memoria (1)

Direccionamiento con desplazamiento (offset)



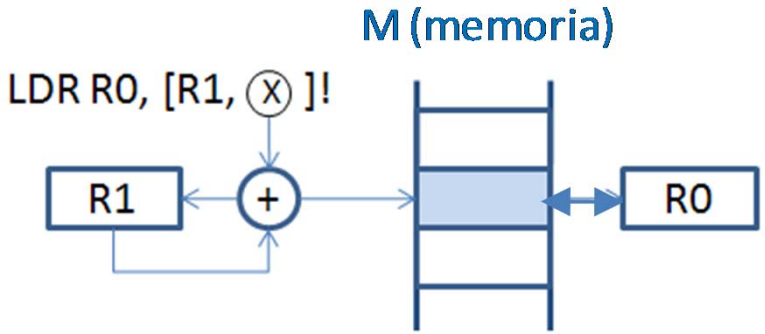

```
LDR     R0, [R1]                @ Carga R0 desde M[R1]
LDR     R0, [R1, #4]            @ Carga R0 desde M[R1 + 4]
LDR     R0, [R1, R2]  @ Carga R0 desde M[R1 + R2]
LDR     R0, [R1, R2, LSL #2]    @ Carga R0 desde M[R1 + R2 * 4]
STR     R0, [R1, #2]  @ Almacena R0 en M[R1 + R2 * 4]
```

3. Direccionamiento

Modos de direccionamiento del ARM (3)

Operandos de acceso a memoria (2)

Direccionamiento pre-indexado



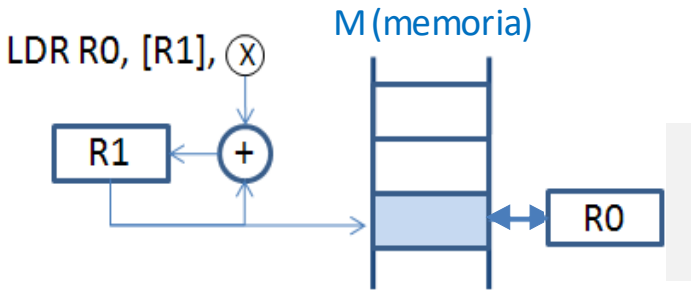

```
LDR     R0, [R1, #4]!          @ Carga R0 desde M[R1+4] y actualiza  R1= R1+4
LDR     R0, [R1, R2]!          @ Carga R0 desde M[R1+R2] y actualiza  R1= R1+R2
LDR     R0, [R1, R2, LSL #2]!  @ Carga R0 desde M[R1+R2*4] y actualiza  R1= R1 + R2*4
STR     R0, [R1, #4]!          @ Almacena R0 en M[R1+4] y actualiza R1 = R1 + 4
```

3. Direccionamiento

Modos de direccionamiento del ARM (4)

Operandos de acceso a memoria (3)

Direccionamiento post-indexado




```
LDR     R0, [R1], #4            @ Carga R0 desde M[R1] y actualiza  R1= R1+4
LDR     R0, [R1], R2  @ Carga R0 desde M[R1] y actualiza  R1= R1+R2
LDR     R0, [R1], R2, LSL #2    @ Carga R0 desde M[R1] y actualiza  R1= R1+R2*4
STR     R2, [R5], #8  @ Almacena R2 en M[R5] y actualiza  R5= R5 + 8
```

4. Instrucciones que operan sobre datos

Tipos de instrucciones que operan sobre datos.

- Movimiento o transferencia de datos
- Desplazamiento y rotación
- Lógicas y manipulación de bits
- Aritméticas y transformación de datos
- Entrada/salida
- Manipulación de direcciones
- Instrucciones paralelas SIMD (Single Instructions Multiple Data)
  - Operan simultáneamente sobre un conjunto de datos del mismo tipo
  - Aceleran las operaciones enteras de las aplicaciones multimedia (MMX)
  - Ampliadas a la coma flotante (SSE: Streaming SIMD Extension.

4. Instrucciones que operan sobre datos

Instrucciones de movimiento o transferencia de datos

- Son el tipo más básico de instrucción máquina.
- Transfieren el contenido de información entre elementos de almacenamiento (registros, memoria y pila)
- Dependiendo de la fuente y destino reciben nombres diferentes:

REG --> REG: transferencia	MEM --> REG: carga (load)	PILA --> REG: extracción (*pop*)
REG -->MEM: almacenamiento (*store*)	MEM --> MEM: movimiento (*move*)	PILA --> MEM: extracción (*pop*)
REG --> PILA: inserción (*push*)	MEM --> PILA: inserción (*push*)	

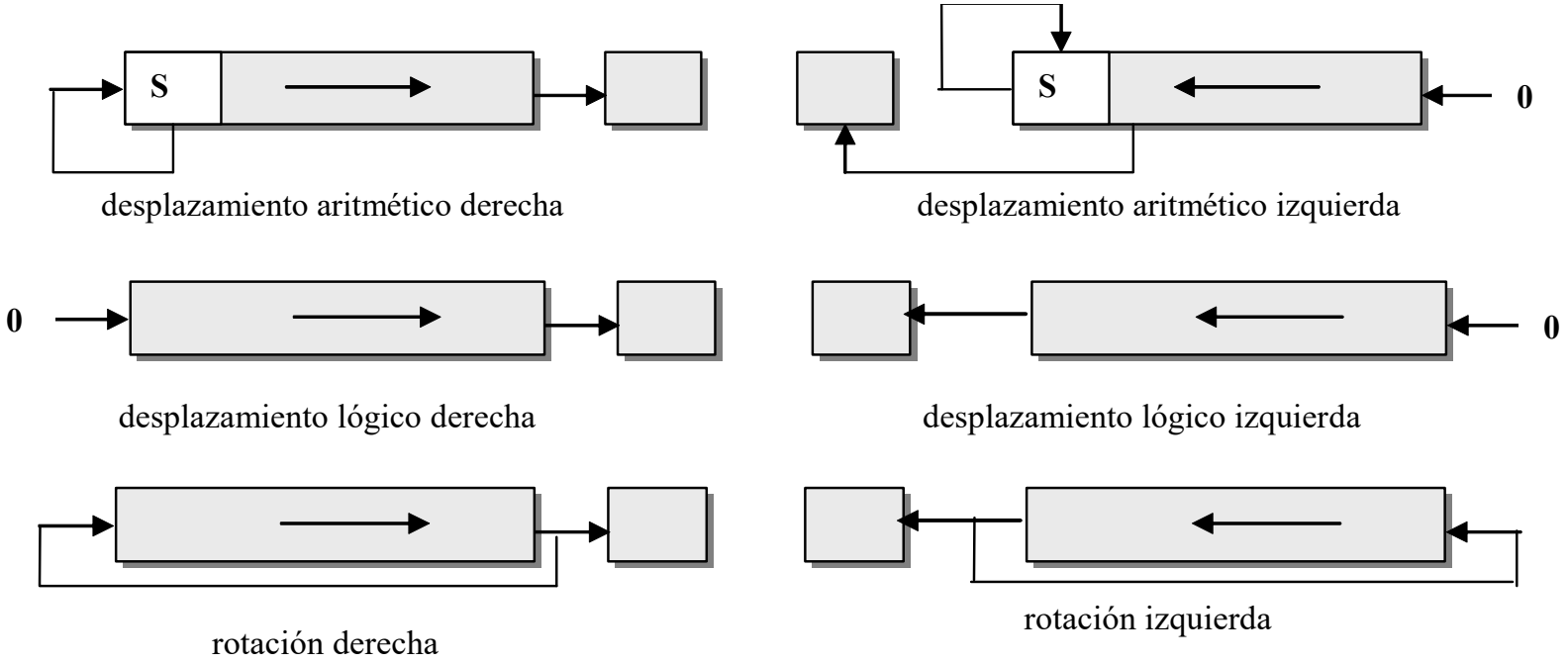
- Una instrucción de este tipo deberá especificar los siguientes elementos:
  - Tipo de transferencia. dos alternativas:
    - Instrucción genérica, con un único CO. Ejemplo MOVE
    - Instrucciones diferentes para cada movimiento. Ejemplos TR, STO, LD, PUSH, POP
  - Direcciones de la fuente y destino de la transferencia.

4. Instrucciones que operan sobre datos

Instrucciones de desplazamiento y rotación

Una instrucción de este tipo deberá especificar los siguientes elementos:

- Tipo de operación: desplazamiento aritmético, lógico, rotación (especificado en CO)
- Cuenta: número de posiciones (normalmente 1 bit, especificado en CO)
- Dirección: izquierda, derecha (especificado en CO o en el signo de Cuenta)
- Tipo de dato: normalmente su longitud



4. Instrucciones que operan sobre datos

Instrucciones lógicas y de manipulación de bits

- Manipulan bits individuales en una unidad direccionable con operaciones booleanas.
- Las instrucciones lógicas operan bit a bit sobre los vectores de bits de sus operandos:
  - NOT (complementación)
  - AND (conjunción lógica)
  - OR (disyunción lógica)
  - XOR (disyunción exclusiva)
  - Equivalencia

Ejemplo: Creación de una máscara sobre los 4 bits menos significativos de R1 (AND con R2) y complementación de los 6 bits centrales (XOR con R3):

<R1> = 1010 0101
<R2> = 0000 1111
<R3> = 0111 1110
<R1> AND <R2> = 0000 0101
<R1> XOR <R3> = 1101 1011

- Las instrucciones de manipulación de bits permiten poner a 0 ó a 1 determinados bits del *registro de estado*.

4. Instrucciones que operan sobre datos

Instrucciones aritméticas

- Casi todos los repertorios disponen de las operaciones aritméticas básicas de suma resta multiplicación y división sobre enteros con signo (coma fija).
- Con frecuencia disponen también de operaciones sobre reales (coma flotante) y decimales (BCD).
  - Suma
  - Resta
  - Multiplicación
  - División
  - Cambio de signo
  - Valor absoluto
  - Incremento
  - Decremento
  - Comparación

Instrucciones de transformación de datos

- Traducción: de una zona de memoria utilizando una tabla de correspondencia.
- Conversión: del formato de representación de los datos.

Ejemplo instrucción *Translate* (TR) del S/370 (BCD → binario o EBCDIC → ASCII)

4. Instrucciones que operan sobre datos

Instrucciones de entrada/salida

• Coinciden con las de referencia a memoria (carga y almacenamiento) en los procesadores que disponen de un único espacio de direcciones compartido para Memoria y E/S.

• Los procesadores con espacios independientes disponen de instrucciones específicas:

- Inicio de la operación: START I/O
- Prueba del estado: TEST I/O
- Bifurcación sobre el estado: BNR (Branch Not Ready)
- Entrada: IN que carga en el procesador un dato de un dispositivo periférico
- Salida: OUT que lo envía a un dispositivo periférico.

Instrucciones de manipulación de direcciones

Calculan la dirección efectiva de un operando para manipularla como un dato.

Ejemplos (68000):

LEA.L opf, An: *dirección fuente --> An*
lleva la dirección efectiva del operando fuente *opf* al registro de direcciones *An*

PEA.L opf: dirección fuente --> Pila
lleva la dirección efectiva del operando fuente *opf* a la Pila de la máquina

5. Instrucciones de control del flujo de ejecución

Instrucciones de control del flujo de ejecución

- Bifurcación condicional (e incondicional)
- Bifurcación a subrutinas

Estas instrucciones desempeñan un triple papel en la programación:

- Toma de decisiones en función de resultados previamente calculados.
- Reutilización de parte del código del programa.
  - Bucles iterativos (instrucciones de bifurcación condicional).
  - Subrutinas (funciones y procedimientos).
- Descomposición funcional del programa.
  - Segundo papel que cumplen las subrutinas dentro de un programa: facilitar la modularidad del mismo, y con ello su depuración.

5. Instrucciones de control del flujo de ejecución

Instrucciones de bifurcación condicional

- Componentes básicos de las instrucciones de salto condicional:
  - código de operación
  - condición de salto
  - dirección destino.

CO	Condición	Dirección

- Semántica:

```
IF Condición = True THEN
        CP <-- Dirección        (CP = Contador de Programa)
ELSE
        CP <-- <CP> + 1
```

- Gestión de la Condición
  - Generación
  - Selección
  - Uso

5. Instrucciones de control del flujo de ejecución

Instrucciones de bifurcación condicional: gestión de la condición

- **Generación**

  Implícita: la condición se genera como efecto lateral de la ejecución de una instrucción de manipulación de datos (ADD, SUB, LOAD, etc.)

  Explícita: existen instrucciones específicas de comparación o test que sólo afectan a las condiciones (CMP, TST, etc.)

- **Selección**

  Simple: afecta a un sólo bit del registro de condiciones (Z, N, C, etc.)

  Compuesta: afecta a una combinación lógica de condiciones simples (C+Z, etc.)

- **Uso**

  - Almacenamiento de la condición en el registro de estado o condición (2 instrucciones):
    - una instrucción de gestión de datos o comparación genera la condición
    - otra instrucción selecciona la condición y bifurca en caso que sea cierta (True)
  - Almacenamiento de la condición en un registro general (2 instrucciones)
    - una instrucción de gestión de datos o comparación genera la condición
    - otra instrucción selecciona la condición y bifurca en caso que sea cierta (True)
  - Sin almacenamiento de la condición (1 instrucción)
    - genera, selecciona la condición y bifurca en caso que sea cierta
      (BRE R1, R2, DIR : salta a DIR si <R1> = <R2>)

5. Instrucciones de control del flujo de ejecución

Condiciones de salto

Simples		Compuestas	
Nemotéctico	**Condición**	**Nemotécnico**	**Condición**
CC (carry clear)	¬C	HI (high)	¬C • ¬Z
CS (carry set)	C	LS (low or same)	C + Z
NE (not equal)	¬Z	HS (high or same)	¬C
EQ (equal)	Z	LO (low)	C
VC (overflow clear)	¬V	GE (greater or equal)	N xor ¬V
VS (overflow set)	V	LT (less than)	N xor V
PL (plus)	¬N	GT (greater than)	(N xor ¬V)and¬Z
MI (minus)	N	LE (less or equal)	(N xor V)orZ

Especificación de la dirección

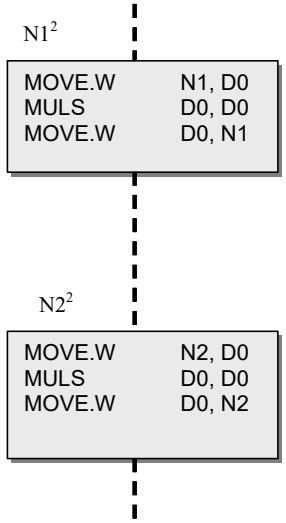
- Direccionamiento directo (Jump)
- Direccionamiento relativo (Branch)
- Direccionamiento implícito (Skip: salta una instrucción si se cumple la condición)
- Instrucciones de procesamiento de datos condicionadas (ARM)

5. Instrucciones de control del flujo de ejecución

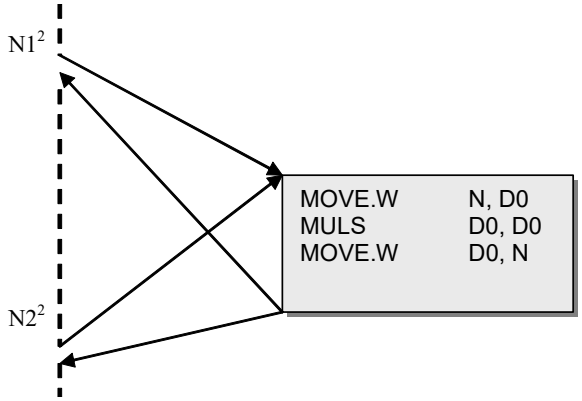
Subrutinas

En diferentes puntos de un programa se realiza la misma operación
(ej. elevar al cuadrado un número): 2 alternativas

MACRO o *subrutina abierta:*
Reescribe el código en los puntos donde se realiza la operación



SUBRUTINA o *subrutina :*
Reutiliza el código bifurcando cada vez que se realiza la operación



Problemas a resolver con las subrutinas:

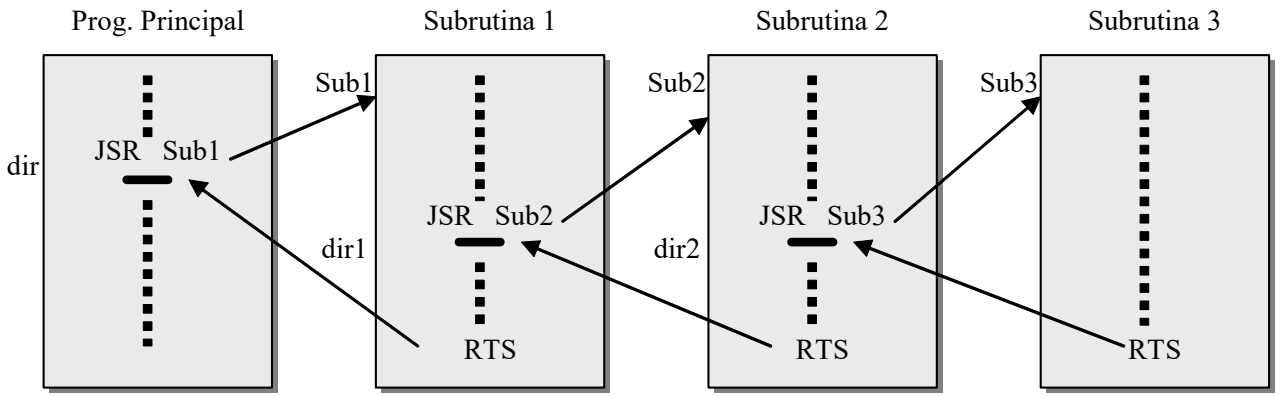
- Control de la dirección de vuelta a la instrucción siguiente
- Paso de parámetros
- Ubicación de las variables locales de la subrutina
- Preservación del contexto del programa

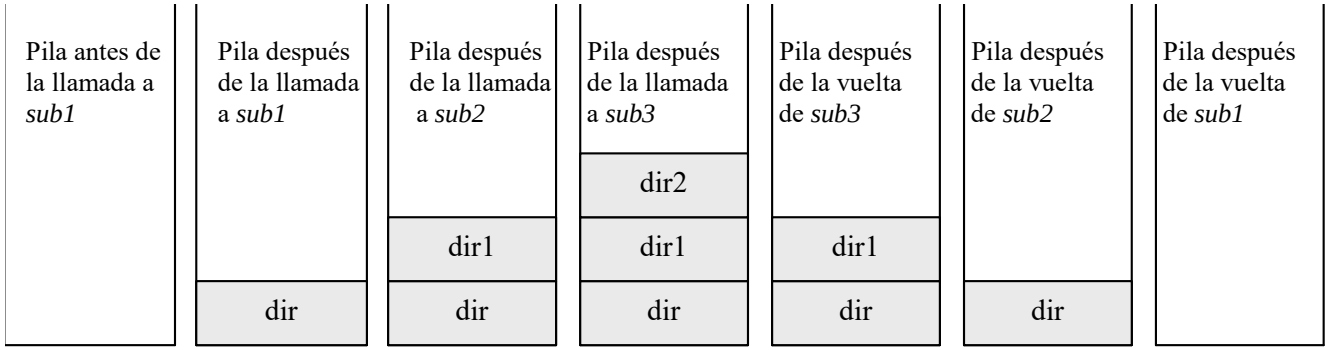
5. Instrucciones de control del flujo de ejecución

Control de la dirección de vuelta

Utilización de una pila (stack)

- Antes de la bifurcación se coloca la dirección de retorno en la pila: instrucción tipo **JSR**
- Lo último que realiza la subrutina es recuperar esta dirección de retorno y llevarla al CP: **RTS**
- Permite el anidamiento y la recursión: ordena los retornos con la extracción de la Pila.

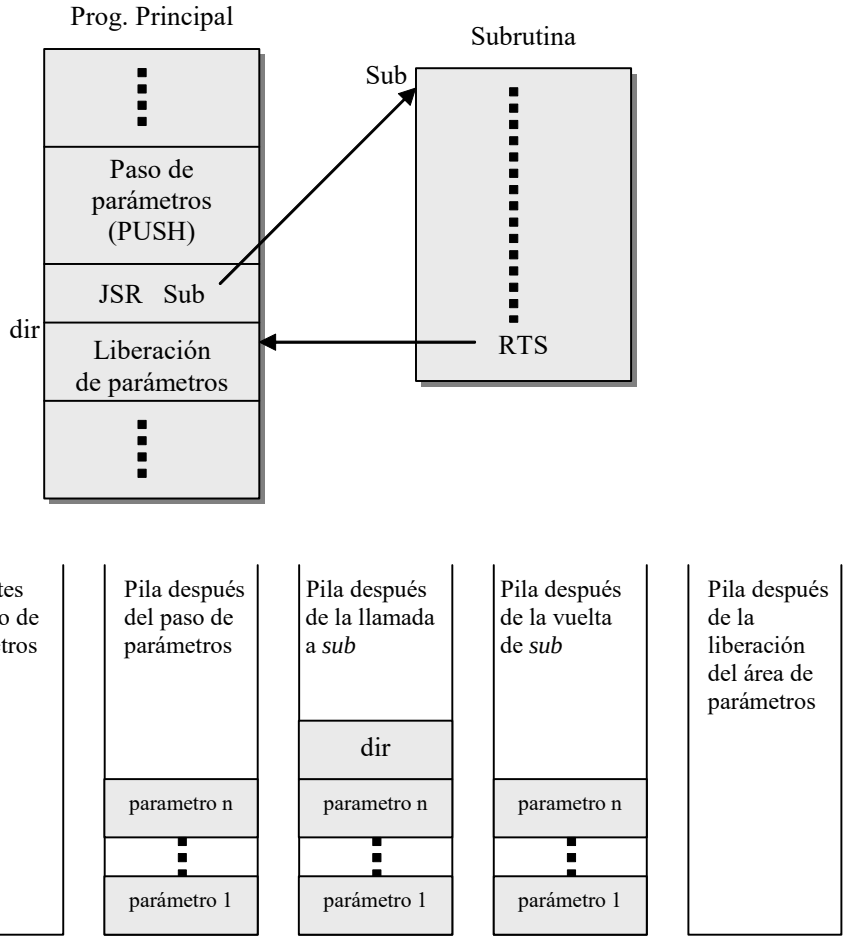




5. Instrucciones de control del flujo de ejecución

Paso de parámetros

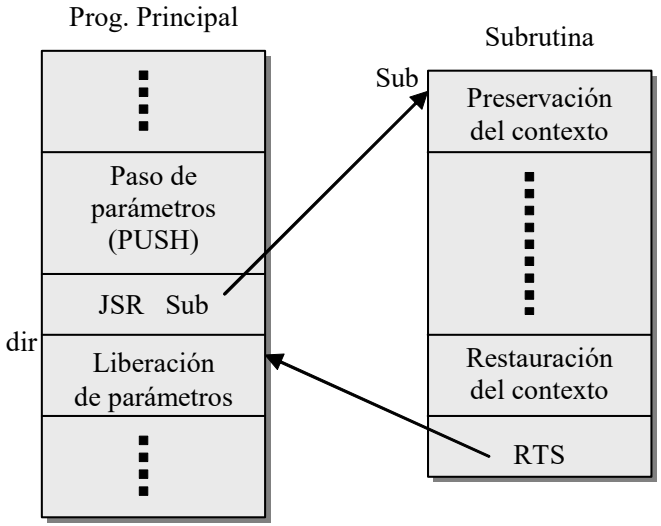
- **También se puede utilizar la misma pila**

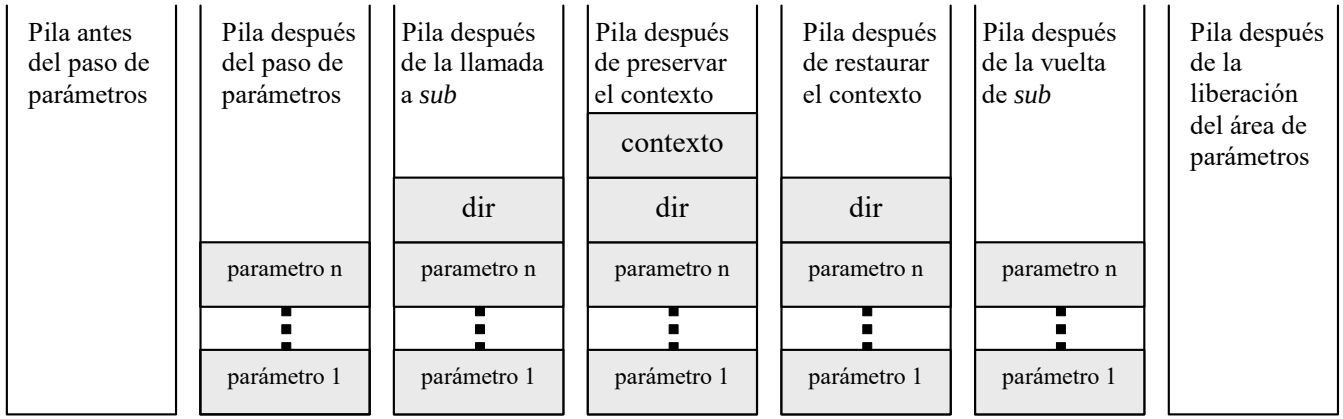


5. Instrucciones de control del flujo de ejecución

Preservación del contexto

- La subrutina comienza salvando todos los registros que se modifican en su ejecución
- Finaliza restaurando el valor de dichos registros

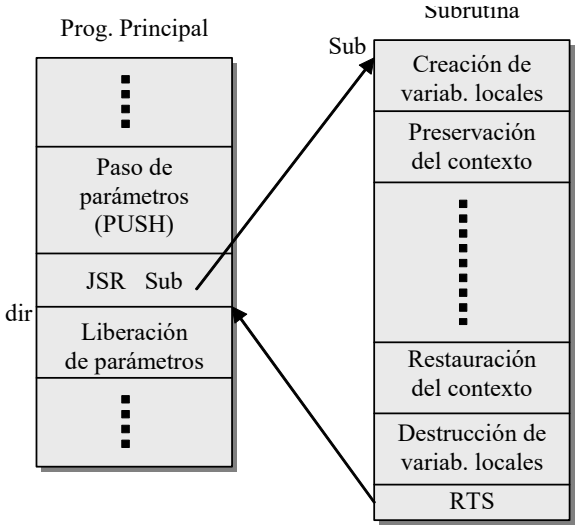


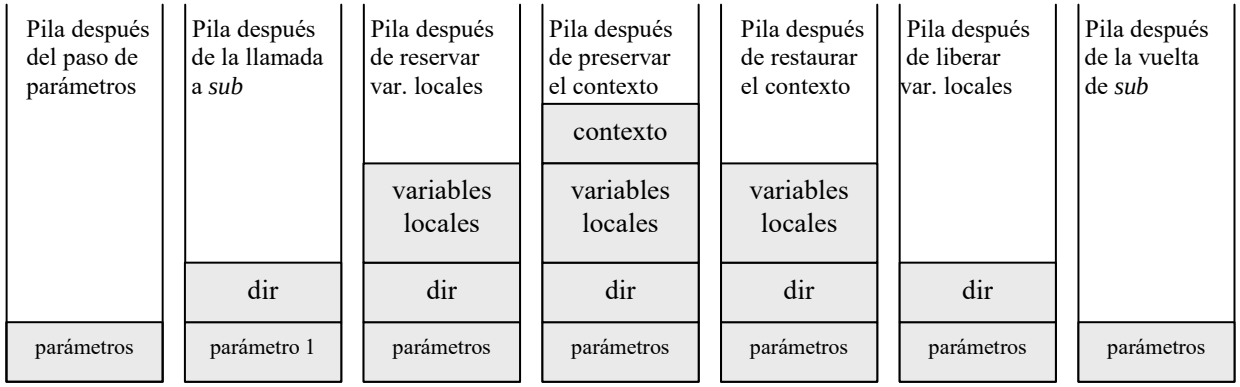


5. Instrucciones de control del flujo de ejecución

Variables locales de la subrutina

- Subrutina reentrante ➔ conjunto nuevo de variables locales por llamada.
- Se utiliza la Pila para el soporte de la gestión dinámica de las llamadas a subrutinas.
- En cada llamada se introduce en la Pila un Registro de Activación (RA)

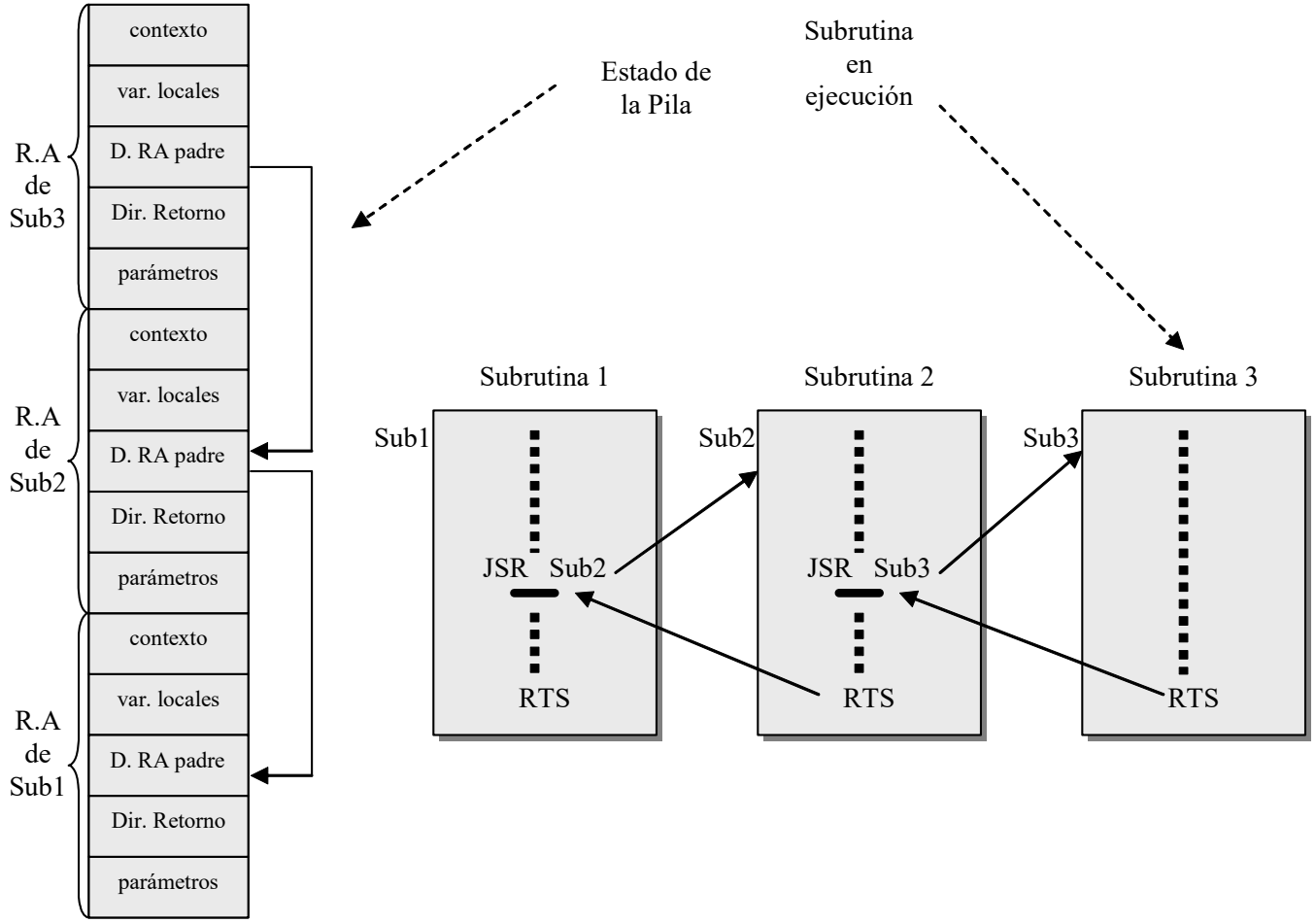




5. Instrucciones de control del flujo de ejecución

Recuperación del RA de una subrutina cuando se vuelve de una llamada

- Antes de realizar la llamada se guarda en el nuevo RA la dirección del antiguo
- Los RAs quedan enlazados a través de este nuevo campo



5. Instrucciones de control del flujo de ejecución

Instrucciones de salto en el ARM

CO	Condición
B	Incondicional
BAL	Siempre
BEQ	Igual
BNE	No igual
BPL	Más
BMI	Negativo
BCC	Carry cero
BLO	Menor
BCS	Carry uno
BHS	Mayor o igual
BVC	Overflow cero
BVS	Overflow uno
BGT	Mayor que
BGE	Mayor o igual
BLT	Menor que
BLE	Menor o igual
BHI	Mayor
BLS	Menor o igual

Instrucciones condicionadas

Código con instrucciones de salto condicional

```
                CMP   r0, #5
                BEQ   Bypass
                ADD  r1, r1, r0
                SUB   r1, r1, r2
Bypass          ……..
```

Código equivalente con instrucciones condicionadas

```
                CMP      r0, #5
                ADDNE  r1, r1, r0
                SUBNE   r1, r1, r2
Bypass          ……..
```

Instrucción básica de salto incondicional

```
                B    etiqueta    ; Salto incondicional a etiqueta
                …….
                …….
Etiqueta        …….
```

Instrucciones de salto condicional

```
                MOV    r0, #10     ;Inicializa el contador del bucle
bucle           …….
                …….
                …….
                SUB    r0, r0, #1    ; Decrementa el contador
                CMP    r0, #0        ; ¿es cero?
                BNE    bucle         ; Bifurca si r0 ≠ 0
```

Bit S permite cambiar el cc en CPSR

```
                MOV    r0, #10     ;Inicializa el contador del bucle
bucle           …….
                …….
                …….
                SUBS   r0, r0, #1    ; Decrementa el contador
                BNE    bucle         ; Bifurca si r0 ≠ 0
```

5. Instrucciones de control del flujo de ejecución

Salto a subrutina en el ARM

Instrucción de salto a subrutina
BL sub

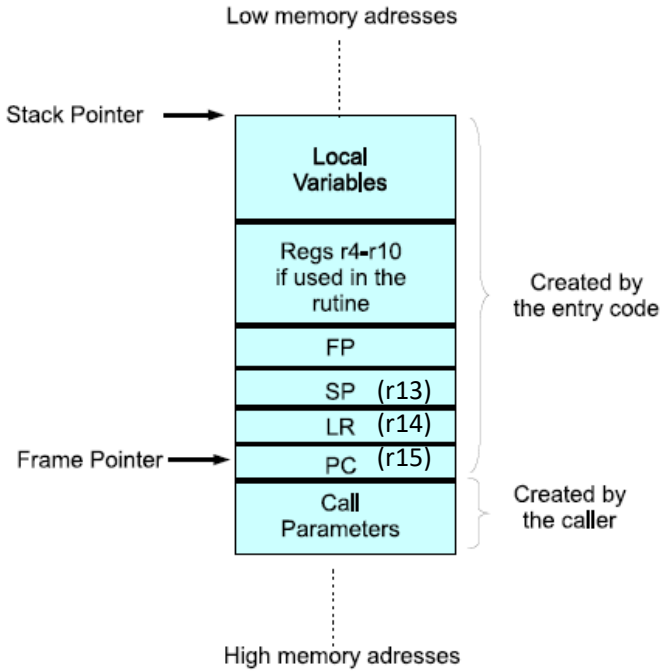
(Branch-and-Link): r14 := r15
r15 := sub (dirección de subrutina)

Instrucción de vuelta de subrutina
MOV r15, r14

Ejemplo

```
                BL   SUB1
                .......

SUB1            STMED  r13!, {r0-r2, r14}     ;salva registros en la pila
                .......
                LDMED  r13!, {r0-r2, r14}     ;recupera registros de la pila
                MOV    r15, r14
```



6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

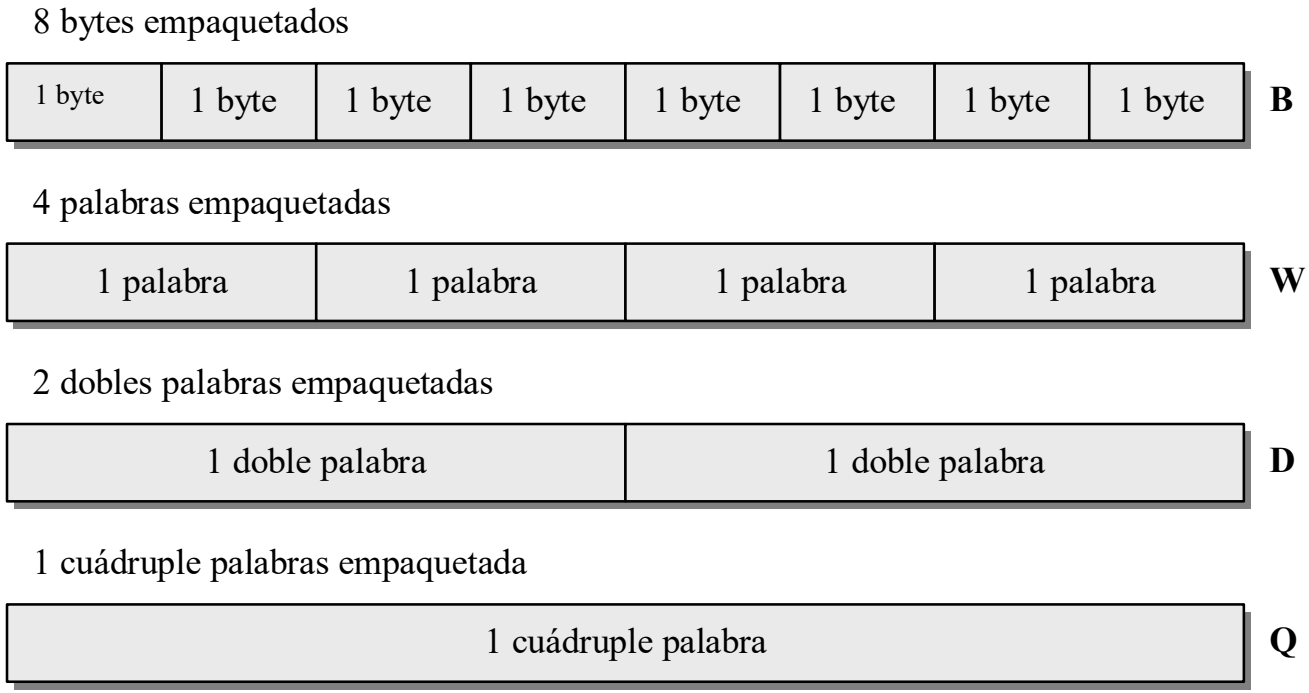
- Instrucciones optimizadas para realizar operaciones multimedia
- Introducidas en la microarquitectura P6 de IA-32 como tecnología MMX del Pentium
- Operan sobre los datos siguiendo un modo SIMD(Single Instruction Multiple Data)
- Mejoran la velocidad sobre la alternativa secuencial del orden de 2 a 8 veces
- Las extensiones SIMD se han utilizado en los repertorios de otros procesadores:
  - SPARC de Sun Microsystems con el nombre de VIS (Visual Instruction Set)
  - PA-RISC de Hewlett-Packard con el nombre de conjunto de instrucciones MAX-2
- Los programas multimedia operan sobre señales muestreadas de vídeo o audio representadas por grandes vectores o matrices de datos escalares

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Tipos de datos

- Las aplicaciones multimedia suelen utilizar los siguientes tamaños de datos:
  - **Gráficos y vídeos**: **8 bits** para cada punto (escala de grises) o componente de color
  - **Audio**: **16 bits** para cada valor muestreado de la señal
  - **Gráficos 3D:** pueden utilizar hasta **32 bits** por punto

- Las instrucciones MMX tienen definidos cuatro tipos de datos:

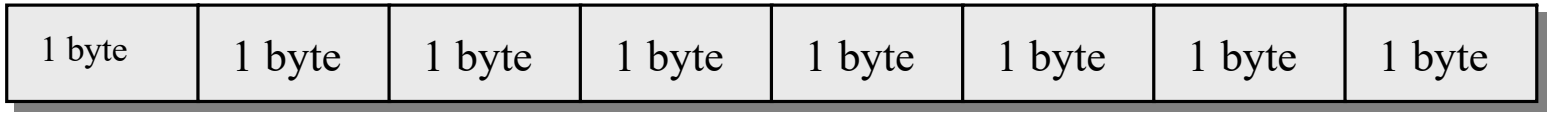


6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Registros MMX

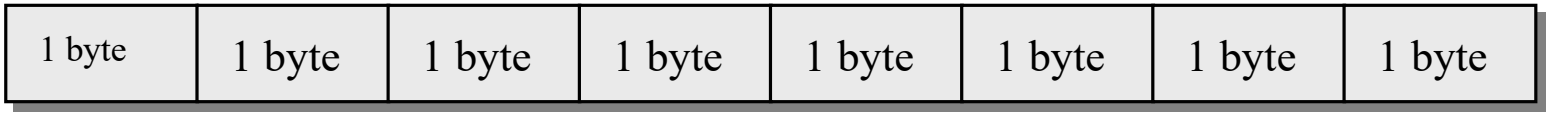
– Existen **8 registros de propósito general de 64 bits** para MMX que se designan como *mmi (i = 0,...7)*

mm0



. . .

mm7



6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

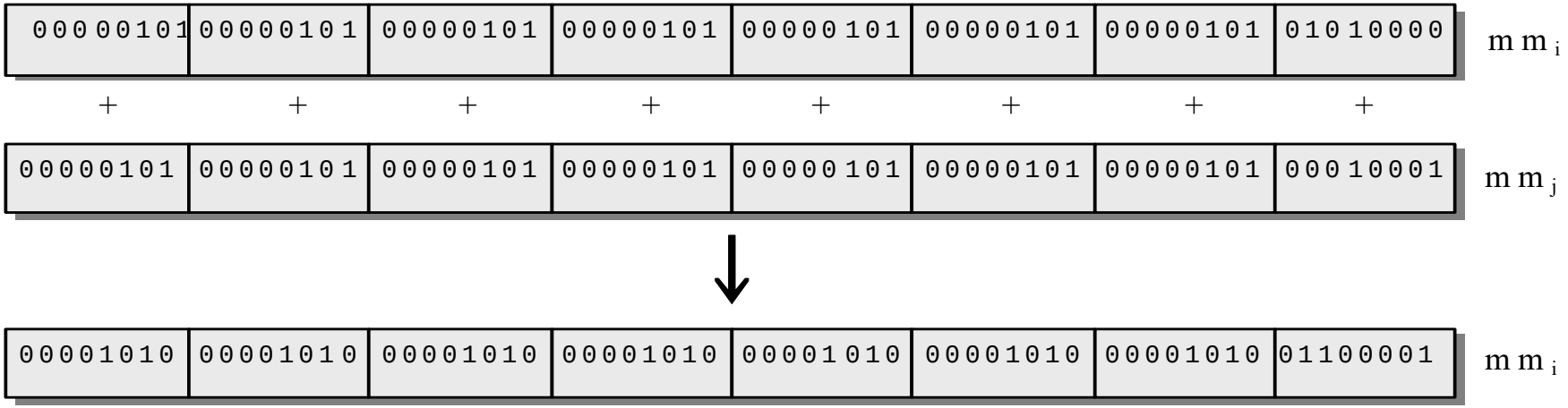
Repertorio de instrucciones MMX

- Formato
  - CSO [<tipo de datos>] <registros MMX>
  - CSO = Código Simbólico de Operación
  - Tipo de datos = B, W, D ó Q

- Ejemplo: Suma paralela con truncamiento de dos registros MMX con 8 bytes

PADDB mm_i, mm_j

$<mm_i> + <mm_j> \rightarrow mm_j$



6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Instrucciones Aritméticas (1)

- Lo forman 9 instrucciones, 3 de suma, 3 de resta y 3 de multiplicación
- Se distingue entre la aritmética con saturación y con truncamiento
- La aritmética con truncamiento elimina el bit de desbordamiento
- La aritmética con saturación utiliza el mayor/menor valor representable cuando hay desbordamiento.

  Ejemplo:

	1111 0000 0000 0000	
+	0011 0000 0000 0000	
rebose --> **1**	0010 0000 0000 0000	resultado en **aritmética con truncamiento**
	1111 1111 1111 1111	resultado en **aritmética con saturación**

- La aritmética con saturación se utiliza cuando se suman dos números que representan intensidad de imagen para que el resultado no sea menor que la suma de las intensidades originales

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

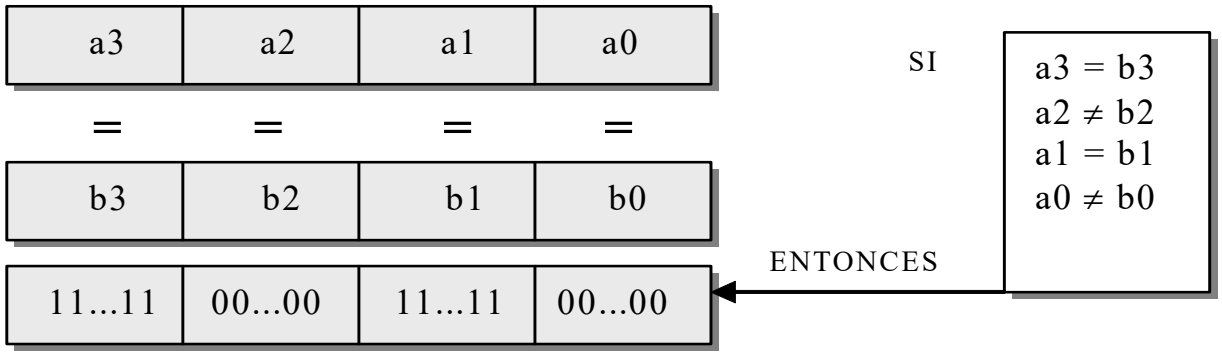
Instrucciones Aritméticas (2)

Instrucciones Aritméticas	Semántica
PADD [B, W, D] mm_i, mm_j	Suma paralela con truncamiento de 8 B, 4 W ó 2 D
PADDS [B, W] mm_i, mm_j	Suma con saturación
PADDUS [B, W] mm_i, mm_j	Suma sin signo con saturación
PSUB [B,W, D] mm_i, mm_j	Resta con truncamiento
PSUBS [B,W] mm_i, mm_j	Resta con saturación
PSUDUS [B, W] mm_i, mm_j	Resta sin signo con saturación
PMULHW *mmi, mmj*	Multiplicación paralela con signo de 4 palabras de 16 bits, con elección de los 16 bits más significativos del resultado de 32 bits
PMULLW mm_i, mm_j	Multiplicación paralela con signo de 4 palabras de 16 bits, con elección de los 16 bits menos significativos del resultado
PMADDWD mm_i, mm_j	Multiplicación paralela con signo de 4 palabras de 16 bits, y suma de pares adyacentes del resultado de 32 bits

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Instrucciones de Comparación

- Las instrucciones MMX de comparación **generan una máscara de bits como resultado**
- Posibilitan la realización de cálculos dependientes de los resultados eliminando la necesidad de instrucciones de bifurcación condicional
- La máscara resultado de una comparación tiene todo 1's cuando la comparación da resultado *true*, y todo 0's cuando la comparación es *false*



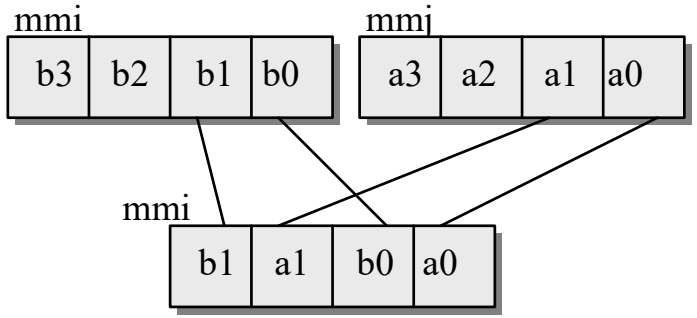
Instrucciones de Comparación	Semántica
PCMPEQ [B, W, D] mm_i, mm_j	Comparación paralela de igualdad, el resultado es una máscara de 1´s si True y de 0´s si False
PCMPGT [B, W, D] mm_i, mm_j	Comparación paralela de > (magnitud), el resultado es una máscara de 1´s si True y de 0´s si False

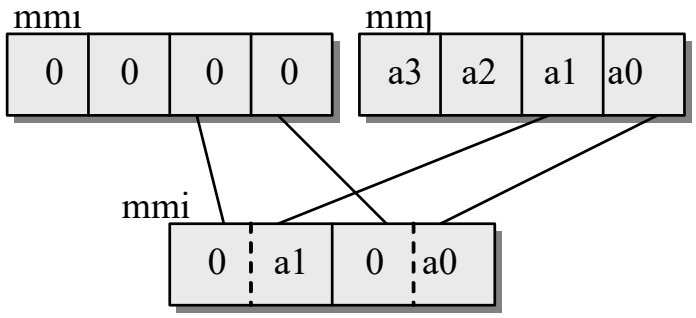
6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Instrucciones de Conversión

MMX utiliza instrucciones para realizar conversiones entre datos:

- *UNPACK* toma dos operandos y los entrelaza. Se utiliza para producir un conjunto de datos de mayor formato a partir de otro menor utilizando un registro con 0's (p.e., de 16 a 32 bits)
- *PACK* realiza el proceso inverso





Instrucciones de Conversión	Semántica
PACKUSWB mm_i, mm_j	Empaqueta palabras en bytes con saturación sin signo
PACKSS [WB, DW] mm_i, mm_j	Empaqueta palabras en bytes con saturación con signo, o palabras dobles en palabras
PUNPCKH [BW, WD, DQ] mm_i, mm_j	Desempaqueta en paralelo los B, W o D más significativos del registro MMX. Mezcla entrelazada
PUNPCKL [BW, WD, DQ] mm_i, mm_j	Desempaqueta en paralelo los B, W o D menos significativos del registro MMX. Mezcla entrelazada

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Instrucciones Lógicas

- Las 4 operaciones lógicas **se realizan bit a bit sobre los 64 bits de los operandos**
- Combinadas con las máscaras resultantes de las instrucciones de comparación permiten la realización de cálculos alternativos en función de resultados

Instrucciones Lógicas	Semántica
PAND mm_i, mm_j	AND lógico bit a bit (64 bits)
PANDN mm_i, mm_j	Primero NOT sobre mm_i *y luego* AND lógico bit a bit (64 bits)
POR mm_i, mm_j	OR lógico bit a bit (64 bits)
PXOR mm_i, mm_j	XOR lógico bit a bit (64 bits)

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Instrucciones de Desplazamiento

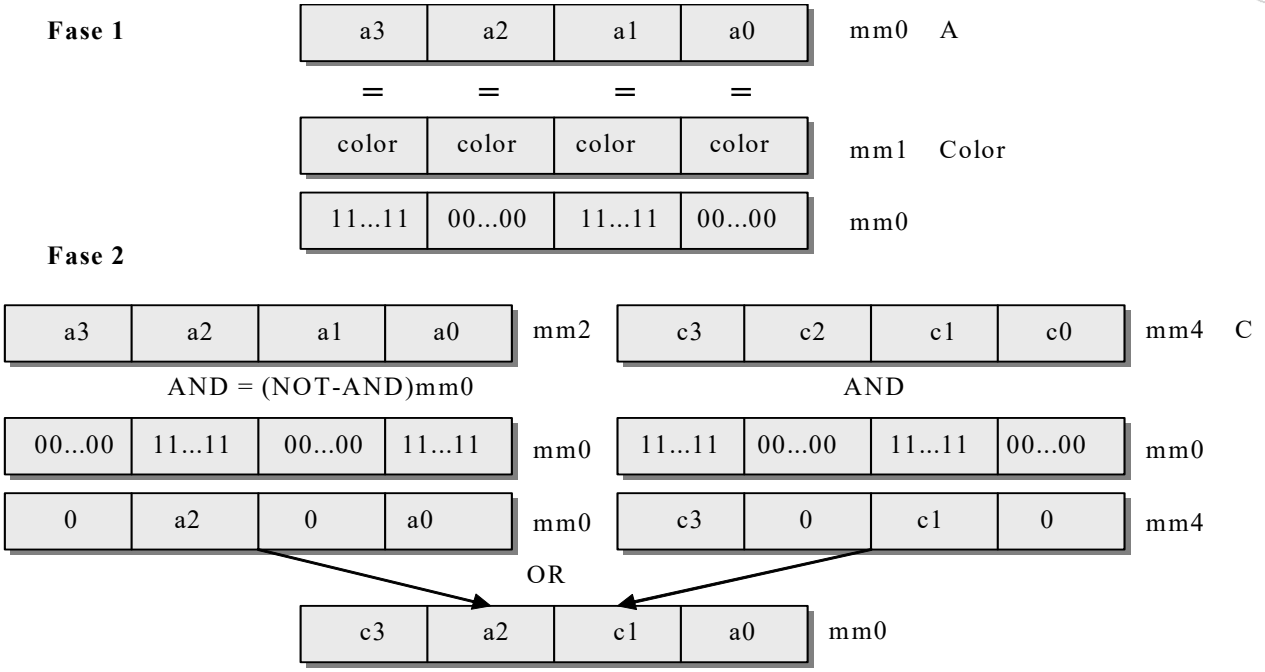
- Realizan desplazamientos y rotaciones sobre cada elemento de datos de uno de sus operandos. El otro indica el número de bits de desplazamiento o rotación.

Instrucciones de Desplazamiento	Semántica
PSLL [W, D, Q] mm_i	Desplazamiento lógico izquierda paralelo de W, D ó Q tantas veces como se indique en el registro MMX o valor inmediato
PSRL [W, D, Q] mm_i	Desplazamiento lógico derecha paralelo de W, D ó Q tantas veces como se indique en el registro MMX o valor inmediato
PSRA [W, D,] mm_i	Desplazamiento aritmético derecha paralelo de W ó D tantas veces como se indique en el registro MMX o valor inmediato

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Ejemplo 1: Codificación MMX de un bucle iterativo

```
for i = 0 to 3
if a[i] = color then
  Salida[i] = c[i]
else Salida[i] = a[i]
```




```
MOVQ     mm0, A      ;carga el array A
MOVQ     mm2, mm0 ;hace copia del array A
MOVQ     mm4, C      ;carga el array C
MOVQ     mm1, Color ; carga Color
PCMPEQW  mm0, mm1 ;compara A con Color
PAND     mm4, mm0 ;selecciona elementos de C
PANDN    mm0, mm2 ;selecciona elementos de A
POR      mm0, mm4 ;genera la salida
```

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Ejemplo 2: Programación de una transición entre imágenes

Función de desvanecimiento gradual de una imagen B y aparición simultanea de otra imagen A

Las dos imágenes A y B se combinan en un proceso que efectúa sobre cada par de *pixels* homólogos la media ponderada por un factor de desvanecimiento *fade* que irá variando de 0 a 1:

$$Pixel\_resultante = Pixel\_A * fade + Pixel\_B*(1 - fade)$$

- Resultado: transformación gradual de la imagen B en la imagen A a medida que *fade* pasa de *0* a *1*

Si una imagen tiene una resolución de 640 X 480 y en el cambio de imagen se emplean 255 valores posibles para *fade* (*fade* codificado con 1 byte), será necesario ejecutar el proceso 640*480*255 = 7.833.600 veces.

Si la imagen fuese en color necesitaríamos esa cantidad multiplicada por 3 (tres componentes básicas del color), es decir 23.500.800.

- Cada pixel ocupa 1 byte y hay que desempaquetarlo previamente a un valore de 16 bits.
- Para secuenciar las operaciones pongamos la expresión anterior de la siguiente forma:

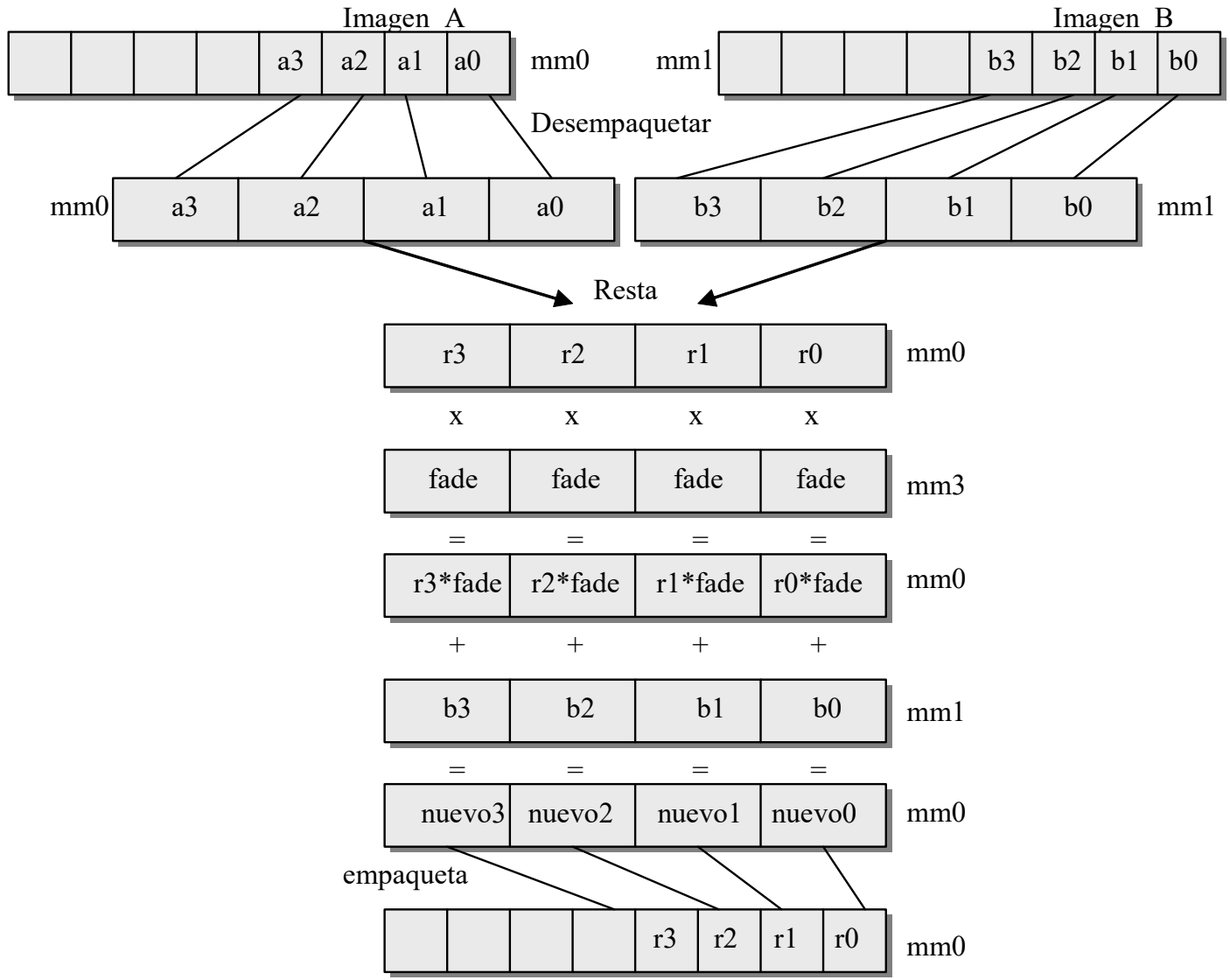
$$Pixel\_resultante = Pixel\_A * fade + Pixel\_B - Pixel\_B*fade =$$

$$(Pixel\_A - Pixel\_B) * fade + Pixel\_B$$

- Se resta a *Pixel_A* el valor de *Pixel_B*, se multiplica el resultado por *fade* y se suma *Pixel_B*
- A estas operaciones hay que añadir las previas de movimiento de datos y desempaquetamiento

6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Ejemplo 2 (continuación): Esquema MMX del proceso:



6. Instrucciones paralelas SIMD (Single Instruction Multiple Data)

Ejemplo 2 (continuación): Programa MMX

```
PXOR      mm7, mm7          ;pone a cero mm7
MOVQ      mm3, valor_fade   ;carga en mm3 el valor de fade replicado 4 veces
MOVD      mm0, imagenA      ;carga en mm0 la intensidad de 4 pixels de la imagen A
MOVD      mm1, imagenB      ;carga en mm1 la intensidad de 4 pixels de la imagen B
PUNPCKLBW mm0, mm7          ;desempaqueta a 16 bits los 4 pixels de mm0
PUNPCKLBW mm1, mm7          ;desempaqueta a 16 bits los 4 pixels de mm1
PSUBW     mm0, mm1          ;resta la intensidad de la imagen B de la de A
PMULHW    mm0, mm3          ;multiplica el resultado anterior por el valor de fade
PADDDW    mm0, mm1          ;suma el resultado anterior a la imagen B
PACKUSWB  mm0, mm7          ;empaqueta en bytes los 4 resultados de 16 bits
```

7. Rendimiento

Medidas del rendimiento de un computador

- Para poder **comparar diferentes procesadores** hace falta establecer una medida del rendimiento que permita cuantificar los resultados de la comparación

  - **Métrica:** establece la unidad de medida, que casi siempre es el tiempo, aunque hay que considerar dos aspectos diferentes del tiempo:
    - **Tiempo de ejecución:** tiempo que tarda en realizarse una tarea determinada
    - **Productividad** (througput): tareas realizadas por unidad de tiempo

  El interés por uno u otro aspecto dependerá del punto de vista de quien realiza la medida: un usuario querrá minimizar el tiempo de respuesta de su tarea, mientras que el responsable de un centro de datos le interesará minimizar el número de trabajos que realiza el centro por unidad de tiempo (productividad)

  - **Patrón de medida:** establece los programas que se utilizan para realizar la medida (**bemchmarks**). Existen muchos posibles benchmarks aunque los más utilizados son:
    - Nucleos de programas reales: SPEC
    - Programas sintéticos: TPC

7. Rendimiento

Tiempo de ejecución

- ***Tiempo de respuesta*:** tiempo para completar una tarea (que percibe el usuario).
- ***Tiempo de CPU:*** tiempo que tarda en ejecutarse un programa, sin contar el tiempo de E/S o el tiempo utilizado para ejecutar otros programas. Se divide en:
  - ***Tiempo de CPU utilizado por el usuario:*** tiempo que la CPU utiliza para ejecutar el programa del usuario sin tener en cuenta el tiempo de espera debido a la E/S
  - ***Tiempo de CPU utilizado por el S.O.:*** tiempo que el S.O. emplea para realizar su gestión interna.
- La función ***time*** de Unix visualiza estas componentes: 90.7u 12.9s 2:39 65%, donde:
  - Tiempo de CPU del usuario = 90.7 segundos
  - Tiempo de CPU utilizado por el sistema = 12.9 segundos
  - Tiempo de CPU= 90.7 seg.+ 12.9seg = 103.6
  - Tiempo de respuesta = 2 minutos 39 segundos =159 segundos
  - Tiempo de CPU = 65% del tiempo de respuesta = 159 segundos*0.65 = 103.6
  - Tiempo esperando operaciones de E/S y/o el tiempo ejecutando otras tareas 35% del tiempo de respuesta = 159 segundos*0.35 = 55.6 segundos

7. Rendimiento

Tiempo de CPU

$$Tiempo\ de\ CPU = NI \bullet CPI \bullet Tc.$$

- ***NI*** (Número de Instrucciones): depende del compilador y la arquitectura utilizada
- ***CPI*** (Ciclos medios por Instrucción): depende de la arquitectura y estructura de la máquina
- ***Tc*** (Tiempo de ciclo): depende de la estructura y la tecnología de la máquina

- El número total de ciclos de reloj de la CPU se calcula como:

$$Número\ de\ ciclos\ de\ la\ CPU = CPI \bullet NI = \sum_{i=1}^{n} CPI_i \bullet NI_i$$

  - NI_i = número de veces que el grupo de instrucciones *i* es ejecutado en un programa
  - CPI_i = número medio de ciclos para el conjunto de instrucciones *i*

- Podemos calcular el *CPI* multiplicando cada CPI_i individual por la fracción de ocurrencias de las instrucciones *i* en el programa o frecuencia de aparción de las instrucciones i en el programa f_i.

$$CPI = \frac{\left(\sum_{i=1}^{n} CPI_i \bullet NI_i\right)}{NI} = \sum_{i=1}^{n} \left(CPI_i \bullet \frac{NI_i}{NI}\right) = \sum_{i=1}^{n} CPI_i \bullet f_i$$

- CPI_i debe ser medido, y no calculado a partir de la tabla del manual de referencia

7. Rendimiento

MIPS (Millones de Instrucciones Por Segundo)

$$MIPS = \frac{NI}{Tiempo\ \_de\ \_ejecución\ \ 10^6} = \frac{1}{CPI * \ 10^6\ * Tc} = \frac{Fc}{CPI * 10^6}$$

$$Tiempo\ de\ ejecución = \frac{NI}{MIPS * 10^6}$$

- **Dependen del repertorio de instrucciones**, por lo que resulta un parámetro difícil de utilizar para comparar máquinas con diferente repertorio
- **Varían entre programas ejecutados en el mismo computador**
- **Pueden variar inversamente al rendimiento**, como ocurre en máquinas con hardware especial para punto flotante

$$MIPS_{relativos} = \frac{Tiempo\ de\ ejecución\ en\ la\ máquina\ de\ referencia}{Tiempo\ de\ ejecución\ en\ la\ máquina\ a\ medir} \bullet MIPS_{refeerencia}$$

$$MIPS_{refeerencia} = 1\ (MIPS\ del\ VAX\text{-}11/780)$$

7. Rendimiento

MFLOPS (Millones de operaciones en punto FLOtante Por Segundo)

$$MFLOPS = \frac{Número\ de\ operaciones\ en\ coma\ flotante\ de\ un\ programa}{Tiempo\ de\ ejecución \ \bullet \ 10^6}$$

- Existen operaciones en coma flotante rápidas (como la suma) o lentas (como la división), por lo que puede resultar una medida poco significativa.
- Se han utilizado los MFLOPS normalizados, que dan distinto peso a las diferentes operaciones.
- Por ejemplo: suma, resta, comparación y multiplicación se les da peso 1; división y raíz cuadrada peso 4; y exponenciación, trigonométricas, etc. peso 8:

Productividad (***throughput***)

- Número de tareas ejecutadas en la unidad de tiempo

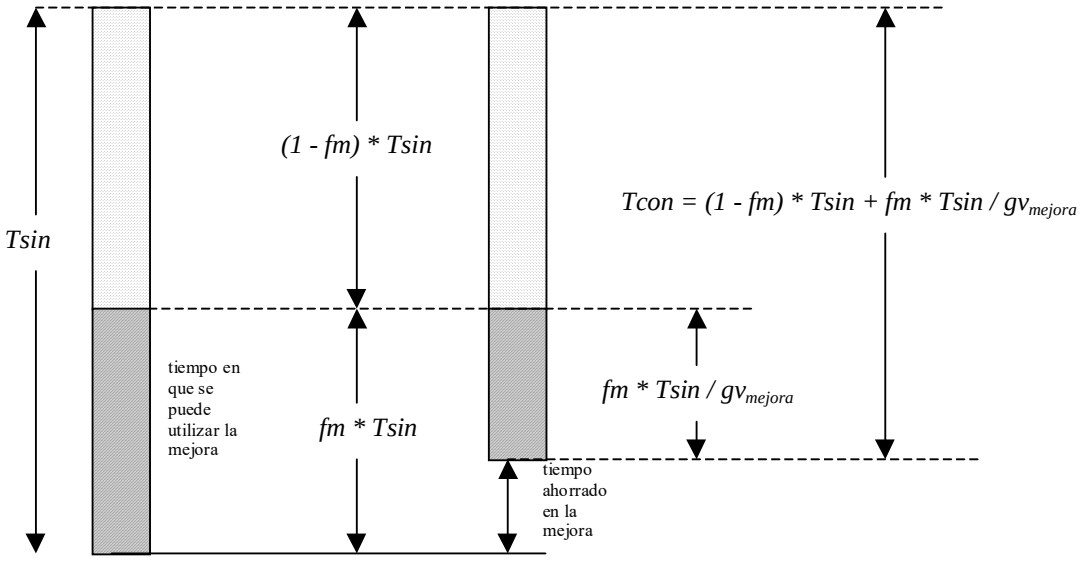
7. Rendimiento

Ganancia de velocidad (*speedup*): Ley de Amdahl

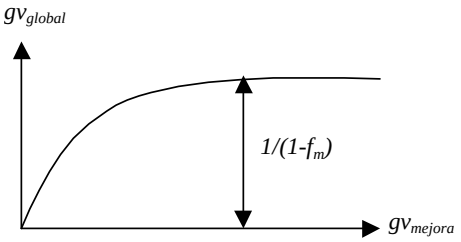
La mejora obtenida en el rendimiento global de un computador al utilizar un modo de ejecución más rápido está limitada por la fracción de tiempo que se puede utilizar dicho modo

•Definición de ganancia de velocidad:

$$gv_{global} = \frac{Tiempo\ de\ ejecución\ sin\ mejora\ (Tsin)}{Tiempo\ de\ ejecución\ con\ mejora\ (Tcon)}$$



$$gv_{global} = \frac{Tsin}{Tcon} = \frac{Tsin}{(1-f_m)*Tsin + \frac{f_m * Tsin}{gv_{mejora}}} = \frac{1}{(1-fm) + \frac{f_m}{gv_{mejora}}}$$



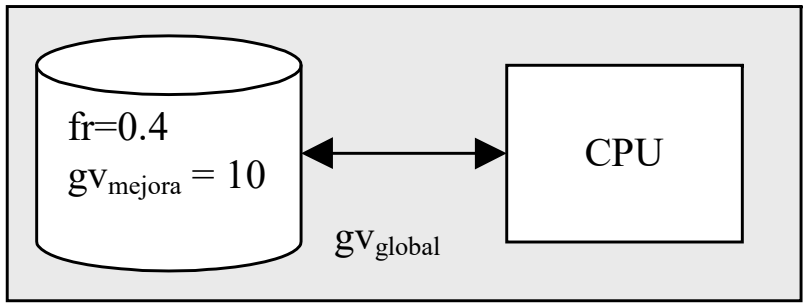
$$lim\_gv_{global} = \frac{1}{1-f_m}$$

$$gv_{mejora} \rightarrow \infty$$

7. Rendimiento

Ejemplo

- En un computador se sustituye el disco magnético por otro10 veces más rápido.
- El disco se utiliza sólo el 40% del tiempo de ejecución.
- ¿Cual es la ganancia de velocidad global?



$gv_{mejora} = 10;\ \ f_m = 0.4 \ \Rightarrow\ gv_{global} = \dfrac{1}{(1-0.4)+\dfrac{0.4}{10}} = \dfrac{1}{0.64} = 1.56$

$lim\ gv_{global} = \dfrac{1}{(1-0.4)} = 1.666$ (máxima ganancia para velocidad del disco infinita)
$gv_{mejora} \dashrightarrow \infty$

7. Rendimiento

Resumen

- Expresión general de la mejora global de rendimiento (ley de Amdahl).

$$gv_{global} == \frac{1}{(1-f_m)+\frac{f_m}{gv_{mejora}}}$$

$f_m =$ fracción de tiempo de utilización del modo que introduce la mejora

$gv_{mejora} =$ ganancia de velocidad relativa al modo que introduce la mejora

- Expresión de la mejora en términos del valor del CPI medio:

$$CPI = \sum_{i=1}^{n} CPI_i \bullet f_i$$

$CPI_i =$ CPI medio del grupo de intrucciones i

$f_i =$ frecuencia de aparción de las instrucciones del grupo i

- Expresión de la mejora en términos de un CPI_{ideal} y penalizaciones introducidas en cada grupo de instrucciones *i* por limitaciones tecnológicas:

$$CPI = CPI_{ideal} + \sum_{i=1}^{n} p_i \bullet f_i$$

$CPI_{ideal} =$ CPI del procesador sin penalizaciones

$p_i =$ penalización introducida en el grupo de instrucciones i

$f_i =$ frecuencia de aparición de las instrucciones del grupo i

$$CPI_{ideal} = \sum_{i=1}^{n} CPI_i \bullet f_i$$

$$CPI = \sum_{i=1}^{n} (CPI_i + p_i) \bullet f_i = \sum_{i=1}^{n} CPI_i \bullet f_i + \sum_{i=1}^{n} p_i \bullet f_i = CPI_{ideal} + \sum_{i=1}^{n} p_i \bullet f_i$$

8. Influencias en el rendimiento de las alternativas de diseño

- Revisaremos las alternativas de diseño de los repertorios de instrucciones (ISA) para seleccionar aquellas que aportan mayor rendimiento al procesador.
- Las alternativas se estudian con datos sobre su presencia en programas reales.
- El resultado caracterizará un repertorio (ISA) que definirá las propiedades generales de los procesadores de tipo RISC.
- Proyectaremos los resultados sobre el procesador DLX, que es un compendio de las principales características de los actuales procesadores RISC:
  - MIPS
  - Power PC
  - Precision Architecture
  - SPARC
  - Alpha.

8. Influencias en el rendimiento de las alternativas de diseño

Tipo de elementos de memoria en la CPU

Tres alternativas

Tipo de máquina	Ventajas	Inconvenientes
Pila		
Acumulador	Instrucciones cortas	Elevado tráfico con Memoria
Registros	Mayor flexibilidad para los compiladores Más velocidad (sin acceso a Memoria)	Instrucciones más largas

- El tráfico con memoria es uno de los cuellos de botella más importantes para el funcionamiento del procesador.

- Se disminuye este tráfico con instrucciones que operen sobre registros: el acceso es más rápido y las referencias a los registros se codifican con menor número de bits.

Conclusión

Se opta por el tipo de máquina con registros de propósito general (RPG)

8. Influencias en el rendimiento de las alternativas de diseño

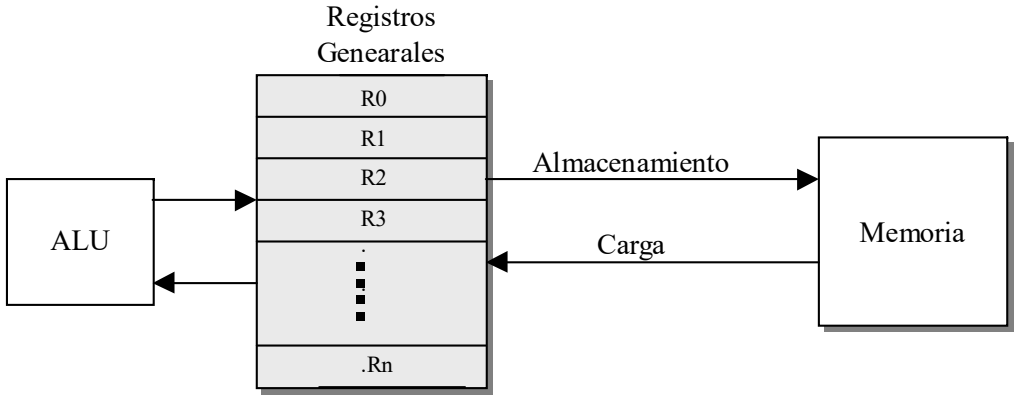
Referencias a memoria en instrucciones ALU

Tres alternativas:

Tipo de máquina	Ventajas	Inconvenientes
Registro-Registro	Ninguna referencia a Memoria Codificación fija => formato simple Generación de código simple	Mayor número de instrucciones por programa
Registro-Memoria	Menor número de instrucciones	Mayor tráfico con Memoria Formato más complejo
Memoria-Memoria	Muchos tipos de direccionamiento Número mínimo de instrucciones por programa	Mucho acceso a memoria Formato complejo

Conclusión

Máquina registro - registro (RR): optimizan el uso de registros, quedando el acceso a memoria limitado a las instrucciones de carga y almacenamiento.



8. Influencias en el rendimiento de las alternativas de diseño

Orden de ubicación de los datos

Dos alternativas:

big-endian	Selección basada en motivos de compatibilidad
litle-endian	

Conclusión

Es indiferente desde el punto de vista del rendimiento.
Serán motivos de compatibilidad los que determinen la elección

Alineamiento de datos

Dos alternativas:

acceso alineado	más rápido
acceso no alineado	más lento en general mayor flexibilidad

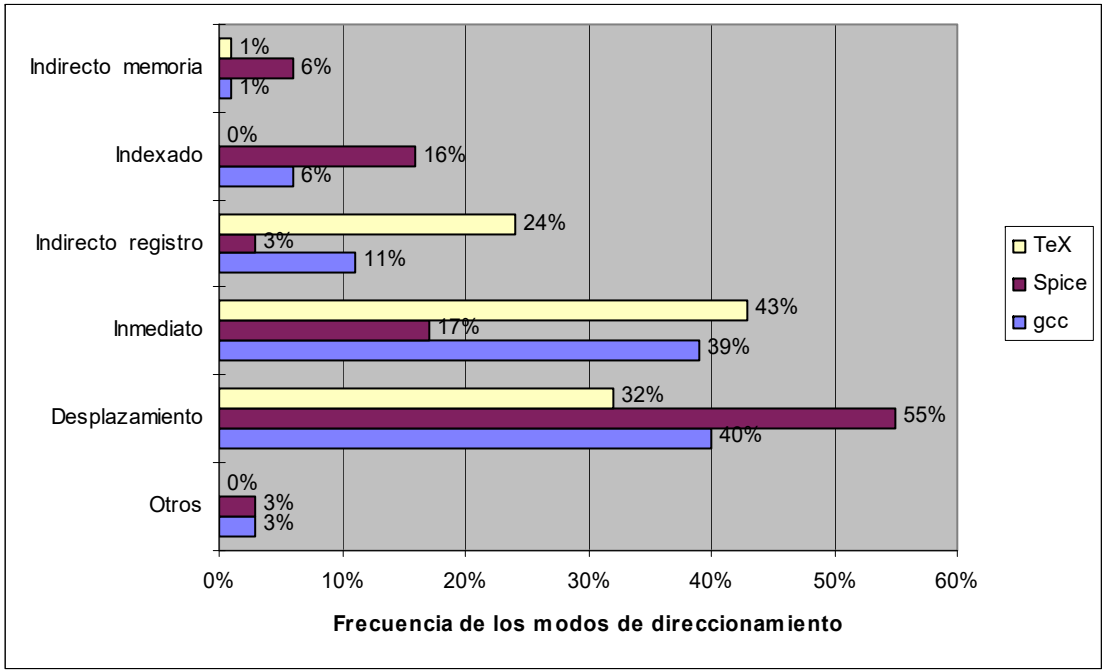
Conclusión

Datos alineados, o si el procesador permite los no alineados,
será el compilador quien genere siempre datos alineados.

8. Influencias en el rendimiento de las alternativas de diseño

Direccionamientos

Resultados de medir los modos de direccionamiento en 3 *SPEC89* sobre el VAX: *Tex, Spice y gcc.*



- Registro + desplazamiento >= 75%.
- Desplazamientos de 12 a 16 bits en un porcentaje del 75% al 99%.
- Campo inmediato de 8 a 16 bits en un porcentaje del 50% al 80%.

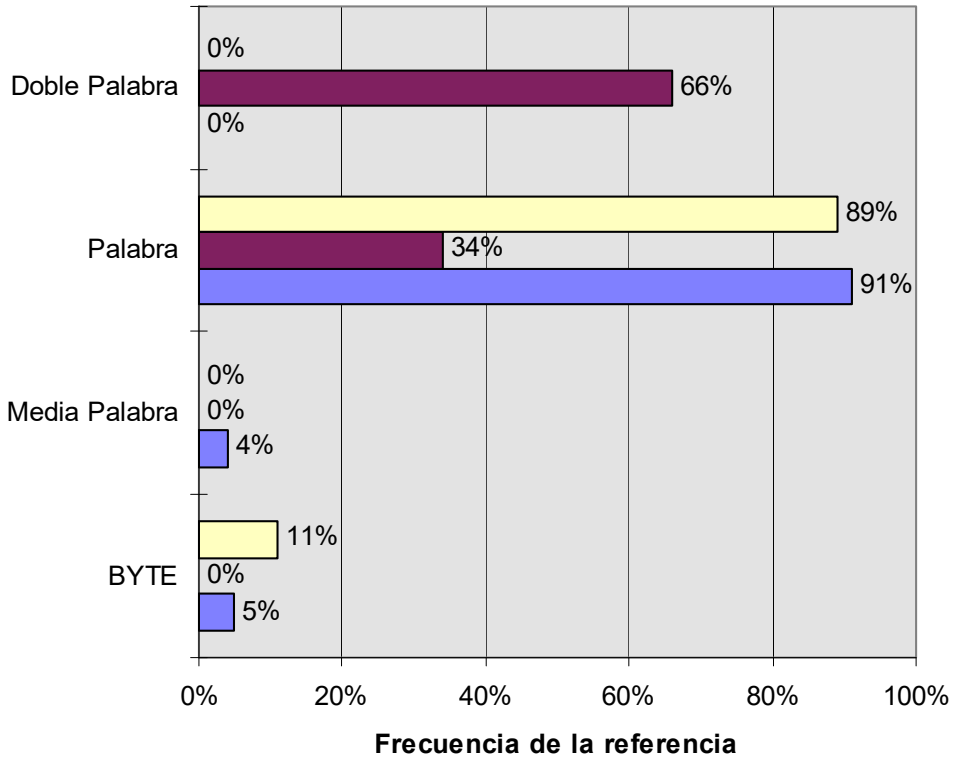
Conclusión

- Direccionamientos registro + desplazamiento y el inmediato
- Tamaños de los desplazamientos de 12 a 16 bits
- Tamaño del dato inmediato de 8 a 16 bits
- La supresión de los modos complejos no afectan decididamente al rendimiento

8. Influencias en el rendimiento de las alternativas de diseño

Datos operando

- Referencias en los *benchmarks* anteriores a los objetos de datos mas usuales (byte, media palabra, palabra y doble palabra)
- Los accesos a datos de longitud palabra o doble palabra dominan sobre los demás.
- Necesidad de acceder al elemento mínimo de resolución del direccionamiento: el byte
- Existencia en los procesadores de operaciones hardware en punto flotante,



Conclusión

- enteros de 16 y 32 bits
- flotantes de 64 bits
- caracteres de 8 bits

8. Influencias en el rendimiento de las alternativas de diseño

Operaciones

- Las operaciones más simples son las que más se repiten en los programas
- Operaciones para programas enteros ejecutándose sobre la familia x86

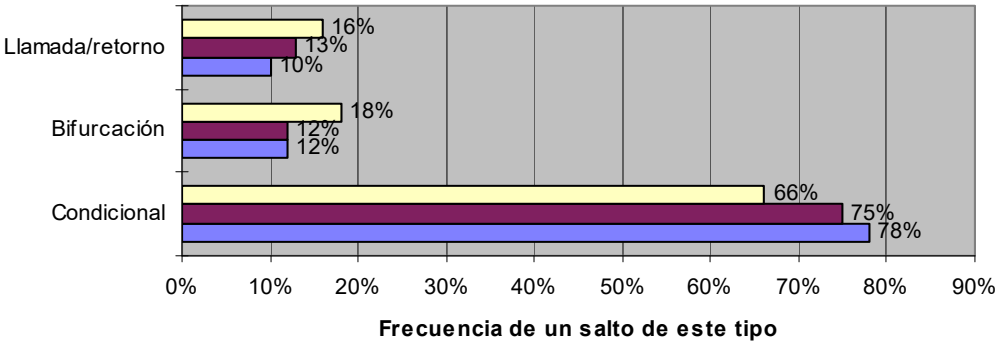
ordenación	instrucción x86	% total ejecutadas
1	carga	22%
2	salto condicional	20%
3	comparación	16%
4	almacenamiento	12%
5	suma	8%
6	and	6%
7	resta	5%
8	transferencia RR	4%
9	salto a subrutina	1%
10	retorno de subrutina	1%
	TOTAL	**96%**

Conclusión

El repertorio ISA de un procesador eficiente no debera incluir muchas más operaciones que las aparecidas en la tabla anterior.

8. Influencias en el rendimiento de las alternativas de diseño

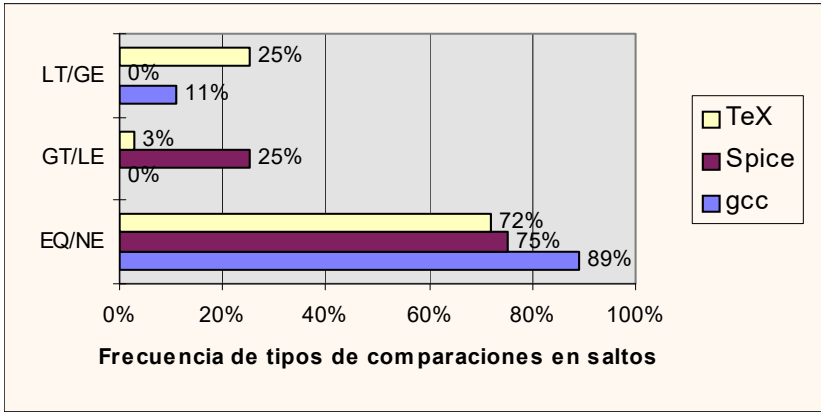
Sentencias de salto condicional



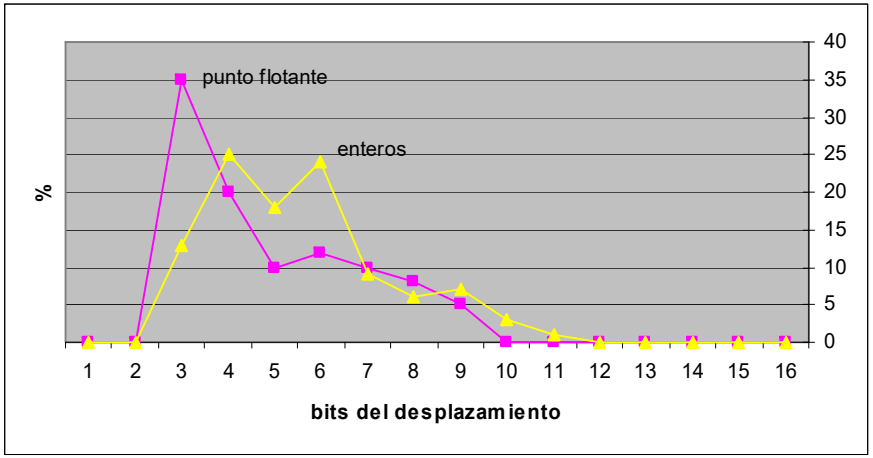
- Aparición de los tres tipos de sentencias.
- Bifurcaciones condicionales >= 75%
- Importante la eficiencia generación/salto .

Distribución de los saltos condicionales

- Saltos sobre igual o diferente >= 70%
- Gran número son comparaciones con cero



Distribución del desplazamiento (número de bits)



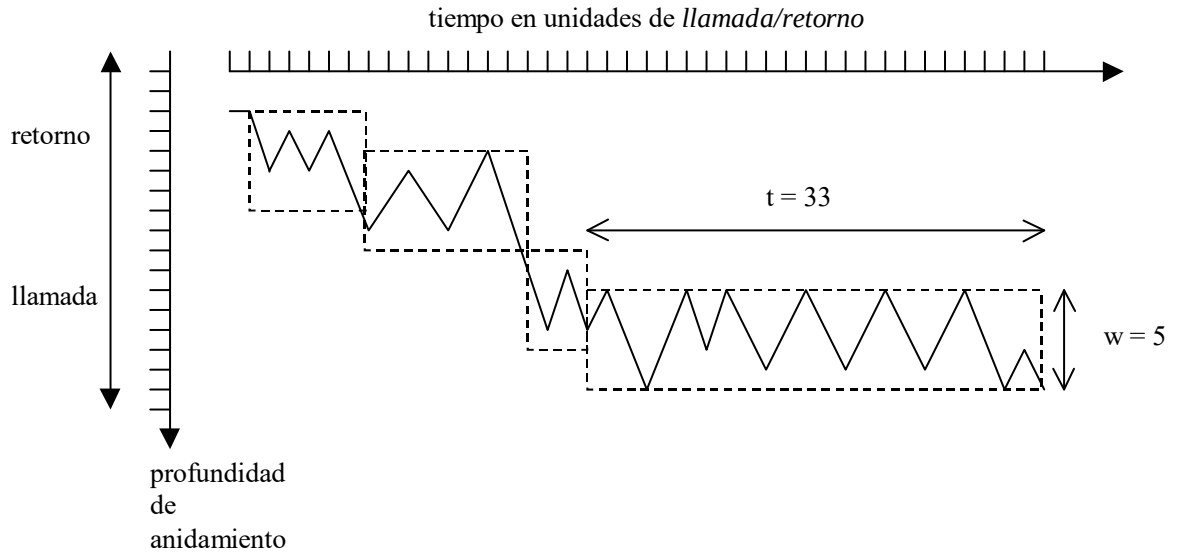
Conclusión

- Instrucciones que integren el test y el salto
- Registro con contenido inalterable e igual a cero.
- Desplazamiento de 8 bits

8. Influencias en el rendimiento de las alternativas de diseño

Llamadas a procedimientos (subrutinas)

- Consumo de tiempo elevado debido a la gestión de los registros de activación (RA)
- Del análisis de los datos se deducen los siguientes hechos:
  - Más del 98% de las llamadas utilizan menos de 6 parámetros
  - Más del 92% de las llamadas utilizan menos de 6 variables locales
  - La variación del nivel de anidamiento en la que se mueve un programa se mantiene relativamente pequeña a lo largo de una secuencia de llamadas/retorno



Conclusión
Conveniencia de utilizar mecanismos específicos para la gestión de los RA.(ventana de registros)

9. Procesadores RISC y CISC

RISC: Reduced-Instruction Set Computer

- Término acuñado por Patterson al principio de los 80
- Primeros diseños:
  - Berkeley RISC-I (Patterson)
  - Stanford MIPS (Hennessy)
  - IBM 801 (Cocke)
- Manifiesto RISC: "Crear ISAs que …"
  - Simplifiquen el diseño de los procesadores
  - Faciliten la optimización del compilador
  - Repertorio de instrucciones relativamente simple
  - Modos de direccionamiento asociados a la instrucción
  - Formatos de instrucción reducidos
  - Hacer rápido lo más usado
- **Ejemplos: PowerPC, ARM, SPARC, Alpha, PA-RISC**

CISC: Complex-Instruction Set Computer

- El término no existía hasta que apareció "RISC"
- **Ejemplos: x86, VAX, Motorola 68000, IBM 360/ 370...**

9. Procesadores RISC y CISC

Características comparativas

CISC(Complex Instruction Set Computers)	RISC(Reduced Instruction Set Computers)
Arquitectura RM y MM	Arquitectura RR (carga/almacenamiento)
Diseñadas sin tener en cuenta la verdadera demanda de los programas	Diseñadas a partir de las mediciones practicadas en los programas a partir de los 80
Objetivo: dar soporte arquitectónico a las funciones requeridas por los LANs	Objetivo: dar soporte eficiente a los casos frecuentes
Muchas operaciones básicas y tipos de direccionamiento complejos	Pocas operaciones básicas y tipos de direccionamiento simples
Instrucciones largas y complejas con formatos muy diversos ==> decodificación compleja (lenta)	Instrucciones de formato simple (tamaño filo) ==> decodificación simple (rápida)
Pocas instrucciones por programa ==> elevado número de ciclos por instrucción (CPI)	Muchas instrucciones por programa ==> reducido número de ciclos por instrucción (CPI)
Muchos tipos de datos ==> interfaz con memoria compleja	Sólo los tipos de datos básicos ==> interfaz con memoria sencilla
Número limitado de registros de propósito general ==> mucho almacenamiento temporal en memoria	Número elevado de registros ==> uso eficiente por el compilador para almacenamiento temporal
Baja ortogonalidad en las instrucciones ==> muchas excepciones para el compilador	Alto grado de ortogonalidad en las instrucciones ==> mucha regularidad para el compilador

9. Procesadores RISC y CISC

CISC/RISC: Ventajas e inconvenientes

- Ecuación de rendimiento:

$$T_{CPU} = N * CPI * t$$

- **CISC**
  - **Reduce N** usando instrucciones complejas, de tamaño variable
  - **Aumenta CPI**
  - **Aumenta t**
    - Instrucciones más complejas tienden utilizar mayor tiempo de ciclo
- **RISC**
  - **Reduce CPI**
    - Instrucciones más sencillas permiten reducir el tiempo de ciclo
  - **Reduce t**
  - **Aumenta N**
    - Nota: Los compiladores tienen una labor más sencilla ⇒ Mejor optimización de código

9. Procesadores RISC y CISC

Ejemplo: la arquitectura MIPS

MIPS: Arquitectura tipo load-store sencilla de 64 bits

- Primer procesador MIPS en 1985
- Describimos un subconjunto llamado MIPS64
  - Un repertorio de instrucciones simple tipo load-store
  - Diseñado para obtener una segmentación eficiente
  - Facilidad de optimización por el compilador

9. Procesadores RISC y CISC

Banco de registros

- **32 registros / 64 bits de propósito general** (GPRs)
  - R0, R1, …, R31, el valor de R0 es siempre 0
  - También llamados registros "enteros"
- **32 registros / 64 bits para aritmética flotante** (FPRs)
  - F0, F1, …, F31
  - Simple precisión (32 bits) o doble precisión (64 bits)
  - Soporte para operaciones de simple y doble precisión

Memoria

- **Memoria direccionable por bytes con dirección de 64 bits**
  - Tiene un bit de modo para seleccionar el alineamiento (little o big endian)
  - Las palabras en memoria están alineadas

9. Procesadores RISC y CISC

Tipos de datos

- **8-, 16-, 32- y 64-bits para aritmética entera**
- **32- y 64-bits para coma flotante** (IEEE 754)
- Las operaciones en MIPS64 operan con enteros de 64 bits o en coma flotante de 32 y 64 bits
  - Operandos enteros de 8-, 16- y 32-bits se rellenan con ceros o con el bit de signo para completar los 64 bits y operar

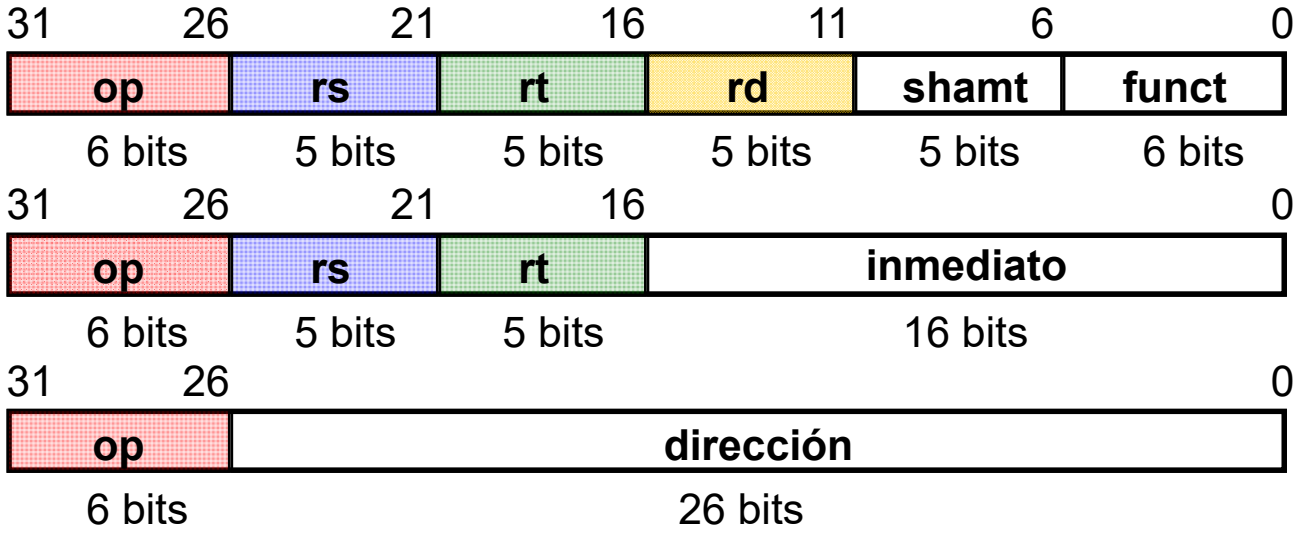
9. Procesadores RISC y CISC

Modos de direccionamiento

- **Sólo tres MDs**
  - **De registro** (campo de 5 bits indicando el registro)
  - **Inmediato** (campo de 16 bits en la instrucción)
  - **Con desplazamiento** (desplazamiento de 16 bits en instrucción)
    - MD indirecto de registro se implementa escribiendo “0” en el campo desplazamiento
    - MD absoluto se implementa usando el registro R0 e indicando la dirección absoluta en el campo desplazamiento

9. Procesadores RISC y CISC

Formatos de instrucción:



- Pocos modos de direccionamiento ⇒ van codificados en el OPCODE
- Instrucciones de 32 bits con un OPCODE de 6 bits

9. Procesadores RISC y CISC

Operaciones

- Cuatro grandes grupos
  - **Load/store**
  - **Operaciones en ALU**
  - **Saltos/bifurcaciones**
  - **Coma flotante**

9. Procesadores RISC y CISC

Operaciones Load/store del MIPS64

- Todos los registros pueden ser utilizados en load/store, salvo el R0 (load sin efecto alguno)

Example instruction	Instruction name	Meaning
LD R1,30(R2)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[30+Regs[R2]]
LD R1,1000(R0)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[1000+0]
LW R1,60(R2)	Load word	Regs[R1] $\leftarrow_{64}$ (Mem[60+Regs[R2]]$_0$)^{32} ## Mem[60+Regs[R2]]
LB R1,40(R3)	Load byte	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{56} ## Mem[40+Regs[R3]]
LBU R1,40(R3)	Load byte unsigned	Regs[R1] $\leftarrow_{64}$ 0^{56} ## Mem[40+Regs[R3]]
LH R1,40(R3)	Load half word	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{48} ## Mem[40+Regs[R3]] ## Mem[41+Regs[R3]]
L.S F0,50(R3)	Load FP single	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D F0,50(R2)	Load FP double	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R2]]
SD R3,500(R4)	Store double word	Mem[500+Regs[R4]] $\leftarrow_{64}$ Regs[R3]
SW R3,500(R4)	Store word	Mem[500+Regs[R4]] $\leftarrow_{32}$ Regs[R3]$_{32..63}$
S.S F0,40(R3)	Store FP single	Mem[40+Regs[R3]] $\leftarrow_{32}$ Regs[F0]$_{0..31}$
S.D F0,40(R3)	Store FP double	Mem[40+Regs[R3]] $\leftarrow_{64}$ Regs[F0]
SH R3,502(R2)	Store half	Mem[502+Regs[R2]] $\leftarrow_{16}$ Regs[R3]$_{48..63}$
SB R2,41(R3)	Store byte	Mem[41+Regs[R3]] $\leftarrow_{8}$ Regs[R2]$_{56..63}$

9. Procesadores RISC y CISC

Ejemplo: la arquitectura MIPS

- Operaciones en ALU
  - Todas son instrucciones registro-registro
  - El registro R0 se usa “como comodín” para implementar algunas instrucciones a partir de otras: Contiene “cero”
    - Ej: Para implementar MOVER R1 a R2 podemos usar: ADD R2, R1, R0

- Instrucciones en coma flotante
  - Los datos en coma flotante / simple precisión ocupan medio registro
  - El formato usado es el IEEE 754
  - Trabajan con los registros de coma flotante
  - Indican cuándo operar en simple o doble precisión
    - Ej.: MOV.S (simple) MOV.D (doble)
  - Para mejorar el rendimiento de rutinas gráficas, MIPS64 implementa instrucciones para realizar dos operaciones simultáneas en precisión simple
    - Cada una en una parte de los registros en coma flotante de 64 bits

9. Procesadores RISC y CISC

Instrucciones de salto y salto condicional

- La condición de salto se indica en la propia instrucción

Example instruction		Instruction name	Meaning
J	name	Jump	$PC_{36..63} \leftarrow$ name
JAL	name	Jump and link	Regs[R31]←PC+8; $PC_{36..63} \leftarrow$ name; ((PC + 4)–2^{27}) ≤ name < ((PC+4)+2^{27})
JALR	R2	Jump and link register	Regs[R31]←PC+8; PC←Regs[R2]
JR	R3	Jump register	PC←Regs[R3]
BEQZ	R4,name	Branch equal zero	if (Regs[R4]==0) PC←name; ((PC+4)–2^{17}) ≤ name < ((PC+4)+2^{17})
BNE	R3,R4,name	Branch not equal zero	if (Regs[R3]!= Regs[R4]) PC←name; ((PC+4)–2^{17}) ≤ name < ((PC+4)+2^{17})
MOVZ	R1,R2,R3	Conditional move if zero	if (Regs[R3]==0) Regs[R1]←Regs[R2]