Estructura de Computadores

Tema 2

Instrucciones y direccionamientos

Índice

- Lenguaje máquina
  - Instrucción máquina
  - Tipos de datos
  - Juego de instrucciones
  - Formato de las instrucciones
- Modos de direccionamiento
- Tipos de instrucciones
- Computadores CISC y RISC

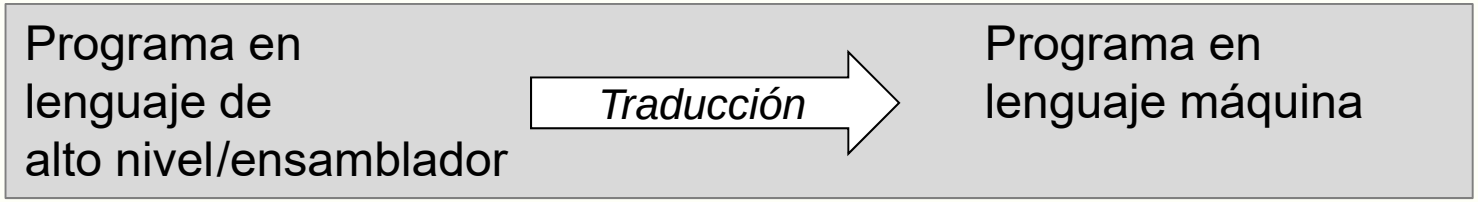
Bibliografía

- De Miguel, P. *"Fundamentos de los computadores"*, Paraninfo, 2004. 9ª edición (capítulo 6).
- Stallings, W. *"Organización y arquitectura de computadores".* Prentice Hall, 2010. 8ª edición (capítulo 10).
- García Clemente y otros. *"Estructura de computadores: problemas resueltos".* RAMA-2006 (capítulo 1)

Introducción

- ## Lenguaje máquina

  Es el que es capaz de interpretar y ejecutar directamente el computador



- ## Características del lenguaje máquina

  - Basado en sistema de representación binario: Las instrucciones se representan en el computador como cadenas de ceros y unos
  - Particular de cada procesador

Lenguaje máquina: Ejemplos

```
int sumar (int a, int b) {
   int sum, aux1, aux2;
   aux1 = a << 2;
   aux2 = b * 3;
   sum = aux1 + aux2;
   return sum;
}
```

```
 0:  e52db004    push        {fp}
 4:  e28db000    add         fp, sp, #0
 8:  e24dd01c    sub         sp, sp, #28
 c:  e50b0018    str         r0, [fp, #-24]
10:  e50b101c    str         r1, [fp, #-28]
14:  e51b3018    ldr         r3, [fp, #-24]
18:  e1a03103    lsl         r3, r3, #2
1c:  e50b3008    str         r3, [fp, #-8]
20:  e51b201c    ldr         r2, [fp, #-28]
24:  e1a03002    mov         r3, r2
28:  e1a03083    lsl         r3, r3, #1
2c:  e0833002    add         r3, r3, r2
30:  e50b300c    str         r3, [fp, #-12]
34:  e51b2008    ldr         r2, [fp, #-8]
38:  e51b300c    ldr         r3, [fp, #-12]
3c:  e0823003    add         r3, r2, r3
40:  e50b3010    str         r3, [fp, #-16]
44:  e51b3010    ldr         r3, [fp, #-16]
48:  e1a00003    mov         r0, r3
4c:  e28bd000    add         sp, fp, #0
50:  e8bd0800    pop         {fp}
54:  e12fff1e    bx lr
```

```
40051d:   55           push   %rbp
40051e:   48 89 e5     mov    %rsp,%rbp
400521:   89 7d ec     mov    %edi,-0x14(%rbp)
400524:   89 75 e8     mov    %esi,-0x18(%rbp)
400527:   8b 45 ec     mov    -0x14(%rbp),%eax
40052a:   c1 e0 02     shl    $0x2,%eax
40052d:   89 45 fc     mov    %eax,-0x4(%rbp)
400530:   8b 55 e8     mov    -0x18(%rbp),%edx
400533:   89 d0        mov    %edx,%eax
400535:   01 c0        add    %eax,%eax
400537:   01 d0        add    %edx,%eax
400539:   89 45 f8     mov    %eax,-0x8(%rbp)
40053c:   8b 45 f8     mov    -0x8(%rbp),%eax
40053f:   8b 55 fc     mov    -0x4(%rbp),%edx
400542:   01 d0        add    %edx,%eax
400544:   89 45 f4     mov    %eax,-0xc(%rbp)
400547:   8b 45 f4     mov    -0xc(%rbp),%eax
40054a:   5d           pop    %rbp
40054b:   c3           retq
```

Lenguaje máquina

Lenguaje ensamblador

Instrucciones máquina

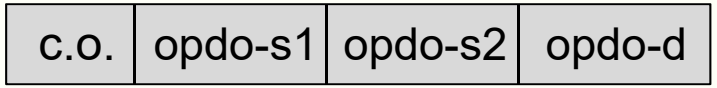
- Propiedades
  - Realizan una única y sencilla función (en general)
  - Operan sobre un número fijo de operandos con una representación determinada
  - Son autocontenidas: Contienen toda la información necesaria para su ejecución:
    - Operación a realizar
    - Operandos o dónde se encuentran
    - Lugar donde dejar el resultado
    - Ubicación de la siguiente instrucción a ejecutar

Instrucciones máquina

Información que contienen:

- Operación a realizar (código de operación)
- Operandos o dónde se encuentran
  - En registros del procesador
  - En memoria
  - En la propia instrucción
- Lugar donde dejar el resultado
  - En registros del procesador
  - En memoria
- Ubicación de la siguiente instrucción a ejecutar
  - De forma implícita: la siguiente instrucción
  - De forma explícita: en las instrucciones de salto

Ejemplo genérico de codificación:



Ejemplos particulares:

```
e0823003    add r3, r2, r3
01 d0       add %edx,%eax
```

Juego de Instrucciones

Conjunto de instrucciones máquina que es capaz de entender y ejecutar la CPU

- El juego de instrucciones de una CPU viene definido por:
  - Las **operaciones** que realiza
    Qué es capaz de hacer el procesador
  - Los **tipos de datos** que maneja, así como su representación
    Sobre qué tipos de información
  - Los **modos de direccionamiento**
    Cómo se especifica el lugar donde están los operandos
  - El **formato** o formatos de las instrucciones
    Cómo se representan/codifican las instrucciones

Todo ello afecta a velocidad de ejecución

Tipos de instrucciones

- Operaciones básicas:
  - Transferencia de datos (*ld, st, move*)
  - Procesamiento (*add, sub, .., and, cmp, …*)
  - De control del flujo de ejecución (*br*, *bz*, *call*, *ret*, *...*)

Tipos y tamaño de los datos

- Tipos de datos básicos:
  - Direcciones
  - Números (enteros con/sin signo, coma flotante)
  - Caracteres (cadenas de caracteres)
  - Datos lógicos
- Tamaños:
  - Palabra (tamaño privilegiado del computador)
  - Doble palabra
  - Media palabra
  - Byte (8 bits)

Direccionamiento de la memoria

- A nivel de palabra: direcciones de memoria consecutivas corresponden a palabras consecutivas
- **A nivel de byte**: direcciones de memoria consecutivas corresponden a bytes consecutivos
  - Dos palabras consecutivas están separadas por el tamaño en bytes de la palabra
  - **Ordenación de los bytes de una palabra en memoria**:
    - Little endian: Byte menos significativo de una palabra en la dirección menor
    - Big endian: Byte menos significativo de una palabra en la dirección mayor

Direccionamiento de la memoria

- A nivel de palabra: direcciones de memoria consecutivas corresponden a palabras consecutivas
- **A nivel de byte**: direcciones de memoria consecutivas corresponden a bytes consecutivos
  - Dos palabras consecutivas están separadas por el tamaño en bytes de la palabra
  - **Ordenación de los bytes de una palabra** en memoria:
    - Little endian: Byte menos significativo de una palabra en la dirección menor
    - Big endian: Byte menos significativo de una palabra en la dirección mayor

---

Ejemplo: número decimal 70.960.543 (0x043AC59F) almacenado en 32 bits en la dirección 184

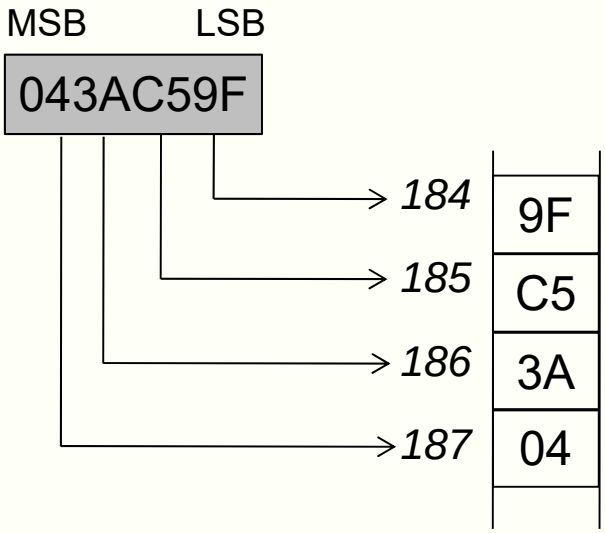
Be:

184	185	186	187
04	3A	C5	9F

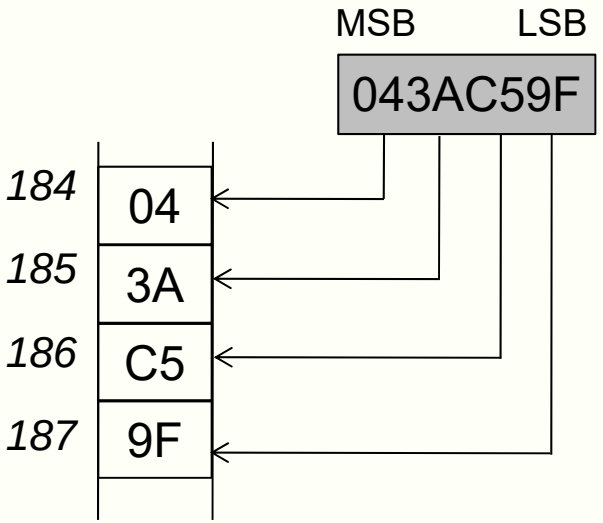
Le:

184	185	186	187
9F	C5	3A	04

Ejemplo Little endian vs Big endian



Little Endian



Big Endian

Direccionamiento de la memoria

- Alineamiento de los datos:

  Un dato que ocupa K bytes está alineado en memoria
  si está almacenado en una dirección múltiplo de K

Ejemplos:

a) Dato de 32 bits: 0x10203040 almacenado en la dirección 185 (NA)
b) Dato de 16 bits: 0x2030 almacenado en la dirección 186 (A)

184		10	20	30
188	40			

- Muchas arquitecturas sólo permiten accesos a datos alineados
- Algunas permiten el acceso a datos no alineados:
  - Los accesos a estos datos son más lentos
  - El acceso a un dato no alineado implica varios accesos a memoria

Modos de direccionamiento

- *Cómo se especifican los operandos de una instrucción* (dato, resultado o instrucción)

  ↪ Procedimiento empleado por la CPU para acceder a ellos

- Definiciones:
  - **Objeto** del direccionamiento: Instrucción, dato o resultado que se desea direccionar. Puede estar en:
    - La propia instrucción (datos)
    - Un registro (datos o resultados)
    - Memoria (datos, resultados e instrucciones)
  - **Dirección** del objeto: Identifica el lugar en el que reside (identificador que especifica el registro o posición de memoria donde se encuentra)

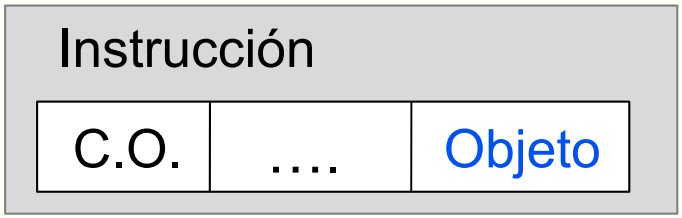
Modos de direccionamiento

- Inmediato . . . . . #valor
- Directo
  - Absoluto
    - a registro . . . . . .registro
    - a memoria . . . . . /direccion
  - Relativo
    - a registro base . . . #desp[.registro_base]
      [.registro_base, #desp]
    - a registro índice
    - a PC . . . . . $desp
- Indirecto . . . . . [ ]
- Implícito

Notación Estándar IEEE 694

Direccionamiento Inmediato

El Objeto está contenido en la propia instrucción



Ejemplos:

```
ADD .R1,#4 ; R1 ← R1+4
LD .R2, #1 ; R2 ← 1
```

- Útil para constantes

✓ Rápido: No requiere accesos adicionales a memoria

✗ Operando de rango limitado

Direccionamiento Inmediato

Posibilidad de indicar el sistema de representación mediante prefijos. En el estándar IEEE 694:

B´valor en binario
H´valor en hexadecimal
D´valor en decimal

Ejemplos:

```
ADD .R1,#4 ; R1 ← R1+4
LD .R2, #1 ; R2 ← 1
```

- Útil para constantes
- ✓ Rápido: No requiere accesos adicionales a memoria
- ✗ Operando de rango limitado

Direccionamiento Directo

En la instrucción está especificada la Dirección donde está almacenado el Objeto

- Absoluto

  La instrucción contiene la dirección completa del Objeto, que puede estar en un registro o en memoria.

- Relativo

  La instrucción contiene un desplazamiento sobre una dirección, que suele estar almacenada en un registro

  La Dirección del Objeto se calcula sumando ambos

  El Objeto está en memoria

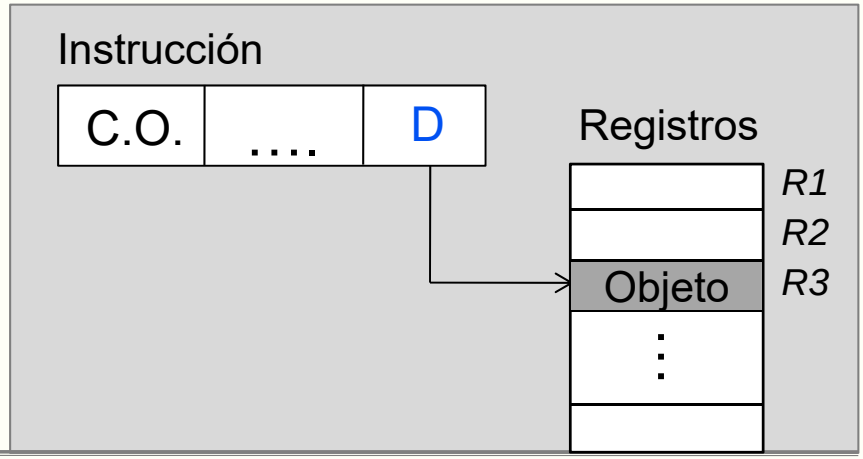
Direccionamiento Directo Absoluto

- **A registro**

  El Objeto está contenido en un registro. La instrucción contiene el *identificador* del registro donde se encuentra

Ejemplo:

```
ADD  .R4, .R3  ;  R4←R4+R3
```

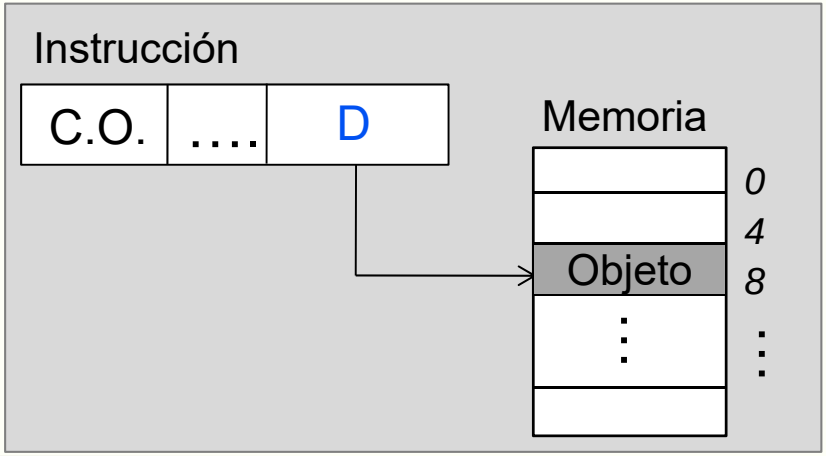


- Útil para variables utilizadas muy frecuentemente
- ✓ El acceso al operando es rápido al estar en un registro
- ✗ Número limitado de registros
- ✓ Se necesitan pocos bits para especificar la Dirección

Direccionamiento Directo Absoluto

- **A memoria**

  El Objeto está contenido en memoria. La instrucción contiene la *dirección* completa de memoria donde se encuentra



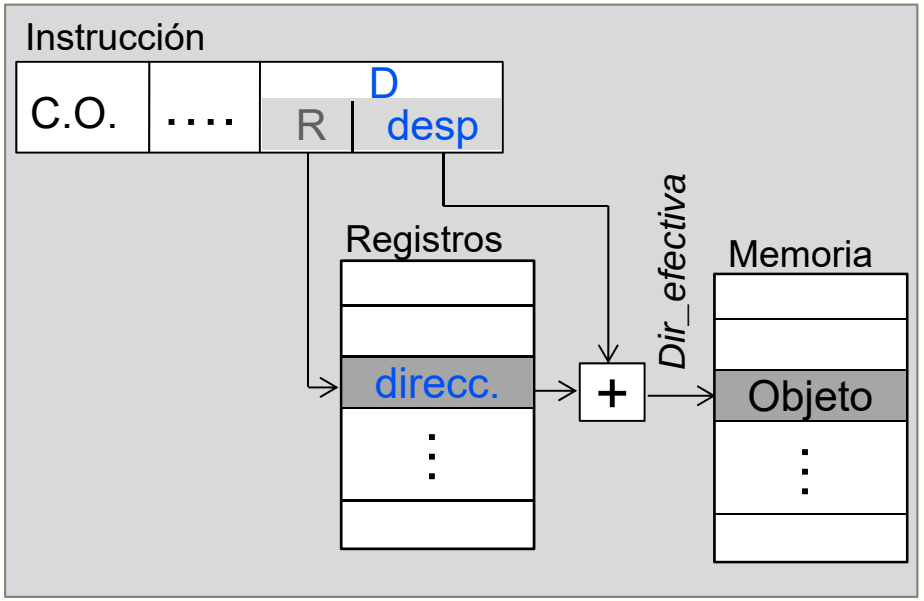
Ejemplo:

```
LD  .R4,/100  ;R4←Mem(100)
BR  /1000
```

- ✗ Número de bits necesarios para especificar la Dirección
- ✓ Acceso a un gran espacio de direcciones
- ✗ El acceso al operando requiere un acceso a memoria (más lento)
- ✗ El código no es reubicable (depende de su ubicación en Mp)

Direccionamiento Relativo

- El Objeto está en memoria. La instrucción contiene la Dirección, especificada en dos "partes":
  - Desplazamiento:
    Valor entero con signo
  - Registro:
    Registro general o de propósito específico que contiene una dirección de memoria



- La dirección efectiva donde se encuentra el Objeto se calcula como: Dir_efectiva = Registro + Desplazamiento

Direccionamiento Relativo a Registro Base

- El registro base se carga con la dirección de memoria donde están almacenados un conjunto de datos
- El desplazamiento indica la posición relativa, respecto a la dirección de comienzo, del dato al que se accede
- El registro base no se modifica

Ejemplo:

```
LD .R1, #4[.R7]   ; R1 ← Mem(R7+4)
LD .R1, [#4,.R7]
```

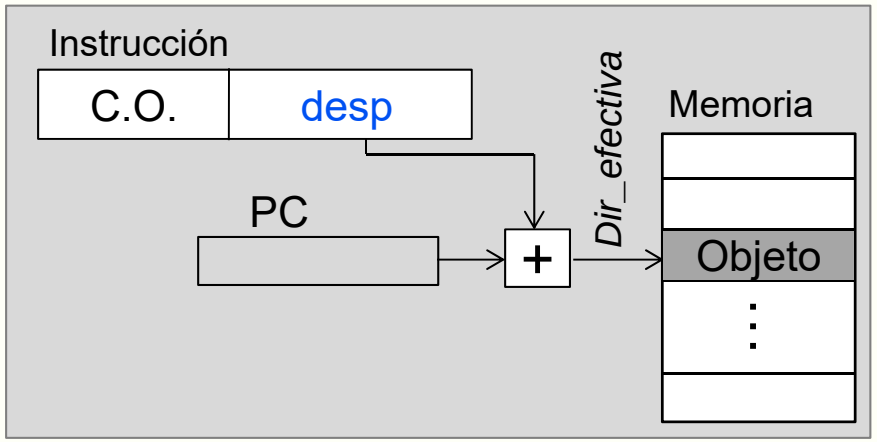
- ✗ El rango de direcciones al que se puede acceder está limitado por el tamaño del desplazamiento
- ✓ Acceso a estructuras de datos y a parámetros en pila
- ✓ Código reubicable (no depende de su ubicación en Mp)

Direccionamiento Relativo a PC

- El registro base es el PC
- Se utiliza en las instrucciones de salto: el Objeto de este direccionamiento es una instrucción

Ejemplo:

```
BR $10   ;PC ← PC+10
```



- ✓ Permite alcanzar instrucciones "cercanas" a la que se está ejecutando: ejecución de saltos "cortos", p.e. bucles, *if then else* ..
- ✓ El código es reubicable (no depende de su ubicación en Mp)

Direccionamiento Relativo a Registro Índice

- Es un direccionamiento relativo en el que el registro base se modifica
  - Preincremento
  - Predecremento
  - Postincremento
  - Postdecremento
- El tamaño del incremento/decremento es igual al tamaño del objeto direccionado

Ejemplos:

```
LD  .R1,#8[++.R7]  ; R7 ← R7+4 y R1 ← Mem(R7+8)
ST  .R1,#8[--.R7]  ; R7 ← R7-4 y Mem(R7+8) ← R1
LD  .R1,#8[.R7++]  ; R1 ← Mem(R7+8) y R7 ← R7+4
SUB .R1,#8[.R7--]  ; R1 ← R1-Mem(R7+8) y R7 ← R7-4
```

✓ Útil para el recorrido de vectores y matrices

Ejercicios

1. Qué hacen las siguientes instrucciones y qué registros y posiciones de memoria se modifican.

```
ADD .R1, #4[.R2], #8[.R3]
ADD #0[.R1], #4[.R2], #8[.R3]
ADD #0[--.R1], #4[.R2++], #8[--.R3]
```

   Valores iniciales de registros: R1 = 100, R2 = 200, R3 = 400

2. Secuencia de instrucciones para sumar los elementos de un array almacenado a partir de la dirección contenida en R1 dejando el resultado en R2

   Inicialmente R2 = 0 y R3 = Nº de elementos

   Instrucciones a utilizar: `ADD`, `SUB` y `BRNZ` de 4Bytes

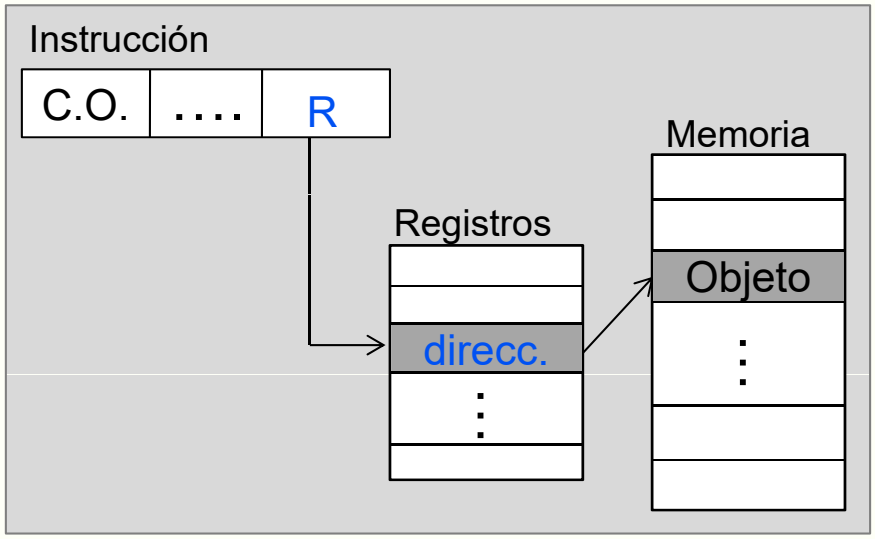
   Direccionamientos: Los vistos hasta ahora

Direccionamiento Indirecto

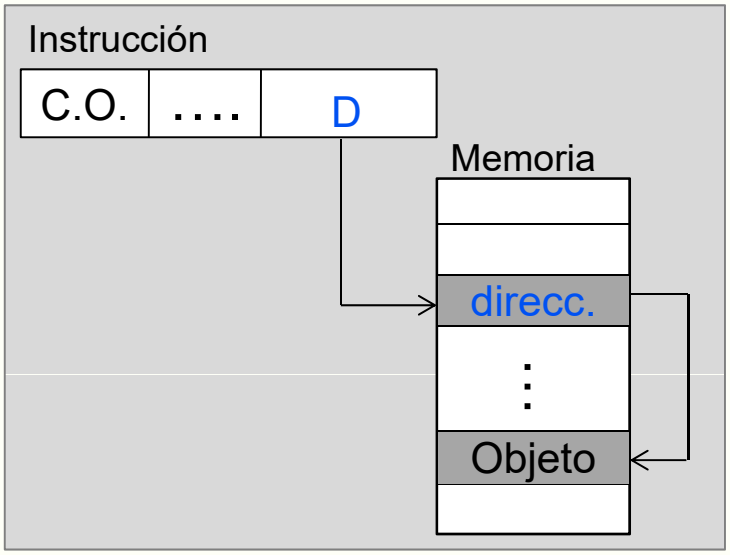
En la instrucción está especificada la dirección donde está almacenada la Dirección en la que se almacena el Objeto.

- Comienza por un direccionamiento Directo, pero la dirección obtenida no apunta al Objeto sino a su Dirección

Ejemplos:



(a) Indirecto a registro



(b) Indirecto a memoria

Direccionamiento Indirecto: Ejemplos

- A registro

```
LD .R1,[.R4]  ; R1 ← Mem(R4)
BR [.R1]      ; PC ← R1
```

- A memoria

```
LD .R1, [/1000] ; R1 ← Mem(Mem(1000))
```

- A registro base

```
LD .R1, [#8[.R4]] ; R1 ← Mem(Mem(R4+8))
```

- A registro índice

```
LD .R1, [#8[.R2++]] ; R1 ← Mem(Mem(R2+8)), R2 ← R2+4
```

✓ Paso de parámetros por referencia

✓ Uso de punteros a variables

Direccionamiento Implícito

La instrucción no contiene ni la Dirección ni el Objeto del direccionamiento

- El Operando se supone ubicado en algún lugar específico de la máquina, por ejemplo:
  - La pila

```
PUSH .R1  (operando destino implícito)
POP .R2   (operando fuente implícito)
```

  - El registro Acumulador (en máquinas de acumulador)

```
ADDA .R1 ; AC ← AC+R1
```

  - El PC

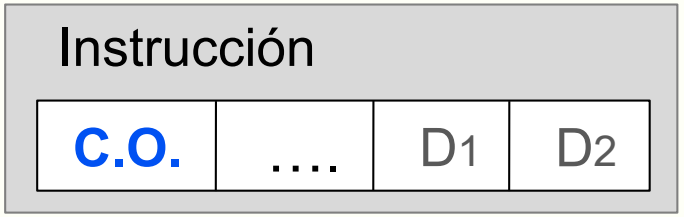
```
BR $-12   ; PC ← PC-12
```

Modos de direccionamiento más utilizados

	Estándar	Intel	Arm	MC88110
Inmediato	`#n`	`$n`	`#n`	`n`
Absoluto a registro	`.R`	`%r`	`r`	`r`
Relativo a Registro	`#n[.R]` `[.R, #n]`	`#n(%r)`	`[r, #n]`	`r, n`

Juego de instrucciones

- **Tipos de Instrucciones**
  - Transferencia
  - Aritméticas y de Comparación
  - Lógicas
  - De bit
  - Desplazamiento y rotación
  - De salto o bifurcación
  - De Entrada/Salida y misceláneas



Transferencia de datos

- Mueven información entre registros, registros y posiciones de memoria o entre posiciones de memoria
- No se modifican los biestables de estado

LD, ST y MOVE

```
LD .R2, #4[.R4]        ; R2 ← MEM(R4+4)
ST .R2, #4[.R4]        ; MEM(R4+4)← R2
MOVE .R2,.R4           ; R4 ← R2
MOVE [.R2],[.R4]       ; MEM(R4)← MEM(R2)
```

PUSH y POP

```
PUSH .R1       ; MEM(SP)← R1; SP ← SP-4
POP  .R1       ; SP ← SP + 4; R1 ← MEM(SP)
```

Transferencia de datos

Posibilidad de indicar, en algunas, el tamaño de la información mediante sufijos y si es con signo o sin el.
En el estándar IEEE 694:

.B	8 bits	**-U**	sin signo
.S	16 bits	**-S**	con signo
.L	32 bits		
.Q	64 bits		

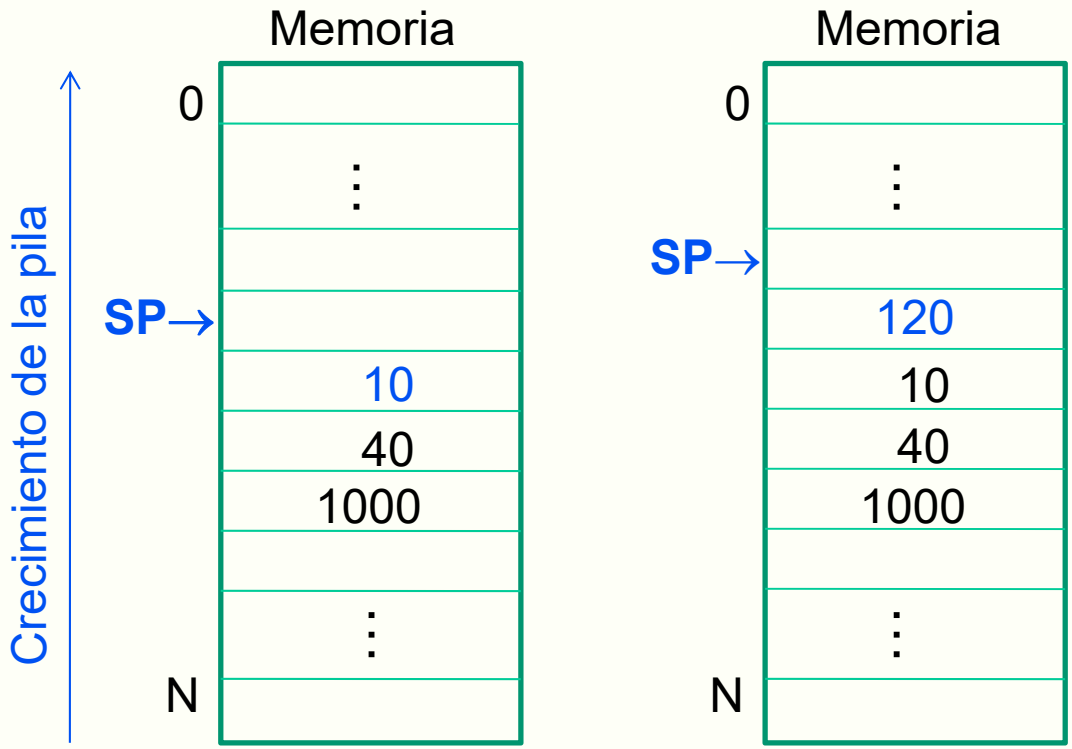
LD, ST y MOVE

```
LD .R2, #4[.R4]        ; R2 ← MEM(R4+4)
ST .R2, #4[.R4]        ; MEM(R4+4)← R2
MOVE .R2,.R4           ; R4 ← R2
MOVE [.R2],[.R4]       ; MEM(R4)← MEM(R2)
```

PUSH y POP

```
PUSH .R1       ; MEM(SP)← R1; SP ← SP-4
POP  .R1       ; SP ← SP + 4; R1 ← MEM(SP)
```

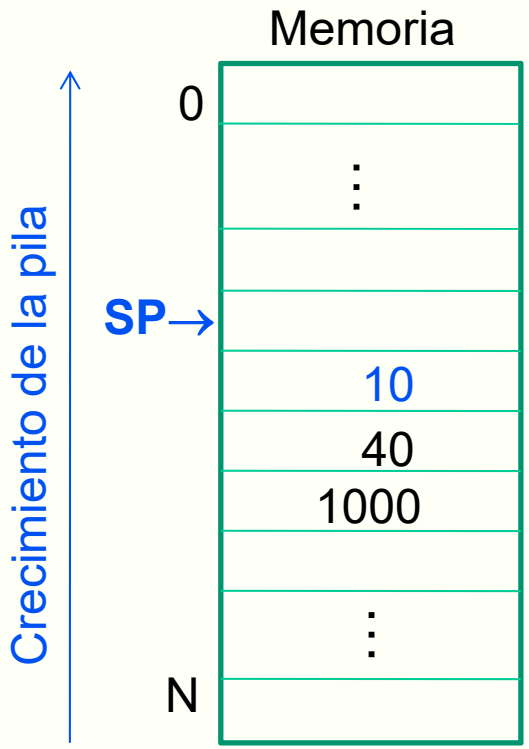
Operaciones PUSH y POP



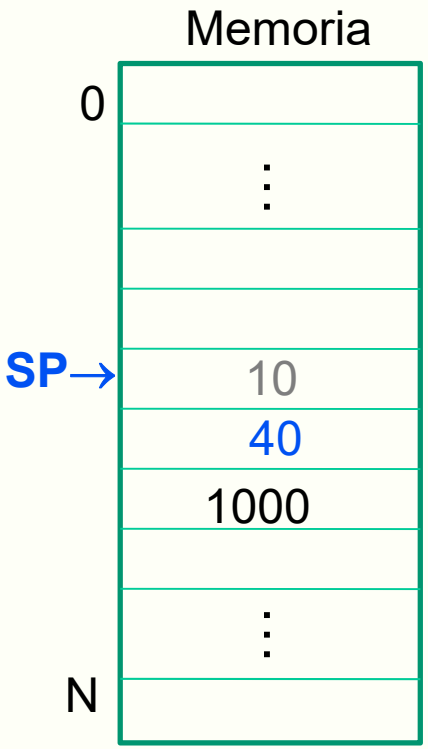
Situación inicial:
- SP apunta a la primera posición libre
- R1 contiene “120”

PUSH .R1
R1 no se modifica

Operaciones PUSH y POP



Situación inicial:
- SP apunta a la primera posición libre
- R1 contiene “120”



POP .R1
R1 contiene ahora “10”

Aritméticas y de Comparación

- Todas modifican los biestables de estado

```
ADD  .R1, .R2        ; R1 ← R1 + R2
SUB  .R1, .R2        ; R1 ← R1 – R2
MUL  .R1, .R2        ; R1 ← R1 * R2
DIV  .R1, .R2        ; R1 ← R1 / R2
ADDC .R1, .R2        ; R1 ← R1 + R2 + C
SUBC .R1, .R2        ; R1 ← R1 - R2 - C
CMP  .R1, .R2        ; R1 – R2 ; No modifica R1 ni R2
INC  .R1             ; R1 ← R1+1
DEC  .R2             ; R2 ← R2-1
```

Aritméticas y de Comparación

Posibilidad de indicar si los datos son con signo o sin el. En el estándar IEEE 694:

- **-U** sin signo
- **-S** con signo

```
ADD  .R1, .R2      ; R1 ← R1 + R2
SUB  .R1, .R2      ; R1 ← R1 – R2
MUL  .R1, .R2      ; R1 ← R1 * R2
DIV  .R1, .R2      ; R1 ← R1 / R2
ADDC .R1, .R2      ; R1 ← R1 + R2 + C
SUBC .R1, .R2      ; R1 ← R1 - R2 - C
CMP  .R1, .R2      ; R1 – R2 ; No modifica R1 ni R2
INC  .R1           ; R1 ← R1+1
DEC  .R2           ; R2 ← R2-1
```

Aritméticas y de Comparación

- La ubicación de los operandos depende del modelo de ejecución del procesador:
  - Registro-Registro

```
ADD .R1,.R2      ; R1 ← R1 + R2
SUB .R1,#4       ; R1 ← R1 - 4
```

  - Registro-Memoria

```
CMP .R1,[.R2++] ; R1 - Mem(R2) ; R2 ← R2+4
```

  - Memoria-Memoria

```
MUL [.R1],#8[.R2] ; Mem(R1) ← Mem(R1)* Mem(R2+8)
```

- Definen el número de direcciones del procesador

Número de direcciones: Ejemplos

- Tres direcciones

```
ADD .R3,.R1,.R2     ; R3 ← R1 + R2
```

- Dos direcciones

```
ADD .R1,.R2         ; R1 ← R1 + R2
```

- Una dirección (máquina de acumulador).

```
ADD .R1             ; AC ← AC + R1
```

- Cero direcciones (máquina de pila)

```
ADD
```

Lógicas

- Realizan la operación indicada, bit a bit
- Modifican los biestables de estado

```
AND .R1,.R2   ; R1 ← R1 AND R2
XOR .R1, #4   ; R1 ← R1 XOR 4
OR .R1,[.R2]  ; R1 ← R1 OR Mem(R2)
NOT #8[.R1]   ; R1 ← NOT Mem(R1+8)
```

- ✓ Útiles para trabajar con máscaras, p.e:

```
AND .R1, #1     ; Si biestable Z = 1 el número es par
AND .R1, #H'0000000F ; Se extrae el byte de menor peso
```

De bit

- Realizan operaciones sobre un bit

```
CLR.I #3,.R1    ; Pone a cero el bit 3 de R1
SET.I #3,.R1    ; Pone a uno el bit 3 de R1
TEST.I #3,.R1   ; biestable Z ← NOT(bit3(R1))
                    ; si bit3(R1)=0 biestable Z ← 1
                    ; si no biestable Z ← 0
```

De bit

- Realizan operaciones sobre un bit

```
CLR.I #3,.R1    ; Pone a cero el bit 3 de R1
SET.I #3,.R1    ; Pone a uno el bit 3 de R1
TEST.I #3,.R1   ; si bit3(R1)=0 biestable Z ← 1
                ; si no biestable Z ← 0
```

Ejercicio:

```
CLR.I #3,.R1
```

¿Utilizando la instrucción AND .R1, #inmediato?

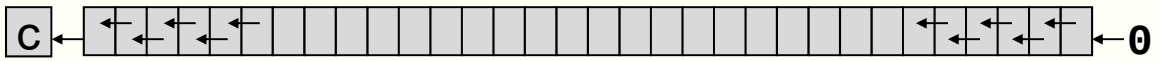
```
SET.I #3,.R1
```

¿Utilizando la instrucción OR .R1, #inmediato?

Desplazamiento

- Realizan desplazamientos de bits a izquierda/derecha (equivalen a multiplicar/dividir por 2)
- Tipos de desplazamiento:
  - **Lógico**

```
SHL  .R1
```

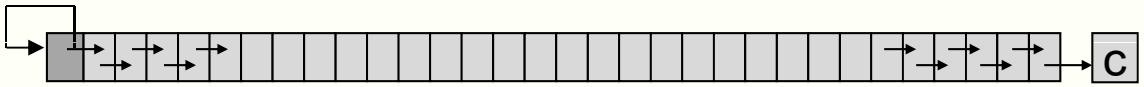



```
SHR  .R2
```

  - **Aritmético**: Para enteros con signo

```
SHLA  .R1
```

```
SHRA  .R2
```



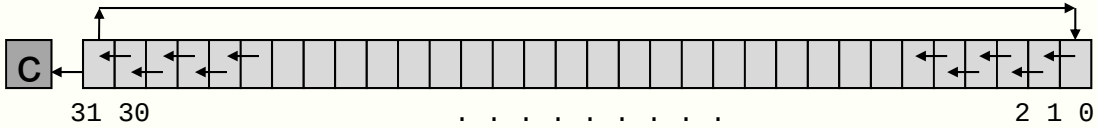
Comportamiento si se utiliza c.a 1 o c.a 2

- Pueden especificar el nº de posiciones a desplazar, p.e. `SHL .R1, #4`

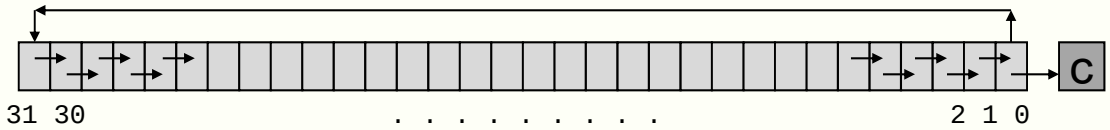
Rotación

- Realizan rotaciones de bits a izquierda/derecha incluyendo o no el biestable de Acarreo
  - Rotación

ROL .R1

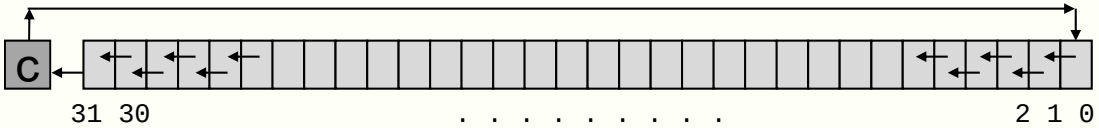


ROR .R1

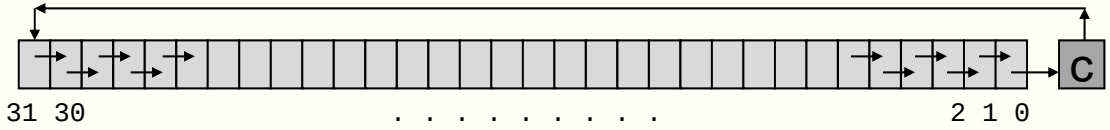


  - Rotación a través del biestable de Acarreo

ROLC .R1

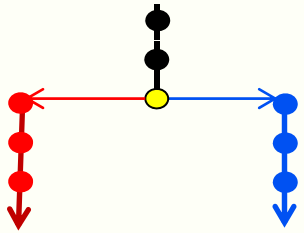


RORC .R1

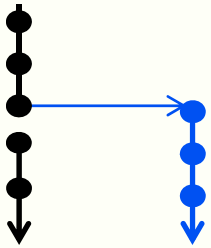


Salto o Bifurcación

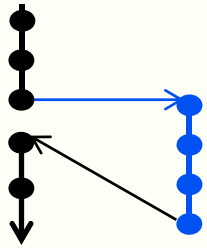
- Modifican la secuencia habitual de ejecución (secuencial)
- No modifican los biestables de estado
- Tipos:
  - Condicionales/Incondicionales
  - Con/Sin Retorno



```
if (cond)
        código-1
else
        código-2
```



goto etiqueta



```
x = sumar(a)
y = x*2
```

Saltos Incondicionales

- El salto se realiza siempre
- Tienen como operando la dirección de memoria donde está la siguiente instrucción a ejecutar

```
BR /1000      ; PC ← 1000
BR #4[.R3]    ; PC ← R3 + 4
BR $10        ; PC ← PC + 10
              ; ¡¡El PC apunta a la dirección de
              ; la siguiente instrucción!!
```

Saltos Condicionales

- El salto se realiza solo si se cumple la condición especificada en la propia instrucción:

  Bcc Direccion

  cc = Z, C, V, P, N, EQ, GT, GE, LT, LE, NZ, NC, NV, NEQ, …

```
BZ /1000      ; si biestable(Z)=1   PC ← 1000
              ; si no PC ← PC+N
BNC #4[.R3]   ; si biestable(C)=0   PC ← R3 + 4
              ; si no PC ← PC+N
BP $10        ; si biestable(S)=0 ← PC + 10
              ; si no PC ← PC+N
```

Saltos Con retorno

- Se utilizan para implementar saltos a subrutinas o llamadas a funciones.
- Una vez ejecutado el código de la subrutina/función se debe retornar a la instrucción siguiente al salto.

```
CALL direccion y RET
```

- Es necesario guardar la dirección de retorno:
  - En un registro de propósito general

```
CALL /1000    ; r1 ← PC , PC ← 1000
RET           ; PC ← R1
```

No permite llamadas anidadas

  - En la pila

Saltos Con retorno

- Guardando la dirección de retorno en la pila

```
CALL /1000    ; Mem(SP) ← PC, SP ← SP-4,
              ; PC ← 1000
RET           ; SP ← SP+4, PC ← Mem(SP)
```

Permite llamadas anidadas

- Si en la subrutina se trabaja con la pila, hay que tener en cuenta que la dirección de retorno ha quedado en la cima de la pila.

Formato de las instrucciones

- Modo en que se disponen los bits que identifican cada uno de los elementos que componen una instrucción: campos
  - Código de operación → Nº de operaciones distintas
  - Operando(s) → Número de direcciones explícitas y modos de direccionamiento
- Se suele emplear más de un formato

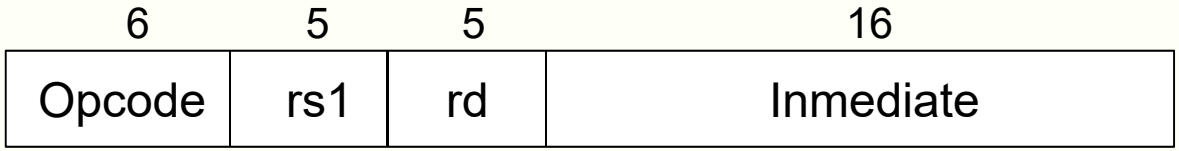
  Cada instrucción encaja en un formato predeterminado
- Aspectos de diseño:
  - Longitud fija o variable
  - Número de formatos distintos
  - Codificación regular

Impacto en la velocidad y sencillez de CPU

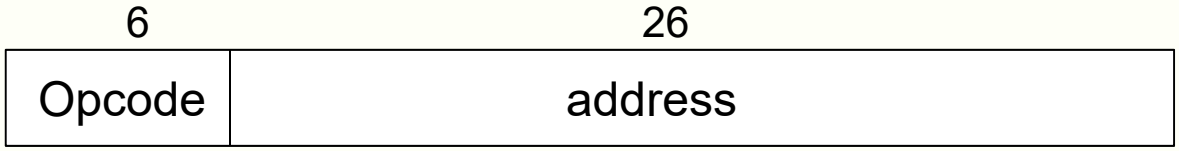
Formatos de instrucción: Ejemplo

DLX

R-Type



I-Type



J-Type

Computadores CISC y RISC

- Dos paradigmas de Arquitectura de un computador
- Responden a la evolución tecnológica en el diseño de procesadores
- CISC (*Complex Instruction Set Computer*)

  Observación:

  Alrededor del 20% de las instrucciones ocupa el 80% del t.ejecución de un programa

- RISC (*Reduced Instruction Set Computer*) - 1980

  Idea básica:

  Hacer más rápidos los casos más frecuentes

CISC vs RISC

- Modelo R-R, R-M y M-M
- Muchas instrucciones y complejas (↑t.ejecución)
- Modos de direccionamiento complejos
- Muchos formatos de instrucción
- Instrucciones de tamaño variable
- U.Control más compleja
- Menor nº de instrucciones por programa
- Ejs: ix86, Amd

- Modelo R-R (solo load y store hacen referencia a memoria)
- Instrucciones sencillas y ortogonales (↓t. ejecución)
- Modos de direccionamiento sencillos
- Pocos formatos de instrucción y codificación uniforme
- Instrucciones de tamaño fijo
- U. Control más sencilla
- Mayor nº de instrucciones por programa
- Ejs: Sparc, Arm, PowerPC

Formatos de instrucción de un CISC

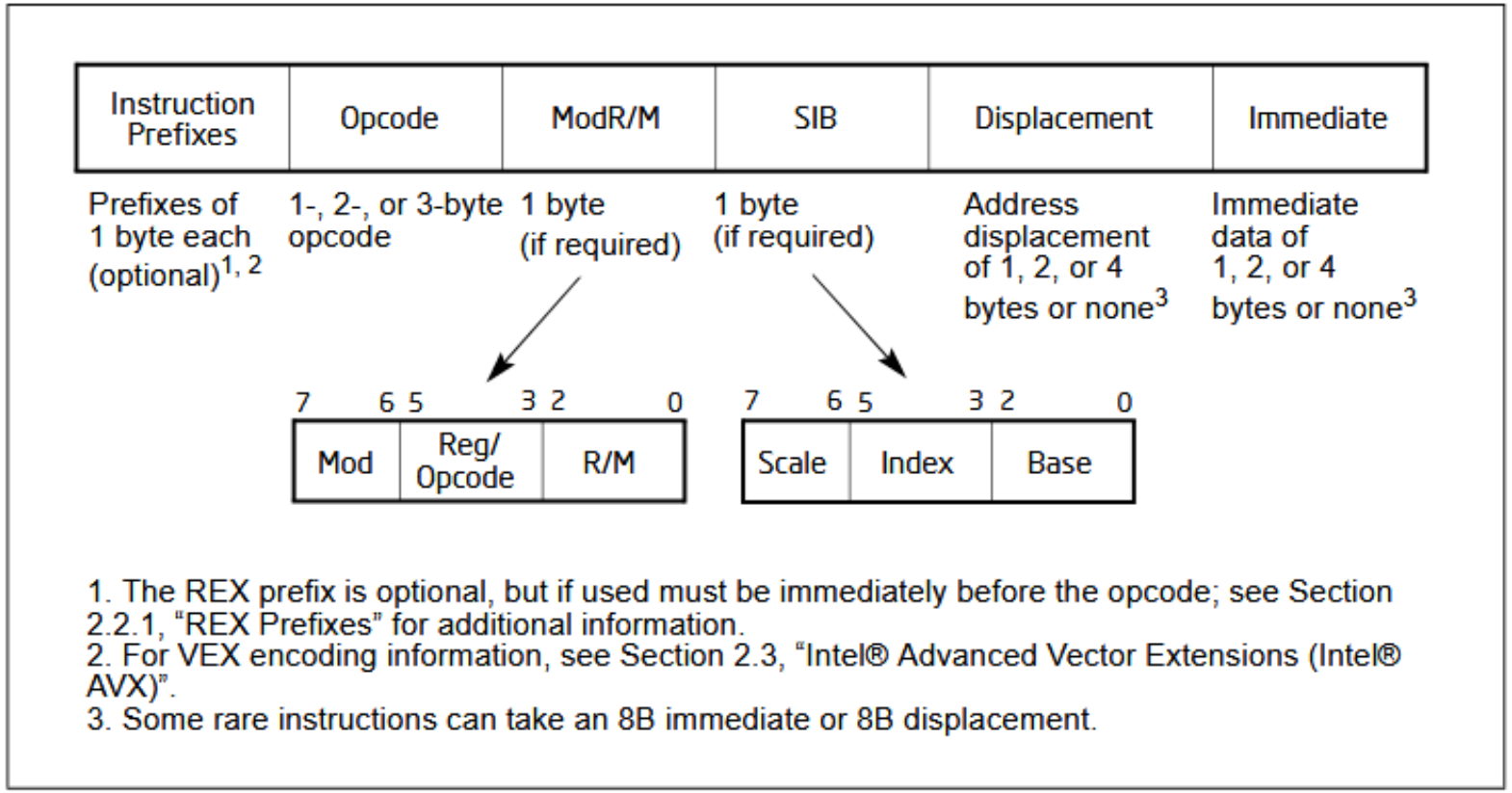
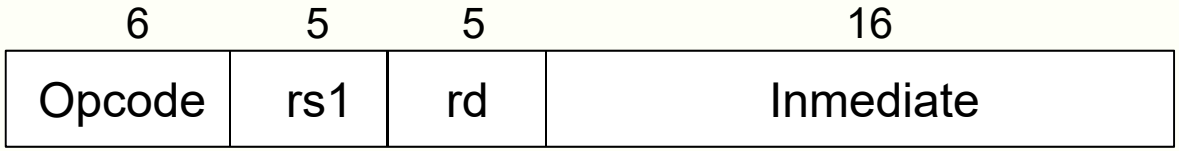


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

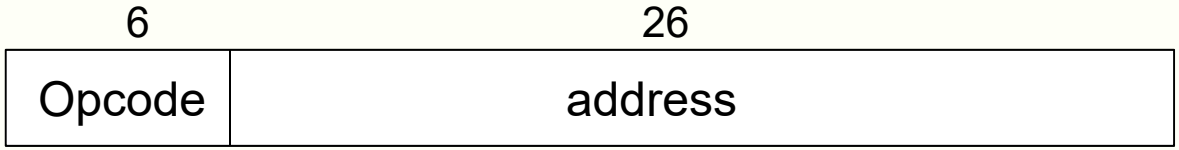
Formatos de instrucción de un RISC

DLX

R-Type

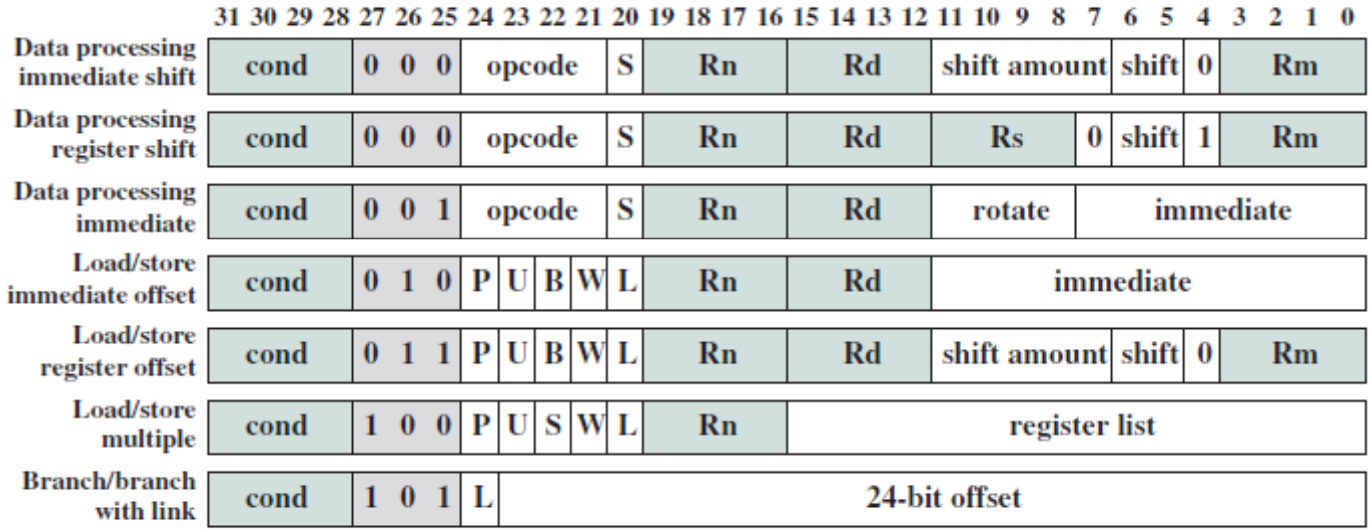


I-Type



J-Type

Formatos de instrucción de un RISC



S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

P, U, W = bits that distinguish among different types of addressing mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

Figure 13.10 ARM Instruction Formats

W. Stallings

Lenguaje máquina: Ejemplos

```
int sumar (int a, int b) {
   int sum, aux1, aux2;
   aux1 = a << 2;
   aux2 = b * 3;
   sum = aux1 + aux2;
   return sum;
}
```

```
 0:  e52db004    push        {fp}
 4:  e28db000    add         fp, sp, #0
 8:  e24dd01c    sub         sp, sp, #28
 c:  e50b0018    str         r0, [fp, #-24]
10:  e50b101c    str         r1, [fp, #-28]
14:  e51b3018    ldr         r3, [fp, #-24]
18:  e1a03103    lsl         r3, r3, #2
1c:  e50b3008    str         r3, [fp, #-8]
20:  e51b201c    ldr         r2, [fp, #-28]
24:  e1a03002    mov         r3, r2
28:  e1a03083    lsl         r3, r3, #1
2c:  e0833002    add         r3, r3, r2
30:  e50b300c    str         r3, [fp, #-12]
34:  e51b2008    ldr         r2, [fp, #-8]
38:  e51b300c    ldr         r3, [fp, #-12]
3c:  e0823003    add         r3, r2, r3
40:  e50b3010    str         r3, [fp, #-16]
44:  e51b3010    ldr         r3, [fp, #-16]
48:  e1a00003    mov         r0, r3
4c:  e28bd000    add         sp, fp, #0
50:  e8bd0800    pop         {fp}
54:  e12fff1e    bx lr
```

```
40051d:   55           push   %rbp
40051e:   48 89 e5     mov    %rsp,%rbp
400521:   89 7d ec     mov    %edi,-0x14(%rbp)
400524:   89 75 e8     mov    %esi,-0x18(%rbp)
400527:   8b 45 ec     mov    -0x14(%rbp),%eax
40052a:   c1 e0 02     shl    $0x2,%eax
40052d:   89 45 fc     mov    %eax,-0x4(%rbp)
400530:   8b 55 e8     mov    -0x18(%rbp),%edx
400533:   89 d0        mov    %edx,%eax
400535:   01 c0        add    %eax,%eax
400537:   01 d0        add    %edx,%eax
400539:   89 45 f8     mov    %eax,-0x8(%rbp)
40053c:   8b 45 f8     mov    -0x8(%rbp),%eax
40053f:   8b 55 fc     mov    -0x4(%rbp),%edx
400542:   01 d0        add    %edx,%eax
400544:   89 45 f4     mov    %eax,-0xc(%rbp)
400547:   8b 45 f4     mov    -0xc(%rbp),%eax
40054a:   5d           pop    %rbp
40054b:   c3           retq
```