Técnicas Digitales II

Arquitectura ARM
Registros - Memoria

Dr.Ing.Steiner Guillermo

- Bibliografía

  Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 6.

  - Hojas 295 a 303

Arquitectura

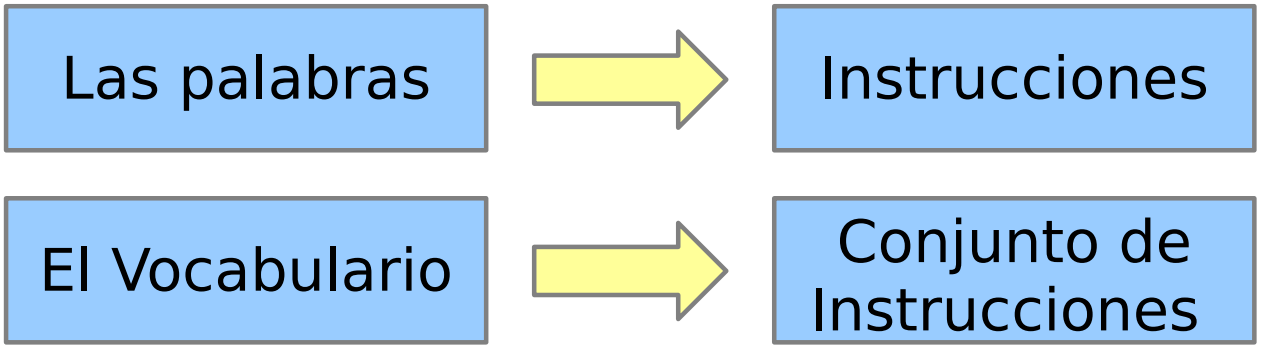
Define el funcionamiento de un microprocesador, es el diseño conceptual y está compuesta por:

- El set de instrucciones.
- Registros
- Modelo de excepciones.
- La ubicación de los operandos con respecto a la cpu y de que manera esta trabaja internamente.

Ejemplo de algunas arquitecturas: ARM, x86 (Intel/AMD), MIPS, SPARC, PowerPC

Arquitectura

- El primer paso para entender una arquitectura es entender su lenguaje.
- En un lenguaje de computación:



- Todos los programas que corren en una computadora posee el mismo conjunto de instrucciones.

Arquitectura

- Cualquier soft que deba correr en un microprocesador es compilado en una serie de instrucciones simples como (sumas, restas y saltos).
- Estas instrucciones poseen la operación y los operandos que utilizará (registros, memoria o un valor en misma operación).
- Como una computadora entiende solo 1 y 0, estas instrucciones están codificadas en binario en un formato denominado Código Máquina. (ARM por ejemplo cada instrucción tiene 32 bits).
- Para que su lectura sea menos tediosa, estas instrucciones se representan en un formato simbólico con el uso de mnemónicos, este nuevo formato se llama Lenguaje Ensamblador.

Arquitectura

- Las diferentes arquitecturas poseen diferentes set de instrucciones.
- Las instrucciones básicas como suma, resta y saltos están presentes en todas las arquitecturas.
- Es por eso que podemos hablar de diferentes dialectos en lugar de diferentes lenguajes.
- Una vez entendido el conjunto de instrucciones de una arquitectura, entender otro es relativamente sencillo.

Arquitectura

- Una arquitectura no define la implementación en hardware, en general existe varias implementaciones para una misma arquitectura.

Ejemplo: Intel y Advanced Micro Devices (AMD) fabrican microprocesadores con arquitectura x86

- Sus implementaciones se realizan con distinto diseño de hardware, poseen diferentes precios y rendimientos.
- Se optimizan con diferentes objetivos, por ejemplo mejora de rendimiento en servidores, economizar energía, etc.

Misma arquitectura significa igual set de instrucciones, pueden correr el mismo software

Arquitectura RISC vs CISC

- CISC (Complex Instruction Set Computer) o Computador con Conjunto de Instrucciones Complejas.

- RISC (Reduced Instruction Set Computer) o Computador con Conjunto de Instrucciones Reducidas.

Arquitectura RISC vs CISC

- CISC (Complex Instruction Set Computer) o Computador con Conjunto de Instrucciones Complejas
  - Esta arquitectura tiene un set de instrucciones mas amplio.
  - Permite operaciones entre memoria o memoria/registro.
  - Motorola 68000, Zilog Z80 y toda la familia Intel x86, AMD x86-64.

Arquitectura RISC vs CISC

- RISC (Reduced Instruction Set Computer) o Computador con Conjunto de Instrucciones Reducidas
  - Set de instrucciones pequeño y de tamaño fijo , esto resulta en hardware para decodificar la instrucción mas simple, pequeño y rápido.
  - Solo existen instrucciones de carga y almacenamiento para acceder a memoria.
  - Suelen tener mayor cantidad de registros.
  - PowerPC, DEC Alpha, MIPS, ARM, SPARC

Set de Instrucciones

- Conjunto de instrucciones que realiza el microprocesador.
- Formada en general por operaciones básicas como sumas y restas.
- Cada instrucción indica la operación y los operandos a usar.
- Los operandos pueden ser registros, memorias o un valor inserto en la misma operación.
- Las instrucciones son codificadas como un número binario en un formato denominado código máquina.
- El ARM representa a cada instrucción con un número de 32 o 16 bits.

ARM

En los capítulos siguientes se desarrollará la arquitectura ARM

- Desarrollada por Acorn Computer Group en la década de 1980.
- Acrom junto con Apple crean en 1990 Advanced RISC Machine (ARM).
- La arquitectura ARM es licenciable. El negocio principal de ARM Holdings es la venta de núcleos.
- Es una de las arquitecturas mas difundidas (Wikipedia)
  - 98% de los mil millones de celulares vendidos anualmente.
  - 90% de microprocesadores RISC.

ARM

- Empresas que poseen licencia ARM: (Wikipedia)

  Alcatel-Lucent, Apple Inc., AppliedMicro, Atmel, Broadcom, Cirrus Logic, Digital Equipment Corporation, Ember, Energy Micro, Freescale, Intel (a través de DEC), LG, Marvell Technology Group, Microsemi, Microsoft, NEC, Nintendo, Nokia, Nuvoton, Nvidia, Sony, MediaTek, NXP (antes Philips Semiconductors), Oki, ON Semiconductor, Psion, Qualcomm, Samsung, Sharp, STMicroelectronics, Symbios Logic, Texas Instruments, VLSI Technology, Yamaha, y ZiiLABS.

Los 4 principios del diseño

Criterios de diseño formulados por David Patterson y John Hennessy en *Computer Organization and Design* utilizado en el diseño de MIPS

1) La regularidad favorece la simplicidad.
2) Hacer rápido el caso mas común.
3) Cuanto mas pequeño mas rápido.
4) Un buen diseño necesita buenas soluciones de compromiso.

Lenguaje Ensamblador

- Para facilitar la lectura de un humano las instrucciones del microprocesador, se representan en un formato simbólico denominado Lenguaje Ensamblador.
- Este lenguaje, representa el conjunto de instrucciones básicas de un microprocesador mediante mnemónicos.
- A diferencias de lenguajes de mas alto nivel, cada microprocesador posee su propio lenguaje ensamblador.
- En general existe una relación de uno a uno desde las sentencias mnemónicas al código máquina.

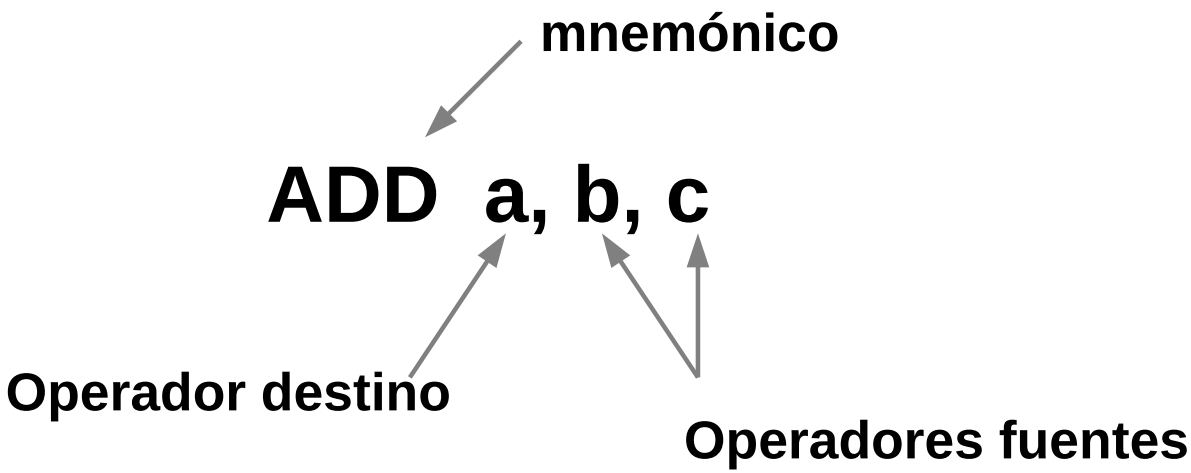
Lenguaje Ensamblador - Instrucciones

- Cada instrucción especifica la operación a realizar y los operandos involucrados.
- Suma en lenguaje ensamblador

Representación en C

a = b + c;

Representación en Ensamblador



Lenguaje Ensamblador - Instrucciones

Resta

```
a = b - c;
```

```
SUB  a, b, c
```

Instrucciones con el mismo formato que ADD 1er principio *La regularidad favorece la simplicidad*

Operación combinada

```
a = b + c - d;
```

```
ADD t, b, c ; t = b + c

SUB a, t, d ; a = t - d
```

Usar múltiples instrucciones para realizar operaciones complejas 2er principio *Hacer rápido el caso mas común*

Lenguaje Ensamblador - Operandos

- Como vimos, las instrucciones requiere de operandos. (a, b, c, ... en los ejemplos).
- Una instrucción requiere una ubicación física de donde se encuentra ese dato.
- Encontramos tres tipos de operandos
  - Almacenados en registros.
  - Almacenados en memoria.
  - Constantes guardadas en la propia instrucción.
- La velocidad de acceso dependerá del tipo de operandos, los registros y las constantes se acceden mas rápidos que aquellos que están en la memoria .

Arquitectura de 32 bits ARM (antes de ARMv8) era de esa arquitectura, porque sus operandos eran de 32 bits.

Registros

Conjunto reducido de registros permiten ser accedido rápidamente 3 er principio
Cuanto mas pequeño mas rápido

- Almacenan los operadores usados en las instrucciones.
- De rápido acceso pero de poca capacidad.
- Su tamaño define el tamaño de la arquitectura.
- ARM posee 16 registros.

a = b + c;

```
; R0=a,R1=b,R2=c
ADD R0,R1,R2
```

a = b + c - d;

```
; R0=a,R1=b,R2=c,R3=d,R4=t
ADD R4,R1,R2 ; t=b+c
SUB R0,R4,R3 ; a=t-d
```

Registros

- Ejemplo: se asume que **a-c** están almacenadas en R0-R2 y **f-j** están almacenadas en R3-R7

a = b – c;
f = (g + h) – (i + j);

```
; ARM assembly code
; R0=a,R1=b,R2=c
; R3=f,R4=g,R5=h,R6=i,R7=j

SUB R0, R1, R2 ; a = b – c
ADD R8, R4, R5 ; R8 = g + h
ADD R9, R6, R7 ; R9 = i + j
SUB R3, R8, R9 ; f=(g+h)–(i+j)
```

Registros

Registro	Uso
R0	Argumento / retorno de valor / variable temporal
R1 - R3	Argumento / variable temporal
R4 - R11	Variables Guardadas
R12	Variable Temporal
R13 (SP)	Puntero del Stack
R14 (LR)	Registro de Enlace o Link
R15 (PC)	Contador de Programa

Constantes / Inmediatos

- Muchas operaciones pueden ser realizadas utilizando una constante o *valor inmediato*.
- Se llama valor inmediato por encontrarse en la misma instrucción y no requerir un nuevo acceso a memoria.

Valores de constantes pequeños

```
a = a + 4;
b = a – 12;
```

```
; R7=a, R8=b
ADD  R7, R7, #4
SUB  R8, R7, #0xC
```

Son precedidas del # y pueden ser escritas en decimal o hexadecimal

```
i = 0;
X = 4080;
```

```
; R4=i, R5=x
MOV R4,#0
MOV R5,#0xFF0
```

Las constantes están limitadas a 8 o 12 bits y no pueden tener una cifra significativa mayor a 8 bits

Constantes / Inmediatos

Valores de constantes grandes

- Si se desea cargar una constante con un numero significativamente grandes.
- Podemos realizarlo utilizando MOV y la instrucción de la or inclusiva ORR

```
int a = 0x7EDC8765;
```

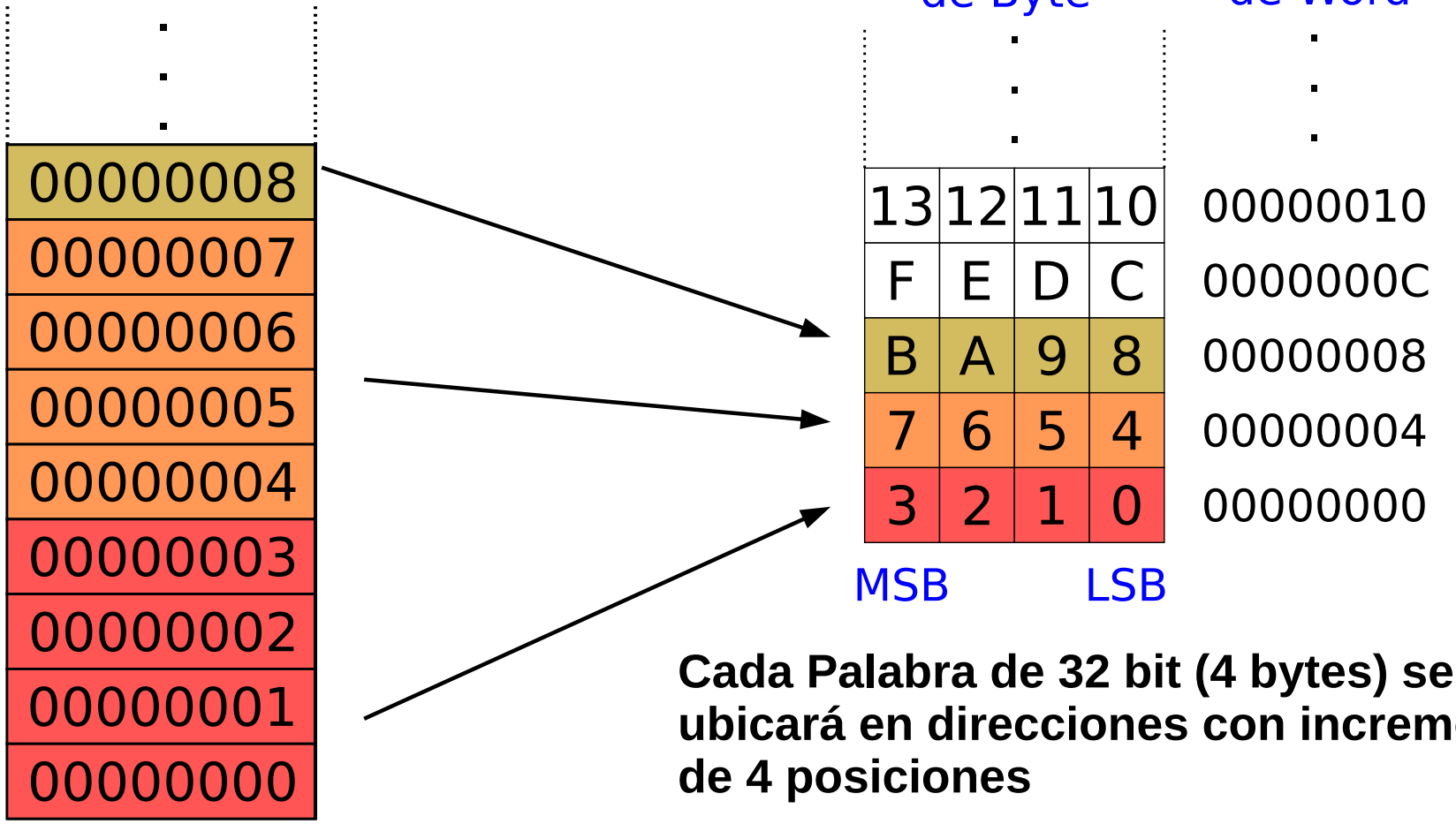
```
; # R0 = a
MOV R0, #0x7E000000
ORR R0, R0, #0xDC0000
ORR R0, R0, #0x8700
ORR R0, R0, #0x65
```

Memoria

- Para sortear la barrera de las 15 variables, se debe mover de la memoria a los registros y de los registros a la memoria.
- En ARM todos las operaciones se realizan entre registros (o registro / inmediato)
- ARM utiliza un modelo de memoria direccionable por bytes (byte-addressable), es decir cada byte tiene una única dirección de memoria.
- La arquitectura ARM utiliza direcciones de memoria de 32 bits y palabras de 32 bits.

Memoria

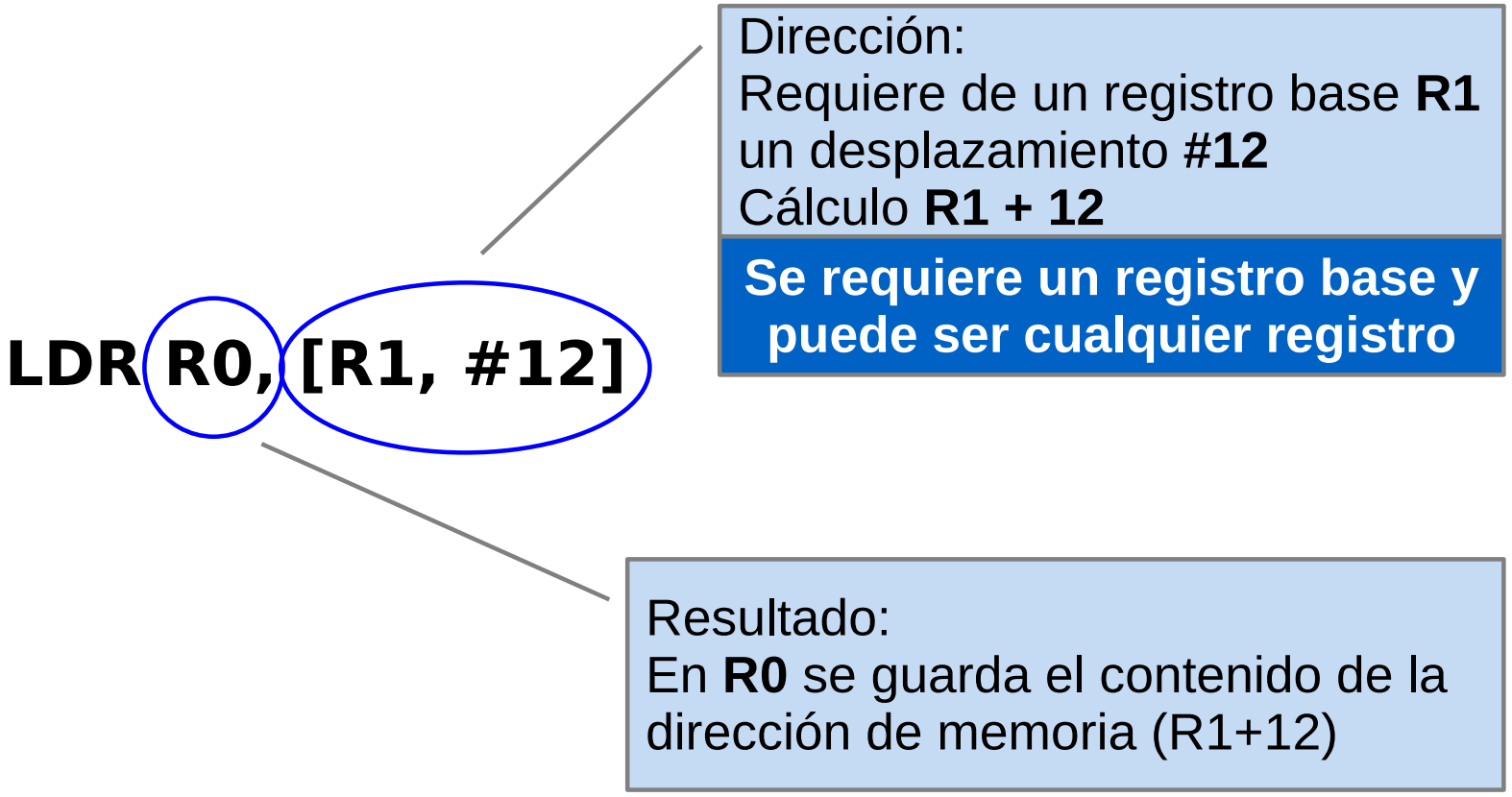
En una memoria orientada a byte
Cada posición representa 1 BYTE



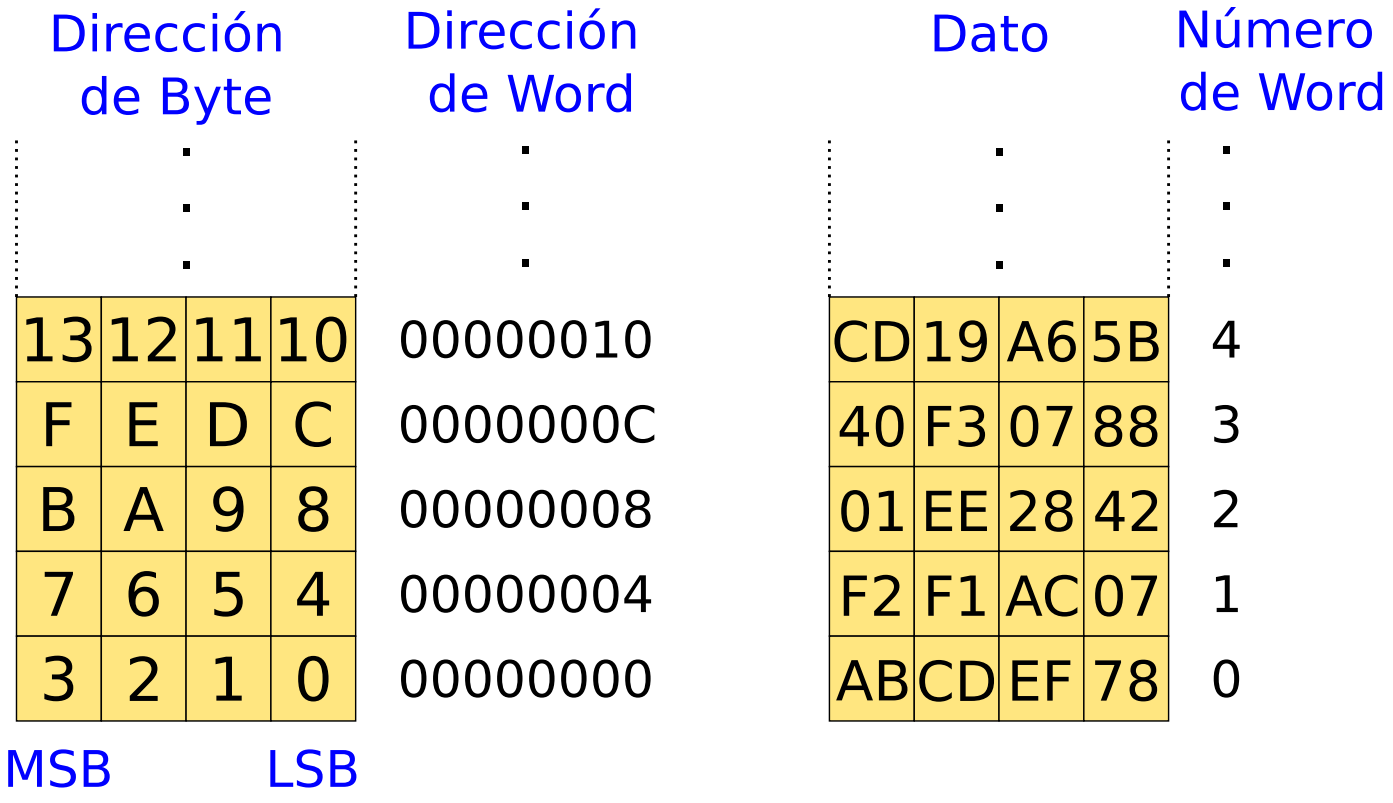
Cada Palabra de 32 bit (4 bytes) se ubicará en direcciones con incremento de 4 posiciones

Lectura de Memoria

- La lectura de memoria se denomina **load**
- El nemónico es **LDR**
- Formato



Ejemplo de lectura de memoria Memoria



Bloque de memoria, cada casillero muestra la DIRECCIÓN donde se encuentra

Ahora cada casillero muestra su contenido

Ejemplo de lectura de memoria Memoria

Leer el elemento 2 de un vector de int (32 bits)

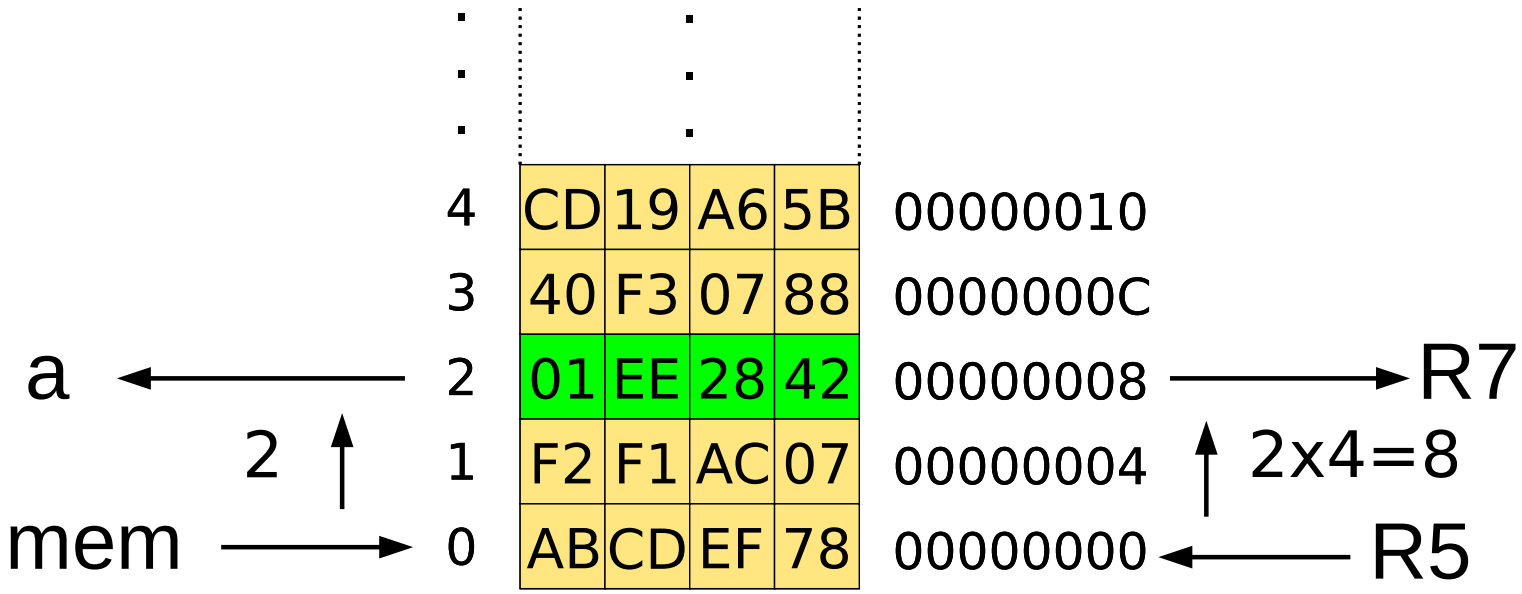
En C

```
a = mem[2];
```

Suponemos inicio del vector = 0

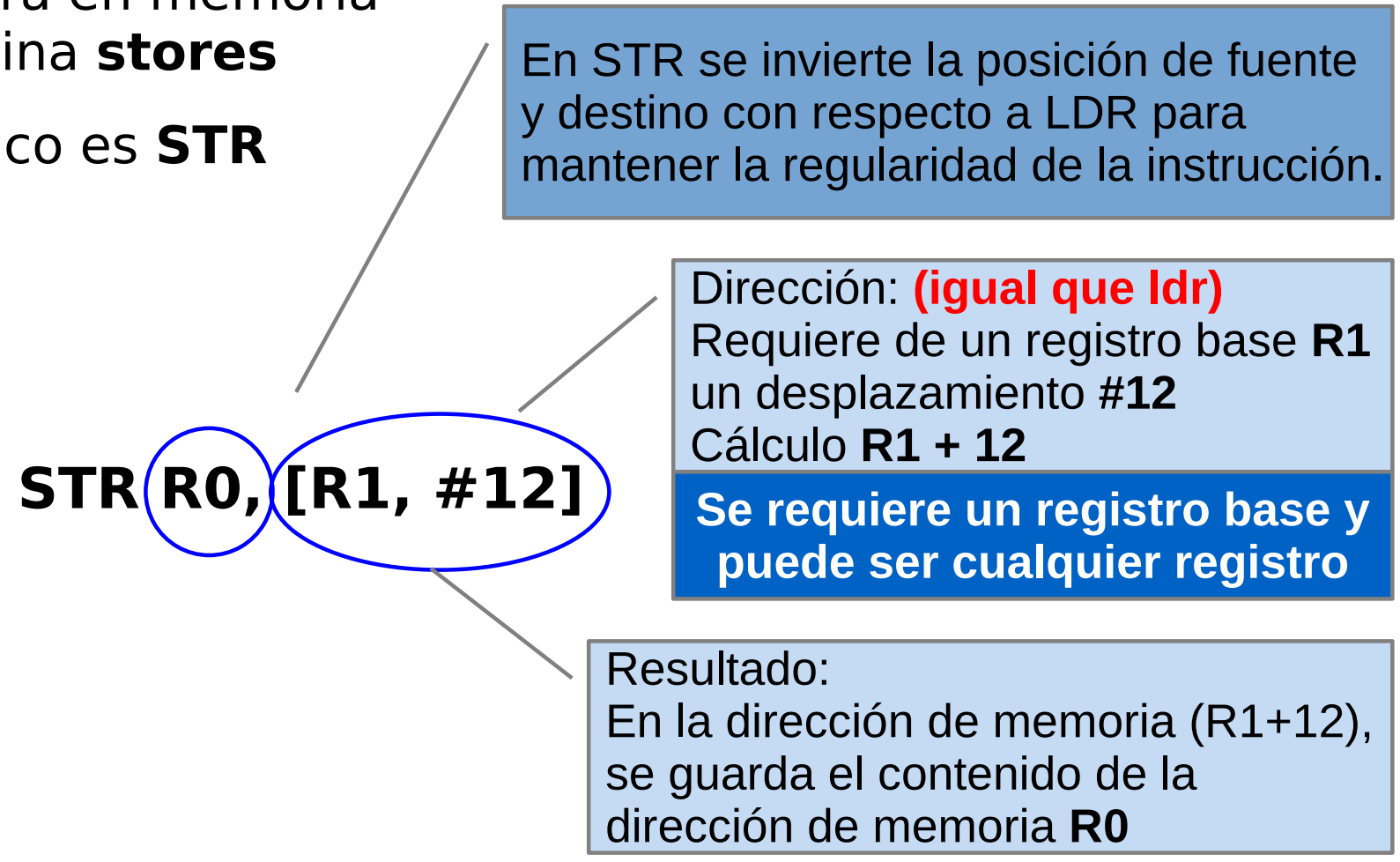
En ensamblador

```
;R7=a
MOV R5, #0
LDR R7,[R5,#8]
```



Escritura en Memoria

- La escritura en memoria se denomina **stores**
- El nemónico es **STR**
- Formato

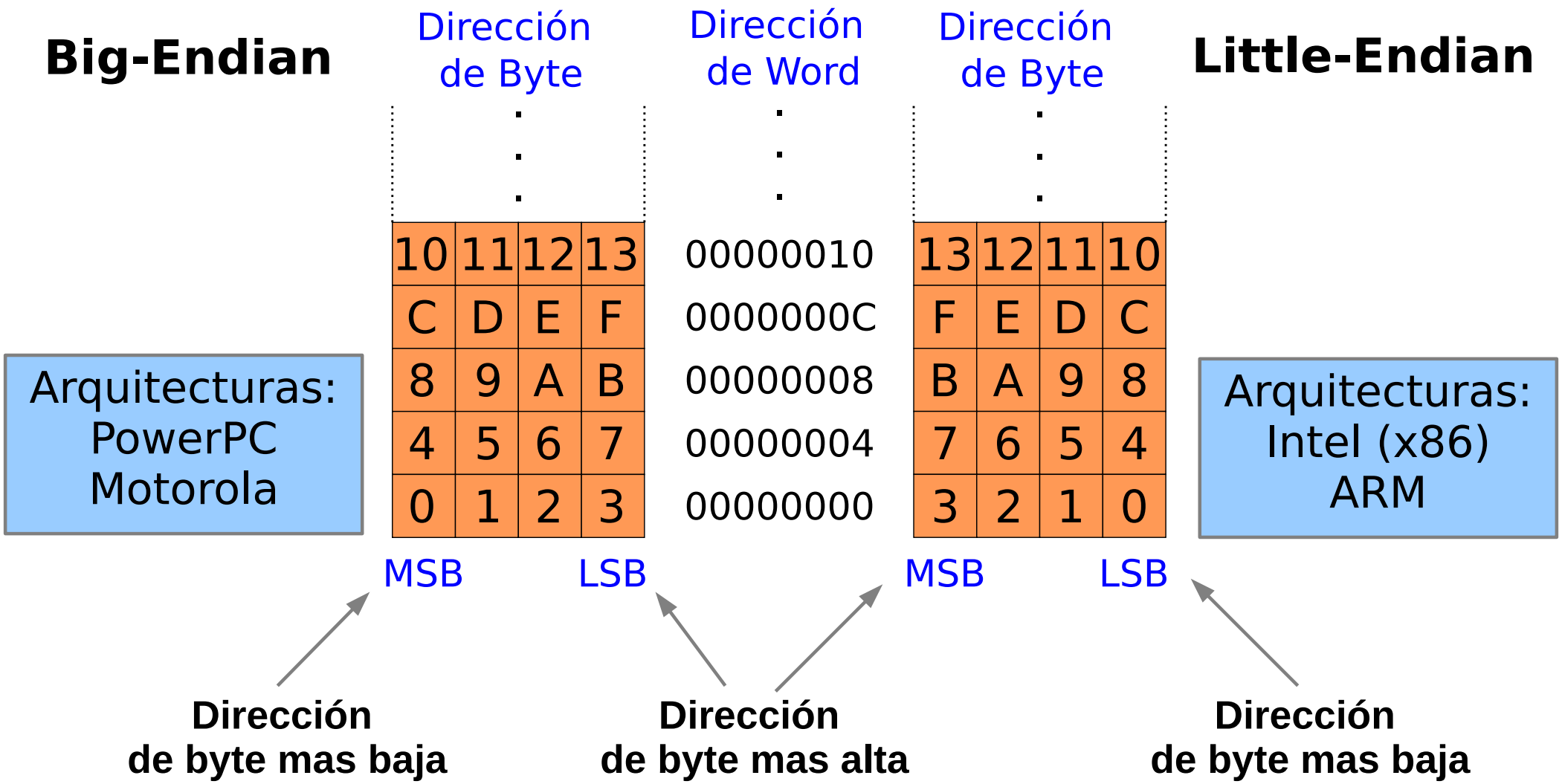


Ejemplo de escritura en memoria

- Guardar en el word 21 el contenido de R7
- Dirección de memoria = 4 x 21 = 84 = 0x54

```
;R7=a
MOV R5, #0
STR R7,[R5,#0x54]
```

Memoria - Endianness



Memoria - Endianness

- **Little-endian**
  - Los bytes del número comienzan por el extremo mas pequeño (little end) o menos significativo.
  - 0x11223344 = {0x44,0x33,0x22,0x11}
  - Arquitecturas: Intel (x86), ARM (acepta los dos, pero en general se implementa little-endian)
- **Big-Endian**
  - Los bytes del número comienzan por el extremo mas grande (big end) o mas significativo.
  - 0x11223344 = {0x11,0x22,0x33,0x44}
  - Arquitecturas: PowerPC, Motorola

Memoria – Endianness - Ejemplo

- ¿ Qué valor lee R7 ?
  donde R5 = 0xABCDEF78

```
STR R5, [R2,#0]
LDRB R7,[R2,#1]
```

Ausencia de subfijo
El STR guarda a partir de la dirección R2+0 el WORD que se encuentra en R5

Subfijo B
El LDR lee el byte que se encuentra en en la direccion R2+0 y lo guarda en R7.

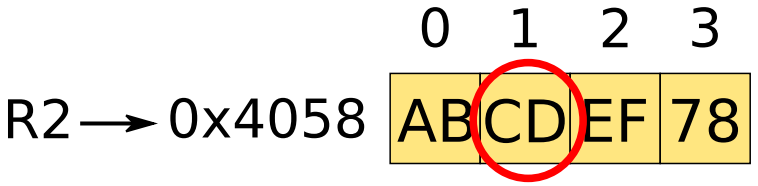
Memoria – Endianness - Ejemplo

- ¿ Qué valor lee R7 ?
  donde R5 = 0xABCDEF78

```
STR R5, [R2,#0]
LDRB R7,[R2,#1]
```

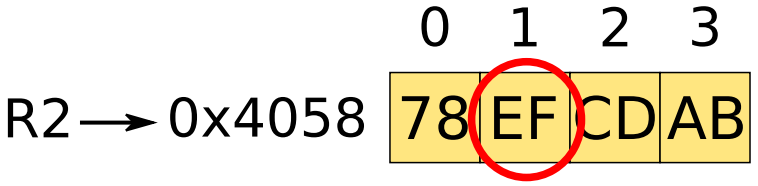
- En big-endian

R7 = 0x000000CD



- En little-endian

R7 = 0x000000EF



- Bibliografía

  Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 6.

¿ Preguntas ?