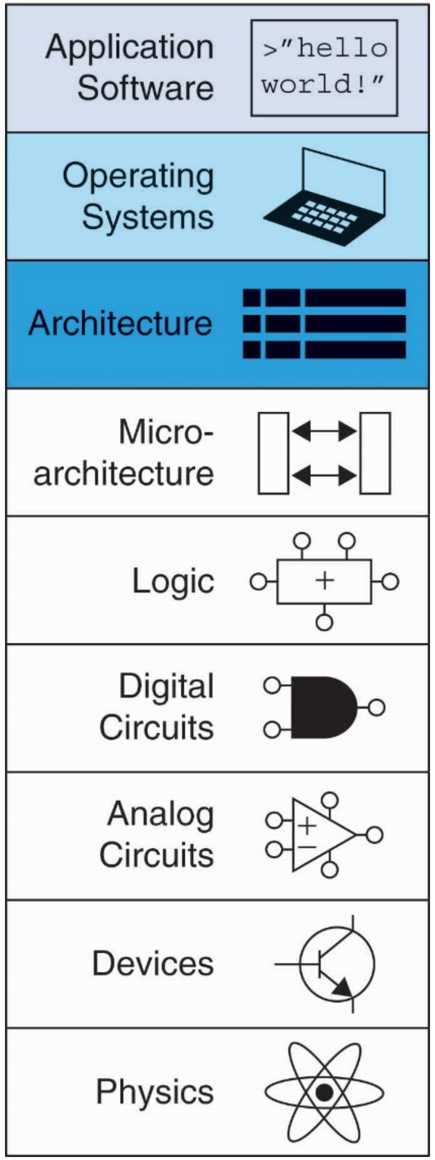
Prof. Mariacarla Staffa
a.a. 2022/2023

Laboratorio di Architettura Degli Elaboratori

Architettura ARM



Introduction

- **Architettura:** descrizione operazionale di computer
    - Come un programmatore *vede* a basso livello un computer
    - Definisce un **insieme di istruzioni** e di registri che fungono da **operandi** per tali istruzioni

Instruzioni

- Istruzioni ARM
  - **Assembly language:** formato human-readable
  - **Machine language:** formato computer-readable

ARM Architecture

- Developed in the 1980's by Advanced RISC (***Reduced Instruction Set Computer***) Machines – now called ARM Holdings
- Nearly 10 billion ARM processors sold/year
- Almost all cell phones and tablets have multiple ARM processors
- Over 75% of humans use products with an ARM processor
- Used in servers, cameras, robots, cars, pinball machines, etc.

Architecture Design Principles

- Principi di progettazione di base di Hennessy and Patterson:
    - **La regolarità favorisce la semplicità**
    - **Rendere veloci le cose frequenti**
    - **Più piccolo è più veloce**
    - **Un buon Progetto richiede buoni compromessi**

Instruction: Addition

C Code

```
a = b + c;
```

ARM Assembly Code

```
ADD a, b, c
```

- **ADD:** mnemonic – indicates operation to perform
- **b, c:** source operands
- **a:** destination operand

Instruction: Subtraction

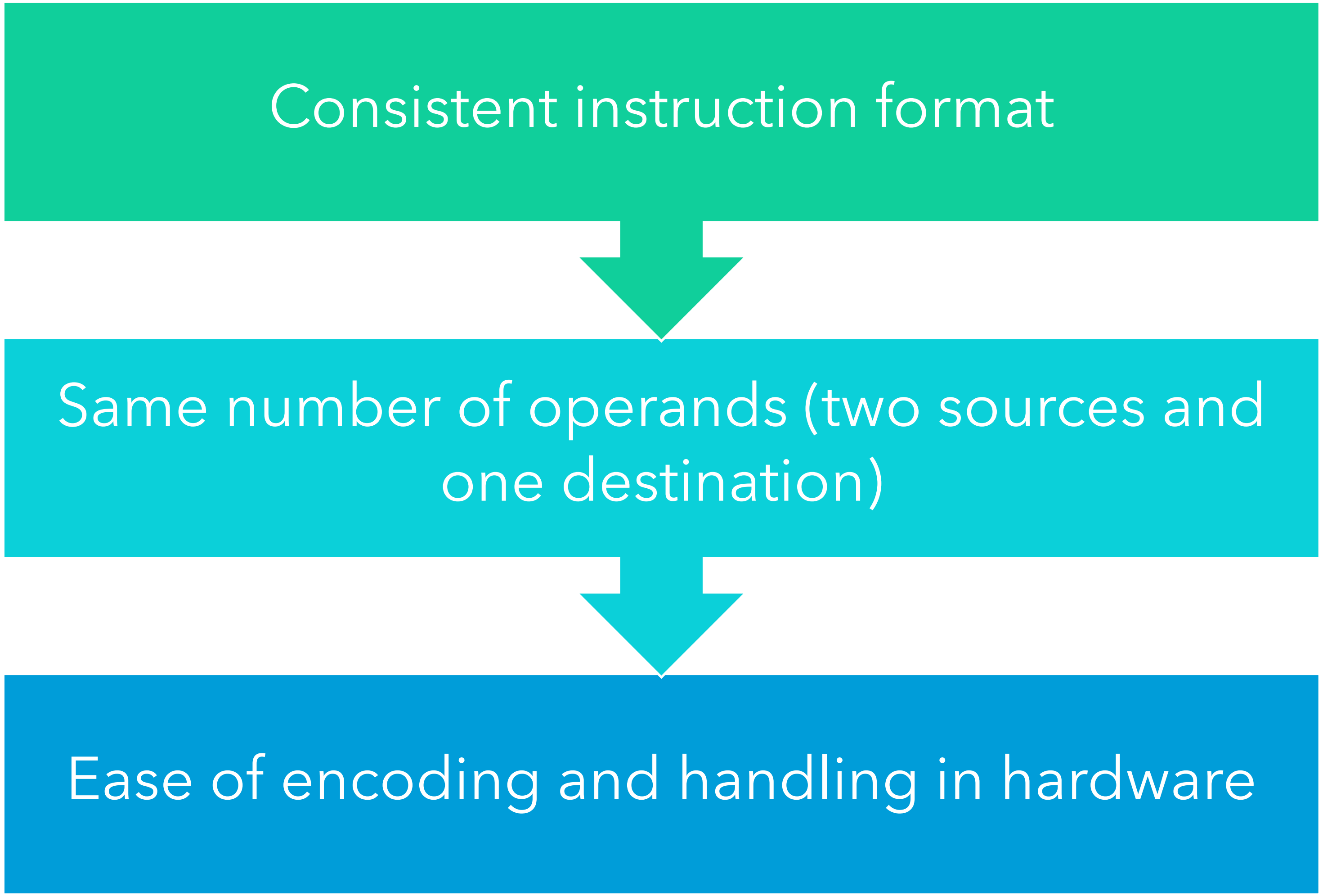
Similar to addition - only mnemonic changes

C Code	ARM assembly code
`a = b - c;`	`SUB a, b, c`

- **SUB:** mnemonic
- **b, c:** source operands
- **a:** destination operand

Design Principle 1

Regularity supports design simplicity



Consistent instruction format

Same number of operands (two sources and one destination)

Ease of encoding and handling in hardware

Multiple Instructions

- More complex code handled by multiple ARM instructions

C Code	**ARM assembly code**
`a = b + c - d;`	`ADD t, b, c  ; t = b + c` `SUB a, t, d  ; a = t - d`

Design Principle 2

Make the common case fast

ARM includes only simple, commonly used instructions

Hardware to decode and execute instructions kept simple, small, and fast

More complex instructions (that are less common) performed using multiple simple instructions

Design Principle 2

Make the common case fast

ARM is a **Reduced Instruction Set Computer (RISC)**, with a small number of simple instructions

Other architectures, such as Intel's x86, are **Complex Instruction Set Computers (CISC)**

Operand Location

- **Physical location in computer**
    - Registers
    - Constants (also called *immediates*)
    - Memory

Operands: Registers

- ARM has 16 registers
- Registers are faster than memory
- Each register is 32 bits
- ARM is called a "32-bit architecture" because it operates on 32-bit data

Design Principle 3

Smaller is faster

ARM includes only a small number of registers

ARM Register Set

ARM ha 16 registri a 32 bit (R0… R15) che sono fisicamente equivalenti fra loro, ma dal punto di vista logico sono usati con scopi specifici

Name	Use
R0	Argument / return value / temporary variable
R1-R3	Argument / temporary variables
R4-R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

Operands: Registers

- **Registers:**
    - R before number, all capitals
    - Example: “R0” or “register zero” or “register R0”

Operands: Registers

- **Registers used for specific purposes:**
    - **Saved registers:** R4-R11 hold variables
    - **Temporary registers:** R0-R3 and R12, hold intermediate values

Stack Pointer

- Lo **stack pointer** è un registro dedicato alla CPU che contiene l'indirizzo della locazione di memoria occupata dal top dello stack per permetterne le operazioni di **push**, che lo incrementerà, e di **pop**, che farà l'inverso, per permettere le operazioni che implicano l'uso dello stack che seguono la logica LIFO ovvero che l'ultimo elemento entrante sarà il primo ad uscire (Last In First Out).
- La memoria stack funziona come una pila di piatti: i piatti vengono aggiunti all'inizio della pila che è lo stack pointer. Invece, quando devono essere ripresi, vengono presi quei "piatti" che sono all'inizio della pila.

Link Register

- Il **Link Register** è un registro di collegamento, ossia un registro che contiene l'indirizzo a cui tornare quando viene completata una chiamata di sottoprogramma.
- Questo è più efficiente dello schema più tradizionale di memorizzazione degli indirizzi di ritorno su uno stack di chiamate, a volte chiamato stack di macchina.

Program Counter

- Nell'architettura dei calcolatori, il **program counter** è un registro della CPU la cui funzione è quella di conservare l'indirizzo di memoria della prossima istruzione da eseguire.
- È un registro puntatore cioè punta a un dato che si trova in memoria all'indirizzo corrispondente al valore contenuto nel registro stesso.

Instructions with Registers

- **Revisit ADD instruction**

C Code	ARM Assembly Code
	`; R0 = a, R1 = b, R2 = c`
`a = b + c`	`ADD R0, R1, R2`

Operands: Constants\Immediates

- Many instructions can use constants or *immediate* operands
- For example: ADD and SUB
- value is *immediate*ly available from instruction

C Code

```

a = a + 4;
b = a - 12;
```

ARM Assembly Code

```
; R0 = a, R1 = b
ADD R0, R0, #4
SUB R1, R0, #12
```

Generazione di costanti

- E' possibile definire costanti con l'istruzione `MOV`:

C Code

```
//int: 32-bit signed word
int a = 23;
int b = 0x45;
```

ARM Assembly Code

```
; R0 = a, R1 = b
MOV R0, #23
MOV R1, #0x45
```

Nota: `MOV` può anche essere usato per spostare il contenuto di un registro in un altro registro:

```
MOV R7, R9
```

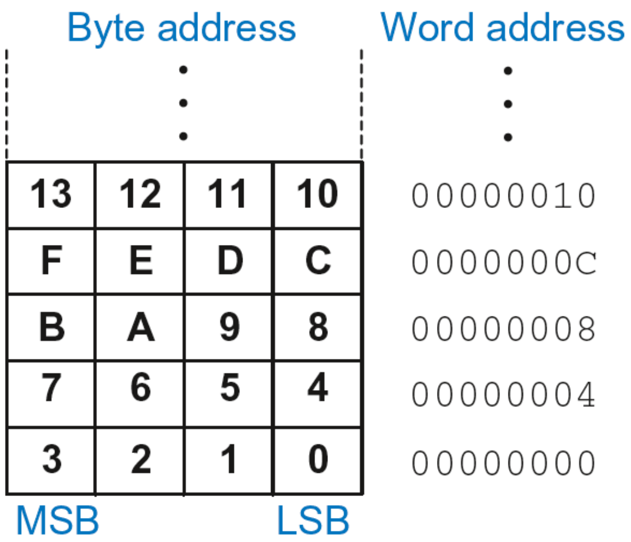
Operands: Memory

- Too much data to fit in only 16 registers
- Store more data in memory
- Memory is large, but slow
- Commonly used variables still kept in registers

- ARM
  - 32 bit per gli indirizzi
  - 32 bit per le parole

Byte-Addressable Memory

- Each data **byte** has unique address
- 32-bit word = 4 bytes, so word address increments by 4



Byte address				Word address
⋮				⋮
13	12	11	10	00000010
F	E	D	C	0000000C
B	A	9	8	00000008
7	6	5	4	00000004
3	2	1	0	00000000
MSB			LSB	

Reading Memory

- Memory read called ***load***
- **Mnemonic:** *load register* (`LDR`)
- **Format:**

```
LDR R0, [R1, #12]
```

LDR loads the value at the address found in R1 to the destination register R0.

Reading Memory

- Memory read called ***load***
- **Mnemonic:** *load register* (`LDR`)
- **Format:**

```
LDR R0, [R1, #12]
```

Address calculation:

- add *base address* (R1) to the *offset* (12)
- address = (R1 + 12)

Result:

- R0 holds the data at memory address (R1 + 12)

Reading Memory

- Memory read called ***load***
- **Mnemonic:** *load register* (`LDR`)
- **Format:**

  ```
  LDR R0, [R1, #12]
  ```

  Address calculation:
  - add *base address* (R1) to the *offset* (12)
  - address = (R1 + 12)

  Result:
  - R0 holds the data at memory address (R1 + 12)

  Any register may be used as base address

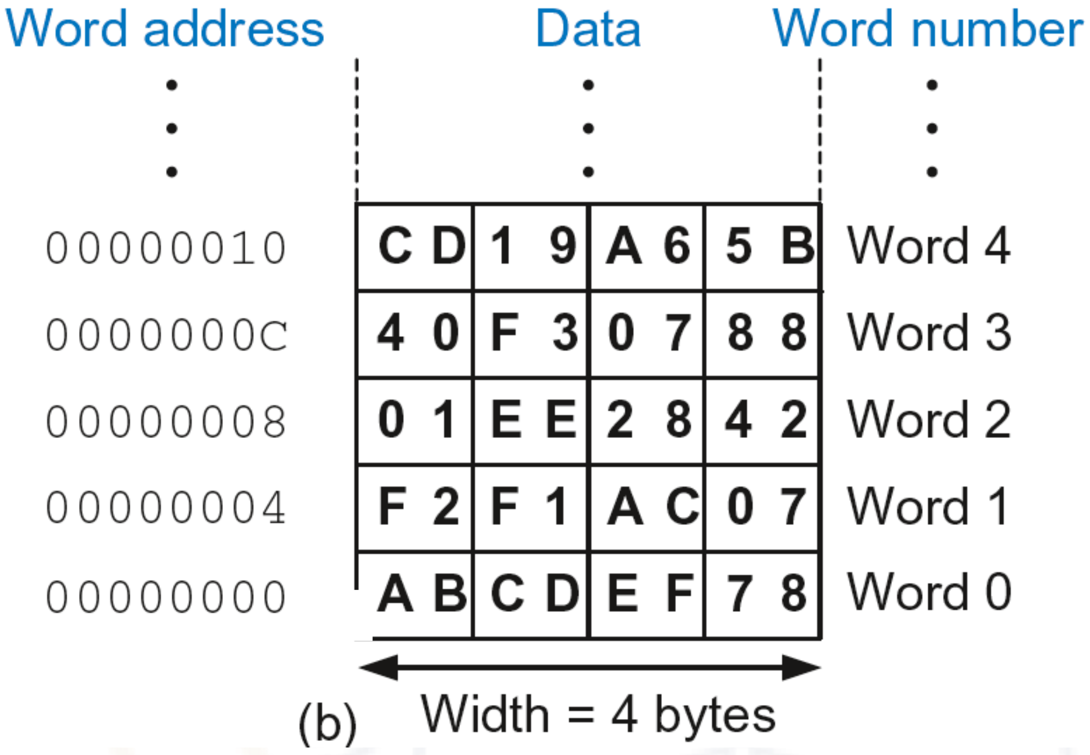
Reading Memory

Example: Read a word of data at memory address 8 into R3

- Address = (R2 + 8) = 8
- R3 = 0x01EE2842 after load

ARM Assembly Code

```
MOV R2, #0
LDR R3, [R2, #8]
```



Word address	Data	Word number
⋮	⋮	⋮
00000010	CD 19 A6 5B	Word 4
0000000C	40 F3 07 88	Word 3
00000008	01 EE 28 42	Word 2
00000004	F2 F1 AC 07	Word 1
00000000	AB CD EF 78	Word 0

(b) Width = 4 bytes

Writing Memory

- Memory write are called ***stores***
- **Mnemonic:** *store register* (`STR`)

Writing Memory

- **Example:** Store the value held in R7 into memory word 21.

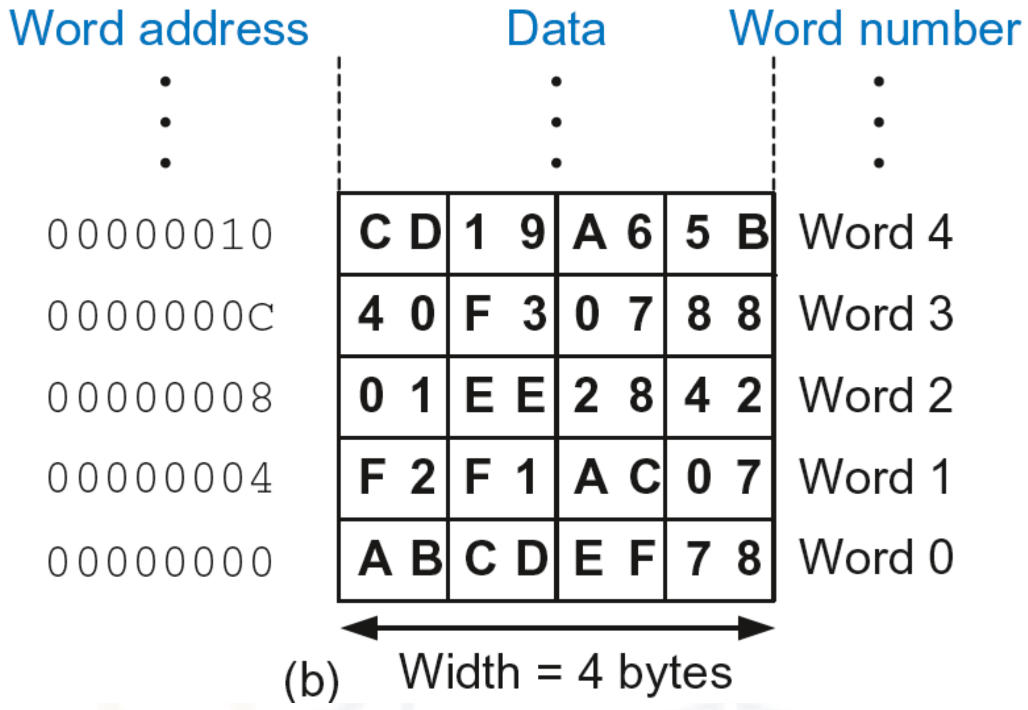
Writing Memory

Example: Store the value held in R7 into memory word 21.

- Memory address = 4 x 21 = 84_{10} = $0x54_{hex}$ =01010100_2

ARM assembly code

```
MOV R5, #0
STR R7, [R5, #0x54]
```



Word address	Data				Word number
⋮	⋮				⋮
00000010	C D	1 9	A 6	5 B	Word 4
0000000C	4 0	F 3	0 7	8 8	Word 3
00000008	0 1	E E	2 8	4 2	Word 2
00000004	F 2	F 1	A C	0 7	Word 1
00000000	A B	C D	E F	7 8	Word 0

(b) Width = 4 bytes

Writing Memory

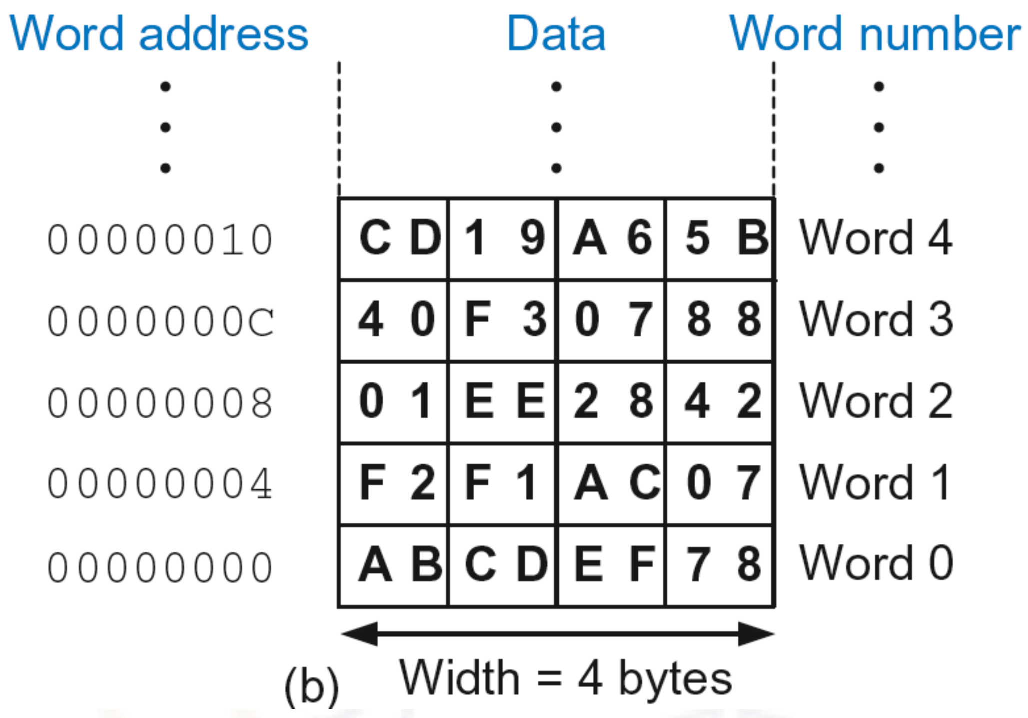
Example: Store the value held in R7 into memory word 21.

- Memory address = 4 x 21 = 84 = 0x54

ARM assembly code

```
MOV R5, #0
STR R7, [R5, #0x54]
```

The offset can be written in decimal or hexadecimal

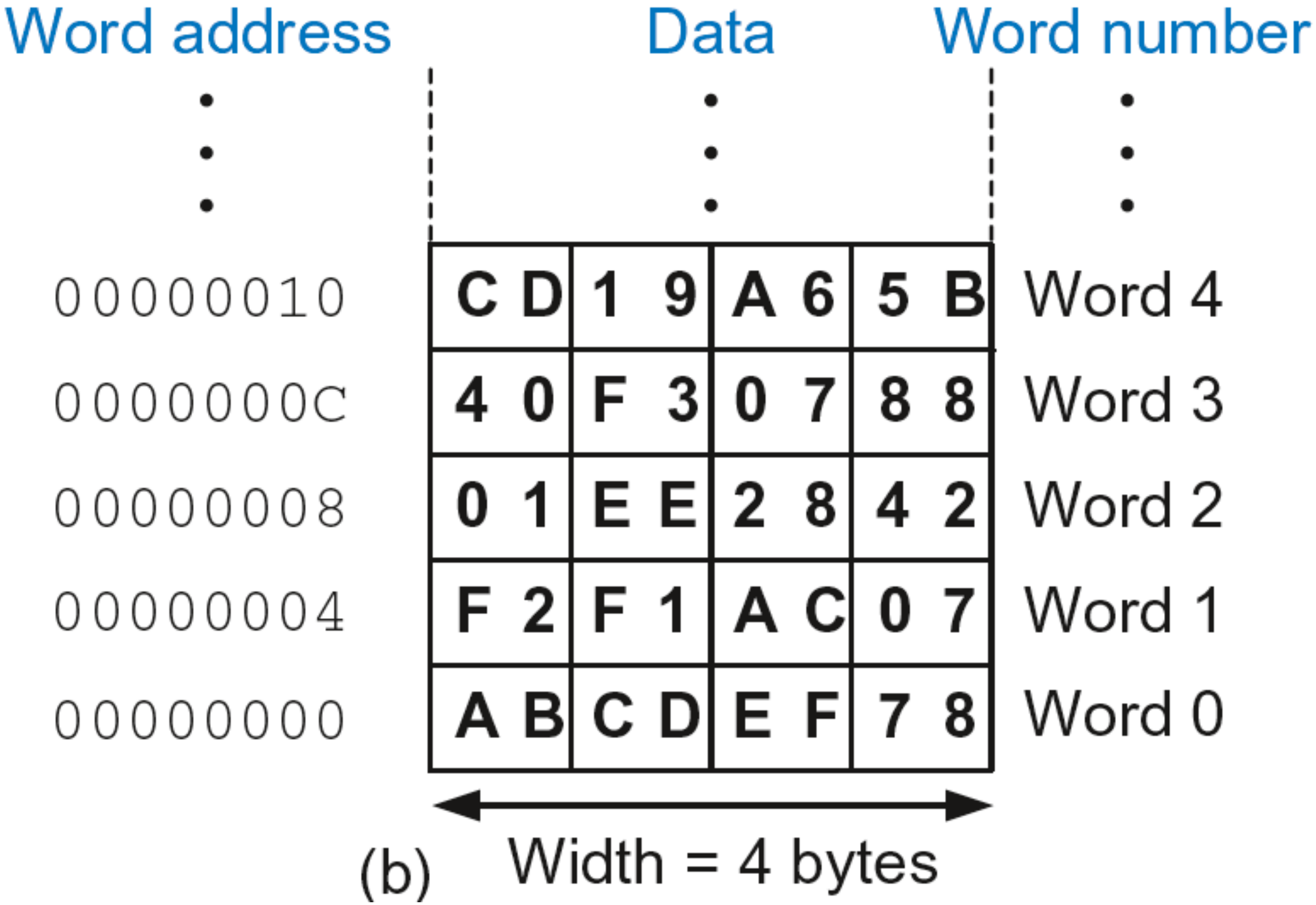


Word address	Data	Word number
⋮	⋮	⋮
00000010	C D 1 9 A 6 5 B	Word 4
0000000C	4 0 F 3 0 7 8 8	Word 3
00000008	0 1 E E 2 8 4 2	Word 2
00000004	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

(b) Width = 4 bytes

Recap: Accessing Memory

- Address of a memory **word** must be multiplied by 4
- **Examples:**
  - Address of memory word 2 = 2 × 4 = 8
  - Address of memory word 10 = 10 × 4 = 40

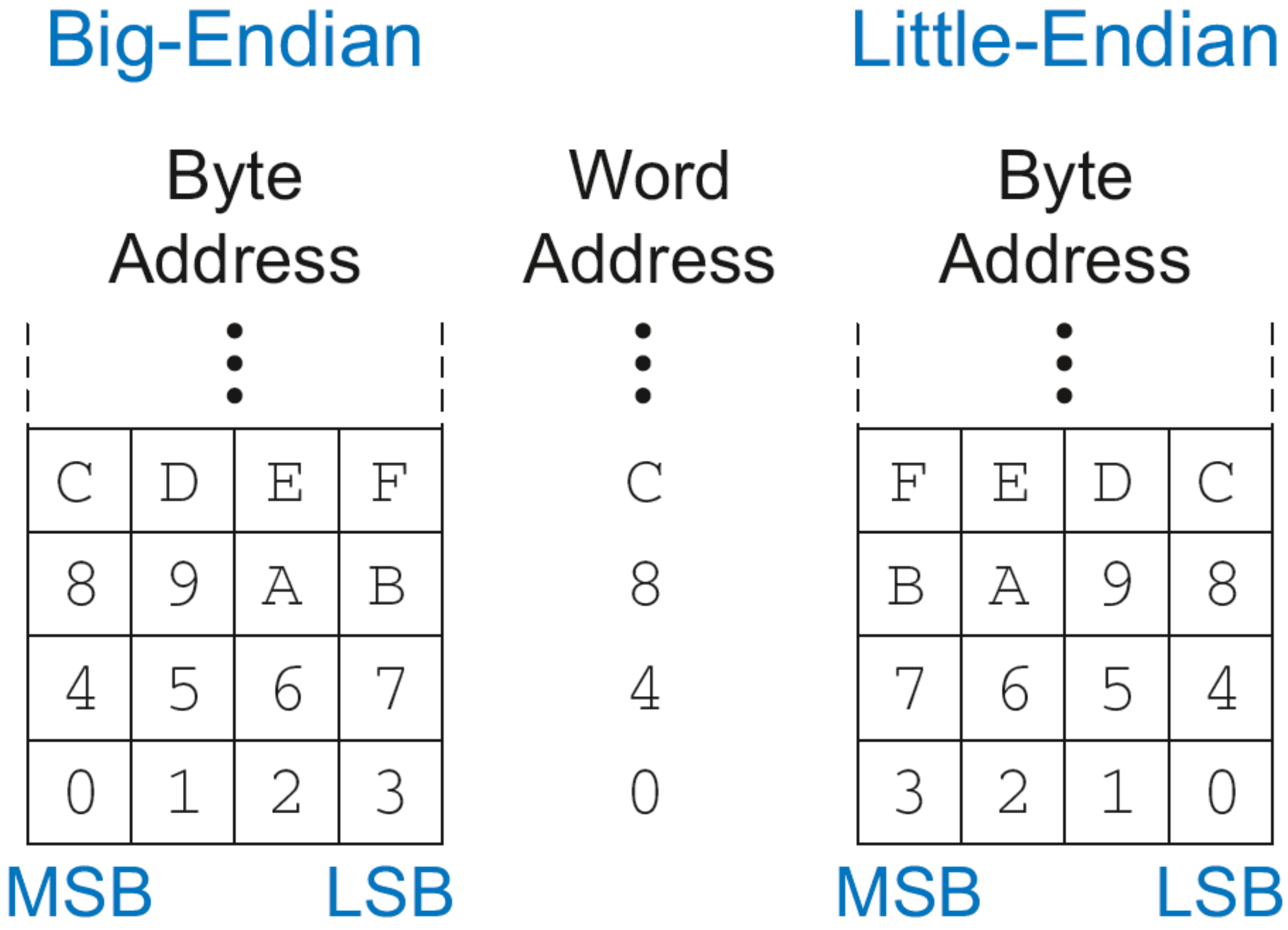


Big-Endian & Little-Endian Memory

- How to number bytes within a word?

Big-Endian & Little-Endian Memory

- **How to number bytes within a word?**
  - **Little-endian:** byte numbers start at the **little** (least significant) end
  - **Big-endian:** byte numbers start at the **big** (most significant) end



Big-Endian					Little-Endian			
Byte Address				Word Address	Byte Address			
⋮				⋮	⋮			
C	D	E	F	C	F	E	D	C
8	9	A	B	8	B	A	9	8
4	5	6	7	4	7	6	5	4
0	1	2	3	0	3	2	1	0
MSB			LSB		MSB			LSB