

# Architettura degli Elaboratori

## Le istruzioni della CPU – parte 1



SAPIENZA  
UNIVERSITÀ DI ROMA

Alessandro Checco

[alessandro.checco@uniroma1.it](mailto:alessandro.checco@uniroma1.it)

[S&PdC] 2.1 – 2.5

Special thanks and credits:

Claudio Di Ciccio, Iacopo Masi, e

Andrea Sterbini



# Logistica e Comunicazioni

Accesso al materiale e comunicazioni: Google Classroom

Architettura degli Elaboratori (a.a. 2023-24)  
Canale 2 (M-Z)

Stream Classwork People

tori...  
23-...

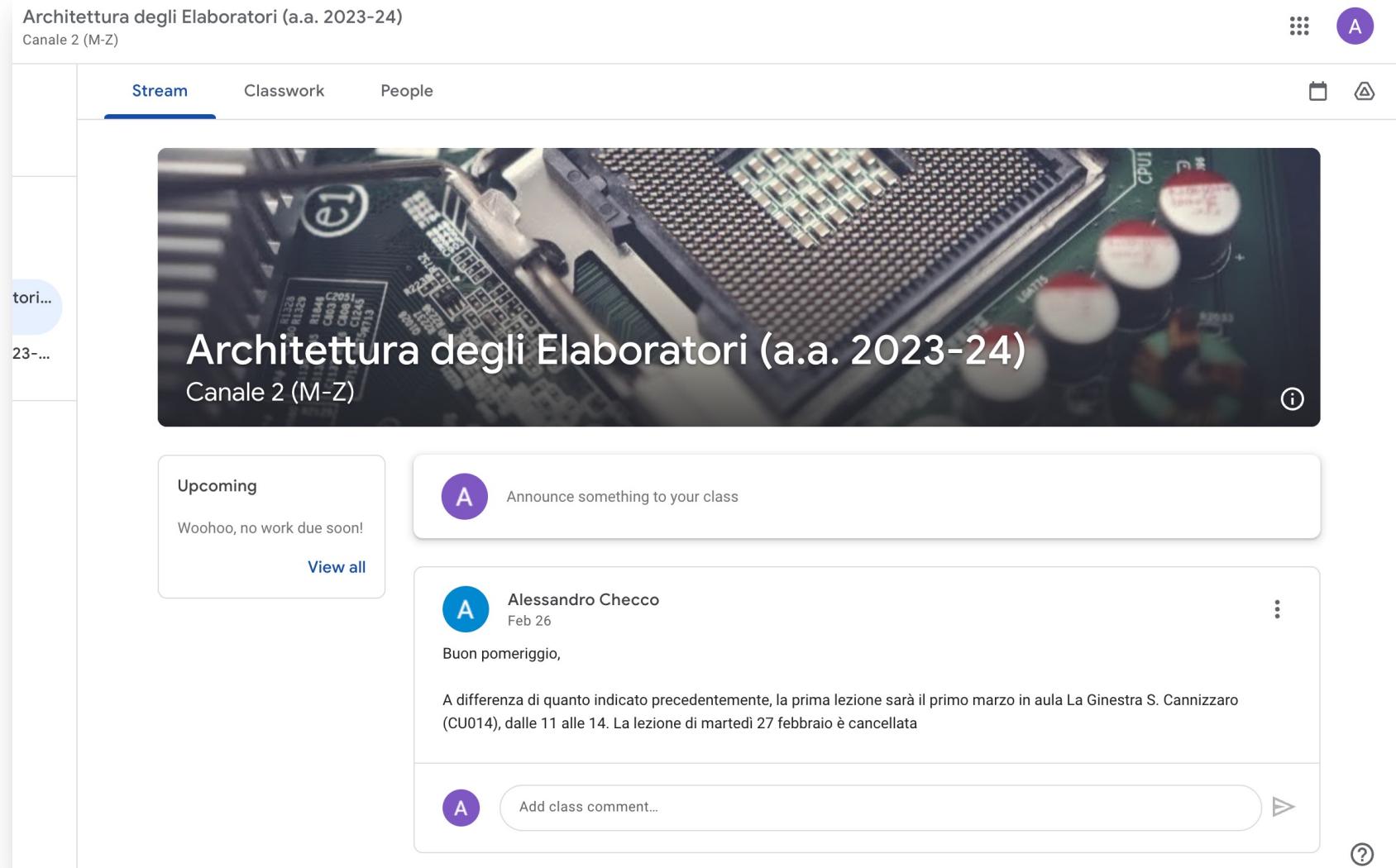
**Architettura degli Elaboratori (a.a. 2023-24)**  
Canale 2 (M-Z)

Upcoming  
Woohoo, no work due soon!  
[View all](#)

A Announce something to your class

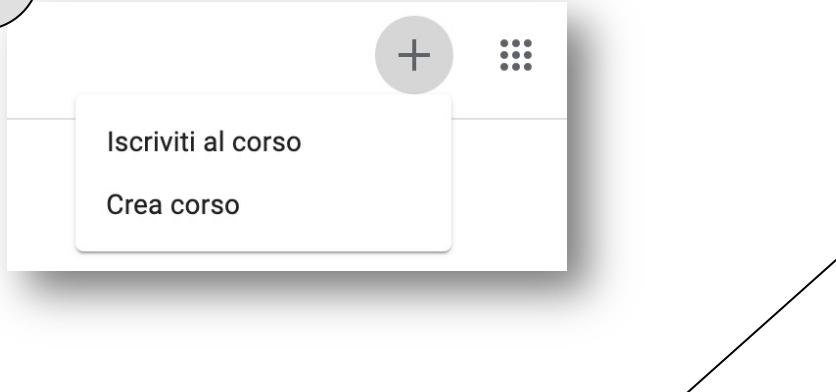
Alessandro Checco  
Feb 26  
Buon pomeriggio,  
  
A differenza di quanto indicato precedentemente, la prima lezione sarà il primo marzo in aula La Ginestra S. Cannizzaro (CU014), dalle 11 alle 14. La lezione di martedì 27 febbraio è cancellata

Add class comment... ▶ ?



# Google Classroom

1



Codice corso  
Chiedi il codice del corso all'insegnante e inseriscilo qui.

Per accedere con un codice di corso

- Utilizza un account autorizzato
- Utilizza un codice corso con 5-7 lettere o numeri, senza spazi né simboli

Se hai problemi a iscriverti al corso, consulta l'[articolo del Centro assistenza](#)

codice

2

Il codice (o link di registrazione) è anche in bacheca:

xz6gckm

# Calendario delle lezioni

Controllare il calendario per potenziali cambi di programma. Se siete iscritti a Google Classroom del corso riceverete una notifica.

22-23) Stream Classwork People

View your work Google Calendar Class Drive folder

Calendario delle lezioni Posted 10:48 AM

Materiale didattico

Today March 2024 Week

MON	TUE	WED	THU	FRI
4	5	6	7	8
7 AM				
8 AM				
9 AM				
10 AM				
11 AM				
12 PM	[Teaching] [AE] Architettura degli Elaboratori 12 – 2pm			
1 PM				
2 PM				
3 PM				
4 PM				

[AE] Architettura degli Elaboratori  
11am – 2pm  
Tullio Levi Civita edificio Castelnuovo

- Tutte le lezioni saranno a Tullio Levi Civita edificio Castelnuovo (aula 3)
- Break a metà lezione?

# Argomenti

---

## Motivazioni

Nel resto del corso vedremo nel dettaglio la progettazione di una CPU MIPS, che:

- Ha un set di istruzioni ristretto (**RISC**)
- È volutamente semplice
- È facile da velocizzare con la pipeline
- È facile da parallelizzare
- È un esempio eclatante di progettazione coordinata dell'hardware col software

## Argomenti della lezione

- Rappresentazione dei numeri
- Interi con Segno (complemento a 2)
- Floating point / Double
- Formati delle istruzioni RISC
- Primi cenni di istruzioni MIPS

Esempi di CPU con architettura RISC:

- **SPARC** di Sun, **MIPS** in sistemi embedded
- **PowerPC** nei Mac e server IBM
- **Cell** delle Play Station Sony
- **ARM** di quasi tutti i cellulari e tablet

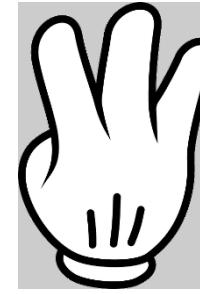
## Ripasso, Preliminari, Sistema Posizionale



SAPIENZA  
UNIVERSITÀ DI ROMA

# Contare

---



# Bits, nibbles e bytes

Un **binary digit**, o **bit**, è un elemento di un insieme binario di simboli.

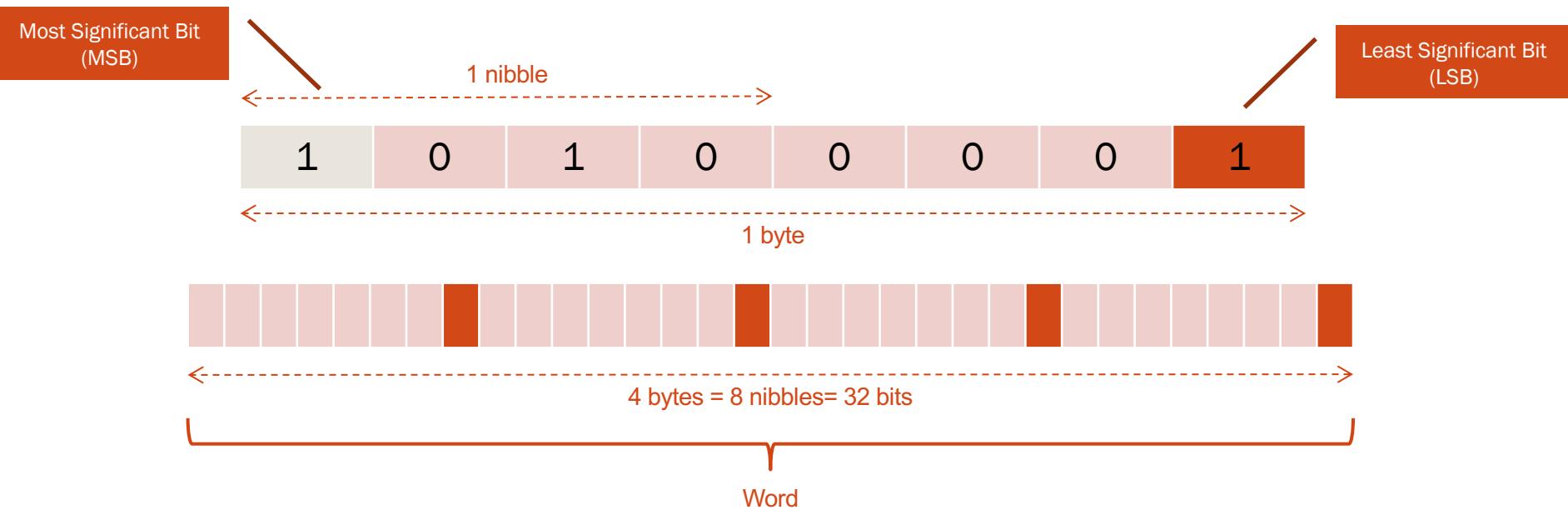
Usiamo:

0 (zero)

1 (uno)

Un **nibble** è una sequenza di 4 bit

Un **byte** è una sequenza di 8 bit



# Bits, nibbles e bytes

Un **binary digit**, o **bit**, è un elemento di un insieme binario di simboli.

Usiamo:

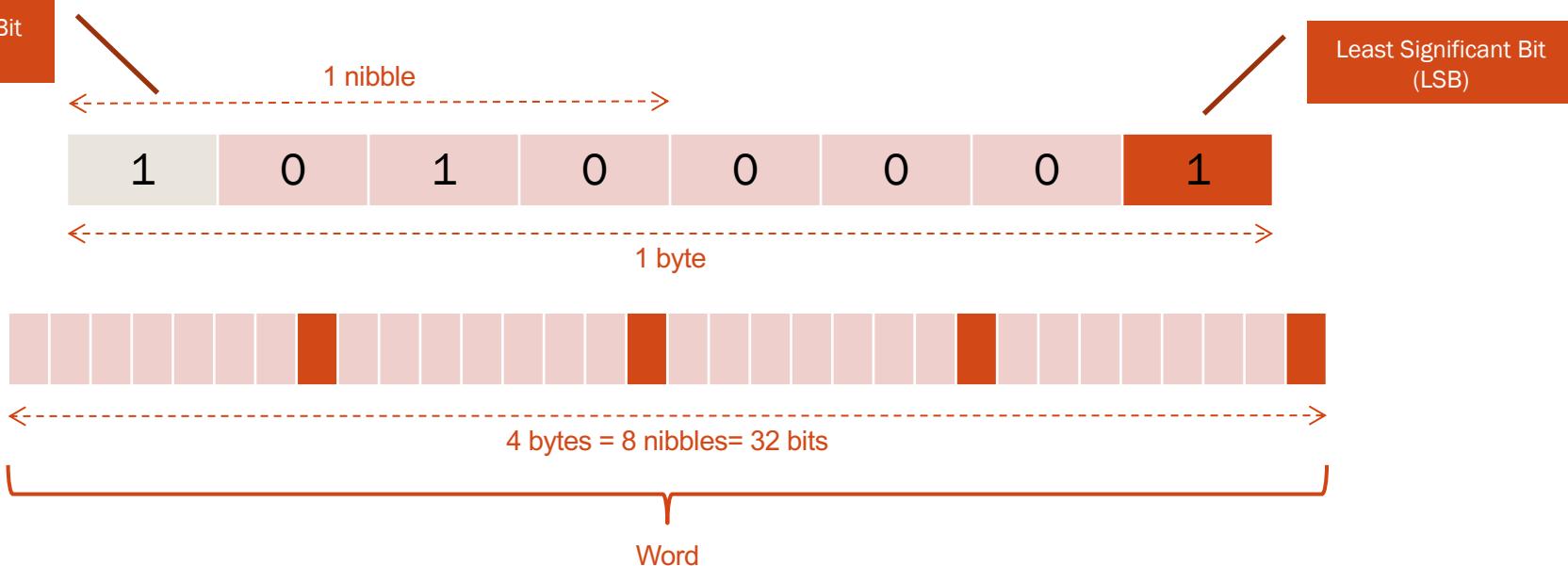
0 (zero)

1 (uno)

Un **nibble** è una sequenza di 4 bit

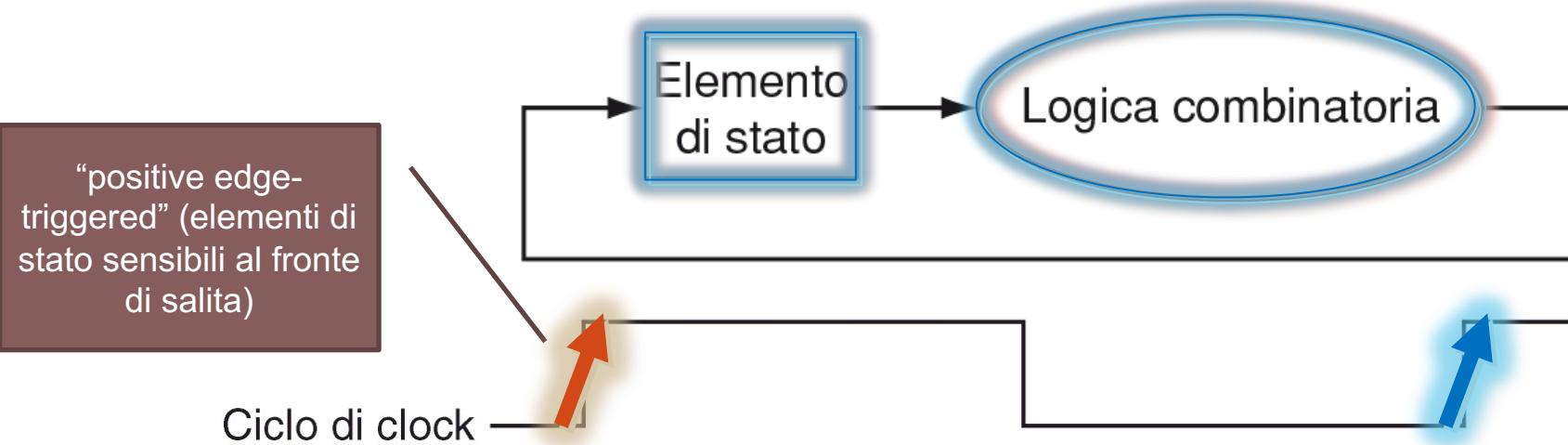
Un **byte** è una sequenza di 8 bit

- 1 bit  $\in \{0, 1\}$
- 8 bit  $\rightarrow$  1 byte
- 32 bit  $\rightarrow$  4 byte  $\rightarrow$  1 word



# Perché solo cifre binarie?

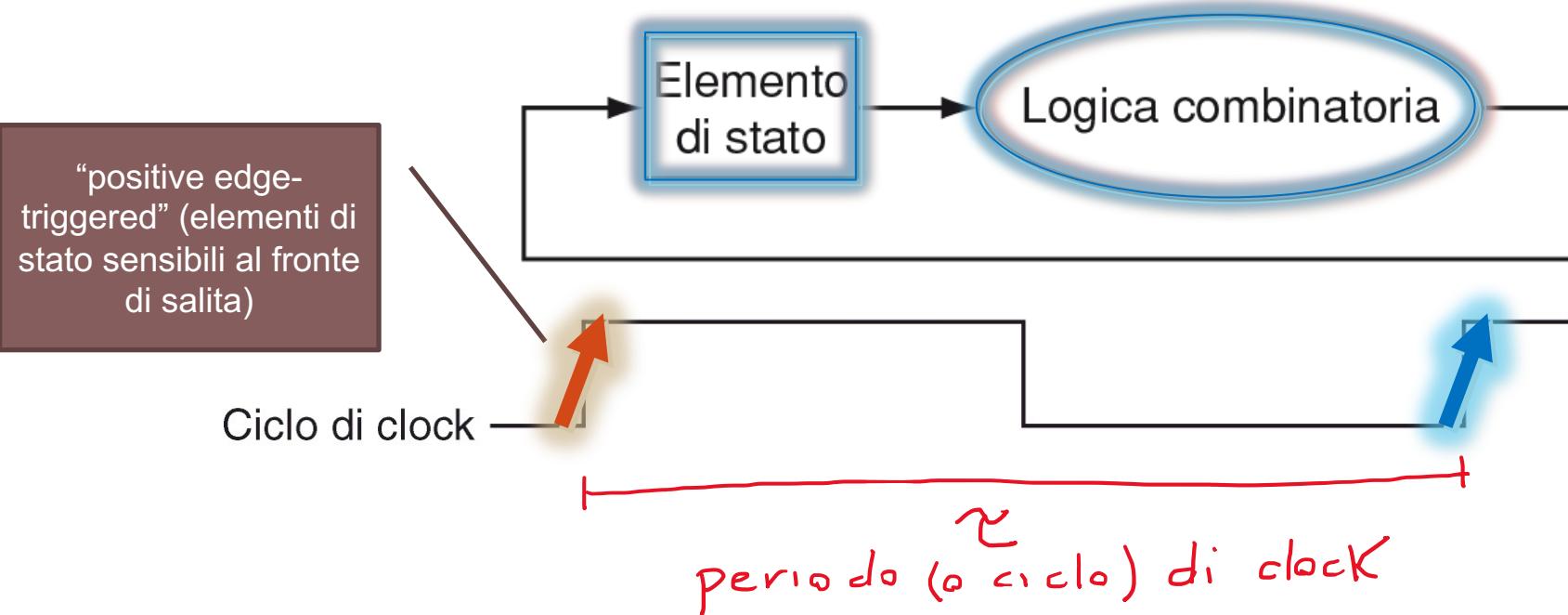
CPU = macchina sequenziale    ovvero    Stato + circuito combinatorio



Nel sistema binario ogni numero può essere scritto con due soli segni, lo 0 e l'1, facilmente traducibili, come dicevamo, in segnali elettrici per il calcolatore e che questo esegue rapidamente operazioni anche molto complesse.

# Perché solo cifre binarie?

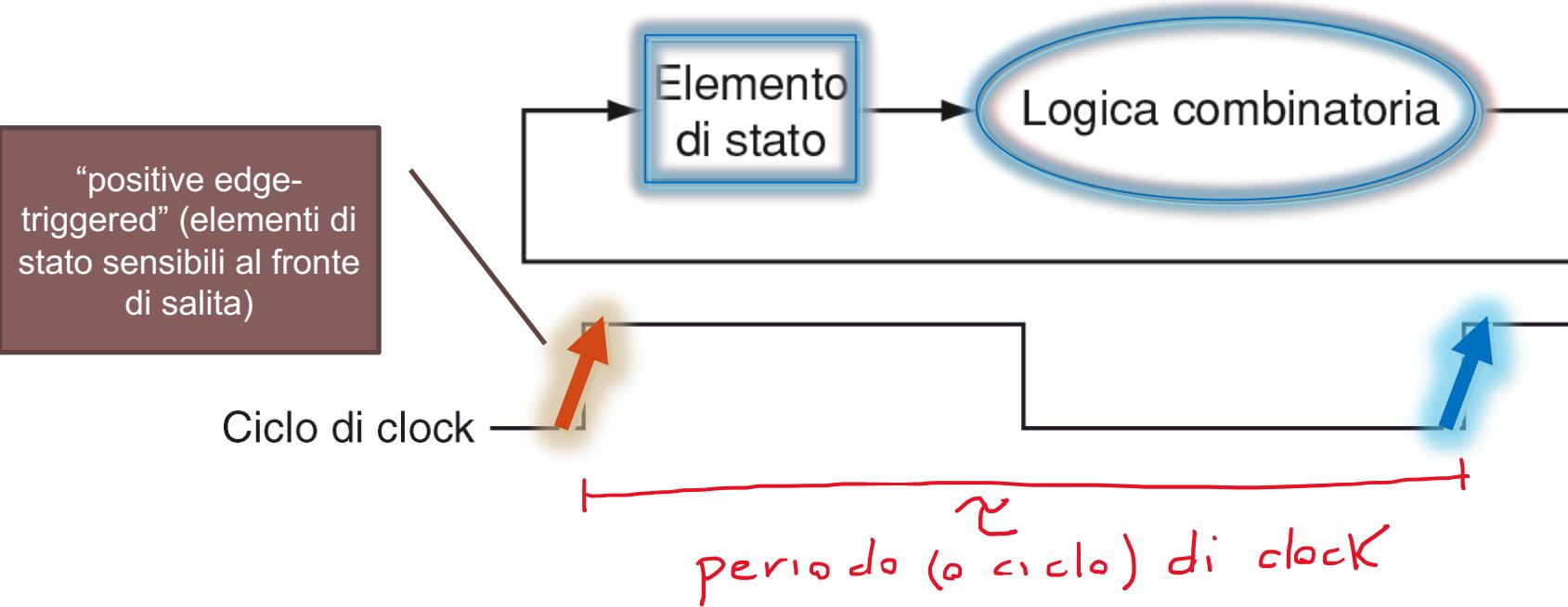
CPU = macchina sequenziale    ovvero    Stato + circuito combinatorio



Nel sistema binario ogni numero può essere scritto con due soli segni, lo 0 e l'1, facilmente traducibili, come dicevamo, in segnali elettrici per il calcolatore e che questo esegue rapidamente operazioni anche molto complesse.

# Perché solo cifre binarie?

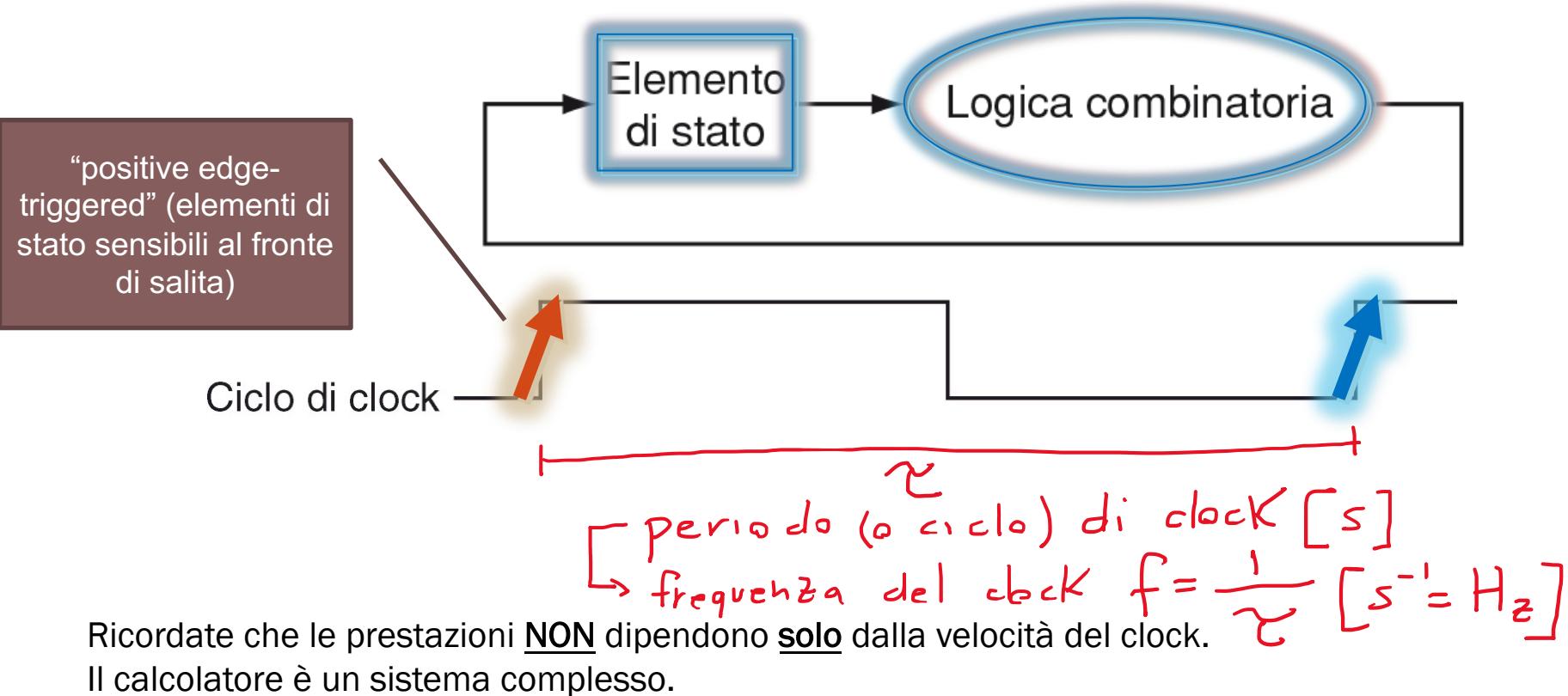
CPU = macchina sequenziale    ovvero    Stato + circuito combinatorio



Ricordate che le prestazioni NON dipendono solo dalla velocità del clock.  
Il calcolatore è un sistema complesso.

# Perché solo cifre binarie?

CPU = macchina sequenziale ovvero Stato + circuito combinatorio



Ripasso: In generale il tempo di esecuzione di un programma dipende da

- 1) # di istruzioni di un programma
- 2) CPI (#medio di clock per istruzione)
- 3) velocità clock

# Prestazioni

$$\text{Tempo di CPU relativo a un programma} = \frac{\text{Cicli di clock della CPU relativi al programma}}{\text{Periodo del clock}}$$

$$\text{Tempo di CPU relativo a un programma} = \frac{\text{Cicli di clock della CPU relativi al programma}}{\text{Frequenza di clock}}$$

$$T = \frac{1}{\nu}$$

## Esercizio

Il nostro programma preferito viene eseguito in 10 secondi dal calcolatore A, che è dotato di un clock a 2 GHz. Stiamo cercando di aiutare un progettista a costruire un calcolatore B in grado di eseguire lo stesso programma in 6 secondi. Il progettista ha concluso che è possibile aumentare in modo significativo la frequenza di clock; questa modifica avrà però un'influenza su tutto il progetto della CPU, facendo sì che il calcolatore B richieda un numero di cicli di clock maggiore di un fattore 1,2 rispetto al calcolatore A per eseguire il programma. Dovendo dare un consiglio al progettista, quale sarà la frequenza di clock che permette di raggiungere l'obiettivo?

$$6 \text{ secondi} = 1,2 \times \#C_{\text{originale}} / x$$

$$\#C_{\text{originale}} = 10 * 2 \text{ GHz}$$

$$x = 1,2 * 10 * 2 \text{ GHz} / 6$$



# Prestazioni

## Esercizio

	CPI per ciascun tipo di istruzione		
	A	B	C
CPI	1	2	3

Sequenza di istruzioni	Numero di istruzioni in linguaggio macchina per ciascun tipo di istruzione		
	A	B	C
Sequenza 1	2	1	2
Sequenza 2	4	1	1

Quale sequenza richiede l'esecuzione di un maggior numero di istruzioni?  
Quale verrà eseguita più velocemente? Qual è il CPI delle due sequenze?

# Prestazioni

## Esercizio

	CPI per ciascun tipo di istruzione		
	A	B	C
CPI	1	2	3

Sequenza di istruzioni	Numero di istruzioni in linguaggio macchina per ciascun tipo di istruzione		
	A	B	C
Sequenza 1	2	1	2
Sequenza 2	4	1	1

Quale sequenza richiede l'esecuzione di un maggior numero di istruzioni?  
Quale verrà eseguita più velocemente? Qual è il CPI delle due sequenze?

→ Seq 2 richiede più istruzioni.

# Prestazioni

## Esercizio

✓ ~~media di colpi di clock per istruzione~~

	CPI per ciascun tipo di istruzione		
	A	B	C
CPI	1	2	3

Sequenza di istruzioni	Numero di istruzioni in linguaggio macchina per ciascun tipo di istruzione		
	A	B	C
Sequenza 1	$2 \times 1 +$	$1 \times 2 +$	$2 \times 3 \rightarrow$
Sequenza 2	4	1	1

Quale sequenza richiede l'esecuzione di un maggior numero di istruzioni?  
Quale verrà eseguita più velocemente? Qual è il CPI delle due sequenze?

→ Seq 2 richiede più istruzioni.

# Prestazioni

## Esercizio

✓ ~~media di colpi di clock per istruzione~~

	CPI per ciascun tipo di istruzione		
	A	B	C
CPI	1	2	3

Sequenza di istruzioni	Numero di istruzioni in linguaggio macchina per ciascun tipo di istruzione		
	A	B	C
Sequenza 1	$2 \times 1 +$	$1 \times 2 +$	$2 \times 3 \rightarrow$
Sequenza 2	$4 \times 1 +$	$1 \times 2 +$	$1 \times 3 \rightarrow 9$ ✓

Quale sequenza richiede l'esecuzione di un maggior numero di istruzioni?  
Quale verrà eseguita più velocemente? Qual è il CPI delle due sequenze?

→ Seq 2 richiede più istruzioni.

→ Seq 2 è più veloce :

# Prestazioni

## Esercizio

✓ ~~media di colpi di clock per istruzione~~

	CPI per ciascun tipo di istruzione		
	A	B	C
CPI	1	2	3

Sequenza di istruzioni	Numero di istruzioni in linguaggio macchina per ciascun tipo di istruzione		
	A	B	C
Sequenza 1	$2 \times 1 +$	$1 \times 2 +$	$2 \times 3 \rightarrow$
Sequenza 2	$4 \times 1 +$	$1 \times 2 +$	$1 \times 3 \rightarrow 9$ ✓

Quale sequenza richiede l'esecuzione di un maggior numero di istruzioni?  
Quale verrà eseguita più velocemente? Qual è il CPI delle due sequenze?

$$CPI_{seq} = \frac{\text{media clock}}{\text{istruzioni}}$$

$$CPI_{seq1} \Rightarrow \frac{10}{5} = 2$$

$$CPI_{seq2} \Rightarrow \frac{9}{6} = 1.5$$
 ✓

# Notazione posizionale

Formula generale per il calcolo del valore  $W$ :

$B$ : base

$n$ : numero di cifre (digits)

$b_i$ : valore della cifra  $i$ -esima (valore nominale)

$B^i$ : position weight

Esempi di notazione:

Sistema decimale (base 10) -> 161

Sistema binario (base 2) -> 10100001<sub>due</sub>



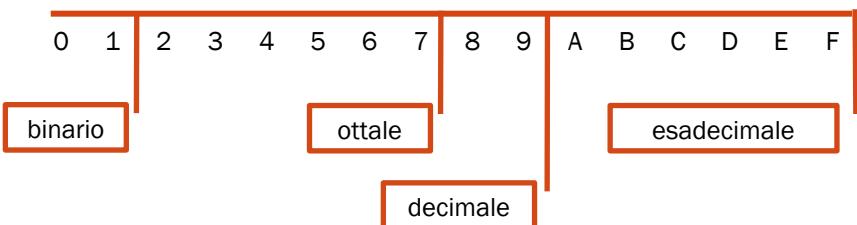
Sistema ottale (base 8) -> 241<sub>otto</sub>

Sistema esadecimale (base 16) -> A1<sub>esa</sub>

Spesso gli esadecimali sono identificati da prefisso 0x

e.g., 0xA1

$$W = \sum_{i=0}^{n-1} b_i \times B^i$$



# Conversione tra sistemi

---

Da binario a decimale:

$$1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 + 0 + 0 = 12$$

# Conversione tra sistemi

Da binario a decimale:

~~3210~~

$$1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 + 0 + 0 = 12$$

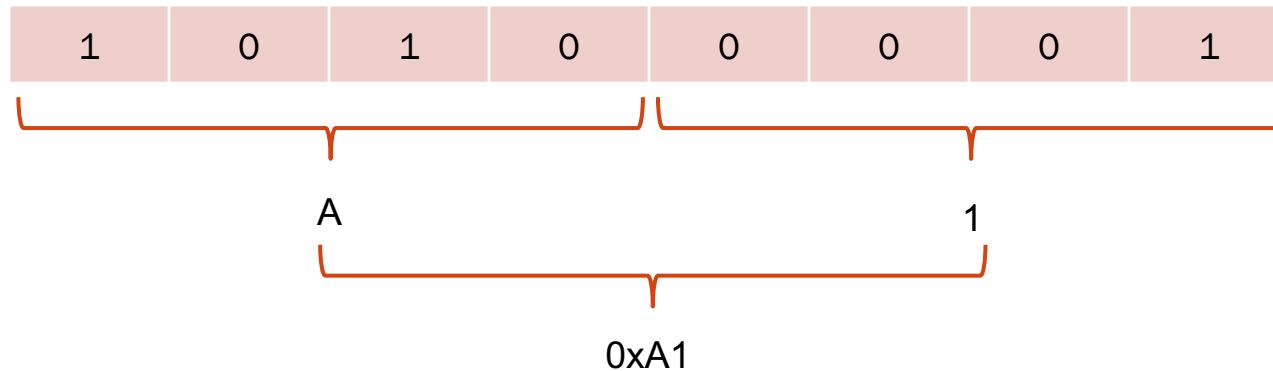
Da decimale ad esadecimale:

$$161 : 16 = 10 \quad \text{Resto: 1}$$

$$10 : 16 = 0 \quad \text{Resto: A}$$

Result: 0xA1

Da binario ad esadecimale:



## Conversione tra sistemi

---

Da binario a decimale:

~~3 2 1 0~~

$$1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 + 0 + 0 = 12$$

# Conversione tra sistemi

Da binario a decimale:

$$1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 + 0 + 0 = 12$$

Da decimale ad esadecimale:

$$x // y \rightarrow x = y \cdot q + r$$

divisione intera  
quoziente      resto

$$161 : 16 = 10 \underline{q} \quad \text{Resto: } 1 \underline{r}$$

$$10 : 16 = 0 \quad \text{Resto: } A$$

Result: 0xA1

# Conversione tra sistemi

Da binario a decimale:

$$1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 + 0 + 0 = 12$$

Da decimale ad esadecimale:

digit position  
// < 1)

$$\begin{array}{rcl} 161 & : & 16 \\ 10 & : & 16 \end{array} = \begin{array}{l} 10q \\ 0 \end{array}$$

$x // y \rightarrow x = y \cdot q + r$

divisione intera  
quoziente resto

Resto: 1 r  
Resto: A

Result: 0xA1

# Conversione tra sistemi

Da binario a decimale:

$$1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 + 0 + 0 = 12$$

Da decimale ad esadecimale:

digit position  
// < 1)

$$\begin{array}{r} 161 \\ : \quad 16 \\ \hline 10 \end{array} \quad \begin{array}{r} 10 \\ : \quad 16 \\ \hline 0 \end{array}$$

$$x // y \rightarrow x = y \cdot q + r$$

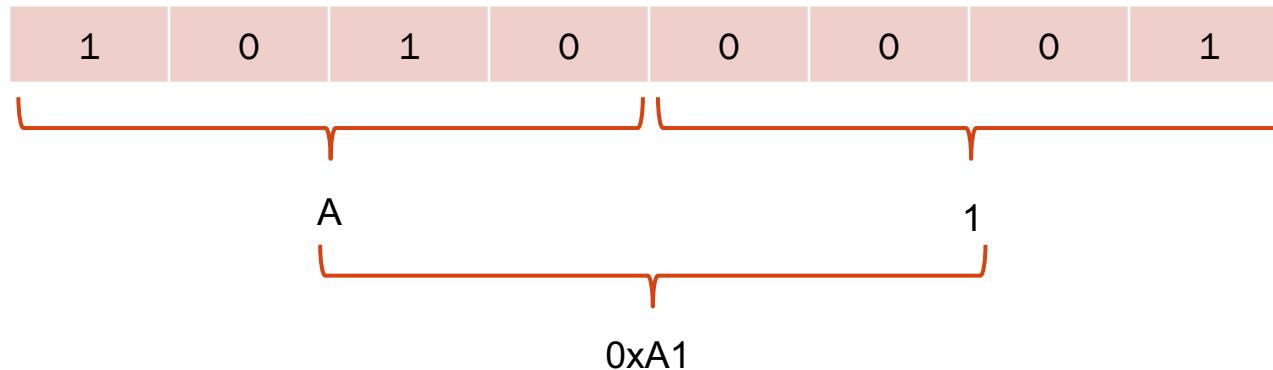
divisione intera  
quoziente resto

Resto: 1  
Resto: A

Result: 0xA1

$$161_{10} \rightarrow A1_{16}$$

Da binario ad esadecimale:



# Conversione tra sistemi

Da binario a decimale:

$$1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 + 0 + 0 = 12$$

Da decimale ad esadecimale:

digit position  
// < 1)

$$\begin{array}{r} 161 \\ \text{---} \\ 10 \end{array} : \begin{array}{r} 16 \\ \text{---} \\ 16 \end{array} = \begin{array}{r} 10 \\ \text{---} \\ 0 \end{array}$$

$$x // y \rightarrow x = y \cdot q + r$$

divisione intera  
quoziente resto

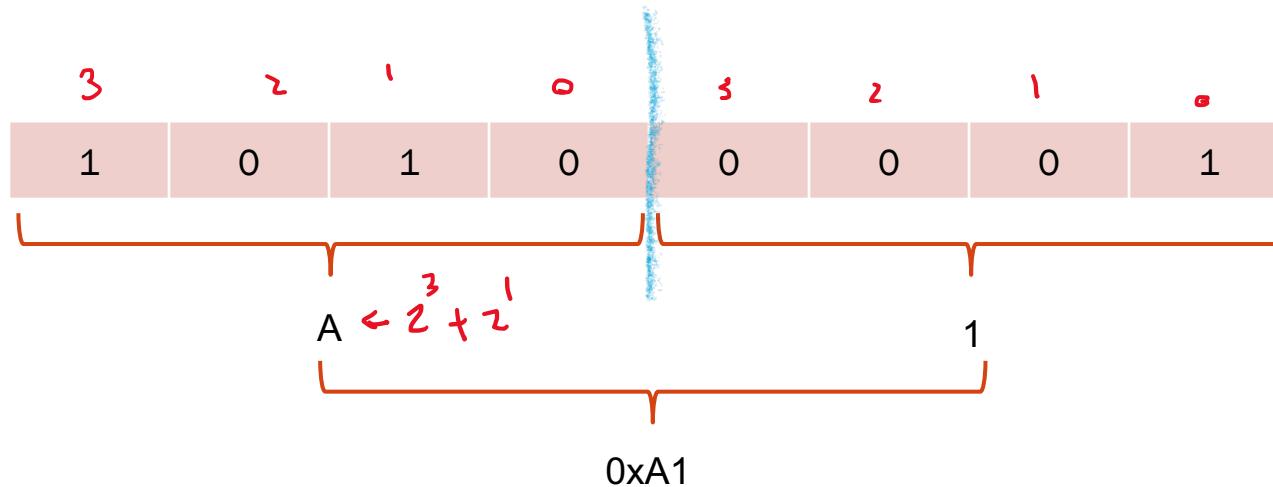
Resto: 1  
Resto: A

Result: 0xA1

Da binario ad esadecimale:

$$1100_2 \rightarrow 12_{10}$$

$$161_{10} \rightarrow A1_{16}$$

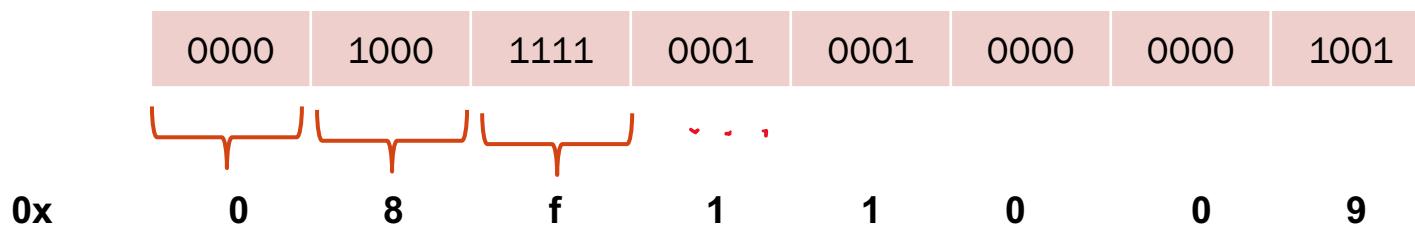


# Perché usare il sistema 0x (esadecimale)?

Le istruzioni del calcolatore sono codificate in lunghe e «noiose» stringhe di numeri binari cioè

- Calcolatori hanno dimensioni dati multipla di 4 (ex.  $32 \bmod 4 = 0$ )
- I numeri hex (base  $2^4=16$ ) sono convertibili in binario raggruppando cifre a 4. Ossia un valore **hex** è possibile esprimere in **bin** su 4 bit.

resto div.  
intero



Molto più facile leggere una rappresentazione compatta come 0x08f11009 che una lunga stringa di 0-1

# Conversione tra sistemi

Massimo valore esprimibile con un byte:

$$11111111_2 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255 = 2^8 - 1$$

$\xleftarrow[8\text{ bit}]{}$

Conversione di interi **senza segno**:

La parola (word) di default del MIPS consta di 32 bit

Si rappresentano 4 miliardi di numeri coh sol. 32 bit  
↳ 4G

Lunghezza	Valore min.	Valore max.
8 bit (1 byte)	0	$255 = 2^8 - 1$
16 bit (2 byte)	0	$65,535 = 2^{16} - 1$
32 bit (4 byte)	0	$4,294,967,295 = 2^{32} - 1$
64 bit (8 byte)	0	$18,446,744,073,709,551,615 = 2^{64} - 1$

# Interi con segno

---

- Modulo e segno
  - $1000\ 0000\ 0000\ 0000\ 0001\ 0000\ 0000_{\text{due}} = -256_{\text{dieci}}$ 
    - Intuitivo, ma dimezza il massimo valore esprimibile (inevitabile)
    - Ed ha due zeri!
- Complemento a due
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0000\ 0000_{\text{due}} = -256_{\text{dieci}}$ 
    - Meno intuitivo, ma molto pratico per somme e sottrazioni
- Complemento a uno
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 0000\ 0000_{\text{due}} = -256_{\text{dieci}}$ 
    - Intuitivo, ma crea problemi con le somme
    - Ed ha due zeri!
- Notazione polarizzata
  - $0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0000\ 0000_{\text{due}} = -256_{\text{dieci}}$ 
    - Poco intuitivo, ma con il medesimo range del complemento a due
- Cosa preferirebbe un architetto di elaboratori? Perché?

# Complemento a due nasce dall'hardware

La soluzione di default che l'hardware implementa naturalmente è il modo migliore per rappresentare i numeri interi negativi

0000	0000	0000	0000	0000	0000	0000	0000	0000	due	=	0	<sub>dec</sub>
0000	0000	0000	0000	0000	0000	0000	0000	0001	due	=	1	<sub>dec</sub>
0000	0000	0000	0000	0000	0000	0000	0000	0010	due	=	2	<sub>dec</sub>
...										...		
0111	1111	1111	1111	1111	1111	1111	1111	1101	due	=	2147483645	<sub>dec</sub>
0111	1111	1111	1111	1111	1111	1111	1111	1110	due	=	2147483646	<sub>dec</sub>
0111	1111	1111	1111	1111	1111	1111	1111	1111	due	=	2147483647	<sub>dec</sub>
1000	0000	0000	0000	0000	0000	0000	0000	0000	due	=	-2147483648	<sub>dec</sub>
1000	0000	0000	0000	0000	0000	0000	0000	0001	due	=	-2147483647	<sub>dec</sub>
1000	0000	0000	0000	0000	0000	0000	0000	0010	due	=	-2147483646	<sub>dec</sub>
...										...		
1111	1111	1111	1111	1111	1111	1111	1111	1101	due	=	-3	<sub>dec</sub>
1111	1111	1111	1111	1111	1111	1111	1111	1110	due	=	-2	<sub>dec</sub>
1111	1111	1111	1111	1111	1111	1111	1111	1111	due	=	-1	<sub>dec</sub>

## Complemento a due nasce dall'hardware

---

La soluzione di default che l'hardware implementa naturalmente è il modo migliore per rappresentare i numeri interi negativi. Ad esempio in una word (32 bit) abbiamo:

$$(x_{31} \times -2^{31}) + (x_{30} \times 2^{30}) + (x_{29} \times 2^{29}) + \dots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

ha il vantaggio che tutti i numeri negativi hanno il bit più significativo uguale a 1. Perciò l'hardware deve controllare soltanto questo bit per verificare se il numero è positivo o negativo (un numero che ha inizio con lo zero viene considerato positivo). Questo bit è a volte chiamato **bit di segno**.

Inoltre le operazioni di somma e sottrazione diventano banali (vedi dopo)

## Complemento a due, esempio

---

La soluzione di default che l'hardware implementa naturalmente è il modo migliore per rappresentare i numeri interi negativi. Ad esempio in una word (32 bit) abbiamo:

1111 1111 1111 1111 1111 1111 1111 1100<sub>due</sub>

Sostituendo il valore dei bit nella formula precedentemente introdotta si ha:

$$\begin{aligned} & (1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \dots + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^{-1}) \\ &= -2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 0 + 0 \\ &= -2\,147\,483\,648_{\text{dec}} + 2\,147\,483\,644_{\text{dec}} \\ &= -4_{\text{dec}} \end{aligned}$$

## Complemento a due, estensione del segno

Word a 32 bit in complemento a due quanto vale?

1111 1111 1111 1111 1111 1111 1111 1100<sub>due</sub>

Sostituendo il valore dei bit nella formula precedentemente introdotta si ha:

$$\begin{aligned} & (1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \dots + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^0) \\ &= -2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 0 + 0 \\ &= -2\,147\,483\,648_{\text{dec}} + 2\,147\,483\,644_{\text{dec}} \\ &= -4_{\text{dec}} \end{aligned}$$

Cosa conterrà un registro a 32 bit 1111 1100<sub>due</sub> se carichiamo dalla memoria il **byte** suddetto ?

## Complemento a due, estensione del segno

Word a 32 bit in complemento a due quanto vale?

1111 1111 1111 1111 1111 1111 1111 1100<sub>due</sub>

Sostituendo il valore dei bit nella formula precedentemente introdotta si ha:

$$\begin{aligned} & (1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \dots + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^0) \\ &= -2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 0 + 0 \\ &= -2\ 147\ 483\ 648_{\text{dec}} + 2\ 147\ 483\ 644_{\text{dec}} \\ &= -4_{\text{dec}} \end{aligned}$$

Cosa conterrà un registro a 32 bit 1111 1100<sub>due</sub> se carichiamo dalla memoria il byte suddetto ?

→ Dipende da come lo carichiamo

Se lo interpretiamo, lo carichiamo senza segno con load byte unsigned (lbu)

## Complemento a due, estensione del segno

Word a 32 bit in complemento a due quanto vale?

1111 1111 1111 1111 1111 1111 1111 1100<sub>due</sub>

Sostituendo il valore dei bit nella formula precedentemente introdotta si ha:

$$\begin{aligned} & (1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \dots + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^0) \\ &= -2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 0 + 0 \\ &= -2\ 147\ 483\ 648_{\text{dec}} + 2\ 147\ 483\ 644_{\text{dec}} \\ &= -4_{\text{dec}} \end{aligned}$$

Cosa conterrà un registro a 32 bit 1111 1100<sub>due</sub> se carichiamo dalla memoria il byte suddetto ?

→ Dipende da come lo carichiamo

Se lo interpretiamo, lo carichiamo senza segno con load byte unsigned (lbu)



## Complemento a due, estensione del segno

Word a 32 bit in complemento a due quanto vale?

1111 1111 1111 1111 1111 1111 1111 1100<sub>due</sub>

Sostituendo il valore dei bit nella formula precedentemente introdotta si ha:

$$\begin{aligned} & (1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \dots + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^0) \\ & = -2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 0 + 0 \\ & = -2\ 147\ 483\ 648_{dec} + 2\ 147\ 483\ 644_{dec} \\ & = -4_{dec} \end{aligned}$$

Cosa conterrà un registro a 32 bit 1111 1100<sub>due</sub> se carichiamo dalla memoria il byte suddetto ?

→ Dipende da come lo carichiamo

Se lo interpretiamo, lo carichiamo con segno con load byte(1b)



## Complemento a due, scorciatoia

---

$x + \bar{x} = -1$ , È la proprietà del complemento a due  
ossia  $\bar{x} + 1 = -x$ .

### Scorciatoia per cambiare di segno un numero

Si cambi di segno il numero  $2_{\text{dec}}$  e si verifichi poi il risultato cambiando di segno il numero  $-2_{\text{dec}}$ .

$$2_{\text{dec}} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{\text{due}}$$

Invertendo questo numero tramite l'inversione dei bit e sommando poi 1, si ottiene:

$$\begin{array}{r} 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{\text{due}} \\ + \qquad \qquad \qquad 1_{\text{due}} \\ \hline = \qquad 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{\text{due}} \\ = \qquad -2_{\text{dec}} \end{array}$$

## Complemento a due, somma e sottrazione

Basta eseguire la somma (ignorando l'overflow) e l'inversione di segno per rappresentare somme e sottrazioni tra numeri con segno!

0000	0000	0000	0000	0000	0000	0000	0000	0000	due	=	0	<sub>dec</sub>
0000	0000	0000	0000	0000	0000	0000	0000	0001	due	=	1	<sub>dec</sub>
0000	0000	0000	0000	0000	0000	0000	0000	0010	due	=	2	<sub>dec</sub>
...										...		
0111	1111	1111	1111	1111	1111	1111	1111	1101	due	=	2147483645	<sub>dec</sub>
0111	1111	1111	1111	1111	1111	1111	1111	1110	due	=	2147483646	<sub>dec</sub>
0111	1111	1111	1111	1111	1111	1111	1111	1111	due	=	2147483647	<sub>dec</sub>
1000	0000	0000	0000	0000	0000	0000	0000	0000	due	=	-2147483648	<sub>dec</sub>
1000	0000	0000	0000	0000	0000	0000	0000	0001	due	=	-2147483647	<sub>dec</sub>
1000	0000	0000	0000	0000	0000	0000	0000	0010	due	=	-2147483646	<sub>dec</sub>
...										...		
1111	1111	1111	1111	1111	1111	1111	1111	1101	due	=	-3	<sub>dec</sub>
1111	1111	1111	1111	1111	1111	1111	1111	1110	due	=	-2	<sub>dec</sub>
1111	1111	1111	1111	1111	1111	1111	1111	1111	due	=	-1	<sub>dec</sub>

# Breve cenno ai numeri reali: standard IEEE754

Come rappresentiamo

$3,14159265\dots$  <sub>dec</sub> ( $\pi$ )

$2,71828\dots$  <sub>dec</sub> ( $e$ )

$0,000000001$  <sub>dec</sub> o  $1,0$  <sub>dec</sub>  $\times 10^{-9}$  (numero di secondi in un nanosecondo)

$3\,155\,760\,000$  <sub>dec</sub> o  $3,15576$  <sub>dec</sub>  $\times 10^9$  (numero di secondi in un secolo medio)

Notazione Scientifica Normalizzata

$(-1)^S \times F \times 2^E$

segno      mantissa      esponente

Singola precisione, float (32bit)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	esponente							mantissa																							

1 bit

8 bit  $\rightarrow$  7 bit + 1 d' segno  
 $2^7 \rightarrow 128$

23 bit

$2^{E=128} \rightarrow$  range  $\in (10^{-38}, 10^{38})$

Come ripartiamo i 32 bit in esponente e mantissa? Compromesso fra precisione del numero (mantissa F) e intervallo di valori (esponente)

# Breve cenno ai numeri reali: standard IEEE754

Come rappresentiamo

$3,14159265\dots$  <sub>dec</sub> ( $\pi$ )

$2,71828\dots$  <sub>dec</sub> ( $e$ )

$0,000000001$  <sub>dec</sub> o  $1,0$  <sub>dec</sub>  $\times 10^{-9}$  (numero di secondi in un nanosecondo)

$3\ 155\ 760\ 000$  <sub>dec</sub> o  $3,15576$  <sub>dec</sub>  $\times 10^9$  (numero di secondi in un secolo medio)

Notazione Scientifica Normalizzata

$(-1)^S \times F \times 2^E$

segno      mantissa      esponente

Doppia precisione, double (64bit)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	esponente											mantissa																			
1 bit	11 bit											20 bit																			
mantissa (continuazione)																															

$$\text{Range} \in \left( -10^{-308}, 10^{308} \right)$$

---

# Le istruzioni del MIPS



SAPIENZA  
UNIVERSITÀ DI ROMA



## Architettura CISC

### (Complex Instruction Set Computer)

#### - Istruzioni di dimensione variabile

Per il fetch della successiva è **necessaria la decodifica**

#### - Formato variabile

Decodifica complessa

#### - Operandi in memoria

Molti accessi alla memoria per istruzione

#### - Pochi registri interni

Maggior numero di accessi in memoria

#### - Modi di indirizzamento complessi

Maggior numero di accessi in memoria

Durata variabile della istruzione

Conflitti tra istruzioni più complicati

#### - Istruz. Complesse: pipeline più complicata

## Architettura RISC

### (Reduced Instruction Set Computer)

#### - Istruzioni di dimensione fissa

Fetch della successiva **senza decodifica della prec.**

#### - Istruzioni di formato uniforme

Per semplificare la fase di decodifica

#### - Operazioni ALU solo tra registri

Senza accesso a memoria

#### - Molti registri interni

Per i risultati parziali senza accessi alla memoria

#### - Modi di indirizzamento semplici

Con spiazzamento, 1 solo accesso a memoria

Durata fissa della istruzione

Conflitti semplici

#### - Istruz. semplici => pipeline più veloce

**Istruz. complesse => molte più istruz. semplici ... MA: girano più velocemente !!!!**

# MIPS – Ingredienti

## Microprocessor without Interlocked Pipelined Stages

### Operandi MIPS

Nome	Esempio	Commenti
32 registri	\$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at	Accesso veloce ai dati. Nel MIPS gli operandi devono essere contenuti nei registri per potere eseguire delle operazioni. Il registro \$zero contiene sempre il valore 0, e il registro \$at viene riservato all'assemblatore per la gestione di costanti molto lunghe.
$2^{30}$ parole di memoria	Memoria[0], Memoria[4], .... Memoria[4294967292]	Alla memoria si accede solamente attraverso le istruzioni di trasferimento dati. Il MIPS utilizza l'indirizzamento al byte, perciò due parole consecutive hanno indirizzi in memoria a una distanza di 4. La memoria consente di memorizzare strutture dati, vettori, o il contenuto dei registri.

#### Importante:

- Memoria **MIPS è indicizzata al byte**
- Se sono **all'indirizzo t** e devo leggere la parola successiva incremento indirizzo come **t+4**, questo perché una parola sono 4 byte ossia 32 bit (1 byte = 8 bit)

# MIPS – Ingredienti

---

## Linguaggio assembler MIPS

Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Aritmetiche	Somma	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	Operandi in tre registri
	Sottrazione	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	Operandi in tre registri
	Somma immediata	addi \$s1,\$s2,20	$\$s1 = \$s2 + 20$	Utilizzata per sommare delle costanti

# MIPS – Ingredienti

---

## Linguaggio assembler MIPS

Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Aritmetiche	Somma	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	Operandi in tre registri
	Sottrazione	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	Operandi in tre registri
	Somma immediata	addi \$s1,\$s2,20	$\$s1 = \$s2 + 20$	Utilizzata per sommare delle costanti
Trasferimento dati	Lettura parola	lw \$s1,20(\$s2)	$\$s1 = \text{Memoria}[\$s2+20]$	Trasferimento di una parola da memoria a registro
	Memorizzazione parola	sw \$s1,20(\$s2)	$\text{Memoria}[\$s2+20] = \$s1$	Trasferimento di una parola da registro a memoria
	Lettura mezza parola	lh \$s1,20(\$s2)	$\$s1 = \text{Memoria}[\$s2+20]$	Trasferimento di una mezza parola da memoria a registro
	Lettura mezza parola, senza segno	lhu \$s1,20(\$s2)	$\$s1 = \text{Memoria}[\$s2+20]$	Trasferimento di una mezza parola da memoria a registro
	Memorizzazione mezza parola	sh \$s1,20(\$s2)	$\text{Memoria}[\$s2+20] = \$s1$	Trasferimento di una mezza parola da registro a memoria
	Lettura byte	lb \$s1,20(\$s2)	$\$s1 = \text{Memoria}[\$s2+20]$	Trasferimento di un byte da memoria a registro
	Lettura byte senza segno	lbu \$s1,20(\$s2)	$\$s1 = \text{Memoria}[\$s2+20]$	Trasferimento di un byte da memoria a registro
	Memorizzazione byte	sb \$s1,20(\$s2)	$\text{Memoria}[\$s2+20] = \$s1$	Trasferimento di un byte da registro a memoria
	Lettura di una parola e blocco	l1 \$s1,20(\$s2)	$\$s1 = \text{Memoria}[\$s2+20]$	Caricamento di una parola come prima fase di un'operazione atomica
	Memorizzazione condizionata di una parola	sc \$s1,20(\$s2)	$\text{Memoria}[\$s2+20] = \$s1; \$s1 = 0 \text{ oppure } 1$	Memorizzazione di una parola come seconda fase di un'operazione atomica
	Caricamento costante nella mezza parola superiore	lui \$s1,20	$\$s1 = 20 * 2^{16}$	Caricamento di una costante nei 16 bit più significativi

# MIPS – Ingredienti

## Linguaggio assembler MIPS

Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Aritmetiche	Somma	add \$s1,\$s2,\$s3	$s1 = s2 + s3$	Operandi in tre registri
	Sottrazione	sub \$s1,\$s2,\$s3	$s1 = s2 - s3$	Operandi in tre registri
	Somma immediata	addi \$s1,\$s2,20	$s1 = s2 + 20$	Utilizzata per sommare delle costanti
Trasferimento dati	Lettura parola	lw \$s1,20(\$s2)	$s1 = \text{Memoria}[s2+20]$	Trasferimento di una parola da memoria a registro
	Memorizzazione parola	sw \$s1,20(\$s2)	$\text{Memoria}[s2+20] = s1$	Trasferimento di una parola da registro a memoria
	Lettura mezza parola	lh \$s1,20(\$s2)	$s1 = \text{Memoria}[s2+20]$	Trasferimento di una mezza parola da memoria a registro
	Lettura mezza parola, senza segno	lhu \$s1,20(\$s2)	$s1 = \text{Memoria}[s2+20]$	Trasferimento di una mezza parola da memoria a registro
	Memorizzazione mezza parola	sh \$s1,20(\$s2)	$\text{Memoria}[s2+20] = s1$	Trasferimento di una mezza parola da registro a memoria
	Lettura byte	lb \$s1,20(\$s2)	$s1 = \text{Memoria}[s2+20]$	Trasferimento di un byte da memoria a registro
	Lettura byte senza segno	lbu \$s1,20(\$s2)	$s1 = \text{Memoria}[s2+20]$	Trasferimento di un byte da memoria a registro
	Memorizzazione byte	sb \$s1,20(\$s2)	$\text{Memoria}[s2+20] = s1$	Trasferimento di un byte da registro a memoria
	Lettura di una parola e blocco	l1 \$s1,20(\$s2)	$s1 = \text{Memoria}[s2+20]$	Caricamento di una parola come prima fase di un'operazione atomica
	Memorizzazione condizionata di una parola	sc \$s1,20(\$s2)	$\text{Memoria}[s2+20] = s1; s1 = 0$ oppure 1	Memorizzazione di una parola come seconda fase di un'operazione atomica
	Caricamento costante nella mezza parola superiore	lui \$s1,20	$s1 = 20 * 2^{16}$	Caricamento di una costante nei 16 bit più significativi
Logiche	And	and \$s1,\$s2,\$s3	$s1 = s2 \& s3$	Operandi in tre registri; AND bit a bit
	Or	or \$s1,\$s2,\$s3	$s1 = s2   s3$	Operandi in tre registri; OR bit a bit
	Nor	nor \$s1,\$s2,\$s3	$s1 = \sim(s2   s3)$	Operandi in tre registri; NOR bit a bit

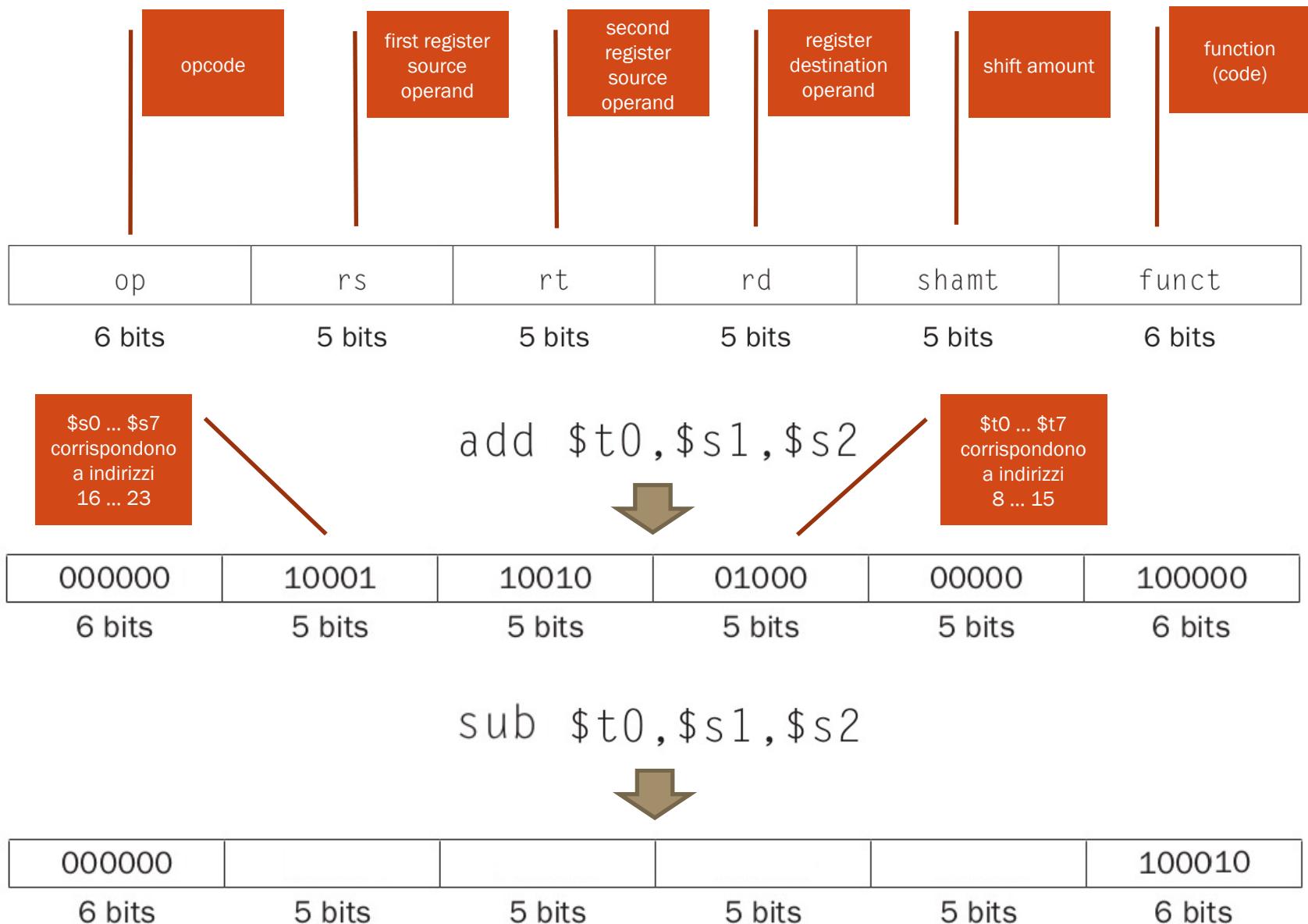
# MIPS – Ingredienti

Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Logiche (segue)	And immediato	andi \$s1,\$s2,20	$\$s1 = \$s2 \& 20$	And bit a bit tra un operando in registro e una costante
	Or immediato	ori \$s1,\$s2,20	$\$s1 = \$s2   20$	OR bit a bit tra un operando in registro e una costante
	Scorrimento logico a sinistra	sll \$s1,\$s2,10	$\$s1 = \$s2 \ll 10$	Spostamento a sinistra del numero di bit specificato dalla costante
	Scorrimento logico a destra	srl \$s1,\$s2,10	$\$s1 = \$s2 \gg 10$	Spostamento a destra del numero di bit specificato dalla costante

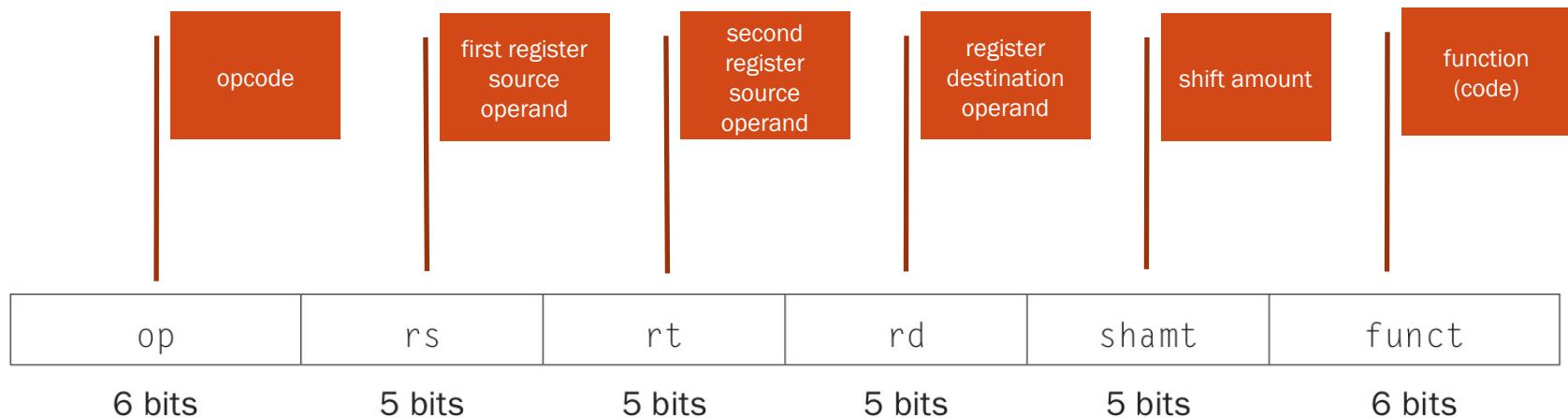
# MIPS – Ingredienti

Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Logiche (segue)	And immediato	andi \$s1,\$s2,20	$\$s1 = \$s2 \& 20$	And bit a bit tra un operando in registro e una costante
	Or immediato	ori \$s1,\$s2,20	$\$s1 = \$s2   20$	OR bit a bit tra un operando in registro e una costante
	Scorrimento logico a sinistra	sll \$s1,\$s2,10	$\$s1 = \$s2 \ll 10$	Spostamento a sinistra del numero di bit specificato dalla costante
	Scorrimento logico a destra	srl \$s1,\$s2,10	$\$s1 = \$s2 \gg 10$	Spostamento a destra del numero di bit specificato dalla costante
Salti condizionati	Salta se uguale	beq \$s1,\$s2,25	Se ( $\$s1 == \$s2$ ) vai a PC+4+100	Test di uguaglianza; salto relativo al PC
	Salta se non è uguale	bne \$s1,\$s2,25	Se ( $\$s1 != \$s2$ ) vai a PC+4+100	Test di disuguaglianza; salto relativo al PC
	Poni uguale a 1 se minore	slt \$s1,\$s2,\$s3	Se ( $\$s2 < \$s3$ ) $\$s1 = 1$ ; altrimenti $\$s1 = 0$	Comparazione di minoranza; utilizzata con bne e beq
	Poni uguale a uno se minore, numeri senza segno	sltu \$s1,\$s2,\$s3	Se ( $\$s2 < \$s3$ ) $\$s1 = 1$ ; altrimenti $\$s1 = 0$	Comparazione di minoranza su numeri senza segno
	Poni uguale a uno se minore, immediato	slti \$s1,\$s2,20	Se ( $\$s2 < 20$ ) $\$s1 = 1$ ; altrimenti $\$s1 = 0$	Comparazione di minoranza con una costante
	Poni uguale a uno se minore, immediato e senza segno	sltiu \$s1,\$s2,20	Se ( $\$s2 < 20$ ) $\$s1 = 1$ ; altrimenti $\$s1 = 0$	Comparazione di minoranza con una costante, con numeri senza segno
	Salto incondizionato	j 2500	Vai a 10000	Salto all'indirizzo della costante
Salti incondizionati	Salto indiretto	jr \$ra	Vai all'indirizzo contenuto in \$ra	Salto all'indirizzo contenuto nel registro, utilizzato per il ritorno da procedura e per i costrutti switch
	Salta e collega	jal 2500	$\$ra = PC + 4$ ; vai a 10000	Chiamata a procedura

# Rappresentazione dell'istruzione (R-type)



# Rappresentazione dell'istruzione (R-type)



\$s0 ... \$s7  
corrispondono  
a indirizzi  
16 ... 23

add \$t0,\$s1,\$s2

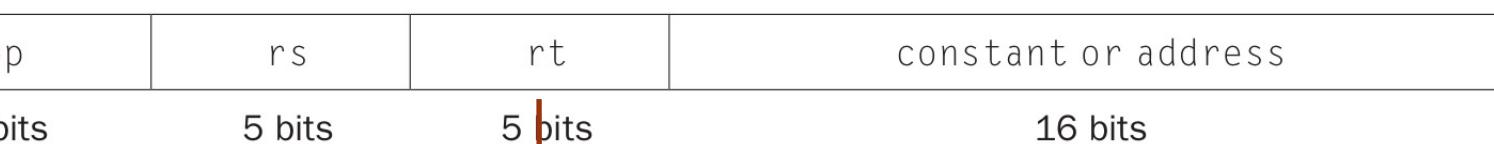
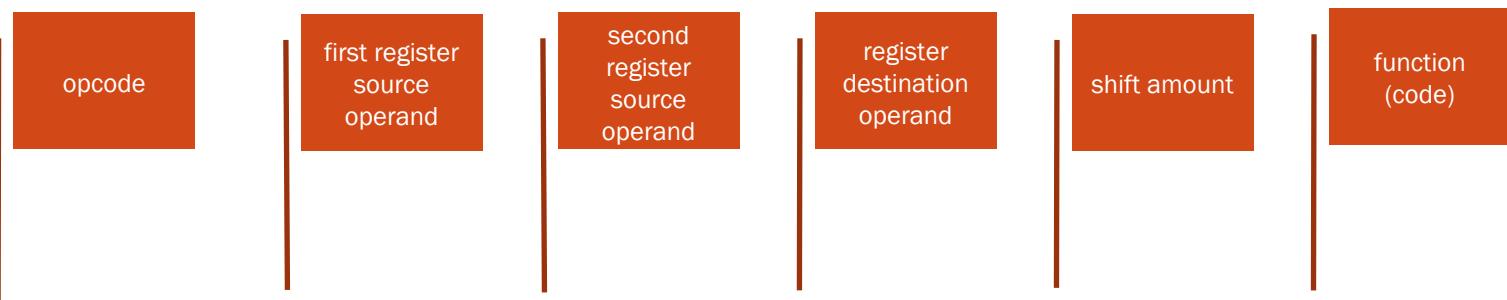
\$t0 ... \$t7  
corrispondono  
a indirizzi  
8 ... 15

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

sub \$t0,\$s1,\$s2

000000	10001	10010	01000	00000	100010
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

# Rappresentazione dell'istruzione (I-type)

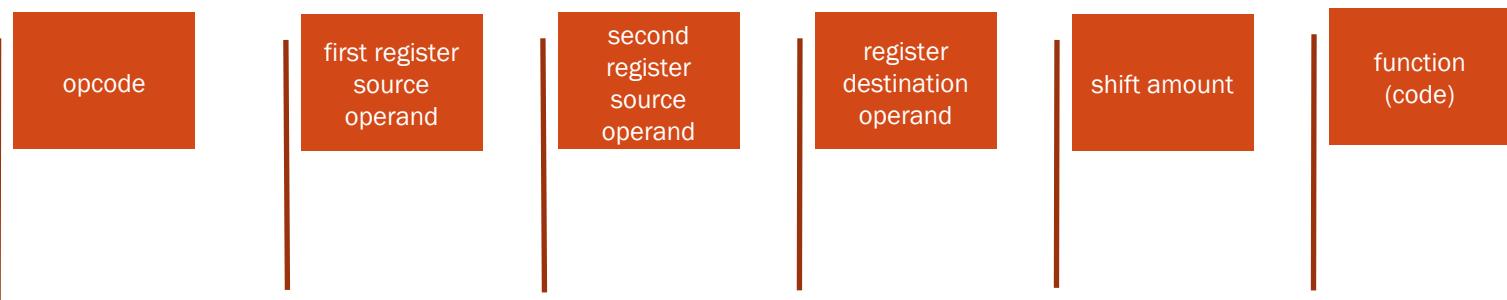


target  
register

addi \$t2,\$s2,4



# Rappresentazione dell'istruzione (I-type)

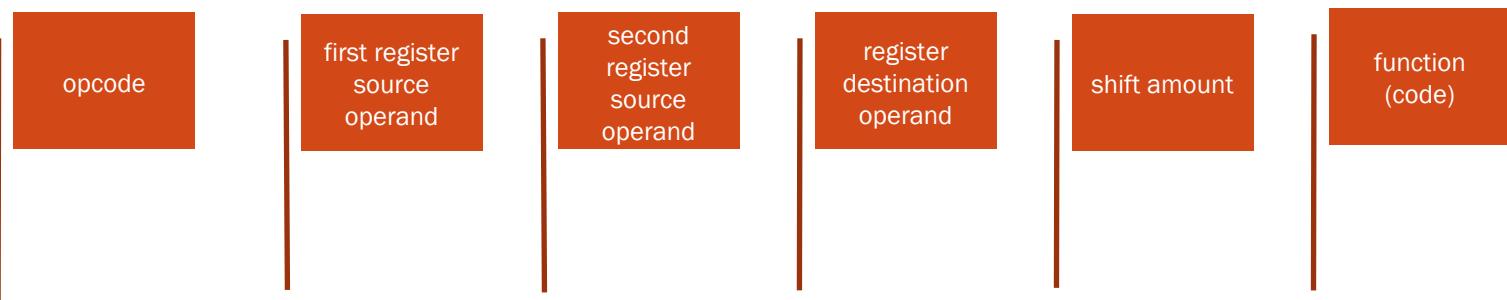


addi \$t2,\$s2,4

0x...?

00 1000 | 0 1010 | 1 1010 | 0000 0000 0000 0100

# Rappresentazione dell'istruzione (I-type)



6 bits      5 bits      5 bits      5 bits      5 bits      6 bits

6 bits      5 bits      5 bits      16 bits

target  
register

addi \$t2,\$s2,4

0x215A0004

00 1000 | 0 1010 | 1 1010 | 0000 0000 0000 0100

# Formato istruzioni

---

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add \$s1,\$s2,\$s3
sub	R	0	18	19	17	0	34	sub \$s1,\$s2,\$s3
addi	I	8	18	17	100			addi \$s1,\$s2,100
lw	I	35	18	17	100			lw \$s1,100(\$s2)
sw	I	43	18	17	100			sw \$s1,100(\$s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

# Formato delle istruzioni MIPS

Istruzioni della CPU tutte da 32 bit con formato molto simile

R-type: (tipo a Registro)

- SENZA accesso alla memoria
- Istruzioni Aritmetico/Logiche

Op 6 bit	Rs 5bit	Rt 5bit	Rd 5bit	Shamt 5 bit	Func 6 bit
-------------	------------	------------	------------	----------------	---------------

Esempio: **add \$t0, \$t1, \$t2**

$\$t0 \leftarrow \$t1 + \$t2$

**sll \$t0, \$t1, 5**

$\$t0 \leftarrow \$t1 \ll 5$

I-type: (tipo Immediato)

- Load/Store
- Salti condizionati (salto relativo al PC)

Op 6 bit	Rs 5bit	Rt/d 5bit	Immediate 16 bit
-------------	------------	--------------	---------------------

Esempi: **lw \$t1, vettore(\$t0)**

$\$t1 \leftarrow \text{MEM}[\text{vettore} + \$t0]$

**beq \$t0, \$t1, offset**

$\text{PC} \leftarrow \text{PC} + 4 + \text{offset} * 4$  SE  $\$t0 == \$t1$

J-type: (tipo Jump)

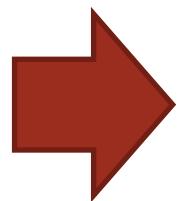
- Salti NON condizionati (salto assoluto)

Op 6 bit	Address 26 bit
-------------	-------------------

Esempio: **j destinazione**

$\text{PC} \leftarrow \text{destinazione} * 4$

# Per la prossima lezione: MARS



The screenshot shows a web browser window titled "MARS MIPS simulator - Missouri State University - Chrome...". The URL in the address bar is "courses.missouristate.edu/Ken...". The page content includes the Missouri State University logo (a building icon and the text "Missouri State UNIVERSITY"), a search bar with the text "Search" and a dropdown menu showing letters "a b c d e f g h i j k l m n o p q r s t u v w x y z", a large image of the planet Mars, and a sidebar with links: Home, Features, Download, License, Papers, Help & Info, and Contact Us.

[courses.missouristate.edu/kenvollmar/mars/download.htm](http://courses.missouristate.edu/kenvollmar/mars/download.htm)