



Università degli Studi di Milano
CdL in Informatica per la comunicazione digitale
AA 2025/2026

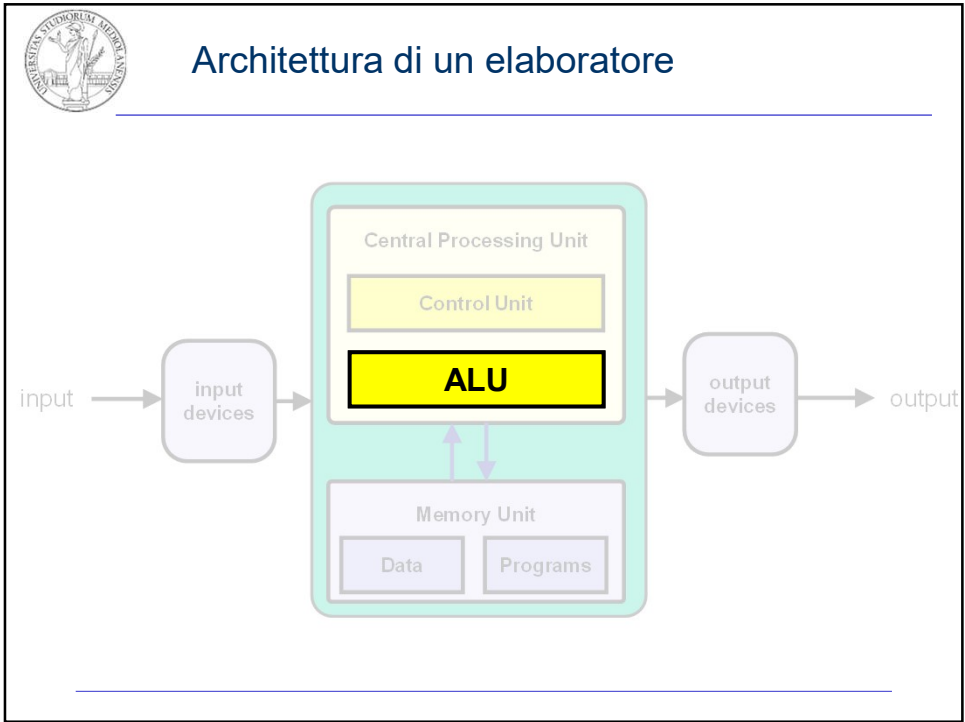
Architettura degli elaboratori - Lez 7:

ALU

l'Unità Aritmetica Logica

Marco Tarini
marco.tarini@unimi.it

1



2

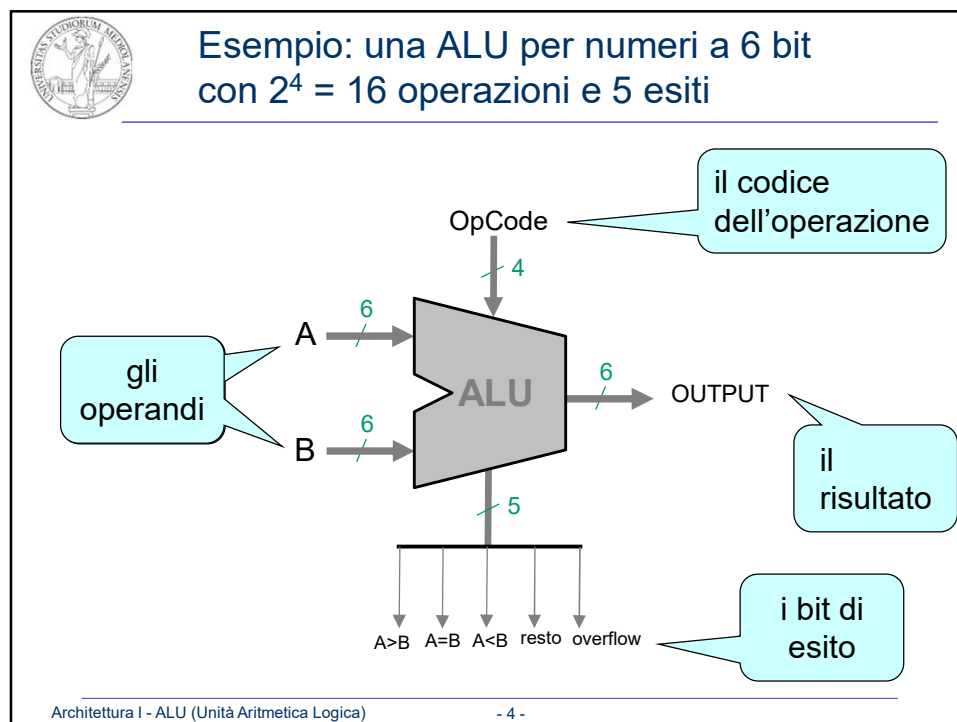


Unità Aritmetico-Logica (ALU)

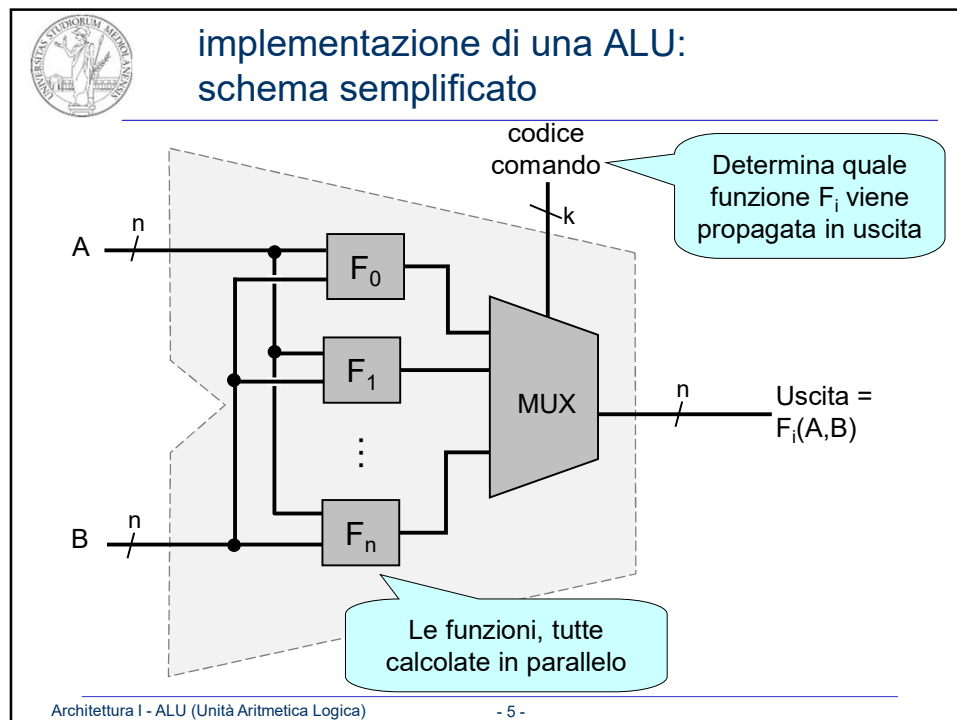
- Parte dell'elaboratore che computa le operazioni logiche e matematiche
- È un circuito multifunzione
 - ▶ È capace di svolgere diverse funzioni, a comando!
- Un comando (opportunamente codificato) seleziona quale delle diverse funzioni deve essere prodotta in uscita
- Le funzioni sono di tipo logico e matematico (dove il nome)
- Input:
 - ▶ almeno due (su alcune ALU, tre) **operandi**, di n bit ciascuno
 - ▶ un **codice comando**, che identifica quale operazione applicare (di k bit, se ho al più 2^k comandi diversi fra cui scegliere)
- Output
 - ▶ il **risultato** dell'operazione richiesta (ancora di n bit)
 - ▶ alcuni **bit di esito** (esempio: «si è verificato overflow?»)
- Non tutti gli ingressi e le uscite sono rilevanti per tutte le operazioni

Architettura I - ALU (Unità Aritmetica Logica) - 3 -

3



4



5



Implementazione di una ALU

- Lo schema precedente è molto semplificato
- In realtà, come abbiamo visto nell'esempio «somma oppure sottrazione» della volta scorsa, molte operazioni possono usufruire di blocchi funzionali e altre parte in comune, per ottimizzare
- Inoltre, le varie operazioni, oltre che a dei risultati, producono anche un certo numero di **bit di esito**, ciascuno dei quali riportano un informazione SI/NO come
 - C'è stato overflow? (se è una operazione che può generarlo)
 - I due numeri sono uguali fra loro, maggiori, minori, diversi...? (per le operazioni di comparazione)
 - E altro
- Anche i bit di esito vengono selezionati da un multiplexer controllato dal codice comando

Architettura I - ALU (Unità Aritmetica Logica) - 6 -

6



Esempio di possibile tabella delle funzioni di una semplice ALU per interi

Codice	Comando	Uscita	Esiti
0	Addizione	A + B	Overflow?
1	Sottrazione	A – B	Overflow?
2	Pass A	A	-
3	Max	Max(A , B)	-
4	Prodotto	A × B	Overflow?
5	Divisione intera	A / B	DivZero?
6	Resto (Modulo)	A % B (A mod B)	DivZero?
7	Compara	-	A=B? A<B? A>B?
8	Valore assoluto	A	Overflow?
9	Shift Left	A << 1 (cioè 2A)	Overflow?
10	Shift Right (aritmetico)	A >> 1 (cioè ½ A)	Resto?
11	Shift Right (logico)	A >> 1 in segno	Resto?
12	And	A ∧ B bit a bit	-
13	Or	A ∨ B bit a bit	-
14	Not	¬A bit a bit	-
15	Azzera	0	-

7

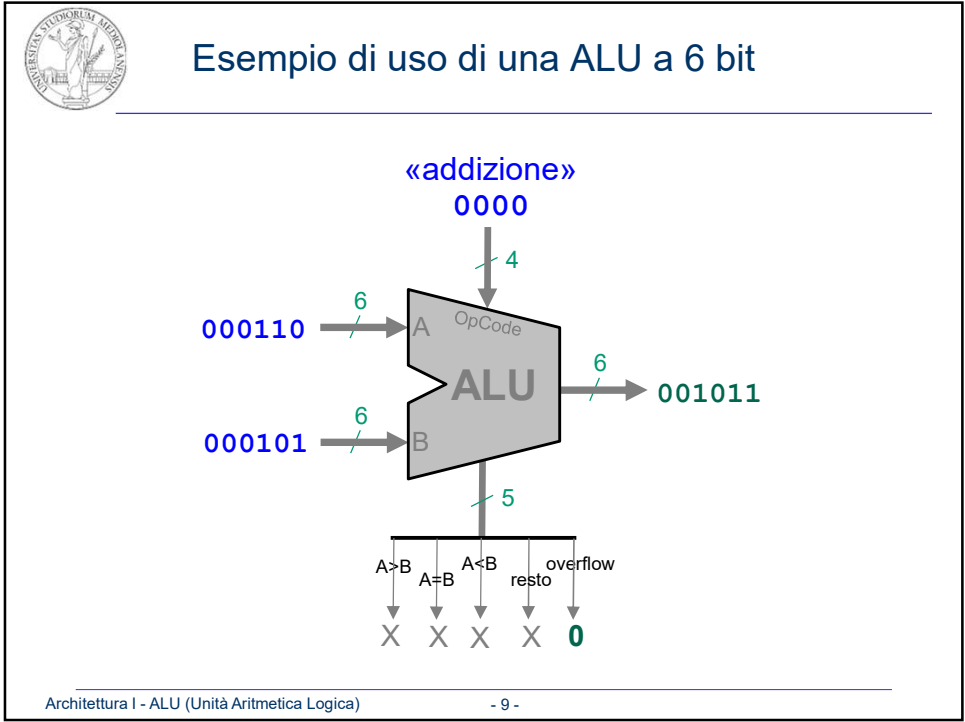


Esempio di possibile tabella delle funzioni di una semplice ALU per interi : note

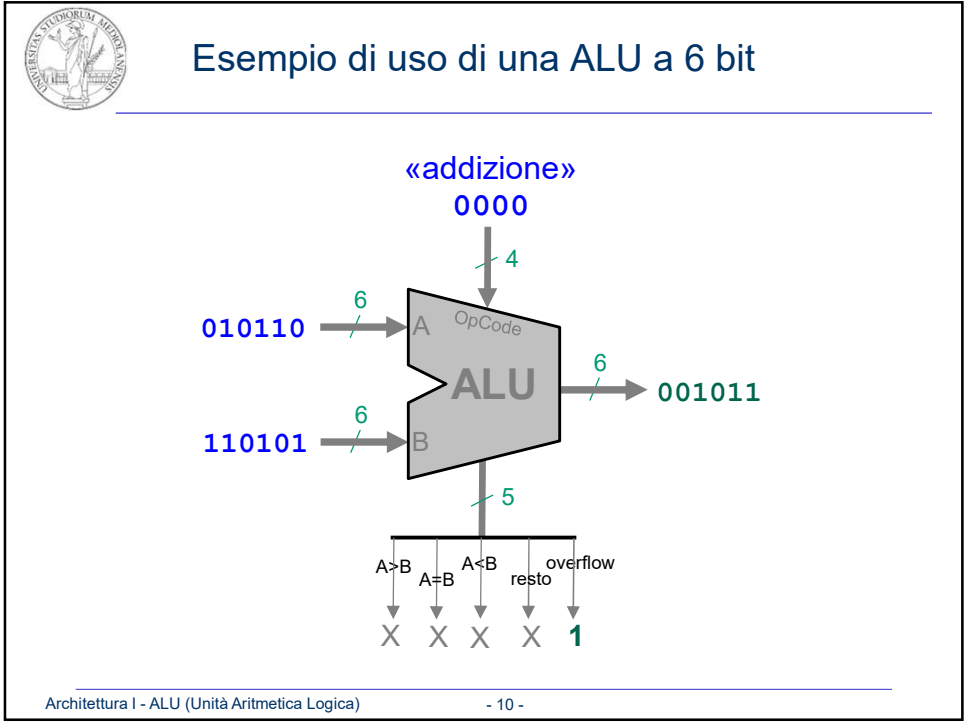
- Alcuni comandi sono molto semplici, ma come vedremo sono utili:
 - Pass A: l'output è direttamente A (B viene ignorato)
 - Azzera: l'output è sempre zero (sia A che B vengono ignorati)
- Alcune operazioni sono aritmetiche, altre logiche
 - Le operazioni logiche, come AND, effettuano l'AND bit a bit
 - Cioè ogni bit dell'output è un AND dei due corrispondenti bit dell'input
- Alcune operazioni producono solo esiti
- Lo shift a destra si differenzia fra aritmetico e logico
 - Logico: i bit inseriti da sinistra sono 0: 100111 >> 1 = 010011
 - Aritmetico: i bit inseriti da sinistra sono il MSB (quello sottolineato) 100111 >> 1 = 110011
questo dimezza (arrotondando *per difetto*) i numeri in CP2 (verifica)
- In questa ALU, le operazioni aritmetiche sono per numeri senza segno
 - Una ALU può avere operazioni distinte (per es, Compara, Add) per numeri in CP2 e senza segno

- 8 -

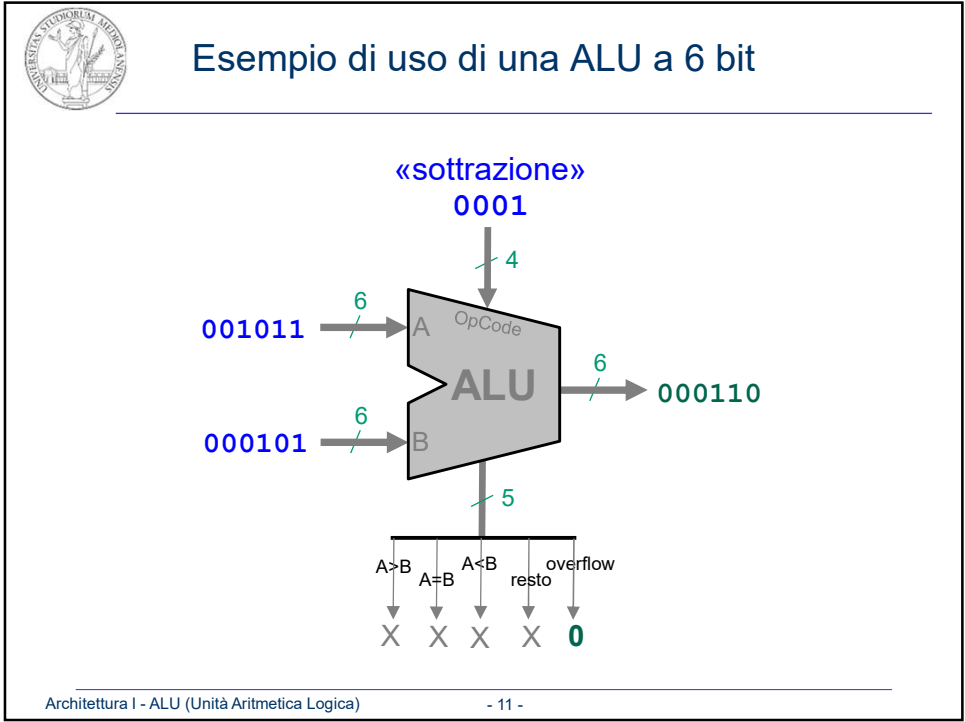
8



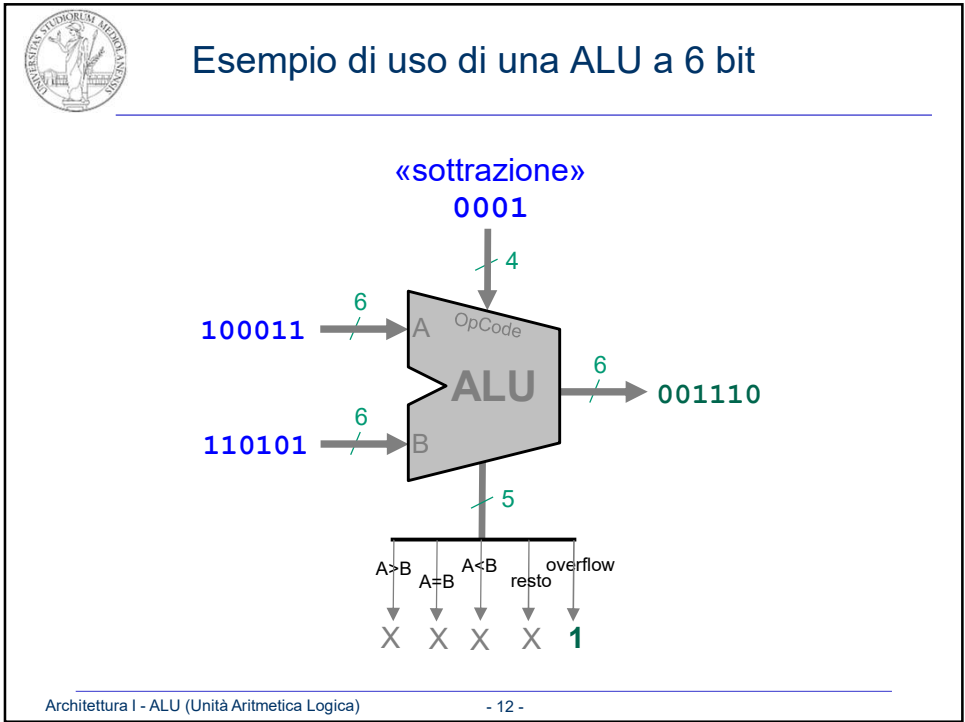
9



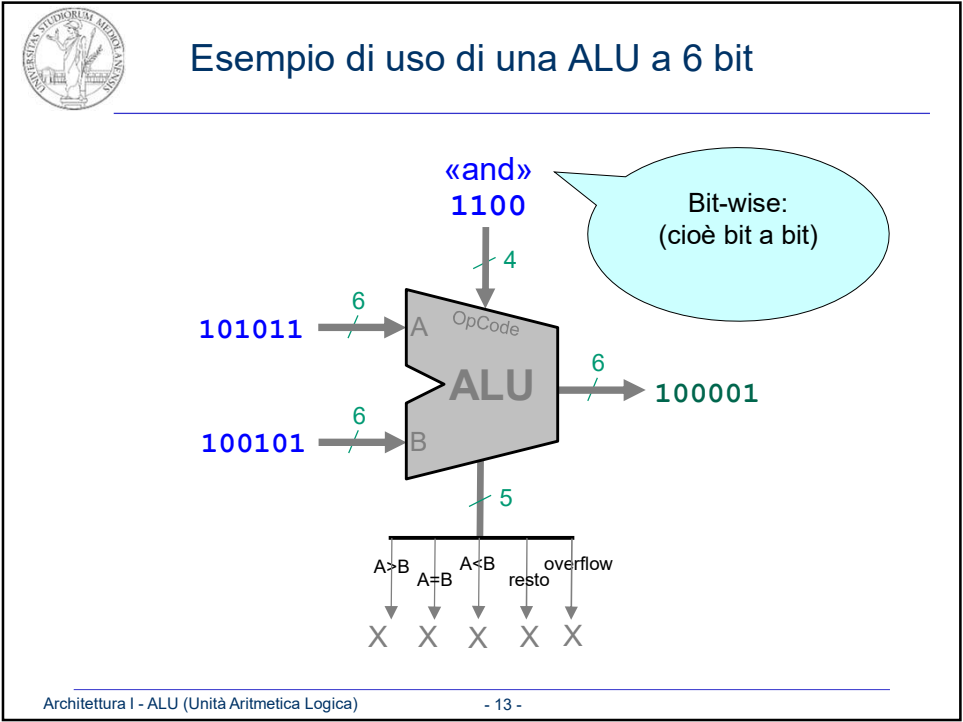
10



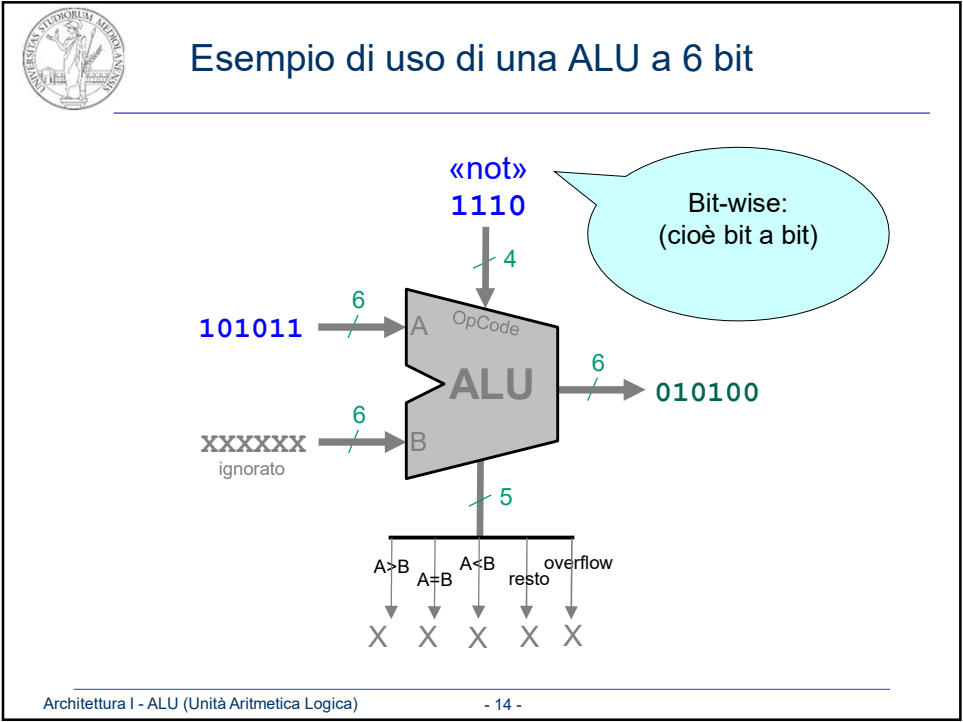
11



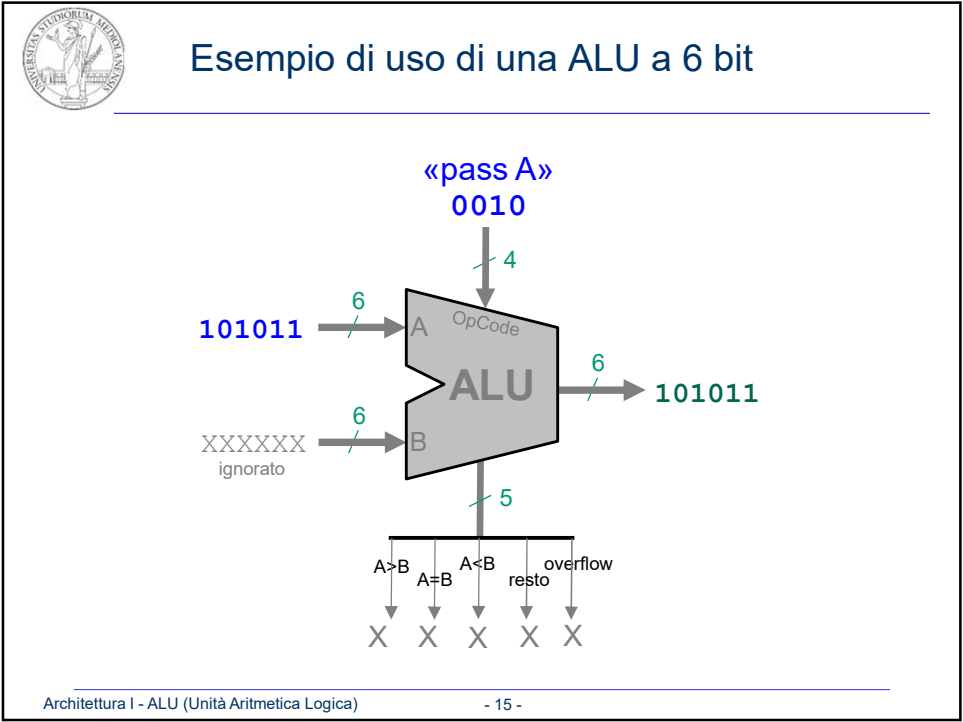
12



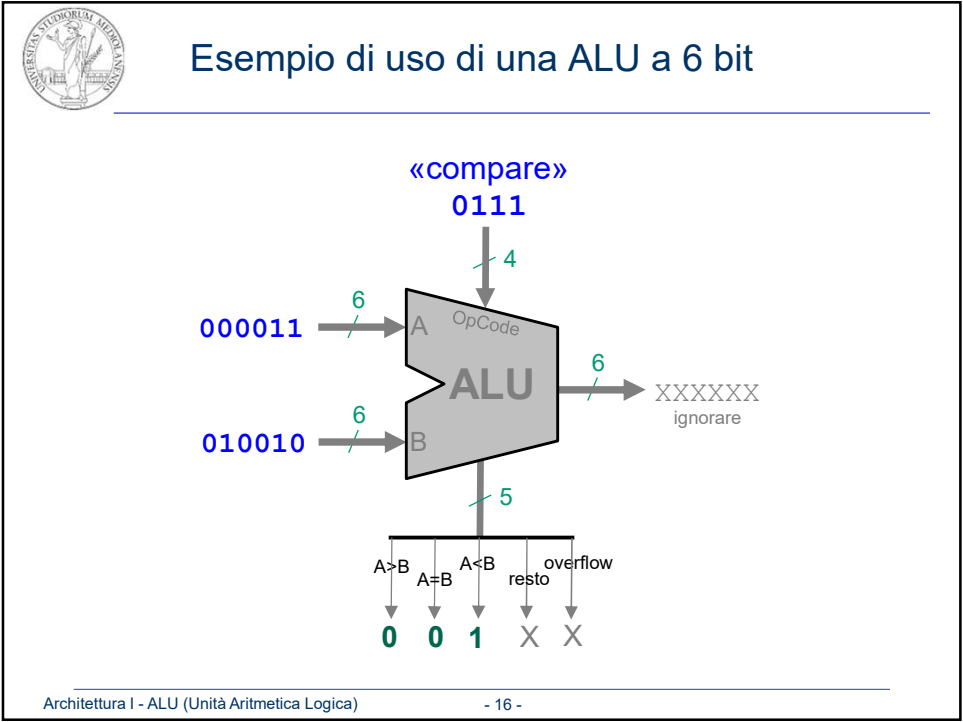
13



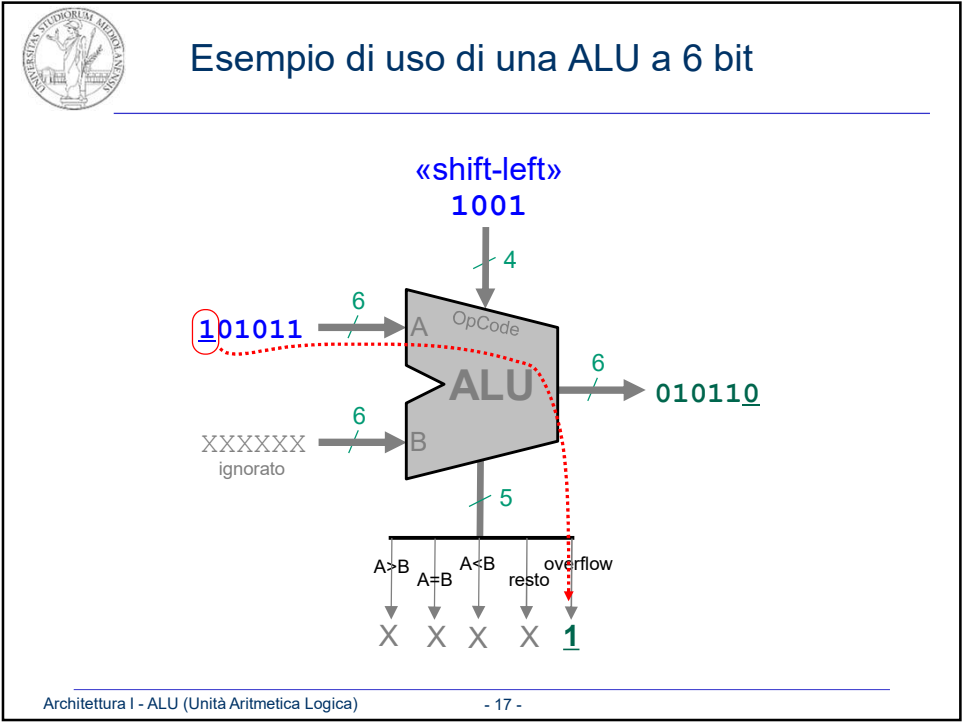
14



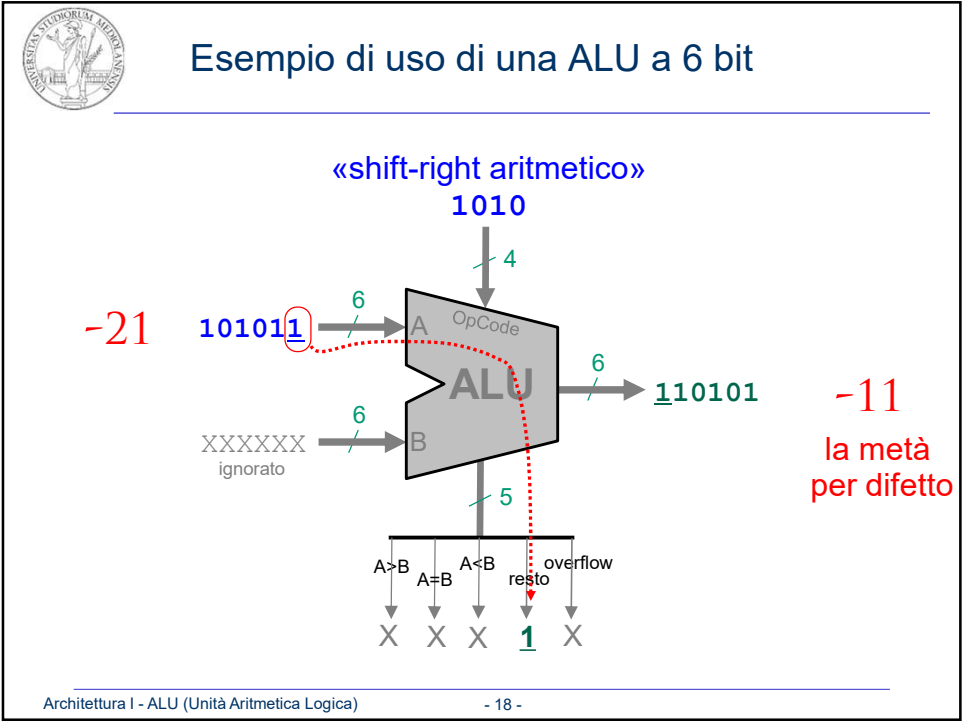
15



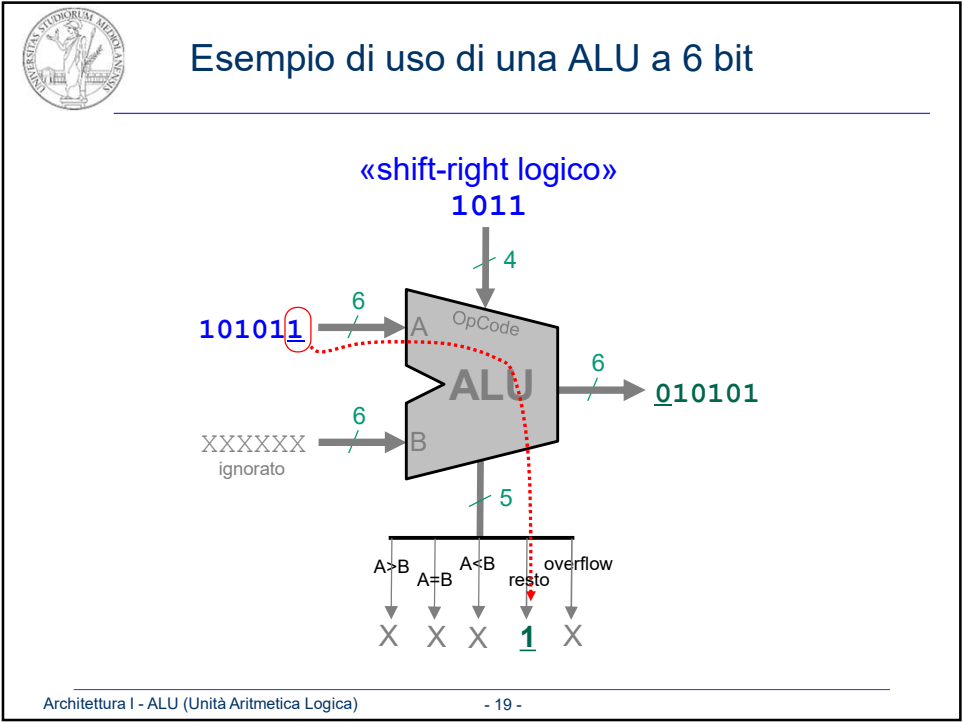
16



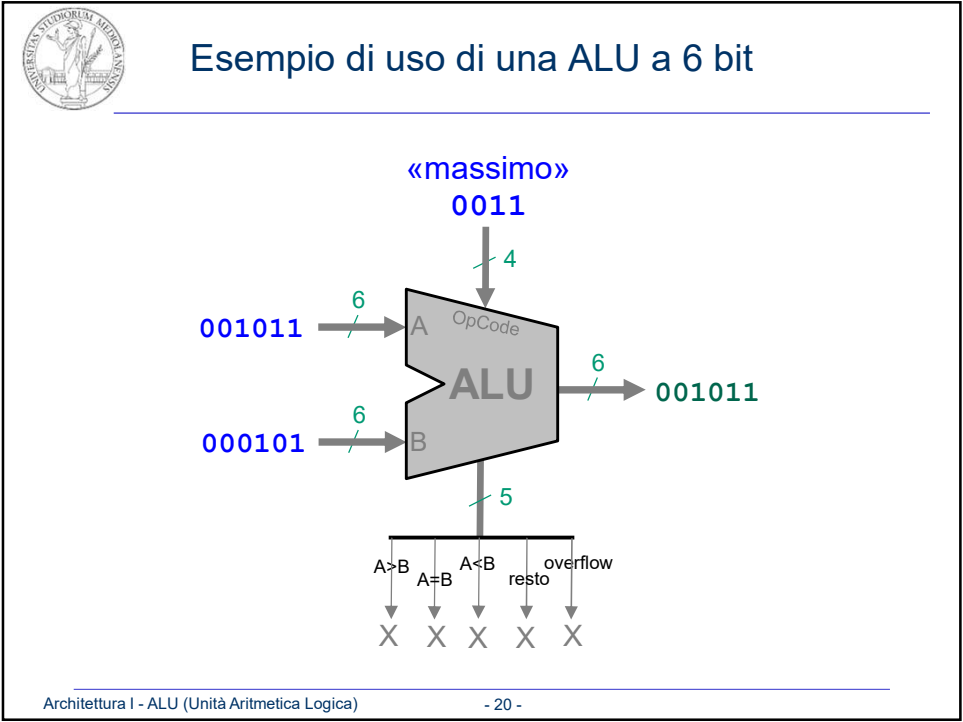
17



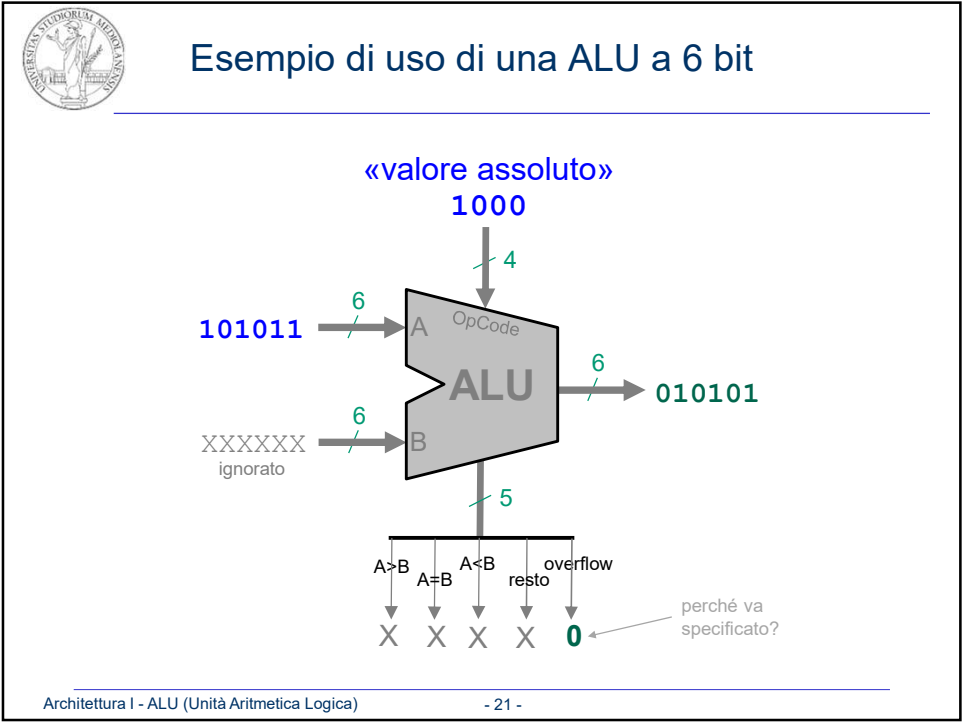
18



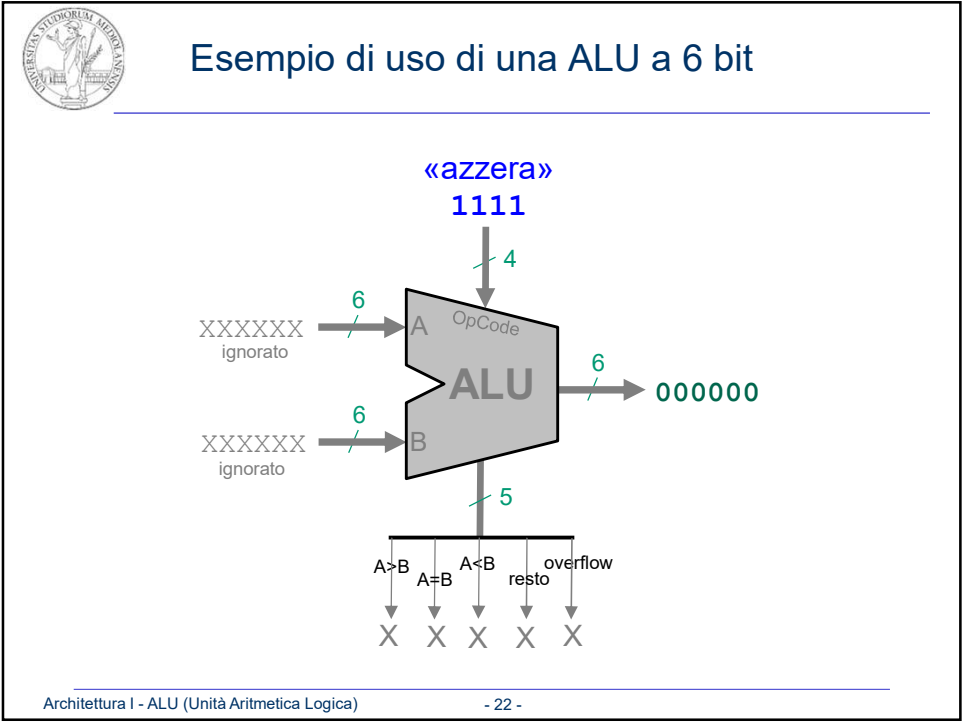
19



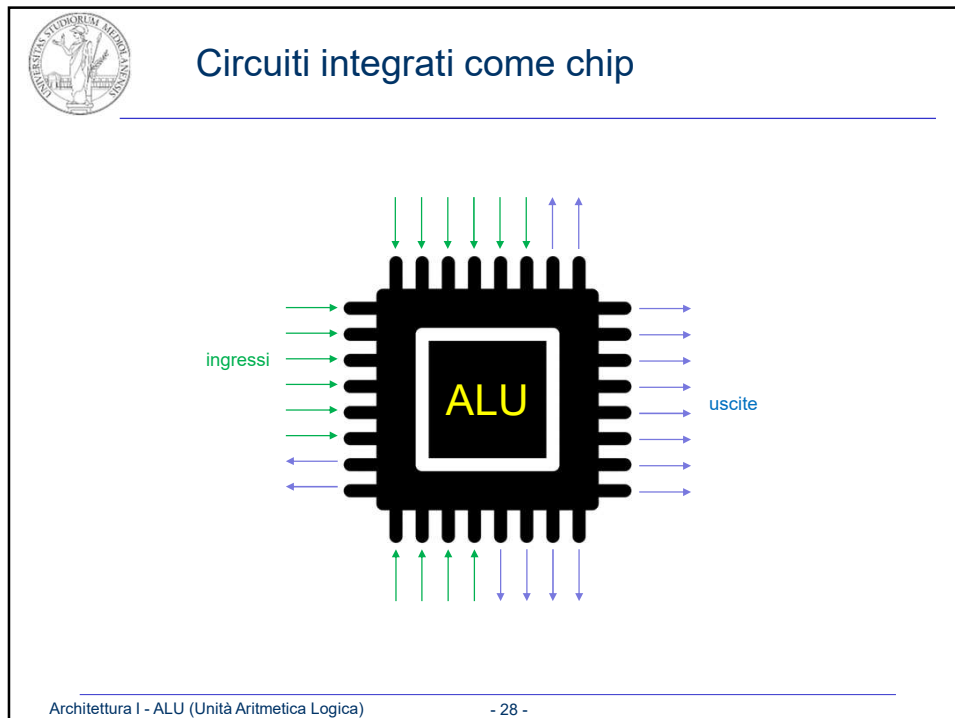
20



21



22

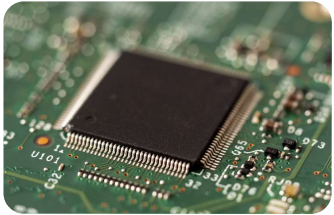
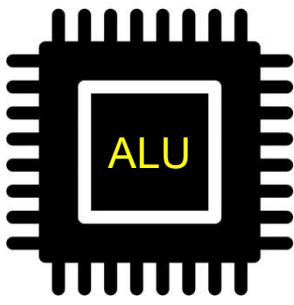


28



ALU come chip (cenni storici)

- Le ALU, come si capisce, sono circuiti molto complessi (diverse migliaia di porte)
- Negli anni 70, le ALU sono state realizzate come singoli chip integrati (realizzati con varie tecnologie)
1 ALU = 1 Chip →
- Oggi, le ALU sono integrate in circuiti ancora più grandi, che implementano tutta la CPU o GPU («microprocessor»)
- Anzi, una CPU può incorporare molte ALU (per es, 8, 16, 32...) per eseguire più operazioni in parallelo
= maggior numero di FLOPS
- Una GPU, specializzata nel calcolo parallelo, incorpora un numero ancora maggiore di ALU: anche migliaia



Architettura I - ALU (Unità Aritmetica Logica) - 29 -

29



ALU. note:

- Le ALU reali operano su operandi con 8, 16, 32, 64 bits
 - ▶ I 2 operandi hanno lo stesso n. di bit del risultato
- Esistono ALU che prevedono più di due operandi in ingresso, per esempio 3, e operazioni che usano tutti e tre gli operandi
- Le ALU possono essere specializzate su tipi interi oppure in virgola mobile (cioè, sia gli operandi che il risultato è espresso in quel tipo)
 - ▶ Oppure prevedere operazioni su interi, e operazioni separate su float

30



Tipi di ALU e varianti

- ALU con tre parametri in ingresso
 - ▶ il terzo è parametro utilizzato solo da alcune operazioni
 - ▶ Per esempio, un'operazione **Multiply-And-Add**(MAD) definita come $MAD(a, b, c) = a \cdot b + c$
- ALU per Floating point
 - ▶ Separate da ALU per interi (che eseguono anche op su CP2, e operazi Logiche)
 - ▶ Nota: le op. in virgola mobile includono op complesse come radici quadrate, elevamento a potenza, log, funzioni trigonom. ...
- Vector computing: ciascuno degli operandi in ingresso sono vettori di (per es) 4 valori.
 - ▶ Es: somma "componente per componente" dei 4 valori = 1 op.
 - ▶ Parallelismo!

31



ALU : scelte progettuali base

- Scelte difficili nel progettare una ALU: *quante operazioni supportare?*
quanto complesse?
- ALU che supporta molte operazioni diverse:
 - ▶ circuito ALU grande e complesso
→
ALU più costosa e difficile da realizzare, e anche più lenta ☹
- ALU che supporta operazioni complesse:
 - ▶ anche le istruzioni semplici vanno lente, tanto quanto quella più complessa (i circuiti sono in parallelo!). ☹
- ALU che supporta un numero minore di operazioni più semplici:
 - ▶ Ciascuna operazione è più veloce ☺
 - ▶ Ma lo stesso risultato necessiterà di *più operazioni* per essere computato!
 - ▶ Es: invece di `compare(A,B)` → `subtract( A , B)`, poi check del segno
 - ▶ Es: invece di `mul(A,B)`, → sequenza di shift di A e somme (caso ipotetico)
 - ▶ Es: invece di `subtract(A,B)`, → flip del segno di B (una op), poi `add(A,B)`
 - ▶ Es: invece di `Pass_A`, → B prende Zero (una op), poi `add(A,B)`