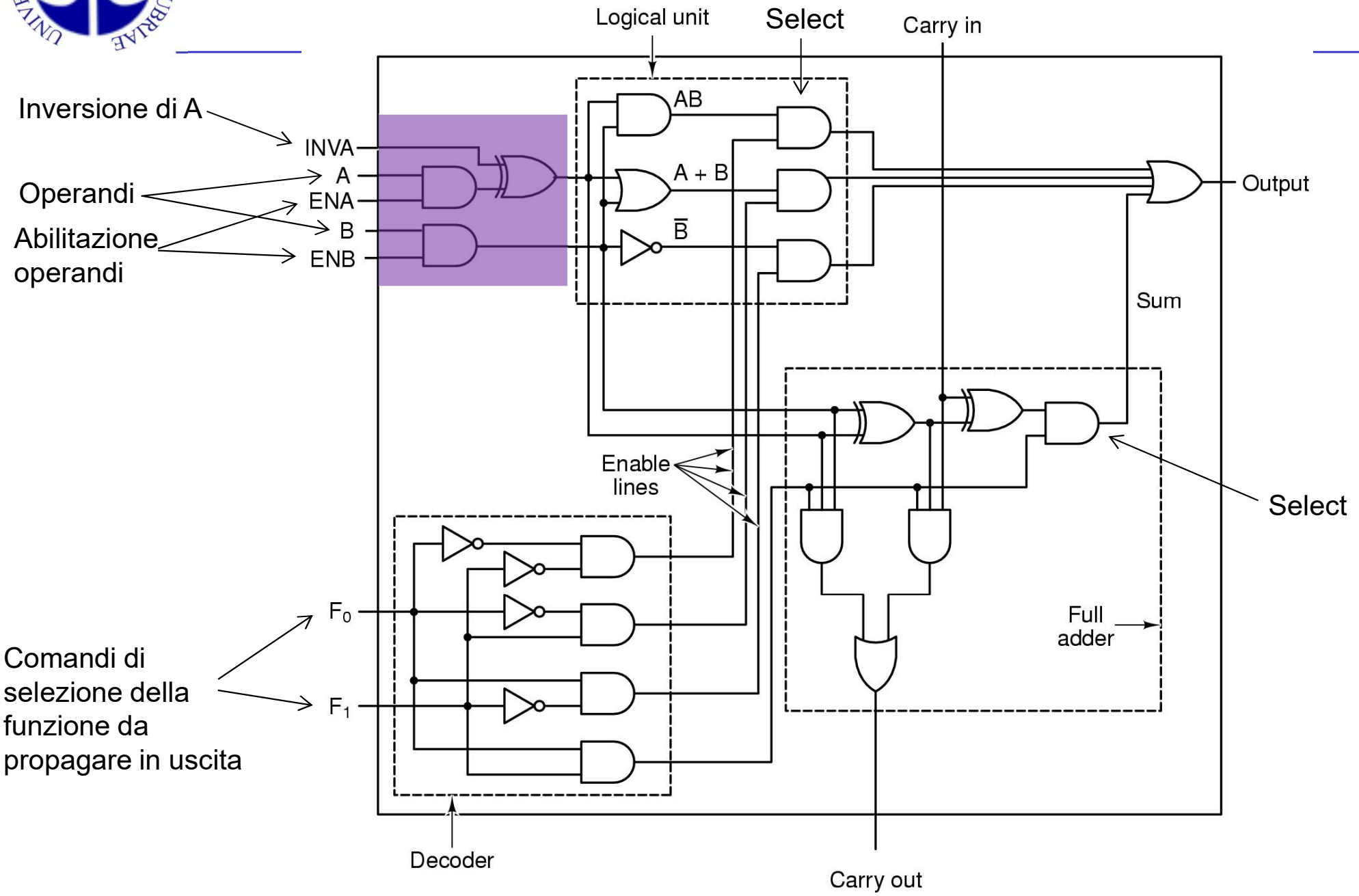
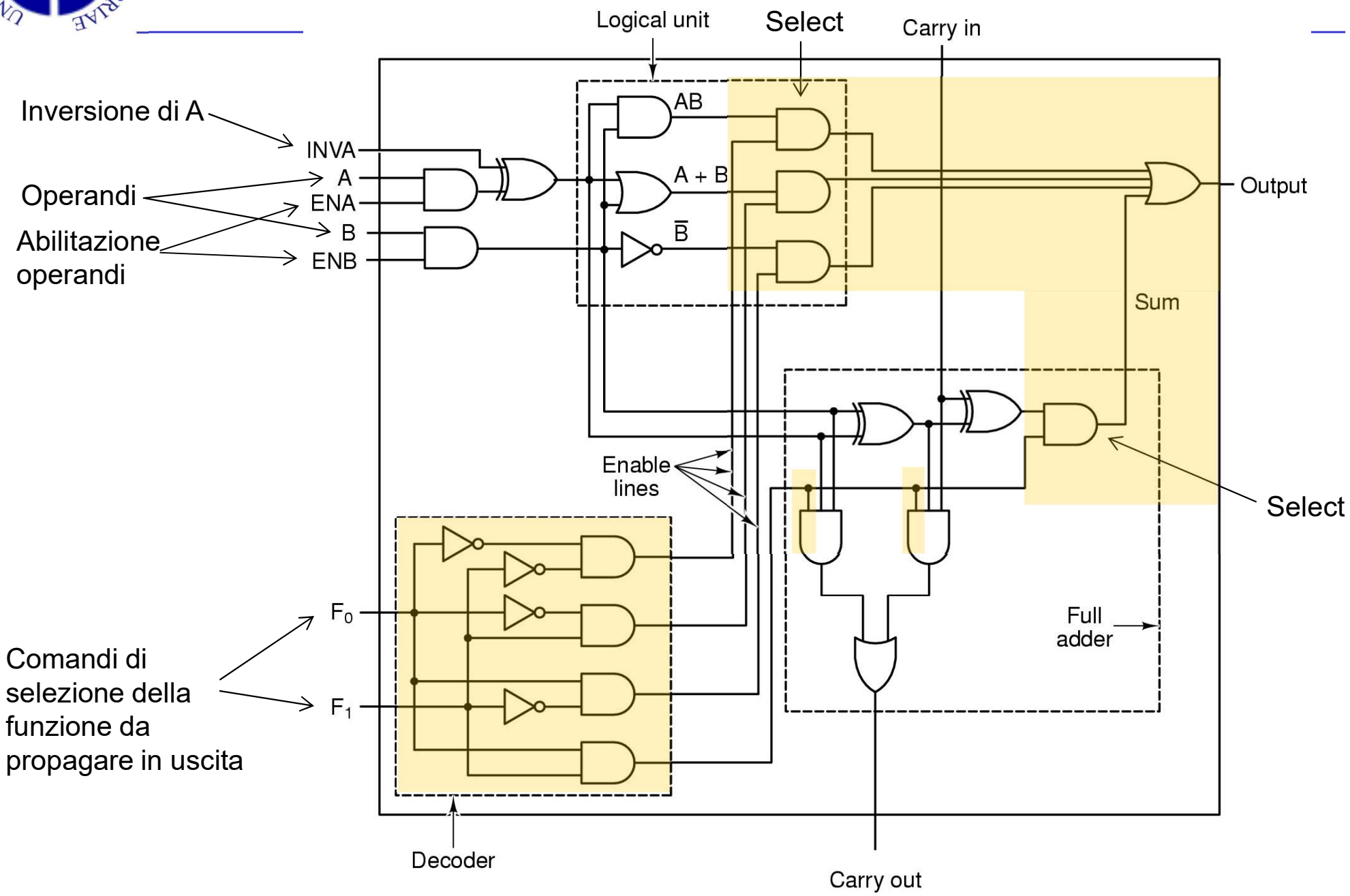
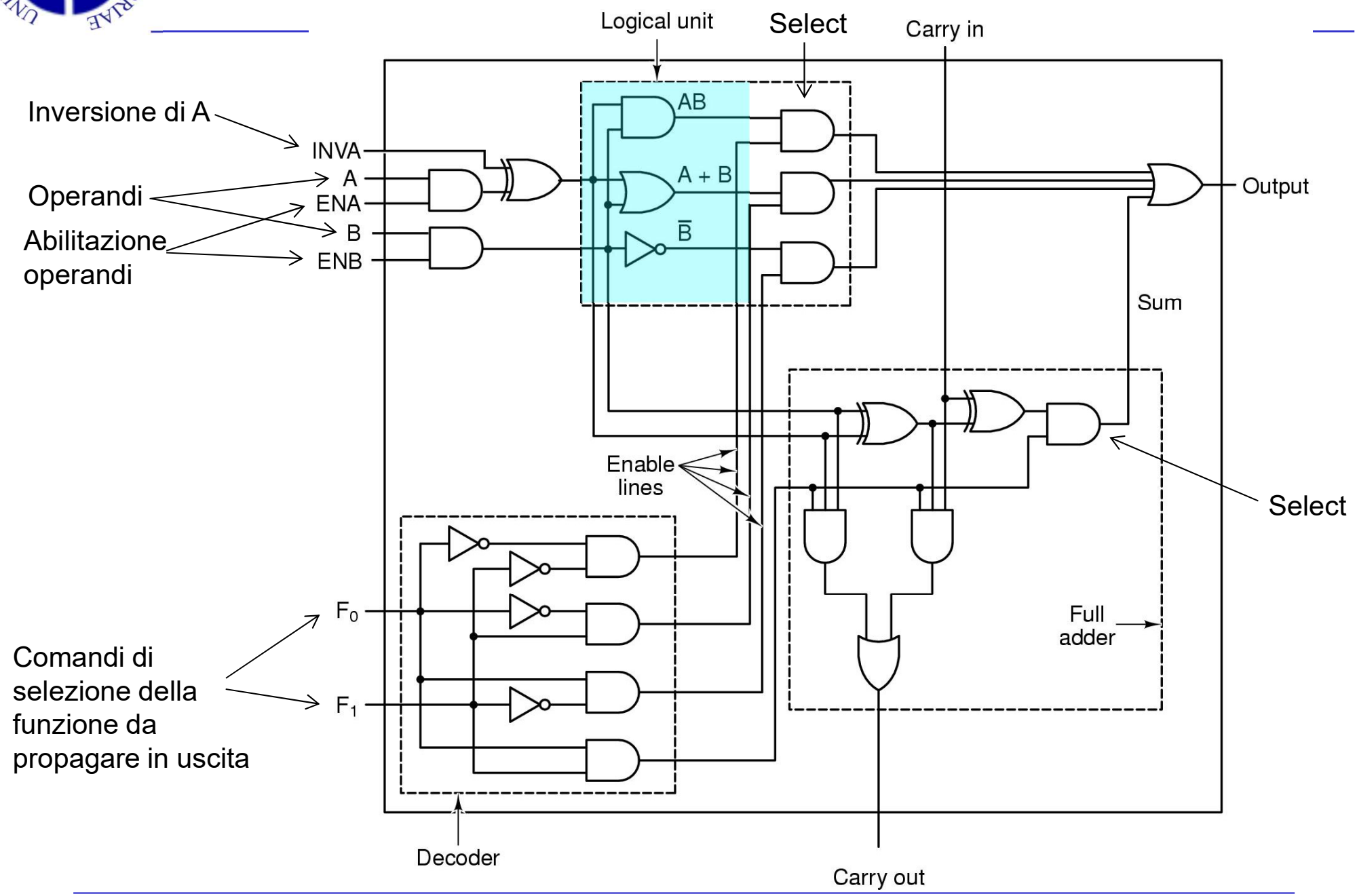
ALU da 1 bit: preprocessing



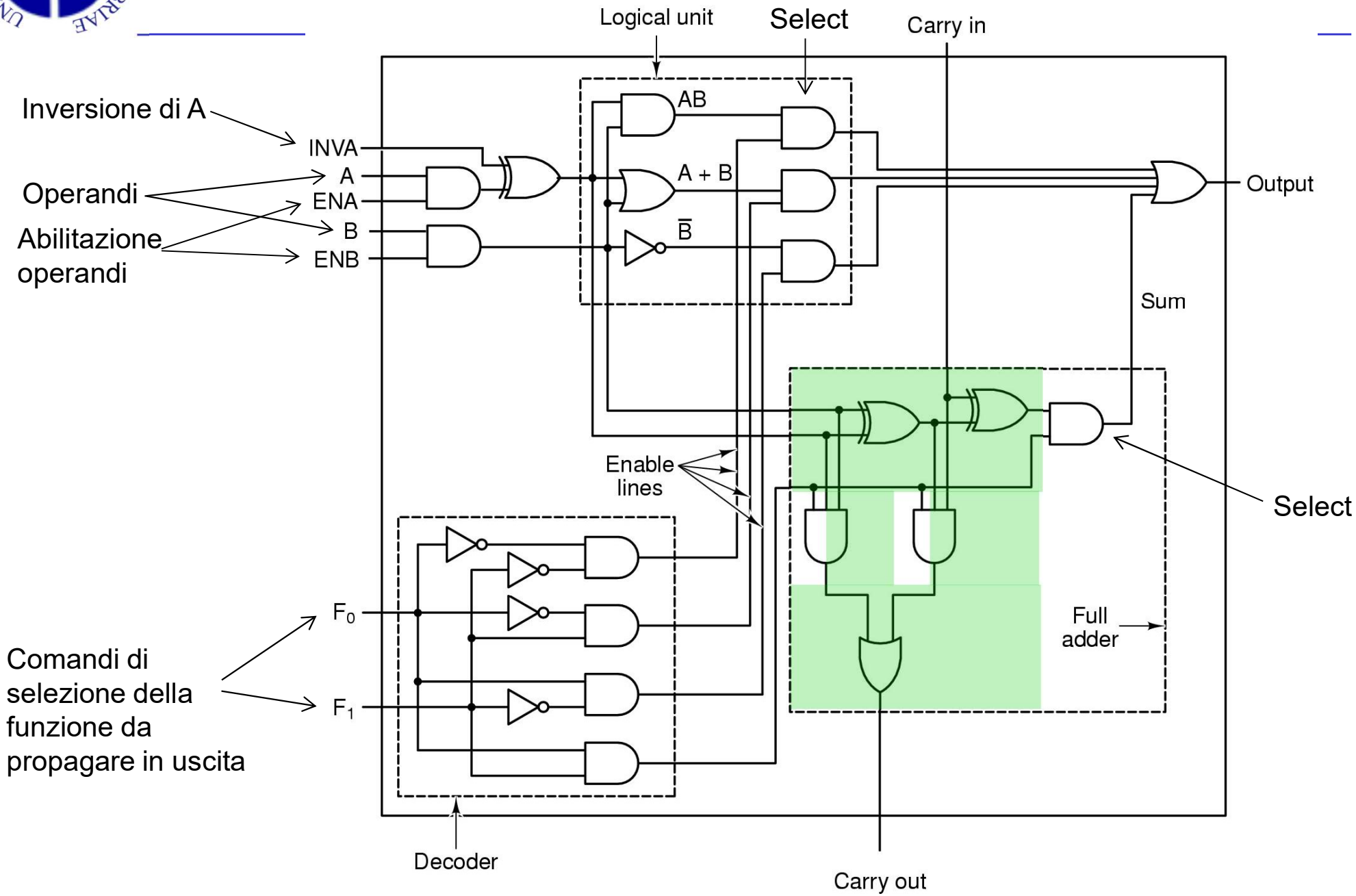
ALU da 1 bit: multiplexer



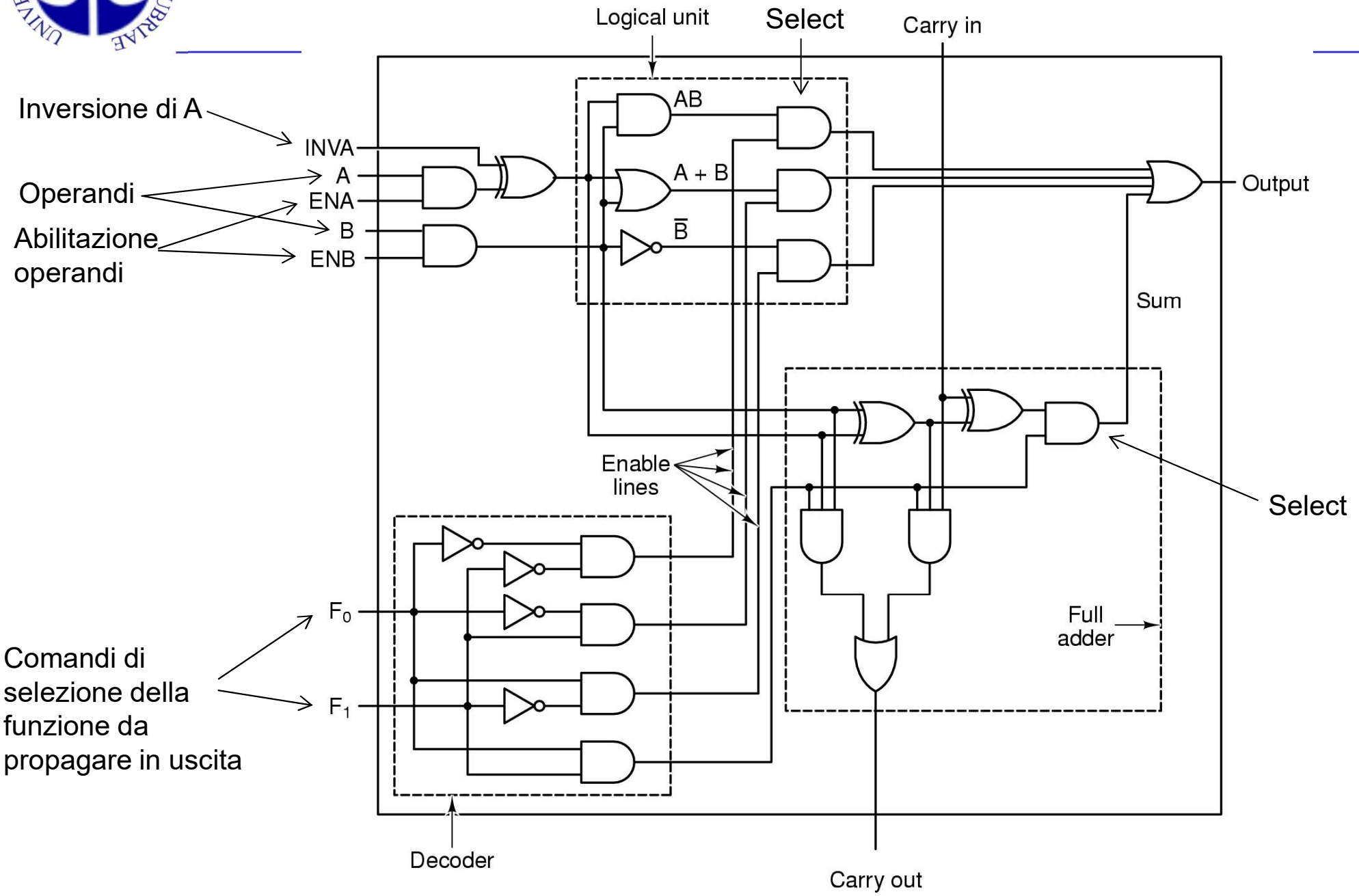
ALU da 1 bit: funzioni logiche



ALU da 1 bit: sommatore



Schema logico di una ALU da 1 bit



ALU note:

- Le ALU reali operano su operandi di un numero molto maggiore di bit (come 8, 16, 32, 64)
- Tipicamente, i due operandi hanno lo stesso numero di bit del risultato
- Le ALU possono essere in grado di eseguire op. anche complesse
  - Come radici quadrate, elevamento a potenza, funzioni trigonom. …
- Operaz. diverse possono interpretare gli operandi in modo diverso
  - es: i due comandi distinti SUM e SUM-FLOAT possono effettuare: somma fra numeri interi, somma fra numeri in virgola mobile
  - spesso, le op in virgola mobile sono le più importanti ed ottimizzate
  - esistono anche ALU specializzate per operazioni in virgola mobile in cui tutti i comandi assumono operatori in virgola mobile
- Le ALU hanno due, oppure tre parametri in ingresso
  - il terzo è utilizzato solo da alcune operazioni
- Sono stampate su un circuito integrato (naturalmente) come parte della CPU

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)