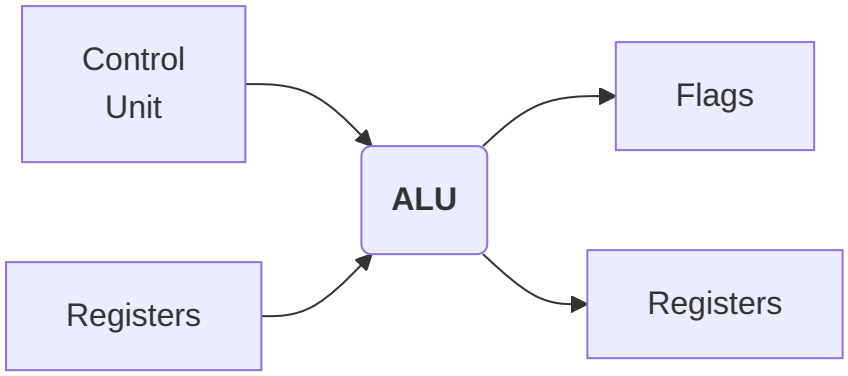
Aritmetica degli elaboratori

Unità aritmetica e logica

- Esegue le operazioni aritmetiche e logiche
- Preleva i dati elaborati dai registri (elaborati in binario)
- Salva il risultato nella propria locazione di memoria (Accumulatore)
- Comunica con la Control Unit
- Gestisce gli interi, ma anche numeri reali

Input e Output ALU



Rappresentazione in modulo e segno

Si considera come segno il bit più a sinistra

- 0 significa positivo
- 1 significa negativo

Esempio

- +18 = 00010010
- -18 = 10010010

Per eseguire operazioni aritmetiche bisogna considerare sia i moduli sia i segni

Complemento a due

- Positivi: da 0 a 2^{n-1}-1
- Negativi:
  - Bit più significativo (a sinistra) a 1
  - I restanti n-1 bit posso avere 2^{n-1} configurazioni diverse

- Per n bit possiamo rappresentare tutti i numeri da -2^{n-1} a $+2^{n-1}-1$

Complemento a due su 3 bit

Bit pattern	Valore rappresentato
011	3
010	2
001	1
111	-1
110	-2
101	-3
100	-4

Decodifica complemento a due

Esempio
2 in biniaro è 0010
ora bisogna invertire gli 0 e 1
diventa quindi 1101
adesso ci sommiamo 1
il risultato è quindi 1110 (-8 +4 +2 = 2)

Benefici del complemento a due

- Facile negazione tramite il metodo visto sopra
- Facili operazioni aritmetiche
- Conversione tra diverse lunghezze
    - da una conversione a n bit a una a m bit con (m > n)

Casi speciali

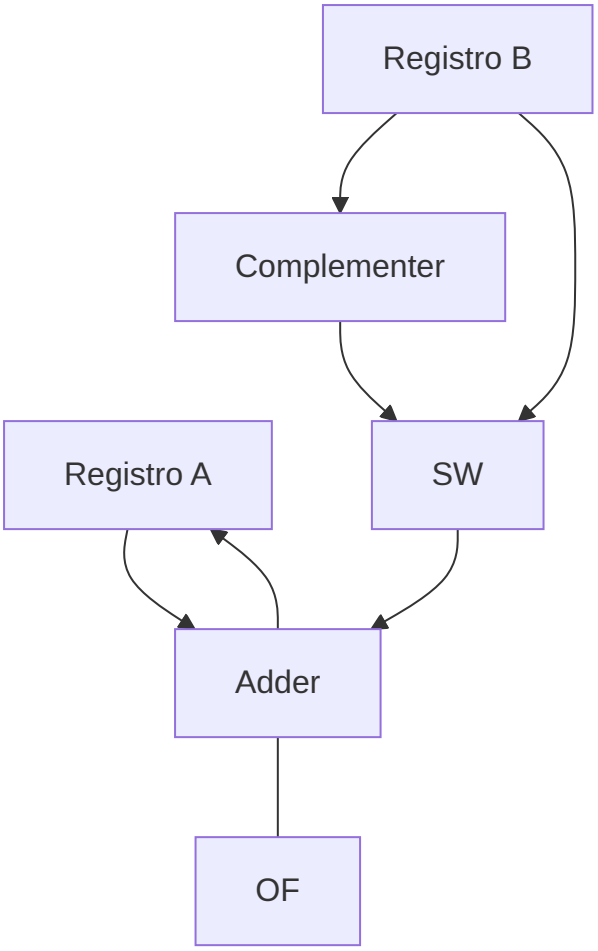
0 = 0000
inverso = 1111
1111 + 1 = 1 0000
1 bit di overflow (quindi ignorato)
= 0000

-128 = 10000000
inverso = 01111111
01111111 + 1 = 10000000
quindi il risultato è sempre -128, questo si verifica perchè i numeri negativi rappresentabili sono sempre 1 in più di quelli positivi, tramite il complemento a due.

Somma e Sottrazione

- La somma è una normale somma binaria
- Per la sottrazione bisogna tenere in considerazione i complementi a due. Per esempio 7-5 = 7 + (-5) = 0111 + 1011 = 1 0010 = 1 bit di overflow, risultato = 2

Hardware per somma e sottrazione



1. **Registri di Input**:
   - Registro A contiene il primo operando
   - Registro B contiene il secondo operando
2. **Elementi di Elaborazione**:
   - Complementer un circuito che genera il complemento del numero contenuto nel registro B
   - Switch è un selettore che sceglie tra B o il suo complemento a due
   - Adder è un sommatore che esegue tale operazione
   - OF è il bit di overflow che segnala il supermento della capacità di elaborazione
3. **Funzionamento**:
   - Nel caso di una somma SW seleziona direttamente il valore di B
   - Nel caso di una sottrazione il Complementer genera il complemento di B e SW seleziona tale valore

- L'adder esegue la somma tra A e B
- Il risultato può essere reinderizzato sul registro A per poter effettuare altre operazioni sullo stesso risultato

Moltiplicazione

La moltiplicazione è invece molto complessa , è necessario calcolare il prodotto parziale per ogni cifra e sommare i prodotti parziali.
Per esempio:
1011(11) x
1101(13) =
1011 + 00000 + 101100 + 1011000 = 10001111(143)
Funziona solo senza segno
Una soluzione è l'utilizzo dell'algoritmo di **Booth**

Divisione

È più complessa della moltiplicazione, utilizza traslazioni, somme e sottrazioni ripetute

Numeri Reali

- **Virgola mobile (o Floating Point)**
  - Ha il vantaggio di poter rappresentare numeri molto grandi o piccoli con poche cifre, valo lo stesso per il binario
  - S: è la mantissa, ossia la parte intera prima della virgola
  - B: è la base
  - E: è l'esponente

La struttura è Bit di segno + esponente polarizzato + mantissa
Il numero si rappresenta come +/- 1.mantissa x $2^{\text{esponente}}$

L'esponente polarizzato significa che è stato aggiunto un valore fisso (bias) all'esponente di base e poi tradotto in binario, per esempio per 8 bit il bit è 127 quindi se l'esponente è 0 diventa:
0 + 127 = 01111111

- **Normalizzazione**
  - L'esponente è aggiustato in modo che il bit più significativo della mantissa sia 1
  - Per ottenerla si sposta la virgola fino a quando il primo bit è 1
  - Ogni spostamento della virgola fa aumentare o diminuire l'esponente

Esempio: 0.0625 è 0.0001 in binario
normalizzato diventa 1.000 x 2^{-4}
-4 + 127 = 123 (01111011)

Per aumentare la precisione della rappresentazione bisogna aumentare il numero di bit con 32 si ha 1 bit per il segno, 8 per l'esponente e 23 per la mantissa. Questa è la **Precisone Singola**. Quella **Doppia** si ha invece con 64 bit.

Riguardo alla **densità** i numeri in virgola mobile non sono distribuiti uniformemente. La densità è maggiore vicino allo zero e diminuisce man mano che ci si allontana, c'è inoltre una maggiore precisione per i numeri più piccoli.
Esempio: con 4 bit (+ segno), 2 per esponente (f e), 2 per mantissa (a b)
Numero= $2^{f \times 2+e-1} \times (1 + a \times 2^{-1} + b \times 2^{-2})$

f	e	a	b	numero
0	0	0	0	0.5
0	0	0	1	0.625
0	0	1	0	0.75
0	0	1	1	0.875
0	1	0	0	1
0	1	0	1	1.25
0	1	1	0	1.5
0	1	1	1	1.75
1	0	0	0	2
1	0	0	1	2.5
1	0	1	0	3
1	0	1	1	3.5
1	1	0	0	4
1	1	0	1	5
1	1	1	0	6
1	1	1	1	7

Qui si nota la densità e la precisione nei numeri vicini allo zero.

Standard IEEE 754

- Standard per i numeri in virgola mobile
- Utilizza un formato singolo a 32 bit e uno doppio a 64 bit
    - sempre 1 bit di segno

    - per quello singolo (8 di esponente e 23 di mantissa)
    - per quello doppio (11 di esponente e 52 di mantissa)
- Si possono usare, anche, formati estesi con più bit per la mantissa ed l'esponente
    - Maggiore precisione e diminuzione di errore dovuto ad arrotondamento ed Overflow

Somma e Sottrazione

Quattro fasi:

1. Controllo delllo zero
    - se uno dei due è 0, il risultato è l'altro numero
2. Allineamento delle mantisse
    - il numero con esponente minore viene spostato a destra per allineare i decimali e rendere uguali gli esponenti
3. Somma o sottrazione delle mantisse (bit per bit)
4. Normalizzazione del risultato
    - shift verso sinistro finchè si ha la cifra più significativa diversa da 0

Moltiplicazione e Divisione

Sei fasi

1. Controllo dello zero
2. Somma/Sottrazione degli esponenti
3. Sottrazione/Somma polarizzazione
    - si sottra/somma il bias per avere l'esponente base
4. Moltiplicazione/Divisione delle mantisse
5. Normalizzazione
6. Arrotondamento
    - la mantissa viene arrotondata per rispettare il formato richiesto

Per maggiore precisione del risultato, si usano i **bit di guardia**, di solito operandi nei registri della ALU che hanno più bit della mantissa + 1 i bit più a destra sono messi a 0 e permettono di non perdere bit se i numeri vengono shiftati a destra.

Un'ulteriore metodo per aumentare la precisione è **l'arrotondamento**. Se il risultato è in un registro più lungo, se portato in virgola mobile, bisogna arrotondarlo. Tramite 3 approcci:

1. Arrotondamento al più vicino
    - sommo 1 se l'ultimo bit è 1, altrimenti 0
    - elimino i bit aggiuntivi che inziano con 0

2. Arrotondamento per eccesso a $+\infty$ o per difetto a $-\infty$
3. Arrotondamento a 0
    - troncati i bit in più