Arithmetic and Logic Unit e moltiplicatore

Engineering Department in Ferrara

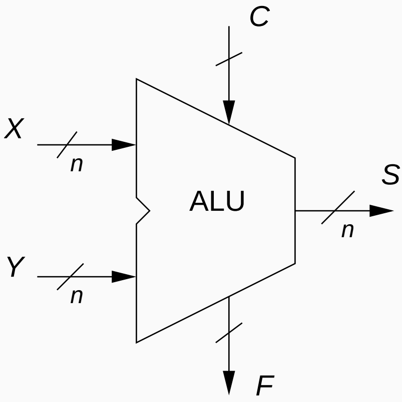
Sommario

Sommario

Introduzione

- **La ALU é un componente in grado di eseguire diversi tipi di operazione di tipo aritmetico e logico su due operandi di *n* bit**
- **Il tipo di operazioni viene determinato dalla configurazione dei bit di controllo**
- **Le prime CPU avevano la parte di elaborazione dati su una ALU (attualmente ne sono presenti piú di una)**
- **La ALU oltre a produrre un risultato di *n* bit produce anche diversi segnali di flag che rappresentano eccezioni o condizioni particolari sul risultato**
- **La sua struttura riflette un compromesso fra costo e prestazioni**

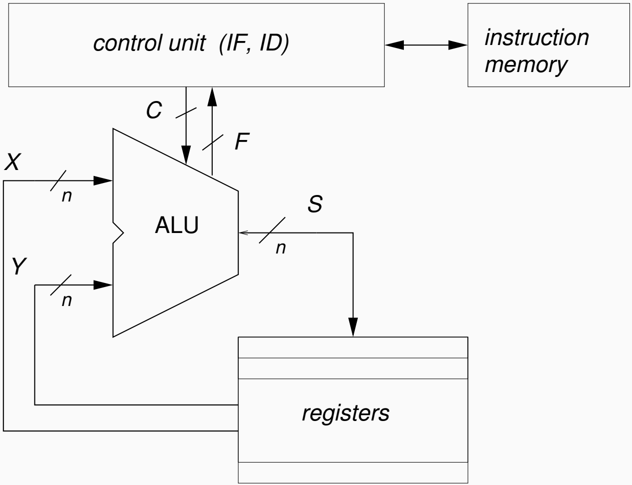
Schema



$S = S(X, Y, C)$ **e** $F = F(X, Y, S)$

Si noti che le operazioni aritmetiche sono in modulo 2^n

Utilizzo della ALU in una semplice CPU



Sommario

Arithmetic and Logic Unit - ALU

Bit sliced ALU

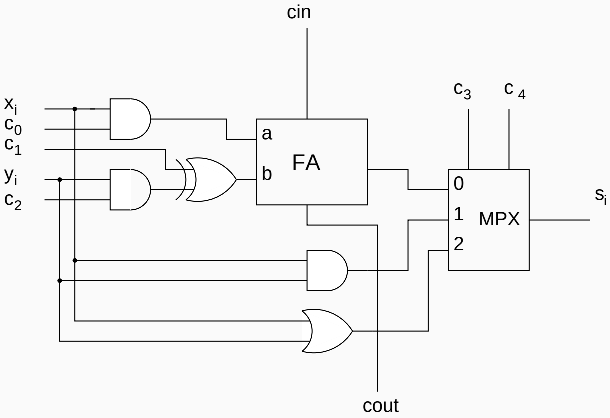
Rappresentazione in complemento a 2

Moltiplicatore

Struttura

- **Sono stati proposti e realizzati diversi tipi di ALU**
- **Forse il tipo piú semplice é quello di tipo bit - sliced in cui una ALU a *n* bit viene costruita a partire da *n* slice, ovvero *n* ALU a 1 bit ciascuna**
- **La ALU é essenzialmente costruita intorno a un *n* bit adder di tipo ripple-carry**
- **La soluzione presenta vantaggi di modularitá e costo, mentre ci sono svantaggi nelle prestazioni**
- **Ciascuna slice é costruita intorno a 1-ALU a 1 bit che é essenzialmente un FA con logica di pre e post elaborazione**

1-bit ALU



1-bit ALU - funzioni aritmetiche

In questo caso il carry-in é significativo

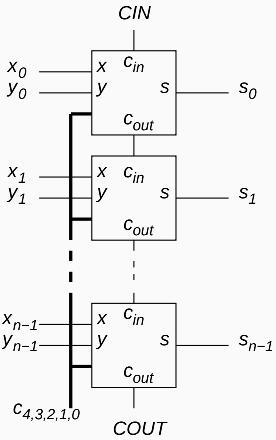
$c_4 c_3$	$c_2 c_1 c_0$	a	b	s	c_{out}
00	**000**	**0**	**0**	c_{in}	**0**
00	**001**	x_i	**0**	$x_i \oplus c_{in}$	$x_i c_{in}$
00	**010**	**0**	**1**	$1 \oplus c_{in}$	c_{in}
00	**011**	x_i	1	$x_i \oplus 1 \oplus c_{in}$	$x_i + c_{in}$
00	**100**	**0**	y_i	$y_i \oplus c_{in}$	$y_i c_{in}$
00	**101**	x_i	y_i	$x_i \oplus y_i \oplus c_{in}$	$x_i y_i + x_i c_{in} + y_i c_{in}$
00	**110**	**0**	y_i'	$y_i' \oplus c_{in}$	$y_i' c_{in}$
00	**111**	x_i	y_i'	$x_i \oplus y_i' \oplus c_{in}$	$x_i y_i' + x_i c_{in} + y_i' c_{in}$

1-bit ALU - funzioni logiche

Si tratta di operazioni logiche bit a bit il cui risultato non dipende dal carry-in

$c_4 c_3$	$c_2 c_1 c_0$	s
01	**- - -**	$x_i y_i$
10	**- - -**	$x_i + y_i$

ALU a *n* bit



CIN
x_0
y_0
x
y
c_{in}
s
c_{out}
s_0
x_1
y_1
x
y
c_{in}
s
c_{out}
s_1
x_{n-1}
y_{n-1}
x
y
c_{in}
s
c_{out}
s_{n-1}
$c_{4,3,2,1,0}$
COUT

Funzioni aritmetiche

- **Lo scopo della logica di pre elaborazione é quello di consentirci di realizzare la sottrazione oltre all'addizione**
- **É anche poi evidente che non ci si puó limitare agli interi senza segno**
- **A questo riguardo sono possibili diverse alternative**

Rappresentazione dei numeri interi con segno

- **Rappresentazione con segno e valore assoluto**
  - **il bit piú significativo viene usato per codificare il segno (0 positivo, 1 negativo)**
  - **segue la codifica del valore assoluto come numero naturale in base 2**
- **si hanno due rappresentazioni per lo 0 (10..0 e 00..0)**
- **il vero problema é dato dal fatto che la rete che realizza la somma e la sottrazione per questo tipo di codifica non risulta conveniente**

Rappresentazione dei numeri interi con segno

- **Rappresentazione in complemento a 2**
- **Dato un numero intero $A \geq 0$ rappresentato con n bit, il suo complemento a 2 é dato da $A^{c2} = 2^n - A$**
- **La rappresentazione di numeri interi in complemento a 2 su n bit codifica i numeri positivi da 0 a $2^{n-1} - 1$ nello stesso modo in cui si codificano i numeri naturali e quelli negativi da -2^{n-1} a -1 utilizzando il complemento a 2 del loro valore assoluto**
- **Anche in questo caso, il bit di maggior peso é 0 se $A \geq 0$ e 1 altrimenti**

Esempio

- **Supponiamo che sia $n = 4$ e quindi si possono rappresentare i numeri da -8 a 7**
- **Il numero 6 si rappresenta come un numero naturale, ovvero come 0110**
- **Il numero -4 si rappresenta con la codifica binaria del complemento a 2 del valore assoluto di -4, quindi come $2^4 - |-4| = 12$ la cui codifica binaria é 1100**
- **Si noti che aggiungendo dei bit nella rappresentazione dei numeri, quelli positivi vanno estesi con degli 0 e quelli negativi con degli 1**
- **Se volessi rappresentare 6 a 8 bit, avrei 00000110 e se volessi rappresentare -4, avrei 11111100**

Calcolo del complemento 2 tramite il complemento a 1

- **Si puó utilizzare un metodo piú rapido**
- **Si consideri un numero naturale binario** $A = a_3 a_2 a_1 a_0$**, il suo valore é dato da** $v(A) = a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$
- **Si consideri ora il complemento a 1 di** A**,** $A^{c1} = a'_3 a'_2 a'_1 a'_0$ **il cui valore é dato da** $v(A)^{c1} = a'_3 2^3 + a'_2 2^2 + a'_1 2^1 + a'_0 2^0$
- **Quanto vale** $A + A^{c1}$ **?**
  $(a_3 + a'_3)2^3 + (a_2 + a'_2)2^2 + (a_1 + a'_1)2^1 + (a_0 + a'_0)2^0 =$
  $2^3 + 2^2 + 2^1 + 2^0 = 15 = 2^4 - 1$
- **Quindi** $A + A^{c1} = 2^n - 1$**, da cui** $A^{c1} = 2^n - 1 - A = A^{c2} - 1$ **e infine** $A^{c2} = A^{c1} + 1$
- **Quindi per ottenere il complemento a 2 di un numero, si possono negare tutti i bit e sommare 1**

Calcolo del complemento 2 tramite il complemento a 1

- **Lo stesso metodo si puó utilizzare per calcolare il valore assoluto di un numero negativo rappresentato in complemento a 2**
- **In tale caso sia A il numero negativo, che sará codificato da** $A^{c2} = 2^n - |A|$
- **Applichiamo il complemento a 2 ad A^{c2}:** $(A^{c2})^{c2} = 2^n - (2^n - |A|) = |A|$
- **Quindi** $|A| = (A^{c2})^{c2} = (A^{c2})^{c1} + 1$
- **Quindi per ottenere il valore assoluto di un numero negativo, si possono negare tutti i bit della rappresentazione in complemento a 2 del numero negativo e sommare 1**

Esempio

- **Il numero -4 si puó calcolare determinando la codifica binaria di 4 (0100), complementando tutti i bit di 0100 e ottenendo quindi 1011 e infine sommando 1 ottenendo cosí 1100**
- **Se** $n = 6$ **e si vuole conoscere il valore di 110100, si complementano i suoi bit ottenendo 001011 e si somma 1 ottenendo cosí 001100 che é il valore assoluto che convertito in decimale da 12 e quindi il numero di partenza é -12**

Somma in complemento a 2

- **Si ricorda che se consideriamo un sommatore binario a n bit con gli ingressi rappresentati da due numeri naturali A e B, gli n bit di uscita forniscono $(A+B)_{mod\ 2^n}$**
- **Supponiamo ora che X e Y rappresentino due numeri interi tali che $X \geq 0$ e $Y < 0$**
- **Supponiamo che siano codificati in complemento a 2**
  - **mentre all'ingresso A dell'adder si ha X con la stessa codifica del numero naturale corrispondente**
  - **all'altro ingresso B dell'adder sará presente la codifica binaria del numero naturale $2^n - |Y|$**
- **All'uscita dell'adder risulterá quindi $(X + 2^n - |Y|)_{mod\ 2^n}$**
- **Siccome $(p+q)_{mod\ q} = p$, $p \geq 0, q > 0$, se $(X - |Y|) \geq 0$, in uscita si ha $X - |Y|$ (ovvero $X + Y$)**

Somma in complemento a 2

- **Se invece** $(X - |Y|) < 0$**, l'uscita rimane** $(X + 2^n - |Y|)_{mod\ 2^n}$ **che possiamo scrivere come** $2^n - |X - |Y|| = (|X - |Y||)^{c2}$
- **Questa é ancora la rappresentazione in complemento a 2 del risultato atteso**
- **Quindi la somma di interi con segno rappresentati in complemento a 2 puó essere eseguita con un adder, questo é il motivo del successo di tale rappresentazione**
- **Conclusioni valide solo se il risultato puó essere rappresentato con *n* bit**

Esempi

Sia $X = 4$ **e** $Y = -1$**, quindi in ingresso a un 4-bit adder abbiamo** 0100 **e** 1111

	0100	**+**
	1111	**=**
1	**0011**	

Si ottiene correttamente 3, si noti che per il momento il carry-out dell'adder non interessa. Vediamo ora il caso in cui $Y = -8$ **che corrisponde a 1000**

	0100	**+**
	1000	**=**
0	**1100**	

Risultato che in complemento a 2 corrisponde a -4.

ALU a n bit: funzioni aritmetiche

- **Fino a questo momento i simboli $\{X, Y, S\}$ sono stati utilizzati nello schema della ALU per denotare parole di n bit**

- **Nel caso delle funzioni aritmetiche della ALU, le configurazioni binarie di tali parole codificano numeri interi rappresentati in complemento a 2 che verranno denotati come $(X)_2,\ (Y)_2,\ (S)_2$, lo stesso vale per le costanti**

- **Nel caso di espressioni il pedice viene riportato solo all'esterno delle parentesi:** $(X + Y + 9)_2 = (X)_2 + (Y)_2 + (9)_2$

- **Le notazioni $(W)_2^{c1}$ e $(W)_2^{c2}$ sono utilizzate per denotare il complemento a 1 e il complemento a 2 di un numero intero positivo W.**

ALU a n bit: funzioni aritmetiche

$c_4 c_3\, c_2 c_1\, c_0$	$S\,(CIN = 0)$	$S\,(CIN = 1)$
00 000	$(0)_2$	$(1)_2$
00 001	$(X)_2$	$(X+1)_2$
00 010	$(-1)_2$	$(0)_2$
00 011	$(X-1)_2$	$(X)_2$
00 100	$(Y)_2$	$(Y+1)_2$
00 101	$(X+Y)_2$ ← **somma**	$(X+Y+1)_2$
00 110	$(Y)_2^{c1}$	$(Y^{c1}+1)_2 = (Y)_2^{c2} =$ $= (-Y)_2$
00 111	$(X+Y^{c1})_2$	$(X+Y^{c1}+1)_2 =$ $= (X+Y^{c2})_2 = (X-Y)_2$ ← **sottrazione**

Funzioni logiche

$c_4 c_3\ c_2 c_1 c_0$	S
01 - - -	XY
10 - - -	$X + Y$

Per la complementazione si puó utilizzare il complemento a 1

Bit di flag

I bit di flag forniscono indicazioni sul risultato

- **bit di zero: vale 1 se il risultato** $S = 0_2$
- **bit di paritá: fornisce la paritá sul risultato**
- **bit di carry: CARRY OUT**
- **bit di overflow: vale 1 se il risultato non é contenuto in** n **bit**
- **bit di segno: vale 1 se il risultato é negativo**

Overflow

Si ha overflow se il segno del risultato é diverso da quello atteso sulla base del tipo di operazione e dei segni degli operandi:

- $X > 0$**,** $Y > 0$ **e** $S < 0$ **con una somma aritmetica**
- $X < 0$**,** $Y < 0$ **e** $S > 0$ **con una somma aritmetica**
- $X > 0$**,** $Y < 0$ **e** $S < 0$ **con una sottrazione aritmetica**
- $X < 0$**,** $Y > 0$ **e** $S > 0$ **con una sottrazione aritmetica**

Sia OV **il bit di overflow:**

$$\begin{aligned} OV = c_4' c_3' (c_2 c_1' c_0 (X_{n-1}' Y_{n-1}' S_{n-1} + X_{n-1} Y_{n-1} S_{n-1}') + \\ c_2 c_1 c_0 (X_{n-1}' Y_{n-1} S_{n-1} + X_{n-1} Y_{n-1}' S_{n-1}')) \end{aligned}$$

Esempi

Operandi A e B in ingresso all'adder contenuto nella ALU

Somma

0	**1**	**1**	**0**	**+**
0	**1**	**0**	**0**	**=**
1	**0**	**1**	**0**	

6+4=-6

1	**1**	**1**	**0**	**+**
1	**0**	**0**	**1**	**=**
0	**1**	**1**	**1**	

-2+(-7)=7

Sottrazione (il secondo operando viene complementato a 2)

0	**1**	**1**	**0**	**+**
0	**1**	**0**	**0**	**=**
1	**0**	**1**	**0**	

6-(-4)=-6

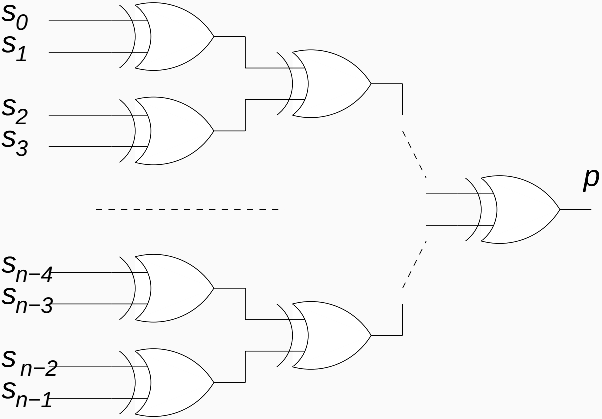
1	**1**	**1**	**0**	**+**
1	**0**	**0**	**1**	**=**
0	**1**	**1**	**1**	

-2-7=-7

Si noti che per la sottrazione sono illustrati i valori agli ingressi della ALU a 1 bit (ovvero dell'adder)

Bit di paritá

Serve per proteggere i dati che vengono scritti nei registri o in memoria



Bit di segno e bit di zero

- **Il bit di segno é banalmente il bit di maggior peso del risultato (che é significativo solo nel caso di operazioni aritmetiche)**
- **Il bit dizero assume il valore 1 solo quando tutti i bit del risultato hanno il valore 0. Puó essere ottenuto tramite un NOR a *n* ingressi**

Sommario

Arithmetic and Logic Unit - ALU

Bit sliced ALU

Rappresentazione in complemento a 2

Moltiplicatore

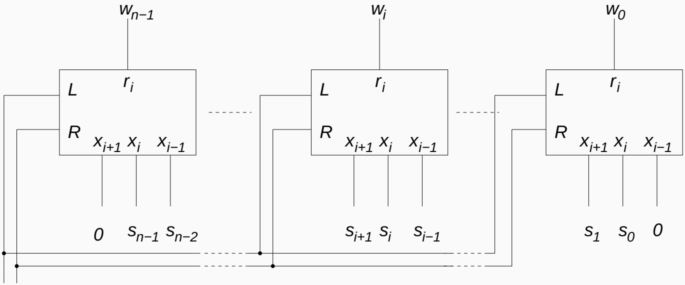
Moltiplicazione

- **La ALU illustrata non conteneva le operazioni di moltiplicazione e divisione**
- **Vedremo come caso particolare la moltiplicazione e divisione per 2 (applicata ai numeri naturali)**
- **Nel caso generale vedremo il prodotto di numeri naturali**

Moltiplicazione e divisione per 2

Moltiplicazione e divisione per 2 possono essere rispettivamente eseguite mediante uno shift a sinistra e uno a destra di una posizione ($\ll 1$ e $\gg 1$ in C)

Circuito (da mettere in uscita alla ALU) in grado di eseguire tali operazioni ($W_i = L'R'S_i + L'RS_{i+1} + LR'S_{i-1}$)



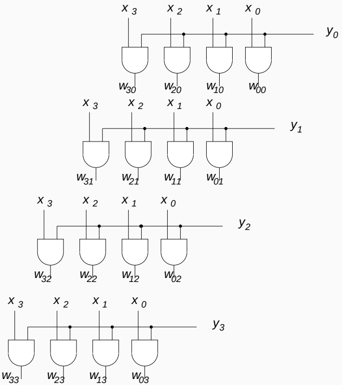
Moltiplicazione fra numeri naturali

L'algoritmo é quello utilizzato in base 10 (somma di prodotti parziali):

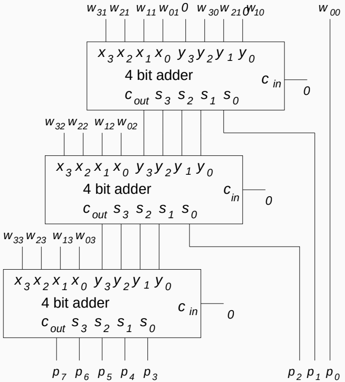
				x_3	x_2	x_1	x_0	$X\ *$
				y_3	y_2	y_1	y_0	$Y\ =$
				$x_3 y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$	$W_0\ +$
			$x_3 y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$	-	$W_1\ +$
		$x_3 y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$	-	-	$W_2\ +$
	$x_3 y_3$	$x_2 y_3$	$x_1 y_3$	$x_0 y_3$	-	-	-	$W_3\ =$
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	

Moltiplicazione fra numeri naturali

Logica AND (prodotti parziali)



Sommatori



Conclusioni

- **Le unitá aritmetiche sono un componente critico del data-path delle ALU**
- **Gli esempi riportati corrispondono alle soluzioni piú semplici**
- **Nelle CPU reali si utilizzano diverse tecniche per migliorare le prestazioni di ALU e moltiplicatori che non si riescono a trattare in questo ambito**