Appunti di Architettura degli Elaboratori

Alessandro Cheli - Prof. Marco Danelutto

A.A 2019-2020

Indice

Capitolo 1

Introduzione al Corso

- **Logica Booleana**
- **Aritmetica Binaria**
- **Reti Logiche**
- **Microarchitettura e Assembler ARM v7 e v8**
- **Gestione della memoria**
- **I/O**

Strumenti Software A differenza degli A.A passati utilizzeremo Verilog e Assembler ARM. Utilizzeremo **iverilog** come compilatore Verilog e **gtkwave** come tool grafico. Un ambiente di sviluppo Verilog completo che vedremo è **Quartus**. Per la seconda parte del corso, Assembler ARM, useremo la **toolchain GNU**, in particolare:

Se non hai una macchina ARM:

- cross-compiler per compilare
- QEMU per una macchina virtuale ARM
- gdb per debugging

Se hai una macchina ARM come un Raspberry Pi:

- Toolchain GNU per compilare
- Cavo Ethernet
- Server SSH sulla macchina ARM per accesso remoto

Storia degli Elaboratori Nei corsi di Architettura degli Elaboratori negli anni 80 i processori studiati erano: il 6502 (8 bit, processore del computer Apple II, noto per essere stato costruito nel garage di Steve Jobs e Wozniak), Z80, processore a 16 bit del famoso computer ZX80 e l'Intel 8088. Tali processori raggiungevano al massimo una velocità di clock (detto molto a grandi linee, operazioni al secondo) dell'ordine di meno di una decina di MHz (Mega Hertz, milioni). I processori odierni raggiungono cicli di clock sull'ordine dei GHz (Giga Hertz, miliardi). Nel corso degli anni fino ad oggi, l'evoluzione dei processori ha seguito la **legge di Moore**. La "legge" spiega

Figura 1.1: Sinclair ZX80

che ogni 18 mesi la potenza dei processori in commercio raddoppia, perché la densità dei transistor contenuti all'interno aumenta. Negli ultimi decenni abbiamo miglioramenti architetturali come super pipeline e super scalari, ciò ha permesso di introdurre processori **multicore**, ovvero che contengono più "nuclei" interni (detti core) che elaborano le istruzioni dei processi in esecuzione in parallelo. Ad oggi il numero di core in uno smartphone raggiunge anche gli 8 core, mentre in processori per server sono stati raggiunti numeri di core anche intorno ai 64. I processori odierni utilizzano core a 64 bit, con architettura X86_64 per Desktop o ARM per dispositivi mobili. Un componente fondamentale dell'evoluzione degli elaboratori è stato anche lo sviluppo dei processori grafici (GPU) con i quali ad oggi è possibile riprodurre grafica su schermo, ambienti tridimensionali molto complessi (videogiochi) o sfruttare la loro capacità di parallelizzazione per l'uso di reti neurali nell'intelligenza artificiale.

Osserveremo i calcolatori a diversi livelli di **astrazione**

I livelli di astrazione sono:

- Applicazioni utente
- Sistema Operativo
- Architettura (ASM, ad es. x86 o ARM)
- Microarchitettura
- Logica
- Circuiti digitali
- Device
- Fisica

Ogni livello si appoggia sul livello inferiore, ovvero è costruito sui componenti offerti dal livello inferiore. Dei principi fondamentali sono: **gerarchi, modularità e regolarità**

La modularità è fondamentale per avere moduli organizzati gerarchicamente, autonomi ed indipendenti.

Set di Istruzioni Distinguiamo due set di istruzioni dei processori, **CISC** e **RISC**. Gli acronimi sono rispettivamente **Complex Instruction Set Computer** e **Reduced Instruction Set Computer**, RISC contiene i processori ARM, che studieremo in dettaglio, mentre CISC comprende i processori più comuni nei desktop (X86 e X86_64)

Capitolo 2

Rappresentazioni Numeriche e Testuali

2.1 Aritmetica Binaria

I calcolatori utilizzano valori discreti (differenze di potenziale) fra 0 e 1 per rappresentare valori numerici. Viene detta Aritmetica Binaria l'aritmetica con i numeri rappresentati in base 2.

Siamo abituati a ragionare in base 10, ad esempio il numero 413 in base 10 è

$$104 = 10^2 \cdot 1 + 10^1 \cdot 0 + 10^0 \cdot 4$$

Lo stesso numero rappresentato in base 2 (codice binario) è

$$104_{10} = 01101000_2 = 2^7 \cdot 0 + 2^6 \cdot 1 + 2^5 \cdot 1 + 2^4 \cdot 0 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 0 + 2^0 \cdot 0$$

Un numero binario di 8 cifre è detto **byte**, un numero di 4 cifre è detto **nibble**. Una **parola** (**word**) è la quantità minima su cui viene rappresentato un intero in un calcolatore. Ad oggi le parole dei calcolatori sono 64 bit, alcuni calcolatori datati hanno parole da 32 bit.

La somma nell'aritmetica binaria è definita normalmente per i numeri positivi. Nei calcolatori i numeri hanno una dimensione finita (numero di bit) che indica il numero di cifre binarie con le quali è possibile rappresentare un numero. I positivi binari rappresentano numeri fino a $2^N - 1$ dove N è il numero di cifre.

Per rappresentare i numeri negativi si utilizza il metodo **segno-magnitudo** dove il bit più a sinistra rappresenta il segno (0 se il numero è positivo e 1 se è negativo). Il problema del metodo segno-magnitudo è che non rispetta la somma aritmetica. Può rappresentare numeri da $[-2^{N-1}, +2^{N-1}]$

Un metodo migliore per rappresentare i numeri negativi è il **complemento a due**. Nel complemento a due la cifra più a sinistra rappresenta sempre 2^{N-1} ma **negativo**. Il resto delle cifre sono positive e vengono sommate alla prima cifra negativa. Questo metodo rispetta la somma aritmetica. Per moltiplicare un numero per -1 si invertono le cifre binarie e si aggiunge 1 al numero. È possibile anche la sottrazione sommando un numero positivo ad uno negativo.

La somma fra due cifre può essere costruita con reti logiche. Il risultato della somma $A + B = A \oplus B$ (operatore XOR) mentre il riporto della somma $= A \wedge B$ (operatore AND)

2.2 Esadecimale

I numeri esadecimali sono numeri in base 16. Siccome non bastano le cifre decimali per rappresentare i numeri maggiori di 9 si usano le prime lettere dell'alfabeto. Una cifra esadecimale rappresenta un nibble (4 bit).

Figura 2.1: Gate XOR

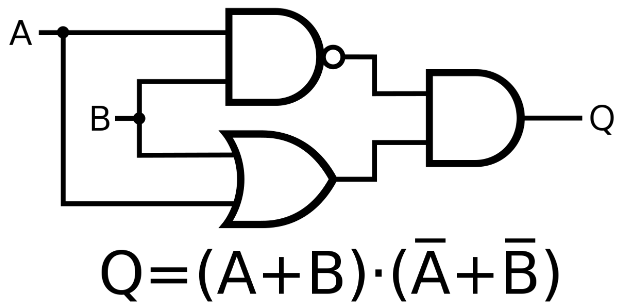
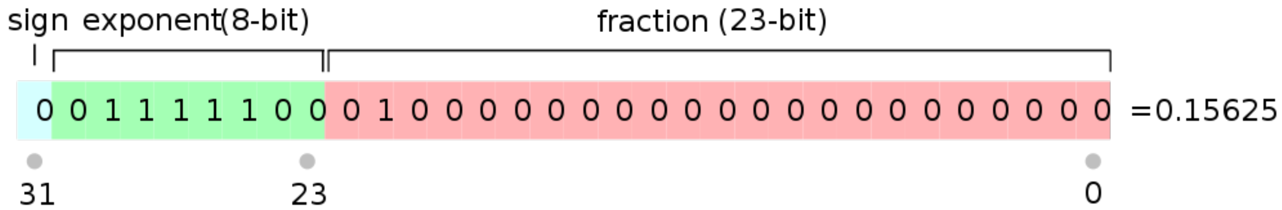


Figura 2.2: Standard IEEE 754 a 32 bit



2.3 Numeri in virgola mobile

I numeri in virgola mobile si rappresentano con lo standard EEE 754 che definisce come si rappresentano i numeri in virgola mobile a singola precisione e doppia precisione (32 e 64 bit)

I bit del numero vengono divisi in 3 parti. Il primo bit denota il segno, la seconda parte rappresenta l'esponente e la terza parte si denota mantissa. L'esponente esprime dove la virgola verrà posizionata, come nella notazione scientifica di una calcolatrice l'esponente rappresenta 10^n dove n è l'esponente. La mantissa è un numero di base moltiplicato per 10^0, e viene successivamente moltiplicato per l'esponente. L'esponente può essere sia positivo che negativo.

Nello standard a 32 bit la sezione esponente ha 8 bit di lunghezza. Un numero ad 8 bit può rappresentare numeri da 0 a 255, per ottenere gli esponenti negativi nello standard dei numeri a virgola mobile il numero a 8 bit rappresenta invece numeri da -127 a +128

Somma dei numeri a virgola mobile Per sommare i numeri a virgola mobile il primo passo è allineare le mantisse, significa osservare gli esponenti e spostarli fino a che le cifre non sono sommabili in colonna. Il secondo passo consiste nel sommare e il terzo passo nel normalizzare la somma. Nei processori la somma floating point viene eseguita in dei moduli appositi che in input ricevono due o più numeri floating point ed eseguono in dei sotto-moduli i tre passaggi della somma in un tempo $1/3t$ dove t è il tempo totale per eseguire una somma. I tre passaggi della

Figura 2.3: IEEE 754 a 64 bit

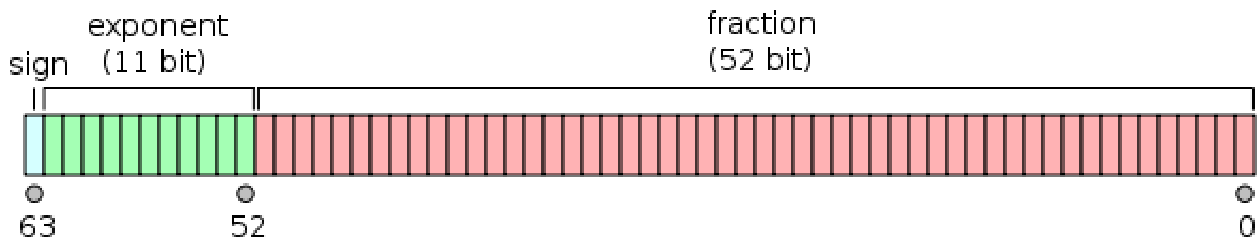


Figura 2.4: Tabella ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

somma possono essere sequenzializzati così che una volta che ogni sotto-modulo ha completato il passo, può ricevere subito l'input successivo (la somma di due numeri FP impiegherà $t + 1/3$ invece che $2t$)

Estensioni vettoriali Alcuni processori permettono di eseguire operazioni contemporaneamente su un registro dividendolo in sottoregistri più piccoli.

2.4 Codifica ASCII

La codifica ASCII è una tabella di codifica di caratteri testuali con interi da 0 a 255 (8 bit). La codifica ASCII estesa è a 16 bit e comprende diversi caratteri non latini.

Capitolo 3

Porte Logiche e Algebra di Boole

I circuiti digitali vengono realizzati utilizzando componenti chiamati **porte logiche**. Sono realizzate con componenti fisici come transistor e resistenze, ma nella progettazione dei circuiti digitali le porte logiche vengono schematizzate con i simboli riportati nella Figura 3.1 per semplificare la progettazione **astraendo** il livello di complessità della circuiteria analogica. Solamente con la porta NAND si possono realizzare tutte le altre porte (NAND è funzionalmente completo), ma le porte in generale si costruiscono singolarmente con componenti appositi. Esse implementano la **logica booleana** che conseguentemente permette di realizzare operazioni di **aritmetica binaria** per costruire unità di calcolo in componenti elettronici e processori.

I componenti elettronici molto piccoli sono sensibili al **rumore**, per ovviare al problema i valori discreti (0 e 1) nei circuiti digitali non seguono un cambiamento istantaneo di differenza di potenziale (voltaggio), ma ammettono un margine per ridurre i problemi causati dal rumore.

I componenti (transistor) con cui si costruiscono porte logiche e circuiti sono realizzati con materiali semiconduttori, che possono essere di diversi tipi. Vedremo il tipo NMOS. Un transistor è composto da materiali come gallio e silicio.

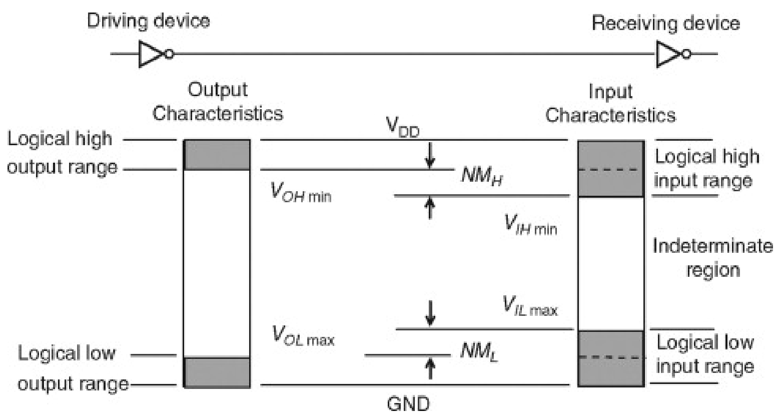


Figura 3.1: Margine di rumore nei circuiti digitali

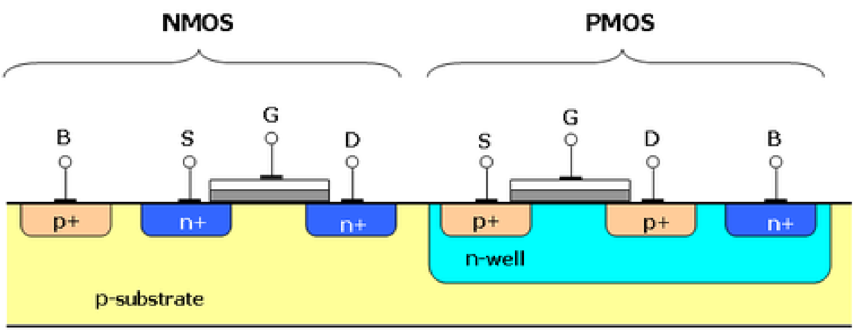
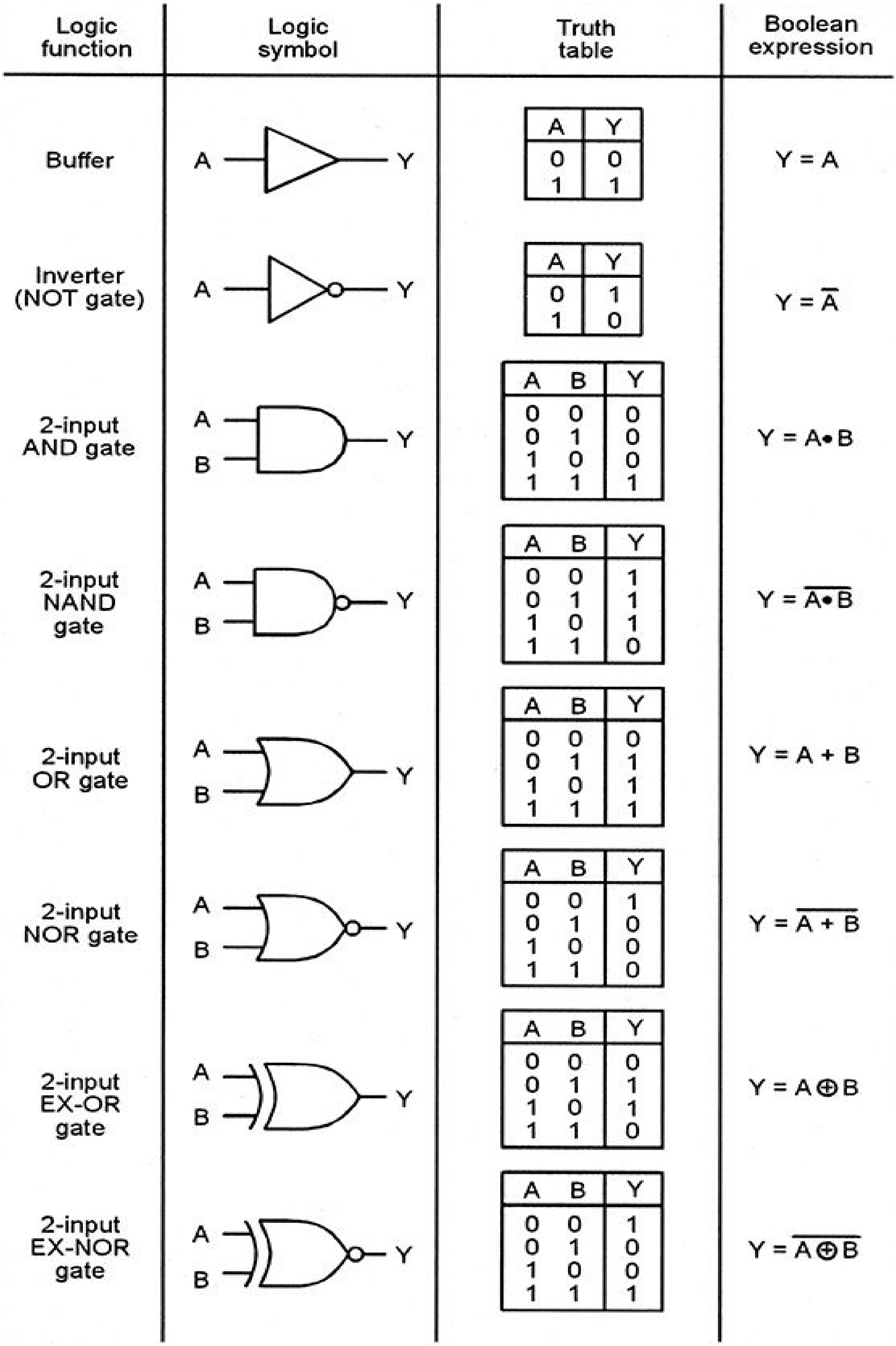


Figura 3.2: Transistor NMOS

Figura 3.3: Tabella delle porte logiche comuni



Logic function	Logic symbol	Truth table	Boolean expression
Buffer	A → Y	A Y 0 0 1 1	$Y = A$
Inverter (NOT gate)	A → Y	A Y 0 1 1 0	$Y = \overline{A}$
2-input AND gate	A, B → Y	A B Y 0 0 0 0 1 0 1 0 0 1 1 1	$Y = A \bullet B$
2-input NAND gate	A, B → Y	A B Y 0 0 1 0 1 1 1 0 1 1 1 0	$Y = \overline{A \bullet B}$
2-input OR gate	A, B → Y	A B Y 0 0 0 0 1 1 1 0 1 1 1 1	$Y = A + B$
2-input NOR gate	A, B → Y	A B Y 0 0 1 0 1 0 1 0 0 1 1 0	$Y = \overline{A + B}$
2-input EX-OR gate	A, B → Y	A B Y 0 0 0 0 1 1 1 0 1 1 1 0	$Y = A \oplus B$
2-input EX-NOR gate	A, B → Y	A B Y 0 0 1 0 1 0 1 0 0 1 1 1	$Y = \overline{A \oplus B}$

Figura 3.4: Porta NOT con transistor PMOS e NMOS

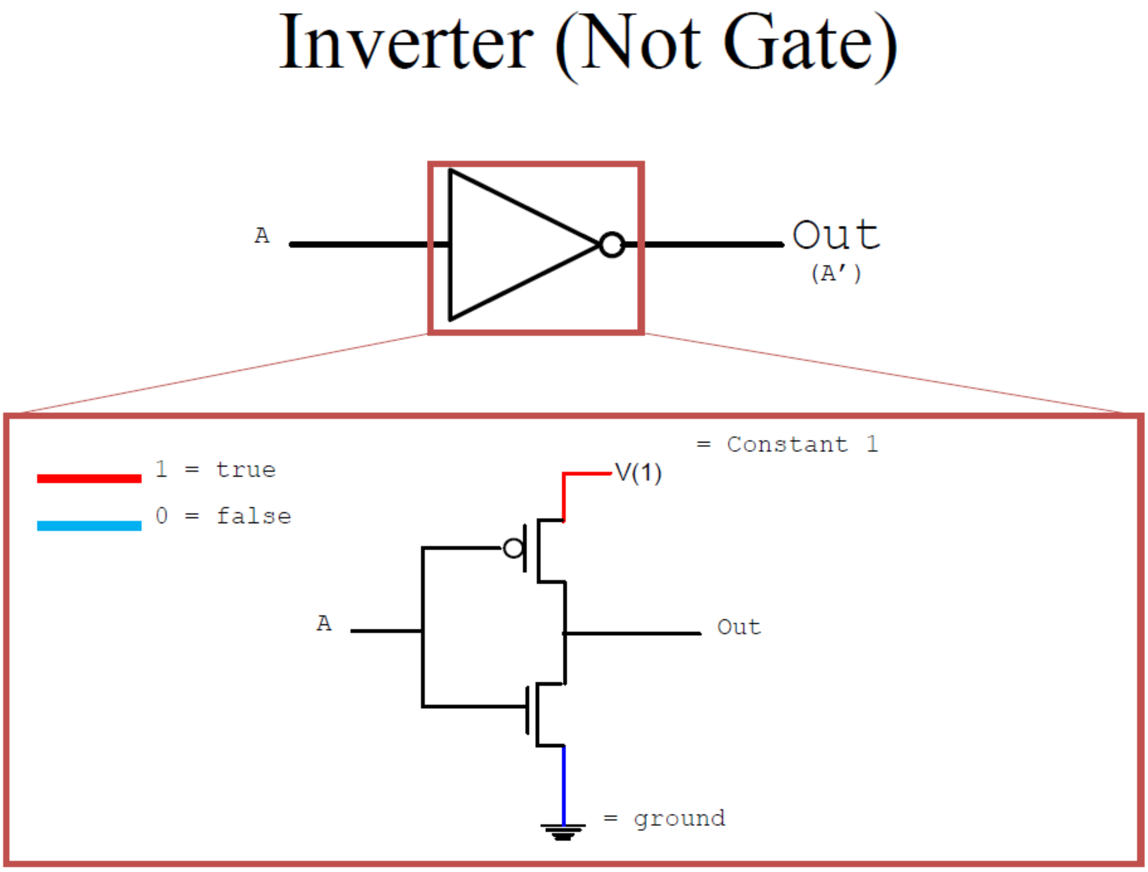
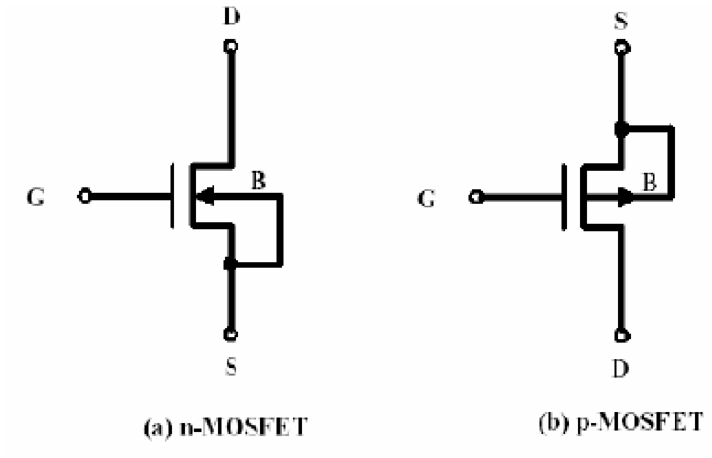


Figura 3.5: Transistor NMOS e PMOS



3.1 Algebra di Boole

Funzione	Notazione Usata	Notazione Logica
NOT(A)	$\overline{A}$	$\neg A$
AND(A,B)	$A \cdot B$	$A \wedge B$
OR(A,B)	$A + B$	$A \vee B$

Tabella 3.1: Notazione usata per l'Algebra di Boole

La forma canonica di espressioni booleane in **somma di prodotti** è:

$$z = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$$

A	B	C	z
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Tabella 3.2: Tabella di verità di z

z si può anche esprimere come prodotto di somme: $z = (A + \overline{B} + C)(\overline{A}BC)(\overline{A}\overline{B}C)$

3.2 Teoremi dell'Algebra Booleana

Breve ripasso dei teoremi della Logica Booleana.

Elemento Identità di prodotto e somma

$$A \cdot 1 = A \iff A \wedge T \equiv A$$

$$A + 0 = 0 \iff A \vee F \equiv A$$

Elemento assorbente

$$A \cdot 0 = 0 \iff A \wedge F \equiv F$$

$$A + 1 = 1 \iff A \vee T \equiv T$$

Idempotenza

$$A \cdot A = A \iff A \wedge A \equiv A$$

$$A + A = A \iff A \vee A \equiv A$$

Complemento

$$A \cdot \overline{A} = 0 \iff A \wedge \neg A \equiv F$$

$$A + \overline{A} = 1 \iff A \vee \neg A \equiv T$$

Commutatività

$$A + B = B + A$$
$$A \cdot B = B \cdot A$$

Associatività

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$
$$(A + B) + C = A + (B + C)$$

Distributività

$$(A \cdot B) + C = (A + C) \cdot (B + C)$$
$$(A + B) \cdot C = AC + BC$$

DeMorgan

$$\overline{(A + B)} = \overline{A} \cdot \overline{B} \iff \neg(A \vee B) \equiv \neg A \wedge \neg B$$
$$\overline{(A \cdot B)} = \overline{A} + \overline{B} \iff \neg(A \wedge B) \equiv \neg A \vee \neg B$$

Esempio Semplifichiamo la formula booleana $z = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$

$$\begin{cases} \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C \equiv \overline{A}\overline{B}(\overline{C} + C) \equiv \overline{A}\overline{B} \\ A\overline{B}C + ABC \equiv AC(\overline{B} + B) \equiv AC \end{cases} \tag{3.1}$$

$$\implies z = \overline{A}\overline{B} + \overline{A}BC + AC \tag{3.2}$$

Le leggi della logica Booleana ci permettono di semplificare molto i componenti realizzati con porte logiche.

3.3 Mappe di Karnaugh

Prendiamo una formula a quattro variabili $f(A, B, C, D)$, una mappa di Karnaugh può essere:

AB\CD	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	0	1	0	0
10	0	0	1	1

Tabella 3.3: Tabella di verità della mappa di Karnaugh di f

Facciamo ad esempio la mappa di Karnaugh di $z = f(A, B, C)$ vista nella sezione precedente:

A\BC	00	01	11	10
0	1	1	1	0
1	0	1	1	0

Tabella 3.4: Tabella di verità della mappa di Karnaugh di z

Possiamo riconoscere un'implicante nella seconda e terza colonna che corrisponde esattamente a C. Osserviamo un'altra implicante nella prima riga, prima e seconda colonna che corrisponde esattamente a $\overline{A}\overline{B}$. Possiamo poi sommare le implicanti per ottenere una formula equivalente a quella di partenza, ciò implica che $z = \overline{A}\overline{B} + C$

Capitolo 4

Algebra di Boole