

IN FASE DI LAVORAZIONE DI UN COMPONENTE ELETTRONICO CI SI PUÒ MUOVERE A PIÙ LIVELLI DI ASTRAZIONE:

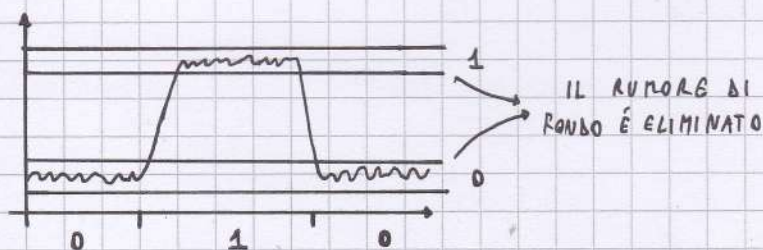
- LIVELLO FISICO: GOVERNATO DALLE LEGGI DELLA FISICA, AD ESEMPIO DELLA MECCANICA QUANTISTICA.
- LIVELLO ELETTRICO: QUI SI CONSIDERANO LE CORRENTI, LE TENSIONI E LE PROPRIETÀ DELLE RETI DI COMPONENTI.
- LIVELLO LOGICO: NON CI SI INTERESSA AL FUNZIONAMENTO INTERNO DI UNA RETE MA SEMPLICEMENTE ALLA FUNZIONALITÀ CHE IMPLEMENTA.
- LIVELLO ARCHITETTURALE: QUI SI COMBINANO RETI DI COMPONENTI PER CREARNE DI PIÙ COMPLESSE.

IMMEDIATAMENTE
IN SEMPRE IN FASE DI LAVORAZIONE È UTILE TENER PRESENTE IL LIVELLO DI ASTRAZIONE PIÙ BASSO, PER CONOSCERE LE PROPRIETÀ DEGLI OGGETTI CON CUI SI LAVORA, E QUELLO IMMEDIATAMENTE PIÙ ALTO, PER SAPERE COME RENDERE PIÙ SEMPLICE IL LAVORO DI CHI DOVRÀ LAVORARE A QUEL LIVELLO DI ASTRAZIONE.

PUÒ ESSERE UTILE IN ALCUNI CASI EFFETTUARE QUALCHE STIMA SUL CONSUMO ~~DE~~ O SU ALTRE CARATTERISTICHE DI UN COMPONENTE ELETTRONICO. IN TAL CASO SI PREFERISCE FARE STIME AD ALTO LIVELLO DI ASTRAZIONE DURANTE LA LAVORAZIONE ~~DE~~ PER EFFETTUARNE SOLO UNA, A PRODOTTO FINITO, A BASSO LIVELLO (SOLITAMENTE ELETTRICO)

DAL FENOMENO FISICO AL SEGNALE DIGITALE

IL FENOMENO FISICO È UNA VIBRAZIONE CHE EVOLVE IN MANIERA CONTINUA



A LIVELLO DIGITALE VIENE ELIMINATO IL RUMORE, A LIVELLO ANALOGICO CIÒ NON AVVIENE

VARIABILE BOOLEANA

Vittorio Alzetta

BOOLEANA DA ROVER BOOL. ESSA È UNA VARIABILE CHE PUÒ ASSUMERE 2 VALORI SOLTANTO, PER SEMPLICITÀ INDICHERO QUESTI DUE STATI CON 0 E 1.

FUNZIONE BOOLEANA

UNA FUNZIONE BOOLEANA HA COME ARGOMENTO UNA O PIÙ VARIABILI BOOLEANE E RESTITUISCE UNA VARIABILE BOOLEANA:

$$f(x_0, x_1, \dots, x_n) = 0, 1$$

PER DEFINIRE UNA FUNZIONE BOOLEANA POSSIAMO RICORRERE AD UNA **TABELLA DI VENTÀ**, CIOÈ UNA TABELLA IN CUI SI STABILISCE IL COMPORTAMENTO DELLA FUNZIONE PER OGNI COMBINAZIONE DI VALORI DELLE VARIABILI NELL'ARGOMENTO.

x_0	x_1	$f(x_0, x_1)$
0	0	0
0	1	1
1	0	1
1	1	0

IN QUESTO CASO, POICHÉ ABBIAMO CONSIDERATO TUTTI I CASI, DIREMO CHE f È **COMPLETAMENTE SPECIFICATA**.

NON SEMPRE SAREMO INTERESSATI AL VALORE CHE f ASSUME PER OGNI COMBINAZIONE DELLE VARIABILI, IN TAL CASO USEREMO LA NOTAZIONE:

x_0	x_1	$f(x_0, x_1)$
0	0	0
0	1	1
1	0	1
1	1	-

→ DON'T CARE (IL RISULTATO CI SARÀ PERO')

FUNZIONE COMPLEMENTARE

Vittorio Alzetta

LA FUNZIONE COMPLEMENTARE q di f SARÀ COSTRUITA IN MODO CHE:

- QUANDO f VALE 0 q VARrà 1
- QUANDO f VALE 1 q VARrà 0

FUNZIONE EQUIVALENTE

LA FUNZIONE EQUIVALENTE q di f FORNISCE SEMPRE GLI STESSI RISULTATI DI f .

ALGEBRA COMMUTAZIONALE

L'ALGEBRA COMMUTAZIONALE SI COMPONE DI ALCUNI ELEMENTI:

- 2 VALORI: $\{0, 1\}$

- L'OPERATORE LOGICO **OR** (INDICATO CON "+") È DEFINITO COME:

x_0	x_1	$x_0 + x_1$
0	0	0
0	1	1
1	0	1
1	1	1

È VERA QUANDO ALMENO UNA DELLE VARIABILI È VERA

- L'OPERATORE LOGICO **AND** (INDICATO CON ".") È DEFINITO COME:

x_0	x_1	$x_0 \cdot x_1$
0	0	0
0	1	0
1	0	0
1	1	1

È VERA QUANDO TUTTE LE VARIABILI SONO VERE

- L'OPERATORE LOGICO **NOT** (INDICATO CON "-") È DEFINITO DA:

x_0	$\overline{x_0}$
0	1
1	0

- L'EQUIVALENZA "="

IDEMPOTENZA

$$x + x = x$$

$$x \cdot x = x$$

x	x+x
0	0
1	1

x	x · x
0	0
1	1

PER LA DIMOSTRAZIONE ABBIAMO USATO IL METODO DELL'INDUZIONE COMPLETA, CIOÈ ABBIAMO CREATO UNA TABELLA E ABBIAMO CONFRONTATO TUTTE LE COMBINAZIONI.

COSTANTE

$$x + 0 = x$$

0 È OPERATORE NEUTRO DELL'OR

$$x + 1 = 1$$

$$x \cdot 0 = 0$$

$$x \cdot 1 = x$$

1 È OPERATORE NEUTRO DELL'AND

COMMUTATIVA

$$x + y = y + x$$

$$x \cdot y = y \cdot x$$

x	y	x+y	y+x
0	1	1	1
0	0	0	0
1	1	1	1
1	0	1	1

x	y	x · y	y · x
0	1	0	0
0	0	0	0
1	1	1	1
1	0	0	0

ASSOCIATIVA

$$(x + y) + z = x + (y + z)$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

x	y	z	(x+y)	(y+z)	(x+y)+z	x+(y+z)
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

NON SERVE BUNAVE SPECIFICARE LA

$$\text{L'ORDINE: } (x \cdot y) \cdot z = x \cdot y \cdot z$$

SE DOVESSIMO CALCOLARE IL RISULTATO DELL'OPERAZIONE LOGICA :

$$X_0 + X_1 + X_2 + X_3 + X_4 + \dots + X_{n-1}$$

POTREMMO PENSARE DI ESEGUIRE ORDINATAMENTE TUTTE LE OPERAZIONI, MA CIÒ comporterebbe INVENTI QUANTI RISORSE CONSUMI DI RISORSE IN TERMINI DI TEMPO.

SI POTREBBE, TENENDO AL FATTO CHE AND E OR GODONO DELL'ASSOCIATIVITÀ, ESEGUIRE A DUE A DUE LE OPERAZIONI, RIDUCENDO IL TEMPO DI ESECUZIONE DA n A $\log_2 n$.

DISTRIBUTIVA

$$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$$

$$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$$

X	Y	Z	X·Y	X·Z	Y+Z	X·(Y+Z)	X·Y + X·Z
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	0
1	0	1	0	1	1	1	1
1	1	0	1	0	1	1	1
1	1	1	1	1	1	1	1

PRECEDENZA DEGLI OPERATORI

NOT → AND → OR, Dunque $(X + Y) \cdot (X + Z) = X + Y \cdot X + Z$

COMPLEMENTAZIONE

$$X + \bar{X} = 1$$

$$X \cdot \bar{X} = 0$$

X	$\bar{X}$	$X + \bar{X}$	$X \cdot \bar{X}$
0	1	1	0
1	0	1	0

ASSOCIAMENTO

$$X + X \cdot Y = X$$

$$X \cdot (X + Y) = X$$

X	Y	XY	X+XY	X+Y	X·(X+Y)
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	1	1
1	1	1	1	1	1

SUPPONENDO VALIDA LA 1 POSSIAMO:

$$X \cdot (X + Y) = X \cdot X + X \cdot Y = X + X \cdot Y = X \quad \square$$

INVOLUZIONE

$$\overline{\overline{X}} = X \quad (\text{DOPPIA RELAZIONE})$$

DUALITÀ

$$\overline{X+Y} = \overline{X} \cdot \overline{Y}$$

$$\overline{X \cdot Y} = \overline{X} + \overline{Y}$$

X	Y	X+Y	$\overline{X+Y}$	$\overline{X} \cdot \overline{Y}$	X·Y	$\overline{X \cdot Y}$	$\overline{X} + \overline{Y}$
0	0	0	1	1	0	1	1
0	1	1	0	0	0	1	1
1	0	1	0	0	0	1	1
1	1	1	0	0	1	0	0

NUOVI OPERATORI LOGICI: NOR, NAND

INTRODUCIAMO DUE NUOVI OPERATORI LOGICI:

• L'OPERATORE **NOR** (ossia $\overline{X+Y}$)

• L'OPERATORE **NAND** (ossia $\overline{X \cdot Y}$)

PROPRITÀ

• $X \text{ NOR } X = \overline{X}$

$X \text{ NAND } X = \overline{X}$

X	$\overline{X}$	X+X	$\overline{X+X}$	X·X	$\overline{X \cdot X}$
0	1	0	1	0	1
1	0	1	0	1	0

• $X \text{ NOR } 0 = \overline{X}$

$X \text{ NOR } 1 = 0$

$X \text{ NAND } 0 = 1$

$X \text{ NAND } 1 = \overline{X}$

• NAND E NOR NON GODONO DELLA PROPRIETÀ ASSOCIATIVA, INFATTI:

$$(X \text{ NOR } Y) \text{ NOR } Z \neq X \text{ NOR } (Y \text{ NOR } Z)$$

$$(X \text{ NAND } Y) \text{ NAND } Z \neq X \text{ NAND } (Y \text{ NAND } Z)$$

• VALE LA PROPRIETÀ **COMMUTATIVA**:

$$X \text{ NOR } Y = Y \text{ NOR } X$$

$$\text{E } X \text{ NAND } Y = Y \text{ NAND } X$$

LO XOR, o "OR ESCLUSIVO", È VERO QUANDO O UNA O L'ALTRA OPERAZIONE È VERA, È COSÌ DEFINITO:

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

ANALOGAMENTE DEFINIAMO LO **XNOR** $(\overline{X \oplus Y})$, CIOÈ LA NEGAZIONE DELLO XOR.

PROPRIETÀ

SIA PER LO XOR CHE PER LO XNOR VALE LA PROPRIETÀ COMMUTATIVA, DUNQUE:

$$(X_1 \oplus X_2) \oplus X_3 = X_1 \oplus (X_2 \oplus X_3)$$

INOLTRE SI OSSERVA CHE $X_1 \oplus X_2 \oplus \dots \oplus X_n$ È VERO QUANDO IL NUMERO DI VARIABILI PAR A 1 È DISPARI.

È DUNQUE FALSA QUANDO IL NUMERO DELLE VARIABILI A 1 È PAR.

GLI OPERATORI UNIVERSALI

OSSERVIAMO CHE A PARTIRE DALL'OPERATORE NOR POSSIAMO RICAVERE:

• NOT : $X \text{ NOR } X = \overline{X + X} = \bar{X}$

• AND : POICHÉ $\overline{\bar{X} + \bar{Y}} = X \cdot Y$ POSSIAMO SCRIVERE

$\bar{X} \text{ NOR } \bar{Y}$ CHE È UGUALE A DIRE $(\overline{X + X}) \text{ NOR } (\overline{Y + Y}) =$
 $= (X \text{ NOR } X) \text{ NOR } (Y \text{ NOR } Y)$

• OR : POICHÉ $\overline{\bar{X} \cdot \bar{Y}} = X + Y$ POSSIAMO SCRIVERE

$\overline{(X \text{ NOR } Y)} = (X \text{ NOR } Y) \text{ NOR } (X \text{ NOR } Y) = X + Y$

MENTRE A PARTIRE DALL'OPERATORE NAND POSSIAMO RICAVARE :

Vittorio Alzetta

• NOT : $X \text{ NAND } X = \overline{X \cdot X} = \overline{X}$

• AND : $\overline{\overline{X \cdot y}} = X \cdot y$ DUNQUE POSSIAMO SCRIVERE

$$(\overline{X \cdot y}) \text{ NAND } (\overline{X \cdot y}) = (X \text{ NAND } y) \text{ NAND } (X \text{ NAND } y) = X \cdot y$$

• OR : $\overline{\overline{X} \cdot \overline{y}} = \overline{\overline{X}} + \overline{\overline{y}} = X + y$ DUNQUE AVREMO

$$(\overline{X \text{ NAND } X}) \cdot (\overline{y \text{ NAND } y}) = (X \text{ NAND } X) \text{ NAND } (y \text{ NAND } y)$$

ABBIAMO RICAVATO TUTTI GLI OPERATORI LOGICI PARTENDO DAL NOR E DAL NAND, PER QUESTO MOTIVO ESSI VERRANNO CHIAMATI **OPERATORI UNIVERSALI**.

DIRAMO INOLTRE CHE I SEGUENTI INSIEMI :

$$\{ \text{OR}, \text{NOT} \} \quad \{ \text{AND}, \text{NOT} \} \quad \{ \text{NOR} \} \quad \{ \text{NAND} \}$$

DAL MOMENTO CHE PERMETTONO DI RICAVARE TUTTI GLI OPERATORI ELEMENTARI SONO **FUNZIONALMENTE COMPLETI**.

IN QUESTO PARAGRAFO DAREMO UN SIMBOLO GRAFICO AGLI OPERATORI LOGICI INTRODOTTI FINO AD ORA.

AND		NOT		OR	
NAND	 NEGAZIONE IN USCITA				
	 NEGAZIONE IN ENTRATA				
XOR		XNOR			

DEFINIRÒ ORA ALCUNI TERMINI CHE SARANNO UTILI NELLO STUDIO DELLE RETI LOGICHE.

LETTERALE: È IL NOME DI UNA VARIABILE CHE PUÒ COMPARE IN UN'ESPRESSIONE, ESSA PUÒ ESSERE IN FORMA DIRETTA O NEGATA:

$$f(x_0, x_1) = \bar{x}_0 x_1 + x_0 \bar{x}_1, \text{ dove } x_0, \bar{x}_0, x_1, \bar{x}_1 \text{ SONO LETTERALI}$$

IMPLICANTE: È UN PRODOTTO DI LETTERALI TALI CHE SE TALE PRODOTTO FOSSE PARI A 1 ALLORA LA FUNZIONE ASSUMEREBBE IL VALORE 1.

NELL'ESEMPIO DI PRIMA SE $\bar{x}_0 x_1 = 1$ ALLORA $f(x_0, x_1) = 1 + x_0 \bar{x}_1 = 1$ E AVINBI $\bar{x}_0 x_1$ È UN IMPLICANTE.

MENTRE SE $x_0 \bar{x}_1 = 1$ ALLORA $f(x_0, x_1) = \bar{x}_0 x_1 + 1 = 1$ E AVINBI $x_0 \bar{x}_1$ È IL SECONDO IMPLICANTE.

MINTERMINE: È UN IMPLICANTE CHE CONTIENE TUTTI I LETTERALI (SE UN LETTERALE COMPARE SIA IN FORMA DIRETTA CHE IN FORMA NEGATA BASTA CHE IL MINTERMINE CONTENGA UNA DELLE DUE FORME)

OSSERVIAMO ANCH'ESSE CHE UNA FUNZIONE A n VARIABILI HA 2^n STATI POSSIBILI,
CONSIDERIAMO IL SEGUENTE ESEMPIO,

$$f(x_0, x_1, x_2) = \bar{x}_0 x_1 x_2 + x_0 x_2$$

n	x_0, x_1, x_2	$f(x_0, x_1, x_2)$
0	0 0 0	0
1	0 0 1	0
2	0 1 0	0
3	0 1 1	1
4	1 0 0	0
5	1 0 1	1
6	1 1 0	0
7	1 1 1	1

DEFINIAMO **ON-SET** L'INSIEME DEI PUNTI IN CUI f VALE 1;

$$\sum (3, 5, 7) \rightarrow \text{SONO MINTERMINI}$$

DEFINIAMO **OFF-SET** L'INSIEME DEI PUNTI IN CUI f VALE 0:

TABELLA DI VERITÀ

$$\prod (0, 1, 2, 4, 6)$$

EVENTUALMENTE POSSIAMO DEFINIRE IL **DC-SET** (~~NO~~ DON'T CARE SET), OVVERO
L'INSIEME DEI PUNTI IN CUI NON È SPECIFICATO IL VALORE DELLA FUNZIONE.

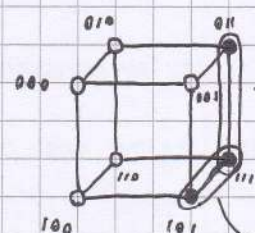
SI OSSERVI CHE DALLA TABELLA DI VERITÀ POSSIAMO RICAVARE ALCUNI IMPLICANTI:

$$011 = \bar{x}_0 x_1 x_2 \quad 101 = x_0 \bar{x}_1 x_2 \quad 111 = x_0 x_1 x_2 \quad \left. \vphantom{\begin{matrix} 011 \\ 101 \\ 111 \end{matrix}} \right\} \text{SONO 3 MINTERMINI}$$

SE QUESTE COMBINAZIONI DI LETTERE SONO VERE ALLORA LO SARÀ ANCHE $f(x_0, x_1, x_2)$.

IMPLICANTE PRIMO: È UN IMPLICANTE A DIMENSIONE MASSIMA, ~~NE~~ ~~NO~~ ~~APPENA~~ ~~FAF~~ 3
~~ESEMPL~~ CHE NON PUÒ ESSERE ESPANSO A SOTTOCUBI PIÙ GRANDI.

MAXTERMINE: È UN PUNTO IN CUI LA FUNZIONE ASSUME IL VALORE 0



IMPLICANTE PRIMO: $\bar{x}_0 x_1 x_2 + x_0 x_1 x_2 = x_1 x_2 (\bar{x}_0 + x_0) = x_1 x_2$

IMPLICANTE PRIMO: $x_0 \bar{x}_1 x_2 + x_0 x_1 x_2 = x_0 x_2 (\bar{x}_1 + x_1) = x_0 x_2$

CHE SUGGERE SE COMBINIAMO IN OR. TUTTI I MINTERMINI! TROVIAMO f .

IMPLICANTE PRIMO ESSENZIALE : È UN IMPLICANTE PRIMO CON ALMENO UN NODO NON CONDIVISO CON ALTRI IMPLICANTI (PRMI SI INTENDE, CREDO).

ENTRAMBI GLI IMPLICANTI PRIMI DELL'ESEMPIO SONO ANCHE ESSENZIALI, INATTI IL PRIMO HA IL NODO 101 NON CONTENUTO IN ALTRI IMPLICANTI (PRMI) E IL SECONDO HA IL NODO 011 NON CONTENUTO IN ALTRI IMPLICANTI (PRMI).

PERCHÉ CI INTERESSIAMO ALLO STUDIO DEGLI IMPLICANTI PRIMI ESSENZIALI?

DI FATTO QUANDO DOVREMO SCRIVERE UNA FUNZIONE A VARABILI BOOLEANE VORREMO INSERIRE IL MINOR NUMERO DI LETTERE POSSIBILE. DUNQUE ~~INSERIREMO~~ COMBINEREMO IN ORGLI IMPLICANTI ~~MINOR POSSIBILI~~ PER RAGGIUNGERE TALE SCOPO, DI QUESTI IMPLICANTI CONSIDEREREMO QUELLI PRMI E FAREMO IN MODO DI METTERE ANCHE TUTTI QUELLI ESSENZIALI.

L'OBIETTIVO È QUELLO DI COPRIRE TUTTA LA FUNZIONE CON IMPLICANTI ESSENZIALI E PRMI.

TEOREMA DELL'ESPANSIONE DI SHANNON

IL ~~TEOREMA~~ TEOREMA DELL'ESPANSIONE DI SHANNON PERMETTE DI RICOSTRUIRE L'ESPRESSIONE DI UNA FUNZIONE NEL SEGUENTE MODO:

$$f(x_0, x_1, \dots) = \bar{x}_0 \cdot f(0, x_1, \dots) + x_0 \cdot f(1, x_1, \dots)$$

VEDIAMOLO APPLICATO AD UN CASO CONCRETO: SUPPONIAMO DI VOLER CALCOLARE L'ESPRESSIONE DELLA FUNZIONE CON LA SEGUENTE TABELLA DI VERITÀ:

	A	B	C	f
0.	0	0	0	1
1.	0	0	1	1
2.	0	1	0	0
3.	0	1	1	0
4.	1	0	0	1
5.	1	0	1	1
6.	1	1	0	0
7.	1	1	1	0

$$f = \sum (0, 1, 4, 5)$$

APPLICHIAMO IL TEOREMA DI SHANNON PIÙ VOLTE :

$$\begin{aligned}
 f(A, B, C) &= \bar{A} \cdot f(0, B, C) + A \cdot f(1, B, C) \\
 &= \bar{A} \cdot [\bar{B} \cdot f(0, 0, C) + B \cdot f(0, 1, C)] + A \cdot [\bar{B} \cdot f(1, 0, C) + B \cdot f(1, 1, C)] \\
 &= \bar{A} \cdot \{ \bar{B} [\bar{C} \cdot f(0, 0, 0) + C \cdot f(0, 0, 1)] + B [\bar{C} \cdot f(0, 1, 0) + C \cdot f(0, 1, 1)] \} + \\
 &\quad + A \cdot \{ \bar{B} [\bar{C} \cdot f(1, 0, 0) + C \cdot f(1, 0, 1)] + B [\bar{C} \cdot f(1, 1, 0) + C \cdot f(1, 1, 1)] \} \\
 &= \bar{A} \cdot \{ \bar{B} [\bar{C} \cdot 1 + C \cdot 1] + B [\bar{C} \cdot 0 + C \cdot 0] \} + A \cdot \{ \bar{B} [\bar{C} \cdot 1 + C \cdot 1] + B [\bar{C} \cdot 0 + C \cdot 0] \} \\
 &= \bar{A} [\bar{B} \cdot (\bar{C} + C)] + A [\bar{B} (\bar{C} + C)] = \bar{A} \cdot \bar{B} + A \cdot \bar{B} \\
 &= \bar{B} (\bar{A} + A) = \bar{B} \cdot 1 = \bar{B}
 \end{aligned}$$

MAPPA DI KARNAUGH

SAREBBE UTILE AVERE UN METODO VISUALE PER ARRIVARE ALLE STESSA CONCLUSIONI CHE
POCO FA ABBIAMO RAGGIUNTO CON MOLTI CALCOLI.

CONSIDERIAMO LA SEGUENTE FUNZIONE:

$$\begin{aligned}
 y(x_0, x_1) &= \bar{x}_0 \bar{x}_1 + \bar{x}_0 x_1 + x_0 x_1 = \bar{x}_0 (\bar{x}_1 + x_1) + x_0 x_1 \\
 &= \bar{x}_0 \cdot 1 + x_0 x_1 = \bar{x}_0 + x_0 x_1
 \end{aligned}$$

$x_0 \ x_1$	y
0 0	1
0 1	1
1 0	0
1 1	1

TABELLA DI VERITÀ DI $y(x_0, x_1)$

CONSTRUIAMO ORA UNA MAPPA CHE RIASSUMA I RISULTATI DELLA TABELLA DI VERITÀ:

	x_0	0	1
x_1	0	1	0
	1	1	1

NOTIAMO CHE INDIFFERENTEMENTE DAL VALORE DI x_1 , SE $x_0 = 0$ ALLORA $f(x_0, x_1) = 1$,
ABBIAMO TROVATO UN IMPLICANTE: $\bar{x}_0$! AGGIUNGIAMO ORA IL SECONDO IMPLICANTE CHE SI HA

NEL CASO IN CUI $X_0 = 1$ E $X_1 = 1$, LIO È $X_0 X_1$ SARÀ IL NOSTRO SECONDO IMPLICANTE.

PERCIÒ CONCLUDIAMO CHE $f(X_0, X_1) = \bar{X}_0 + X_0 X_1$

OSSERVIAMO PERÒ CHE CONSIDERANDO COME SECONDO IMPLICANTE UN IMPLICANTE PRIMO ESSENZIALE POSSIAMO OTTENERE UN'ESPRESSIONE CON 2 LETTERE ANZICHÉ 3!

INFATTI, INDIFFERENTEMENTE DAL VALORE DI X_0 SE $X_1 = 1$ ALLORA $f(X_0, X_1) = 1$ E DUNQUE X_1 SARÀ IL NOSTRO NUOVO IMPLICANTE.

IN CONCLUSIONE $f(X_0, X_1) = \bar{X}_0 + X_1$

K-MAP A 3 VARABILI

LA MAPPA CHE ABBIAMO DISEGNATO PRIMA FACEVA RIFERIMENTO AD UNA FUNZIONE A 2 VARABILI.

IMPRENDIAMO ORA L'ESEMPIO ~~ANTERIORE~~ f FATTO PER COMPRENDERE IL TEOREMA DELL'ESPANSIONE DI SHANNON.

DISEGNAMO LA K-MAP A 3 VARABILI:

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

	A	B	C	f
0.	0	0	0	1
1.	0	0	1	1
2.	0	1	0	0
3.	0	1	1	0
4.	1	0	0	1
5.	1	0	1	1
6.	1	1	0	0
7.	1	1	1	0

OCCORRE PREMETTERE CHE UN IMPLICANTE PUÒ ESSERE ESPANSO AD UN ALTRO DA n LETTERE PUNCHÉ QUEST'ULTIMO ABBI A $n = 2^k$ LETTERE.

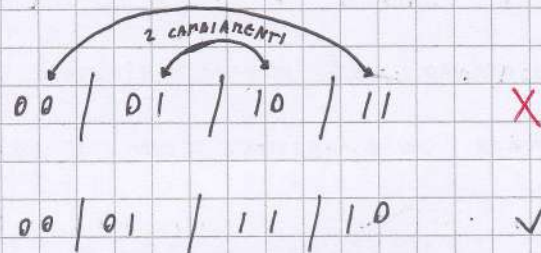
CONSIDERIAMO QUINDI L'IMPLICANTE $(m_0 + m_1)$ E NOTIAMO CHE NONOSTANTE LA VARIABILE C CAMBIA $f(A, B, C) = 1$ E QUINDI ECCO IL PRIMO IMPLICANTE: $\bar{A} \bar{B}$

POSSIAMO PERÒ SPINGERCI OLTRE, RADDOPPIANDO LA GRANDEZZA DELL'IMPLICANTE A 4!

NEL BLOCCO $(m_1 + m_2 + m_4 + m_5)$ COS'È CHE VARIA (E QUINDI PUÒ ESSERE SCARTATO)

E COSA RIMANE COSTANTE (E QUINDI VA CONSIDERATO)? L'UNICA COSTANTE È $B = 0$, CHE CI ASSICURA CHE $f(A, B, C)$, QUINDI LA NOSTRA ESPRESSIONE SARÀ $f(A, B, C) = \bar{B}$.

OSSERVIAMO, PRIMA DI ANDARE OLTRE, CHE NELLA COSTRUZIONE DELLA K-MAP BISOLNA IMPOSTARE CHE NEL PASSAGGIO DA UNA CELLA AD UNA ADIACENTE CAMBI IL VALORE DI UNA SOLA VARIABILE:



K-MAP A 4 VARIABILI

RAPPRESENTIAMO GRAFICAMENTE UNA K-MAP PER FUNZIONI A 4 VARIABILI:

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

- GLI SPAZI VUOTI INDICANO LA PRESENZA DI UNO 0
- I '-' INDICANO LA PRESENZA DI UN DON'T CARE
- È POSSIBILE ANDARE OLTRE IL BORDO PER UNIRSI AL BORDO OPPOSTO

CONSIDERIAMO COME PRIMO IMPLICANTE PRIMO ESSENZIALE L'UNIONE IN OR DI m_8 E m_{10} , NOTIAMO CHE A TUTTARE COSTANTI SONO $A = 1$, $B = 0$ E $D = 0$, PERTANTO IL NOSTRO PRIMO IMPLICANTE SARÀ: $A \bar{B} \bar{D}$.

OSSERVIAMO CHE I DON'T CARE POSSONO ESSERE SFRUTTATI ANCHE, ASSUNTI COME 1, PER CREARE L'IMPLICANTE PRIMO ESSENZIALE $(m_4 + m_5 + m_7 + m_6)$ DOVE CI

ACCORIAMO CHE RIMANGONO COSTANTI SOLO $A = 0$ E $B = 1$, PERTANTO IL SECONDO IMPLICANTE SARÀ: $\bar{A} B$.

IN CONCLUSIONE $f(A, B, C, D) = \bar{A} B + A \bar{B} \bar{D}$

LE K-MAP SI POSSONO ESTENDERE A 5 VARIABILI ACCOSTANDO 2 K-MAP A 4 VARIABILI; AD UNA SARA' ASSOCIATO IL VALORE 0 DI UNA DELLE 5 VARIABILI, ALL'ALTRA IL VALORE 1 DELLA VARIABILE SCELTA.

QUESTE 2 MAPPE PERMETTONO IL PASSAGGIO TRA PUNTI CON LA STESSA COORDINATA, E' COME SE FOSSE SOVRAPPOSTE IN UNO SPAZIO TRIDIMENSIONALE.

$X_3 X_4 \backslash X_1 X_2$	00	01	11	10
00	1			
01		-	-	
11		1	1	1
10	1			

$X_0 = 0$

$X_3 X_4 \backslash X_1 X_2$	00	01	11	10
00	1			
01				
11				1
10	1			

$X_0 = 1$

CONSIDERO INIZIALMENTE L'IMPLICANTE COMPOSTO DA 4 MINITERMINI 00101, 00111, 01101, 01111. OTTENIAMO $\bar{X}_0 X_2 X_4$

POI MI ACCORRO CHE ANCHE IL MINITERMINE 01011 E 11011 SONO AMICENTI, E COSI' LI UNISCO. OTTENIAMO $X_1 \bar{X}_2 X_3 X_4$

IN ULTIMO NE CONSIDERO UNO DA 4 CHE SI ESPANDE IN ENTRAMBE LE MAPPE: 00000, 00010, 10000, 10010. OTTENIAMO $\bar{X}_1 \bar{X}_2 \bar{X}_4$

ADUNQUE LA NOSTRA f SARA' :

$$f(X_0, X_1, X_2, X_3, X_4) = \bar{X}_0 X_2 X_4 + \bar{X}_1 \bar{X}_2 \bar{X}_4 + X_1 \bar{X}_2 X_3 X_4$$

SIMILMENTE A QUANTO FATTO CON LA K-MAP A 5 VARABILI DOVREMO DISEGNARE PIÙ K-MAP (4) CHE NE FORMERANNO 1 PIÙ GRANDE A CUI ASSOCIEREMO LE VARABILI 2 VARABILI RIMANENTI (4 NELLE K-MAP + 2 ESTERNE).

$X_4 X_5 \backslash X_2 X_3$		$X_2 X_3$							
		00	01	11	10	00	01	11	10
00				1		00		1	
01			1			01		1	
11						11			
10	-	-	-	-		10	-	1	1
		00	01	11	10				
00				-		00		1	
01			1			01		-	
11						11			
10	1	1	1	1		10	1	1	1

$$X_0 = 0$$

$$X_0 = 1$$

$$X_1 = 0$$

$$X_1 = 1$$

ANCHE QUI, COME PRIMA, È POSSIBILE PASSARE DA UNA MAPPA AD UN'ALTRA ADIACENTE SE SI RIMANE SULLE STESSA COORDINATE.

PERTANTO AVREMO UN PRIMO IMPLICANTE ALLE COORDINATE 001101, 001100 CHE SI ESPANDERÀ AD UNO DI 8 ELEMENTI NELLE ALTRE TABELLE, DIVENTANDO: $X_2 X_3 \bar{X}_4$.

IL SECONDO IMPLICANTE AVRÀ 16 ELEMENTI E SARÀ: $X_4 \bar{X}_5$

IN CONCLUSIONE L'ESPRESSIONE DELLA NOSTRA FUNZIONE SARÀ:

$$f(X_0, X_1, X_2, X_3, X_4, X_5) = X_2 X_3 \bar{X}_4 + X_4 \bar{X}_5$$

LA DISTANZA DI HAMMING

Vittorio Alzetta

SCRIVIAMO ANZITUTTO I PRIMI 8 NUMERI DECIMALI IN BINARIO, SEPARANDO NE LE CIPRE:

K	b_2	b_1	b_0	H
0	0	0	0	1
1	0	0	1	2
2	0	1	0	1
3	0	1	1	3
4	1	0	0	1
5	1	0	1	2
6	1	1	0	1
7	1	1	1	3

PER SEMPLICITÀ HO AGGIUNTO ALLA TABELLA UNA COLONNA RAPPRESENTANTE LA DISTANZA DI HAMMING TRA IL NUMERO CORRENTE E IL SUCCESSIVO, OVVERO IL NUMERO DI BIT CHE CAMBIANO VALORE NEL PASSAGGIO AL NUMERO SUCCESSIVO.

CI CHIEDIAMO ORA SE SIA POSSIBILE OTTIMIZZARE LA DISTANZA DI HAMMING FACENDO IN MODO CHE SIA SEMPRE PAR A 1. EBBENE, GRAZIE ALLA CODIFICA DI GRAY È POSSIBILE.

DI SEGUITO UN ESEMPIO PER I PRIMI 16 NUMERI NATURALI:

	g_3	g_2	g_1	g_0	H
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	1	1
3	0	0	1	0	1
4	0	1	1	0	1
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	1
14	1	0	0	1	1
15	1	0	0	0	1

QUESTA CODIFICA PERMETTE DI MINIMIZZARE IL CONSUMO ENERGETICO NEL CONTEGGIO.

METODO DI QUINE - MCCLUSKEY

QUESTO METODO PER LA SINTESI DI FUNZIONI LOGICHE PERMETTE DI OPERARE CON FUNZIONI A PIÙ DI 6 VARIABILI (MEGLIO DELLE MAPPE DI KARNAUGH).

DETERMINIAMO ANZITUTTO L'ON-SET DI UNA FUNZIONE f E DISEGNIAMO LA K-MAP CORRISPONDENTE (QUESTO PER COMPRENDERE MEGLIO IL METODO DI QUINE-MCCLUSKEY):

$X_2 X_1 \backslash X_0 X_3$	00	01	11	10
00			1	1
01	1	1	1	1
11	1			
10				

$$f(X_0, X_1, X_2, X_3) = \sum (1, 3, 5, 8, 9, 12, 13)$$

SI OSSERVI CHE NEI RAGGRUPPAMENTI CHE EFFETTUIAMO NELLA K-MAP TRA UN ELEMENTO E UN ALTRO CAMBIA SEMPRE E SOLO UN BIT, SE NEL MINTERMINE DI PARTENZA CI SONO n UNO ALLORA IN QUELLI ADIACENTI CI SARANNO $n+1$ UNO O $n-1$ UNO.

TENENDO QUESTO A MENTE CREIAMO UNA TABELLA CHE DIVIDA I MINTERMINI IN BASE AL NUMERO DI UNO:

		X_0	X_1	X_2	X_3	
QUESTI SIMBOLI SONO STATI INSERITI PER LA CREAZIONE DELLA PROSSIMA TABELLA	✓	m_1	0	0	0	1
	✓	m_8	1	0	0	0
	✓	m_3	0	0	1	1
	✓	m_5	0	1	0	1
	✓	m_9	1	0	0	1
	✓	m_{12}	1	1	0	0
	✓	m_{13}	1	1	0	1
						1 UNO
						2 UNO
						3 UNO

PER PROCEDERE DOVREMO ACCOPPIARE MINTERMINI DI GRUPPI ADIACENTI TRA CUI VARRA UN SOLO BIT. NEL FARE CIÒ PRENDEREMO NOTA DI QUELLI ACCOPPIATI E DI QUELLI NON ACCOPPIATI

NELLA PROSSIMA TABELLA CONTRASSEGNERO' CON "-" LA CELLA CHE VARIA TRA
LA RAPPRESENTAZIONE DI UN MINITERMINE E QUELLA DELL'ALTRO.

	X_0	X_1	X_2	X_3			
	0	0	-	1	m_1, m_3	A	È RIMASTO NON COBERTO PERTANTO SARÀ IL NOSTRO PRIMO IMPLICANTE
ACCOPPIAMENTI TRA	0	-	0	1	m_1, m_5	✓	
GRUPPO 1 E GRUPPO 2	-	0	0	1	m_1, m_3	✓	
	1	0	0	-	m_8, m_9	✓	
	1	-	0	0	m_8, m_{12}	✓	
	-	1	0	1	m_5, m_{13}	✓	
ACCOPPIAMENTI TRA	1	-	0	1	m_9, m_{13}	✓	
GRUPPO 2 E GRUPPO 3	1	1	0	-	m_{12}, m_{13}	✓	

ORA PROCEDIAMO CON LA CREAZIONE DELLA PROSSIMA TABELLA, FACENDO ATTENZIONE AL
SIMBOLO "-", CHE NON VA INTESO COME UN DON'T CARE MA COME UN VALORE
SPECIFICO (COME 0 E 1). SIAMO DI NUOVO INTERESSATI ALI ~~CON~~ ACCOPPIAMENTI CON
CAMBIAMENTO DI 1 SOLA VARIABILE:

	X_0	X_1	X_2	X_3	
B	-	-	0	1	m_1, m_5, m_9, m_{13}
C	1	-	0	-	m_8, m_9, m_{12}, m_{13}

ANUVATI A QUEST'ULTIMA TABELLA ABBIAMO TROVATO TUTTI GLI IMPLICANTI DI f , E

~~PROPRIO VARIABILI~~

$$f(X_0, X_1, X_2, X_3) = A + B + C$$

$$f(X_0, X_1, X_2, X_3) = \bar{X}_0 \bar{X}_1 X_3 + \bar{X}_2 X_3 + X_0 X_2$$

PER RICAVARE LA FORMULA F OCCORRE INDIVIDUARE, TRA GLI ACCOPPIAMENTI IN
MINTERMINI FATTI, QUELLI CHE COPRONO TUTTI I MINTERMINI DELL'ON-SET.

	A	B	C
m_1	X	X	
m_3	X		
m_5		X	
m_8			X
m_9		X	X
m_{12}			X
m_{13}		X	X

POICHE A È L'UNICO IMPLICANTE A COMPRENDERE IL MINTERMINE 3, B L'UNICO A
COMPRENDERE IL MINTERMINE 5 E C L'UNICO A COMPRENDERE IL MINTERMINE 8 E
12 CONCLUDIAMO CHE SONO TUTTI ESSENZIALI E PERTANTO:

$$F(x_0, x_1, x_2, x_3) = A + B + C$$

$$F(x_0, x_1, x_2, x_3) = \overline{x_0} \overline{x_1} x_3 + \overline{x_2} x_3 + x_0 \overline{x_2}$$

SI OSSERVA CHE IN CASO DI DC-SET SI ESCLUDONO SOLO DALLA TABELLA FINALE
(IN QUESTA PAGINA) I MINTERMINI DEL DC-SET.

METODO DI PETRIK

IL METODO DI PETRIK SI APPLICA ALLA TABELLA FINALE; RIPRENDIAMO QUINDI L'ESEMPIO FATTO
SOPRA, PER RICAVARE LA FORMULA FINALE AVREMO POTUTO SCEGLIERE UN IMPLICANTE ALLA
VOLTA PER OGNI MINTERMINE, CIOÈ:

$$\underbrace{(A + B)}_{m_1} \cdot \underbrace{A}_{m_3} \cdot \underbrace{B}_{m_5} \cdot \underbrace{C}_{m_8} \cdot \underbrace{(B + C)}_{m_9} \cdot \underbrace{C}_{m_{12}} \cdot \underbrace{(B + C)}_{m_{13}} =$$

SCELTA DI:

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

$$= ABC$$

IN CONCLUSIONE SAREMO OBBLIGATI A PRENDERE TUTTI GLI IMPLICANTI PER COPRIRE LA FUNZIONE.

SE, AL CONTRARIO, CI POSSIAMO RITROVARE IN UNA SITUAZIONE DEL GENERE:

$$AB + CEF + DGH$$

ALLORA AVREMO POTUTO SCEGLIERE UNA DELLE TRE COMBINAZIONI DI IMPLICANTI (QUELLA CON MENO IMPLICANTI) TALI CHE SIANO IL PIÙ GRANDI POSSIBILE).

RAGRUPPARE GLI ZERI

IN ALCUNI CASI PUÒ ESSERE PIÙ FACILE RAGRUPPARE GLI ZERI PIUTTOSTO CHE GLI UNO, IN TAL CASO AL POSTO DI "AND" METTEREMO "OR" E VICEVERSA (QUANDO SI TRATTA DI SCRIVERE L'ESPRESSIONE DELLA FUNZIONE). LE FORME DIRETTE DIVENTANO INVERSE E VICEVERSA.

RICONOSCERE UN OPERATORE LOGICO CON LA K-MAP

PROVIAMO AD IMMAGINARCI LA PEGGIOR MAPPA DI Karnaugh, CIÒ QUELLA CHE RICHIEDE LA PRESENZA, NELL'ESPRESSIONE DI f , DEL MAGGIOR NUMERO DI IMPLICANTI POSSIBILE. QUESTA MAPPA È LA SEGUENTE:

	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

ORA BISOGNA LA K-MAP CORRISPONDENTE ALLO XOR SU 4 VARIABILI ($P(x_0, x_1, x_2, x_3) = x_0 \oplus x_1 \oplus x_2 \oplus x_3$); RICORDANDO CHE È VERO QUANDO IL NUMERO DI VARIABILI VERE È DISPARI:

	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

NOTIAMO IMMEDIATAMENTE CHE LE DUE MAPPE SONO IDENTICHE, E PERTANTO LA PRIMA SI RIVELA ESSERE UNA BUONA MAPPA, IMPLEMENTABILE TRATTATA MA UN SOLO OPERATORE LOGICO, LO XOR.

IMMAGINIAMO ORA DI DOVER IMPLEMENTARE LA SEGUENTE FUNZIONE:

	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1			

$$f(x_0, x_1, x_2, x_3)$$

POTREMMO PENSARE DI SFRUTTARE LO XOR VISTA LA SOMILIANZA, MA NEPREMMO ANCHE LA CELLA 1110 CON UN 1.

SI POTREBBERO ALLORA UNIRE IN UNA 2 FUNZIONI, LA PRIMA CORRISPONDENTE ALLO XOR MENTRE LA SECONDA TALE PER CUI LA CELLA IN 1110 VALGA 0 (UNITO IN UNA AND CON LA XOR DAREBBE 0 COME RISULTATO).

EFFATTAMENTE VORREMO UNA COSA DEL GENERE:

	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

•

	00	01	11	10
00	-	1	-	1
01	1	-	1	-
11	-	1	-	1
10	1	-	0	-

$$g(x_0, x_1, x_2, x_3)$$

LA SECONDA FUNZIONE SI RICAVA SUBITO SFRUTTANDO IL DON'T CARE ALLA SINISTRA DELLO 0 E RAPPRESENTANDO QUEST'ULTIMO ANCHIE LOI 1.

$$g(x_0, x_1, x_2, x_3) = x_1, x_2, \bar{x}_3 \quad (\text{SE FOSSEMO 1 QUELLI RAPPRESENTATI})$$

↓

↓

$$g(x_0, x_1, x_2, x_3) = \bar{x}_1 + \bar{x}_2 + x_3$$

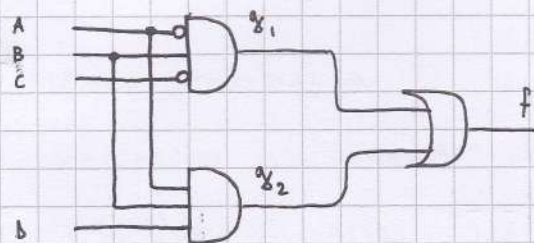
OVVERO IN CONCLUSIONE AVREMO:

$$\begin{aligned} f(x_0, x_1, x_2, x_3) &= (x_0 \oplus x_1 \oplus x_2 \oplus x_3) \cdot g(x_0, x_1, x_2, x_3) \\ &= (x_0 \oplus x_1 \oplus x_2 \oplus x_3) \cdot (\bar{x}_1 + \bar{x}_2 + x_3) \end{aligned}$$

IL GLITCH

CONSIDERIAMO LA SEGUENTE FUNZIONE E DISEGNIAMONE IL RISPETTIVO CIRCUITO:

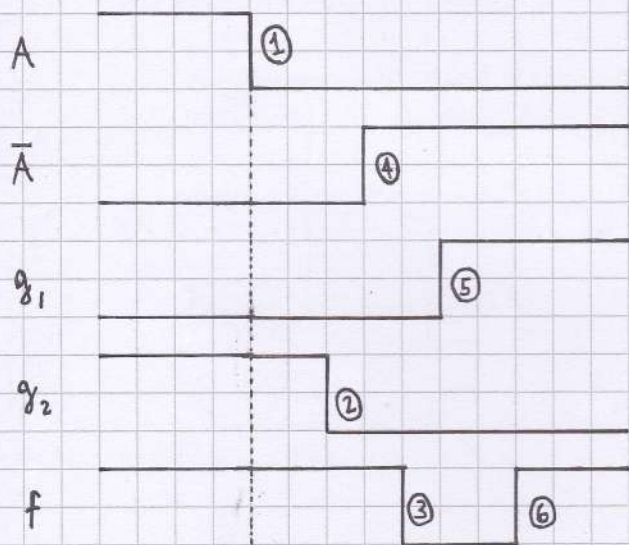
$$f(A, B, C, D) = \bar{A} B \bar{C} + A B D$$



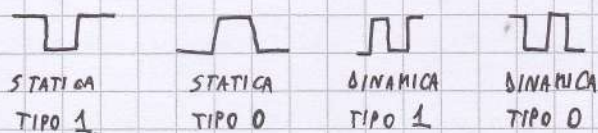
ORA PROVIAMO A VEDERE COSA SUCCEDERÀ SE SI PASSA DALLA COMBINAZIONE $(1, 1, 0, 1)$ A QUELLA $(0, 1, 0, 1)$ DOVE CAMBIA LA VARIABILE A.

NEL FARLO TERREMO CONTO DEI RITARDI DOVUTI AL PASSAGGIO DEL SEGNALE ATTRAVERSO UN OGGETTO FISICO.

PRIMA DI INIZIARE POSSIAMO CALCOLARE IL VALORE DELLA FUNZIONE f NELLE 2 SITUAZIONI E POICHÉ $f(0, 1, 0, 1) = f(1, 1, 0, 1) = 1$ CI ASPETTIAMO CHE f RIMANGA COSTANTE A 1.



QUESTA È CHIAMATA **AREA GLITCH** E NE ESISTONO DIVERSE:



1 - INIZIALMENTE A CAMBIA STATO PASSANDO DA 1 A 0.

2 - DOPO UN BREVE DELAY $g_2(0, 1)$ DIVENTA PARI A 0

3 - POICHÉ $g_1 = g_2 = 0$ f CAMBIA STATO PASSANDO A 0

4 - DOPO IL CAMBIAMENTO DI g_2 AVVIENE LA REINIZIAZIONE DI A

5 - $g_1(0, 1, 0) = 1$ (CAMBIA STATO)

6 - INFINE, CON UN PICCOLO RITARDO $f = 1$

PER EVITARE IL GLITCH, SE NECESSARIO, POSSIAMO AIUTARCI CON LA MAPPA

DI KARNAUGH:

	00	01	11	10
00		1		
01		1	1	
11			1	
10				

DOVREMO CERCARE UNA COPPIA DI 1 ADIACENTI COPERTA DA IMPLICANTI DIVERSI, QUINDI ALLUNGIAMO UN IMPLICANTE PER COPRIRE TALE COPPIA.

NEL NOSTRO CASO L'IMPLICANTE $\bar{A}\bar{B}C$ COPRE GLI UNO IN POSIZIONE (0100) E (0101) MENTRE L'IMPLICANTE ABD COPRE QUELLI IN (1101) E (1111).

ALLUNGIAMO L'IMPLICANTE ~~SE~~ CHE RAGGRUPPA GLI UNO SEGNATI IN ROSSO, COSÌ FACENDO AVREMO:

$$F(A, B, C, D) = \bar{A}\bar{B}C + ABD + B\bar{C}D$$

DOVE SI OSSERVA CHE L'ULTIMO IMPLICANTE NON DIPENDE DA A E PERTANTO RIMANE COSTANTE AL SUO VALORE ($= 1$) IMPEDENDO IL VERIFICARSI DI GLITCH (LA FUNZIONE UNISCE IN OR 1 3 IMPLICANTI, E ~~PER~~ L'ULTIMO È SEMPRE VERO).

FUNZIONI A PIÙ USCITE

UNA FUNZIONE A PIÙ USCITE PUÒ ESSERE DEFINITA COME SEGRE:

$$f(x_0, x_1, \dots) = \begin{cases} y_1(x_0, x_1, \dots) \\ \vdots \\ y_n(x_0, x_1, \dots) \end{cases}$$

NE FARÒ ORA UN ESEMPIO CONCRETO IN MODO DA FISSARE MEGLIO LE IDEE:

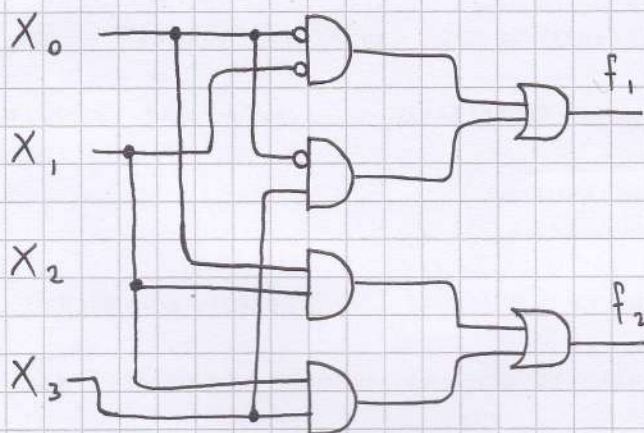
	00	01	11	10
00	1			
01	1	1		
11	1	1		
10	1			

	00	01	11	10
00			1	
01		1	1	
11		1	1	
10			1	

$$f_1 = \bar{X}_0 \bar{X}_1 + \bar{X}_0 X_3$$

$$f_2 = X_0 X_1 + X_1 X_3$$

HO RAPPRESENTATO LE DUE FUNZIONI CON LA K-MAP, E ORA NE RISVELNERÒ IL CIRCUITO:



MINIMIZZAZIONE COSTO

SI PUÒ OSSERVARE CHE IL COSTO DI UN CIRCUITO È PROPORZIONALE ALLA SUA AREA, ESSA, D'ALTRO CANTO, PUÒ ESSERE APPROSSIMATA SOMMANDO TRA LORO IL NUMERO DI INGRESSI ALLE SINGOLE PORTE LOGICHE.

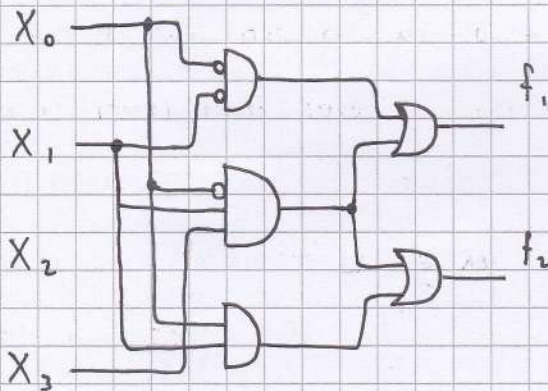
QUESTO VORREBBE DIRE CHE IL NOSTRO CIRCUITO HA AREA PROPORZIONALE A 12.

È POSSIBILE RIDURRE QUESTO VALORE OSSERVANDO CHE L'IMPLICANTE CHE COPRE I MINITERMINI (0101) E (0111) È COMUNE AD ENTRAMBE LE FUNZIONI, PERTANTO POSSONO ESSERE RISCritte COME:

$$f_1 = \overline{X_0} \overline{X_1} + \overline{X_0} X_1 X_3$$

$$f_2 = X_0 X_1 + \overline{X_0} X_1 X_3$$

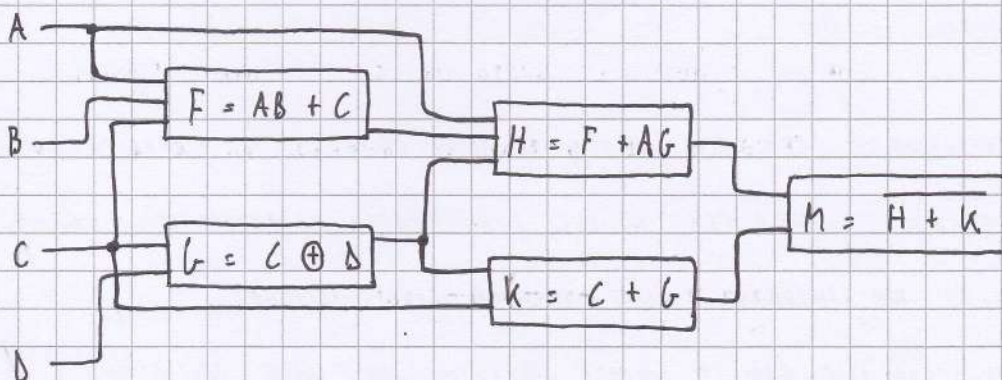
IL FATTO CHE CI SIA UN IMPLICANTE COMUNE CI PERMETTERÀ DI CREARE UN CIRCUITO CON AREA MINORE, INFATTI:



IN QUESTO CASO L'AREA SARÀ PROPORZIONALE A $2 + 3 + 2 + 2 + 2 = 11$, UN RISULTATO MIGLIORE RISPETTO A PRIMA.

È POSSIBILE MODIFICARE L'ALGORITMO DI QUINE-MCCLUSKEY PER RICAVARE LE ESPRESSIONI DELLE FUNZIONI CHE MINIMIZZANO IL COSTO DEL CIRCUITO, MA NON LO VEDREMO QUI.

D'ORA IN POI, DATO CHE LE FUNZIONI POTREBBERO DIVENTARE MOLTO COMPLESSE, DISGNEREMO I CIRCUITI SFRUTTANDO DEI BLOCCHI FUNZIONALI DEL TIPO:



METRICHE IMPORTANTI NELLA CREAZIONE DI UN CIRCUITO

Vittorio Alzetta

DESCRIVERO' ORA ALCUNE DELLE METRICHE DA TENERE IN CONSIDERAZIONE NELLA CREAZIONE DI UN BUON CIRCUITO:

- **AREA**: PER MINIMIZZARE L'AREA ABBIAMO CERCATO DI RIDURRE AL MINIMO IL NUMERO E LA DIMENSIONE DEGLI IMPLICANTI DI UNA FUNZIONE.

QUESTA È UNA METRICA DI TIPO **AVERAGE**, CIOÈ, UN MIGLIORAMENTO IN UNA QUALSIASI PORTA LOGICA PORTA A DEI BENEFICI IN TUTTO IL CIRCUITO.

- **RITARDO**: RAPPRESENTA IL RITARDO TRA L'INGRESSO DEGLI INPUT E L'USCITA DEGLI OUTPUT, VOGLIAMO MINIMIZZARLO. OGNI PORTA LOGICA HA IL SUO **DELAY**.

SIAMO INTERESSATI AL **PERCORSO CRITICO**, CIOÈ IL TRACCATO CHE NECESSITA DI PIÙ TEMPO PER ESSERE PERCORSO.

QUESTA È UNA METRICA DI TIPO **WORST-CASE**, CIO' SIGNIFICA CHE UN MIGLIORAMENTO AI TRACCATI NON PEGGIORI NON PORTERÀ A BENEFICI GENERALI. DOVEREMO AGIRE SUI NODI DEL PERCORSO CRITICO.

- **POTENZA**: OGNI VOLTA CHE UNA TRANSDAZIONE FISICA CAMBIA VALORE C'È UN CONSUMO DI ENERGIA (CON CONSUMO SI INTENDE IL PASSAGGIO DA ENERGIA ELETTRICA A ENERGIA TERMICA, INUTILIZZABILE AI NOSTRI FINI). QUESTO CONSUMO È DETTO **E_{dyn}** (CONSUMO DINAMICO) E DIPENDE DA:

- LA CAPACITÀ DELLE PORTE LOGICHE A VALLE (CHG VENGONO CARICATE), CIOÈ L'ENERGIA NECESSARIA A FAR RILEVARE UN CAMBIAMENTO ALLE PORTE SUCCESSIVE. È PROPORTIONALE AL **FAN OUT**
N° DI PORTE IN USCITA

- IL NUMERO DI VOLTE IN CUI CAMBIA IL PROPRIO VALORE, SI INTENDE IL NUMERO DI PASSAGGI DA 0 A 1 O VICEVERSA DEL CIRCUITO, CIO' CORRISPONDE ALLA **SWITCHING ACTIVITY** E DIPENDE DAL COMPORTAMENTO DEL CIRCUITO.

OLTRE ALL'ENERGIA DINAMICA BISOGNA CONSIDERARE CHE PER IL SOLO FATTO DI AVER ACCESSO IL CIRCUITO SI HA UNA PERDITA DI ENERGIA, CHIAMATA E_{STAT} (STATICA) O E_{LEAK} (DA LEAKAGE, PERDITA).

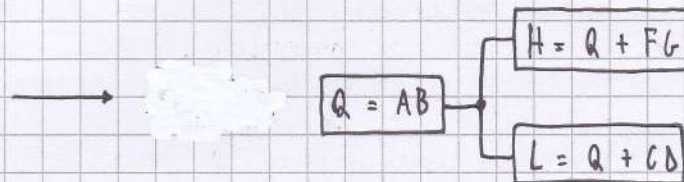
UN FATTORE CHE INFLUENZA SIA LA E_{DYN} SIA LA E_{STAT} È LA V.D.D. O TENSIONE DI ALIMENTAZIONE, CHE ALL'AUMENTARE CAUSA PERDITE MIGLIORI DI E_{STAT} E INCAIDE PIÙ E_{DYN} PER IL FUNZIONAMENTO.

MANIPOLAZIONE DEL CIRCUITO

PER MIGLIORARE UN CIRCUITO SECONDO LE METRICHE APPENA VISTE SI POSSONO OPERARE MANIPOLAZIONI DI TIPO ALGEBRAICO:

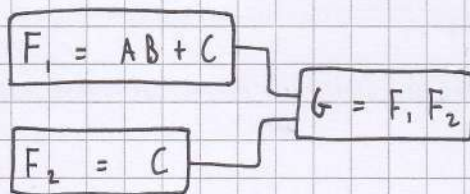
$$H = AB + FG$$

$$L = AB + CD$$



DOVE HO RIUNITO UN TERMINE CHE COMPARIVA SEPARATAMENTE SU ENTRAMBE LE FUNZIONI H E L .

UN ALTRO TIPO DI MANIPOLAZIONE PUÒ ESSERE DI TIPO BOOLEANO, CONSIDERIAMO IL SEGUENTE ESEMPIO:



	00	01	11	10
0			1	
1	1	1	1	1

F_1

	00	01	11	10
0				
1	1	1	1	1

F_2

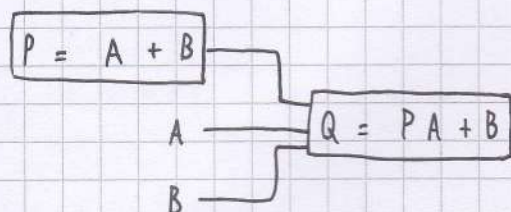
OSSERVIAMO CHE SE CAMBIASSIMO LA SECONDA RIGA DI F_1 (QUELLA CON $C=0$) CON UNA QUALSIASI COMBINAZIONE DI 0 E 1 IL RISULTATO NON CAMBIEREBBE, PERCHÉ FINIREBBERO IN ANO CON LEI 0 DELLA F_2 E DAREBBERO COME RISULTATO FINALE 0.

È PER QUESTO MOTIVO CHE POSSIAMO RISCRIVERE LA K-MAP DI F_1 COME:

	00	01	11	10
0	-	-	1	-
1	1	1	1	1

← NON CI INTERESSA
CIÒ CHE C'È QUI.

PRIMA DI INTRODURRE QUESTI DUE NUOVI CONCETTI CONSIDERIAMO IL SEGUENTE ESEMPIO, A CUI FAREMO RIFERIMENTO IN SEGUITO:



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

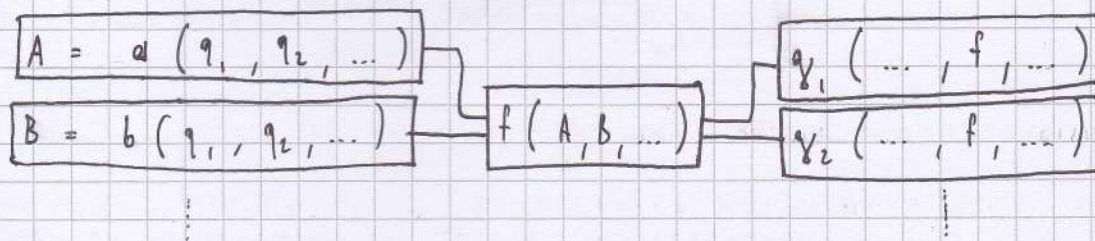
OSSERVIAMO SUBITO CHE È IMPOSSIBILE AVERE $P=0$ MENTRE $A=0$ E $B=1$, PER DEFINIZIONE $P=A+B$, QUINDI $P=0+1=1$. SIMILMENTE SE $A=1$ E $B=0$ ALLORA P NON PUÒ ESSERE 0, INFINE SE $P=1$ E $A=B=0$ SIAMO DI NUOVO DI FRONTE AD UN CASO IMPOSSIBILE.

QUESTO RAGIONAMENTO CI PORTA AD ESCLUDERE DAI NOSTRI CALCOLI ALCUNE CELLE, CHE PERÒ POTRANNO ESSERE UTILIZZATE COME DEI DON'T CARE AL FINE DI MINIMIZZARE. LA NUOVA K-MAP DI Q SARÀ QUINDI:

AB \ P	00	01	11	10
0		-	-	
1	-	1	1	1

~~~~~  $Q = P = A + B$

ORA FACCIAMO UN ESEMPIO UN PO' PIÙ ELABORATO:



A SINISTRA TUTTE LE FUNZIONI CHE GENERANO LE VARIABILI DI  $f$ , A DESTRA TUTTE LE FUNZIONI CHE A CUI  $f$  PASSA IL PROPRIO RISULTATO.



DEFINIAMO IL **CONTROLLABILITY DC-SET** (O CDC-SET) COME L'INSIEME DELLE COMBINAZIONI DI MINTERMINI CHE NON SI POSSONO VERIFICARE. RIFERENDOCI ALL'ESEMPIO DI PRIMA LA COMBINAZIONE DI ABP PARIA A (010) o (110) o {ANCORA (001) NON POTEVA VERIFICARSI.

IN FORMULE AVREMO CHE SE:

$$A \neq a(q_1, \dots)$$

CIOE A E DIVERSO DALLA FUNZIONE CHE LO GENERA (NEL CASO DI PRIMA  $1 = P \neq A+B=0+0$ )

IL CASO SARÀ IMPOSSIBILE, UNENDO TUTTI QUESTI CASI IMPOSSIBILI A OTTERREMO:

$$CDC-SET = [A \oplus a(q_1, q_2, \dots)] + [B \oplus b(q_1, q_2, \dots)] + \dots$$

SE SONO UGUALI CI DARÀ 0 E IL TERMINE SCONPARRÀ DALLA SOMMA DI OR  
LO XOR RESTITUISCE 1 SE I DUE TERMINI SONO DIVERSI

DOBBIAMO UNIRE TUTTE LE COMBINAZIONI IMPOSSIBILI

DEFINIAMO INVECE **OSSERVABILITY DC-SET** (O ODC-SET) L'INSIEME DEI PUNTI

PER CUI IL RISULTATO DI f NON HA IMPORTANZA E QUINDI NON INFLUENZA IL RISULTATO DELLE FUNZIONI CHE HANNO TRA LE VARIABILI IL RISULTATO DI f.

IN FORMULE SI AVrà:

$$ODC-SET = [\overline{y_1(f=0)} \oplus \overline{y_1(f=1)}] \cdot [\overline{y_2(f=0)} \oplus \overline{y_2(f=1)}] \cdot \dots$$

RESTITUISCE 0 SE SONO UGUALI, POI VIENE NEGATO E DÀ 1, SCONPARENDO

DEVE VALERE PER TUTTE LE FUNZIONI

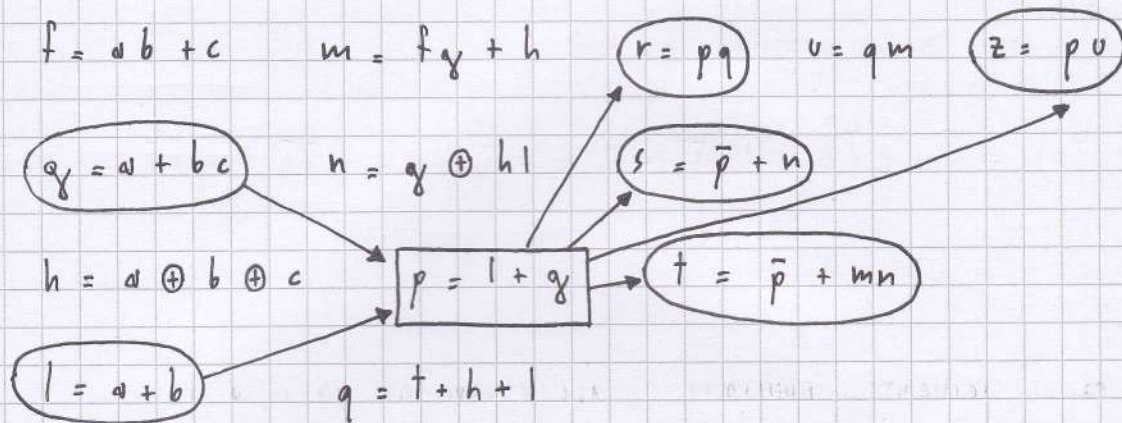
IN SOSTANZA DEVE ESSERE INIRRUENTE PER TUTTE LE FUNZIONI IN CUI COMPARE IL RIS.

IN CONCLUSIONE FACCIO NOTARE CHE:

$$DC-SET = OBSERVABILITY DC-SET + CONTROLLABILITY DC-SET$$



CONSIDERIAMO IL SEGUENTE CIRCUITO, DEFINITO DALLE SEGUENTI FUNZIONI:



HO INDICATO LA FUNZIONE DI CUI VOGLIAMO CALCOLARE IL DC-SET MENTRE HO CERCATO LE FUNZIONI CHE DEFINISCONO LE VARIABILI DI  $p$  E QUELLE A CUI  $p$  FORNISCE IL PRIMO RISULTATO.

PROCEDIAMO CON IL DETERMINARE IL DC-SET:

$$\begin{aligned}
 DC(p) &= [l \oplus (a+b)] + [y \oplus (a+bc)] \\
 &= \bar{l}(a+b) + l(\overline{a+b}) + \bar{y}(a+bc) + y(\overline{a+bc}) \\
 &= \bar{l}a + \bar{l}b + l\bar{a}\bar{b} + \bar{y}a + \bar{y}bc + y\bar{a}\bar{b} \\
 &= \bar{l}a + \bar{l}b + l\bar{a}\bar{b} + \bar{y}a + \bar{y}bc + y\bar{a}\bar{b} + y\bar{a}\bar{c}
 \end{aligned}$$

SIAMO ARRIVATI AD ESPRIMERE IL DC-SET CON OPERATORI DI BASE, ORA DETERMINIAMO IL DC-SET:

$$DC(p) = DC(r/p) \cdot DC(s/p) \cdot DC(z/p) \cdot DC(t/p)$$

$$DC(r/p) = \overline{r(p=0) \oplus r(p=1)} = \overline{0 \oplus q} = \overline{1 \cdot q + 0 \cdot \bar{q}} = \bar{q}$$

$$DC(s/p) = \overline{s(p=0) \oplus s(p=1)} = \overline{1 \oplus n} = \overline{0 \cdot n + 1 \cdot \bar{n}} = n$$

$$DC(z/p) = \overline{z(p=0) \oplus z(p=1)} = \overline{0 \oplus v} = \overline{1 \cdot v + 0 \cdot \bar{v}} = \bar{v}$$



$$\text{CDC } \dagger / p = \overline{\dagger(p=0)} \oplus \dagger(p=1) = mn \oplus 1 = 0 \cdot mn + 1 \cdot \overline{mn} = mn$$

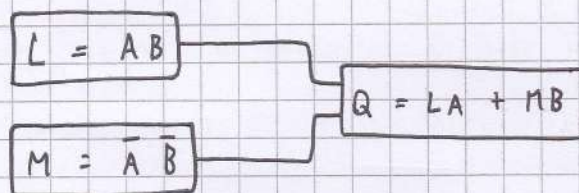
$$\text{CDC}(p) = \bar{q} \cdot n \cdot \bar{u} \cdot mn = \bar{q} n \bar{u} m$$

DUNQUE IL DC-SET DELLA FUNZIONE  $p$  SARÁ:

$$\text{DC-SET}(p) = \bar{1}a + \bar{1}b + 1\bar{a}\bar{b} + \bar{q}a + \bar{q}bc + q\bar{a}\bar{b} + q\bar{a}\bar{c} + \bar{q}n\bar{u}mn$$

### ESEMPIO CDC-SET

CONSIDERAMO LA SEGUENTE FUNZIONE, ALL'INTERNO DI UN CIRCUITO:



ORA MOSTRIAMO LA K-MAP DI  $Q$ :

| AB<br>LM | AB |    |    |    |
|----------|----|----|----|----|
|          | 00 | 01 | 11 | 10 |
| 00       |    |    |    |    |
| 01       |    | 1  | 1  |    |
| 11       |    | 1  | 1  | 1  |
| 10       |    |    | 1  | 1  |

E ORA DETERMINIAMO IL CONTROLLABILITY DC-SET:

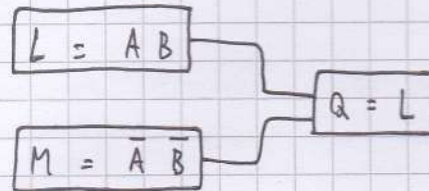
$$\begin{aligned} \text{CDC}(Q) &= (L \oplus AB) + (M \oplus \bar{A}\bar{B}) = \bar{L}AB + L\bar{A}\bar{B} + \bar{M}\bar{A}\bar{B} + M\bar{A}\bar{B} \\ &= \bar{L}AB + L\bar{A}\bar{B} + L\bar{B} + \bar{M}\bar{A}\bar{B} + \bar{M}\bar{A} + \bar{M}\bar{B} = \bar{L}AB + \bar{L}\bar{A} + \bar{L}\bar{B} + \bar{M}\bar{A}\bar{B} + \bar{M}\bar{A} + \bar{M}\bar{B} \end{aligned}$$

SAPPIAMO CHE QUANDO IL CDC( $Q$ ) VALE 1 C'È UNA VARIABILE CHE È IN UNO STATO NON GENERATO DALLA FUNZIONE CHE DEFINISCE LA VARIABILE STESSA, PERTANTO POSSIAMO ALLUNGERE ALLA K-MAP I DON'T CARE IN CORRISPONDENZA DEI PUNTI IN CUI IL CDC( $Q$ ) VALE 1:



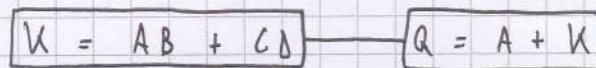
| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00      | -  | -  | -  | -  |
| 01      | -  | -  | -  | -  |
| 11      | -  | -  | -  | -  |
| 10      | -  | -  | 1  | -  |

E ANCHE LA VERSIONE SEMPLIFICATA DI  $Q$  SARÀ:



### ESEMPIO OBC-SET

CONSIDERIAMO UN ALTRO CIRCUITO:



È MOLTO SEMPLICE MA CI PERMETTE DI OSSERVARE LE CONSEGUENZE CHE L'OBC-SET HA SULLA K-MAP DELLA FUNZIONE  $K$ :

| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00      |    |    | 1  |    |
| 01      |    |    | 1  |    |
| 11      | 1  | 1  | 1  | 1  |
| 10      |    |    | 1  |    |

ORA CALCOLIAMO L'OBSERVABILITY OBC-SET:

$$\begin{aligned}
 OBC(K) &= OBC\ Q/K = \overline{Q(K=0)} \oplus \overline{Q(K=1)} = \overline{A} \oplus 1 = \overline{\overline{A} \cdot 1 + A \cdot 0} \\
 &= \overline{\overline{A}} = A
 \end{aligned}$$

ANCHE LA NUOVA K-MAP SARÀ:

| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00      |    |    | -  | -  |
| 01      |    |    | -  | -  |
| 11      | 1  | 1  | -  | -  |
| 10      |    |    | -  | -  |

$$K = CD$$



# IL TEMPO DI PROPAGAZIONE

FINO AD ORA ABBIAMO ~~DEMO~~ CERCATO DI RIDURRE L'AREA DEL NOSTRO CIRCUITO, ORA INVECE STUDIEREMO UN ALGORITMO CHE CI PERMETTERÀ DI CALCOLARE IL TEMPO DI PROPAGAZIONE DEL SEGNALE ASSOCIATO AL PERCORSO CRITICO.

QUESTO ALGORITMO È ALLA BASE DELL' **ANALISI STATICA** DEL TEMPO DI PROPAGAZIONE.

IN SOSTANZA CI STIAMO CHIEDENDO QUANTO TEMPO È NECESSARIO PER ESSERE SICURI DI AVERE UN RISULTATO.

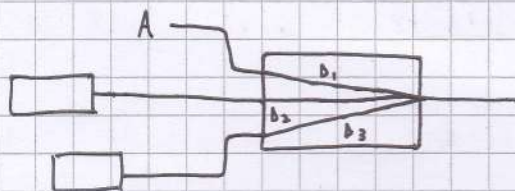
LA RISPOSTA SI AVRÀ CALCOLANDO IL SEGUENTE VALORE:

$$T_d = D + \alpha FI + \beta FO$$

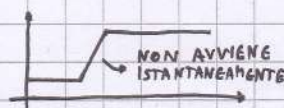
DOVE  $D$  È IL RITARDO INTINSECO DELLA ~~CHIAMATA~~ PORTA LOGICA,  $\alpha FI$  È UN FATTORE PROPORZIONALE AL NUMERO DI INGRESSI DELLA PORTA E  $\beta FO$  È UN FATTORE PROPORZIONALE ~~AL NUMERO~~ ALLA MANOBBRA DELLE PORTE DA CANGIARE A VALLE (TANTA PIÙ CANGI RICHIEDONO TANTO PIÙ TEMPO RICHIEDERÀ CANGIARE), PER SEMPLICITÀ IN QUEST'ULTIMO CASO SI CONSIDERA SEMPLICEMENTE IL NUMERO DI PORTE ALL'USCITA PER UN FATTORE  $\beta$ .

UN MODELLO PIÙ PRECISO E COMPLESSO CONSIDERA ULTERIORI FATTORI, QUALI:

- DELAY SPECIFICI PER OGNI INGRESSO DELLA PORTA LOGICA



- PENDENZA DEL SEGNALE NEL CAMBIAMENTO DA 0 A 1 O VICEVERSA:



- ALTEZZA DELLA VARIAZIONE DEL SEGNALE, SE SI PASSA DA 0 A 1 È DIVERSO RISPETTO AL CAMBIAMENTO DA 1 A 0.



# ANALISI STATICA TEMPO DI PROPAGAZIONE

IN QUESTO CAPITOLO VEDREMO I PASSI NECESSARI ALL'INDIVIDUAZIONE DEL PERCORSO CRITICO (IL PIÙ LUNGO DA PERCORRERE PER IL SEGNALE).

PER PRIMA COSA SARÀ NECESSARIO ORDINARE TOPOLOGICAMENTE IL NOSTRO CIRCUITO, IN SOSTANZA DOVREMO RAPPRESENTARE GRAFICAMENTE UN SISTEMA DI FUNZIONI PER POI ELABORARE OGNI NODO DEL CIRCUITO.

CONSIDERIAMO LE SEGUENTI FUNZIONI:

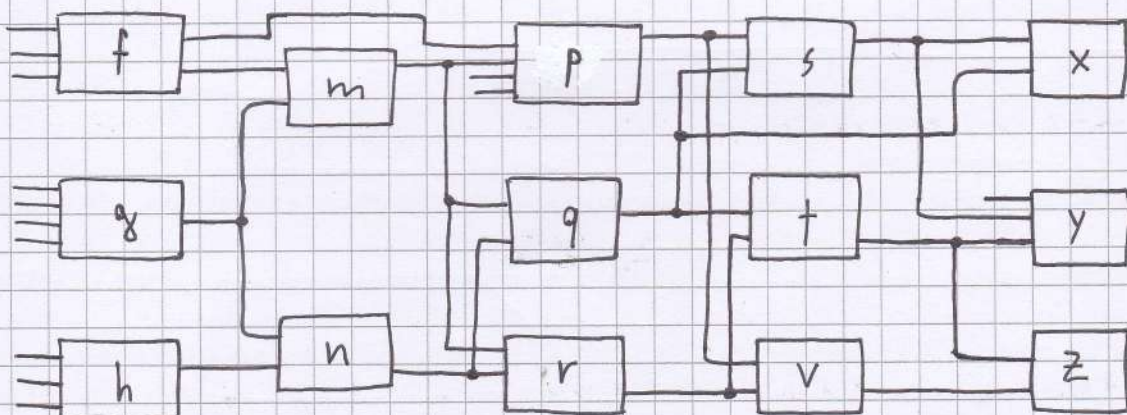
$$\begin{aligned} f &= ab + c & m &= \bar{f}q & q &= \bar{n} + m & t &= q + r & x &= q \oplus \bar{s} \\ g &= ab + cd & n &= q \oplus h & r &= \overline{n + m} & v &= pr & y &= c \oplus \bar{s}t \\ h &= \bar{b}c + d & p &= d \oplus (bf + m) & s &= pq & z &= t \oplus v \end{aligned}$$

DOVE  $a, b, c, d$  SONO LE VARIABILI DI INGRESSO E  $x, y, z$  SONO LE FUNZIONI DI USCITA.

IL TEMPO CHE IL SEGNALE IMPIETERÀ PER ATTRAVERSARE UN LATE SARÀ:

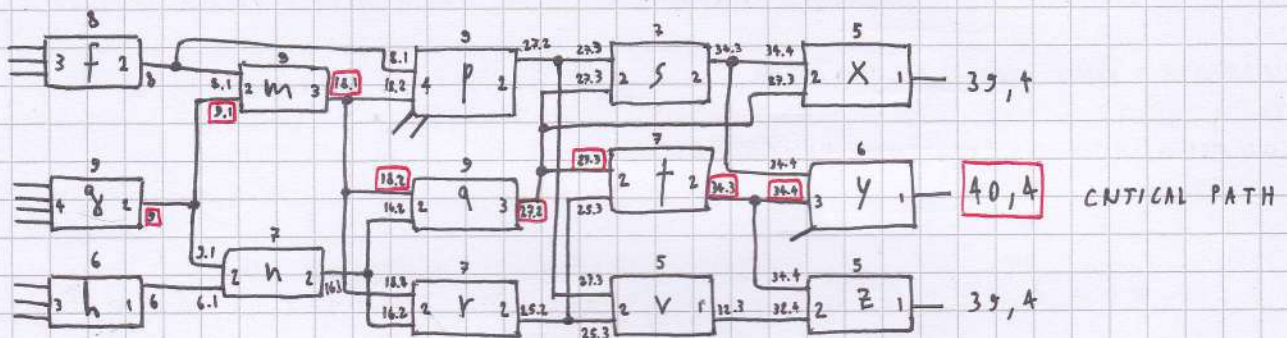
$$T_L = 1 + 1 \cdot F_I + 2 \cdot F_O$$

COME DETTO PRECEDENTEMENTE, AGGIUNGIAMO ANCHE UN TEMPO DI ATTRAVERSAMENTO DEI FILI ELETTRICI  $T_w = 0,1$  COSTANTE. PROCEDIAMO CON LA RAPPRESENTAZIONE DEL CIRCUITO:





CIÒ CHE FAREMO ORA SARÀ SEGNARE A SINISTRA E DESTRA DEL NOME DELLA FUNZIONE  
RISPETTIVAMENTE IL NUMERO DI INPUT E DI OUTPUT, SOPRA SCRIVEREMO IL TEMPO NECESSARIO  
ALL'ATTRAVERSAIMENTO DI QUEL GATE SPECIFICO E LATERALMENTE, SUI FILI, IL TEMPO CHE  
IL SEGNALE HA IMPIEGATO PER ARRIVARE FINO A LÌ:



NEL SEGNARE I TEMPI ALL'USCITA DEI GATE HO SEMPRE PRESO IN CONSIDERAZIONE  
IL VALORE PIÙ ALTO TRA QUELLI IN INGRESSO, SOMMANDOCI POI IL  $T_G$ .

L'ESEMPIO NON È DEI MIGLIORI IN QUANTO PERCORRENDO AL CONTRARIO IL ~~CIRCUITO~~  
CIRCUITO PER TROVARE IL CRITICAL PATH CI TROVIAMO DI FRONTE A DEI BIVI, CIÒ  
NON TOGLIE CHE IL PERCORSO EVIDENZIATO SIA COMUNQUE UN PERCORSO CRITICO E  
IL SUO TEMPO DI PERCORRENZA SIA PARI A 40,4 UNITÀ DI TEMPO.

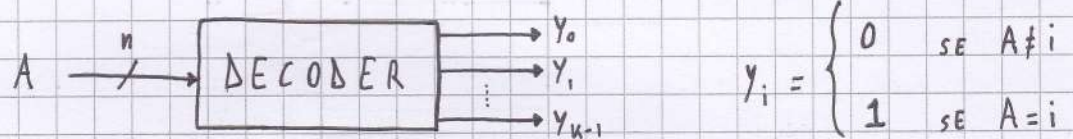


## IL DECODER

INTRODUCIAMO ORA ALCUNI CONCETTI CHE SARANNO UTILISSIMI QUANDO VOIAMO FARE DELL'ALGEBRA CON I CIRCUITI.

IL PRIMO STRUMENTO CHE VERAMENTE È IL **DECODER**, CIOÈ UN OGGETTO (UN CIRCUITO) CHE RICEVE IN INGRESSO UNA COMBINAZIONE DI BIT E FORNISCE IN USCITA SVANATI BIT, PONENDO A 1 SOLO QUELLO CORRISPONDENTE AL VALORE NUMERICO RAPPRESENTATO DALL'INPUT.

AVREMO UN CIRCUITO DEL TIPO:



IL SIMBOLO CHE HO USATO NELLA FRECCIA DI INGRESSO VUOLE DIRE CHE CI SONO  $n$  INPUT,  $K$  E  $n$  SONO LEGATI DALLA SEGUENTE RELAZIONE:

$$K = 2^n$$

PROVIAMO A SCRIVERE LA TABELLA DI VERITÀ DI UN DECODER A 2 BIT (DI INGRESSO):

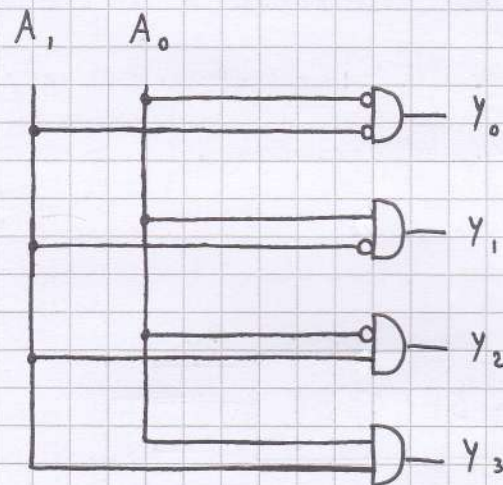
| $A_1$ | $A_0$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 1     |
| 0     | 1     | 0     | 0     | 1     | 0     |
| 1     | 0     | 0     | 1     | 0     | 0     |
| 1     | 1     | 1     | 0     | 0     | 0     |

NOTIAMO CHE QUANDO LA COMBINAZIONE IN INGRESSO VALE 11 ALLORA SI ACCENDE IL BIT IN USCITA PARTE A 3, INFATTI  $11_2 = 3_{10}$ .

ORA DOBBIAMO PENSARE A COME IMPLEMENTARE UNA FUNZIONE DI QUESTO TIPO IN UN CIRCUITO.



È ABBASTANZA IMMEDIATO PENSARE DI DOVER COLLEGARE GLI INPUT  $A$  A DELLE PORTE  
AND, E PER EVITARE CHE SI ATTIVINO TUTTE CONTEMPORANEAMENTE BASTERÀ NEGLARE IN  
ENTRATA OPPORTUNAMENTE ALCUNI DEI SEGNALI.  
QUINDI IL CIRCUITO SARÀ:

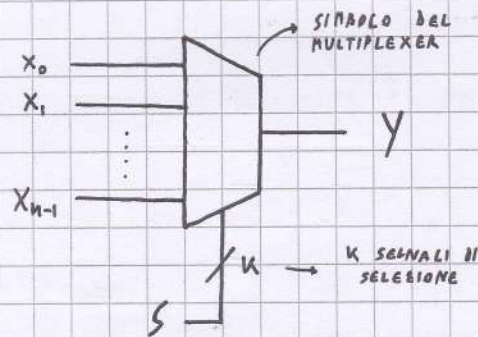




# IL MULTIPLEXER

IL CIRCUITO CHE CREEREMO ADESSO PRENDE IL NOME DI **MULTIPLEXER** (O BREVEMENTE MUX). ESSO CI PERMETTERÀ DI FARE DELLE SCELTE E IN BASE A QUESTE OTTENERE RISULTATI DIVERSI.

IN PARTICOLARE IL MULTIPLEXER SI COMPORRÀ DI  $K$  IN SEGNALI DI INGRESSO DETTI DI **SELEZIONE** E ALTRI  $n$  DETTI INVECE INGRESSI **DATI**, IN USCITA CI RESTITUIRÀ L' $i$ -ESIMO INGRESSO DATO, DOVE  $i$  È IL VALORE NUMERICO RAPPRESENTATO DALL'INGRESSO DI SELEZIONE.



$$n = 2^k$$

PROVIAMO A DISEGNARE LA TABELLA DI VENTÀ DI UN MULTIPLEXER CON 2 BIT DI SELEZIONE:

| $S_1$ | $S_0$ | $Y$   |
|-------|-------|-------|
| 0     | 0     | $X_0$ |
| 0     | 1     | $X_1$ |
| 1     | 0     | $X_2$ |
| 1     | 1     | $X_3$ |

TENENDO BEN CHIARO QUELLO A CUI DOBBIAMO ARRIVARE, CIOÈ LA TABELLA SOPRA, NOTIAMO CHE IL MEDESIMO RISULTATO PUÒ ESSERE OTTENUTO SCOMPONENDO LA FUNZIONE  $Y$  IN ALTRE PIÙ SEMPLICI:



| $s_1$ | $s_0$ | $f_3$ | $f_2$ | $f_1$ | $f_0$ |
|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | $x_0$ |
| 0     | 1     | 0     | 0     | $x_1$ | 0     |
| 1     | 0     | 0     | $x_2$ | 0     | 0     |
| 1     | 1     | $x_3$ | 0     | 0     | 0     |

ORA NON POSSIAMO CHE NOTARE 2 COSE, ANZITUTTO IL FATTO CHE LA FUNZIONE  $y$  PUÓ ESSERE OTTENUTA COME COMBINAZIONE IN OR DI  $f_0, f_1, f_2, f_3$ ; POI É CHIARA LA SOMIGLIANZA DI QUESTA TABELLA CON QUELLA DEL DECODER A 2 BIT.

SUPPONIAMO PER UN ATTIMO CHE ANCHE  $x_0, x_1, x_2, x_3$  CI SIANO SOLO 1, IN TAL CASO PER DEFINIRE LE FUNZIONI  $f$  BASTEREBBE:

$$f_0 = \overline{s_0} \cdot \overline{s_1}$$

$$f_1 = \overline{s_0} \cdot s_1$$

$$f_2 = s_0 \cdot \overline{s_1}$$

$$f_3 = s_0 \cdot s_1$$

NOI, ANCHE VOLERE SOLO 1, VOLLAMO LE VARIABILI  $x_0, x_1, x_2, x_3$ , PERTANTO CI BASTERÁ EFFETTUARE LA SEGUENTE PICCOLA MODIFICA:

$$f_0 = \overline{s_0} \cdot \overline{s_1} \cdot x_0$$

$$f_1 = \overline{s_0} \cdot s_1 \cdot x_1$$

$$f_2 = s_0 \cdot \overline{s_1} \cdot x_2$$

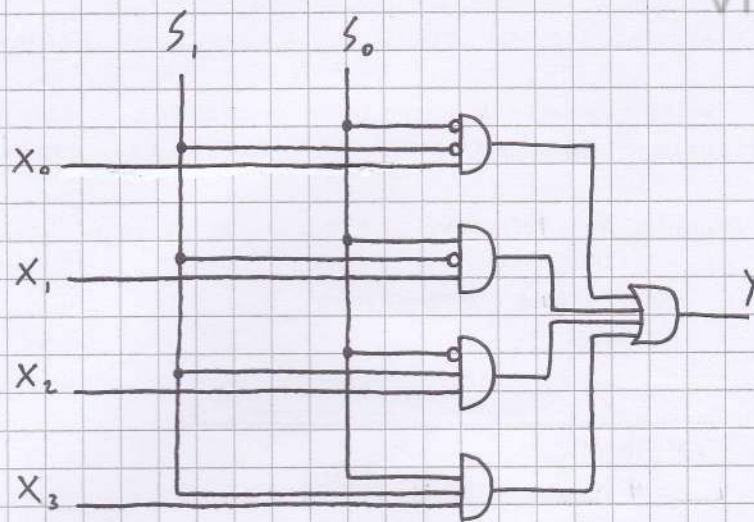
$$f_3 = s_0 \cdot s_1 \cdot x_3$$

DA CUI SI PUÓ FACILMENTE OTTENERE  $y$  COMBINANDO IN OR LE  $f$ :

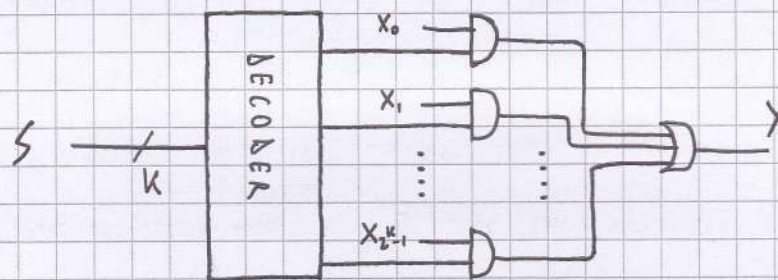
$$y = f_0 + f_1 + f_2 + f_3$$

DISSEGNAHMO QUINDI IL CIRCUITO, TENENDO CONTO DEL LAVORO NÁ FATTO ORA PER IL DECODER:





NOTIAMO CHE LO STESSO CIRCUITO PUÒ ESSERE SCRITTO ~~ANALOGAMENTE~~, PER  $K$  SEGNALI DI SELEZIONE E  $2^K$  SEGNALI DATI, NEL SEGUENTE MODO:

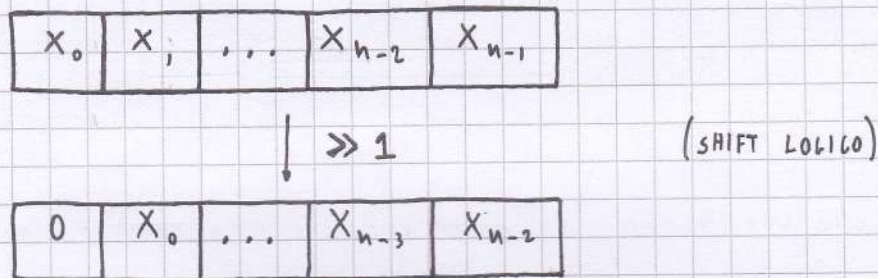




## LO SHIFTER

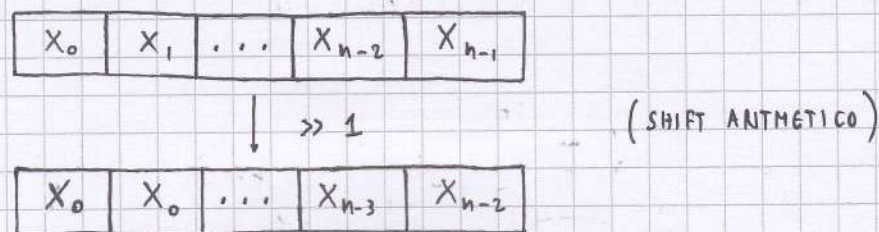
VOLLIAMO ORA COSTRUIRE UN CIRCUITO IN GRADO DI SPOSTARE UNA SEQUENZA DI BIT DI UN NUMERO DI POSIZIONI DATO.

PER OPERAZIONE DI TRASLAZIONE INTENDIAMO:



QUELLO APPENA FATTO PRENDE IL NOME DI **SHIFT RIGHT**, ANALOGAMENTE SI HA LO **SHIFT LEFT**.

È INOLTRE POSSIBILE DISTINGUERE TRA SHIFT DI TIPO **LOGICO**, DOVE LA NUOVA POSIZIONE LIBERA VIENE OCCUPATA DA UNO 0, E **ARITH SHIFT** DI TIPO **ARITMETICO**, DOVE IL BIT PIÙ SIGNIFICATIVO RIENTRA AL POSTO DELLO 0, CIOÈ:



LA PRINCIPALE DIFFERENZA TRA I 2 TIPI DI SHIFT (LOGICO E ARITMETICO) CONSISTE NELLA PERMANENZA DEL SEGNO, INFATTI, CONSIDERANDO I NUMERI <sup>COME</sup> SCRITTI IN COMPLEMENTO A 2, POICHÉ IN UN CASO (SHIFT ARITMETICO) SI CONSERVA IL BIT PIÙ SIGNIFICATIVO SI HA CHE IL SEGNO RIMANE COSTANTE. NELLO SHIFT LOGICO IL SEGNO DIVENTA SEMPRE POSITIVO.



L'OPERAZIONE CHE EFFETTIAMO SU UNA SEQUENZA DI BIT QUANDO APPLICHIAMO UNO SHIFT DESTRA, SINISTRA, LOGICO O ARITMETICO È SCHEMATIZZATA QUI SOTTO:

SLL - K <sup>NUMERO DI SHIFT</sup>

SLLA - K

$$y = x \cdot 2^k$$

SRL - K

SRA - K

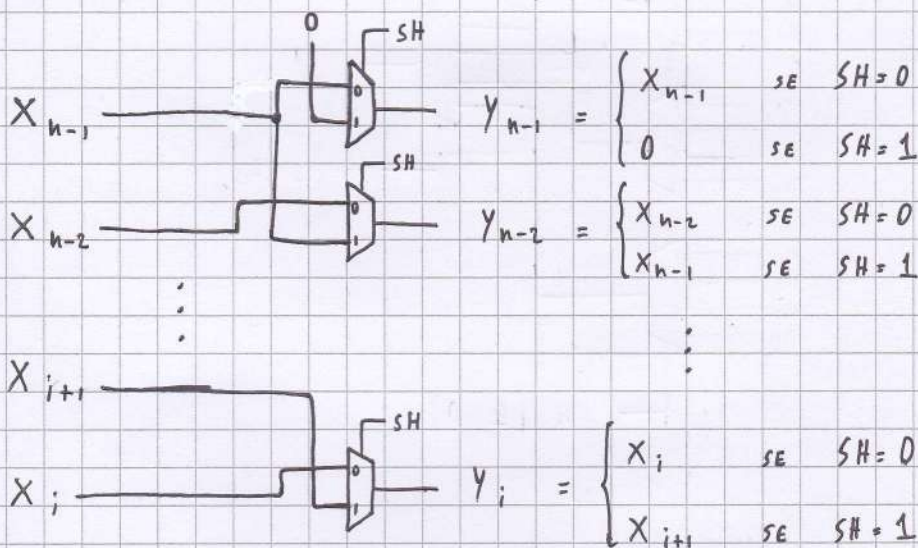
$$y = x \cdot 2^{-k} \begin{cases} \text{NON MANTIENE IL SEGNO} \\ \text{MANTIENE IL SEGNO} \end{cases}$$

OVVIAMENTE IL TUTTO A MENO DI OVERFLOW!

ORA SUPPONIAMO DI VOLER IMPLEMENTARE UN CIRCUITO CHE, IN BASE AD UN SEGNALE DI INGRESSO, DECIDA SE APPLICARE UNO SHIFT LOGICO DI UNA POSIZIONE O NON FARE NULLA:

$$SH = \begin{cases} 0 & \text{NO SHIFT} \\ 1 & \text{SRL} - 1 \end{cases}$$

PERTANTO, VISTO CHE DOVREMO PRENDERE UNA DECISIONE SULLA BASE DEL VALORE DI SH, SFRUTTEREMO QUINDI IL MULTIPLEXER PER IMPLEMENTARE LO SHIFTER:

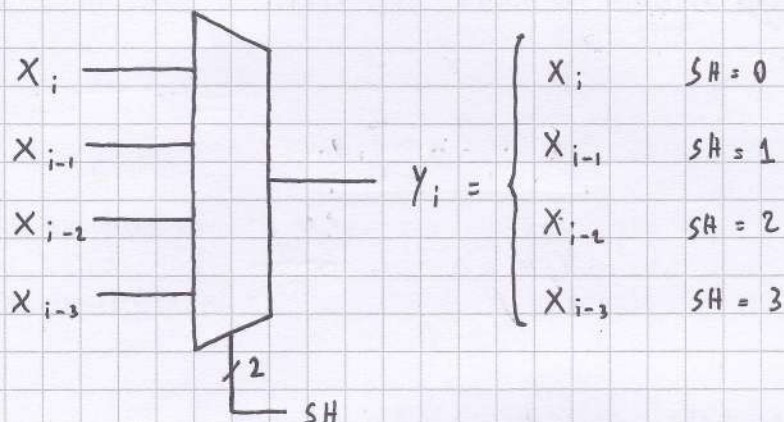




ORA IMMAGINIAMO DI DOVER CREARE UNO SHIFTER CHE IN BASE AL SEGNALE DI INGRESSO

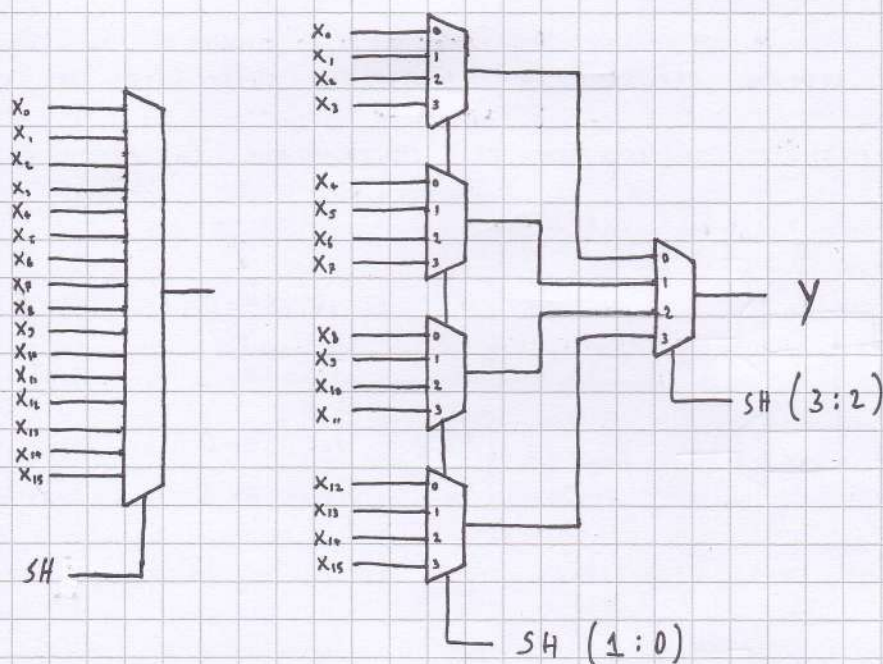
DECIDA SE SHIFTERE DI 0, 1, 2, 3 POSIZIONI LA NOSTRA SEQUENZA.

CI TROVEREMO NELLA SEGUENTE SITUAZIONE (CASO SAL):



QUESTO SINGOLO SHIFTER VIENE CHIAMATO **SHIFTER A BANDE**.

SOLITAMENTE NON CI SI SPINGE OLTRE I MULTIPLEXER CON 4 INGRESSI DATI, PIUTTOSTO SI UNISCONO A CASCATA, DI SEGUITO VEDIAMO COME IMPLEMENTARE UN MULTIPLEXER A 16 INGRESSI DATI CON 5 LA 4 INGRESSI DATI:

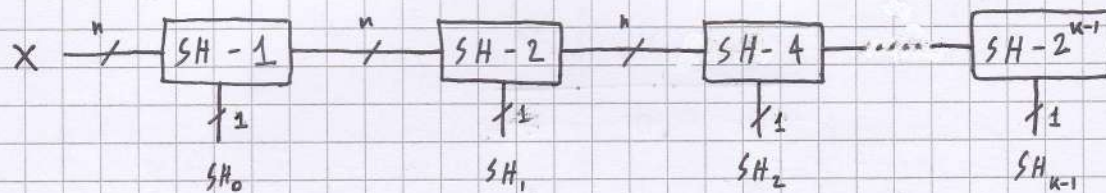


PER LA PRIMA SELEZIONE PRENDEREMO IL SECONDO E IL PRIMO BIT DI SH, INFATTI DIVIDENDO IN 4 BLOCCHI E SCAFIANDO IL 3° E 4° BIT CI LIMITIAMO A CONSIDERARE LA POSIZIONE DEL VALORE INDICATO DA SH ALL'INTERNO DEI 4 BLOCCHI, POI CON IL TERZO E QUARTO BIT ANDIAMO A SCEGLIERE A VALORE DEI 4 BLOCCHI (NON RICORDO)



APPARTIENE IL VALORE CHE STIAMO CERCANDO.

SAPENDO CHE MULTIPLEXER CON PIÙ INGRESSI DATI SONO PIÙ COSTOSI È INFATTI  
 RIDURRE LA VARIANTE COLLEGANDOLI PIUTTOSTO A CASCATA, CREANDO PIÙ LIVELLI.  
 IN PARTICOLARE CONSIDERIAMO LA SEGUENTE STRUTTURA:



DOVE OGNI SHIFTER È CONTROLLATO DA SEGNALI DI CONTROLLO DA 1 BIT, CHE  
 FARÀ SHIFTARE DI 0 O  $2^n$  POSIZIONI, DOVE  $n$  È IL PENICO DEL SEGNALE DI  
 CONTROLLO.

IN QUESTO MODO USEREMO SOLO MULTIPLEXER DA 2 INGRESSI DATI COLLEGATI  
 PERÒ A CASCATA. CI SARÀ POSSIBILE QUINDI, TRAMITE  $k-1$  SEGNALI  
 DI CONTROLLO, EFFETTUARE SPOSTAMENTI DA 0 FINO A  $2^k - 1$  BIT.

~~DATI  $k$  SEGNALI DI CONTROLLO~~ SE VOLESSIMO AVERE LA POSSIBILITÀ  
 DI EFFETTUARE SPOSTAMENTI DA 0 A  $k$  BIT CI SERVIREBBERO:

$$\log_2 (k + 1)$$

LIVELLI DI MULTIPLEXER. PER QUESTO MOTIVO LO CHIAMEREMO **SHIFTER LOGARITMICO**,  
 CHE È UN'ALTERNATIVA AL BARREL SHIFTER.

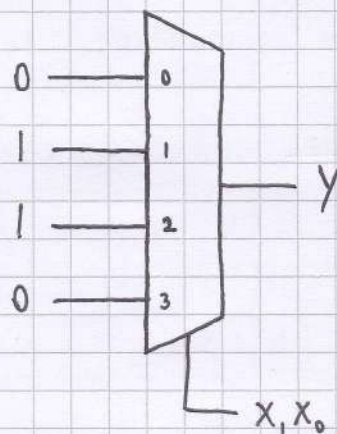


IN QUESTO BREVE PARAGRAFO VEDREMO COME SFRUTTARE LE FUNZIONALITÀ DI UN MULTIPLEXER PER RAPPRESENTARE UNA FUNZIONE.

CONSIDERIAMO IL SEGUENTE ESEMPIO:

| $X_1$ | $X_0$ | $Y$ |
|-------|-------|-----|
| 0     | 0     | 0   |
| 0     | 1     | 1   |
| 1     | 0     | 1   |
| 1     | 1     | 0   |

L'IMPLEMENTAZIONE DI  $Y$  TRAMITE MULTIPLEXER È BANALISSIMA, INFATTI CI BASTERÀ METTERE COME INGRESSI DI SEGNALI  $X_0$  E  $X_1$ , E COME INGRESSI DATI I RISPETTIVI VALORI DI  $Y$ .



DIVERSO È IL CASO IN CUI CI VENISSE DATA UNA FUNZIONE A 4 VARIABILI, CHE QUINDI RICHIEREBBE 16 INGRESSI DATI NEL MULTIPLEXER, E POSSIAMO OBBLIGATI AD UTILIZZARE MULTIPLEXER DA 8 INGRESSI DATI.

SUPPONIAMO, AD ESEMPIO, DI AVERE:

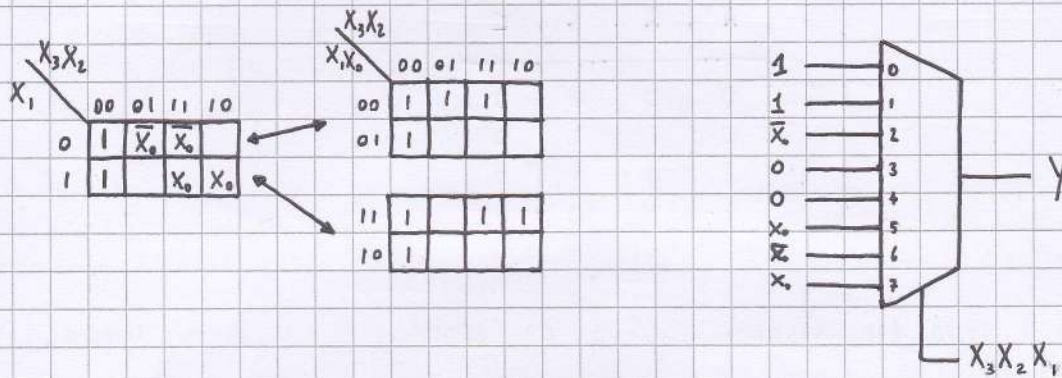
| $X_3 X_2$ | $X_1 X_0$ | 00 | 01 | 11 | 10 |
|-----------|-----------|----|----|----|----|
| 00        |           | 1  | 1  | 1  |    |
| 01        |           | 1  |    |    |    |
| 11        |           | 1  |    | 1  | 1  |
| 10        |           | 1  |    |    |    |



ORA, PER POTER IMPLEMENTARE LA FUNZIONE CON UN MULTIPLEXER DA 3 BIT  
DOBBIAMO RESTRINGERE LA K-MAP.

IN SOSTANZA NON POTREMO AVERE DEI VALORI COSTANTI IN INGRESSO ~~MA~~ <sup>E</sup> IN ALCUNI  
DI ESSI DOVRÀ COMPARIRE UNA DELLE 4 VARIABILI DELLA FUNZIONE.

SEGLIAMO LA VARIABILE  $X_0$  E RESTRINGIAMO LA K-MAP:



A SINISTRA LA K-MAP SINTETIZZATA E A DESTRA IL MULTIPLEXER CON LA VARIABILE  
 $X_0$  IN INGRESSO.



## I SOMMATORI

DOPO IL DECODER, IL MULTIPLEXER E LO SHIFTER PROVAMO A COSTRUIRE UN CIRCUITO CHE CI PERMETTA DI CALCOLARE LA SOMMA TRA DUE NUMERI.

ANZITUTTO CONSIDERIAMO UN SEMPLICE ESEMPIO:

| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

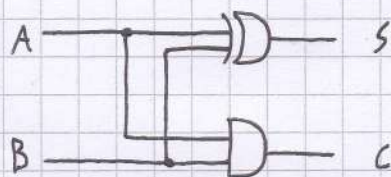
DOVE A E B SONO LEI ADDENDI, S È LA SOMMA E C È IL RIPORE.

DOBBIAMO GUARDARE ALLA TABELLA COME UN ELENCO DEI POSSIBILI CASI CHE CI SI POTRANNO PRESENTARE CON I RELATIVI RISULTATI.

È IMMEDIATO NOTARE CHE LA SOMMA CONSISTE IN UNO XOR E IL RIPORE IN UN AND, CIÒ:

$$S = A \oplus B \quad C = A \cdot B$$

NOTIAMO PERÒ CHE SE DOVESSIMO EFFETTUARE LA SOMMA DI 2 NUMERI QUESTA TABELLA ANDREBBE SICURAMENTE BENE PER IL PRIMO CASO (SOMMA DELLE CIFRE MENO SIGNIFICATIVE) MA LÌ AL SECONDO CASO POTREBBE PERDERSI IL RIPORE DELLA PRIMA SOMMA (NON È INFATTI PREVISTO UN RIPORE IN ENTRATA, MA SOLO IN USCITA). È PER QUESTO MOTIVO CHE IL SEGUENTE CIRCUITO:



PRENDE IL NOME DI SEMI SOMMATORE.



ORA PERFEZIONIAMO LA NOSTRA FUNZIONE IN MANIERA DA PREVEDERE

ANCHE UN PORTO IN ENTRATA, ESSA SARÀ COSÌ DEFINITA:

| A | B | $C_{in}$ | S | $C_{out}$ |
|---|---|----------|---|-----------|
| 0 | 0 | 0        | 0 | 0         |
| 0 | 1 | 0        | 1 | 0         |
| 1 | 0 | 0        | 1 | 0         |
| 1 | 1 | 0        | 0 | 1         |
| 0 | 0 | 1        | 1 | 0         |
| 0 | 1 | 1        | 0 | 1         |
| 1 | 0 | 1        | 0 | 1         |
| 1 | 1 | 1        | 1 | 1         |

ECCO QUI LA TABELLA CHE NASCUME TUTTI I CASI, PROVAMO ORA A CAPIRE COME OTTENERE S E  $C_{out}$ . INIZIAMO DA S:

| $C_{in}$ | AB |    |    |    |
|----------|----|----|----|----|
|          | 00 | 01 | 11 | 10 |
| 0        | 0  | 1  | 0  | 1  |
| 1        | 1  | 0  | 1  | 0  |

NONOCCIAMO SUBITO IL PATTERN A SCACCHIERA DELLA FUNZIONE XOR, PERTANTO AVREMO:

$$S = A \oplus B \oplus C_{in}$$

DIVERSO È IL CASO DEL CARRY OUT, INFATTI LA SUA K-MAP SARÀ:

| $C_{in}$ | AB |    |    |    |
|----------|----|----|----|----|
|          | 00 | 01 | 11 | 10 |
| 0        | 0  | 0  | 1  | 0  |
| 1        | 0  | 1  | 1  | 1  |

DUNQUE, CERCANDO DI MINIMIZZARE LA FUNZIONE CHE ESPRIME  $C_{out}$ , AVREMO:

$$C_{out} = A \cdot B + C_{in} \cdot B + C_{in} \cdot A$$

SI POTREBBE PERÒ, VISTO CHE ABBIAMO GIÀ INTRODOTTO IL SEM SOMMATORE, CERCARE DI "RICICLARE" LE OPERAZIONI FATTE IN QUELLO PER POTERLO SFRUTTARE IN QUESTO CASO. RISCRIVIAMO QUINDI LA FUNZIONE CON GLI IMPLICANTI VERBI:

$$C_{out} = AB + \bar{A}B C_{in} + A\bar{B} C_{in}$$

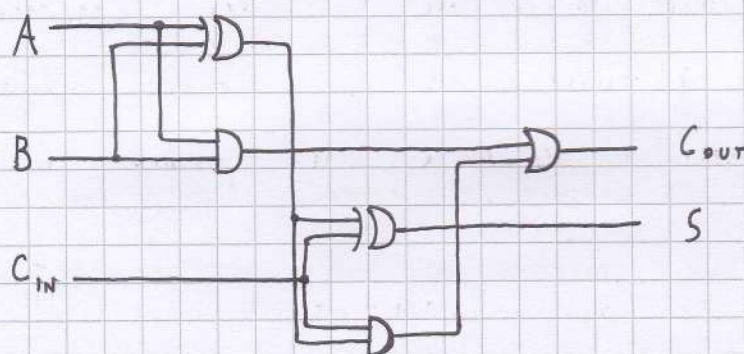


CON UN SEMPLICE ACCOLLIMENTO OTTENIAMO:

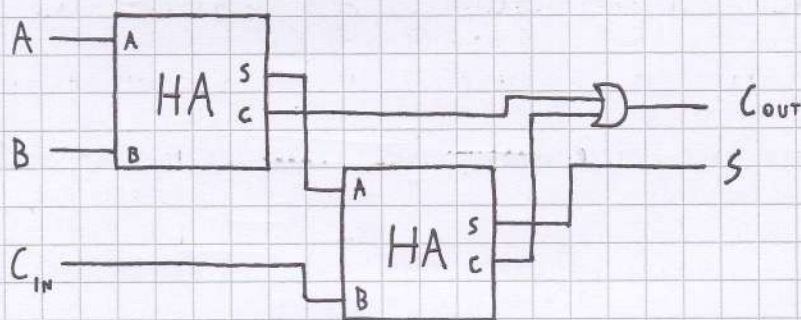
$$C_{OUT} = AB + C_{IN} (A \oplus B)$$

CHE RUTILIZZA 2 DELLE FUNZIONI USATE NEL SEMI SOMMATORE.

DISEGNAMO QUINDI IL CIRCUITO DEL **SOMMATORE COMPLETO**:



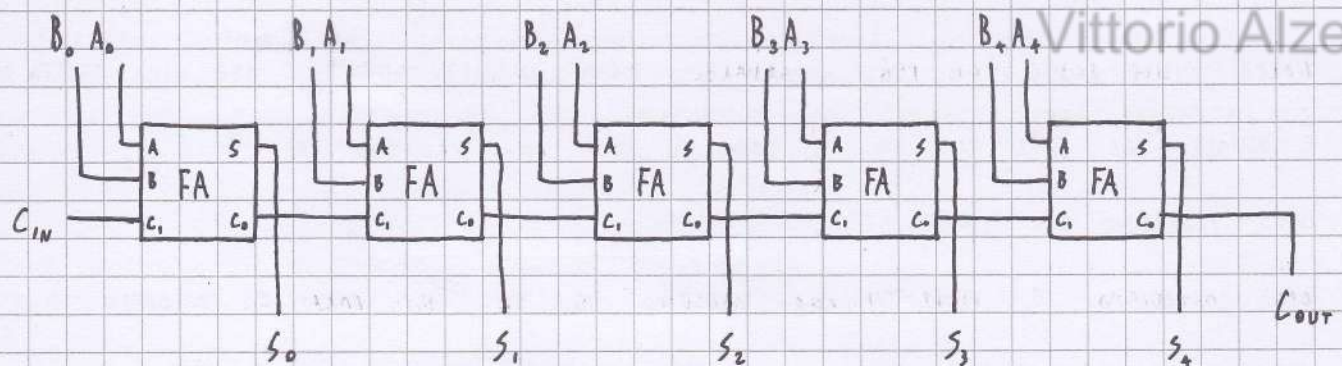
SI OSSERVI CHE IL FATTO DI AVER RUTILIZZATO LE FUNZIONI TROVATE PER IL SEMI SOMMATORE CI PERMETTE DI RISPONDERE IL CIRCUITO COME:



ORA CHE ABBIAMO TUTTI GLI STRUMENTI PER EFFETTUARE LA SOMMA DI 3 BIT NON CI RESTA CHE UNIRE AN PIÙ AB. FULL ADDER (O SOMMATORI COMPLETI) PER POTER SOMMARE PIÙ BIT INSIEME.

SI OSSERVI CHE PER LA SOMMA DEI BIT MENO SIGNIFICATIVI SI POTREBBE USARE UN HALF ADDER (SEMI SOMMATORE), TUTTAVIA IN ALCUNI CASI È CONVENIENTE DIVIDERE IN PIRELLI UNA SOMMA E PER QUESTO MOTIVO POTREBBE ESSERCI BISOGNO DI AVERE UN CARRY IN ANCHE NELLA SOMMA DEI BIT MENO SIGNIFICATIVI, QUINDI DISEGNAMO IL CIRCUITO:





NOTIAMO SUBITO CHE NONOSTANTE LE PORTE XOR PER IL CALCOLO DELLE SOMME SIANO PIÙ DISPENDIOSE IN TERMINI DI TEMPO, IL SEGNALE RELATIVO AL CARRY OUT DEVE PASSARE PER TUTTI I FULL ADDER PRIMA DI ESSERE CALCOLATO.

DUNQUE INTUIAMO CHE IL CRITICAL PATH RIVARSI LA PROPAGAZIONE DEL CARRY, PER QUESTO MOTIVO CHIAMAMO QUESTO SOMMATORE **RIPPLE CARRY** (RIPPLE = ONDULAZIONE).

IL TEMPO DI PROPAGAZIONE DEL RIPORTO SARÀ UN VALORE DEL TIPO:

$$T = n \cdot T_{\text{CARRY}}$$

DOVE  $T_{\text{CARRY}}$  È IL TEMPO CHE CI METTE OGNI SINGOLO FA AD ELABORARE IL CARRY.

UN TEMPO LINEARE PER UN'OPERAZIONE COSÌ IMPORTANTE PERÒ NON CI VA BENE,

PER OTTIMIZZARLO CERCHIAMO DI CAPIRE QUALI INFORMAZIONI POSSONO ESSERE ESTRATTE DAI VALORI DI A E B RIGUARDANTI IL CARRY IN USCITA.

SCRIVIAMO QUINDI LA TABELLA DI VERITÀ DELLA FUNZIONE CHE SOMMA 3 BIT:

| A | B | C <sub>IN</sub> | S | C <sub>OUT</sub> |
|---|---|-----------------|---|------------------|
| 0 | 0 | 0               | 0 | 0                |
| 0 | 1 | 0               | 1 | 0                |
| 1 | 0 | 0               | 1 | 0                |
| 1 | 1 | 0               | 0 | 1                |
| 0 | 0 | 1               | 1 | 0                |
| 0 | 1 | 1               | 0 | 1                |
| 1 | 0 | 1               | 0 | 1                |
| 1 | 1 | 1               | 1 | 1                |

OVVIAMENTE NOTIAMO CHE (NELLE VERDI) QUANDO SIA A CHE B SONO PARI A 1

IL C<sub>OUT</sub> SARÀ 1, IN TAL CASO SI DIRÀ CHE IL CARRY È STATO **GENERATO**

E POTRÀ ESSERE CALCOLATO SENZA ASPETTARE IL C<sub>IN</sub>.



NELLE ALTE ROSSE SI PUÒ OSSERVARE, MENO INTUITIVAMENTE, CHE IL CARRY OUT È UGUALE AL CARRY IN SE È VERA 1 SOLO UNA VARIABILE TRA A E B, IN QUESTO CASO PARLEREMO DI CARRY PROPAGATO.

ORA COSTRUIAMO 2 FUNZIONI CHE VALGONO 1 SE IL CARRY È GENERATO O PROPAGATO, ANDRETTUTTO RACCIAMONE UNA CHE CHIAMEREMO G, VERA SE IL CARRY È GENERATO:

$$G = A \cdot B$$

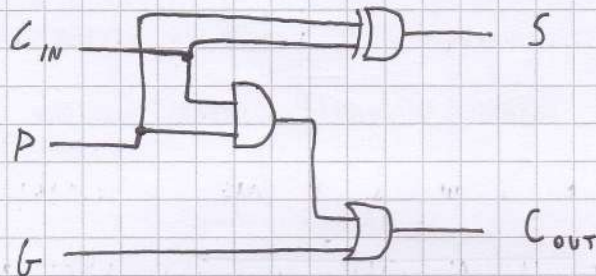
ORA UNA, CHE CHIAMEREMO P, VERA QUANDO IL CARRY È PROPAGATO:

$$P = A \oplus B$$

BENE, ORA NON CI RESTA CHE COSTRUIRE LA FUNZIONE  $C_{i+1}$  VERA QUANDO IL  $C_{OUT}$  VALE 1:

$$C_{i+1} = G + C_i \cdot P$$

IN QUESTO MOMENTO POSSIAMO RISCRIVERE IL CIRCUITO DEL FULL ADDER IN QUESTA MANIERA:



DOVE P E G SONO CALCOLATE IN UNA FASE DI SETUP.

ORA, A RIGOR DI LOGICA, DOVREMO INDIVIDUARE LE CONDIZIONI CHE FANNO VERIFICARE IL CASO PEGGIORE (PROPAGAZIONE DEL CARRY PER TUTTO IL NIPPLE ADDER) E OTTIMIZZARLO. AVENDO DEFINITO LA FUNZIONE P SAPPIAMO CHE IL CASO PEGGIORE SI AVRÀ PER:

$$P_0 = P_1 = P_2 = \dots = P_n = 1$$

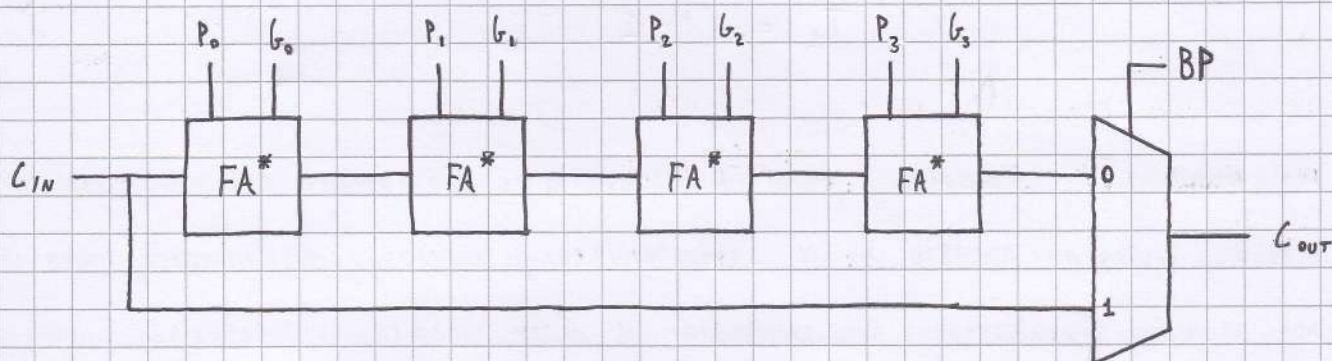
E IN QUESTO SPECIFICO CASO SI AVRÀ  $C_{IN} = C_{OUT}$ .



OSSERVIAMO QUINDI CHE IMPIEGANDO UN MULTIPLEXER CI SARÀ POSSIBILE DECIDERE SE FAR ATTRAVERSARE AL CARRY TUTTO IL CIRCUITO O PORTARLO SUBITO ALLA FINE IN QUANTO PROPAGATO IN TUTTI I GATE. COME SEGNALE DI CONTROLLO USEREMO OVVIAMENTE:

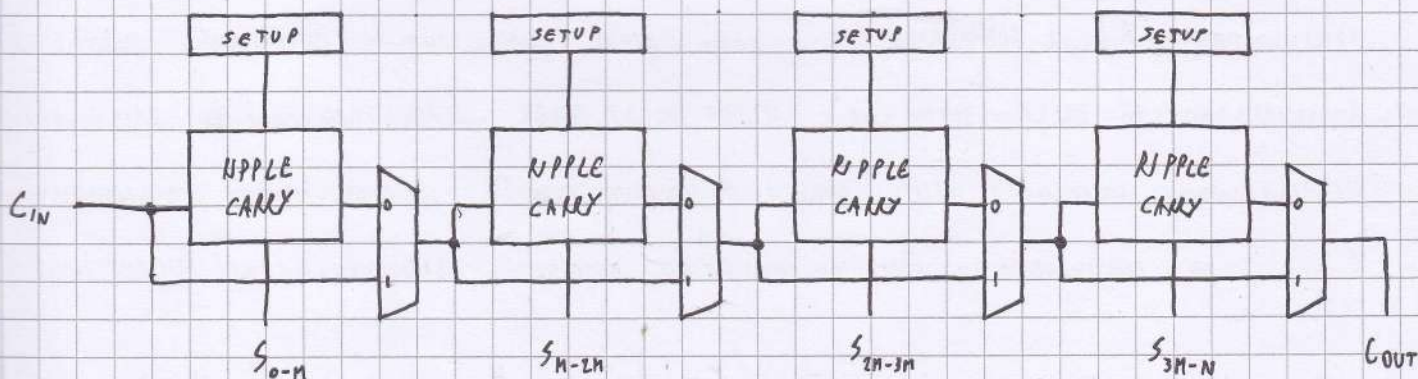
$$BP = P_0 \cdot P_1 \cdot P_2 \cdot \dots \cdot P_n$$

ADUNQUE IL NOSTRO NUOVO CIRCUITO SARÀ IL SEGUENTE (USERÒ DEI FULL ADDER ~~DEI~~ MODIFICATI, SI ASSUMA CHE LE FUNZIONI P E G VENGANO CALCOLATE IN UNA PRECEDENTE FASE DI SETUP):



NONOSTANTE L'INTRODUZIONE DEL MULTIPLEXER NON CI ABBIAMO RISPARMIATO MOLTO, DI FATTO IL CASO PEGGIORE OMA ANCHÉ ESSERE LA PROPAGAZIONE DEL CARRY PER N FA È LA PROPAGAZIONE PER N-1 FA.

È PER QUESTO MOTIVO CHE DECIDIAMO DI DIVIDERE I NUMERI DA SOMMARE IN BLOCCHI, IN MODO DA RIDURRE GLI EFFETTI NEGATIVI DEL CASO DESCRITTO SOPRA. CREIAMO QUINDI IL SEGUENTE MODELLO, CHE PRENDERÀ IL NOME DI **CARRY BYPASS**:





CERCHIAMO ORA DI CAPIRE IL TEMPO NECESSARIO AD ELABORARE IL CARRY NEL CASO PEGGIORE. PER INNA COSA NOTIAMO CHE NON CI SARÁ POSSIBILE FARE QUESTA ANALISI CON I METODI GIÁ VISTI (ANALISI STATICA), INFATTI ESSA CI PORTEREBBE A INDIVIDUARE COME CASO PEGGIORE QUELLO IN CUI IL CARRY VIENE PROPAGATO SEMPRE, MA QUESTO É PROPRIO IL CASO CHE ANZICHÉ EVITIAMO UNGE AI MULTIPLEXER.

É NECESSARIO RAZIONARE GUARDANDO ALLA STRUTTURA DEL CARRY BYPASS.

PER IL PRIMO BLOCCO IL CASO PEGGIORE É RAPPRESENTATO DA UN CARRY GENERATO NEL PRIMO FA E PROPAGATO ATTRAVERSO TUTTI GLI ALTA, DUNQUE ~~UN~~ RICHIEDE UN TEMPO PAR A:

$$T_1 = (N-1) \cdot t_{\text{CARRY}}$$

NEI CASI SUCCESSIVI, SE SI PRESENTASSE LO STESSO CASO DEL PRIMO BLOCCO, NON SI AVREBBE ~~UN~~ IL CASO PEGGIORE, INFATTI, SE IN ANCHE SOLO UNO DEI BLOCCHI SUCCESSIVI SI GENERASSE IL CARRY ALLORA SI POTREBBE INIZIARE LÍ NELLA PROPAGAZIONE. OVVIAMENTE IL CASO PEGGIORE NEI BLOCCHI SUCCESSIVI É RAPPRESENTATO DAL PASSAGGIO DEL CARRY ATTRAVERSO TUTTI I MULTIPLEXER, QUINDI:

$$T_2 = \left( \frac{N}{M} - 1 \right) \cdot t_{\text{MUX}}$$

DOVE  $N/M$  RAPPRESENTA IL NUMERO DI BLOCCHI (E QUINDI DI MULTIPLEXER) E IL  $-1$  SERVE A NON CONSIDERARE L'ULTIMO, INFATTI NELL'ULTIMO BLOCCO NON SI PÓV AVERE UNA PROPAGAZIONE TOTALE, NON SAREBBE IL CASO PEGGIORE.

BISOLTA FARE IN MODO CHE IL BLOCCO DEBBA ASPETTARE IN INGRESSO IL CARRY IN, NELL'ULTIMO FA SI GENERERÁ ~~PERÓ~~, COSÍ SIAMO SICURI CHE IL MULTIPLEXER NON PARÁ ATTRAVERSARE AL CARRY TUTTO IL RIPLE CARRY ADDER (PER CALCOLARE IL BIT DI SOMMA). RACCONTANDO ESSENDO IL CARRY GENERATO ~~IN~~ NELL'ULTIMO FA SI AVRÁ SUBITO IL RISULTATO DEL COUT, MA PER FARE LA SOMMA DI QUEL BLOCCO BISOGNERÁ ASPETTARE CHE SI PROPAGHI IN  $M-1$  BIT

$$T_3 = (N-1) \cdot t_{\text{CARRY}} + t_{\text{SUM}}$$

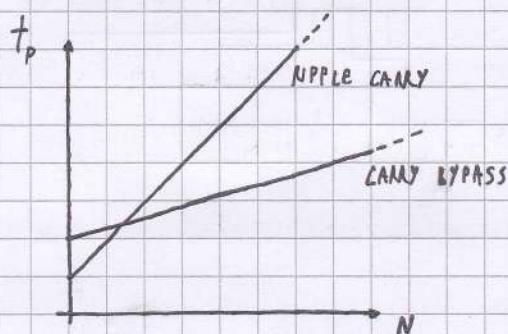


IN CONCLUSIONE, IL CASO PEGGIORE AVRÀ IL SEGUENTE TEMPO DI ESECUZIONE:

$$t_{\text{adder}} = t_{\text{setup}} + (n-1)t_{\text{carry}} + \left(\frac{N}{M}-1\right) \cdot t_{\text{mux}} + (n-1) \cdot t_{\text{carry}} + t_{\text{sum}}$$

NOTIAMO CHE IL TEMPO DI PROPAGAZIONE È ANCORA IN FORMA LINEARE, TUTTAVIA È POSSIBILE TROVARE UN BILANCIAMENTO ANDANDO A MODIFICARE  $N$ .

CI RITROVEREMO IN UNA SITUAZIONE DEL GENERE:



SI NOTA CHE PER  $N$  BASSI RISULTA MIGLIORE IL NIPPLE CARRY, QUESTO PERCHÉ VOLENDO IMPLEMENTARE UN CARRY SKIP (O BYPASS) CI SI TROVA A DOVER INSERIRE DEI MULTIPLEXER, CHE PER  $N$  PICCOLI NON OTTIMIZZANO MOLTO <sup>MA</sup> RAPPRESENTANO UN COSTO IMPORTANTE.

## CARRY SELECT ADDER

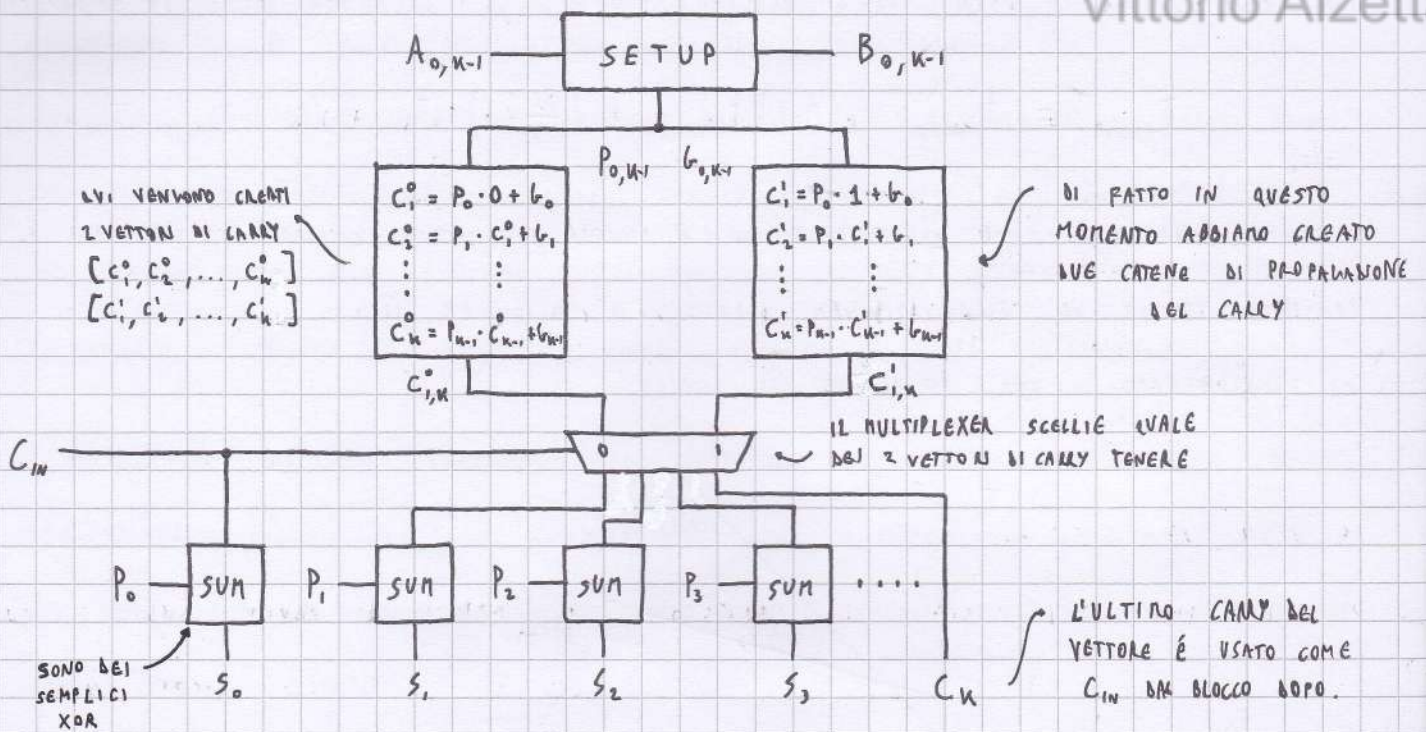
DOPO AVER VISTO IL NIPPLE CARRY E IL CARRY SKIP INTRODUCIAMO UN NUOVO SOMMATTORE, IL CARRY SELECT.

IL PRINCIPIO CHE STA ALLA BASE DI QUESTO SOMMATTORE È QUELLO DEL PRECALCOLO, INFATTI È POSSIBILE CALCOLARE IN ANTICIPO QUALE SARÀ IL  $C_{out}$  A SECONDA CHE IL  $C_{in}$  SIA 0 O 1.

FACENDO COSÌ, QUANDO ARRIVERÀ IL VERO  $C_{in}$ , SAPREMO SUBITO (CON UN MULTIPLEXER) IL VALORE DEL CARRY IN USCITA. QUESTO METODO È FATTIBILE IN QUANTO I POSSIBILI VALORI CHE LA VARIABILE DI INGRESSO (IL  $C_{in}$ ) PUÒ AVERE SONO SOLO 2 (0 E 1).

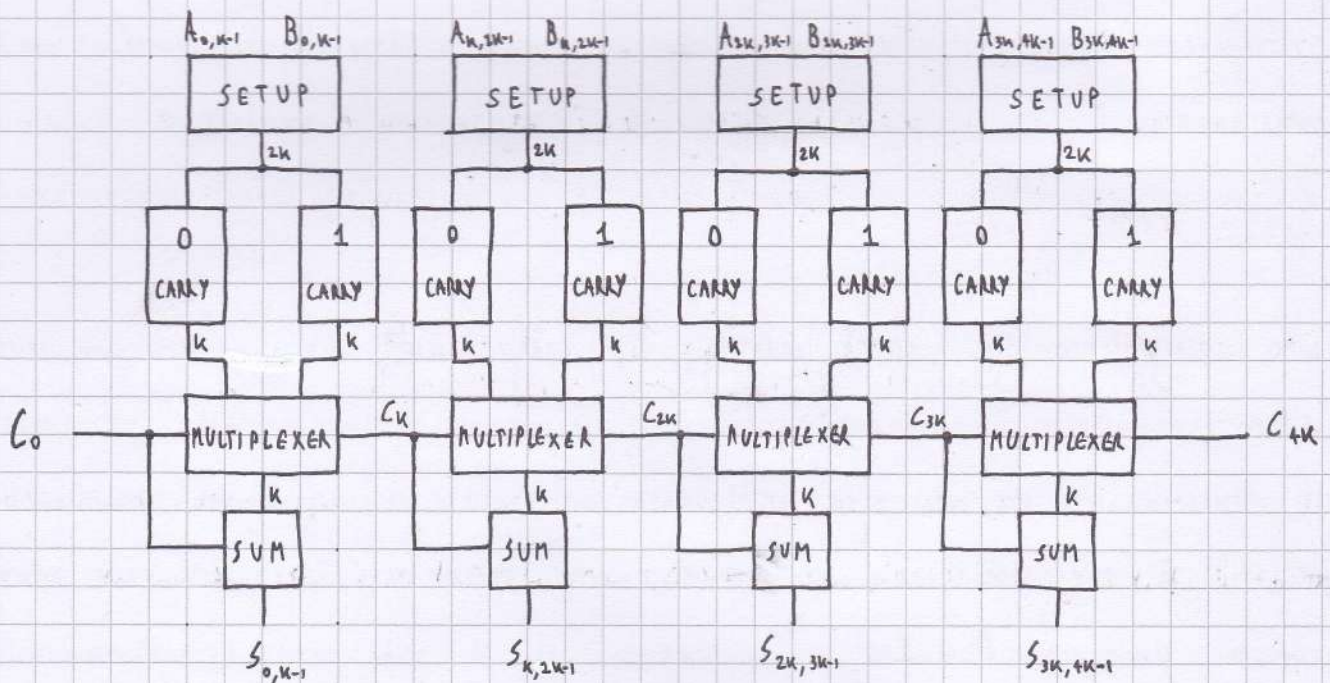
IN PARTICOLARE ORA LA STRUTTURA DI UN BLOCCO SARÀ LA SEGUENTE:





IN QUESTO CASO DA COSA SARÀ RAPPRESENTATO IL CASO PELLIDARE?

PER CAPIRE ANCH'UNO RAPPRESENTIAMO MARCIAMENTE PIÙ BLOCCHI DEL CARRY SELECT ADDER:



IL TEMPO DI PROPAGAZIONE DOVRA' TENERE CONTO DI UN TEMPO PER IL SETUP (VIALE PER TUTTI, AVVIENE CONTEMPORANEAMENTE), DEL TEMPO NECESSARIO A CALCOLARE I VETTORI DI CARRY DA K ELEMENTI (SONO CALCOLATI TUTTI CONTEMPORANEAMENTE), DELL'AVVIO ALL'ULTIMO BLOCCO DEL  $C_{IN}$  E ~~DOPO~~ ANCHE DELLA SOMMA DELL'ULTIMO BLOCCO.



LA FORMULA SARÁ PERTANTO:

Vittorio Alzetta

$$t_p = t_{\text{setup}} + K \cdot t_{\text{carry}} + \left(\frac{N}{K}\right) \cdot t_{\text{mux}} + t_{\text{sum}}$$

NOTIAMO PERÓ CHE I RILLECIONAMENTI IN TERMINI TEMPORALI RISPETTO AL CARRY SKIP NON SONO GIUSTIFICATI DAL RADDOPPIO DELLA LARGHEZZA DEL CIRCUITO (SONO RADDOPPIATE LE CATENE DI PROPAGAZIONE DEL CARRY).

TUTTAVIA, GRAZIE AD UN PICCOLO ACCORRIMENTO CI SARÁ POSSIBILE CAMBIARE COMPLETAMENTE LA SITUAZIONE.

NOI DOBBIAMO CAPIRE CHE IL CALCOLO DEI VETTORI DI CARRY AVVIENE IN CONTEMPORANEA PER TUTTI I BLOCCHI, QUINDI POSSIAMO DIRE CHE L'ULTIMO BLOCCO RIMANE SENZA FAR NULLA FINO ALL'ARRIVO DEL CARRY IN ELABORATO DA TUTTI GLI ALTRI BLOCCHI.

SI POTREBBE ASTUTAMENTE FAR CALCOLARE AI BLOCCHI DOPO IL PRIMO UN BIT IN PIÚ NELLA SOMMA E NEL VETTORE DI CARRY, IN MODO TALE DA OCCUPARE IL TEMPO CHE BUTTEREBBERO VIA MENTRE ASPETTANO L'ARRIVO DEL  $C_{in}$ .

QUESTA SCELTA CI PERMETTE DI FAR CALARE DISTRATTAMENTE IL TEMPO DI PROPAGAZIONE, SE INRATTI CONSIDERIAMO IL CASO PEGGIORE (CIOÉ QUEI TEMPI CHE RILVARDANO L'ULTIMO BLOCCO) AVREMO:

$$t_p = t_{\text{setup}} + K \cdot t_{\text{carry}} + t_{\text{mux}} + t_{\text{sum}}$$

AVREMO UN SOLO MULTIPLEXER DA CONTARE, PERCHÉ QUANDO IL SECONDO L'ULTIMO BLOCCO AVRÁ ELABORATO I 2 VETTORI (SI NOTI CHE QUELLI DELL'ULTIMO BLOCCO SONO I PIÚ GRANDI) IL SEGNALE SA DEL  $C_{in}$  SARÁ GIÁ PRONTO PER SCEGLIERE UNO DEI DUE VETTORI.

SI PUÓ DIMOSTRARE CHE PER  $N$  GRANDI  $K \approx \sqrt{2N}$  E QUINDI  $t_p$  PER QUESTO ABBERÁ AVRÁ UNA DIPENDENZA NON LINEARE DAL NUMERO DI BIT DELLA SOMMA ( $N$ ). LA DIMOSTRAZIONE É BANALE:

PER QUESTO LO CHIAMEREMO  
SQUARE ROOT CARRY SELECT ADDER

$$2+3+4+\dots+K = N \Leftrightarrow 1+2+\dots+K = N+1 \Leftrightarrow \frac{K(K+1)}{2} = N+1 \Rightarrow K^2 \approx 2N \Rightarrow K \approx \sqrt{2N}$$



INTRODUCIAMO ORA UN NUOVO TIPO DI SOMMATORE, CHE GRAZIE AL FATTO CHE LA SOMMA LOBE DELLA PROPRIETÀ ASSOCIATIVA (QUINDI PERMETTE DI DIVIDERE LA SOMMA IN BLOCCHI) CI PERMETTERÀ DI SCENDERE A TEMPI DI ELABORAZIONE PROPORZIONALI AL LOGARITMO DEL NUMERO DI BIT IN INGRESSO.

CHIAMEREMO QUESTO SOMMATORE **CARRY LOOK-HEAD ADDER**, CHE IN TERMINI DI AREA SARÀ IL PIÙ COSTOSO.

IL NOSTRO OBIETTIVO SARÀ QUELLO DI OTTENERE UN'ESPRESSIONE PER I VARI CARRY CHE DIPENDA SOLO DAI BIT DI PROPAGATE E GENERATE (SI HANNO SUBITO, SENZA TEMPI DI ATTESA).

A TAL FINE ESPRIMEREMO  $C_1$ ,  $C_2$  E  $C_3$  SFRUTTANDO SOLO I BIT  $P$  E  $G$ :

$$C_1 = P_0 C + G_0$$

$$C_2 = P_1 C_1 + G_1 = P_1 (P_0 C + G_0) + G_1 = P_1 P_0 C + P_1 G_0 + G_1$$

$$\begin{aligned} C_3 &= P_2 C_2 + G_2 = P_2 [P_1 (P_0 C + G_0) + G_1] + G_2 \\ &= P_2 P_1 P_0 C + P_2 P_1 G_0 + P_2 G_1 + G_2 \end{aligned}$$

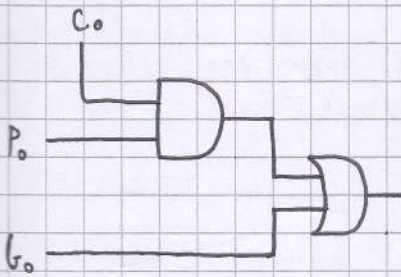
CHIAMEREMO QUESTE ESPRESSIONI, CHE CONTENGONO SEMPRE PIÙ LETTERALI, FACENDO RIFERIMENTO ALLE VARIABILI PRESENTI:

$$CG-1 (C, P_0, G_0) \quad CG-2 (C, P_0, P_1, G_0, G_1) \quad CG-3 (C, P_0, P_1, P_2, G_0, G_1, G_2)$$

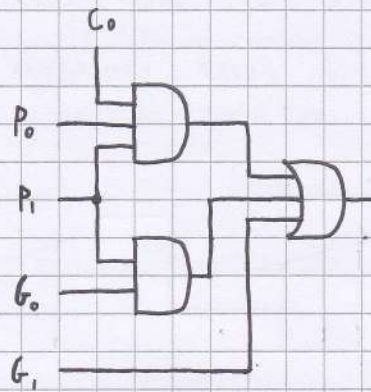
QUESTE FUNZIONI CI PERMETTONO DI CALCOLARE IMMEDIATAMENTE IL VALORE DEL CARRY 3 POSIZIONI PIÙ AVANTI.

RAPPRESENTIAMO IL CIRCUITO:

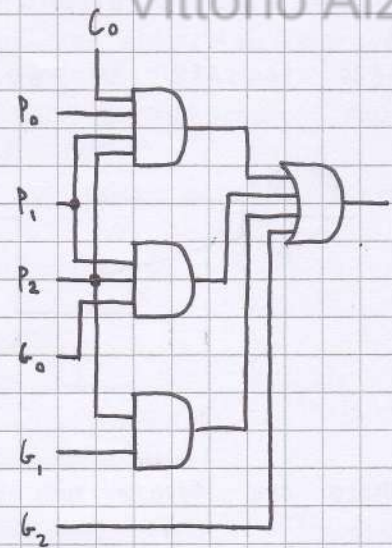




$$C_1 = P_0 C_0 + G_0$$

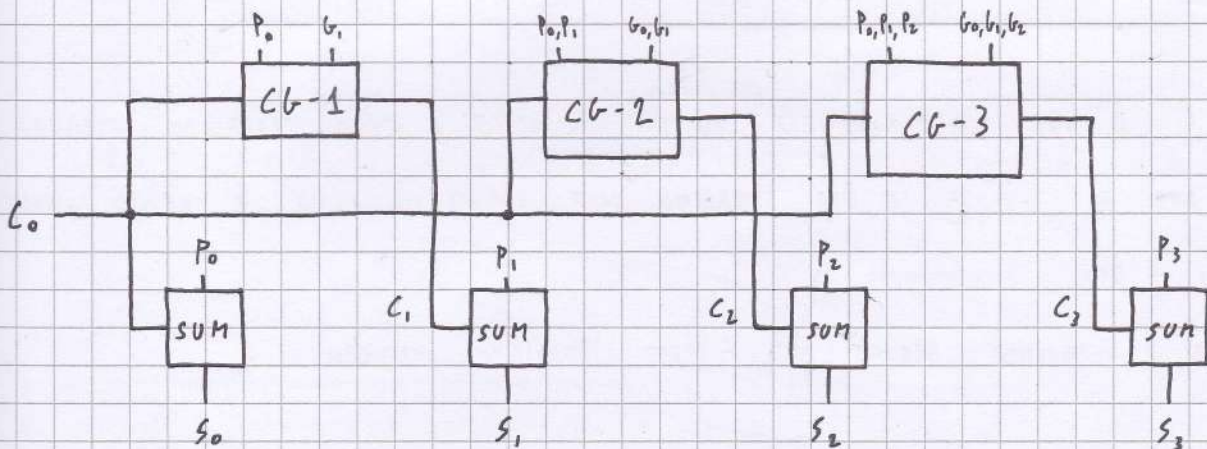


$$C_2 = P_1 C_1 + G_1$$

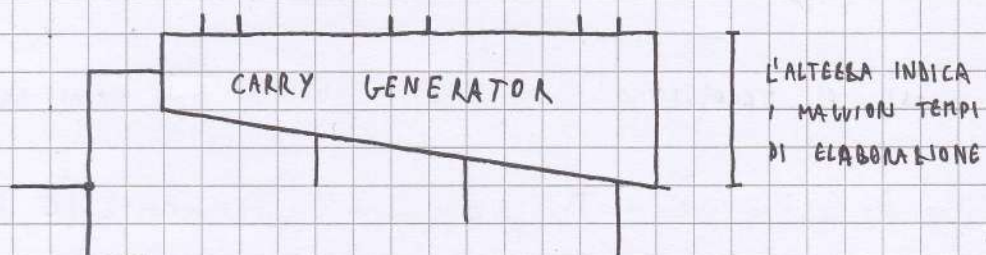


$$C_3 = P_2 C_2 + G_2$$

SI POTREBBE ANCHE PENSARE DI COSTRUIRE UN SOMMATORE CHE IMPLEMENTI QUESTI 3 CIRCUITI AD UNA IDEATA :

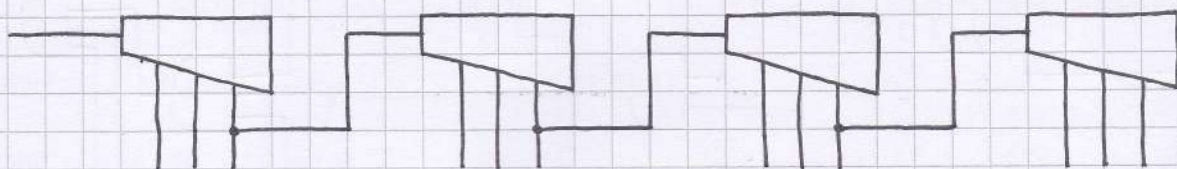


PER INDICARE IL FATTO CHE CG-3 HA TEMPI DI ELABORAZIONE MOLTO PIÙ LUNGI RISPETTO A CG-2 E CG-1 SOLITAMENTE SI USA LA SEGUENTE NOTAZIONE :





QUELLO CHE ABBIAMO OTTENUTO È UN CARRY LOOK-AHEAD DA 4-BIT, SI POTREBBE PENSARE DI UNIRE PIÙ CARRY GENERATOR FRA LORO PER OTTENERE:



MA QUELLO CHE AVREMO NON SARÀ ALTRO CHE UN PROPAGARSI DI BLOCCO IN BLOCCO DEL CARRY (TEMPO DI ESECUZIONE LINEARE).

PER QUESTO MOTIVO PROVIAMO A SPINGERCI OLTRE CALCOLANDO IL CARRY 4 POSIZIONI PIÙ AVANTI:

$$C_4 = P_3 C_3 + G_3 = \underbrace{P_3 P_2 P_1 P_0}_{P_{[0,3]}} C_0 + \underbrace{P_3 P_2 P_1 G_0 + P_3 P_2 G_1 + P_3 G_2 + G_3}_{G_{[0,3]}}$$

DOVE  $P_{[0,3]}$  È UNA FUNZIONE VERA QUANDO IL CARRY È STATO PROPAGATO ATTRAVERSO I PRIMI 4 BIT E  $G_{[0,3]}$  È UNA FUNZIONE VERA QUANDO IL CARRY È STATO GENERATO IN UNO DEI 4 BIT.

DUNQUE PER IL SECONDO BLOCCO DA 4 BIT POSSIAMO SCRIVERE:

$$C_5 = (P_4 C_4 + G_4) = CG-1 (C_4, P_4, G_4)$$

$$C_6 = (P_5 C_5 + G_5) = CG-2 (C_4, P_4, P_5, G_4, G_5)$$

$$C_7 = (P_6 C_6 + G_6) = CG-3 (C_4, P_4, P_5, P_6, G_4, G_5, G_6)$$

PROCEDENDO COSÌ CI ACCORGIAMO CHE LA STRUTTURA NON VARIA MAI E:

$$C_8 = P_{[4,7]} P_{[0,3]} C + P_{[4,7]} G_{[0,3]} + G_{[4,7]} = CG-2 (C, P_{[4,7]}, P_{[0,3]}, G_{[4,7]}, G_{[0,3]})$$

$$C_{12} = P_{[8,11]} P_{[4,7]} P_{[0,3]} C + P_{[8,11]} P_{[4,7]} G_{[0,3]} + P_{[8,11]} G_{[4,7]} + G_{[8,11]} = CG-3 (C, P_{[8,11]}, P_{[4,7]}, P_{[0,3]}, G_{[8,11]}, G_{[4,7]}, G_{[0,3]})$$



OSSERVIAMO CHE PER I PRIMI 16 BIT SI VIENE A CREARE UNA SORTA DI STRUTTURA AD ALBERO, DOVE POSSIAMO PENSARE DI AVERE UN CARRY GENERATOR IN CIMA (PER CALCOLARE SUBITO  $C_4, C_8, C_{12}$ ) DATO CHE IN UNA PRECEDENTE FASE DI SETUP ABBIAMO PREVENTIVAMENTE CALCOLATO:

$$P_{[0,3]}, P_{[4,7]}, P_{[8,11]}, G_{[0,3]}, G_{[4,7]}, G_{[8,11]}$$

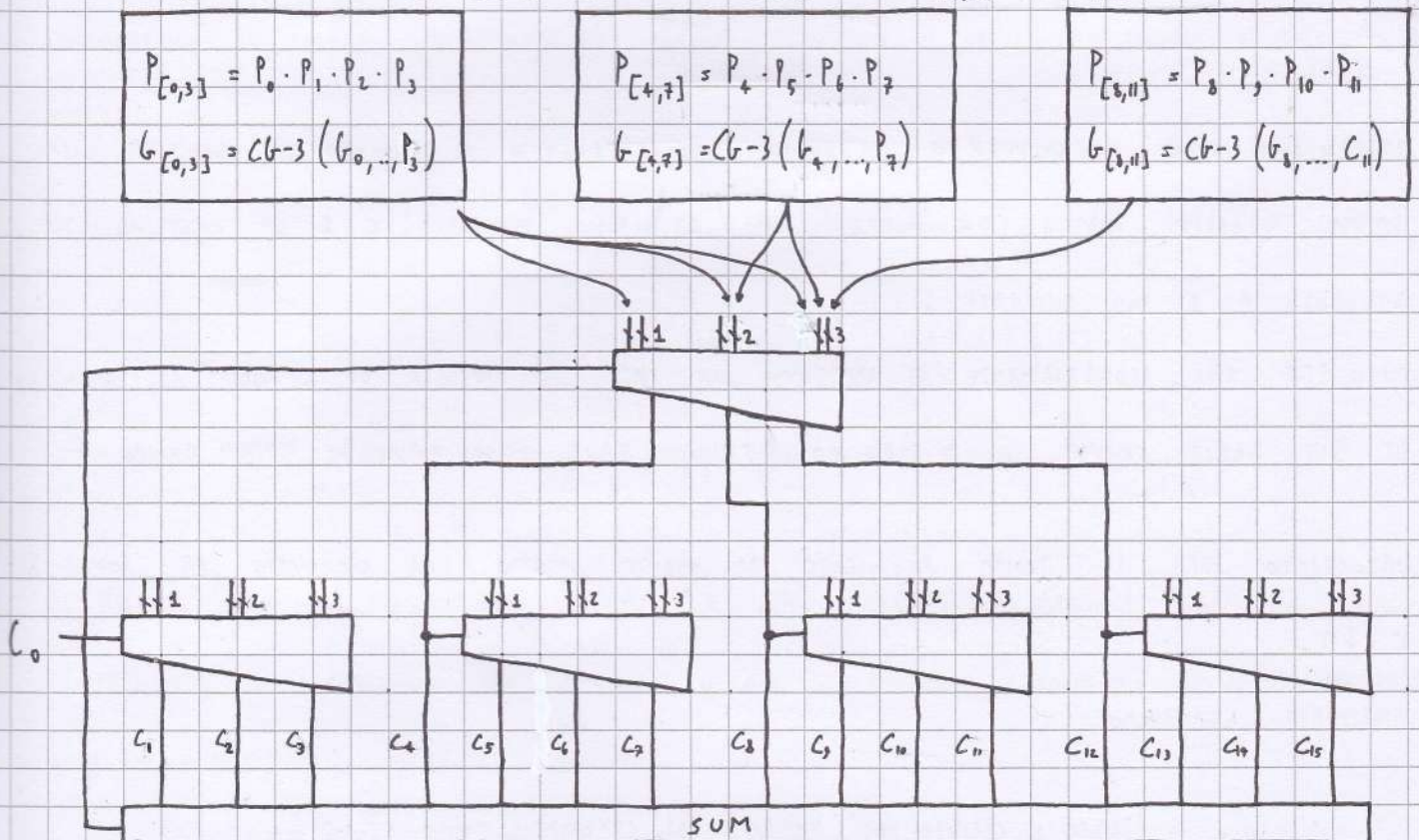
PER QUESTI È INFATTI SUFFICIENTE AVERE  $P_0, \dots, P_n$  E  $G_0, \dots, G_n$  (CALCOLATI IN UNA FASE DI SETUP ANCORA PRECEDENTE).

AVUTI  $C_4, C_8$  E  $C_{12}$  I RISULTATI SARANNO PASSATI AD UN SECONDO

LIVELLO DI CARRY GENERATOR CHE SI OCCUPERANNO DI CALCOLARE TUTTI I SINGOLI

CARRY CHE POI VERANNO SRAVITATI PER FARE LA SOMMA FINALE.

(IN REALTÀ NEL SETUP C'È UN BLOCCO CHE CALCOLA I BIT  $P_{[12,15]}$  E  $G_{[12,15]}$ , NE CAPIREMOPI IL SENSO)

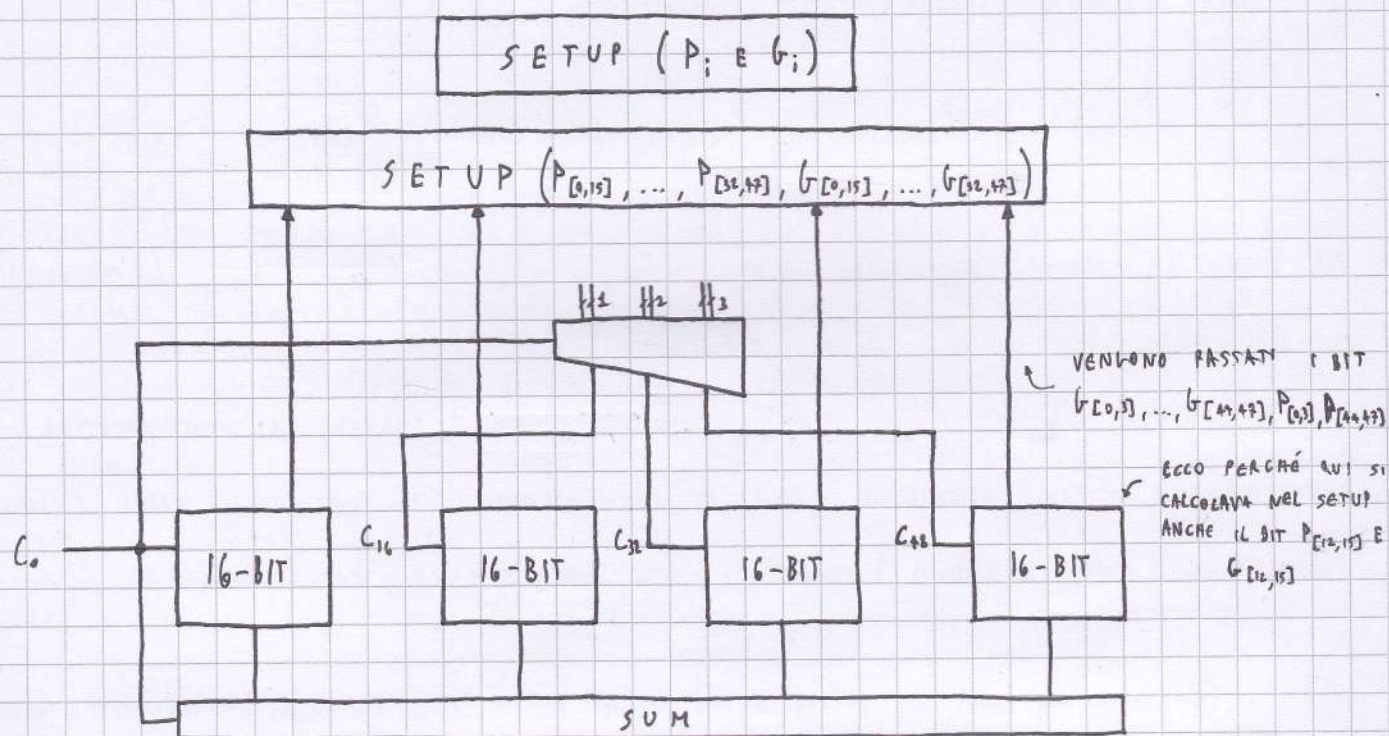


ORA IMMAGINIAMO DI PRENDERE IL SOMMATORE DA 16-BIT CHE ABBIAMO APPENA CREATO PER ANUNCIARNE ALTRI 3 ACCANTO. DI CONSEGUENZA AVREMO RISOLTO DI UN



CARRY GENERATOR IN CIMA A QUESTI SONO 4 BLOCCHI, CHE GENERANO IL BIT DI CARRY IN POSIZIONE 16, 32 E 48.

LA STRUTTURA, SENZA DISCERNERE I PARTICOLARI SARA:



OSSERVIAMO CHE NONOSTANTE ABBIAMO QUADRUPPLICATO IL NUMERO DI BIT DELLA SOMMA ABBIAMO DOVUTO AGLIUNGERE SOLO 1 LIVELLO DI SETUP E 1 DI CARRY GENERATOR, ARRIVANDO A 3 PER ENTRAMBI.

INTUIAMO CHE PROBABILMENTE LA RELAZIONE CHE LEGA IL NUMERO DI LIVELLI AL NUMERO DI BIT DELLA SOMMA È DI TIPO LOGARITMICO E LA BASE POTREBBE PROPRIO ESSERE 4.

CALCOLIAMO ORA IL TEMPO RICHIESTO DA QUESTO OBIETTO PER CALCOLARE LA SOMMA DI  $n$  BIT.

ANLITUTTO DERIVIAMO:

- $T_s$  = TEMPO DI CALCOLO DEL SETUP PER CALCOLARE SOLO  $P_i$  E  $G_i$ ;
- $L$  = NUMERO DI LIVELLI DI CARRY GENERATOR (E QUINDI DI SETUP)
- $T_{sum}$  = TEMPO RICHIESTO PER LA SOMMA (CIÒ UNA PAROLA SOLA, QUINDI POCO)
- $T_{CG-3}$  = TEMPO RICHIESTO PER IL CALCOLO DEL RISULTATO DI UN CG-3 (OVVERO UN CARRY-GENERATOR A 3 ORING).



OSSEVATO CHE IL NUMERO DI LIVELLI VA CON IL LOGARITMO IN BASE 4 DEL NUMERO DI BIT DA SOMMARE:

$$L = \log_4 n$$

E CHE IL  $T_{CL-3}$  NON È ALTRO CHE L'ATTRAVERSAMENTO DI 2 LIVELLI DI PORTE LOGICHE (E DUNQUE  $T_{sum}$  L'ATTRAVERSAMENTO DI UN SOLO LIVELLO DI PORTE LOGICHE):

$$T_{CL-3} = T_G \cdot 2 \quad T_{sum} = T_G$$

OTTENIAMO LA SEGUENTE ESPRESSIONE:

$$T_{CLA} = T_s + (L-1)T_{CL-3} + L \cdot T_{CL-3} + T_{sum}$$

DOVE  $(L-1)T_{CL-3}$  È DOVUTO ALLE  $L-1$  (UN LIVELLO DI SETUP È IL PRIMO, PER  $P_i$  E  $L_i$ , <sup>DOVE NON SI CALCOLA UN CL-3</sup> QUINDI SONO  $L-1$ ) LIVELLI DI SETUP IN CUI BISOGNA CALCOLARE IL RISULTATO DI UN CL-3 E  $L \cdot T_{CL-3}$  È DOVUTO ALLE  $L$  LIVELLI DI CARRY GENERATOR, IL CUI PERCORSO PIÙ DISPENDIOSO IN TERMINI DI TEMPO È PROPRIO L'ULTIMO (QUELLO CHE CALCOLA IL CL-3).  
DUNQUE:

$$\begin{aligned} T_{CLA} &\approx T_s + (L-1) \cdot 2 \cdot T_G + L \cdot 2 \cdot T_G + T_G \\ &\approx 2T_G + 2LT_G - 2T_G + 2LT_G \\ &\approx (4 \log_4 n) T_G \end{aligned}$$

E DUNQUE ABBIAMO TROVATO UN ADDER IL CUI TEMPO DI ELABORAZIONE NON PUÒ ESSERE MIGLIORATO, IL LOGARITMO DEL NUMERO DI BIT DA SOMMARE RAPPRESENTA UN LIMITE INFERIORE ALLA COMPLESSITÀ DI QUESTO PROBLEMA.

A SECONDA DELL'ATTIVITÀ CHE DOVRÒ SVOLGERE SCELGERÒ OPPORTUNAMENTE L'ADDER CHE FARÀ AL CASO MIO TRA QUELLI CHE ABBIAMO VISTO.

LA DECISIONE VERRÀ PRESA SULLA BASE DI QUANTO DOVRÒ ANDARE VELOCE O QUANTA AREA SARÒ DISPOSTO A USARE.



# I SOTTRATTORI

IN QUESTO CASO, VOLENDO IMPLEMENTARE UN SOTTRATTORE, NON PARLEREMO PIÙ DI CARRY (RIPORTO) MA DI BORROW (RESTO).

OSSERVIAMO PERÒ QUESTO FATTO:

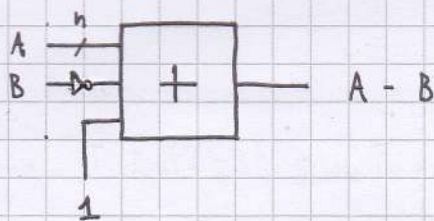
$$-X = \overline{X} + 1$$

QUANDO AVREMO A CHE FARE CON NUMERI NEGATIVI LAVOREREMO QUASI SEMPRE IN COMPLEMENTO A DUE, DUNQUE CI SARÀ POSSIBILE SCRIVERE:

$$A - B = A + (-B) = A + \overline{B} + 1$$

DOVE SIAMO PASSATI DALL'AVERE SOTTRAZIONI ALL'AVERE SOLO SOMME.

QUINDI SCELTO OPPORTUNAMENTE UN SOMMATORE POTREMO IMPLEMENTARE LA SOTTRAZIONE COME:

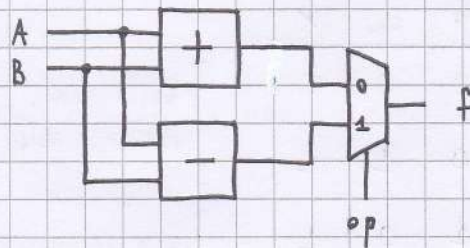


POTREBBE ESSERE COME UTILE IN ALCUNI CASI IMPLEMENTARE UN CIRCUITO PER LA SEGUENTE FUNZIONE:

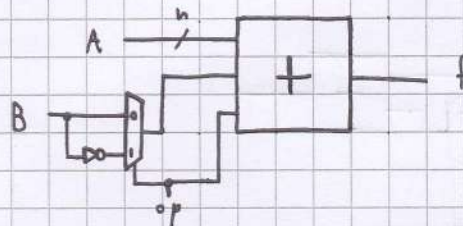
$$f(A, B, op) = \begin{cases} A + B & op = 0 \\ A - B & op = 1 \end{cases}$$

POICHÉ DOBBIAMO AVERE DIVERSAMENTE IN BASE AL SEGNALE DI CONTROLLO  $op$  POTREMMO PROVARE AD IMPLEMENTARLA CON UN MULTIPLEXER:





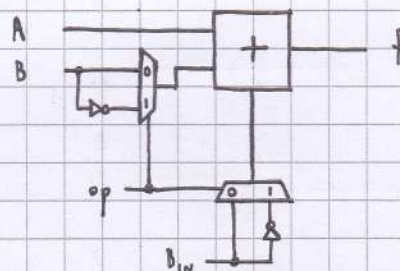
NON SIAMO CONTENTI DI QUESTO RISULTATO, È TROPPO DISPENDIOSO IN TERMINI DI SPAZIO, PROVIAMO A SPOSTARE DA IL MULTIPLEXER IN MODO DA FARE PRIMA LA SCELTA E POI LA SOMMA:



SI POTREBBE PERÒ ANCHE AVERE UN  $B_{in}$  (BORROW IN INGRESSO) E ALLORA LA FUNZIONE  $f$  DIVENTEREBBE:

$$f(A, B, B_{in}, op) = \begin{cases} A + B + B_{in} & op = 0 \\ A + \bar{B} + 1 - B_{in} & op = 1 \end{cases}$$

OSSERVATO CHE  $1 - B_{in} = \bar{B}_{in} + 1 + 1 = \bar{B}_{in}$  POSSIAMO SCRIVERE IL CIRCUITO CHE IMPLEMENTA  $f$  COME:

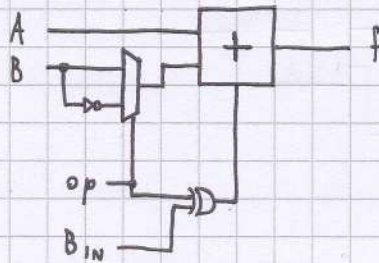


O, IN ALTERNATIVA, POSSIAMO OSSERVARE CHE IL VALORE CHE EFFETTIVAMENTE ENTRA NEL SOMMATORE COME BORROW IN È CALcolato A PARTIRE DALLA SEGUENTE TABELLA DI VERITÀ:



| op | $B_{in}$ | X |                                                          |
|----|----------|---|----------------------------------------------------------|
| 0  | 0        | 0 | NELLA SOMMA<br>NON CAMBIA NULLA                          |
| 0  | 1        | 1 |                                                          |
| 1  | 0        | 1 | NELLA SOTTRAZIONE<br>PRENDIAMO IL<br>CONTRO DEL $B_{in}$ |
| 1  | 1        | 0 |                                                          |

DA CUI MOSTRANO SUBITO CHE  $X = op \oplus B_{in}$ , IL CHE CI PERMETTE DI  
SEMPLIFICARE IL CIRCUITO DI PRIMA IMPLEMENTANDO UNA PORTA XOR ANZICHÉ UN MULTIPLEXER:



## CODIFICA DI GRAY

PRENDIAMO ORA UN ARGOMENTO CHE AVEVAMO GIÀ TRATTATO IN PASSATO, LA CODIFICA  
DI GRAY.

PROVIAMO A COSTRUIRE UN CIRCUITO CHE CI PERMETTA DI EFFETTUARE LA CONVERSIONE  
DA CODICE BINARIO A CODICE GRAY, DUNQUE PER PRIMA COSA SCRIVEREMO LA  
TABELLA DI VERITÀ (PER 3 BIT):

| $b_2$ | $b_1$ | $b_0$ | $g_2$ | $g_1$ | $g_0$ |
|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 0     |
| 0     | 0     | 1     | 0     | 0     | 1     |
| 0     | 1     | 0     | 0     | 1     | 1     |
| 0     | 1     | 1     | 0     | 1     | 0     |
| 1     | 0     | 0     | 1     | 1     | 0     |
| 1     | 0     | 1     | 1     | 1     | 1     |
| 1     | 1     | 0     | 1     | 0     | 1     |
| 1     | 1     | 1     | 1     | 0     | 0     |



ORA CI SCRIVIAMO LA K-MAP PER LE 3 TABELLE DI VERITÀ (UNA PER  $q_2$  COME RISULTATO, UNA PER  $q_1$ , E UNA PER  $q_0$ ):

| $b_2 \backslash b_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| 0                    | 0  | 1  | 1  | 0  |
| 1                    | 1  | 0  | 0  | 1  |

$$q_0 = \bar{b}_0 b_1 + b_0 \bar{b}_1 = b_0 \oplus b_1$$

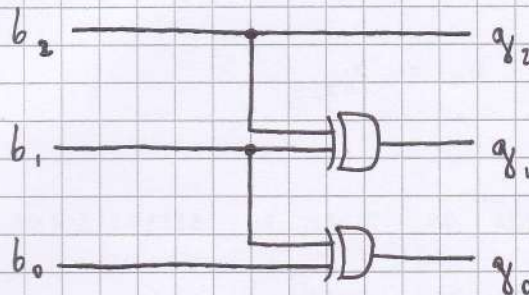
| $b_2 \backslash b_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| 0                    | 0  | 1  | 0  | 1  |
| 1                    | 0  | 1  | 0  | 1  |

$$q_1 = b_1 \bar{b}_2 + \bar{b}_1 b_2 = b_1 \oplus b_2$$

| $b_2 \backslash b_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| 0                    | 0  | 0  | 1  | 1  |
| 1                    | 0  | 0  | 1  | 1  |

$$q_2 = b_2$$

DA QUI POSSIAMO TRARRE SUBITO IL CIRCUITO DI CONVERSIONE, IMPLEMENTABILE TRAMITE PORTE LOGICHE DA 2 INGRESSI:



DI FATTO POSSIAMO CONCLUDERE CHE PER L' $i$ -ESIMO BIT (AD ESCLUSIONE DI QUELLO PIÙ SIGNIFICATIVO) IL SUO VALORE CONVERTITO SARÀ:

$$q_i = b_{i+1} \oplus b_i$$

DIVERSO È SE PROViamo A COSTRUIRE UN CIRCUITO CHE IMPLEMENTI LA CONVERSIONE DA CODICE GRAY A CODICE BINARIO.

SCRIVIAMO ANCHE IN QUESTO CASO LA TABELLA DI VERITÀ:



| $q_2$ | $q_1$ | $q_0$ |
|-------|-------|-------|
| 0     | 0     | 0     |
| 0     | 0     | 1     |
| 0     | 1     | 0     |
| 0     | 1     | 1     |
| 1     | 0     | 0     |
| 1     | 0     | 1     |
| 1     | 1     | 0     |
| 1     | 1     | 1     |

| $b_2$ | $b_1$ | $b_0$ |
|-------|-------|-------|
| 0     | 0     | 0     |
| 0     | 0     | 1     |
| 0     | 1     | 1     |
| 0     | 1     | 0     |
| 1     | 1     | 1     |
| 1     | 1     | 0     |
| 1     | 0     | 0     |
| 1     | 0     | 1     |

SCRIVIAMO QUINDI LE RELATIVE K-MAP:

| $q_2 \backslash q_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| $q_0$                | 0  | 0  | 1  | 0  |
|                      | 1  | 1  | 0  | 1  |

$$b_0 = (q_2 \oplus q_1) \oplus q_0 = q_2 \oplus q_1 \oplus q_0$$

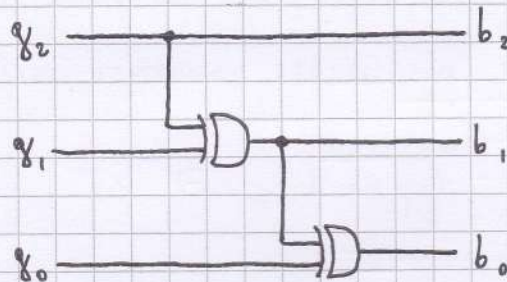
| $q_2 \backslash q_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| $q_0$                | 0  | 0  | 1  | 0  |
|                      | 1  | 0  | 1  | 0  |

$$b_1 = \bar{q}_2 q_1 + q_2 \bar{q}_1 = q_2 \oplus q_1$$

| $q_2 \backslash q_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| $q_0$                | 0  | 0  | 0  | 1  |
|                      | 1  | 0  | 0  | 1  |

$$b_2 = q_2$$

E DUNQUE IL CIRCUITO RICHIEDERÀ UN TEMPO DI ELABORAZIONE MAGGIORE RISPETTO A QUELLO PER LA CONVERSIONE ORROSTA, AVREMO:



CI TROVIAMO QUINDI DI FRONTE AD UNA SORTA DI CASCATA DI PORTE XOR, CHE DEVONO ASPETTARE IL RISULTATO DELLA PRECEDENTE.



# I Moltiplicatori

Vittorio Alzetta

VOLLIAMO, IN QUESTO CAPITOLO, COSTRUIRE UN CIRCUITO EFFICIENTE IN GRADO DI EFFETTUARE LA MOLTIPLICAZIONE TRA DUE NUMERI IN BINARIO.

ANBITUTTO CAPIAMO COME SI FA LA MOLTIPLICAZIONE TRA NUMERI IN BINARIO:

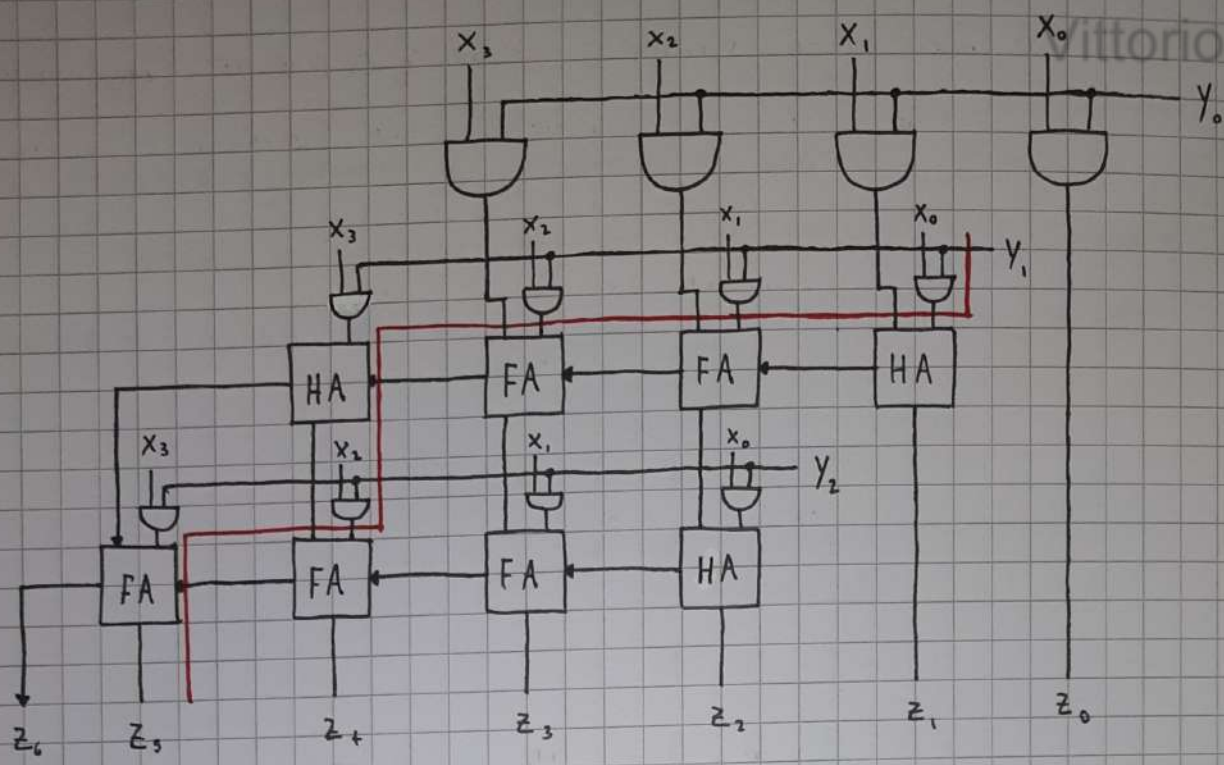
$$\begin{array}{r} 1011 \times 1001 \\ \hline 1011 \\ 0000 \\ 0000 \\ 1011 \\ \hline 110001 \end{array}$$

DA QUESTO ESEMPIO DOVREBBE APPARE CHIARO CHE LA MOLTIPLICAZIONE TRA 2 BIT CORRISPONDE ALL'AND TRA 1 2 BIT:

| A | B | AxB |
|---|---|-----|
| 0 | 0 | 0   |
| 0 | 1 | 0   |
| 1 | 0 | 0   |
| 1 | 1 | 1   |

IN UN PRIMO MOMENTO POTREMMO PENSARE DI UNIRE IN VERTICALE PIÙ LIVELLI DI ADDER, IN MODO DA OTTENERE UN CIRCUITO MOLTO SEMPLICE A CUI DAREMO IL NOME DI **ARRAY MULTIPLIER**:





IN QUESTO SCHEMA LE FRECCE CONTRASSEGNAVO IL PERCORSO DEI BIT DI CARRY.

STABILITO CHE STIAMO MOLTIPLICANDO TRA LORO 2 NUMERI GENERICI:

$$X_0, X_1, \dots, X_n \quad \text{E} \quad Y_0, Y_1, \dots, Y_m$$

INDIVIDUIAMO IL PERCORSO CRITICO DEL MOLTIPLICATORE (IN ROSSO) E FACCIAMO UN' ANALISI STATICA DEL TEMPO DI PROPAGAZIONE:

$$T_{MUL} = T_{AND} + (n-1) T_{CARRY} + (m-2)(T_{SUM} + T_{CARRY}) + T_{SUM}$$

INFATTI, IN UN PRIMO MOMENTO ATTRAVERSA (IL SEGNALE) UNA SERIE DI PORTE AND IN PARALLELO.

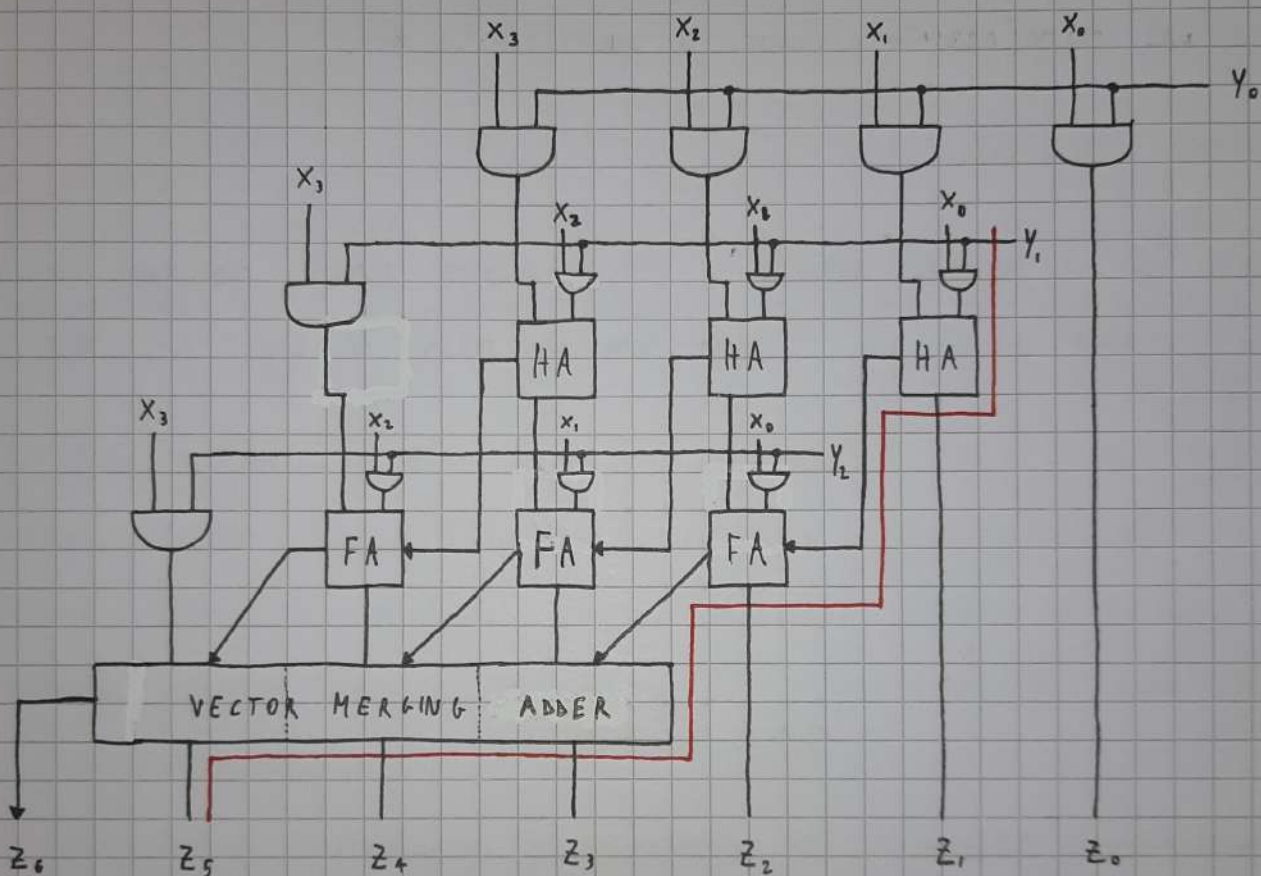
POI IL CARRY SI PROPAGA ATTRAVERSO  $n-1$  ADDER (CI SONO 3 FRECCE) E INFINE DOVRÁ PASSARE (SEMPRE IL SEGNALE) ATTRAVERSO  $m-2$  "BLOCCHI" SOMMA + PROPAGAZIONE CARRY ( $m-2$  PERCHÉ PER  $Y_0$  BASTANO DELLE PORTE AND E PER  $Y_{m-1}$  ABBIAMO L'ULTIMO SCALINO).

POSSIAMO RISCRIVERE  $T_{MUL}$  COSÌ:

$$T_{MUL} = T_{AND} + (n+m-3) T_{CARRY} + (m-1) T_{SUM}$$



OSSERVIAMO CHE CON UNA PICCOLA MODIFICA AL PERCORSO SEGUITO DAL CARRY SI POTREBBE COSTRUIRE UN CIRCUITO PIÙ EFFICIENTE SENZA L'ULTIMA DIAGONALE DI ADDER (DI FATTO CI BASTA CHE IL CARRY, O NPORTO, VENGA PASSATO ALLA COLONNA SUCCESSIVA, NON CI IMPORTA DOVE IN TERMINI DI POSIZIONE VERTICALE). INTRODUCIAMO QUINDI IL **CARRY-SAVE MULTIPLIER**:



OSSERVIAMO CHE PROPAGANDO IL CARRY IN DIAGONALE ~~UNQUE~~ GLI ALLE ADDER NELLA DIAGONALE FINALE DEL PRECEDENTE SCHEMA NON AVREBBERO PIÙ SENSO (RICEVEREBBERO O PERÙ UN SOLO VALORE IN INGRESSO, NON AVREBBERO NULLA DA SOMMARE), QUINDI LI MUOVIAMO.

FACCIO INOLTRE NOTARE CHE IL VECTOR MERGING ADDER NON RAPPRESENTA ALTRO CHE UN ADDER VELOCE (QUANTO CI PARE) CHE EFFETTUA LE ULTIME SOMME (NON POSSIAMO METTERLO OVUNQUE PERCHÉ AVREMMO UN CIRCUITO TROPPO VASTO).



Vittorio Alzetta

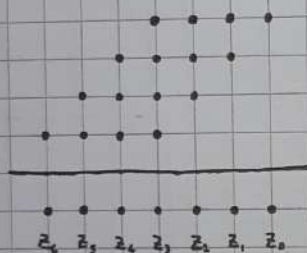
ANALIZZANDO DI NUOVO IL TEMPO DI ELABORAZIONE TROVIAMO CHE IL PERCORSO CRITICO (IN ROSSO) DIPENDE DALLA SEGUENTE RELAZIONE:

$$T_{MUL} = T_{AND} + (m-1)T_{CARRY} + T_{ADDER}$$

IN SOSTANZA SE  $m \approx n$  QUESTO NUOVO MOLTIPLICATORE SARÀ IL DOPIPIO PIÙ VELOCE DEL PRIMO (ABBIAMO ELIMINATO LA DIPENDENZA DA  $m$ ).

### RIDUZIONE A 2 LIVELLI

PROVIAMO ORA A CAMBIARE APPROCCIO, NEL SEGUENTE SCHEMA INDICHERÒ CON I PALLINI LE PORTE LOGICHE AND (NELLA PRIMA RIGA AVREMO IN ORDINE  $x_3y_0, x_2y_0, x_1y_0, x_0y_0$ ):



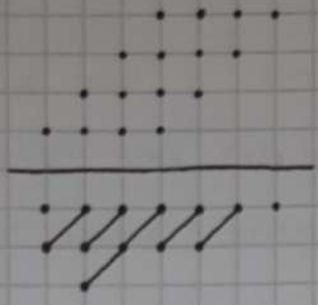
È INDOUBBIAMENTE PIÙ SEMPLICE DA COMPRENDERE E RAPPRESENTARE.

OSSERVIAMO PERÒ CHE SOMMANDO TRA LORO 1, 2, 4, 8 BIT SI RENDERÀ NECESSARIO PROPAGARE UN CERTO CARRY ALLE COLONNE SUCCESSIVE, IN PARTICOLARE AVREMO CHE:

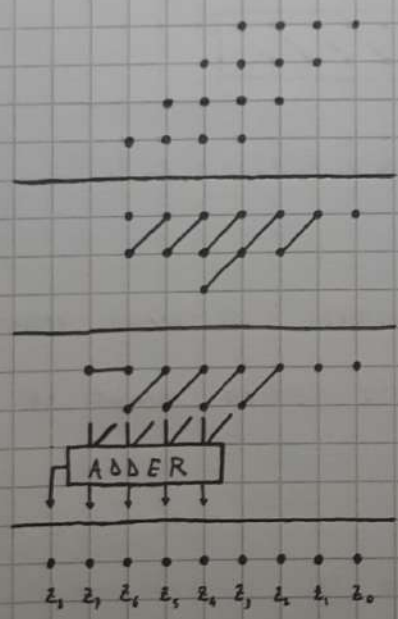
- CON 1 BIT NON CI SARÀ RIPORTO, O È 0 O È 1
- CON 2 BIT SI HA RIPORTO QUANDO DOVRESSERO ESSERE ENTRAMBI 1
- CON 4 BIT SI PUÒ AVERE RIPORTO ANCHE ALLA COLONNA CHE STA DUE POSIZIONI RIVIN LÁ, DOBBIAMO PENSARLA COME FORSE UNA SOMMA IN DECIMALE, SE SOMMANDO LE UNITÀ OTTENIAMO PIÙ DI 100 IL RIPORTO ANDRÀ 2 COLONNE PIÙ IN LÁ (ALLE CENTINAIA).
- CON 8 BIT CI SPINGEREMO AD AVERE UN POSSIBILE RIPORTO FINO A 3 COLONNE PIÙ IN LÁ.



NEL SEGUENTE SCHEMA, DOPO LA LINEA ORIZZONTALE, SEGNERÒ NEL PRIMO LIVELLO I  
BIT DI SOMMA DELLA COLONNA SOPRASTANTE MENTRE COLLEGHERÒ AD ESSI I BIT DI  
RIPORTO PRELEVATI DALLA STESSA COLONNA:



OSSERVIAMO CHE NEL CASO IN CUI DOVRESSIMO CALCOLARE IL BIT DI SOMMA E I  
2 BIT DI CARRY DELLA 4<sup>a</sup> COLONNA DOVREMMO IMPLEMENTARE UN CIRCUITO AD HOC,  
MENTRE SE SI TRATTASSE DI SOMMARE 2 O 3 BIT CON I RELATIVI BIT  
DI RIPORTO POTREMMO UTILIZARE I GIÀ STUDIATI HALF E FULL ADDER  
(PRENDONO IN INPUT 2 O 3 BIT IN INGRESSO E NE RESTITUISCONO 2).  
DECIDIAMO QUINDI DI PROCEDERE CON LA ADDIZIONE FINO A QUANDO NON  
OTTENIAMO COLONNE CON AL MASSIMO 2 BIT, POI FAREMO LA SOMMA FINALE  
CON L'OPPORTUNO ADDER:



QUESTA STRUTTURA PRESENTA DELLE INCONVENIENZE NELLA MANIPOLAZIONE DELLE COLONNE,



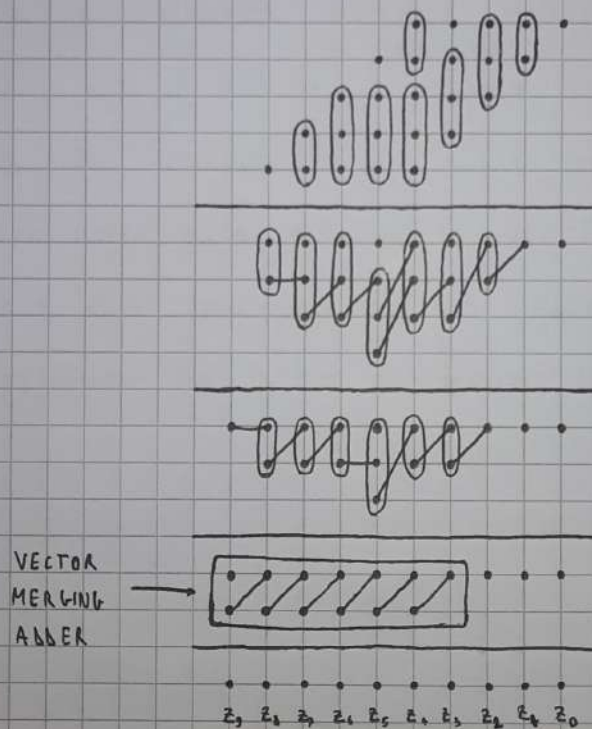
PER QUESTO MOTIVO DECIDIAMO DI ADOTTARE UN ALGORITMO, DETTO **WALLACE - TREE**,  
PER ACCOPPIARE I BIT DELLE COLONNE.

CI RIFAREMO AI FULL ADDER E GLI HALF ADDER PER ANDARE A 2 LIVELLI LE  
COLONNE.

OSSERVATO CHE IN OGNI FULL ADDER CI SONO 3 INGRESSI E 2 USCITE CI

ASPETTIAMO UNA RIDUZIONE NELL'ALTEZZA DELLE COLONNE DI UN FATTORE 1,5.

NEL SEGUENTE SCHEMA USEREMO SEMPRE, QUANDO POTREMO, DEI FULL ADDER, IN  
CASO CONTRARIO DEGLI HALF ADDER E ANCORA IN ALTERNATIVA PROPORCHEREMO  
SEMPLICEMENTE IL BIT SINGOLO:



IL TEMPO DI ELABORAZIONE SARÁ LOGARITMICO, SI VIENE INFATTI A CREARE UNA STRUTTURA  
A ALBERO. LO SVANTAGGIO DI QUESTO METODO RIMANE LA SUA IRREGOLARITÀ NELL'ALTEZZA  
DELLE COLONNE.

É POSSIBILE USARE UN ALTRO CRITERIO PER L'ACCOPPIAMENTO DEI BIT DELLE COLONNE,  
IL **DADD - TREE**.

ESSO PREVEDE DI UTILIZZARE INIZIALMENTE DEI FULL ADDER E POI, DI USARE



DELLI HALF ADDER SOLO SE NELLA COLONNA CHE SI VIENE A CREARE SOTTO AVREMO  
UN NUMERO DI BIT MULTIPLO DI 3, IN CASO CONTRARIO PREVEDE DI PROPALARE  
SEMPLICEMENTE QUESTI BIT ALLO STADIO SUCCESSIVO. (NELL'ULTIMO STADIO USO ANCHE GLI HA)  
SENZA RESTRIZIONI

SOLITAMENTE IL DADDA-TREE RICHIEDE MENO LIVELLI DI RIDUZIONE MA HA PIÙ  
BIT NEL VECTOR MERGING ADDER, DUNQUE NON È MOLTO APPARENTE AL WALLACE-TREE.

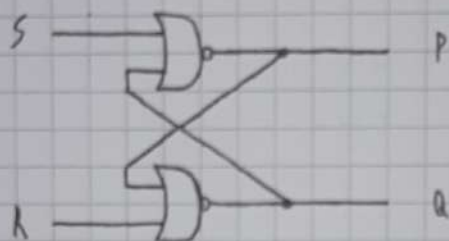


## I CIRCUITI SEQUENZIALI

TUTTO QUELLO CHE ABBIAMO VISTO FINO AD OGGI SI BASA SULLA LOGICA DEI CIRCUITI COMBINATORI, OVVERO CIRCUITI IL CUI RISULTATO DIPENDE SOLTANTO DALLA COMBINAZIONE DEI VALORI IN INGRESSO.

IN QUESTO CAPITOLO INTRODUCIAMO I CIRCUITI SEQUENZIALI, CHE DIPENDERANNO ANCHE DALLA SEQUENZA DEGLI INGRESSI CHE ARRIVANO NEL TEMPO.

ABBIAMO QUINDI BISOGNO DI CREARE UN OGGETTO CHE MANTENGA L'INFORMAZIONE NEL TEMPO, E BUNAVE CREIAMO IL SEGUENTE CIRCUITO:



RICORDIAMO CHE DATA UNA PORTA NOR, SE UNO DEGLI INGRESSI È 1 IL RISULTATO SARÀ 0 MENTRE SE UNO DEGLI INGRESSI È 0 ALLORA IL RISULTATO SARÀ L'ALTRO INGRESSO NEGATO.

PROVIAMO A COSTRUIRE LA TABELLA DI VERITÀ DI QUESTO CIRCUITO, TENENDO CONTO CHE:

$$\overline{S + Q} = P, \quad \overline{R + P} = Q$$

LA TABELLA SARÀ:

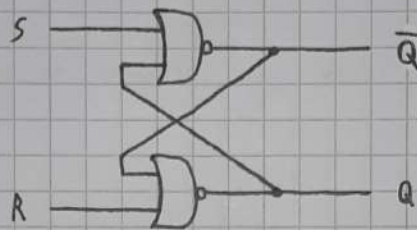
| S | R | P         | Q         |       |
|---|---|-----------|-----------|-------|
| 0 | 0 | $\bar{Q}$ | $\bar{P}$ |       |
| 0 | 1 | $\bar{Q}$ | 0         | (1 0) |
| 1 | 0 | 0         | $\bar{P}$ | (0 1) |
| 1 | 1 | 0         | 0         |       |



NOTIAMO CHE NELLE PRIME TRE RIGHE IL RISULTATO DI  $Q$  E  $P$  È SEMPRE OPPOSTO, MENTRE SOLO NELL'ULTIMA SI HA LO STESSO VALORE PER  $P$  E  $Q$ , PER QUESTO MOTIVO NON CONSIDEREREMO PIÙ IL CASO  $S = R = 1$ .

IL NOSTRO OBIETTO SARÀ QUINDI STABILE E  $P = \bar{Q}$ , ABBIAMO ANINDI LA POSSIBILITÀ DI SALVARCI DENTRO UN VALORE E POI USARLO PIÙ AVANTI NEL TEMPO.

È PER QUESTO MOTIVO CHE CHIAMIAMO QUESTO CIRCUITO **LATCH** (QUALCOSA CHE CATTURA) **ASINCRONO** DI TIPO **SR**. CAPIAMO MEGLIO PERCHÉ LO CHIAMIAMO LATCH DI TIPO **SR**:

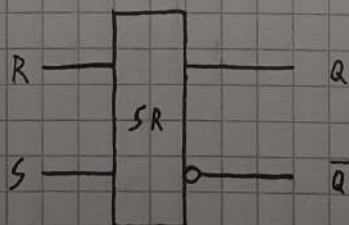


POICHÉ CI PONIAMO SEMPRE DIVERSA DA  $S = R = 1$  POSSIAMO SCRIVERE  $\bar{Q}$  AL POSTO DI  $P$ , INOLTRE CONSIDEREREMO  $Q$  COME VALORE DI USCITA (CI INTERESSEREMO A QUELLO).

OVVERE QUANDO  $S = 1$  E  $R = 0$  STIAMO PONENDO  $Q = 1$ , IN UN CERTO SENSO LO STIAMO ATTIVANDO (UN OPERAZIONE DI SET).

QUANDO INVECE  $R = 1$  E  $S = 0$  STIAMO PONENDO  $Q = 0$ , OVVERE ABBIAMO FORNITO UNO 0 NEL CIRCUITO (UN OPERAZIONE DI RESET).

SUPPONIAMO ORA CHE NEL SEGUENTE CIRCUITO  $Q$  ABBAIA GIÀ UN VALORE (0 o 1) E PROVIAMO A SCRIVERE LA TABELLA DI VERITÀ DEL VALORE CHE ASSUMERÀ POI  $Q$ :





ORA CREIAMO LA TABELLA DI VERITÀ, DOV'  $Q'$  SARÀ IL NUOVO VALORE DI  $Q$   
DATI  $S, R$  E  $Q$  (VALORE DI PARTENZA DI  $Q$ ):

|                                                                       | $S$ | $R$ | $Q$ | $Q'$ |
|-----------------------------------------------------------------------|-----|-----|-----|------|
| QUI STO PRESERVANDO<br>IL VALORE DI PARTENZA<br>DELLA VARIABILE $Q$ . | 0   | 0   | 0   | 0    |
|                                                                       | 0   | 0   | 1   | 1    |
| QUI LO STO RESETTANDO<br>A 0.                                         | 0   | 1   | 0   | 0    |
|                                                                       | 0   | 1   | 1   | 0    |
| QUI LO STO SETTANDO<br>A 1.                                           | 1   | 0   | 0   | 1    |
|                                                                       | 1   | 0   | 1   | 1    |

LA PRECEDENTE TABELLA PUÒ MOLTO SEMPLICEMENTE ESSERE RASSUNTA NELLA SEGUENTE:

| $S$ | $R$ | $Q'$ |
|-----|-----|------|
| 0   | 0   | $Q$  |
| 0   | 1   | 0    |
| 1   | 0   | 1    |

POSSIAMO ANCHE SCRIVERE LA SEGUENTE VARIANTE, CHE PRENDE IL NOME DI **TABELLA DELLE ECCITAZIONI** (È FACILE COSTRUIRELA SE SI GUARDA ALLA INIZIALE TABELLA):

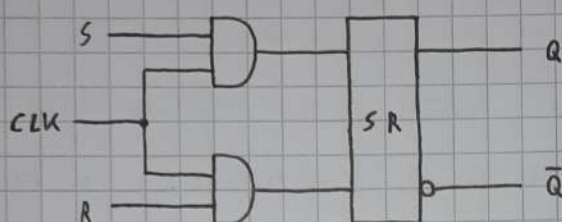
| $Q$ | $Q'$ | $S$ | $R$ |
|-----|------|-----|-----|
| 0   | 0    | 0   | —   |
| 0   | 1    | 1   | 0   |
| 1   | 0    | 0   | 1   |
| 1   | 1    | —   | 0   |

CI DICE QUALI VALORI DI  $S$  E  $R$  BASTANO PER OTTENERE UNA DETERMINATA TRANSIZIONE.

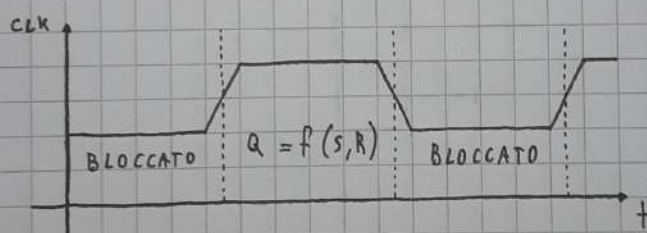


## IL CLOCK

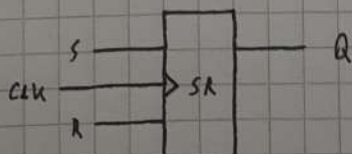
ABBIAMO DETTO CHE VOGLIAMO ESSERE IN LAUDO DI DISTINGUERE SEQUENZE DI INGRESSI DI VERBA NEL TEMPO, PERTANTO NOBBIAMO BISOLNO DI QUALCOSA CHE BATTA IL TEMPO. VOGLIAMO RVINDI IMPLEMENTARE UN CIRCUITO CHE, PRESO IN INGRESSO IL VALORE CLK (CLOCK), SI COMPORNI COME UN LATCH ASINCRONO SR QUANDO CLK = 1 E SI CONSERVI (COME SE  $S + R = 0$ ) QUANDO CLK = 0. QUESTO É FACILE DA OTTENERE, BASTA AGGIUNGERE DELLE PORTE AND:



IL RISULTATO SARÁ CHE NEL TEMPO AVREMO I SEGUENTI COMPORTAMENTI:



VISTO CHE QUESTO NUOVO CIRCUITO HA UN SEGNALE CHE BATTE IL TEMPO DECIDIAMO DI CHIAMARLO **LATCH SINCRONO** DI TIPO SR. LO RAPPRESENTEREMO NEL SEGUENTE MODO:



DOVE IL TRIANGOLINO INDICA CHE CLK NON É UN SEGNALE DI CONTROLLO MA DI SINCRONIZZAZIONE.



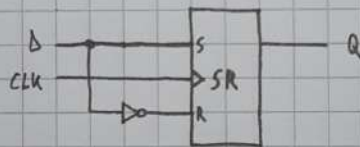
INTRODUCIAMO ORA UN NUOVO TIPO DI LATCH SINCRONO, ESSO PRENDERÀ SOLO UN VALORE IN INGRESSO ~~IN~~ E L'USCITA SARÀ UGUALE ALL'INGRESSO QUANDO IL CLOCK AVrà VALORE 1, NON SEGUIRÀ IL VALORE DI INGRESSO QUANDO  $CLK = 0$ . LO CHIAMEREMO **LATCH SINCRONO DI TIPO D**, LA SUA TABELLA DI TRANSIZIONE E IL SUO SCHEMA SONO:



| D | Q' |
|---|----|
| 0 | 0  |
| 1 | 1  |

IN SOSTANZA QUANDO  $CLK = 0$  QUESTO LATCH MOSTRA IN USCITA IL POTENZIALMENTE VECCHIO VALORE DI  $D$  (MENTRE  $CLK = 0$   $D$  PUÒ CAMBIARE), È PER QUESTO CHE USIAMO LA  $D$  DI DELAYED.

È FACILISSIMO DA IMPLEMENTARE CON IL LATCH SINCRONO DI TIPO SR:



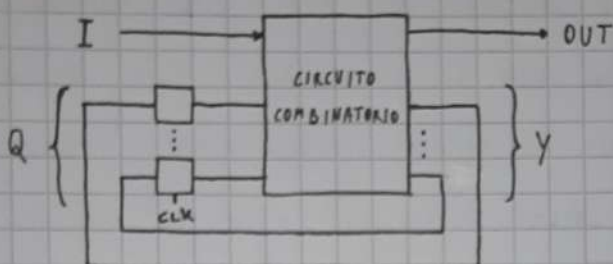
SE  $D = 1$  IL LATCH SI SETTA A 1, SE  $D = 0$  IL LATCH SI RESETTA A 0.



## IL FLIP-FLOP

IL MOTIVO PER IL QUALE ABBIAMO INTRODOTTO DUE OGGETTI IN GRADO DI MEMORIZZARE AL LORO INTERNO DELLE INFORMAZIONI È QUELLO DI POTER COSTRUIRE DEI CIRCUITI CHE FACCIANO LE OPERAZIONI UN PASSO ALLA VOLTA, ALTERNANDO UNA FASE DI ELABORAZIONE AL UNA DI LETTURA.

UNA STRUTTURA DI ESEMPIO PUÒ ESSERE LA SEGUENTE:



VOLLIAMO QUINDI CHE IN QUESTO CIRCUITO GLI OUTPUT DIVENTANO DALLI INGRESSI E GLI OUTPUT DIVENTINO I NUOVI INGRESSI:

$$\begin{cases} y(k) = f(Q(k), I) \\ Q(k+1) = y(k) \end{cases}$$

DOVE  $y(k)$  È IL VALORE DI  $y$  ALL'INTERVALLO  $k$ -ESIMO E  $Q(k)$  È IL VALORE DEGLI INPUT NELL'INTERVALLO  $k$ -ESIMO.

OSSERVIAMO CHE DATI IN PARTENZA DEI VALORI AD  $I$  E  $Q$  QUANDO  $CLK = 0$  QUESTI VALORI VERRANNO MEMORIZZATI E IL CIRCUITO COMBINATORIO AVRÀ IL TEMPO DI ELABORARE  $f(Q(k), I)$  E DI FAR ARRIVARE ALL'INGRESSO DEI LATCH IL NUOVO VALORE (IL  $CLK$  È ANCORA 0 QUINDI I LATCH SONO ANCORA BLOCCATI).

APPENA  $CLK$  DIVENTA 1 PASSANO I NUOVI VALORI ATTRAVERSO IL LATCH, CHE ORA È DIVENTATO TRASPARENTE E VENGONO SUBITO ELABORATI DAL CIRCUITO COMBINATORIO.

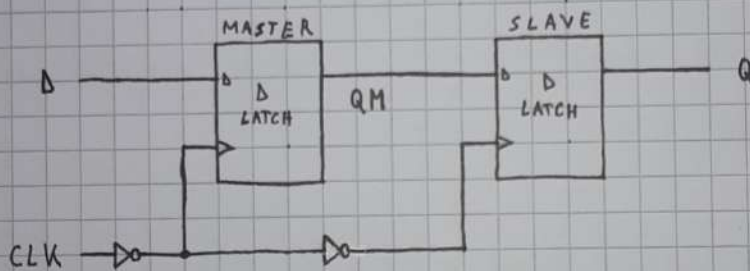


IL PROBLEMA È CHE IL SEGNALE DI CLOCK RIMANE A 1 PER UN PÓ E QUESTO PROVOCA UN ~~SALENDO~~ LOOP COMBINATORIO DOVE I NUOVI VALORI VENGONO ELABORATI DAL CIRCUITO COMBINATORIO E POI SONO IMMEDIATAMENTE PASSATI AI LATCH (CHE SONO ANCORA TRASPARENTI) PER ESSERE ELABORATI.

NOI VOLLIAMO CHE QUANDO CLK DIVENTA 1 I NOSTRI DATI VENGANO ELABORATI SOLTANTO UNA VOLTA PER UN INTERVALLO, QUINDI DOBBIAMO INTRODURRE UNA MODIFICA.

FINO AD ORA I LATCH CHE ABBIAMO VISTO AVEVANO DEGLI INTERI INTERVALLI DI CAMPIONAMENTO (CHE GENERANO DEI LOOP), IL NOSTRO OBIETTIVO È QUELLO DI COSTRUIRE UN CIRCUITO IN GRADO DI MEMORIZZARE I DATI IN UN SINGOLO ISTANTE PER POI PASSARE I RISULTATI ED ESSERE IMMEDIATAMENTE "IBERNATO" (O BLOCCATO).

QUESTO COMPONENTE SI CHIAMA **FLIP-FLOP** E HA LA SEGUENTE STRUTTURA:



CERCHERÒ ORA, PER QUANTO MI È POSSIBILE, DI SPIEGARE COME MAI QUESTO SEMPLICE CIRCUITO PERMETTE IL PASSAGGIO DEL SEGNALE D SOLO IN UN ISTANTE E NON IN UN INTERVALLO.

SUPPONIAMO CHE CLK SIA 0 E CHE D SIA UN SEGNALE CHE VARIA IN CONTINUAZIONE (AIUTA A IMMAGINARSELO IN FUNZIONE).

SE  $CLK = 0$  ALLORA IL MASTER È TRASPARENTE E QUINDI  $Q_M$  VARIA IN CONTINUAZIONE, LO SLAVE INVECE È BLOCCATO E QUINDI  $Q$  NON STA VARIANDO.

APPENA  $CLK = 1$  IL MASTER SI BLOCCA E QUINDI  $Q_M$  SI RISSA ALL'ULTIMO VALORE CHE AVEVA ASSUNTO D PRIMA DELLA TRANSIZIONE DEL CLOCK, LO SLAVE INVECE VIENE APERTO (DOPPIA NEGAZIONE DEL CLK) E TRASMETTE A Q IL VALORE DI  $Q_M$  (CIOÈ L'ULTIMO ASSUNTO DA D PRIMA DELLA TRANSIZIONE).



COME POSSIAMO OSSERVARE, NONOSTANTE  $CLK = 1$  NON C'È NESSUN LOOP, NON C'È NESSUN PASSAGGIO CONTINUO DI VALORI, È SEMPLICEMENTE STATO FISSATO L'ULTIMO VALORE DI  $D$  ALLA FINE DELLA TRANSIZIONE.

APPENA  $CLK = 0$  (TORNA A 0) SI RITORNA ALLA CONDIZIONE INIZIALE, DOVE  $Q$  NON VARIA E  $QM$  VARIA COME E ASSUME LO STESSO VALORE DI  $D$ .

CHIAMEREMO QUESTO PARTICOLARE FLIP-FLOP **D-FLIP-FLOP POSITIVE-EDGE TRIGGERED**, IN QUANTO SI ATTIVA QUANDO LA TRANSIZIONE DEL CLOCK È DA 0 A 1. LO INDIRICHEREMO CON:

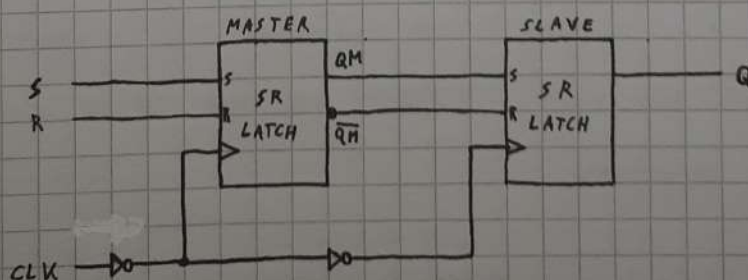


IL SIMBOLO ALL'INTERNO INDICA CHE IL VALORE DI  $Q$  È CAMPIONATO IN CORRISPONDENZA DI UN FRONTE DI SALITA.

PER AVERE UN FLIP-FLOP CHE CAMPIONA NEL FRONTE DI DISCESA BASTA CAMBIARE LA POSIZIONE DEL NOT NEL CIRCUITO.

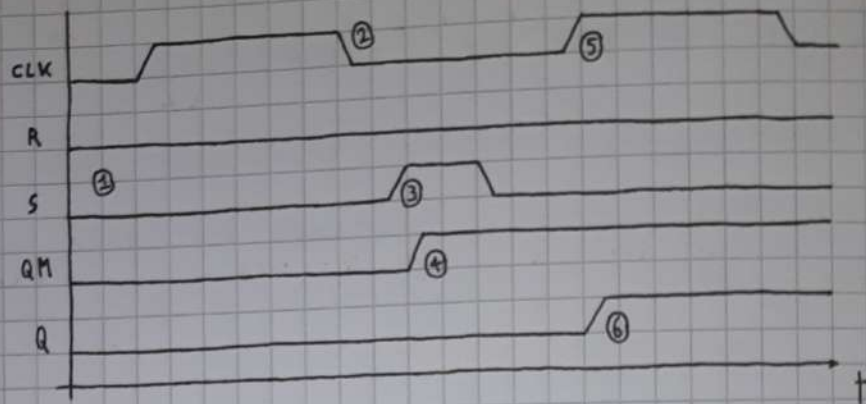
CI POTREMMO CHIEDERE MA SE SIA POSSIBILE CREARE UN FLIP-FLOP CON DEI LATCH SINCRONI DI TIPO SR.

IL CIRCUITO SAREBBE QVINDÌ:





ORA PROVIAMO A RAPPRESENTARE GRAFICAMENTE UN POSSIBILE ANDAMENTO DEI VALORI NEL TEMPO:



ORA, PROCEDENDO PER PUNTI DESCRIVENDO IL PROBLEMA IN CUI INCORRIAMO USANDO DEI FLIP-FLOP DI TIPO SR:

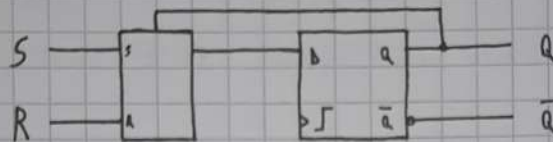
- 1 - INIZIALMENTE  $R$  E  $S$  SONO A 0, QUINDI IL VALORE DI  $Q_M$  NON VARIA E RIMANE A 0, IL MASTER ORA È CHIUSO QUINDI ANCHE  $Q$  NON VARIA.
- 2 - IL SEGNALE DI CLOCK PASSA A 0, QUINDI SI APRE IL MASTER MENTRE LO SLAVE RIMANE CHIUSO (ORA  $Q_M$  PUÒ VARARE).
- 3 - IN QUESTO PUNTO ABBIAMO UN GLITCH E  $S$  DIVENTA 1 PER UN PICCOLO INTERVALLO DI TEMPO (TORNA A 0 PRIMA DELLA TRANSIZIONE DEL CLOCK).
- 4 - A CAUSA DEL GLITCH IL VALORE DI  $Q_M$  VARIA (DALLA TABELLA ABBIAMO CHE CON  $R=0$  E  $S=1$  IL VALORE DI  $Q_M$  DEVE ESSERE 1), NONOSTANTE POI  $S$  TORNA A 0  $Q_M$  NON VARIA, INRATTI NEL CASO IN CUI  $S=R=0$  IL VALORE DI  $Q_M$  VIENE MEMORIZZATO.
- 5 - ORA CHE  $CLK=1$  IL MASTER SI CHIUSO E LO SLAVE SI APRE FACENDO COMUNICARE  $Q_M$  E  $Q$ .
- 6 - ADUNQUE LO SLAVE SI VEDrà IN INGRESSO  $S=Q_M=1$  E  $R=\overline{Q_M}=0$ ,



QUINDI NON PUÒ FAR ALTRO CHE FAR PASSARE  $Q$  DA 0 A 1.

NOTIAMO QUINDI CHE NONOSTANTE ALL'ISTANTE DI CAMPIONAMENTO  $S$  E  $R$  FOSSERO ENTRAMBI 0 (IL CLITCH AVVIENE INTA) E QUINDI, A RIFER DI LOWCA, NON AVREBBE DOVUTO CAMBIARE IL VALORE DI  $Q$ , CAMBIANDO IL VALORE DI  $Q$ . QUESTO È UN ERRORE, ED È PER QUESTO CHE NON È COSÌ SEMPLICE COSTRUIRE UN FLIP-FLOP CON DUE LATCH DI TIPO SR (SINCROMI).

QUELLO CHE VORREMMO OTTENERE È UN **FLIP-FLOP SR**, MA SAPPIAMO COME COSTRUIRE SOLO IL FF-D, QUINDI DOVREMO FARE QUALCOSA DEL TIPO:



IL MOTIVO PER IL NOME COLLEGHIAMO  $Q$  ALL'INPUT "DISTURBO" È PERCHÉ, SE ABBIAMO INTENZIONE DI COSTRUIRE UN FF-SR, ABBIAMO BISOGNO DI SAPERE IL VALORE DI  $Q$  QUANDO  $S=R=0$  (FASE DI MEMORIZZAZIONE, DOVE NON DEVE VARARE L'OUTPUT) PER EVITARE CHE  $Q$  CAMBI (ALTRIMENTI AL PRIMO DI CLOCK  $Q$  VARIA ANCHE  $Q$ ).

QUINDI PRENDEREMO ANCHE  $Q$  COME INPUT.

SCRIVIAMO QUINDI LA MAPPA DI KARNAUGH PER CAPIRE CHE CIRCUITO IMPLEMENTARE VOLENDO OTTENERE I SEGUENTI RISULTATI:

$$S=R=0 \Rightarrow Q' = Q$$

$$S=1, R=0 \Rightarrow Q' = 1$$

$$S=0, R=1 \Rightarrow Q' = 0$$

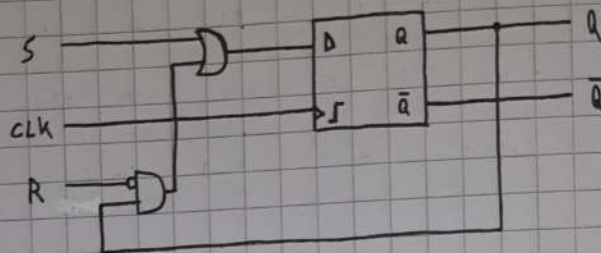
|   |   | SR |    |    |    |
|---|---|----|----|----|----|
|   |   | 00 | 01 | 11 | 10 |
| Q | 0 | 0  | 0  | -  | 1  |
|   | 1 | 1  | 0  | -  | 1  |

DA CUI SI RICAVA SUBITO L'ESPRESSIONE PER  $D$ :

$$D = S + Q\bar{R}$$



POSSIAMO QUINDI ORA COSTRUIRE IL FF-SR:



ARRIVATI A QUESTO PUNTO POTREMMO PENSARE DI DARE FINALMENTE UN SIGNIFICATO ALLA COMBINAZIONE  $S=R=1$ .

DECIDIAMO CHE IN TAL CASO  $Q' = \bar{Q}$ , PER MANTENERE UNA CERTA SIMMETRIA. PER COSTRUIROLO SCRIVEREMO ANCHE LE TABELLE DELLE TRANSIZIONI E DELLE ECCITAZIONI.

| J | K | $Q(k+1)$     |
|---|---|--------------|
| 0 | 0 | $Q(k)$       |
| 0 | 1 | 0            |
| 1 | 0 | 1            |
| 1 | 1 | $\bar{Q}(k)$ |

| Q | Q' | S | R |
|---|----|---|---|
| 0 | 0  | 0 | - |
| 0 | 1  | 1 | 0 |
| 1 | 0  | 0 | 1 |
| 1 | 1  | - | 0 |

SI NOTI CHE NELLA PRIMA HO USATO J E K IN QUANTO QUELLO CHE STIAMO COSTRUIENDO PRENDERÀ IL NOME DI **FLIP-FLOP JK**.

COSTRUIAMO QUINDI LE K-MAP PER CAPIRE COSA METTERE TRA J, K E S, R:

| JK | 00 | 01 | 11 | 10 |
|----|----|----|----|----|
| Q  | 0  | 0  | 1  | 1  |
| 1  | -  | 0  | 0  | -  |

$$S = J\bar{Q}$$

| JK | 00 | 01 | 11 | 10 |
|----|----|----|----|----|
| Q  | 0  | -  | 0  | 0  |
| 1  | 0  | 1  | 1  | 0  |

$$R = KQ$$

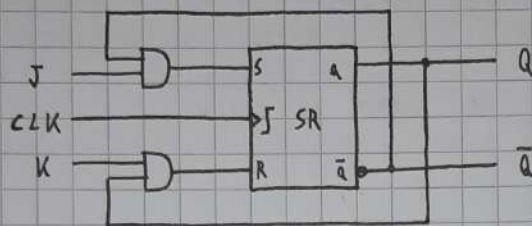
SPIEGHERÒ BREVEMENTE COME HO RICAVATO I PRIMI 2 VALORI NELLA PRIMA TABELLA



(È ANALOGO PER LE ALTRE CELLE).

MI SONO CHIESTO COSA SUCCEDDE QUANDO  $J = K = 0$ , HO LETTO SULLA TABELLA 1 CHE  $Q(k+1) = Q(k)$ , QUINDI SONO ANDATO SULLA TABELLA 2 E HO CERCATO I CASI IN CUI  $Q' = Q$ , MA SAPENDO CHE  $Q = 0$  MI RIDUCCO AD UN SOLO CASO (IL PRIMO), E LÌ  $S = 0$ ,  $R = -$  (A NOI PERÒ INTERESSA SOLO  $S$  IN QUESTA TABELLA).

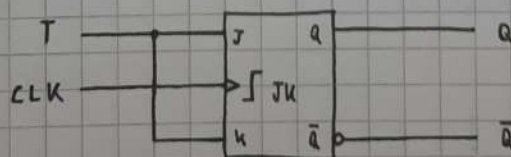
IL CIRCUITO SARÀ PERTANTO UNA COSA DEL GENERE:



ORA CREIAMO UN ALTRO TIPO DI FLIP-FLOP, VOGLIAMO CHE OR IN BASE AL VALORE IN INGRESSO DECIDA SE INVERTIRE O MENO IL VALORE IN USCITA SECONDO LA SEGUENTE TABELLA:

| T | $Q(k+1)$          |
|---|-------------------|
| 0 | $Q(k)$            |
| 1 | $\overline{Q(k)}$ |

LO CHIAMEREMO **FLIP-FLOP T** E IMPLEMENTARLO CON UN FF-JK È BANALE:



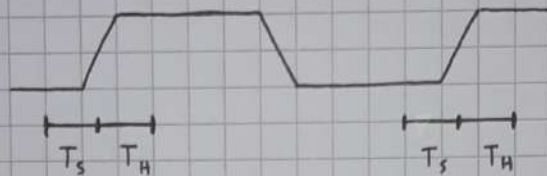
QUESTO SARÀ L'ELEMENTO BASE CON IL QUALE VERRANNO COSTRUITI I CONTATORI.



## METASTABILITÀ

ABBIAMO VISTO DIVERSI FLIP-FLOP E ABBIAMO OSSERVATO COME ESSI, AL CONTRARIO DEI LATCH, CAMPIONANO IN SINGOLI ISTANTI E NON IN INTERVALLI. QUESTO È PARZIALMENTE VELO, INFATTI ANCHE I FLIP-FLOP CAMPIONANO IN INTERVALLI, SEPPUR BREVISSIMI RISPETTO A QUELLI DEI LATCH.

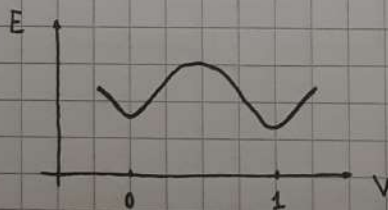
IN QUESTI PICCOLI INTERVALLI L'INGRESSO NON DEVE VARIARE, CHIAMEREMO TEMPO DI **SETUP** L'INTERVALLO IN CUI L'INPUT DEVE RIMANERE STABILE PRIMA DEL FRONTE DI CLOCK, MENTRE DIREMO TEMPO DI **HOLD** PER RIFERIRCI ALL'INTERVALLO IN CUI L'INPUT DEVE RIMANERE STABILE SUCCESSIVO AL FRONTE DI CLOCK.



L'UNIONE DEL TEMPO DI **SETUP** E DEL TEMPO DI **HOLD** VIENE CHIAMATA **FINESTRA DI STABILITÀ** (PER INDICARE CHE LÌ IL SEGNALE NON DEVE VARIARE).

DOBBIAMO ANCHE CONSIDERARE IL FATTO CHE IL CIRCUITO NON DARÀ SUBITO IL RISULTATO (AL FRONTE DI CLOCK) MA LO FARÀ DOPO UN TEMPO **CLOCK TO OUTPUT**.

IL FATTO CHE IL SEGNALE DI CLOCK PASSI DA 0 A 1 VUOLE DIRE CHE È VARIATO IL SUO LIVELLO DI ENERGIA:



AFFINCHÉ SIA STABILE IN 0 E 1 DEVE AVERE DUE MINIMI DI ENERGIA (ALTRIMENTI NON SI AVREBBE EQUILIBRIO STABILE, VEDI FISICA 1).



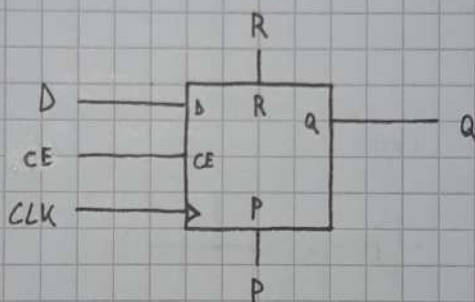
NEL MOMENTO IN CUI VIENE VIOLATA LA FINESTRA DI STABILITÀ CI SI POTREBBE TROVARE IN UNA SITUAZIONE DI EQUILIBRIO INSTABILE Dove LA VARIABILE NON È NÉ 0 NÉ 1, MA SI TROVA IN MEZZO.

QUESTA CONDIZIONE VIENE DETTA DI **METASTABILITÀ** E RENDE IMPOSSIBILE PREVEDERE CHE VALORE ASSUMERÀ LA VARIABILE UNA VOLTA DECAVUTA (DA UNA PARTE O DALL'ALTRA).

## ALTRI INPUT NEI FLIP-FLOP

ANZITUTTO FACCIO OSSERVARE CHE IL FLIP-FLOP PIÙ IMPORTANTE È QUELLO DI TIPO D (POI VIENE QUELLO DI TIPO T).

IN UN FF-D POSSIAMO AVERE I SEGUENTI INGRESSI:



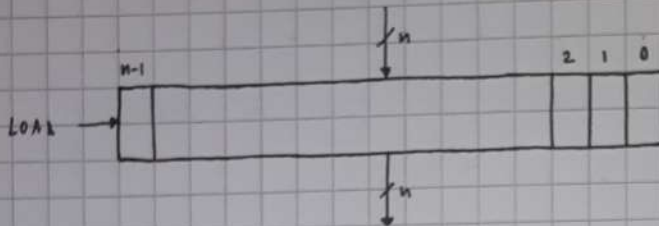
Dove GLI INGRESSI NUOVI IMPLEMENTANO LE SEGUENTI FUNZIONALITÀ:

- R : RESET, PORTA SEMPRE IL VALORE DI Q A 0
- CE : CLOCK ENABLE, QUANDO VALE 1 Q SELVE EVENTUALMENTE D (A SECONDA DEL VALORE DEL CLOCK) ALTIMENTI QUANDO VALE 0 BLOCCA L'USCITA.  
"LOAD" ↗
- P : PRESET, PORTA SEMPRE IL VALORE DI Q A 1

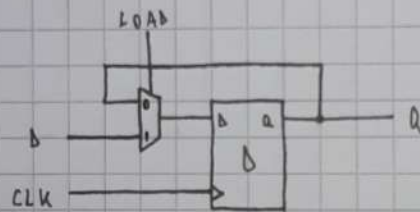


# I REGISTRI

PER ADESSO ABBIAMO VISTO SOLO CIRCUITI IN GRADO DI MEMORIZZARE SINGOLI BIT, MA NOI ABBIAMO BISOGNO DI QUALCOSA CHE MEMORIZZI PAROLE INTERE, PER QUESTO INTRODUCIAMO I REGISTRI, CIOÉ CIRCUITI CHE HANNO AL PROPRIO INTERNO NUMEROSI FF - D :



DOVE GLI ELEMENTI CHE HO NUMERATO DA 0 A  $n-1$  SONO I FF - D, POSSIAMO PENSARE AL SEGNALE DI LOAD (CE) COME :



INMA HO DETTO CHE I FF - T SONO GLI ELEMENTI BASE DEI CONTATORI, EBBENE, ORA CHE ABBIAMO UN OGGETTO IN GRADO DI MEMORIZZARE PIÙ BIT PROVIAMO A COSTRUIRE UNO (DI CONTATORE).

VOLIAMO CHE SIA A 3 BIT, PERTANTO AD OGNI FRONTE DI CLOCK DOVRÀ PASSARE ALLA RIGA SUCCESSIVA DI QUESTA TABELLA:

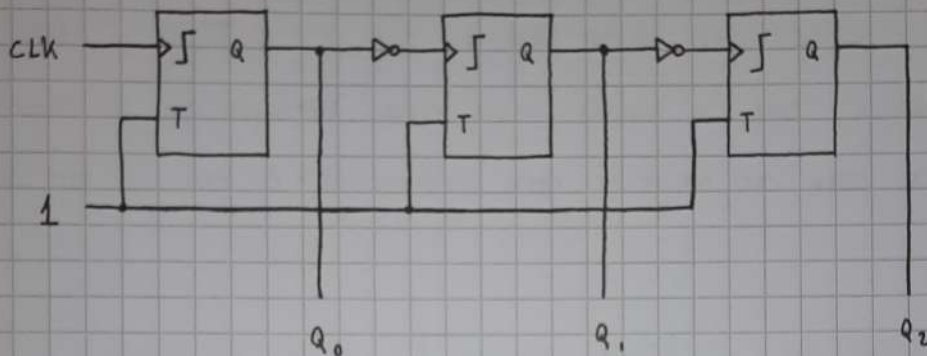
| $Q_2$ | $Q_1$ | $Q_0$ |
|-------|-------|-------|
| 0     | 0     | 0     |
| 0     | 0     | 1     |
| 0     | 1     | 0     |
| 0     | 1     | 1     |
| 1     | 0     | 0     |
| 1     | 0     | 1     |
| 1     | 1     | 0     |
| 1     | 1     | 1     |



OSSERVIAMO CHE OGNI VOLTA CHE IN UNA COLONNA SI PASSA DA 1 A 0 (FRONTE DI DISCESA) NELLA COLONNA A SINISTRA SI CAMBIA STATO.

LA COLONNA  $Q_0$  (CIFRA MENO SIGNIFICATIVA) CAMBIA STATO SEMPRE ALT-CANANDO 0 E 1.

POTREMO QUINDI PENSARE DI COSTRUIRE UN CIRCUITO DEL GENERE:



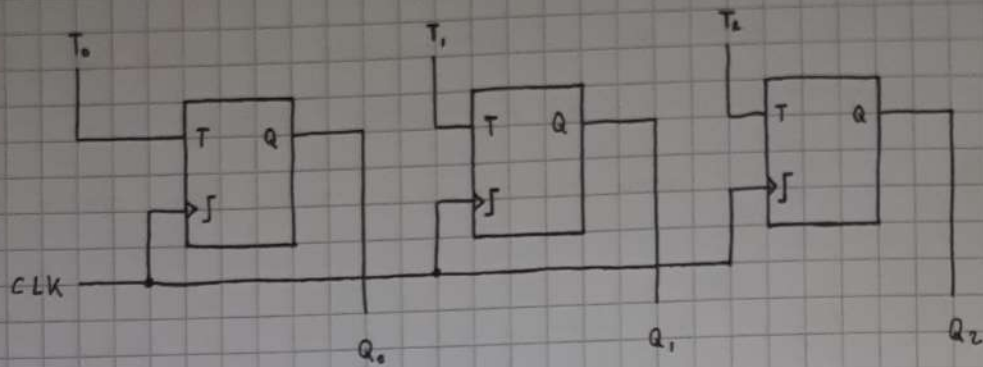
IN SOSTANZA  $Q_0$  VARIA AD OGNI FRONTE POSITIVO DEL CLOCK MENTRE I BIT SUCCESSIVI CAMBIANO SOLO QUANDO QUELLO PRECEDENTE VENGA UN FRONTE NEGATIVO (POI C'È IL NEGATO PERCHÉ IL FF-T SI ATTIVA CON IL FRONTE POSITIVO).

SI OSSERVI CHE IL BIT  $Q_2$  DEVE ASPETTARE, QUANDO TUTTI I BIT DEVONO CAMBIARE, DEVE ASPETTARE CHE IL SEGNALE SI PROPAGHI ATTRAVERSO TUTTI I FF-T PRECEDENTI.

IL TUTTO DEVE ANCHE AVVENIRE PRIMA DEL SUCCESSIVO FRONTE DI SALITA, DUNQUE SCARTIAMO QUESTO **RIPPLE COUNTER** E PROVIAMO A COSTRUIRE UNO MLLIORE.

IL SEGUENTE SCHEMA RAPPRESENTA UN CIRCUITO FORMATO DA FF-T IN PARALLELO CHE RICEVONO IL SEGNALE DI CLOCK CONTEMPORANEAMENTE, NON VIENE PROPAGATO COME PRIMA:



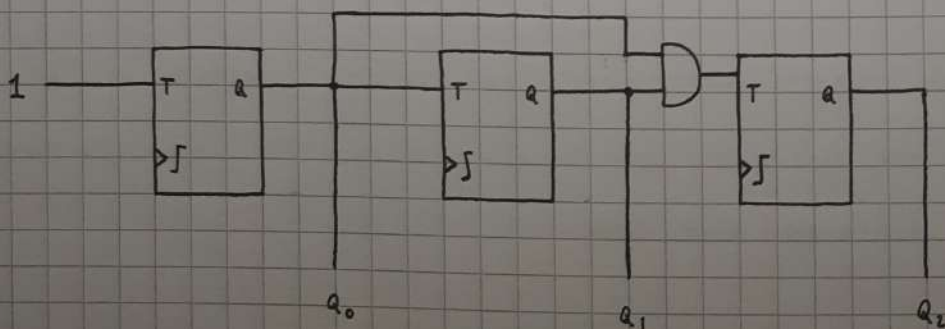


È ORA DI CAPIRE CHE VALORI DARE A  $T_0$ ,  $T_1$ , E  $T_2$ , PER FARLO CREIAMO UNA TABELLA CON IL VALORE DELLA CIFRA RAPPRESENTATA AL MOMENTO, CON IL VALORE DELLA CIFRA CHE AVREMO AL PASSO SUCCESSIVO E CON IL BIT CHE DOVrà CAMBIARE:

| $Q_2$ | $Q_1$ | $Q_0$ | $Q'_2$ | $Q'_1$ | $Q'_0$ | $T_2$ | $T_1$ | $T_0$ |
|-------|-------|-------|--------|--------|--------|-------|-------|-------|
| 0     | 0     | 0     | 0      | 0      | 1      | 0     | 0     | 1     |
| 0     | 0     | 1     | 0      | 1      | 0      | 0     | 1     | 1     |
| 0     | 1     | 0     | 0      | 1      | 1      | 0     | 0     | 1     |
| 0     | 1     | 1     | 1      | 0      | 0      | 1     | 1     | 1     |
| 1     | 0     | 0     | 1      | 0      | 1      | 0     | 0     | 1     |
| 1     | 0     | 1     | 1      | 1      | 0      | 0     | 1     | 1     |
| 1     | 1     | 0     | 1      | 1      | 1      | 0     | 0     | 1     |
| 1     | 1     | 1     | 0      | 0      | 0      | 1     | 1     | 1     |

È IMMEDIATO VERIFICARE CHE  $T_0 = 1$ ,  $T_1 = Q_0$ , E  $T_2 = Q_0 \cdot Q_1$  (IL PATTERN CONTINUEREBBE CON  $T_3 = Q_0 \cdot Q_1 \cdot Q_2$ ,  $T_4 = Q_0 \cdot Q_1 \cdot Q_2 \cdot Q_3$ ).

ADUNQUE COSTRUIAMO IL CIRCUITO DEFINITIVO:



QUESTO PRENDERà IL NOME DI FF-T COUNTER SINCRONO.

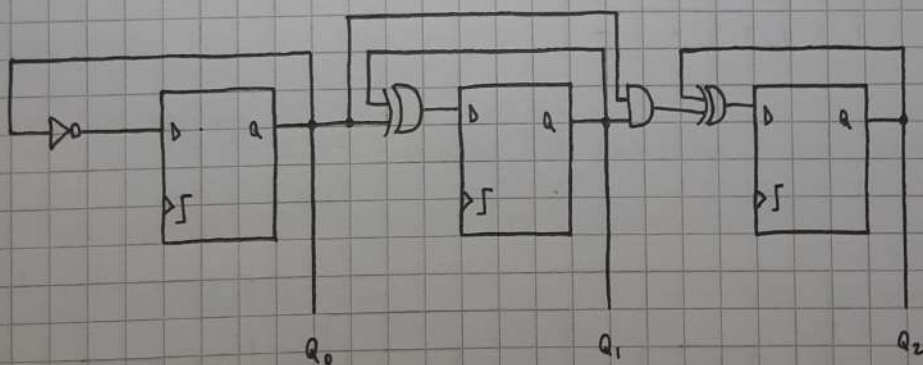


PRIVIAMO AM A CREARE UN COUNTER CON DEI FF-D. COME ABBIAMO GIÀ VISTO  
 CI CONVIENE COMPILARE UNA PICCOLA TABELLA DOVE INDICHIAMO IL VALORE CHE  
 DEVIAMO ASSUMERE  $D_0, D_1, D_2$  (EVENTUALMENTE SI PUÒ CONTINUARE CON  $D_3, \dots, D_n$ ):

| $Q_2$ | $Q_1$ | $Q_0$ | $Q'_2$ | $Q'_1$ | $Q'_0$ | $D_2$ | $D_1$ | $D_0$ |
|-------|-------|-------|--------|--------|--------|-------|-------|-------|
| 0     | 0     | 0     | 0      | 0      | 1      | 0     | 0     | 1     |
| 0     | 0     | 1     | 0      | 1      | 0      | 0     | 1     | 0     |
| 0     | 1     | 0     | 0      | 1      | 1      | 0     | 1     | 1     |
| 0     | 1     | 1     | 1      | 0      | 0      | 1     | 0     | 0     |
| 1     | 0     | 0     | 1      | 0      | 1      | 1     | 0     | 1     |
| 1     | 0     | 1     | 1      | 1      | 0      | 1     | 1     | 0     |
| 1     | 1     | 0     | 1      | 1      | 1      | 1     | 1     | 1     |
| 1     | 1     | 1     | 0      | 0      | 0      | 0     | 0     | 0     |

OVVIAMENTE, ESSENDO IL FF-D UN SEMPLICE DIKETTO CHE TRASMETTE IN USCITA CIÒ  
 CHE ARRIVA IN ENTRATA, AVREMO  $D_i = Q'_i$ , MA NOI VOGLIAMO ESPRIMERE  $D_i$  CON  
 $Q_i$ .

SI VERIFICA ANCHE QUI ABBASTANZA FACILMENTE CHE  $D_0 = \bar{Q}_0$ ,  $D_1 = Q_1 \oplus Q_0$  E  
 $D_2 = Q_2 \oplus (Q_1 \cdot Q_0)$ , SI POTREBBE CONTINUARE CON  $D_3 = Q_3 \oplus \underbrace{(Q_2 \cdot Q_1 \cdot Q_0)}_{T_3} \dots$   
 DISEGNAMO QUINDI IL CIRCUITO:



CHIAMEREMO QUESTO CONTATORE **FF-D COUNTER SINCRONO**.



SUPPONIAMO ORA DI VOLER IMPLEMENTARE UN CONTATORE IN MODULO, CIOÉ CHE CONTI DA 0 A  $n$  E POI RICOMINCI.

AD ESEMPIO PROVIAMO A CREARNE UNO MODULO 5, COMPILIAMO QUINDI LA SOLITA TABELLA:

| $Q_2$ | $Q_1$ | $Q_0$ | $Q_2'$ | $Q_1'$ | $Q_0'$ | $T_2$ | $T_1$ | $T_0$ |
|-------|-------|-------|--------|--------|--------|-------|-------|-------|
| 0     | 0     | 0     | 0      | 0      | 1      | 0     | 0     | 1     |
| 0     | 0     | 1     | 0      | 1      | 0      | 0     | 1     | 1     |
| 0     | 1     | 0     | 0      | 1      | 1      | 0     | 0     | 1     |
| 0     | 1     | 1     | 1      | 0      | 0      | 1     | 1     | 1     |
| 1     | 0     | 0     | 0      | 0      | 0      | 1     | 0     | 0     |

COSTAVIAMO BUNQUE LE K-MAP PER DEFINIRE LE FUNZIONI DI  $T_2$ ,  $T_1$  E  $T_0$  (SI OSSERVI CHE ALCUNI CASI, UGUALI SUCCESSIVI A 100, METTEREMO IL DON'T CARE SULLE K-MAP):

| $Q_2 \backslash Q_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| 0                    | 0  | 0  | -  | 1  |
| 1                    | 0  | 1  | -  | -  |

$$T_2 = Q_2 + Q_1 Q_0$$

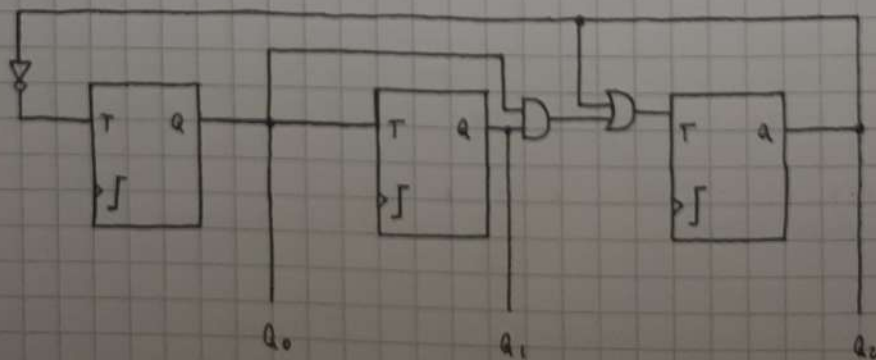
| $Q_2 \backslash Q_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| 0                    | 0  | 0  | -  | 0  |
| 1                    | 1  | 1  | -  | -  |

$$T_1 = Q_0$$

| $Q_2 \backslash Q_1$ | 00 | 01 | 11 | 10 |
|----------------------|----|----|----|----|
| 0                    | 1  | 1  | -  | 0  |
| 1                    | 1  | 1  | -  | -  |

$$T_0 = \bar{Q}_2$$

È BUNQUE IMMEDIATO COSTAVIAMO IL CIRCUITO:





# MACCHINE A STATI FINITI

LE MACCHINE A STATI FINITI SONO CIRCUITI IL CUI RISULTATO DIPENDE ANCHE DA CIÒ CHE L'USCITA VALEVA NEI PRECEDENTI CICLI DI CLOCK.

LE DEFINIAMO TRAMITE UNA SESTUPLA DI OGGETTI CHE ORA DEFINIRÒ:

- $S$  : INSIEME DEGLI STATI ( FINITO E NON VUOTO ).
- $I$  : INSIEME DEI SIMBOLI CHE POSSONO ARRIVARE IN INGRESSO ( QUALUNQUE COSA ASTRATTA PURCHÉ FINITA ), È DETTO ANCHE ALFABETO.
- $O$  : INSIEME DEI SIMBOLI CHE LA FSM ( FINITE STATE MACHINE ) PUÒ GENERARE IN USCITA.
- $\delta$  : RAPPRESENTA LA FUNZIONE CHE CI DICE QUALE STATO DEVE ASSUMERE LA MACCHINA AL PROSSIMO CICLO.
- $\lambda$  : RAPPRESENTA LA FUNZIONE CHE CI DICE CICLO PER CICLO QUAL'È L'USCITA ( L'OUTPUT ) DELLA MACCHINA.
- $s_0$  : È UN ELEMENTO DI  $S$  E RAPPRESENTA LO STATO INIZIALE DELLA MACCHINA.

RIGUARDO LA FUNZIONE  $\delta$  POSSIAMO DIRE CHE DEVE DIPENDERE DA:

- $S$  , PER TENERE TRACCIA DELLO STORICO
- $I$  , DEVE PUR DIPENDERE DAGLI INPUT CORRENTI

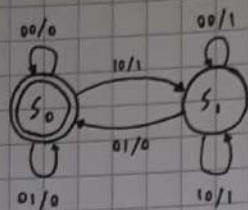
DIVERSO È PER LA FUNZIONE  $\lambda$  , CHE SE DOVESSE DIPENDERE SOLO DALLLO STORICO DEGLI STATI (  $S$  ) DAREBBE IL NOME DI **MOORE FSM** ALLA MACCHINA. MENTRE SE DOVESSE DIPENDERE DALLLO STORICO E DAGLI INPUT CORRENTI (  $S, I$  ) DAREBBE IL NOME DI **MEALY FSM** ALLA MACCHINA:

$$y(n) = f(x(0), \dots, x(n-1)) \quad \text{MOORE}$$

$$y(n) = f(x(0), \dots, x(n-1), x(n)) \quad \text{MEALY}$$



FACCIAMO ORA UN ESEMPIO DI MACCHINA A STATI FINITI DI MEALY,  
MOSTRANDO COME VA RAPPRESENTATA:



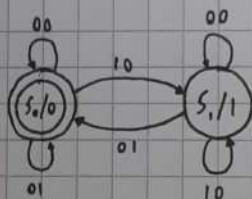
|       | 00      | 01      | 10      |
|-------|---------|---------|---------|
| $S_0$ | $S_0/0$ | $S_0/0$ | $S_1/1$ |
| $S_1$ | $S_1/1$ | $S_0/0$ | $S_1/1$ |

COME POSSIAMO NOTARE 00, 01, 10 RAPPRESENTANO GLI INPUT MENTRE 0 E 1 SONO GLI OUTPUT.

POSSIAMO DIRE CHE SI TRATTA DI UNA MEALY FSM PERCHÉ L'OUTPUT ( $\lambda$ ) SI TROVA SU OGNI SINGOLA FRECCIA E QUINDI DIPENDE SIA DAL VALORE DI QUELLA FRECCIA ( $L(INPUT, I)$ ) SIA DALLO STATO DA CUI È PARTITA ( $S$ ).

IL DOPIO CERCHIO SU  $S_0$  INDICA CHE QUELLO È IL PRIMO STATO IN CUI SI TROVERÀ LA FSM ALL'ACCENSIONE.

PROVIAMO ORA A FARE UN ESEMPIO DI MOORE FSM:



|       | 00    | 01    | 10    |
|-------|-------|-------|-------|
| $S_0$ | $S_0$ | $S_0$ | $S_1$ |
| $S_1$ | $S_1$ | $S_0$ | $S_1$ |

| $S_0$ | 0 |
|-------|---|
| $S_1$ | 1 |

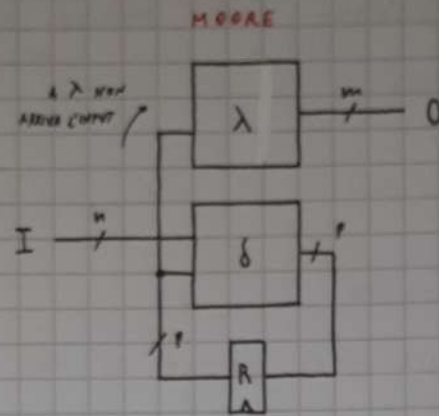
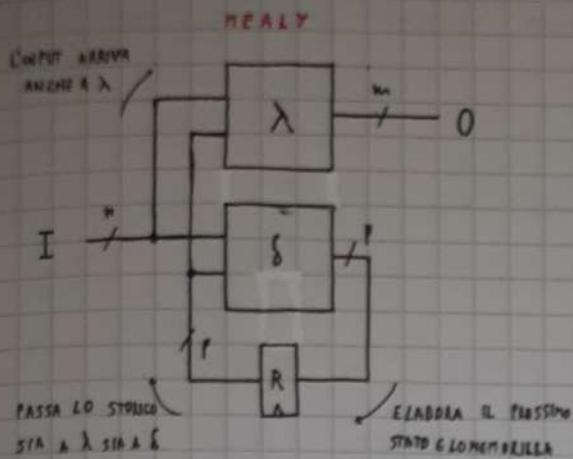
NOTIAMO SUBITO CHE L'OUTPUT ( $\lambda$ ) NON SI TROVA PIÙ NELLE FRECCIE MA ALL'INTERNO DI OGNI STATO, INFATTI DEVE DIPENDERE SOLTANTO DALLO STATO ( $S$ ).

È INOLTRE NECESSARIA UN'ALTRA TABELLA PER ASSEGNARE UN VALORE AI SIMBOLI DEGLI STATI.

PER IMPLEMENTARE CON DEI CIRCUITI LE MACCHINE A STATI FINITI CI RIPAREMO AL MODELLO DI HUFFMAN, CHE PREVEDE L'UTILIZZO DI REGISTRI FORMATI DA FF-D PER LA MEMORIZZAZIONE DELLO STATO DEGLI STATI.

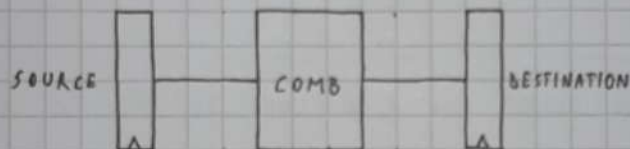
DI SEGUITO 2 ESEMPI (MEALY E MOORE):





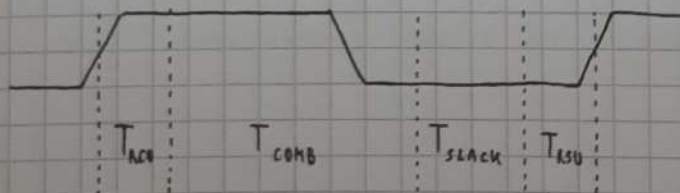
IL FATTO È CHE AD OGNI CICLO DI CLOCK IL REGISTRO DEVE AVERE NUOVI ELEMENTI, ALTRIMENTI POTREBBAMO ESSERE DEI PROBLEMI (A  $\lambda$  POTREBBE ARRIVARE UNO STORICO NON AGGIORNATO E L'OUTPUT POTREBBE RIMANERE SBALECIATO).

CONSIDERIAMO IL SEGUENTE SCHEMA:



DOVE SOURCE E DESTINATION, COME ABBIAMO VISTO PRIMA, POSSONO CONSIDERARE.

CI ASPETTIAMO CHE IL TEMPO PER PASSARE DA UN REGISTRO ALL'ALTRO SIA MINORE DI UN CLOCK. RIASSUMO QUESTO RAGIONAMENTO IN UN DIAGRAMMA:



DOVE  $T_{ACO}$  È IL TEMPO DI LANCIO ATTRAVERSO IL PRIMO REGISTRO,

$T_{COMB}$  È IL TEMPO DI ELABORAZIONE CARATTERISTICO DELLA RETE COMBINATORIA,

$T_{RSU}$  È IL TEMPO DI CATTURA DEL SECONDO REGISTRO E

$T_{SLACK}$  È IL TEMPO RIMANENTE



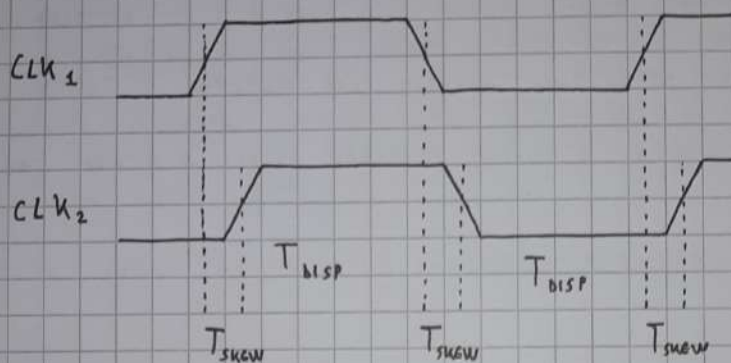
NE CONCLUDIAMO CHE AFFINCHÉ NON CI SIANO PROBLEMI DOVE VERIFICARSI LA

DISUGUAGLIANZA:

$$T_{CLK} \geq T_{ACO} + T_{COMB} + T_{RSU}$$

E, SE VOGLIAMO,  $T_{SEACH} \geq 0$ .

NELLA REALTÀ FISICA SUCCEDDE CHE NON TUTTI I COMPONENTI DEL CIRCUITO VEDONO E ANALIZZARE IL CLOCK NELLO STESSO MOMENTO, TUTTAVIA AL NOSTRO LIVELLO DI ASTRUZIONE NON POSSIAMO STIMARLO.

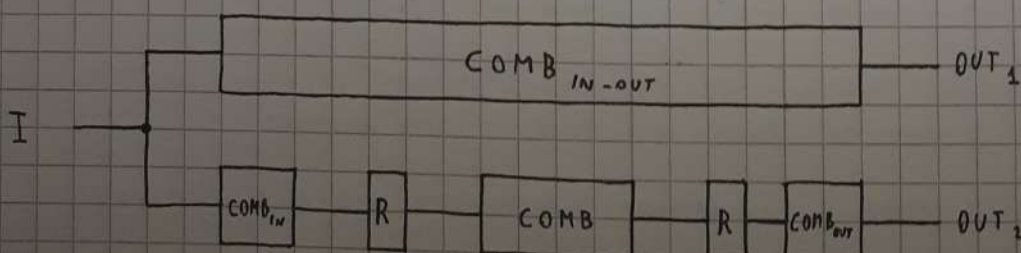


INDICIAMO CON  $T_{skew}$  IL RITARDO TRA LA RICEZIONE DA PARTE DI COMPONENTI DIVERSI DEL CLOCK. SI OSSERVA SUBITO CHE  $T_{disp} < T_{CLK}$ .

NEI NOSTRI ESERCIZI CI VERRÀ DATO UN VALORE MASSIMO PER  $T_{skew}$ , PER IL QUALE DOVREMO AVERE:

$$T_{CLK} \geq T_{ACO} + T_{COMB} + T_{RSU} + T_{skew}$$

OGNI CIRCUITO SEQUENZIALE PUÒ ESSERE SCHEMATIZZATO COME SEGUE:

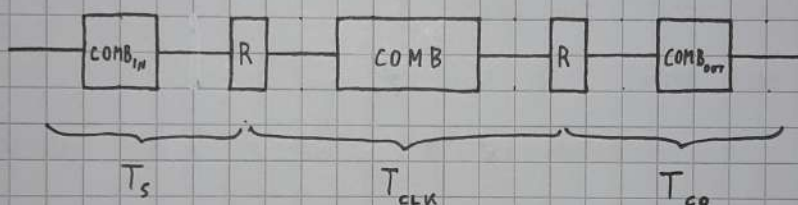




È POSSIBILE, COME NEL CASO DELLE MACCHINE MEALY E MOORE, CHE I due REGISTRI COINCIDANO.

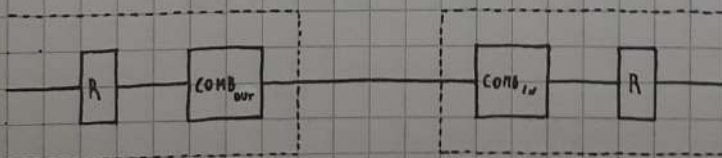
AD OGNI MODO IL RISULTATO FINALE, PUÒ ESSERE CALCOLATO TRAMITE UN SEMPLICE CIRCUITO COMBINATORIO (CHE RICHIEDEREBBE UN TEMPO  $T_{\text{comb}}$ ) O ATTRAVERSO UN CIRCUITO SEQUENZIALE (CHE PREVEDE L'IMPIEGO DI REGISTRI).

IN QUESTO SECONDO CASO IL PASSAGGIO DA REGISTRO A REGISTRO DEVE RICHIEDERE MENO DI UN TEMPO DI CLOCK, INOLTRE L'ELABORAZIONE DELL'INPUT DA PARTE DEL  $\text{COMB}_{\text{in}}$  E LA RICEZIONE DEI RISULTATI DA PARTE DEL PRIMO REGISTRO DEVE AVVENIRE PRIMA DEL FRONTE DI CLOCK DI UN TEMPO  $T_{\text{setup}}$ , INFINE I RISULTATI DEVONO USCIRE DAL SECONDO REGISTRO ED ESSERE ELABORATI DAL  $\text{COMB}_{\text{out}}$  E CIÒ RICHIEDERÀ UN TEMPO  $T_{\text{clock-to-output}}$  O  $T_{\text{co}}$ .



NOTIAMO CHE È LA STESSA STRUTTURA, IN TERMINI DI TEMPI, CHE ABBIAMO VISTO PER IL FLIP-FLOP, SOLO CHE IN QUESTO CASO  $T_s$  È IL TEMPO DI SETUP DEL CIRCUITO INTERO (STESSA COSA PER  $T_{\text{co}}$ ).

POTREMMO ORA CHIEDERCI COSA SUCCEDERÀ SE COLLEGHIAMO TRA LORO 2 DISPOSITIVI:



IN QUESTO CASO SI TRATTA DI PASSARE DA UN REGISTRO ALL'ALTRO, QUINDI IL TEMPO DI CLOCK DOVrà ESSERE MAGGIORE O UGUALE AL TEMPO  $T_{R_1-R_2}$ .



IN PARTICOLARE IL TEMPO DI CLOCK DI QUESTO NUOVO CIRCUITO (UNIONE DI DUE DIVERSI) DOVRÀ ESSERE SICURAMENTE MAGGIORE DI ENTRAMBI I  $T_{CLK}$  DEI DUE CIRCUITI INIZIALI (ALTAMENTE, ASSUNTO CHE I REUSTI SENTANO IL PRIMO CLOCK INSIEME, UNO DEI DUE [IL PIÙ VELOCE] SENTIRÀ UN ALTRO CLOCK QUANDO IL SEGNALE NON SARÀ ANCORA PASSATO DA UN CIRCUITO ALL'ALTRO, IN REALTÀ DA UN REUSTO ALL'ALTRO).  
INOLTRE DOVRÀ ESSERE MAGGIORE ANCHE DEL TEMPO CHE CI METTE AD USCIRE DAL DISPOSITIVO 1 + IL TEMPO CHE CI METTE AD ENTRARE NEL CIRCUITO O DISPOSITIVO 2:

$$T_{CLK} \geq \max \{ T_{CLK_1}, T_{CLK_2}, T_{CO_1} + T_{S_2} \}$$

### ESEMPIO MEALY

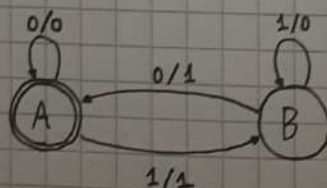
VOLIAMO COSTRUIRE UNA CIRCUITO CON LE SEGUENTI CARATTERISTICHE:

- HA UN SOLO INPUT CHE DIREMO  $X$
- HA UN SOLO OUTPUT CHE DIREMO  $Y$
- SE  $X_i = X_{i-1}$ , ALLORA  $Y = 0$ , SE  $X_i \neq X_{i-1}$ , ALLORA  $Y = 1$

LA MACCHINA PUÒ ESSERE COSTRUITA A PARTIRE DALLA DOMANDA: COSA MI SERVE TENERE MEMORIZZATO PER RISPONDERE ALLA DOMANDA DEL PUNTO 3?

EBBENE DATO CHE STIAMO CREANDO UNA MACCHINA DI MEALY CI BASTA MEMORIZZARE IL VALORE DEL BIT ENTRATO IN INGRESSO AL CICLO PRECEDENTE (DATO CHE QUELLO CORRENTE DEL CICLO ATTUALE VIENE ELABORATO DALLA  $X$ ).

AVREMO QUINDI BISOGNO DI 2 STATI, ~~MA~~ ~~NO~~ CI TROVEREMO IN A SE IL BIT PRECEDENTE ERA 0 E IN B SE IL BIT PRECEDENTE ERA 1:



DECIDIAMO DI CODIFICARE A CON 0 E B CON 1 E COSTRUIAMO LA TABELLA DELLE TRANSIZIONI:



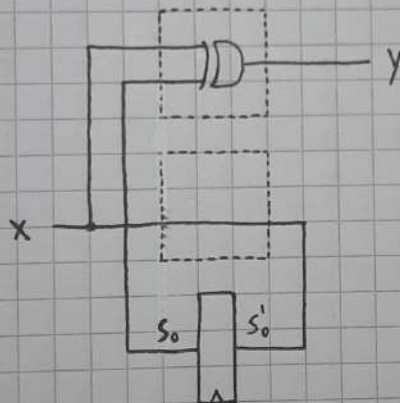
| $s_0$ | $x$ | $s'_0$ | $y$ |
|-------|-----|--------|-----|
| 0     | 0   | 0      | 0   |
| 0     | 1   | 1      | 1   |
| 1     | 0   | 0      | 1   |
| 1     | 1   | 1      | 0   |

DOVE  $s_0$  RAPPRESENTA L'USCITA CORRENTE DAL REGISTRO (DI FATTO RAPPRESENTA IL VALORE DELL'ULTIMO BIT) E  $s'_0$  IL VALORE CHE VIENE MEMORIZZATO NEL REGISTRO ADESSO.

È IMMEDIATO VEDERE CHE:

$$s'_0 = x \quad y = s_0 \oplus x$$

DA CUI SI RICAVALA SUBITO IL CIRCUITO.



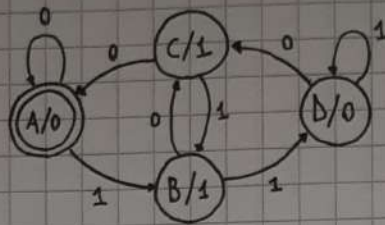
## ESEMPIO MOORE

SUPPONIAMO DI DOVER COSTRUIRE UNA MACCHINA DI MOORE CHE PRESENTI LE STESSA CARATTERISTICHE DELL'ESEMPIO PRECEDENTE.

DOBBIAMO QUINDI TENER CONTO CHE PER GENERARE L'USCITA NON POTREMO PIÙ USARE L'INPUT CORRENTE, MA POTREMO SOLO USARE LO STORICO DEGLI STATI.

PROCEDEREMO IN MANIERA INCREMENTALE, A PARTIRE DALLO STATO A CORRISPONDENTE ALLA SITUAZIONE ULTIMA 2 BIT PARI A 0 CI CHIEDEREMO COSA SUCCEDE SE ARRIVA IN INPUT UNO 0 O UN 1.





DOVE GLI STATI A, B, C, D HANNO LE CARATTERISTICHE:

|   | ULTIMO | PENULTIMO |
|---|--------|-----------|
| A | 0      | 0         |
| B | 1      | 0         |
| C | 0      | 1         |
| D | 1      | 1         |

DECHO DI CONFIGURARE A, B, C E D COME MOSTRATO NELLA FIGURA SOPRA, SARANNO BASTANTI NECESSARI 2 BIT PER OGNI STATO E DUNQUE 2 REGISTRI. LA TABELLA DELLE TRANSIZIONI SARÀ:

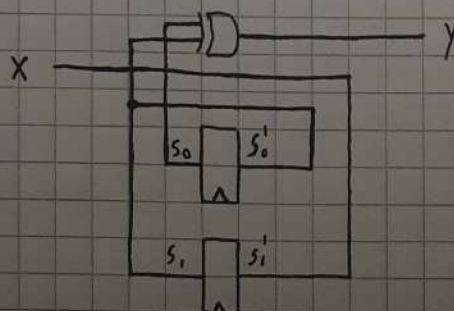
| $s_1$ | $s_0$ | X | $s'_1$ | $s'_0$ |
|-------|-------|---|--------|--------|
| 0     | 0     | 0 | 0      | 0      |
| 0     | 0     | 1 | 1      | 0      |
| 0     | 1     | 0 | 0      | 0      |
| 0     | 1     | 1 | 1      | 0      |
| 1     | 0     | 0 | 0      | 1      |
| 1     | 0     | 1 | 1      | 1      |
| 1     | 1     | 0 | 0      | 1      |
| 1     | 1     | 1 | 1      | 1      |

| $s_1$ | $s_0$ | Y |
|-------|-------|---|
| 0     | 0     | 0 |
| 1     | 0     | 1 |
| 0     | 1     | 1 |
| 1     | 1     | 0 |

DA CUI SI CARIOSCE SUBITO CHE:

$$y = s_0 \oplus s_1, \quad s'_0 = s_1, \quad s'_1 = x$$

DA CUI SI RICAHA IL CIRCUITO:



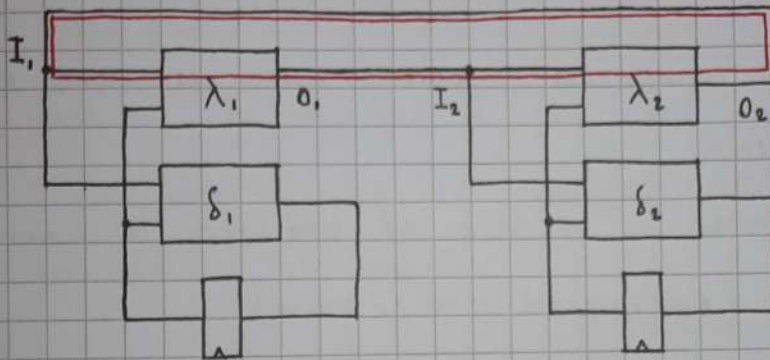


## UNIONE DI FSM

ABBIAMO GIÀ DISCUSSO SU QUALI DEBBANO ESSERE I TEMPI DI ELABORAZIONE DI UN DISPOSITIVO RICAVATO A PARTIRE DALL'UNIONE DI 2 DISPOSITIVI INIZIALI. PROVIAMO ORA AD UNIRE TRA LORO 2 MACCHINE A STATI FINITI, FACENDO IN MODO CHE L'OUTPUT DI UNA SIA L'INPUT DELL'ALTRA.

MEALY - MEALY

ANZITUTTO DISEGNIAMO LO SCHEMA:



SEMBREBBE ANDARE TUTTO BENE, SE NON FOSSE CHE IL PERCORSO CHE HO EVIDENZIATO RAPPRESENTA UN LOOP PURAMENTE COMBINATORIO (DA EVITARE).

IN QUESTA SITUAZIONE AVREMO:

$$O_2 = \lambda_2(I_2) = \lambda_2(O_1) = \lambda_2(\lambda_1(I_1)) = \lambda_2(\lambda_1(O_2))$$

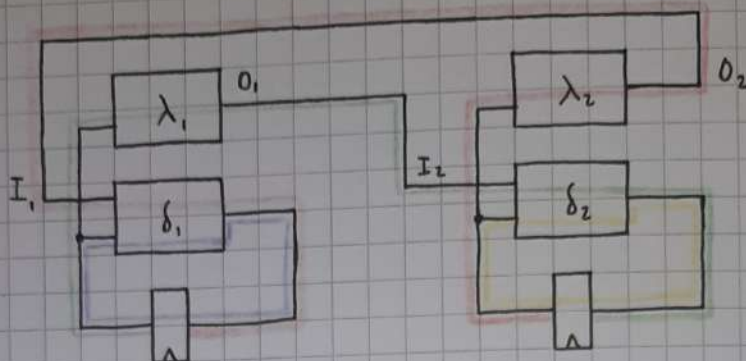
IN REALTÀ NON DIPENDE SOLO DA  $O_2$ , LAMBDA NEVE ANCHE LO STORICO DEGLI STATI IN INGRESSO MA L'HO FATTO PER EVIDENZIARE LA DIPENDENZA DI  $O_2$  DA SE STESSO.

QUINDI SCARTIAMO QUESTO DISPOSITIVO.



MOORE - MOORE

ANCHE QUI RISEGNAMO LO SCHEMA:



QUI NON SONO PRESENTI LOOP COMBINATORI, POSSIAMO QUINDI PROCEDERE E DETERMINARE UN VALORE MINIMO PER IL CLOCK.

SI OSSERVA CHE BASTA PRENDERE IL PERCORSO PIÙ DISPENDIOSO (IN TERMINI DI TEMPO) CHE VA DA REGISTRO A REGISTRO (EVENTUALMENTE PURE LO STESSO).

MI SERVIRÒ DEI COLORI PER EVIDENZIARE I VARI PERCORSI:

$$T_{CLK} \geq T_{ACO} + T_{\delta_1} + T_{RSU}$$

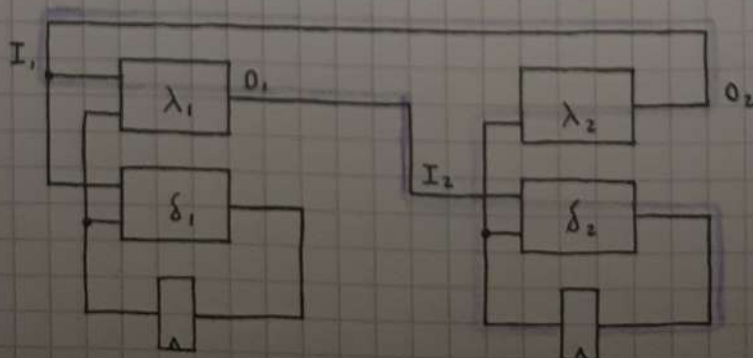
$$T_{CLK} \geq T_{ACO} + T_{\lambda_1} + T_{\delta_2} + T_{RSU}$$

$$T_{CLK} \geq T_{ACO} + T_{\delta_2} + T_{RSU}$$

$$T_{CLK} \geq T_{ACO} + T_{\lambda_2} + T_{\delta_1} + T_{RSU}$$

MEALY - MOORE

LO SCHEMA SARÀ:





SIMILMENTE A QUANTO DETTO PER IL DISPOSITIVO MOORE-MOORE NON CI SONO LOOP COMBINATORI.

VALGONO INOLTRE LE STESSSE 4 CONDIZIONI GIÀ DISCUSSE A CUI (ESSENDOCI UNA MEALY) SE NE AGGIUNGE UNA QUINTA CHE EVIDENZIARÒ:

$$T_{CLK} \geq T_{REC} + T_{\lambda_2} + T_{\lambda_1} + T_{\delta_2} + T_{RSU}$$

SI RICORDA CHE  $T_{REC}$  E  $T_{RSU}$  SONO I TEMPI DI INGRESSO E USCITA DEI DATI DA I REGISTRI DELLE DUE MACCHINE.

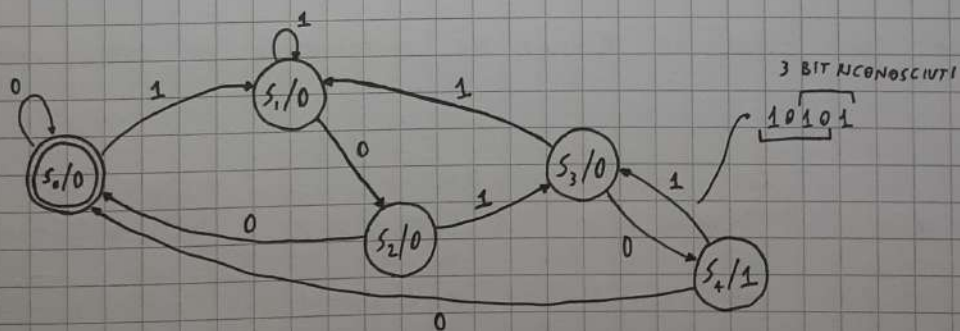
## ESEMPI DI MACCHINE A STATI FINITI

PROVIAMO A COSTRUIRE UNA MACCHINA A STATI FINITI CHE RICONOSCA LA SEQUENZA 1010 E DIA COME RISULTATO 1 QUANDO LA TROVA (SEQUENZE CONSECUTIVE POSSONO SOVRAPPORSI). AD ESEMPIO:

$$I = 000101000101010$$

$$y = 0000001000000101$$

COSTRUIAMO ANCHE LA MACCHINA IN MANIERA INCREMENTALE, CIOÈ QUANDO CI TROVIAMO IN UNA NUOVA SITUAZIONE AGGIUNGIAMO UNO STATO.



DOVE  $S_0 \Rightarrow$  NESSUN BIT RICONOSCIUTO

$S_2 \Rightarrow (10)$  2 BIT RICONOSCIUTI

$S_4 \Rightarrow (1010)$  SEQUENZA RICONOSCIUTA

$S_1 \Rightarrow (1)$  UN BIT RICONOSCIUTO

$S_3 \Rightarrow (101)$  3 BIT RICONOSCIUTI



A QUESTO PUNTO SFRUTTIAMO LA CODIFICA BINARIA PER CODIFICARE I 5 STATI DELLA MACCHINA (3 COMBINAZIONI SARANNO INUTILIZZATE, PERCIÒ CORRISPONDERANNO A DEI DON'T CARE):

|       | $s_2$ | $s_1$ | $s_0$ |
|-------|-------|-------|-------|
| $s_0$ | 0     | 0     | 0     |
| $s_1$ | 0     | 0     | 1     |
| $s_2$ | 0     | 1     | 0     |
| $s_3$ | 0     | 1     | 1     |
| $s_4$ | 1     | 0     | 0     |

ORA SCRIVO LA MACCHINA A STATI FINITI IN FORMA TABULARE:

| $s_2$ | $s_1$ | $s_0$ | $x$ | $s_2'$ | $s_1'$ | $s_0'$ |
|-------|-------|-------|-----|--------|--------|--------|
| 0     | 0     | 0     | 0   | 0      | 0      | 0      |
| 0     | 0     | 0     | 1   | 0      | 0      | 1      |
| 0     | 0     | 1     | 0   | 0      | 1      | 0      |
| 0     | 0     | 1     | 1   | 0      | 0      | 1      |
| 0     | 1     | 0     | 0   | 0      | 0      | 0      |
| 0     | 1     | 0     | 1   | 0      | 1      | 1      |
| 0     | 1     | 1     | 0   | 1      | 0      | 0      |
| 0     | 1     | 1     | 1   | 0      | 0      | 1      |
| 1     | 0     | 0     | 0   | 0      | 0      | 0      |
| 1     | 0     | 0     | 1   | 0      | 1      | 1      |

|       | $s_2$ | $s_1$ | $s_0$ | $y$ |
|-------|-------|-------|-------|-----|
| $s_0$ | 0     | 0     | 0     | 0   |
| $s_1$ | 0     | 0     | 1     | 0   |
| $s_2$ | 0     | 1     | 0     | 0   |
| $s_3$ | 0     | 1     | 1     | 0   |
| $s_4$ | 1     | 0     | 0     | 1   |

SI POSSONO ORA CREARE LE K-MAP PER DETERMINARE LE ESPRESSIONI DI  $s_2'$ ,  $s_1'$ ,  $s_0'$  E  $y$ :

| $s_2 s_1$ | $s_0 x$ | 00 | 01 | 10 | 11 |
|-----------|---------|----|----|----|----|
| 00        |         | -  | -  | -  | -  |
| 01        |         | -  | -  | -  | -  |
| 11        |         | -  | -  | -  | -  |
| 10        |         | 1  | -  | -  | -  |

| $s_2 s_1$ | $s_0 x$ | 00 | 01 | 11 | 10 |
|-----------|---------|----|----|----|----|
| 00        |         | -  | -  | -  | -  |
| 01        |         | 1  | -  | 1  | -  |
| 11        |         | -  | -  | -  | -  |
| 10        |         | 1  | -  | -  | -  |

| $s_2 s_1$ | $s_0 x$ | 00 | 01 | 11 | 10 |
|-----------|---------|----|----|----|----|
| 00        |         | -  | -  | -  | -  |
| 01        |         | 1  | 1  | -  | 1  |
| 11        |         | 1  | 1  | -  | -  |
| 10        |         | -  | -  | -  | -  |

| $s_2 s_1$ | $s_0$ | 00 | 01 | 11 | 10 |
|-----------|-------|----|----|----|----|
| 0         |       | -  | 1  | -  | -  |
| 1         |       | -  | -  | -  | -  |

$$s_2' = s_1 \cdot s_0 \cdot \bar{x}$$

$$s_1' = s_2 \cdot x + s_1 \cdot \bar{s}_0 \cdot x + \bar{s}_1 \cdot s_0 \cdot x$$

$$s_0' = x$$

$$y = s_2$$

UNA VOLTA TROVATE QUESTE FUNZIONI POSSIAMO FERMARCI, SENZA BISOGNO DI DOVERE DISEGNARE IL CIRCUITO.



SI OSSERVA CHE LA CODIFICA BINARIA NON È LA SOLA, ESISTE ANCHE LA CODIFICA ONE-HOT CHE CONSISTE IN:

|       | $s_4$ | $s_3$ | $s_2$ | $s_1$ | $s_0$ |
|-------|-------|-------|-------|-------|-------|
| $s_0$ | 0     | 0     | 0     | 0     | 1     |
| $s_1$ | 0     | 0     | 0     | 1     | 0     |
| $s_2$ | 0     | 0     | 1     | 0     | 0     |
| $s_3$ | 0     | 1     | 0     | 0     | 0     |
| $s_4$ | 1     | 0     | 0     | 0     | 0     |

SOSTANZIALMENTE SI USANO TANTI BIT PER LA CODIFICA QUANTI STATI HA LA MACCHINA, QUESTO HA DEI VANTAGGI NEL CALCOLO DELLE FUNZIONI (BASILMENTE AVENDO UN 1 PER COLONNA SI POSSONO COMBINARE PER OTTENERE TUTTE LE COMBINAZIONI CHE SI VOGLIONO) MA DEI SVANTAGGI IN TERMINI DI AREA, INFATTI SERVIRANNO REGISTRI PIÙ GRANDI E DUNQUE PIÙ FLIP-FLOP.

### ESEMPIO COMPLICATO DI FSM

SUPPONIAMO DI VOLER PROGETTARE UNA MACCHINA A STATI FINITI CHE RICEVE 2 INPUT A, B ALLA VOLTA E QUANDO NE HA RICEVUTI 4 (DUNQUE 8 IN TOTALE, 4 A E 4 B) RESTITUISCE 1 SE IL NUMERO RAPPRESENTATO DAI VALORI DI A È MAGGIORE DI QUELLO RAPPRESENTATO DAI VALORI DI B, 0 ALTRIMENTI. SE NON SONO STATI RICEVUTI 4 BIT LA MACCHINA RESTITUISCE DEI DON'T CARE.

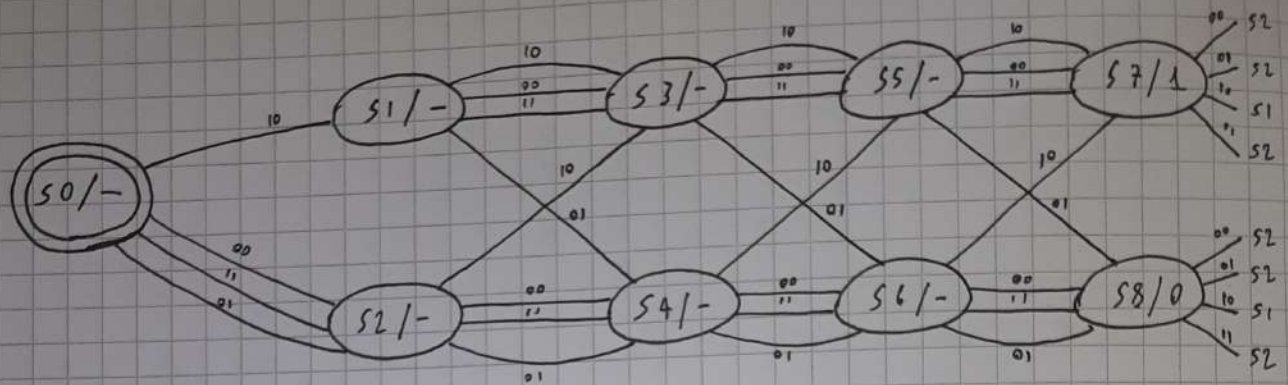
FACCIAMO UN ESEMPIO:

|   | 12 |   |   |   | 13 |   |   |   | 3 |   |   |   | 6  |   |   |   |
|---|----|---|---|---|----|---|---|---|---|---|---|---|----|---|---|---|
| A | 0  | 0 | 1 | 1 | 1  | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0  | 1 | 1 | 0 |
|   | 7  |   |   |   | 10 |   |   |   | 8 |   |   |   | 13 |   |   |   |
| B | 1  | 1 | 1 | 0 | 0  | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1  | 0 | 1 | 1 |
| y | -  | - | - | 1 | -  | - | - | 1 | - | - | - | 0 | -  | - | - | 0 |

SI OSSERVA CHE DA SPECIFICHE IL PRIMO BIT RICEVUTO È IL MENO SIGNIFICANTE, QUINDI I NUMERI VANNO LETTI DA DX A SX.



CIÒ CHE LA NOSTRA MACCHINA DEVE SAPERE È IL RISULTATO DEL CONFRONTO  
FINO AD ADESSO, DUNQUE:



QUESTO SEMPLICE SCHEMA DOVREBBE FAR CAPIRE MEGLIO LA LOGICA CON LA QUALE È  
STATA COSTRUITA LA NOSTRA MACCHINA:

1° COPPIA DI BIT

$A > B$                       ↑  
 A 0 0 1 1  
 B 0 1 0 1  
 $A \leq B$                       ↓ ↓ ↓

SUCCESSIVE COPPIE DI BIT

$A > B$                       ↑  
 A 0 0 1 1  
 =  
 B 0 1 0 1  
 $A \leq B$                       ↓

I VARI STATI RAPPRESENTANO LE SEGUENTI SITUAZIONI:

S0 - 0 BIT RICEVUTI

S1 - 1 BIT RICEVUTO  $A > B$

S2 - 1 BIT RICEVUTO  $A \leq B$

S3 - 2 BIT RICEVUTI  $A > B$

S4 - 2 BIT RICEVUTI  $A \leq B$

S5 - 3 BIT RICEVUTI  $A > B$

S6 - 3 BIT RICEVUTI  $A \leq B$

S7 - 4 BIT RICEVUTI  $A > B$

S8 - 4 BIT RICEVUTI  $A \leq B$



PER SCRIVERE LA FUNZIONE DI OUTPUT E QUELLA DI STATO PROSSIMO (SONO 8 IN REALTÀ) MI SERVIRÀ DELLA CODIFICA ONE-HOT, CHE PREVEDE:

|       | $S_0$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_8$ | $S_9$ |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $S_0$ | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| $S_1$ | 0     | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| $S_2$ | 0     | 0     | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| $S_3$ | 0     | 0     | 0     | 1     | 0     | 0     | 0     | 0     | 0     | 0     |
| $S_4$ | 0     | 0     | 0     | 0     | 1     | 0     | 0     | 0     | 0     | 0     |
| $S_5$ | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 0     | 0     | 0     |
| $S_6$ | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 0     | 0     |
| $S_7$ | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 0     |
| $S_8$ | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     |
| $S_9$ | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1     |

DA CUI SI POSSONO FACILMENTE RICAVARE LE FUNZIONI:

$$Y = S_7$$

$$S'_1 = S_0 \cdot \bar{A}\bar{B} + S_7 \cdot \bar{A}B + S_8 \cdot \bar{A}\bar{B}$$

$$S'_2 = S_0 \cdot \bar{A}\bar{B} + S_0 \cdot \bar{A}B + S_0 \cdot AB + S_7 \cdot \bar{A}\bar{B} + S_7 \cdot \bar{A}B + S_7 \cdot AB + S_8 \cdot \bar{A}\bar{B} + S_8 \cdot \bar{A}B + S_8 \cdot AB$$

$$S'_3 = S_1 \cdot \bar{A}\bar{B} + S_1 \cdot \bar{A}B + S_1 \cdot AB + S_2 \cdot \bar{A}\bar{B}$$

$$S'_4 = S_2 \cdot \bar{A}\bar{B} + S_2 \cdot \bar{A}B + S_2 \cdot AB + S_1 \cdot \bar{A}\bar{B}$$

$$S'_5 = S_3 \cdot \bar{A}\bar{B} + S_3 \cdot \bar{A}B + S_3 \cdot AB + S_4 \cdot \bar{A}\bar{B}$$

$$S'_6 = S_4 \cdot \bar{A}\bar{B} + S_4 \cdot \bar{A}B + S_4 \cdot AB + S_3 \cdot \bar{A}\bar{B}$$

$$S'_7 = S_5 \cdot \bar{A}\bar{B} + S_5 \cdot \bar{A}B + S_5 \cdot AB + S_6 \cdot \bar{A}\bar{B}$$

$$S'_8 = S_6 \cdot \bar{A}\bar{B} + S_6 \cdot \bar{A}B + S_6 \cdot AB + S_5 \cdot \bar{A}\bar{B}$$

E QUI TERMINA IL CORSO DI RETI LOGICHE TENUTO DAL PROFESSOR MIRKO LOGHI,  
PROSEGUIREMO CON IL CORSO DI ARCHITETTURA DEI CALCOLATORI.



# ARCHITETTURA DEI CALCOLATORI

IN QUESTA PARTE DEL CORSO METTAMO IN PRATICA CIÒ CHE ABBIAMO VISTO FINO AD ORESSO, VEDREMO COME SI UTILIZZANO I MODULI STUDIATI PER COSTRUIRE UN COMPUTER.

USCIREMO UN SIMULATORE PER CAPIRE IN PROFONDITÀ COSA SUCCEDDE ALL'INTERNO DEI PC E SALIREMO DI LIVELLO DI ASTRAZIONE.

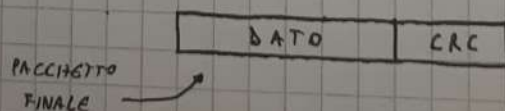
## LA CODIFICA DEI DATI

ORMAI IN QUALSIASI APPLICAZIONE DELL'INFORMATICA AVVIENE UN TRASFERIMENTO DI DATI, DUNQUE DOBBIAMO ASSICURARCI CHE I DATI RICEVUTI SIANO QUELLI CORRETTI, O COMunque, SE DOVESSERO PRESENTARE ERRORI, DOVREMO FARLI IN MODO ~~DA~~ DI ACCORGERCENE.

SPESSE CAPITA CHE QUESTI CONTROLLI DEGLI ERRORI VENGANO DIRETTAMENTE FATTI IN HARDWARE, CON RETI LOGICHE DEDICATE.

IN UN PRIMO MOMENTO SI POTREBBE PENSARE DI ALLEGGERIRE, AL MESSAGGIO DEI BIT DIRETTAMENTE COLLEGATI AL CONTENUTO DEL DATO DA TRASMETTERE. CHIAMEREMO **RIDONDANZA** QUESTA CARATTERISTICA E CI DOTEREMO DI UNA FUNZIONE IN GRADO DI ELABORARE I BIT DEL MESSAGGIO E FORNIRE UN **CRC** (CODICE A RIDONDANZA CICLICA).

IL NOSTRO DATO TRASMESSO SARÀ QUINDI:



ALLA RICEZIONE DEL MESSAGGIO SEPAREREMO IL DATO DAL CRC E RI-CALCOLEREMO IL CRC CON IL DATO CHE È RICEVUTO.

QUESTO CI PERMETTE DI CONFRONTARE I 2 CRC STABILENDO SE DURANTE LA



TRASMISSIONE CI SONO STATI DEGLI ERRORI.

IN REALTÀ ESISTONO COMBINAZIONI DI ERRORI MULTIPLI CHE POTREBBERO FAR RISULTARE I CRC IDENTICI NONOSTANTE DEGLI ERRORI NEL DATO, MA L'EQUIVALENZA TRA CRC È UNO RESTA COMUNQUE UN OTTIMO INDICATORE.

VEDIAMO ORA ALCUNI DEI PRINCIPALI METODI UTILIZZATI PER LA CREAZIONE DEL CODICE A RIDONDANZA CICLICA.

## PARITÀ

QUESTA TECNICA PREVEDE DI AGGIUNGERE, OGNI 7 BIT, UN BIT DI PARITÀ IN MODO TALE CHE NEL BYTE CI SIANO UN NUMERO PAR (0 DISPARI, A SECONDA DELLA PARITÀ) DI 1.

|   |   |   |   |   |   |                  |   |
|---|---|---|---|---|---|------------------|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1                | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1                | 1 |
|   |   |   |   |   |   | ↑                |   |
|   |   |   |   |   |   | BIT DI<br>PARITÀ |   |

PER DETERMINARE IL BIT DI PARITÀ CI È SUFFICIENTE UNIRE IN XOR I PRIMI

7 BIT:  $P = A_1 \oplus A_2 \oplus \dots \oplus A_7$ .

SI OSSERVI CHE POTREBBE ESSERE IN CORRETTO, ALLA RECESSIONE, ANCHE IL BIT DI PARITÀ.

## LA DISTANZA DI HAMMING

QUELLA CHE INTRODUCIAMO ORA NON È UNA TECNICA DI CODIFICA MA PIUTTOSTO UNA TECNICA CHE CI PERMETTE DI VALUTARE LE TECNICHE DI CODIFICA E LA LORO SOLIDITÀ.

INOLTRE DEFINIAMO LA **PAROLA DI CODICE**, CIOÈ UN PACCHETTO DATO + CRC COMPLETO (SE VERIFICHIAMO IL CRC NON TROVIAMO DIFFERENZE).



Vittorio Alzetta

ORA IMMAGINIAMO CHE I PALLINI SIANO PAROLE DI CODICE, MENTRE I QUADRATI SIANO PAROLE NON DI CODICE:



CIÒ CHE HO RAPPRESENTATO MOSTRA COME PER PASSARE DA UNA PAROLA DI CODICE ALL'ALTRA SIA NECESSARIO IL PASSAGGIO ATTRAVERSO UNA SOLA PAROLA NON DI CODICE, DOVE OGNI FRECCIA CORRISPONDE AL CAMBIAMENTO DI UN BIT DEL DATO + CRC.

IN PARTICOLARE NELLA CODIFICA DELLA PARITÀ SE CAMBIAMO 1 BIT CI ACCORGIAMO DELL'ERRORE, MENTRE SE NE CAMBIO 2, NONOSTANTE IL MESSAGGIO SIA ERRATO, NON CI POTREMO ACCORGERE DELLO SBAGLIO.

LA DISTANZA DI HAMMING È LA DISTANZA MINIMA TRA 2 PAROLE DI CODICE, IN QUESTO CASO VALE 2.

OSSERVATO CHE SE CAMBIO 2 PAROLE DI CODICE DIVERSE NON POTREMO MAI FINIRE SULLA STESSA PAROLA NON DI CODICE AVREMO CHE:



QUESTO RAPPRESENTA IL PASSAGGIO DA UNA PAROLA DI CODICE ALL'ALTRA ATTRAVERSO 4 CAMBIAMENTI (4 BIT MODIFICATI, ANCHE LO STESSO RIVOLTE).

TUTTAVIA, ESSENDO QUESTO PERCORSO UNICO, SE CI TROVIAMO AL PUNTO ① SAREMO IN GRADO NON SOLO DI RILEVARE L'ERRORE, MA ADDIRITTURA DI CORREGGERLO PROCEDENDO AL CONTRARIO VERSO LA PAROLA DI CODICE PIÙ VICINA.

GIÀ SE CI FOSSIMO TROVATI IN ② NON AVREMO AVUTO LA POSSIBILITÀ DI CORREGGERE L'ERRORE. IN ② LA P.D.C. PIÙ VICINA È ④ MA È SBAGLIATA, QUINDI CI PERVIAMO A ① POSSIAMO OSSERVARE CHE VOLENDO SOLAMENTE ODORARE RILEVARE UN ERRORE



CI BASTERÀ PORRE  $D.H. = K + 1$ , DOVE  $K$  È IL NUMERO DI ERRORI CHE VOGLIAMO ALEVARE.

SE INVECE VOGLIAMO ANCHE CORREGGERE L'ERRORE DOVREMO PORRE

$D.H. = 2K + 1$ , IN QUANTO NOI POSSIAMO RISALIRE ALLA PAROLA DI CODICE CORRETTA FINO ALLA METÀ (METÀ ESCLUSA, EVENTUALMENTE) DEL PERCORSO CHE CI PORTA ALLA PAROLA DI CODICE SBAGLIATA.

TORNANDO ALLA CODIFICA DELLA PANTÀ VEDIAMO UN ESEMPIO (IN QUESTO CASO LA PANTÀ È PAN):

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |   |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

← BIT DI PANTÀ NERE

← BIT DI PANTÀ COLONNE

SUPPONIAMO CHE IL PRECEDENTE SIA IL MESSAGGIO CHE ABBIAMO RICEVUTO, SI OSSERVA FACILMENTE CHE È PRESENTE UN ERRORE NELLA RIGA E NELLA COLONNA CHE HO EVIDENZIATO IN ROSSO, QUESTO CI PERMETTE DI INDIVIDUARE E CORREGGERE IL BIT IN POSIZIONE (2,3). QUESTO SE HO AL MASSIMO UN ERRORE, ALTIMENTI NON SIAMO IN GRADO DI CORREGGERLI.

## 1 CODICI POLINOMIALI

SONO STATI SVILUPPATI BELLI ALGORITMI CHE HANNO LA CAPACITÀ DI PROTEGGERE UN DATO A PIACIMENTO. ESSI SI BASANO SULLA SCELTA DI ALCUNI VALORI AD

ESEMPIO:

1 0 0 0 1 1 0 1

$$X^7 + X^3 + X^2 + 1$$



DOVE LA SECONDA È UNA NOTAZIONE PIÙ COMPATTA CHE HA IL NOME AL  
CODICE: POLINOMIALE.

UTILIZZEREMO DUNQUE UN POLINOMIO (CIOÈ UNA SEQUENZA DI BIT) PER  
GENERARE  $n$  BIT DI CRC, CONSIDERANDO LA SEGUENTE RELAZIONE:

$$\frac{M(x) \cdot 2^n}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

DOVE  $M(x)$  È IL MESSAGGIO E  $M(x) \cdot 2^n$  È IL MESSAGGIO SPOSTATO A SX DI  $n$   
POSIZIONI,  $G(x)$  È IL POLINOMIO GENERATORE E  $Q(x)$ ,  $R(x)$  SONO IL QUOZIENTE  
E IL RESTO.

SOMMANDO  $R(x)/G(x)$  AD ENTRAMBI I MEMBRI OTTIENIAMO:

$$\frac{M(x) \cdot 2^n}{G(x)} + \frac{R(x)}{G(x)} = Q(x) + \underbrace{\frac{R(x)}{G(x)}}_A + \underbrace{\frac{R(x)}{G(x)}}_A$$

MA OSSERVATO CHE NELL'ARITMETICA MODULO 2  $A + A = 0$  AVREMO:

$$\frac{M(x) \cdot 2^n + R(x)}{G(x)} = Q(x)$$

[1]

DOVE  $M(x) \cdot 2^n + R(x)$  È IL MESSAGGIO A CUI ABBIAMO ATTACCATO IL RESTO.

SI NOTI CHE IN FASE DI CODIFICA SIAMO INTERESSATI A RICAVARE IL RESTO E  
IN FASE DI DECODIFICA ANCHE, IN QUANTO PRIMA DOBBIAMO ATTACCARLO AL MESSAGGIO  
E POI DOBBIAMO VERIFICARE, DIVIDENDO PER  $G(x)$ , CHE SIA 0, DATO CHE NELLA  
[1] COMPARE SOLO IL QUOZIENTE.

POICHÉ NON SIAMO INTERESSATI AL QUOZIENTE NEL SEGUENTE ESEMPIO NI LIMITERÒ  
A RICAVARE IL RESTO, CONSIDERANDO:

$$M(x) = 1110$$

$$G(x) = 101$$



PROCEDIAMO AGGIUNGENDO 2 ZERI ALLA DESTRA DEL MESSAGGIO (SE IL POLINOMIO GENERATORE HA 3 CIFRE IL RESTO NE AVRA' MASSIMO 2) E DIVIDENDO PER  $g(x)$ :

$$\begin{array}{r}
 \text{SPAZIO PER IL RESTO} \\
 1 \ 1 \ 1 \ 0 \ 0 \ 0 : 1 \ 0 \ 1 \\
 \underline{1 \ 0 \ 1} \\
 0 \ 1 \ 0 \ 0 \\
 \underline{1 \ 0 \ 1} \\
 0 \ 0 \ 1 \ 0 \ 0 \\
 \underline{1 \ 0 \ 1} \\
 0 \ 0 \ 1 \rightarrow \text{QUESTO SARÁ IL CRC}
 \end{array}$$

QUINDI IL MESSAGGIO CODIFICATO SARÁ:  $\overbrace{1 \ 1 \ 1 \ 0 \ 0}^{\text{MESSAGGIO}} \overbrace{0 \ 1}^{\text{CRC}}$

INVIAMO ORA A DECODIFICARLO, CIÓE DIVIDENDO PER IL POLINOMIO GENERATORE E VERIFICARE CHE IL RESTO SIA ZERO:

$$\begin{array}{r}
 1 \ 1 \ 1 \ 0 \ 0 \ 1 : 1 \ 0 \ 1 \\
 \underline{1 \ 0 \ 1} \\
 0 \ 1 \ 0 \ 0 \\
 \underline{1 \ 0 \ 1} \\
 0 \ 0 \ 1 \ 0 \ 1 \\
 \underline{1 \ 0 \ 1} \\
 0 \ 0 \ 0 \rightarrow \text{IL RESTO É ZERO}
 \end{array}$$

E QUESTO CONFERMA CHE CON BUONA PROBABILITÀ IL MESSAGGIO É STATO TRASMESSO CORRETTAMENTE.

L'UTILITÀ DI QUESTA CODIFICA STA NEL FATTO CHE POSSIAMO USARE LO STESSO HARDWARE SIA PER LA CODIFICA CHE PER LA DECODIFICA, BASTA CHE SIA IN GRADO DI CALCOLARE IL RESTO DI UNA DIVISIONE.



SONO STATI ANALIZZATI SIA IL NUMERO DI ERRORI SIA LA POSIZIONE IN CUI ESSI OCCORRONO PIÙ SPESSE E SONO STATI CREATI DEI POLINOMI GENERATORI STANDARD, AD ESEMPIO: CRC-8, CRC-16, CRC-32, CRC-CCITT.

## I CODICI DI HAMMING

QUESTA CODIFICA PREVEDE INIZIALMENTE LA DISTRIBUZIONE DEL MESSAGGIO LUNGO UNA SEQUENZA DI BIT IN MODO TALE CHE RIMANGANO VUOTE LE POSIZIONI CORRISPONDENTI ALLE POTENZE DI 2:

1 0 0 1 1 0 1

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|----|----|---|---|---|---|---|---|---|---|---|
| 1  | 0  | 0 |   | 1 | 1 | 0 |   | 1 |   |   |

LE POSIZIONI LIBERE VERRANNO OCCUPATE CON IL CRC, PER CALCOLARLO SOMMIAMO LE RAPPRESENTAZIONI IN BINARIO DELLE POSIZIONI IN CUI COMPAIONO BIT A 1, COSÌ:

|    |   |   |   |   |
|----|---|---|---|---|
| 3  | 0 | 0 | 1 | 1 |
| 6  | 0 | 1 | 1 | 0 |
| 7  | 0 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 |

1 0 0 1 } È IL CRC

DUNQUE IL MESSAGGIO CODIFICATO SARÀ:

1 0 0 1 1 1 0 0 1 0 1

PRIMA DI PROCEDERE CON LA DECODIFICA DOBBIAMO OSSERVARE CHE SE OM SOMMASSIMO I BIT A 1 VECCHI (3, 6, 7, 11) E CON IL CRC LA SOMMA SAREBBE ZERO, INATTI FARE  $3 + 6 + 7 + 11 + \text{CRC} = (3 + 6 + 7 + 11) + (3 + 6 + 7 + 11) = 0$  IN ARITMETICA MODULO 2.

E IL FATTO DI AVER POSIZIONATO IL CRC NELLE POTENZE DI 2 CI PERMETTE DI STABILIRE LA SEGUENTE REGOLA PER LA DECODIFICA: IL MESSAGGIO È STATO TRASMESSO CORRETTAMENTE SE LA SOMMA DELLE POSIZIONI DEI BIT A 1 VALE 0.



IN QUESTO CASO AVREMO :

|         |   |   |   |   |
|---------|---|---|---|---|
| 3       | 0 | 0 | 1 | 1 |
| 6       | 0 | 1 | 1 | 0 |
| 7       | 0 | 1 | 1 | 1 |
| 11      | 1 | 0 | 1 | 1 |
| 1       | 0 | 0 | 0 | 1 |
| 8       | 1 | 0 | 0 | 0 |
| 0 0 0 0 |   |   |   |   |

SE INVECE IL BIT IN POSIZIONE 6 FOSSE CAMBIATO DA 1 A 0 ALLORA NELLA SOMMA SAREBBE VENUTO A MANCARE IL CONTRIBUTO DI 0110, PORTANDO LA SOMMA FINALE AD ESSERE DIVERSA DA 0000 MA UGUALE PROPRIO A 0110:

|              |              |              |              |              |
|--------------|--------------|--------------|--------------|--------------|
| 3            | 0            | 0            | 1            | 1            |
| <del>6</del> | <del>0</del> | <del>1</del> | <del>1</del> | <del>0</del> |
| 7            | 0            | 1            | 1            | 1            |
| 11           | 1            | 0            | 1            | 1            |
| 1            | 0            | 0            | 0            | 1            |
| 8            | 1            | 0            | 0            | 0            |
| 0 1 1 0      |              |              |              |              |

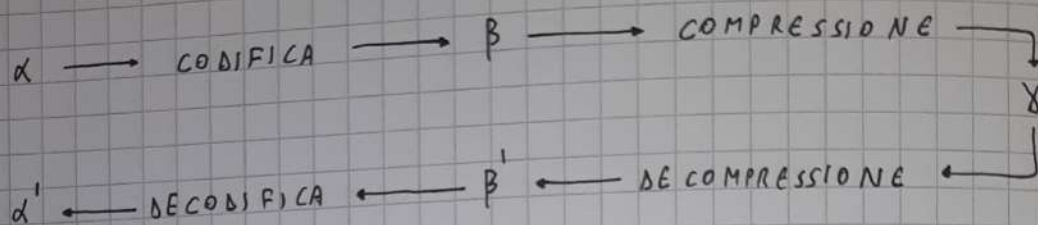
DUNQUE IN FASE DI DECODIFICA POSSIAMO CORRELARE IL VALORE DEL BIT IN POSIZIONE 0110 = 6. EVENTUALMENTE POTREMO TROVARE LA POSIZIONE DI UN BIT CHE È CAMBIATO DA 0 A 1 E DUNQUE È DI TROPPO NELLA SOMMA.

## I CODICI A CONVOLUZIONE

DURANTE LA TRASMISSIONE DI PACCHETTI DI INFORMAZIONI POSSONO ESSERCI MOLTI ERRORI E DUNQUE IL DATO VIAGGIA PER PAZI O INSIEME AI DATI SUCCESSIVI. NEI CASI CHE ABBIAMO VISTO NOI DATO + CRC VIAGGIANDO SEMPRE INSIEME, IN QUESTO CASO IL RISULTATO DELLA DECODIFICA È FUNZIONE ANCHE DEI DATI RECEVUTI IN PRECEDENZA.

# TECNICHE DI COMPRESSIONE

IL NOSTRO OBIETTIVO È QUELLO DI COMPRIERE UN DATO SENZA PERDERE IL SUO SIGNIFICATO, CONSIDERANDO IL SEGUENTE SCHEMA:



QUESTO È LO SCHEMA GENERALE CHE UN DATO ATTRAVERSA DURANTE LA TRASMISSIONE. ESISTONO 2 TIPI DI TECNICHE DI COMPRESSIONE:

- **LOSSLESS**, SONO TECNICHE DI COMPRESSIONE SENZA PERDITA DOVE

$$\text{sizeof}(\gamma) < \text{sizeof}(\beta), \quad \alpha = \alpha', \quad \beta = \beta'$$

SONO USATE PER COMPRIERE FILE DI TESTO, FILE ESECUIBILI, DATI NUMERICI CHE NON POSSONO SUBIRE MODIFICHE.

- **LOSSY**, SONO TECNICHE DI COMPRESSIONE CON PERDITA DOVE

$$\text{sizeof}(\gamma) \ll \text{sizeof}(\beta), \quad \alpha \approx \alpha', \quad \beta \neq \beta'$$

QUESTE SONO PRINCIPALMENTE USATE PER COMPRIERE FILE MULTIMEDIALI SFRUTTANDO LE LIMITAZIONI DEL NOSTRO SISTEMA PERCETTIVO, DI FATTO NON CI ACCORIAMO DELLA PERDITA DI INFORMAZIONI.

HANNO ANCHE LA CARATTERISTICA (LE TECNICHE LOSSY) DI POTER DECIDERE QUANTI DATI PERDERE.



## LE TECNICHE LOSS LESS

IN QUESTO CAPITOLO ANALIZZEREMO ALCUNE TECNICHE DI COMPRESSIONE SENZA PERDITA.

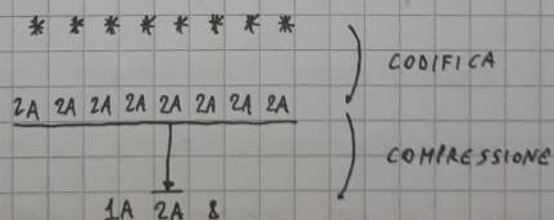
### RUN LENGTH ENCODING (RLE)

QUESTA TECNICA COMPRIME SEQUENZE DI BYTE UGUALI, AD ESEMPIO SE UN CARATTERE SI RIPETE 100 VOLTE NON LO SCRIVEREMO 100 VOLTE MA DIREMO CON UNA PARTICOLARE SEQUENZA DI CARATTERI CHE SI RIPETE 100 VOLTE.

QUESTA TECNICA RISULTA PARTICOLARMENTE ADATTA ALLA COMPRESSIONE DI FILE MULTIMEDIALI GIÀ COMPRESSI, CHE SPESSE PRESENTANO LUNGHE SEQUENZE DI BIT UGUALI.

#### CON CARATTERE RISERVATO

IN QUESTO CASO PER SAPERE QUANDO LEGERE I VALORI E QUANDO HO IL NUMERO DI RIPETIZIONI USIAMO UN CARATTERE RISERVATO, IL CODICE 1A (SUB, O SOSTITUITO), A SEGUIRE ALLUNGEREMO IL SIMBOLO DA RIPETERE E IL NUMERO DI RIPETIZIONI. AD ESEMPIO



QUESTO TUTTAVIA FUNZIONA PER STRINGHE DI TESTO, MA QUANDO CI TROVIAMO A LAVORARE CON ALTRI TIPI DI DATO È DIFFICILE CHE IL CARATTERE 1A SIA LIBERO PER POTER ESSERE USATO COME CODICE RISERVATO, DUNQUE È NECESSARIO INTRODURRE UN METODO CHE NON NE NECESSITI.

#### SENZA CARATTERE RISERVATO

IN QUESTO CASO IL COMPILATORE SI OWNI VOLTA CHE UN CARATTERE SI RIPETE

Vittorio Alzetta

3 o PIÙ VOLTE ARRIVARE AL TERMINE DELLA TRIPLETTA IL NUMERO DI RIPETIZIONI  
TOTALI DEL CARATTERE, AD ESEMPIO:

2A 2A 2A 2A 2A 2A 2A → 2A 2A 2A 7

2A 2A 2A 2A → 2A 2A 2A 4

2A 2A 2A → 2A 2A 2A 3

COME SI OSSERVA NEI PRIMI DUE CASI LA COMPRESSIONE HA PORTATO AD UNA RIDUZIONE  
DEL NUMERO DI CARATTERI MENTRE NELL'ULTIMO CASO SE N'È ARRIVATO UNO, QUESTO È  
IL PREZZO DA PAGARE PER NON AVERE UN CODICE RISERVATO.

## LA CODIFICA ENTROPICA

QUELLA DI CUI PARLEREMO ORA NON È UNA VERA E PROPRIA TECNICA DI COMPRESSIONE  
MA PIUTTOSTO UNA INTELLIGENTE TECNICA DI CODIFICA, CHE PREVEDE DI ASSOCIARE  
CODICI PIÙ CORTI (MENO BIT) A SIMBOLI PIÙ PROBABILI (CHE COMPaiono PIÙ  
SPESSE). QUESTA CODIFICA RIMANE UN'IDEA, ABBIAMO BISOGNO DI STRUMENTI PER METTERLA IN PRATICA.  
IN SOSTANZA ALLUNGHIAMO ~~LA~~ I CARATTERI POCHÉ VOLTE E LI ACCORCIAMO MOLTE  
VOLTE.

MEGLIO SE  
BASSA

SI CHIAMA ENTROPICA PERCHÉ GRAZIE ALL'ENTROPIA DEL SISTEMA I SIMBOLI NON  
COMPaiono TUTTI CON LA STESSA PROBABILITÀ E PERMETTONO DI RARE O DI CREARE IL DESIDERATO.  
CI POTREMMO PERÒ TROVARE IN UNA SITUAZIONE DEL GENERE:

e 1 0 1 0

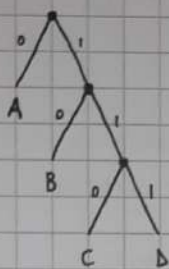
q 1 0 1 0 0 1 ...

NOTIAMO CHE IL CODICE DELLA LETTERA e È LO STESSO CHE COMPARE ALL'INIZIO  
DELLA q, QUESTO CREA DELLE AMBIGUITÀ DAL MOMENTO CHE IL DECOMPRESSORE  
NON SA SE DECODIFICARE LA e O LA q.



SI INTRODUCONO QUINDI I CODICI A PREFISSO, ESSI PREVEDONO CHE SE LA LETTERA  $e$  VIENE CODIFICATA CON 1010 ALLORA NESSUN ALTRO SIMBOLO POTRÀ INIZIARE CON LA SEQUENZA 1010.

PER CODIFICARE I SIMBOLI IN QUESTO MODO SI PUÒ CREARE UN ALBERO BINARIO COMPLETAMENTE SBILANCIATO:



$A = 0$

$B = 10$

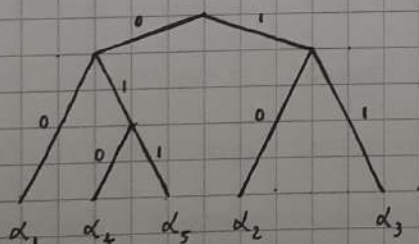
$C = 110$

$D = 111$

LA QUANTITÀ DI BIT NECESSARI ALLA RAPPRESENTAZIONE DEI SIMBOLI DEVE RAPPRESENTARE IL PIÙ REBELMENTE POSSIBILE LA DISTRIBUZIONE DELLA PROBABILITÀ.

IL CODICE A PREFISSO È LO STRUMENTO CHE PERMETTE L'IMPLEMENTAZIONE DELLA CODIFICA ENTROPICA CON SIMBOLI MEMORIZZATI CON SEQUENZE DI LUNGHEZZA VARIABILE SENZA PRESENTARE AMBIGUITÀ.

L'ALBERO COMPLETAMENTE SBILANCIATO NON RAPPRESENTA PERÒ LA SCELTA MIGLIORE, ESISTE INFATTI LA CODIFICA DI HUFFMAN CHE PREVEDE LA COSTRUZIONE DI UN ALBERO IN FUNZIONE DELLA FREQUENZA RELATIVA O RICORRENZA DEI DATI. NE È UN ESEMPIO:



$$P = \left\{ \begin{matrix} 1 & 2 & 3 & 4 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0,25 & 0,25 & 0,2 & 0,15 & 0,15 \end{matrix} \right\}$$

$$\alpha_1 = 00 \quad \alpha_2 = 10 \quad \alpha_3 = 11$$

$$\alpha_4 = 010 \quad \alpha_5 = 011$$

SI PUÒ OSSERVARE COME I SIMBOLI PIÙ FREQUENTI SIANO RAPPRESENTATI DA MENO BIT.

Vittorio Alzetta

UN ALTRO MODO, PIÙ SEMPLICE, PER CONFIGURARE A PREFISSO ED EVITARE DI AVERE UN ALBERO CON 256 FOGLIE È DETTO **MNP-5**, MA PRIMA DI DISCUTERNE APRIAMO UNA PICCOLA PARENTESI.

SPESSE CAPITA CHE ALCUNE LETTERE PRECEDANO QUASI SEMPRE ALTRE LETTERE, AD ESEMPIO LA "U" È PRECEDUTA DALLA "q" IN MOLTI CASI, È QUINDI POSSIBILE INTRODURRE DELLE COMPLICANZE CHE SI BASANO SU QUESTO FENOMENO DI PROBABILITÀ CONDIZIONATA.

TORNANDO ALLA CODIFICA MNP-5 ~~PRODOTTO~~ AVREMO CHE COMUNQUE AVrà UNA STRUTTURA CHE LA RENDE UN CODICE A PREFISSO (PUR NON UTILIZZANDO UN ALBERO) MA SOPRATTUTTO GESTISCE IN MODO COMPLETAMENTE AUTOMATICO LA PROBABILITÀ CHE QUINDI NON È PIÙ FISSA.

OGNI SIMBOLO È ASSOCIATO AD UN HEADER ED UN BODY COME NELLA SEGUENTE

TABELLA:

| BYTE IN INPUT | HEADER | BODY | OUTPUT |
|---------------|--------|------|--------|
| 00000000      | 000    | 0    | 0000   |
| 00000001      | 000    | 1    | 0001   |
| 00000010      | 001    | 0    | 0010   |
| 00000011      | 001    | 1    | 0011   |
| 00000100      | 010    | 00   | 01000  |
| 00000101      | 010    | 01   | 01001  |
| 00000110      | 010    | 10   | 01010  |
| 00000111      | 010    | 11   | 01011  |
| 00001000      | 011    | 000  | 011000 |

DOVE L'HEADER CI DICE QUANTE CIFRE DEL BODY DOBBIAMO LEGERE.

SI PRESENTANO 2 CASI SPECIALI, IL PRIMO SI HA DOVE L'HEADER VALE 000 E LEGGIAMO COMUNQUE 1 BIT (LO SI FA PER AVERE TUTTE LE COMBINAZIONI POSSIBILI), IL SECONDO SI HA ALLA FINE QUANDO L'HEADER VALE 111 E IL BODY NON È 111111 MA 1111110 (LO SI FA PER NON AVERE UNA SEQUENZA DI 1 CHE GENERALMENTE È ASSOCIATA AL -1).



DURANTE LA FASE DI CODIFICA SI PARTE CON UNA TABELLA COME QUELLA CHE HO RAPPRESENTATO PRIMA E AD OGNI SIMBOLO IN ~~INPUT~~ INPUT SI ALLIQUONA UN PRECISO CONTATORE, AD ESEMPIO:

00000001  $\longrightarrow$   $V[1]++$   $\longrightarrow$  0001  
 00000100  $\longrightarrow$   $V[4]++$   $\longrightarrow$  01000

VIENE QUINDI SEGUITO UNO SCHEMA DEL TIPO:



LA PARTICOLARITÀ DI QUESTA CODIFICA ARRIVA QUANDO IL CONTATORE RELATIVO ALLA 1<sup>a</sup> RIGA SUPERA DI UNA CERTA SOLLIA IL CONTATORE RELATIVO ALLA 1<sup>a</sup> RIGA, PERCHÉ A QUEL PUNTO VERRANNO INVERTITI I CONTATORI (CIOÈ SCAMBIATI I LORO VALORI) E I RELATIVI BYTE IN INPUT. RIPRENENDO L'ESEMPIO DI PRIMA AVREMO:

00000100  $\longrightarrow$   $V[1]++$   $\longrightarrow$  0001  
 00000001  $\longrightarrow$   $V[4]++$   $\longrightarrow$  01000

QUESTO AVVIENE SIA IN CODIFICA CHE IN DECODIFICA, DATO CHE IN QUEST'ULTIMO PASSAGGIO SI ALLIQUONA NO I CONTATORI ALLO STESSO MODO

PER ADESSO QUINDI LA TECNICA DI CODIFICA E COMPRESSIONE PARVEDE L'IMPIEGHO DI UNA ~~MA~~ VETTORE DI 256 CONTATORI, MA POTREMMO SPINGerci PIÙ IN LÁ CONSIDERANDO UN VETTORE DI VETTORI DI 256 CONTATORI PER INTRODURRE PRATICAMENTE IL CONCETTO DI PROBABILITÀ CONDIZIONATA.

AD OGNI SIMBOLO IN INPUT ALLIQUONEREMO IL SEGUENTE CONTATORE:

$V[\text{SIMBOLO\_PRECEDENTE}][\text{SIMBOLO\_CORRENTE}]++$

Vittorio Alzetta

UN'ALTRA IMPORTANTE CODIFICA È CHIAMATA CODIFICA A DIZIONARIO, dove CON ALCUNO SI INTENDE UNA TABELLA IN CUI COMPaiono delle SEQUENZE DI BIT.

FACCIO OSSERVARE PRELIMINARIAMENTE CHE L'ALGORITMO SU CUI SI BASA È DETTO **LZW** (DA ZIV-LEMPER-WELCH) E VIENE UTILIZZATO DA MOLTI SOFTWARE COME WINZIP.

QUESTA CODIFICA RAPPRESENTERÀ I SIMBOLI TUTTI CON LA STESSA LUNGHEZZA MA VIA VIA CHE L'ALGORITMO LAVORERÀ LE SEQUENZE DI BIT CHE RAPPRESENTANO SIMBOLI INIZIERANNO A RAPPRESENTARE SEQUENZE SEMPRE MAGGIORI DI SIMBOLI.

OSSERVIAMO CHE SE VOGLIAMO RAPPRESENTARE PIÙ SIMBOLI CON UN CODICE ESSO DOVRÀ ESSERE COMPOSTO DA PIÙ DI 8 BIT, ALTIMENTI SI OCCUPEREBBERO TUTTE LE POSIZIONI DEL DIZIONARIO CON I SOLITI 256 SIMBOLI E NON CI SAREBBE SPALIO PER AGGIUNGERE CODICI ASSOCIATI A COMBINAZIONI DI PIÙ SIMBOLI.

UNA SCELTA CLASSICA SONO I 12 BIT, AVREMO DUNQUE UNA TABELLA DEL TIPO:

|                                                      |                 |     |
|------------------------------------------------------|-----------------|-----|
| CHIAMEREMO STRINGHE<br>GLI ELEMENTI DELLA<br>TABELLA | 0               | 0   |
|                                                      | 1               | 1   |
|                                                      | 2               | 2   |
|                                                      | 3               | 3   |
|                                                      | ⋮               |     |
|                                                      | 255             | 255 |
|                                                      | 256             |     |
|                                                      | ⋮               |     |
|                                                      | 2 <sup>12</sup> |     |
|                                                      |                 |     |
|                                                      |                 |     |
|                                                      |                 |     |

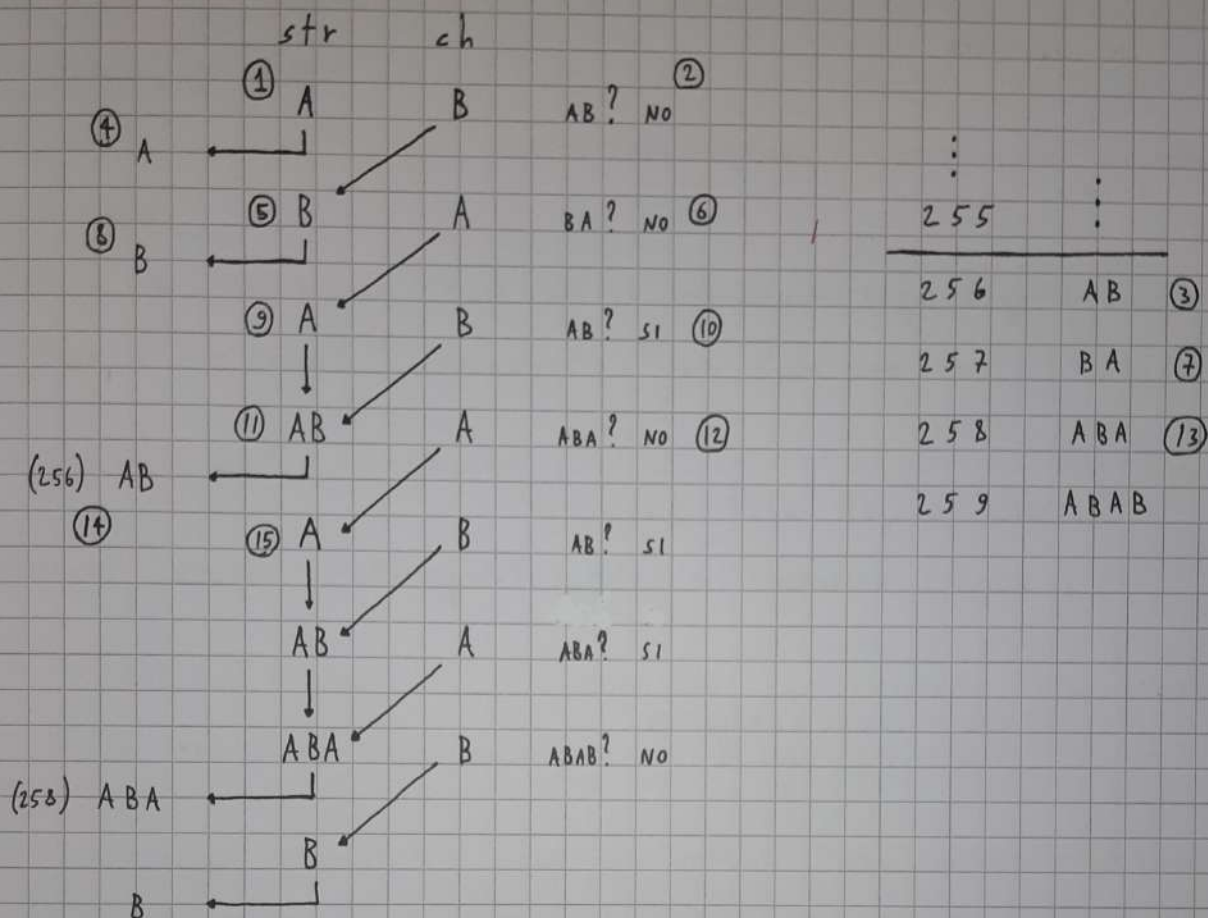
QUESTA PARTE VIENE COSTRUITA  
DALL'ALGORITMO E INIZIALMENTE  
È VUOTA.

DOBBIAMO QUINDI CREARE UN OBIETTO IN GRADO DI IMPARARE (COME LA CODIFICA MNP-5) QUALI SONO I SIMBOLI PIÙ PROBABILI E QUALI SONO LE COMBINAZIONI DI SIMBOLI PIÙ FREQUENTI.

VEDIAMO UN ESEMPIO CHE MOSTRA IN DETTAGLIO COME AVISCE L'ALGORITMO LZW PER AGGIUNGERE ELEMENTI AL DIZIONARIO.



SUPPONIAMO DI RICEVERE IN INPUT LA STRINGA ABABABAB E DI  
DOVERLA CODIFICARE.



SI INIZIA PONENDO LA STRINGA PARI AL PRIMO SIMBOLO DEL MESSAGGIO E  
IL CARATTERE PARI AL SECONDO (1), POI CI CHIEDIAMO SE STRINGA + CARATTERE  
FANNO PARTE DELLA TABELLA (2), DATO CHE LA RISPOSTA È NO LI AGGIUNGIAMO NEL  
PRIMO POSTO DISPONIBILE NELLA TABELLA (3). ORA BUTTIAMO FUORI IL SIMBOLO  
MEMORIZZATO SULLA STRINGA (4) E LO SOSTITUIAMO CON IL CARATTERE (5) (POI  
AL POSTO DEL CARATTERE METTEREMO IL PROSSIMO SIMBOLO DEL MESSAGGIO) E CI  
CHIEDIAMO SE STRINGA + CARATTERE C'È NELLA TABELLA (6). LA RISPOSTA È NO E  
DUNQUE AGGIUNGIAMO AB ALLA TABELLA (7), POI CODIFICHIAMO B E LO BUTTIAMO  
FUORI (8), METTIAMO IL CARATTERE NELLA STRINGA (9) E CI CHIEDIAMO SE AB È  
PRESENTI NELLA TABELLA (10). LA RISPOSTA ORA È SÌ E DUNQUE NON BUTTIAMO  
FUORI LA STRINGA MA LA UNIAMO AL CARATTERE E LA PORTIAMO LÙ (11). ORA CI

CI CHIEDIAMO SE ABA C'È NELLA TABELLA (12), DATO CHE NON C'È LO  
 ANNUNCIAMO (13) E SOVRASCRIVIAMO LA STRINGA CON IL CARATTERE DOPO AVERLA  
 CODIFICATA (14) (15), SI PROCEDE COSÌ FINO ALLA FINE QUANDO NON SONO RIMASTI  
 SIMBOLI DA CODIFICARE.

L'ALGORITMO IN FORMA TESTUALE SI PRESENTA COME:

```

STRING = get input character
while (there are still input characters) do
{
    CHARACTER = get input character
    if (STRING + CHARACTER is in the string table)
    {
        STRING = STRING + CHARACTER
    } else
    {
        output the code for STRING
        add STRING + CHARACTER to the string table
        STRING = CHARACTER
    }
}
output the code for STRING
  
```

NEL NOSTRO ESEMPIO IL MESSAGGIO ORIGINALE E CODIFICATO È:

ABABABAB → A B AB ABA B → A-B-256-258-B

IL DECODIFICATORE FA LA STESSA COSA, INIZIALMENTE SI CHIEDE SE AB C'È NELLA  
 TABELLA, E ALLA RISPOSTA NEGATIVA LO ANNUNCE ALLA TABELLA. QUESTO LO PERMETTE  
 DI INTERPRETARE CORRETTAMENTE IL 256 CHE ARRIVA SUBITO DOPO.



## LE TECNICHE LOSSY

QUI VEDREMO ALCUNE TECNICHE DI COMPRESSIONE CON PERDITA, INFATTI SE CI LIMITASSIMO ALLE TECNICHE LOSSLESS LA COMPRESSIONE DI FILE MULTIMEDIALI RISULTEREBBE PRATICAMENTE INUTILE. QUESTI TIPI DI FILE PRESENTANO SPESSE SEQUENZE DI BIT SIMILI MA DIVERSI, IL CHE RENDE DIFFICILE E POCO EFFICIENTE LA COMPRESSIONE SENZA PERDITA.

VEDREMO IN PARTICOLARE COME VENGONO CODIFICATE E COMPRESSE LE IMMAGINI, I VIDEO E GLI AUDIO.

## LA CODIFICA DELLE IMMAGINI

ANZITUTTO OCCORRE DISTINGUERE 2 TIPI DI IMMAGINI:

- IMMAGINI **VEETONALI**, UTILIZZATE PER LA MECCANICA, L'ARCHITETTURA O IN GENERALE LA RAPPRESENTAZIONE DI DISegni GEOMETRICI
- IMMAGINI **BITMAP**, SONO LE IMMAGINI CHE SCATTIAMO QUOTIDIANAMENTE, HANNO SFUMATURE E SONO MOLTO IRREGOLARI.

ABBIAMO QUINDE BISOGNO DI 2 METODI DIVERSI PER CODIFICARE QUESTI 2 TIPI DI IMMAGINI DIVERSE.

### CODIFICA IMMAGINI VETTONALI

PER CODIFICARE UN'IMMAGINE VETTONALE SARÁ SUFFICIENTE ESPRIMERE MATEMATICAMENTE L'Oggetto GEOMETRICO CHE VERRÁ STAMPATO, QUINDE SI TRATTERÁ DI SEMPLICI FILE ASCII. NE É UN ESEMPIO:



circle 98 66 50

polyline 0 48 88 152 88 48 0 48

TIPICAMENTE IL FORMATO É DXF (DRAWING EXCHANGE FORMAT).

CODIFICA IMMAGINI BITMAP

QUANDO VIENE SCATTATA UN'IMMAGINE I PUNTI LUMINOSI DELL'IMMAGINE REALE VENGONO PROIETTATI SULLA MACCHINA, CHE CAMPIONA L'IMMAGINE SUDDIVIDENDOLA IN UN INSIEME DI PUNTI DETTI **PIXEL** (~~essa~~ PICTURE ELEMENT). SE AD ESEMPIO SCATTIAMO UN'IMMAGINE CON UNA FOTOCAMERA DA 10 MPX ESSA AVRA 10 MILIONI DI PIXEL.

NEL CASO DI VERE IMMAGINI IN BIANCO E NERO LA CODIFICA È INUTILE INTUITIVA, DATO CHE OGNI PIXEL DEVE ESSERE DESCRITTO DA UN CERTO NUMERO DI BIT E DOBBIAMO RAPPRESENTARE SOLO IL BIANCO E IL NERO POSSIAMO VEDERE CHE CI SERVIREMO DI UN SOLO BIT (0 = BIANCO, 1 = NERO):



|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

IN ALTERNATIVA SI PUÒ CODIFICARE CIASCUN PIXEL CON 8 BIT, IN MODO DA AVERE 256 POSSIBILI LIVELLI DI GRIGIO.

È CHIARO CHE UN'IMMAGINE BITMAP HA UNA **RSOLUZIONE** LIMITATA, IN QUANTO OGNI PIXEL È DI PER SÉ UN'APPROSSIMAZIONE DI UNA PICCOLA AREA DELL'IMMAGINE REALE.

NEL MOMENTO IN CUI VUOLIAMO PASSARE DA UNA RAPPRESENTAZIONE CON SCALA DI GRIGI AD UNA A COLORI DOBBIAMO CONSIDERARE I 2 SEGUENTI METODI DI SINTESI:

- **SINTESI SOTTATTIVA**, CONSISTE NELLA STESSA OPERAZIONE CHE SI SVOLGE SOVRAPPONENDO CANE COLORATE SEMITRASPARENTI, CIOÈ PREVEDE DI FORNIRE ~~DA~~ LUNGHEZZE D'ONDA ALLA LUCE BIANCA. MOLTE STAMPANTI ADOPERANO QUESTO METODO MESCOLANDO TRA LORO INCHIOSTRO MAGENTA, CIANO E GIALLO (SONO DETTE A GETTO D'INCHIOSTRO).



- SINTESI ADDITTIVA, IN QUESTO CASO VIENE PRESA LUCE DI PIÙ COLORI PER ESSERE MESCOLOTA E GENERARE LUCE DI UN ALTRO COLORE. I DISPLAY, AD ESEMPIO, HANNO 3 SORGENTI LUMINOSE PER PIXEL E REGOLANDO L'INTENSITÀ DELLE 3 SORGENTI LUMINOSE (RGB - RED GREEN BLUE) GENERA COLORI DIVERSI.

NELLE IMMAGINI BITMAP, DUNQUE, AD OGNI PIXEL È ASSOCIATA UNA TRIPLETTA DI VALORI, CHE RAPPRESENTANO L'INTENSITÀ DEI 3 COLORI PRIMARI.

È FACILE VEDERCI CHE IL NUMERO DI COLORI RAPPRESENTABILI, DAL MOMENTO CHE OGNI COLORE PRIMARIO È RAPPRESENTATO DA 1 BYTE, SARÀ:

$$\# \text{ COLORI} = 2^8 \cdot 2^8 \cdot 2^8 = 2^{24} \approx 16 \text{ MILIONI}$$

DUNQUE CON QUESTA CODIFICA DEI PIXEL, CHE CHIAMEREMO TRUE COLOR, UN'IMMAGINE A 10 Mpx AVRÀ UN PESO DI 30 MB ( $\frac{3B}{px} \cdot 10 Mpx$ ), IL CHE RENDE NECESSARIO LO SVILUPPO DI TECNICHE DI COMPRESSIONE.

PRIMA DI PASSARE ALLA TECNICA DI COMPRESSIONE JPEG SI OSSERVI CHE ANCHE IN UN'IMMAGINE DA 10 Mpx (MOLTO GRANDE) NON SIAMO IN GRADO DI UTILIZZARE TUTTI I COLORI (SONO 16 MILIONI), DUNQUE SI È DECISO DI CREARE UNA PALETTE DEI COLORI COMPOSTA DALLE TRIPLETTES DI COLORI DI OGNI PIXEL.

SE QUINDI 2 PIXEL HANNO LO STESSO COLORE FAREMO RIFERIMENTO ALLA STESSA RIGA DELLA TABELLA, INSERENDO UN SOLO NUMERO IN OGNI PIXEL (L'INDICE DELLA TABELLA).

## LA COMPRESSIONE JPEG

IN QUESTO PARAGRAFO VEDREMO UNA TECNICA DI COMPRESSIONE DELLE IMMAGINI SVILUPPATA DAL JOINT PHOTOGRAPHIC EXPERTS GROUP CHE SI BASA SULLA DCT (DISCRETE COSINE TRANSFORM).

PRIMA DI ENTRARE NEI DETTAGLI OCCORRE OSSERVARE CHE L'OCCHIO UMANO NON È MOLTO SENSIBILE ALLE VARIAZIONI NELLE CARATTERISTICHE DEI PIXEL.

QUANDO QUESTE SONO MOLTO RAVVICINATE NELLO SPAZIO. UNO STRUMENTO CHE CI PERMETTE DI CONVERTIRE LA MATRICE DI PIXEL IN UNA MATRICE DI FREQUENZE SPAZIALI (INDICANO LA RAPIDITÀ DELLA VARIAZIONE DELLE CARATTERISTICHE NELLE 2 DIREZIONI) È PROPRIO LA DCT.

OTTENUTA LA MATRICE DI FREQUENZE SPAZIALI RIUSCIAMO FACILMENTE AD INDIVIDUARE LE ZONE A FREQUENZE SPAZIALI ALTE E BASSE. DATO CHE NON SIAMO SENSIBILI A QUELLE ALTE APPLICHIAMO UN PROCESSO DI QUANTIZZAZIONE DOVE ANDIAMO A DIVIDERE LA MATRICE DI FREQUENZE PER UNA DETERMINATA QUANTIZZAZIONE PER POI TRONCARE I RISULTATI:

SUPPONIAMO DI AVERE UNA MATRICE CON FREQUENZE ALTE IN BASSO A DX E BASSE IN ALTO A SX:

|                 |                |    |    |    |    |    |
|-----------------|----------------|----|----|----|----|----|
| FREQUENZE BASSE | 16             | 25 | 30 | 32 | 40 | 45 |
|                 | 22             | 28 | 27 | 30 | 45 | 50 |
|                 | 23             | 20 | 29 | 39 | 42 | 51 |
|                 | 24             | 32 | 30 | 40 | 50 | 55 |
|                 | 30             | 35 | 49 | 42 | 52 | 62 |
|                 | 32             | 40 | 52 | 63 | 67 | 69 |
|                 | FREQUENZE ALTE |    |    |    |    |    |

LA NOSTRA MATRICE DI QUANTIZZAZIONE AVRÀ QUINDI VALORI ALTI IN BASSO A DX E BASSI IN ALTO A SX, IN MODO TALE DA TRONCARE EVENTUALMENTE A 0 I VALORI NELLA ZONA ROSSA (VERRANNO APPROSSIMATI MOLTO) E LASCIARE INVARIATI O QASI I VALORI NELLA ZONA VERDE.

IN GENERALE IL PROCESSO DI CODIFICA CONSISTE DEI SEGUENTI PASSAGGI:

- ① L'IMMAGINE VIENE SEPARATA IN 3 COMPONENTI, UNA DI LUMINANZA (Y) E 2 DI CROMINANZA (U e V). DOVE Y RAPPRESENTA UNA MEDIA PONDERATA DEI VALORI R, G e B E U, V SI CALCOLANO COME:

$$Y = 0,299 R + 0,587 G + 0,114 B$$

$$U = (B - Y) \cdot 0,565$$

$$V = (R - Y) \cdot 0,713$$



QUESTO VIENE FATTO PERCHÉ L'OCCHIO È PIÙ SENSIBILE ALLE VARIAZIONI DI LUMINANZA (CHE COMPAREMO MENO) INVECE CHE A QUELLE DI COLORE (CHE COMPAREMO DI PIÙ).

② VENLONO CREATE LE MATRICI DELLE FREQUENZE SPAZIALI (APPLICANDO LA DCT) DI TUTTI I BLOCCHI DI PIXEL  $8 \times 8$  CONTENUTI NELLE IMMAGINI.

③ LA MATRICE DI FREQUENZE SPAZIALI VIENE QUANTIZZATA CON UN'APPOSITA MATRICE DI QUANTIZZAZIONE.

④ VIENE SCANNERIZZATA LA MATRICE QUANTIZZATA A ZIG-ZAG, CERCANDO DI MASSIMIZZARE LE SERIENZE DI BIT UGUALI.

⑤ VENLONO COMPRESSE LE SERIENZE DI BIT CON LE TECNICHE DI COMPRESSIONE SENZA PERDITA, IN PARTICOLARE RLE E CODIFICA DI HUFFMAN.

### ESEMPIO:

NEL SEGUENTE ESEMPIO CONSIDEREREMO UNA IMMAGINE RAPPRESENTATA IN SCALA DI GRIGI, DUNQUE NON SEPAREREMO LE COMPONENTI DI LUMINANZA E CROMINANZA.

|     |     |     |     |
|-----|-----|-----|-----|
| 136 | 142 | 151 | 214 |
| 134 | 142 | 202 | 211 |
| 131 | 187 | 203 | 208 |
| 165 | 185 | 197 | 210 |

SI SOTTRA LA COSTANTE 128

|    |    |    |    |
|----|----|----|----|
| 8  | 14 | 23 | 86 |
| 6  | 14 | 74 | 83 |
| 3  | 59 | 75 | 80 |
| 77 | 57 | 69 | 82 |

APPLICHIAMO LA TRASFORMATA DI SCAEFA

MATRICE  
DI QUANTIZZAZIONE

|     |     |     |     |
|-----|-----|-----|-----|
| 16  | 11  | 10  | 77  |
| 12  | 12  | 113 | 92  |
| 14  | 121 | 120 | 101 |
| 112 | 100 | 103 | 99  |

DIVISIONE

|        |        |      |      |
|--------|--------|------|------|
| -327,5 | -215,2 | 16,1 | -0,7 |
| 16,1   | 3,4    | -0,1 | 2,1  |
| 2,9    | -0,5   | 0,9  | 2,8  |
| 0,2    | 1,9    | -0,6 | 0,7  |

|     |     |   |   |
|-----|-----|---|---|
| -20 | -20 | 2 | 0 |
| 2   | 0   | 0 | 0 |
| 0   | 0   | 0 | 0 |
| 0   | 0   | 0 | 0 |

Vittorio Alzetta

SI NOTI CHE IN CORRISPONDENZA DEI VALORI PIÙ ALTI NELLA MATRICE DELLE FREQUENZE SPAZIALI ABBIAMO I TENORI PIÙ ALTI ANCHE NELLA MATRICE DI QUANTIZZAZIONE IDEM PER I PIÙ BASSI (SONO QUELLI NEGATIVI NELLA MATRICE DELLE FREQUENZE SPAZIALI).

IL RISULTATO È UNA MATRICE CON UNA LUNGA SERVENZA DI EEN (SE LETTA A RIGHE) IL CHE LA RENDE FACILMENTE COMPRESSIBILE.

SI OSSERVI CHE IL PROCESSO DI CODIFICA È INTERAMENTE INVERTIBILE, INFATTI LA TRASFORMATA È UNA SEMPLICE FUNZIONE MATEMATICA, LA MATRICE DI QUANTIZZAZIONE PUÒ ESSERE Moltiplicata PER QUELLA DELLE FREQUENZE SPAZIALI E I BIT CODIFICATI CON HUFFMAN E LA RLE POSSONO ESSERE DECODIFICATI.

LA CODIFICA JPEG NON SI ADATTA BENE ALLA COMPRESSIONE DI IMMAGINI NON FOTOREALISTICHE, INFATTI ALLA BASE DELLA DEFINIZIONE DELLA MATRICE DI QUANTIZZAZIONE CI SONO LE SEGNATURE (PRESENTI IN GENERALE IN IMMAGINI FOTOREALISTICHE)

## LA CODIFICA DEI VIDEO

PER ILLUDERE IL CERVELLO UMANO DI STAR GUARDANDO UN MOVIMENTO ANIMATO UNA SERIE DI FOTO È SUFFICIENTE MOSTRARLE IN RAPIDA SUCCESSIONE.

AL CINEMA TIPICAMENTE I FILM VENGONO RIPRODOTTI A 24 FRAME/S, MENTRE NELLE VECCHIE TELEVISIONI ANALOGICHE EUROPEE E AMERICANE RISPETTIVAMENTE A 25 FRAME/S E 30 FRAME/S (QUESTO PER VIA DELLE FREQUENZE DELLA CORRENTE, 50 HZ IN EUROPA E 60 HZ IN AMERICA).

IN REALTÀ L'OCCHIO È CONTINUAMENTE IN GRADO DI PERCEPIRE UNO SFARFALCIO TRA UN FRAME E L'ALTRO, DUNQUE NEL TEMPO SONO STATI AUMENTATI I FRAME-RATE PER EVITARE QUESTO FENOMENO.

SAPENDO CHE OGNI FOTOGRAFIA DEVE ESSERE CODIFICATA CON LE TECNICHE VISTE PER LE IMMAGINI BITMAP RISULTA CHE PER GUARDARE UN FILMATO IN QUALITÀ



FULL-HD A 30 fps CON 16 bit PER PIXEL (NEANCHE IN TRUE COLOR A 3 BYTE!) SONO E NECESSARIO TRASMETTERE:

$$(1920 \cdot 1080)_{px} \cdot 2B \cdot 30 \frac{s}{s} = 125 \frac{MB}{s}$$

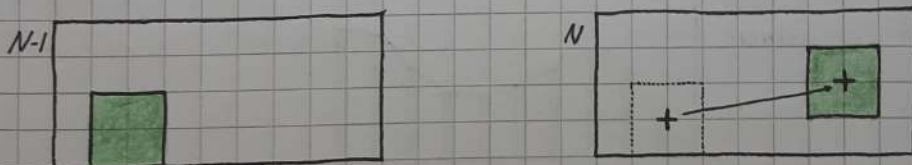
SI RENDE QUINDI OBBLIGATO SVILUPPARE DELLE TECNICHE DI COMPRESSIONE

## LA COMPRESSIONE MPEG

QUESTA TECNICA DI COMPRESSIONE È STATA SVILUPPATA DAL MOTION PICTURES EXPERTS GROUP E UNA DELLE SUE TANTE VERSIONI È ANCHE UTILIZZATA PER COMPRIERE L'AUDIO.

ANZITUTTO FACCIO OSSERVARE CHE IN UN FILM SOLITAMENTE 2 FOTOCAMERE CONSECUTIVI SONO MOLTO SIMILI, A MENO DI CAMBI DI SCENA, EBBENE QUESTA TECNICA PREVEDE DI PRENDERE UN BLOCCO DA  $16 \times 16$  PIXEL NEL FOTOCAMERA N E ANDARE A CERCARE IL BLOCCO  $16 \times 16$  PIXEL PIÙ SIMILE NEL FOTOCAMERA N-1 (CALCOLANDO ANCHE L'ERRORE TRA UN BLOCCO E L'ALTRO, SI TRATTA DI UNA SEMPLICE DIFFERENZA BIT A BIT).

DUNQUE PER IL FOTOCAMERA N VERRÀ MEMORIZZATO IL VETTORE SPOSTAMENTO DEL BLOCCO  $16 \times 16$  E IL BLOCCO DI ERRORE, COME ILLUSTRATO QUI SOTTO:



FACENDO COSÌ SI STANNO CONCATENANDO TRA LORO TUTTI I FOTOCAMERE, IL CHE RENDE IMPOSSIBILE PARTIRE DA UN PUNTO NEL MECCO (BISOLVEREBBE ASPETTARE CHE I FOTOCAMERE LE INFORMAZIONI DAL PRECEDENTE E NECHIEDEREBBE MOLTO TEMPO), QUINDI OGNI N FOTOCAMERA VENGONO INSERITI DEI FOTOCAMERE DI TIPO I (INTRA-PICTURES) CHE NON HANNO PRECEDENTI, NELLA CODIFICA, AD ALTRI FOTOCAMERE.

TAM I FOTOLAMMI NORMALI SE NE INSEGNANO ANCHE DI TIPO B (BI-DIRECTIONAL PREDICTER) CHE HANNO UNA CODIFICA BASATA SIA SUL FOTOLAMMA PRECEDENTE SIA SUL SUCCESSIVO, PER DARE LA POSSIBILITÀ DI ANDARE INDIETRO.

## LA CODIFICA DEGLI AUDIO

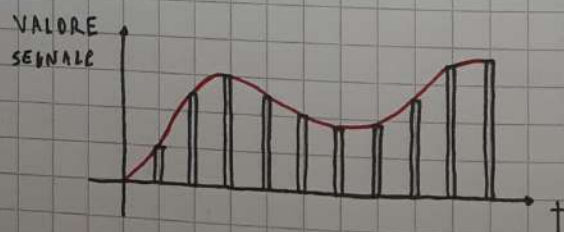
ANZITUTTO CAPIAMO COME IL SUONO VIENE REGISTRATO: LE ONDE SONORE CHE NOI EMIAMMO SONO PERCEPITE DA UNA MEMBRANA CHE INIZIA A VIBRARE INSIEME AD UNA BOBINA AD ESSA COLLEGATA, LA BOBINA POI GENERA UNA CORRENTE INDOTTA CHE VIENE ALLEVATA DAL COMPUTER O DAL REGISTRATORE.

IL RISULTATO È IL SEGNALE AUDIO ANALOGICO, CHE È CONTINUO NEL TEMPO E NELLE AMPIEZZE.

PER CODIFICARE IL SEGNALE AUDIO DIGITALMENTE SI RENDE NECESSARIO DISCRETIZZARE IL SEGNALE, PER QUANTO RIGUARDA LA CONTINUITÀ NEL TEMPO EFFETTUEREMO UN CAMPIONAMENTO, PER LA CONTINUITÀ NELLE AMPIEZZE EFFETTUEREMO UNA QUANTIZZAZIONE.

### CAMPIONAMENTO:

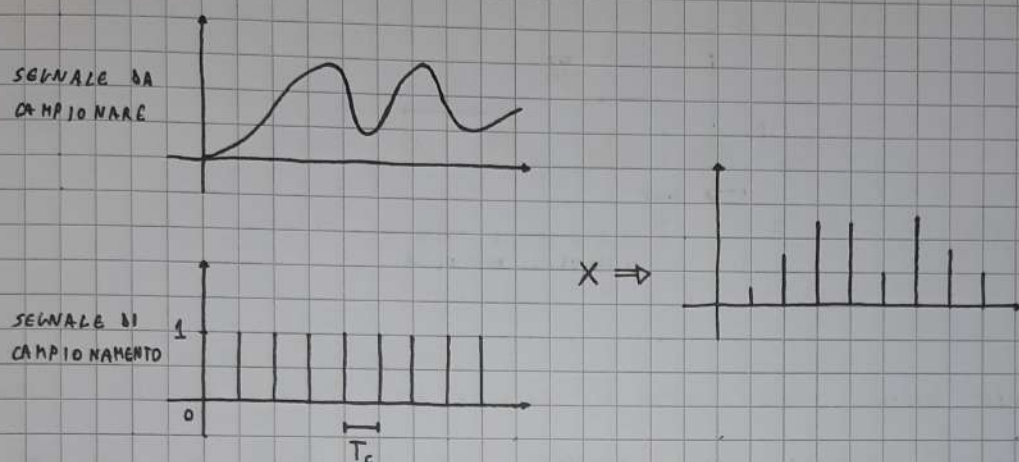
IN QUESTA FASE ANDEREMO A CAMPIONARE IL SEGNALE AUDIO IN ALCUNI PUNTI, CERCANDO DI OTTENERE UN RISULTATO DEL TIPO:



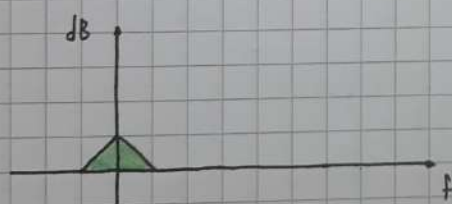
DOVE IN ROSSO ABBIAMO IL SEGNALE AUDIO CONTINUO E I RETTANGOLI RAPPRESENTANO IL SEGNALE QUANDO CAMPIONATO. IL NOSTRO OBIETTIVO È QUELLO DI CAMPIONARE IL SEGNALE SENZA PERDERE LE INFORMAZIONI PIÙ IMPORTANTI (CIÒ È QUELLE A CUI SIAMO SENSIBILI).



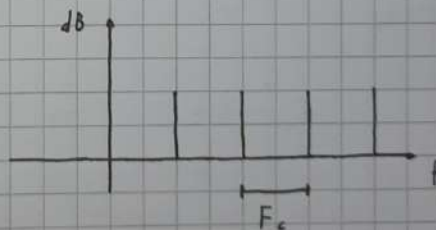
ANZITUTTO BISOLNA OSSERVARE CHE NEL DOMINIO DEL TEMPO, L'OPERAZIONE CHE DOVREMO EFFETTUARE PER PASSARE DA SEGNALE DA CAMPIONARE A SEGNALE CAMPIONATO SARÁ UNA MOLTIPLICAZIONE PER UN SEGNALE DI CAMPIONAMENTO (CHE VARÁ 0 OVUNQUE TRANNE NEI PUNTI DI CAMPIONAMENTO IN CUI VARÁ 1):



CI RIMANE SOLO DA CAPIRE CHE VALORE DARE AL TEMPO DI CAMPIONAMENTO  $T_c$ , PER DETERMINARLO PASSIAMO AL DOMINIO DELLE FREQUENZE APPLICANDO LA TRASFORMATA DI FOURIER AI 2 GRAFICI CHE SONO:

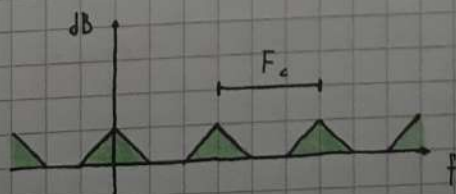


SPETTRO DI POTENZA DEL SEGNALE DA CAMPIONARE



SPETTRO DI POTENZA DEL SEGNALE DI CAMPIONAMENTO

IN QUESTO CASO, PER DETERMINARE LO SPETTRO DI FREQUENZA DEL SEGNALE CAMPIONATO NON DOVREMO MOLTIPLICARE I 2 GRAFICI MA EFFETTUARE UNA CONVOLUZIONE, IL RISULTATO SARÁ:







QUANTIZZAZIONE.

IL CAMPIONAMENTO PRODUCE DEI NUMERI REALI, CHE ANDREBBERO ESPRESSI CON UN NUMERO INFINITO DI BIT. LIMITANDO AD UN NUMERO FISSO I BIT CHE VERRANNO UTILIZZATI PER ESPRIMERE I NUMERI REALI SI INTRODUCE UN ERRORE CHE È DETTO **RUMORE DI QUANTIZZAZIONE.**

ESEMPI:

- LA VOCE NELLA TELEFONIA DIGITALE HA BANDA 3400 Hz, FREQUENZA DI CAMPIONAMENTO 8 KHz E QUANTIZZA A 8 bit/campione.

DUINQUE OGNI SECONDO NECESSITERÀ DI:

$$\# b = 8'000 \frac{\text{CAMPIONI}}{s} \cdot 8 \frac{\text{bit}}{\text{CAMPIONE}} = 64 \frac{\text{Kb}}{s}$$

E DUINQUE 1 min DI REGISTRAZIONE OCCUPERÀ 480 Kb.

- LA MUSICA NEI CD AUDIO HA BANDA 20 KHz, FREQUENZA DI CAMPIONAMENTO 44,1 KHz E QUANTIZZA A 16 bit/campione (x2 CANALI, DATO CHE È STEREO).

DUINQUE OGNI SECONDO NECESSITERÀ DI:

$$\# b = 44'100 \frac{\text{CAMPIONI}}{s} \cdot 16 \frac{b}{\text{CAMPIONE}} \cdot 2 = 1,411 \frac{\text{Mb}}{s}$$

E DUINQUE 1 min DI REGISTRAZIONE OCCUPERÀ  $1,411 \frac{\text{Mb}}{s} \cdot 60 = 10,6 \text{ MB}$

LA COMPRESSIONE DEGLI AUDIO

LA COMPRESSIONE DEGLI AUDIO SFRUTTA DI NUOVO LE LIMITAZIONI DELL'ORECCHIO UMANO. IN PARTICOLARE LA TELEFONIA DIGITALE APPLICA UN RAPPORTO DI

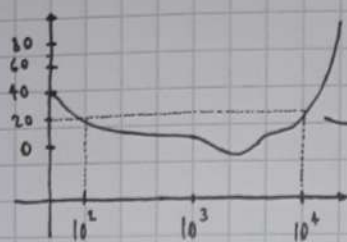
COMPRESSIONE 1:8 (64 Kb/s → 8 Kb/s), PER LA TRASMISSIONE

DI VOCE E MUSICA SU INTERNET SI USA LA TECNICA MPEG LAYER III, ANCH'È DETTA MP3.

MP3

QUESTA TECNICA LIMITA LA CODIFICA DEL SUONO ALLE INFORMAZIONI RILEVANTI PER L'ORECCHIO UMANO, SFRUTTA INNCIPALMENTE LE SEGUENTI CARATTERISTICHE:

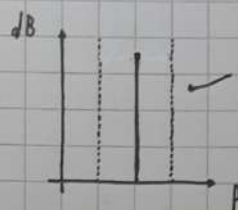
- SENSIBILITÀ DELL'ORECCHIO UMANO, INFATTI A DETERMINATE PRESSIONI SONORE (dB) L'ORECCHIO UMANO NON SENTE TUTTE LE FREQUENZE DA 20 Hz A 20 kHz MA SOLO ALCUNE:



IN QUESTO ESEMPIO POICHÉ SIAMO A 20 dB SENTIREMO SOLO LE FREQUENZE TRA  $10^2$  E  $10^4$  Hz, DUNQUE ASCOLTEREMO LE ALTRE

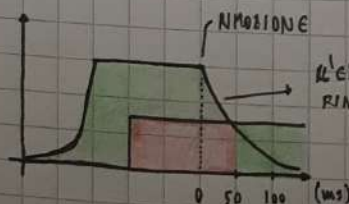
direttamente a  
piccola banda

- EFFETTO DI MASCHERATURA IN FREQUENZA, SI APPLICA LA TRASFORMATA DI FOURIER PER POI ANALIZZARE SIMULTANEMENTE BANDE DA POCHIE DECINE O CENTINAIA DI HERTZ, INFATTI IN OGNUNA DI QUESTE BANDE CI SARA' UN SUONO CON ENERGIA RILEVANTE RISPETTO AGLI ALTRI E DUNQUE POICHÉ IL NOSTRO ORECCHIO SENTIRÁ SOLO QUELLO MASCHERATO QUELLA BANDA CON QUEL SUONO:



ELIMINATO GLI ALTRI  
SVONI NATO CHE NON  
VENGONO SENTITI,

- EFFETTO DI MASCHERATURA NEL TEMPO, SUPPONIAMO CHE LA MASCHERATURA VENGA RIMOSSA DI COLPO, ALLORA L'ORECCHIO UMANO NON PERCEPIRÁ SUBITO LA SUA PRESENZA MA LO FARÁ DOPO UN BREVE INTERVALLO:



L'EFFETTO DELLA MASCHERA SI FA SENTIRE  
FINO A 50 ms IN QUESTO CASO

NON HA SENSO INIZIARE A CODIFICARE IL SEGNALE DEL SUONO MASCHERATO IN  $t=0$  ms, INFATTI VIENE MASCHERATO FINO A  $t=50$  ms, DA LÍ INIZIEREMO A CODIFICARLO.



# ARCHITETTURA E FUNZIONAMENTO DEL CALCOLATORE

QUI INIZIA LA VERA E PROPRIA 2<sup>a</sup> PARTE DEL CORSO, DOPO UNA BREVE PARENTESI SULLE TECNICHE DI CODIFICA E COMPRESSIONE INIZIAMO A PARLARE DEL CALCOLATORE E DEL SUO FUNZIONAMENTO.

## MODELLO DI CALCOLATORE

QUANDO SI PROGETTARONO I PRIMI COMPUTER SI PRESENTÒ LA NECESSITÀ DI DOVER SVOLGERE COMPITI SEMPRE PIÙ COMPLESSI. PER MAE CIÒ SI PENSÒ CHE SERVISSERO ISTRUZIONI PIÙ COMPLESSE E DUNQUE HARDWARE PIÙ ELABORATI.

QUESTO IN REALTÀ NON PORTÒ VANTAGGI PRESTAZIONALI, INFATTI POTEVA CAPITARE CHE L'HARDWARE FOSSE IN GRADO DI SVOLGERE ISTRUZIONI MOLTO COMPLESSE SENZA PERÒ AVERE IL SOFTWARE ADATTO A DARE TALI ISTRUZIONI (AD ESEMPIO L'HARDWARE POTEVA ESSERE IN GRADO DI MANIPOLARE INTERE STRINCE QUANDO I LINGUAGGI DI PROGRAMMAZIONE PERMETTEVANO DI ELABORARE UN CARATTERE ALLA VOLTA).

TALI MODELLI DI CALCOLATORI PRENDONO IL NOME DI **CISC** (COMPLEX INSTRUCTION SET COMPUTER).

IN SEGUITO VENNE FATTO UNO STUDIO STATISTICO SULLE FUNZIONI IN HARDWARE MENO UTILIZZATE, CHE PORTÒ ALLA LORO ELIMINAZIONE.

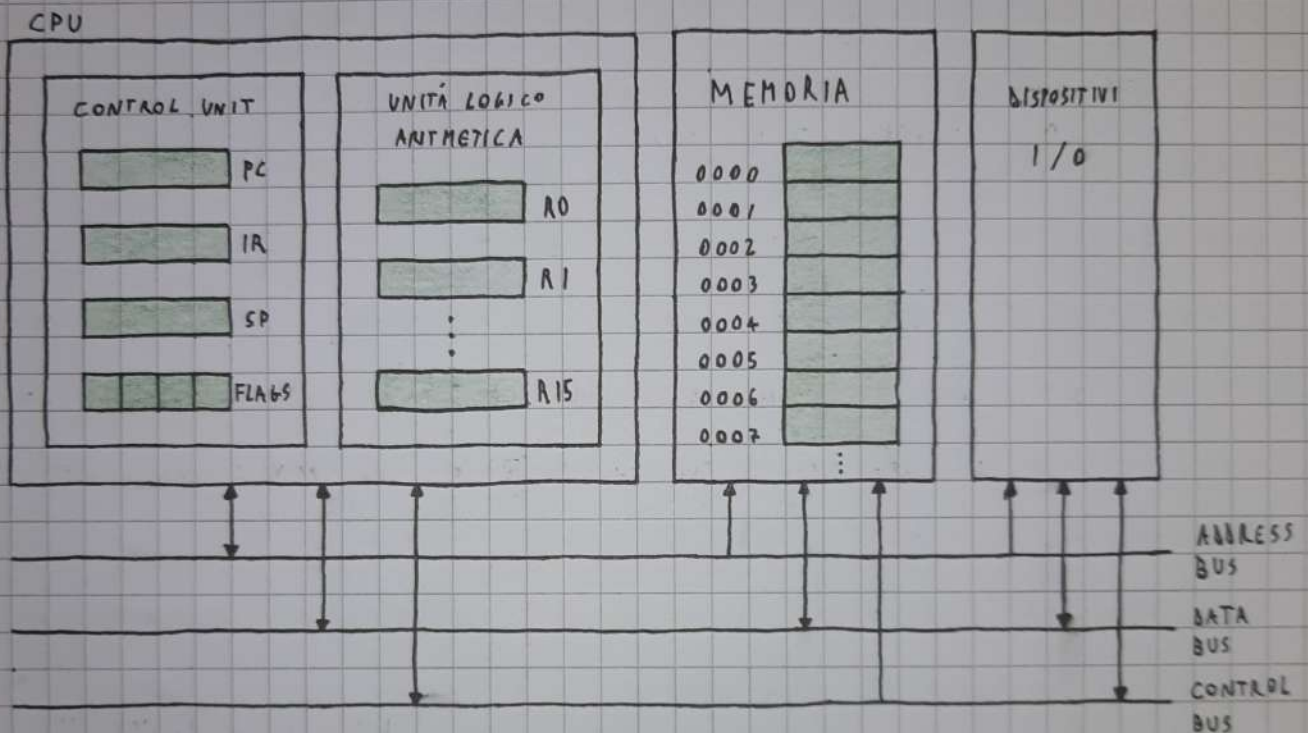
IL BENEFICIO DI AVER RICREATO TALI FUNZIONI IN SOFTWARE STA NEL FATTO CHE ORA LA "COMPLESSITÀ" CHE AVANZA NEL PROCESSORE VIENE SFRUTTATA PER OTTIMIZZARE LE OPERAZIONI BASE CHE VENGONO SVOLTE DIRETTAMENTE DALL'HARDWARE.

QUESTI MODELLI DI COMPUTER DOVE VIENE SEMPLIFICATO E OTTIMIZZATO LO SPAZIO CHE HA NEL CHIP DI SILICIO (CPU) SONO DETTI **RISC** (REDUCED INSTRUCTION SET COMPUTER).

ORA IN PD) USEREMO E STUDIEREMO UN MODELLO DI PROCESSORE BASATO SULLE RISC, IN FUTURO LO SIMULEREMO ANCHE AL COMPUTER, MA PER ORA CI LIMITIAMO COMPRENDERNE LA STRUTTURA.

## ARCHITETTURA DEL PROCESSORE

IL SEGUENTE MODELLO VERRA' SEMPRE USATO D'ORA IN AVANTI:



I PROCESSORI SONO TRA LORO DIVERSI, E DUNQUE HANNO CARATTERISTICHE DIFFERENTI. NEL NOSTRO MODELLO AVREMO:

- **PARALLELISMO** A 16 BIT, CIOE' IL NOSTRO PROCESSORE SARA' IN GRADO DI LAVORARE IN PARALLELO SU 16 BIT, DUNQUE SI VIRA CHE L'ARCHITETTURA DEL NOSTRO MODELLO PREVEDE WORD DA 16 BIT (ORamai SONO QUASI TUTTI A 64 BIT).
- 16 REGISTRI GENERAL PURPOSE DA 16 BIT L'UNO UTILIZZATI PER MEMORIZZARE DATI, INDIRIZZI DI MEMORIA, RISULTATI INTERMEDI DI OPERAZIONI...



- 3 REGISTRI SPECIAL PURPOSE NON DIRETTAMENTE INNESSABILI ALLE ISTRUZIONI (A CUI NON SI HA ACCESSO DIRETTO):

- **PROGRAM COUNTER (PC)**, CONTIENE L'INDIRIZZO DI MEMORIA DELLA PROSSIMA ISTRUZIONE CHE DOVRÀ ESSERE ESECUITA.
- **INSTRUCTION REGISTER (IR)**, CONTIENE L'ISTRUZIONE CHE STA CORRENTEMENTE VENENDO ESECUITA O DECODIFICATA (IN SOSTANZA I BIT DELL'INSTRUCTION REGISTER ABILITANO LE PORTE LOGICHE CHE ESEGUONO UNA SPECIFICA ISTRUZIONE).
- **STACK POINTER (SP)**, CONTIENE L'INDIRIZZO DI MEMORIA DELL'ULTIMO ELEMENTO INSERITO NELLO STACK (CIOÈ UNA STRUTTURA DATI CHIAMATA PILA E DUNQUE CON CRITERIO DI ESTRATTORE LIFO, LAST IN FIRST OUT). L'ULTIMO ELEMENTO CORRISPONDE ALLA TESTA DELLA PILA.

- **FLAG**, NE SONO PRESENTI 4 NELL'UNITÀ DI CONTROLLO E VENGONO UTILIZZATI PER COMPIERE OPERAZIONI SULLA BASE DELLE PRECEDENTI, IN PARTICOLARE AVREMO:

- **Z (ZERO)**, CHE PUÒ AD ESEMPIO ESSERE SFRUTTATO PER STABILIRE SE DUE NUMERI SONO UGUALI (IN TAL CASO LA DIFFERENZA SAREBBE NULLA E IL FLAG SI ATTIVEREBBE).
- **N (NEGATIVE)**, CHE PUÒ ESSERE SFRUTTATO PER CONFRONTARE 2 VALORI (SE IL FLAG SI ATTIVA VUOL DIRE CHE IL SOTTAMENDO, CIOÈ IL NUMERO SOTTATTO, È MAGGIORE DEL MINUENDO, CIOÈ IL NUMERO A CUI È STATO SOTTATTO QUALCOSA).

- **C (CARRY)**, SE PRENDIAMO 2 NUMERI DA 16 BIT E SOMMANDOLI COMPARE UN RIPOUNTO IN 17<sup>ma</sup> POSIZIONE SI ATTIVA QUESTO FLAG (NEL REGISTRI ABBIAMO 16 BIT DI SPAZIO).
- **V (OVERFLOW)**, SE STIAMO SOMMANDO 2 NUMERI IN COMPLEMENTO A 2 È POSSIBILE CHE SI VERIFI OVERFLOW, CIOÈ CHE IL RISULTATO SIA INCORRETTO A CAUSA DI SPAZIO INSUFFICIENTE (SI VERIFICA QUANDO  $C_{n-1} \oplus C_n = 1$ ), IN TAL CASO SI ATTIVA IL FLAG.
- **ISOLATED I/O**, PER PARLARE CON I DISPOSITIVI I/O CI SONO ISTRUZIONI DIVERSE DA QUELLE UTILIZZATE PER I DISPOSITIVI DI MEMORIA (SERVONO 16 BIT PER RIPETERSI AD UN INDIRIZZO DI I/O).
- **LOAD & STORE**, AL CONTRARIO DELLE MACCHINE CISC, CHE LAVORAVANO DIRETTAMENTE CON LA MEMORIA, LE RISC PERMETTONO L'ACCESSO ALLA MEMORIA SOLO ALLE ISTRUZIONI CHE TRASFERISCONO DATI REGISTRI → MEMORIA O MEMORIA → REGISTRI. (LE ALTRE OPERAZIONI OPERANO SOLO SUI REGISTRI).

## MODI DI INDIRIZZAMENTO

NELL'ACCESSO AI BYTE DELLA MEMORIA DOBBIAMO FORNIRE UN INDIRIZZO ATTRAVERSO L'ADDRESS BUS, PER FARLO CI SONO 3 METODI:

- **OPERANDO IMMEDIATO**, IN QUESTO CASO LA WORD CHE SEGLIE L'ISTRUZIONE (ISTRUZIONE CONTENUTA NELL'IR) CONTIENE DIRETTAMENTE IL DATO, MA SAPPIAMO CHE L'INDIRIZZO DELLA PROSSIMA ISTRUZIONE È CONTENUTO NEL PC E SEMPLICEMENTE L'INDIRIZZO DEL DATO SARÀ NEL PC.
- **INDIRIZZO ASSOLUTO**, LA WORD CHE SEGLIE L'ISTRUZIONE (IL CUI INDIRIZZO STA NEL PC) CONTIENE L'INDIRIZZO DEL DATO. VEDEREMO POI IN DETTAGLIO COSA ACCADE.



- **INDIRIZZO IN REGISTRO**, L'INDIRIZZO DEL DATO È CONTENUTO NEL REGISTRO SPECIFICATO DALL'ISTRUZIONE CORRENTE.

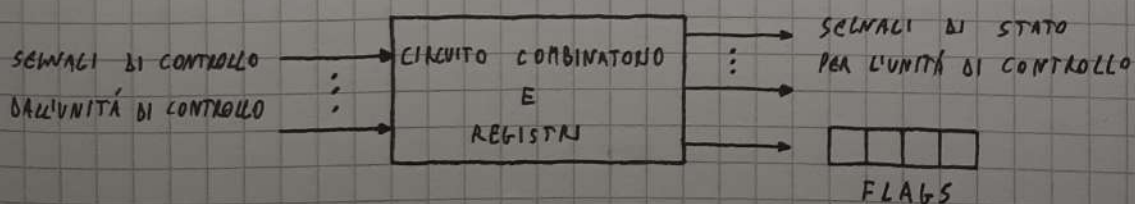
## L'UNITÀ DI CONTROLLO

L'UNITÀ DI CONTROLLO È UN CIRCUITO SEQUENZIALE SINCRONO (PERCHÉ RICEVE IN INPUT ANCHE IL SEGNALE DI CLOCK) CHE RICEVE DAL PROCESSORE DEI SEGNALE DI STATO E SI OCCUPA DI DECODIFICARE L'ISTRUZIONE CONTENUTA NELL' IR GENERANDO DEI SEGNALE DI CONTROLLO PER IL PROCESSORE.  
 RASSUMENDO:



## UNITÀ ARITMETICO LOGICA

L'UNITÀ ARITMETICO-LOGICA È UN CIRCUITO COMBINATORIO CHE RICEVE IN INGRESSO DEI SEGNALE DI CONTROLLO DA PARTE DELL'UNITÀ DI CONTROLLO E GENERA IN USCITA DEI SEGNALE DI STATO PER L'UNITÀ DI CONTROLLO MODIFICANDO EVENTUALMENTE I SUOI REGISTRI O I FLAG DELLA CONTROL UNIT.  
 RASSUMENDO



# 1 BUS

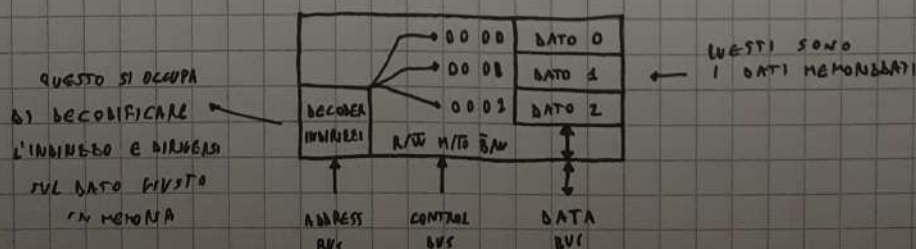
L'ADDRESS BUS E IL DATA BUS HANNO LO STESSO PARALLELISMO DEL PROCESSORE (IN REALTÀ NEI PROCESSORI A 64b L'ADDRESS BUS NON HA 64 FILI, NE HA MENO PERCHÉ LA MEMORIA, O RAM, HA BISOGNO DI MENO SPASSO).  
DUNQUE NEL NOSTRO CASO ADDRESS E DATA BUS AVRANNO 16 FILI.

DIVERSO È IL CASO DEL CONTROL BUS, IL CUI SCOPO È QUELLO DI STABILIRE LE 3 SEGUENTI PROPRIETÀ:

- $R/\overline{W}$ , LA CONTROL UNIT PONE A 1 QUESTO FILO SE LO SCOPO DELL'OPERAZIONE È QUELLO DI LEGGERE, ALTRIMENTI A 0 PER OPERAZIONI DI SCRITTURA
- $M/\overline{IO}$ , SI PONE A 0 PER LEGGERE O SCRIVERE SU DISPOSITIVI DI INPUT O OUTPUT, SI PONE A 1 PER LAVORARE SULLA MEMORIA
- $\overline{W}/\overline{B}$ , SI PONE A 1 SE SI VOGLIE FARE UN'OPERAZIONE DI LETTURA O SCRITTURA DI ~~1~~ UNA WORD, SI PONE A 0 PER LAVORARE SU UN SINGOLO BYTE

## LA MEMORIA RAM

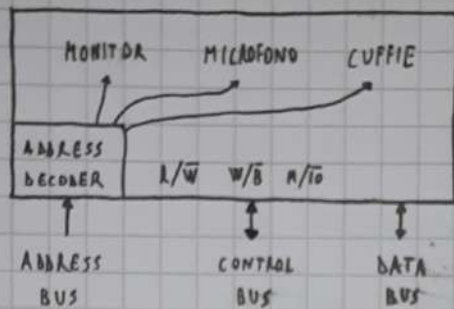
LA MEMORIA RAM RICEVE IN INGRESSO UN INDIRIZZO (DALL'ADDRESS BUS) E DEI VALORI DI CONTROLLO (DAL CONTROL BUS) PER CAPIRE SE SCRIVERE O SPERIRE UN VALORE O SE OPERARE SU UNA WORD O SU UN BYTE (SI SUPPONE CHE  $M = \overline{IO} = 1$ , DATO CHE STIANO LAVORANDO SULLA MEMORIA).





## I DISPOSITIVI INPUT / OUTPUT

SIMILMENTE A QUANTO VISTO PER LA MEMORIA RAM I DISPOSITIVI DI INPUT / OUTPUT  
 RICEVONO UN INDIRIZZO (CHE SI RIFERISCE AL DISPOSITIVO CON IL QUALE INTERAGIRE,  
 AD ESEMPIO TASTIERA O MONITOR) E TRASMETTONO O RICEVONO BYTE / WORD IN BASE  
 A CIÒ CHE DICE IL CONTROL BUS. DI SEGUITO UNO SCHEMA RASSUNTIVO.



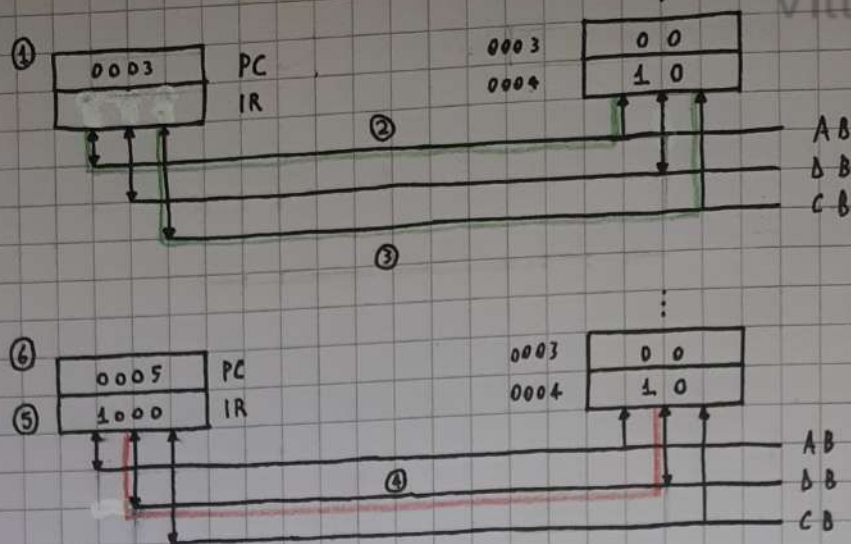
## LETTURA DELLA MEMORIA

IN QUESTO CAPITOLO VEDREMO COME UN PROGRAMMA SALVATO IN MEMORIA PUÒ ESSERE  
 LETTO ED ESEGUITO. SARÀ NECESSARIO, IN QUESTI ESAMI, DISSEGNARE RIPETUTAMENTE IL  
 MODELLO DI PROCESSORE CHE ABBIAMO VISTO PRIMA, PER SEMPLICITÀ MI LIMITERÒ A  
 RIPRODURRE SOLTANTO I LISTINI E LE COMPONENTI CHE SARANNO NECESSARIE ALLA  
 LETTURA DEL PROGRAMMA EVITANDO DI AGGIUNGERE PARTICOLARI SUPERFLUI CHE RENDEREBBERO  
 SOLO PIÙ DIFFICILE LA COMPrensIONE DEL MECCANISMO.

### 1. INSTRUCTION FETCH

IN QUESTA PRIMA FASE VIENE ESEGUITA LA PRIMA ISTRUZIONE DEL NOSTRO PROGRAMMA, COME  
 ABBIAMO VISTO IL PC CONTERrà L'INDIRIZZO IN MEMORIA DELLA PROSSIMA ISTRUZIONE E POICHÉ  
 NON CE NE SONO STATE PRIMA SARÀ ANCHE QUELLA INIZIALE.

( DISENDO DIETRO )

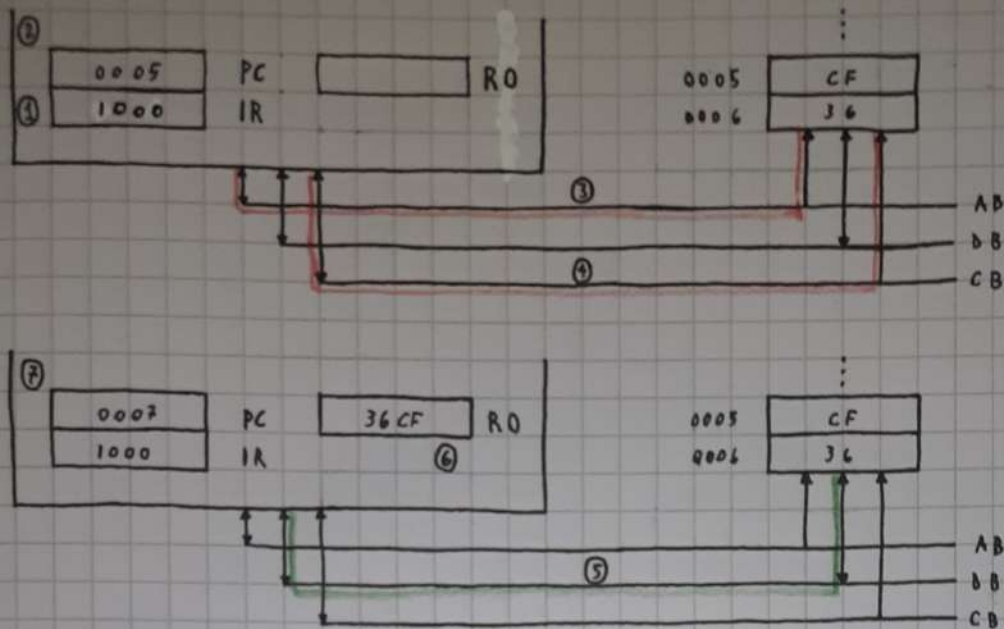


- ① INIZIALMENTE IL PROGRAM COUNTER È IMPOSTATO SU 0003, INDICANDO L'INDIRIZZO DELLA PRIMA ISTRUZIONE DA ESEGUIRE ALL'INDIRIZZO 0003.
- ② SULL'ADDRESS BUS VIENE PASSATO IL VALORE 0003.
- ③ SUL CONTROL BUS VENGONO PASSATI I VALORI  $R/\overline{W} = 1$ ,  $M/\overline{IO} = 1$ ,  $\overline{W/B} = 1$  (CIÒ È LETTURA IN MEMORIA DI UNA WORD).
- ④ LA MEMORIA METTE QUINDI SUL DATA BUS IL VALORE LETTO ALLI INDIRIZZI 0003 E 0004 (SI TRATTA DI UNA WORD):  
 \*\* I BIT DELLA CELLA CON INDIRIZZO PIÙ ALTO SARANNO QUELLI PIÙ SIGNIFICATIVI \*\*
- ⑤ L'ISTRUZIONE LETTA È PRONTA PER ESSERE DECODIFICATA ED ESEGUIITA DATO CHE ORA SI TROVA NELL'INSTRUCTION REGISTER (CONTIENE L'ISTRUZIONE CORRENTE).
- ⑥ OIA CHE ABBIAMO LETTO ED ESECUITO LA PRIMA ISTRUZIONE SI PASSA ALLA PROSSIMA INCREMENTANDO (IN QUESTO CASO, DATO CHE ABBIAMO LETTO UNA WORD, DI 2) IL VALORE DEL PROGRAM COUNTER.

## 2. OPERAND FETCH - OPERANDO IMMEDIATO

ORA CHE ABBIAMO LETTO LA PRIMA ISTRUZIONE POSSIAMO PROCEDERE CON IL RESTO DEL PROGRAMMA. IN QUESTO CASO SUPPONIAMO CHE LE ISTRUZIONI PREVEDANO UNA LETTURA IN MEMORIA CON OPERANDO IMMEDIATO (UN MODO DI INDIRIZZAMENTO).



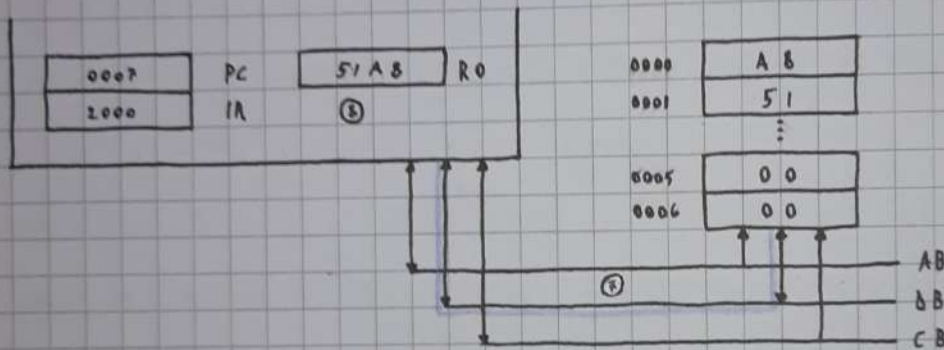
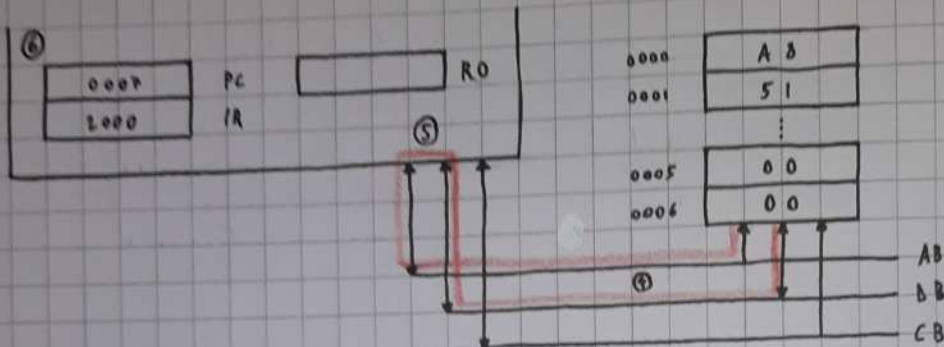
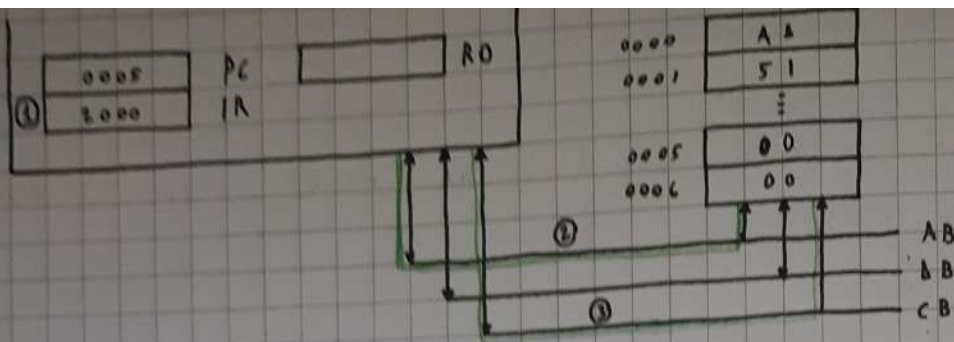


- ① L'ISTRUZIONE CORRENTE CI DICE DI MEMORIZZARE NEL REGISTRO 0 IL VALORE CHE SERVE NEL PROGRAMMA. (OPERANDO IMMEDIATO)
- ② L'INDIRIZZO DEL VALORE CHE SERVE NEL PROGRAMMA È CONTENUTO NEL PROGRAM COUNTER
- ③ L'INDIRIZZO VIENE TRASMESSO ALLA MEMORIA TRAMITE L'ADDRESS BUS
- ④ SUL CONTROL BUS SONO TRASMESSI I VALORI  $R/\bar{W} = 1$ ,  $W/\bar{B} = 1$ ,  $M/\bar{I/O} = 1$ .
- ⑤ LA MEMORIA TRASMETTE SUL DATA BUS IL VALORE 36 CF (INDIRIZZI PIÙ GRANDI, CIFRE PIÙ SIGNIFICATIVE)
- ⑥ IL VALORE LETTO VIENE INSERITO NEL REGISTRO R0.
- ⑦ IL PROGRAM COUNTER INCREMENTA DI 2 PERCHÉ ABBIAMO LETTO UNA WORD.

### 3. OPERAND FETCH - INDIRIZZO ASSOLUTO

IN QUESTO CASO LA WORD CHE SEGLIMERÀ L'ISTRUZIONE CORRENTE NEL PROGRAMMA CONTERRÀ L'INDIRIZZO DEL DATO CHE VORRÀMO SCRIVERE IN UNO DEI REGISTRI.

SARÀ UN PO' PIÙ LABORIOSO E LUNGO IL PERCORSO CHE CI PERMETTERÀ DI MEMORIZZARE IL DATO.

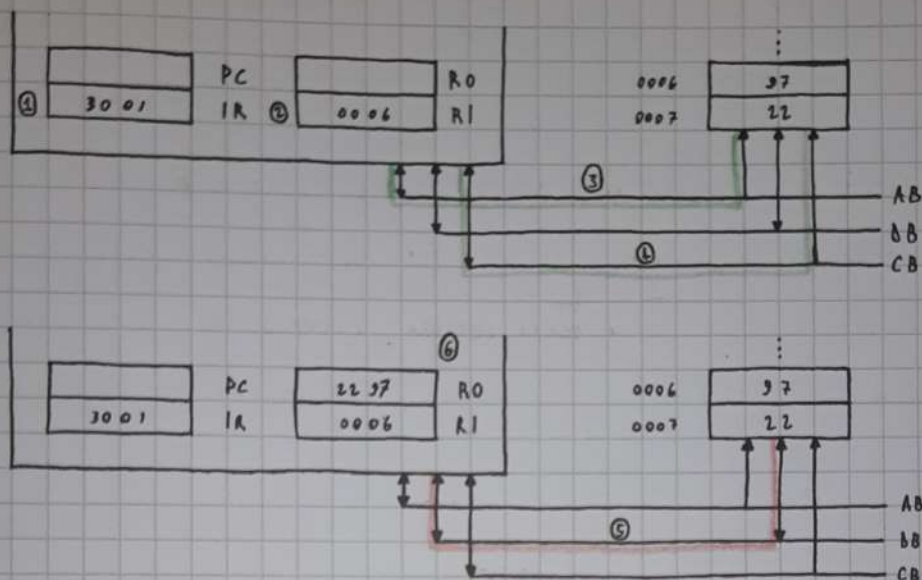


- ① L'ISTRUZIONE CONTENUTA NELL'INSTRUCTION REGISTER PREVEDE DI MEMORIZZARE NEL REGISTRO 0 IL DATO IL CUI INDIRIZZO È CONTENUTO NELLA PROSSIMA ISTRUZIONE.
- ② PER LEGERE LA PROSSIMA ISTRUZIONE SI SPUNTA IL PC ~~PER MEMORIZZARE IL DATO~~
- ③ VENGONO PASSATI I SOLITI PARAMETRI SUL CONTROL BUS
- ④ L'INDIRIZZO DEL DATO CHE VOGLIAMO MEMORIZZARE CI VIENE RESTITUITO COME DATO
- ⑤ TUTTAVIA IL PROCESSORE SA CHE STIAMO EFFETTUANDO UN INDIRIZZAMENTO CON ~~IL~~ INDIRIZZO ASSOLUTO E DUNQUE INSERISCE NELL'ADDRESS BUS IL DATO RICEVUTO DALLA MEMORIA
- ⑥ VIENE AZIONATO IL PC INCREMENTANDOLO DI 2 (È STATA LETTA UNA WORD).
- ⑦ LA MEMORIA RESTITUISCE IL DATO PRESENTE ALL'INDIRIZZO CONTENUTO NELLE CELLE CHE ABBIAMO LETTO PRIMA
- ⑧ IL DATO FINALE VIENE MEMORIZZATO NEL REGISTRO 0



#### 4. OPERAND FETCH - INDIRIZZO IN REGISTRO

IN QUESTO ULTIMO CASO L'INDIRIZZO DEL DATO CHE VOGLIAMO LEGERE E MEMORIZZARE È CONTENUTO IN UN REGISTRO.



- ① L'ISTRUZIONE CORRENTE STABILISCE CHE DOBBIAMO LEGERE UN VALORE DALLA MEMORIA E SALVARLO NEL REGISTRO 0 CON INDIRIZZAMENTO IN REGISTRO.
- ② IL REGISTRO DAL VALORE DONNEMO PRENDERE L'INDIRIZZO DEL DATO DA MEMORIZZARE È L'1.
- ③ VIENE UNQUE PASSATO IL CONTENUTO DI R1 SULL'ADDRESS BUS.
- ④ LA MODALITÀ È LA SOLITA: LETTURA DI UNA WORD DALLA MEMORIA.
- ⑤ VIENE RESTITUITO SUL DATA BUS IL CONTENUTO DELLA CELLA PUNTATA DA R1.
- ⑥ VIENE MEMORIZZATO IN RO IL DATO PASSATO.

#### SCRITTURA IN MEMORIA

IN QUESTO CAPITOLO NON PARLÓ NISUNO MA ILLUSTRERÓ BREVEMENTE A PAROLE COME IVÓ ESSERE EFFETTUATA LA SCRITTURA DI UNA WORD O IN BYTE IN MEMORIA.

#### 1. MEMORY STORE - INDIRIZZO IN REGISTRO

L'INDIRIZZO IN REGISTRO NEL CASO DI LETTURA DELLA MEMORIA E NEL CASO

DELLA SCRITTURA IN MEMORIA DIFFERISCE PER UN SEMPLICE PARTICOLARE, CIÒ CHE QUANDO SI ACCADE FINALMENTE ALLA POSIZIONE IN MEMORIA PUNTATA DAL REGISTRO 1 NELL'ESEMPIO DI PRIMA IN UN CASO SI ESTRAEVA IL VALORE CONTENUTO IN MEMORIA PER SALVARLO IN R0 MENTRE IN QUESTO CASO (SCRITTURA) SI ESTRAE IL VALORE CONTENUTO IN R0 PER SALVARLO IN MEMORIA.

## 2. MEMORY STORE - INDIRIZZO ASSOLUTO

ANCHE IN QUESTO CASO QUANDO FINALMENTE ABBIAMO ACCESSO ALLA POSIZIONE PUNTATA DALL'1<sup>a</sup> ISTRUZIONE SUCCESSIVA A QUELLA CORRENTE (A CUI ABBIAMO AVUTO ACCESSO TRAMITE IL PROGRAM COUNTER) NEL CASO DELLA LETTURA MEMORIZZAVAMO IL DATO ~~MA~~ IN UN REGISTRO MENTRE ORA RILEVAMO IL CONTANEO, SCRIVEREMO IN MEMORIA IL CONTENUTO IN UN REGISTRO.



## IL LINGUAGGIO MACCHINA E ASSEMBLY

IN QUESTO CAPITOLO VEDREMO COME VIENE SCRITTO UN PROGRAMMA IN LINGUAGGIO MACCHINA, CIOÈ IN CODICE BINARIO E COME SI È ANDATO A SVILUPPARE DEGLI ASSEMBLATORI, CIOÈ DEI TRADUTTORI IN GRADO DI RENDERE PIÙ FACILMENTE MEMORIZZABILI LE ISTRUZIONI.

### IL FORMATO DELLE ISTRUZIONI

I BIT DELL'ISTRUZIONE NON SONO SCELTI A CASO, SONO SCELTI IN MANIERA TALE DA AVERE UNA PAGURA SUFFICIENTE. DI FATTO AL GIORNO D'OGGI ESISTONO DELLE ARCHITETTURE PIÙ BEN CONSOLIDATE E QUINDI SI TENDE AD USARE UN LINGUAGGIO COMPATIBILE CON MOLTI ALTRI PIUTTOSTO CHE INVENTARE DA ZERO UNA NUOVA ARCHITETTURA.

IL CRITERIO DI SCELTA DEL FORMATO DELLE ISTRUZIONI È ARBITRARIO E LOGICO, DEVE INFATTI SEMPLIFICARE AL MASSIMO LA RETE LOGICA DI DECODIFICA.

IN GENERALE LAVOREREMO SU QUESTO FORMATO:



### MODI DI INDIRIZZAMENTO

ABBIAMO VISTO 3 MODI DI INDIRIZZAMENTO, I RELATIVI CODICI SARANNO:

- OPERANDO IMMEDIATO     $01_2$     I (MNEMONICO)
- INDIRIZZO ASSOLUTO     $10_2$     A (MNEMONICO)
- INDIRIZZO IN REGISTRO     $11_2$     R (MNEMONICO)

SI OSSERVA SCHEMARE LE ISTRUZIONI IN BINARIO PUÒ RISULTARE MOLTO DIFFICILE

È QUINDI QUANDO SI SCRIVEVANO LUNGHE PORZIONI DI CODICE SI CREÒ LA  
NECESSITÀ DI COSTRUIRE DEI PROGRAMMI (IN LINGUAGGIO MACCHINA, AVENDO BISSATO) DETTI  
ASSEMBLATORI CHE TRADUCEVANO IN LINGUAGGIO MACCHINA DEI CODICI SPECIFICI PIÙ  
SEMPLICI DA SCRIVERE E RICORDARE CHIAMATO ABBONTO CODICI MNEMONICI (IN QUESTO CASO  
SONO I, A, R)

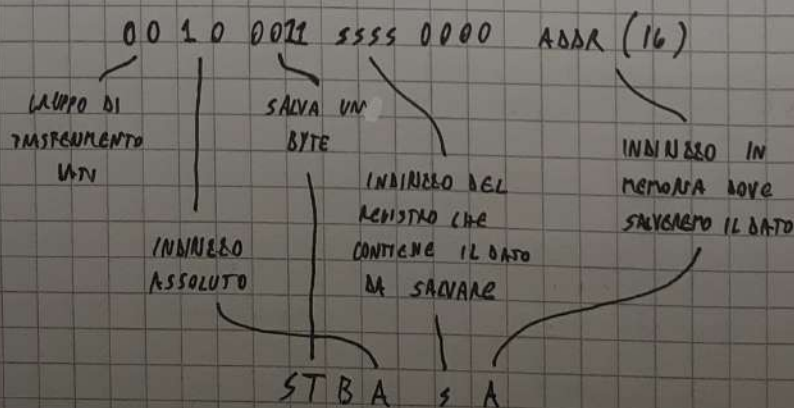
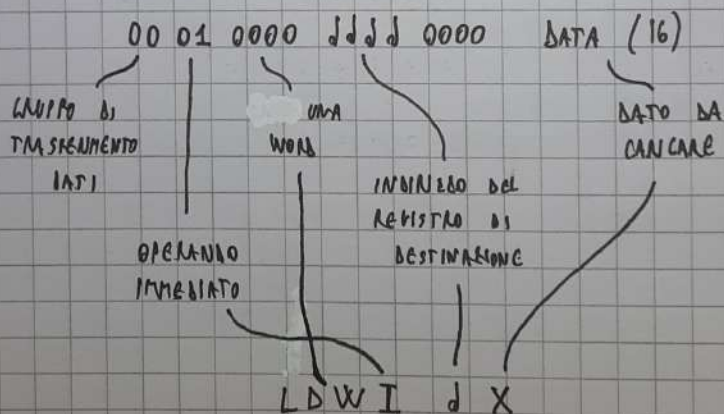
1 GRUPPE

ESISTONO 4 DIVERSI GRUPPI, A CUI CORRISPONDONO I SEGUENTI CODICI

- TRASFERIMENTO DATI ( DA E VERSO LA MEMORIA O TRA REGISTRI )

CODICE 0 , 002.

FAI' UNA QUALCHE ESERCIZIO DI ISTRUZIONE PER IL TRASFERIMENTO DI DATI E  
POI FORNIR' UNA TABELLA CON TUTTE LE OPERAZIONI POSSIBILI CON QUESTO  
GRUPPO:





|      |     |            |                                 |           |
|------|-----|------------|---------------------------------|-----------|
| LDWI | d X | LOAD WORD  | 0 0 0 1 0 0 0 0 d d d d 0 0 0 0 | DATA (16) |
| LDWA | d A | LOAD WORD  | 0 0 1 0 0 0 0 0 d d d d 0 0 0 0 | ADDR (16) |
| LDWR | d a | LOAD WORD  | 0 0 1 1 0 0 0 0 d d d d a a a a |           |
| LDBI | d X | LOAD BYTE  | 0 0 0 1 0 0 0 1 d d d d 0 0 0 0 | DATA (8)  |
| LDBA | d A | LOAD BYTE  | 0 0 1 0 0 0 0 1 d d d d 0 0 0 0 | ADDR (16) |
| LDBR | d a | LOAD BYTE  | 0 0 1 1 0 0 0 1 d d d d a a a a |           |
| STWA | s A | STORE WORD | 0 0 1 0 0 0 1 0 s s s s 0 0 0 0 | ADDR (16) |
| STWR | s a | STORE WORD | 0 0 1 1 0 0 1 0 s s s s a a a a |           |
| STBA | s A | STORE BYTE | 0 0 1 0 0 0 1 1 s s s s 0 0 0 0 | ADDR (16) |
| STBR | s a | STORE BYTE | 0 0 1 1 0 0 1 1 s s s s a a a a |           |
| MV   | s d | MOVE       | 0 0 0 0 0 1 0 0 s s s s d d d d |           |
| PUSH | s   | PUSH       | 0 0 0 0 1 0 0 0 s s s s 0 0 0 0 |           |
| POP  | d   | POP        | 0 0 0 0 1 0 0 1 d d d d 0 0 0 0 |           |
| SPRD | d   | LEVI SP    | 0 0 0 0 1 1 0 1 d d d d 0 0 0 0 |           |
| SPWR | s   | SCRVI SP   | 0 0 0 0 1 1 1 0 s s s s 0 0 0 0 |           |

LE ULTIME 4 OPERANO SULLA STACK POINTER MENTRE LA MOVE SPESITA UN DATO DA UN REGISTRO ALL'ALTRO.

UN ESEMPIO DI PROGRAMMA CHE LEGGE UN DATO DALLA MEMORIA E LO SALVA IN

UN REGISTRO E' IL SEGUENTE:

LDWI R15 ABAC →  $\underbrace{0001}_1 \underbrace{0000}_0 \underbrace{1111}_F \underbrace{0000}_0$  (ABAC)

|         |    |                 |
|---------|----|-----------------|
| 0 0 0 0 | FO | PRIMA VIENE     |
| 0 0 0 1 | 10 | L'ISTRUZIONE    |
| 0 0 0 2 | AC | POI L'OPERANDO  |
| 0 0 0 3 | AB | SU CUI LAVORARE |

# • ARITMETICO - LOGICHE

CODICE  $1_1$ ,  $01_2$ .

IN QUESTO CASO NON È MOLTO DA DIRE SULLE OPERAZIONI, SONO SEMPLICI E MANIPOLAZIONI DEI BIT DI REGISTRI. NOTIAMO PERO CHE NON ESSENDOCI TRASFERIMENTO DI DATI NON AVRÀ SENSO INDICARE UN MODO DI INDIRIZZAMENTO E QUINDI VI SARÀ IL CODICE 00, CHE NON CORRISPONDE AD ALCUN CODICE DI INDIRIZZAMENTO.

|     |   |   |             |          |      |      |
|-----|---|---|-------------|----------|------|------|
| ADD | S | d | ADD         | 01000000 | SSSS | dddd |
| SUB | S | d | SUBTRACT    | 01000001 | SSSS | dddd |
| NOT | r |   | BITWISE NOT | 01000010 | rrrr | 0000 |
| AND | S | d | BITWISE AND | 01000011 | SSSS | dddd |
| OR  | S | d | BITWISE OR  | 01000100 | SSSS | dddd |
| XOR | S | d | BITWISE XOR | 01000101 | SSSS | dddd |
| INC | r |   | INCREMENT   | 01001000 | rrrr | 0000 |
| DEC | r |   | DECREMENT   | 01001001 | rrrr | 0000 |
| LSH | r |   | LEFT SHIFT  | 01001010 | rrrr | 0000 |
| ASH | r |   | RIGHT SHIFT | 01001011 | rrrr | 0000 |

SI NOTI INOLTRE CHE NEL CASO DELLA SOMMA SI STA DICENDO DI FARE LA SOMMA TRA IL CONTENUTO DI S E d, E DI METTERE IL RISULTATO IN d (UN PÒ COME SI FACEVA IN C CON IL  $+=$ ).

UN ESEMPIO DI CODICE CHE ESegue LA SOTTRAZIONE TRA R8 E R10 E LA SALVA IN R10 È IL SEGUENTE:

SUB R8 R10 →

|      |      |      |      |
|------|------|------|------|
| 0100 | 0001 | 1000 | 1010 |
| 4    | 1    | 8    | A    |

↓

|      |    |                        |
|------|----|------------------------|
| 0000 | 8A | } SI LEVE AL CONTRARIO |
| 0001 | 41 |                        |



• INPUT - OUTPUT

CODICE 2 , 10.

I DISPOSITIVI DI INPUT E OUTPUT HANNO UNA MEMORIA E SE SI VOLESSE LEGGERE O SCRIVERE AD UN DETERMINATO INDIRIZZO DI QUELLE MEMORIE (DEI DISPOSITIVI DI IO) SI UTILIZZERANNO QUESTE ISTRUZIONI.

OVVIAMENTE NOI LEGEREMO I DATI DEI DISPOSITIVI DI INPUT E SCRIVEREMO DATI NEI DISPOSITIVI DI OUTPUT.

|      |   |   |             |                                 |          |
|------|---|---|-------------|---------------------------------|----------|
| INW  | d | A | INPUT WORD  | 1 0 0 0 0 0 0 0 d d d d 0 0 0 0 | IN_ADDR  |
| INB  | d | A | INPUT BYTE  | 1 0 0 0 0 0 0 1 d d d d 0 0 0 0 | IN_ADDR  |
| OUTW | s | A | OUTPUT WORD | 1 0 0 0 0 0 1 0 s s s s 0 0 0 0 | OUT_ADDR |
| OUTB | s | A | OUTPUT BYTE | 1 0 0 0 0 0 1 1 s s s s 0 0 0 0 | OUT_ADDR |
| TSTI | A |   | TEST INPUT  | 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 | IN_ADDR  |
| TSTO | A |   | TEST OUTPUT | 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 | OUT_ADDR |

PER CAMBIARE IL SIGNIFICATO DELLE 2 ULTIME ISTRUZIONI DOBBIAMO PARTIRE DAL FATTO CHE IL PROCESSORE CAMBIA LO STATO DELLA CONTROL UNIT AL RITMO DEL CLOCK. QUANDO ESSO INTERAGISCE CON UN DISPOSITIVO ESTERNO ~~DA MEMORIA~~ (CHE IN LEGGERE HA RITMO PIÙ LENTI) SI METTEVA MAGARI A CONTROLLARE IL CONTENUTO DEGLI INPUT 1 MILIARDO N VOLTE AL SECONDO QUANDO LO STESSO DISPOSITIVO DI INPUT LO AGGIORNA 1000 VOLTE AL SECONDO.

PER EVITARE DI CONTROLLARE INVILMENTE SI UTILIZZANO DELLE TECNICHE VIA SOFTWARE (COME IL POLLING) CHE PREVEDONO DI RIMANERE IN ATTESA DEL PROSSIMO INPUT FINO AL SUO ARRIVO.

PER CAMBIARE SE L'INPUT È ARRIVATO O NO VIENE UTILIZZATA LA FUNZIONE DI TEST, CHE AGGIORNA IL FLAG Z A SECONDA CHE SIA O NO ARRIVATO IL SEGNALE DI INPUT / OUTPUT.

# • CONTROLLO

CODICE 3 , 11<sub>2</sub>

Al servizio l'elenco di istruzioni di controllo che possiamo effettuare:

|       |   |                        |                                          |
|-------|---|------------------------|------------------------------------------|
| BR    | A | BRANCH                 | 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ADDR |
| JMP   | f | JUMP                   | 1 1 0 0 0 0 0 1 f f f f f f f f          |
| JMPZ  | f | JUMP IF ZERO           | 1 1 0 0 0 0 1 0 f f f f f f f f          |
| JMPNZ | f | JUMP IF NOT ZERO       | 1 1 0 0 0 0 1 1 f f f f f f f f          |
| JMPN  | f | JUMP IF NEGATIVE       | 1 1 0 0 0 1 0 0 f f f f f f f f          |
| JMPNN | f | JUMP IF NOT NEGATIVE   | 1 1 0 0 0 1 0 1 f f f f f f f f          |
| JMPC  | f | JUMP IF CARRY          | 1 1 0 0 0 1 1 0 f f f f f f f f          |
| JMPV  | f | JUMP IF OVERFLOW       | 1 1 0 0 0 1 1 1 f f f f f f f f          |
| RET   |   | RETURN FROM SUBROUTINE | 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0          |
| CALL  | A | SUBROUTINE CALL        | 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 ADDR     |
| HLT   |   | HALT                   | 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0          |

Dove BR e JMP sono gli unici salti non condizionati e danno la possibilità di saltare ad un indirizzo qualsiasi (BR) o di un determinato offset a partire dalla posizione corrente (JMP).

Oss. 1: IL JMP VIENE ESECUITO DOPO IL L'INCREMENTO DEL PROGRAM COUNTER (PC).  
BISOGNA CONSIDERARE CHE L'OFFSET SI HA A PARTIRE DALL'ISTRUZIONE SUCCESSIVA.

Oss. 2: IL VALORE DELL'OFFSET È IN COMPLEMENTO A 2 E QUINDI POICHÉ NUMERI NEGATIVI VOLLONO DIRE ANDARE INDIETRO NELLA MEMORIA SI POSSONO CREARE DEI CICLI.

LA ISTRUZIONE HALT SPENDE LA CPU, IL C NON PREVEDE QUESTO COMANDO PERCHÉ AL TERMINE DEL PROGRAMMA NON ARRESTA IL SISTEMA OPERATIVO MA LI



RESTITUISCE IL CONTROLLO CON IL RETURN.

## L'ASSEMBLATORE

L'ASSEMBLATORE, COME ACCENNAVO IN PRECEDENZA, È UN PROGRAMMA CHE SI OCCUPA DI TRADURRE LA LINGUAGGIO ASSEMBLY A LINGUAGGIO MACCHINA.

QUESTO SI OCCUPA ANCHE DI CALCOLARE GLI INDIRIZZI DI SALTO (INFATTI NOI IN ASSEMBLY NON HAVEMO L'INDIRIZZO SPECIFICO A CUI SALTARE MA L'ETICHETTA, CIOÈ UN NOME LEGATO AD UNA RIGA DEL PROGRAMMA). RISERVA IN MEMORIA ANCHE DELLO SPAZIO PER I DATI (ALL'INIZIO LO FA).

### ESEMPIO PROGRAMMA IN ASSEMBLY

COME ESEMPIO SCRIVERÒ ORA UN PROGRAMMA CHE SOMMA  $N$  NUMERI (DOVE  $N$  SARÀ SCRITTO SU UNA WORD E VARIA 3) SCRITTI SU 8 BIT.

```
N: WORD 3 // HO SALVATO NELLO SPAZIO DI MEMORIA RISERVATO AI DATI LA
DATA: BYTE 5 // WORD 3 E I 3 BYTE CHE RAPPRESENTANO I VALORI CHE
      BYTE 9 // DOVRÒ FARE SOMME.
      BYTE 0C //
LDWA R1 N // HO CARICATO IN R1 IL NUMERO 3
LDWI R2 DATA // HO CARICATO IN R2 L'INDIRIZZO AL QUALE SI TROVA IL PRIMO DATO
LDWI R10 0 // HO RESETTATO IL CONTENUTO DI R10
LOOP: LDBR R11 R2 // HO CARICATO IL DATO PUNTATO DA R2 IN R11
      ADD R11 R10 // HO SOMMATO IL CONTENUTO DI R10 E R11 E L'HO SALVATO IN R10
      INC R2 // HO INCREMENTATO R2, ORA PUNTA AL DATO SUCCESSIVO
      DEC R1 // HO DECREMENTATO IL NUMERO DI DATI DA SOMMARE ANCORA
      JMP NZ LOOP // VERIFICO SE ABBIAMO TERMINATO LE SOMME, SE SI PROSEGUE ALTRA VOLTA AL LOOP
      HLT // TERMINO IL PROGRAMMA, ORA IN R10 C'È LA SOMMA FINALE.
```

ESEMPIO PROGRAMMA IN LINGUAGGIO MACCHINA

PRENDIAMO IL PRECEDENTE ESEMPIO E TRADUCIAMOLO IN LINGUAGGIO MACCHINA SFRUTTANDO LE ISTRUZIONI VISTE NELLE PRECEDENTI PAGINE.

03 } N (È UNA WORD, QUINDI 16 BIT)  
00

05 }  
09 DATA  
0C

001000000000100000000000<sub>h</sub> (PRIMA IL BYTE MENO SIGNIFICATIVO)

10  
20  
00  
00

LDWA R1 N

000100000001000000000000<sub>h</sub>

20  
10  
02  
00

LDWI R2 DATA

000100000101000000000000<sub>h</sub>

A0  
10  
00  
00

LDWI R10 0

0011000110110010 // LOOP

B2  
31

LDBR R11 R2

0100000010111010

B A  
4 0

ADD R11 R10

0100100000100000

20  
4 8

INC R1

010010010000100000

10  
4 9

DEC R1



1 1 0 0 0 0 1 1 [ OFFSET ]

OFFSET = F6

C3

JMP NE LOOP

HO CONTATO 10 POSIZIONI IN INDIETRO PER ARRIVARE ALL'ETICHETTA DEL LOOP, QUINDI SCRIVIAMO -10 IN COMPLEMENTO A 2 SU 8 BIT:

$-10 = -00001010 = 11110110 = F6$

1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0

0 0

C F

HLT

← IL PC SI TROVA QUI AL MOMENTO DEL SALTO

## LE LIBRERIE

DURANTE IL CORSO DEI FONDAMENTI DI PROGRAMMAZIONE ABBIAMO FATTO AMPIO USO DELLE LIBRERIE, CIOÈ LUOGHI DOVE VENIVANO RUNITI PIÙ FUNZIONI PER POTER ESSERE CHIAMATE IN GRUPPO O SINGOLARMENTE DA ALTRE PROGRAMMI.

NEL SEGUENTE ESEMPIO VEDREMO LA LIBRERIA CHE PERMETTERÀ DI FARE UNA MOLTIPLICAZIONE E IL PROGRAMMA CHE LA CHIAMERÀ.

### MOLTIPLICAZIONE .IXX

|           |      |          |    |                                                                                                    |
|-----------|------|----------|----|----------------------------------------------------------------------------------------------------|
| MUL:      | LDWI | R10      | 0  | AZZERO LA CELLA CON IL CONTATORE DELLE SOMME                                                       |
|           | LDWI | R0       | 0  | AZZERO LA CELLA DOVE SCRIVERÒ IL RISULTATO                                                         |
| MUL_LOOP: | MV   | R2       | R3 | SPOSTO R2 IN R3 ALTAMENTE CON LA SUB LO PERDO                                                      |
|           | SUB  | R10      | R3 | ESEGUO LA SOTTRAZIONE TRA R10 E R3                                                                 |
|           | JMPZ | MUL_END  |    | ] POICHÉ NON C'È UNA CONDIZIONE UNICA IN CASO DI NUMERI NON POSITIVI NE UNISCO 2 (ZERO E NEGATIVO) |
|           | JMPN | MUL_END  |    |                                                                                                    |
|           | ADD  | R1       | R0 | ESEGUO UNA SOMMA                                                                                   |
|           | INC  | R10      |    | INCREMENTO IL CONTATORE DI SOMME                                                                   |
|           | JMP  | MUL_LOOP |    | SALTO INCONDIZIONATAMENTE ALL'INIZIO DEL LOOP                                                      |
| MUL_END:  | RET  |          |    | RESTITUISCO IL CONTROLLO AL MAIN                                                                   |

MOLTIPLICAZIONE - MAIN . AXX

~~##~~ # include "moltiplicazione . ixx"

~~##~~

|         |               |                                                                           |
|---------|---------------|---------------------------------------------------------------------------|
| A :     | word 85       | ] QUESTI SONO I 2 VALORI CHE<br>MOLTIPLICHERO TRA LORO                    |
| B :     | word 92       |                                                                           |
| START : | LAWI R15 0100 | SALVO IN R15 IL VALORE DI INIZIALIZZAZIONE DELLO SP                       |
|         | SPWA R15      | INIZIALIZZO LO STACK POINTER                                              |
|         | LDA R1 A      | ] CARICO I 2 VALORI DA MOLTIPLICARE                                       |
|         | LDA R2 B      |                                                                           |
|         | PUSH R1       | ] MI ASSICURO DI NON PERDERE I VALORI DI<br>R1 E R2 SALVANOLI NELLO STACK |
|         | PUSH R2       |                                                                           |
|         | CALL MUL      | CHIAMO LA FUNZIONE DELLA MOLTIPLICAZIONE                                  |
|         | POP R2        | ] ESTRAUO DALLO STACK I VALORI DI R1<br>E R2                              |
|         | POP R1        |                                                                           |
|         | HLT           | TERMINO IL PROGRAMMA.                                                     |

OSSERVAZIONI IMPORTANTI

DI SEGUITO ALCUNE INFORMAZIONI SULLA LOGICA CON LA QUALE VIENE ASSEMBLATO ED ESECUITO UN PROGRAMMA COME QUELLO CHE HO APPENA PROPOSTO. ~~QUESTE~~ ALIUNDO CHE QUESTE OSSERVAZIONI SONO RITORTE DI ALCUNE ORE DI ESPERIMENTI CON IL PROGRAMMA SIMCPU :

- PER CONVENZIONE SI È DECISO DI ADOBIARE I REGISTRI ~~RA~~ R1, R2 (ED EVENTUALMENTE I SUCCESSIVI) ALI ARGOMENTI DELLE FUNZIONI (ECCO PERCHÉ HO SALVATO IN R1 E R2 I VALORI DA MOLTIPLICARE) MENTRE R0 È UTILIZZATO COME VALORE DI RITORNO DELLE FUNZIONI.

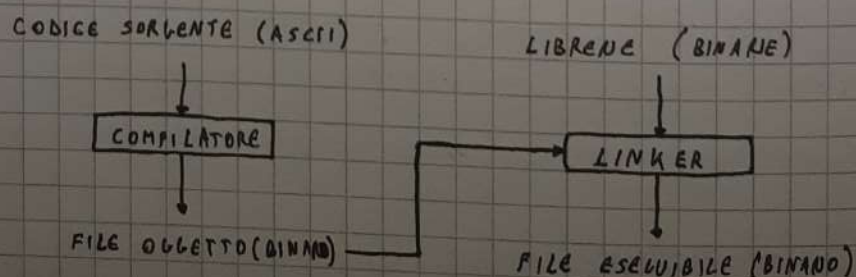


- LO STACK NON È ALTRO CHE UNA PARTE DELLA MEMORIA. NOI ABBIAMO UN REGISTRO (LO STACK POINTER) CHE PUNTA AD UN ELEMENTO DI QUESTA PARTE DELLA MEMORIA (CHE CHIAMEREMO STACK). QUESTO PUNTATORE VA INIZIALMENTE INIZIALIZZATO ED È PER QUESTO CHE HO SALVATO 0100 IN R15. IL NOSTRO STACK SARÀ COMPOSTO DALLE CELLE DELLA MEMORIA CON INDIRIZZO UGUALE E SUPERIORE A 0100.
- POP AD OGNI POP IL PROCESSORE DECREMENTA LO STACK POINTER (DEVE ESTRARRE UN ELEMENTO!) (POI SI RICORDA CHE LO SP PUNTA ALLA PRIMA CELLA LIBERA DELLO STACK, QUINDI PER ACCEDERE ALL'ULTIMO ELEMENTO INSERITO DOBBIAMO PER FORZA SPOSTARCI INDIETRO DI UNO), LEGGE IL VALORE PUNTATO DALLO STACK POINTER E LO ELABORA.
- PUSH AD OGNI PUSH IL PROCESSORE ~~LEGE~~ LEGGE IL VALORE DA MEMORIZZARE, LEGGE IL VALORE DELLO STACK POINTER E VI SALVA SUBITO IL DATO LÌ (LO SI PUNTA ALLA PRIMA CELLA LIBERA), INFINE INCREMENTA LO STACK POINTER.
- SI CAPISCE QUINDI CHE A MENO DI SOVRASCRIZIONI (CIOÈ POP SEGUITE DA PUSH) LA MEMORIA NELLO STACK NON CAMBIA, VARIANO SOLO LA POSIZIONE DELLO STACK POINTER.
- OGNI VOLTA CHE SETTIAMO UN REGISTRO CON UN VALORE O ESEGUIAMO ~~UNA POP~~ UNA POP (CHE DIRETTO VUOL PROPRIO DIRE SETTARE UN REGISTRO CON UN VALORE, INFATTI DA QUALCHE PARTE DOVREMO PUR METTERE I VALORI CHE ABBIAMO ESTRATTO CON LA POP DALLO STACK) VENGONO ALLORAVANTI I FLAG ZERO E NEGATIVO.
- OGNI VOLTA INVECE CHE ESEGUIAMO UN'OPERAZIONE AD ESEMPIO INC, DEC, SUB, ADD VENGONO ALLORAVANTI TUTTI I FLAG.

- QUANDO UN PLA VIENE IMPOSTATO (AD ESEMPIO A SEGUITO DI UNA SOTTADONE) ESSO NON CAMBIERÀ FINCHÉ NON VENIRÀ ESEUITA UN'ALTRA OPERAZIONE CHE LO MODIFICHERÀ.
- QUANDO VIENE FATTA UNA CALL AD UNA FUNZIONE IL PROGRAMMA DEVE MEMORIZZARE IN QUALCHE MODO IL PUNTO DELL'ESECUZIONE A CUI ERA ARRIVATO, COSÌ AL TERMINE DELLA FUNZIONE POTRÀ RIPRENDERE DA DOVE AVEVA LASCIATO. PER FARE CIÒ SALVA NELLO STACK AN IL VALORE INCREMENTATO DI UNO ~~INCREMENTO~~ DELL'INDIRIZZO DELLA ISTRUZIONE CALL (DI FATTO SALVA NELLO STACK IL PROGRAM COUNTER INCREMENTATO) CON UNA PUSH. QUANDO VIENE FATTA UNA RET IL PROGRAM COUNTER VIENE IMPOSTATO CON IL VALORE LETTO DA UNA POP DELLO STACK, DOVE PRIMA CI ERAMO PREOCCUPATI DI SALVARE IL PUNTO A CUI ERAVAMO ARRIVATI.
- È CHIAMO CHE LE FUNZIONI RICORSIVE SONO POSSIBILI SOLO GRAZIE ALLA PRESENZA DI QUESTO STACK, SENZA IL QUALE NON SI SAREBBERO POTUTI SALVARE GLI INDIRIZZI DA CUI PARTIRE AL TERMINE DI UNA FUNZIONE.

## IL COMPILATORE

IL COMPILATORE È IL PROGRAMMA CHE SI OCCUPA DI TRADURRE IL CODICE SORLENTE (UN FILE ASCII) IN UN FILE OGGETTO (IN BINARIO). QUEST'ULTIMO VENIRÀ POI REMPLITO CON TUTTE LE FUNZIONI DI LIBRERIA CHE GLI SERVIRANNO, INFINE SI AVRÀ IL FILE ESECUIBILE (IN BINARIO):





IN PARTICOLARE IL COMPILATORE SI OCCUPA DI TRADURRE:

- DICHIARAZIONI DI VARIABILI
- ESPRESSIONI (ISTRUZIONI ARITMETICO-LOGICHE)
- ISTRUZIONI DI CONTROLLO (IF, WHILE ...)

DI SEGUITO ANALIZZEREMO IN DETTAGLIO COME VENGONO TRADOTTE DAL COMPILATORE QUESTE 3 CATEGORIE DI ISTRUZIONI.

## DICHIARAZIONI

INIZIAMO CON UN ESEMPIO:

int n = 5;

n : word 5

char v[5];

v : byte 0 ← AL PRIMO VALORE

byte 0 VIENE ASSOCIATA UN'

byte 0 ETICHETTA

byte 0

byte 0

└ LO ZERO INDICA CHE

NON SONO INIZIALIZZATI

SI POSSA VEDI CHE NON VIENE  
MEMORIZZATO DA NESSUNA PARTE  
CHE V È LUNGO 5, STA AL  
PILLOTTATORE TENERNE TRACCA

## ESPRESSIONI

PER LE ESPRESSIONI LA TRADUZIONE NON È COSÌ LINEARE, NEL SENSO CHE  
LE OPERAZIONI DOVRANNO ESSERE DIVISE E QUINDI ESEGUITE IN PIÙ FASI:

$n = x + y + 3$

LDWI R0 3

LDWA R1 x

ADD R1 R0

LDWA R1 y

ADD R1 R0

STWA R0 n

ANZITUTTO VENGONO  
CARICATI I VALORI COSTANTI

POI SI PROCEDE  
CON LE VARIABILI

int n, x = 12, y = 64

n : word 0 ← NON È INIZIALIZZATA

x : word 12 ← SONO IN INIZIALIZZAZIONE

y : word 64 ← SONO IN INIZIALIZZAZIONE

ISTRUZIONI DI CONTROLLO

NEI SEGUENTI ESEMPI (IN TUTTI QUANTI !!) SI FÀ LA SEGUENTE

IPOTESI :

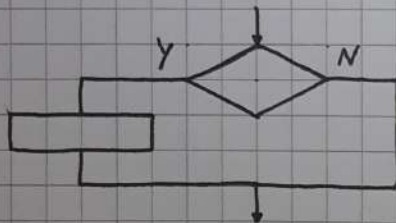
FLAG Z = 0  $\Leftrightarrow$  CONDIZIONE VERA  
 FLAG Z = 1  $\Leftrightarrow$  CONDIZIONE FALSA

IN REALTÀ PUÒ ANCHE ESSERE UTILIZZATO IL FLAG N PER STABILIRE IL RISULTATO DI UNA DISUGUAGLIANZA (SI EFFETTUA LA SOTTRAZIONE E POI SI GUARDA AL FLAG N).

AD OGNI MODO ECCO COME VENGONO TRADOTTE LE ISTRUZIONI DI CONTROLLO NEL LINGUAGGIO C :

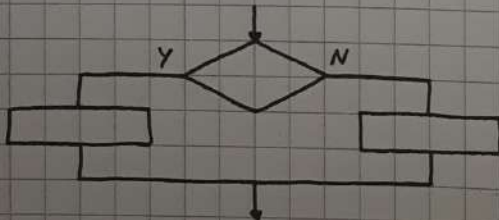
IF

IN QUESTO E NEI PROSSIMI CASI PROPORRÒ LO SCHEMA A BLOCCHI DELL'ISTRUZIONE CON A PIANCO LA TRADUZIONE :



SE Z = 1  $\leftarrow$  JMPZ cont  
 SALTA QUI  
 ...  
 cont : ...

ISTRUZIONI DELL'IF  
 CONTINUA DEL PROGRAMMA

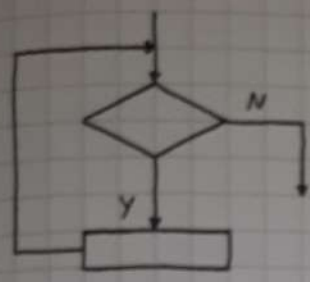
IF ELSE

JMPZ else  
 ...  
 SE Z = 1  
 OLTRE ALTRIMENTI  
 SALTO ELSE  
 JMP cont  
 else : ...  
 cont : ...

ISTRUZIONI "THEN"  
 ISTRUZIONI "ELSE"  
 CONTINUA DEL PROGRAMMA



WHILE



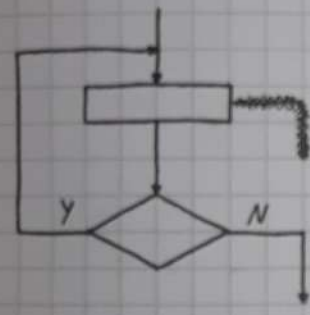
loop: . . . } QUI VIENE VALUTATA LA  
                  . . . } CONDIZIONE, E SE RIENTRA  
                              . . . } ALL'INIZIO IL FLAG

SE Z=1 LA ← JMPZ cont  
CONDIZIONE È FALSA  
E AVIZI SALTA . . . QUI CI SONO LE ISTRUZIONI  
                              . . . DEL CICLO

JMP loop

cont: . . . CONTINUA AGLI INDIRIZZI

DO WHILE



loop: . . . ISTRUZIONI DEL CICLO  
                  . . .  
                  . . . } QUI VIENE VALUTATA LA  
                              . . . } CONDIZIONE E VENGONO  
                                      . . . } ATTORNATI I FLAG

SE Z=0 LA  
CONDIZIONE È FALSA  
E AVIZI SALTA IL CICLO ← JMPNZ loop

cont: . . . CONTINUA AGLI INDIRIZZI

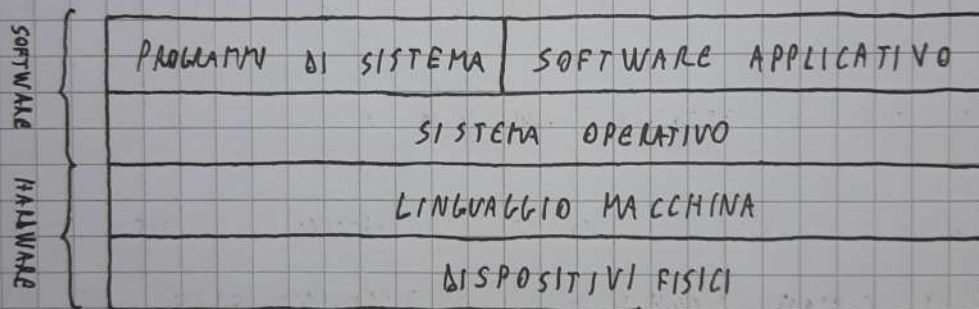
## IL SISTEMA OPERATIVO

I MODERNI CALCOLATORI HANNO UN HARDWARE COSÌ COMPLICATO DA RENDERNE IMPOSSIBILE LA GESTIONE DIRETTA, ISTRUZIONE PER ISTRUZIONE, DA PARTE DELL'UTENTE.

È PER QUESTO MOTIVO CHE DISPONGONO TUTTI DI UN SOFTWARE DI BASE, CHIAMATO **SISTEMA OPERATIVO**, CHE FILTRA (o SEMPLIFICA) LE OPERAZIONI DELL'UTENTE SULL'HARDWARE.

DI FATTO VIENE MESSA A DISPOSIZIONE DELL'UTENTE UNA **MACCHINA VIRTUALE**, CIÒ È UNA VERSIONE DIVERSA (VIRTUALE) E SEMPLIFICATA DEL REALE HARDWARE CHE COMPONE IL COMPUTER.

IL SEGUENTE SCHEMA RIASSUME LA GERARCHIA DI UN PC:



DOVE CON PROGRAMMI DI SISTEMA SI INTENDONO TUTTI QUELLI CHE NON SONO FINALIZZATI AD UN OBIETTIVO SPECIFICO (ES. COMPILATORI, INTERPRETI...).

MENTRE CON SOFTWARE APPLICATIVO SI INTENDONO I PROGRAMMI CHE USATO NORMALMENTE (ES. SINCRU, WORD, EXCEL...).

## GESTIONE DELLE RISORSE

TUTTI I PROGRAMMI CHE CONOSCIAMO POGGIANO SUL SISTEMA OPERATIVO, ESSO PERMETTE DI GESTIRE FACILMENTE LE RISORSE HARDWARE. PER ESEMPIO ANCHE DISPOSITIVI SEMPLICI COME MONITOR E TASTIERE RICHIEDONO UN SACCO DI SOFTWARE PER PILOTARE L'HARDWARE (SI RICHIEDONO DISPOSITIVI).



DI FATTO NEL MONITOR C'É UN PICCOLO COMPUTER E IL SISTEMA OPERATIVO DEL NOSTRO COMPUTER DEVE WINDI CONTENERE IL SOFTWARE NECESSARIO ALLA COMUNICAZIONE CON IL COMPUTER, CHIAMEMO QUESTI MODULI SOFTWARE **DRIVER**.

PER ESEMPIO, AL SEMPLICE COMANDO:

```
fprintf (fp, "%s %d", str, num);
```

LA MACCHINA VIRTUALE MESSA A DISPOSIZIONE DAL SISTEMA OPERATIVO DEVE SVOLGERE PARECCHIE OPERAZIONI, TRA CUI:

- GESTIONE DEL BUFFER IN MEMORIA
- GESTIONE DELL' INTERRUPT
- DIALOGO CON IL CONTROLLER DEL DISCO PER AGGIORNARE LE INFORMAZIONI IN MEMORIA

LE VEDREMO PIÚ IN DETTAGLIO PIÙSSIMAMENTE.

TRA LE ALTRE COSE LA MACCHINA VIRTUALE É ANCHE IN CARICO DI GESTIRE LE RISORSE CONDIVISE CONTEMPORANEAMENTE. IN SOSTANZA SI POSSONO AVERE PIÚ UTENTI COLLEGATI AL PC CONTEMPORANEAMENTE SENZA IL RISCHIO DI PROBLEMI (BANALMENTE ANCHE PIÚ SCHEDE APERTE DA UNO STESSO UTENTE ~~PARALLELAMENTE~~ CONTEMPORANEAMENTE HANNO BISOGNO DI UNA GESTIONE DELLE RISORSE CONDIVISA).

OVERBORD IL TUTTO DEVE ANCHE LAVORARE IN AGITO.

## PROTEZIONE

DALLE CONSIDERAZIONI APPENA FATTE SULLA GESTIONE DELLE RISORSE SI PRESENTA UN GRANDE PROBLEMA LEGATO ALLA SICUREZZA INFORMATICA, TUTTI I DATI CHE VENGONO GESTITI DEVONO ESSERE IN UN QUALCUN MODO PROTETTI DA AZIONI MALDESTRATE O MALEVOLE DELL'UTENTE.

PER PROTEGGERE I DATI SONO STATI CREATI DAI SISTEMISTI ( COLORE  
CHE ~~DA~~ DANNO VITA AI SISTEMI OPERATIVI ) GLI UTENTI .  
ALLI UTENTI SONO ASSOCIATI DEGLI ACCOUNT ( IL TERMINE DERIVA DAL  
GERMO FINANZIARIO PERCHÉ INIZIALMENTE , QUANDO I COMPUTER ERANO POCCHI IN  
TUTTO IL MONDO , SI PAGAVA PER OGNI SECONDO DI UTILIZZO DELLA  
CPU DI QUESTI MAINFRAME ) I QUALI PERMETTONO DI AVERE ACCESSO  
SOLO A DETERMINATE "ISTRUZIONI" . DI FATTO UTENTI DIVERSI HANNO  
DIVERSE LIMITAZIONI NELLA GESTIONE DEL CALCOLATORE .

TIPICAMENTE ESISTONO 2 TIPI DI UTENTI :

- UTENTE **AMMINISTRATORE** : SI OCCUPA DI GESTIRE IL CALCOLATORE  
INSTALLANDO O DISINSTALLANDO PROGRAMMI , HA LA POSSIBILITÀ DI  
MODIFICARE IL SISTEMA OPERATIVO .
- UTENTE **NORMALE** : NON HA LA POSSIBILITÀ DI INSTALLARE O  
AGGIORNARE I PROGRAMMI E NON PUÒ MODIFICARE IL SISTEMA OPERATIVO .

QUESTA DISTINZIONE TRA I PRIVILEGI DEI 2 UTENTI È DI FONDAMENTALE IMPORTANZA,  
SE INFATTI CI TROVIAMO A LAVORARE CON L'UTENTE NORMALE E VENIAMO  
ATTACCATI DA UN MALWARE , QUEST'ULTIMO NON POTRÀ CAUSARE TROPPI  
DANNI DAL MOMENTO CHE IL NOSTRO ACCESSO ALLE FUNZIONALITÀ DEL COMPUTER  
È GIÀ LIMITATO . SE INVECE NELLA STESSA SITUAZIONE SIAMO LOGGATI CON  
L'UTENTE AMMINISTRATORE ALLORA IL MALWARE PUÒ AVERE EFFETTI DEVASTANTI .

IN REALTÀ NEI SISTEMI OPERATIVI PIÙ GRANDI SONO PRESENTI 3 TIPI  
DI UTENTI :

- SISTEMISTA , PUÒ FARE TUTTO
- OPERATORE , HA PER ESEMPIO ACCESSO AI FILE DI TUTTI GLI UTENTI
- UTENTE , HA ACCESSO SOLO AI SUOI FILE



## ESEMPIO

IL MECCANISMO CHE VIENE MESSO IN ATTO PER PROTEGGERE I DATI È ESTREMAMENTE SPECIFICO E ORA LO ANALIZZEREMO IN DETTAGLIO.

SUPPONIAMO CHE L'UTENTE 1 VUOLIA CANCELLARE UN FILE DELL'UTENTE 2. QUINDI L'UTENTE 1 FA LA RICHIESTA DI CANCELLAZIONE E QUESTA VIENE PRELEVATA DA UN DRIVER CHE SI OCCUPA DI ADEGUATAMENTE SVOLGERE TALE OPERAZIONE (DI CANCELLAZIONE). IL DRIVER IGÌ È STATO PROGRAMMATO PER VERIFICARE DI CHI SIA IL FILE CHE STA PER CANCELLARE, PERTANTO BLOCCA L'ESECUZIONE DELL'ISTRUZIONE DAL MOMENTO CHE A RICHIESTA NON È STATO IL PROPRIETARIO DEL FILE DA CANCELLARE. A QUESTO PUNTO L'UTENTE 1 POTREBBE PROVARE A MODIFICARE IL CODICE RELATIVO A QUESTO DRIVER, ELIMINANDO LE PARTI DOVE VIENE ESEUITO IL CONTROLLO DEL PROPRIETARIO.

È NECESSARIO ORA SPIEGARE CHE LA CPU PUÒ OPERARE IN 2 DIVERSE MODALITÀ: **USER MODE** E **KERNEL MODE**. IN MODALITÀ **KERNEL** SI PUÒ ACCEDERE DIRETTAMENTE ALL'HARDWARE E QUINDI SI PUÒ MODIFICARE IL SISTEMA OPERATIVO. NELLA MODALITÀ **USER** È COME SE ALCUNE ISTRUZIONI NON ESISTESSERO PIÙ, QUINDI È IMPOSSIBILE MODIFICARE IL CODICE DEL SISTEMA OPERATIVO (E QUINDI I DRIVER).

IL NOSTRO UTENTE 1 DOVRÀ QUINDI PROVARE AD ENTRARE IN **KERNEL MODE** PER RUSCIRE NEL SUO INTENTO. TUTTAVIA PER PASSARE IN **KERNEL MODE** È NECESSARIO EFFETTUARE LA RICHIESTA CON UN PROGRAMMA CHE DEVE RESIDERE NELLA ZONA DI MEMORIA DEDICATA A CONTENERE IL SISTEMA OPERATIVO (LA CPU È IN GRADO DI CANTARE DA CHE ZONA DELLA MEMORIA PROVIENE UN PROGRAMMA).

QUINDI AL NOSTRO UTENTE 1 NON RESTA CHE SCRIVERE QUESTO PROGRAMMA E PORLO NELLA ZONA DI MEMORIA DEDICATA AL SISTEMA OPERATIVO,

MA SORGE UN PROBLEMA, INEFFATTI QUANDO PROVIAMO A CANCELLARE DOVE ABBIAMO DETTO IL SISTEMA OPERATIVO È GIÀ STATO ESECUITO. E SEMBRA GIÀ STATO ESECUITO È STATO ANCHE GIÀ CANCELLATO IL MODULO CHE SI OCCUPA DI CARICARE I PROGRAMMI IN MEMORIA E QUEST'ULTIMO È PREESISTENTE ALLA NON CANCELLAZIONE NELLA STESSA ZONA DI MEMORIA DOVE STA IL SISTEMA OPERATIVO.

DUNQUE L'UTENTE SI DOVRÀ APPOLLIARSI ALLE FUNZIONI DEL SISTEMA OPERATIVO E COME ABBIAMO GIÀ VISTO ESSE ESECUONO IL CONTROLLO SUL MANUTENIMENTO DEL FILE.

## L'INTERPRETE

L'INTERPRETE È UN PROGRAMMA CHE ANALIZZA UNA PER UNA LE ISTRUZIONI DI UN CODICE SORLENTE, DIRETTAMENTE DA TASTIERA O DA FILE ASCII (PUÒ ESSERE USATA COME INTERFACCIA UTENTE DAL SISTEMA OPERATIVO).

L'INTERPRETE DEI COMANDI È UN PROGRAMMA ESECUIBILE CHE VIENE LANCIATO AUTOMATICAMENTE ALL'AVVIO DEL COMPUTER. ESSO RIMANE IN ATTESA DEI COMANDI DA PARTE DELL'UTENTE.

OGNI SISTEMA OPERATIVO DOTATO DI INTERPRETE DEI COMANDI PERMETTE DI AUTOMATIZZARE ~~VARIE~~ OPERAZIONI ~~DEI~~ CHE RICHIEDONO LUNGHE E RIPEITIVE SEQUENZE DI COMANDI.

L'INTERPRETE POI TRADUCE LE ISTRUZIONI UNA ALLA VOLTA (SE C'È UN CICLO CHE SI RIPETE 5 VOLTE CON 10 ISTRUZIONI ALL'INTERNO TRADURRÀ 50 ISTRUZIONI). AL CONTRARIO DEL COMPILATORE CHE LE TRADUCE TUTTE IN UNA (NEL COMPILATION-TIME).

## COMPONENTI DEL SISTEMA OPERATIVO

IL SEGUENTE ELENCO MOSTRA 4 ELEMENTI CHE IL SISTEMA OPERATIVO DEVE ESSERE IN GRADO DI GESTIRE:

- GESTIONE DEI PROCESSI, CIÒ È LA CAPACITÀ DI FAR GIRARE IN



PROGRAMMI CONTEMPORANEAMENTE (NON DIPENDE DAL NUMERO DI CORE).

- GESTIONE DELLA MEMORIA CENTRALE, CIÒ È LA CAPACITÀ DI AVERE IN MEMORIA PIÙ PROGRAMMI
- GESTIONE DELLA MEMORIA DI MASSA, CIÒ È LA CAPACITÀ DI ORGANIZZARE I FILE SU DISCO E IL LORO ACCESSO
- GESTIONE DELL'INPUT/OUTPUT, CIÒ È LA CAPACITÀ DI COMUNICARE CON I DISPOSITIVI PERIFERICI

PRIMA DI ANALIZZARE SINGOLARMENTE QUESTI 4 ASPETTI DARÒ QUALCHE DEFINIZIONE DI BASE PER POTER PROCEDERE PIÙ VELOCEMENTE DOPO.

### PROCESSI

UN PROGRAMMA IN ESECUZIONE È CHIAMATO **PROCESSO** (o TASK), AD OGNI PROCESSO È ASSOCIATO UN **PID** (PROCESS IDENTIFIER) CIÒ È UN NUMERO. OLTRE AL PID VIENE ASSOCIATO AD OGNI PROCESSO ANCHE UN **PCB** (PROCESS CONTROL BLOCK), CIÒ È UN INSIEME DI INFORMAZIONI SUL PROCESSO IN QUESTIONE (PID, STATO DI AVANGAMENTO, PRIORITÀ, MEMORIA ALLOCATA, CPU USATA, ...).

### DEFINIZIONI:

PROGRAMMI IN MEMORIA { MONOPROGRAMMAZIONE: 1 PROGRAMMA ALLA VOLTA OCCUPA LA MEMORIA  
MULTIPROGRAMMAZIONE: PIÙ PROGRAMMI ALLA VOLTA OCCUPANO LA MEMORIA

ESECUZIONE DEI PROGRAMMI { MONOTASKING: 1 PROGRAMMA ALLA VOLTA IN ESECUZIONE  
MULTITASKING: PIÙ PROGRAMMI IN ESECUZIONE CONTEMPORANEAMENTE

UTENTI { MONOUTENTE: 1 UTENTE PER VOLTA PUÒ LAVORARE  
MULTIUTENTE: PIÙ UTENTI POSSONO LAVORARE CONTEMPORANEAMENTE

## GESTIONE DEI PROCESSI

ESISTONO DIVERSI TIPI DI SISTEMI OPERATIVI, IN GRADO DI GESTIRE OPPORTUNAMENTE I PROCESSI, NE VEDIAMO ALCUNI ESEMPLI:

### OS MONITOR

SONO PRINCIPALMENTE UTILIZZATI PER ELETTRODOMESTICI (IL FAMOSO TASTIERINO DEL MICROONDE), SONO MONOTASKING E MANUTENTE. SONO MOLTO SEMPLICI E LETTI I COMANDI DA TASTIERA MOSTRANO I MESSAGGI SUL DISPLAY.

### OS BATCH

QUANDO LE CPU ERANO COSTOSE E NON ESISTEVA IL MULTITASKING QUESTI SISTEMI OPERATIVI ERANO IN GRADO DI PROCEDERE CON L'ESECUZIONE DI UN PROGRAMMA IMMEDIATAMENTE DOPO IL TERMINE DEL PRECEDENTE.

DA QUI SONO NATI I FILE .BAT CHE GESTIVONO (NEI MODERNI CALCOLATORI) IN AUTOMATICO UNA SEQUENZA DI COMANDI DEL SISTEMA OPERATIVO (QUELLI CHE NORMALMENTE SCRIVIAMO NEL CMD).

QUESTI FILE SONO IMPORTANTI PERCHÉ PERMETTONO DI AUTOMATIZZARE PROCESSI CHE POTREBBERO DURARE ORE O GIORNI (NON POSSIAMO STARE TUTTO IL TEMPO AVANTI AL PC AD ASPETTARE CHE TERMININO).

### OS MULTITASKING

QUESTI SISTEMI SI BASANO SULLA MULTIPROGRAMMABILE (CONSIGLIATA AI PIÙ PROGRAMMI IN MEMORIA) CHE VEDREMO PIÙ IN DETTAGLIO IN FUTURO. E SULLA CONDIVISIONE DELLE CAPACITÀ ELABORATIVE DELLA CPU (CHE ESEGUE 1 ISTRUZIONE ALLA VOLTA), DETTA **TIME-SHARING**.

MA COME FA LA CPU AD ESEGUIRE PIÙ PROGRAMMI CONTEMPORANEAMENTE?

IN REALTÀ LA CPU CONTINUA AD ESEGUIRE UN PROGRAMMA ALLA VOLTA, PERÒ

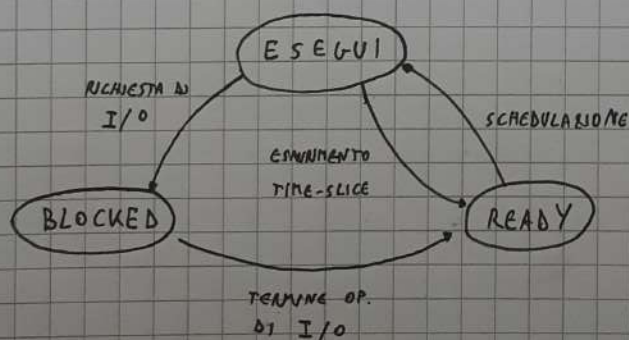


LO FA ALTERNANDO UN PEZZO DI UNO AD UN PEZZO DI UN ALTRO. QUESTA COSA AVVIENE COSÌ RAPIDAMENTE CHE NON CI ACCORGIAMO CHE OLV, AD ESEMPIO 10ms, CAMBIA PROGRAMMA E LASCIA IN STAND-BY QUELLO CHE ABBIAMO DAVANTI. CHIAMEREMO **TIME-SLICE** IL TEMPO CHE PASSA SU UN PROGRAMMA PRIMA DI PASSARE AD UN ALTRO.

OSSERVIAMO CHE LA TECNICA DI POLLING (SI RIMANE IN ATTESA DEL SEGNALE DI INPUT/OUTPUT) NON È CONCILIABILE CON QUESTO MULTITASKING DEL PROCESSORE\*. ECCO CHE VENGONO INTRODOTTI GLI **INTERRUPT**, CIOÈ UN COMANDO CHE METTE IN PAUSA UN PROCESSO QUANDO ESSO RICHIEDE UN I/O, PERMETTENDO ALLA CPU DI PROSEGUIRE CON L'ESECUZIONE DEGLI ALTRI PROCESSI. IL PROCESSO MESSO IN PAUSA VIENE RATTIVATO SOLO QUANDO VIENE RICEVUTO IL SEGNALE DI INPUT/OUTPUT RICHIESTO, LA CPU SI ACCORGE DI QUESTA COSA TRAMITE UN SEGNALE HARDWARE CHE LE VIENE INDIRIZZATO.

OGNI VOLTA CHE UN PROCESSO VIENE INTERRUPTO (PER ESAURIMENTO DELLA TIME SLICE O PER RICHIESTA OPERAZIONE DI I/O) IL SISTEMA OPERATIVO NE SALVA IL CONTESTO (**PSW**, PROCESS STATUS WORD, È SIMILE AL PCB) IN MODO TALE DA POTERLO FARE RIPARTIRE TALE COME A COME L'AVEVA LASCIATO E ~~PER~~ RATTIVA IL PRIMO PROCESSO IN ORDINE DI PRIORITÀ (QUESTA DECISIONE È PRESA DALLO **SCHEDULER**).

IL SEGUENTE SCHEMA (È UNA MACCHINA A STATI FINITI) RAPPRESENTA QUESTE CONSIDERAZIONI:



\* INFATTI AD OGNI RICHIESTA DI I/O DI UN PROCESSO LA CPU SI BLOCCHEREBBE SU QUEL PROCESSO SENZA POTER CONTINUARE L'ESECUZIONE DEGLI ALTRI PROCESSI.

OCCORRE FARE UNA DISTINZIONE TRA IL TEMPO TOTALE PER IL QUALE IL NOSTRO PROCESSO HA IMPIEGATO LA CPU (CPU TIME) E QUELLO CHE È SEMPLICEMENTE TRASCORSO DALLA SUA ATTIVAZIONE, MISURABILE CON IL CRONOMETRO (ELAPSED-TIME). SI AVrà SEMPRE:

$$\text{CPU-TIME} \leq \text{ELAPSED-TIME}$$

DOVE VALE L'UGUAGLIANZA SE E SOLO SE VIENE ESECUITO 1 PROCESSO E BASTA.



# GESTIONE DELLA MEMORIA CENTRALE

ESISTONO DIVERSE TECNICHE PER GESTIRE LA MEMORIA CENTRALE, ALCUNE DI ESSE SI BASANO SULLE PARTIZIONI MENTRE ALTRE SULLA MEMORIA VIRTUALE.

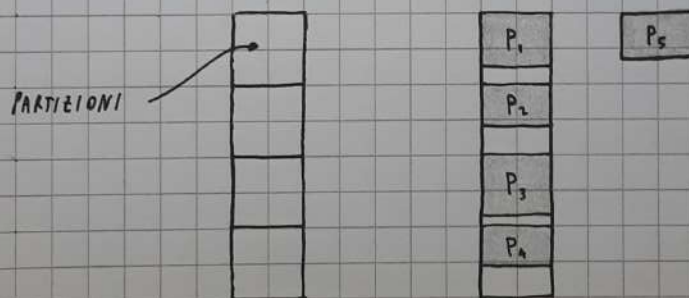
LO SCOPO RIMANE PERÒ COMUNE A TUTTE QUANTE:

- MASSIMIZZARE IL NUMERO DI PROCESSI CHE POSSONO UTILIZZARE LA MEMORIA
- MINIMIZZARE LO SPAZIO IN MEMORIA INVULGIZATO (NO FRAGMENTAZIONE)
- ESSERE SEMPLICE ED EFFICIENTE

PRIMA DI PARLARE DELLE TECNICHE REALMENTE UTILIZZATE NEI CALCOLATORI MODERNI È UTILE CAPIRE PERCHÉ SONO STATE SCARTATE QUELLE BASATE SULLE PARTIZIONI.

## PARTIZIONI FISSE

CON QUESTA TECNICA LA MEMORIA VIENE DIVISA IN PARTIZIONI DELLA STESSA GRANDIEZZA, OGNUNA DELLE QUALI CONTERRÀ UN PROCESSO. CONSIDERAMO IL SEGUENTE ESEMPIO:

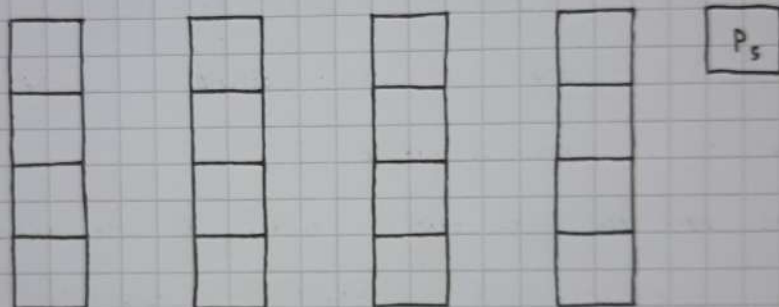


INIZIALMENTE LA MEMORIA È VUOTA E VIENE DIVISA IN PARTIZIONI, POI INIZIAMO A REMPLIRLA CON I PROCESSI  $P_1, P_2, P_3, P_4$  E CI ACCORGIAMO <sup>CHE</sup> NON REMPIONO TOTALMENTE LO SPAZIO MESSO A DISPOSIZIONE DALLE SINGOLE PARTIZIONI. SOMMANDO GLI SPAZI LIBERI DI OGNI PARTIZIONE CI SAREBBE IL POSTO PER INSERIRE IL PROCESSO  $P_5$  MA NON ESSENDO POSSIBILE UN RIANNUOVAMENTO NON POTREMO INSERIRLO (QUESTO FATTO SI CHIAMA FRAGMENTAZIONE INTERNA).

PUO' AVVENIRE UN CASO SIMILE CON UN PROCESSO CHE RICHIEDEREBBE, PER ESSERE INSENTO IN MEMORIA, PIU' DI UNA PARTIZIONE. IN QUESTO CASO, L'IMPOSSIBILITA' DI POTER INSERIRE QUESTO GRANDE PROCESSO SI DICE **FRAGMENTAZIONE ESTERNA**. COMPRESE LE LIMITAZIONI DELLE TECNICHE A PARTIZIONI FISSE VEDIAMO UN'ALTRA.

## **PARTIZIONI VARIABILI**

IN QUESTO CASO, COME SUGGERISCE IL NOME, CONSIDEREREMO UNA DIVISIONE DELLA MEMORIA IN PARTIZIONI DI GRANDEZZA VARIABILE. IN SOSTANZA OCCUPEREMO TUTTA LA MEMORIA INSERENDO I NUOVI PROCESSI SUBITO DOPO IL PUNTO DOVE TERMINANO I PRECEDENTI:



INIZIALMENTE ABBIAMO RIPIENO LA MEMORIA CON I PROCESSI  $P_1, P_2, P_3$ , AL TERMINE DEL SECONDO PROCESSO ABBIAMO INSERITO  $P_4$  E POI, AL TERMINE DI  $P_4$ , CI SIAMO DI NUOVO RIPIENITI CON UNA MEMORIA FRAGMENTATA DOVE E' IMPOSSIBILE FARE STARE IL PROCESSO  $P_5$ .

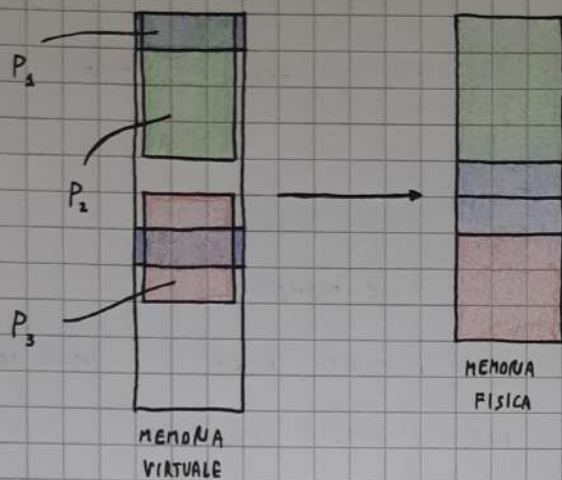
## **MEMORIA VIRTUALE**

LE TECNICHE CHE SI BASANO SULLA MEMORIA VIRTUALE HANNO COME SCOPO PRINCIPALE QUELLO DI CREARE UNA CORRESPONDENZA TRA INDIRIZZI DELLA MEMORIA FISICA E INDIRIZZI VISTI DAL PROGRAMMA (VIRTUALI).

DI FATTO OGNI PROGRAMMA CREDERE DI ESSERE L'UNICO (PERCHE' IN MEMORIA VIRTUALE PUO' USARE TUTTI I BYTE DISPONIBILI), IL SEGUENTE SCHEMA MOSTRA IL RUOLO DEL SISTEMA OPERATIVO NEL PASSARE DA MEMORIA ~~PHYSICAL~~ VIRTUALE A



## MEMORIA FISICA :



COME POSSIAMO OSSERVARE NELLA MEMORIA VIRTUALE È POSSIBILE CHE ALCUNI SI SOVRAPPONGANO (OCCUPINO) GLI STESSI INDIRIZZI, MENTRE NELLA MEMORIA FISICA VENGONO POSIZIONATI EFFICIENTEMENTE.

IL SISTEMA OPERATIVO EFFETTUA IL CALCOLO DELLE POSIZIONI IN MEMORIA FISICA ALLA **MEMORY MANAGEMENT UNIT (MMU)**, POSTA TRA LA CONTROL UNIT E L'ADDRESS BUS.

PRIMA DI PROCEDERE CON LE TECNICHE DI TRADUZIONE DELL'INDIRIZZO DA VIRTUALE A REALE OCCORRE SOTTOLINEARE CHE :

- OGNI PROCESSO HA ACCESSO A 2 <sup>PARALLELISMO</sup> INDIRIZZI DIVERSI NELLA MEMORIA VIRTUALE
- OGNI INDIRIZZO VIRTUALE PUÒ ESSERE TRADOTTO IN 2 <sup>FILE DELL'ADDRESS BUS</sup> INDIRIZZI DIVERSI NELLA MEMORIA FISICA (IN REALTÀ NON DIPENDE SOLO DAI FILE DELL'A.B. MA ANCHE DALLA QUANTITÀ DI MEMORIA RAM, CHE TALORA NON È STATA ESPANSA AL MASSIMO E QUINDI NON COINCIDE CON I FILE DELL'A.B.)
- OGNI PROCESSO PUÒ OCCUPARE TANTO SPAZIO QUANTO NE È STATO LASCIATO DALLI ALTRI PROCESSI IN MEMORIA.

DIAMO ORA 3 SEMPLICI DEFINIZIONI CHE SERVIRANNO A CAPIRE I PROSSIMI PARAGRAFI:

- **SEGMENTO**: È UN BLOCCO LOGICO DEL PROGRAMMA (PUÒ ESSERE IL PROGRAMMA INTERO COME UNA SINGOLA ISTRUZIONE)
- **PAGINA**: PARTIZIONE DELLA MEMORIA VIRTUALE DI UN PROCESSO DELLA STESSA DIMENSIONE DI UN PAGE FRAME
- **PAGE-FRAME**: PARTIZIONE DELLA MEMORIA FISICA DI DIMENSIONE FISSA (TUTTI I PAGE FRAME HANNO LA STESSA DIMENSIONE)

## LA SEGMENTAZIONE

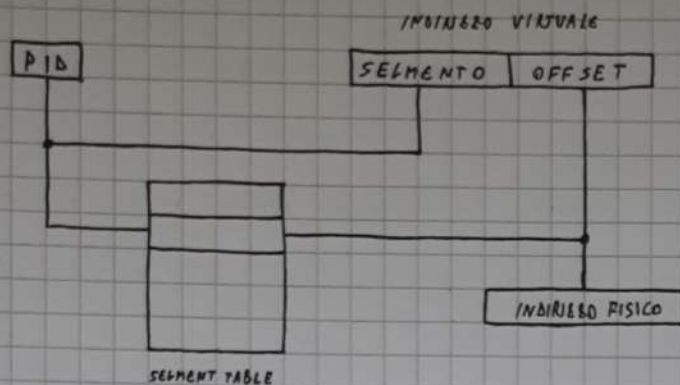
QUESTA TECNICA PREVEDE UNA DIVISIONE DEL PROGRAMMI E DEI DATI IN SEGMENTI. OGNI SEGMENTO CONTIENI UNA O PIÙ ISTRUZIONI, E QUESTE SARANNO MEMORIZZATE INSEITE NELLA MEMORIA VIRTUALE CON UN AGGREGATO AD UN INDIRIZZO CHE TIENE CONTO DEL SEGMENTO IN QUESTIONE E DELL'OFFSET TRA LA POSIZIONE DELL'ISTRUZIONE E L'INIZIO DEL SEGMENTO IN CUI SI TROVA.



OGNI PROCESSO HA INOLTRE UNA SPECIFICA **SEGMENT TABLE**, CIÒ È UNA TABELLA DOVE SONO MEMORIZZATI GLI INDIRIZZI DI BASE DI OGNI SEGMENTO PIÙ ULTERIORI INFORMAZIONI CHE PERMETTONO DI ASSEGNARE DELLE ETICHETTE AI VARI SEGMENTI (ES. SOLO LETTURA, SOLO SCRITTURA, PARTE DELL'OS ...).

UN TIPICO SCHEMA DI TRADUZIONE DELL'INDIRIZZO VIRTUALE IN LIVELLO FISICO È MOSTRATO QUI DI SEGUITO:





IN SOSTANZA IL PID PERMETTE DI AVERE L'INDIRIZZO DI BASE DELLA SEGMENT TABLE DEL PROCESSO IN CORSO (LA TABELLA È SEMPLICEMENTE CONTENUTA IN MEMORIA).

INDIVIDUATA LA SEGMENT TABLE SI VA A VERGARE L'INDIRIZZO DI BASE DEL SEGMENTO CHE CI INTERESSA (IL NUMERO DEL SEGMENTO INDIVIDUA LA RIGA DELLA TABELLA ALLA QUALE TROVEREMO L'INDIRIZZO DI BASE). UNA VOLTA FATTO CIÒ CI BASTA SOMMARE L'OFFSET ALL'INDIRIZZO DI BASE DEL SEGMENTO PER OTTENERE L'INDIRIZZO FISICO DELL'ISTRUZIONE.

UNO DEI VANTAGGI, OLTRE ALLA POSSIBILITÀ DI ETICHETTARE I SEGMENTI, STA NEL POTER RILOCARE I SEGMENTI IN QUANTO LE ISTRUZIONI AL LORO INTERNO FANNO SEMPRE RIFERIMENTO ALL'INIZIO DEL SEGMENTO.

## LA PAGINAZIONE

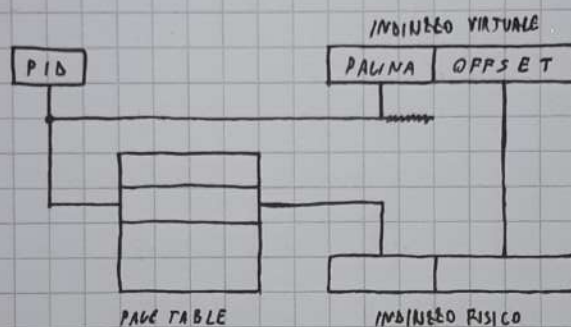
POICHÉ LA SEGMENTAZIONE HA IL PROBLEMA DI DIVIDERE IL PROGRAMMA IN SEGMENTI DI DIMENSIONE VARIABILE SI VERIFICA LA FRAMMENTAZIONE ESTERNA (MALGRADO LO SPAZIO PER INSERIRE IN MEMORIA C'È POCO È FRAMMENTATO IN BLOCCHI NON CONTIGUI). LA PAGINAZIONE DIVIDE IL PROGRAMMA IN PAGINE (PORZIONI DELLA MEMORIA VIRTUALE) E LA MEMORIA FISICA IN PARTIZIONI DELLA STESSA GRANDezza, CHIAMATE PAGE FRAME.

IL VANTAGGIO DI QUESTA SCELTA STA NEL FATTO CHE NONOSTANTE UN PROGRAMMA VEDA LE PAGINE CONTIGUE NON È DETTO CHE LO SIANO ANCHE I PAGE FRAME ALL'INTERNO DELLA MEMORIA FISICA, ANZI, È MOLTO PROBABILE CHE LÍ NON LO SIANO, PRIMO PER QUESTIONI DI EFFICIENZA.

AD ESEMPIO SE DOVESSIMO ~~avere~~ INSERIRE IN MEMORIA (CON PAGINE DA 4 KB)  
UN PROCESSO DA 16 KB POTREMO DOVER IMPIEGARE LE PAGINE FRAME 1, 3, 4, 8  
(NON PER FORA DEVONO ESSERE CONTIGUE). A PATTO TANTO CHE VENGANO MEMORIZZATI  
IN UNA **PAGE TABLE** GLI INDIRIZZI IN MEMORIA FISICA DELLE PAGINE DEL PROGRAMMA.

~~Però~~ LE PAGINE, E QUINDI LE PAGINE FRAME, SONO SEMPRE FORMATE DA  $2^n$   
BIT. IN QUESTO MODO GLI INDIRIZZI SIA DELLE PAGINE SIA DEI PAGE FRAME  
SARANNO MULTIPLI DI  $2^n$ , MENTRE GLI OFFSET SARANNO SEMPRE SOTTOMULTIPLI DI  
 $2^n$ . QUESTO RENDE IMMEDIATA LA SOMMA DI INDIRIZZO DI BASE DELLA PAGE FRAME  
E OFFSET (DI FATTO SI TRATTA DI UN ACCOSTAMENTO DEI VIGNONI).

DI SEGUITO UN TIPICO SCHEMA DI TRASLATIONE DELL'INDIRIZZO VIRTUALE IN  
LIVELLO FISICO:



MA L'INDIRIZZO FISICO DI UN PAGE FRAME E L'OFFSET POTREBBANO ASSOMIGLIARE A  
QUALCOSA DEL TIPO:

|                           |                             |
|---------------------------|-----------------------------|
| 1 0 1 1 1 0 1 0 0 0 0 0   | INDIRIZZO PAGE FRAME        |
| 0 1 1 0 1 0               | OFFSET                      |
| <hr/>                     |                             |
| 1 0 1 1 1 0 1 0 1 1 0 1 0 | INDIRIZZO FISICO ISTAURIONE |

## PAGINAZIONE PIÙ SEGMENTAZIONE

DAL MOMENTO CHE LA PAGINAZIONE PUÒ PORTARE A PAGE TABLE MOLTO GROSSE  
SI PUÒ DECIDERE DI USARE UNA PAGE TABLE DIVERSA PER OGNI SEGMENTO.



QUESTO PORTA POI A DOVER TENERE TRACCIA DEL SEGMENTO, DELLA PAGINA E DELL'OFFSET ALL'INTERNO DELL'INDIRIZZO VIRTUALE.

## PAGE FILE

QUANDO UN PROGRAMMA È IN ESECUZIONE NON È DETTO CHE SERVA AVERE TUTTE LE SUE PAGINE IN MEMORIA CENTRALE, PERTANTO VIENE SFRUTTATO IL **PAGE FILE**, CIOÈ UNA ZONA DEL DISCO DEDICATA A CONTENERE LE PAGINE TEMPORANEAMENTE INUTILIZZATE DEL PROGRAMMA IN ESECUZIONE.

NEL MOMENTO IN CUI LA RAM DOVESSA AVERE BISOGNO DI UNA PAGINA NE VERRÀ CERCATO L'INDIRIZZO TRAMITE UNA PAGE TABLE, SE SI DOVESSA SCOPRIRE CHE NON È PRESENTE IN MEMORIA SI VERIFICHEREBBE IL **PAGE FAULT**.

IN SOSTANZA SI VUOLE EFFETTUARE UN INSTRUCTION FETCH (PRIMA ISTRUZIONE DEL PROGRAMMA) MA NON SI HA IN MEMORIA L'ISTRUZIONE, QUINDI VIENE FATTA LA RICHIESTA AL DISCO <sup>AL PAGE FILE</sup> RITROVANDO QUELLA PAGINA.

TUTTAVIA POTREBBE NON ESSERCI PIÙ SPAZIO IN MEMORIA PER INSERIRE UNA NUOVA PAGINA, È QUI CHE SI RICORRE AD UN ALGORITMO DI GESTIONE DEI PAGE FAULT CHIAMATO **LEAST RECENTLY USED (LRU)**. QUEST'ULTIMO È UN ALGORITMO PIÙ MODIFICATO, NEL SENSO CHE TENDENZIALMENTE ~~IL PRIMO~~ IL PRIMO ELEMENTO AD ESSERE STATO INSERITO IN MEMORIA SARÀ IL PRIMO AD ESSERE MESSO SU DISCO (NEL PAGE FILE), TUTTAVIA SE DURANTE L'ESECUZIONE UNA PAGINA VIENE RUTILIZZATA ESSA TORNERÀ IN CIMA ALLA <sup>LISTA</sup> ~~PRIMA~~, FACENDO IN MODO CHE AD ESSERE SPOSTATE NEL PAGE FILE SIANO LE PAGINE MENO UTILIZZATE.

QUANDO ARRIVA IL MOMENTO DI SPOSTARE UNA PAGINA DALLA RAM AL DISCO HO INTERESSE NEL NO-CANCELLARLA SOLO SE È STATA MODIFICATA, ALTRIMENTI POSSO ANDARE AVANTI COME SE NULLA POSSA (LE PAGINE SONO GIÀ PRESENTI NEL DISCO, VERMANNO COPIATE DALLA RAM AL DISCO SOLO NEL CASO IN CUI POSSA STATA EFFETTUATA QUALCUNA MODIFICA).

TUTTAVIA PER SAPERE SE UNA PAGINA È STATA MODIFICATA O NO OCCORRE

AVERNE TENUTO TRACCIA, IN PARTICOLARE SI AGGIUNGERANNO ATTRIBUTI ALLA PAGE TABLE PER MEMORIZZARE, TRA LE ALTRE COSE (ES. SOLO SCRITTURA/LETTURA, PAGINA DI SISTEMA OPERATIVO...), ANCHE SE È STATA MODIFICATA (DIRTY BIT).

I PAGE FAULT RALLENTANO MOLTO IL COMPUTER, LE RICHIESTE CHE VENGONO FATTE AL DISCO RICHIEDONO INRATTI MOLTO TEMPO. PUÒ SUCCEDERE CHE UN PROCESSORE ENTRO IN UN CICLO DOVE OCCURRA TUTTO IL PROPRIO TEMPO AD GESTIRE PAGE FAULT (C'È BISOGNO IN CONTINUAZIONE DI PAGINE DAL DISCO). IN TAL CASO SI DICE CHE IL PROCESSORE È IN UNA CONDIZIONE DI THASHING E VENGONO COMPLETAMENTE IBERNATI ALCUNI PROCESSI. VENGONO NUOVAMENTE ATTIVATI QUANDO SARÀ POSSIBILE



## MEMORIA CACHE

Ci si accorge presto che la CPU è in grado di lavorare a velocità più alte rispetto a quelle richieste per l'accesso in memoria centrale. Per questo viene introdotta la **memoria cache**, una memoria posta fra la RAM (memoria centrale) e la CPU. Essa è estremamente più piccola e molto più veloce della RAM (nei calcolatori moderni è posta internamente al processore) e ha lo scopo di mettere a disposizione della CPU immediatamente alcune informazioni (data la sua grandezza non è in grado di contenere interi programmi ma solo alcune istruzioni).

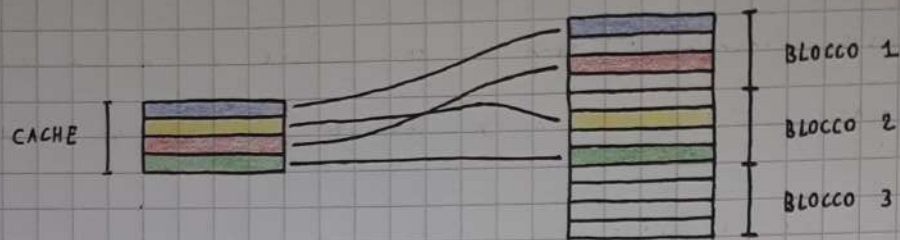
Ma quindi il processore ha bisogno di capire molto velocemente se il dato che sta cercando si trova o no all'interno della cache. Per rispondere a questa domanda, e quindi stabilire se ci troviamo nello stato **HIT** (il dato è nella cache) o **MISS** (il dato non è nella cache), si sfrutta il principio di località dei programmi. Quest'ultimo afferma che quando accediamo ad un dato in memoria è altamente probabile che con la prossima istruzione ci troviamo vicini spazialmente al dato di partenza (solitamente i nostri programmi sono sequenziali) o comunque al massimo distanti qualche decina di istruzioni.

Di seguito vedremo i principali modelli di memoria cache e i principi di funzionamento.

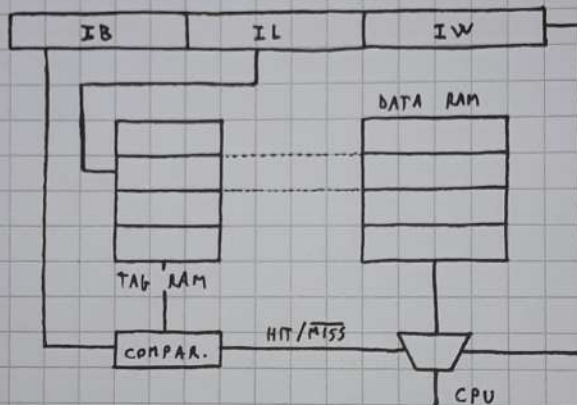
### CACHE A MAPPATURA DIRETTA

Con questo modello ogni volta che accedo alla cache so se il dato c'è o devo prenderlo dalla memoria RAM, non ci sono altre alternative (in altri modelli ci sono più posti dove cercare). In particolare la cache è divisa in  $2^L$  linee, le quali contengono  $2^W$  word, in seguito la memoria centrale viene divisa in  $2^b$  blocchi grandi quanto la cache.

LA POSIZIONE DELLE LINEE NEI BLOCCHI (QUINDI STIAMO PARLANDO DELLE LINEE NELLA RAM) CORRISPONDE ALLA POSIZIONE CHE AVRANNO QUANDO SARANNO NELLA CACHE:



DI SERVITO INVECE TROVIAMO UN DIAGRAMMA CHE RAPPRESENTA LA LOGICA CON LA QUALE SI CARISCE SE IL DATO È PRESENTE O NO NELLA CACHE.



IN SOSTANZA LE LINEE VENGONO DIVISE IN 3 PARTI, IW SERVE AD IDENTIFICARE LA WORD DI INTERESSE ALL'INTERNO DELLA LINEA (SE AVREMO HIT=1), IL IDENTIFICA LA LINEA, CIÒ È LA POSIZIONE ALL'INTERNO DEL BLOCCO. NEL TAB RAM (È UNA TABELLA) C'È SCRITTO IL BLOCCO AL CUI RIVIENE LA LINEA IL, CHE VIENE CONFRONTATO CON IB, CIÒ È IL BLOCCO DAL QUALE VOGLIAMO LEGERE LA LINEA IL. IN CASO ALTERNATIVO SI LEGGE IL CONTENUTO DELLA LINEA IL DAL DATA RAM, IN CASO CONTRARIO USANDO LA MEMORIA CENTRALE PER IL DATO.

ESISTONO TRE TIPI DI CACHE: **PARTIALMENTE ASSOCIATIVE** CHE PREVEDONO L'USO DI PIÙ TAB RAM E QUINDI DATA RAM O ASSIETTURA **COMPLETAMENTE ASSOCIATIVE**, CHE QUINDI PERMETTONO DI COPIARE OGNI LINEA DELLA MEMORIA IN OGNI POSIZIONE



NELLA CACHE (PUOI VEDERLA COME UNA CACHE CON  $2^b$  TAG MM, ANCHE SE NON È VERO, PERÒ RENDO L'IDEA). QUESTI MODELLI POSSONO PRESENTARE IL PROBLEMA DI DOVER DECIDERE QUALE DATO (MA QUALE TAG MM) ESTIMARE PER LASCIARE SPAZIO AD UNO NUOVO IN CASO DI MISS. LA SCELTA DEV'ESSERE PRESA MOLTO RAPIDAMENTE, ED È PER QUESTO CHE MA (VARI) APPROCCI ABBIAMO:

- LRU (LAST RECENTLY USED)
- FIFO (FIRST-IN ~~AND~~ FIRST-OUT)
- RAND (A CASO) ( $\uparrow$  VELOCITÀ,  $\downarrow$  EFFICIENZA)

## INTERRUPT

L'INTERRUPT NASCE COME ALTERNATIVA PIÙ EFFICIENTE E NECESSARIA AL POLLING PER I PROCESSORI CHE LAVORANO IN TIME SHARING, E WINDI NON POSSONO ESSERE BLOCCATI FINO ALL'ARRIVO DI UN EVENTUALE SEGNALE DI I/O.

ABBIAMO VISTO IN SIM CPU CHE IL SEGNALE HARDWARE DI INTERRUPT PUÒ ESSERE ASSOCIATO AD UNA FUNZIONE, CHE VIENE CHIAMATA ALL'ARRIVO DEL SEGNALE.

I DISPOSITIVI DI INPUT/OUTPUT SONO IN GRADO DI DARE QUESTO SEGNALE TRAMITE IL CONTROL BUS. IN PARTICOLARE, OGNI VOLTA CHE ARRIVA QUESTO SEGNALE VENGONO SVOLTI I SEGUENTI PASSAGGI:

- SI SALVA IL PROGRAM COUNTER NELLO STACK
- SI ESOLVE LA ROUTINE DI SERVICE ~~LA ROUTINE DI SERVICE È LA ROUTINE DI SERVICE~~
- VIENE SALVATA LA PSW (PROCESS STATUS WORD)
- VIENE CHIAMATA LA FUNZIONE CHE PUÒ OPERARE SUL DATO NECESSARIO (EVENTUALMENTE)
- VIENE RIPULITO IL PC E LO STATO DEI REGISTRI E DEL PROCESSORE.

IL FATTO CHE UN INTERRUPT NECESSITA DEL TEMPO PER ESSERE NECESSARIO ED ESEGUITO RENDE NECESSARIO MASCHERARE (DISABILITARE) GLI ALTRI INTERRUPT CHE POTREBBERO SOVRAPPORSI AL PRECEDENTE ANCHE IN ESECUZIONE. TUTTAVIA

ESISTONO SEMPRE ALCUNE LINEE DI INTERRUPT NON MASCHERABILI, AD ESEMPIO QUELLI RELATI AL SOTTOCARICAMENTO DEL COMPUTER, O QUELLI ALLA CASSA DI ALIMENTAZIONE, IN GENERALE QUELLI RELATI ALLE EMERGENZE. I MODERNI CALCOLATORI RENDONO DIFFICILE RUSCIRE A TROVARE SPAZIO PER L'HANDWARE DEGLI INTERRUPT, PERTANTO VENGONO ADOTTATI 2 APPROCCI DIVERSI:

- LINEE DI RICHIESTA INDIPENDENTI: OGNI INTERRUPT È DIRETTAMENTE COLLEGATO AL PROCESSORE
- CONTROLLER ESTERNO; GLI INTERRUPT SONO GESTITI DA UN DISPOSITIVO ESTERNO ALLA CPU CHE TRAMITE UN BUS (M) SELEZIONA L'INTERUP TO DA SEGUIRE E LO TRASMETTE ALLA CPU.

## DMA

SUPPONIAMO DI VOLER TRASFERIRE UNA GROSSA QUANTITÀ DI ~~NUMERO~~ BYTE NELLA MEMORIA DA UNA PERIFERIA. SOLITAMENTE QUESTA OPERAZIONE RICHIEDEREBBE ALCUNE ISTRUZIONI PER OGNI SINGOLO BYTE/WORD, IL CHE RENDEREBBE MOLTO LENTO IL PROCEDIMENTO. IN QUESTI CASI SI FA USO DEL DMA, CIOÈ UN <sup>E DELL'HANDWARE</sup> CONTROLLER DOTATO DI UN CONTATORE CHE ESERCE IL TRASFERIMENTO IN BLOCCO DI UN CERTO NUMERO DI DATI ~~E DELL'HANDWARE~~ (LO FA UN ARTO ALLA VOSTRA E IL CONTATORE MONITORA IL TRASFERIMENTO). VENGONO QUINDI ESERCITE SUI BUS SOLO LE OPERAZIONI DI LETTURA DA UNA PARTE E SCRITTURA ALL'ALTRA. È PER QUESTO MOTIVO CHE LA CPU VIENE SCOLLEGATA DAL BUS PER FARE IN MODO CHE ~~QUESTA~~ POSSA ENTRARE IN GIOCO IL CONTROLLER DEL DMA (CHE È COSTRUITO PER FARE UNA COSA SOLTANTO).



## GESTIONE INPUT - OUTPUT

SUPPONIAMO DI VOLER RIPRODURRE UN AUDIO SALVATO IN MEMORIA SU UNA SCHEMA AUDIO CHE CONVERTE IL SEGNALE NUMERALE IN ANALOGICO (DAC). I CAMPIONI DEVONO PERÒ ARRIVARE CON UNA CONFIGURAZIONE PREDEFINITA, SUPPONIAMO 2 CANALI DA 16 BIT CON FREQUENZA 44'100 HZ, MA COME SI SCRIVE IN C UN PROGRAMMA CHE ASPETTI  $1/44'100$  S PRIMA DI PASSARE I 32 BIT? NON SI PUÒ FARE, INFATTI NON SAREBBE COMPATIBILE CON IL TIME SPANNING, SECONDO CUI IL PROGRAMMA LIRA QUANDO HA A DISPOSIZIONE UNA TIME-SLICE. IN PRATICA VENLONO UTILIZZATI I **DRIVER**, CIOÈ INTERFACCIE TIPICAMENTE FORNITE DAI PRODUTTORI DI UN DISPOSITIVO PER POTER COMUNICARE CON QUESTULTIMO:



LE TECNICHE USATE PER SCAMBIARE DATI CON L'UNITÀ CENTRALE SONO I SOLITI:

- POLLING
- INTERRUPT
- DMA
- DMA + INTERRUPT

LA TRASMISSIONE AVVIENE UN BYTE ALLA VOLTA O INTERI BLOCCHI ALLA VOLTA (DMA) SI PRESENTA PERÒ UN PROBLEMA, UNO LEGATO ALLE TEMPISTICHE, NEL SENSO CHE IL NOSTRO PROGRAMMA NON SA QUANDO FORNIRE I DATI AL DRIVER. È UNO LEGATO AL FATTO CHE IL DRIVER È PRESENTE NELLA MEMORIA DEL PC, E COME TALE, PER POTER UTILIZZARE I RISULTATI DEL NOSTRO PROGRAMMA HA BISOGNO CHE

IL NOSTRO PROGRAMMA SIA LINKATO AL SUO INTERNO, MA QUESTO È IMPOSSIBILE,  
È ESCLUSO CHE PER STAMPARE UN FOGLIO WORD L'UTENTE DEGAZI I SOLENTI DI  
WORD E OGNA STAMPANTE PER FAR SI CHE STAMPINO IL SUO MOLETO.

PER QUESTO ESISTONO LE FUNZIONI DI **CALLBACK**, CIOÈ FUNZIONI CHE PERMETTONO  
DI COMUNICARE AL DRIVER IL NOME DELLA FUNZIONE CHE DEVE ESSERE A DISPOSIZIONE  
DELL'UTENTE PER METTERE A DISPOSIZIONE DEL DRIVER I DATI. (IN SOSTANZA  
LO SI DICE CHE QUANDO AVRA' BISOGNO DI DATI DEVE CHIEDERLI A UNA  
FUNZIONE). E SARÀ PROPRIO LA SOGGETTA A CHIAMARCI QUANDO AVRA'  
BISOGNO DI DATI.



## GESTIONE DEI FILE

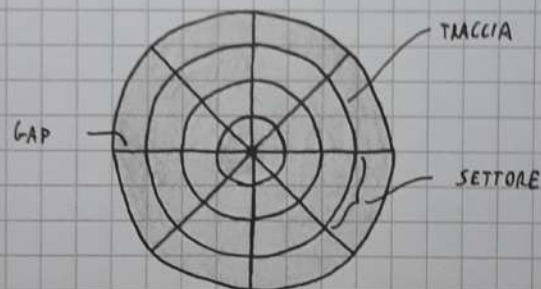
OGNI SISTEMA OPERATIVO PERMETTE DI MEMORIZZARE I DATI NELLA MEMORIA DI MASSA SOTTO FORMA DI **FILE**. QUESTI SONO POI ORGANIZZATI SECONDO STRUTTURE GERARCHICHE (HFS) CHE PREVEDONO L'ESISTENZA DI ALCUNI FILE PARTICOLARI, DETTI **DIRETTORI**, UTILIZZATI PER RAGGRUPPARE I FILE.

CHIAMIAMO INVECE **RADICE** IL PADRE (SONO I DIRETTORI A CUI APPARTIENE UN FILE) DI PIÙ ALTO LIVELLO. MENTRE I SISTEMI UNIX HANNO UN UNICA RADICE I SISTEMI WINDOWS HANNO UNA RADICE PER OGNI PER DISCO.

DI SEGUITO VEDREMO 2 METODI USATI PER LA MEMORIZZAZIONE DEI FILE IN MEMORIA.

## DISCHI MAGNETICI

I DISCHI MAGNETICI SONO COSÌ STRUTTURATI:



IL **SETTORE** È L'UNITÀ MINIMA DI RISTRUTTURAZIONE DEI FILE NEL DISCO. QUANDO ESEGUIAMO UNA `printf` I DATI VENGONO SALVATI SU DISCO QUANDO IL BUFFER CONTIENE ABBASTANZA BYTE DA RIEMPIRE UN SETTORE. LA **TRACCIA**, RAPPRESENTATA DAI ~~NUMERI~~ CERCHI CONCENTRICI, È UN INSIEME DI SETTORI.

PER IDENTIFICARE UN SETTORE SONO NECESSARIE 3 COORDINATE: DISCO (SPESSO SONO UNITI VERTICALMENTE A FORMARE DEI CILINDRI), TRACCIA, SETTORE.

LA CAPACITÀ DI MEMORIZZAZIONE DI UN DISCO DIPENDE DAI SETTORI PIÙ PICCOLI, INRARI IN ESSI I BIT SARANNO PIÙ VICINI, DAL MOMENTO CHE OGNI SETTORE PERIFERICO IL NUMERO DI BIT DEVE ESSERE LO STESSO (PER COMUNITÀ SI VUOLE CHE LA CAPACITÀ NON CAMBI) VERRÀ

SFRUTTATA MENO LA CAPACITÀ DI MEMORIZZAZIONE DELLA SUPERFICIE.

IN FASE DI LETTURA LA VELOCITÀ ANGOLARE CON LA QUALE VIENE FATTO RUOTARE IL DISCO È COSTANTE. IN PARTICOLARE CHIAMEREMO **SEEK TIME** IL TEMPO NECESSARIO AFFINCHÉ LA TESTINA SI POSIZIONI SULLA TRACCIA CORRETTA, MENTRE CHIAMEREMO **LATENCY TIME** IL TEMPO NECESSARIO PERCHÉ LA TESTINA SI TROVI SOTTO IL SETTORE CERCATO.

## DISCHI OTTICI

I DISCHI OTTICI SONO PENSATI DA UN'UNICA TRACCIA A SPIRALE DAL CENTRO DEL DISCO VERSO L'ESTERNO. QUESTO PERMETTE, MANTENENDO FISSA LA MANIGLIA DEI SETTORI, DI SFRUTTARE AL MASSIMO LA CAPACITÀ DI MEMORIZZAZIONE DELLA SUPERFICIE. IN FASE DI LETTURA / SCRITTURA LA VELOCITÀ LINEARE RIMANE LA STESSA, PERTANTO MANO A MANO CHE CI SPOSTIAMO VERSO L'ESTERNO LA VELOCITÀ ANGOLARE DIMINUISCE.

## GESTIONE DEI DISCHI

IL SISTEMA OPERATIVO ASSOCIA AD OGNI DISCO UNA **FAT** (FILE ALLOCATION TABLE), CIOÈ UNA TABELLA CHE ~~memorizza~~ PERMETTE DI LAVORARE CON I NOMI DEI FILE. È INFATTI POSSIBILE ESTRARRE DA QUESTA TABELLA ALCUNE INFORMAZIONI NECESSANTI I FILE (ES. LUNGEZZA, ~~la~~ POSIZIONE NEL DISCO, SCRITTURA / LETTURA, ...).