MICROCONTROLLORE : CPU, ROM, RAM, I/O per.

- CPU : gestisce ALU tramite REGISTRI
- MEMORIE :
  - **MEMORIA di PROGRAMMA** : contenitori da 16 bit
  - **REGISTRI INTERNI** : 32 registri da 8 bit (divisi in due banchi)
    - R1 R0
    - ⋮
    - R15 R14
    - R17 R16
    - ⋮
    - XH XL => X
    - YH YL => Y
    - ZH ZL => Z
  - MEMORIA dati estesa : 64Kb celle da 8 bit
    - REGISTRI PARTICOLARI
      - Program Counter : indirizzo istruzione che si sta per eseguire
      - SREG : bit di valore logico che gestiscono l'esecuzione (FLAG)

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C

        - C : CARRY => C = 1 risultato non rappresentabile (riporto nella somma)
        - Z : ZERO => Z = 1 risultato nullo
        - N : NEGATIVE => N = 1 risultato negativo
        - V : OVERFLOW => V = 1 risultato non rappr. (interi con segno)
        - S : segno effettivo operazione => S = N ⊕ V
        - H : riporto a metà operazione
        - T : "bit di appoggio"; memorizza/riprende bit di altri registri
        - I : bit di interruzione
      - Stack Pointer : meccanismo gestito come pila; contiene l'indirizzo della prima cella libera in cima allo STACK; usato per gestire operazioni di inserimento e prelievo
        - PUSH : inserimento nello stack (scrittura); oggetto letto viene posto nell'indirizzo indicato da SP, che viene decrementato
        - POP : prelievo dallo stack (lettura); valore di SP incrementato e viene prelevato il valore indicato

- INDIRIZZAMENTO :
  - IMMEDIATO : operando incorporato (costante numerica); lavorano con i registri da R16 in poi
  - DIRETTO : operando dichiarato direttamente mediante la memoria che lo contiene
  - INDIRETTO : operando è un puntatore (X, Y, Z)
  - INDIRETTO con AUTOMODIFICA : modifica il puntatore

master COPY

STRUTTURE di PROGRAMMAZIONE

- **SALTI CONDIZIONATI**: permettono di seguire strade diverse sulla base del risultato delle operazioni precedenti, in particolare in base ai flag
  - **SKIP**: condizione vera fa saltare l'istruzione seguente
  - **BRANCH**: condizione vera fa saltare all'indirizzo specificato in modo (ASSOLUTO ("indirizzo k"); RELATIVO ("avanti di n"))

- **RIPETI N-VOLTE (FOR)**: - si usa un registro contatore, decrementandolo K volte e, se ≠ 0, si ripete con BRANCH;
  - utilizza FLAG Z e salto BRANCH;
  - esegue il ciclo almeno una volta;
  - si inizializza un registro e dopo si pone la label che permette di ripetere il ciclo;
  - le istruzioni non devono interferire con il registro del conteggio

- **RIPETI FINCHÈ (WHILE)**: - struttura di attesa della condizione;
  - itera finché la condizione non è verificata;
  - no registri di appoggio; condizione valutata ad ogni iterazione e prima o poi deve permettere di uscire

- **RIPETI MENTRE (IF)**: - struttura sbloccata dal venire meno di una condizione
  - no registri di appoggio; condizione valutata ad ogni iterazione

---

- **SUBROUTINE**: sottoprogramma al termine del quale si torna al chiamante.
  Occorre salvare sullo stack l'indirizzo di ritorno; ciò avviene mediante una RCALL. Con la funzione RET si ripristina il PC
  - RCALL: chiama il sottoprogramma
  - RET: ritorna al punto di chiamata

N.B.: occorre sempre bilanciare PUSH e POP

- **INTERRUZIONE**: reazione automatica ad una richiesta o chiamata implicita ad una subroutine. Vengono abilitate con il flag I e se avviene l'evento previsto avviene il salto a INTERRUPT VECTOR (indirizzi della memoria che corrispondono a subroutine = INTERRUPT HADLER). Si ritorna alla situazione precedente con una RETI

master COPY

ALGEBRA di BOOLE

- **POSTULATI di HUNTINGTON**: dato un insieme B, due elementi $x, y \in B$ e due operazioni $+$ (OR) e $\cdot$ (AND):
  - $x+y \in B$ e $x \cdot y \in B$
  - $x+0 = 0+x = x$ elemento neutro OR; $x \cdot 1 = 1 \cdot x = x$ // // AND
  - $x+y = y+x$; $x \cdot y = y \cdot x$
  - $x+(y \cdot z) = (x+y)(x+z)$; $x \cdot (y+z) = xy + xz$
  - $x + \bar{x} = 1$; $x \cdot \bar{x} = 0$
  - RECIPROCITÀ: $+ \Leftrightarrow \cdot$; $0 \Leftrightarrow 1$

- **TEOREMI**:
  - elemento neutro (altro) dominante: $x+1=1$, $x \cdot 0 = 0$
  - complementarietà elementi neutri: $\bar{0}=1$, $\bar{1}=0$
  - idempotenza: $x+x=x$, $x \cdot x = x$
  - assorbimento: $x + xy = x$, $x(x+y) = x$ $(y \subseteq x)$
  - involuzione: $\bar{\bar{x}} = x$
  - somma col complemento: $x + \bar{x}y = x+y$, $x(\bar{x}+y) = xy$
  - associatività: $x+y+z = x+(y+z)$, $x \cdot y \cdot z = x(yz)$
  - legge di De Morgan: $\overline{x+y} = \bar{x} \cdot \bar{y}$, $\overline{xy} = \bar{x} + \bar{y}$

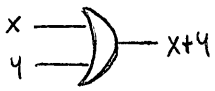
master COPY

FUNZIONI BOOLEANE

- **OBIETTIVO**: descrizione funzione tramite tabella di verità e ricavare da essa un'ESPRESSIONE BOOLEANA
- **DEFINIZIONI**:
  - LETTERALI: ciascuna occorrenza di una variabile
  - TERMINE PRODOTTO: singolo letterale o prodotto di essi
- **FORME NORMALI**
  - FORMA SP: somma di prodotti
  - FORMA PS: prodotto di somme
- **FORME CANONICHE**: i termini contengono tutte le variabili
  - FORMA SP: Σ MINTERMINI ~ termine prodotto costituito da tutte le variabili, che vale 1 solo per una combinazione
  - FORMA PS: Σ MAXTERMINI ~ termine somma costituito da tutte le variabili, che vale 0 solo per una combinazione

  Si considerano le righe della tabella per cui la funzione vale **1 (0)**, considerando le variabili affermate o negate
- **ESPANSIONE di SHANNON**: qualunque espressione booleana può essere ricondotta ad una forma canonica
  - SP: $f(x_1, x_2, \dots, x_i, \dots, x_n) = \bar{x}_i f(x_1, x_2, \dots, 0, \dots, x_n) + x_i f(x_1, x_2, \dots, 1, \dots, x_n)$
  - PS: $f(x_1, x_2, \dots, x_i, \dots, x_n) = [\bar{x}_i + f(x_1, x_2, \dots, 0, \dots, x_n)][x_i + f(x_1, x_2, \dots, 1, \dots, x_n)]$

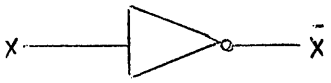
PORTE LOGICHE



OR



AND



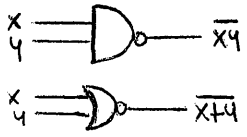
NOT

master COPY

RETI COMBINATORIE

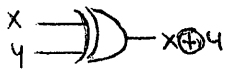
- implementano funzioni logiche tramite porte logiche interconnesse
- uscita completamente determinata dagli ingressi (assenza loop ingresso/uscita ⇒ reti ACICLICHE)
- CONDIZIONI DON'T CARE: ⇝ non fisicamente possibili o si ha un ingresso di cui non importa conoscere il comportamento della rete

- PORTE COMPLESSE:
  - PORTA NAND: prodotto logico negato
  - PORTA NOR: somma logica negata



PORTE UNIVERSALI: si possono utilizzare per realizzare tutte le altre porte

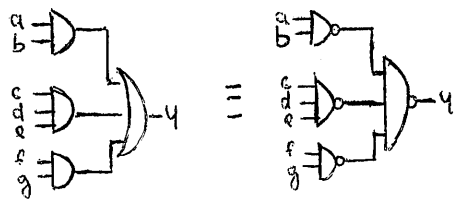
  - PORTA XOR: or esclusivo $x \oplus y = x\bar{y} + \bar{x}y = (x+y)(\bar{x}+\bar{y})$
    VERA solo se le variabili sono DIVERSE



↳ PROPRIETÀ:
- ⇝ $x \oplus 0 = x$ } l'OR esclusivo è un INVERTITORE a COMANDO, che cambia o lascia invariata x a seconda se diamo 0, 1
- ⇝ $x \oplus 1 = \bar{x}$ }
- ⇝ $x \oplus x = 0$, $x \oplus \bar{x} = 1$
- ⇝ $x \oplus y = y \oplus x$
- ⇝ $x \oplus (y \oplus z) = x \oplus y \oplus z$
- ⇝ $x(y \oplus z) = xy \oplus xz$
- ⇝ $x + y = x \oplus y \oplus xy$ ⇝ si ottiene la OR tramite AND e XOR
- ⇝ se $x \oplus y = z \Rightarrow y \oplus z = x$ e $x \oplus z = y$

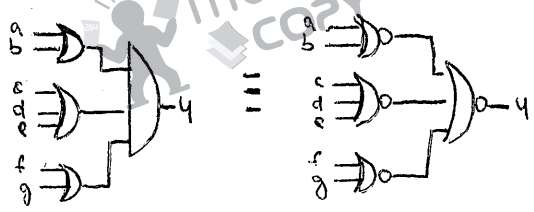
- FORMA NAND-NAND: corrisponde a SP

↳ $y = ab + cde + fg = \overline{\overline{ab} \cdot \overline{cde} \cdot \overline{fg}}$



- FORMA NOR-NOR: corrisponde a PS

↳ $y = (a+b)(c+d+e)(f+g) = \overline{\overline{(a+b)} + \overline{(c+d+e)} + \overline{(f+g)}}$



tramite de Morgan realizziamo funzioni come le porte SP e PS facendo uso di uno stesso tipo di porta universale

SINTESI di RETI COMBINATORIE

↳ **SINTESI con CRITERIO di OTTIMO**: • forme normali SP e PS con due livelli di logica
• minimo numero di letterali

↳ **TEOREMA di OTTIMALITÀ**: una forma normale SP ottima è composta esclusivamente da implicanti principali

↳ **IMPLICANTE PRINCIPALE**: implicante non contenuto in altri implicanti di ordine maggiore (e quindi con meno letterali)

↓

UTILIZZO delle **MAPPE di KARNAUGH** al fine di trovare un'espressione fatta di implicanti principali che formano la forma SP in maniera non ridondante

↳ **CRITERIO di ESSENZIALITÀ**: se tutte le caselle contenenti 1, adiacenti ad un mintermine, appartengono ad un medesimo implicante, allora questo è **ESSENZIALE**

↳ occorre trovare prima gli implicanti essenziali e poi quelli di ordine maggiore: se la funzione è coperta da tutti essenziali si ha la sintesi ottima

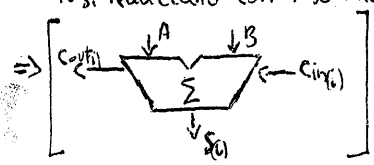
N.B.: i DON'T CARE sono usati come jolly

COMPONENTI COMBINATORI

• **SOMMATORI BINARI**: algoritmo somma binaria ~ $X+Y = \sum_{i=0}^{n-1}(x_i+y_i)2^i = c_n \cdot 2^n + \sum_{i=0}^{n-1} s_i \cdot 2^i$

~ 2·n ingressi e n+1 uscite (numeri a n cifre binarie)

~ si realizzano con n sommatori da 3 bit, cioè con i **FULL ADDER** ⇒



$S_i = |A+B+C_{in}(i)|_2$

$Cout_i = \frac{A+B+C_{in}(i)}{2}$ → 1 se $S_i \geq 2$, 0 altrimenti

A, B: addendi
Cin: riporto cifra precedente
Cout: riporto cifra successiva

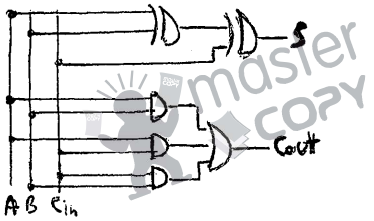
~ **SINTESI**:

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

MAPPE ~

$[S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in} = A \oplus B \oplus C_{in}]$

$[Cout = AB + AC_{in} + BC_{in}]$



~ **SOMMATORE PARALLELO RIPPLE-CARRY**: il bit del riporto si propaga dai bit meno significativi ai più significativi

master COPY