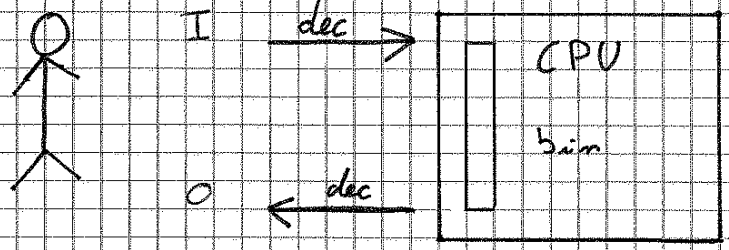
NUMERO: 694 DATA: 07/10/2013

A P P U N T I

STUDENTE: Martini

MATERIA: Calcolatori Elettronici

Prof. Reorda



Il programmatore lavora in decimale e la CPU in binario, affinché al programmatore tornino i risultati in decimale è necessaria un'operazione di conversione, che nei ling. di alto livello viene fatta dalle procedure di I/O (printf, scanf, ecc).

Programmando in assembler le conversioni dobbiamo farcele da soli, nel senso che dobbiamo inserire delle procedure che fanno le conversioni

Se l'omino inserisce 2197 da tastiera (cod. ASCII) una procedura assembler trasforma qst 4 cod. ASCII nel numero binario 2197 su 16 bit, ti fa tutte le operazioni e poi eventualmente produce un risultato che fa il comando inverso.

Il BCD è un formato in cui si prende ciascun carattere, quindi ciascuna cifra decimale e la si rappresenta su 4 bit.

Un numero BCD è un numero composto da tanti gruppi di 4 bit, detti NIBBLE, quante sono le cifre decimali che lo compongono.

L'×86 lavora in BCD ⇒ quindi possiede una serie di istruzioni • che permettono di fare oper. aritmetiche tra 2 numeri rappresentati in BCD.

• supportate dall'hardware

Se però in AL e DX ci sono 2 valori con segno → Signed
il processo è più complicato perché in AL c'è un numero in CA2,
nel bit 7 c'è il segno (a 0 o a 1 a seconda che sia positivo
o negativo).
Sugli altri 7 bit di AL c'è il modulo se il num. è positivo, ma
il valore in CA2 (negato +1) se il num è negativo.

Se il numero è positivo posso utilizzare il metodo precedente, ma
se è negativo devo ricordare che il primo bit di AL è 1 e
indica il valore negativo ma gli altri bit sono in CA2.

AL | 1 | . . . . . |

Se voglio estendere qst valore su 16 bit (qui AL) devo mettere il
primo bit a 1 per mantenere la negatività e poi riempire gli
altri 7 bit in 1, riporto 0 1 e in fondo vi sono i bit rimasti.

(1) . . . . . .

| 1 | 1 1 1 1 1 1 1 | 1 | . . . . . |

Quindi se voglio estendere su 16 bit un numero negativo che sta
originariamente su 8 bit dobbiamo aggiungere sugli 8 bit alti
tutti 1.

Ora diventa più difficile andare ad eseguire l'istruzione di
partenza (quella di sommare 2 registri di cui uno su 8 bit e uno
su 16 bit), inizialmente devo scoprire qual è il valore sul bit più
alto (1 o 0) per sapere se qst valore contenuto nel registro è negativo
o positivo ⇒ e scegliere il processo opportuno! *

Procedimento:
→ testare se il numero è positivo o negativo
→ mettere sugli 8 bit alti (qui di AH) o tutti 1 o tutti 0
→ infine sommo i 2 registri, ora entrambi su 16 bit.

- NEG operando

Serve per cambiare di segno un operando: il proc non modifica, quindi, solo il bit più significativo ma esegue una trasformazione da modulo a C2, e viceversa.

Lezione 16

- MUL operando → moltiplica 2 op. senza segno
- IMUL operando
  ↓
  moltiplica 2 op con segno

Operando: registro, cella di memoria, NO → costante
↓
può essere sia WORD sia BYTE

operandi
8 × 8 → 16 \
16 × 16 → 32 / risultato

Moltiplicazione su 16 bit:

MUL CX
CX * (AX)
↓
il 2° operando è implicito ed è sempre AX

⚠ → è su 16 bit
→ il risultato è sempre su DX, AX
(DX: ⊕, AX: ⊖)

Mul e Imul settano il flag di carry se la parte più significativa è a 0 o meno.

NB in una moltiplicazione tra word, i flag CF e OF valgono 0 se il registro DX è nullo

IMPORTANTE

La parte ALTA di RES va inizializzata a 0 perché non so se poi il risultato necessiterà anche dei 16 bit alti, quindi per non avere valori strani la metto a 0.
Altrimenti posso risolvere il problema con:

RES DD ∅

Ma se le istruzioni vengono riseguite, nessuno mi assicura che la parte alta di RES sia ancora a 0, ecco il perché di qsl istruzione .

- DIV operando → divisione tra num interi senza segno
- IDIV operando
  ↓
  divisione tra num interi con segno

Operando: registro, cella di memoria
NO → costante

↓

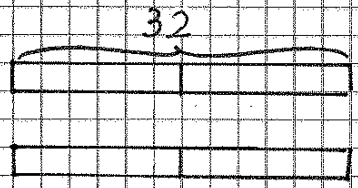
può essere sia WORD sia BYTE

Divisione su 8 bit:

	dividendo	divisore	quoziente	resto
byte	16	8	8	8
	1X	10	1A	AH

L'operando che io specifico è il divisore, gli altri sono impliciti.

Somma tra num. interi su 32 bit



Ma ho registri da 32 bit come risolvere il problema?

→ Li divido in: una parte + significativa e una parte meno significativa!

parte  significativa del risultato → risultato

Sommo, per dar vita al risultato, le parti - sign. tra loro, e faccio lo stesso con quelle + sign.

Devo però considerare che la somma tra le parti - sign. può generare un (carry) da riportare alla parte più significativa del risultato (cioè dal bit 15 al 16).

Quindi, innanzitutto, sommo i bit - sign dei 2 operandi: se non mi generano (carry) → OK! → sommo anche i bit + sign e ho finito. Altrimenti se ho un carry → sommo 1 alla parte + sign. del risultato

ALGORITMO

Sommi i bit - sign. dei 2 operandi

OTTIENI

bit - sign del risultato

se genera un carry

sommi i bit + sign degli op.

OTTIENI

bit + sign. del risultato

+1

se l'OK produce un carry

sommi i bit + sign. degli oper.

OTTIENI

bit + sign del risultato

```
MOV AX, WORD PTR NUM1+2
ADC AX, WORD PTR NUM2+2
MOV WORD PTR NUM3+2, AX
```

-> sommo la parte alta di NUM1, la parte alta di NUM2 e un eventuale carry

Ma la somma può anche generare OVERFLOW, posso controllare sul flag di overflow.

-> overflow

```
  0101 +
  1011 =
 -----
 10000
```

(1) overflow

NB -> posso sommare anche numeri su 64 bit, utilizzando ADD e ADC, sommando di volta in volta le word partendo da quelle meno significative.

- SBB dest, sorg

Ast istruzione considera il carry in una sottrazione: se CF=0 si comporta come una SUB, ma se CF=1 sottrae 1 al risultato con una SUB.

Lezione 17

Istruzioni di salto

Utilizzando le istruzioni di salto è possibile forzare il processore a continuare l'esecuzione senza eseguire l'istruzione successiva, ma saltando ad una diversa istruzione.

ista. di salto INCONDIZIONATO → istruz. che fa sì che il proc. salti
ista di salto CONDIZIONATO ad un'ista. che non è quella successiva.

quando il proc incontra qst ista., fa un test e a seconda del "risultato" salta o prosegue per ordine.

Istruzioni di salto INCONDIZIONATO

- JMP dest
  ↳ etichetta

Ogni qualvolta il proc. incontra qst ista salta, senza effettuare ulteriori verifiche dell'istruzione "dest".

Da un punto di vista dell'hardware del proc qst ista va a modificare il program counter o anche detto IP (per la terminologia Intel).

Ad ogni ista l'IP viene incrementato per accedere all'ista. successiva, con qst ista di salto il valore dell'IP viene modificato, opportunamente, per raggiungere l'ista. denominata "dest" al fetch successivo.

Quindi, (dal punto di vista hardware) il proc modifica l'IP, durante l'esecuzione di un'ista di salto, con l'offset dell'istruzione associata all'etichetta.

- CMP → usata da salto cond.

CMP dest, sorg

Esegue un confronto tra i 2 operandi, in pratica esegue la sottrazione dest - sorg senza restituire il risultato.

La CMP aggiorna opportunamente tutti i flag di stato.

Istruzioni di salto CONDIZIONATO relative ad un singolo FLAG

- JZ	salta se	ZF = 1		
- JNZ	"	ZF = 0		
- JS	"	SF = 1		
- JNS	"	SF = 0		
- JO	"	OF 1	↘ ↗	overflow
- JNO	"	OF 0	↘ ↗	overflow
- JC	"	CF 1	↘ ↗	carry
- JNC	"	CF 0	↘ ↗	carry
- JP o JPE	"	PF 1	↘ ↗	parità
- JNP o JPO	"	PF 0	↘ ↗	parità

NB "SALTO" → vai all'etichetta

Nella maggior parte dei casi

↓

```
[CMP dest, sorg
[Jxxx label
```

Semplifichiamo il tutto → Confronto tra numeri CON segno

JL o JNGE	salta se	dest < sorg	TESTANDO FLAG
JG o JNLE	"	dest > sorg	TESTANDO FLAG
JLE o JNG	"	dest ≤ sorg	TESTANDO 2 FLAG
JGE o JNL	"	dest ≥ sorg	TESTANDO 2 FLAG
JE = JZ	"	dest = sorg	
JNE = JNZ	"	dest <> sorg	

Istruzioni di salto CONDIZIONATO dal contenuto di CX

- JCXZ label → salta se CX=0 a label

Esempio

```
     MOV CX, K
ciclo: JCXZ lab
       ========
       DEC CX
       JMP ciclo
lab:
```

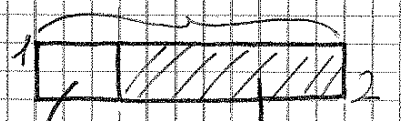
oppure

```
CMP CX, 0
JE label
```

Nel caso delle istr. di salto condizionato, se ha il test e se esso mi da un certo risultato avviene il salto e quindi se ha la modifica dell'IP.

Nel caso di salto incondizionati la modifica dell'IP avviene già al tempo d'esecuzione.

JMP label



cod macchina dell'istr.

↑ ISTR. DI SALTO

Al FETCH dell'istruzione successiva verrà caricata l'istr. a cui vogliamo saltare.

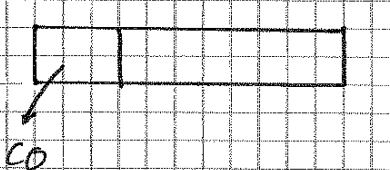
Quant'è lunga quest'istr.?!

(1 + 2) byte

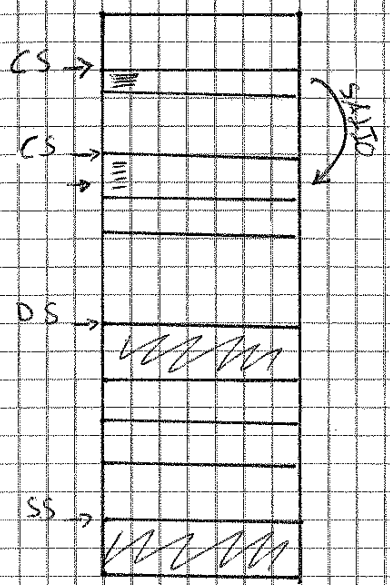
C. op ↙ ↘ offset

ISTR. DI SALTO

JE label



I Programmi "grandi" (veri e propri) stanno su vari segmenti: CS, DS ed SS sono quelli iniziali come sempre ma poi ve ne sono altri!

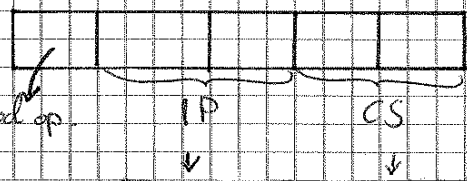


Finché il programma sta nel 1° segm, bisogna modificare solamente l'IP, ma se mi devo spostare in un segm. successivo anche il CS va modificato, oltre all'IP, qst passaggio è fattibile tramite un'istr. di SALTO e

SMP → su 5 byte

SMP

SMP label



specifica che è un'istr. di salto encond.

2 byte contengono il valore da caricare nell'IP e 2 nel CS

L'istr. succ viene ora caricata nel nuovo segm, l'ass. ha calcolato i valori da inserire nel cod. macchina

Riepilogo sui FORMATI:

- SHORT → istr di salto cond. → (2) byte
- NEAR → " " " encond. → (3) byte
- FAR → " " " " che permettono di saltare da un segm. all'altro → (5) byte

```
salta: INC SI          → passo al successivo carattere da
       DEC CX            analizzare
       CMP CX, 0
       JNE ciclo       MOV MINUSCOLE, AX
           MINUSCOLE, AX
       ...
```

Istruzioni di iterazione

- LOOP
- LOOPE
- LOOPNE

} facilitano l'implementazione di costrutti di tipo iterativo

- LOOP label

```
DEC CX
CMP CX, 0        LOOP implementa con una sola istr. queste 3
JNE ciclo
```

Quindi LOOP permette che il proc:

→ decrementi CX
→ controlla il valore di CX
  → se CX ≠ 0 salta all'istr. dell'etichetta
  → altrimenti esegue l'istr successiva

- LOOPE label

LOOPE e LOOPNE sono più complicate.

LOOPE permette che il proc:

→ decrementi CX
→ controlli i valori di CX e ZF
  → se CX ≠ 0 E ZF = 1 salta all'istr. dell'etichetta.

- LOOPNE label

Fa sì che il proc:

-> decrementi CX

-> controlli i valori di CX e ZF

-> se CX≠0 ZF=0 salta all'istr. dell'etichetta

-> altrimenti esegue l'istr. successiva.

Le istruzioni che modificano i flag:

STC -> CF=1
CLC -> CF=0
} settano forzatamente il bit di carry

Utili per passare il risultato di una procedura.

Istruzioni per la manipolazione dei BIT

Si suddividono in: IST. LOGICHE -> permettono di controllare o modificare uno o più bit

IST. DI SCORRIMENTO -> permettono di cambiare la posizione dei bit.

Es: istr. logiche -> AND, OR, ecc.
Es: istr di scorrimento -> ≪ ≫

AND AX, BX -> AND bit a bit tra AX e BX, il risultato va in AX
AX ← AX & BX

A	B	AND
0	1	0
1	0	0
0	0	0
1	1	1

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

Esempio

"Conversione da ASCII a binario"

```
[...]
ASCII DB ?
NUM DB ?
  [...]
MOV AL, ASCII
AND AL, 0FH   → mette a 0 i 4 bit alti
MOV NUM, AL
...
```

"Conversione da binario a ASCII"

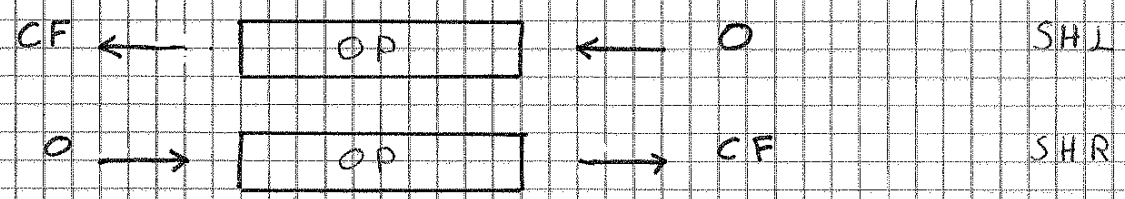
```
  [...]
ASCII DB ?
NUM DB ?
  [...]
MOV AL, NUM
OR  AL, 30H
MOV ASCII, AL
...
```

- TEST dest, sorg

Esegue una AND ma non mette il risultato in dest → NON modifica dest ma setta i FLAG

```
TEST AL, 0001 0000b   → il risultato sarà 0 se il 5° bit di AL è 0
JE lab
```

SCHEMA



Applicazioni

SHR → equivale ad una divisione per 2^n per interi senza segno.
SHL → equivale ad una moltiplicazione per 2^n per interi senza segno.

SHR AX, 2 → $AX/4$ $\quad 2^2 = 4$

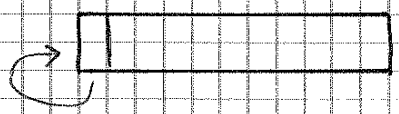
SHL AX, 4 → $AX \cdot 16$ $\quad 2^4 = 16$

~~DIV~~ e ~~MUL~~

NB vale per numeri SENZA segno!

E se il numero è CON segno?
È importante ricordare che il bit + alto è il SEGNO e deve rimanere tale.

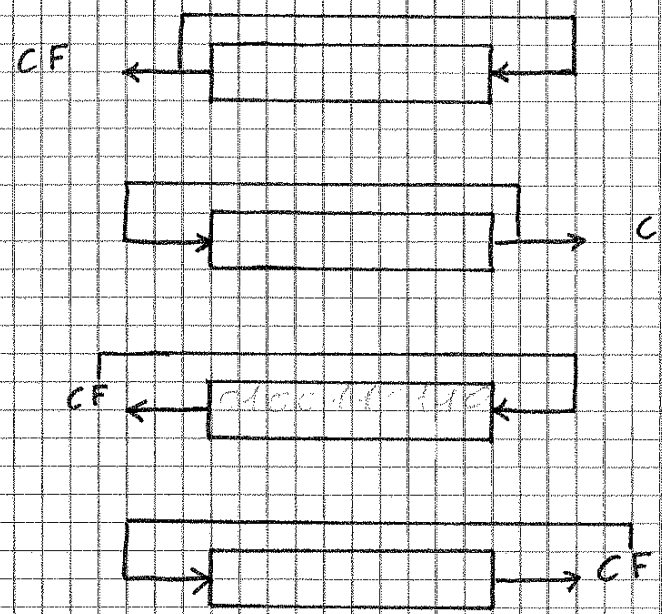
Quando se voglio dividere un numero con segno per una potenza di 2, devo:



1) garantire che il bit più significativo resti tale.
2) inserire, a partire dal 2° bit da sinistra, dei bit a 1 o a 0 a seconda che il numero originale era positivo o negativo

Istruzioni di Rotazione

Permettono di inserire, i bit che escono da un lato dall'altro.
→ Li fanno rientrare



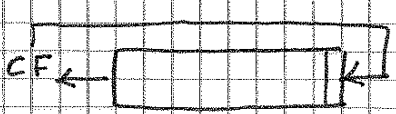
- ROL
- ROR

NB l'ultimo bit spostato assegna un valore a CF

- RCL
- RCR

CF fa parte della rotazione, quindi se faccio uscire 3 bit solo 2 rientreranno dall'altro lato perché l'ultima a uscire è nel CF!

RCL AX,3



Il progetto si concretizza a diversi livelli di dettaglio, ogni livello utilizza le librerie di componenti disponibili a livello inferiore.

C'è molta interazione tra i vari livelli. ←

Sistemi combinatori e sequenziali

↙ se i valori delle uscite dip esclusivamente dai valori applicati sui suoi ingressi in quell'istante.

↓

es. sommatore

↘ se i valori delle uscite dip sia dai valori correnti degli ingressi, sia dai valori applicati negli istanti precedenti.

↓

es. contatore

Per descrivere la funzione che voglio venga svolta da un modulo COMBINATORIO ho 2 modalità: - TAVOLA DI VERITÀ (per ciascuna combinazione di ingresso specifica la corrispondente combinazione di uscita)

- FUNZIONE BOOLEANA (implementata dalle uscite)

Se vogliamo fornire la specifica di un sistema SEQUENZIALE dobbiamo descrivere come le uscite dipendano dalla storia precedente.
Per fare ciò bisogna considerare la VARIABILE DI STATO che tiene conto della storia precedente, essa ci permette di conoscere tutte le informazioni degne di importanza.

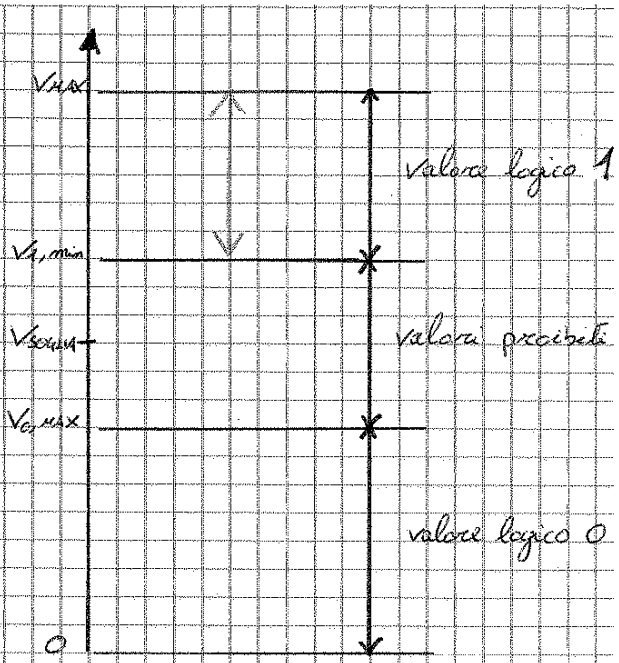
↓

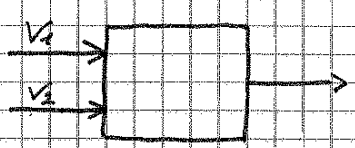
è il num. di transizioni che ci sono state sul segnale dall'ultima volta che vi è stato un RESET.

Esistono 3 modalità per spiegare il comportamento di un sistema sequenziale:

- TAVOLA DEGLI STATI/USCITE
- DIAGRAMMA DEGLI STATI/USCITE
- FUNZIONE DI TRANSIZIONE DEGLI STATI E FUNZIONE DELLE USCITE

Quando il componente "trova" una tensione che va da 0 a $V_{0,MAX}$ si ha 0.
Se la tensione è compresa tra $V_{1,min}$ e V_{MAX} si ha 1.
Il componente che il progettista sta realizzando ha degli ingressi e un'uscita.
→ tensione





Se V_1 e V_2 ricadono nella 1ª o nella 3ª "parte" allora OK! → il componente li interpreta come 0 e 1 rispettivamente
Se la tensione finisce nella zona alta (si brucia tutto, quindi V_{MAX}) sarà opportuno come specificato che nell'ingresso non siano applicate tensioni maggiori di V_{MAX}.

Se vengono applicate tensioni negative (al di sotto della 1ª zona) nessun assicura su quello che può succedere, se invece vengono applicate nella zona di mezzo, che è identificata come valori proibiti, probabilmente il dispositivo non si danneggia ma l'uscita sarà indeterminata.

L'impo di qst schema è permettere il passaggio da valori analogici (tensioni) a valori discreti binari in modo tale che da qst momento in poi si debba lavorare solo con 0 e 1.

Il sistema va messo insieme in maniera tale che possibilmente tutti i componenti a qst livello ricevano sempre delle tensioni che rientrino nelle zone previste, la zona dell'1 e la zona dello 0, se non in momenti transitori in cui la tensione passa dallo 0 all'1 attraversando la zona proibita, però l'impo è che non stia violando così le specifiche.

Se metto insieme 2 transistor (disegno a sinistra) ognuno con una tensione di controllo che rappresenta il segnale di ingresso, ottengo un componente che in uscita ha una tensione il cui valore dipenderà dalle tensioni in ingresso. Se una delle tensioni in ingresso è al valore basso vuole dire che uno dei 2 transistor è aperto, per contro se tutte e due le tensioni di ingresso sono al valore alto vuole dire che entrambi i transistor sono chiusi, e se entrambi i transistor sono chiusi V_{out} viene portato al valore basso (a massa)

A fronte di 2 valori alti in ingresso, in uscita ho un valore basso.
Con 2 valori bassi in ingresso, entrambi i transistor sono aperti e V_{out} non va a massa, e l'uscita ha valore alto.

$1\ 1 \rightarrow 0$
$0\ 0 \rightarrow 1$

Se in ingresso ho un valore alto e uno basso → i 2 transistor saranno uno chiuso e uno aperto, il cammino da V_{out} a massa è INTERROTTO e perciò rimane un valore alto.

$0\ 1 \rightarrow 1$
$1\ 0 \rightarrow 1$

1 1	0
0 1	1
1 0	1
0 0	1

Tavola di verità → trasformazione tra valori di tensione e bit.

↓

descritta tramite il simbolismo della porta NAND

Lezione 20

Reti nelle porte logiche

L'insieme di porte logiche da vita ad 1 circuito combinatorio, o meglio un modulo che implementa una funzione combinatoria.

es sommatore

> La domanda è: in che modo devo mettere insieme le porte logiche?

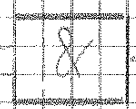
Per fare qst progetto posso lavorare sui (BIT).

AND

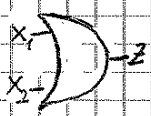
$Z = X_1 X_2 = X_1 \wedge X_2$

NAND

$Z = \overline{X_1 X_2} = \overline{X_1 \wedge X_2}$



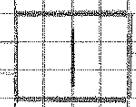
OR



$Z = X_1 + X_2 = X_1 \vee X_2$

NOR

$Z = \overline{X_1 + X_2} = \overline{X_1 \vee X_2}$



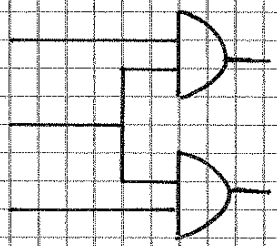
EXOR

$Z = X_1 \oplus X_2$

EXNOR

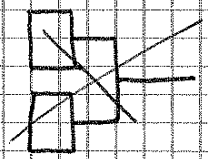
$Z = \overline{X_1 \oplus X_2}$

- attivare in parallelo 2 ingressi di 2 moduli (o sistemi) diversi ha un CCBF



- <u>non devono essere presenti cicli</u>

Per esempio <u>non</u> si possono collegare insieme 2 uscite 



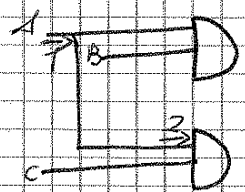
Fanin → numero di segnali in ingresso ad una porta

Fanout
↘ numero di altre porte pilotate dall'uscita di una porta.

, (num di porte influenzate dall'uscita di una det. porta)

Anche l'ingresso può avere fanout > 1.

A → fanout 2



Lezione 21

Voglio trovare il (circuito minimo) con il numero minimo di porte per implementare una determinata funzione booleana. → MINOR COSTO

Ciò può essere fatto in 2 modi: o implementando la funz. booleana, ovvero avendo la specifica del circuito sotto forma di funz. booleana qst può essere manipolata al fine di essere semplificata ma mantenendo lo stesso significato.

Prima di eseguire il passaggio da funz. booleana a CIRCUITO è (una) necessario che la funz. venga semplificata, una volta fatto è definita SOMMA DI PRODOTTI.

Algebra Booleana

È basata su: elementi (che possono essere a 0 o 1)

operatori (AND (·), OR (+), NOT)

assiomi:

- chiusura: $a \in K, b \in K \Rightarrow ab \in K, a+b \in K$
- identità: $a+0=a$, $a \cdot 1, a$
- commutatività: $a+b=b+a$, $a \cdot b = b \cdot a$
- distributività: $a(b+c) = a \cdot b + a \cdot c$, $a+(b \cdot c) = (a+b) \cdot (a+c)$
- inverso: $a\bar{a}=0$, $a+\bar{a}=1$

Leggi dell'algebra booleana

↘ dipendono dagli assiomi precedenti

- Associatività
  $a+(b+c) = (a+b)+c$
  $a(bc) = (ab)c$
- Idempotenza
  $a+a=a$
  $aa=a$
- Involuzione
  $\bar{\bar{a}} = a$
- De Morgan
  $\overline{a+b} = \bar{a}\bar{b}$
  $\overline{ab} = \bar{a}+\bar{b}$

Mappe di Karnaugh

↓

Per ottenere CIRCUITI MINIMIZZATI partendo dalla tavola della verità

Il metodo delle MdK può essere applicato sia a partire da una tavola della verità sia a partire da un funz. booleana e si basa sulla costruzione di una rappresentazione.

Esempio su 3 variabili d'ingresso

x_1	x_2	x_3	f_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Tav della verità

Però considerando che voglio il MINOR numero di parole possibili posso:

x_3 \ x_1x_2	00	01	11	10
0	1	0	0	0
1	1	1	1	0

Mappa di Karnaugh

↓

Permette di eseguire l'ottimizzazione, ovvero cercare di individuare i gruppi di 1 da aggregare insieme.

↓ NB

1	0	0	0
1	1	1	0

Ast è un modo per raggruppare gli 1 in gruppi di dim. potenza di 2 tenendo conto che gli 1 devono essere ADIACENTI

1	0	0	0
1	1	1	0

Ognuno di qst gruppi corrisponde ad un CUBO.

Mappe di Karnaugh con 4 ingressi

1)

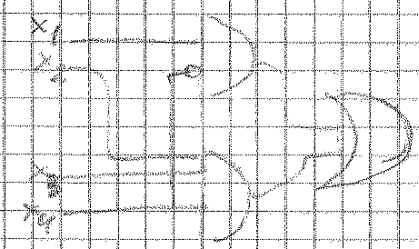
x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	1	0	0	1
10	0	0	1	1

queste righe sono ADIACENTI

$\overline{x}_2 x_3 x_4$

$x_1 \overline{x}_4$

$f = \overline{x}_2 x_3 x_4 + x_1 \overline{x}_4$



2) Da svolgere

x_3x_4 \ x_1x_2	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

$\overline{x}_3 x_2$

x_4

Deve lavorare il num MINIMO di subi di dim. MASSIMA

$f = x_4 + x_2 \overline{x}_3$

Lezione 22

Procedura di sintesi

Una volta costruita la Mappa di Karnaugh corrispondente alla tavola di verità si deve:

- identificare il minimo insieme di cubi che coprono tutti gli 1 nella mappa di Karnaugh (cercando quelli di dimensione maggiore); i cubi possono eventualmente sovrapporsi.
- trasformare i cubi nella corrispondente espressione in forma di somma di prodotti.
- derivare il corrispondente circuito.

Il problema della sintesi di un circuito combinatorio minimo è un problema con complessità esponenziale → maggiore è il numero di ingressi maggiore è la difficoltà nell'eseguire la sintesi.

Don't care

In alcuni casi le specifiche per un sistema combinatorio possono non prevedere alcun valore in uscita in corrispondenza di alcune combinazioni in ingresso, ad esempio perché qst non si presentano mai.

In tal caso si parla di valori DON'T CARE

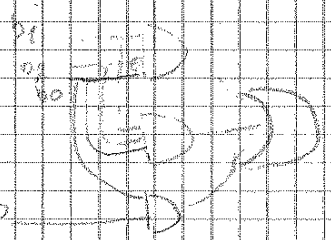
Perciò il processo di sintesi può sfruttare qst valori per minimizzare il circuito

Possiamo costruire una mappa di Karnaugh con 6 caselle contenenti il valore dont care.

b_1b_0 \ b_3b_2	00	01	11	10
00	0	0	X	0
01	0	0	X	1
11	1	0	X	X
10	0	1	X	X

Possiamo scegliere per ogni X il valore più appropriato per ottenere il circuito minimo.

b_1b_0 \ b_3b_2	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	0	1	1
10	0	1	1	0



$b_3 b_0$

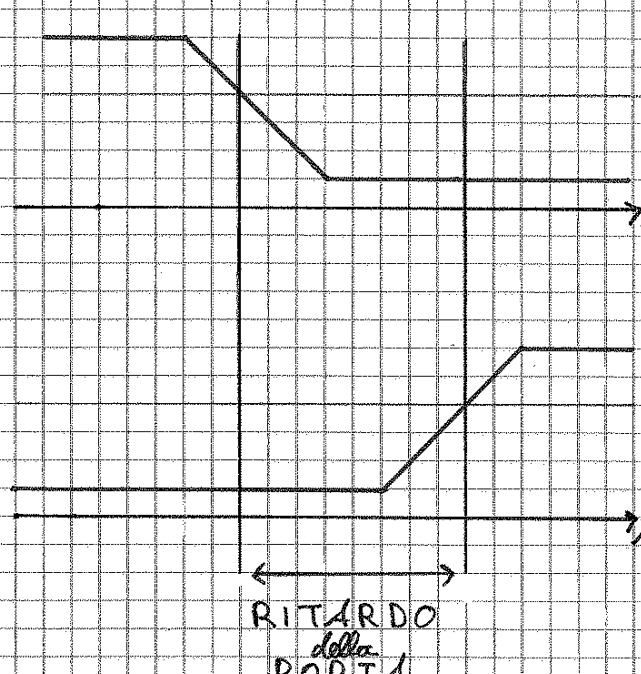
$\bar{b}_2 b_0 b_1$

$b_0 b_1 b_2$

$$f = b_3 b_0 + \bar{b}_2 b_0 b_1 + b_0 b_1 b_2$$

Ritardi

Tempo necessario affinché, date delle combinazioni in ingresso, si abbiano le corrispondenti combinazioni in uscita.



Ritardo K: è il tempo che intercorre dal momento in cui si verifica una Δ sugli ingressi ($t_0 + t$) al momento in cui l'uscita assume il valore corrispondente.

Il ritardo del circuito dipende anche dalla PROFONDITÀ DEL CIRCUITO ovvero il massimo numero di porte che si incontrano lungo un qualsiasi cammino da un'ingresso ad un'uscita.

Se le porte hanno lo stesso ritardo allora trovo il cammino che incontra più porte, ma se hanno ritardi diversi trovo il cammino le cui porte provocano un ritardo massimo (maggiore del ritardo delle porte su un cammino differente)

Cammino critico

↓

Ritardo dato dal tempo tra l'ingresso e l'uscita

Cammino in cui una transizione impiega il massimo tempo per andare da un ingresso a un'uscita.

Considerando l'esempio precedente: se su tutti gli ingressi assumono 1, l'uscita andrà 0 e se ad un certo t l'ultimo ingresso assume il valore 0 allora, essendo tutte porte AND, in uscita avrò un 0! Poiché in qst caso lo 0 in entrata sull'ultimo ingresso (in basso) incontra una sola porta lungo il suo cammino si avrà un ritardo K legato, appunto, alla porta.

(AND→)

Se invece fosse il 2° ingresso (partendo dal basso) ad assumere il valore 0, poiché su quel cammino vi sono 2 porte il ritardo sarebbe 2K. (ogni porta ha ritardo K)

Il CAMMINO CRITICO di qst circuito è quello corrispondente al 1° o 2° ingresso! → evidenziato nell'esempio

↓

RITARDO = 4K

I cammini critici mi permettono di calcolare il ritardo massimo con cui le uscite commutano.

In generale una porta è al livello i se ha almeno un ingresso al livello i-1.

Lezione 23

Circuiti sequenziali

Variabile di stato → tiene conto della storia precedente degli ingressi

↓

Struttura di memorizzazione che contiene un valore finché non ghene viene scritto uno nuovo sopra

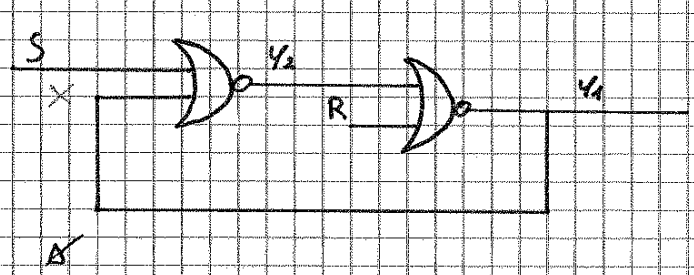
chiamate Variabili di memoria

↓

si basano sui ritardi delle porte e permettono di creare degli elementi in grado di memorizzare un bit

Flip - flop:

✗ Viola le regole dei cosidetti circuiti ben formati (CBF), infatti vi è un ciclo.



È in grado di mantenere l'informazione per un periodo illimitato di tempo.

2 porte NOR

NOR → assumono il valore 0 solo se sugli ingressi ci sono 2 1.

Ma $S=R=1$ configurazione vietata

Se $S=R=1$ y_2 è $=0$ e y_1 è $=0$, su $\bar{y}$ è riportato 0, quindi il ciclo permane

⬇

Transizione $S=R=1 \rightarrow S=R=0$

Nel passaggio dalla config. $S=R=1$ alla config. $S=R=0$, il circuito assume una configurazione che dipende da quale dei 2 NOR è più veloce a commutare.

Se $K_2 < K_1$ → la porta di sinistra è più veloce di quella di destra

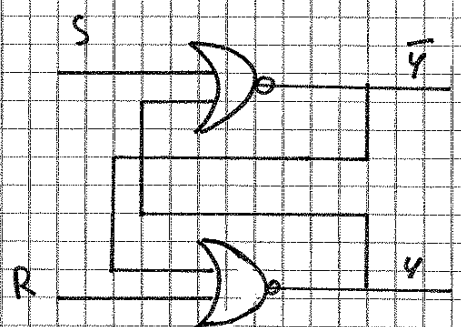
Se $K_2 > K_1$ → " " " destra " " " " " " sinistra

Nella pratica le porte non avranno MAI lo stesso ritardo. Ast circuito ha quindi un comportamento NON prevedibile.

⬇

ecco perché configurazione vietata

Flip-Flop SR asincrono:



y \ SR	00	01	10
0	0	0	1
1	1	0	1

Tavola degli stati

⬇

variabile di stato (= uscita)

TAVOLA DEGLI STATI:
- Se applico la config. 10 forzo a 1 l'uscita
- Se applico la config. 01 forzo a 0 l'uscita
- Se applico la config. 00 mantengo il valore dell'uscita.

La tabella non possiede la config. 11 perché è VIETATA.

Clock → mi avvisa quando il circuito può commutare!

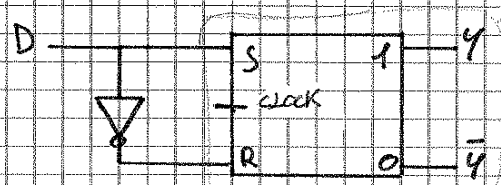
Clock = 0 → il circuito è "congelato" mantenendo il valore interno a cui l'avevo forzato precedentemente.

Clock = 1 → la config. del sistema dipende da S e R che possono forzare a 1 o a 0 il valore di y.
S = R = 1 → vietata

CIRCUITI SINCRONI circuiti con segnale di clock che permette di decidere se procedere con la commutazione o meno.
(inteso come abilitare)

Flip-Flop D → più comune nella pratica

È creato a partire da un flip-flop sincrono, con l'aggiunta di un invertec.



Tutto il circuito è pilotato da un ingresso D.
Per la presenza dell'INVERTER S e R avranno sempre valori opposti.

Perciò al sincrono posso applicare una configurazione che forza a 1 o a 0.
Da ricordare però la presenza del clock! clock = 0 . . .
clock = 1 . . .

Se il clock = 1 considero il valore di D.

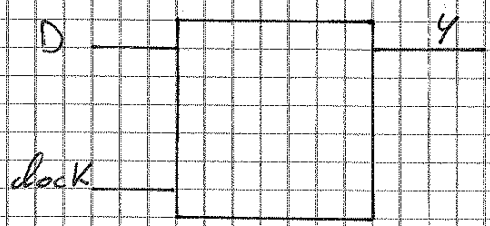
y \ D	0	1
0	0	1
1	0	1

} tavola degli stati

Se D = 0, quando il segnale di clock va a 1, l'uscita assume il valore di D, ma anche quando D = 1 l'uscita assume il valore di D

- D=1 → S=1, R=0 → y=1
  con 1e 0 il SR sincrono
  forza l'uscita a 1

Il flip-flop D è un elemento di MEMORIA.

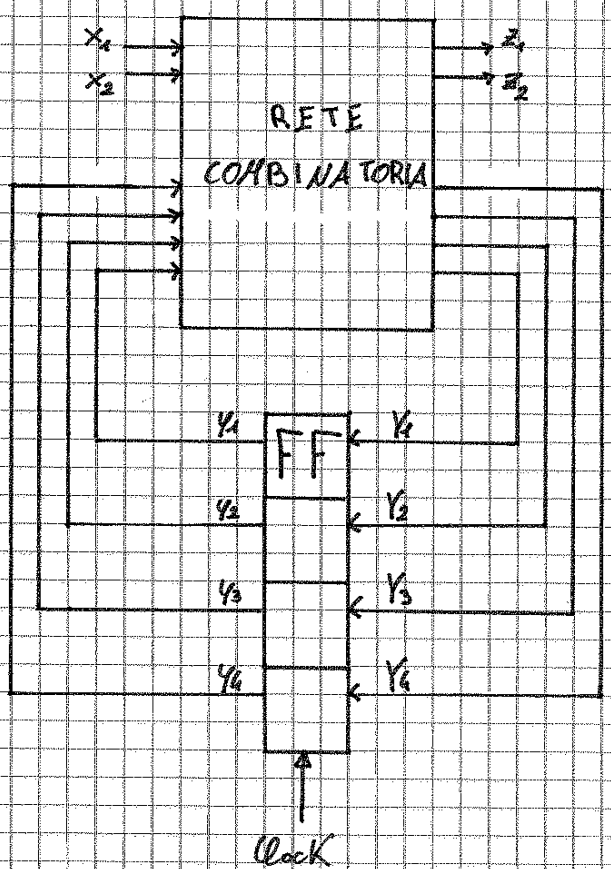


1 entrata → D
1 uscita → y

Circuito Sequenziale Sincrono

É un circuito composto da porte logiche e da flip-flop di tipo D. Detto SINCRONO perché è presente un segnale di clock che è comune a tutti i flip-flop componenti il circuito.

Ast circuito può essere schematizzato secondo il modello di Huffman.



→ porte logiche

FF → flip-flop connessi a uno stesso clock

stabili.

Ad un segnale di clock le uscite cambiano, è possibile siano cambiati anche i valori degli ingressi, e cambiano le USCITE! TUTTO CIO' SI VERIFICA AD OGNI COLPO DI CLOCK.

- valore I, valore flip-flop ⇒ si determinano O e Y ⇒ si determina (colpo di CLOCK)
- Y → ricomincia el ciclo
- → comportamento di un circuito sequenziale

↓

in cui l'istante di campionamento viene determinato dal clock che fa muovere il circuito da un certo stato presentato dai valori dei flip-flop in un nuovo stato.

Un circuito SEQUENZIALE SINCRONO secondo il modello di Huffman implementa un SISTEMA SEQUENZIALE descrivibile attr. il diagramma degli stati.

I flip-flop sono coloro che permettono di memorizzare la VARIABILE DI STATO.

Il valore delle USCITE dip dal valore della VARIABILE DI STATO e dagli INGRESSI.

L'assegnazione degli stati presuppone che io sappia quanti stati sono necessari, devo identificare innanzitutto il numero di flip-flop necessari per codificare il valore dello stato.

Se ho N stati il numero di flip-flop necessari è pari al $\lceil \log_2 N \rceil$ prendendo l'intero superiore.

Se ho tra 0 e 3 stati mi occorrono 2 FF.

//	4 e 7	//	3 //
	8 15		4
	16 31		5

Una volta scoperto il numero di FF posso assegnare, in prima battuta a caso, a ciascuno stato una rappresentazione binaria dei FF che ho identificato.

La tabella degli stati, una volta fatta la trasformazione dei nomi simbolici degli stati in valori binari, diventa la TAVOLA DI VERITÀ della rete combinatoria che troviamo nel modello di Huffman.

Con le mappe di Karnaugh, o con qualche altro metodo più sofisticato se la tabella è troppo grande, ci costruiamo per

NB ciascuna uscita della rete combinatoria il circuito che la implementa. -> quindi eseguiamo la SINTESI o PROGETTO della rete combinatoria e otteniamo il nostro modello di Huffman implementato sotto forma di FF da un lato e porte logiche dall'altro e abbiamo costruito il CIRCUITO SEQUENZIALE SINCRONO che si comporta come il sistema sequenziale delle specifiche iniziali.

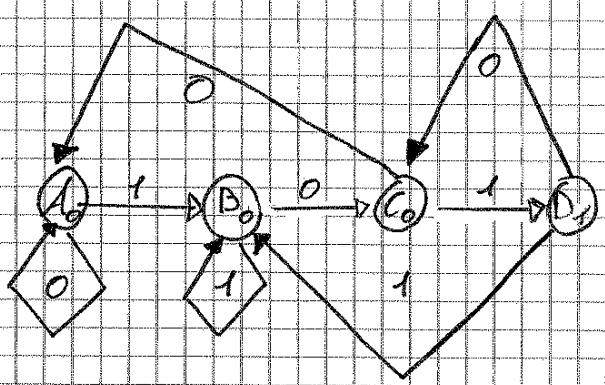
STATI S
INGRESSI I

$\#\ LINEE = S * 2^I$

Però il diagramma di stato può essere semplificato tramite l'eliminazione di stati equivalenti.

2 stati sono EQUIVALENTI se a fronte della stessa sequenza di ingressi producono la stessa sequenza di uscite.

È possibile MINIMIZZARE il diagramma degli stati ottenendo un diagramma di minor dimensioni che però abbia lo stesso comportamento del precedente.



4 stati
2 FF

→ assegno un valore agli stati e sostituisco il valore allo stato nella tabella di transizione.

Esempio tabella di transizione.

I	Y		Y	O
f(0,00)		→	00	0
f(1,00)		→	01	0
f(0,01)		→	10	0
f(1,01)		→	01	0
f(0,10)		→	00	0
f(1,10)		→	11	0
f(0,11)		→	10	1
f(1,11)		→	01	1

A = 00
B = 01
C = 10
D = 11

Costruisco la rete combinatoria la quale ha 3 ingressi: 1 I e 2 Y dati dai 2 FF; e 3 uscite: 1 O e 2 Y per alimentare i FF

- Mappa di Karnaugh di Y

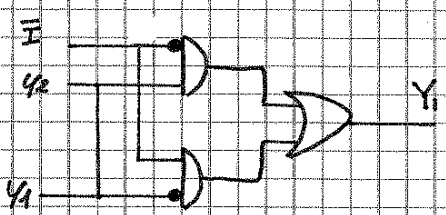
-1ª COLONNA

I \ y_1y_2	00	01	11	10
0	0	1	1	0
1	0	0	0	1

Y
00
01
10
01
00
11
10
01

ho esaminato la 1ª COLONNA → 1° FF

$Y_1 = \bar{I} \cdot y_2 + I\, y_1 \bar{y}_2$



→ circuito che produce Y_1

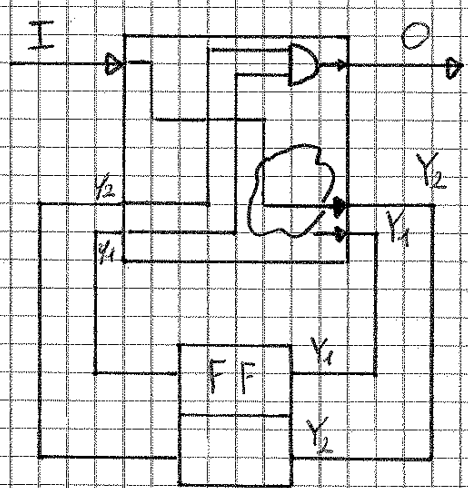
-2ª COLONNA

Y_2 implementa y_2

I \ y_1y_2	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$Y_2 = I$

-Y_2 dipende solo da I



circuito SEQUENZIALE SINCRONO che implementa il RICONOSCITORE di SEQUENZA

4) FF D scelgo la risposta vera.

risposta corretta: Ad ogni colpo di clock viene memorizzato il valore presente sull'ingresso D.

5) Si consideri un sistema sequenziale con 3 ingressi, 10 stati e 2 uscite, di quante righe è composta la tabella di verità della funzione di transizione degli stati?

risposta corretta: 80

$\#\,INGRESSI = STATI * 2^{I}$

$10 * 2^{3} = 80$

6) Si consideri il modello di Huffman dei circuiti sequenziali sincroni: a cosa corrisponde il blocco posto sotto la rete combinatoria?

risposta corretta: Ad un insieme di FF di tipo D

7) Considerazioni sulla 7ª domanda!

FSM -> macchine a stati finiti tipicamente implementate da un circuito sequenziale sincrono.

FSM che implementano il modello di Mealy
e il modello di Moore
↓
le uscite dip. solo dal valore dello stato in quel momento
↓
nel diagramma degli stati il valore di uscita è associato al vertice.

↓
le uscite dip dal valore dello stato e degli ingressi in quel momento.

La falsa è FSM di Mealy!
O dip solo dal VALORE DELLO STATO.

2) Si progetti un circuito sequenziale che gestisca un distributore (minimo) di bevande.

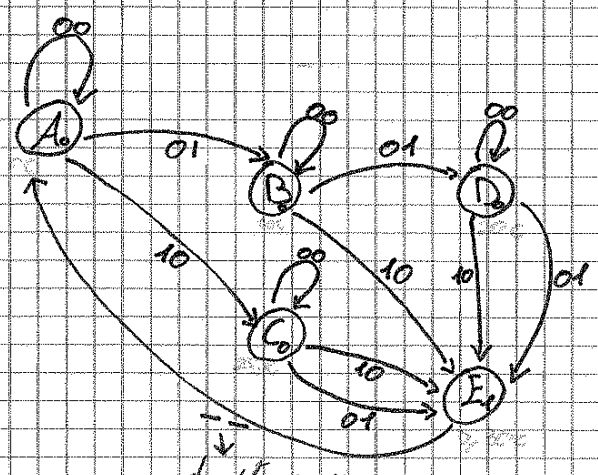
Il sistema possiede

- 2 ingressi x_1 e x_2 che segnalano l'introduzione di una moneta da 25 a 10 centesimi, rispettivamente; ad ogni istante uno solo dei 2 ingressi può essere attivo.
- un'uscita z che segnala se le monete introdotte hanno valore superiore a 30 centesimi, e se quindi la bevanda dev'essere fornita all'utente.
- il sistema non fornisce resto.

$x_1 \rightarrow 25$
$x_2 \rightarrow 10$

circuito sequenziale sincrono

Risolvibile o con un diagramma degli stati o con una tabella di transizione.



per ogni stato posso avere 3 archi perché vi sono 3 possibili combinazioni: 00, 01, 10

i 2 ingressi non si attivano contemporaneamente (XX)

Tabella di transizione

5 stati → 4 stato 3 possibili ingressi → 15 righe

I x1x0	Y	z	Y
00	A 000	0	A 000
01	A 000	0	B 001
10	A 000	0	C 010
00	B 001	0	B 001
01	B 001	0	D 011
10	B 001	0	E 100
00	C 010	0	C 010
01	C 010	0	E 100
10	C 010	0	E 100
00	D 011	0	D 011
01	D 011	0	E 100
10	D 011	0	E 100
00	E 100	1	A 000
01	E 100	1	A 000
10	E 100	1	A 000

NB → Z è riferito a Y e non a Y, quindi allo stato CORRENTE e non quello futuro.

5 STATI → k=5 → 3 FF necessari

Assegnazione degli stati

A	000
B	001
C	010
D	011
E	100

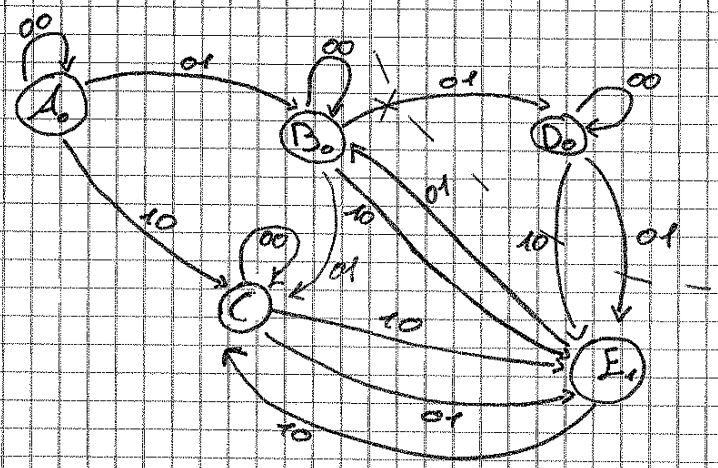
Ho ottenuto una funz. BOOLEANA, una tabella che corrisponde a un circuito combinatorio (del modello di Huffman) che per ciascuna riga ho le uscite e i valori dei FF.

funz. booleana con 5 ingressi e 4 uscite, utilizzando le uscite costruisco la Kappa di Karnaugh.

NB Moore non è più dispendioso in numero di stati, DIPENDE!

Entrambi i diagrammi non sono minimizzati.

Per esempio minimizziamo quello di Moore!



L'importo in più non viene cancellato

C e D hanno un comportamento molto simile!

$\xrightarrow{01}$ E

$\xrightarrow{10}$ E

C0 D0 hanno la stessa uscita!

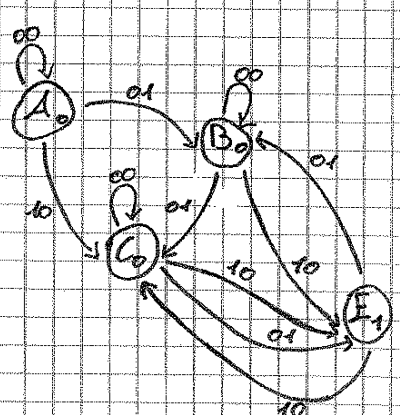
C/D $\xrightarrow{00}$ C/D

A fronte della stessa uscita se sottoposti alla stessa transizione si comportano nello STESSO modo → EQUIVALENTI

EQUIVALENTI → se compatto C e D il sistema non cambia

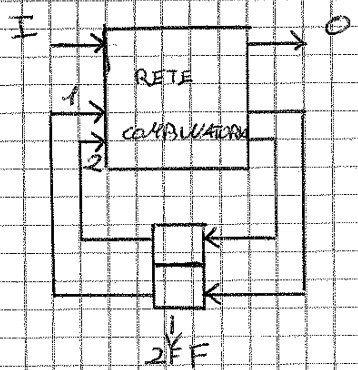
↓

associati alla stessa uscita e a fronte delle stesse combinazioni di ingresso vanno nello stesso stato.



L'importo in più non viene cancellato.

2 FF → 2 variabili per lo stato futuro



- Mappa di Karnaugh del flip-flop Y_2

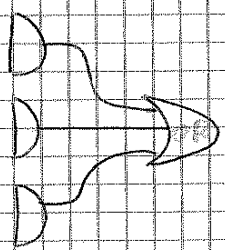
$y_2 y_1$ \ $x_1 x_2$	00	01	11	10
00	0	0	d	1
01	0	1	d	0
11	1	0	d	0
10	1	0	d	0

i don't care posso usarli come 0 o 1 a piacimento

↳ qst 3 d sono = 0

$$Y_2 = \bar{x}_1 \bar{x}_2 y_2 + \bar{y}_2 y_1 x_2 + \bar{y}_2 \bar{y}_1 x_1$$

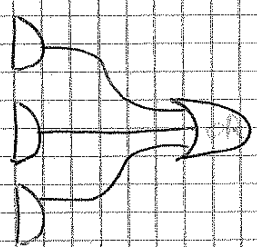
3 AND



- Mappa di Karnaugh per Y_1

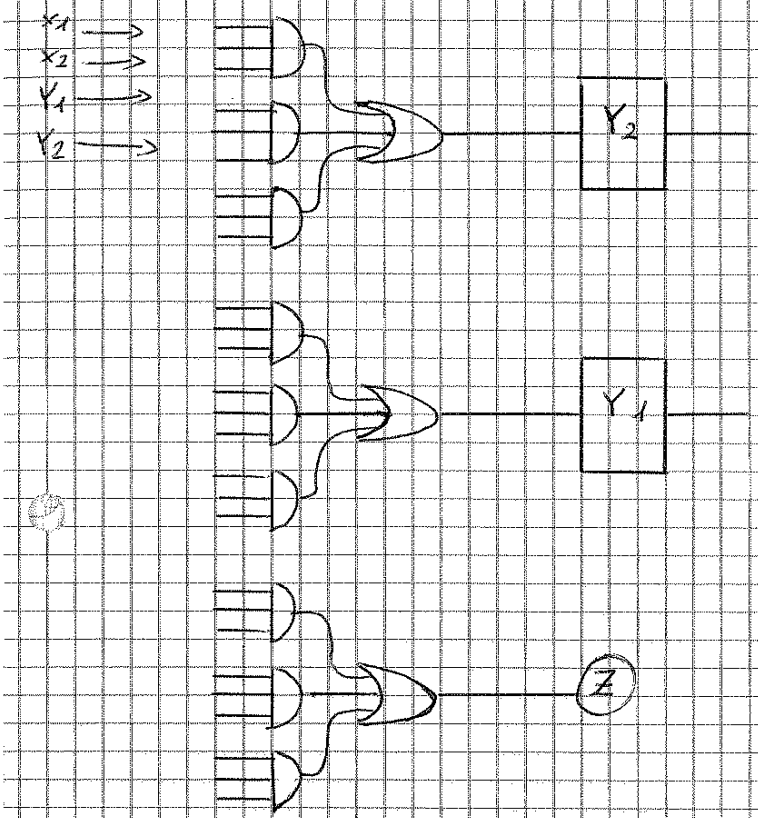
$y_2 y_1$ \ $x_1 x_2$	00	01	11	10
00	0	1	d	1
01	1	0	d	0
11	1	0	d	0
10	0	0	d	0

$$Y_1 = \bar{x}_1 \bar{x}_2 y_1 + \bar{y}_2 \bar{y}_1 x_2 + \bar{y}_2 \bar{y}_1 x_1$$



Ho 3 funzioni combinatorie a 1 uscita, ciascuna con 3 variabili in ingresso

Il circuito finale è:



Fine parte esercizi

Livello register transfer RT

Il progettista lavora con componenti più complessi delle porte logiche, anche le specifiche saranno tali.

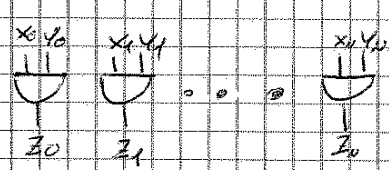
↓ es registri

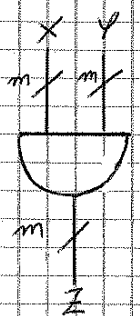
L'unità di dato è la parola, l'unità di tempo è maggiore.
I componenti principali sono formati da porte logiche.

Elenco di componenti è:
- porte logiche operanti su parole — COMBINATORIE
- multiplexer — C
- decodificatori e codificatori — C
- PLD
- moduli aritmetici (ALU, sommatore, ecc) — C
- registri
- contatori
- FPGA
- bus
- memorie

Porte logiche operanti su parole

Esegue un'operazione logica (AND, OR, ecc) bit a bit fra 2 parole.

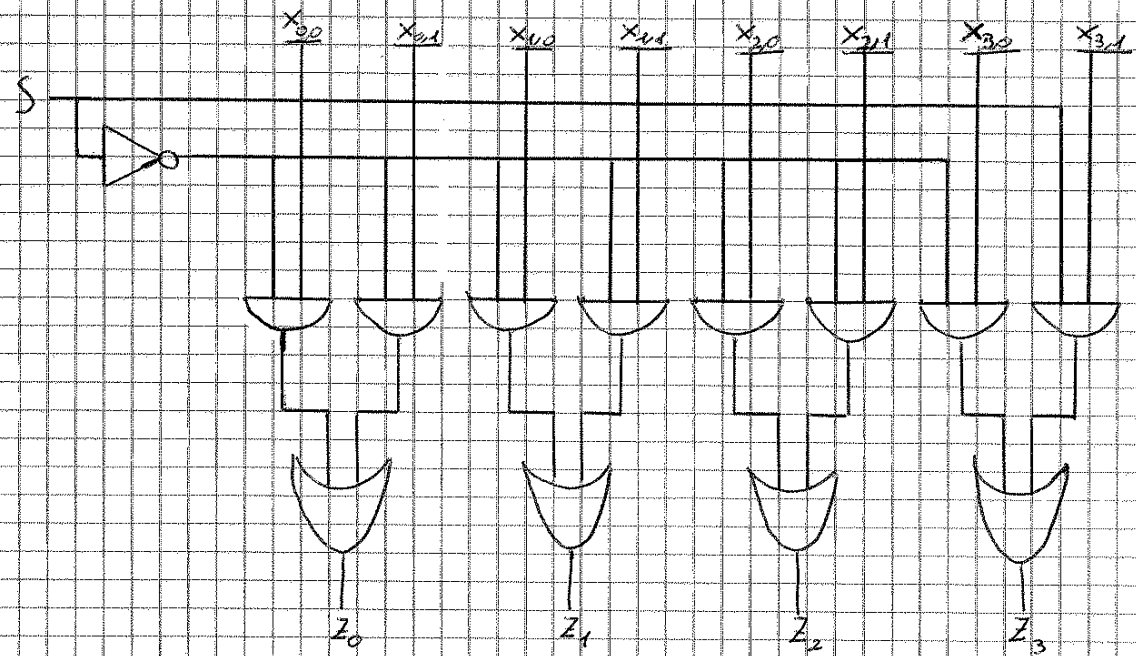




Multiplexer

È un modulo con K segnali in ingresso e 1 uscita, esso seleziona un segnale in ingresso e lo fa comparire in uscita, sono i segnali di controllo S ad indicargli quale ingresso selezionare. →

La realizzazione del multiplexer → esempio

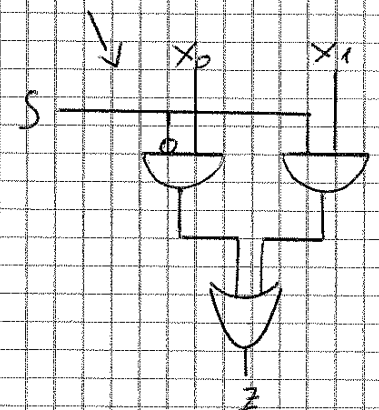


Multiplexer da 2 a 1, quindi 2 ingressi e 1 uscita, e parallelism pari a 4!

1° ingresso → $x_{0,0}$, $x_{1,0}$, $x_{2,0}$, $x_{3,0}$
2° ingresso → $x_{0,1}$, $x_{1,1}$, $x_{2,1}$, $x_{3,1}$
} ciascuno da 4 bit

A seconda del valore di S ciascuna AND fa passare uno o l'altro valore e poi l'OR porta all'uscita.

Multiplexer minimo



Il generico multiplexer mi permette di implementare una qualunque funzione combinatoria.

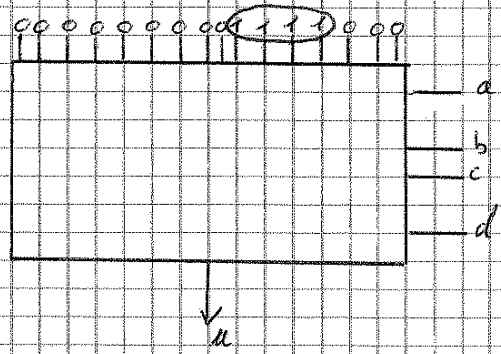
Esempio di sintesi

circuito con 4 ingressi a, b, c, d la cui u valga 1 quando $8 < (abcd)_2$

0000	0
0001	0
0010	0
.	
.	
.	
1000	0
1001	1
1010	1
1011	1
1100	1
1101	0
1110	0

SINTESI → MPX

↓ la v. della Verità



Non è minimo

Prendo un MPX di queste dimensioni, tanti segnali di controllo quanti sono i miei ingressi, poi collego gli ingressi a 0 e 1, a seconda di quale deve essere l'uscita in corrispondenza di una combinazione d'ingresso dei segnali.

Ad ogni ingresso di dato corrisponde un mintermine, dove è espresso in funzione di un segnale di controllo, ognuno dei quali è un letterale.

Un decodificatore è un modulo combinatorio, i valori di uscita dipendono dagli ingressi in quel momento.

Possiamo utilizzare una TAVOLA DI VERITÀ per spiegarne il comportamento.

Esempio

decodificatore 2 → 4

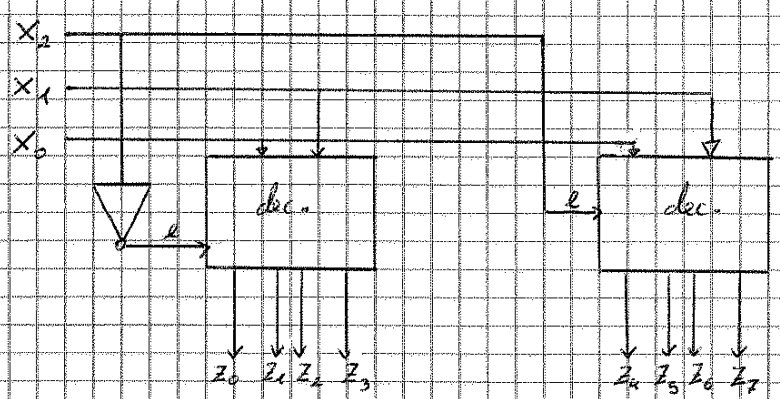
X	e	$Z_{3\,2\,1\,0}$
00	1	0001
01	1	0010
10	1	0010
11	1	1000
-	0	0000

↓ disattivo

→ 01 = 1 → attivo la porta con indice 1

Connessione a cascata

1) decodificatore 3 → 8



X_2 collegato agli enable, decide quale dei 2 decodificatori deve attivarsi.

Quindi escluso X_2, che comanda gli ENABLE, le uscite dipendono dai 2 bit meno significativi della tabella a fianco

X_2	X_1	X_0	
0	0	0	$X_2 = 0$ si attiva il dec di sinistra
0	0	1	
0	1	0	
0	1	1	
1	0	0	$X_2 = 1$ si attiva il dec di destra
1	0	1	
1	1	0	
1	1	1	

È un modulo combinatorio, posso utilizzare la TAVOLA DELLA VERITÀ per spiegarne il comportamento.

Esempio

codificatore 4-2

X_3 X_2 X_1 X_0	Z
0001	00
0010	01
0100	10
1000	11

→ indice 2 → Z = 2

Se più ingressi vengono attivati contemporaneamente bisogna parlare di codificatori prioritari.

Nei cod. prioritari qualunque configurazione d'ingresso è lecita, non ci sono limitazioni.
Se vi è una sola linea d'ingresso attiva il codificatore prioritario si comporta come un codificatore normale.
Se vi sono più linee d'ingresso attive cmq lui può mandarne solo 1 in uscita, scegliere quale si basa sul concetto di PRIORITÀ.

All'interno del modulo ogni linea avrà una certa priorità, maggiore per alcune minore per altre, il cod. prioritario manda in uscita l'indice della linea d'ingresso con priorità massima tra quelle attive.

Moduli Aritmetici

SOMMATORE
SOTTRATTORE
ecc,

I Mod. Ar. possono avere complessità variabile a seconda di:

- tipo di dati supportati (interi, interi con segno, decimali)
- tipo di operazioni supportate (somma, sottraz., moltiplicaz., divisione, oper. trigonometriche)
- velocità (soluzioni combinatorie o sequenziali)

- Sommatori

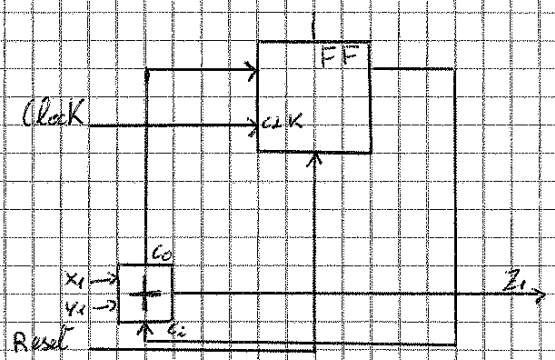
Possono essere realizzati seguendo 3 soluzioni alternative:

1. sommatori seriali
2. sommatori combinatori
3. sommatori combinatori modulari

NB i sommatori qui considerati sommano numeri INTERI e SENZA SEGNO.

1. Sommatore Seriale

Il costo in termini di hardware è minimo poiché ha una struttura molto semplice.



Ad ogni colpo di clock viene prodotto un bit in uscita a partire dal meno significativo. All'inizio il carry deve essere nullo.

Full Adder (+) è l'elemento base dei sommatori, prende 2 bit in ingresso e produce in uscita i bit d'uscita e il carry!

x	y	z	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Il sommatore combinatorio funziona bene con n piccolo, la complessità cresce esponenzialmente con n.

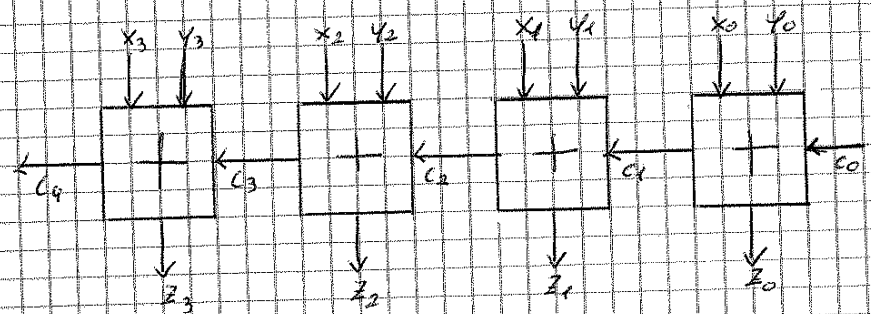
Vi sono altre soluzioni caratterizzate dalla MODULARITÀ, soluzione tale per cui il progettista abbia una soluzione con un certo parallelismo e avendo certi componenti possa costruire qualunque sommatore.

Le soluzioni modulari cercano di comporre dei MODULI ELEMENTARI per costruire il generico modulo.

Ripple Carry Adder

↓ si basa su più full-adder

Se dovessi per esempio sommare 2 numeri ognuno su 4 bit, potrei utilizzare 4 full-adder uno affianco all'altro.



Le c riportano il carry!

Questo metodo è estensibile all'infinito aggiungendo sempre full-adder.

Ogni Full-Adder mi riporta in uscita un bit del risultato, fino ad avere il valore totale dato dall'insieme dei full-adder.

Vantaggio del RIPPLE CARRY ADDER: facilità di costruzione.

↓ è un circuito combinatorio

↓ ha un ritardo

Si può costruire un'espressione che ci indica come il carry look. può costruire il valore del generico carry da mandare al generico full-adder.

Carry con ritardo → $c_{i+1} = x_i y_i + x_i c_{i-1} + y_i c_{i-1} = g_i + p_i c_{i-1}$ ← Prodotto senza ritardo

dove $g = x_i y_i$, $p = x_i + y_i$

Analogamente $c_{i-1} = g_{i-1} + p_{i-1} c_{i-2}$

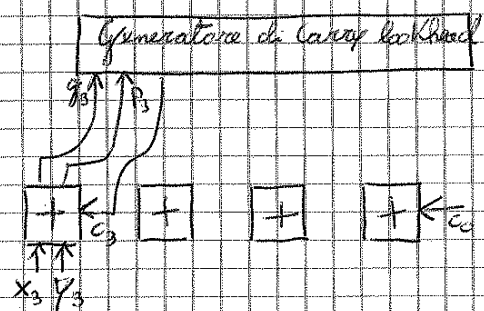
Sostituendo c_{i-1} in c_{i+1} ottengo

$$c_i = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-2}$$

Posso poi sostituire c_{i-2} e così via.

Il carry lookhead genera qst espressioni.

→ IMPLEMENTAZIONE



→ implementa le espressioni e genera i vari carry

Qst sommatore funziona su 2 fasi: 1ª fase → applico x e y ai vari full-adder i quali generano g e p, il ritardo tra l'inserimento dei valori sull'ingresso e la formazione di g e p è pari a Δ ovvero il ritardo del full-adder.

↓

lavorano in parallelo

Il generatore è veloce! Passato Δ vengono introdotti i valori dei carry nei full-adder, ora tutti i f.-ad hanno stabili i valori di x, y e c e producono z.

t in cui x e y sono STABILI

Prima di ottenere le uscite, partendo da un tempo t_0, vi sarà il ritardo (Δ) dei f.-ad per produrre g e p, il ritardo del generatore pari a Δ e nuovamente il (Δ) ritardo dei f-add che a qst punto hanno ricevuto i carry e producono le uscite Z.