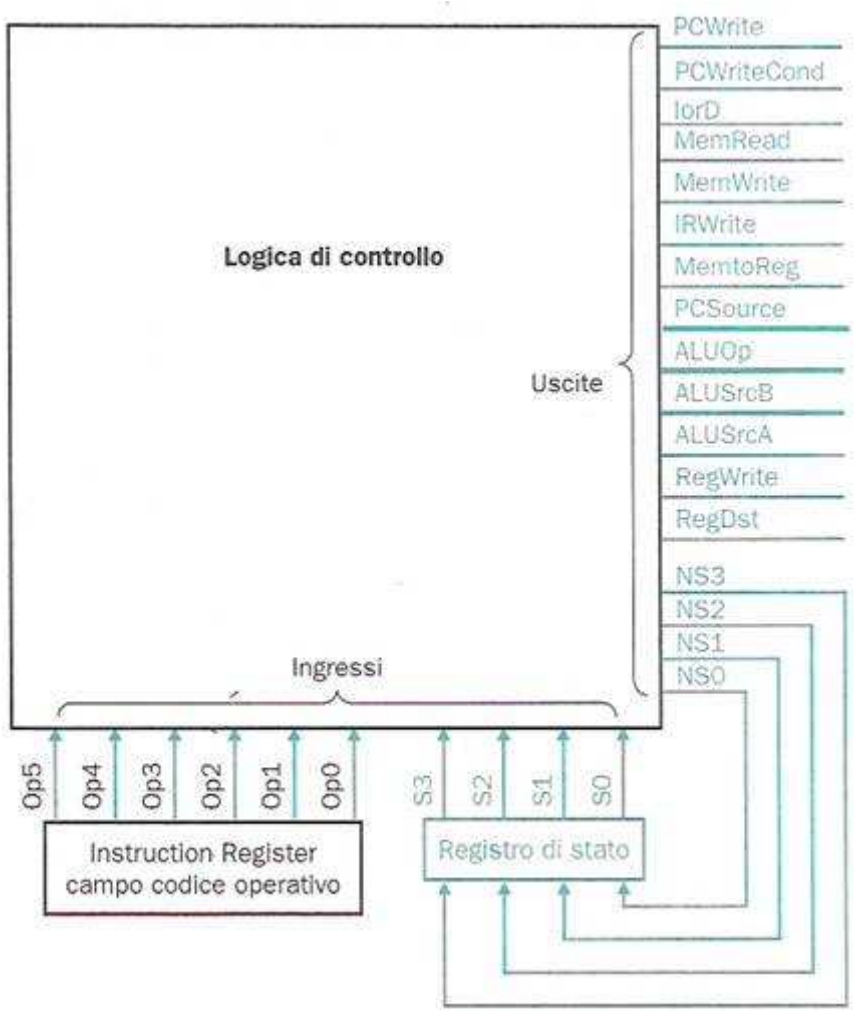
SCO MIPS Multiciclo



- Logica Combinatoria di Controllo
- Registro di Stato Corrente
- Uscite:
  - Segnali di Controllo (SC)
  - Stato Futuro (SF)
  - **Modello Moore**:
    - SC dipende solo da stato corrente
    - SF dipende anche da opcode
    - Pro:
      - logica controllo + semplice
      - segnali di controllo calcolati + velocemente
  - Modello Mealy:
    - SC e SF dipendono sia da stato corrente che da opcode
    - Pro:
      - possibile riduzione numero di stati della macchina a stati

Step I: equazioni logiche per ogni singola uscita

Uscita	Stato corrente	Operazione
PCWrite	stato0 + stato9	
PCWriteCond	stato8	
IorD	stato3 + stato5	
MemRead	stato0 + stato3	
MemWrite	stato5	
IRWrite	stato0	
MemtoReg	stato4	
PCSource1	stato9	
PCSource0	stato8	
ALUOp1	stato6	
ALUOp0	stato8	
ALUSrcB1	stato1 + stato2	
ALUSrcB0	stato0 + stato1	
ALUSrcA	stato2 + stato6 + stato8	
RegWrite	stato4 + stato7	
RegDst	stato7	
StatoFuturo0	stato4 + stato5 + stato7 + stato8+ stato9	
StatoFuturo1	stato0	
StatoFuturo2	stato1	(Op='lw')+(Op='sw')
StatoFuturo3	stato2	(Op='lw')
StatoFuturo4	stato3	
StatoFuturo5	stato2	(Op='sw')
StatoFuturo6	stato1	(Op='tipo-R')
StatoFuturo7	stato6	
StatoFuturo8	stato1	(Op='beq')
StatoFuturo9	stato1	(Op='jmp')

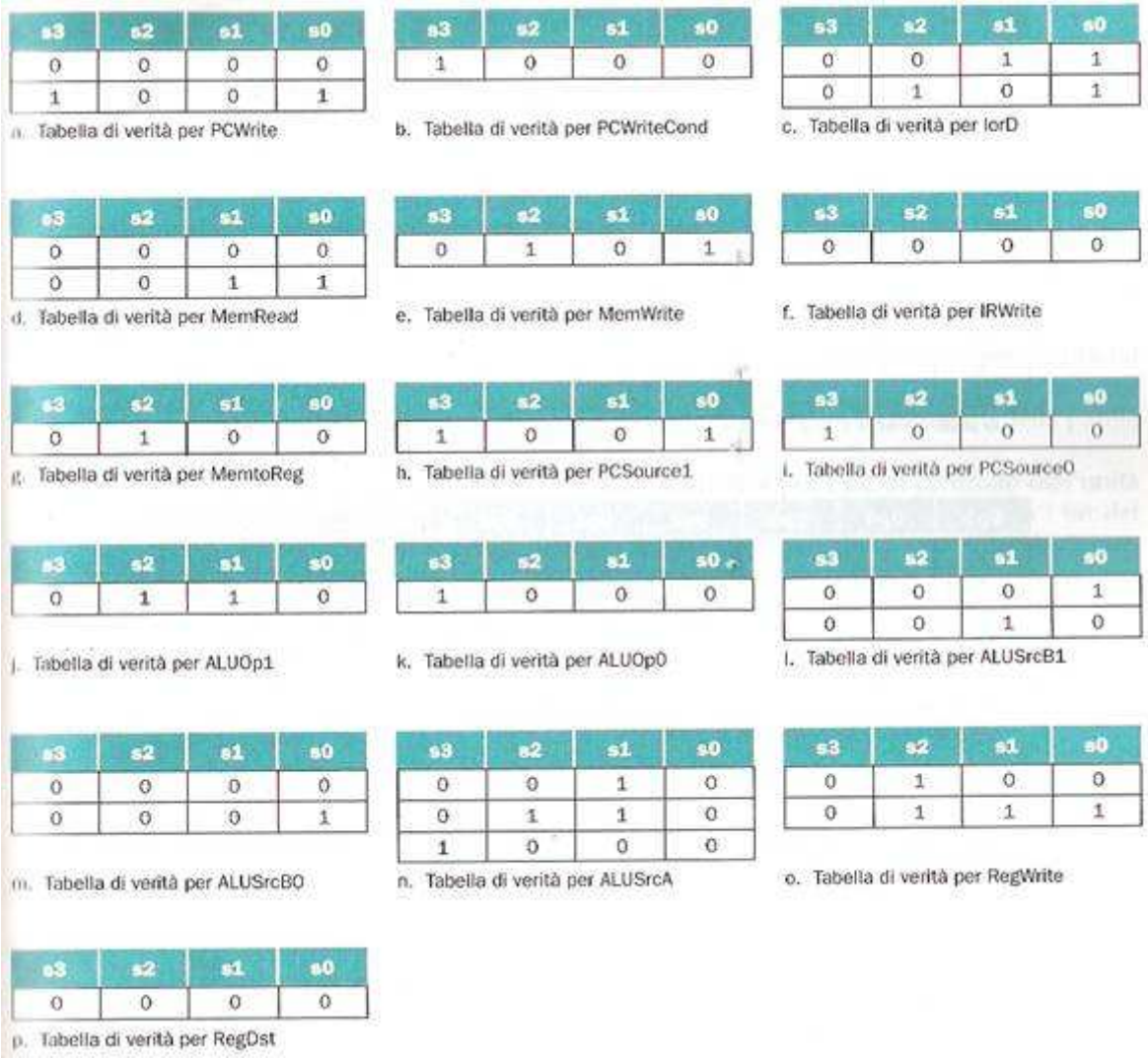
Si hanno complessivamente 20 uscite:

- 16 segnali di controllo
- 4 bits per la codifica di 10 stati:
  - ancora da codificare…

Per ciascuna uscita si identificano le condizioni logiche abilitanti:

- Modello di Moore:
  - SC dipendono solo da stato corrente
  - Conviene separare logica di controllo per SC:
    - indipendente da opcode
    - piu' semplice
  - e logica di controllo per SF:
    - maggiore complessità (+ ingressi)

Step II: codifica segnali di controllo (1)



a. Tabella di verità per PCWrite

s3	s2	s1	s0
0	0	0	0
1	0	0	1

b. Tabella di verità per PCWriteCond

s3	s2	s1	s0
1	0	0	0

c. Tabella di verità per IorD

s3	s2	s1	s0
0	0	1	1
0	1	0	1

d. Tabella di verità per MemRead

s3	s2	s1	s0
0	0	0	0
0	0	1	1

e. Tabella di verità per MemWrite

s3	s2	s1	s0
0	1	0	1

f. Tabella di verità per IRWrite

s3	s2	s1	s0
0	0	0	0

g. Tabella di verità per MemtoReg

s3	s2	s1	s0
0	1	0	0

h. Tabella di verità per PCSource1

s3	s2	s1	s0
1	0	0	1

i. Tabella di verità per PCSource0

s3	s2	s1	s0
1	0	0	0

j. Tabella di verità per ALUOp1

s3	s2	s1	s0
0	1	1	0

k. Tabella di verità per ALUOp0

s3	s2	s1	s0
1	0	0	0

l. Tabella di verità per ALUSrcB1

s3	s2	s1	s0
0	0	0	1
0	0	1	0

m. Tabella di verità per ALUSrcB0

s3	s2	s1	s0
0	0	0	0
0	0	0	1

n. Tabella di verità per ALUSrcA

s3	s2	s1	s0
0	0	1	0
0	1	1	0
1	0	0	0

o. Tabella di verità per RegWrite

s3	s2	s1	s0
0	1	0	0
0	1	1	1

p. Tabella di verità per RegDst

s3	s2	s1	s0
0	0	0	0

- Innanzitutto occorre codificare i 10 possibili stati:
  - sono sufficienti i 4 bits S3,S2,S1,S0
- Per ciascun SC, si costruisce una tabella in cui si riportano gli stati in cui sono abilitati:
  - I segnali di controllo dipendono solo dallo stato corrente (Moore):
    - Gli opcode possono essere visti come don’t care conditions

Step II: codifica segnali di controllo (2)

$S_3S_2S_1S_0$	Segnali di Controllo
0000	1001010000001000
0001	0000000000011000
0010	0000000000010100
0011	0011000000000000
0100	0000001000000010
0101	0010100000000000
0110	0000000001000100
0111	0000000000000011
1000	0100000010100100
1001	1000000100000000

- Si può ottenere una rappresentazione equivalente, ma più compatta codificando i segnali di controllo nell'ordine in cui sono mostrati a pagina 1
  - PCWrite, PCWriteCond,IorD …
- …e associare a ciascuno stato l'intera configurazione dei segnali di controllo abilitati

Step III: codifica stato futuro

StatoFuturo0	stato4 + stato5 + stato7 + stato8+ stato9	
StatoFuturo1	stato0	
StatoFuturo2	stato1	(Op='lw')+(Op='sw')
StatoFuturo3	stato2	(Op='lw')
StatoFuturo4	stato3	
StatoFuturo5	stato2	(Op='sw')
StatoFuturo6	stato1	(Op='tipo-R')
StatoFuturo7	stato6	
StatoFuturo8	stato1	(Op='beq')
StatoFuturo9	stato1	(Op='jmp')

Ingressi Uscite

opcode						stato corrente				stato futuro			
x	x	x	x	x	x	0	0	0	0	0	0	0	1
l	o	a	d	w		0	0	0	1	0	0	1	0
s	t	o	r	e	w	0	0	0	1	0	0	1	0
b	e	q				0	0	0	1	1	0	0	0
j	m	p				0	0	0	1	1	0	0	1
..	..	..	..	..	..	..	..	..	..	..	..	..	..

Per ogni bit di codifica dello stato futuro (SF3,SF2,SF1,SF0) si costruisce una tabella di verità, ad es:

- il bit SF3 serve per codificare solo gli stati futuri 8 e 9
- SF 8 iff:
  - stato corrente=1 AND opcode=beq (00001)
- SF 9 iff:
  - stato corrente=1 AND opcode=jmp (00010)

Step III: codifica stato futuro

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	0	1

a. Tabella di verità per l'uscita SF3, che è attiva quando lo stato futuro è 8 o 9. Il segnale viene attivato quando lo stato corrente è 1.

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	1
1	0	1	0	1	1	0	0	1	0
X	X	X	X	X	X	0	0	1	1
X	X	X	X	X	X	0	1	1	0

b. Tabella di verità per l'uscita SF2, che è attiva quando lo stato futuro è 4, 5, 6 o 7. Questo succede solo se lo stato corrente è 1, 2, 3 o 6.

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	0	1
1	0	1	0	1	1	0	0	0	1
1	0	0	0	1	1	0	0	1	0
X	X	X	X	X	X	0	1	1	0

c. Tabella di verità per l'uscita SF1, che è attiva quando lo stato futuro è 2, 3, 6 o 7. Lo stato futuro è 2, 3, 6 o 7 solo se lo stato corrente è 1, 2 o 6.

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
X	X	X	X	X	X	0	0	0	0
1	0	0	0	1	1	0	0	1	0
1	0	1	0	1	1	0	0	1	0
X	X	X	X	X	X	0	1	1	0
0	0	0	0	1	0	0	0	0	1

d. Tabella di verità per l'uscita SF0, che è attiva quando lo stato futuro è 1, 3, 5, 7 o 9. Questo succede solo se lo stato corrente è 0, 1, 2 o 6.

- Per ogni bit di codifica dello stato futuro (SF3,SF2,SF1,SF0) si costruisce una tabella di verità, ad es:
  - il bit SF3 serve per codificare solo gli stati futuri 8 e 9
  - SF 8 iff:
    - stato corrente=1 AND opcode=beq (00001)
  - SF 9 iff:
    - stato corrente=1 AND opcode=jmp (00010)

Alternative x l'implementazione: ROM vs PLA

ROM (Read Only Memory)

- ciascuna configurazione dei 10 bit di ingresso (6 opcode + 4 bits di stato) è utilizzata per indirizzare una delle 2^10 celle di memoria
- ciascuna cella ha ampiezza 20 bit, tante quanto sono le uscite:
  - 16 segnali di controllo
  - 4 bits di stato
- capacità complessiva = 2^10 * 20 bits=20 Kbit
- Pro:
  - semplicità
- Contro:
  - forte ridondanza, specialmente in presenza di funzioni logiche sparse:
    - don't care conditions (d.c.c) non sono sfruttate:
      - » una riga con *n* d.c.c viene copiata 2^n volte
    - la maggior parte delle uscite (SC) dipendono solo da 4 ingressi (stato corrente)
  - per ridurre ridondanza è possibile considerare una ROM per SC e una per SF:
    - ROM SC: ingresso=4 bits, uscite=16 bits, capacità = 2^4*16=256 bits
    - ROM SF: ingresso=10 bits, uscite=4 bits, capacità = 2^10*4=4096 bits
    - Capacità ROM SF + ROM SC = 4,3 Kbits, un quarto rispetto alla RAM unica

Implementazione tramite ROM: Segnali di Controllo

4 bit meno significativi dell'indirizzo	Bit 19-4 della parola
0000	1001010000001000
0001	0000000000011000
0010	0000000000010100
0011	0011000000000000
0100	0000001000000010
0101	0010100000000000
0110	0000000001000100
0111	0000000000000011
1000	0100000010100100
1001	1000000100000000

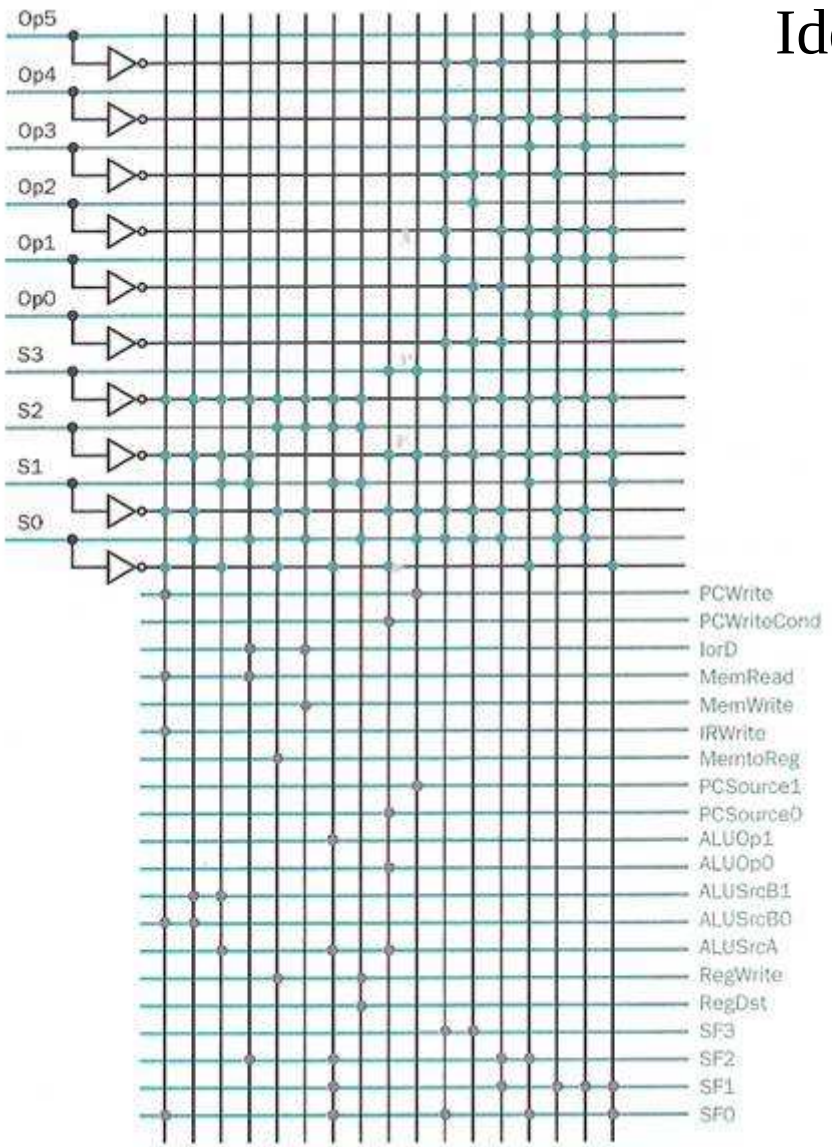
- Porzione ROM dedicata alla codifica dei SC:
  - solo stato corrente (codificato nei 4 bits meno significativi degli indirizzi della ROM) influenzano i SC (codificati nei 16 bit più significativi di ciascuna parola)
- Se non si utilizza ROM dedicate per SC e SF, queste informazioni saranno ripetute 64 volte (6 bits rimanenti indirizzo=2^6=64)

Implementazione tramite ROM: Stato Futuro

	Op [5-0]					
Stato corrente	000000	000010	000100	100011	101011	Ogni altro valore
0000	0001	0001	0001	0001	0001	0001
0001	0110	1001	1000	0010	0010	illegale
0010	XXXX	XXXX	XXXX	0011	0101	illegale
0011	0100	0100	0100	0100	0100	illegale
0100	0000	0000	0000	0000	0000	illegale
0101	0000	0000	0000	0000	0000	illegale
0110	0111	0111	0111	0111	0111	illegale
0111	0000	0000	0000	0000	0000	illegale
1000	0000	0000	0000	0000	0000	illegale
1001	0000	0000	0000	0000	0000	illegale

- d.c.c. associate a opcodes legali (appartenenti al set), ma non possibili visto lo stato attuale (es. stato 2)
- dopo fetch, si verifica se l'opcode prelevato dall'IR appartenga all'insieme degli opcodes legali:
  - altrimenti è necessario generare un'eccezione (x esercizio)
- Inefficienze solo parzialmente risolte usando ROM dedicata per SF:
  - Allo stato 0 sono associate 2^6 parole (tutti i possibili opcodes) anche se SF è sempre 1…

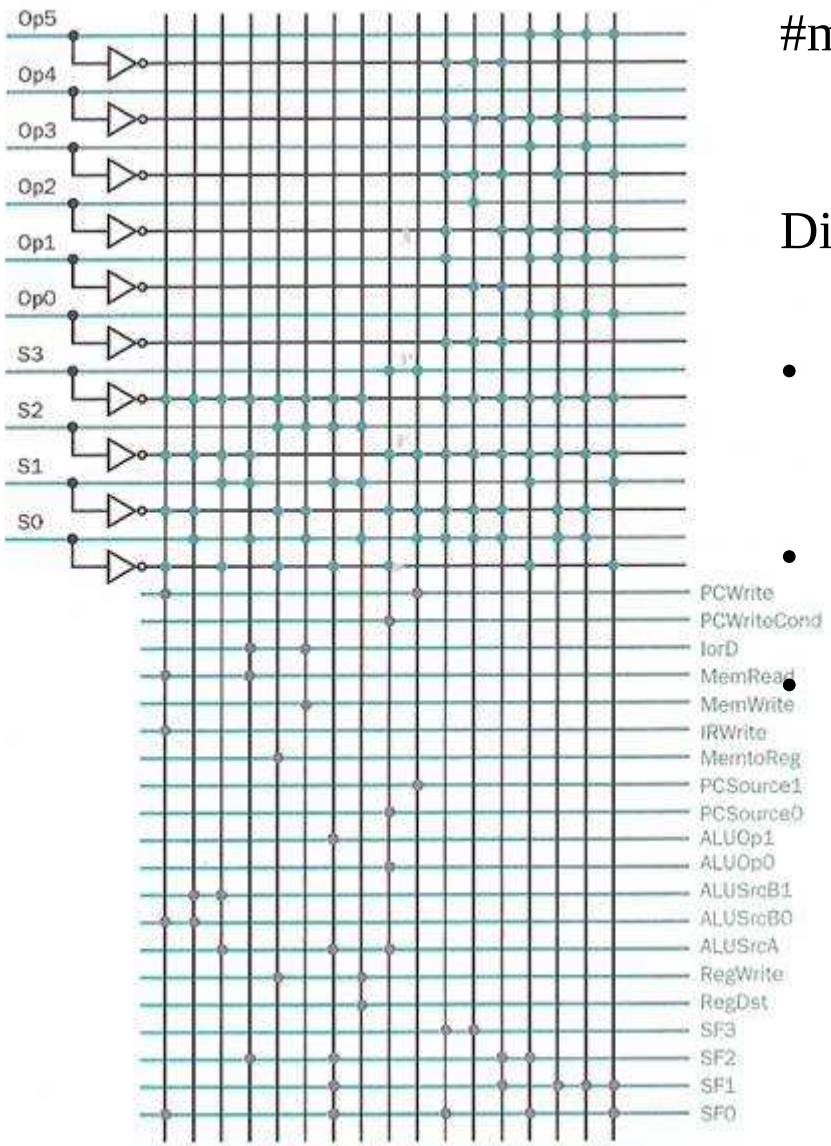
Implementazione tramite PLA (Programmable Logic Array)



Idea di base:

- codificare **solo** le combinazioni di ingresso richieste:
  - circuito logico in forma somma-prodotto
- Piano AND:
  - Input:
    - 1 riga orizzontale per ogni ingresso (+1 per negato)
  - Output:
    - 1 riga verticale per ciascun mintermine del circuito (prodotto logico ingressi)
    - es: mintermine 1= $\overline{S3}$,$\overline{S2}$,$\overline{S1}$,$\overline{S0}$ (in verde)
  - Dimensione: #mintermini*#ingressi
- Piano OR:
  - Input:
    - 1 riga verticale per ogni mintermine
  - Output
    - 1 riga orizzontale per ogni uscita (somma logica dei mintermini in grigio)
  - Dimensione: #mintermini*#uscite

Implementazione tramite PLA



#mintermini unici: 17

- ottenuti da tabelle di verità in Step II e III
- #mintermini indipendenti da opcode: 10

Dimensione Complessiva:

(10*17)+(17*20)=460 celle PLA

- la dimensione fisica della cella base del PLA è solo leggermente più grande della cella base per una ROM!
- per la ROM erano stati necessari 20Kbit (o 4Kbit usando 2 ROMS)
- Usando 2 PLAs (1 per SF e 1 per SC):
  - PLA_SC:
    - 4 ingressi, 10 mintermini, 16 uscite
    - (4*10)+(10*16)=200 celle PLA
  - PLA_SF:
    - 10 ingressi,7 mintermini, 4 uscite
    - (10*7)+(7*4)=98 celle PLA
  - Dimensione complessiva: 298 celle PLA