Università degli Studi di Padova
Corso di Laurea in Matematica
A.A. 2008-2009

INTRODUZIONE ALLA PROGRAMMAZIONE

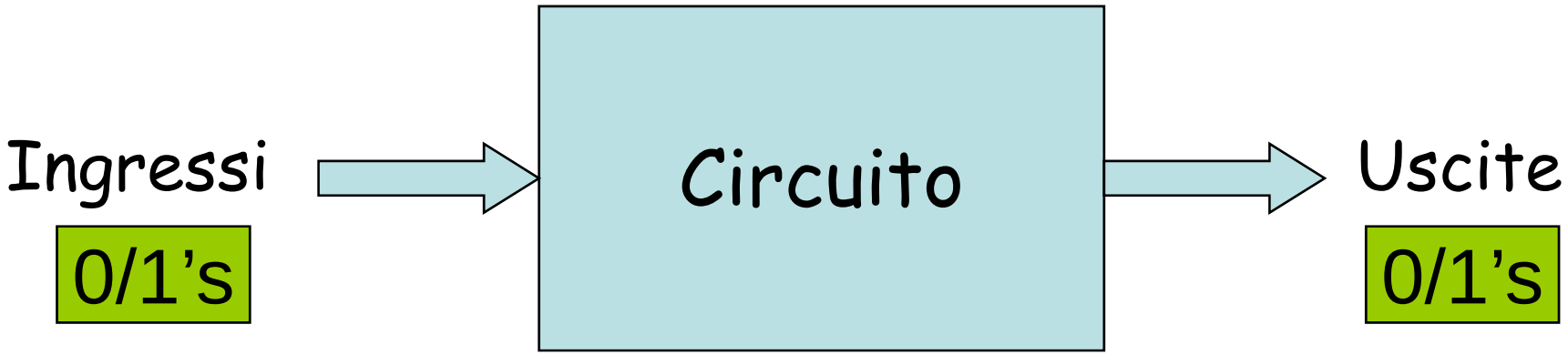
Barbara Di Camillo

Si ringraziano il Dott. Enrico Grisan e il Dott. Fabio Aiolli per il materiale didattico fornito

Parte III

Hardware

Circuiti Digitali



- Un circuito (digitale) puo' essere descritto
  - Mostrando i dettagli realizzativi in termini di circuiti elementari (porte)
  - Mediante una tavola di verita' che mostra i valori in uscita x tutti i possibili ingressi

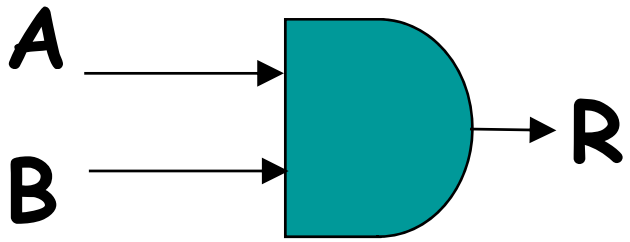
L' Hardware di un computer

I componenti di un computer sono realizzati con un **gran numero** di componenti elettroniche "molto semplici" dette *porte logiche*

Solamente 3 tipi di porte (base):

- AND ("e")
- OR ("o")
- NOT ("non")

AND



Fornisce tensione all'output **R** se e solamente se vi e` tensione in **entrambi** gli input **A** e **B**

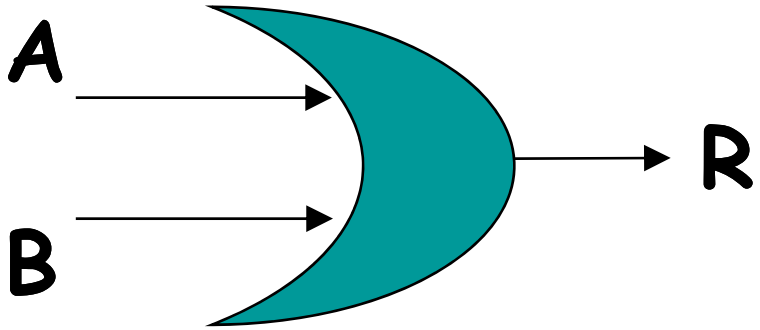
A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

Tavola di verità

1 = VERO, 0 = FALSO

1 = tensione, 0 = no tensione

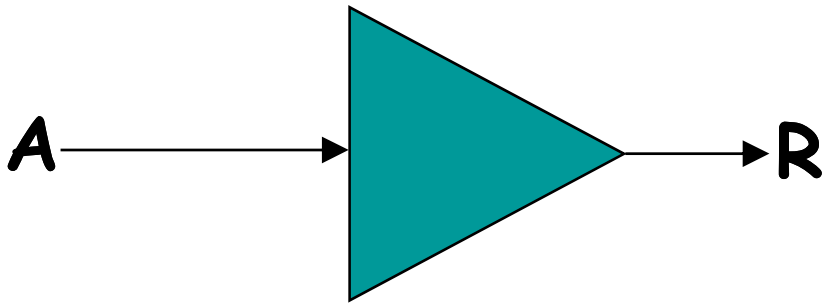
OR



A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

Fornisce tensione all'output **R**
se e solamente se vi e` tensione
in **almeno uno** degli input **A** e **B**

NOT



A	R
0	1
1	0

Fornisce tensione all'output **R** se e solamente se **non** vi e` tensione all'input ***A***

Assemblando queste componenti possiamo costruire nuovi circuiti piu'complessi..

A	Ris
0	0/1
1	0/1

A	B	Ris
0	0	0/1
0	1	0/1
1	0	0/1
1	1	0/1

Ci sono $4 = 2^{(2^1)}$ possibili tabelle di verita` ad una entrata
e $16 = 2^{(2^2)}$ possibili tabelle di verita`a due entrate.
In generale, ci sono $2^{(2^N)}$ tabelle di verita` con N entrate.
I circuiti corrispondenti **si possono tutti realizzare**
componendo i circuiti elementari AND, OR e NOT .

Tavola di verita' $A \Rightarrow B$

IMPLICAZIONE LOGICA

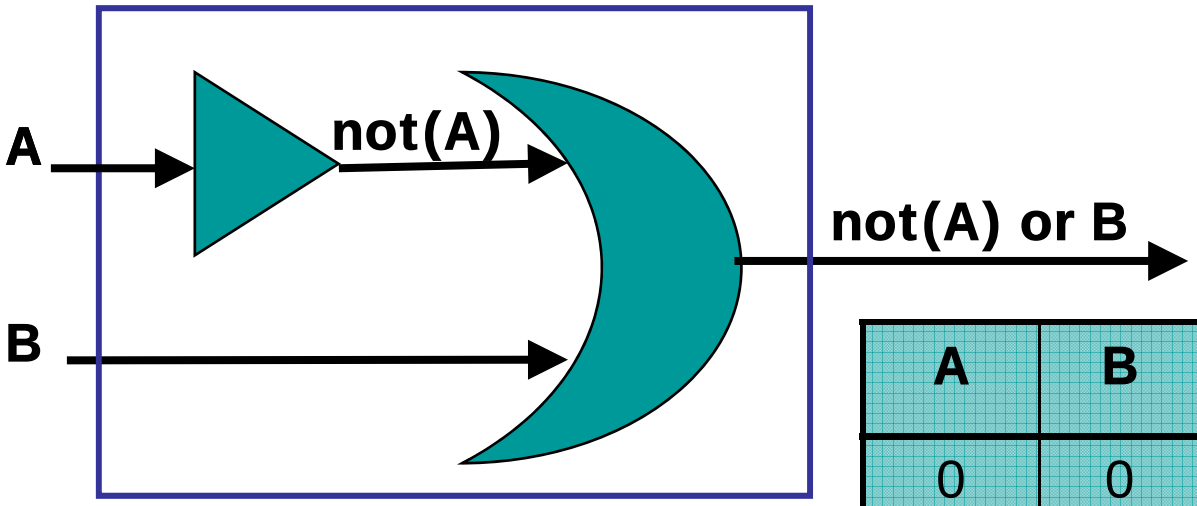
se A allora B $A \Rightarrow B$

ogni volta che A e' VERO anche B deve essere VERO

A	B	A⇒B
0	0	1
0	1	1
1	0	0
1	1	1

Come costruire un circuito che realizza questa tavola di verita`?

Il seguente circuito realizza l'implicazione logica $A \Rightarrow B$



A	B	not(A)	not(A) or B
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

Infatti vale che

$A \Rightarrow B \cong \text{not}(A) \text{ or } B$

tavola di verità ⇔ circuito

- Dato un qualsiasi circuito e' sempre possibile definire la tavola di verita' (in un solo modo)

- Data una tavola di verita' si possono costruire in generale piu' circuiti che la realizzano

Equivalenza Logica

A ⇔B Ogni volta che A è VERO anche B è VERO e viceversa.

Ossia, A e' VERO se e solo se B e' VERO

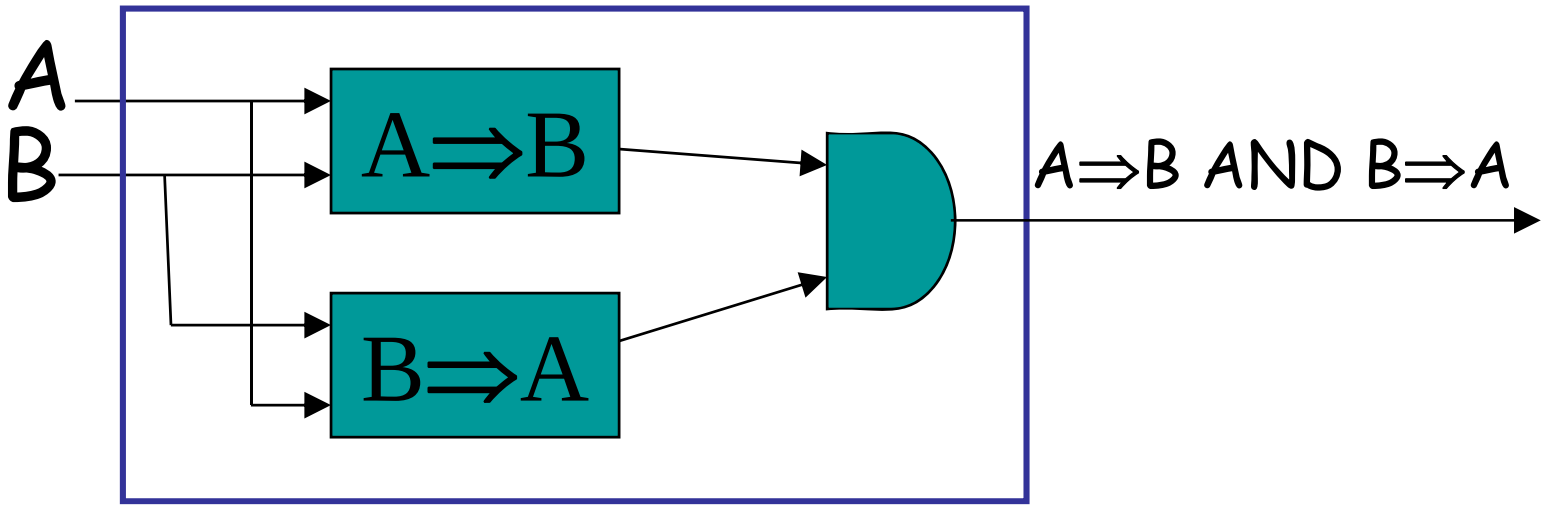
A	B	A⇔B
0	0	1
0	1	0
1	0	0
1	1	1

Notiamo che l'equivalenza è una doppia implicazione:

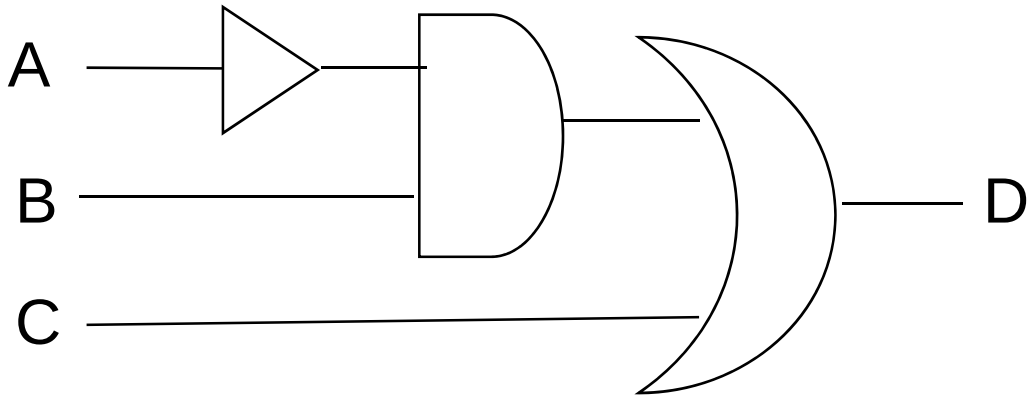
$$A \Leftrightarrow B \quad \cong \quad (A \Rightarrow B) \quad AND \quad (B \Rightarrow A)$$

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

Possiamo quindi realizzare l'equivalenza logica tramite il seguente circuito



Come ottenere una tabella di verita' dato un circuito?



D = ……

A	B	C	D
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Due Modi

- Modo 1) Si calcola, per ogni possibile configurazione degli ingressi, l'uscita delle porte fino alle uscite del circuito

- Modo 2) Si calcola la formula logica corrispondente al circuito e si calcolano le tabelle di verita' partendo dalle formule intermedie piu' semplici fino alla formula data

Come si costruisce un circuito che implementi una tavola di verita'?

Il modo + semplice

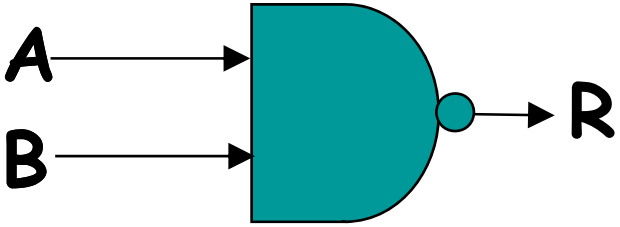
Passo 1) Per ogni uscita nella tabella uguale a 1, si crea un 'circuito riconoscitore' della corrispondente configurazione di ingressi, ovvero una AND tra gli ingressi, negati quando uguali a zero

Passo 2) OR tra le uscite delle AND costruite al passo precedente

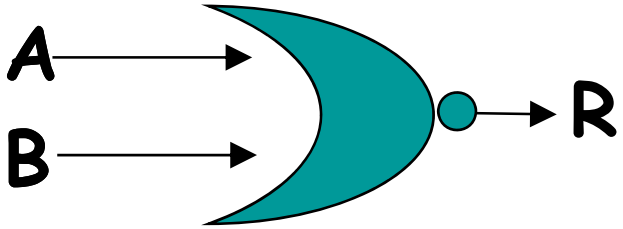
Chiaramente il circuito cosi' costruito non necessariamente e' il piu' semplice (ovvero con il minor numero possibile di porte logiche utilizzate)

Porte NAND e NOR

Si puo' dimostrare che vale il seguente fatto:
ogni tavola di verita` si puo' realizzare componendo un solo circuito elementare tra i due seguenti:
NAND, cioe` **not (A and B)**
NOR, cioe` **not(A or B)**



A	B	R
0	0	1
0	1	1
1	0	1
1	1	0



A	B	R
0	0	1
0	1	0
1	0	0
1	1	0

Porte NAND e NOR

Si puo` dimostrare che
- A and B ≅ (A nand B) nand (A nand B)
- A or B ≅ (B nand B) nand (A nand A)
- not(A) ≅ A nand A

Esercizio 1:

Verificare che le tabelle di verita' coincidono nei tre casi sopra.

Esercizio 2:

Mostrare come è possibile realizzare i circuiti AND, OR e NOT utilizzando solo porte NOR

Verificare il risultato usando le tabelle di verita'

Porte NAND e NOR

Una CPU si puo` realizzare stampando su silicio una griglia di milioni di porte logiche tutte uguali: NAND o NOR.

I vantaggi dell'architettura NOR sono un'elevata affidabilità nel tempo ed un veloce accesso in lettura.
Le memorie NAND, hanno in genere densità più elevate e migliori velocità in scrittura.

Riassumendo: Si possono comporre circuiti semplici per ottenenere circuiti che realizzano nuove operazioni piu` complesse

Ma cosa si riesce a fare con le porte logiche?

Somma tra numeri binari (Ripasso)

La somma di due numeri interi in rappresentazione binaria si effettua in modo analogo alla somma di interi in rappresentazione decimale: si sommano a due a due le cifre corrispondenti a partire da destra e se la somma e` maggiore o uguale alla base (2 per la rappresentazione binaria, 10 per la rappresentazione decimale, ecc.) si ha un **riporto** che si deve sommare alle due cifre successive.

```
1    11       riporti
 1010011 +
 1100011 =
----------
10110110
```

Proviamo a costruire un circuito per sommare due numeri interi

```
10000110     riporti
 1010011 +
 1100011 =
----------
10110110
```

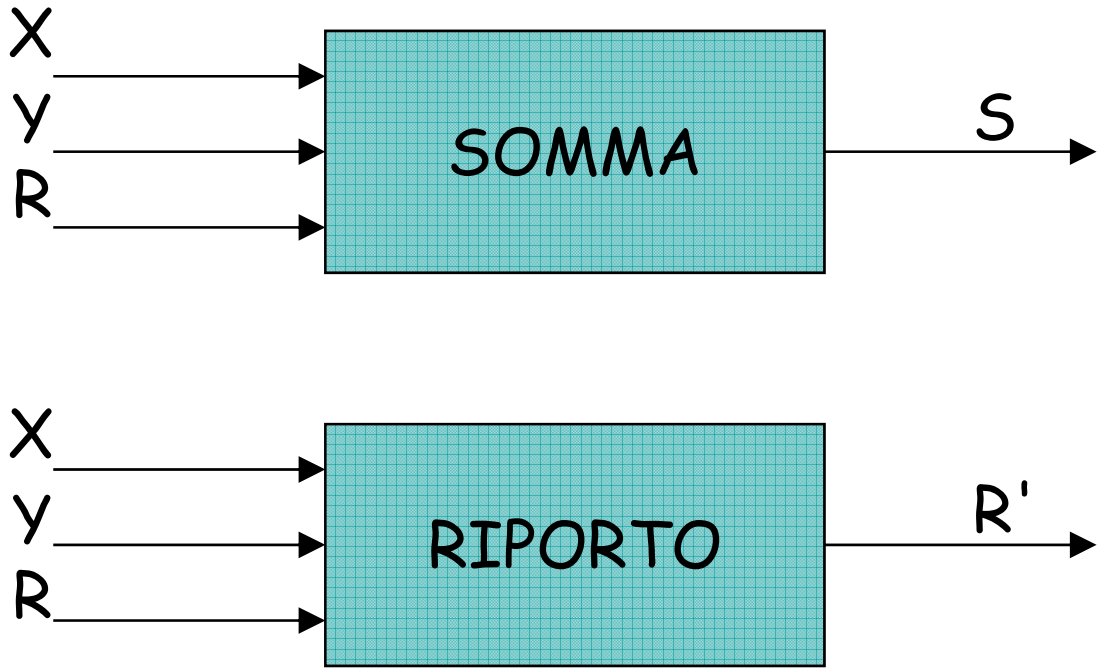
Iniziamo con un circuito che faccia la somma su una colonna

Abbiamo tre cifre binarie X, Y, R in input mentre in output vogliamo ottenere la somma S ed il riporto R'

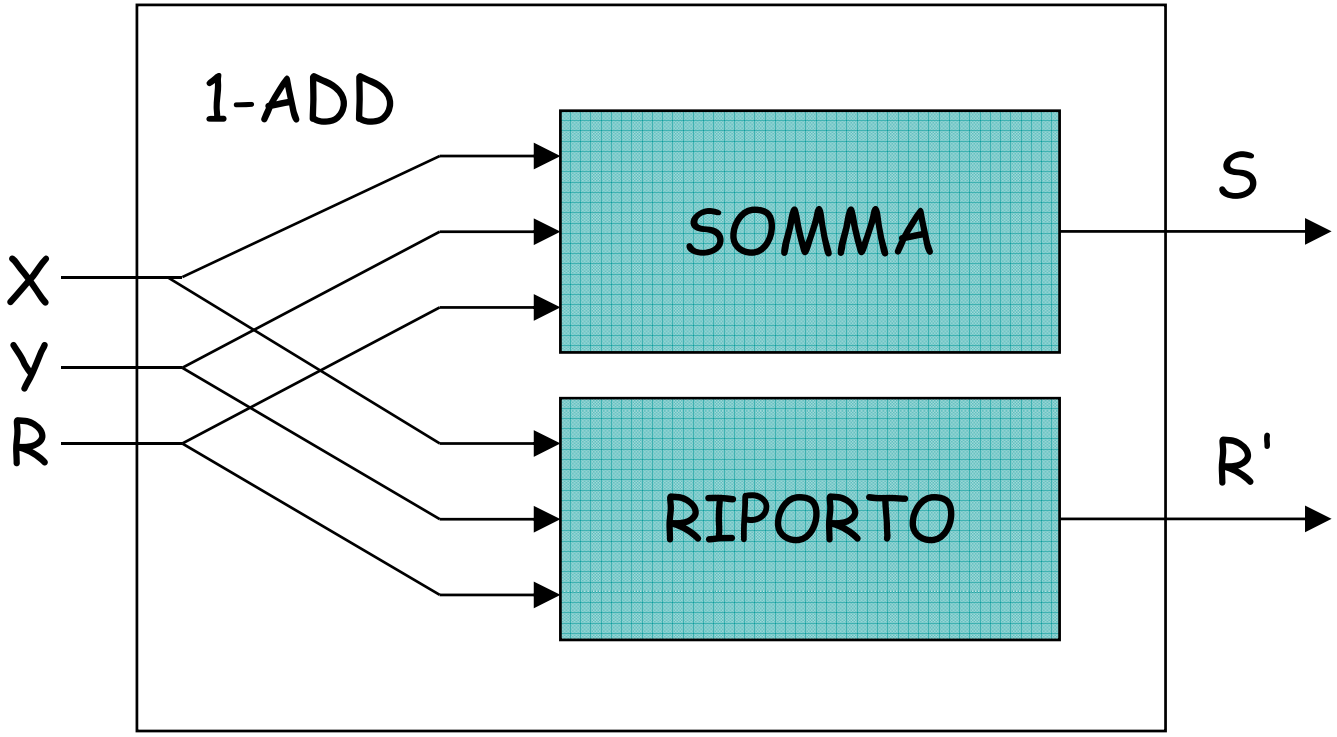
Tabella di verità

X	Y	R	S	R'
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

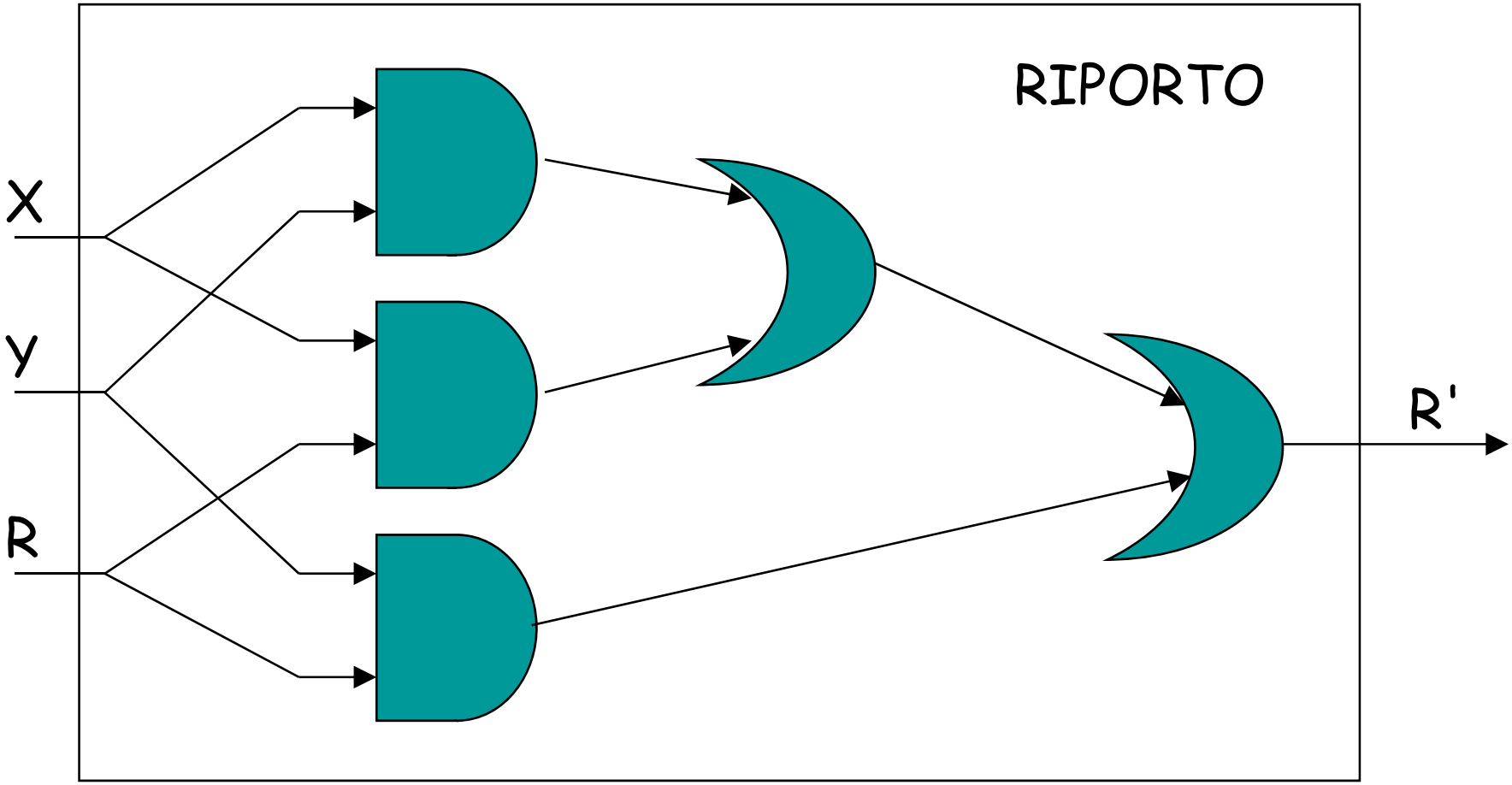
Supponiamo di avere i circuiti che calcolano somma e riporto



Possiamo allora combinare i circuiti SOMMA e RIPORTO per ottenere il seguente circuito 1-ADD

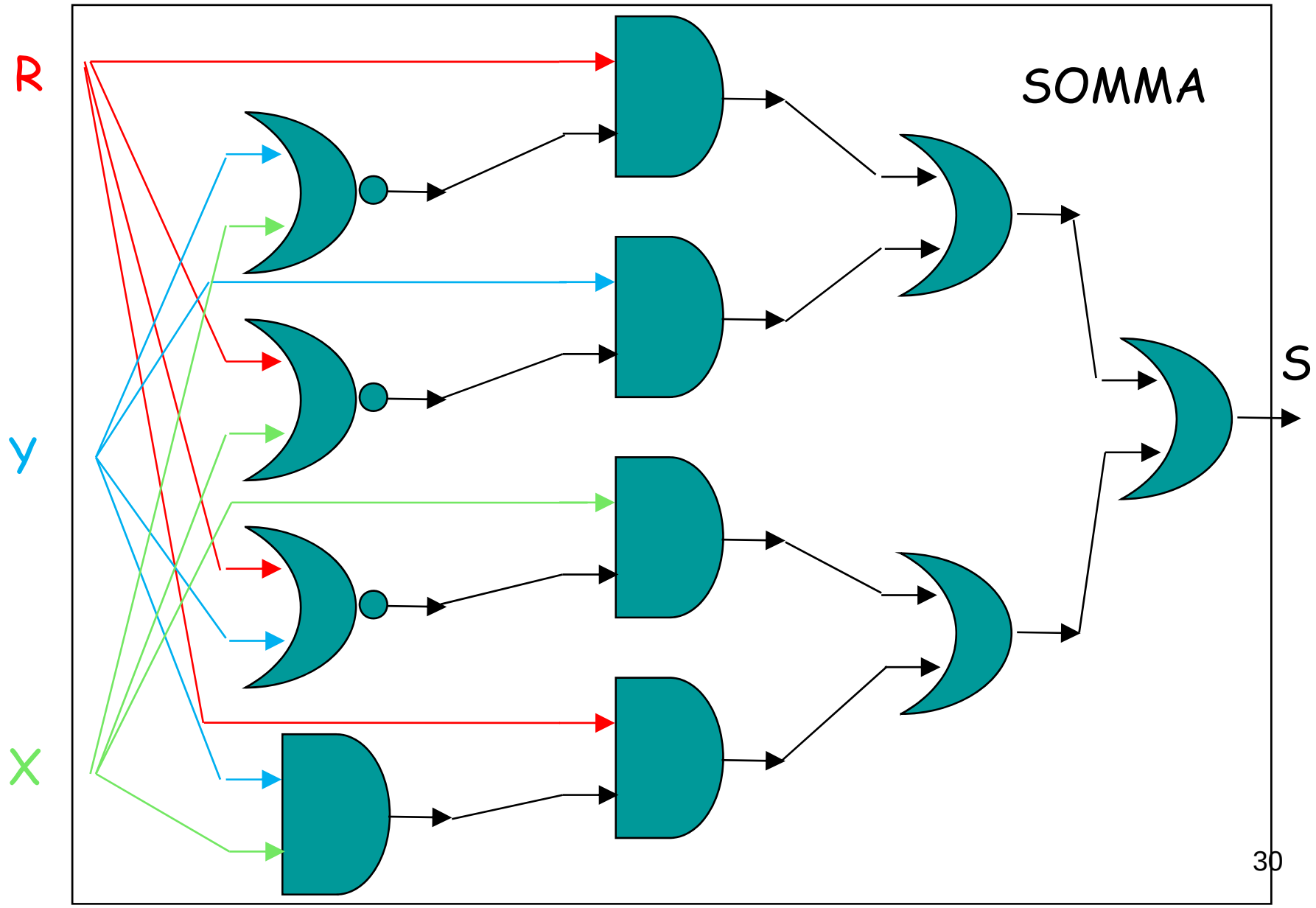


Il circuito RIPORTO puo` essere realizzato nel seguente modo



Basta infatti verificare la corrispondente tabella di verita'.

Il circuito SOMMA puo` essere realizzato nel seguente modo



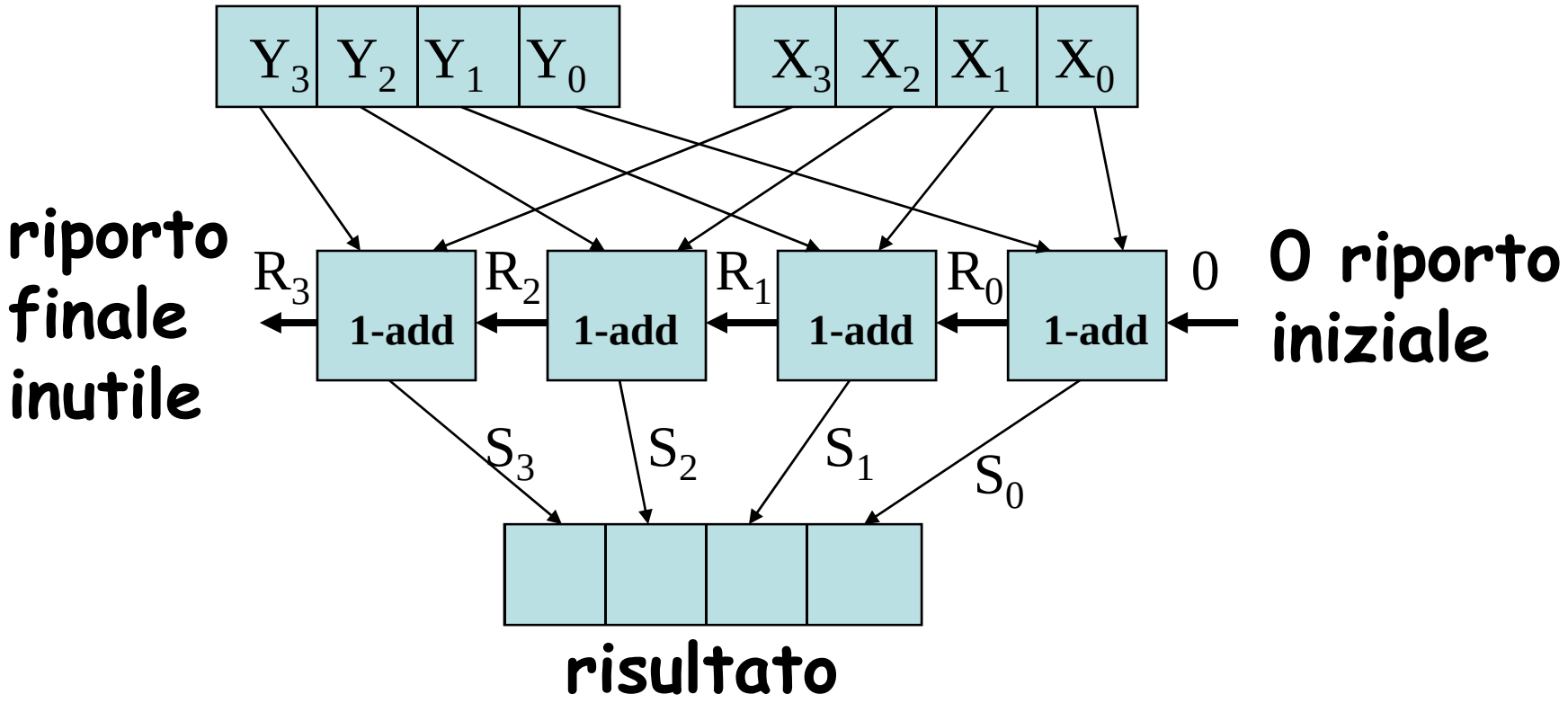
Esercizio:

1. Verificare la tabella di verita per il circuito RIPORTO e del circuito SOMMA

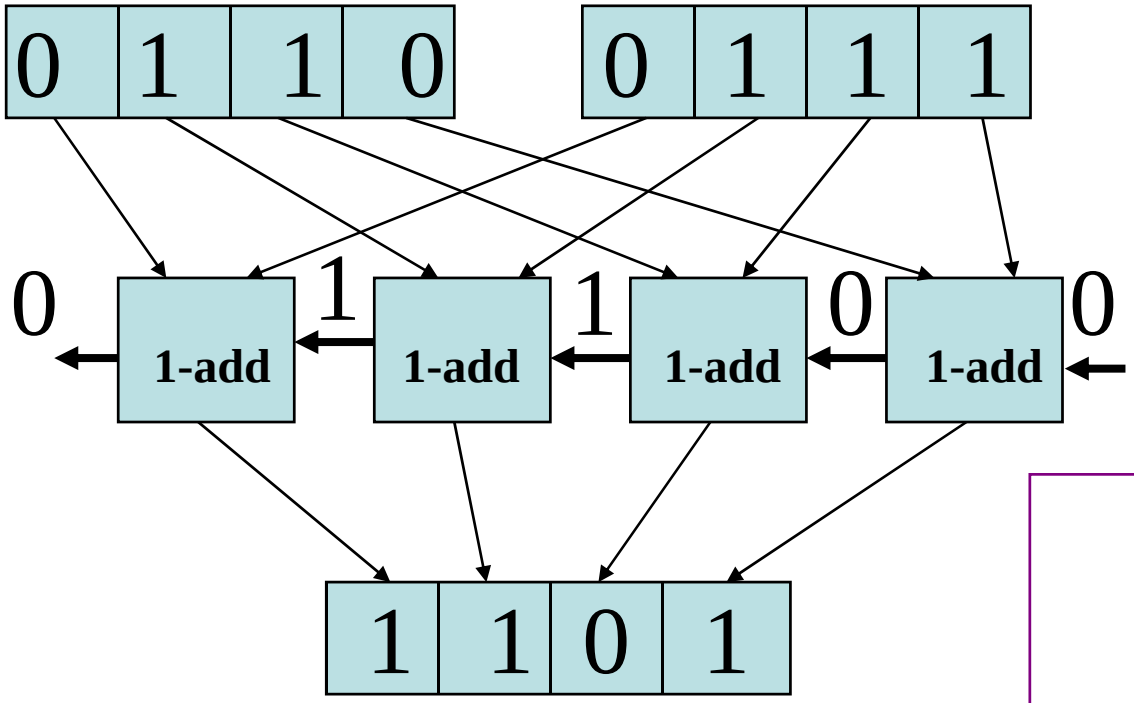
A questo punto componendo K circuiti 1-ADD e` possibile realizzare un circuito K-ADD che somma due numeri binari di K cifre.

Vediamo l'esempio della somma di due numeri binari di 4 cifre.

Somma di numeri di 4 bit



esempio




```
0111 +
0110 =
------
1101
```

Attenzione

Si e` trascurato il problema del cosiddetto **overflow**, ad esempio:

```
  0111 +
  1110 =
 ------
 10101
```

Esercizio

Dire se è vera o no la seguente implicazione:

$$(A \Rightarrow (B \text{ and } C)) \overset{?}{\Rightarrow} (B \Rightarrow A)$$

A	B	C	$A \Rightarrow$ (B and C)	$B \Rightarrow A$	Valore
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	1

QUINDI:

$B \Rightarrow A$ non è conseguenza

Esercizio

Dire se sono vere o no le seguenti implicazioni:

$$(A \Rightarrow (B \text{ and } C)) \overset{?}{\Rightarrow} (A \Rightarrow B)$$

$$(A \Rightarrow B) \overset{?}{\Rightarrow} (\text{not } B \Rightarrow \text{not } A)$$

Esercizio

Definire la tabella di verità ed un circuito per questa operazione

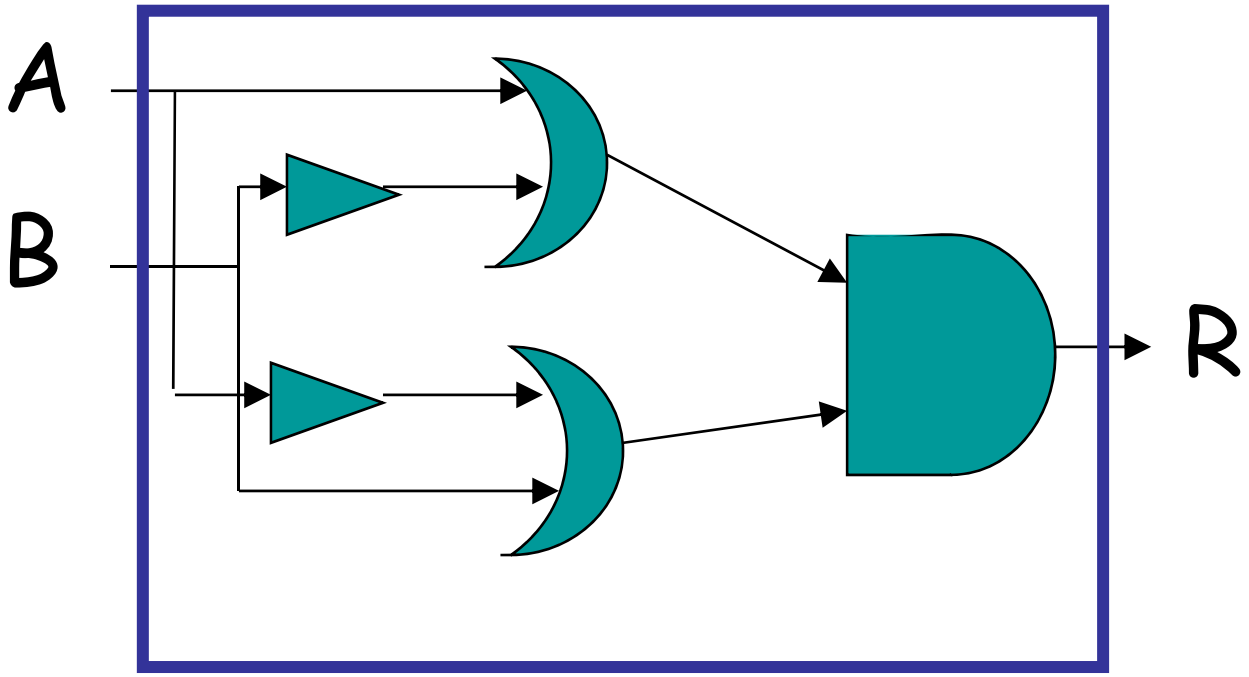
OR esclusivo: XOR

A XOR B è vera quando esattamente uno tra A e B è vero

Suggerimento: possiamo osservare che la tabella di verita` e` "duale" alla tabella di verita` dell'equivalenza.

Esercizio

Determinare la tavola di verità del seguente circuito. E` una tavola nota?



Parte IV

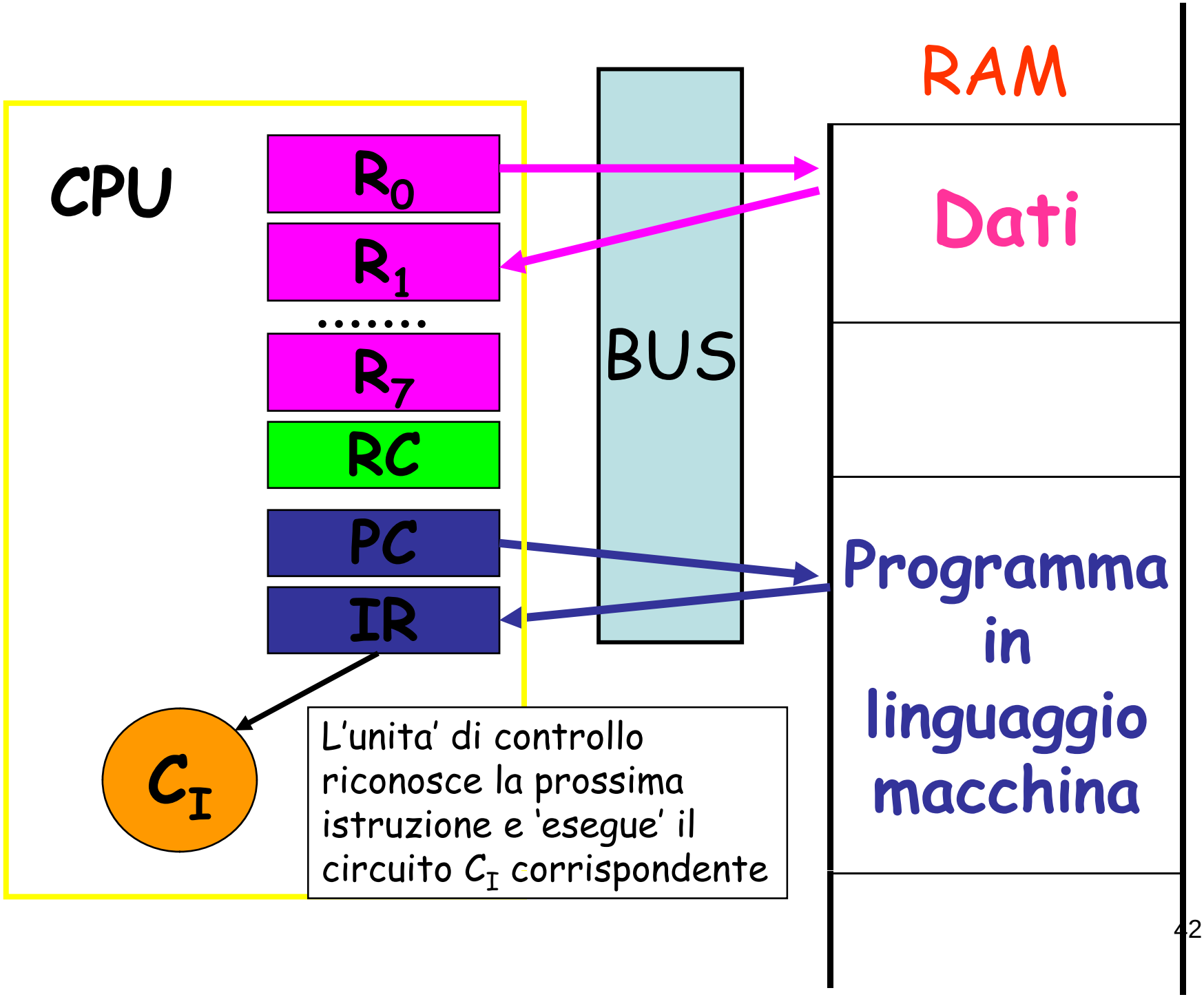
Linguaggio Macchina e Assembler

LINGUAGGIO MACCHINA

Descriveremo una CPU "MINIMA" dotata di un certo insieme di istruzioni I ciascuna realizzata da un corrispondente circuito C_I.

Questo insieme di istruzioni della CPU "MINIMA" costituisce il linguaggio macchina di "MINIMA".

Semplificheremo l'approccio considerando che ogni istruzione sia memorizzata in una parola di memoria di 32 bit.


RAM
CPU
R_0
R_1
.....
R_7
RC
PC
IR
BUS
Dati
Programma in linguaggio macchina
C_I
L'unita' di controllo riconosce la prossima istruzione e 'esegue' il circuito C_I corrispondente

Registro PC (P)

Il Registro PC (P), cioé Program Counter

- ✓ Contiene l'indirizzo in memoria centrale della prossima istruzione da eseguire
- ✓ All'inizio dell'esecuzione di un programma viene caricato con l'indirizzo della prima istruzione di quel programma
- ✓ Ad ogni istruzione eseguita il PC viene modificato per contenere l'indirizzo della istruzione successiva

Registro IR (IP)

Il Registro IR (IP), cioè Instruction Register

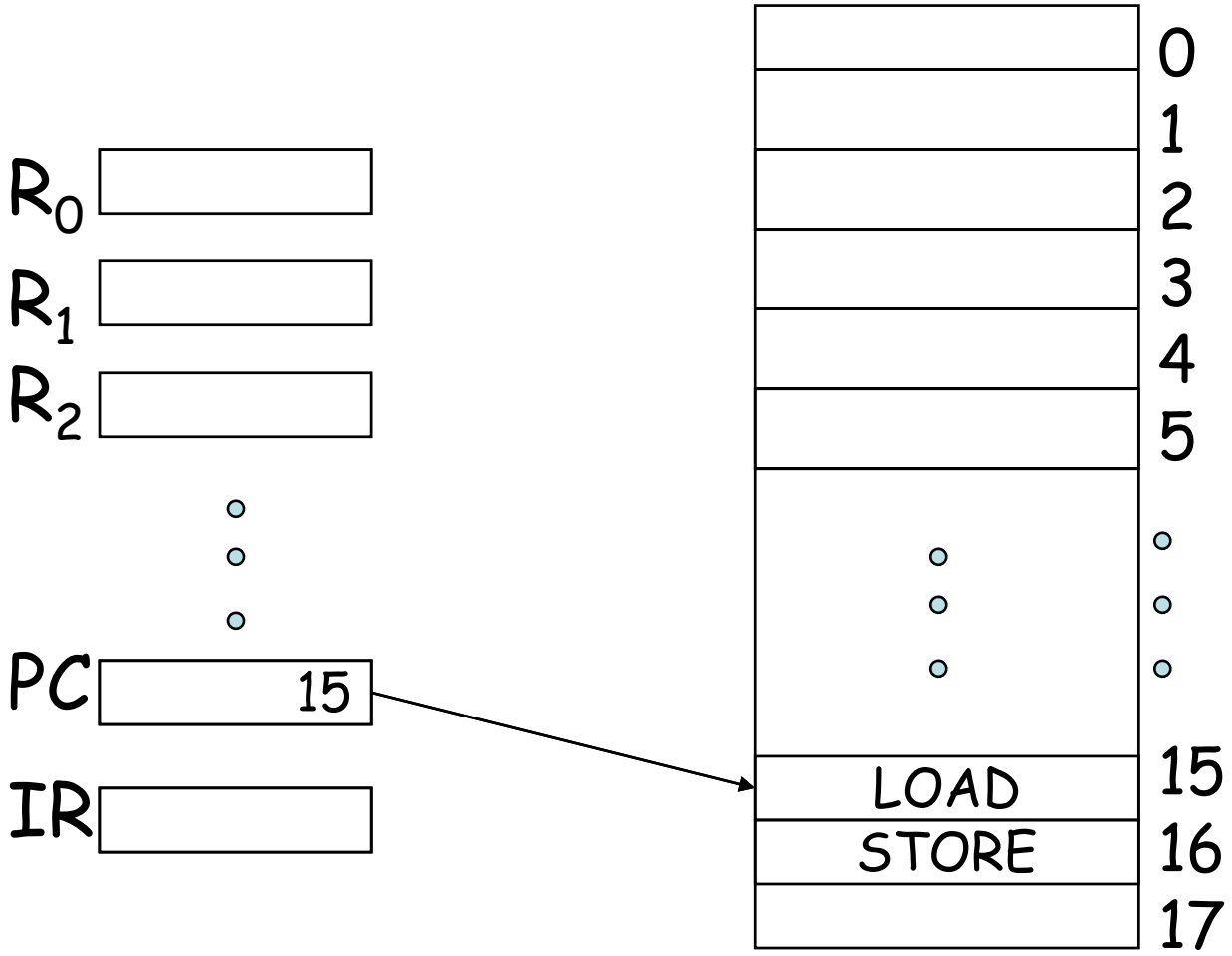
- ✓ Contiene l'istruzione correntemente in esecuzione
- ✓ La CPU legge l'istruzione contenuta nell'instruction register e la esegue

3 tipi di operazioni

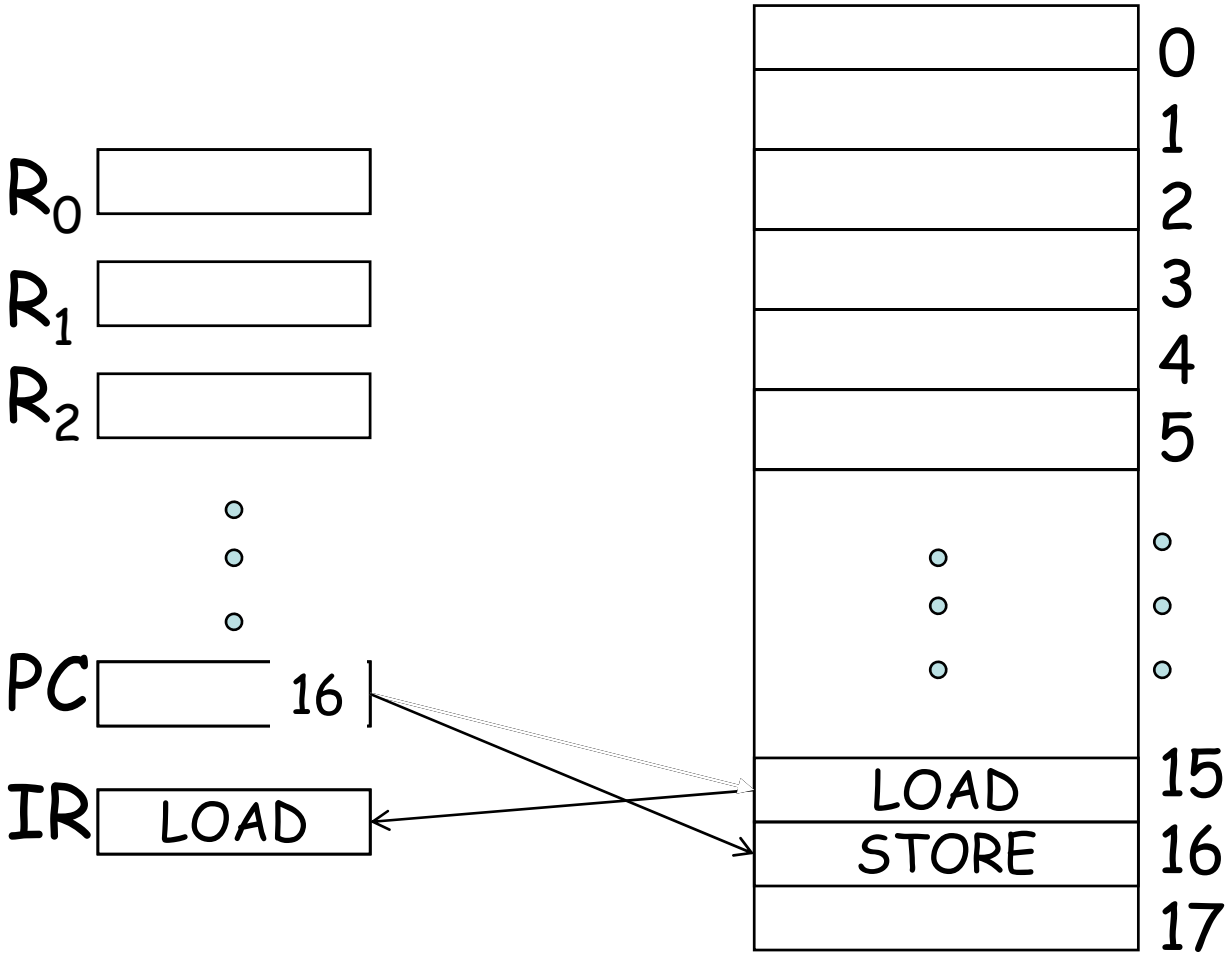
La CPU deve eseguire 3 tipi di operazioni

1) trasferimento di dati tra RAM e registri R0, …, R1 o tra RAM e dispositivi di Input/Output
2) operazioni aritmetiche: somma, differenza, moltiplicazione e divisione
3) operazioni di controllo: confronto, salto e stop

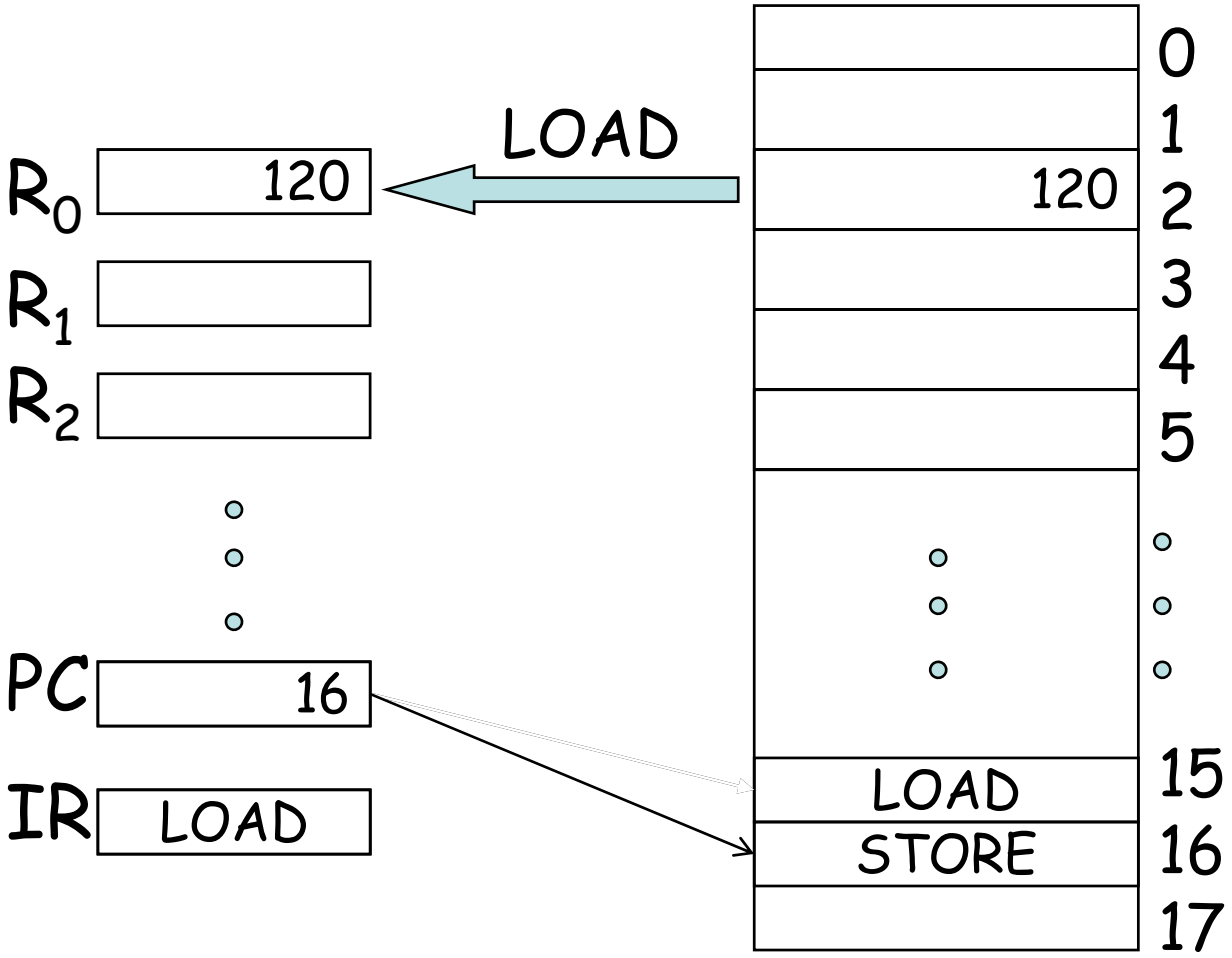
Esempio di utilizzo dei registri durante un'istruzioni di trasferimento: registri ⇔ RAM



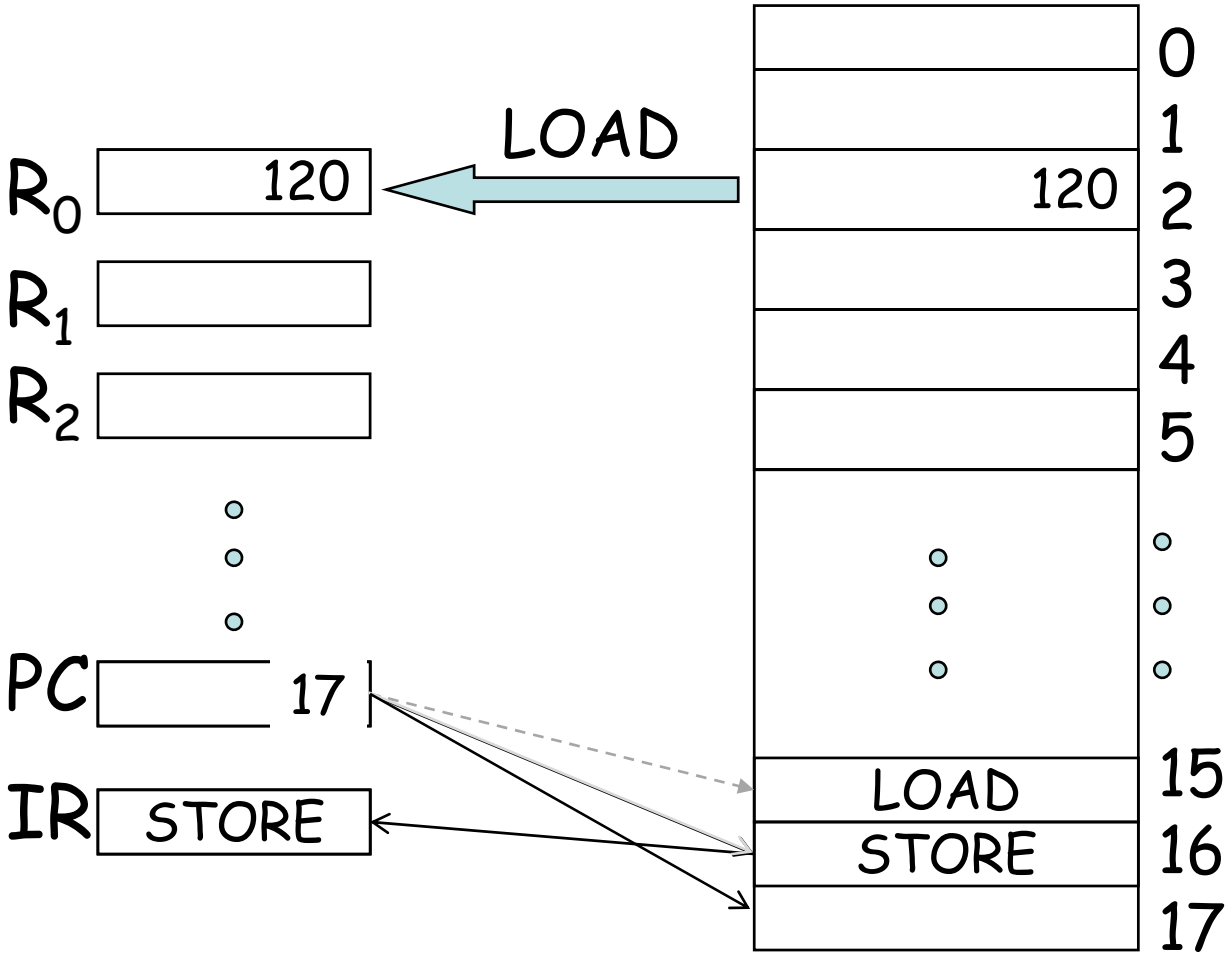
Esempio di utilizzo dei registri durante un'istruzioni di trasferimento: registri ⇔ RAM



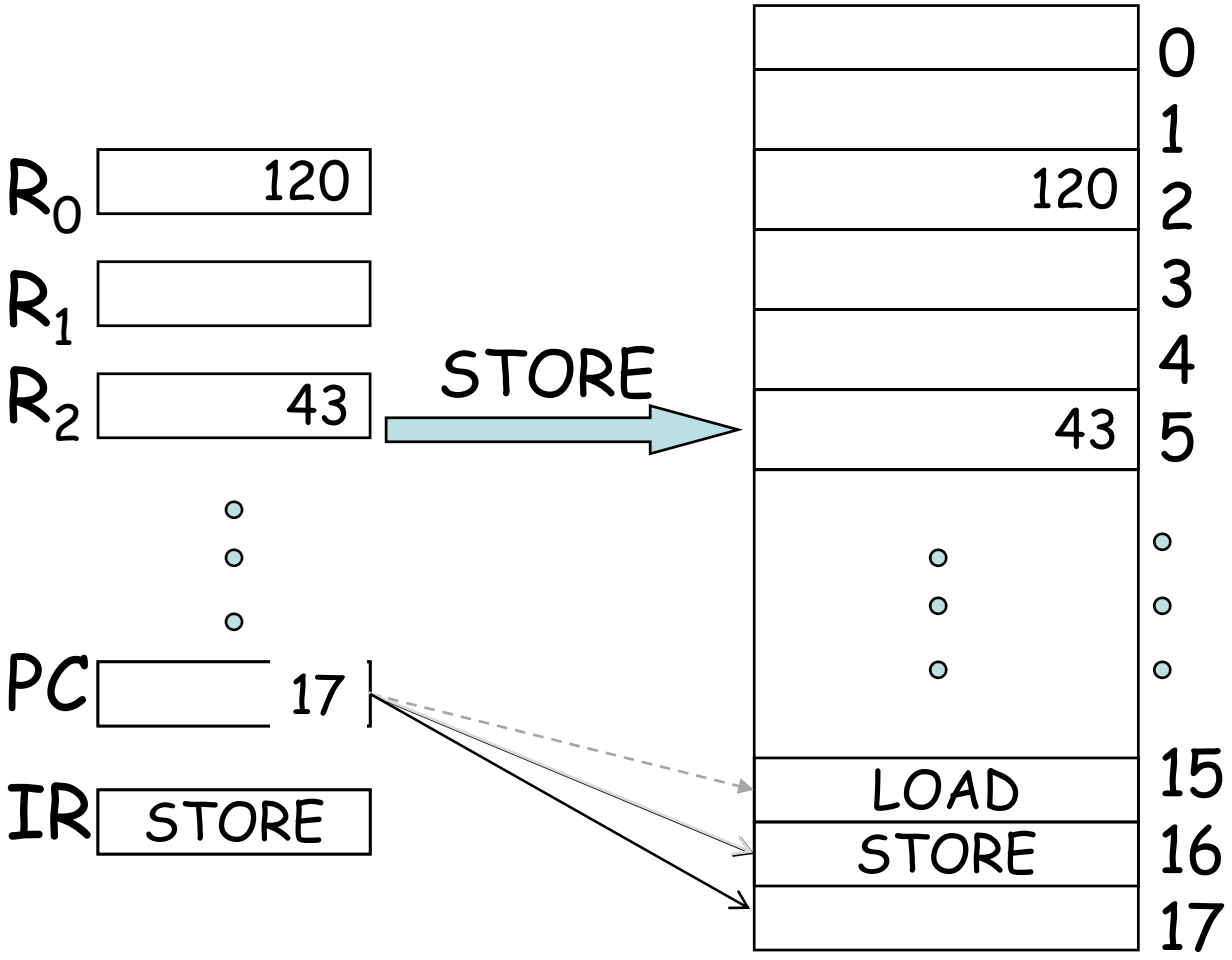
Esempio di utilizzo dei registri durante un'istruzioni di trasferimento: registri ⇔ RAM



Esempio di utilizzo dei registri durante un'istruzioni di trasferimento: registri ⇔ RAM



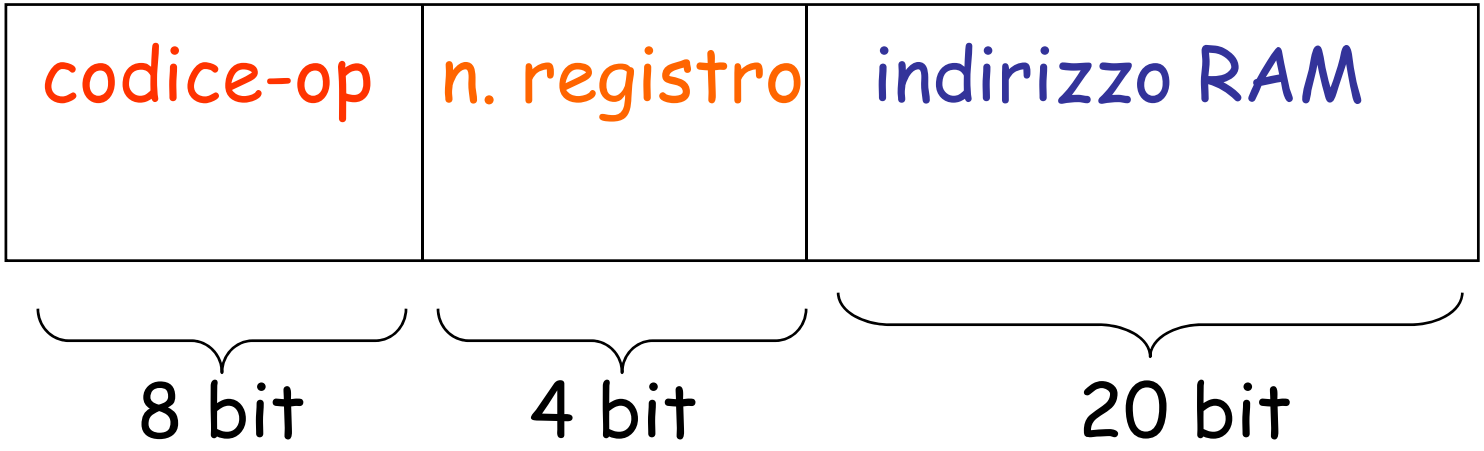
Esempio di utilizzo dei registri durante un'istruzioni di trasferimento: registri ⇔ RAM



Istruzioni di trasferimento: registri $\Leftrightarrow$ RAM

Nella RAM è contenuto un codice per il tipo di istruzione di trasferimento, un codice con l'indirizzo del registro Ri$\in${R0 … Rn} e un codice con l'indirizzo della RAM in cui andare a prelevare o caricare i dati.

Formato delle istruzioni di trasferimento



Codici:

LOAD	**00000000**
STORE	**00000001**

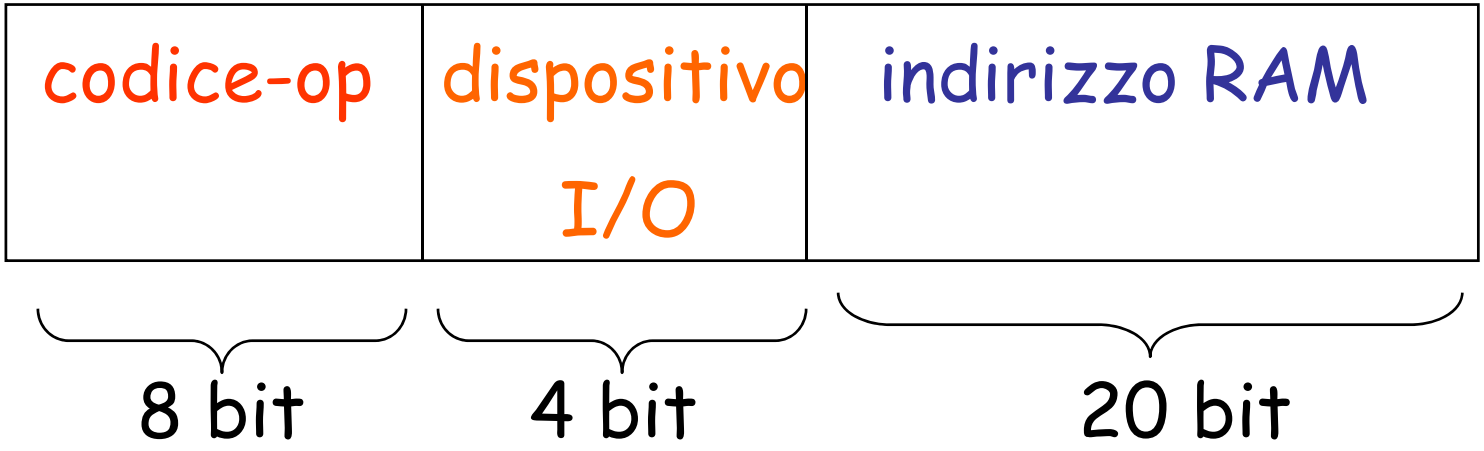
Esempio:

```
00000000 0000 00000000000000000010
00000001 0010 00000000000000000101
```

Istruzioni di trasferimento: input/output ⇔ RAM

Nella RAM è contenuto un codice per il tipo di istruzione di trasferimento, un codice con l'indirizzo del dispositivo di input/output e un codice con l'indirizzo della RAM in cui andare a prelevare o caricare i dati.

Formato delle istruzioni di trasferimento



Codici:

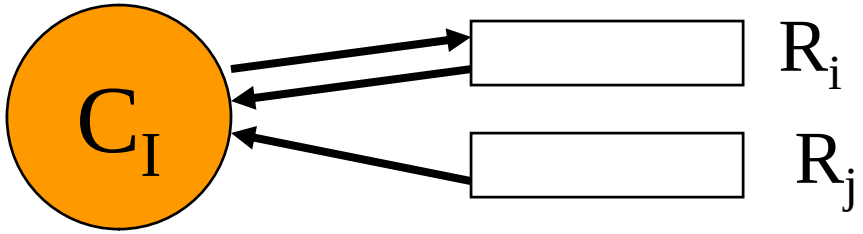
READ	**00010000**
WRITE	**00010001**

Esempio:

00010000 0000 00000000000000000010
00010001 0010 00000000000000000101

Istruzioni aritmetiche

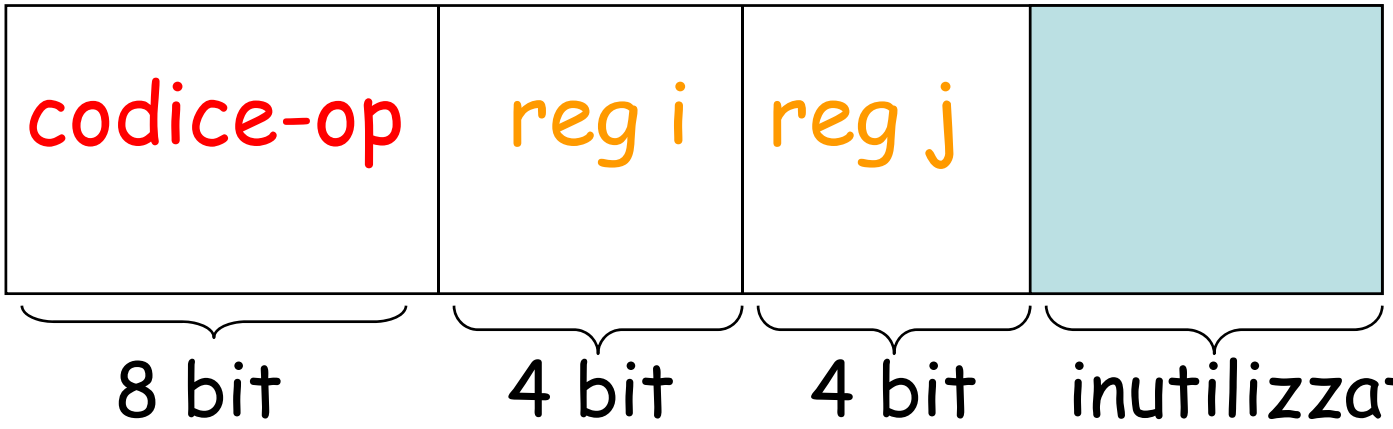
Eseguono somma, differenza, moltiplicazione e divisione usando i registri come operandi. Memorizzano il risultato nel primo registro argomento.



Istruzioni aritmetiche

Nella RAM è contenuto un codice per il tipo di istruzione aritmetica, un codice per il registro $Ri \in \{R0 \ldots Rn\}$ che contiene l'operando 1 e un codice per il registro $Rj \in \{R0 \ldots Rn\}$ che contiene l'operando 2.

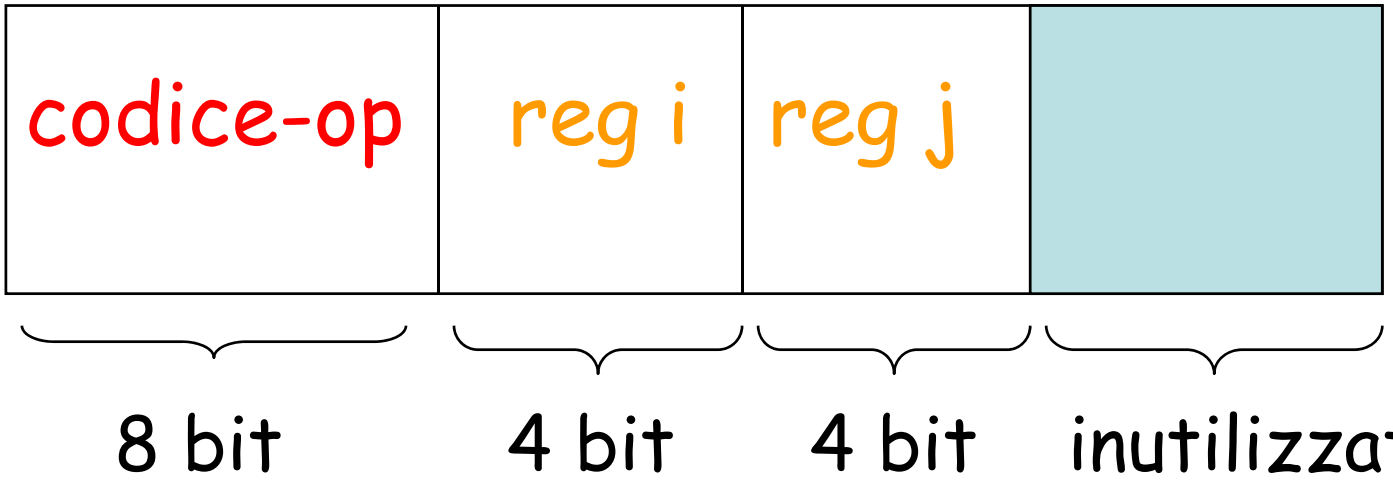
Formato delle istruzioni aritmetiche



Codici Operazioni

Tra Interi		Tra reali	
ADD	00000010	FADD	00000011
SUB	00000100	FSUB	00000101
MULT	00000110	FMULT	00000111
DIV	00001000	FDIV	00001001
MOD	00001010		

Formato delle istruzioni aritmetiche



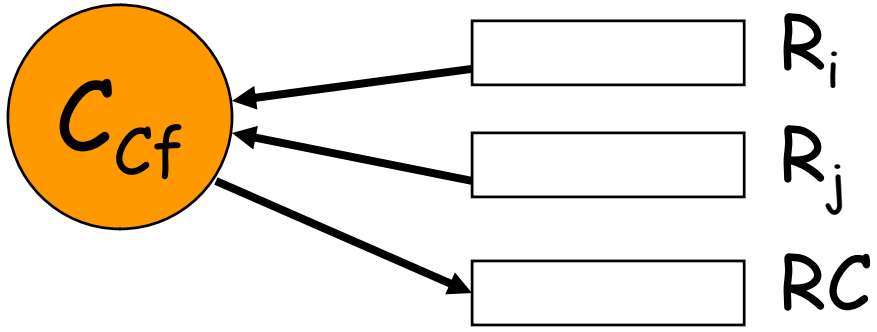
Esempio: **00000010 0011 0001** xxxxxxxxxxxxxxxx
00000011 0011 0001 xxxxxxxxxxxxxxxx

Istruzione di confronto

Paragona il contenuto di 2 registri R_i e R_j

- se $R_i < R_j$ memorizza -1 nel registro RC
- se $R_i = R_j$ memorizza 0 in RC
- se $R_i > R_j$ memorizza 1 in RC

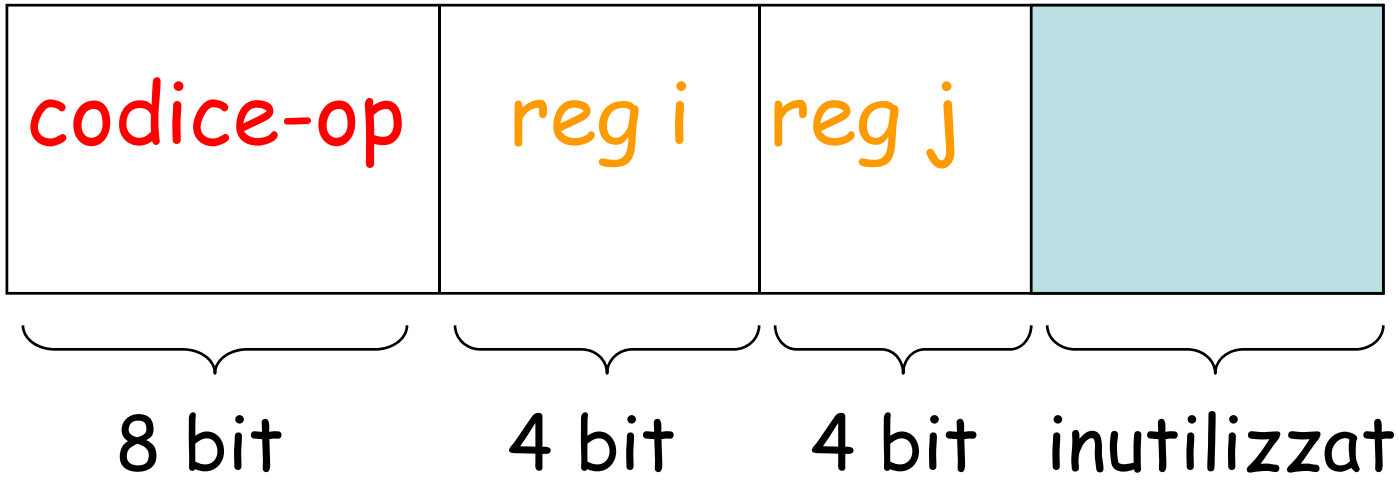
RC: Registro dei Confronti



Istruzioni di confronto

Nella RAM è contenuto un codice per il tipo di istruzione di confronto, un codice per il registro Ri$\in${R0 … Rn} che contiene l'operando 1 e un codice per il registro Rj$\in${R0 … Rn} che contiene l'operando 2.

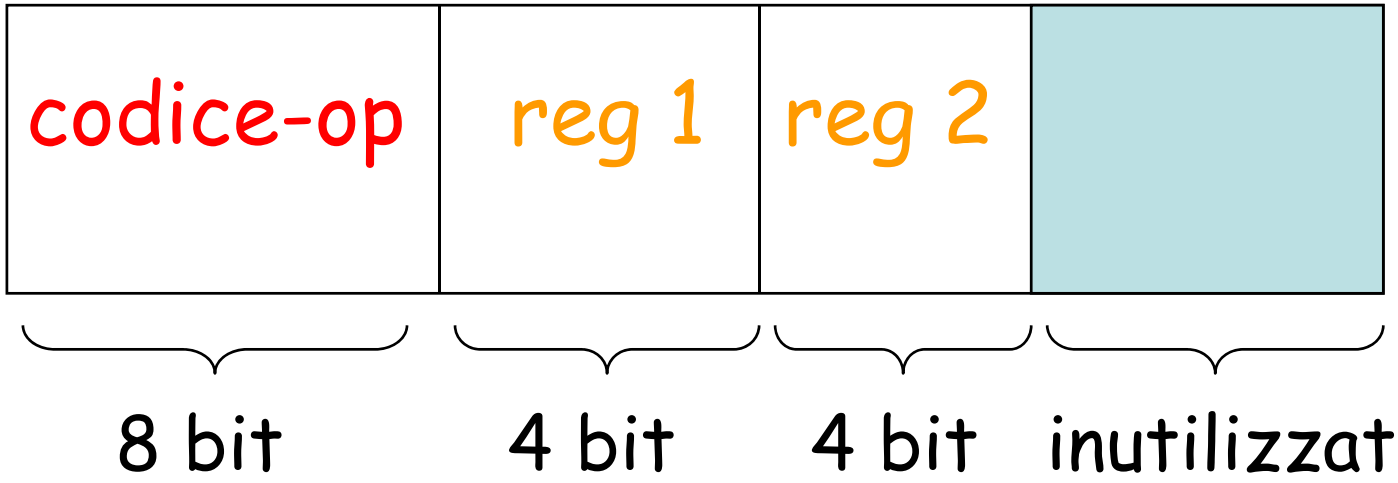
Formato dell'istruzione di confronto



Codici:

COMP	**00100000**
FCOMP	**00100001**

Formato dell'istruzione di confronto



Esempio: **00100000 0010 0101 xxxxxxxxxxxxxxxx**
00100001 0010 0101 xxxxxxxxxxxxxxxx

Istruzione di salto

Permette di "saltare" ad un'altra istruzione del programma a seconda del contenuto del registro RC (cioè a seconda del risultato di un confronto)

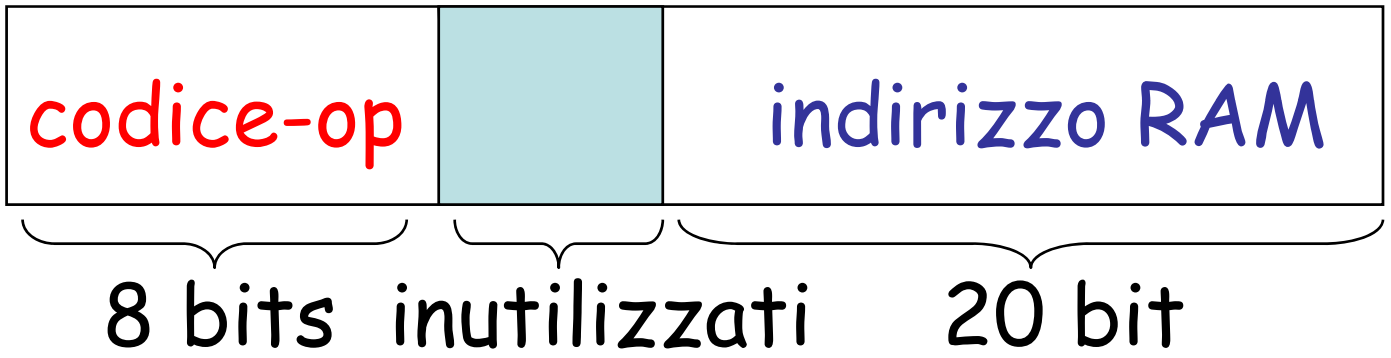
Istruzioni di salto

Nella RAM è contenuto un codice per il tipo di istruzione di salto, e un codice con l'indirizzo della RAM a cui saltare

L'effetto di una istruzione di salto ad un indirizzo M e` quindi quello di memorizzare M nel registro PC se si verifica la relativa condizione nel registro RC

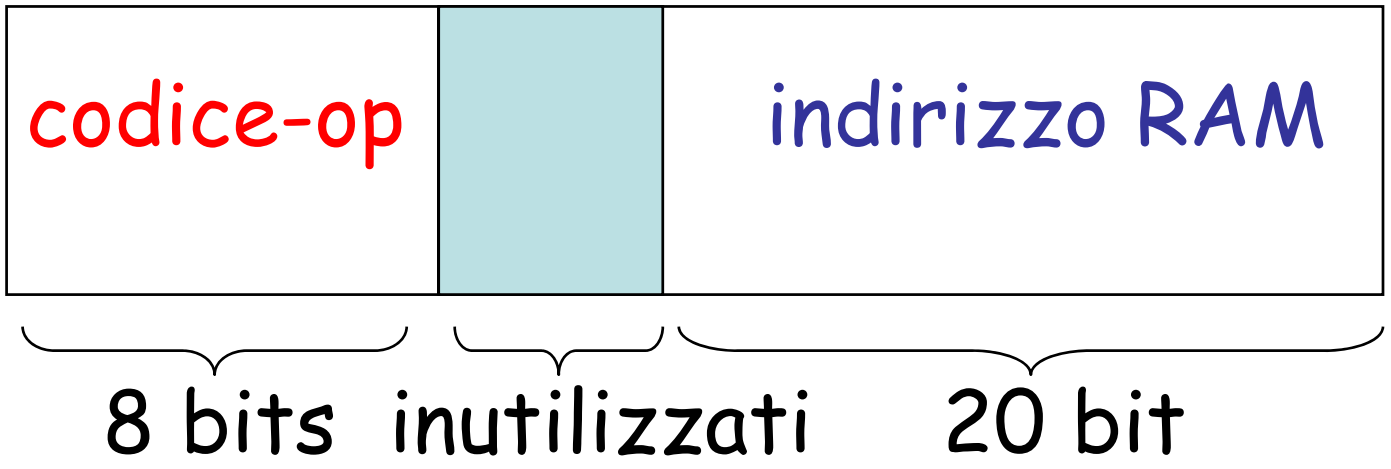
Slide modificata
Modificare tab. 1.16 pg. 33 del testo

Formato dell'istruzione di salto



BRLT	(BRanch if Lower Than)	**01000001**	**se RC=-1poni PC=M**
BRLE	(BRanch if Lower or Equal)	**01000010**	**se RC<=0poni PC=M**
BREQ	(BRanch if Equal)	**01000011**	**se RC=0 poni PC=M**
BRNE	(BRanch if Not Equal)	**01000100**	**se RC≠0 poni PC=M**
BRGT	(BRanch if Greater Than)	**01000101**	**se RC=1 poni PC=M**
BRGE	(BRanch if Greater or Equal)	**01000110**	**se RC>=0poni PC=M**
BRANCH		**10000000**	**poni comunque PC=M**

Formato dell'istruzione di salto



Esempio:

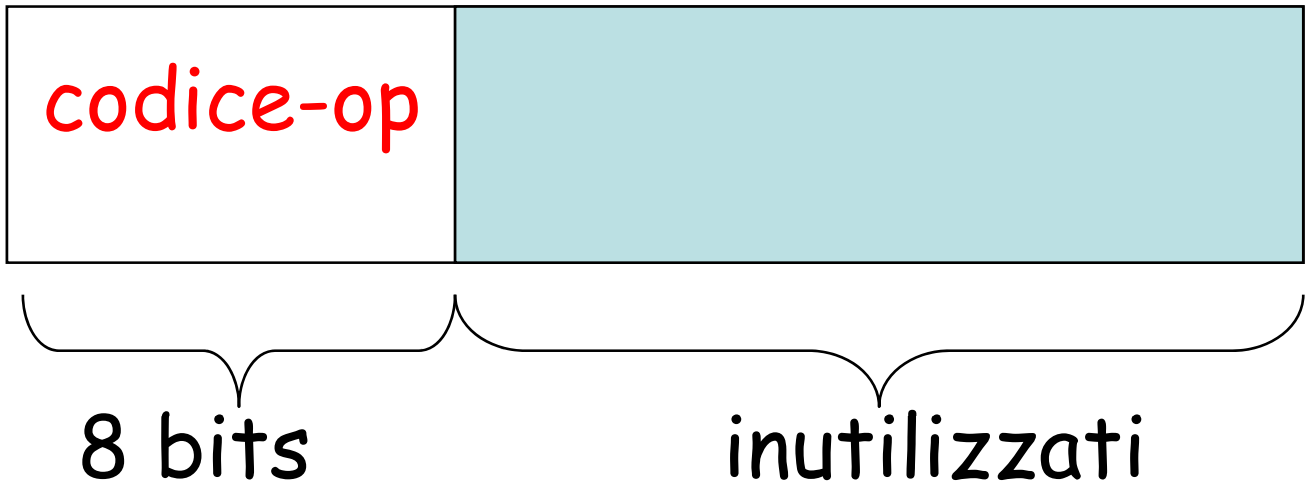
```
01000001 xxxx 00000000000000001001
10000000 xxxx 00000000000000001010
```

Istruzione di stop

Semplicemente termina il programma

Codice: **STOP 10000001**

Formato istruzione di stop

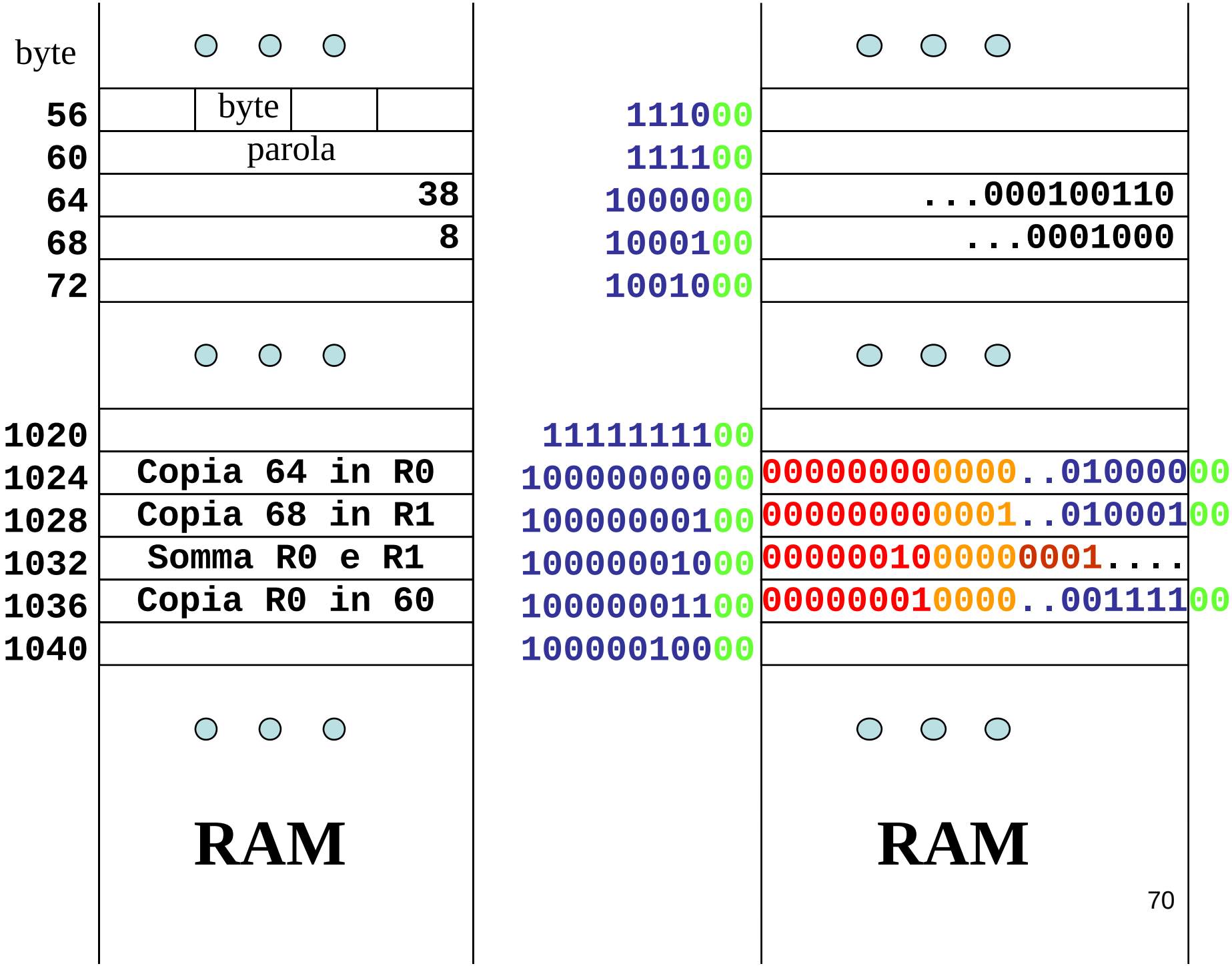


Esempio: **10000001** xxxxxxxxxxxxxxxxxxxxxxxx

Esempio

Scriviamo un programma in linguaggio macchina che:

- trasferisce il contenuto delle 2 parole della RAM di indirizzi 64 e 68 nei registri R_0 e R_1
- somma i contenuti dei registri R_0 ed R_1
- trasferisce il risultato nella parola della RAM all'indirizzo 60


byte
byte
56
parola
60
38
64
8
68
72
1020
Copia 64 in R0
1024
Copia 68 in R1
1028
Somma R0 e R1
1032
Copia R0 in 60
1036
1040
RAM
111000
111100
1000000
1000100
1001000
1111111100
1000000000
1000000100
1000001000
1000001100
1000010000
...000100110
...0001000
000000000000..01000000
000000000001..01000100
000000100000001....
000000010000..00111100
RAM

Svantaggi del linguaggio macchina

- programmi in binario sono difficili da scrivere, capire e modificare
- il programmatore deve occuparsi della gestione degli indirizzi RAM

primo passo $\Rightarrow$ Assembler

Caratteristiche dell'Assembler

L'Assembler e' un rudimentale linguaggio di programmazione

- codici mnemonici per le operazioni
- nomi mnemonici, detti *variabili*, al posto degli indirizzi RAM dei dati
- nomi mnemonici, detti *etichette*, al posto degli indirizzi RAM delle istruzioni usati nei salti
- categorie di valori: si tratta dei cosiddetti *tipi di dato* **INT** e **FLOAT** (normalmente non presenti)

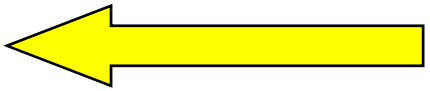
Codici mnemonici di operazioni

- trasferimento: **LOAD** (RAM → CPU) e **STORE** (CPU → RAM)
- aritmetiche: **ADD,SUB,MULT,DIV,MOD, FADD,FSUB,FMULT,FDIV**
- test: **COMP,FCOMP**
- salto: **BREQ,BRGT,BRLT,BRGE,BRLE, BRANCH**
- terminazione: **STOP**

Esempio precedente in Assembler

```
Z : INT ;
X : INT 38;
Y : INT 8;
LOAD R0 X;
LOAD R1 Y;
ADD  R0 R1;
STORE R0 Z;
```

dichiarazioni di variabili

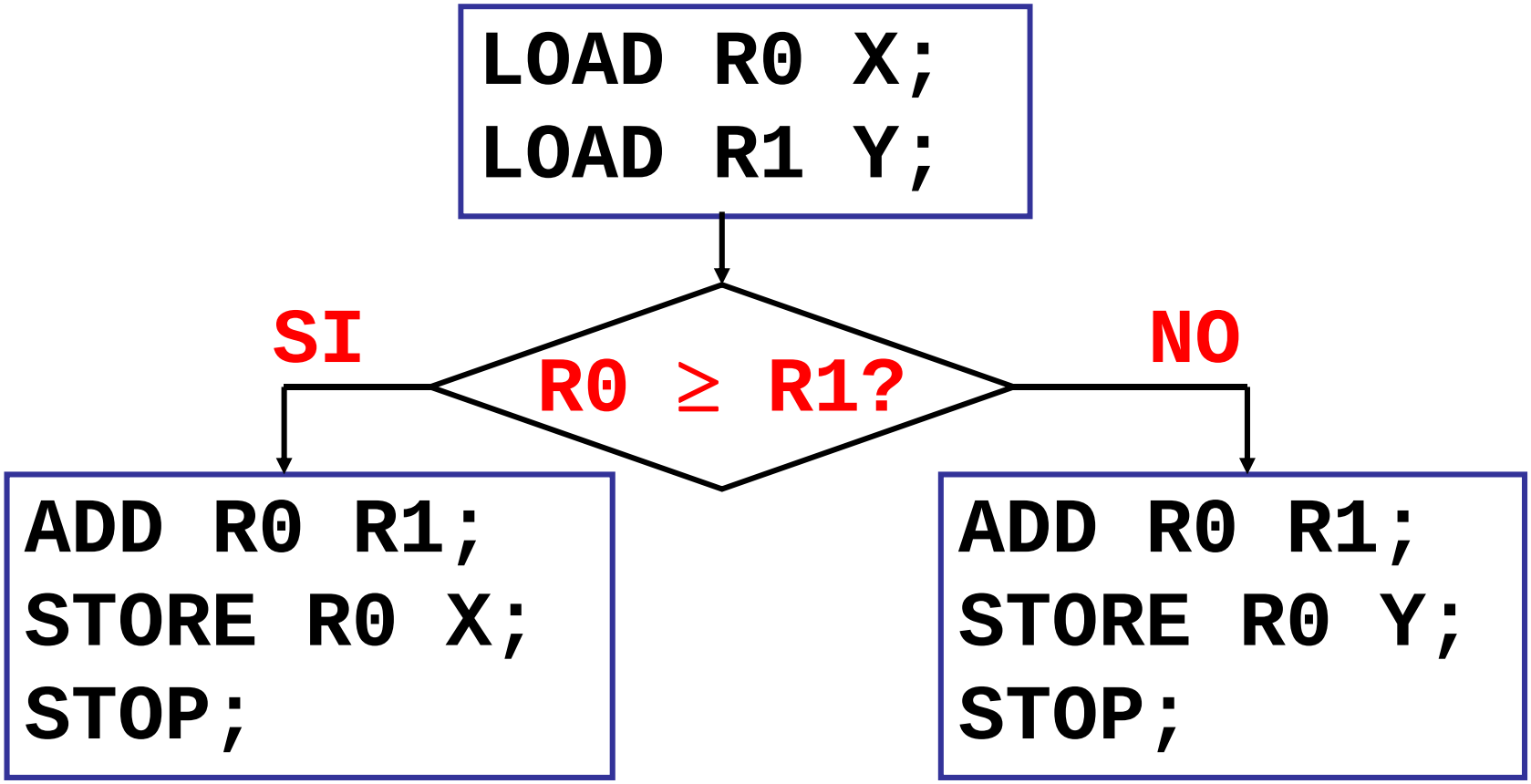


istruzioni

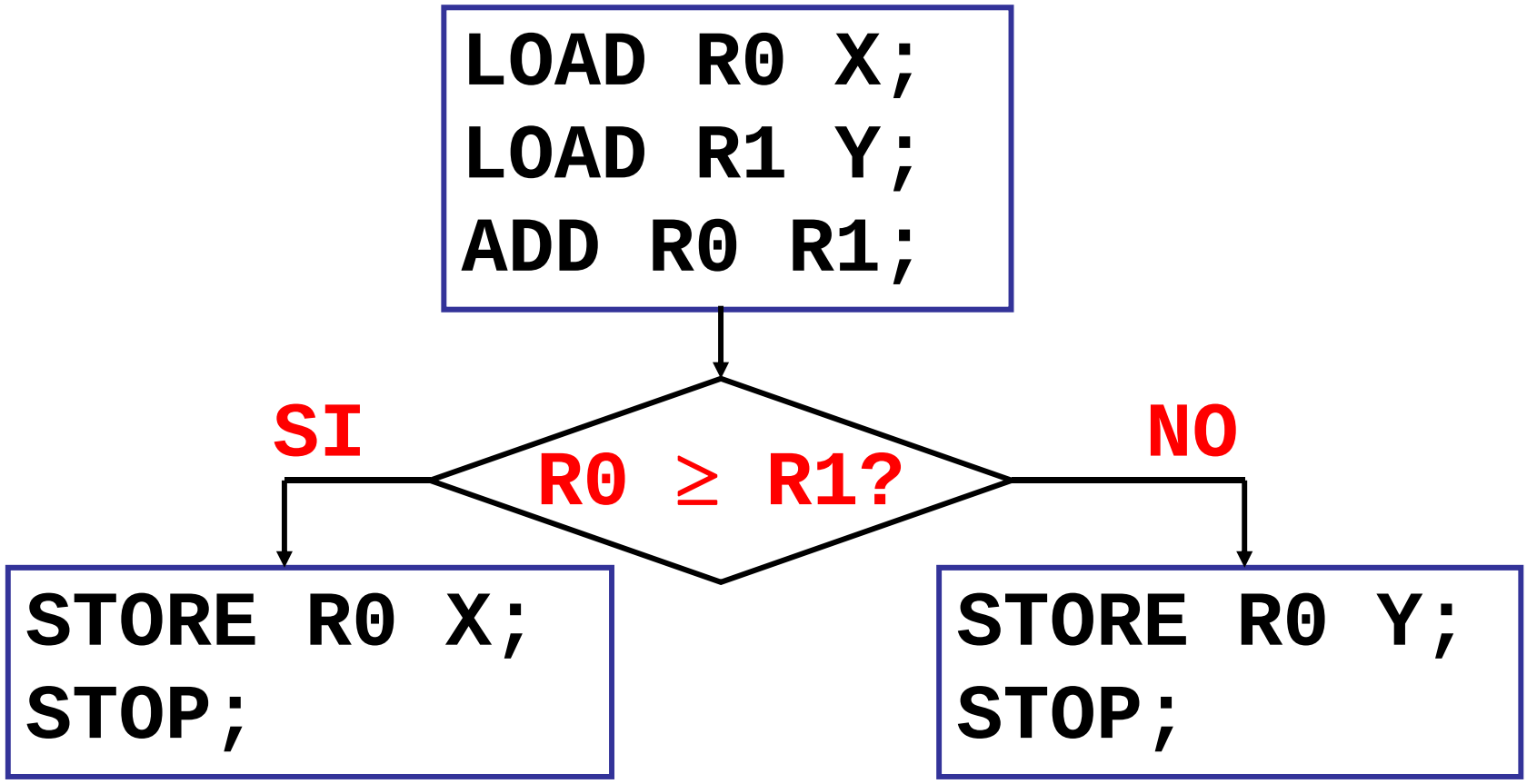
Altro esempio

Carica due valori dalla RAM, li somma e memorizza il risultato al posto del maggiore dei 2 numeri sommati (nel caso siano uguali, in uno qualsiasi dei 2 indirizzi RAM)

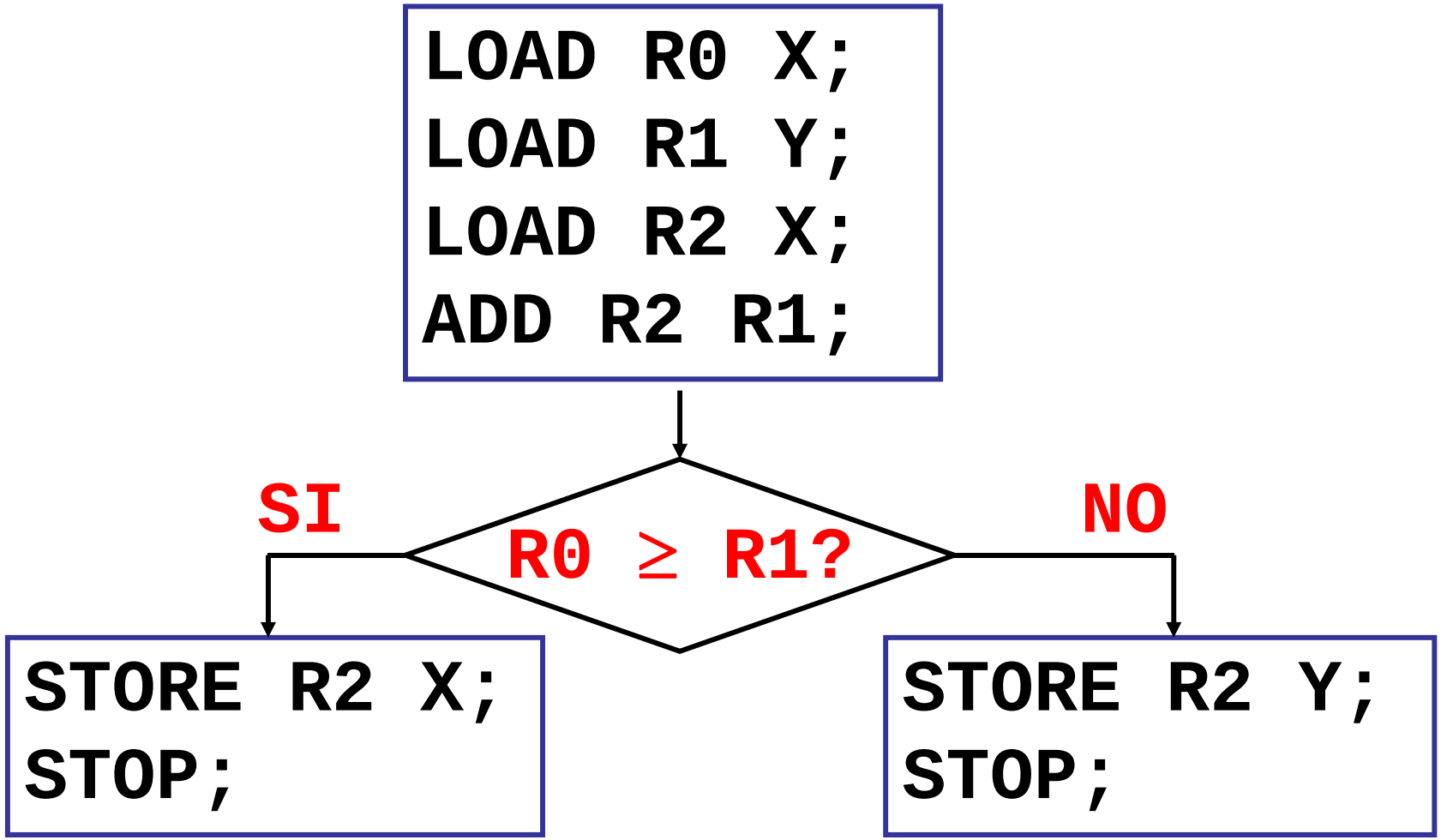
Algoritmo 1


LOAD R0 X;
LOAD R1 Y;
SI
R0 ≥ R1?
NO
ADD R0 R1;
STORE R0 X;
STOP;
ADD R0 R1;
STORE R0 Y;
STOP;

Questo algoritmo funziona come il precedente?



Algoritmo 2


LOAD R0 X;
LOAD R1 Y;
LOAD R2 X;
ADD R2 R1;
R0 ≥ R1?
SI
NO
STORE R2 X;
STOP;
STORE R2 Y;
STOP;

Codice assembler algoritmo 1

```
X: INT 38;
Y: INT 8;
     LOAD R0 X;
     LOAD R1 Y;
     COMP R0 R1;
     BRGE maggiore;
     ADD R0 R1;
     STORE R0 Y;
     STOP;
maggiore:
     ADD R0 R1;
     STORE R0 X;
     STOP;
```

Codice assembler algoritmo 2

```
X: INT 38;
Y: INT 8;
    LOAD R0 X;
    LOAD R1 Y;
    LOAD R2 X;
    ADD R2 R1;
    COMP R0 R1;
    BRGE maggiore;
    STORE R2 Y;
    STOP;
maggiore:
    STORE R2 X;
    STOP;
```

Conclusioni

3 costrutti fondamentali:

1) X ← valore **ASSEGNAZIONE**

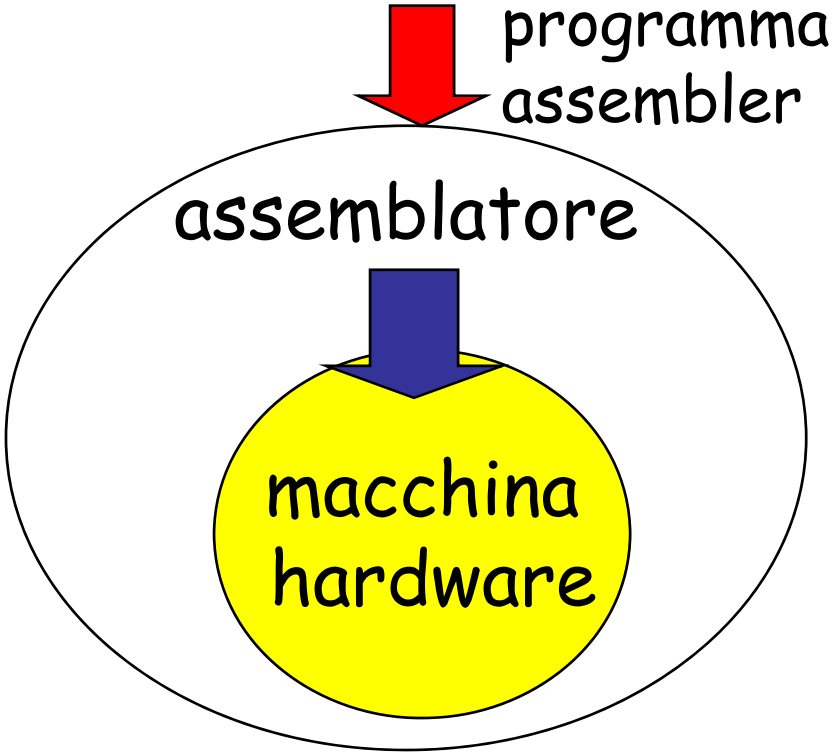
2) TEST di una condizione che determina quale azione intraprendere successivamente

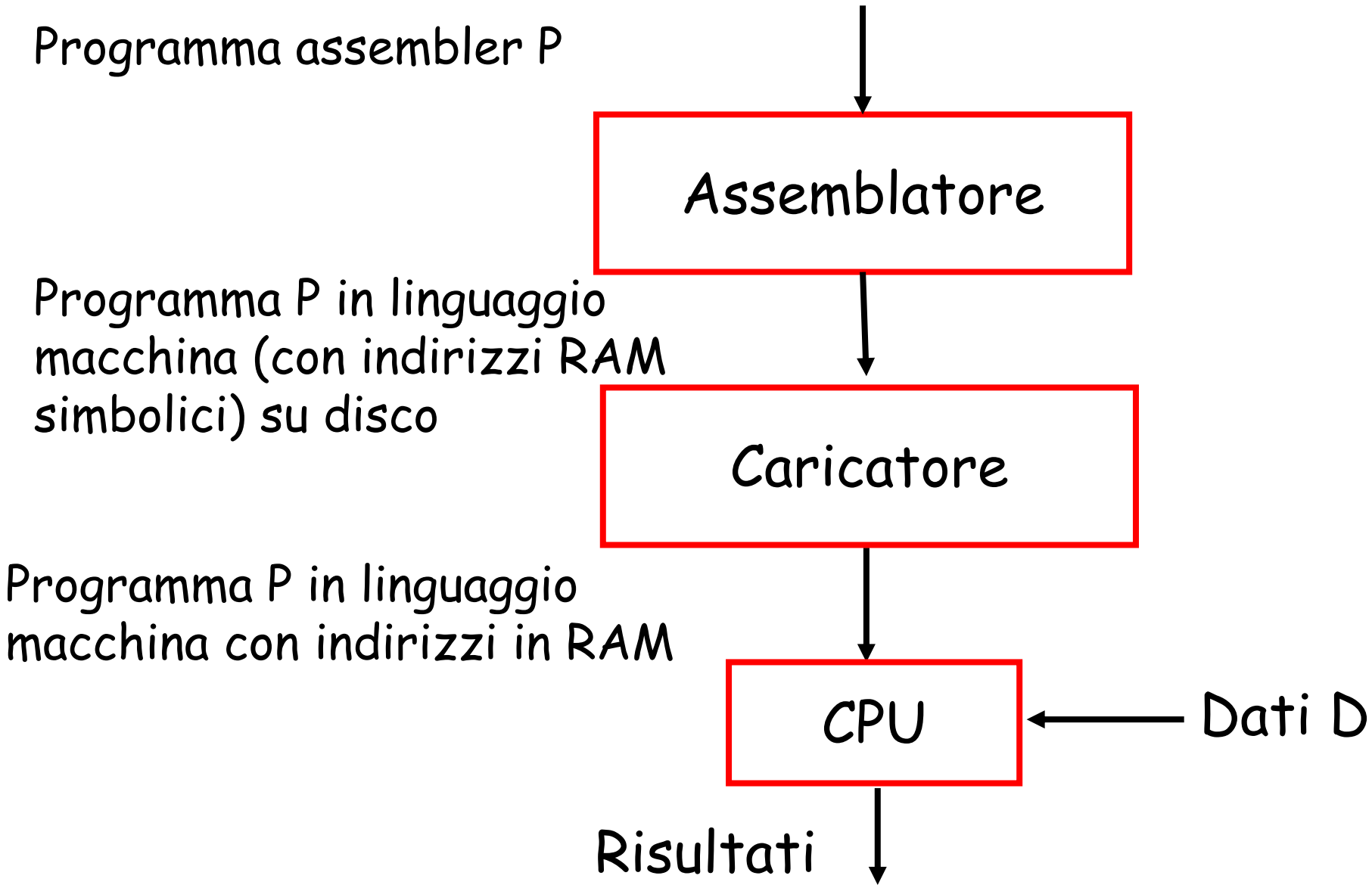
3) CICLO che si ripete finchè una data condizione è vera

SONO COSTRUTTI FONDAMENTALI in ogni LINGUAGGIO di PROGRAMMAZIONE

La CPU non "capisce" l'assembler !!

Il programma assembler deve essere tradotto in un programma macchina




Programma assembler P
Assemblatore
Programma P in linguaggio macchina (con indirizzi RAM simbolici) su disco
Caricatore
Programma P in linguaggio macchina con indirizzi in RAM
CPU
Dati D
Risultati