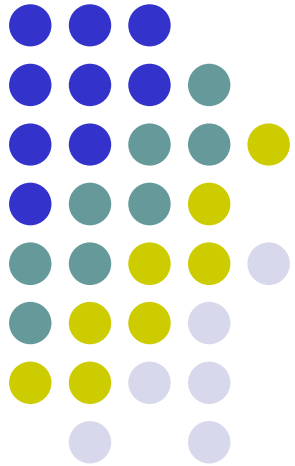
Laboratorio di Calcolatori 1

Corso di Laurea in Fisica

A.A. 2006/2007



Dott.Davide Di Ruscio

Dipartimento di Informatica

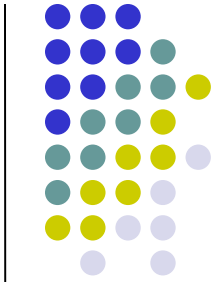
Università degli Studi di L'Aquila

Lezione del 29/01/07

Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele

Prova intermedia

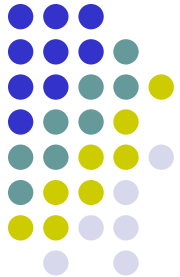


- Lunedì 12 Febbraio 2007
  - 11:30-13:30 A.M.

- Gli esercizi di preparazione alla prova sono disponibili on line all'indirizzo:

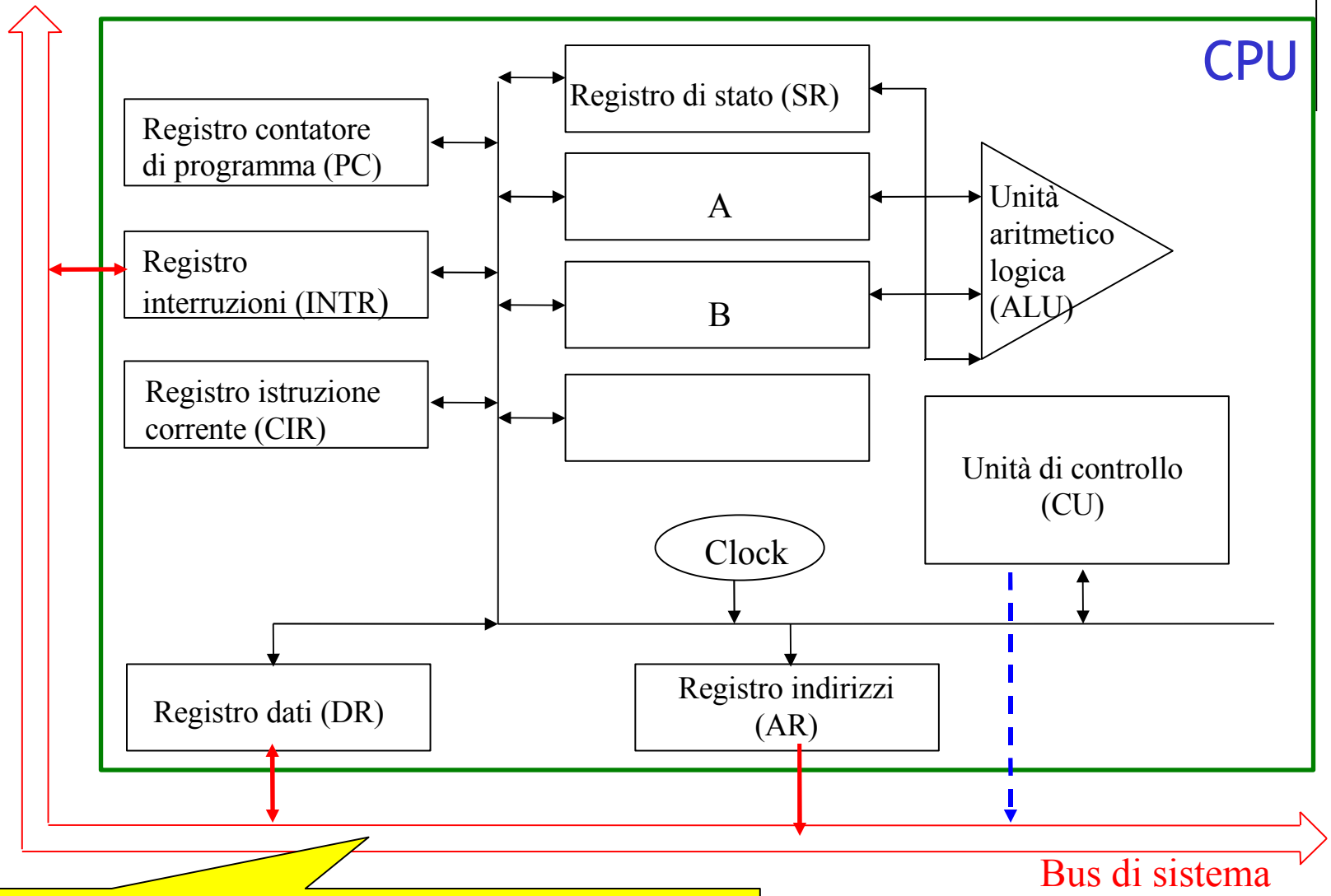
www.di.univaq.it/diruscio/labcalc1/esercizi.html

Sommario (I parte)

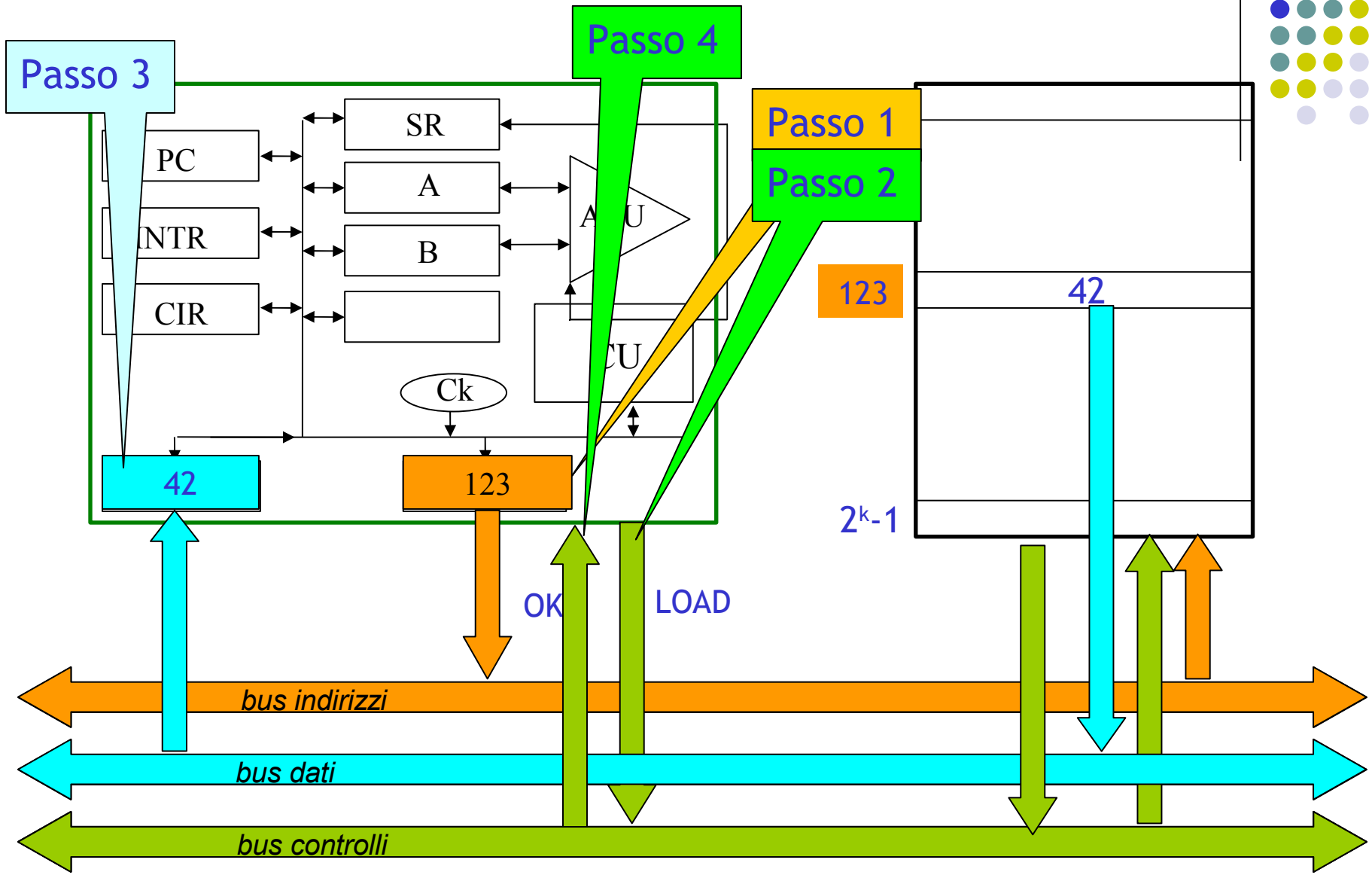


- Concetti fondamentali
- Aspetti architetturali di un sistema di calcolo
  - hardware
  - software
    - software di base
    - software applicativo
- Codifica dell'informazione
  - numeri naturali, interi, reali
  - caratteri
  - immagini
- Macchina di Von Neumann
  - CPU (UC, ALU, registri, clock)
  - memoria centrale
  - bus di bistema
  - periferiche
- Linguaggio macchina
- Linguaggio assembler
- Sistema operativo
- Ambiente di programmazione

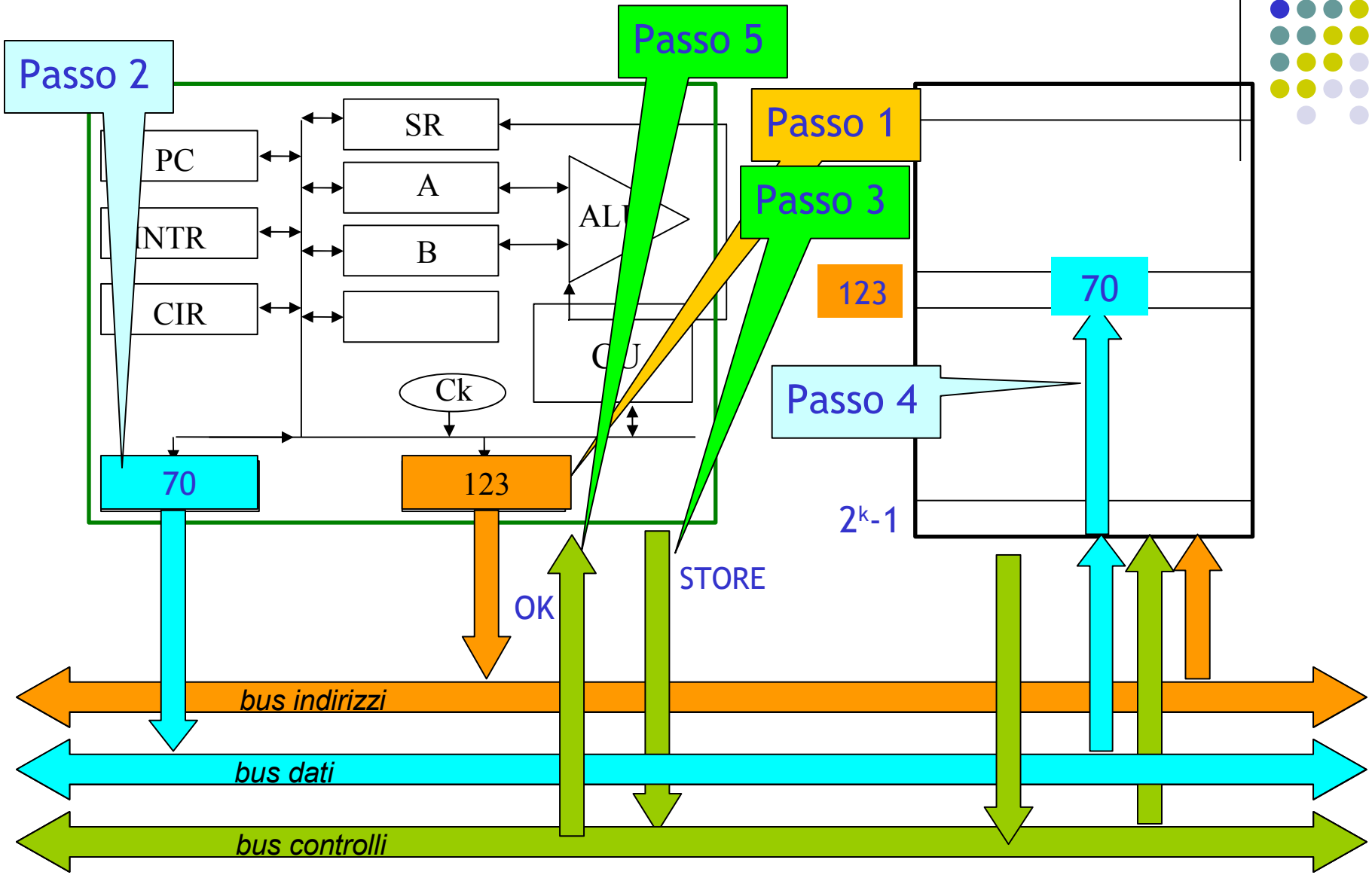
Il bus di sistema



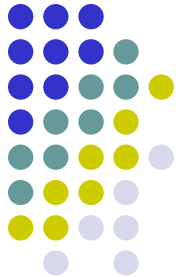
Sequenza di lettura in MM (load)



Sequenza di scrittura in MM (store)



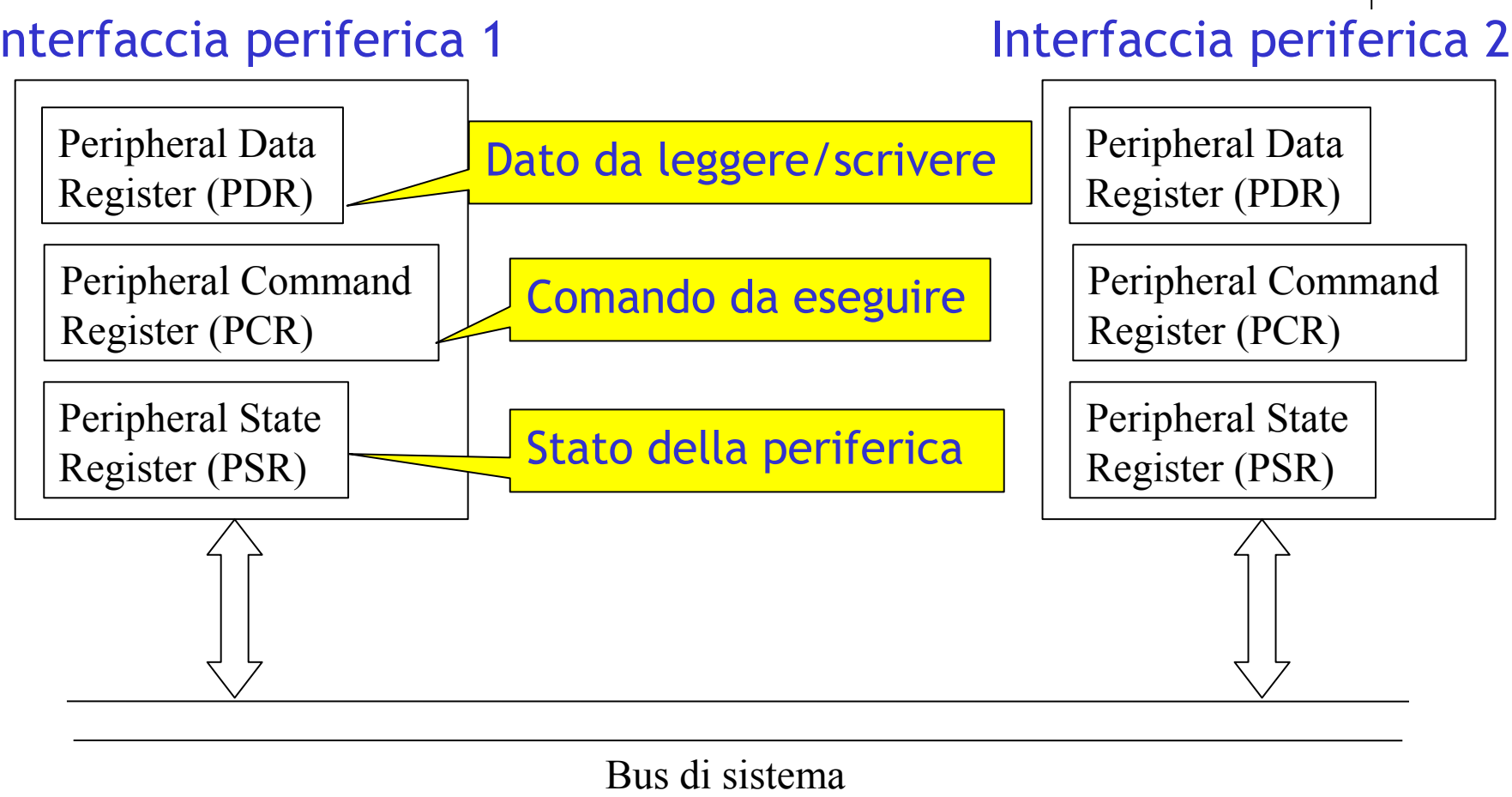
- REGISTRI: dispositivi elettronici capaci di memorizzare sequenze di bit fungendo da piccole memorie interne alla C.P.U.
  - **DR** (registro dati): in esso transitano le parole da leggere (load) o scrivere (store) nella memoria centrale
  - **AR** (registro indirizzi): contiene l'indirizzo della cella di memoria coinvolta nell'operazione di load o store
  - **CIR** (registro istruzione corrente): contiene l'istruzione correntemente in esecuzione
  - **PC** (registro contatore di programma): contiene l'indirizzo della cella di memoria in cui si trova la prossima istruzione da eseguire
  - **INTR** (registro interruzioni): per la gestione di eventi asincroni, contiene istruzioni circa lo stato di funzionamento delle periferiche



- **A,B**: memorizzano gli operandi e successivamente i risultati delle operazioni svolte dall'ALU. In particolare,
  - in precedenza essi vengono caricati con i valori dei due operandi
  - dopo l'esecuzione di un'operazione il registro A è caricato con il risultato dell'operazione.
  - nel caso della divisione intera, il resto della divisione è caricato nel registro B, mentre per le altre operazioni il contenuto del registro B non è definito
- **SR** (registro di stato): contiene informazioni circa l'esito delle operazioni svolte dall'ALU, ed in particolare i bit di carry (presenza di riporto), zero (presenza di valore nullo in A), segno (segno del risultato del'operazione svolta dall'ALU), overflow (per rilevare eventuali condizioni di overflow)
- …

…

Le interfacce delle periferiche



Le interfacce delle periferiche

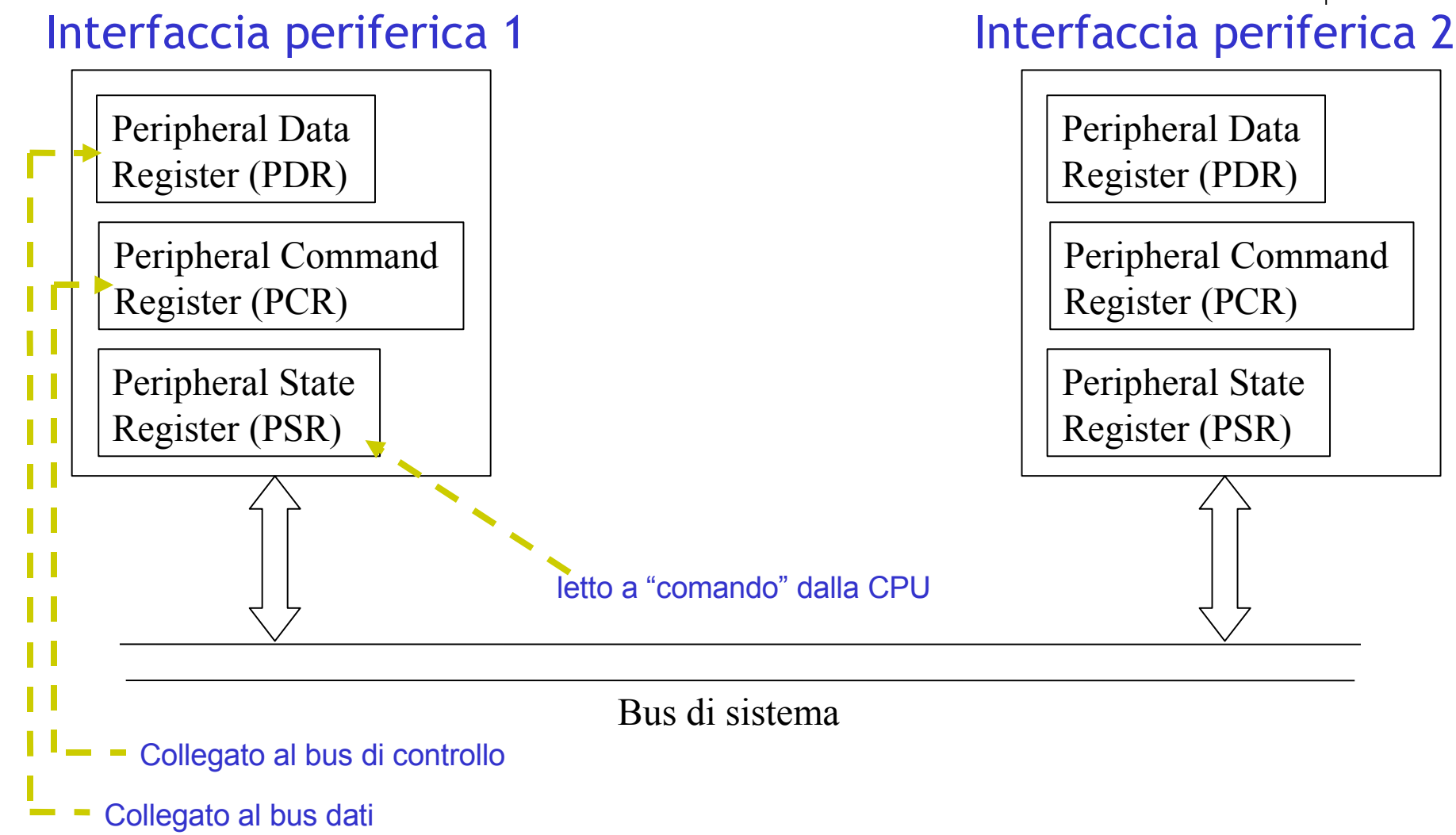
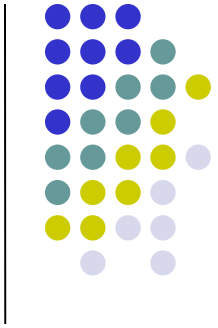


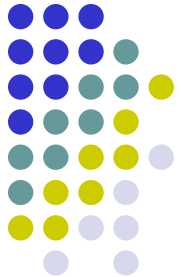
Tabella o elenco delle istruzioni



lettura in memoria	LOADA	0000
	LOADB	0001
scrittura in memoria	STOREA	0010
	STOREB	0011
lettura dalla periferica	READ	0100
scrittura sulla periferica	WRITE	0101
operazioni aritmetiche	ADD	0110
	DIF	0111
	MUL	1000
	DIV	1001
salto a branch	JUMP	1010
	JUMPZ	1011
nessuna operazione	NOP	1100
terminazione	HALT	1101

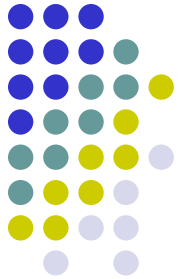
Dato che il linguaggio comprende *14* istruzioni diverse, per il codice operativo sono necessari $m \geq 4$ bit

L'istruzione *read*



- *read ind*: acquisisce un valore dalla periferica di input (ad es. una tastiera) e lo inserisce nella cella di memoria di indirizzo *ind*

- Viene eseguita dalle seguenti micro-istruzioni:
  - Il contenuto del registro PDR dell'interfaccia della periferica viene trasferito tramite il bus in DR
  - L'operando in OP(CIR) viene trasferito in AR
  - Viene eseguita una scrittura in memoria centrale, ossia Il contenuto di DR viene memorizzato nella cella di memoria il cui indirizzo è specificato in AR, ossia in MEM[AR]

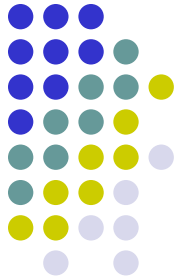


In sintesi, viene eseguita dalle seguenti tre micro-istruzioni :

- PDR→DR
- OP(CIR)→AR
- DR→MEM[AR]

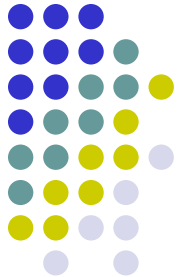
N.B. Per poter effettuare l'operazione, il registro PDR deve contenere il dato letto dalla periferica; questa condizione può essere verificata utilizzando il registro di stato PSR della periferica o tramite il registro RINT dell'unità centrale

L'istruzione *write*



- *Write ind*: stampa il contenuto della cella di memoria di indirizzo *ind* sulla periferica di output

- Viene eseguita dalle seguenti micro-istruzioni:
  - Il contenuto in OP(CIR) viene trasferito in AR
  - Il contenuto della cella il cui indirizzo è specificato da AR viene trasferito in DR
  - Il contenuto di DR viene trasferito tramite il bus nel registro PDR presente nell'interfaccia della periferica

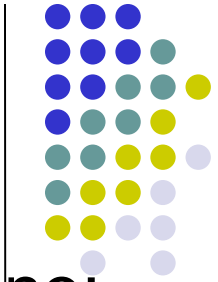


In sintesi, viene eseguita dalle seguenti tre micro-istruzioni:

- OP(CIR)→AR
- MEM[AR] →DR
- DR→PDR

N.B. Per poter effettuare l'operazione, la periferica di output deve essere pronta a ricevere il dato da stampare nel registro PDR; come per la read, questa condizione può essere verificata sfruttando il registro di stato PSR della periferica o tramite il registro RINT dell'unità centrale

Le istruzioni aritmetche

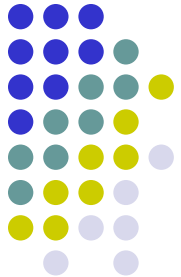


- Le operazioni numeriche effettuate dalla ALU sono:
  - *add* - somma
  - *dif* - differenza
  - *mul* - moltiplicazione
  - *div* - divisione

- L'effetto di ciascuna è quello di effettuare l'operazione corrispondente prendendo come operandi i contenuti dei registri A e B, e di memorizzare quindi il risultato nel registro A, ricoprendo il valore precedente

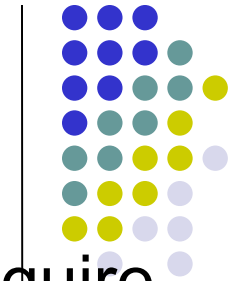
- L'unica eccezione si ha nel caso di divisione, in cui il quoziente viene memorizzato in A ed il resto in B

Esempio: somma di due interi



LOADA 16	(carico il registro A con il contenuto della cella 16)
LOADB 17	(carico il registro B con il contenuto della cella 17)
ADD	(faccio la somma, il risultato viene messo nel registro A)

Le istruzioni di salto (*jump)*



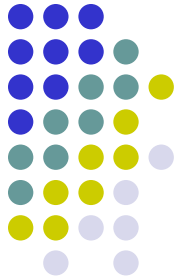
- Le *istruzioni di salto* consentono di far proseguire l'esecuzione non dall'istruzione contenuta nella cella di memoria successiva, bensì da quella il cui indirizzo corrispondente è specificato dall'operando

- Possono essere fondamentalmente di due tipi:

  - Salto incondizionato - *jump ind:* fa proseguire l'esecuzione del programma dall'istruzione contenuta nella locazione di memoria di indirizzo *ind*

  - Salto condizionato - *jumpz ind:* fa avvenire il salto del programma all'indirizzo *ind* solo se il contenuto del registro A è nullo

L'istruzione *jump* (salto incondizionato)



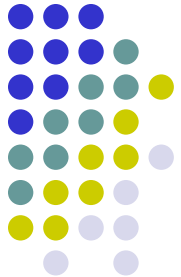
- *jump ind:* fa proseguire l'esecuzione del programma dall'istruzione contenuta nella locazione di memoria di indirizzo *ind*
- Viene eseguita dalla sola micro-istruzione
  - OP(CIR)→PC
- Essa ha l'effetto di predisporre la fase di fetch seguente in modo che prelevi l'istruzione da eseguire dall'indirizzo specificato nell'operando

Esempio

100 LOADA 16
101 LOADB 17
102 ADD
103 JUMP 100

JUMP100 fa tornare ad eseguire l'istruzione nell'indirizzo 100

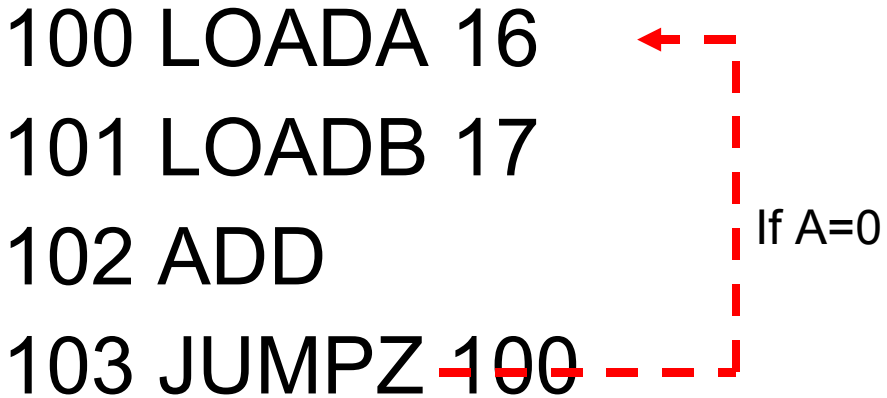
L'istruzione *jumpz* (salto condizionato)



- *jumpz ind:* fa proseguire l'esecuzione del programma dall'istruzione contenuta nella locazione di memoria di indirizzo *ind* solo se il contenuto del registro A è nullo

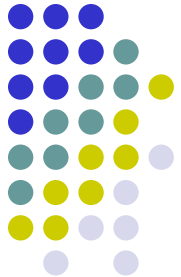
- Viene eseguita dalla sola micro-istruzione
  - If A=0 OP(CIR)→PC

Esempio



JUMPZ 100 fa tornare ad eseguire l'istruzione nell'indirizzo 100 solo se il valore di A dopo l'esecuzione dell'istruzione ADD è uguale a 0

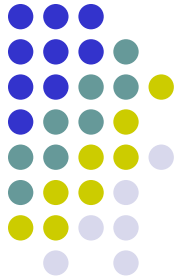
Le istruzioni di pausa e arresto



- *nop*: ha la fase di execute vuota, ossia ha l'effetto di far passare la fase senza che venga effettuata alcuna operazione; in pratica viene utilizzata per introdurre temporizzazioni o ritardi, ad esempio per una visualizzazione animata dell'output

- *halt:* è l'istruzione di arresto ed ha come effetto la terminazione dell'esecuzione del programma

Esempio: prodotto di interi

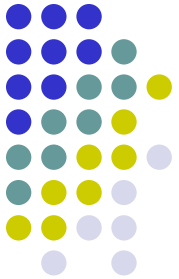


Questo semplice programma riportato in un linguaggio simbolico acquisisce due numeri interi dal terminale e stampa sul video il valore del loro prodotto.

Le istruzioni del programma occupano le prime 8 celle di memoria.

Le celle 8, 9 e 10 sono predisposte per contenere dati di tipo intero

```
 0    READ     8
 1    READ     9
 2    LOADA    8
 3    LOADB    9
 4    MUL
 5    STOREA  10
 6    WRITE   10
 7    HALT
 8    INT
 9    INT
10    INT
```

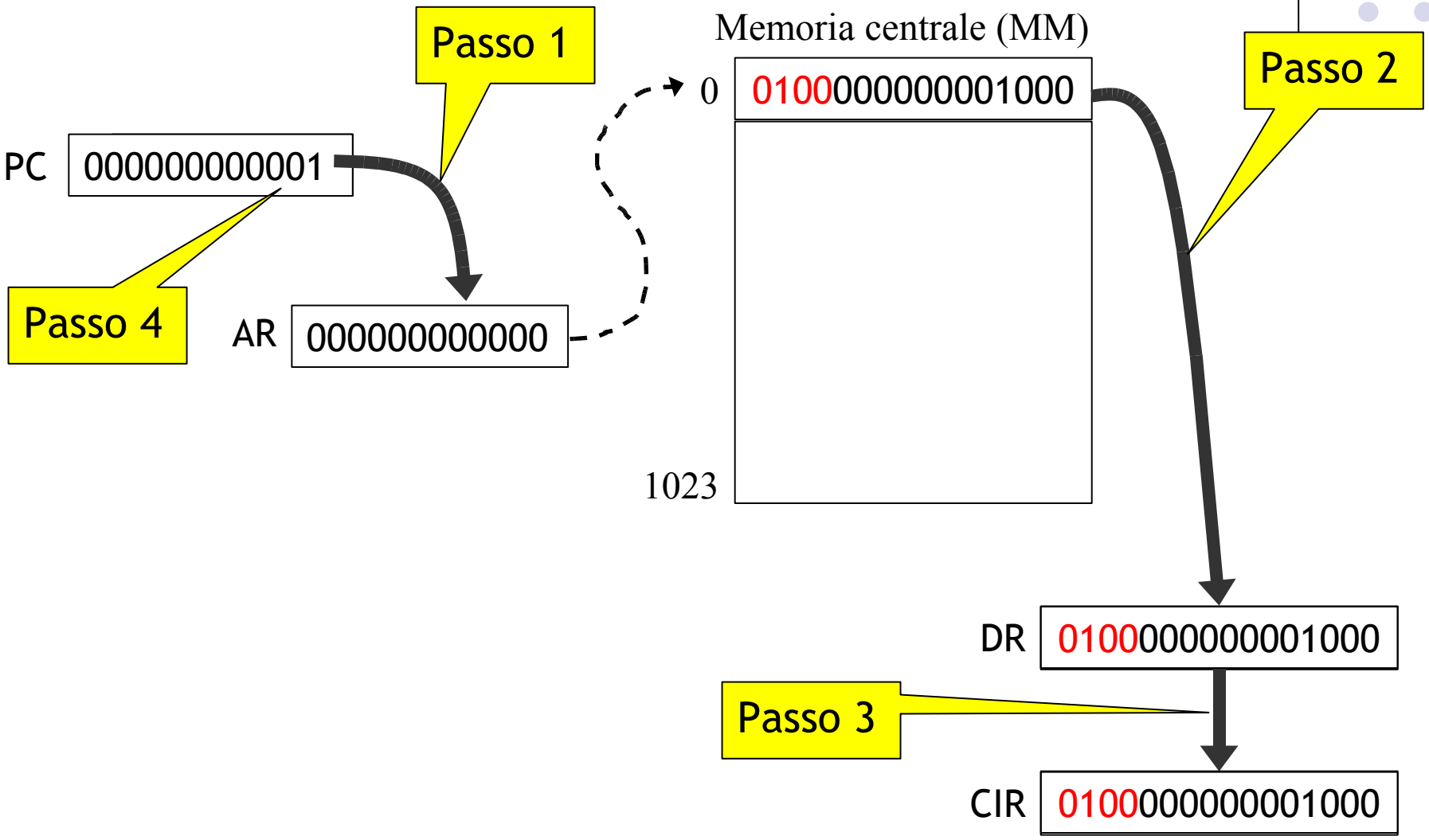


- Nella forma definitiva, le istruzioni devono comunque essere espresse in formato binario:

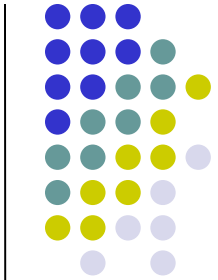
Numero cella di memoria	Contenuto della cella di memoria
0	0100000000001000
1	0100000000001001
2	0000000000001000
3	0001000000001001
4	1000000000000000
5	0010000000001010
6	0101000000001010
7	1101000000000000
8	…
9	…
10	…

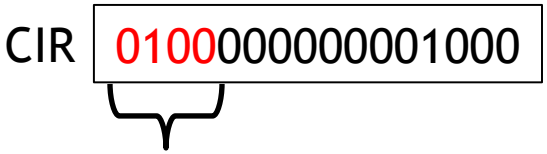
- Leggere, comprendere, correggere e modificare il significato di un programma presenta non poche difficoltà...

Fase di fetch 1ª istruzione



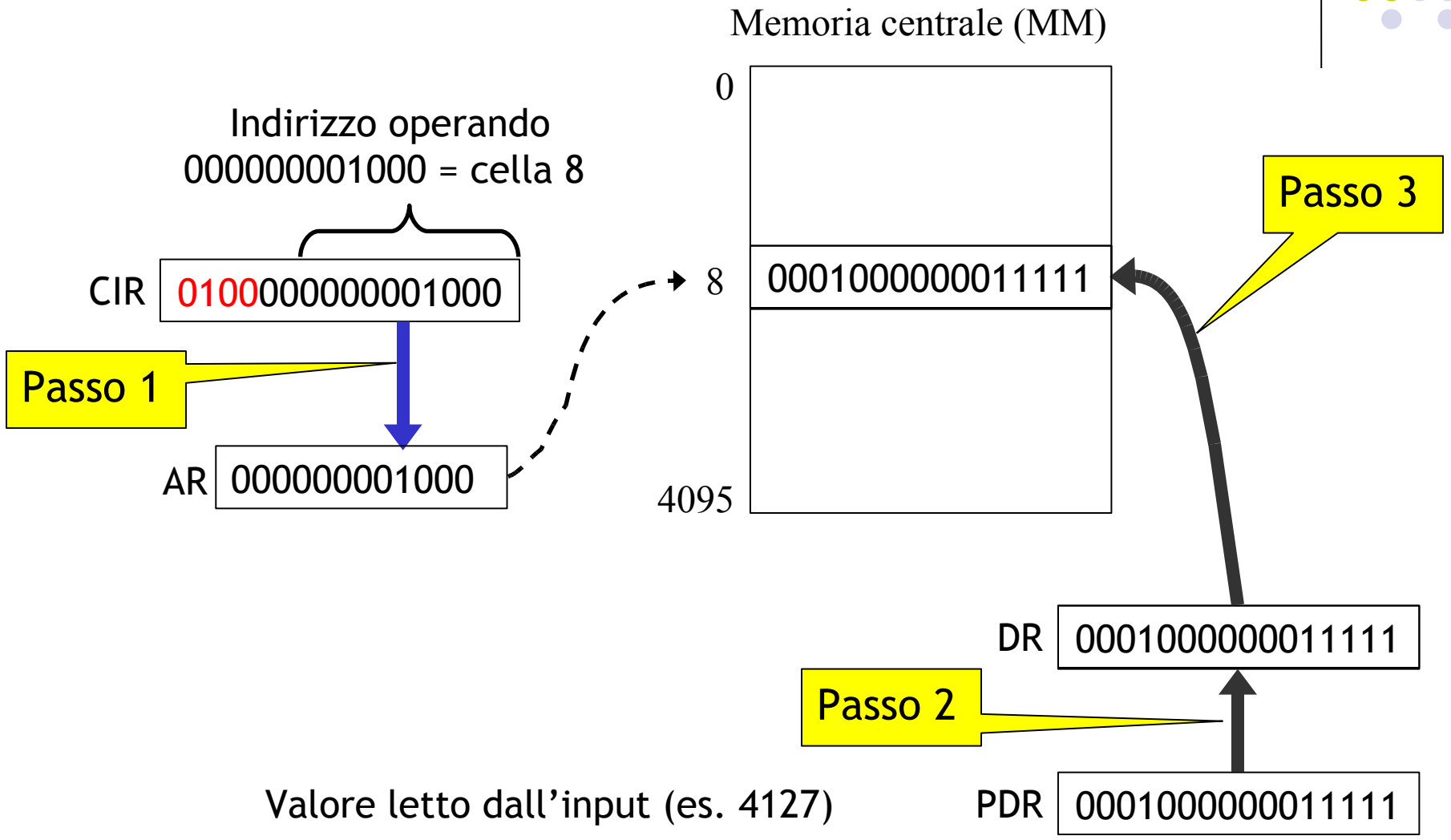
Fase di interpretazione 1ª istruzione



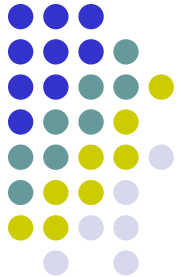


Codice operativo 0100 = read, ossia leggi da input

Fase di esecuzione 1ª istruzione

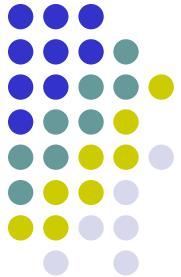


Svantaggi del linguaggio macchina

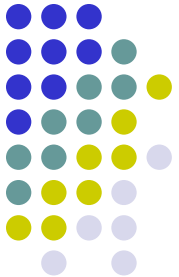


- I programmi in binario sono difficili da scrivere, capire e modificare
- Il programmatore deve occuparsi di gestire la RAM: difficile ed inefficiente

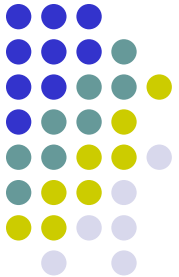
Il linguaggio assemblatore



- Per evitare questo problema sono stati inventati linguaggi a più *alto livello*
- Il programmatore potrà quindi concentrarsi sulla stesura di programmi comprensibili, poiché scritti in un linguaggio più simbolico e vicino al suo modo di esprimersi
- La traduzione inversa, da un programma scritto in un linguaggio simbolico o sorgente all'equivalente in linguaggio macchina è un procedimento meccanico e può essere delegato ad un altro programma, scritto in linguaggio macchina (assemblatore, compilatore, interprete, ….)
- Tanto maggiore è l'astrazione, ossia la distanza dal linguaggio macchina, tanto maggiore sarà la complessità del programma che provvederà a tradurre il programma sorgente nel linguaggio macchina.



- Il primo passo di astrazione di permette di ottenere un linguaggio che si avvicina al linguaggio macchina e non aggiunge potenza descrittiva
- Questo linguaggio è chiamato *Assembler* e in generale ha le seguenti caratteristiche
  - Le istruzioni corrispondono in maniera biunivoca alle istruzioni di un linguaggio macchina
  - Vengono utilizzati nomi simbolici per codificare le istruzioni

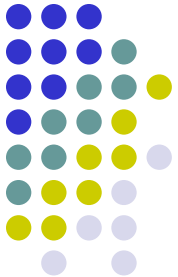


Più precisamente si utilizzano nomi simbolici per:

- I codici operativi, dando luogo ai cosiddetti codici mnemonici (loada, storea, read, …)
- Indirizzi di memoria, dando luogo a
  - Etichette, per celle di memoria contenenti istruzioni
  - Variabili, per celle di memoria contenenti dati

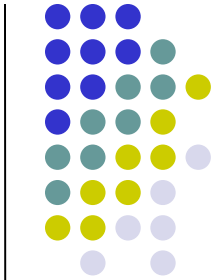
- Chiaramente tutti i nomi simbolici (codici mnemonici, etichette e variabili) devono essere distinti
- Il programma che effettua la traduzione da linguaggio assembler a linguaggio macchina si chiama *Assemblatore*

Esempio: prodotto di interi



I nomi simbolici
X e Y corrispondono
a due variabili contenenti
numeri interi

```
        READ    X
        READ    Y
        LOADA   X
        LOADB   Y
        MUL
        STOREA  Z
        WRITE   Z
        HALT
X       INT
Y       INT
Z       INT
```

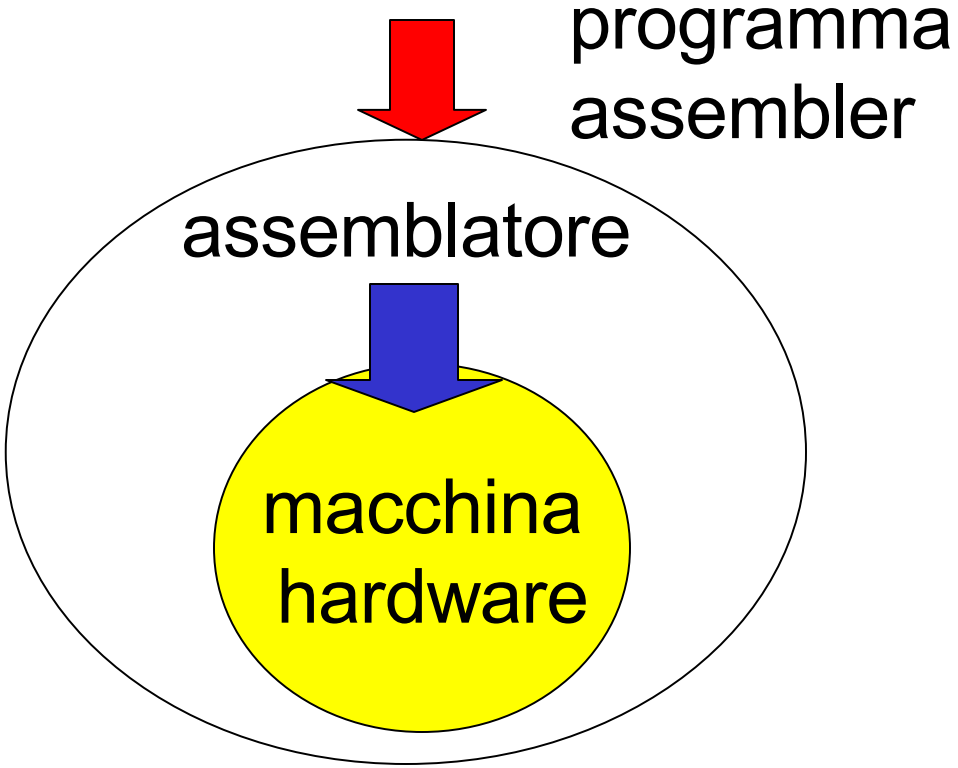


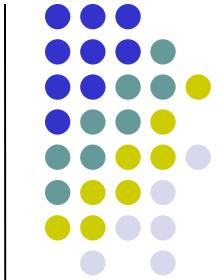
Quindi:

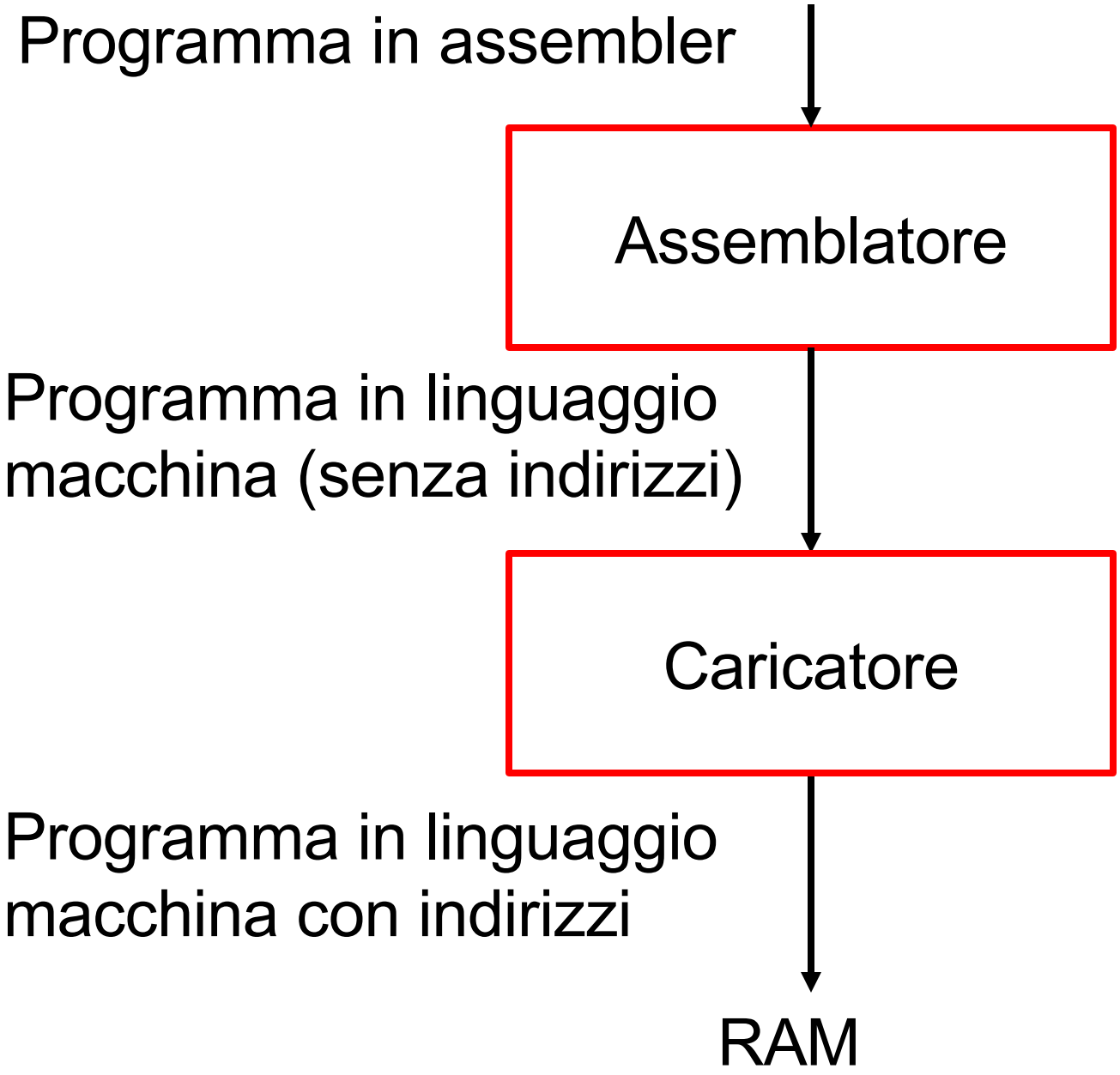
- codici mnemonici per le operazioni
- nomi mnemonici (identificatori) al posto degli indirizzi RAM per i dati (e indirizzi RAM delle istruzioni usate nei salti)
- avanzate: tipi dei dati **INT** e **FLOAT**

La CPU non “capisce” l’assembler

Il programma assembler deve essere tradotto in un programma macchina






Programma in assembler
Assemblatore
Programma in linguaggio macchina (senza indirizzi)
Caricatore
Programma in linguaggio macchina con indirizzi
RAM