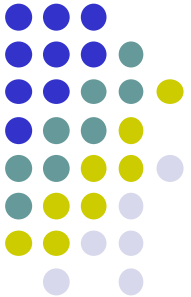
MODELLO STACK

R1: 20; R2: 50; R3: 10

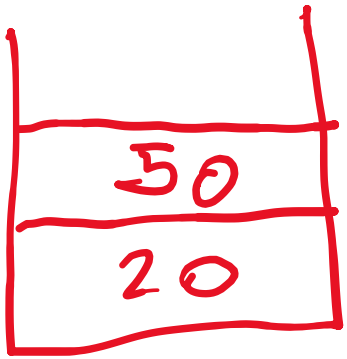
```
PUSH R1
PUSH R2
MUL
PUSH R3
POP R2
POP R3
PUSH R1
```

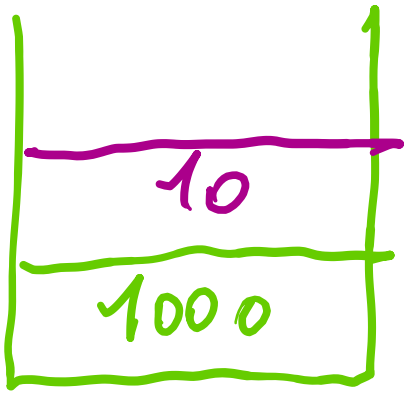


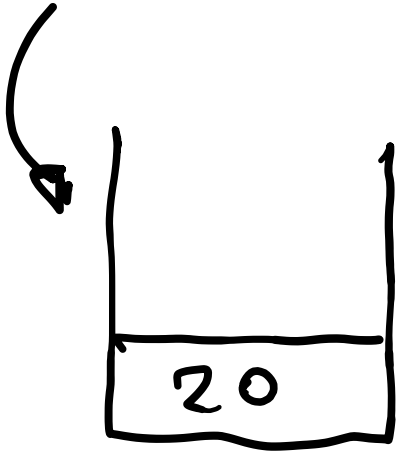
① Contenuto dello STACK?

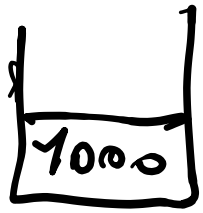
② i valori dei registri?

```
PUSH R1
PUSH R2
MUL
PUSH R3
POP R2
POP R3
PUSH R1
```





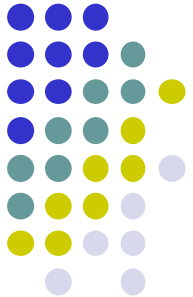




R2: 10
R3: 1000

R1: 20

Esecuzione e sequenzializzazione delle istruzioni

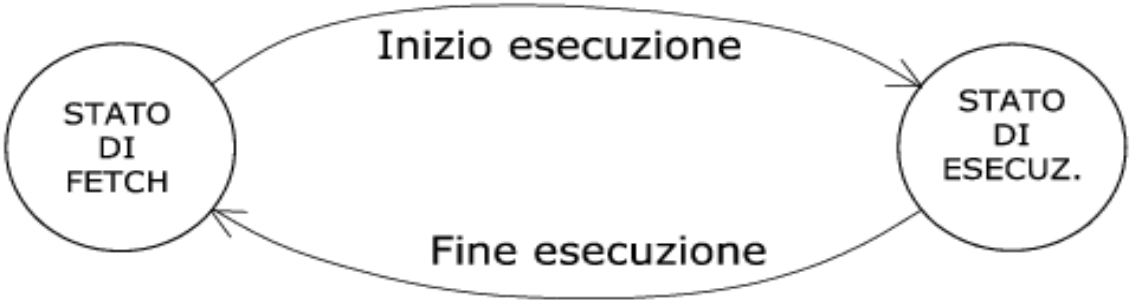


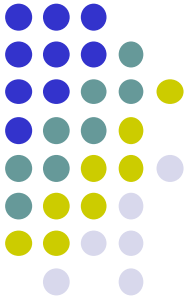
- Il normale funzionamento dell'elaboratore consiste nell'esecuzione in sequenza delle istruzioni dei programmi in linguaggio macchina

- L'esecuzione di una singola istruzione, prevede due fasi fondamentali:
  - Fase di Fetch: acquisizione dell'istruzione da eseguire dalla memoria centrale
  - Fase di Execute: interpretazione ed esecuzione dell'istruzione stessa

- Tutte le istruzioni hanno una fase di fetch identica e si distinguono solo per la fase di execute

- Le fasi si alternano secondo il seguente diagramma





- Per eseguire le fasi di fetch ed execute la CPU si serve di due registri appositamente predisposti:
  - ***IR*** (Instruction Register):
    - usato per contenere l'istruzione in corso di esecuzione.
    - caricato in fase di fetch
    - rappresenta l'ingresso che determina le azioni svolte durante la fase di esecuzione.
  - ***PC*** (Program Counter):
    - tiene traccia dell'esecuzione del programma
    - contiene di volta in volta l'indirizzo della cella di memoria in cui si trova la prossima istruzione da eseguire
    - viene incrementato (di *4*) durante la fase di fetch per predisporre la fase di fetch dell'istruzione successiva

- Vediamo un esempio relativo all'esecuzione del segmento di codice corrispondente all'istruzione *a=b+c*

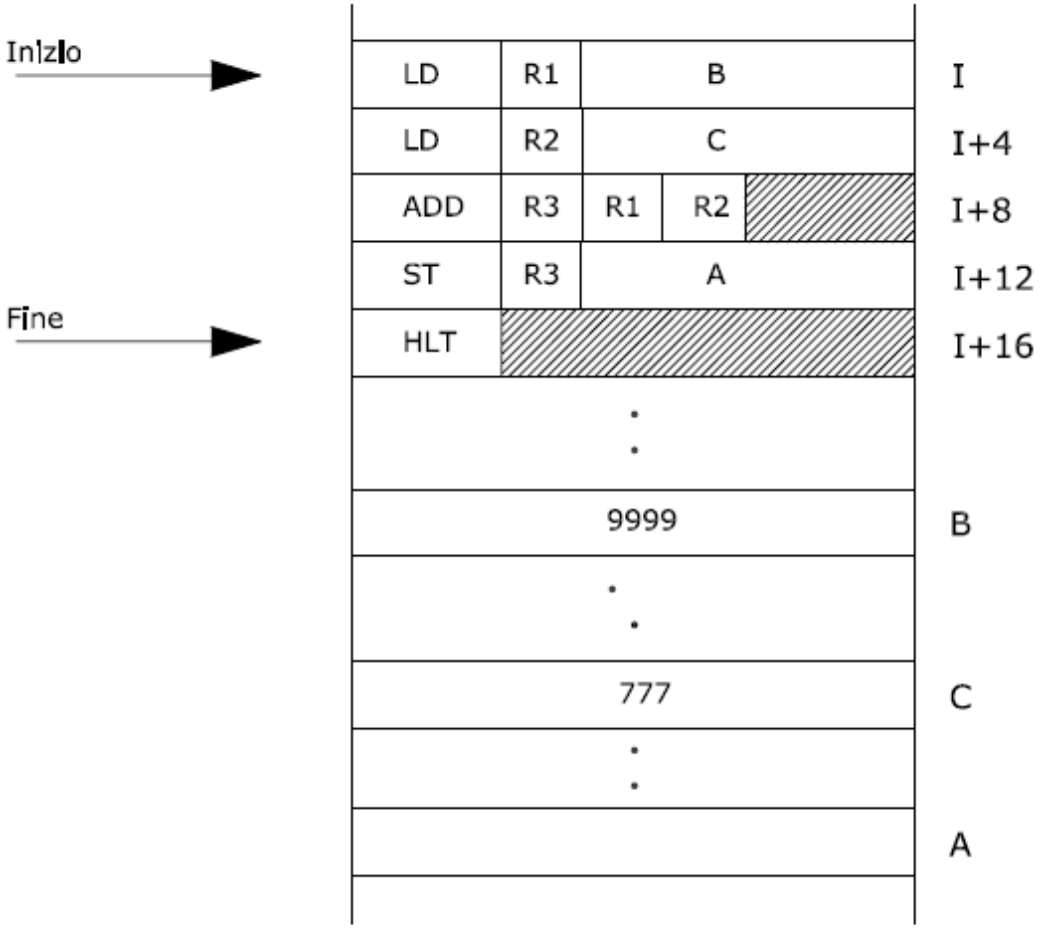
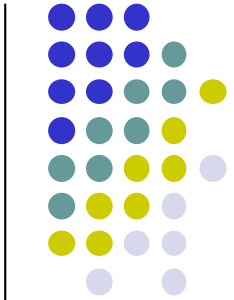


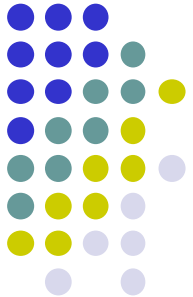
Figura 5.4 Mappa della memoria con le tre variabili e la sequenza di istruzioni equivalenti allo statement C "`a = b+c`" per il caso della seconda soluzione. Si è fatto l'ipotesi che la variabile `b` valga 9999 e che la variabile `c` valga 777. Dopo l'esecuzione del tratto di codice, nella posizione `A` si trova il numero 10776, qualunque fosse il contenuto precedente della cella, mentre il contenuto delle altre due è immodificato.

Il controllo del flusso



- L'istruzione *HLT* in coda al segmento di programma precedente è un primo esempio intervento sul flusso di esecuzione o controllo del programma
- Serve a bloccare l'avanzamento impedendo che il *PC* vada oltre l'ultima istruzione utile
- In generale per poter considerare le varie alternative che possono verificarsi durante l'elaborazione, il repertorio deve includere istruzioni che consentato di trasferire il controllo
- Tali istruzioni vengono chiamate di *salto* o *diramazione*, perché fanno sì che la prossima istruzione eseguita sia quella specificata, e non quella direttamente successiva (in memoria) alla istruzione correntemente in esecuzione

Salto incondizionato

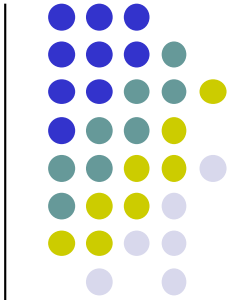


JMP DEST

- Ha l'effetto di trasferire il controllo incondizionatamente all'indirizzo specificato
- In altre parole, la prossima istruzione eseguita sarà quella contenuta nella locazione di indirizzo *DEST*
- In forma compatta l'effetto di tale istruzione può essere descritto con

PC ← *DEST*

Salto condizionato



JE Ra,Rb DEST

- Ha l'effetto di trasferire il controllo all'indirizzo specificato se i contenuti dei registri *Ra* e *Rb* coincidono
- In forma compatta l'effetto di tale istruzione può essere descritto con

  if (Ra=Rb) then PC ← DEST

- Vediamo i possibili formati delle istruzioni di salto

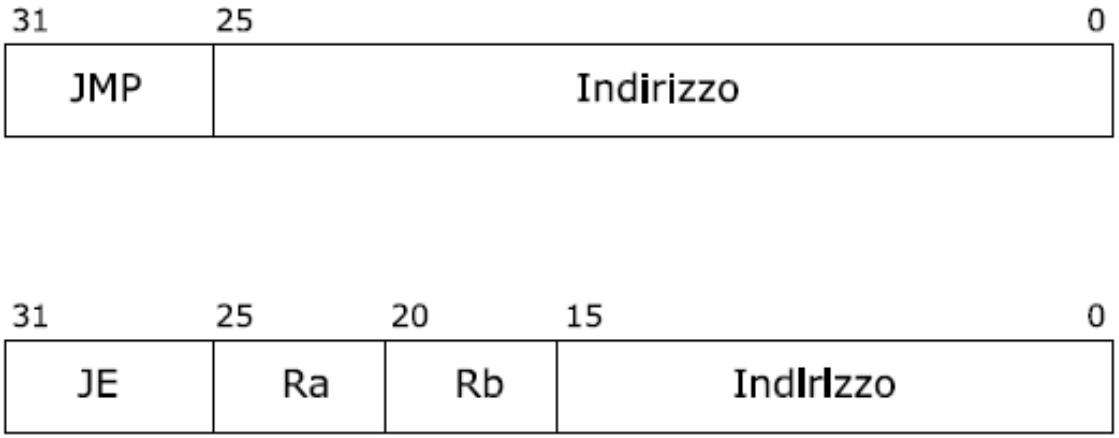
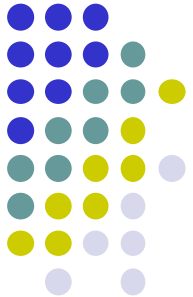
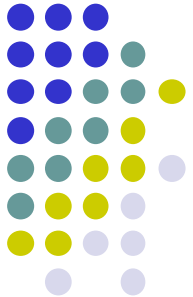


Figura 5.5 Possibili formati per le istruzioni di salto incondizionato e salto condizionato.

Codici di condizione e registro di stato



- Il salto condizionato richiede la verifica di una determinata condizione
- Ciò può avvenire sia confrontando direttamente i contenuti dei due registri coinvolti (stile RISC, si veda nel seguito)
- Oppure può avvenire dotando la macchina di un registro di stato (stile CISC), chiamato solitamente *Processor Status Word (PSW)*
- I bit di *PSW* rappresentano il risultato di operazioni e possono essere esaminati tramite istruzioni di salto condizionato



- Alcuni bit di stato sempre contenuti in *PSW*:
  - *Z*(zero): il risultato dell'operazione precedente è 0
  - *S*(segno): il risultato dell'operazione precedente è negativo
  - *O*(overflow): l'operazione precedente ha prodotto traboccamento
  - *C*(carry): l'operazione precedente ha prodotto riporto

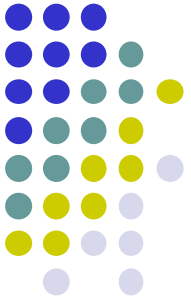
- La macchina è dotata di istruzioni in grado di esaminare uno o più bit di stato: *JZ* (jump if zero), *JNZ* (jump if not zero), *JG* (jump if greater)

- Prima di un'istruzione di salto, in caso di utilizzo di *PSW* può essere necessaria un'istruzione di confronto per determinare i valori dei bit di *PSW:*

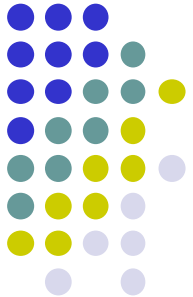
```
CMP  R1, R2          CMP  R1, R2
JE DEST              JG DEST
```

- Dopo la *CMP, JE* salta se *R1=R2* e *JG* se *R1>R2*

Altre istruzioni di controllo del flusso

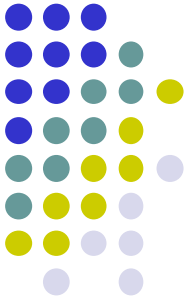


- Altre istruzioni che possono alterare il normale flusso di esecuzione del programma riguardano la gestione dei sottoprogrammi



- Sottoprogrammi:
  - nella programmazione è spesso necessario che una determinata sequenza di azioni venga ripetuta su dati diversi, come ad esempio il calcolo di una funzione del tipo *y=sin(x)*
  - il programmatore definisce quindi un *sottoprogramma* dal nome *sin* che verrà chiamato di volta in volta passandogli come *parametro* il valore su cui deve calcolare il seno
  - in termini di linguaggio macchina il sottoprogramma è costituito da un blocco di istruzioni (consecutive) e la chiamata un trasferimento ad esso del controllo
  - il sottoprogramma deve poter essere chiamato da qualsiasi punto del programma e al termine il controllo deve tornare al punto della chiamata
  - ciò richiede che vengano stabiliti meccanismi e convenzioni riguardo al modo di
    - effettuare il collegamento tra programma chiamante e sottoprogramma
    - tenere traccia dell'indirizzo di ritorno
    - attuare il passaggio di parametri
  - Li vedrete nei prossimi corsi

Verso il repertorio delle istruzioni



Vediamo ora alcune istruzioni che risultano essere molto utili nella scrittura di programmi in linguaggio macchina

- *LD Ra,V(Rb)* *;Ra ← M[V+Rb]*
  - modifica l'istruzione di load vista precedentemente per permettere l'indicizzazione, ossia
  - il calcolo dell'indirizzo della cella coinvolta è dato dalla somma di un indirizzo base (V) più un indice corrispondente al registro *Rb*

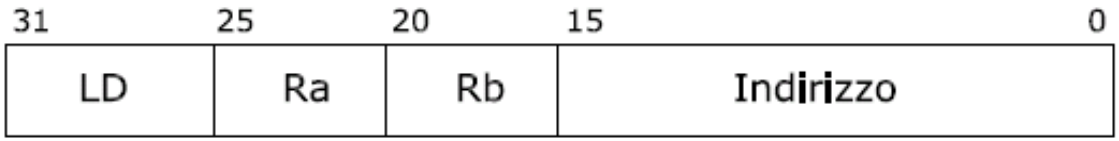
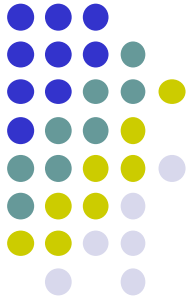
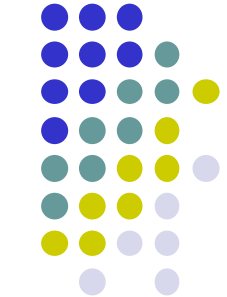


Figura 5.6 Formato dell'istruzione `LD` per consentire il riferimento relativo a un registro. L'istruzione ha l'effetto di portare in Rd il contenuto della posizione di memoria il cui indirizzo è dato dalla somma del contenuto di Rb con il contenuto del campo Indirizzo.



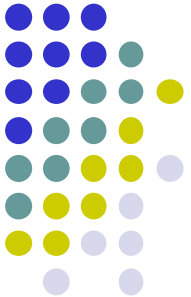
- *ST V(Rb),Ra ; M[V+Rb] ← Ra*
  - equivalente alla precedente per la store
- *ADDI Ra,Rb,<numero> ;Ra ← Rb+<numero>*
  - somma con indirizzamento immediato (add immediate)
  - consente di specificare direttamente il numero coinvolto nella somma, senza doverlo andare a prelevare in memoria o in un registro
- *SUB Rd,Ra,Rb ;Rd ← Ra-Rb*
- *SUBI Rd,Ra,<numero> ;Rd ← Ra-<numero>*
- *LDI Ra,<numero> ;Ra ← <numero>*
  - caricamento immediato (load immediate)
- …

Il repertorio delle istruzioni



- Abbiamo appena visto alcune delle principali funzionalità offerte dalle istruzioni del linguaggio macchina
- Come già accennato, possiamo classificare i tipi di istruzione come segue:
  - elaborazione: istruzioni aritmetiche e logiche
  - memorizzazione: istruzioni sulla memoria
  - trasferimento: istruzioni di I/O
  - controllo: istruzioni di verifica e di salto
- Il numero di istruzioni, le potenzialità delle singole istruzioni, il formato e la loro codifica hanno grande influenza sulle prestazioni
- Si possono identificare due approcci opposti:
  - *RISC (Reduced Instruction Set Computers)*
  - *CISC (Complex Instruction Set Computers)*
- Essi identificano macchine progettate con filosofie contrastanti
- Vediamole brevemente

RISC



Nelle architetture RISC il repertorio delle istruzioni è progettato secondo i seguenti criteri

- le istruzioni hanno tutte la stessa dimensione, per esempio *32* bit
- il codice operativo occupa un spazio predefinito
- esiste un numero estremamente limitato di formati
- il codice operativo identifica in modo univoco il formato
- nel passare da formato a formato, i campi che identificano uguali entità occupano sempre la stessa posizione

- Si noti che tali criteri sono quelli seguiti finora nella descrizione del repertorio delle istruzioni

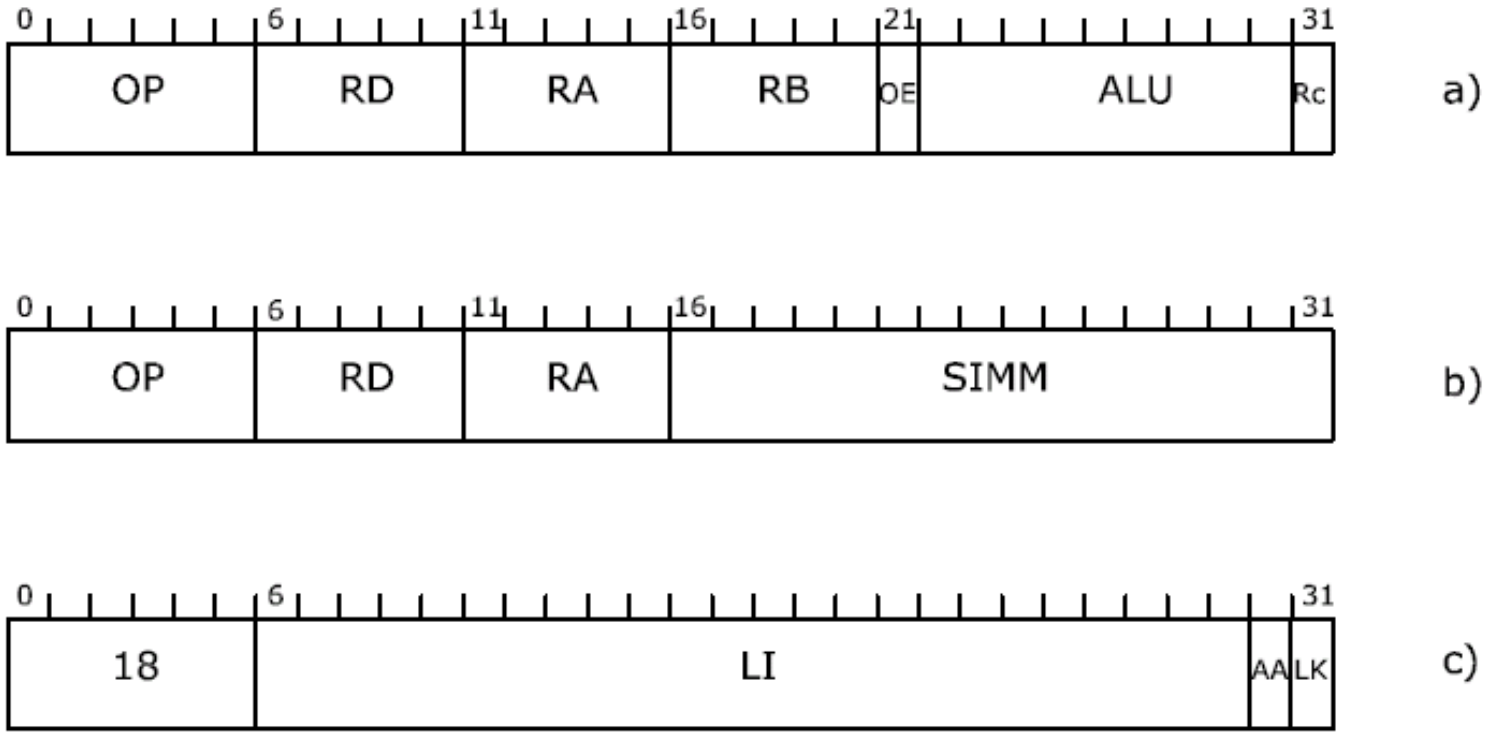
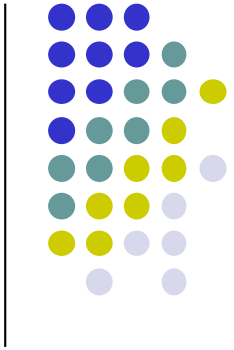


Figura 5.12 Esempio di formati di istruzioni del PowerPC. Tutte le istruzioni aritmetiche tra interi a tre registri prevedono `OP`=31. L'istruzione di somma è identificata attraverso la codifica ALU=266.

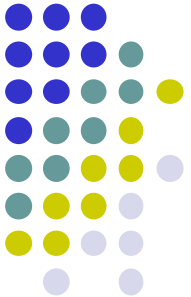


- formato a)
  - corrisponde alle operazioni aritmetiche
  - ognuna può essere specificata con OP=31 variando il campo ALU
  - i campi di un bit OE e Rc specificano quali bit di alcuni registri di stato possono essere modificati e i loro valore corrisponde a codici mnemonici differente (es. add, add., addo, addo.)

- formato b)
  - include le istruzioni di load e store
  - es: lwz RD,d(RA) carica il contenuto della cella di memoria di indirizzo d+RA nel registro RD (d è codificato nel campo SIMM)
  - se RA=r0, l'indirizzo è assoluto, altrimenti si ottiene sommando a SIMM il contenuto di RA
  - formato usato anche per operazioni aritmetiche con operando immediato, tipo addi r1,r2,35 (35 codificato in SIMM)
  - formato usato anche per istruzioni di salto condizionato, tipo be r1,r2,DEST, che salta se r1=r2, ed è tale che in SIMM viene codificato l'indirizzo relativamente a PC, ossia lo spostamento da PC per ottenere DEST
- formato c)
  - per le istruzioni di salto incondizionato
  - in questo caso AA fa interpretare l'indirizzo come assoluto o relativo rispetto al PC

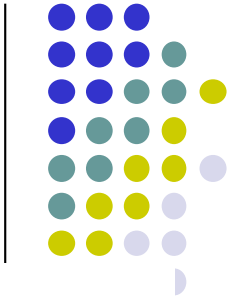
- IMPORTANTE: struttura regolare che rende efficiente la decodifica: decodifica in parallelo dei vari campi

CISC



Nelle architetture CISC il repertorio delle istruzioni è progettato secondo i seguenti criteri

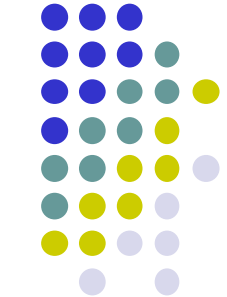
- le istruzioni non hanno dimensione fissa
- il codice operativo può occupare più o meno bit
- esiste un numero alquanto elevato di formati
- il codice operativo, nelle sue possibili variazioni, identifica lo specifico formato delle istruzioni
- nel passare da formato a formato, i campi che identificano uguali entità possono occupare posizioni differenti



PREFISSI				ISTRUZIONE				
Istruzione	Dimensione Indirizzo	Dimensione Operando	Segmento	Codice	MOD R/M	SIB	Scostamento	Immediato
0/1	0/1	0/1	0/1	1/2	0/1	0/1	0/1/2/4	0/1/2/4

Figura 5.13 Formato delle istruzioni nell'architettura ×86, per macchine a 32 bit. Si noti che l'istruzione vera e propria può essere preceduta da ben quattro tipi di prefisso, che ne alterano l'interpretazione usuale. Il campo del codice di operazione può essere di uno o due byte. Tre bit del campo MOD R/M sono da considerare come estensione del codice di operazione. MOD R/M determina quali registri sono implicati, oltre a indicare se c'è riferimento alla memoria e a condizionare la presenza del campo SIB (Scale-Index-Base). Il campo SIB determina il fattore di scala nell'accesso ai dati. Infine il campo scostamento e il campo immediato rappresentano rispettivamente lo scostamento dell'operando in memoria o una quantità codificata nell'istruzione.

Indirizzamento



- Riguarda le modalità di indirizzamento della memoria, ossia le modalità con cui le istruzioni si riferiscono alle celle, sia per quanto riguarda i dati che il controllo
- Come già detto gli indirizzi vengono assegnati ai byte, ma una singola operazione di memoria può richiedere il trasferimento di semiparole e parole
- Prima di introdurre le varie modalità, soffermiamoci su alcune problematiche, tra cui quella dell'allineamento
- Infatti, semiparole o parole disallineate comportano due accessi consecutivi, poiché ogni operazione può considerare solo i quattro banchi di 8 bit che giaggiono sulla stessa riga
- Oltre a comportare rallentamenti, ciò richiede la presenza di logica aggiuntiva per il riordinamento dei byte:

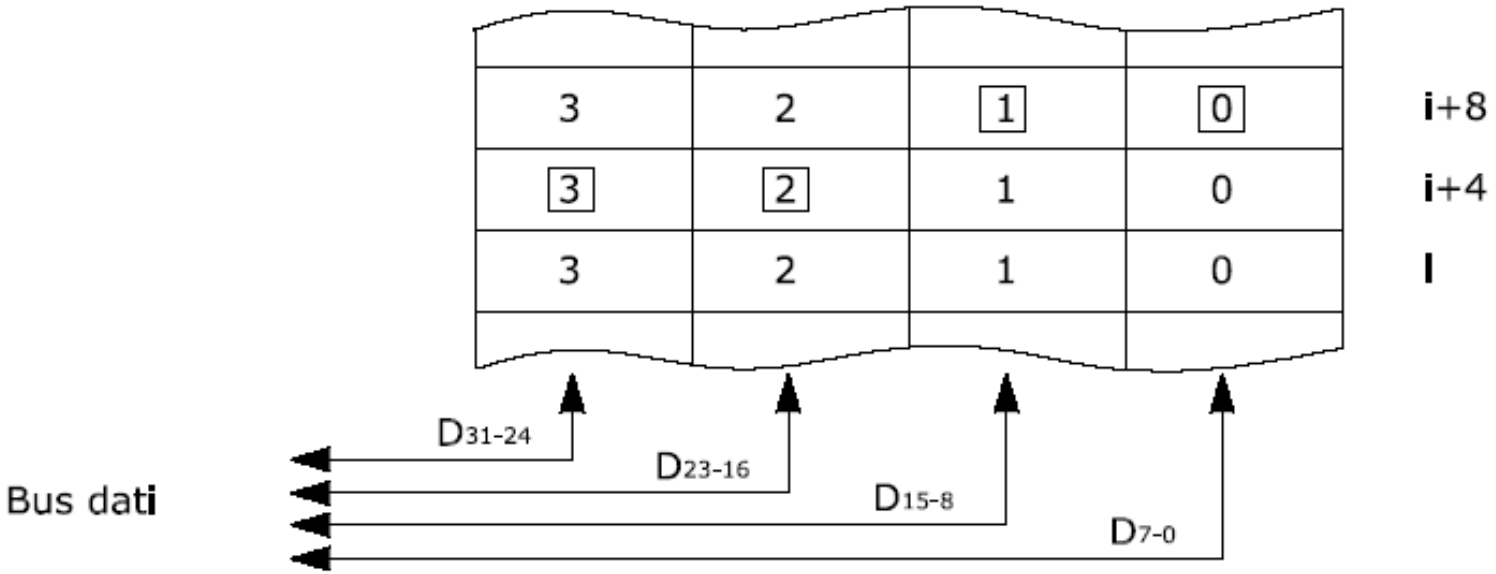
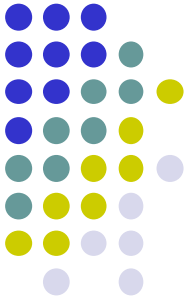
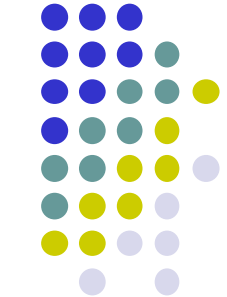


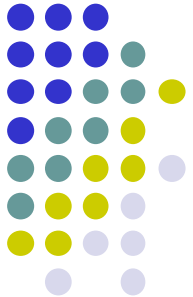
Figura 5.17 Schema del collegamento della memoria al bus dei dati. Una parola allineata al generico indirizzo "i" richiede un solo accesso alla memoria. La parola non allineata, tratteggiata in figura, ha la parte meno significativa nei due byte più significativi alla posizione "i+4" e la parte più significativa nei due byte meno significativi della posizione "i+8". La sua lettura richiede due accessi: uno per ciascuna metà. Inoltre, siccome sul bus le posizioni risultano scambiate, la CPU deve essere dotata della logica per riportare nella giusta posizione le due metà.



- L'architettura Intel consente il disallineamento, ma a partire dal 486 consente al programmatore di "forzare" l'allineamento
- Nelle RISC invece tutte le operazioni di memoria richiedono l'allineamento, sia in riferimento ai dati che alle istruzioni, che occupano sempre una parola (ossia una riga)
- Il tentativo di accesso disallineato quando non consentito solitamente genera un'*eccezione* (tipo di interruzione)
- Anche in caso di allineamento, la logica deve comunque prevedere la possibilità di operazioni di memoria che coinvolgono byte o semiparole, posizionandoli opportunamente sul bus dati (es. byte sulle linee $D_{23\text{-}16}$ anziché $D_{7\text{-}0}$)
- Prima di andare a trattare separatamente le modalità di indirizzamento dei dati e del controllo, risolviamo un'ultima questione



- Essa riguarda la traduzione delle istruzioni assembler in linguaggio macchina
- A tal proposito si consideri un'istruzione del tipo *LD R1,Var*
- Da quanto detto finora
  - ad *LD* corrisponde il codice operativo *110011*
  - ad *R1* la sequenza di cinque bit *00001*
  - e a *Var*?
- Domanda: in base a quale criterio l'assemblatore (o il compilatore) può calcolare IND a partire dal nome simbolico *Var?*
- La risposta a priori non è così banale, perché una volta tradotto il programma può essere "caricato" in memoria centrale per l'esecuzione in posizioni diverse
- Consideriamo separatamente due casi: memoria lineare e memoria segmentata



Memoria lineare

- La memoria è vista come un unico blocco, dall'indirizzo *0* ad *M-1*

- *Rilocazione:*
  - l'assemblatore determina l'indirizzo di *Var* rispetto all'indirizzo *0,* assunto come indirizzo di partenza del programma, ossia della prima istruzione
  - all'atto del caricamento del programma in memoria, a IND viene aggiunto l'effettivo indirizzo di partenza
  - questo processo e' svolto dal *loader* (caricatore)

- Vediamo un esempio

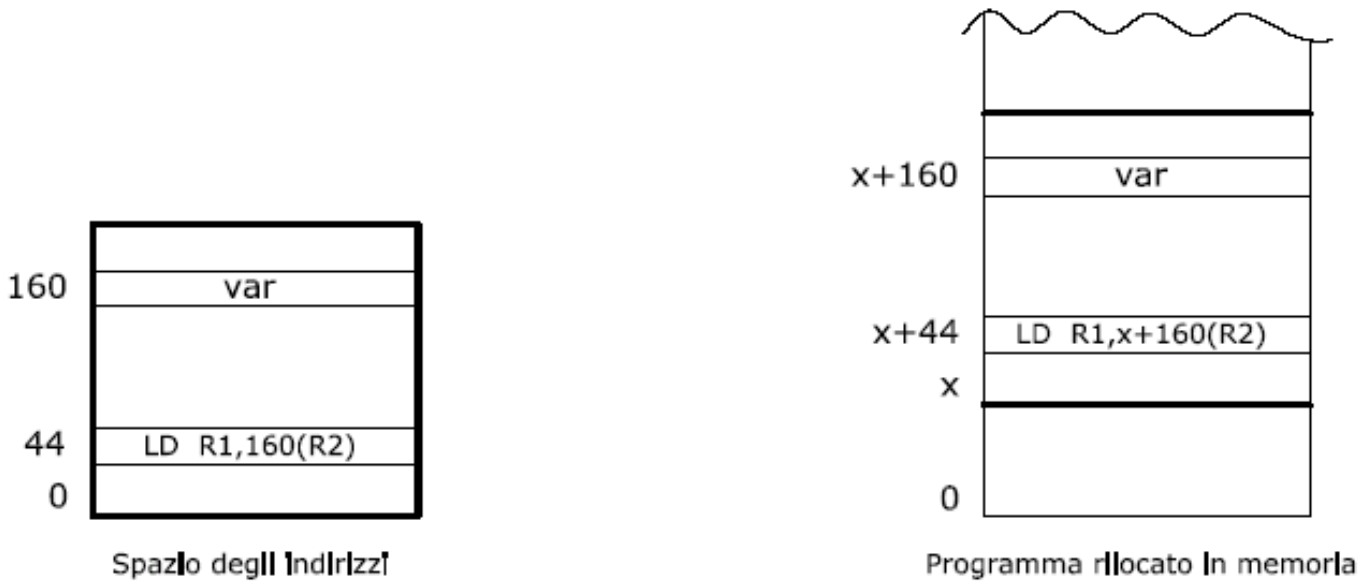
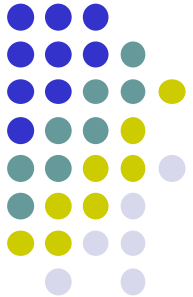


Figura 5.19 Rilocazione di un programma in uno spazio di memoria lineare. I campi relativi agli indirizzi sono calcolati dall'assemblatore o dal compilatore in riferimento alla posizione convenzionale 0 nello spazio degli indirizzi. All'atto del caricamento in memoria, questi campi vengono modificati in modo da tener conto della posizione di caricamento del programma in memoria (indicata con x).



Memoria segmentata

- La memoria è divisa in segmenti, di posizione e dimensioni variabili
- Esistono registri che al tempo di esecuzione contengono l'indirizzo di partenza (la *base*) dei segmenti
- Essi sono sempre dietro le quinte e intervengono ogni qual volta viene generato un indirizzo
- Tutte le tecniche utilizzabili nel modello lineare sono trasportabili al modello segmentato
- Per tale motivo si usa il termine di *indirizzo effettivo* per indicare l'indirizzo calcolato dalla CPU, indipendentemente dalla presenza dei registri segmento
- Se la memoria è lineare, l'indirizzo fisico corrisponde a quello effettivo, mentre se è segmentata per ottenere l'indirizzo fisico all'indirizzo effettivo deve essere sommato il contenuto del registro di segmento:

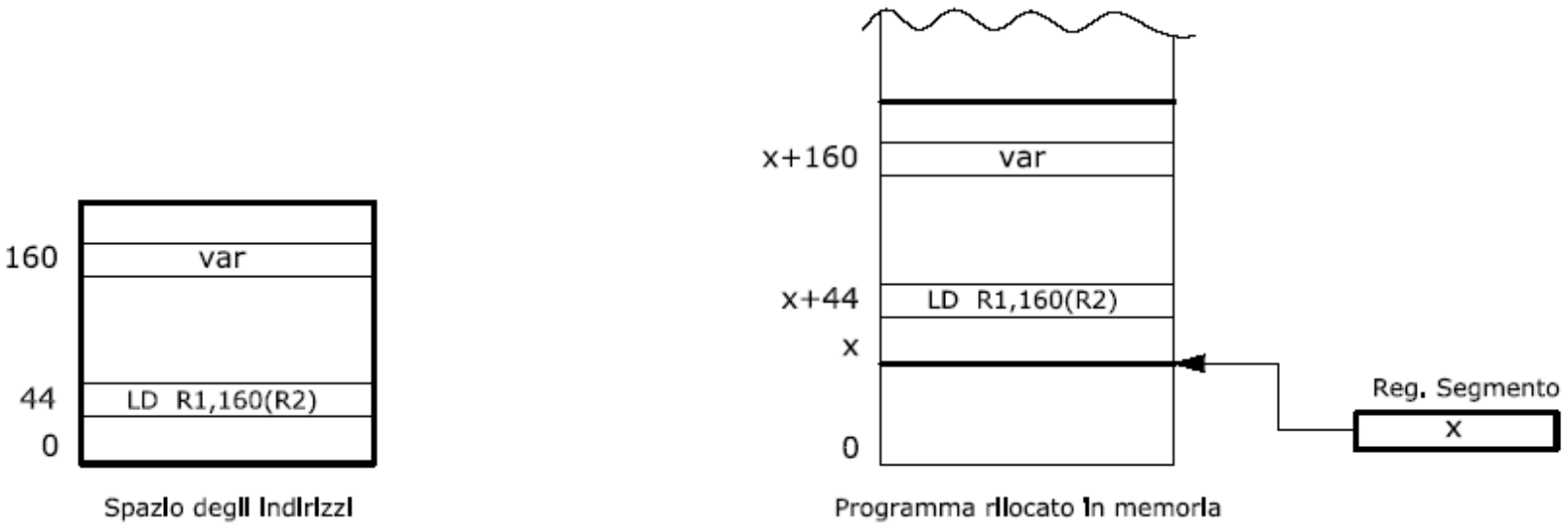
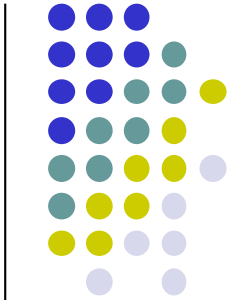


Figura 5.20 Rilocazione di un programma in uno spazio di memoria segmentata. I campi relativi agli indirizzi, calcolati dall'assemblatore o dal compilatore in riferimento alla posizione convenzionale 0, non necessitano di essere modificati. Per tener conto della posizione di caricamento del programma viene usato un *registro di segmento*. Naturalmente l'esecuzione dell'istruzione richiede che venga prima calcolato l'indirizzo effettivo come somma dei contenuti del campo `IND` e di Rb e che questo venga sommato al contenuto del registro di segmento per determinare l'indirizzo fisico in memoria.

Indirizzamento dei dati



- Esistono diverse modalità, sorte anche con lo scopo di risolvere il problema della codifica dell'indirizzo nelle istruzioni in linguaggio macchina
- Infatti spesso il campo indirizzo non è sufficientemente ampio per poter indirizzare tutta la memoria, ossia specificare tutti i possibili indirizzi
- Una possibile soluzione per superare tale limitazione è quello di interpretare l'indirizzo non come effettivo, ma come scostamento rispetto ad un registro di riferimento, come ad esempio *PC*
- In questo caso si determina una *finestra* all'interno della quale il programma può indirizzare che segue l'esecuzione del programma stesso
- Nelle architetture correnti si preferisce interpretare l'indirizzo come scostamento rispetto ad altri registri, mentre il *PC* funge da riferimento esclusivamente per le istruzioni di salto
- L'ulteriore registro deve essere codificato nell'istruzione, riducendo ulteriormente la dimensione del campo indirizzo
- Vediamo un esempio nel modello registro-memoria relativo ad una tipica istruzione
  LD RA,Var

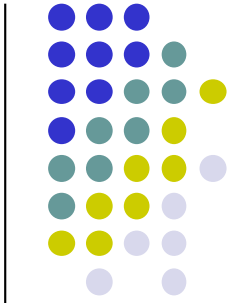
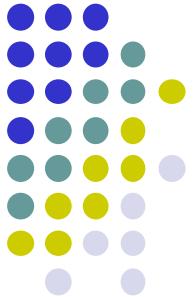


Figura 5.18 Possibili formati di istruzioni che indirizzano la memoria. Il campo IND contiene l'indirizzo della cella di memoria cui l'istruzione fa riferimento. Questo campo è di norma insufficiente a rappresentare la dimensione massima della memoria. Se per esempio la macchina ha un bus di indirizzi a 32 bit e le istruzioni sono su 32 bit è indubbio che la dimensione del campo IND dell'istruzione in alto sarà comunque inferiore a 32. Occorre prevedere uno schema in cui IND è solo una componente dell'indirizzo calcolato dalla CPU. Nel caso dell'istruzione in basso, l'indirizzo viene calcolato come somma del contenuto del campo IND e del contenuto del registro Rb. L'interpretazione di IND dipende dallo schema di indirizzamento, nel senso che esso può rappresentare uno scostamento o un indirizzo assoluto (si veda al Paragrafo 5.10.1).

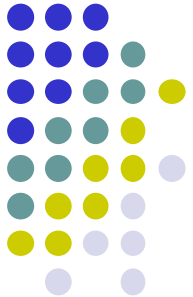


- In tale esempio l'istruzione diventa del tipo *LD RA,Var(RB),* con *RA* corrispondente ad *Rsd* e *RB* a *Rb*

- Si indica con *indirizzo effettivo (effective address)* l'indirizzo calcolato attraverso i componenti espliciti codificati nell'istruzione

- Ad esempio nell'esempio precedente l'indirizzo effettivo nel primo caso è esattamente IND, mentre nel secondo è dato dalla somma di IND e *Rb*

- Come abbiamo visto, l'indirizzo effettivo può però non essere quello definitivo o *fisico,* cioè quello della locazione di memoria coinvolta nell'operazione (può essere implicitamente sommato l'indirizzo di base del segmento)



A partire dalle considerazioni precedenti, elenchiamo alcune possibili modalità di indirizzamento di dati:

- diretto: è il più semplice, il campo IND viene interpretato come indirizzo effettivo

  Es: *LD R1,Var* *;EA = IND (R1 ← M[EA])*

- relativo ai registri: l'indirizzo effettivo è dato dalla somma del campo IND e del contenuto del registro *Rb*

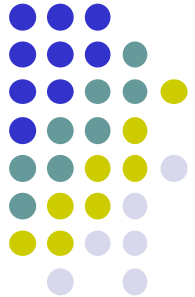
  Es: *ST Var(R2),R5* *;EA = IND + R2*

- indiretto rispetto ai registri: simile al precedente, ma senza campo indirizzo nell'istruzione

  Es: *LD R1,(R2)* *;EA = R2*

- indiretto relativo ai registri, scalato e con indice: si aggiunge un secondo registro con funzione di *indice;* molto utile con strutture dati tipo vettori, in cui il secondo registro viene usato come indice nella struttura scalando in base alla dimensione del singolo elemento

  Es: *LD R1,Var(R2)(RX)* *; EA = IND + R2+RX*d* (*d*=dim. elemento)



- indiretto rispetto ai registri con autoincremento: consente di incrementare automaticamente il contenuto del registro; utile negli stessi casi del precedente, esiste anche versione con decremento

  Es: *LD R1, (R2)+* *;EA = R2,R2←R2+d (d*=dim.elem.)

- immediato: non è propriamente un indirizzamento in memoria; consente al programmatore di specificare un numero o un'espressione riconducibile a un numero direttamente nell'istruzione; non c'e' quindi accesso in memoria, in quanto il dato è già presente in CPU

  Es: *LD R1, 2467* *;R1 ← 2467*

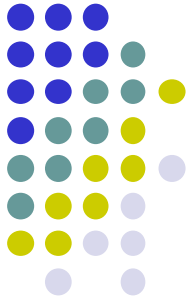
- dei registri: neanche questo è propriamente un indirizzamento in memoria; i registri vanno di norma specificati dal programmatore, anche se in certi casi uno può essere predefinito:

  Es: *LD R1, R2* *;R1 ← R2*

- delle porte di I/O: per le operazioni di I/O; l'indirizzo della porta di norma e' codificato in un campo dell'istruzione, anche se sono possibili soluzioni alternative come contenerlo in un registro

  Es: *IN AL,PORTA* *;AL ← PORTA di ingresso*

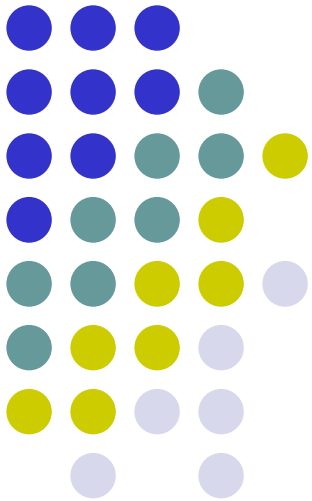
Indirizzamento nel trasferimento del controllo

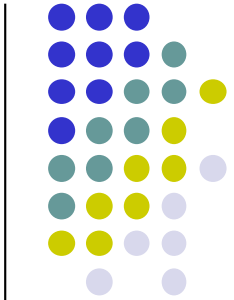


- Si ottiene con le istruzioni di salto, di chiamata e di ritorno dai sottoprogrammi (interruzioni a parte)
- Da studi effettuati, risulta che i salti condizionati tendono a modificare il *PC* di una misura contenuta
- I salti incondizionati invece spesso corrispondono all'abbandono di un tratto di codice ed è più probabile che la misura del salto sia meno contenuta
- L'indirizzamento si riduce a due modalità essenziali:
  - indirizzamento assoluto:
    - solitamente usato nei salti incondizionati
    - nell'istruzione viene codificato l'indirizzo di destinazione
  - indirizzamento relativo :
    - solitamente usato nei salti condizionati
    - l'indirizzo di destinazione viene di norma calcolato come scostamento dal *PC,* ma a volte possono essere utilizzati anche registri generali

La CPU

- La struttura della CPU
- Logica cablata/microprogrammazione
- Prestazioni

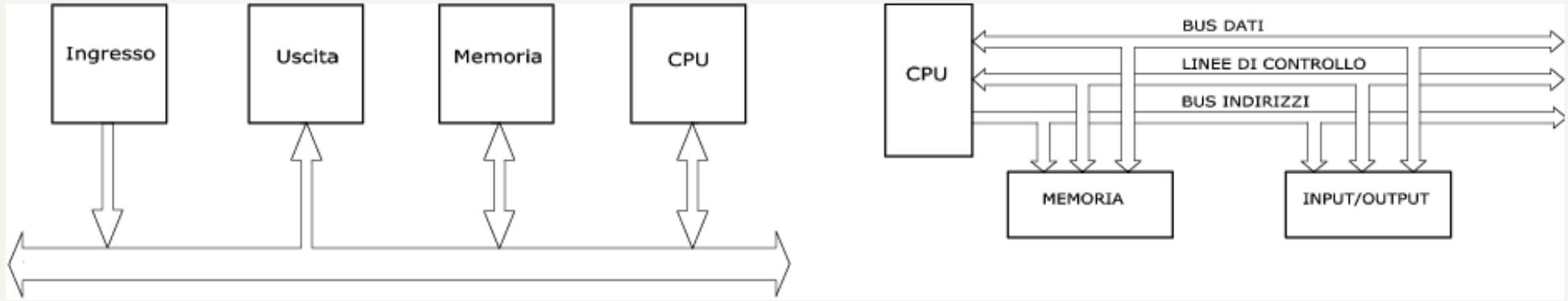




- Approfondiamo ora gli aspetti relativi all'esecuzione delle istruzioni da parte della CPU
- A tale scopo forniamo un modello semplificato di CPU descrivendo il suo funzionamento interno
- Esamineremo in dettaglio il processo di esecuzione, ricavando le forme d'onda dei segnali generati dalla CPU e le relative espressioni logiche
- Infine considereremo brevemente il problema della valutazione delle prestazioni

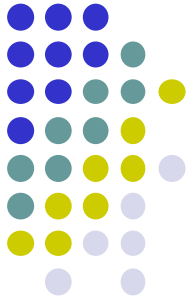
Schema di riferimento

- Per il momento lo schema di riferimento sarà quello nella figura seguente



- Corrisponde allo schema dei PC anni 80
- Tuttora in largo uso nei sistemi di controllo
- A seconda dello schema organizzativo del sistema CPU-memoria, sono possibili due architetture di riferimento

Architettura di Von Neumann



- memoria indifferenziata per dati o istruzioni
- è il modello a cui faremo riferimento

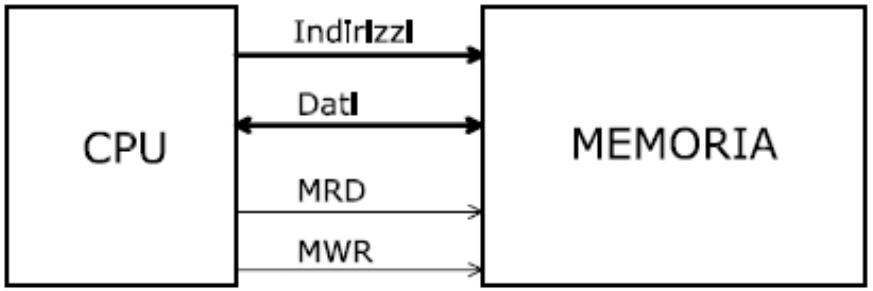
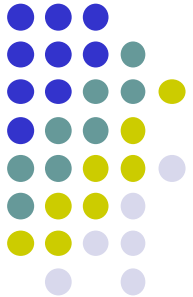


Figura 6.1 Architettura di Von Neuman. La memoria è indifferenziata per dati o istruzioni, solo l'interpretazione da parte della CPU stabilisce se una determinata configurazione di bit è da riguardarsi come un dato o come un'istruzione.

Architettura Harward



- due memorie distinte: la memoria istruzioni e la memoria dati
- trova impiego in specifici campi applicativi, come l'elaborazione di segnali, ma non i sistemi di uso generale

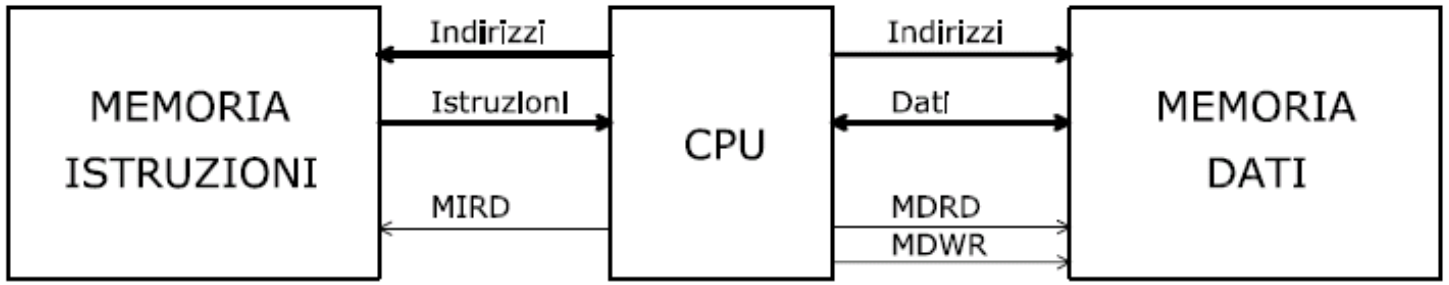
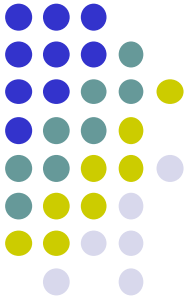


Figura 6.2 Architettura Harvard. Si noti che per quanto si riferisce alla sola esecuzione di un programma nella memoria istruzioni, la linea di comando MIRD è del tutto superflua, in quanto si può immaginare che questa memoria sia sempre e soltanto letta.



- Tenendo conto del modello di Von Neumann, la CPU osservata dall'esterno compie essenzialmente due operazioni: letture e scritture in memoria
- Vediamo un esempio di temporizzazione
- Si suppone che la CPU effettui un'operazione di lettura in 3 cicli di clock
- Sul ciclo T1 la CPU asserisce l'indirizzo e il comando di lettura MRD
- La memoria risponde immettendo nel bus dati il contenuto della cella indirizzata
- Nell'esempio i dati saranno disponibili nel ciclo T3, durante il quale vengono acquisiti dalla CPU
- Se la memoria è più lenta, la logica deve prevedere un segnale di ingresso solitamente denominato WAIT, asserito dall'esterno, che ha l'effetto di far inserire dalla CPU uno o più cicli di wait per ritardare il ciclo T3

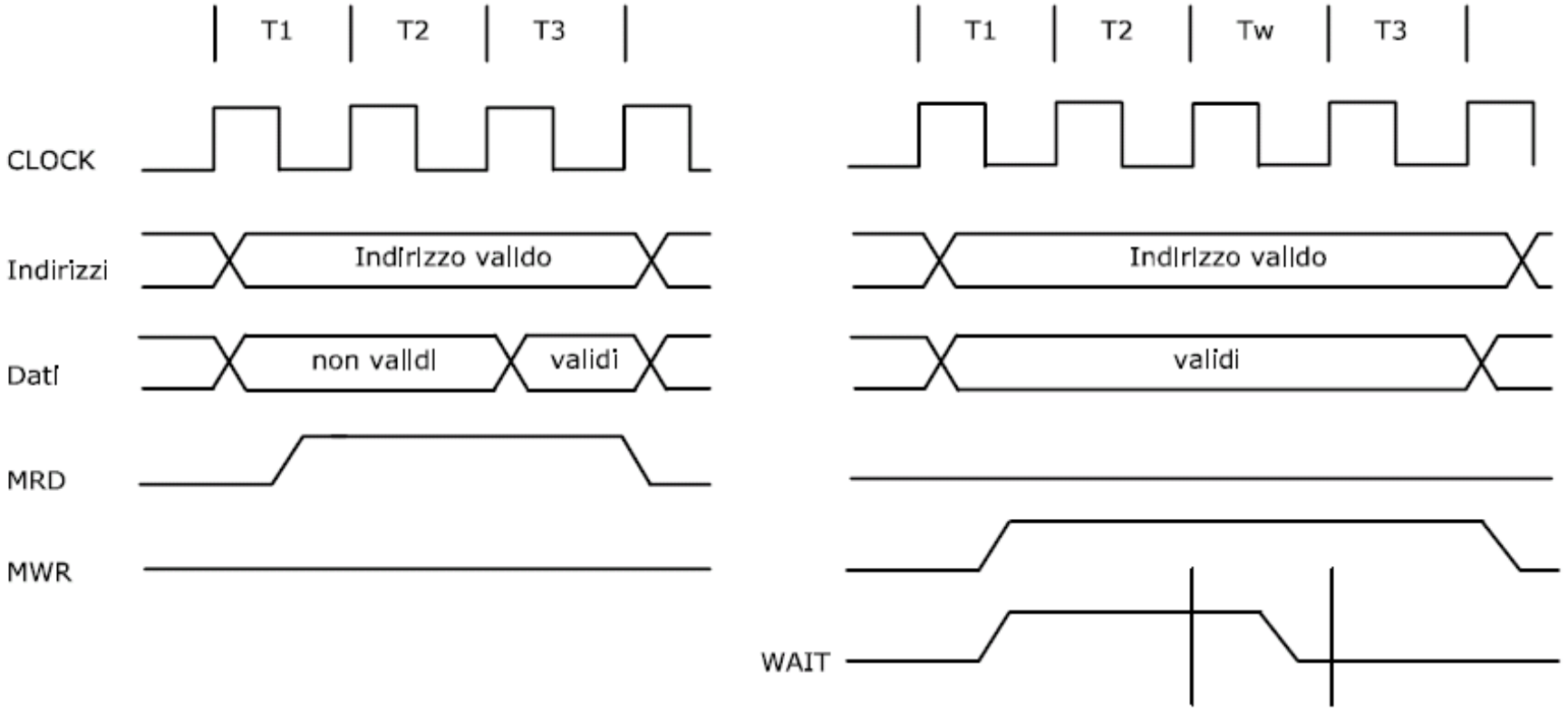
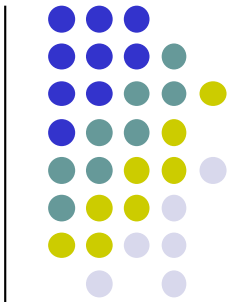


Figura 6.3 Temporizzazione qualitativa dei cicli di lettura e scrittura. Nel caso specifico, la scrittura presenta un ciclo di *wait*. Si ipotizza che la logica di CPU controlli l'ingresso WAIT sul fronte di salita di T2 (e su quello di Tw) e se questo è asserito inserisca un ciclo Tw.

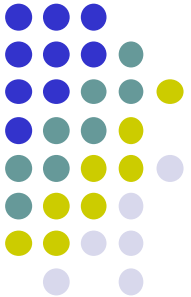
La CPU



- Possiamo fornire una schematizzazione aggregata in cui la CPU è divisa in due parti: unità di controllo e unità operativa

- Unità di controllo (UC):
  - responsabile dell'esecuzione delle istruzioni
  - comanda l'unità operativa affinché svolga le azioni previste dall'istruzione
  - in pratica interpreta il codice operativo traducendolo in sequenze temporizzate di comandi all'unità operativa

- Unità operativa (UO):
  - svolge la funzione di manipolazione e trasformazione dell'informazione, trasformando i dati di ingresso X in dati di uscita Y
  - include l'unità logico-aritmetica (ALU)
  - possiede diversi registri per depositare l'informazione manipolata
  - comunica all'unità di controllo le condizioni di stato, in modo da informarla circa i risultati intermedi



- Entrambe le unità sono reti sequenziali sincrone
- Mentre la UO potrebbe essere una rete combinatoria, l'UC e' necessariamente sequenziale
- L'UC può essere descritta da un automa che segue il proprio diagramma di stati
- L'esecuzione di un'istruzione normalmente richiede più cicli di clock, corrispondenti al passaggio attraverso altrettanti stati dell'automa

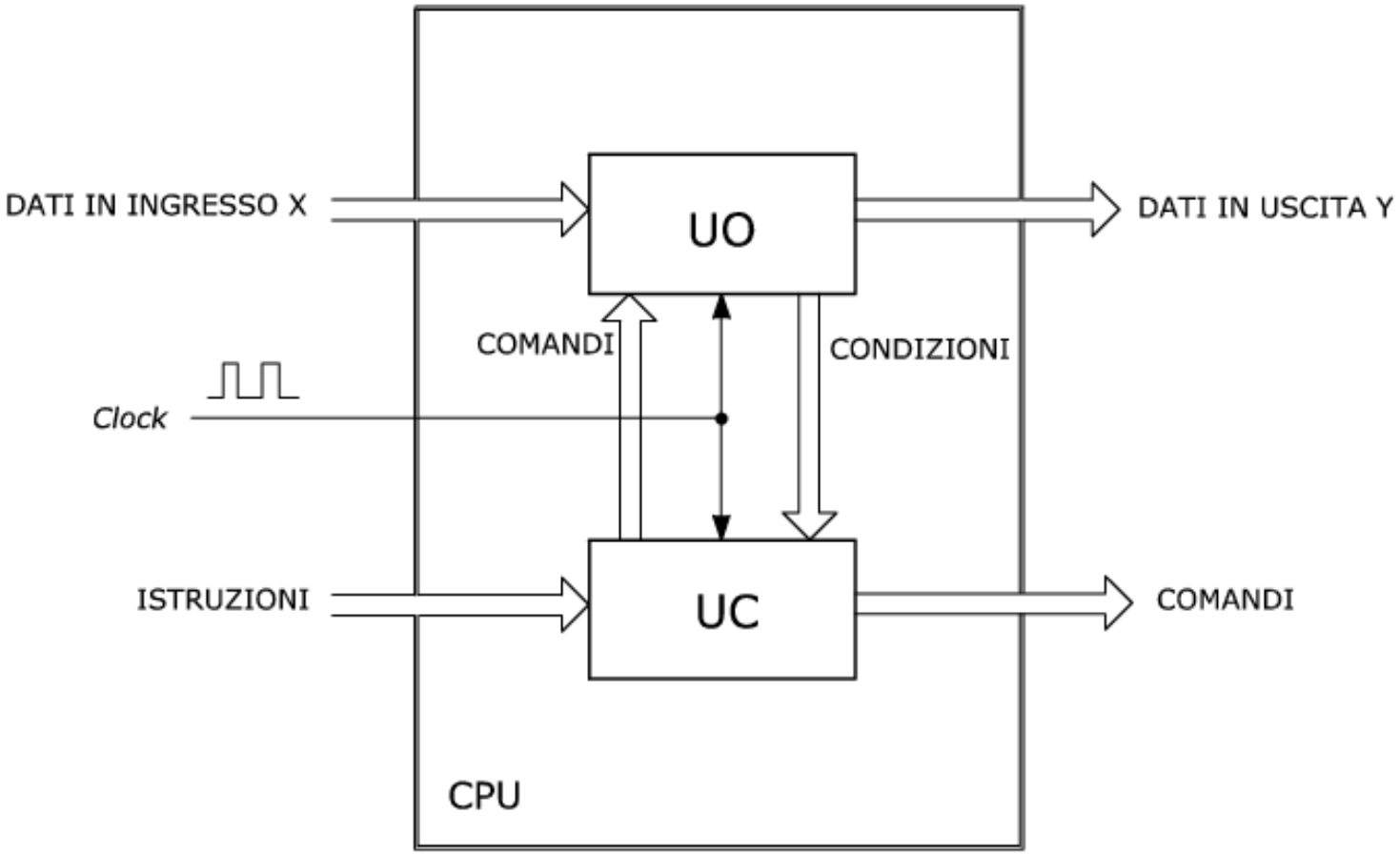
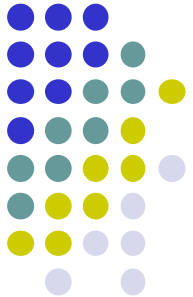


Figura 6.4 Struttura generale della CPU. Lo schema mette in evidenza due parti: l'unità operativa (UO) e l'unità di controllo (UC). Lo schema mostra anche i flussi informativi tra UC e UO e tra l'intera CPU e l'esterno [LP86].



- Come già visto, l'UC può essere in due macrostati: *fetch* e *execute*

- *Fetch*:
  - Prelievo dalla memoria e decodifica dell'istruzione
  - Fase comune a tutte le istruzioni
- *Execute*:
  - Fase in cui vengono eseguite le azioni previste dal codice di operazione
  - Diversa da istruzione a istruzione

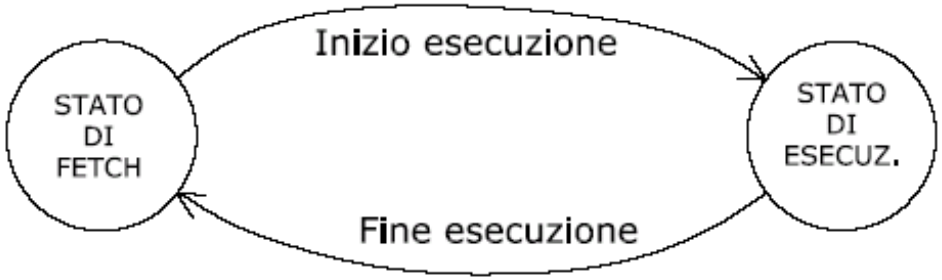
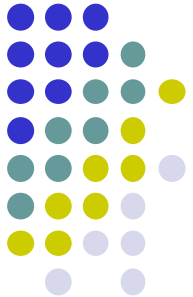


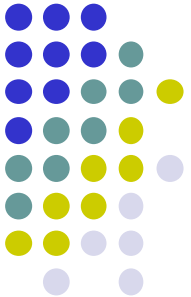
Figura 6.5 Diagramma di stato aggregato del funzionamento dell'unità di controllo. Il primo macrostato, detto stato di *fetch*, comprende il prelievo e la decodifica della prossima istruzione; il secondo macrostato, detto stato di *execute*, comprende tutte le azioni che vengono svolte per quella specifica istruzione.

Registri della CPU



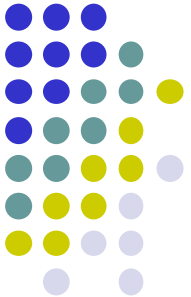
Ecco una lista dei principali registri interni alla CPU e della loro funzione:

- **IR**:
  - usato per contenere l'istruzione in corso di esecuzione
  - caricato in fase di fetch
  - rappresenta l'ingresso della logica di controllo che determina le azioni svolte durante la fase di esecuzione
- **PC**:
  - tiene traccia dell'esecuzione del programma, mantenendo l'indirizzo in memoria della prossima istruzione da eseguire
  - aggiornato durante la fase di fetch per predisporre la successiva



- **MAR**:
  - contiene l'indirizzo della locazione di memoria da leggere o scrivere
  - l'uscita è sul bus indirizzi
  - il flusso di dati è unidirezionale (dalla CPU verso la memoria)
  - di norma l'uscita è abilitata durante le operazioni di memoria, in terzo stato altrimenti, anche se per semplicità la supporremo sempre abilitata
  - La dimensione di MAR determina l'ampiezza dello spazio di memoria fisica; dalla fine degli anni '80 vengono prodotti microprocessori con bus indirizzi a 32 bit

- **DTR**:
  - registro attraverso il quale viene scambiata l'informazione tra la CPU e il resto del sistema
  - l'uscita è sul bus dati
  - il flusso di dati è bidirezionale (dall'esterno alla CPU in lettura, e dalla CPU verso l'esterno in scrittura)
  - tradizionalmente di DTR dà la misura del grado di parallelismo della macchina (8, 16, 32, 64 bit)
  - La dimensione di DTR dalla metà degli anni '90 è di 64 bit

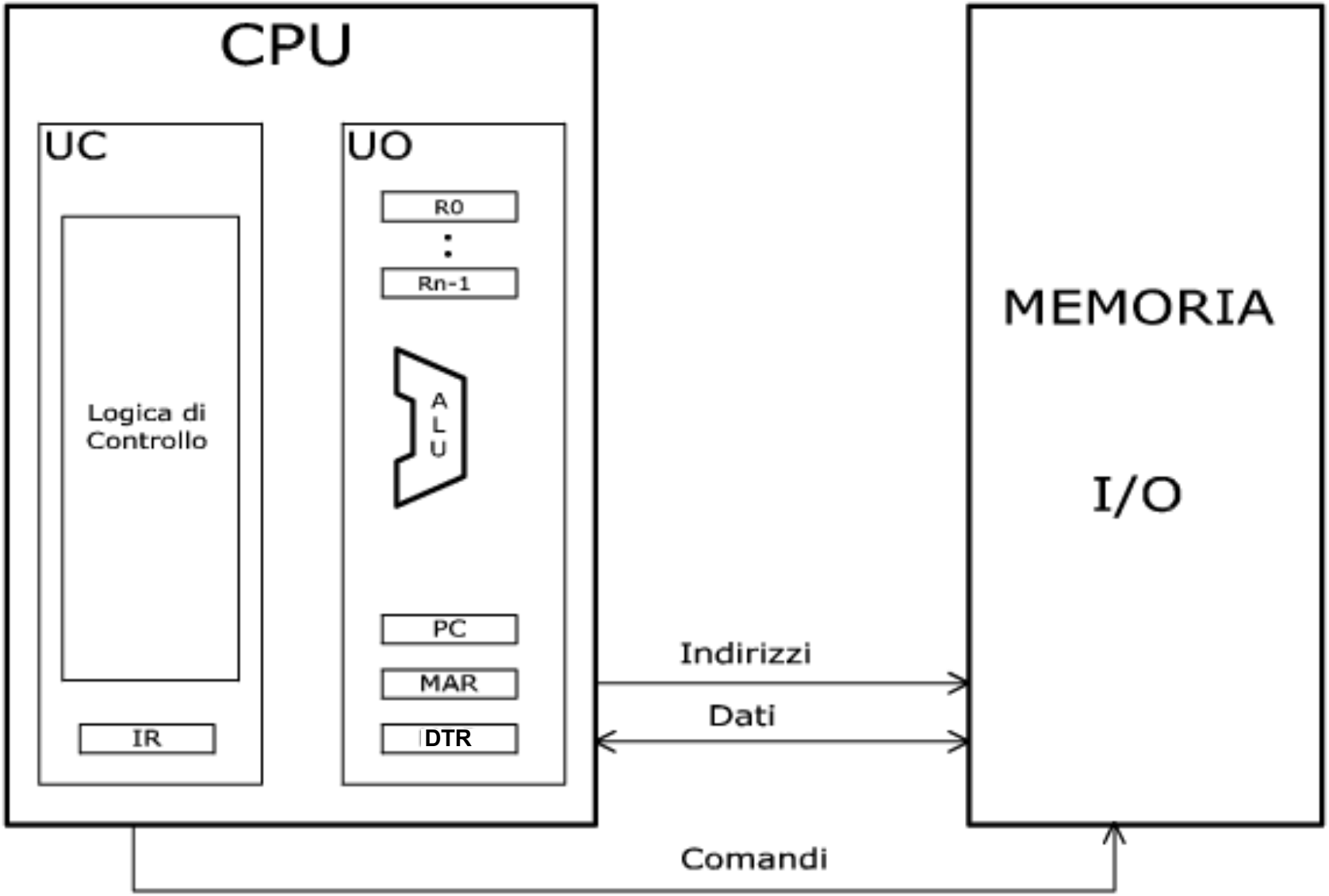


- **R0, R1,...Rn**:
  - registri di uso generale impiegati per mantenere i dati manipolati dall'ALU
  - nelle architetture moderne c'e' la tendenza ad averne un numero elevato e di utilizzarli in modo intercambiabile
  - in passato i registri erano più specializzati (accumulatore, necessariamente coinvolto nelle operazioni aritmetiche, chiamato AX nell'architettura X86)

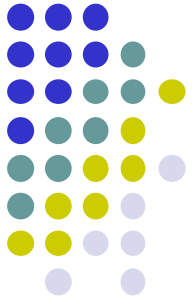
- I registri MAR e DTR non sono essenziali, poiché per prelevare le istruzioni basterebbe presentare PC sul bus indirizzi, mentre per i dati basterebbe stabilire un percorso tra il bus dati e il registro implicato

- Ad ogni modo vengono sempre usati come tramite nelle operazioni di lettura/scrittura, in quanto consentono di disaccoppiare CPU e memoria

- Riassumendo, abbiamo i seguenti componenti essenziali:



Struttura interna



- Per descrivere i passi di esecuzione di un'istruzione, bisogna tener conto dei percorsi dei dati
- Mostriamo ora una possibile struttura interna della CPU a bus singolo
- Ciò implica che può essere effettuato un solo trasferimento alla volta da una singola sorgente ad una o più destinazioni
- L'esecuzione delle istruzioni avviene in più passi corrispondenti alle azioni elementari elencate nel seguito
- Nel funzionamento descritto, si assume che i registri operino come visto precedentemente

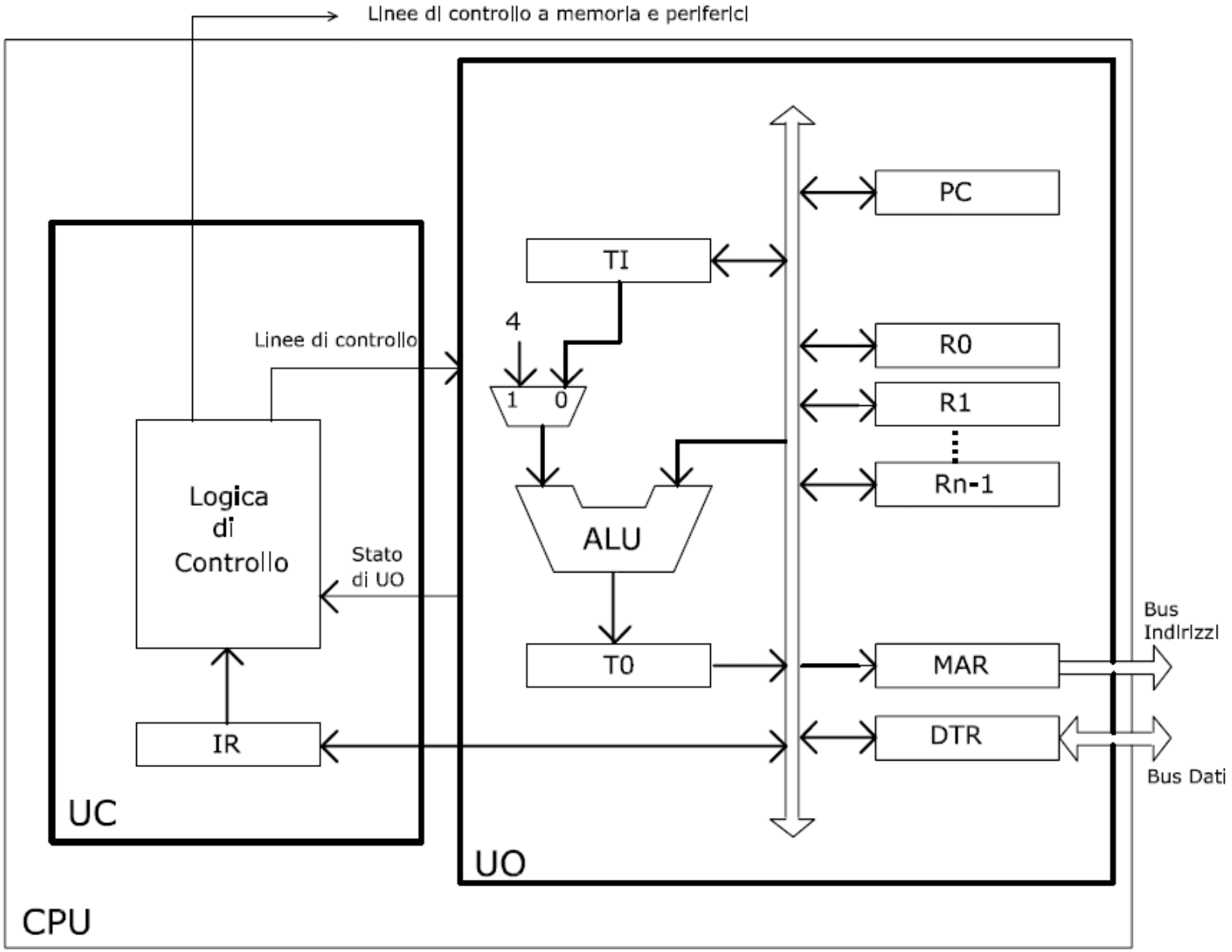


Figura 6.6 Organizzazione del percorso dati a singolo bus interno.