Struttura e Funzione del Processore

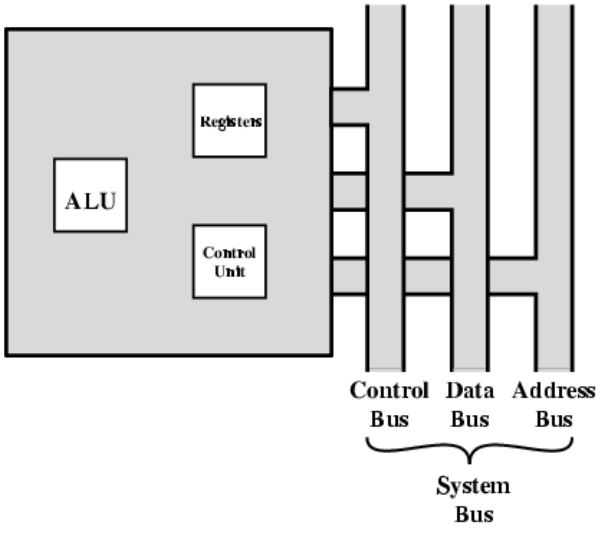
Capitolo 12

Struttura CPU

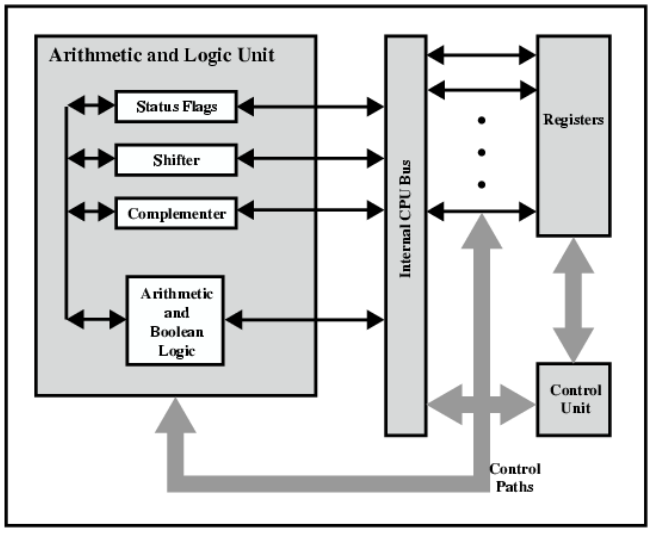
Compiti CPU:

- Prelevare istruzioni 
- Interpretare istruzioni 
- Prelevare dati 
- Elaborare dati 
- Scrivere (memorizzare) dati 

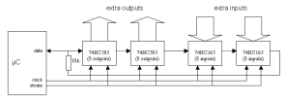
CPU con bus di sistema



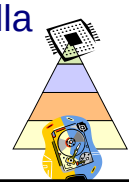
Struttura interna CPU



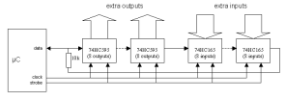
Registri



- CPU ha bisogno di uno “spazio di lavoro” dove memorizzare i dati
- Questo “spazio di lavoro” è costituito dai **registri**
- Numero e funzioni svolte dai registri varia a seconda dell’impianto progettuale della CPU
- Scelta progettuale molto importante
- I registri costituiscono il livello più alto della così detta “Gerarchia della memoria”



Registri



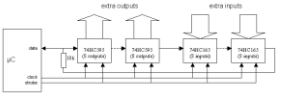
- Registri utente
  - usati dal “programmatore” per memorizzare internamente alla CPU i dati da elaborare
- Registri di controllo e di stato
  - usati dall’unità di controllo per monitorare le operazioni della CPU
  - usati dai programmi del Sistema Operativo (SO) per controllare l’esecuzione dei programmi

“programmatore”

1. Umano che programma in assembler
2. Compilatore che genera codice assembler a partire da un programma scritto in un linguaggio ad alto livello (C, C++, Java,…)

Ricordarsi che un programma in assembler è trasformato in codice macchina dall’assemblatore (+ linker) che trasforma il codice mnemonico delle istruzioni in codice macchina

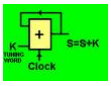
Registri visibili all’utente: registri utente



- Ad uso generale
- Per la memorizzazione di dati
- Per la memorizzazione di indirizzi
- Per la memorizzazione di codici di condizione

Registri ad uso generale

- Possono essere veramente ad uso generale
- …oppure dedicati a funzioni particolari
- Possono essere usati per contenere dati o indirizzi
- Dati
  - Ad esempio: accumulatore



- Indirizzi
  - Ad esempio: indirizzo base di un segmento di memoria

Segmento di memoria

- La memoria principale può essere organizzata, dal punto di vista logico (cioè concettuale), come un insieme di segmenti o spazi di indirizzamento multipli:
  - "visibili" al "programmatore", che riferisce logicamente una locazione di memoria riferendo il segmento e la posizione della locazione all'interno del segmento:
    es. segmento 4, locazione 1024
  - come supporto a questa "visione" della memoria, occorre poter indicare **dove**, all'interno della memoria fisica, inizia il segmento (*base*) e la sua **lunghezza** (*limite*)
    es. il segmento 4 ha base = $00EF9445_{hex}$ e limite = 4MB
  - quindi occorrono dei registri dove memorizzare tali informazioni

Registri ad uso generale

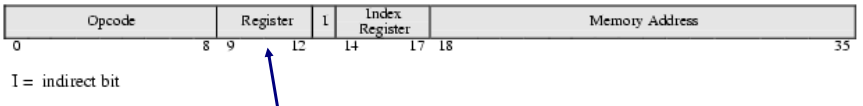
- Registri veramente ad uso generale
  - Aumentano la flessibilità e le opzioni disponibili al programmatore “a basso livello”
  - Aumentano la dimensione dell’istruzione e la sua complessità (perché ?)
- Registri specializzati
  - Istruzioni più piccole e più veloci
  - Meno flessibili

Perché aumenta dimensione e complessità ?

- Facciamo l’esempio di istruzioni a formato fisso:

PDP-10

Campo non necessario se non avessi registri generali: *formato semplificato !*



I = indirect bit

4 bit ➡ 2^4 = 16 registri generali
Se si avessero 32 registri generali sarebbero necessari 5 bit di indirizzamento

Formato più lungo! 37 bit ⇦

Quanti registri generali?

- Tipicamente tra 8 e 32
- Meno di 8 = più riferimenti (accessi) alla memoria principale (perché ?)
- Più di 32 non riducono i riferimenti alla memoria ed occupano molto spazio nella CPU
- Nelle architetture RISC tipicamente si hanno più di 32 registri generali

Perché più accessi ?

ESEMPIO: supponiamo di dover calcolare:

```
mem[4] = mem[0]+mem[1]+mem[2]+mem[3]
mem[5] = mem[0]*mem[1]*mem[2]*mem[3]
mem[6] = mem[5]-mem[4]
```

```
 4 registri
R0 ← mem[0];
R1 ← mem[1];
R2 ← mem[2];
R3 ← mem[3];
R0 ← R0+R1;
R0 ← R0+R2;
R0 ← R0+R3;
R1 ← R1*R2;
R1 ← R1*R3;
R2 ← mem[0];
R1 ← R1*R2;
R0 ← R1-R0;
```

7 op
5 mem

```
 6 registri
R0 ← mem[0];
R1 ← mem[1];
R2 ← mem[2];
R3 ← mem[3];
R4 ← R0+R1;
R4 ← R2+R4;
R4 ← R3+R4;
R5 ← R0*R1;
R5 ← R2*R5;
R5 ← R3*R5;
R0 ← R5-R4;
```

7 op
4 mem

Quanto lunghi (in bit) ?

- Abbastanza grandi da contenere un indirizzo della memoria principale
- Abbastanza grandi da contenere una “full word”
- E’ spesso possibile combinare due registri dati in modo da ottenerne uno di dimensione doppia
  - Es.: programmazione in C
    - double int a;
    - long int a;

Registri per la memorizzazione di Codici di Condizione

- Insiemi di bit individuali
  - es. Il risultato dell’ultima operazione era zero
- Possono essere letti (implicitamente) da programma
  - es. “Jump if zero” (salta se zero)
- Non possono (tipicamente) essere impostati da programma

Registri di Controllo e di Stato

- Program Counter (PC)
- Instruction Register (IR)
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)

Program Status Word

- Un insieme di bit

	0	0	0	0	0	0	0	0	Value after reset
PSW	CY	AC	F0	RS1	RS0	OV		P	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

- Include Codici di Condizione
  - Segno dell'ultimo risultato
  - Zero
  - Riporto
  - Uguale
  - Overflow
  - Abilitazione/disabilitazione Interrupt
  - Supervisore

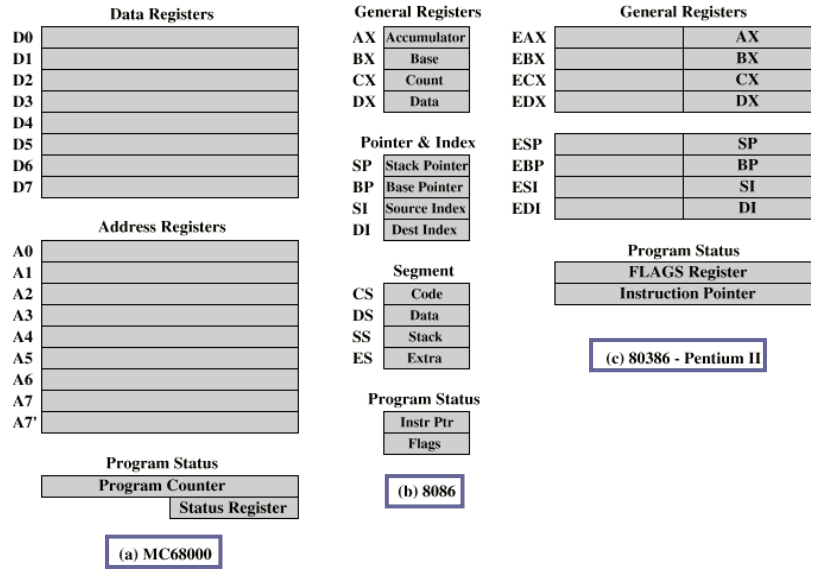
Modo Supervisore

- Permette al sistema operativo di utilizzare le procedure del Kernel, che agiscono su componenti critiche del sistema
- In particolare permette l'esecuzione di istruzioni "privilegiate"
- Disponibile esclusivamente al sistema operativo
- Non disponibile all'utente programmatore
- Lo studierete in dettaglio nel corso di Sistemi Operativi

Altri registri

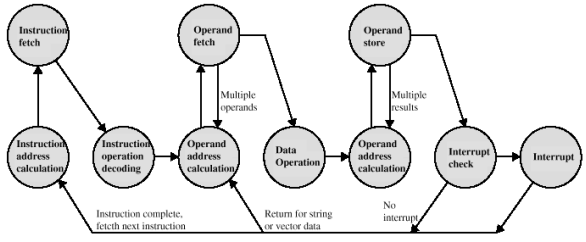
- Ci possono essere registri che puntano a:
  - Process control blocks (sistemi operativi)
  - Interrupt Vectors (sistemi operativi)
  - Tabella delle pagine della memoria virtuale
- La progettazione della CPU e quella dei sistemi operativi sono strettamente correlate

Esempi di Organizzazione di Registri



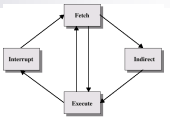
Ciclo esecutivo delle istruzioni: Fetch/Execute

- Lo avete già visto



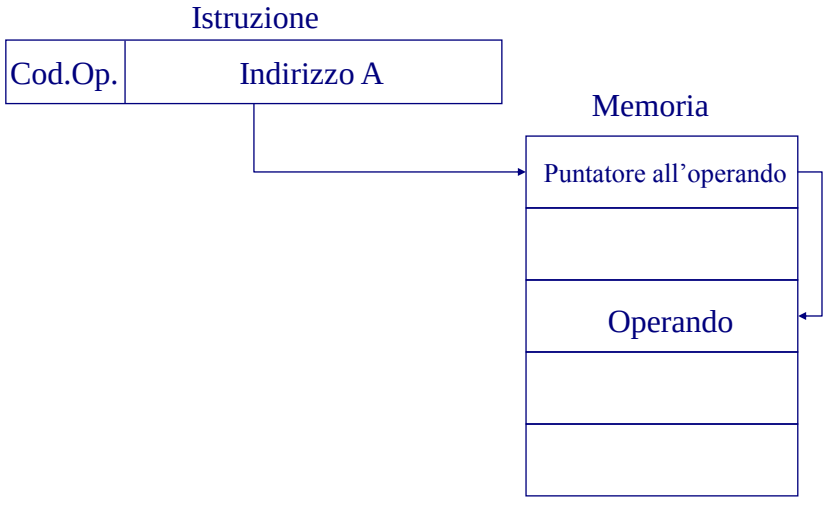
- Stallings, Capitolo 3
- Ne vediamo una versione revisionata

Indirettezza

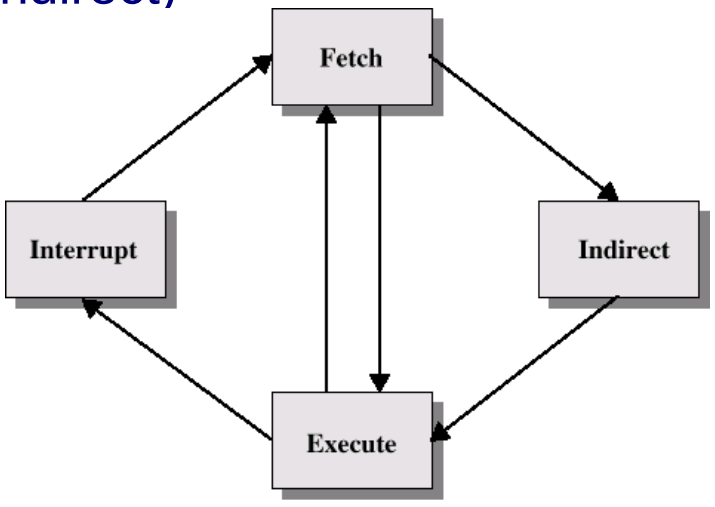


- Per recuperare gli operandi di una istruzione può essere necessario accedere alla memoria
- La modalità di indirizzamento indiretto per specificare la locazione in memoria degli operandi richiede più accessi in memoria
- L'indirettezza si può considerare come un sottociclo del ciclo fetch/execute

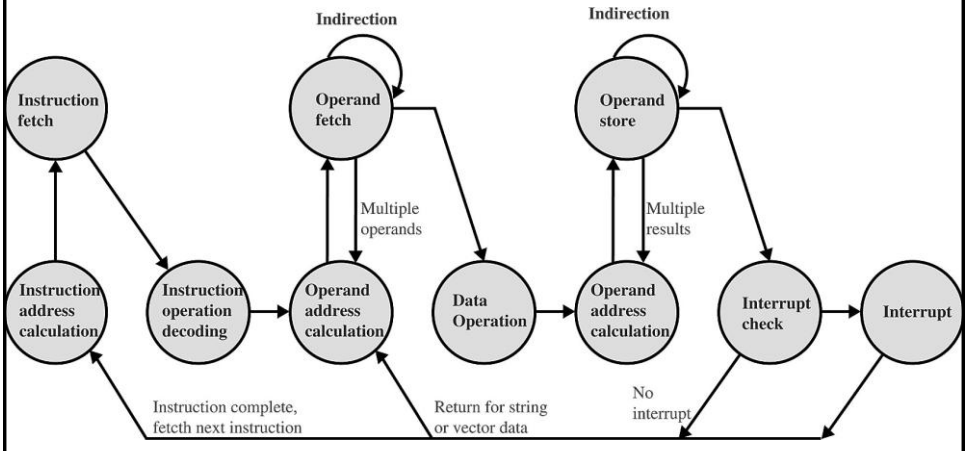
Indirizzamento indiretto



Introduzione della Indirettezza (indirect)



Fetch/Execute: ancora più in dettaglio

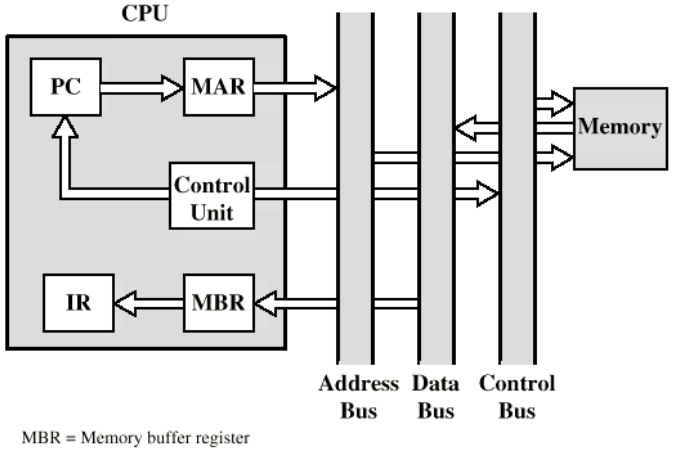


Flusso dei dati (Instruction Fetch)

Dipende dalla architettura della CPU, in generale:

- Fetch
  - PC contiene l'indirizzo della istruzione successiva
  - Tale indirizzo viene spostato in MAR
  - L'indirizzo viene emesso sul bus degli indirizzi
  - La unità di controllo richiede una lettura in memoria principale
  - Il risultato della lettura in memoria principale viene inviato nel bus dati, copiato in MBR, ed infine in IR
  - Contemporaneamente il PC viene incrementato

Flusso dei dati (Diagramma di Fetch)

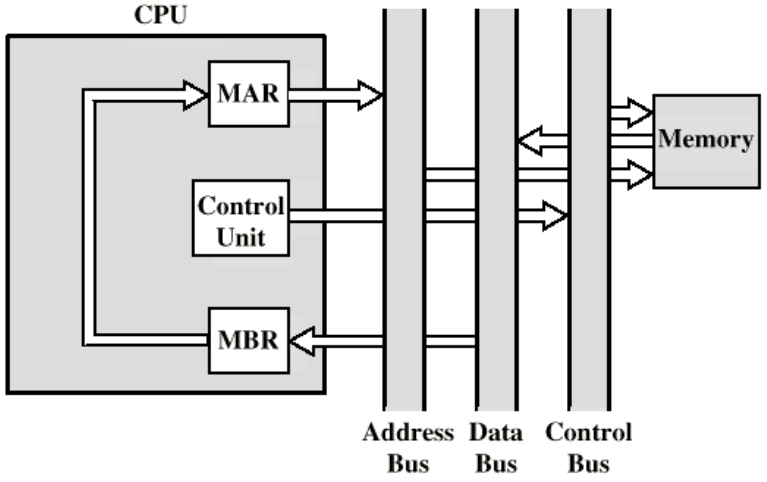


MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Flusso dei dati (Data Fetch)

- IR è esaminato
- Se il codice operativo della istruzione richiede un indirizzamento indiretto, si esegue il ciclo di indirettezza
  - gli N bit più a destra di MBR vengono trasferiti nel MAR
  - L'unità di controllo richiede la lettura dalla memoria principale
  - Il risultato della lettura (indirizzo dell'operando) viene trasferito in MBR

Flusso dei dati (Diagramma di Indirettezza)



Flusso dei dati (Execute)

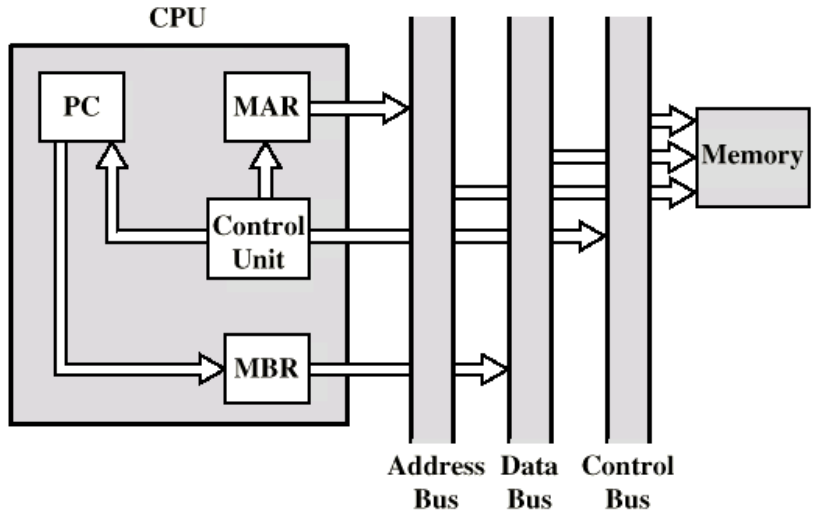
- Può assumere molte forme
- Dipende dalla istruzione da eseguire
- Può includere
  - lettura/scrittura della memoria
  - Input/Output
  - Trasferimento di dati fra registri e/o in registri
  - Operazioni della ALU

Flusso dei dati (Interrupt)

Semplice e prevedibile:

- Contenuto corrente del PC deve essere salvato per permettere il ripristino della esecuzione dopo la gestione dell'interruzione
  - Contenuto PC copiato in MBR
  - Indirizzo di locazione di memoria speciale (es. stack pointer) caricato in MAR
  - Contenuto di MBR scritto in memoria
- PC caricato con l'indirizzo della prima istruzione della routine di gestione della interruzione
- Fetch della istruzione puntata da PC

Flusso dei dati (Diagram. di Interrupt)



Prefetch

- La fase di prelievo della istruzione accede alla memoria principale
- La fase di esecuzione di solito non accede alla memoria principale
- Si può prelevare l'istruzione successiva durante l'esecuzione della istruzione corrente
- Questa operazione si chiama "instruction prefetch"

Prefetch

(a) Simplified view

Miglioramento delle prestazioni

- Il prefetch non raddoppia le prestazioni:
  - L'esecuzione di istruzioni jump o branch possono rendere vano il prefetch (perché ?)
  - La fase di prelievo è tipicamente più breve della fase di esecuzione
    - Prefetch di più istruzioni ?
- Aggiungere più fasi per migliorare le prestazioni

Il prefetch può essere inutile perché…

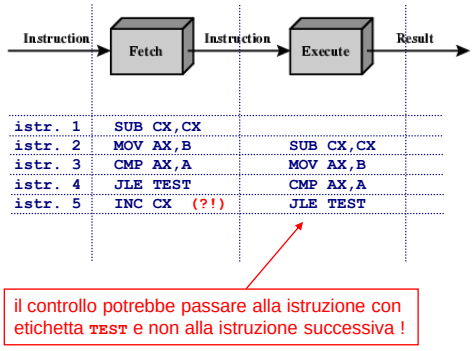
Per il seguente costrutto

```
if (A > B) then
```

un compilatore potrebbe generare il seguente codice 80x86

```
      SUB CX,CX; CX ← 0
      MOV AX,B ; AX ← mem[B]
      CMP AX,A ; paragona [AX] con mem[A]
      JLE TEST ; salta se A ≤ B
      INC CX   ; CX ← CX+1
TEST  JCXZ OUT ; salta se [CX] è 0
THEN

OUT
```



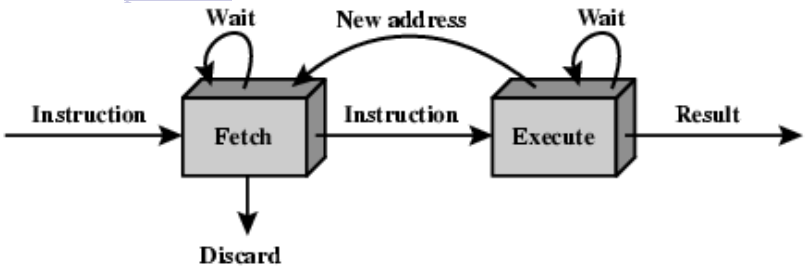
il controllo potrebbe passare alla istruzione con etichetta `TEST` e non alla istruzione successiva !

… si deve caricare una istruzione diversa dalla successiva !

Prefetch

(a) Simplified view

perché ?

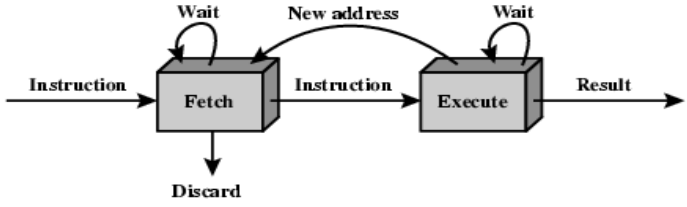


(b) Expanded view

perché…

su processore 286

	istruzione	cicli di clock impiegati dall'istr.
istr. 1	SUB CX,CX	2
istr. 2	MOV AX,B	5
istr. 3	CMP AX,A	6
istr. 4	JLE TEST	3 se non salta,>7 altrimenti
istr. 5	INC CX	2
istr. 6	JCXZ	4 se non salta,>8 altrimenti



(b) Expanded view