

Risposta alle domande da pag 139 a 142

1. E mnemonico il linguaggio macchina o quello assembly? Perché?
2. Da quali campi è formata un'istruzione assembly?
3. Quali sono e che cosa indicano i tre flag di controllo del registro di stato?
4. Perché le istruzioni jump condizionate devono essere precedute dall'istruzione compare?
5. A quale registro si appoggia l'istruzione loop per eseguire il numero di cicli previsti?
6. Che cosa fa la cpu dopo aver ricevuto un interrupt?
7. che cos'è uno stack?

1. È mnemonico il linguaggio macchina o quello assembly? Perché?

→ È **mnemonico** il **linguaggio assembly**, perché usa parole abbreviate (mnemonici) che rappresentano istruzioni, come MOV, ADD, SUB, ecc.

Il **linguaggio macchina** invece è fatto solo di **numeri binari** (0 e 1), che sono difficili da leggere per le persone.

2. Da quali campi è formata un'istruzione assembly?

→ Un'istruzione assembly è formata da:

- **Etichetta** (facoltativa): un nome per identificare una posizione nel codice
- **Operazione (mnemonico)**: il comando da eseguire, es. MOV
- **Operandi**: i dati su cui lavora l'istruzione, es. AX, BX
- **Commento** (facoltativo): spiegazione del codice, inizia con ;

Esempio:

```
start:  MOV AX, BX    ; copia il contenuto di BX in AX
```

3. Quali sono e che cosa indicano i tre flag di controllo del registro di stato?

ZF – Zero Flag

Vale 1 se il risultato è zero, altrimenti 0.

Esempio: CMP AX, AX → risultato = 0 → ZF = 1

SF – Sign Flag

Indica se il risultato è negativo (vale 1) o positivo (vale 0).

CF – Carry Flag

Vale 1 se c'è stato un riporto (carry) o un prestito (borrow) in un'operazione aritmetica.

Serve per l'aritmetica con numeri senza segno.

PF – Parity Flag

Vale 1 se il numero di bit a 1 nel risultato è pari, 0 se è dispari.

Usato in controlli di parità, ad esempio nelle comunicazioni.

AF – Auxiliary Carry Flag

Vale 1 se c'è stato un riporto dal bit 3 al bit 4 (utile per operazioni con numeri BCD – Binary-Coded Decimal).

È più tecnico e specifico, spesso usato in aritmetica su numeri decimali.

OF – Overflow Flag

Vale 1 se c'è stato un overflow: cioè se il risultato non entra nei bit previsti.

Serve per l'aritmetica con numeri con segno.

7. Che cos'è uno stack?

→ Lo **stack** è una **zona di memoria** usata per salvare dati temporanei, come:

- indirizzi
- variabili
- registri

Funziona come una pila:

- con PUSH **metti** un dato sopra

- con POP **togli** l'ultimo dato messo

Risposta alle domande da pag 147 a 153

1. Perché non ha senso scrivere MOV 55 CCh,AX?
2. Che tipo di indirizzamento utilizza l'istruzione MOV AX,[BX][SI]?
3. Che tipo di indirizzamento utilizza l'istruzione MOV 03[DI],AX?
4. Che cos'è il base address?
5. Che cos'è offset?
6. Che cos'è il displacement?
7. Che cos'è e come si calcola l'effective address?
8. Descrivi l'indirizzamento diretto da memoria a registro
9. Descrivi l'indirizzamento indiretto tramite registri base e indice e con displacement.

1. Perché non ha senso scrivere MOV 55CCh, AX?

Perché in Assembly **non puoi spostare dati verso un numero**.

55CCh è un **valore immediato (una costante)**, non un registro o una cella di memoria.

È come dire: "scrivi qualcosa dentro a un numero" – non ha senso.

✓ Corretto: MOV AX, 55CCh (mette il valore 55 CCh dentro AX)

✗ Sbagliato: MOV 55CCh, AX (non puoi mettere AX dentro un numero)

2. Che tipo di indirizzamento utilizza l'istruzione MOV AX, [BX][SI]?

Usa **indirizzamento indiretto con base e indice**.

👉 [BX][SI] significa: prendi il contenuto della **memoria all'indirizzo BX + SI**, e mettilo in AX.

3. Che tipo di indirizzamento utilizza l'istruzione MOV 03[DI], AX?

Questa è una scrittura che indica **indirizzamento indiretto con indice + displacement (scostamento)**.

👉 03[DI] = prendi il contenuto della **memoria a DI + 03h**, e metti AX lì.

Si chiama: **indirizzamento con indice e displacement**.

4. Che cos'è il base address?

È un **indirizzo di partenza** usato per calcolare dove andare a leggere o scrivere in memoria. Di solito è contenuto in registri come **BX** o **BP**.

5. Che cos'è l'offset?

L'**offset** è la **distanza** (in byte) tra l'inizio di un segmento e un indirizzo specifico dentro quel segmento.

Esempio: se il segmento inizia a 1000h e voglio andare alla cella 100Ah → offset = Ah.

6. Che cos'è il displacement?

È un **valore costante** che si **aggiunge** a un registro per trovare un indirizzo. È come dire: "vai a partire da DI, ma spostati avanti di 3".

Esempio: `MOV AX, [DI+03h]` → 03h è il **displacement**.

7. Che cos'è e come si calcola l'effective address?

L'**effective address** (indirizzo effettivo) è l'**indirizzo finale** in memoria che la CPU calcola usando registri e displacement.

Si calcola con:

 **Registro base + Registro indice + Displacement**

Esempio:

`MOV AX, [BX + SI + 04h]`

→ L'indirizzo effettivo = contenuto di BX + contenuto di SI + 4

8. Descrivi l'indirizzamento diretto da memoria a registro

In questo tipo:

- l'indirizzo è **scritto direttamente nel codice**
- e il dato viene preso da lì e messo nel registro.

Esempio:

```
MOV AX, [1234h]
```

→ Prende il contenuto della cella di memoria 1234h e lo mette in AX.

9. Indirizzamento indiretto tramite registri base e indice con displacement

 **Cos'è?**

È un tipo di indirizzamento **indiretto**, in cui la CPU **calcola l'indirizzo di memoria** sommando 3 elementi:

- **Un registro base** → ad esempio: BX o BP
- **Un registro indice** → ad esempio: SI o DI
- **Un valore costante (displacement)** → ad esempio: +04h

Quindi la formula è:

Indirizzo effettivo = Base + Indice + Displacement

 **Esempio pratico:**

asm

```
MOV AX, [BX + SI + 04h]
```

 Questo vuol dire:

1. Prendi il **contenuto del registro BX**
2. Aggiungi il **contenuto del registro SI**

3. Aggiungi **04h** (il displacement)
4. Vai in **quell'indirizzo di memoria**
5. Copia il **valore che c'è lì dentro** nel registro AX

DOMANDE FATTE DA ME

- **quali sono i processi di traduzione del codice assembly:**

La **traduzione del codice Assembly** in un programma eseguibile coinvolge diversi passaggi fondamentali. Anche se l'Assembly è un linguaggio già molto vicino al linguaggio macchina, c'è comunque un processo strutturato per arrivare a un file eseguibile funzionante.

Ecco i **principali processi di traduzione del codice Assembly**:

1. Scrittura del codice sorgente (Assembly)

- Il programmatore scrive il codice sorgente in Assembly, con estensione tipica `.asm` o `.s`.
-

2. Assemblamento (Assemblaggio)

- Il codice Assembly viene tradotto in **linguaggio macchina (object code)** tramite un **assembler**.
 - Output: un file oggetto (`.obj`), contenente codice binario ma **non ancora eseguibile**.
-

3. Linking (Collegamento)

- Il file oggetto viene passato a un **linker**, che:
 - Risolve riferimenti esterni (es. chiamate a librerie o funzioni definite altrove).
 - Unisce più file oggetto se necessario.
 - Aggiunge librerie standard o esterne richieste.
- Output: un **file eseguibile** (es. .exe su Windows, nessuna estensione su Linux).

- quali registri contiene un processore intel 8086?

✓ 1. Registro dei flag

- **FR**
 - Contiene i **bit di stato e controllo**: CF, ZF, SF, OF, IF, DF, TF, ecc.
-

✓ 2. Registri di lavoro (general)

- **AX, BX, CX, DX**
 - Usati per calcoli, trasferimenti di dati, moltiplicazioni, I/O, loop, ecc.
 - Si possono dividere in registri da 8 bit: AH/AL, BH/BL, ecc.
-

✓ 3. Registri indice

- **DI (Destination Index)**
- **SI (Source Index)**
 - Utilizzati soprattutto per operazioni su **stringhe** (es. MOVS, LODS, STOS, ecc).

✓ 4. Registri di segmento

- DS (Data Segment)
- ES (Extra Segment)
- CS (Code Segment)
- SS (Stack Segment)
 - Servono a indicare i **segmenti di memoria** attivi.

✓ 5. Registri puntatore



- IP (Instruction Pointer) → indica l'indirizzo della **prossima istruzione** da eseguire.
- BP (Base Pointer) → usato per accedere ai **parametri di funzione nello stack**.
- SP (Stack Pointer) → punta alla **cima dello stack**.

- quanti sono i flag del registro di stato dei processori intel x86?



sono 9 e sono usati per fotografare lo stato della cpu dopo l'esecuzione di un'istruzione

- quali sono i flag di controllo?



1. TF – Trap Flag

- **Serve per il debugging:** se attivo, il processore esegue **una sola istruzione alla volta**, poi **genera un'eccezione** (interruzione) chiamata "debug".
 - Utile per eseguire il programma **istruzione per istruzione**.
 -  Si attiva con: STI
 -  Si disattiva con: CLI
-

2. IF – Interrupt Flag

- Controlla se il processore può **rispondere alle interruzioni hardware**.
 - Se **IF = 1**, il processore accetta le interruzioni.
 - Se **IF = 0**, **ignora le interruzioni** (utile in alcune operazioni critiche).
 -  Si attiva con: STI (Set Interrupt)
 -  Si disattiva con: CLI (Clear Interrupt)
-

↔ 3. DF – Direction Flag

- Determina **in che direzione** si muovono gli indirizzi in **operazioni su stringhe** (MOVS, LODS, STOS, ecc).
- Se **DF = 0** → l'indirizzo aumenta (scorre **avanti** nella memoria).
- Se **DF = 1** → l'indirizzo diminuisce (scorre **indietro** nella memoria).
-  Si imposta con: STD (Set Direction Flag)
-  Si azzera con: CLD (Clear Direction Flag)

Risposte kahoot

- **Nella modalità di indirizzamento cosa si intende per OFFSET ?**

distanza da sommare all'indirizzo base

- **Cosa si intende per indirizzo di base?**

Indirizzo della cella iniziale del blocco di codice da elaborare

- **Indica qual e l'istruzione corretta**

Destinazione Mov origine

- **Cosa si intende per spostamento**

ulteriore Spostamento rispetto all'indirizzo base e offset

- **i metodi di indirizzamento dell i8086 posso essere**

Immediato,Diretto,Indiretto

- **Nell'indirizzamento diretto da registrazione a registrazione l'unico bus coinvolto e il bus dati**

falso

- **Nell'indirizzamento immediato, l'istruzione Mov 4Dh, AX e corretta**

falso

- **Indica l'istruzione rappresentata dalla seguente immagine**

Mov AX, 03 [SI]

- **L'istruzione Mov AX, 02[BX][SI] coinvolge anche il bus dati**

Vero

- **Nell'indirizzamento indiretto, quando sono coinvolti anche i registri SI e DI e necessario eseguire anche**

una somma

- **nell'istruzione Mov AX,BX non e coinvolta a memoria centrale**

Vero

- **L'istruzione Mov AX[5Bh] e valida anche se si invertono gli operandi**

Risposta alle domande pag 157

1. Quali sono i registri fondamentali della CPU?

I registri fondamentali, presenti in tutte le CPU (pur con nomi differenti a seconda dell'azienda produttrice di microprocessori), sono

- AR (Address Register): memorizza gli indirizzi per gli accessi in memoria
- DR (Data Register): memorizza i dati provenienti dalla memoria diretti alla CPU e dalla CPU diretti alla memoria.
- IR (Instruction Register): memorizza il codice operativo (opcode) dell'istruzione da eseguire.
- PC (Program Counter): memorizza l'indirizzo della prossima istruzione da eseguire.
- SR (Status Register): memorizza, tramite una serie di flag, lo stato del processore, successivo all'esecuzione dell'ultima operazione
- SP (Stack Pointer): memorizza l'indirizzo top dello stack. Lo stack è un'area della memoria in cui i dati sono letti/scritti in modalità Last-In-First-Out (LIFO).
- R0, ..., Rn: registri di lavoro che memorizzano i risultati temporanei in ingresso e in uscita dall'ALU.

2. Descrivi nel dettaglio cosa succede nella fase ID (Instruction Decode) del ciclo macchina.

L'opcode dell'istruzione è decodificato e in base a esso si caricano nei registri di lavoro (R0, ..., Rn) gli operandi necessari all'istruzione e si attivano le microcircuiterie necessarie a svolgere l'operazione richiesta.

3. Quali sono i principali problemi di gestione delle pipeline?

Questa tecnica funziona molto bene se non vi sono legami troppo stretti tra due istruzioni; in particolare se a un'istruzione serve il risultato di un'istruzione precedente, essa non potrà entrare nella fase EX fino a quando l'istruzione precedente non è giunta alla fase WB. Un altro problema che si presenta è quello conseguente ai cosiddetti salti di esecuzione: blocchi di istruzioni non sono eseguite se non sono verificate determinate condizioni; in questo caso il microprocessore passa a eseguire istruzioni memorizzate "distanti" dalle precedenti con la necessità di svuotare la pipeline prima di eseguire il salto, con conseguente rallentamento dell'esecuzione.

4. Il processo che traduce il codice assembly nel corrispondente codice macchina eseguibile direttamente dal microprocessore, si compone di due passaggi. Descrivili.

Il processo che traduce il codice assembly nel corrispondente codice macchina eseguibile direttamente dal microprocessore si compone di due passaggi:

- **ASSEMBLER**: è un processo di traduzione che non richiede alcuna intelligenza; esiste il processo inverso, detto disassemblaggio o disassembly;
- **LINKER**: serve a collegare moduli e librerie di cui si compone il programma e a distribuire il codice oggetto nello spazio di indirizzi di memoria centrale assegnato al programma.

5. Quali sono e cosa indicano i 6 flag di stato del registro di stato?

I flag di stato sono:

- **C (Carry)**: è settato a 1 in presenza di riporto (carry) o prestito (borrow) a seguito di operazioni aritmetiche nell'ALU, altrimenti è resettato a 0;
- **P (Parity)**: è il bit di parità; in parità pari è settato a 1 per avere un numero pari di bit a 1 nel dato, viceversa in parità dispari; serve per la rilevazione degli errori di trasmissione;
- **A (Auxiliary)**: è usato per carry e borrow da un nibble (pari a un semibyte, cioè 4 bit) a quello immediatamente successivo;
- **Z (Zero)**: è settato a 1 quando il risultato di un'operazione aritmetica è 0, altrimenti è resettato 0;
- **S (Sign)**: è settato a 1 quando il risultato di un'operazione aritmetica è negativo (in complemento a 2 il bit segno vale 1 per i numeri negativi), altrimenti è resettato 0;
- **O (Overflow)**: settato a 1 segnala un "traboccamento", cioè segnala se il risultato di un'operazione aritmetica tra interi produce un risultato che va a occupare il bit dedicato al segno del numero stesso.

6. Descrivi il funzionamento dell'istruzione LOOP.

L'Istruzione LOOP serve per l'esecuzione e il controllo dei cicli. Ve ne sono di tre tipi:

- **LOOP**: esegue il ciclo se CX è diverso da 0;
- **LOOPE**: preceduta dalla CMP, esegue il ciclo se CX è diverso da 0 e il flag Z = 0;
- **LOOPNE**: preceduta dalla CMP, esegue il ciclo se CX è diverso da 0 e il flag Z = 1.

7. Come si dichiara una subroutine?

Le subroutine (procedure) si dichiarano nel formato Nome_sub PROC NEAR, seguito dalle istruzioni che devono terminare con l'istruzione RET (per il ritorno alla subroutine o

al main chiamante), seguita dalla parola chiave endp (end procedure); le PROC NEAR si scrivono in fondo al code segment. Se si lavora a moduli (su più file .asm) e la subroutine chiamata non è nello stesso file del chiamante, occorre dichiarare la subroutine come: Nome_sub PROC FAR.

8.I metodi indirizzamento si suddividono in tre tipi. Quali?

I metodi di indirizzamento si suddividono in tre tipi, in base al modo in cui l'istruzione tratta il dato da elaborare: • immediato: nell'istruzione c'è il dato; • diretto: nell'istruzione c'è l'indirizzo di memoria o il registro in cui si trova il dato; • indiretto: nell'istruzione c'è l'indirizzo di memoria o il registro dove si trova l'indirizzo di memoria in cui c'è il dato.

9.Che differenza c'è tra indirizzamento immediato e indirizzamento diretto?

L'indirizzamento immediato coinvolge un solo registro e non consente l'inversione degli operandi. Invece l'indirizzamento diretto coinvolge uno o due registri o una locazione di memoria.