

## Assembler

Quando si installa un programma si creano dei file che sono memorizzati nell'hard-disk del PC. Quando si "lancia" il programma parte di questi file è copiata nella RAM (memoria di lavoro) e da questo momento la CPU può eseguire le istruzioni del programma.

Per capire meglio cos'è esecuzione di una serie di istruzioni in una shell MSDOS lanciamo il programma debug che ci mette a disposizione un emulatore di PC con gli strumenti per analizzarne il funzionamento.

Il programma ha solo tre istruzioni, la terza altera la prima. In assembler la CPU fa esattamente quello che le istruzioni indicano, anche modificare una parte del programma stesso.

```
C:\Users\claudio>debug
-a 100
17C6:0100 mov ax,1234
17C6:0103 mov bx,0100
17C6:0106 mov [bx],ax
17C6:0108
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17C6 ES=17C6 SS=17C6 CS=17C6 IP=0100 NV UP EI PL NZ NA PO NC
17C6:0100 B83412 MOV AX,1234
-t=100,1

AX=1234 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17C6 ES=17C6 SS=17C6 CS=17C6 IP=0103 NV UP EI PL NZ NA PO NC
17C6:0103 BB0001 MOV BX,0100
-t 1

AX=1234 BX=0100 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17C6 ES=17C6 SS=17C6 CS=17C6 IP=0106 NV UP EI PL NZ NA PO NC
17C6:0106 8907 MOV [BX],AX DS:0100=34B8
-d 100,107
17C6:0100 B8 34 12 BB 00 01 89 07 .4.....
-t 1

AX=1234 BX=0100 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17C6 ES=17C6 SS=17C6 CS=17C6 IP=0108 NV UP EI PL NZ NA PO NC
17C6:0108 0000 ADD [BX+SI],AL DS:0100=34
-d 100,107
17C6:0100 34 12 12 BB 00 01 89 07 4.....
-q
C:\Users\claudio>
```

In verde sono evidenziati i comandi del debug

```
a 100
```

Attiva la scrittura di istruzioni assembler nella locazione 0100.

```
17C6:0100 mov ax,1234
```

Sposta il valore 1234 nel registro AX

```
17C6:0103 mov bx,0100
```

Sposta il valore 0100 nel registro BX

```
17C6:0106 mov [bx],ax
```

Sposta il valore nel registro AX nella locazione puntata da BX

```
-r
```

Visualizza i registri della CPU,  
AX contiene 0000,

BX contiene 0000,  
IP contiene 0100 che punta all'istruzione MOV AX,1234.

-t=100, 1

Esegue un'istruzione contenuta a partire dalla locazione 0100 e successivamente visualizza lo stato dei registri della CPU,

AX contiene 1234,

IP contiene 0103 e punta all'istruzione MOV BX,0100

-t 1

Esegue un'istruzione contenuta a partire dalla locazione puntata da IP e successivamente visualizza lo stato dei registri della CPU,

BX contiene 0100,

IP contiene 0106 e punta all'istruzione MOV [BX],AX

-t 1

Esegue un'istruzione contenuta a partire dalla locazione puntata da IP e successivamente visualizza lo stato dei registri della CPU,

la locazione 0100 contiene 34,

la locazione 0101 contiene 12.

Il programma di tre istruzioni occupa 8 byte. Ogni istruzione è suddivisa in codice operativo e dato. Il codice operativo è sempre presente, il dato può non esserci.

indirizzo	valore	informazione	istruzione	descrizione
0100	B8	Codice operativo	MOV AX,1234	Indirizzamento immediato il valore 1234 è spostato nel registro AX
0101	34	Dato		
0102	12	Dato		
0103	BB	Codice operativo	MOV BX,0100	Indirizzamento immediato il valore 0100 è spostato nel registro BX
0104	00	Dato		
0105	01	Dato		
0106	89	Codice operativo	MOV [BX],AX	Indirizzamento indiretto AX è spostato nelle locazioni puntate da BX
0107	07	Codice operativo		

Con il termine **fetch** si intende l'operazione di caricamento dell'istruzione dalla RAM alla CPU che poi la decodifica e la esegue nella fase di **execute**.

Si può suddividere l'esecuzione delle tre istruzioni in una serie di letture e scritture in RAM.

address	AX	BX	IP	data	Registro istruzione	fase	R/W	0100	0101
0100	0000	0000	0100	B8	B8	fetch	R	B8	34
0101	0000	0000	0101	34	B8 34	fetch	R	B8	34
0102	0000	0000	0102	12	B8 34 12	fetch	R	B8	34
	1234	0000	0103		B8 34 12	execute		B8	34
0103	1234	0000	0103	BB	BB	fetch	R	B8	34
0104	1234	0000	0104	00	BB 00	fetch	R	B8	34
0105	1234	0000	0105	01	BB 00 01	fetch	R	B8	34
	1234	0100	0106		BB 00 01	execute		B8	34
0106	1234	0100	0106	89	89	fetch	R	B8	34
0107	1234	0100	0107	07	89 07	fetch	R	B8	34
0100	1234	0100	0108	34	89 07	execute	W	34	34
0101	1234	0100	0108	12	89 07	execute	W	34	12

Questa descrizione è semplificata rispetto al funzionamento di una CPU attuale, si tratta della tecnologia usata nelle CPU dei primi anni '80. Per aumentare la velocità di esecuzione sono state introdotte diverse tecniche.

**Prefetch.** Mentre è ancora in esecuzione l'istruzione si procede al fetch dell'istruzione successiva.

**Branch prediction.** Aumentando la dimensione della memoria dedicata al prefetch si pone il problema di come gestire le istruzioni condizionali. Sono stati studiati sofisticati algoritmi per gestire al meglio queste situazioni.

**Dual channel.** Dopo aver aumentato la dimensione del bus da 8 a 16 e 32 bit il data bus per velocizzare il sistema si sono affiancati due moduli per ottenere un bus a 64 bit.

I **compilatori** (C, C++, Pascal, Basic) trasformano un file che contiene il codice sorgente in un file che contiene il codice macchina che è una serie di istruzioni tipo quelle viste in precedenza. Il codice macchina è il sistema più efficiente per sfruttare la potenza del PC. L'inconveniente principale è l'accesso diretto della CPU alla memoria (Nell'esempio precedente il programma altera se stesso). Altro inconveniente è il fatto che al cambio di ogni CPU si deve procedere all'ottimizzazione del codice macchina con una nuova compilazione.

Gli **interpreti** (Java, .net) partendo dal codice sorgente non generano codice macchina ma un codice intermedio identico per tutte le piattaforme. Ogni piattaforma ha una propria versione dell'interprete in grado di eseguire il codice intermedio senza nessuna modifica. L'inserimento di un livello ulteriore consente una gestione più sofisticata della sicurezza evitando che processi distinti interferiscano tra di loro.