SISTEMI E RETI: IL LINGUAGGIO MACCHINA

FONTE: HTTPS://WWW.EDUTECNICA.IT/

LINGUAGGIO MACCHINA

La quasi totalità dei microprocessori esistenti attualmente sul mercato, sono stati realizzati per essere il nucleo di un sistema di calcolo che implementa una macchina di Von Neumann.

I microprocessori sono sistemi che operano su programmi sequenziali, nei quali ogni istruzione è caratterizzata da un indirizzo di memoria, mentre il flusso delle istruzioni è guidato da un puntatore (contatore di istruzioni) che percorre lo spazio di questi indirizzi di memoria secondo delle regole imposte dal programma stesso.

Questa architettura è denominata come sistema a singola istruzione e a singolo flusso di dati SISD (Single Instruction stream- Single Data stream); il sistema elabora le istruzioni una alla volta nell'ordine in cui sono scritte, a meno che l'istruzione stessa non imponga di modificare la sequenza di esecuzione. Ogni singola istruzione viene eseguita in due fasi:

- Fetch: lettura dell'istruzione dalla memoria.
- Execute : interpretazione ed esecuzione effettiva dell'istruzione.

In quest'ordine di cose, i parametri che differenziano tra loro i microprocessori sono:

1. Set di istruzioni
2. Modalità di indirizzamento degli operandi.
3. Numero di indirizzi.
4. Ortogonalità.
5. Organizzazione dello spazio degli indirizzi.
6. Parallelismo del bus dati
7. Bus degli indirizzi
8. Ordine di memorizzazione delle informazioni.
9. Frequenza di lavoro.
10. Gestione dell'input/output.
11. Metodi di esecuzione delle istruzioni.

1 Set di istruzioni

Ogni microprocessore possiede un set di istruzioni logiche fondamentali (AND OR NOT) ed almeno delle istruzioni aritmetiche di somma (ADD) e di sottrazione (SUB). Le altre, meno fondamentali possono essere o non essere presenti, dato che la moltiplicazione (MUL) può essere considerata una somma ripetuta e la divisione (DIV) una sottrazione ripetuta. Maggiore è il set di istruzioni minore è l'insieme di istruzioni che devono essere scritte manualmente dal programmatore. Bisogna tener conto che in linea generale:

Quello che viene realizzato via hardware è più veloce della stessa cosa realizzata via software a parità di condizioni.

2 Modalità di indirizzamento degli operandi

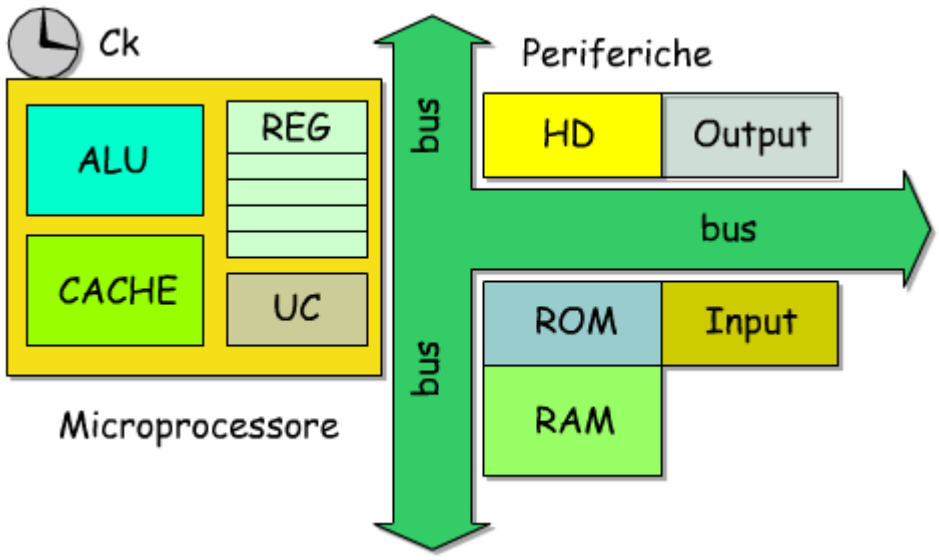
Ogni istruzione è costituita fondamentalmente da due elementi: il codice operativo e l'operando. In informatica si possono usare diversi modi per assegnare il valore 3 ad una variabile (indirizzamento) ad es.

- **`A=3`** o **`B=3`** indirizzamento immediato cioè l'istruzione stessa contiene il dato
- **`A=B`** indirizzamento diretto, si specifica che il valore si trova all'indirizzo di memoria B
- **`A[i]=3`** indirizzamento indicizzato (vettore) etc..

Bisogna sempre conoscere i modi di indirizzamento di un µP per sapere quali tipi di dato si possono implementare direttamente e quali invece si devono simulare. Ad esempio se un µP non permette il metodo indicizzato non è possibile realizzare algoritmi in cui siano presenti dei vettori.

3 Numero di indirizzi

Noi, usiamo descrivere l'architettura di un elaboratore in modo semplice, attraverso il seguente disegno, dove si riconosce la presenza del microprocessore (CPU=Central Processing Unit). Tutto quello che non è il microprocessore è considerato una periferica collegata alla CPU da delle linee chiamate BUS.



Nella CPU si evidenzia la ALU (arithmetic Logic Unit) che serve ad eseguire operazioni aritmetiche, prevalentemente somme. Per eseguire un'operazione aritmetica ci vogliono degli operandi sui quali attuare l'operazione. La sigla REG che si vede indica i registri interni

al processore che sono delle locazioni di memoria dove vengono inseriti gli operandi sui quali eseguire le operazioni.

In genere il numero di operandi per eseguire una operazione è al massimo tre, se consideriamo una tipica operazione di somma C=A+B; partendo da questo assunto possiamo classificare i microprocessori in classi a:

3 indirizzi: consentono al programmatore di decidere tutti e tre gli indirizzi, cioè si può scrivere **`ADD C, A, B`** cioè C=A+B
2 indirizzi: consentono al programmatore di decidere due indirizzi mentre il terzo viene stabilito dal microprocessore, cioè si può scrivere **`ADD B, A`** che esplicitato significa B=B+A. In questo caso si possono specificare i due addendi mentre la variabile che deve contenere il risultato viene imposta dal µP; in questo caso B significa sia l'addendo prima dell'operazione che la somma dopo l'operazione.
1 indirizzo: consente al programmatore di scegliere un solo operando mentre gli altri due vengono imposti dal microprocessore, ad es. **`ADD B`** che significa A=A+B; solo la variabile B viene scelta dal programmatore, mentre l'altro operando, in questo caso il registro accumulatore A, così come il risultato, sono impliciti nel codice operativo.
0 indirizzi: sono processori che permettono di scrivere solo il codice operativo mentre gli operandi vengono imposti; è il caso delle stack machine, architetture nelle quali la memoria viene vista come una pila (stack) e si può accedere solo ai dati che stanno in cima alla catasta. Ad es. **`ADD`** il µP prende i primi due dati in cima allo stack, ne calcola la somma e rimette il risultato in cima allo stack.

4 Ortogonalità

Un microprocessore si dice ortogonale quando tutti i registri sono in grado di fornire gli stessi servizi utilizzabili in qualsiasi istruzione; in caso contrario il microprocessore si dice non ortogonale.

La quasi totalità dei microprocessori esistenti in commercio non sono ortogonali; per cui occorre conoscere preventivamente quali sono i registri che permettono di eseguire determinate istruzioni e quali no.

5 Organizzazione dello spazio degli indirizzi

È necessario conoscere l'organizzazione della memoria dal punto di vista logico che possono essere di due tipi

- lineare
- segmentata.

Nell'organizzazione lineare la memoria è unidimensionale, cioè lo spazio degli indirizzi è organizzato come un'unica sequenza di locazioni, ciascuna dotata di un indirizzo. Per indirizzare le locazioni di memoria è sufficiente un solo registro (Program Counter) e quindi quando si scrivono le istruzioni dobbiamo indicare un solo argomento per individuare una locazione. Tradizionalmente i microprocessori della famiglia Motorola utilizzavano (almeno inizialmente) questo tipo di organizzazione.

Nell'organizzazione segmentata il programmatore definisce i segmenti e al microprocessore vanno forniti due parametri per referenziare la locazione di memoria: il segmento e la posizione relativa al segmento (offset). Il processore deve essere dotato di un registro per il segmento e un registro per il displacement (spiazzamento: distanza dall'indirizzo base del segmento). I microprocessori della famiglia Intel/AMD adottano questo tipo di organizzazione.

6 Parallelismo del bus dati

Con il termine parallelismo si indica la dimensione in bit degli operandi che i microprocessori possono elaborare nelle loro operazioni. In un microprocessore ad 8 bit gli operandi hanno la dimensione di un byte (8 bit) In un microprocessore a 64 bit si possono gestire operandi che hanno la dimensione di 8 byte.

7 Bus degli indirizzi

In del famiglio generale qualsiasi elaboratore è dotato di tre tipi di bus:

- bus dei dati
- bus degli indirizzi
- bus di controllo

anche se esistono elaboratori che usano le stesse linee di trasmissione sia per i dati che per gli indirizzi. La dimensione del bus degli indirizzi è molto importante perché definisce la quantità di memoria fisica che il microprocessore può gestire. Un µP a 16 bit può indirizzare 2^{16} byte di memoria un processore con n bit indirizza 2^n byte di memoria.

8 Ordine di memorizzazione delle informazioni

Nelle operazioni computazionali si può avere a che fare con diversi tipi di dati aventi dimensioni diverse (interi, reali a virgola fissa, reali a virgola mobile etc..) che devono essere rappresentati in memoria all'interno di locazioni la cui dimensione è un multiplo di un byte. Noi dobbiamo tener presente che dal punto di vista fisico, la memoria è una sequenza di locazioni da un byte; quindi se un dato occupa 16bit (2 byte) nelle due aree adiacenti assegnate vengono prima memorizzati gli 8 bit più significativi e poi gli 8 bit meno significativi o viceversa. I processori Intel usano questa convenzione, i processori Motorola ad es. usano la convenzione inversa.

9 Frequenza di lavoro

Quando andiamo ad acquistare un elaboratore elettronico o anche un dispositivo mobile, uno dei primi parametri che viene fornito dal commerciante è la frequenza di lavoro (del clock) la cui velocità viene espressa in GHz (10^9 Hz). Questo valore esprime il numero di cicli macchina eseguiti dal microprocessore in un secondo. Per eseguire una operazione, vengono impiegati alcuni cicli macchina: la durata (periodo) di un ciclo macchina è l'inverso della frequenza. La frequenza di clock non è il parametro decisivo per stabilire se un sistema di calcolo è più veloce di un altro perché ci sono altre caratteristiche del sistema che possono influenzarne l'efficienza come la RAM disponibile o la velocità di rotazione del disco con il suo tempo medio di accesso.

10 Gestione dell'input/output

Ogni elaboratore si interfaccia col mondo esterno tramite le periferiche di input e output ed ogni interfaccia può essere vista dal punto di vista software come un gruppo di tre indirizzi:

- uno per la parte relativa ai registri di controllo
- uno relativo ai registri di comando
- uno relativo al porto (port) dell'interfaccia.

Nei microprocessori Intel è sempre stata adottata una filosofia di I/O isolato con un set di istruzioni specifico per l'input-output. Nei dispositivi Motorola si è sempre adottato un sistema I/O mappato, dove non c'è un set di istruzioni specifico dedicato ma si usano le stesse istruzioni adoperate per la lettura/scrittura in memoria. Quest'ultima politica è considerata, ovviamente, più versatile.

11 Metodi di esecuzione delle istruzioni

Tutti i microprocessori implementano la logica di Von Neumann ma ognuno può adottare accorgimenti specifici per migliorare le prestazioni come le seguenti:

- Prefetch. In questo caso il microprocessore viene interpretato come un sistema costituito da due sezioni: la sezione esecutiva e la sezione di decodifica, controllo e temporizzazione. Dato che la sezione esecutiva non è interessata da operazioni di fetching, si organizza il lavoro del µP in modo da eseguire la fase di fetching dell'istruzione successiva mentre si è in fase di esecuzione dell'istruzione in corso. I processori Intel adottano questa filosofia.

- Pipeline. In questo caso la fase esecutiva viene suddivisa in sottofasi, come in una catena di montaggio (pipeline): si hanno simultaneamente diverse fasi di esecuzione di diverse istruzioni, eliminando in questo modo i tempi delle fasi intermedie.

- Memoria cache. Per velocizzare le operazioni, tutti i sistemi sono dotati di una memoria tampone (cache) dove vengono immagazzinati dati ed istruzioni eseguiti più frequentemente e più recentemente. In questo modo il sistema è in grado di anticipare i dati prima che il microprocessore li debba chiedere.

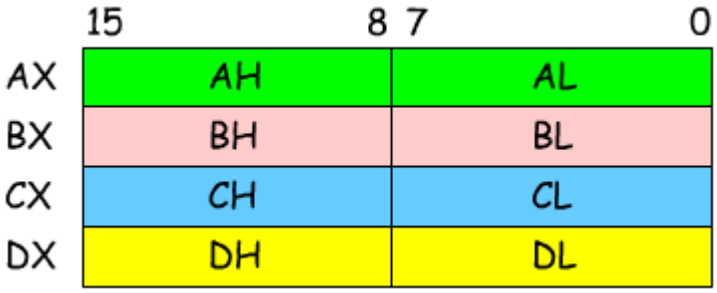
MICROPROCESSORE 8088/86

L'importanza di questo microprocessore della Intel è dovuta all'introduzione e alla successiva grande diffusione dei personal computer basati su sistema operativo MS-DOS che si è avuta negli anni '80, in Italia ben rappresentati dall' "iconico" elaboratore Olivetti M24 con parallelismo dati di 16 bit e con ben 20 bit per il bus degli indirizzi.

Si tratta di un processore non ortogonale dotato di 14 registri raggruppabili per tipologia.

Registri accumulatori

I registri accumulatori (general purpose=di utilizzo generico) sono 4 registri a 16 bit denominati AX, BX, CX, DX.



Con le sigle AX, BX, CX, DX si referenziano tutti i 16 bit di ciascun registro. Indicando AH, BH, CH, DH vengono indirizzati gli 8 bit più significativi dei registri suddetti. Indicando AL, BL, CL, DL vengono indirizzati gli 8 bit meno significativi dei registri suddetti.

Il registro AX, oltre che come accumulatore, viene coinvolto automaticamente in altre operazioni, come la moltiplicazione, la divisione e le operazioni di I/O.

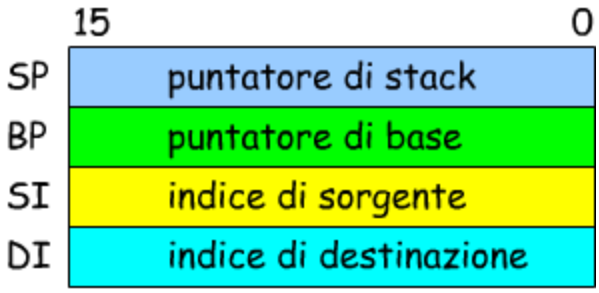
Il registro BX viene usato come indice di un vettore o come base di una tabella.

Il registro CX viene usato anche come contatore a 16 bit in particolari istruzioni come i LOOP.

Il registro DX viene anche automaticamente usato nelle operazioni di moltiplicazione per contenere la parte più significativa di un prodotto o nelle divisioni per memorizzare il resto. Nei sistemi MS-DOS è coinvolto nelle operazioni di I/O contenendo l'indirizzo del punto di ingresso o di uscita di un dato per una periferica.

Registri Puntatori e Indice

Esistono 4 registri a 16 bit, chiamati registri indice usati per organizzare vettori e strutture record e per gestire le procedure dello stack. In questi registri si possono referenziare solo tutti e 16 i bit disponibili.



Il registro SP indica le prime locazioni disponibili nello stack.

Il registro BP permette di organizzare lo stack come se fosse un vettore per la gestione dei parametri quando si usano le procedure (PROC).

I registri SI e DI sono usati per implementare vettori; vengono poi automaticamente coinvolti nelle operazioni sulle stringhe: SI indica la stringa sorgente, DI la stringa di destinazione.

Registri di segmento e puntatore di istruzione

Come abbiamo detto il microprocessore Intel 8086 vede la memoria in modo segmentato, esso può referenziare una locazione di memoria tramite due valori: il segmento e la posizione relativa al segmento (offset displacement).

L'indirizzo del segmento è contenuto in uno dei registri di segmento relativo all'area che si sta referenziando, più di preciso:

Il registro CS (Code Segment) che indica la base del segmento di memoria in cui è stato allocato il codice del programma.

Il registro DS (Data Segment) indica la base del segmento in cui sono stati allocati i dati.

Il registro SS (Stack Segment) che indica la base del segmento definito come area di stack, utile per implementare procedure con passaggio di parametri.

Al contenuto del registro di segmento bisogna aggiungere:

- l'offset che viene fornito dal registro IP (Istruction Pointer), se si deve referenziare un'istruzione (come nei salti condizionali) oppure dalla variabile che inseriamo nell'istruzione, se viene referenziato un dato.

- il registro SP (Stack Pointer) se usiamo istruzioni che si riferiscono allo stack.

Il registro ES (Extra Segment) è un segmento extra per i dati, quindi possiamo accedere a due segmenti per dati. Nella pratica ES viene usato nelle istruzioni che anno riferimento alle stringhe e spesso referenzia la stessa area dati definita in DS.

I microprocessori della classe 8088/8086 eseguono le istruzioni in modo sequenziale una alla volta nell'ordine scritto dalla prima all'ultima a meno che l'istruzione stessa non imponga l'esecuzione di un'istruzione che si trova in una posizione diversa da quella dell'istruzione successiva (come avviene nel caso dei salti condizionali).

La coppia di registri che permettono la sequenzializzazione è data da **`CS:IP`**.

La memoria fisica di un sistema di calcolo può essere pensata come un array monodimensionale, per cui il processore deve trasformare un indirizzo bidimensionale in uno lineare, cioè deve calcolare l'indirizzo effettivo EA (effective address). Il calcolo viene fatto attraverso la seguente formula:

`EA=Segmento×16+Offset`

questa formula è adottata nel processore 8086 che è dotato di un bus degli indirizzi di 20bit cioè 2^{20}=1.048.576 byte= 1 MB.

Il segmento lungo 16bit, moltiplicato per 16 viene trasformato in 20 bit aggiungendo come parte meno significativa 4 bit di valore 0 e sommando quindi il relativo offset, anch'esso di 16 bit.

Utilizzando l'aritmetica in esadecimale questa operazione viene eseguita facilmente:

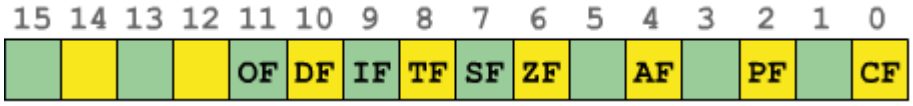
Segmento=FFF0
Offset=EEE0

applicando la formula si ha

EA=FFF00+EEE0=10EDE0

Registro dei flag

Il registro dei flag indicato anche come registro di stato è un registro a 16 bit contenente 9 bit (flag), che vengono usati per indicare, in modo indipendente, differenti condizioni durante l'esecuzione di un programma.



i bit 0, 2, 4, 6, 7, 11 contengono flag di stato che denotano risultati di operazioni di programma; i bit 8, 9, 10 contengono flag di controllo; gli altri non vengono utilizzati.

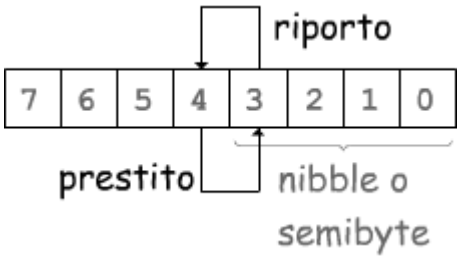
I flag possono essere analizzati dopo aver fatto eseguire opportune istruzioni. il loro stato (set = 1 o reset = 0) condiziona le istruzioni di salto condizionato.

Flag di stato

- Flag di Carry (CF): Questo flag viene impostato a 1 (settato) se dall'esecuzione di un'istruzione di addizione, per esempio, risulta un riporto (carry), o se in una sottrazione risulta un prestito (borrow). in caso contrario il flag è resettato. questo è significativo se le due operazioni di addizione e sottrazione sono applicati a dati che rappresentano numeri naturali.

- Flag di Parità (PF): Questo flag è impostato a 1 se il peso (somma di bit di valore 1) di un dato è pari, resettato quando il peso è dispari. questo flag era utilizzato nei processori precedenti alla classe x86, nei quali l'input e l'output veniva fatto direttamente dal microprocessore per controllare gli eventuali errori di trasmissione e ricezione. Nella comunicazione seriale asincrona il codice a controllo di peso è usato come codice di rilevazione degli errori: questo codice è noto come codice di controllo parità.

- Flag di Carry ausiliario (AF): Opera in modo analogo al flag di carry, ma viene usato per indicare un riporto dal bit 3 al bit 4 in una somma a 8 bit o un prestito dal bit 4 al bit 3 in una sottrazione a 8 bit.



praticamente, il flag viene settato quando in una somma ce un rapporto da un nibble (4 bit o semi byte) a quello immediatamente più significativo, oppure quando, in una sottrazione, si ha un prestito da un semibyte a quello immediatamente meno significativo.

●Flag di Zero (ZF): Il flag di 0 viene settato quando il risultato di un'operazione ha valore 0. questo può accadere, per esempio, dopo l'esecuzione di un'istruzione di sottrazione o di decremento oppure dopo l'esecuzione di un confronto di due numeri dello stesso valore. Per risultati diversi da 0 il flag viene resettato.

●Flag di Segno(SF): Il flag viene usato per controllare se un numero è positivo o negativo, oppure per verificare se un istruzione ha prodotto un risultato positivo o negativo, in termini di complemento a 2 aritmetico. Il bit più significativo di un numero in complemento a 2 è usato per indicare un valore positivo (resettato) o negativo (settato): questo è il bit che viene copiato nel bit 7 del registro dei flag. In alcuni microprocessori viene indicato come flag N (flag di negativo).

●Flag di Overflow (OF): Si tratta di un flag molto utile nelle istruzioni di addizione e sottrazione con numeri interi e indica un overflow, cioè l'operazione tra due valori concordi ha prodotto come risultato un valore discorde (e viceversa). Ha la stessa funzione del flag di carry nelle operazioni tra numeri naturali.

Flag di controllo

I flag di questa categoria servono a gestire alcune modalità operative del microprocessore e possono essere impostati da programma.

●Flag di Trap (TF): Questo flag viene settato quando si vuole eseguire un programma passo passo (step by step), cioè si vuole imporre al microprocessore di passare il controllo dell'utente dopo l'esecuzione di ogni istruzione.

●Flag di Interrupt (IF): Viene resettato per disabilitare le interruzioni (eventi asincroni) oppure viene settato per abilitarli.
Il flag **`IF`** è controllato dalle istruzioni **`STI`** (Set Interrupt) e **`CLI`** (Clear Interrupt).

- Flag di Direzione(DF): Le istruzioni di stringa sono le istruzione che terminano per S, come **MOVS**, **SCAS**, **CMPS**, **STOS** e così via: queste istruzioni servono, per esempio, per copiare un certo numero di locazioni da una zona ad un'altra della memoria oppure per confrontare due zone di memoria. Il flag di direzione viene usato per indicare la direzione secondo la quale queste istruzioni agiscono, in accordo con gli indirizzi di memoria contenuti nei registri **SI** e **DI**. Il flag è resettato per stringhe che si stanno elaborando nella direzione della memoria per indirizzi crescenti e settato per stringhe in elaborazione per indirizzi decrescenti di memoria.

LINGUAGGIO ASSEMBLY 8086

Per programmare un microprocessore si usa un linguaggio simbolico chiamato linguaggio assembly (o assembler), costituito da istruzioni che rappresentano il funzionamento interno del processore e delle sue operazioni elementari.

Si dice che l'assembly è un linguaggio orientato alla macchina, perché non si può scrivere software in assembly se non si conosce l'architettura del microprocessore che si vuole programmare. Questo significa che esistono molteplici assembly, a secondo del microprocessore che si vuole programmare, anche se tutti gli assembly hanno come denominatore comune le caratteristiche enumerate in precedenza.

Per tradurre un programma sorgente scritto in assembly x86, si usa un programma assemblatore, che accetta istruzioni assembly mnemoniche e li traduce automaticamente in codice oggetto. L'assemblatore assegna, quindi, all'istruzione scritta il modo sintatticamente corretto il rispettivo codice macchina. L'assemblatore viene anche detto traduttore 1:1.,perché ad un'istruzione in assembly corrisponde un'istruzione in linguaggio macchina.

A questo proposito, bisogna dire che al suo interno, un elaboratore, memorizza dati ed istruzioni, codificandoli in esadecimale; questo per ovvi motivi:

$(14)_{10}=(1110)_{2}=(E)_{16}$

come si vede, per codificare ad es. il numero 14 ci vogliono due cifre decimali, quattro cifre binarie e soltanto una cifra esadecimale; una informazione in codice esadecimale è meno ingombrante da rappresentare. Il computer memorizza in esadecimale non solo i dati ma anche le operazioni come l'addizione, la sottrazione o il salto tra due istruzioni; diventa dunque comodo associare dei codici mnemonici (facili da ricordare) alle operazioni usate per programmare la macchina.

Operazione	Codice	Termine mnemonico
Addizione	30H	ADD
Sottrazione	31H	SUB
Salto a	40H	JMP

L'assembly, come tutti i linguaggi, possiede una grammatica che consente di costruire frasi(istruzioni)in accordo ad una sintassi, un insieme di istruzioni, dirette al microprocessore, e un insieme di pseudo-istruzioni o direttive, che sono i comandi diretti all'assemblatore.

Implementa inoltre un paradigma di programmazione di tipo imperativo o procedurale secondo il teorema di Jacopini-Bohm che afferma: per codificare una qualsiasi procedura sono necessarie 3 strutture di controllo, la sequenza, la selezione e l'iterazione. (l'informatica è tutta qui)

Il modello generale di un'istruzione in assembly è il seguente:

```
[etichetta [:]] COP [Destinazione [, Sorgente] ] [;Commento]
```

come si vede dalla sintassi l'unica istruzione obbligatoria è il codice operativo (COP) tutti gli argomenti restanti sono opzionali (dato che sono contenuti in parentesi quadre).

L'etichetta è opzionale, serve solo a ad associare un nome simbolico ad una variabile o all'indirizzo di una istruzione del programma. Bisogna tener conto che l'etichetta può essere accompagnata dal carattere ":" che è obbligatorio nel caso che l'etichetta indichi l'indirizzo di una istruzione, mentre non si mette quando l'etichetta indica identifica una locazione di memoria assegnata ad una variabile.

Il COP può avere al massimo due operandi (destinazione, sorgente). Anche la presenza di eventuali commenti preceduti dal punto e virgola non sono obbligatori anche se consigliati.

Istruzioni per il processore 8086

Il set di istruzioni per l'assembly x86 può essere classificato per categoria

Istruzioni di uso generale	
MOV	Sposta i dati dalla memoria al processore e viceversa
NOP	Indica che non bisogna eseguire alcuna operazione
PUSH	Inserisce un dato nello stack
POP	Preleva un dato dallo stack
XCHG	Scambia il contenuto di due registri oppure di un registro e di una locazione di memoria, oppure un registro ed il contenuto di una locazione di memoria. Le dimensioni degli argomenti devono essere le stesse.

Istruzioni per l'input/output	
IN	Legge il valore da un porto
OUT	Scrive un dato su un porto

addizione	
ADD	Addizione
ADC	Addizione con carry
DAA	Addizione con aggiustamento decimale nel caso gli operandi siano in codice BCD
INC	incremento

Sottrazione	
SUB	Sottrazione
SBB	Sottrazione con prestito
CMP	Confronto
NEG	Negazione (complemento a 2)
DEC	Decremento
DAS	Aggiustamento decimale nel caso gli operandi siano in codice BCD

Moltiplicazione	
MUL	Moltiplicazione
IMUL	Moltiplicazione con numeri interi

Divisione	
DIV	Divisione con numeri naturali
IDIV	Divisione con numeri interi
CBW	Conversione di byte in parola (word)

CWD	Conversione di word in doppia word

Istruzioni per operazioni logiche	
AND	And logico
NOT	Not logico o complemento a 1
OR	Or logico o inclusivo
TEST	Confronto logico
XOR	Or esclusivo

Istruzioni per operazioni di shift e rotazione	
RCL	Rotazione a sinistra con carry
RCR	Rotazione a destra con carry
ROL	Per la rotazione a sinistra
ROR	Per la rotazione a destra
SAL	Shift aritmetico a sinistra
SAR	Shift aritmetico a destra
SHL	Shift logico a sinistra
SHR	Shift logico a destra

Istruzioni per le operazioni sui flag	
CLC	Azzera i flag di carry
CLD	Azzera i flag di direzione
CLI	Azzera i flag di interrupt
CMC	Complemento del flag di carry
STC	Setta il flag di carry
STD	Setta i flag di direzione
STI	Setta i flag di interrupt

Istruzioni per le operazioni di controllo dei trasferimenti	
CALL	Chiamata ad una procedura
JMP	Salto
RET	Ritorno da una procedura

Nei controlli con trasferimento condizionato, l'esecuzione del salto è determinata dal risultato di precedenti istruzioni di confronto oppure dal valore attuale dei registri di flag.

Trasferimenti condizionati	
JA	Salta se maggiore nel caso di numeri naturali
JAE	Salta se maggiore o uguale con numeri naturali
JB	Salta se minore nel caso di numeri naturali

JBE	Salta se minore o uguale nel caso di numeri naturali
JC	Salta se il carry è settato
JCXZ	Salta se il registro CX è uguale a zero
JE	Salta se gli operandi sono uguali
JG	Salta se maggiore con i numeri interi
JGE	Salta se maggiore o uguale con i numeri interi
JL	Salta se minore con i numeri interi
JLE	Salta se minore o uguale con i numeri interi
JNC	Salta se i carry è resettato
JNE	Salta se diverso
JNO	Salta se non c'è overflow
JNS	Salta se il segno è positivo
JO	Salta se c'è overflow
JS	Salta se il segno è negativo
JPE	Salta se la parità è pari
JPO	Salta se la parità è dispari

Istruzioni per il controllo dei cicli	
LOOP	Esegue un ciclo
LOOPNE	Esegue un ciclo se uguale
LOOPNE	Esegue un ciclo se non è uguale

Modi di indirizzamento

I modi di indirizzamento usati per i microprocessori della classe x86 sono:

1 Indirizzamento immediato

L'istruzione contiene il valore per cui il sorgente è un valore costante. Ad es.

```
ADD AL,7
```

significa addizionare 7 al valore del registro AL

```
CMP BL, 55
```

confronta il contenuto del registro BL con 55.

Le costanti possono essere espresse in una delle seguenti quattro basi: decimale, binario, ottale ed esadecimale.

Se si vuole esprimere una costante in una base diversa dalla base 10, basta aggiungere alla costante

- B se la base è binaria, ad es. 1101B
- O se la base è ottale ad es. 232O
- H se la costante è esadecimale, ad esempio 7FH

2 Indirizzamento con registro

In questo caso gli operandi sono contenuti in specifici registri e, nel caso dei registri accumulatori, possiamo usare sia la lunghezza word (16bit) che la lunghezza byte (8bit) chiaramente sia l'operando sorgente che l'operando destinazione devono avere la stessa dimensione. Ad es.

```
MOV AX, BX
ADD DL, AL
CMP AX, DX
```

con l'istruzione **`MOV CX, BH`** che è sbagliata perché i due registri non hanno la stessa dimensione.

3 Indirizzamento diretto (o assoluto)

Uno dei due operandi si trova nel segmento dato ed identificato da un'etichetta. Per esempio:

```
MOV AX, pippo
MOV pluto, AL
MOV pippo, 5
```

L'istruzione **`MOV 5, AL`** è' errata, perché l' operando di destinazione non può essere una costante; anche l' istruzione

```
MOV marco, andrea
```

è errata perché i due operandi non possono essere entrambi locazione di memoria.

4 Indirizzamento indiretto

L' operando si trova ad un indirizzo puntato dai registri base o indice quali: BP, BX, SI, BI Per esempio nell' istruzione

```
MOV CX, [DI]
```

Se non si vuole utilizzare la regola di default, occorre esplicitare anche il segmento (regola del segment override). Per esempio nell' istruzione

```
MOV AX, DS: [BP]
```

Il contenuto del registro BP viene sommato al contenuto del registro del segmento DS, anziché a SS come imporrebbe il default. Quando si usa questo metodo di indirizzamento, nel caso in cui non si possa ricavare dall' istruzione le dimensioni dell'operando, si verifica una situazione di errore. Per esempio, nell' istruzione **`INC [SI]`** non si può ricavare dall'istruzione stessa la dimensione della locazione di memoria che deve essere incrementata; lo stesso problema si ha con l'istruzione **`MOV [BX], 5.`** Per risolvere il problema occorre comunicare al processo un'informazione aggiuntiva, nel seguente modo:

`INC word ptr [SI]` se la variabile ha dimensione word, oppure

`MOV byte ptr [BX], 5` se la variabile puntata ha dimensione byte.

Altri prefissi sono **`dword ptr`** se si punta alla doppia word, oppure **`qword ptr`** se si punta alla quadrupla word. Quindi quando non si può ricavare dall' istruzione la dimensione degli operandi, questa deve essere esplicita. Una variante al metodo precedente consiste nell' aggiungere o togliere un valore costante. Per esempio

`MOV AX [SI + 5]` oppure **`MOV, [DI - 2]`**

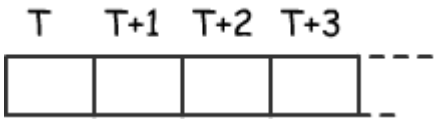
al valore contenuto nel registro viene aggiunto o tolto il valore costante indicato, prima di addizionarlo al segmento moltiplicato per 16.

5 Indirizzamento indicizzato

Nell' indirizzamento indicizzato vengono usati i registri SI e DI. In questo caso il registro rappresenta l'offset. Per esempio, nell' istruzione

```
MOV AX, T [SI]
```

La variabile T deve essere immaginata come un array ed è la base del blocco di partenza, mentre SI è l'offset.



Se SI contiene il valore 2, viene referenziata la locazione etichettata con T + 2. Anche in questo caso valgono le regole indicate nel punto precedente, con la possibilità di aggiungere o togliere un valore costante al contenuto del registro.

6 Indirizzamento con base ed indice

In questo tipo di indirizzamento l'indirizzo richiesto viene indicato tramite un registro (BX o BP) un registro indice (SI o DI) ed uno spiazzamento opzionale, come ad es.

```
MOV [AX,[BX+SI+costante]]
```

Modelli di memoria

Nei sistemi Intel, non è consentito all'utente di decidere in quale zona di memoria allocare il programma, in essi, la gestione della memoria segmentata l'utente decide il numero ed il tipo di segmenti da usare, l'utente stesso, deve esplicitamente richiedere la memoria al modulo che effettua il linking, indicando il modello di memoria da usare. I modelli di memoria consentiti sono.

.TINY L'utente deve organizzare il programma mettendo in un unico segmento sia i dati sia il codice: di conseguenza, la dimensione massima di un programma è uguale alla massima dimensione di un segmento, cioè 64K. Questo è l'unico modello di memoria che consente di creare i file con estensione .com.

.SMALL L'utente deve organizzare il programma utilizzando due segmenti: uno per il codice e uno per i dati.

.COMPACT Il programma è organizzato con un segmento per il codice e più segmenti per i dati.

.MEDIUM Il programma è organizzato con un solo segmento per i dati e più segmenti per il codice.

.LARGE Il programma è organizzato con più segmenti per i dati e più segmenti per il codice, ma non è consentito definire variabili che abbiano una dimensione pari a 64K

.HUGE Il programma è organizzato con più segmenti per i dati e più segmenti per il codice ed è consentito definire variabili che abbiano una dimensione pari a 64K.

Per quelle che sono le nostre esigenze, in tutti i programmi che seguono utilizzeremo il modello .SMALL. La struttura del nostro programma assembler 8086 avrà di solito il seguente schema

```
.MODEL SMALL
.STACK 100H
.DATA
dichiarazione
di variabili
.CODE
MOV AX,@DATA
MOV DS,AX
istruzioni
dell'algoritmo
MOV AX,4C00H
INT 21H
END
```

Quasi sempre ricorrono le seguenti direttive:

.STACK **[dimensione]**

Questa direttiva definisce il segmento di stack, indicando al posto dell'argomento "dimensione" un numero che rappresenta la quantità desiderata, espressa in byte; la massima dimensione nei sistemi MS_DOS è 64K. Il valore della dimensione viene specificato ad esempio come

.STACK **100H**

che richiede uno stack segment di 256 locazioni (100 in esadecimale). Si tratta di una direttiva non obbligatoria ma è opportuno che ci sia soprattutto quando si usano le istruzioni che utilizzano il segmento di stack (PUSH, POP, indirizzamenti indicizzati o indiretti che usano il registro BP)

.DATA

Questa direttiva definisce il primo segmento per i dati. La dimensione è calcolata automaticamente come totale dello spazio usato da tutte le variabili.

.CODE

La direttiva .CODE definisce il primo segmento per il codice eseguibile. La dimensione viene calcolata automaticamente come totale dello spazio usato dalle singole istruzioni.

Solitamente (di default) non si impone di indicare nelle istruzioni gli indirizzi in modo completo Segmento: Offset, ma solamente l'offset, occorre a tal proposito inizializzare il registro DS con l'indirizzo di partenza assegnato al segmento dati. Questo valore è deciso dal sistema al momento del caricamento e viene messo nella variabile d'ambiente @DATA. Poiché non è possibile passare direttamente il contenuto

della variabile @DATA al registro DS (in quanto non è disponibile l'istruzione per fare questo), occorre usare un altro registro di appoggio per il quale questa assegnazione è possibile, normalmente il registro AX tramite la seguente coppia di istruzioni.

```
MOV AX, @DATA
MOV DS, AX
```

Le due istruzioni suddette servono per inizializzare correttamente il registro di segmento DS, in modo da usare nel codice, come indirizzi, solo gli offset. Per terminare il programma, useremo invece le due seguenti istruzioni

```
MOV AX,4C00H
INT 21H
```

L'effetto della prima istruzione è quello di mettere la costante esadecimale 4C nel registro AH e la costante 00 in AL. La seconda istruzione INT 21H richiede uno dei servizi del DOS: il 4C (in ogni chiamata al DOS il numero del servizio neve essere contenuto in AH) perché è quello che restituisce il controllo al sistema operativo. In AL viene messo il codice di errore: il valore zero indica che tutto si è svolto correttamente. Seguirà la pseudo-istruzione **END** che rappresenta la fine fisica del programma; comanda, infatti all'assemblatore di terminare la fase di assemblaggio.

```
TITLE X
.model small
.stack
.data messaggio db "Ciao a tutti sto imparando l'assembly","$"
.code
mov ax,@DATA
mov ds,ax
mov ah,09
lea dx,messaggio
int 21h
mov ax,4c00h
int 21h
end
```

viene messo il valore 9 nel registro AX che predispone il processore alla stampa a video di una stringa. La stringa da stampare viene posta nel registro DX attraverso l'istruzione

```
LEA DX,MESSAGGIO
```

L'interrupt 21H si occupa di attuare l'operazione di stampa. Come si vede, l'assembly, al contrario di certi linguaggi ad alto livello, non è case-sensitive, cioè, è indifferente alle maiuscole/minuscole. Importante è invece l'operazione di concatenamento della stringa in output con il carattere **$** indicatore della terminazione della stringa stessa. Dopo aver salvato il codice in un file di testo con nome ed estensione x.asm nella cartella TASM dove sono contenuti il file del compilatore, assemblare il programma dalla riga di comando con

```
C:\TASM>tasm x
```

linkare il programma col comando:

```
C:\TASM>tlink x
```

eseguire il programma digitando:

```
C:\TASM>x
```

Se invece di una intera stringa vogliamo stampare a video un singolo carattere, occorrerà usare il metodo illustrato nel seguente programma

```
TITLE Y
.model small
.stack
.data
.code
mov ax,@DATA
mov ds,ax
mov dl,66
mov ah,02
int 21h
mov ax,4c00h
int 21h
end
```

Per stampare a video il contenuto di un registro basta inserire il contenuto da stampare nel registro DL, settare a 2 il valore di AH e invocare l'interruzione 21.

```
MOV AH,2
MOV DL,66
INT 21H
```

In questo caso viene stampato a video il carattere 'B', che corrisponde al 66° elemento del codice ASCII. Ogni registro ha due parti, ad esempio, AX ha una parte bassa AL e una parte alta AH. Sia AL che AH sono costituiti da 8 bit e possono dunque computare fino a $256=2^8$. Editare e salvare il seguente file col nome y.asm.

```
C:\TASM>tasm y
```

linkare il programma col comando:

```
C:\TASM>tlink y
```

eseguire il programma digitando:

```
C:\TASM>y
```

Se invece di stampare un carattere, vogliamo stampare un numero (una cifra) possiamo usare l'accorgimento che si vede nel seguente programma

```
TITLE Z
.model small
.stack
.data
op db 3
.code
mov ax,@DATA
mov ds,ax
mov dl,op
add dl,48
mov ah,02
int 21h
mov ax,4c00h
int 21h
end
```

L'operando op=3 viene messo nel registro DL; allo stesso registro viene poi aggiunto 48 (48 è il codice ASCII che rappresenta lo 0 a cui seguono gli altri numeri fino al 9). In questo modo, il programma, invece di stampare il 3° carattere del codice ASCII stamperà il 3. Possiamo poi compilare e linkare il programma scritto come visto negli esempi precedenti.

COMPILATORE

Come abbiamo visto, per sviluppare programmi scritti in codice assembler sono necessari alcuni strumenti software essenziali: l'editor di testo, il programma assemblatore, il linker.

L'editor di testi o text editor è un programma che permette di scrivere un documento costituito esclusivamente da testo; l'editor di testi più comunemente usato per programmare sui sistemi Windows è il blocco note (Noteblock). Alla fine della scrittura del codice sorgente in linguaggio assembler il documento realizzato deve essere salvato con estensione .asm. Nel programma sorgente ogni carattere viene codificato nel codice macchina secondo il set di caratteri ASCII.

Come assemblatore può essere usato tipicamente MASM (Microsoft Macro Assembler) oppure, come usiamo noi, il TASM (Turbo Assembler). L'assemblatore è' un programma di traduzione di procedure scritte in linguaggio assembly: esso assegna alle frasi sintatticamente corretto il rispettivo codice macchina. L'input al programma è costituito dalla procedura dell'utente scritta in linguaggio assemblativo (sorgente)con l'impiego di un editor di testo e memorizzato sul disco in un file .asm.

Il programma assemblatore traduce sia i dati che le istruzioni del programma sorgente come un unico insieme di dati, detto data set di input, e fornisce in uscita il data set di output scritto in codice oggetto. Fisicamente, ad una lista di byte del programma sorgente corrisponde in uscita dal programma assemblatore, una diversa lista di byte del programma oggetto (File.obj).

L'assemblatore, fa uso di una variabile chiamata termine location counter utilizzata per determinare gli indirizzi occupati dai singoli byte del data set di output. Il valore iniziale di questo parametro di solito è 0. Nella prima passata il programma assemblatore genera una tabella che mette in corrispondenza i nomi simbolici assegnati alle etichette con il valore corrente del location counter. Questa tabella, detta symbol table, contiene anche i valori attribuiti ai nomi simbolici posti in ciascun campo operando delle istruzioni e definite da opportune pseudo-istruzioni. La seconda passata si esaurisce semplicemente assegnando ai nomi simbolici i valori ottenuti dalla symbol table. L'assemblatore deve essere in grado di fornire messaggi di errore nel caso non venga rispettata la sintassi. In generale se un programma non ha errori di sintassi non significa necessariamente che "giri", ma semplicemente che la sintassi è corretta.

LINKER

L'assemblatore generalmente traduce il programma sorgente calcolando gli indirizzi a partire dall'indirizzo 0. Il programma oggetto può quindi essere eseguito correttamente solo se viene allocato in memoria a partire proprio dalla locazione di indirizzo 0. Questo in genere non è possibile in quanto le prime locazioni vengono utilizzate dal sistema operativo stesso e il programma utente è allocato in memoria centrale solo a partire da un altro indirizzo. Indichiamo con IBR l'indirizzo della locazione di memoria a partire dalla quale il

programma è allocato in memoria per la sua esecuzione. Affinché il programma possa essere eseguito correttamente tutti gli indirizzi che compaiono all'interno del programma oggetto devono essere incrementati della quantità IBR. L'indirizzo IBR si chiama indirizzo base di rilocazione. Queste operazioni di ricalcolo sugli indirizzi sono svolte dal programma linker(collegatore). Oltre a questo il linker costruisce un programma unico a partire dai programmi oggetto separati (che possono essere procedure esterne e librerie) prodotti dall'assemblatore, completando, con l'aiuto delle informazioni che si trovano nel programma sorgente stesso, i riferimenti che erano rimasti in sospeso. Esso giustappone il programma, le eventuali routine esterne e le librerie una di seguito all'altro; inoltre, dove necessario, trasforma gli indirizzi in modo da tenere conto di questo concatenamento.

FONTE: HTTPS://WWW.EDUTECNICA.IT/