Enrico Tombelli
Docente presso
ITC "A. Volta" - Bagno a Ripoli - Firenze
(e.tombelli@libero.it)

MICROPROCESSORI a 16/32 BIT (e Intel 8086)

MICROPROCESSORI A 16/32 BIT (e Intel 8086)

Le principali esigenze che hanno portato allo sviluppo della tecnologia verso i microprocessori a 16/32 bit sono le seguenti:

- Velocizzazione dell'elaborazione delle informazioni con particolare riguardo all'ampliamento della capacità di calcolo e alla velocità d'esecuzione delle istruzioni. Ciò si è ottenuto non soltanto con l'aumento della velocità del clock che comunque risulta preponderante[1], ma anche rendendo più efficiente lo svolgimento stesso delle istruzioni. Infatti, già nella versione INTEL 8086 dei primi anni 80' si hanno strutture multiprocessore dove ci sono più blocchi che lavorano in parallelo (vedi p.e. il prefetching). Nel PENTIUM si hanno addirittura due microprocessori della generazione precedente che operano contemporaneamente. Si capisce quindi come la potenza di calcolo intesa come numero di istruzioni elementari eseguibili nell'unità di tempo sia quasi diecimila volte superiore[2].

- Occhio di riguardo all'utilizzo di linguaggi ad alto livello. A questo scopo sono stati particolarmente curati i vari modi di indirizzamento e l'organizzazione dei dati, cosicché i compilatori che operano la traduzione dei programmi scritti in linguaggio alto livello (Visual Basic, C, Pascal ecc.) risultano più efficienti nel produrre le versioni in linguaggio macchina.

- Un'altra innovazione è la possibilità di gestire la multiutenza[3] utilizzando servizi che sono stati implementati già a livello hardware.

Breve parentesi sulla multiutenza:

É la possibilità di eseguire più processi contemporaneamente ed è praticamente nata con i primi computer. Questi ultimi essendo particolarmente costosi non potevano essere utilizzati da un solo utente (al contrario del PC). Infatti l'esecuzione di un programma comporta che la CPU deve rimanere spesso in attesa per adattarsi alla velocità di un'altra risorsa come una stampante (p.e.). Questi tempi di attesa non potevano assolutamente essere perduti visto l'enorme costo della macchina. Si è quindi subito pensato di utilizzare i tempi morti per servire altri utenti arrivando addirittura ad estendere tale tecnica assegnando un "QUANTO" di tempo di esecuzione ad ogni processo in modo che tutti potessero comunque essere serviti. Partendo da questo concetto la tecnologia e i metodi di assegnazione della CPU per l'esecuzione contemporanea di più programmi si allarga per poter risolvere tutte le problematiche che si vengono a creare. Sta di fatto che ogni utente dispone di una potenza di calcolo continua ed ha la sensazione di avere la disponibilità completa dell'elaboratore. Tutto questo può essere efficientemente realizzato solamente se i processi da eseguire si trovano tutti contemporaneamente in memoria. Questa tecnica si dice appunto MULTIUTENZA[4].

---

[1] Basta pensare che nei primi anni 80 (MP 8086) i microprocessori dell'epoca funzionavano con frequenze di clock dell'ordine del MHz. Attualmente, nelle ultime generazioni di MP si arriva a frequenze dell'ordine del GHz, ovvero quasi 1000 volte superiori senza tenere conto che tale rapporto diventa continuamente più alto.

[2] Il numero di milioni di istruzioni che è possibile eseguire in un secondo è detto MIPS (million of instructions per second)

[3] Possibilità di utilizzo contemporaneo della CPU da parte di più utenti, che hanno inoltre la possibilità di condividere anche altre risorse come la stampante, le memorie di massa ecc.

[4] Si deve pensare che attualmente prima di parlare di multiutenza si tratta di MULTITASKING. Questa situazione si può avere p.e. nei PC dove è possibile avere più programmi in esecuzione ma appartenenti allo stesso utente. (P.es col sist. Op. WINDOWS è possibile attivare più TASK, ovvero più partizioni con

La presenza di più programmi in memoria comporta numerosi problemi di sicurezza e affidabilità, specialmente se i processi sono di utenti diversi. Si deve continuamente controllare che l'esecuzione di un processo non invada aree di memoria nelle quali risiedono processi diverso (in pratica si deve fare in modo che p.e. un'istruzione di salto non faccia riferimento a celle di memoria di altri processi. Le zone di memoria (PARTIZIONI O TASK) devono quindi essere ben separate e controllate continuamente. Quest'incombenza è affidata al SISTEMA OPERATIVO o comunque ad un programma di MONITORIZZAZIONE che intercetta le operazioni di salto e controlla che queste avvengano correttamente. Il controllo, nei sistemi a microprocessore ad 8 bit avveniva in modo ambiguo in quanto la gestione LINEARE (ovvero ogni cella ha semplicemente un indirizzo a partire dalla cella 0 fino all'ultima) dell'indirizzamento non favoriva certo il processo di monitoraggio che risultava alquanto precario. Nei microprocessori dell'ultima generazione si tiene conto di questa necessità in quanto l'hardware è stato progettato già con alcune funzioni accessorie che permettono questo tipo di controllo in modo più efficiente.

In particolare, per poter allocare efficientemente i vari processi da eseguire in memoria è stato adottato un particolare sistema di gestione, consistente nella ripartizione della memoria stessa in vari blocchi , dentro ognuno dei quali viene riposta una parte del processo in esecuzione. In effetti questo sistema comporta un notevole vantaggio, in quanto non è necessario che tutto il processo in esecuzione sia contemporaneamente in memoria, ma basta che vi sia solo la parte attualmente in esecuzione, il resto può stare sul disco ed essere trasferito in memoria solo nel momento in cui deve essere effettivamente eseguito necessitando così di una capacità di memoria (fisica) molto più ristretta di quella effettivamente necessaria (MEMORIA VIRTUALE).

Tutto ciò è realizzato scomponendo ogni indirizzo in due parti; l'indirizzo del segmento e l'indirizzo della cella all'interno del segmento, trattando poi separatamente le due parti. La composizione dell'indirizzo effettivo, ovvero la concatenazione del riferimento al segmento e del riferimento alla cella all'interno del segmento è realizzata tramite un dispositivo detto MMU (Memory Menagement Unit), che può trovarsi integrato nella CPU (vedi Intel 8086/8088) oppure in un chip separato (vedi Zilog Z8000).

Nell'evoluzione che ha portato alla costruzione di architetture a 16/32 bit è stato tenuto conto il più possibile della compatibilità con quelle della generazione precedente a 8 bit .

- ORGANIZZAZIONE DELLA MEMORIA.

Lo spazio di indirizzamento della memoria può essere gestito principalmente utilizzando una di queste tre tecniche:

- Spazio di indirizzamento **LINEARE** (Sistema Motorola; p.e. 68000).

  Questo sistema tratta la memoria come un insieme di celle consecutive, ciascuna delle quali è selezionata da una singola componente di indirizzo, il cui range di valori ne determina la dimensione massima.

  La lunghezza del codice di indirizzo (16,20,24,32) definisce le dimensioni del bus indirizzi e dei registri puntatori della CPU.

---

all'interno un processo in esecuzione. è il caso in cui si attiva il processo di stampa in una finestra e in un'altra si da il via alla formattazione di un floppy. I due processi vengono eseguiti contemporaneamente). Quando invece i processi in esecuzione appartengono a più utenti si parla di MULTIUTENZA.

É la tecnica più semplice, ma non presenta nessuna capacità intrinseca di protezione e virtualizzazione della memoria.

- Spazio di indirizzamento **MAPPATO IN PAGINE** (Sistema Zilog ; p.e. Z8000)

La memoria e suddivisa in blocchi di ampiezza fissa al loro interno lineari dette PAGINE che non possono sovrapporsi, nemmeno parzialmente, in quanto la locazione iniziale di ogni singola pagina è stata prefissata a priori ed è adiacente all'ultima cella della pagina che la precede nell'ordine sequenziale.

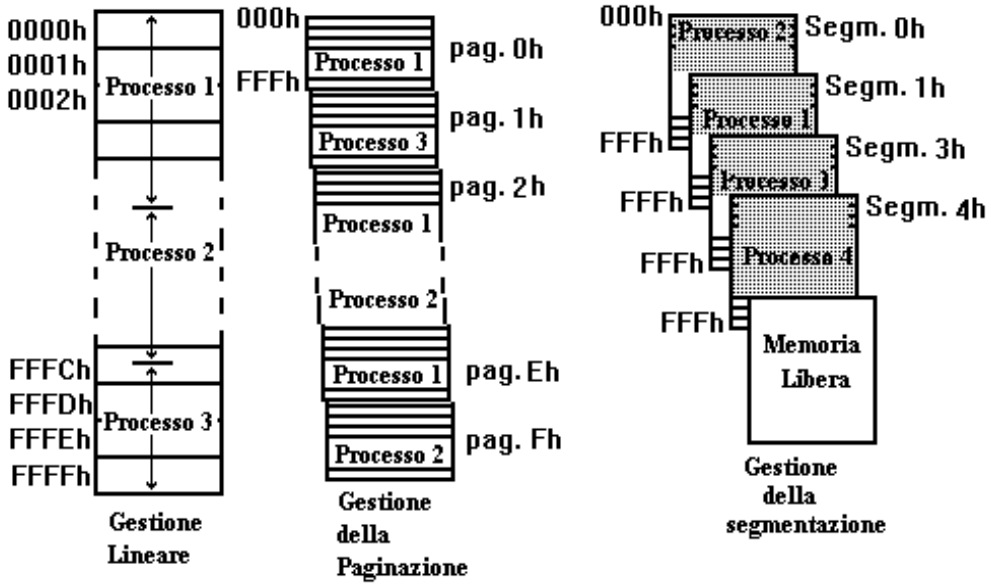
L'indirizzo fisico di ogni cella è identificato in modo rigido dalla composizione dell'indirizzo della pagina con l'indirizzo della cella all'interno della pagina.

P.e. 1 Mcelle di memoria suddivise in 16 pagine (0, 1, ...15) di 64 Kcelle ciascuna. La cella A8D1h della pagina 7 ha come indirizzo fisico 7A8D1h (20 bit)

- Spazio di indirizzamento **SEGMENTATO** (Sistema Intel; p.e. 8086/8088).

Questo sistema è intermedio a quelli visti precedentemente, infatti, la memoria è logicamente suddivisa in un insieme di aree, al loro interno lineari, dette SEGMENTI, disposti in modo apparentemente casuale (l'ordine è quello cronologico stabilito al momento dell'allocazione in memoria del segmento) così da coprire lo spazio fisico disponibile.

Ogni cella della memoria è quindi identificata da due parametri, ovvero il selettore di segmento a 16



GESTIONE DELLA MEMORIA

bit e l'identificatore della cella all'interno del segmento (displacement, anch'esso a 16 bit) che ne definisce la posizione rispetto alla locazione iniziale del segmento stesso.

Tutte le istruzioni di un segmento in esecuzione contengono riferimenti a celle del segmento stesso, mentre è lasciata al sistema operativo la gestione degli identificatori del segmento quando avviene un

salto ad un segmento diverso. Questa tecnica non garantisce implicitamente l'impossibilità di sconfinamento in altre aree di memoria ove potrebbero risiedere altri processi in attesa in quanto i segmenti possono parzialmente sovrapporsi, ma ha il vantaggio di essere più flessibile e permette di utilizzare efficientemente lo spazio di memoria.

- MEMORY MENAGEMENT UNIT (MMU)

Il MMU è un dispositivo piuttosto complicato, che ha numerosi compiti, fra i quali la traduzione da indirizzo logico, (formato dai due descrittori di indirizzo) a indirizzo fisico. Tale traduzione avviene in modo controllato nel senso che prima di accedere alla cella indirizzata, p.e. nel caso del prelievo della istruzione prevista per l'esecuzione, viene verificato se tale cella appartiene ad uno dei blocchi associati al processo attualmente in esecuzione. Questo per evitare sconfinamenti da parte del processo in aree non a lui destinate, e che potrebbero portare a malfunzionamenti.

A questo scopo, il MMU dispone di una tabella che lui stesso gestisce e che associa ad ogni blocco della memoria le relative informazioni; per esempio:

- Numero identificativo del blocco.
- Indirizzo iniziale del blocco.
- Nome del processo (o parte di esso) riposto al suo interno.
- Lunghezza del blocco occupato dal processo.
- Attributi vari (solo lettura, protezioni varie ecc.)

Fra i vari attributi associati al blocco, c'è un identificatore di privilegio che segnala alla CPU il modo operativo del processo residente nel blocco, ovvero l'insieme delle operazioni ammesse dal processo. In genere il modo operativo può essere :

- System (Per il sistema operativo)
- User (Per tutti gli altri processi)

Nel modo "System" il processo è abilitato ad effettuare qualsiasi tipo di operazione.

Il MMU dispone anche di una memoria, generalmente interna, di tipo associativo (ovvero un tipo di memoria nella quale l'accesso non viene effettuato in base all'indirizzo, ma direttamente in base al contenuto, grazie alla interrogazione in parallelo di tutte le celle) detta "Cache Memory" nella quale vengono riposti i blocchi di memoria appena utilizzati in modo che essi possano essere recuperati più velocemente nel caso debbano essere riutilizzati a breve scadenza senza andare a recuperarli nella memoria di massa.

- PREFETCHING DELLE ISTRUZIONI

Il prefetching delle istruzioni consiste nel sovrapporre nel tempo le fasi di fetch e d'esecuzione. E' attuato distinguendo, all'interno della CPU due processori indipendenti rispetto all'interfacciamento col bus:

- I circuiti che provvedono al recupero delle istruzioni da eseguire, sfruttano i tempi di non utilizzo del bus durante l'esecuzione per la fase di fetch. Non viene prelevata solo l'istruzione da eseguire prossimamente, ma anche alcune successive, che vengono riposte in una struttura FIFO in attesa della esecuzione.
- Il secondo processore, invece ha il compito di eseguire le istruzioni prendendoli dalla struttura FIFO

Con il sistema di prefetching si riduce notevolmente (azzera in teoria) il tempo necessario per l'acquisizione dell'istruzione, ovvero della fase di fetch, in quanto questa è sovrapposta alla esecuzione vera e propria.

In effetti, può verificarsi che la presenza di istruzioni di salto nella coda vanifichi il beneficio, in quanto viene variata la normale sequenza di esecuzione. Alcune istruzioni della coda non vengono quindi eseguite e la coda viene vuotata.

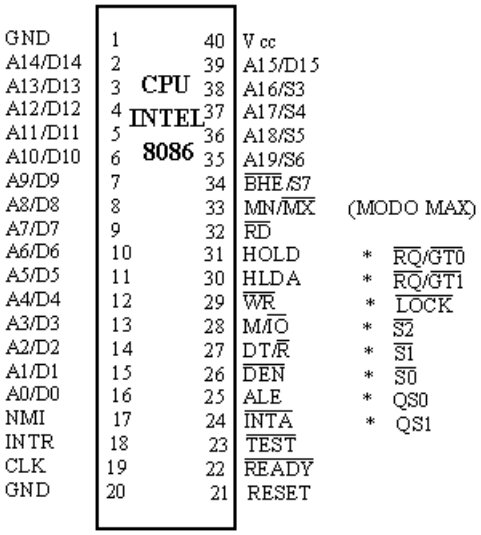
Nonostante questo problema, l'operazione di prefetching risulta comunque conveniente se il numero di istruzioni in coda è sufficientemente limitato.

- DEBUGGING FACILITIES

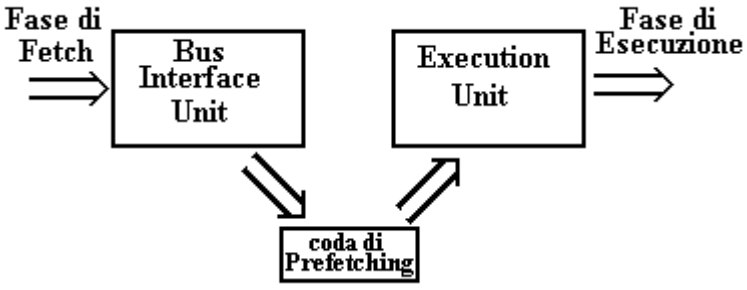
Alcune CPU dell'ultima generazione dispongono di strutture logiche integrate nella CPU stessa che permettono di supportare alcuni degli aspetti più delicati relativi alla messa a punto dei programmi, come l'impostazione di "break-point" (punti prefissati sui quali il processo si ferma permettendo la consultazione dello stato della CPU, dei suoi registri interni e della memoria) e il "Tracing", ovvero l'esecuzione "passo-passo" del programma e presentazione della relativa situazione.

- MICROPROCESSORE INTEL 8086

Le funzionalità interne della CPU sono suddivise logicamente in due unità indipendenti:



- **BIU (Bus Interface Unit)**: provvede alla gestione degli indirizzamenti sul bus, curando le sincronizzazioni con i dispositivi esterni. É dotata di quattro registri per l'indirizzamento del blocco di memoria (Registri di segmento CS, SS, DS, ES) del registro contatore di programma, che in questa CPU assume il nome di "Instruction Pointer" (IP) e della "Instruction Queque", ovvero di una struttura FIFO di 6 Byte entro la quale vengono riposte le istruzioni accodandole in sequenza. Quest'ultimo dispositivo permette di realizzare il meccanismo di "Prefetching" che consente di prelevare in modo asincrono le istruzioni nel corso di stati interni che non richiedono accessi alla memoria, permettendo così di sfruttare efficientemente il bus.
- **EU (Execution Unit)**: É dotata di otto registri operativi e di puntamento, della ALU e del Registro di stato (Registro FLAG). Un bus interno alla CPU connette tutte le sue parti fra di loro.



- ORGANIZZAZIONE LOGICA

DELLA MEMORIA

Come già accennato, il tipo di organizzazione adottato per la memoria è la SEGMENTAZIONE. Infatti, la memoria è suddivisa in BLOCCHI di 64 KByte ciascuno (segmento) e ad ognuno di essi si fa riferimento con un indirizzo di 16 bit.

Ogni segmento può essere di uno dei quattro tipi sotto descritti a seconda del tipo di registro puntatore che contiene il relativo indirizzo base:

Registro	**Tipo segmento**	**Tipo d'informazione contenuta**
CS	Code Segment	Processo da eseguire
DS	Data "	Dati da elaborare
SS	Stack "	Area di Stack
ES	Extra Data Segm.	Dati Supplementari (blocchi)

Il riferimento ad ogni cella di memoria è realizzato tramite due informazioni:

- INDIRIZZO BASE del segmento, contenuto in uno dei registri di segmento.
- DISPLACEMENT, che rappresenta l'offset da comporre con l'indirizzo base e che realizza il riferimento della cella all'interno del segmento.

La composizione dell'indirizzo base con il displacement per la formazione dell'indirizzo fisico è semplicemente la somma fra i displacement e l'indirizzo base (quest'ultimo moltiplicato per 16 ovvero shiftato verso sinistra di 4 bit o equivalentemente di una cifra esadecimale).

P.E.

Indirizzo Segmento	675Ah
Diplacement	AB79h

INDIR. FISICO = 675Ah (x 10h) + AB79h = 675A0h+AB79h = 72119h (20 bit)

Poiché da questa logica risulta che l'indirizzo iniziale dei segmenti può essere un qualunque valore multiplo di 16, ne deriva che i segmenti possono sovrapporsi, consentendo di dimensionare in modo ottimizzato la memoria fisica.

Non viene eseguito peraltro nessun controllo automatico da parte della CPU, sul range di displacement definito dai registri. É necessario quindi prestare le dovute cautele affinché siano evitati accessi scorretti al di fuori del segmento.

- AREE DI MEMORIA RISERVATE

Alcune parti della memoria sono riservate per specifiche operatività della CPU:

- La parte bassa della Memoria (da 00000h a 003FFh : 1KByte) contiene il vettore di interrupt[5].
- La parte alta (da FFFF0h a FFFFFh : 16 Byte) invece contiene la procedura di Bootstrap al reset e il codice di salto al programma utente (Nel caso di PC al Sistema operativo). A parte le prime tre celle a

[5] L'INTERRUPT è una tecnica che permette di lanciare una procedura attivando un segnale fisico.

partire da FFFF0h che ospitano il salto incondizionato al programma utente, la rimanente parte di area deve essere lasciata libera.

Da notare che il reset inizializza il registro CS a FFFFh e il registro IP a 0000h, e che pertanto la prima istruzione da eseguire si trova all'indirizzo fisico FFFF0h.

- REGISTRI INTERNI

Nella BIU ci sono i quattro registri di segmento e il registro "IP" (Instruction Pointer). Il Registro IP a 16 bit contiene l'indirizzo della prossima istruzione da recuperare. La formazione dell'indirizzo dell'istruzione non dipende però solo dal registro IP, ma anche da registro CS. Insieme, formano l'indirizzo fisico della prossima istruzione da eseguire (IP contiene il displacement, mentre il CS contiene l'indirizzo del segmento).

Oltre ai quattro registri di segmento già visti (CS, DS, ES, e SS) e al registro contatore di programma (IP) che formano il primo gruppo (sono i cinque registri che si trovano nella BIU), la CPU contiene al suo interno (nella EU) altri nove registri raggruppati come segue:

- Registri Accumulatori (AX, BX, CX, DX) a 16 bit che vengono utilizzati per scopi generali. Tali registri possono essere utilizzati anche scomposti in 8 bit (Parte alta e parte bassa). In questo modo essi formano 8 registri a 8 bit ciascuno, denominati AL, AH, BL, BH, CL, CH, DL, DH. Questa caratteristica vale solo per questo gruppo di registri.
- Registri Puntatori e Indici (16 bit ciascuno).

  SP Stack Pointer
  BP Base Pointer
  SI Source Index
  DI Destination Index

  Questi registri sono utilizzati genericamente come puntatori o indici e contengono l'indirizzo di displacement ovvero l'offset da aggiungere all'indirizzo di segmento per la formazione dell'indirizzo fisico.
- Registro Flags (16 bit non tutti utilizzati). è il registro di stato della CPU, e contiene le seguenti informazioni:

  - CF (Carry): Indica (quando posto a 1) che nell'ultima operazione aritmetica c'è stato un riporto o un prestito.
  - AF (Auxiliary Carry): Come il flag precedente indica che si è propagato un riporto, ma fra i bit 3 e 4 del risultato. Viene utilizzato per operazioni fra numeri di tipo BCD.
  - DF (Direction): Questo flag indica la direzione di incremento o decremento automatico nelle operazioni per la manipolazione di blocchi interi di memoria (Stringhe) effettuate tramite apposite istruzioni di loop.
  - IF (Interrupt Enable): è il flag che controlla l'abilitazione del sistema alla interruzione (1 Interrupt abilitato).
  - OF (Overflow): Annuncia che nell'ultima operazione aritmetica si è verificato un overflow, quando è a 1.
  - PF (Parity):Indica la parità utilizzata nei controlli, ovvero è 1 se la parità è di tipo pari.
  - SF (Sign): Viene posto a 1 se il risultato dell'ultima operazione aritmetica è stato negativo, ovvero se l'MSB è a 1.

- TF (Trap): Quando viene forzato a 1 dall'operatore, la CPU si predispone nello stato "Single Steep", ed è possibile eseguire una istruzione per volta dopo di che si pone in attesa, permettendo all'operatore l'analisi della situazione allo scopo di favorirne il debug.
- ZF (Zero): Viene posto a 1 quando il risultato dell'ultima operazione aritmetica è stato nullo.

Programmi che non definiscono ne manipolano registri di segmento, risultano dinamicamente rilocabili, cioè possono essere allocati in memoria in zone qualunque senza la necessità di ricompilarli. Questo perché la definizione dell'indirizzo fisico di segmento è lasciata al controllo del sistema operativo.

- MODI DI INDIRIZZAMENTO

I trasferimenti che possono essere effettuati tramite la CPU di tipo 8086 o 8088 possono essere di due tipi:

- Da registro interno a registro interno
- Dalla memoria ad un registro interno o viceversa.

Non sono ammessi trasferimenti diretti (tramite una sola istruzione) da una cella di memoria ad un'altra cella di memoria.

Il recupero dell'operando di un'istruzione Assembler può avvenire facendo comparire direttamente il nome del registro interessato, nel caso che venga utilizzato un registro interno alla CPU, oppure identificando l'indirizzo fisico della memoria, nel caso che l'operazione da effettuare riguardi una cella.

In effetti l'identificazione dell'indirizzo fisico della cella di memoria dipende come già detto da due parametri a 16 bit, ovvero:

- Il selettore o indirizzo del segmento
- Il displacement

Questi vengono sommati fra loro facendo però pesare il codice di segmento 16 volte di più (Infatti viene shiftato di una cifra esadecimale verso sinistra prima della somma) per la formazione dell'indirizzo fisico in cui si trova l'operando.

I due parametri vengono a loro volta recuperati come segue:

- Indirizzo di segmento

  É riposto in uno dei registri di segmento (CS, SS, DS, ES) a seconda dei casi. In particolare viene utilizzato il registro CS quando il segmento contiene un processo in esecuzione o una sua parte. Questo registro agisce in coppia col registro IP. Questi due registri compongono il puntatore all'istruzione (Program Counter) da prelevare per l'esecuzione.

  per accedere allo stack (Operazioni di PUSH e POP) è sempre utilizzato il registro SS, che agisce in coppia col registro SP. Insieme, i due registri, formano il "PUNTATORE ALLO STACK".

  Nella manipolazione di dati può essere utilizzato sia il registro DS che il registro ES secondo i casi:

  - ES nella manipolazione di stringhe (blocchi di caratteri) e in particolare definisce la destinazione implicita della stringa (insieme al registro DI).

- DS può essere adoperato in due casi, ovvero nella manipolazione di stringhe (in questo caso definisce il segmento sorgente e agisce in coppia col registro SI), oppure nella trattazione di dati generici agendo con i registri SI oppure DI.

Quanto detto è riassunto dalla tabella:

Reg. Operando	Reg. segmento	Note
IP	CS	Acquisizione Istruzioni
SP	SS	Istruzione accesso Stack
BP	SS	" " "
SI	DS	Manipol. dati generici
SI	DS	" stringhe (Sorgente)
DI	ES	" stringhe (Destinazione)

L'associazione (di default) può, in alcuni casi, essere variata facendo precedere il codice della istruzione da un prefisso di un byte detto "Segment Override Prefix".

S.O.P. = 001 Id. Reg. 110

Id. Registro	Registro segmento
00	ES
01	CS
10	SS
11	DS

(1 byte), dove l'identificatore del registro è definito come segue:

L'utilizzo di questo prefisso permette di variare l'assegnazione come in tabella.

Per indicare la variazione del registro di segmento rispetto a quello di default, nell'istruzione in linguaggio Assembler, si fa comparire il nome del registro di segmento prescelto seguito da "**:**".

Reg. Oper.	Note	Reg. segm.
BP		DS, ES o CS
SI	Manipol. dati generici	ES, SS o CS
SI	" stringhe (Dest.)	ES, SS o CS

P.e. Nel caso che il registro sia ES si ha MOV AX,ES:7Ah[BP][DI]

- Displacement.

Il displacement identifica lo spiazzamento della locazione rispetto all'indirizzo base del segmento identificato dal registro di segmento.

É composto dalla somma di tre parti:

- L'OFFSET, è un valore fisso (di 8 o 16 bit a seconda dei casi) e che compare direttamente nell'operando della istruzione stessa.
- L'INDIRIZZO BASE, è un valore a 16 bit, contenuto in un registro base (BX o BP). compare nella istruzione fra parentesi quadre, p.e. [BX] o [BP].
- L'INDICE, è anch'esso un valore a 16 bit e viene immagazzinato in un registro indice (DI o SI). Come nel caso dell'indirizzo base, è indicato nell'istruzione fra parentesi quadre, p.e. [SI] o [DI].

Esempio:

MOV AX,6h[BX][DI]

AX è uno dei registri accumulatori, e rappresenta, in questo caso la destinazione del risultato.

La sorgente è identificata da BX (contiene l'indirizzo base p.e. 4AE8h), DI (Indice p.e. AA56h) e dal valore 6h (offset.

Il displacement è calcolato come somma di BX+DI+6, ovvero, nel nostro esempio 4AE8h+AA56h+6 = F544h (Displacement)

L'indirizzo del segmento (ovvero l'indirizzo di partenza) è contenuto in un registro di segmento, e in particolare, se non si è forzato l'assegnazione di default tramite il Segment Override Prefix, viene utilizzato il registro dati DS (p.e. 116Dh)

La sorgente del dato da trasferire è la cella di memoria il cui indirizzo fisico è calcolato come segue:

Indirizzo segmento	116D0h	+
Displacement	F544h	=

Indirizzo Fisico	20C14h	

Il modo di indirizzamento è definito abbastanza semplicemente dalla configurazione del displacement, a seconda della presenza o no degli elementi che lo definiscono (Offset, Base e Indice).

Da ciò si capisce che facendo tutte le possibili combinazioni si ottiene un numero di modi piuttosto elevato (circa una ventina). Sulla classificazione dei vari modi ogni testo analizzato propone una sua versione che comunque si può mediare come segue (Negli esempi si fa riferimento alla sorgente della istruzione "MOV", che trasferisce il contenuto della sorgente ovvero l'operando di sinistra, alla destinazione, ovvero l'operando di destra) :

MOV Destinazione,Sorgente

- MODO REGISTRO (Register Mode).

Istruzione	Cod. Macchina	Funzione	Tipo
MOV AX,BX	89 D8	(BX→AX)	
MOV AL,BH	88 F8	(BH→AL)	

- MODO DIRETTO (Direct Mode).

MOV CX,[0100]	8B 0E [00 01]	([0100h] →CX)	

- MODO IMMEDIATO (Immediate Mode).

MOV DX,045A	BA [5A 04]	(45Ah→DX)	

- MODO INDIRETTO (Base e/o Index Indirect Mode).

MOV AX,19[BX]	8B 47 [19]	([BX+19h] →AX)	(Base Indir.)
MOV AX,[BX]	8B 07	([BX] →AX)	
MOV AX,[DI+46]	8B 45 [46]	([DI+46h] →AX)	(Index Indir.)
MOV AX,[DI]	8B 05	([DI] →AX)	

MOV AX,13[BP][SI]	8B 42 [13]	([BP+SI+13h] →AX)	(Base Index Indir.)
MOV AX,[BP][SI]	8B 02	([BP+SI] →AX)	

Si può osservare che l'offset può essere o no presente.

A questi tipi di indirizzamento, possiamo aggiungere le tecniche riguardanti i salti, associate alle istruzioni di SALTO. Esse servono per la modifica della normale sequenza d'esecuzione delle istruzioni. Si prevedono due tipi di salti:

- ASSOLUTO: quando la cella puntata e unicamente identificata da un indirizzo specifico.

  P.e. (FFh 26h 00h 01h 4 byte) JMP [0100h] (salta incondizionatamente alla cella 100h)

  Ovviamente il riferimento della cella parte dall'indirizzo 0000h del segmento dove risiede il programma, cioè quello identificato dal registro CS (salto INTRASEGMENTALE). Si può osservare che l'istruzione codificata in linguaggio macchina è composta da 4 byte di cui i primi due (FFh e 26h) compongono il Codice operativo, mentre gli altri due sono (00h e 01h) sono l'indirizzo al quale saltare. è possibile effettuare un salto anche al di fuori del segmento ove risiede il programma (salto INTERSEGMENTALE). In questo caso si deve specificare nell'istruzione anche il valore del selettore di segmento.

  P.e. (EAh 00h 01h 34h 12h 5 byte) JMP 1234:0100h (salta alla cella 100h del segmento 1234h).

  Si osserva che i byte che formano l'istruzione sono 5 di cui il primo è il Cod. Operativo (EAh), i due seguenti (00h e 01h) sono il displacement e gli ultimi due sono il selettore del segmento (34h e 12h).

- RELATIVO: il salto avviene ad un riferimento relativo alla cella sulla quale si trova l'istruzione di salto ("salta 3 celle in avanti" o "salta 8 celle indietro"). Nell'istruzione quindi non compare direttamente l'indirizzo al quale saltare, ma l'OFFSET da recuperare per effettuare lo spostamento del controllo esecuzione.

```
0100 B8 00 01     MOV     AX,0100h
0103 EB 05        JMP     010Ah
0105 89 D9        MOV     CX,BX
0107 47           INC     DI
0108 EB F6        JMP     0100h
010A 48           DEC     AX
```

  Si osserva che i due salti nel programma sono di tipo relativo. Il primo (JMP 010Ah) è un salto in avanti di 5 celle che corrisponde pertanto al valore dell'offset (05h) ovvero al secondo byte dell'istruzione. Si deve tenere conto che il conteggio delle celle da saltare, parte dalla cella attualmente puntata dal registro IP ovvero quella immediatamente dopo l'istruzione di salto (indirizzo 0105h). Il secondo salto è invece all'indietro, pertanto l'offset è negativo ed è di -10 celle. Tale valore lo troviamo come secondo byte dell'istruzione (F6h) codificato in complemento a 2[6].

La differenza da un punto di vista applicativo fra i due tipi di salto (e quindi di tecniche di indirizzamento è che il salto assoluto ha una "potenza" maggiore in quanto può saltare in qualsiasi punto della memoria, mentre il salto relativo, avendo dedicato all'offset solo un byte può effettuare spostamenti al max di 128

[6] Il valore F6h corrisponde in rappresentazione in complemento a 2 al valore -10, infatti se sommiamo il valore 10, ovvero 0Ah ad F6h otteniamo 00h (vedi rappresentazione binaria in complemento a due)

celle indietro e 127 in avanti. Di contro però, essendo il salto relativo codificato con istruzioni a meno byte esso occupa meno memoria ed ha una esecuzione più veloce. Ha inoltre il grosso vantaggio che i riferimenti di salto sono rilocabili[7].

- ISTRUZIONI INTEL 8086/8088 (VEDI REPERTORIO IN APPENDICE)

Il set di istruzioni Assembler della CPU 8086 o 8088 prevede la classificazione secondo le seguenti operazioni:

- TRASFERIMENTO DATI

  Effettuano trasferimenti di dati da registri interni alla CPU ad altri registri interni o a celle di memoria ovvero a porte di I/O. Tali trasferimenti possono ovviamente avvenire anche in senso inverso, ma mai da una cella di memoria ad un'altra cella di memoria o porta di I/O. Alcune di esse possono operare sui registri accumulatori sia a 16 che a 8 bit .

  Le istruzioni che appartengono a questo gruppo sono:

MOV, PUSH, POP, XCHG	(General Porpose)
IN, INW, OUT, OUTW, XLAT	(Specifiche per l'I/O)
LEA, LDS, LES	(Trasferimento Indirizzi)
LAHF, SAHF, PUSHF, POPF	(" Registro Flag)

- OPERAZIONI ARITMETICHE E LOGICHE

  Sono le istruzioni che utilizzano l'ALU, insieme a quelle logiche, e configurano di conseguenza i flag di Zero, Segno, Overflow e Carry del registro di stato.

  Oltre alle operazioni aritmetiche ordinarie, come la somma, sottrazione ecc. sono implementate anche operazioni più complesse come la moltiplicazione e la divisione, nonché operazioni per la manipolazione di numeri in forma BCD sia compattato che no (Ascii compatibile).

  Fanno parte delle istruzioni Aritmetico-Logiche:

  Aritmetiche

ADD, ADC, INC, AAA, DAA	(Risultato nell'accumulatore)
SUB, SUBB, DEC, NEG, CMP, AAS,DAS	(Sottrazioni)
MUL, IMUL, AMM	(Moltiplicazioni)
DIV, IDIV, AAD, CBW, CWD	(Divisioni)

  Logiche

NOT, SHL, SHR, SAL, SAR, ROL, ROR RCL, RCR	(Monadiche)
AND, TEST, OR, XOR	(Diadiche)

[7]Un programma si dice rilocabile quando è possibile allocarlo in memoria per l'esecuzione a partire da qualsiasi indirizzo in quanto all'interno del codice non ci sono riferimenti assoluti, ma solo relativi (vedi "RILOCABILITÀ")

- MANIPOLAZIONE STRINGHE E ITERAZIONI

Una stringa non è altro che una sequenza di caratteri, cioè un blocco di dati di un byte ciascuno depositato in memoria a partire da una certa locazione.

Le operazioni per la manipolazione delle stringhe sono di tipo elementare (primitive) ed eseguono confronti o trasferimenti. Possono essere replicate in modo iterativo tramite un opportuno PREFISSO DI ITERAZIONE ("REP" = replicate) o ponendo al termine del nucleo di iterazione opportune istruzioni di controllo ("LOOP" = ciclo), che determinano le modalità di esecuzione del ciclo iterativo.

Si capisce che la capacità di manipolazione delle stringhe offre una notevole potenza d'elaborazione, in quanto permette di trattare in blocco intere strutture di dati.

Come già predetto, i segmenti utilizzati come sorgente e destinazione del trasferimento sono identificati dai registri:

ES per la destinazione.
DS per la sorgente.

Mentre non è possibile utilizzare altri registri al posto di ES per la destinazione, è possibile identificare il segmento sorgente con qualsiasi registro di segmento, tramite l'utilizzo del Prefisso di Override (Segment Override Prefix)

La parte relativa al displacement è invece identificata dai registri Indice (SI o DI)

Sono istruzioni primitive per il trattamento delle stringhe:

MOVS, CMPS, SCAS, LODS, STOS.

Esse possono processare o un singolo byte o una singola parola per volta. Per specificare questo si aggiunge al codice mnemonico un carattere di identificazione (B o W).

P.e.	MOVSB	(Un byte per volta)
	MOVSW	(16 bit per volta)

Gli operandi di queste istruzioni sono impliciti, in quanto la situazione di default prevede:

	Reg. Segmento	Displacement
SORGENTE	DS	SI
DESTINAZIONE	ES	DI

La situazione di default può essere alterata tramite l'utilizzo del Prefisso di Override (SOP).

Le istruzioni per la manipolazione delle stringhe aggiornano automaticamente i registri indice (DI e SI), ma non vengono alterati i registri di segmento (DS e ES).

P.e. per lo spostamento di un blocco di dati lungo 67h caratteri può essere utilizzata l'istruzione MOVSB preceduta da un prefisso di ripetizione (REP).

N.B. L'informazione relativa alla lunghezza del blocco deve essere precaricata nel Registro CX.

			Esempio col "DEBUG" del DOS		
MOV	CX,67h	0100	B9 67 00	MOV	CX,0067
REP	MOVSB	0103	F3	REPZ	
		0104	A4	MOVSB	

Questa routine sposta un blocco di 67h caratteri dalla locazione puntata dai registri DS e SI alla locazione puntata dai registri ES e DI.

Ogni qualvolta che un carattere viene spostato, il registro CX viene decrementato automaticamente alla logica della CPU. Pertanto, l'operazione complessiva cessa quando il registro CX e arrivato a zero.

Oppure

			Esempio col "DEBUG" del DOS		
	MOV CX,67h	0100	B9 67 00	MOV	CX,0067
103:	**iterazione**	0103	42	INC	DX
	.....	0104	E2 FD	LOOP	103
	LOOP 103				

L’istruzione LOOP PIPPO decrementa il registro CX e salta all’etichetta "PIPPO" se tale registro è diverso da zero (itera finché CX#0). Per questo si precarica il registro CX con il valore 67h ovvero il numero delle iterazioni. Ognuna di queste ultime corrisponde all’insieme d'istruzioni comprese fra l’etichetta "PIPPO" e l’istruzione "LOOP".

- CONTROLLO DEL FLUSSO (salti e chiamate a sottoprogrammi)

  Queste istruzioni permettono il controllo della normale sequenza di esecuzione.

  Si può notare che non esistono chiamate a sottoprogrammi (istruzioni CALL e RET) di tipo condizionato. Per quanto riguarda i salti, la condizione può essere imposta solo su quelli di tipo relativo alla posizione attuale (Reg. IP) e all'interno dello stesso segmento, con Displacement ad un solo byte con segno. Pertanto possono essere saltate in modo condizionato solo 127 celle in avanti o 128 indietro.

  Con i salti di tipo assoluto (CALL e JMP) è possibile, invece, saltare a qualsiasi cella della memoria, anche al di fuori del segmento attuale.

  Appartengono a questo gruppo le istruzioni:

  JMP, J Cond., CALL, RET, LOOP, LOOPZ, LOOPNZ, JCXZ INT, INTO, IRET.

- CONTROLLO CPU

  CLC, CMC, STC, CLD, STD, CLI, STI, HLT, WAIT, ESC, LOCK.

- LINGUAGGIO ASSEMBLER

La programmazione in linguaggio macchina risulta alquanto complessa a causa della codifica poco adatta al modo col quale vengono rappresentate le singole azioni (ovvero in binario). In parziale aiuto ci viene la codifica esadecimale che ci permette una rappresentazione più sintetica. Dato però che la rappresentazione intrinseca delle istruzioni è molto schematica e non ambigua, è possibile utilizzare alcuni artifizi grazie ai quali si può programmare utilizzando un linguaggio più vicino a quello umano. Ovviamente è necessario comunque tradurre poi il programma in linguaggio macchina. Il linguaggio che ne deriva da questo tipo di programmazione è detto ASSEMBLER e sfrutta alcune facilitazioni:

- CODICI MNEMONICI
  É possibile associare ad ogni codice operativo un testo mnemonico ovvero, che richiami alla memoria l'operazione da svolgere (p.e. MOV per trasferire un dato, ADD per sommarlo, JMP per effettuare un salto ecc.). E' quindi necessario disporre di un sistema di traduzione che associ ad ogni codice OPERATIVO (binario) il relativo codice MNEMONICO. Quest'associazione è costituita dalla TABELLA DEI CODICI (vedi p.e. Tabella di traduzione dell'8080).

  Inoltre, gli operandi della istruzione possono fare riferimento direttamente al nome dei registri e agli indirizzi delle celle rappresentati in esadecimale, in decimale, ottale o binario.

  Una istruzione Assembler che trasferisce il contenuto del registro AX in quello BX, si presenterà per esempio nella seguente forma (AX→BX):

  MOV BX,AX

  L'istruzione che trasferisce il valore immediato 45h nel registro AX (45h→AX) è:

  MOV AX,45h

- ETICHETTE

  Al posto dei riferimenti fisici (p.e. indirizzi) possono essere utilizzati dei nomi (ETICHETTE)[8] col vantaggio di poter modificare il programma inserendo o eliminando istruzioni senza modificare il riferimento associato all'etichetta.

Indir	Cod Macchina	Etichetta	Istruz Assembler
0100	EB 08		JMP 010Ah
0102	89 D8		MOV AX,BX
0104	01 C8		ADD AX,CX
0106	F8		CLC
0107	8B 0F		MOV CX,[BX]
0109	F8		CLC
010A	89 C0		MOV AX,AX
010C	47		INC DI

Indir	Cod Macchina	Etichetta	Istruz assembler
0100	EB 08		JMP **ETIK**
0102	89 D8		MOV AX,BX
0104	01 C8		ADD AX,CX
0106	F8		CLC
0107	8B 0F		MOV CX,[BX]
0109	F8		CLC
010A	89 C0	**ETIK:**	MOV AX,AX
010C	47		INC DI

Nell'esempio si osservano due programmi equivalenti; il primo fa riferimento direttamente all'indirizzo 010Ah (JMP 010Ah) per poter effettuare un salto alla cella 010Ah dove si trova l'istruzione MOV AX,AX. Nel secondo caso la stessa cosa è ottenuta mettendo al posto dell'indirizzo fisico (010Ah) un nome ovvero una etichetta che lo identifica biunivocamente. Infatti, l'istruzione puntata dal salto

(MOV AX,AX)

[8] NB: Il "DEBUG" del DOS non permette l'uso di etichette

è associata al nome "ETIK" che identifica appunto il riferimento. Si deve notare che l'etichetta è seguita da "**:**" per identificare il riferimento logico. L'associazione fra etichette (riferimenti logici) e gli indirizzi (riferimenti fisici) è contenuta in una tabella chiamata TAVOLA DEI SIMBOLI e che è costruita durante l'operazione di traduzione in linguaggio macchina.
Si osserva che inserendo un'ulteriore istruzione nel programma, nel primo caso è necessario modificare l'indirizzo fisico (010Ah) in quanto l'istruzione destinazione del salto è spostata a indirizzi più alti. Nel secondo caso invece, quest'operazione non è necessaria in quanto è svolta automaticamente in fase di traduzione

- COMMENTI

É possibile inserire nel corpo del programma, dei testi che permettono il commento esplicativo di un'istruzione o di un blocco, nonché il titolo del programma stesso. Questi commenti sono posti dopo l'istruzione stessa e identificati da "punto e virgola".

Tavola dei simboli	
ETICHETTA	INDIRIZZO FISICO
ETIK	010Ah

p.e.) PIPPO: MOV AX,45h ;Trasferisce il valore 45h nel registro AX

- ALTRE AGEVOLAZIONI

A seconda del tipo di linguaggio Assembler è possibile inserire nell'istruzione semplici operazioni elementari come somme o sottrazioni che permettono l'incremento di un indirizzo fisico o logico (p.e. PIPPO+3: tre celle dopo quella identificata dall'etichetta "PIPP"); è possibile utilizzare notazioni relative a codifiche binarie, ottali, Ascii, decimali o esadecimali)

Alcuni linguaggi Assembler prevedono la possibilità di utilizzare strumenti più complessi come

- FRASI DICHIARATIVE; per informare il traduttore di certe modalità di conversione
- DIRETTIVE; comandi diretti al traduttore che eseguirà in fase di traduzione
- MACROISTRUZIONI (o MACRO); sequenze di istruzioni già tradotte il linguaggio macchina e che vengono inserite in una espansione locale del programma nel punto dove si trova la macroistruzione, nel momento della traduzione.

Tutte queste entità non sono istruzioni vere e proprie ma s specifiche direttive al programma di traduzione e non alla CPU. Esse servono per dirigere convenientemente e più efficientemente la fase di traduzione. Quindi non producono direttamente istruzioni per la CPU e sono quindi dette PSEUDO-ISTRUZIONI

NB: a parte le frasi dichiarative, le direttive e le macroistruzioni, il corpo del programma è costituito da istruzioni Assembler che hanno corrispondenza biunivoca con quelle in linguaggio macchina. Pertanto, ogni istruzione in Assembler produrrà una istruzione in linguaggio macchina. Tale caratteristica permette di rendere REVERSIBILE la traduzione; infatti, è possibile convertire un programma da linguaggio macchina in Assembler (ovviamente vengono tradotti solo i codici operativi e non etichette ne macroistruzioni ne direttive).

- SINTASSI DEL LINGUAGGIO ASSEMBLER 8086/8088

Ogni istruzione in linguaggio Assembler è costituita da quattro tipi di campi:

- Campo etichetta.

  Identifica un indirizzo logico da associare ad una cella di memoria. è un nome costituito da caratteri alfanumerici che non sia un codice mnemonico. è caratterizzata da "**:**" in coda alla stringa (solo nelle istruzioni di tipo ordinario; nelle pseudoistruzioni non vengono messi i "**:**").

- Campo codice Mnemonico.

  É una stringa di caratteri che definisce in maniera univoca un codice operativo, ovvero la funzione che l'istruzione deve eseguire.

- Campi operandi.

  Sono i campi sui quali agisce la funzione.

- Campo commento.

  É una nota aggiuntiva che può essere aggiunta alla istruzione allo scopo di chiarire meglio all'operatore la funzione svolta dalla specifica istruzione. È identificato dal carattere iniziale "**;**".

p.e. PIPPO: MOV AX,BX ; carica in AX il contenuto di BX

Esiste la possibilità di utilizzare gli operatori aritmetici somma (+) e sottrazione (-) per la definizione dell'indirizzo:

p.e. PIPPO: MOV AX,PLUTO+3

Carica nel registro AX il contenuto della terza cella dopo quella identificata della etichetta "PLUTO".

Per quanto riguarda la notazione numerica si ha:

B:	Numero Binario	(1100110B)
H:	" Esadecimale	(5AH)
:	" Decimale	(56)
O:	" Ottale	(26O)
“-”:	Stringa di caratteri	("PIPPO")

- CODIFICA IN LINGUAGGIO MACCHINA (vedi repertorio INTEL 8086)

Scorrendo il manuale del linguaggio macchina si può osservare che il primo byte che identifica una generica istruzione è composto, oltre che da cifre binarie ben definite, anche da parametri dipendenti dal tipo di istruzione stessa e dal contesto nella quale l'istruzione deve operare.

- "d" Definisce la direzione nella quale il dato deve viaggiare, ovvero se il trasferimento avviene dalla memoria ad un registro interno (d=1) oppure viceversa (d=0)

- "W" Indica la lunghezza della destinazione del dato da trasferire, ovvero se di 1 byte (W=0); viceversa se è lungo 2 byte (W=1).

- "s" Indica la dimensione di un dato immediato (Sorgente), ovvero s=0 ⇒0 bit , altrimenti s=1 ⇒ 16 bit

  NOTA: s e W possono agire in coppia nel caso di indirizzamento immediato, secondo la seguente tabella:

s:W	Sorgente	Destinaz.	Note
0 0	8 bit	8 bit	
0 1	" "	16 bit	(Sorgente estesa secondo l'aritmetica del complemento a due)
1 0	-	-	
1 1	16 bit	16 bit	

- "Reg" è un codice di tre bit che definisce il registro a 16 o a 8 bit interessato al trasferimento.

Il secondo byte, quando presente contiene i seguenti parametri:

- "Mod" Indica la presenza e il tipo di offset (8 o 16 bit)
- "Reg" Come sopra.
- "R/M" è il modo di indirizzamento, ovvero indica la composizione del displacement.

ESEMPIO

ADD 6[BX][DI],DX

Questa istruzione somma il contenuto del registro DX a quello della cella il cui indirizzo è specificato dal displacement 6+BX+DI

Il risultato viene riposto nella stessa cella dalla quale è stato prelevato uno degli operandi.
Consultando il manuale, l'istruzione in oggetto è identificata, per l'indirizzamento di tipo indiretto, dal codice operativo

000000dW	mod. reg. R/M
Byte 1	Byte 2

- direzione: registro→Memoria	⇒	d = 0
- N. Byte = 2 (Come la sorgente)	⇒	W = 1
- Offset presente a 8 bit	⇒	Mod. = 01
- Registro DX (010)	⇒	Reg. = 010
- DI+BX+Displ	⇒	R/M = 001

Da cui si ottiene il codice operativo

dW	Mod	Reg.	. R/M
00000001	01	010	001

Al codice operativo va aggiunto un Byte per l'offset (06h)

⇒ 01h, 51h, 06h (Codice Macchina)

- ESEMPIO DI PROGRAMMA IN ASSEMBLER

Questo programma genera un ritardo tramite il decremento di due registri accumulatori AX (più significativo) e BX.

Indir.	Codice Oggetto		Codice Sorgente		N. T
100 h	B8h YYh XXh		MOV	AX,XXYYh	4
103 h	BBh FFh FFh	CICLO:	MOV	CX,655354	
106 h	4Ah	LOOP:	DEC	CX	2
107 h	75h FDh		JNZ	LOOP	8/4
109 h	48h		DEC	AX	2
10A h	75h F7h		JNZ	CICLO	8/4

CICLO = 103h (tavola dei simboli)
LOOP = 106h

La durata dell'esecuzione del programma è approssimativamente di:

[(8+2)*65535+4+2+8]* XXYYh + 4
=655000*XXYYh stati interni.

P.e. XXYYh=0040h ⇒ 41 Mega stati interni

Frequenza Clock =2 MHz ⇒ Durata = 41/2 = 20 sec.

- COMPILATORI e ASSEMBLATORI

Si può constatare che l'operazione di traduzione di un programma da linguaggio Assembler a linguaggi Macchina è una operazione "MECCANICA" che non richiede particolare competenza, se non quella di conoscere le regole elementari che governano tale operazione. D'altra parte tali regole sono estremamente semplici da codificare e pertanto è possibile "automatizzare" la traduzione facendola quindi assolvere da un programma che è detto "ASSEMBLATORE". Esistono programmi che svolgono l'operazione inversa e che sono detti DISASSEMBLATORI.

Estendendo il linguaggio Assembler a forme ancora più complesse è possibile creare linguaggi più efficienti ognuno corredato di un traduttore proprio che osserva sintassi ben più complicate, ma che producono funzioni estremamente potenti. Si hanno inoltre delle righe di istruzione (STATEMENT) che utilizzano parole più vicine al linguaggio umano anche se non è possibile utilizzarlo direttamente per la sua forte ambiguità. Si sono quindi evoluti negli ultimi tempi dei linguaggi di programmazione detti di ALTO LIVELLO (per differenziarli da quelli a basso livello come l'Assembler e il linguaggio macchina) p.e. Linguaggio "C", PASCAL, BASIC, FORTRAN e moltissimi altri. I programmi che svolgono la traduzione da linguaggio ad alto livello a quello macchina sono detti in generale "COMPILATORI"

Nonostante la comodità di utilizzo di questi linguaggi ad alto livello, il linguaggio Assembler è molto spesso utilizzato per la sua immediata azione sulla CPU che viene altrimenti mascherata dalla complessità delle istruzioni dei linguaggi evoluti. Infatti una istruzione in linguaggio alto livello, per la sua potenza di esecuzione, corrisponde a varie istruzioni in linguaggi macchina. Si perde così la controllabilità diretta sulla CPU e la reversibilità della traduzione.

- FASI DI TRADUZIONE DI UN PROGRAMMA ASSEMBLER

	Codice Sorgente	
	MOV	AX,45A7h
CICLO:	MOV	CX,FFFFh
LOOP:	DEC	CX
	JNZ	LOOP
	DEC	AX
	JNZ	CICLO

L'operazione di compilazione Assembler avviene in due fasi distinte:

1. La prima fase scorre il testo del programma. Vengono immediatamente sostituiti i codici mnemonici con i codici operativi tramite la TABELLA DEI CODICI e tradotti i valori assoluti in binario. Viene, quindi, costruita la TAVOLA DEI SIMBOLI nella quale vengono riportate le etichette e i rispettivi valori man mano che vengono acquisite le definizioni. Si deve notare che la traduzione è quasi completamente effettuata; manca solamente l'inserimento dei valori fisici al posto delle etichette. Tale operazione non è possibile svolgerla durante la prima passata perché non sono noti nel momento i valori associati alle etichette non ancora incontrate e quindi indefinite. Pertanto, in questa fase al posto delle etichette vengono inseriti dei riferimenti provvisori (XX per "LOOP" e YY per "CICLO" nella figura). Tali riferimenti provvisori verranno sostituiti con i valori fisici durante la seconda passata.

Indir.	Codice Oggetto		Codice	Sorgente
000 h	B8h A7h 45h		MOV	AX,45A7h
003 h	BBh FFh FFh	CICLO:	MOV	CX,FFFFh
006 h	4Ah	LOOP:	DEC	CX
007 h	75h **XX**		JNZ	LOOP
009 h	48h		DEC	AX
00A h	75h **YY**		JNZ	CICLO

tavola dei simboli
CICLO = 103h (YY riferimento provvisorio)
LOOP = 106h (XX riferimento provvisorio)

000 h	B8h	A7h	45h	MOV	AX,45A7h
003 h	BBh	FFh	FFh	CICLO: MOV	CX,FFFFh
006 h	4Ah			LOOP: DEC	CX
007 h	75h	**FDh**		JNZ	LOOP
009 h	48h			DEC	AX
00A h	75h	**F7h**		JNZ	CICLO

2. La seconda passata utilizza la tavola dei simboli appena costruita e sostituisce i riferimenti provvisori con quelli fisici ottenendo la traduzione completa del programma.
   NB: Non conoscendo la posizione che il programma acquisirà in memoria al momento della esecuzione, esso viene tradotto presupponendo che venga allocato a partire dall'indirizzo 0000h

- SOTTOPROGRAMMI

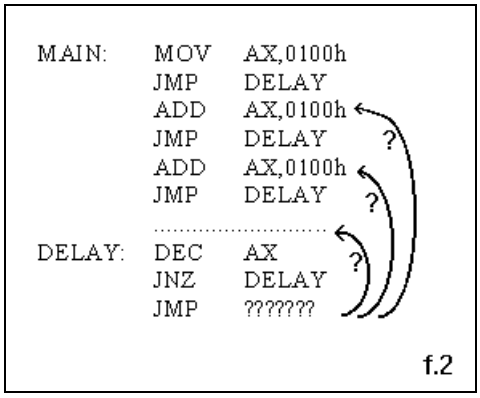
Può succedere che alcune parti di un programma debbano essere ripetute più volte, come in fig. 1. Infatti la parte relativa al ritardo (seconda e terza istruzione) deve essere eseguita 3 volte. Per risparmio di spazio e ridurre la complessità si potrebbe modificare la sequenza come in fig. 2. Ovvero riportare in coda al programma le due istruzioni che devono essere ripetute e saltando ogni volta alla prima di esse. Questa soluzione presenta però l'inconveniente che il ritorno alla parte principale non può essere svolta da un salto ordinario (JMP) in quanto per esso è previsto solo una destinazione, mentre ne servono tre diverse[9]; una per ogni "CHIAMATA" in modo da proseguire nel programma l'istruzione successiva a quella della chiamata.

```
MAIN:   MOV   AX,0100h
R1:     DEC   AX
        JNZ   R1
        ADD   AX,0100h
R2:     DEC   AX
        JNZ   R2
        ADD   AX,0100h
R3:     DEC   AX
        JNZ   R3
```
f.1

Per ovviare a questo possono essere utilizzate istruzioni apposite (dette

[9] Esiste la possibilità di rendere più versatile il salto salvando l'IP nel registro CX ed utilizzare come istruzione di ritorno JMP CX. Rimane, comunque, il problema di trasferire l'IP nel registro CX ed inoltre non è possibile la "nidificazione" delle chiamate.

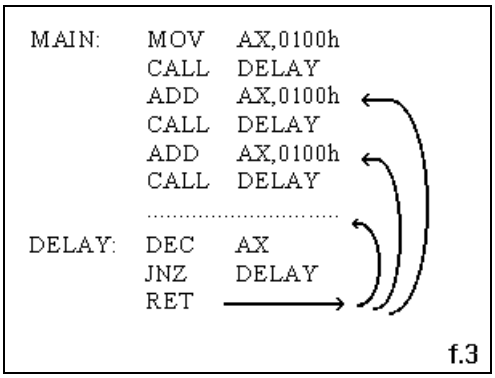
chiamate a sottoprogrammi) che memorizzano il punto di chiamata (istruzione CALL) in una zona apposita della memoria. La parte eseguita più volte è detta SOTTOPROGRAMMA, ovvero "Sequenza di istruzioni che può essere eseguita e chiamata più volte durante l'esecuzione del programma".



Il sottoprogramma a sua volta deve concludersi con un "salto" al punto memorizzato dall'istruzione di chiamata. Questo tipo di salto è svolto dall'istruzione "RET" (return - Ritorno vedi fig. 3)

Istruzione **CALL**: (le istruzioni di salto interessano i registri "IP" e "CS", ovvero i registri che puntano all'istruzione. pertanto, alla combinazione CS:IP verrà fatto riferimento con l'appellativo PROGRAM COUNTER ovvero "**PC**")

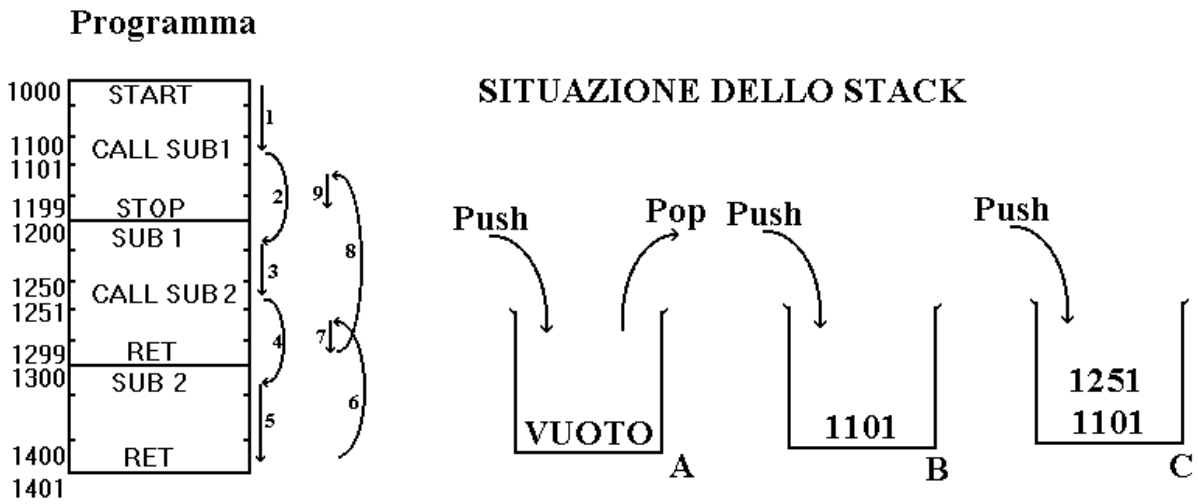
1. Salva l'indirizzo contenuto nel PC nello STACK)
2. Mette nel PC il nuovo indirizzo al quale saltare



Istruzione **RET**

1. Ripristina il PC col valore originario prima della chiamata precedentemente salvato nello STACK

La zona di memoria dove è memorizzato l'indirizzo al quale ritornare dopo l'esecuzione del sottoprogramma deve essere organizzata a STACK (LIFO) per permettere



la chiamata NIDIFICATA dei sottoprogrammi. Infatti è possibile che un sottoprogramma chiami un altro sottoprogramma per utilizzarne i servizi. A sua volta è possibile che il sottoprogramma ne chiami un altro e così via.

Sequenza di esecuzione:

1. Si parte dalla prima istruzione (START - indirizzo 1000 - stack A) del programma principale (detto MAIN) e lo stack è inizialmente vuoto. Si eseguono in sequenza le istruzioni fino a trovare la chiamata al primo sottoprogramma (CALL SUB1). L'istruzione salva il contenuto attuale del PC (indirizzo 1101 - stack B) nello stack e provvede a rimpiazzare nel PC l'indirizzo del salto (1200).
2. La prossima istruzione eseguita è quindi la prima del sottoprogramma SUB1
3. vengono eseguite in sequenza le istruzioni della SUB1 fino alla chiamata al secondo sottoprogramma (SUB 2) all'indirizzo 1250. La chiamata provvede a salvare il contenuto del PC (1251 - stack C) nello stack e lo rimpiazza col nuovo indirizzo (1300)
4. Viene quindi svolto il salto alla SUB2 all'indirizzo 1300.

5. Vengono eseguite le istruzioni della SUB2 fino alla fine dove viene incontrata l'istruzione RET (indirizzo 1400).
6. L'esecuzione della RET consiste nel prelevare l'indirizzo del salto dallo stack (1251) per metterlo nel PC in modo da eseguire l'istruzione consecutiva alla seconda chiamata. NB: Questa istruzione di RET deve prelevare l'ultimo indirizzo inserito in modo da saltare al sottoprogramma che precede quello attuale nell'ordine di chiamata. Questo evidenzia la funzione dello stack.
7. Partendo dalla istruzione immediatamente dopo la CALL SUB2 ovvero dall'indirizzo 1251 si finisce di eseguire la SUB1 fino alla relativa istruzione di RET
8. Viene ripetuta l'operazione 6) con la variante che l'indirizzo prelevato dallo stack e quindi posto nel PC è 1101, ovvero quello della istruzione seguente la CALL SUB1. Viene quindi eseguito il ritorno al programma principale
9. Il programma principale viene finito di eseguire fino all'ultima istruzione (indirizzo 1100)

I sottoprogrammi sono entità simili alle MACROISTRUZIONI ma differiscono da essi in quanto la loro chiamata è diretta alla CPU e non all'assemblatore. Infatti mentre nelle "MACRO" vengono inserite fisicamente le istruzioni nel punto di chiamata, le istruzioni del sottoprogramma si trovano invece normalmente in coda la programma principale. Se le MACRO evitano il tempo perso per la chiamata (in quanto le istruzioni la sostituiscono in toto) i sottoprogrammi hanno il vantaggio di permettere il risparmio di memoria in quanto non vengono replicati come le MACRO per ogni chiamata.

Entrambi hanno comunque il vantaggio di rendere più leggibile e quindi manutenibile il programma.

- DIRETTIVE ALL'ASSEMBLATORE (INTEL 8086)

Le direttive (Pseudoistruzioni) per l'assemblatore sono dei comandi per la produzione di dati con certe modalità (p.e. per la disposizione più opportuna del programma in memoria ecc.) Le direttive pertanto non producono niente direttamente in codice macchina.

NOTA: Le etichette associate alle direttive, non vengono fatte seguire dai due punti (:), come accade per le istruzioni di tipo ordinario.

- ORG nnnn (Origin).

  Informa l'assemblatore che la procedura in linguaggio macchina deve essere allocata in memoria a partire dalla cella "nnnn", che identifica anche l'indirizzo della prima istruzione da eseguire.

```
ORG  562Ah
```

- EQU nnnn (Equal).

  Questa direttiva associa ad una etichetta un valore ben preciso, ovvero il valore nel campo operando.

```
PIPPO  EQU 56DEh
```

  Fa si che all'etichetta "PIPPO" sia associato l'indirizzo 56DEh.

- DB nn (Data Byte)

  Permette di allocare un carattere in una cella di memoria.

PIPPO DB DAh

Carica nella cella associata all'etichetta "PIPPO" il carattere "DAh".

Con questo tipo di direttiva è possibile anche definire in memoria interi blocchi di caratteri.

PIPPO DB "TIZIO, CAIO, SEMPRONIO"

Nell'esempio, un blocco costituito da 22 caratteri viene definito in memoria a partire dalla cella identificata dalla etichetta "PIPPO".

- DW nnnn (Data Word)

Funziona come la direttiva precedente ma permette di definire dati di due byte (Word a 16 bit).

PIPPO DB E456h

NOTA: è possibile, quando necessita disporre in memoria una sequenza di dati tutti uguali, utilizzare l'operatore

- n DUP(x)

Dove "**n**" è il numero di ripetizioni e "**x**" è il dato da ripetere.

PIPPO DB 100 DUP(0) oppure PIPPO DW 50 DUP(0)

Queste pseudoistruzioni caricano in memoria una sequenza di 100 "0" a partire dalla cella identificata dalla etichetta "PIPPO".

Si deve notare che le due direttive (DB e DW), pur producendo lo stesso effetto in memoria, ovvero il caricamento della sequenza di 100 celle azzerate, assegnano alla etichetta "PIPPO" ATTRIBUTI differenti. Infatti, nel caso che l'operazione sia fatta tramite "DB", "PIPPO" ha un attributo di tipo "BYTE", mentre nell'altro caso ha un attributo di tipo "WORD".

L'attributo assegnato ad una etichetta è essenziale per la generazione del codice macchina.

ADD PIPPO,5

Dispone il risultato su 8 o 16 bit a seconda del tipo di attributo associato all'etichetta "PIPPO".

- DD nnnnnnnn (Double Data Word)

Permette di definire dati a 4 byte.

PIPPO DB DCE52346h

- END (Fine Procedura)
Identifica la fine fisica della procedura. Può presentare un campo operando che è l'etichetta associata alla prima istruzione della procedura da eseguire.

END PIPPO

- DIRETTIVE PER LA IDENTIFICAZIONE DEI SEGMENTI

Il programma sorgente deve contenere una parte di PREDEFINIZIONE della segmentazione, ovvero della definizione in memoria dei vari spazi associati alla procedura, ai dati e allo stack.

L'operazione di predefinizione consiste nell'assegnare gli opportuni valori ai registri di segmento (CS, DS, ES, SS), che identificano l'indirizzo iniziale delle rispettive aree di lavoro.

Per fare ciò vengono utilizzate delle apposite direttive:

- PROC e ENDP (per la procedura).
- SEGMENT e ENDS (Per le altre aree).

- DIRETTIVE PROC E ENDP

La direttiva "PROC" identifica l'inizio della procedura (Mentre "ENDP" identifica la fine) e ne definisce certi attributi, come la sua allocazione all'interno o all'esterno del segmento ove risiede il programma chiamante.

Per identificare l'allocazione, la direttiva PROC dispone di un parametro che può essere

- NEAR (Modo Intrasegmentale)
- FAR (Modo Intersegmentale)

Col parametro "NEAR" la procedura si trova nello stesso segmento ove risiede il programma chiamante, mentre se si trova all'esterno del segmento viene utilizzato il parametro FAR (p.e., nel caso di una procedura principale, il programma chiamante è il sistema operativo, e pertanto, essendo, il modo quasi sempre di tipo Intersegmentale, viene pertanto utilizzato il parametro "FAR").

Il parametro "NEAR" (o "FAR") stabilisce anche certi attributi per le etichette che compaiono nella procedura, nel senso che le etichette di una procedura "FAR" sono di tipo Intersegmentali, mentre quelle di una procedura di tipo "NEAR" sono di tipo Intrasegmentali

La differenza fra i due tipi di attributi, è utilizzata dall'assemblatore per la codifica in linguaggio macchina, in quanto le etichette di tipo "NEAR" possono trovarsi solo all'interno del segmento, mentre quelle di tipo "FAR" possono essere dei riferimenti anche al di fuori del segmento ove risiede la procedura.

N.B. La definizione "NEAR" o "FAR" può essere NIDIFICATA, e in questo caso, la definizione della procedura più interna definirà l'attributo delle etichette all'interno di essa.

- DIRETTIVE SEGMENT E ENDS

Queste direttive funzionano in modo simile a quelle già viste, e permettono di dividere il programma in segmenti.

I segmenti che devono essere utilizzati dal programma sono:

- Segmento della procedura (CS).
- Segmento dei dati (DS).
- Segmento dello Stack (SS).

La direttiva "SEGMENT" permette di associare un segmento ad una etichetta che lo identificherà univocamente.

Può avere alcuni parametri come:

- "PARA" (Paragrafo); In questo caso il segmento inizia ad una locazione standard ovvero ad un indirizzo multiplo di 16.
- "STACK" ; Identifica la definizione del segmento di stack.
- "'PUBLIC" ; Identifica la definizione del segmento dei dati o della procedura.

```
p.e.        STACK   SEGMENT  PARA       'STACK'
            DATA    SEGMENT  PARA PUBLIC 'DATA'
            CODE    SEGMENT  PARA PUBLIC 'CODE'
```

Con queste direttive, il segmento di codice (Procedura), è identificato dalla etichetta "CODE", quello di dati dall'etichetta "DATA" e quello dello stack dall'etichetta "STACK". Tutti e tre i segmenti iniziano a indirizzi standard, ovvero multipli di 16.

- DIRETTIVA ASSUME

Questa direttiva serve per inizializzare i registri di segmento all'interno della procedura, per farli puntare ai relativi segmenti.

P. Es PIPPO ASSUME CS: CODE

Carica nel registro CS l'indirizzo base del segmento di codice (Procedura),ovvero quello associato alla etichetta "CODE".

NOTA: Il segmento di stack non richiede nessuna istruzione "ASSUME"

- ESEMPIO DI SCRITTURA DI UNA PROCEDURA IN ASSEMBLER INTEL 8086.

Riprendiamo il programma generatore di ritardo e "Confezioniamolo" per la compilazione tramite l'assemblatore "MASM".

```
1.    STACK  SEGMENT   PARA STACK  'STACK'   ; DEFINISCE L'AREA DI
2.           DB        1024 DUP (0FFh)       ; STACK DI 1K ASSOCIANDO
3.    STACK  ENDS                            ; L'ETICHETTA. "STACK"


4.    DATA   SEGMENT   PARA PUBLIC 'DATÀ     ; DEFINISCE L' AREA DI
5.    PIPPO  DW        0040h                 ; DATI A PARTIRE DALLA
6.    DATA   ENDS                            ; ETICHETTA "DATA"

7.    CODE   SEGMENT   PARA PUBLIC 'CODÉ
```

```
8.    MAIN   PROC     FAR              ; QUESTA PARTE E' DI
9.           ASSUME   CS: CODE         ; TIPO STANDARD. SERVE
10.          PUSH     DS               ; PER L'ASSEGNAZIONE
11.          MOV      AX,0             ; DEI VARI REGISTRI DI
12.          PUSH     AX               ; SEGMENTO E PER SALVARE
13.          MOV      AX,DATA          ; L'INDIRIZZO DELLA
14.          MOV      DS,AX            ; AREA DI "PSP" PER IL
15.          ASSUME   DS:DATA          ; RITORNO AL SIST. OPERATIVO.

16.          MOV      BX,DS:PIPPO      ; PROCEDURA PER IL
17.          MOV      AX,[BX]          ; RITARDO. TRASCORSO
18.   CICLO: MOV      CX,65535         ; IL RITARDO RIPASSA
19.   LOOP:  DEC      CX               ; IL CONTROLLO AL
20.          JNZ      LOOP             ; SISTEMA OPERATIVO.
21.          DEC      AX
22.          JNZ      CICLO
23.          RET

24.   MAIN   ENDP                       ; CHIUSURA DEL SEGMENTO
25.   CODE   ENDS                       ; DELLA PROCEDURA.

26.          END      MAIN             ;
```

Per come sono stati definiti i vari segmenti, la procedura completa viene riposta in memoria dal sistema operativo, a partire dal segmento di stack, seguito dal segmento dati e seguito a sua volta dal segmento della procedura vera e propria.

Questa disposizione è quella più corretta, perché permette di definire anticipatamente tutte le etichette relative ai dati, e quindi i rispettivi attributi, che serviranno nella fase di compilazione successiva.

NB: nel gruppo di istruzioni da 8 a 15, si salva il contenuto del registro DS che è stato inizializzato dal S.O. (MS/DOS) per puntare alla prima cella dell'area PSP (cella di inizio segmento di codice; La procedura vera e propria inizia al displacement=100h). Tale cella contiene l'istruzione INT 20 che serve per il ritorno al sistema operativo. Senza tale istruzione il programma potrebbe causare dei malfunzionamenti.

- COMPILATORE, LINKER, LOADER E RILOCABILITÀ D'UN PROGRAMMA

Programma principale (MAIN.ASM)

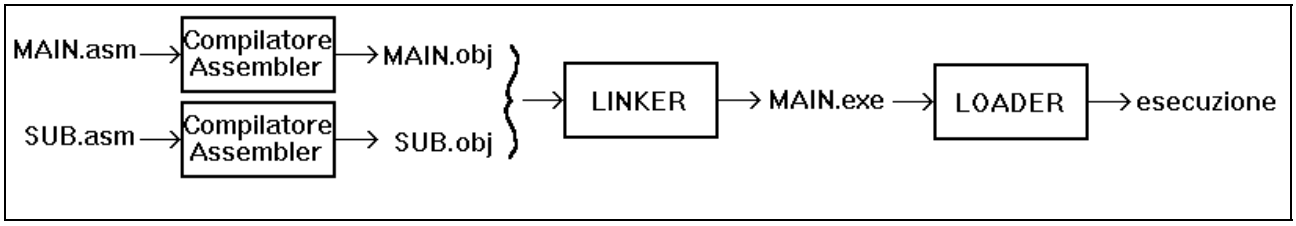
```
0000  B8 7A 46        MOV  AX,467Ah
0003  E8 ?? ?? CICLO: CALL DELAY (????h)
0006  48              DEC  AX
0007  75 [FA]         JNZ  CICLO (0003h)
```

Sottoprogramma (SUB. ASM)

```
0000  B9 FF FF DELAY: MOV  CX,FFFFh
0003  49       LOOP:  DEC  CX
0004  75 [FD]         JNZ  LOOP (0003h)
0006  C3              RET
```

Consideriamo i due programmi a lato (principale e sottoprogramma) che costituiscono ognuno un file autonomo (MAIN.ASM e SUB.ASM). Notare che i file contenenti programmi in Assembler hanno l'estensione .ASM (programmi SORGENTE). Se essi vengono compilati separatamente ognuno dei due produrrà un file a parte con l'estensione .OBJ (programmi OGGETTO). Si osserva che entrambi vengono tradotti presupponendo che

per l'esecuzione partano dall'indirizzo 0000h. Ovviamente ciò non è possibile, in quanto almeno uno dei due sarà in coda all'altro; per esempio SUB.obj sarà allocato a partire dalla cella 0009h (sempre



ammesso e non concesso che MAIN.obj sia allocato a partire dalla cella 0000h). Quindi nella compilazione si deve tenere conto che i valori fisici associati ad alcune etichette potranno cambiare nel momento in cui il programma verrà caricato in memoria per l'esecuzione (p.e. i riferimenti fisici associati ai salti assoluti. D'altra parte ci sono delle etichette il cui riferimento fisico non dipende dalla posizione del programma in memoria, p.e. le etichette dei salti relativi. Tali etichette sono dette RILOCABILI. Quando un programma oggetto contiene tutti riferimenti relativi ovvero etichette rilocabili il programma è detto RILOCABILE ed è possibile allocarlo in qualsiasi punto della memoria senza nessun intervento dopo la compilazione. Notare che nel programma MAIN l'etichetta CICLO è rilocabile, mentre non lo è l'etichetta DELAY. D'altra parte l'etichetta DELAY è rilocabile nel sottoprogramma SUB. Quindi mentre SUB è un programma rilocabile, non lo è MAIN.

```
Programma dopo il "LINKAGGIO"(MAIN.EXE)

0000 B8 7A 46            MOV   AX,467Ah
0003 E8 [09 00] CICLO:   CALL  0009h(DELAY)
0006 48                  DEC   AX
0007 75 [FA]             JNZ   0003h(CICLO)
0009 B9 FF FF   DELAY:   MOV   CX,FFFFh
000C 49           LOOP:  DEC   CX
000D 75 [FD]             JNZ   000Ch(LOOP)
000F C3                  RET
```

Inoltre potremo avere delle etichette che fanno riferimento a indirizzi esterni al programma oggetto. Un esempio è l'etichetta "DELAY" che pur trovandosi nel programma principale MAIN si riferisce al nome del sottoprogramma SUB. è ovvio che tale etichetta sarà lasciata in sospeso dalla compilazione in attesa di conoscere la posizione in memoria del sottoprogramma.

É quindi necessario operare dopo la compilazione di tutti i moduli allo scopo di allocarli in sequenza ed eventualmente agganciarli tramite i mutui riferimenti. Questa operazione è svolta da un programma detto LINKER. L'azione del Linker sui due moduli MAIN.obj e SUB.obj produce il file a lato.

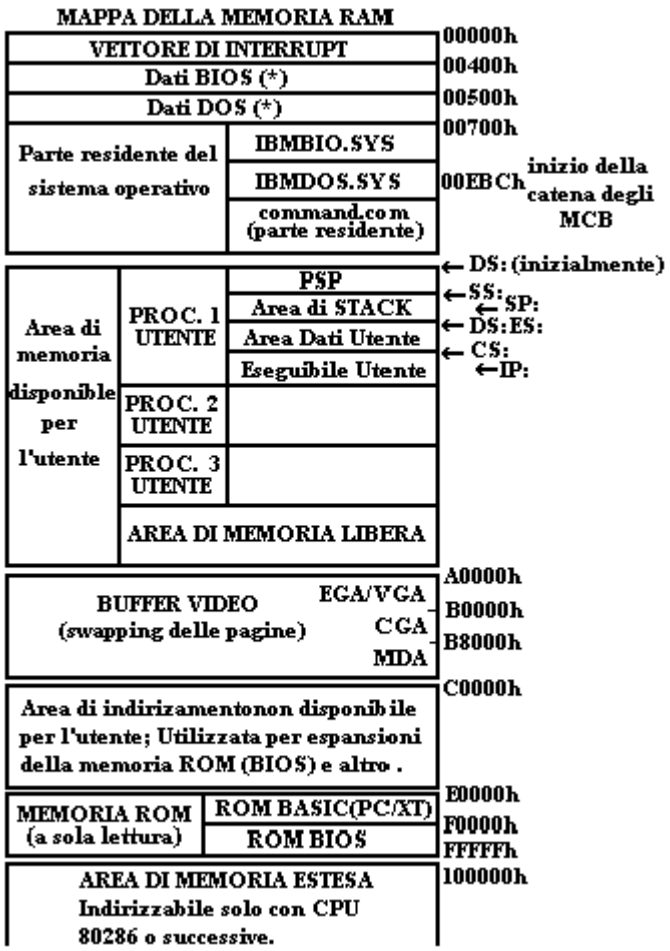
Si deve notare che il valore associato all'etichetta DELAY è 0009h, ma esso è noto solo dopo l'operazione di aggancio da parte del LINKER.

Una etichetta non rilocabile ha un riferimento assoluto associato che varia in funzione della posizione della cella alla quale si riferisce e quindi dalla posizione di allocazione in memoria del blocco contenente la cella (programma o blocco di dati). Supponiamo che in fase di compilazione una etichetta non rilocabile sia temporaneamente associata ad un indirizzo assoluto (p.e. 0100h) ovvero presupponendo che il programma o il blocco dati sia allocato a partire dall'indirizzo 0000h. Una allocazione diversa del blocco, p.e. a partire dalla cella 035Ah, provoca uno spostamento di pari valore della cella puntata dall'etichetta. Quindi anche il riferimento assoluto relativo all'etichetta deve essere incrementato di questo valore (nell'esempio è 0100h+035Ah=045Ah). L'aggancio tramite il LINKER prevede quindi l'incremento (o il decremento) quindi dei riferimenti fisici associati alle etichette non rilocabili del valore pari all'indirizzo della prima cella del blocco allocato in memoria.

```
Programma dopo il caricamento in memoria

015A B8 7A 46          MOV     AX,467Ah
015D E8 [63 01]  CICLO:CALL    0163h(DELAY)
0160 48                DEC     AX
0161 75 [FA]           JNZ     015Dh(CICLO)
0163 B9 FF FF          MOV     CX,FFFFh
0167 49          DELAY:DEC     CX
0168 75 [FD]           JNZ     0167h(DELAY)
```

Il LINKER produce un file eseguibile (MAIN.EXE) che verrà riposto in memoria di massa.
Si deve ancora tenere conto che nel momento della esecuzione il programma dovrà essere allocato in memoria. Quest'operazione non è del tutto banale, perché, ancora una volta non è nota a priori il punto di allocazione del programma eseguibile. Dato che il LINKER ha presupposto che il programma eseguibile venga allocato a
partire dall'indirizzo 0000h è necessario incrementare ulteriormente i riferimenti fisici associati alle etichette rilocabili in modo da accordare i puntamenti. P.e. se il programma MAIN.exe sarà allocato a partire dalla cella 015Ah l'etichetta "DELAY" (rif. fisico 0009h) che non è rilocabile assumerà il riferimento fisico 0009h+015Ah=0163h). Quest'ultima operazione è svolta da un altro programma dello "LOADER" il quale svolge anche altri compiti come il passaggio del controllo della CPU al programma eseguibile (MAIN.exe).



- *MAPPA DELLA MEMORIA CONVENZIONALE IN UN PC IBM (O COMPATIBILE)*

La memoria convenzionale di un PC ha una estensione di 1 MB e, partendo dall'indirizzo fisico 00000h arriva fino all'indirizzo FFFFFh. Oltre tale indirizzo si ha la memoria estesa che però non è gestita dalla CPU Intel 8086 in quanto sono disponibili solo 20 linee di indirizzo. Può comunque essere utilizzata dalle CPU Intel 80286 e successive. La memoria convenzionale non è disponibile completamente per l'utente, ma solo una parte di essa. Infatti la parte oltre 640K è riservata alla memoria video, alla memoria a sola lettura (BIOS) e a parti accessorie. Inoltre, della parte inferiore una quota consistente è occupata dal vettore di INT e (se il sistema è gestito da un S.O. MS/DOS) da alcuni parametri e dati vari relativi alla configurazione attuale del sistema. Questa ultima viene caricata in RAM al momento dell'accensione (BOOTSTRAP).

N.B: Il sistema operativo, subito prima della procedura completa, riserva una zona di controllo, detta "PSP" che permette il ritorno al sistema operativo, una volta che la procedura è stata eseguita. Al momento della esecuzione, il codice di segmento dati (DS) punta al segmento PSP, perciò la sua memorizzazione nello stack, provoca al ritorno il salto alla cella iniziale del PSP, ove si trova l'istruzione "INT 20h", corrispondente alla routine di ritorno al sistema operativo.

L'area di PSP, permette anche il passaggio dei parametri alla procedura da eseguire.

* NOTA : per maggiori dettagli di come viene sfruttata questa area vedere "PS/2 & PC IBM", Guida del Programmatore di Peter Norton, C.E. Jackson, pag. 58

- CPU INTEL 80286

La CPU 286 differisce dal suo predecessore (8086) per alcune prestazioni aggiuntive, come la capacità di operare più velocemente non solo per l'effetto acceleratore della più sostenuta velocità del clock (fino a 25 MHz), ma anche per l'efficienza più spinta con la quale è stata progettata (Con freq. di clock di 10 MHz si ottengono velocità 6 volte superiori a quelle ottenute con la CPU 8086 funzionante a 5 MHz).

Il bus dati è identico a quello dell'8086 (16 linee), ma è capace di pilotare 24 linee di indirizzo, disponendo così di una capacità di indirizzamento di 16 Mbyte. Inoltre il bus non è multiplexato, ed il chip è fornito di 68 pin.

La INTEL, per mantenere la compatibilità con la CPU 8086, in modo da poter utilizzare tutto il software per essa sviluppato (a cominciare da sistema operativo DOS), ha predisposto il "286" di due modi operativi:

- Modo REALE (Real Mode), che permette l'emulazione dell'8086, limitando così la capacità di indirizzamento ad 1 Mbyte di celle di memoria. Questo è praticamente il modo di funzionamento della CPU 286 con il sistema operativo "DOS".
- Modo PROTETTO o Virtuale (Protect Mode), col quale è possibile utilizzare a pieno le prestazioni della CPU. Questo modo di funzionamento permette di sfruttare efficientemente il meccanismo della Memoria Virtuale rendendo possibile già a livello Hardware l'isolamento fra le varie partizioni (Task) nelle quali risiedono processi diversi. Questo sistema può però funzionare solo sotto la supervisione di un sistema operativo più evoluto del DOS, come per esempio OS/2 o UNIX.

L'architettura interna di questa CPU è principalmente uguale a quella dell'8086, salvo alcuni particolari. Ha gli stessi registri dell'8086, ma varia solamente la configurazione del registro di stato. Questo è formato dagli stessi flags dell'8086 (ovvero vedi sotto)

- CF (Carry)
- AF (Auxiliary Carry)
- DF (Direction)
- IF (Interrupt Enable)
- OF (Overflow)
- PF (Parity)
- SF (Sign)
- TF (Trap)
- ZF (Zero)

ma ha in più i flags:

- NT (Nexted Task). Usato solo in modalità protetta, stabilisce se il processo in esecuzione dispone o no di un puntatore valido ad un altro processo concatenato, in attesa, o se gestisce direttamente l'hardware, eventualmente per conto di un altro processo.
- IOPL (Input/output Privilege Level). Usato solo nel modo protetto e specifica il massimo livello di privilegio col quale il processo attualmente in esecuzione può eseguire istruzioni di I/O.
- ET (Processor Extension Type). Indica il tipo di estensione del processore.
- TS (Task Switch). Viene posto a 1 dalla CPU quando il controllo della unità centrale passa in modo sincrono da un processo ad un altro.
- EM (Processor Extension Emulated). Indica alla CPU che le funzioni del coprocessore matematico devono essere emulate da software.
- MP (Monitor Processor Extension). Dichiara la presenza del Coprocessore matematico (Chip complementare per lo svolgimento dei calcoli matematici).

- PE (Protection Enable). Identifica la modalità di funzionamento (PE=1 ⇒Protect Mode, PE=0⇒Real Mode)

N.B. ET,TS,EM,MP,PE sono flags che identificano lo stato del sistema, pertanto possono essere considerati come un registro a parte (16 bit della parte alta del registro flags). Ad esso viene dato il nome di parola di "Stato della macchina", e viene indicata con MSW (Machine Status Word).

Nel caso di funzionamento in modalità protetta, i registri di segmento, identificano un selettore in una tabella che contiene le informazioni relative ai vari processi in attesa di esecuzione.

L'80286, rispetto all'8086 ha in più una serie di registri di sistema che vengono utilizzati solo in modalità protetta. La loro funzione è l'indirizzamento di alcune tabelle contenenti le informazioni per la allocazione dei segmenti in memoria RAM.

- GDTR (48 bit) Global Descriptor Table.
- IDTR (" ") Interrupt Descriptor Table.

Questi registri a 48 bit sono costituiti da due parti, ovvero una parte base (32 bit) e l'altra (16 bit) contenente la dimensione per le aree di GDT e IDT.

- LDTR (16 bit) Local Descriptor Table.
- TR (" ")

Questi due registri contengono il selettore di 16 bit ai segmenti LDT e TSS del processo attualmente in esecuzione.

- TSS ( ) Task State Segment

Quest'ultimo registro contiene le informazioni sullo stato del processo attualmente in esecuzione.

- INDIRIZZAMENTO IN MODALITÀ PROTETTA

Il sistema di traduzione da indirizzo logico a fisico nel modo protetto è diverso da quello visto per l'8086 o per lo stesso 80296 in modalità protetta.

Con questa modalità operativa ogni registro di segmento seleziona la riga di una tabella di descrittori del segmento stesso, nella quale è registrato l' indirizzo di segmento (24 bit), e con esso anche altri dati relativi al processo residente nel segmento stesso (attributo del segmento, dati di programma, sola lettura, livello di priorità del processo, ecc.).

P.e. indirizzo 0038:4321

0038h registro di segmento
4321h offset o displacement

0038 punta ad una riga della tabella dei descrittori, nella quale è riportato il valore 012340h a 24 bit .

0038h → 012340h (Indirizzo base del segmento)

La cella indirizzata si trova alla locazione di indirizzo fisico ricavato dalla somma fra l'indirizzo base (012340h) trovato nella tabella e l'offset (4321h), ovvero

```
012340h+
   4321h=
----------
016661h
```

Con questo sistema è possibile far partire un segmento da una qualsiasi delle 16 M celle di memoria ed è lungo sempre 64 Kbyte.

- CPU INTEL 80386

Questa CPU detiene tutte le prestazioni della CPU 80286, ma ha inoltre una architettura leggermente diversa. ha infatti registri a 32 bit invece che a 16, anche se il tipo dei registri è essenzialmente lo stesso, e ciò gli permette di gestire efficientemente un bus dati di 32 bit. Anche il bus indirizzi è ampliato, ed è costituito da 32 linee portando così 4 Gbyte la capacità di indirizzamento (in modo protetto).

L'architettura del sistema sfrutta un meccanismo di PIPELINE, ovvero di sovrapposizione temporale di molteplici funzioni, che permette di svolgere contemporaneamente più compiti diversi. Questo grazie alla presenza di più stadi di esecuzione delle istruzioni svolti in parallelo da unità dedicate.

Dispone inoltre di un sistema di bus con una più ampia banda passante, e di un sistema di decodifica hardware degli indirizzi logici in fisici (che sfrutta una memoria Cache interna), e ciò le permette di ridurre considerevolmente i tempi di esecuzione raggiungendo velocità dell'ordine di 3-4 MIPs (Milioni di istruzioni al secondo).

Possiede internamente un sistema di autodiagnosi e di 4 registri di break-point, nonché della possibilità di accedere da software al sistema di decodifica di indirizzi logici in fisici.

L'architettura interna dell'80386 prevede tutti i registri della CPU 80286, con alcune varianti più altri registri specifici. Infatti sono presenti :

- - Registri generali, AX, BX, CX, DX, DI, SI, BP e SP, che indicano lo spiazzamento rispetto all'inizio del segmento. Dato che questi registri sono a 32 bit e possono essere utilizzati con capacità variabile di 8, 16 e 32 bit , è possibile dimensionare in modo variabile la lunghezza di ogni segmento, ovvero di 256, 64K 4 G byte. Per identificare la lunghezza di ogni registro, si deve specificare, nella istruzione, il nome preceduto da un prefisso come segue:

32 bit	16 bit	8 bit
EAX	AX	AH AL
EBXBXBHBL		
ECXCXCHCL		
EDXDXDHDL		
ESPSP--		
EDIDI--		
ESISI--		
EBPBP--		

- Registri di segmento, di 16 bit sono identici a quelli del 286 (CS,SS,DS,ES), ai quali si aggiungono altri due registri (FS e GS) di 16 bit per l'uso più flessibile delle istruzioni di accesso a strutture dati da base e indice.

- Registri EIP (Instruction Pointer) e EFLAGS. Anche questi registri sono simili a quelli della CPU 80286 ma sono a 32 bit ed inoltre sono presenti due flags in più per la gestione del modo protetto:

  VM : Flag per la modalità 8086 virtuale
  RF : Flag di ripristino del controllo di correttezza (Resume). Vedi meglio "Assembler 80286/386" pg.71.

- Registri di sistema, esattamente uguali a quelli visti per l'80286.

- Registri di controllo: Sono 3 registri a 32 bit , dei quali solo uno di essi è parzialmente utilizzato (CR0). Gli altri (CR1 e CR2) sono stati previsti per l'evoluzione futura della architettura interna. La loro funzione è quella di rappresentare lo stato del sistema relativo a situazioni indipendenti dal processo attualmente in esecuzione, e possono essere accessibili tramite le istruzioni "LOAD" e "STORE". I flags presenti in CR0 sono:

  ET,TS,EM,MP,PE, già visti per il 286.

  NB. La parte bassa di CR0 è la "Machine Status Word" (MSW), in modo che in modalità protetta, l'80386 sia pienamente compatibile con l'80286, alla quale vi si può accedere tramite le istruzioni LMSW e SMSW presenti sia sul 286 che sul 386.

- Registri di debug e di ricerca. Sono 10 registri a 32 bit e vengono indicati con DR0, ..., DR7, TR6 e TR7.

  I primi 8 sono per il debug, e fra questi i DR4 e DR5 sono utilizzati direttamente dal costruttore (Intel) per scopi interni. Gli ultimi due invece sono utilizzati per la ricerca (TR6 e TR7) e servono per verificare il contenuto della RAM e della memoria Associativa CAM (Content Addressable Memory); in particolare vengono sfruttati per la traduzione degli indirizzi logici in fisici.

  La funzione di ogni registro è la seguente:

DR0 :	Indirizzo lineare breakpoint 0
DR1 :	" " " 1
DR2 :	" " " 2
DR3 :	" " " 3
DR4 :	Riservato Intel
DR5 :	" "
DR6 :	Stato breakpoint
DR7 :	Controllo breakpoint
TR6 :	Controllo ricerca
TR7 :	Stato ricerca

  L'indirizzamento in modalità protetta è lo stesso di quello per la CPU 286, ma con la sola differenza che l'indirizzo base registrato nella tabella dei descrittori è a 32 bit e che l'offset registrato nei registri

generali è a lunghezza variabile e può raggiungere la dimensione massima di 32 bit, potendo così avere segmenti lunghi fino a 4 Gbyte e che possono partire da qualsiasi indirizzo base.

APPENDICE

- MODELLO INTEL 8086

AX:	**AH**	**AL**	**accumulator**
BX:	**BH**	**BL**	**Base**
CX:	**CH**	**CL**	**Countat**
DX:	**DH**	**DL**	**Data**
	SP		**Stack Pointer**
	BP		**base Pointer**
	SI		**Source Index**
	DI		**Destination Index**
	IP		**Instruction Pointer**
	FLAGS H	**FLAGS L**	**Status Flags**
	CS		**Code Segment**
	DS		**Data Segment**
	SS		**Stack Segment**
	ES		**Extra Segment**

r/m	Indirizzo Operando
000	[BX] + [SI] + DISP
001	[BX] + [DI] + DISP
010	[BP] + [SI] + DISP
011	[BP] + [DI] + DISP
100	[SI] + DISP
101	[DI] + DISP
110	[BP] + DISP (segue secondo byte dell'istruzione prima del dato se richiesto)
111	[BX] + DISP

Campo "reg" (assegnamento)

16 - bit (W=1)	8 - bit (W=0)	SEGMENTO
000AX	000AL	00ES
001CX	001CL	01CS
010DX	010DL	10SS
011BX	011BL	11DS
100SP	100AH	
101BP	101CH	
110SI	110DH	
111DI	111BH	

SECONDO BYTE DELLA ISTRUZIONE

mod	XXX	r/m

mod	DISPLACEMENT
00	DISP=0 *: Disp Low and DISP High sono assenti (* escluso se mod=00 e r/m = 100 allora EA = DISP-High:DISP-Low)
01	DISP-Low sign-extended to 16-bit, DISP-High sono assenti
10	DISP= DISP-High: DISP-Low
11	r/m è trattato come "reg" field

AF:	**Auxliary Carry - BCD**
CF:	**Carry Flag**
PF:	**Parity Flag**
SF:	**Sign Flag**
ZF:	**Zero Flag**
DF:	**Direction Flag (string)**
IF:	**Interrupt Enable Flag**
OF:	**Overflow Flag (CF xor SF)**
TF:	**Trap Flag - Single Step Flag**

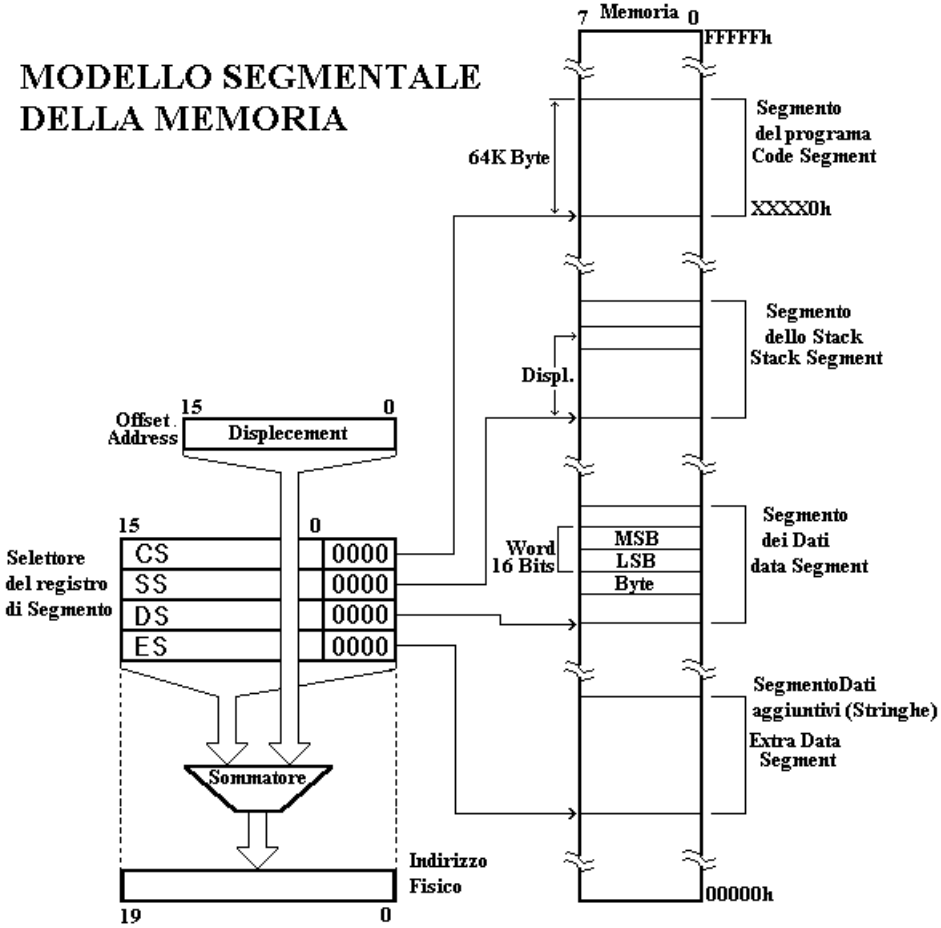
STATUS FLAGS

X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

SEGMENT OVERRIDE PREFIX
0 0 1 reg 1 1 0

Uso del Segment Override Segment

REGISTRO OPERANDO	Default	Con Override Prefix
IP (Indirizzamento Codice programma)	CS	mai
SP (Indirizamento Stack)	SS	mai
BP (Indirizamento Stack o Stack Maker))	SS	BP+DS o ES, o CS
SI o **DI** (escluso Stringhe)	DS	ES, SS, o CS
SI (Indirizzo sorgente implicito per le stringhe)	DS	ES, SS, o CS
DI (Indirizzo destinazione implicito per le stringhe)	ES	mai



REPERTORIO ISTRUZIONI INTEL 8086

			1° BYTE	2° BYTE	3° BYTE	4° BYTE	5° BYTE
	ISTRUZIONI DI TRASFERIMENTO						
MOV	MOVE (SPOSTA)	REG/MEM to/from Register	100010 d W	mod reg r/m			
		IMMEDIATE to REG/MEM	1100011 W	mod 000 r/m	DATA	DATA if W=1	
		IMMEDIATE to Register	1011W Reg	DATA	DATA if W=1		
		MEMORY to Accumulator.	1010000 W	ADDR-Low	ADDR-High		
		Accumulator To Memory	1010001 W	ADDR-Low	ADDR-High		
		REG/MEM to segment Register	10001110	mod 0 reg r/m			
		Segment reg. To REG/MEM	10001100	mod 0 reg r/m			
PUSH	PUSH	REG/MEM	11111111	mod 110 r/m			
		Register	01010 reg				
		Segmente Register	000 reg 110				
POP	POP	REG/MEM	10001111	mod 000 r/m			
		Register	010111 reg				
		Segment Reg.	000 reg 111				
XCHG	EXCHANGE (SCAMBIA)	REG/MEM with Register	1000011 W	mod reg r/m			
		Register with Accumulator	10010 reg				
IN	INPUT TO AL/AX FROM (INGRESSO)	Fixed PORT	1110010 W	PORT			
		variable PORT (DX)	1110110 W				
OUT	OUTPUT FROM AL/AX TO (USCITA)	Fixed PORT	1110011 W				
		Variable PORT (DX)	1110110 W				
XLAT	TRANSATE BYTE TO AL		11010111				
LEA	LOAD EA TO REGISTER		10001101	mod reg r/m			
LDS	LOAD POINTER TO DS		11000101	mod reg r/m			
LES	LOAD POINTER TO ES		11000100	mod reg r/m			
LAHF	LOAD AH WITH FLAGS		10011111				
SAHF	STORE AH INTO FLAGS		10011110				
PUSHF	PUSH FLAGS		10011100				
POPF	POP FLAGS		10011101				
	ISTRUZIONI ARITMETICHE						
ADD	ADDITION	REG/MEM with Register to eiter	000000 d W	mod reg r/m			
		IMMEDIATE to REG/MEM	100000 s W	mod 000 r/m			
		IMMEDIATE to Accumulator	0000010 W	DATA	DATA if W=1		

ADC	ADDITION WITH CARRY	REG/MEM and Register to eiter	000100 d W	mod 000 r/m		
		IMMEDIATE to REG/MEM	100000 s W	mod 010 r/m	DATA	DATA if W=1
		IMMEDIATE to Accumulator	0001010 W	DATA	DATA if W=1	
INC	INCREMENT	REG/MEM	1111111 W	mod 000 r/m		
		Register	01000 reg			
AAA	ASCII ADJUST FOR ADDITION		00110111			
DAA	DECIMAL ADJUST FOR ADDITION		00100111			
SUB	SUBTRACT	REG/MEM and Register to eiter	001010 d W	mod reg r/m		
		IMMEDIATE from REG/MEM	100000 s W	mod 101 r/m	DATA	DATA if W=1
		IMMEDIATE from Accumulator	0010110 W	DATA	DATA if W=1	
SBB	SUBTRACT WITH BORROW	REG/MEM and Register to eiter	000110 d W	mod reg r/m		
		IMMEDIATE from REG/MEM	100000 s W	mod 011 r/m	DATA	DATA if sW=01
		IMMEDIATE from Accumulator	0001110 W	DATA	DATA if W=1	
DEC	DECREMENT	REG/MEM	1111111 W	mod 001 r/m		
		Register	01001 reg			
NEG	CHANGE SIGN		1111011 W	mod 011 r/m		
CMP	COMPARE	REG/MEM and Register	001110 d W	mod reg r/m		
		IMMEDIATE with REG/MEM	100000 s W	mod 111 r/m	DATA	DATA if sW=01
		IMMEDIATE ith Accumulator	0011110 W	DATA	DATA if W=1	
AAS	ASCII ADJUST FOR SUBTRACT		00111111			
DAS	DECIMAL ADJUST FOR SUBTRACT		00101111			
MUL	MULTIPLY (UNSIGNED)		1111011 W	mod 100 r/m		
IMUL	INTEGER MULTIPLY (SIGNED)		1111011 W	mod 101 r/m		
AAM	ASCII ADJUST FOR MULTIPLY		11010100	00001010		
DIV	DIVIDE (UNSIGNED)		1111011 W	mod 110 r/m		
IDIV	INTEGER DIVIDE (SIGNED)		1111011 W	mod 110 r/m		
AAD	ASCII ADJUST FOR DIVIDE		11010101	00001010		
CBW	CONVERT BYTE TO WORD		10011000			
CWD	CONVERT WORD TO DOUBLE-WORD		10011001			
	ISTRUZIONI DI TIPO LOGICO					
NOT	INVERT		1111011 W	mod 010 r/m		
SHL/SAL	SHIFT LOGICAL/ARITHMETIC LEFT		110100 V W	mod 100 r/m		
SHR	SHIFT LOGICAL RIGHT		110100 V W	mod 101 r/m		
SAR	SHIFT ARITHMETIC RIGHT		110100 V W	mod 111 r/m		
ROL	ROTATE LEFT		110100 V W	mod 000 r/m		

ROR	ROTATE RIGHT		110100 V W	mod 001 r/m			
RCL	ROTATE THROUGH CARRY LEFT		110100 V W	mod 010 r/m			
RCR	ROTATE THROUGH CARRY RIGHT		110100 V W	mod 011 r/m			
AND	AND	REG/MEM and Register to eiter	001000 d W	mod reg r/m			
		IMMEDIATE to REG/MEM	1000000 W	mod 100 r/m	DATA	DATA if W=1	
		IMMEDIATE to Accumulator	0010010 W	DATA	DATA if W=1		
TEST	AND FUNCTION TO FLAG (NO RESULT)	REG/MEM and Register	1000010 W	mod reg r/m			
		IMMEDIATE data and REG/MEM	1111011 W	mod 000 r/m	DATA	DATA if W=1	
		IMMEDIATE data and Accumulator	1010100 W	DATA	DATA if W=1		
OR	OR	REG/MEM and Register to eiter	000010 d W	mod reg r/m			
		IMMEDIATE to REG/MEM	1000000 W	mod 001 r/m	DATA	DATA if W=1	
		IMMEDIATE to Accumulator	0000110 W	DATA	DATA if W=1		
XOR	EXCLUSIVE OR	REG/MEM and Register to eiter	001100 d W	mod reg r/m			
		IMMEDIATE to REG/MEM	1000000 W	mod 110 r/m	DATA	DATA if W=1	
		IMMEDIATE to Accumulator	0011010 W	DATA	DATA if W=1		
	MANIPOLAZIONE STRINGHE						
REP	REPEAT		1111001 z				
MOVS	MOVE STRING		1010010 W				
CMPS	COMPARE STRING		1010011 W				
SCAS	SCAN STRING		1010111 W				
LODS	LOAD STRING		1010110 W				
STOS	STORE STRING		1010101 W				
	SALTO E CONTROLLO FLUSSO						
CALL	CALL	Direct within Segment	11101000	DISP-Low	DISP- High		
		Indirect within Segment	11111111	mod 010 r/m			
		Direct Intersegment	10011010	OFFSET-Low	OFFSET-High	SEG-Low	SEG-High
		Indirect Intersegment	11111111	mod 010 r/m			
JMP	UNCONDITIONAL JUMP	Direct within Segment	11101001	DISP-Low	DISP-High		
		Direct within Segment-Short	11101011	DISPL.			
		Indirect within Segment	11111111	mod 100 r/m			
		Direct Intersegment	11101010	OFFSET-Low	OFFSET-High	SEG-Low	SEG-High
		Indirect Intersegment	11111111	mod 101 r/m			
RET	RETURN FROM CALL	Within Segment	11000011				
		Within Segment Adding IMMEDIATE to SP	11000010	DATA-Low	DATA-High		

		SP			
		Intersegment	11001011		
		Intersegment Adding IMMEDIATE to SP	11000010	DATA-Low	DATA-High
JE or JZ	JUMP ON EQUAL or ZERO		01110100	DISPL.	
JL or JNGE	JUMP ON LESS or NOT GREATER OR EQUAL		01111100	DISPL.	
JLE or JNG	JUMP ON LESS OR EQUAL or NOT GREATER		01111110	DISPL.	
JB or JNAE	JUMP ON BELOW* or NOT ABOVE* OR EQUAL		01110010	DISPL.	
JBE or JNA	JUMP ON BELOW* OR EQUAL or NOT ABOVE*		01110110	DISPL.	
JP or JPE	JUMP ON PARITY or PARITY EVEN		01111010	DISPL.	
JO	JUMP ON OVERFLOW		01110000	DISPL.	
JS	JUMP ON SIGN		01111000	DISPL.	
JNE or JNZ	JUMP ON NOT EQUAL or NOT ZERO		01110101	DISPL.	
JNL or JGE	JUMP ON NOT LESS or GREATER OR EQUAL		01111101	DISPL.	
JNLE or JG	JUMP ON NOT LESS OR EQUAL or GREATER		01111111	DISPL.	
JNB or JAE	JUMP ON NOT BELOW* or ABOVE* OR EQUAL		01110011	DISPL.	
JNB or JA	JUMP ON NOT BELOW* OR EQUAL or ABOVE*		01110111	DISPL.	
JNP or JPO	JUMP ON NOT PARTY or PARITY ODD		01111011	DISPL.	
JNO	JUMP ON NOT OVERFLOW		01110001	DISPL.	
JNS	JUMP ON NOT SIGN		01111001	DISPL.	
LOOP	LOOP CX TIME		11100010	DISPL.	
LOOPZ/LOOPE	LOOP WHILE ZERO/EQUAL		11100001	DISPL.	
LOOPNZ/LOOP	LOOP WHILE NOT ZERO/NOT		11100000	DISPL.	
NE	EQUAL				
JCXZ	JUMP ON CX ZERO		11100011	DISPL.	
	CONTROLLO PROCESSO				
INT	INTERRUPT	Typo Specified	11001101	TYPE	
		Type 3	11001100		
INTO	INTERRUPT ON OVERFLOW		11001110		
IRET	INTERRUPT RETURN		11001111		
CLC	CLEAR CARRY		11111000		
CMC	COMPLEMENTARY CARRY		11110101		
CLD	CLEAR DIRECTION		11111100		
CLI	CLEAR INTERRUPT		11111010		

* "Above" and "Below" si riferiscono alle relazioni fra due valkori senza segno mentre "Greater" and "Less" si riferiscono a relazioni fra due valori con segno.

HLT	HALT	11110100	
LOCK	BUS LOCK PREFIX	11110000	
STC	SET CARRY	11111001	
NOP	NOT OPERATION	10010000	
STD	SET DIRECTION	11111101	
STI	SET INTERRUPT	11111011	
WAIT	WAIT	10011011	
ESC	ESCAPE (TO EXTERNAL DEVICE)	11011xxx	mod xxx r/m

NOTE:
se d=1 allora “TO”; se d=0 allora “FROM”
se W=1 allora WORD instruction; se W=0 allora BYTE Instruction
se sW=01 allora 16 bit di dato immediato nella parte operando
se sW=01 allora il dato immediato costituente l’operando è esteso a 16 bit
se V=0 allora “COUNT”=1; se V=1 allora “COUNT” in (CL)
x = non usato
Z e usato nelle istruzioni stringa per confronto. modifica lo ZF FLAG
AL = 8-bit Accumulator
AX = 16-bit Accumulator
CX = Count Register
DS = Data Segment
DX = Variable Port register
ES = Extra Segment
ABOVE/BELOW si riferisce ad un unsigned value
GREATER = more Positive (strettamente maggiore)
LESS = Less Positive (more Negative) signed value (strettamente minore)

DOMANDE: MICROPROCESSORI A 16- 32 bit (8086) **E. Tombelli**

1 Spiegare dettagliatamente quali sono le principali innovazioni apportate ai microprocessori dell'ultima generazione (16 bit e successivi) rispetto a quelli della precedente a 8 bit.

MP

2 Spiegare il funzionamento della MULTIUTENZA, e che utilità e problematiche comporta.

MP

3 Spiegare quali sono le problematiche relative alla MULTIUTENZA e come vengono affrontate nei microprocessori dell'ultima generazione.

MP

4 Spiegare quali sono i principali sistemi per la gestione logica della memoria, facendo degli esempi opportuni.

MP

5 Spiegare la differenza fra i due tipi di gestione della memoria: PAGINATO e SEGMENTATO, facendo degli esempi riguardo al sistema di indirizzamento.

MP

6 Spiegare qual è la funzione del MMU e del PREFETCHING.

MP

7 Spiegare la funzione del PREFETCHING e del DEBUGGING.

MP

8 Spiegare la funzione della memoria CACHE e del MMU.

MP

9 Spiegare l'architettura interna del microprocessore 8086.

MP

10 Spiegare com'è organizzata logicamente la memoria in un sistema con microprocessore 8086.

MP

11 Dire quali sono i registri interni dell'8086 e la loro funzione

MP

12 Dire quali sono le aree di memoria riservate nel sistema 8086 e perché; Specificare la funzione dei flag nel registro di stato.

MP

13 Dire a che cosa serve la memoria CACHE e la funzione dei FLAG nel registro di stato dell'8086.

MP

14 Specificare qual è la logica che permette l'indirizzamento da parte dell'8086 (MODI DI INDIRIZZAMENTO).

MP

15 Definire come vengono allocate le informazioni nei registri interni dell'8086 allo scopo di indirizzare una cella di memoria.

MP

16 Descrivere la funzione del S.O.P. e come avviene l'associazione dei registri interni allo scopo di indirizzare una cella di memoria. Dire qual è il contesto se il S.O.P. non viene utilizzato nelle istruzioni
MP 8086.

17 Fare alcuni esempi di TIPO DI INDIRIZZAMENTO relativo all'8086 e spiegarne la funzione nel modo più completo, magari con un esempio.

MP

18 Scrivere un'istruzione 8086 che utilizzi l'indirizzamento INDIRETTO in modo che sia la più completa possibile.

MP

19 Spiegare quali sono le possibilità per effettuare un "SALTO" in un programma d'istruzioni 8086. Fare riferimento alla modalità d'individuazione dell'indirizzo al quale saltare.

MP

20 Proporre un esempio di salto RELATIVO all'indietro e spiegare come avviene il calcolo dell'offset.

MP

21 Spiegare quali sono le possibili istruzioni per il trasferimento dei dati nel sistema 8086. Fare alcuni esempi.

MP

22 Spiegare quali sono le possibili istruzioni che permettono il calcolo logico-aritmetico nell'8086. Fare alcuni esempi.

MP

---

23 Spiegare il funzionamento delle istruzioni per la manipolazione delle stringhe nel sistema 8086. Fare alcuni esempi.

MP

---

24 Spiegare il funzionamento delle istruzioni di salto e quali sono le possibilità nel sistema 8086. Fare alcuni esempi.

MP

---

25 Spiegare in che cosa consiste il linguaggio ASSEMBLER.

MP

---

26 Spiegare quali sono principali agevolazioni dovute all'utilizzo del linguaggio ASSEMBLER. Fare alcuni esempi.

MP

---

27 Specificare la funzione delle FRASI DICHIARATIVE nel linguaggio ASSEMBLER. Fare alcuni esempi.

MP

---

28 Dire il significato degli oggetti definiti dall'insieme delle PSEUDO-ISTRUZIONI e perché vengono chiamate così.

MP

---

29 Spiegare la funzione delle MACROISTRUZIONI, DIRETTIVE in genere e FRASI DICHIARATIVE.

MP

---

30 Spiegare in che cosa consiste la REVERSIBILITA' di un programma scritto in linguaggio ASSEMBLER. Spiegare la differenza fra un linguaggio a basso livello ed uno ad alto livello.

MP

---

31 Spiegare la funzione di un compilatore e la differenza fra linguaggio ad alto e basso livello. Dire quali sono i pregi e i difetti.

MP

---

32 Spiegare come avviene la "compilazione" di un programma scritto in ASSEMBLER, facendo magari un esempio.

MP

---

33 Spiegare la ragione dell'utilità delle ETICHETTE nel linguaggio ASSEMBLER

MP

---

34 Spiegare a cosa servono la tavola dei SIMBOLI e la tabella dei CODICI.

MP

---

35 Spiegare la funzione dei SOTTOPROGRAMMI e come possono essere realizzati.

MP

---

36 Dire la differenza fra le MACROISTRUZIONI e i SOTTOPROGRAMMI

MP

37 Con un semplice esempio, illustrare come la CPU provvede all'esecuzione in un sottoprogramma e spiegare perché e come viene utilizzato lo STACK.

MP

38 Spiegare il ruolo dello STACK nella gestione dei sottoprogrammi e quali sono le istruzioni per realizzare questi ultimi.

MP

39 Spiegare come si evolve il contenuto del Program Counter (CS:IP nell'8086) durante l'esecuzione di un sottoprogramma.

MP

40 Definire il ruolo delle istruzioni CALL e RET e la loro funzione specifica. Fare un esempio.

MP

41 Spiegare la differenza fra COMPILATORE, LINKER, LOADER. Dire in che cosa consiste la rilocabilità di un programma.

MP

42 Specificare la funzione del LINKER e del LOADER. Fare un esempio.

MP

43

MP

44

MP

45

MP

46

MP

47

MP

48

MP

49

MP