Appunti di Sistemi di Elaborazione
Generalità sul processore Intel 8086

Cenni storici sulla Intel

La Intel Corporation fu fondata nel 1968 allo scopo di costruire CHIP DI MEMORIA. Successivamente, su precise richieste di costruttori di calcolatori, essa costruì le prime 2 CPU al mondo che fossero basate su un unico chip: erano il **4004** (a 4 bit) ed l' **8008** (a 8 bit). Un passo decisivo fu poi compiuto con la costruzione dell' **8080**, una piccola CPU, di uso molto più generale delle precedenti, che riscontrò un enorme quanto inaspettato successo. Nel 1876 uscì l' **8085**, che aveva semplicemente maggiori capacità di Input/Output rispetto all'8080. Immediatamente dopo comparve l' **8086**, vera CPU a 16 bit costruita su di un unico chip. Dopo l'8086 comparve l' **8088**: questo aveva la stessa architettura del predecessore 8086 e poteva perciò eseguire gli stessi programmi, tuttavia aveva un bus dati da 8 bit (anziché da 16 bit) il che lo rendeva più lento ma anche più economico.

Il limite dell'8086 e dell'8088 era l'indirizzamento della memoria: non si poteva indirizzare più di 1 Megabyte di memoria. Nacque allora, agli inizi degli anni '80, l' **80286**, che era una versione avanzata e compatibile dell'8086. Anche se il repertorio di istruzioni era grossomodo lo stesso dei predecessori, l'organizzazione della memoria era più complessa.

Infine, nacque l' **80386**, che era una vera CPU a 32 bit sempre su un solo chip. Da notare che il passaggio dalla prima CPU a 4 bit alla CPU 80386 fu compiuto in soli 15 anni e la differenza tra i due chip è incredibile. In un certo senso, la storia della famiglia INTEL e della evoluzione delle CPU rispecchia un po’ quello che è stata l'evoluzione contemporanea dell'industria informatica.

Il concetto di "coda"

Il processore 8086 utilizza il concetto di **accodamento delle istruzioni** al fine di aumentare la velocità del computer; vediamo in cosa consiste:

- all'interno del chip del processore, c'è una piccola area (6 byte), detta appunto **coda di istruzioni**, la quale è destinata a contenere diversi byte di istruzioni;
- durante il ciclo di esecuzione di un programma, quando il computer è pronto per eseguire l'istruzione successiva, non deve necessariamente andare a prenderla tutta dalla memoria, in quanto è possibile che tutta o parte dell'istruzione sia già contenuta nella coda; in questo modo, risulta ridotto l'accesso del processore al bus del sistema, che può essere utilizzato da altri dispositivi.

Inoltre, il vantaggio di usare una coda è anche il seguente: se la coda contiene sia l'istruzione in corso di esecuzione sia la prossima da eseguire, è possibile leggere la nuova istruzione mentre la vecchia viene eseguita. In questo modo, per esempio, le istruzioni che presentano **operandi immediati** possono essere eseguite quasi alla stessa velocità di quelle che invece usano dati contenuti nei registri della CPU.

L'indirizzamento e la segmentazione

L'8086 può avere accesso ad **1 MB** di memoria, che corrisponde cioè ad indirizzi che vanno da 0 a 2^{20}-1: infatti, questo processore dispone di un **bus indirizzi di 20 bit**. Dato che i registri dell'8086 sono tutti a 16 bit, il processore utilizza un particolare schema di indirizzamento della memoria, che prende il nome di **segmentazione**.

Lo spazio di I/O

L'8086 ha un'area separata dalla memoria centrale, chiamata **spazio di I/O**; tale area di memoria può essere pensata come un ulteriore spazio di indirizzamento, nel quale solitamente vengono posti gli indirizzi dei dispositivi collegati al processore. D'altra parte, comunque, tali indirizzi potrebbero anche essere sistemati direttamente nella memoria principale.

Lo spazio di I/O utilizza degli indirizzi a 16 bit, il che significa che la dimensione massima di quest'area di memoria è di 64 Kbyte. Spesso, in quest'area, vengono sistemati i controllori dei dispositivi e la memoria associata al processore **8089** (se presente).

La struttura di Interrupt

Mentre quasi tutti i microprocessori a 8 bit richiedono l'intervento di chip addizionali per la gestione degli interrupt, al contrario l'8086 è dotato di una propria potente struttura di interrupt. Infatti, l'8086 dispone dei mezzi seguenti:

- intanto, ci sono circa 1000 byte riservati per contenere fino a 256 puntatori ai **vettori di interrupt**;
- inoltre, c'è uno spazio esterno alla memoria, detto "area di I/O", che può avere la dimensione massima di 64Kb, che è destinato alle operazioni di I/O.

L'8086 è dotato di **controllore di interrupt**, che si indica con l'abbreviazione **PIC** (che sta per "Controllore programmabile di Interrupt"), il quale si occupa del trattamento diretto degli interrupt per un numero di dispositivi che può arrivare fino ad 8. Questo controllore si comporta come un impiegato alla *reception* di un hotel, mentre ogni dispositivo si comporta come un cliente che voglia parlare con il capo, che ovviamente è la CPU. Il controllore ha il compito di lasciar passare un cliente per volta, tenendo conto delle opportune priorità. L'aggettivo "programmabile" indica la possibilità di indicare al controllore di ignorare o controllare determinate richieste di interrupt; la selezione viene determinata per mezzo della **maschera di interrupt**, che consiste in un byte che la CPU invia al PIC attraverso una particolare **porta**. Gli 8 bit della maschera corrispondono agli 8 dispositivi: per disabilitare l'interrupt proveniente da un dispositivo, basta porre ad 1 il corrispondente bit della maschera. Ad esempio, se inviassimo al

PIC una maschera pari a 11111111, esso ignorerà tutti e 8 i dispositivi, mentre invece, se la maschera è 00000000, allora risponderà a ciascuno di essi.

Quando due o più dispositivi (non mascherati) richiedono il servizio del PIC nello stesso istante, il PIC determina quale debba passare per primo in base ad opportune **priorità** (che l'utente stesso può definire) . I dispositivi che non vengono serviti subito, vengono messi "in attesa" e poi vengono serviti non appena è possibile.

Vediamo allora dal punto di vista pratico COME viene servita una richiesta di interrupt:

- quando arriva al PIC una richiesta di interruzione ed il PIC stabilisce, in base alle priorità prima citate, che può essere servita, esso invia alla CPU un segnale che avverte della richiesta di interrupt;
- appena può, la CPU salva lo stato dell'elaborazione in corso (in modo da poterla riprendere dallo stesso punto) e risponde inviando un altro segnale al PIC per indicare che ha riconosciuto la richiesta ed è disponibile;
- a questo punto il PIC invia alla CPU un byte per comunicarle il tipo di interrupt da eseguire: in pratica, non fa altro che indicare la locazione, nella tabella degli interrupt, dell'indirizzo della **routine di gestione dell'interrupt** in questione.

In definitiva, quindi, le richieste di interruzione arrivano direttamente al PIC; questo prima seleziona quelle che vanno servite e quelle che vanno ignorate e poi, in base alle rispettive priorità, "chiama" la CPU e le indica di servire le richieste nell'ordine che lui stesso ha stabilito. Si deduce quindi come venga alleggerito il lavoro della CPU, la quale deve "solo" occuparsi di eseguire, nei momenti e nell'ordine indicati dal PIC, le routine di gestione degli interrupt.

I chip coprocessori per l'8086

Sono disponibili, in aggiunta all'8086, due chip coprocessori: il primo è il **processore di dati numerici 8087**, il quale garantisce operazioni in virgola mobile estremamente veloci; il secondo è il **processore di I/O 8089**, espressamente progettato per un efficiente spostamento di blocchi di dati.

Entrambi questi coprocessori dispongono di un proprio set di istruzioni: quando viene eseguito un programma, essi controllano il flusso del programma e, quando incontrano una propria istruzione, la eseguono senza alcun intervento da parte dell'8086. Si dice che essi lavorano in **co-processing** con il processore centrale, cioè l'8086.

L'8086 svolge le funzioni di direzione generale dei lavori dell'intero sistema, in grado di delegare alcuni compiti agli altri 2 processori.

Struttura generale dell'8086

L'8086 è un **processore a 16 bit** ; esso si divide in 2 **sottoprocessori**: c'è infatti una **unità esecutiva** (EU) ed una **unità di interfaccia con il bus** (BIU). L'unità esecutiva è quella che si occupa di decodificare ed eseguire le istruzioni, cioè tutti i calcoli, mentre l'unità BIU deve accedere ai dati ed alle istruzioni che provengono dal mondo esterno, interagendo con il bus e con i registri interni. Ognuno dei due sottoprocessori forma una unità di lavoro a sé stante, indipendente dall'altra ed in grado di svolgere solo determinati compiti. Quando la BIU preleva un byte di *codice oggetto* dalla memoria lo pone nella coda; in particolare, nell'8086, trattandosi di processore a 16 bit, il prelevamento avviene 2 byte alla volta.

L'importanza dei 16 bit

Nonostante i sistemi a 8 bit siano evidentemente più semplici e meno costosi da realizzare, i sistemi a 16 bit come l'8086 presentano senza dubbio migliori prestazioni: questo per via del fatto che un bus dati di 16 bit può trasferire dati fino a 2 volte più in fretta tra componenti a 16 bit, compresi la CPU, la memoria ed il sistema di visualizzazione grafica; in particolare, risultano molto più veloci le operazioni di "fetch" delle istruzioni e di visualizzazione delle figure.

Ad ogni modo, *il fatto, per esempio, che il bus dati dell'8086 sia il doppio di quello dell'8088 non implica affatto che l'8086 funziona ad una velocità doppia dell'8088*. I motivi che impediscono questo sono molteplici: intanto, bisogna tener conto che molte applicazioni, pur girando su processori a 16 bit, richiedono che i dati siano comunque passati 8 bit per volta; un altro motivo è che, comunque, il processore deve svolgere molteplici altre attività oltre quella di muovere dati tra un dispositivo e l'altro. Tuttavia, il motivo vero è legato ad alcune caratteristiche dell'8086 che abbiamo introdotto prima: queste sono la divisione del processore in 2 sottoprocessori (UE e BIU) e l'uso della coda di istruzioni.

Le operazioni aritmetiche

Sappiamo che l'8086 possiede, nel suo set di istruzioni, alcune istruzioni che eseguono operazioni matematiche sia su numeri con segno sia su numeri senza segno. Tuttavia, a causa del modo di funzionamento dell'aritmetica in **complemento a 2** (usata per la rappresentazione dei numeri negativi), le operazioni aritmetiche che realmente vengono effettuate sono di fatto le stesse per i numeri con segno e per quelli senza. L'unica differenza sta nell'intervallo oltre il quale si verifica l' **overflow** (il quale overflow viene segnalato da un opportuno flag).

Nell'aritmetica dei **numeri senza segno**, l'overflow si verifica quando c'è un riporto dalla cifra binaria più a sinistra: in questo caso, il **flag del riporto CF** viene impostato ad 1 dalla CPU.

Per quanto riguarda invece i **numeri con segno**, la situazione risulta più complicata. In questo caso il flag che viene utilizzato non è più CF bensì **OF**. L'overflow in una operazione con numeri segnati si verifica quando il segno del risultato risulta sbagliato (a causa proprio di un riporto di troppo): per esempio, nella somma di 2 numeri positivi, può accadere che il risultato risulti negativo proprio perchè è troppo grande per poter essere contenuto nello spazio che gli è stato riservato (8 o 16 bit). Allora, la CPU è dotata di circuiti che sono in grado di riconoscere questa situazione e di impostare ad 1 il flag OF. Da notare che, nel caso sempre della somma algebrica, non ci sarà mai overflow se i due numeri da sommare hanno segno diverso, in quanto il risultato sarà comunque più piccolo del maggiore tra i due addendi e quindi potrà essere contenuto senza problema nel suo spazio.

L'input/output nel processore Intel 8086

Per poter arrivare a descrivere come il microprocessore 8086 gestisce l'input/output, è necessario fare una breve quanto essenziale premessa. Quando viene progettato un calcolatore e quindi tutte le componenti che lo costituiscono, uno degli obbiettivi di fondo da ottenere è di rendere il sistema il più veloce possibile: bisogna cioè metterlo in grado di eseguire i programmi nel modo più veloce possibile. Ora, ci sono vari fattori che influiscono sulla velocità complessiva di elaborazione di un sistema: tralasciando le prestazioni del microprocessore e dei dispositivi ad esso collegati, ci sono 2 fattori fondamentali che influiscono sulla velocità di elaborazione: il primo fattore sono i cosiddetti **tempi morti** del microprocessore, ossia quegli intervalli di tempo durante i quali il microprocessore non svolge alcun compito in quanto deve aspettare che qualche altro dispositivo termini una certa operazione; il secondo fattore è l'**occupazione del bus**, ossia di quel dispositivo che, collegando tra loro tutti i dispositivi che costituiscono il calcolatore, permette l'interscambio dei dati sia tra i dispositivi sia tra il calcolatore ed il mondo esterno. Per poter raggiungere alte velocità di elaborazione, il sistema deve essere concepito in modo sia da ridurre quanto più è possibile i tempi di attesa del microprocessore sia da ridurre l'accesso di ogni dispositivo al bus al minimo indispensabile. I motivi sono i seguenti:

- in primo luogo, la possibilità di dare al microprocessore del lavoro continuo (lavoro utile, naturalmente) da svolgere consente evidentemente di velocizzare notevolmente i tempi di esecuzione;
- in secondo luogo, quanto minore è il tempo durante il quale ogni dispositivo accede al bus , tanto maggiore sarà la possibilità di ogni dispositivo di accedere al bus stesso ogni volta che ne ha bisogno.

Allora, entrambi questi obbiettivi possono essere perseguiti rendendo le operazioni svolte dal sistema, in particolare quelle di I/O, quanto più indipendenti possibile dal controllo del microprocessore; si tratta cioè di affidare ad altri dispositivi la gestione "parziale" dell'I/O, in modo che la CPU, dopo le necessarie operazioni di controllo, possa dedicarsi ad altri

compiti contemporaneamente allo svolgimento delle operazioni di I/O. Inoltre, se la CPU interviene solo in maniera minima sulla gestione dell'I/O, è ovvio che accede per un tempo estremamente piccolo al bus, rendendolo perciò più disponibile per gli altri dispositivi: difatti, non dimentichiamoci che le operazioni di I/O consistono in flussi di dati che fluiscono o dal sistema verso l'esterno o viceversa, passando ogni volta per il bus.

Nell'8086, come del resto anche in molti altri chip, specialmente quelli moderni, per rendere la gestione dell'I/O il più indipendente possibile dalla CPU si usano i cosiddetti **controllori di dispositivi di I/O**: in pratica, ogni dispositivo di I/O è dotato di un proprio "controllore" ed è questo controllore che collega il dispositivo al BUS; si tratta di un normale chip che si occupa appunto della gestione del dispositivo in questione e del suo accesso al bus. In particolare, molti controllori vengono messi in grado di accedere alla memoria, dove si trovano i dati per l'I/O, senza alcun intervento vigile della CPU: si dice in questo caso che il controllore è dotato di **Accesso Diretto alla Memoria**; in pratica, l'operazione di I/O avviene tramite l'interazione del controllore, del corrispondente dispositivo di I/O e di un alto particolare chip, detto appunto **chip di DMA** che si occupa di gestire l'accesso del controllore alla memoria.

In poche parole, il chip DMA prende il comando del BUS di sistema per trasferire direttamente informazioni da una parte all'altra del sistema. In pratica, quando c'è da svolgere una operazione di I/O, la CPU fornisce al chip DMA tutte le informazioni che consentono di identificare l'operazione da svolgere (Input o Output), quale dispositivo è interessato e i dati che verranno utilizzati : nel caso di output, viene indicato l'indirizzo di memoria del primo byte da trasferire e il numero totale di byte da trasferire; nel caso di input, viene indicato l'indirizzo di memoria del primo byte da riempire ed il numero totale di byte da ricevere. Il tutto viene fatto aggiornando alcuni registri interni al chip DMA. Fatto questo, la CPU si disinteressa dell'operazione di I/O, che invece viene avviata e gestita dal chip DMA in collaborazione con il controllore. Appena terminata l'operazione, viene inviato un interrupt alla CPU per avvisarla.

Questo accade dunque quando il controllore del dispositivo utilizzato ha un accesso diretto in memoria; nel caso invece non ce l'abbia, allora la CPU interagisce direttamente con il controllore, senza la "mediazione" del chip DMA: in questo caso, la CPU viene parzialmente assorbita nel controllo dell'intera operazione, con conseguente rallentamento delle operazioni.

Ad ogni modo è bene dire che anche la tecnica di accesso diretto alla memoria non risolve completamente il problema della velocizzazione dell'elaborazione: difatti, la ridotta velocità dei dispositivi di I/O condiziona comunque il processo di I/O, nel senso che sono richiesti diversi accessi al bus prima che l'operazione venga terminata. Dato, allora, che questi dispositivi hanno maggiore priorità, per l'accesso al bus, rispetto alla CPU, può capitare che anche la CPU richieda l'accesso al bus e sia costretta ad aspettare (i già citati tempi morti) prima di poterlo ottenere. Questo processo è noto come **furto dei cicli**, nel senso che i dispositivi di I/O, utilizzando il bus, impediscono alla CPU di fare altrettanto.

A proposito dell' 8086, possiamo anche aggiungere che spesso esso viene affiancato da un altro processore, l'8089, che si occupa per intero della gestione dell'I/O: si tratta di un processore che, mentre è in corso

l'esecuzione di un certo programma, si prende carico della esecuzione delle istruzioni di I/O, che vengono invece ignorate dall'8086.

Vediamo altre ulteriori considerazioni circa il problema dell'I/O al fine di introdurre brevemente il funzionamento del coprocessore 8089.

Quando parliamo di **controllo dell'I/O** ci riferiamo al trattamento dei vari dispositivi di I/O, allo scopo di trasformare i segnali elettrici del computer nei normali segnali che l'operatore è in grado percepire e comprendere. Di solito, quando c'è la trasmissione di un certo flusso di dati, all'interno di tali dati ci sono anche delle **parole di controllo**, ossia di una specie di istruzioni che indicano, a chi di dovere, di svolgere una qualche operazione: ad esempio, quando c'è da visualizzare un testo sul video, pensiamo a quei caratteri che indicano di andare a capo o di lasciare uno spazio; questi caratteri non vengono visualizzati ma provocano appunto particolari azioni. La quantità di tempo che è necessario all'esecuzione di tali istruzioni che, tutte insieme, compongono un vero e proprio programma, varia ovviamente da dispositivo a dispositivo: nell'esempio prima citato se il testo va stampato su video, verrà impiegato un certo tempo, mentre invece ne verrà impiegato di più se la stampa avviene tramite una stampante.

Altri problemi legati all'I/O, oltre quelli finora descritti, sono senz'altro le "conversioni dei dati" e gli "schemi di correzione degli errori": per **conversioni dei dati** intendiamo la trasformazione di un certo flusso di dati da un certo codice (ad esempio l'ASCII) ad un altro (ad esempio l'EBCDIC) con regole del tutto diverse; per **schemi di correzione degli errori** intendiamo invece tutti quegli accorgimenti, molto complessi, che permettono di correggere eventuali errori di trasmissione dovuti ad eventi hardware indesiderati quanto inevitabili (pensiamo a surriscaldamenti, urti,...).

La memoria ROM nel processore Intel 8086

La **memoria ROM**, nell'8086, è utilizzata per memorizzare un breve programma, detto di **bootstrap**, cioè di inizializzazione, che permette di avviare il controllore del floppy disk: il programma fa’ si che il primo settore del disco sia caricato in memoria; tale settore contiene a sua volta un programma che carica il resto del sistema operativo; il sistema operativo, in seguito, inizializza i vari controllori e, quando è pronto per rispondere ai comandi, dà un segnale di attesa al sistema.

Il generatore di clock

Questo dispositivo serve a generare il cosiddetto **segnale di clock** di un processore; tale segnale serve a stabilire la velocità alla quale il sistema lavora; nel caso dell'8086 la velocità massima è di 5 MHz, che corrisponde a circa 200 nanosecondi per ciclo (= $1 / 5*10^6$).

Uno dei segnali che collegano il generatore di clock con il processore è il cosiddetto segnale di "READY" (pronto): la funzione di questo segnale è quella di sincronizzare il processore con dispositivi esterni più lenti. Quando

il processore richiede l'accesso ad un dispositivo che non è pronto (cioè NOT READY), il dispositivo invia uno 0 sulla linea READY; quando il processore riceve questo segnale, resta in attesa fino a che non arriva un 1 sulla linea READY; solo allora esso prosegue con il programma.

La logica per l'interfaccia dei bus

Il collegamento tra il processore ed il bus di sistema non è "diretto", bensì **interfacciato**, ossia avviene mediante dispositivi intermedi; si parla quindi di **interfacciamento del bus** e si rende necessario per due motivi: in primo luogo, i segnali che provengono dal processore possono non essere abbastanza potenti per pilotare il resto del sistema, per cui vanno amplificati opportunamente; in secondo luogo, i segnali prodotti dal processore potrebbero non corrispondere direttamente a quelli richiesti dal resto del sistema e vanno perciò opportunamente modificati.

Tra i dispositivi di interfacciamento troviamo i seguenti:

- il controllore di bus;
- il trasmettitore/ricevitore di dati ottali;
- il "latch ottale".

Oltre a questi, nei sistemi dotati di più processori, si trova anche l' **Arbitro del Bus** che si occupa di ripartire le risorse per l'accesso al bus principale.

Sappiamo che il bus principale consta di 4 **sottobus**: alimentazione, controllo, indirizzi e dati. Tralasciando il primo, vediamo come gli altri tre vengono interfacciati con il processore tramite i dispositivi prima citati.

Il **controllore del bus** è un chip a 20 pin usato per creare l'interfaccia tra il processore ed il **bus di controllo**; esso produce dei segnali di controllo molto importanti, tra i quali citiamo il controllo della lettura e/o della scrittura della memoria, il controllo della scrittura di I/O, l'abilitazione alla memorizzazione dei dati e quella per la memorizzazione degli indirizzi. Alcuni di questi segnali (ad esempio quelli di controllo per lettura e scrittura) sono destinati al bus di sistema, mentre altri, come i controlli di abilitazione, sono destinati ad altri chip che realizzano anch'essi l'interfaccia tra processore e bus.

Il **trasmettitore/ricevitore** è un altro chip a 20 pin che serve per interfacciare il processore al **bus dati**: esso memorizza in una particolare area di memoria, detta **buffer**, i dati che vanno e vengono dal processore.

Il **latch ottale** è un chip a 20 pin utilizzato per l'interfaccia tra le linee dati/indirizzi del processore ed il **bus indirizzi** del sistema. Esso resta in attesa del momento in cui le informazioni di indirizzo compaiono sui pin; in quell'istante, esso prende tali informazioni e le conserva nel bus indirizzi del sistema.

Coprocessing e multiprocessing

Un modo per ottenere migliori prestazioni da un sistema dotato di più processori è quello di definire dei processori periferici, detti **slave** (ossia dipendenti), sotto il controllo di un processore centrale, detto **master**. Usando i chip 8086, 8089 (per l'I/O) e 8087 (per i calcoli), è possibile utilizzare 2 metodi: il "coprocessing" ed il "multiprocessing".

In un sistema a **coprocessing**, due o più processori condividono lo stesso flusso di istruzioni, ossia entrambi seguono lo stesso programma; tuttavia, essi eseguono "a turno" le istruzioni, nel senso che alcune istruzioni sono più adatte all'uno dei due, altre all'altro. Un esempio di coprocessing si ha tra l'8086 e l'8087.

Al contrario, in un sistema a **multiprocessing**, due o più processori condividono un'area comune di memoria, ma operano su diversi flussi di istruzioni: uno dei due potrebbe essere il master e dirigere l'altro mediante messaggi posti in memoria; una volta partita l'esecuzione, ciascun processore esegue il proprio programma. Un esempio di multiprocessing si ha tra l'8086 e l'8089.

Il grande vantaggio di entrambi i metodi è che il processore centrale è sollevato da gran parte del lavoro e può dedicarsi completamente a far funzionare il sistema come un tutt'uno: esso infatti invia i diversi lavori ai diversi processori che sono specializzati e quindi più adatti. Naturalmente, per far funzionare questi meccanismi sono necessarie speciali istruzioni che abilitino un processore a cooperare con gli altri. Per esempio, nel coprocessing, le istruzioni devono essere necessariamente divise in gruppi, ciascuno per un processore: se un processore riceve una istruzione non destinata a lui, non deve cercare di eseguirla, ma deve lasciare l'esecuzione al processore cui l'istruzione era destinata.

Il Controllore programmabile di CRT 8275

Questo dispositivo è un chip a 20 pin che si occupa della **gestione dello schermo**: in particolare, esso si occupa di fornire la temporizzazione ed il controllo per la visualizzazione di un testo. In primo luogo, il testo da visualizzare deve essere memorizzato in una specifica zona della memoria principale, detta **Ram di visualizzazione dei testi**. Questa memorizzazione avviene nel modo seguente: la CPU sistema in queste locazioni di memoria i codici ASCII corrispondenti ad ogni singolo carattere da visualizzare. Sulla base di questi codici, il controllore produce i segnali video necessari per visualizzare sullo schermo i simboli corrispondenti. Per esempio, se il primo byte della RAM del video contenesse il valore esadecimale 41, che è il codice ASCII del carattere "A", il controllore farebbe apparire nell'angolo in alto a sinistra dello schermo una A maiuscola. Il resto della memoria viene rappresentato sullo schermo in righe successive, da sinistra a destra e dall'alto verso il basso.

Sulla base di ciò, è utile fare delle considerazioni su come è possibile far stampare dei caratteri sullo schermo: una volta noto l'indirizzo a partire dal quale comincia la memoria RAM per il video, basta sistemare nelle locazioni che partono da questo indirizzo i codici ASCII corrispondenti ai caratteri da

stampare; ci pensa poi il controllore ad effettuare la visualizzazione. Da notare anche che può essere l'utente stesso, tramite opportune istruzioni, a fissare quale deve essere la zona di memoria destinata alla RAM per il video.

Metodi di indirizzamento nell'8086

Per spiegare i vari metodi di indirizzamento in questo processore ci serviremo di una delle più comuni **istruzioni Assembler** tra quelle che utilizzano 2 diversi operandi: si tratta dell'istruzione **MOV**, la quale trasferisce un byte oppure una parola dall' *operando sorgente* all' *operando destinazione*. Il formato dell'istruzione è

MOV operando1,operando2

Dobbiamo adesso vedere quali dati è possibile specificare al posto di "operando1" e "operando2".

Indirizzamento immediato

L'**indirizzamento immediato** è il più semplice: mentre, di solito, gli operandi servono ad indicare la posizione (memoria o registro) del dato da utilizzare, nell'indirizzamento immediato il dato da utilizzare viene "immediatamente" indicato nell'istruzione stessa. Facendo riferimento alla nostra istruzione avremo ad esempio

MOV AX,564

Questa istruzione indica di inserire nel **registro AX** la costante 564. In tal modo, quindi, il numero 564 non va ricercato in qualche particolare posizione di memoria ma tra i byte stessi che contengono l'istruzione.

Indirizzamento a registro

Un altro modo abbastanza facile di indirizzamento è l' **indirizzamento a registro**: in questo caso il dato da utilizzare si trova in un registro, del quale va ovviamente specificato il nome. Un esempio di questo tipo di indirizzamento è

MOV AX,BX

per cui l'istruzione indica di *copiare* nel registro AX il contenuto del **registro BX** (si parla di *copiare* in quanto il contenuto di BX non viene in alcun modo modificato). Chiaramente, i registri in questione sono a 16 bit. E' possibile anche indicare registri ad 8 bit, quali sono, per esempio, **AL** e **BL**: in questo caso si avrà

MOV AL,BL

Da notare che le operazioni di trasferimento dati tra registri sono molto più veloci di quelle che coinvolgono la memoria: infatti, se bisogna accedere alla memoria, è necessario sia generare un opportuno indirizzo (con il metodo visto prima) sia utilizzare il bus, il tutto con perdite di tempo (relativamente) influenti.

Indirizzamento diretto

Il metodo di **indirizzamento diretto** è il più classico: in questo caso l'operando contiene l'indirizzo di memoria del dato cui corrisponde. Da notare, però, che l'indirizzo specificato non è quello fisico, ma solo il valore di **offset**, per cui sarà necessaria una generazione di indirizzo per determinare l'indirizzo assoluto. Un esempio di indirizzamento diretto è il seguente: l'istruzione

MOV AX,Alfa

dice di inserire nel registro AX il dato contenuto nell'indirizzo di memoria il cui offset è Alfa. A meno di ulteriori specificazioni nella istruzione, l'indirizzo fisico viene calcolato con la formula

(DS*16)+Alfa

usando cioè il **registro DS** di default. Infatti, il registro di segmento DS è quello utilizzato per individuare il segmento di memoria contenente specificamente i dati. Questo non vuol dire, però, che solo il registro DS possa essere usato per la generazione di questi indirizzi. E' infatti possibile specificare nella istruzione quale registro di segmento si intende utilizzare: ad esempio, l'istruzione

MOV AX,ES:Alfa

dice di usare il registro ES.

Indirizzamento indiretto tramite registro

I metodi di **indirizzamento indiretto**, che possono essere di vario tipo, consistono nello specificare la posizione di una locazione (memoria o registro) nella quale è indicato l'indirizzo di memoria in cui si trova il dato da utilizzare.

Un primo caso è quello dell'**indirizzamento indiretto tramite registro**: in questo caso, l'offset con cui calcolare l'indirizzo fisico di memoria si trova o in un **registro base** (**SI**,**DI**) o nel **registro indice BP** o nel **registro generale BX**. Il registro di segmento utilizzato è uno qualsiasi tra i 4. Un esempio è il seguente:

MOV AX,[SI]

Questa istruzione dice di spostare nel registro AX il dato contenuto in memoria all'indirizzo il cui offset è contenuto in SI. Allora, per la generazione dell'indirizzo fisico il processore usa la formula

(DS*16)+SI

Se invece scrivessimo

MV AX,ES:[SI]

allora la formula sarebbe la stessa con la sola eccezione di usare il registro ES al posto di DS.

Indirizzamento indiretto tramite registro con spostamento

Questo tipo di indirizzamento unisce gli ultimi due modi precedenti: infatti, per ottenere l'offset è necessario sommare il valore contenuto in un **registro interno** (specificato nell'istruzione) al valore di una **costante** (specificata anch'essa nell'istruzione). Ad esempio, se scrivessimo l'istruzione

MOV AX,Beta [DI]

vorrebbe dire che l'offset è Beta+DI,per cui l'indirizzo fisico di memoria va calcolato tramite la formula

(DS*16) + Beta + DI

La costante Beta è appunto lo "spostamento". Se volessimo usare un registro di segmento diverso da DS allora ci comporteremmo come nei casi precedenti: volendo usare ES dovremmo scrivere

MOV AX,ES:Beta [DI]

Indirizzamento diretto tramite registro con base e registro indice

In questo caso, l'offset si ottiene somma i contenuti di 2 registri interni del processore. Ad esempio l'istruzione

MOV AX,[BX] [SI]

dice che l'offset è la somma dei contenuti di BX ed SI. In tal modo, l'indirizzo fisico sarà dato da

(DS*16)+BX+SI

Anche qui è possibile cambiare il registro di segmento come nei casi visti in precedenza.

Indirizzamento indiretto tramite registro con base+indice+costante

Questo è l'ultimo possibile metodo di indirizzamento. Siamo sempre nell'ambito dell'indirizzamento indiretto; in particolare, la situazione è analoga alla precedente, con la differenza che l'offset si ottiene sommando ai contenuti di due registri interni anche il valore di una costante specificata nell'istruzione. Un facile esempio di questo tipo di indirizzamento è il seguente:

MOV AX,Alfa [BX] [DI]

In questo caso l'offset è la somma dei contenuti di BX e DI con la costante Alfa. L'indirizzo fisico è perciò

(DS*16)+Alfa+BX+DI

E' sempre possibile modificare il registro di segmento.

Registri e Flag nel processore 8086

Il processore Intel 8086 possiede 14 registri, definiti **registri interni**, tutti a 16 bit. Talvolta si dice che alcuni di questi registri hanno una propria *personalità*, nel senso che sono utilizzati per un unico scopo, mentre altri hanno qualche tratto caratteristico in comune, nel senso che possono essere utilizzati per gli stessi scopi. Vediamo allora nel dettaglio i nomi e le caratteristiche d'uso di questi registri.

Intanto, possiamo suddividerli in 4 gruppi:

- **4 registri generali** (AX,BX,CX,DX), usati prevalentemente per calcoli matematici ma anche per altri scopi;
- **4 registri di segmento** (CS,DS,ES,SS), usati esclusivamente per indirizzare la memoria, ossia per contenere indirizzi;
- **4 registri indice** (SP,BP,SI,DI), usati per contenere esclusivamente indirizzi di memoria;
- **1 registro dei flag** (FLAG), il quale contiene una serie di bit di stato del processore;
- **1 registro puntatore alle istruzioni** (IP), usato per contenere appunto solo indirizzi di memoria che si riferiscono solo ad istruzioni.

Prima di passare ad esaminarli uno per uno, premettiamo che i 4 registri generali AX, BX, CX e DX hanno la caratteristica di poter essere utilizzati sia come registri da 16 bit sia come coppie di registri da 8 bit; in questo caso, AX sarà diviso in AL e AH, BX in BL e BH, CX in CL e CH, DX in DL e DH.

Registro AX

Il registro AX è il registro accumulatore per eccellenza, nel senso che è quello utilizzato per i calcoli matematici con numeri da 16 bit (perciò solo interi). Per esempio, dovendo sommare 2 numeri, il primo viene caricato in AX, il secondo viene sommato ad AX ed il risultato viene inserito in AX. Inoltre, questo stesso registro viene usato per le operazioni con numeri a 32 bit (sia interi che reali): esso viene usato per contenere la parte più significativa di un numero a 32 bit (la rimanente parte viene inserita in DX).

Registro BX

Il registro BX è un registro detto "non specializzato", nel senso che, al contrario di AX, può essere usato per compiti diversi. Talvolta lo si usa come registro base per l'indirizzamento della memoria; talvolta viene utilizzato per contenere semplicemente numeri a 16 bit; mai viene utilizzato per contenere parti di numeri a 32 bit.

Registro CX

L'uso prevalente di questo registro è come contatore di un qualche ciclo di istruzioni. Talvolta lo si usa per contenere numeri a 16 bit, ma mai lo si usa per l'indirizzamento.

Registro DX

L'uso quasi esclusivo di questo registro è quello di memorizzare numeri: può trattarsi di numeri a 16 bit oppure della parte meno significativa di un numero a 32 bit (la parte più significativa viene inserita in AX). Non può invece essere usato per l'indirizzamento.

Registri SI e DI

I registri SI e DI sono esclusivamente utilizzati per l'indirizzamento indiretto della memoria. Non li si usa mai per contenere numeri. Questo è il motivo per cui si parla di **registri dedicati**.

Spesso l'uso di questi registri è connesso ad istruzioni che operano su stringhe: ad esempio, una tipica operazione su stringhe richiede un indirizzo sorgente, che si troverà in SI, e uno di destinazione, che si troverà in DI.

Registri BP e SP

Questi registri sono utilizzati esclusivamente per la gestione della memoria in modo **LIFO**, ossia per la gestione di uno stack di memoria. Il registro BP punta al fondo dello stack, ossia alla cella di memoria a partire dalla quale è stato realizzata lo stack; viceversa, il registro SP punta all'elemento affiorante dello stack. Ad esempio, se in un programma da

eseguire è presente una procedura che fa uso di una variabile locale, l'offset di tale variabile sarà contenuto in BP.

Registri CS,DS,ES,SS

Questi4 registri di segmento servono ad individuare porzioni di memoria, ognuna da 64K, ossia ognuna composta da 65536 celle. Il registro CS contiene l'indirizzo (segmento) a partire dal quale comincia l'**area delle istruzioni**; il registro DS contiene l'indirizzo di partenza dell'**area dati**; il registro ES contiene l'indirizzo di partenza dell'**area EXTRA**, che può essere usata a proprio piacimento; il registro SS contiene l'indirizzo di partenza dell'**area Stack**.

Dal punto di vista pratico l'uso di questi registri è il seguente:

- quando abbiamo l'offset di una istruzione, l'indirizzo fisico si otterrà sommando tale offset al contenuto di CS moltiplicato per 16;
- se l'offset si riferisce invece ad un dato, allora l'indirizzo fisico del dato si otterrà sommando l'offset al contenuto di DS moltiplicato per 16;
- l'indirizzo fisico della cima o del fondo dello stack si otterrà sommando il contenuto di SP o di BP al contenuto di SS moltiplicato per 16.

Per il registro ES dipende invece da che uso si è scelto di farne.

Registro IP

Il registro IP prende il nome di **Program counter** o anche "Contatore delle istruzioni" o anche "Contatore di programma". Esso, quando si sta eseguendo un programma, contiene sempre l'indirizzo (ovviamente l'offset) dell'istruzione da eseguire successivamente. Non viene utilizzato per nessun altro scopo.

Registro FLAG

Questo registro, talvolta indicato anche con il nome di **Registro Stato del Programma**, contiene una serie di bit che forniscono al processore una serie di informazioni utili sulle operazioni che sta compiendo o che ha compiuto in precedenza. Non si tratta perciò di un *registro normale*, nel senso che non serve per fornire un unico valore numerico. Vengono invece considerati separatamente i valori dei singoli bit. Vediamo allora i nomi associati a tali bit ed il loro scopo:

- bit 1 - **CF** - bit del riporto
- bit 2 - non utilizzato
- bit 3 - **PF** - bit di parità
- bit 4 - non utilizzato
- bit 5 - **AF** - bit di riporto ausiliario
- bit 6 - non utilizzato

- bit 7 - **ZF** - bit zero
- bit 8 - **SF** - bit del segno
- bit 9 - **TF** - bit di trap
- bit 10 - **IF** - bit di interrupt
- bit 11 - **DF** - bit di direzione
- bit 12 - **OF** - bit di overflow
- bit 13 - non utilizzato
- bit 14 - non utilizzato
- bit 15 - non utilizzato
- bit 16 - non utilizzato

Il bit CF rappresenta il riporto alla fine degli operandi.
Il bit PF dà la parità (pari o dispari) del risultato di una istruzione aritmetica.
Il bit AF rappresenta il riporto della metà degli operandi.
Il bit ZF vale 1 se il risultato di una operazione aritmetica è 0 e vale 0 in caso contrario.
Il bit SF vale 1 se il risultato di una operazione aritmetica è negativo e vale zero se è positivo.
Il bit TF autorizza i trap utilizzati per la messa a punto di programmi (Debug):
Il bit IF autorizza gli interrupt.
Il bit DF determina la direzione delle operazioni su stringhe.
Il bit OF vale 1 se il risultato di una operazione matematiche ha portato ad un overflow mentre vale 0 se tale eventualità non si è verificata.

La rappresentazione dei numeri nell'8086

Nell'8086 è possibile memorizzare i **numeri interi** in due modi differenti:

- il primo e più consueto modo è quello di memorizzare i numeri mediante il loro **valore binario**: in altre parole, se si vuole memorizzare il numero decimale N, nella memoria sarà contenuto il suo valore binario, ottenuto mediante le normali regole di conversione tra il sistema decimale e quello binario;
- un altro modo (utilizzato per lo più per calcoli di tipo finanziario) è quello di utilizzare la cosiddetta "codifica binaria" di numeri decimali, meglio conosciuto come **metodo BCD**.

Vediamo allora di parlare proprio di questo metodo. In linea generale, questo metodo si base sul fatto di rappresentare in binario, nel modo che vedremo, non più il valore complessivo del numero, bensì le sue singole cifre decimali. Esistono due tipi di codifica BCD: quello "non impaccato" e quello "impaccato"

BCD non impaccato

Se si vuole memorizzare un numero decimale mediante il **metodo BCD non impaccato**, si usa questo metodo: ogni cifra decimale del numero (cioè da 0 a 9 oppure da 0 ad F se siamo in esadecimale) viene memorizzata in 1 byte (mediante ovviamente il suo valore binario). Questo significa che, per memorizzare un numero decimale di N cifre, serviranno N byte, uno per ogni cifra.

Ad esempio, se il numero fosse 159, la sua codifica sarebbe

```
00000001 00000101 00001001
1        5        9
```

Ovviamente, è un metodo molto dispendioso di byte, in quanto è ovvio che ogni byte potrebbe contenere 256 numeri diversi anziché solo 10, cioè le cifre decimali da 0 a 9.

BCD impaccato

Certamente più economico (in termini di byte occupati) è il **metodo BCD impaccato**, che è identico al precedente, tranne per il fatto che, ad ogni cifra decimale viene assegnato un semibyte (detto **nibble**) anziché un byte intero. Con questo metodo, il numero 159 sarebbe rappresentato come

```
0000 0001 0101 1001
 0    1    5    9
```

Ovviamente, è necessario aggiungere una cifra 0 non significativa per completare il byte di sinistra.

Operazioni aritmetiche per la codifica BCD

La possibilità di usare questo tipo di rappresentazione dei numeri è garantita dalla presenza di particolari istruzioni che "aggiustano" i vari bit in modo da poter effettuare le normali operazioni matematiche anche tra questo tipo di numeri.

Operazioni di aggiustamento ASCII

Vediamo allora come è possibile effettuare delle operazioni tra numeri codificati con il metodo BCD non impaccato.

Per questi numeri sono previste le seguenti istruzioni di **aggiustamento ASCII**:

- **AAA**: aggiustamento ASCII per l'addizione;
- **AAS**: aggiustamento ASCII per la sottrazione;
- **AAM**: aggiustamento ASCII per la moltiplicazione;
- **AAD**: aggiustamento ASCII per la divisione.

Istruzione AAA

Supponiamo di voler addizionare 7 a 5: intanto, la rappresentazione nella memoria di questi due numeri sarà

7: 00000111
5: 00000101

Se l'operazione di somma fosse fatta tra due numeri in normale rappresentazione binaria, il risultato sarebbe 12 e sarebbe rappresentato in memoria semplicemente da

00001100

Questo invece non avviene con la codifica BCD non impaccata. Quello che accade con la codifica BCD è questo: viene per prima cosa effettuata la normale somma tra le due cifre (istruzione **ADD**) ed il risultato viene posto in AL: quindi AL contiene adesso

00001100

A questo punto, interviene l'istruzione **AAA**: per prima cosa, essa controlla il valore decimale del nibble inferiore di AL, cioè dei 4 bit più a destra; nel caso questo valore sia minore o uguale a 9, non compie alcuna operazione ed il risultato è quello contenuto in AL; nel caso, invece, come nel nostro esempio, il valore di questo nibble sia maggiore di 9 (il che significa che il risultato presenta più di una cifra decimale), essa esegue queste azioni:

- aggiunge 6 ad AL;
- aggiunge 1 (che sarebbe il riporto) ad AH (che era stato posto a 0);
- pone il nibble superiore di AL (i 4 bit più a sinistra) uguali a 0.

In tal modo, il risultato della somma è

00000001 00000010

che è esattamente la codifica BCD non impaccata del numero decimale 12, cioè di 7+5.

Ovviamente, questo è un caso semplice in quanto la somma è tra 2 numeri decimali con 1 sola cifra. Nel caso, invece, le cifre siano più di una, è necessario costruire un opportuno ciclo di istruzioni che eseguano le operazioni sopra descritte su ogni coppia di cifre corrispondenti.

Istruzione AAS

Come l'istruzione AAA, anche la **AAS** va usata DOPO aver eseguito la sottrazione tra due operandi decimali non impaccati validi. L'effetto di questa istruzione è di cambiare il contenuto di AL in un numero decimale non impaccato: se il valore del nibble inferiore è maggiore di 9, ad AL viene sottratto 6, ad AH viene aggiunto 1 ed il nibble superiore di AL viene azzerato.

Istruzione AAM

L'istruzione **AAM** corregge il risultato di una moltiplicazione tra due numeri BCD non impaccati. Essa funziona così: il contenuto di AL viene diviso per 10; il quoziente della divisione viene messo in AH, mentre il resto viene messo in AL.

Istruzione AAD

L'istruzione **AAD** cambia i contenuti dei registri AL ed AH (ciascuno dei quali contiene un numero BCD non impaccato) in un numero binario che è equivalente al contenuto del registro AL. Questo consente al programmatore di usare l'istruzione IDIV per ottenere il risultato corretto. Il registro AH deve essere 0 prima dell'istruzione IDIV. Dopo l'istruzione IDIV, il quoziente è posto in AL ed il resto è posto in AH. Entrambi i nibble superiori di AL ed AH vengono azzerati.

Ad ogni modo, la divisione con molte cifre è piuttosto complicata, non discuteremo il modo in cui è ottenuta.

Autore: **Sandro Petrizzelli**
e-mail: sandry@iol.it
sito personale: http://users.iol.it/sandry
succursale: http://digilander.iol.it/sandry1