1 Architettura 8086

L'8086, capostipite della famiglia 80X86, fu una dei primi microprocessori a 16 bit. In un 8086 sia il bus dati che la ALU erano a 16 bit, mentre esso poteva indirizzare fino a 1 MByte di memoria (20 bit di address bus). Questa memoria, vista la capacità di 64 kByte tipica dei microprocessori del tempo, era considerata una quantità enorme.
La memoria di un 8086 è organizzata in locazioni di 8 bit, per cui, dato che il data bus è di 16 bit, ogni accesso alla memoria trasferisce due Byte contemporaneamente.
L'8086 fu venduto per la prima volta nel 1978, contemporaneamente ad esso fu commercializzato l'8088, che ne è diverso solo perché ha un data bus di 8 bit. Questo permetteva di usare le tecnologie esistenti per la produzione a basso costo delle schede madre dei computer, che erano calibrate sulle CPU precedenti ad 8 bit. L'8088 ebbe un successo commerciale anche superiore all'8086, dato che fu adottato nella prima versione del PC IBM (XT). Funzionalmente era identico all'8086 solo che per trasferire dalla memoria numeri da 16 bit impiegava due cicli di memoria in sequenza, invece di concludere in un solo ciclo, come l'8086.
L'accesso alla memoria un Byte alla volta era gestito in modo automatico dalla CPU ed era del tutto "trasparente" al programmatore, così che scrivere programmi per un 8086 o un 8088 era esattamente la stessa cosa.
E' ormai molto tempo che l'8086 non viene più prodotto, ma i suoi successori, che ne condividono il codice binario, sono ancora oggi i tipi di CPU di maggior successo commerciale, per cui saper programmare un 8086 è ancora importante per utilizzare fino in fondo tutti i PC di oggi.

1.1 EU e BIU, prefetching

Se ragioniamo bene sul funzionamento di una CPU durante le sue sequenze di fetch e execute, ci rendiamo conto che, conclusa la fase di fetch, durante la fase di execute i bus possono rimanere inattivi. Ciò accade quando l'istruzione che è stata presa durante la fase di fetch lavora solo su registri interni. Il fatto che in questi casi i bus non vengano utilizzati si può sfruttare per rendere la CPU più veloce.
Invece che lasciare i bus inattivi si può leggere il codice operativo dell'istruzione successiva a quella che viene correntemente eseguita e immagazzinarlo in una coda all'interno della CPU. Quando la fase di execute sarà terminata la CPU potrà prendere il codice operativo della prossima istruzione dalla coda invece che dalla memoria, dato che la coda è già all'interno della CPU questo comporterà un grande risparmio di tempo. Questo significa che la fase di fetch e quella di execute si possono svolgere contemporaneamente, almeno in alcuni momenti.
Questo meccanismo, che configura una sorta di "cache delle istruzioni", era presente anche nel primo 8086 ed era detto "**prefetching**". Meccanismi del genere, molto più sofisticati, sono utilizzati anche nelle CPU più moderne.
La cache delle istruzioni di un 8086 è detta "coda di prefetch" ed è grande 6 Byte. Essa è molto piccola, considerando che le cache di istruzioni all'interno delle CPU odierne possono essere migliaia di kByte.
Dunque un 8086 durante l'esecuzione delle istruzioni che non coinvolgono la memoria legge le istruzioni successive a quella puntata dal Program Counter e le memorizza nella coda di prefetch, se essa non è già piena. Tutto ciò avviene in modo completamente automatico e trasparente al programmatore.
Per poter funzionare in questo modo l'8086 è suddiviso in due parti: una, detta **E**xecution **U**nit (EU), sovrintende alla sola fase di execute, mentre la BIU (**B**us **I**nterface **U**nit) è la parte che fa l'accesso ai bus e che, essendo indipendente dalla EU, è in grado fare il prefetching, contemporaneamente alla fase di execute. Questo fatto è messo in evidenza nella Figura 3, nella quale la EU e la BIU sono separate da una linea tratteggiata).
Se alla conclusione dell'execute la coda contiene qualcosa il fetch è già fatto e la EU può prendere dalla coda l'istruzione da eseguire. Se la coda è vuota si procede a fare una normale fase di fetch.
Naturalmente durante un'istruzione che fa accesso alla memoria, come ad esempio:

```
ADD [10], AL
```

il prefetching non potrà avere luogo, dato che la BIU è impegnata a leggere l'operando dalla locazione 10 ed a scrivere il risultato dell'istruzione alla stessa locazione.

!!!! da fare!

Figura 1: prefetching

1.1.1 Integrazione del coprocessore

Un'altra caratteristica avanzata dell'8086 rispetto alle CPU del suo tempo era la stretta integrazione con un prodotto complementare che poteva fare complesse operazioni aritmetiche in hardware, sgravando il software tecnico – scientifico di un peso molto consistente.
Si trattava del coprocessore aritmetico, in grado di effettuare i calcoli in virgola mobili e le funzioni trascendenti (seno, coseno) direttamente in hardware.

Il coprocessore studiato per l'8086 si chiama 8087 ed ha un set d'istruzioni speciale, che estende quello della CPU. Le istruzioni per il coprocessore vengono memorizzate insieme al programma normale, come se dovessero essere eseguite direttamente dalla CPU.
La CPU provvede al fetching delle istruzioni del coprocessore come fossero istruzioni proprie, poi, quando riconosce il codice operativo di una istruzione del coprocessore, glielo spedisce direttamente, fermandosi ad attendere i risultati.
In questo modo chi scrive il software può utilizzare le istruzioni dell'8087 come se fossero parte del set d'istruzioni dell'8086.
Dal 486 in poi il coprocessore è stato integrato all'interno del chip di ogni CPU della famiglia X86, per cui le istruzioni per l'aritmetica a virgola mobile sono oggi entrate a far parte a tutti gli effetti del set d'istruzioni della famiglia X86.

1.2 Registri dell'8086

1.2.1 I registri generali

I registri 8086 non hanno indirizzo. Per un programmatore Assembly il loro "indirizzo" è il loro "nome". Con l'Assembly 8086 per utilizzare un registro useremo il suo nome nell'istruzione relativa. Per esempio l'istruzione MOV AX, BX fa uso di due registri interni alla CPU i cui nomi sono AX e BX.

L'8086 ha quattro registri generali: AX, BX, CX e DX.

La lettera X dei nomi dei registri a 16 bit si può interpretare come "e**x**tended", perché, come vedremo, costituiscono un'estensione di registri a 8 bit.

Un operando o due operandi?

Consideriamo lo stesso tipo di istruzione, ma che usa registri diversi, per esempio:

```
; l'istruzione MOV (da "move" muovi) trasferisce nell'operando
; indicato a sinistra il valore dell'operando di destra
;
  Istruzione         Linguaggio macchina
  MOV AX, 0          B8 0000

  MOV BX, 0          BB 0000
```

Queste sono istruzioni a due operandi, un operando "sorgente" (0) ed una "destinazione" (AX o BX).
Se analizziamo il codice in linguaggio macchina, che abbiamo scritto accanto alle due istruzioni, ci rendiamo conto che l'unico operando indicato esplicitamente è il numero 0, mentre il registro viene indicato usando un diverso codice operativo.
Infatti la prima MOV ha codice operativo B8, mentre la seconda ha BB.
Il codice operativo indica quindi non solo che l'istruzione deve essere una MOV, ma quale deve essere il registro di destinazione del trasferimento. L'operando 0000 è invece identico nelle due istruzioni (ambedue devono trasferire 0 nel registro di destinazione).
Viste dalla CPU queste sono due istruzioni diverse, in quanto il loro codice operativo è diverso. Sarà compito del compilatore generare i codici operativi giusti, secondo quanto sta scritto nel file sorgente.
A livello di macchina le due istruzioni hanno quindi un operando solo, che vale 0, ma dal punto di vista logico, per il programmatore Assembly, le due istruzioni appena viste sono la stessa istruzione con operandi diversi (AX e BX).

Un registro, tre registri

Ogni registro generale di un 8086 è uno e trino. Nello stesso hardware, utilizzando gli stessi transistor per la memorizzazione, si mantiene il contenuto di tre registri.
Uno è un registro da 16 bit, due sono registri da 8 bit. I registri da 8 bit sono parte del registro a 16; uno dei due ne usa i bit della parte bassa, l'altro ne prende la parte alta.
Prendiamo per esempio il registro AX. Esso è diviso in due "sottoregistri" da 8 bit ciascuno, chiamati AL ed AH. A**L** condivide i bit della parte bassa di AX (**L**ow end), mentre A**H** usa quelli della parte alta (**H**igh end).

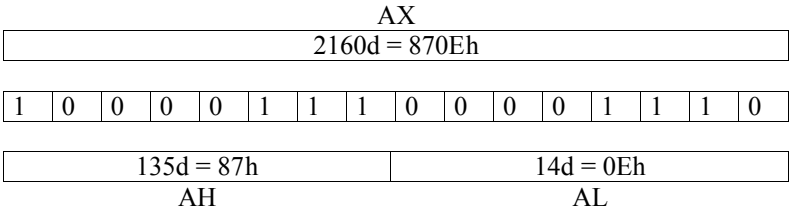


Figura 2: tre registri in uno

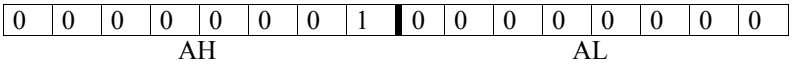
Quanto detto per AX vale anche per gli altri registri generali, quindi esistono i registri BL e BH, CL e CH, DL e DH, sono parte rispettivamente di BX, CX e DX.
Il registro a 8 bit ed il corrispondente a 16 bit funzionalmente sono registri "diversi". Le istruzioni fatte su AX sono a prima vista del tutto diverse da quelle fatte su AL. Peraltro AL ed AX condividono gli stessi bit, funzionano con gli stessi transistor, per cui cambiare uno di essi significa modificare anche il secondo. Ciò può essere un po' pericoloso, perché si presta ad effetti collaterali subdoli. Vediamo un esempio:

```
MOV BH, 20   ; uso BH per qualche scopo, che qui non interessa

.. faccio tante altre cose, che qui non interessano ..

MOV BX, 0    ; azzero BX, per qualche scopo sconosciuto, MA
             ; non ricordo che così azzero ANCHE BH!
; se da qui uso BH pensando che ci sia 20 la vita di questo
; programma diventa pericolosa! (in verità BH ora vale 0!)
```

E' importante notare che i registri da 8 bit "alti" o "bassi" sono registri completamente diversi fra loro: le operazioni effettuate su AL non hanno alcun effetto su AH, né su nessun altro registro, tranne AX di cui AL è la parte bassa. Ciò significa che, per esempio, non c'è nulla di diverso nell'usare i registri AL e DH piuttosto che AL e AH.
Vediamo un esempio che dovrebbe chiarire la relazione fra il registro da 16 bit ed i due da 8 bit:

```
MOV AL, 0    ; scrive 0 negli 8 bit meno
             ; significativi di AX
MOV AH, 1    ; scrive 1 nella parte alta di AX
```

Il risultato di queste operazioni è il seguente:



Volendo ottenere lo stesso risultato con una sola istruzione si può fare così:

```
MOV AX, 0100h; scrive 256 in AX
```

Un dubbio tipico, che è meglio fugare subito, è il seguente: se dopo un'operazione aritmetica "non c'è più posto in AL", parte del risultato va a finire in AH?

```
MOV AH, 127
MOV AL, 255
```

La situazione del registro AX dopo queste due istruzioni è la seguente:

AH	01111111b	11111111b	AL

Supponiamo ora di aggiungere uno ad AL:

```
ADD AL, 1
; qui Carry flag = 1, Zero flag = 1, Overflow flag = 0, Sign flag = 0
; CF = 1 perché l'istruzione ha dato un riporto fra numeri senza segno
; ZF = 1 perché l'istruzione ha dato il risultato di zero (in AL!)
; OF = 0 perché l'istruzione fra numeri con segno è corretta (-1+1=0)
; SF = 0 perché il risultato ha il bit più significativo = 0
```

Dato che dopo la ADD il risultato dovrebbe essere 256, che non ci sta in AL, a qualcuno potrebbe venire il dubbio che AL vada ad "invadere" AH, per cui il risultato di quelle tre istruzioni sia il seguente:

AH	00000001b	00000000b	AL	IPOTESI ERRATA!

Invece **non** è così. Dato che i due registri da 8 bit sono completamente indipendenti, il contenuto di AH non deve assolutamente essere toccato dalla ADD ed il vero risultato finale sarà il seguente:

AH	01111111b	00000000b	AL	IPOTESI GIUSTA!

Si noti che AH rimane esattamente uguale a prima, cioè è 127. In AL il risultato giusto non ci sta, negli otto bit a disposizione il risultato è zero, che è sbagliato, per cui verrà segnalato un errore aritmetico, con il flag di carry a 1.

Se l'ultima addizione fosse stata a 16 bit, come nell'istruzione seguente:

```
ADD AX, 1
; qui Carry flag = 0, Zero flag = 0, Overflow flag = 1, Sign flag = 1
```

Il risultato sarebbe stato la somma del numero a 16 bit 01111111 1111111b (7FFFh = 32767d) e di 1, per cui:

AH	1000000b	00000000b	AL
	1000000000000000b		AX

Dopo la ADD precedente i flag assumono i valori indicati per le seguenti cause:
- CF = 0 perché non c'è nessun riporto fra numeri senza segno (AX è di 16 bit!)
- ZF = 0 perché il risultato è diverso da zero (in AX!)
- OF = 1 perché sommando uno ad un numero positivo è venuto un numero negativo! (0111111111111111 + 1 = 1000000000000000)
- SF = 1 perché il risultato ha il bit più significativo = 1, ed è perciò negativo.

Vedremo più avanti che esiste un'istruzione particolare che, operando inizialmente con un numero da 8 bit "invade" un registro da 16 bit.
Infatti l'istruzione di moltiplicazione "parte" da AL e "finisce" in AX. Peraltro c'è da dire che l'istruzione MUL dell'8086 è davvero un po' "strana" e che l'"invasione" di AX avviene sempre, e non solo quando non c'è più posto in AL.

Uso specifico dei registri generali

I registri generali possono essere usati per tutti gli scopi, possono essere l'origine e la destinazione di tutte le operazioni "normali" che un 8086 può svolgere. Ciascuno dei registri generali ha anche una utilizzazione particolare, per la quale solo esso può essere usato. Vediamo quali sono questi utilizzi, con l'indicazione che la spiegazione in dettaglio verrà data quando si tratterà delle istruzioni coinvolte in queste utilizzazioni particolari.
Questo fatto può dare effetti collaterali se non si sa quali sono le conseguenze di ciascuna istruzione.

Il registro AX è l'**A**ccumulatore. Alcune particolari istruzioni usano AX, e solo AX, per contenere uno degli operandi di istruzioni aritmetiche e per memorizzarne il risultato
BX è detto "**B**ase" register. Come registro base viene usato per puntare alla memoria. Tramite BX è possibile accedere a locazioni di memoria, scrivendo preventivamente l'indirizzo al suo interno.
CX è il **C**ontatore (Counter) dell'8086 e viene usato da quelle istruzioni che necessitano di contare automaticamente.
DX viene detto registro **D**ati. Il suo uso non è particolare, viene usato per le elaborazioni. E' usato come parte dell'accumulatore nelle istruzioni di moltiplicazione e divisione (MUL e DIV).

Si fa notare che esistono codici operativi specifici per le istruzioni aritmetiche che usano AX come operando, per cui se è possibile è meglio usare AX, AL o AH per le operazioni aritmetiche. Se si farà così il codice occuperà meno memoria e potrà essere un po' più veloce nell'esecuzione.

Registro	Uso
AX	Registro di uso generale e **A**ccumulatore
BX	Registro di uso generale e Registro puntatore **B**ase
CX	Registro di uso generale e **C**ontatore
DX	Registro di uso generale e Registro **D**ati

Tabella 1: Uso particolare dei registri generali

Registri puntatori

I registri puntatori contengono indirizzi. Per il tramite di un registro puntatore è possibile fare riferimento alla memoria in modo indiretto. Prima si mette il valore di un indirizzo in un registro puntatore, poi si potrà leggere o scrivere alla locazione di memoria che ha quell'indirizzo.
Sull'accesso indiretto alla memoria si dirà molto in seguito, in questo momento ci interessa solo sapere quali sono i registri dell'8086 che possono essere usati come puntatori alla memoria.
Questi registri sono quattro, due sono detti registri "base", gli altri due registri "indice".
I registri base sono gli unici il cui nome inizia con la B, cioè BX e BP, mentre i registri indice sono quelli che terminano con una I, cioè SI e DI.
I registri puntatori di un 8086 sono sempre di 16 bit ed indicano una parte dell'indirizzo a 20 bit che viene detta "offset" (vedi più avanti "segmentazione" per i dettagli). Come offset non possono essere usati registri di 8 bit, anche se fanno parte di BX, come BH e BL. La CPU X86 dal 386 in poi possono fare uso di indirizzi di offset di 16, ma anche di 32 bit.
I nomi "Source Index" e "Destination Index" assegnati rispettivamente a SI e DI fanno riferimento ad alcune istruzioni particolari dette "istruzioni di stringa", che vedremo più avanti.
Per quanto esso non sia lo scopo per cui sono stati inventati, i registri puntatori possono essere anche usati per effettuare su di essi operazioni aritmetiche varie. Possono perciò essere usati anche alla stregua di registri generali, tanto che alcuni aggiungono ai registri generali ance SI, DI e BP.
La differenza importante con quelli che noi abbiamo chiamato registri generali è che SI, DI e BP non si possono dividere in due registri di 8 bit.

Registro Base	Uso
BX	Puntatore Base (e registro generale)
BP	Puntatore Base (**B**ase **P**ointer)
Registro Indice	Uso
SI	Puntatore Indice e Indice della stringa sorgente (**S**ource **I**ndex, nelle operazioni di stringa)
DI	Puntatore Indice e Indice della stringa destinazione (**D**estination **I**ndex, nelle operazioni di stringa)

Tabella 2: registri puntatori

Registri puntatori gestiti dalla CPU
Due registri a 16 bit dell'8086 contengono sempre indirizzi, sono dunque registri puntatori, ma sono gestiti in modo del tutto automatico dalla CPU. Dato che ad aggiornarli ci pensa la CPU il programmatore non dovrebbe mai cambiare i valori contenuti in questi registri, tranne che in casi del tutto eccezionali e decisamente "strani".
Il primo di questi due puntatori è IP (**I**nstruction **P**ointer). Come si può intuire da nome, IP punta alla prossima istruzione da eseguire e viene aggiornato automaticamente dalla CPU ad ogni istruzione eseguita. IP è dunque il Program Counter di un 8086, anche se per essere precisi bisognerebbe dire che IP è l'"offset" del Program Counter. Il ruolo di Program Counter spetta alla coppia di registri CS:IP, come avremo anche modo di esaminare quando tratteremo della segmentazione.
L'altro puntatore automatico è SP, lo **s**tack **p**ointer, che punta alla locazione dell'ultimo valore contenuto in una importante struttura dati detta "stack", che avremo modo di trattare in dettaglio più avanti.
SP è solo la parte di offset dell'indirizzo dello stack. L'indirizzo completo dello stack è dato dalla coppia di registri SS:SP

Registro	Uso
IP	Puntatore alla prossima istruzione (offset) (**I**nstruction **P**ointer)
SP	Puntatore allo stack (offset) (**S**tack **P**ointer)

Registro di stato (o registro dei flag)
Il registro di stato di un 8086 è di 16 bit ed è chiamato "**FLAGS** register".
Non tutti i bit di FLAGS sono utilizzati: nell'8086 se ne usano 9, nel 286 12.

I flag più importanti della famiglia X86 sono:

Flag aritmetici (status flags):

- O **O**verflow, indica l'errore aritmetico nelle operazioni con numeri con segno
- S **S**egno (Sign) è il bit più significativo del risultato, 1 se il risultato è negativo
- Z **Z**ero = 1 se il risultato dell'istruzione che l'ha sistemato è uguale a 0, 0 altrimenti
- A **A**djust flag (o Auxiliary Carry), analogo al riporto od al prestito per le operazioni in BCD (le istruzioni per l'aritmetica BCD non saranno illustrate in questo testo)
- P **P**arità (parity) = 1 se gli 8 bit più bassi del risultato contengono un numero pari di bit a uno
- C **C**arry (riporto o prestito) = 1 se c'è riporto o prestito sul risultato (operazioni senza segno), 0 altrimenti

Flag di controllo:

T **T**rap flag, utile al debugger per funzionare passo – passo

I **I**nterrupt flag, se a uno abilita il sistema d'interruzione (vedi prossimo volume)

D **D**irection flag, usato nelle istruzioni di stringa. Se = 1 l'istruzione di stringa procede decrementando gli indirizzi (per le istruzioni di stringa vedi capitolo "Altre istruzioni")

Flag di errore:

NT **N**ested **T**ask, per le operazioni multitasking (286>)

R **R**esume flag, per le operazioni multitasking (386>)

Nell'appendice "Set d'istruzioni della famiglia X86" si trova indicato il comportamento di ogni istruzione riguardo a ciascuno di questi flag.
Il dettaglio sul funzionamento di ciascun flag verrà dato quando tratteremo le istruzioni che ne fanno uso. Esistono anche altri flag, che introdurremo quando sarà opportuno.
Il registro di stato di un 386 è di 32 bit ed è chiamato **EFLAGS**. La sua parte bassa di 16 bit è identica a FLAGS del 286, che estende il registro FLAGS dell'8086 mantenendosi compatibile con esso. Il registro EFLAGS del 386 usa 14 flag.

1.2.2 Registri di segmento

I registri il cui nome finisce con S hanno un uso molto speciale. Non possono mai contenere dati da elaborare ma solo particolari puntatori, che indicano gli indirizzi di memoria d'inizio dei "segmenti", come vedremo più avanti.
Dato che i registri di segmento servono ogni volta che si accede alla memoria, essi sono stati dislocati nella BIU.
Per scoraggiare l'uso di questi registri per scopi diversi da quelli per cui sono stati progettati, non è possibile effettuare calcoli aritmetici su di essi, né trasferirvi direttamente dei numeri. L'unica cosa che è possibile fare sui registri di segmento è copiare su di essi il valore contenuto in un registro generale.

DS Puntatore all'inizio del segmento dati (**D**ata **S**egment)

SS Puntatore all'inizio del segmento stack (**S**tack **S**egment)

CS Puntatore all'inizio del segmento codice (**C**ode **S**egment)

ES Puntatore all'inizio dell'**E**xtra **S**egment

FS Puntatore all'inizio di un altro segmento extra (solo 386 e >)

GS Puntatore all'inizio di un altro segmento extra (solo 386 e >)

Per quanto riguarda la spiegazione dell'accesso in memoria tramite i registri di segmento si rimanda al capitolo relativo alla segmentazione.
Notiamo soltanto che, come si vede nella Figura 3, per la produzione dell'indirizzo di memoria è utilizzato un particolare "sommatore d'indirizzo". Questo sommatore realizza una strana somma fra due quantità di 16 bit. Il risultato di questa somma è l'indirizzo di 20 bit. Uno degli operandi di questa somma è sempre e comunque uno dei registri di segmento, come si può apprezzare anche dalla figura.
Nella documentazione Intel ordinaria non sono indicati registri temporanei, che comunque non sono interessanti per il programmatore, che, come già visto, non può farne uso.

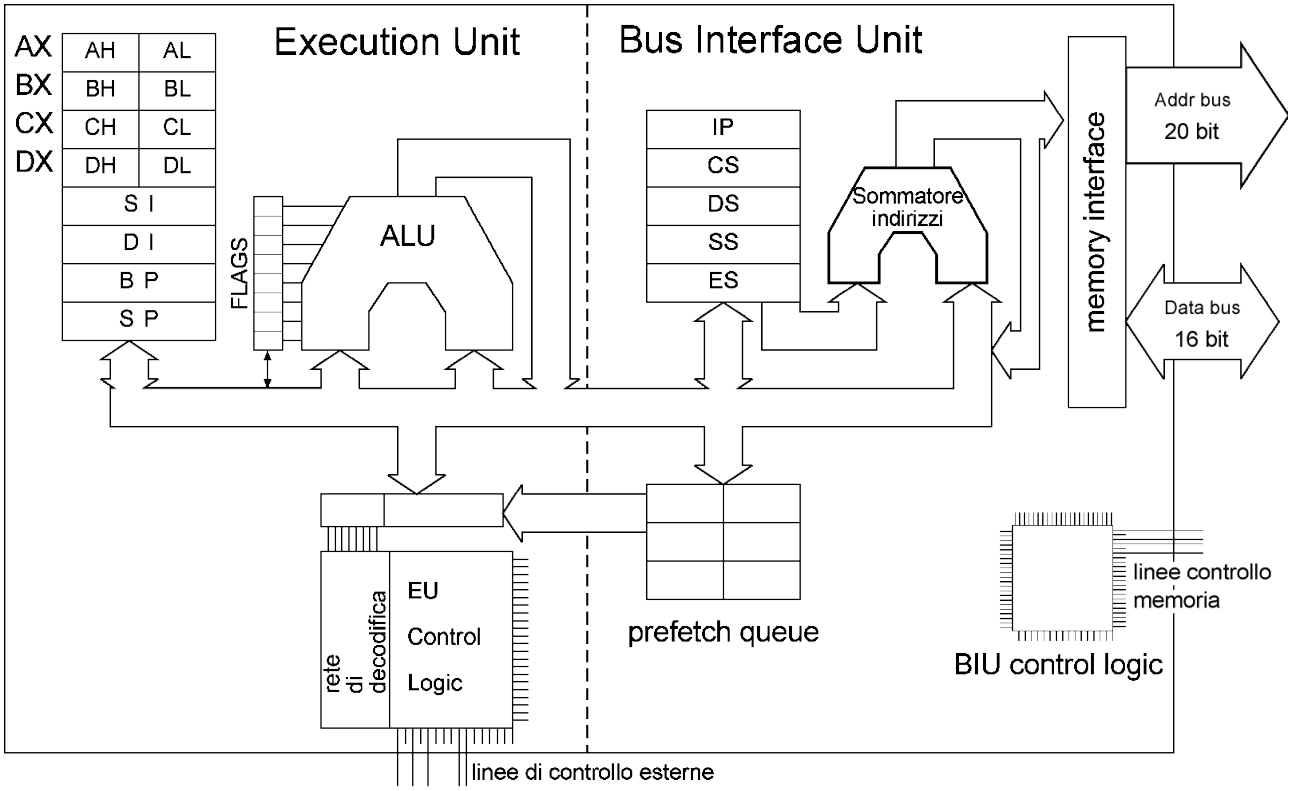


Figura 3: schema semplificato dell'architettura dell'8086

La figura non è esattamente uguale a quella proposta dalla casa costruttrice nella sua documentazione.
Viene disegnato un solo bus interno, mentre Intel ne indica tre.
E' indicato anche un collegamento fra la cache delle istruzioni ed un registro temporaneo d'istruzione, che la casa costruttrice non menziona. Questo solo per rendere più chiaro il fatto che il codice operativo della prossima istruzione da eseguire può provenire dal data bus, facendo il percorso normale di una fase di fetch oppure può provenire direttamente dalla coda di prefetch.

1.3 Inizializzazione del sistema.

Dopo la ricezione del segnale di reset l'8086 si procura il codice della prima istruzione da eseguire all'indirizzo a 20 bit 0FFFF0h, cioè 16 Byte prima della fine della sua memoria.
A quella locazione è presente una memoria elettronica non volatile, ROM o EPROM. La prima istruzione farà saltare ad un piccolo un programma di test e di caricamento, residente in ROM. Questo programma, normalmente detto "POST" (**P**ower **O**n **S**elf **T**est) verificherà il funzionamento dei sottosistemi del computer, caricherà l'ora dall'orologio in tempo reale, poi leggerà dall'hard disk la parte del Sistema Operativo che deve rimanere sempre in memoria, la caricherà in memoria e la farà eseguire.
A quel punto il computer sarà pronto da usare.
La sequenza di accensione di un computer, con il caricamento iniziale del Sistema Operativo, viene anche detta sequenza di "**bootstrap**" (lacci di stivale!), o "**boot**".

1.4 Cenni alla famiglia X86

Dopo l'8086 (1978) vennero realizzati l'80186 e l'80286 (1982). Mentre l'80186 non aveva sostanziali novità per quanto riguarda il software, il "286" introduceva la "modalità protetta", sulla quale si tornerà in dettaglio nel prossimo volume. Questo modo di funzionare della CPU aiuta il software nella realizzazione di Sistemi Operativi "multiprogrammati" (su di essi si spenderanno alcuni capitoli del Volume secondo).
Le più grandi novità software della famiglia 80X86 furono introdotte con l'80386 (1985), una CPU a parallelismo interno di 32 bit.
Il funzionamento in modo protetto del "386" era molto più completo di quello della CPU precedente. I Sistemi Operativi per PC del giorno d'oggi (es. Windows, Unix, Linux) sono stati scritti per il set d'istruzioni del 386 e non possono funzionare con le CPU precedenti.
Il 386 ha registri a 32 bit che inglobano quelli a 16 bit delle CPU precedenti della famiglia. I nomi dei registri a 32 bit del 386 iniziano con una E (**e**nhanced, "migliorato"), seguita dal nome del corrispondente registro 8086.
Nel 386 ci sono perciò i registri da 32 bit EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP.
I registri di segmento rimangono a 16 bit, ma ne sono aggiunti due. Essi sono perciò CS, DS, SS, ES, FS, GS.

Il 386 ha molte nuove istruzioni, per lavorare con i registri a 32 bit e per gestire la modalità protetta.
Le CPU X86 successive al 386 non hanno rivoluzionato il modo di scrivere il software per X86; dal punto di vista del programmatore, 486 (1991), Pentium (1993), Pentium II (1995) e le CPU concorrenti AMD, sono solo dei 386 molto più potenti.
Le CPU AMD della serie "Opteron" sono un'estensione a 64 bit dell'architettura X86.
Nel seguito di questo volume si farà cenno ad alcune delle particolarità del set d'istruzioni e dei registri del 386, pur senza scendere eccessivamente nei dettagli. Una trattazione più esaustiva verrà fatta nel Volume secondo.

1.4.1 Multiplexing dei bus

L'8086 ha una caratteristica hardware particolare, condivisa anche da altre CPU modestamente complesse, detta "multiplexing dei bus".
Questa tecnica permette di condividere gli stessi piedini del circuito integrato della CPU per le funzioni di address bus e data bus. La cosa è possibile perché ogni volta che si deve accedere ai dati in memoria bisogna prima sempre emettere l'indirizzo sull'address bus. Quindi un accesso alla memoria si sviluppa sempre in due fasi:
1 - emissione dell'indirizzo
2 – trasferimento dei dati
Sfruttando questa sequenzialità alcuni tipi di CPU usano gli stessi piedini per gli indirizzi durante la fase 1 e per i dati da trasferire durante la fase 2.
L'indirizzo però deve essere mantenuto attivo per tutto il tempo dell'accesso alla memoria per cui sono necessari dei dispositivi logici esterni ("latch") che memorizzano le linee dei bus nel momenti in cui è presente l'indirizzo e lo mantengono anche quando il significato dei piedini della CPU cambia, divenendo linee di dati.
Il multiplexing dei bus viene usato per risparmiare nel numero dei piedini del microprocessore, cosa che implica una maggiore affidabilità del chip (quello che non c'è non si rompe) ed un minor costo della scheda madre.
CPU molto semplici come certi microcontrollori per sistemi embedded di solito non ne fanno uso per limitare il numero dei componenti esterni.
Anche microprocessori molto complessi, che hanno diverse centinaia di piedini, di solito non usano piedini in multiplex, dato che il problema di avere molti piedini in quei µp deve essere risolto comunque ..
Le CPU X86 dal 286 in poi non usano il multiplexing dei bus.

!!!! da fare!

Figura 4: multiplexing dei bus

Curiosità

Nel romanzo fantastico "Le avventure del Barone di Munchhausen " (R.E. Raspe , 1785) il protagonista riesce a volare semplicemente tirando forte sui lacci dei suoi stivali (bootstraps!).
Il termine sequenza di "bootstrap", usato per indicare la sequenza di avviamento di un computer, fa riferimento a questo racconto. Come il Barone di Munchausen si "autosolleva" con i suoi stessi lacci, così un computer, durante il "bootstrap", carica il suo software con il suo stesso software.