NUMERO : 217 DATA : 23/02/2012

# A P P U N T I

STUDENTE : Garraffa

MATERIA : Sistemi a Microprocessori

Prof. Mezzalama

# Capitolo 1

## Classificazione elaboratori

Un microprocessore è un'implementazione fisica di un processore (CPU) realizzata su un singolo circuito integrato.
Possiamo individuare diverse categorie di microprocessori, relativamente ad ogni categoria alcuni parametri sono di maggiore importanza rispetto ad altri, per cui non si guarda sempre e solo alla potenza di calcolo come principale parametro di riferimento.
Ad esempio, un sistema **Desktop** viene valutato in base a dei parametri differenti rispetto ad un sistema **Server**. Infatti per i sistemi desktop sono rilevanti le **prestazioni grafiche** e il **rapporto prezzo/prestazioni**, mentre per i sistemi server i due parametri più importanti sono il **throughput** e la **dependability** (affidabilità).
Il throughput stabilisce il numero di operazioni che vengono effettuate dal sistema nell'unità di tempo.
La dependability, o affidabilità, indica la capacità di fornire un servizio che può essere considerato affidabile. Il parametro che viene considerato per la valutazione della dependability è il **MTBF (mean time between failure)**, dato dalla somma tra il MTTF (mean time to failure) e il MTTB (mean time to repair).
Una fondamentale classificazione tra i microprocessori distingue le seguenti tre macrocategorie:

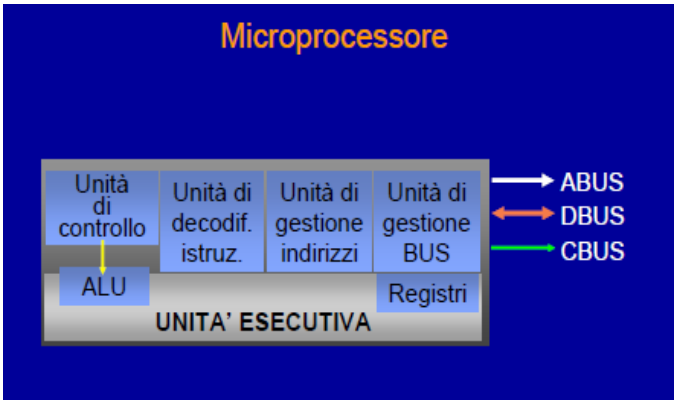
- **General purpose**, sono microprocessori ad alte prestazioni e versatili, non limitati ad un solo utilizzo. A questa categoria appartengono i processori dei PC.
- **Special purpose**, sono microprocessori utilizzati per scopi specifici (come quelli grafici).
- **Embedded**, sono microprocessori realizzati per svolgere compiti specifici su determinate piattaforme hardware, il loro utilizzo è limitato a tali piattaforme.

I sistemi embedded non hanno un sistema operativo che li gestisce. Il sistema operativo fa in modo che nei sistemi general purpose venga realizzata un'architettura a livelli (layered) che permetta di nascondere i dettagli specifici delle applicazioni e di realizzare un'economia di scala.
Ad oggi il mercato è costituito per il 98% da microprocessori per applicazioni embedded, tuttavia i microprocessori per i personal computer, pur rappresentando solo lo 0,2% del numero totale di pezzi prodotti, assorbono circa la metà del fatturato complessivo.

## Architettura dei microprocessori

Un microprocessore è composto da più unità:

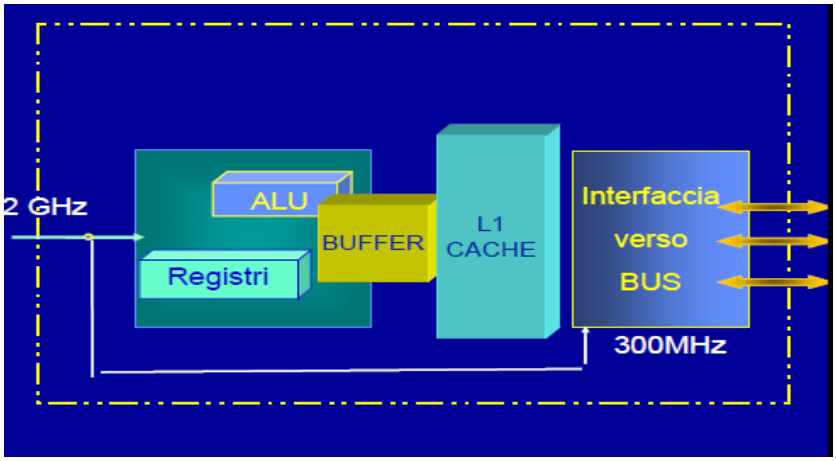
- una unità di controllo;
- una unità di decodifica delle istruzioni;
- una unità di gestione degli indirizzi;
- una unità di gestione del bus;
- una unità di esecuzione, che contiene la ALU (Algebric Logical Unit) e i registri.



Un altro modo per incrementare le prestazioni di un sistema è cambiarne l'architettura, ottenendo quindi uno **sviluppo architetturale**.
I due principali fattori che hanno determinato uno sviluppo architetturale negli ultimi anni sono:

- **l'aumento del parallelismo di esecuzione**;
- **il disaccoppiamento dei bus di sistema dalla cpu mediante buffer**.

L'aumento del parallelismo può essere effettuato in più modi, uno di questi è sicuramente l'utilizzo di pipeline, e più recentemente si è puntato molto su processori superscalari.
A queste due innovazioni architetturali sono associati due tipi di parallelismo: quello a livello di istruzione (ILP) e quello a livello di thread (TLP).
Nei sistemi recenti multicore, si è puntato più ad ottimizzare il TLP, rendendo le pipeline dei singoli core più corte, e dovendo gestire le problematiche legate alla gestione dell'esecuzione di più threads su più core.
Il numero di core in ogni processore è destinato ad aumentare nei prossimi anni, secondo la **nuova legge di Moore** raddoppierà ogni 18 mesi.
Il disaccoppiamento dei buffer di sistema dalla cpu viene effettuato con le code di prefetch e le cache.




I bus non dialogano direttamente con la CPU, ma comunicano con la L1 cache tramite un interfaccia (BIE, Bus Interface Unit), tramite le cache viene riempito opportunamente un buffer di prefetch, che deve contenere le istruzioni che più probabilmente verranno eseguite dopo quella corrente.
Il riempimento di tale buffer viene effettuato tramite algoritmi di predizione opportuni, che nei processori Intel fanno uso di un Branch Target Buffer (BTB).

risulta essere di 4GB ($2^{32}$ bytes). Questa modalità permette l'utilizzo di strutture di gestione del multitasking, della protezione della memoria e del paging.
Si passa a tale modalità modificando alcuni registri di controllo e alcuni flag.
Infine, la terza modalità è la **Virtual 8086 Mode**, che permette l'esecuzione di software scritto in Real Mode che non possono essere eseguiti in sistemi che usano la Protected Mode, emulando l'ambiente IA-16.
L'emulazione di IA-16 può pertanto essere effettuata tramite la Real Mode, che limita semplicemente il set di istruzioni utilizzabili a quelle a 16 bit, e la Virtual 8086 Mode che la effettua utilizzando un altro task che emula il comportamento logico dell'8086.

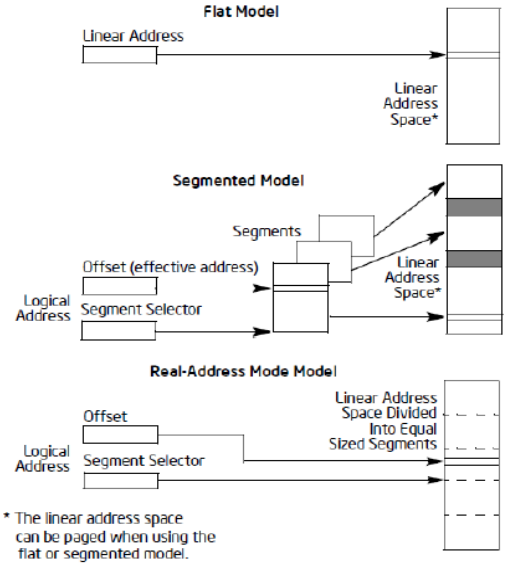
## Registri in IA-32

A seconda della diversa modalità in cui si opera, in IA-32 si ha un diverso numero di registri visibili e dei loro bit.

| VM86 | | | Real Mode | | | Protected Mode | | |
|---|---|---|---|---|---|---|---|---|
| **Registro** | **Descrizione** | **Bit** | **Registro** | **Descrizione** | **Bit** | **Registro** | **Descrizione** | **Bit** |
| IP | Instruction Pointer | 16 | EIP | Instruction Pointer | 32 | EIP | Instruction Pointer | 32 |
| SI | Source Index | 16 | ESI | Source Index | 32 | ESI | Source Index | 32 |
| DI | Destination Index | 16 | EDI | Destination Index | 32 | EDI | Destination Index | 32 |
| FLAG | Status Flags | 16 | EFLAG | Status Flags | 32 | EFLAG | Status Flags | 32 |
| AX | Accumulator | 16 | EAX | Accumulator | 32 | EAX | Accumulator | 32 |
| BX | Base | 16 | EBX | Base | 32 | EBX | Base | 32 |
| CX | Counter | 16 | ECX | Counter | 32 | ECX | Counter | 32 |
| DX | Data | 16 | EDX | Data | 32 | EDX | Data | 32 |
| SP | Stack Pointer | 16 | ESP | Stack Pointer | 32 | ESP | Stack Pointer | 32 |
| BP | Base Pointer | 16 | EBP | Base Pointer | 32 | EBP | Base Pointer | 32 |
| CS | Code Segment | 16 | ECS | Code Segment | 16 | ECS | Code Segment | 16 |
| SS | Stack Segment | 16 | SS | Stack Segment | 16 | SS | Stack Segment | 16 |
| DS | Data Segment | 16 | DS | Data Segment | 16 | DS | Data Segment | 16 |
| ES | Extra Segment | 16 | ES | Extra Segment | 16 | ES | Extra Segment | 16 |
| | | | FS | Extra Segment | 16 | FS | Extra Segment | 16 |
| | | | GS | Extra Segment | 16 | GS | Extra Segment | 16 |
| | | | | | | TR | Task | 16\|32\|16 |
| | | | | | | LDTR | Local Desc. Table | 16\|32\|16 |
| | | | | | | IDTR | Interrupt Desc. Table | 32\|16 |
| | | | | | | GDTR | Global Desc. Table | 32\|16 |
| | | | | | | CR3 | Control | 32 |
| | | | | | | CR2 | Control | 32 |
| | | | | | | CR1 | Control | 32 |
| | | | | | | CR0 | Control | 32 |
| | | | TR7 | Test | 32 | TR7 | Test | 32 |
| | | | TR6 | Test | 32 | TR6 | Test | 32 |
| | | | | | | DR7 | Debug | 32 |
| | | | | | | DR6 | Debug | 32 |
| | | | | | | DR5 | Debug | 32 |
| | | | | | | DR4 | Debug | 32 |
| | | | | | | DR3 | Debug | 32 |
| | | | | | | DR2 | Debug | 32 |
| | | | | | | DR1 | Debug | 32 |
| | | | | | | DR0 | Debug | 32 |

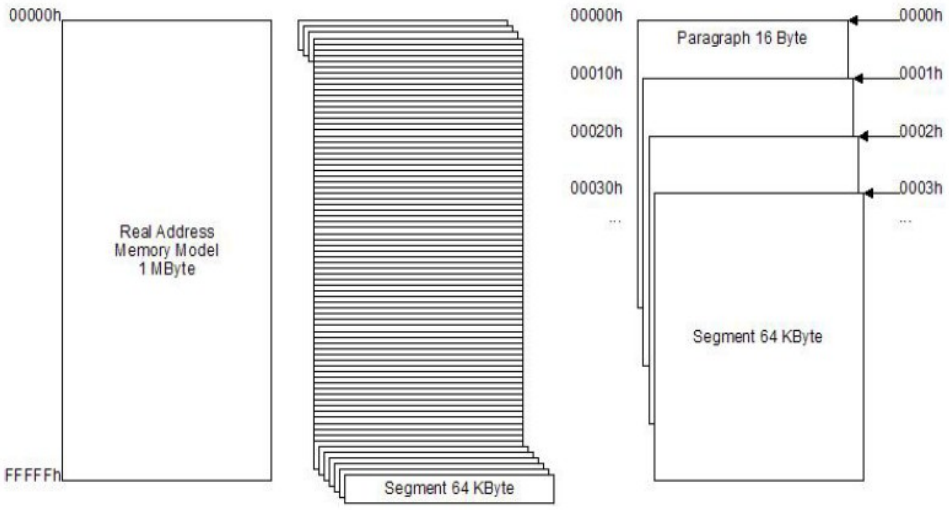
Analizzando la precedente tabella, notiamo che i registri di segmento sono ampi 16 bit, facendo in modo che siano considerabili un totale di $2^{16}$ segmenti, mentre i registri di indirizzamento possono avere 16 o 32 bit a seconda della modalità di funzionamento.
I registri di controllo e di debug sono presenti solo nella protected mode, mentre i registri di test sono presenti nella protected mode e nella real mode.
Nella virtual mode 8086 vengono annullate le estensioni a 32 bit, in quanto l'8086 era a 16 bit.
Esistono 4 tipi di registri:

- registri dato;
- registri di controllo;



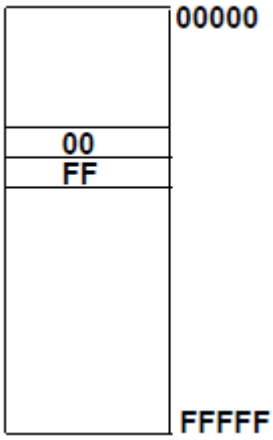

# Real address mode memory model

I segmenti sono visibili al programmatore a livello di istruzione, permettono di organizzare la memoria dividendola in porzioni omogenee (dati, codice e stack), anche più di una per tipo.
I segmenti sono da 64 Kbyte, in overlapping di 16 Byte ognuno, tale distanza viene detta paragrafo.

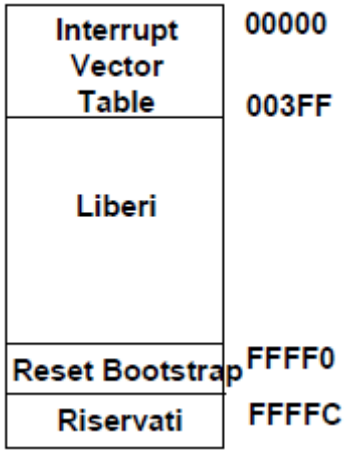



L'indirizzo fisico (**Physical Address**) viene ottenuto fornendo un indirizzo di segmento (Segment Address) e un offset (Effective Address), entrambi di 16 bit. Ogni qual volta l'architettura deve generare un indirizzo fisico da mettere sull' A-bus, l'indirizzo di segmento viene shiftato di 4 bit (ciò equivale ad una moltiplicazione per 16) e viene sommato all'offset. Pertanto l'indirizzo fisico è lungo 20 bit, e permette di indirizzare 1MB di memoria. Una volta fissati i valori dei registri di segmento, un indirizzo fisico è determinato univocamente dal valore di un registro di indirizzamento a 16 bit.

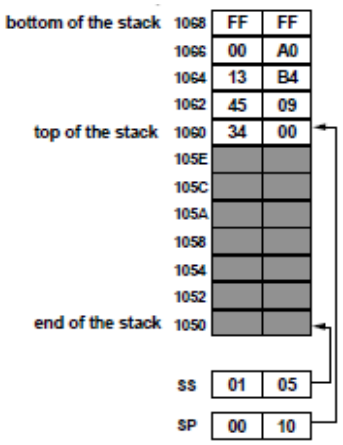
specificato:




Alcune parti della memoria non sono libere, ma riservate. Vediamo quali sono le parti riservate della memoria nell'architettura 8086.




I byte da 00000H a 003FFH sono occupati dalla **interrupt vector table (IVT)**, che contiene i 256 puntatori alle routine di gestione degli interrupt.
In modo reale, ciascun puntatore è formato da 4 byte, due dei quali sono per l'indirizzo del registro di segmento, mentre gli altri due sono per l'offset. I primi 32 puntatori sono riservati a eccezioni interne del processore.
I byte da FFFF0H a FFFFCH sono riservati al **reset bootstrap**, che contiene una JMP alla procedura di bootstrap del sistema.
Infine, i byte da FFFFCH a FFFFFH, sono riservati alla **versione del bios in uso**.
L'8086 prevede, inoltre, alcune strutture hardware per la gestione di uno **stack,** che corrisponde all'area di memoria la cui testa è puntata dal registro di segmento SS.



svuotato e la Execution Unit deve attendere il fetch della nuova istruzione.
La BIU, per ottimizzare il processamento delle istruzioni, effettua il **prefetch** delle istruzioni, cioè carica preventivamente la coda delle istruzioni.

# Pipeline

Il ciclo di vita di un'istruzione può essere scomposto in più fasi, che possono essere svolte spesso in maniera indipendente l'una dall'altra.
Ad esempio possiamo individuare 4 fasi: il **fetch**, che consiste nel caricamento dell'istruzione dalla memoria, la **decode**, che consiste nella decodifica del codice operativo dell'istruzione, la **execute**, che consiste nell'esecuzione dell'operazione associata all'istruzione, e la **write back**, che consiste nella scrittura del risultato.
Ci sono delle condizioni in cui tali fasi richiedono più tempo del normale per essere completate, si dice in questo caso che la fase entra in **criticità**.
La fase di fetch entra in criticità se l'istruzione deve essere prelevata dalla memoria e non dalla cache, la fase di decode se l'operando deve essere prelevato dalla memoria, la fase di execute se l'operazione da eseguire è complessa (come una divisione), infine la fase di write back entra in criticità se il risultato va scritto in memoria e non in un registro.
Il meccanismo del pipeline consiste nel fare in modo che mentre un'istruzione effettua la fase n-esima, l'istruzione successiva sia in grado di eseguire la fase (n-1)-esima a meno che non si verifichino determinate condizioni di stallo.
L'idea è quella di aumentare il parallelismo di esecuzione delle istruzioni, al fine di **aumentare il throughput**.
Il tempo di esecuzione delle varie fasi può essere diverso, pertanto le istruzioni che si trovano nelle fasi più lente devono attendere che le fasi più lente (e quindi più complesse) terminino. In ogni caso il valore maggiore tra i tempi di esecuzione delle varie fasi, determina il **clock della pipeline**, che permette lo svolgimento in maniera sincrona delle operazioni e non è legato strettamente al clock del sistema.
Se la differenza tra il tempo di esecuzione di due fasi è molto ampia, l'efficienza della pipeline stessa diminuisce.
A tal proposito parliamo di **pipeline ideale** qualora ogni fase abbia uguale durata; in questo caso il tempo di esecuzione di una istruzione si riduce di un fattore pari al numero degli stadi k, rispetto al caso senza pipeline (cioè il throughput aumenta di un fattore k).
Se la durata delle varie fasi non è uguale, si parla di **pipeline imperfetto.**
Sia Ts il tempo di esecuzione di una istruzione senza pipeline, e Tp il tempo di esecuzione di una istruzione con pipeline, nel caso generico di pipeline imperfetto si ha:

$$Tp <= Ts$$

Nel caso di pipeline ideale, le prestazioni migliorano linearmente con l'aumentare delle fasi k, tuttavia per k molto alto non è possibile mantenere il vincolo di pipeline ideale, in quanto l'overhead dovuto alla gestione delle varie fasi va aumentando ed è difficile scomporre le istruzioni in un numero elevato di fasi di uguale durata.
Dimostriamo adesso che nel caso di pipeline ideale l'incremento prestazionale è pari al numero di fasi (o stadi) k.
Sia $\tau_i$ la durata dello stadio i-esimo e $\tau_{max} = max(\tau_i)$.
Il tempo di esecuzione di una istruzione risulta essere pari a $k\tau_{max}$(pari al tempo che impiega l'istruzione per scorrere tutta la pipeline)**,** il tempo di esecuzione di N istruzioni senza pipeline risulta essere $N\ k\ \tau_{max}$ e il tempo di esecuzione di N istruzioni con pipeline risulta essere $(N-1)\ \tau_{max} + k\ \tau_{max}$.
Calcoliamo il fattore di accelerazione, determinato come il rapporto tra il tempo di esecuzione di N istruzioni senza pipeline e il tempo di esecuzione di N istruzioni con pipeline.

# I/O nell'architettura 8086

L'accesso alle periferiche avviene spesso attraverso speciali locazioni associate ad un certo indirizzo.
L'accesso a tali locazioni può essere effettuato:

- in memory mapped, l'accesso avviene attraverso una normale istruzione che accede ad una locazione di memoria, parte dello spazio di indirizzamento è assegnato alle periferiche.
- In isolated I/O, l'accesso avviene attraverso speciali istruzioni (IN e OUT), alle periferiche vengono associati degli indirizzi dedicati.

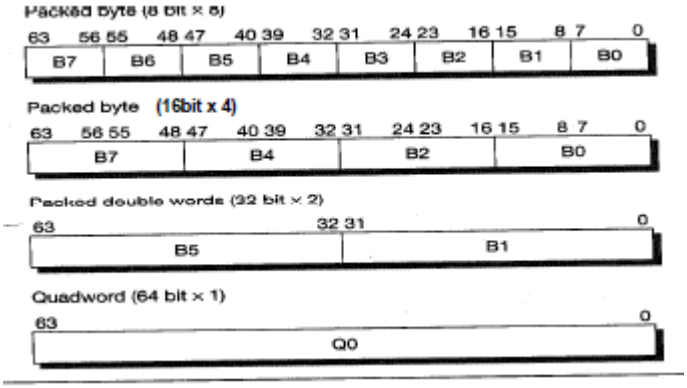
Lo spazio di indirizzamento è al più di 64KB, in quanto gli indirizzi riferiti ai periferici sono espressi, al massimo, su 16 bit.

# Registri e feature speciali

L'evoluzione dell'architettura ha portato all'introduzione di registri special purpose per sfruttare le nuove features.
La **FPU (Floating Point Unit)** permette di eseguire operazioni in virgola mobile via hardware. Esegue operazioni fixed, floating point e calcola funzioni trascendenti e trigonometriche (infatti per il calcolo del coseno e del seno, esistono istruzioni come FCOS e FSEN). L'unità è un processore con architettura a stack e fa uso dei seguenti nuovi registri specifici:

- FPU Registers Stack (80 bit);
- FPU Instruction Register (48 bit);
- FPU Data Pointer (48 bit);
- FPU Control, Status, Tag Registers (16 bit);
- FPU Opcode Register (11 bit).

La **MMX (MultiMedia Extension)** è una tecnologia che permette di elaborare dati organizzati a blocchi da 64 bit. I dati vengono interpretati come sequenze di bit, pertanto senza ulteriori estensioni non vi è il supporto per l'aritmetica in complemento a 2 o quella in virgola mobile. E' divenuta una necessità a causa del crescente utilizzo di interfacce grafiche a finestra. Si hanno a disposizione ulteriori registri e istruzioni dedicate.
Fa uso di 8 MMX Registers di 64 bit, tali registri sono utilizzabili in diversi formati, cioè come sequenze di 8 byte, come quaterna di word o come coppia di double word. Le istruzioni al massimo operano su 64 bit e sono istruzioni che prevedono operazioni aritmetiche, di trasferimento e logiche.

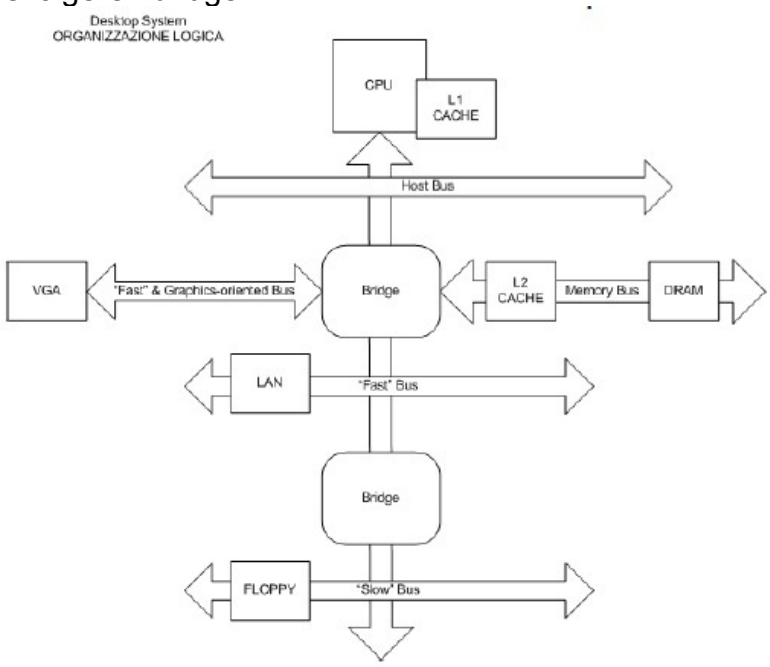



Le **SSE (streaming SIMD Extension)** estendono la tecnologia MMX, aggiungendo tutto il necessario per manipolare valori floating point contenuti in registri a 128 bit. Tale tecnologia mette a disposizione nuovi registri e nuove istruzioni.
Fa uso di 8 XMM Registers di 128 bit, e di un MXCSR Register di 32 bit, che contiene flag di controllo e di stato associati alle operazioni eseguite.
Permette l'utilizzo di istruzioni che effettuano operazioni aritmetiche intere e floating point

# Capitolo 3

## Architettura a bus

I sistemi desktop odierni hanno in comune il fatto che gestiscono più bus, in genere ne vengono gestiti almeno due o tre.
Ciò è dovuto al fatto che in ogni sistema esistono dispositivi che comunicano a diverse velocità con la CPU, pertanto usando un singolo bus, tale bus dovrebbe lavorare alla frequenza più bassa tra quella dei dispositivi ad esso collegati, e ciò comporterebbe un notevole degrado prestazionale.
Per cui gli odierni calcolatori implementano un'**architettura a bus multilivello**, in cui ad ogni livello vengono collegati dispositivi con velocità simili.
I vari bus sono interconnessi tramite dei **bridge**, che effettuano una sorta di routing dei segnali e gestiscono il cambio della frequenza di trasferimento da un bus all'altro.
Aumentando i livelli di bus, aumenta il ritardo che un dispositivo collegato al bus più lontano dalla CPU, deve affrontare per comunicare con essa, e ovviamente, aumenta il lavoro che devono svolgere i bridge.




Vediamo adesso come l'organizzazione logica descritta, viene realizzata nell'architettura 8086, usata dai sistemi desktop di oggi.
Esistono due bridge: il **Northbridge** e il **Southbridge**.
Il Northbridge si occupa della gestione efficace della memoria attraverso il **cache control** (che gestisce i miss e gli hit in memoria aggiornando opportunamente le cache tramite delle transazioni), e non gestisce gli I/O.
Ha come bus di riferimento il **Front Side Bus (FSB)**, che lo collega alla CPU, e il bus orientato alla grafica **AGP**.
Il Southbridge gestisce le periferiche di I/O, può gestire connessioni con le periferiche quasi punto-punto, tramite bus, e di tipo PCI.
E' collegato al bus **LPC (Low Pin Count)**, che sostituisce il vecchio bus ISA minimizzando il numero di pin di connessione con le periferiche. A tale bus è collegato il Super I/O che funge da interfaccia di comunicazione nei confronti delle periferiche più lente, come la porta seriale e parallela e le porte PS/2.
Anche il **Bios** è collegato al bus LPC, pertanto, quando un processo invoca un'istruzione del Bios (questa circostanza si verifica in alcuni sistemi operativi), l'istruzione deve

# Il ciclo di bus

Il ciclo di bus è una sequenza di eventi attraverso cui il microprocessore comunica con la memoria, le periferiche o l'interrupt controller.
In funzione della CPU, la durata di un ciclo di bus può essere di 2 o 4 cicli di clock del bus, in ogni caso è un numero fisso nell'architettura 8086, in quanto tale architettura implementa un bus sincrono.
Il throughput del bus cresce proporzionalmente alla crescita del parallelismo e della frequenza di clock del bus stesso.
Attualmente il front side bus arriva a una frequenza di lavoro di 200MHz, non è possibile aumentare tale frequenza per problemi legati a disturbi elettromagnetici e trasmissioni di linea.
Ricordando che:

$$\lambda = c/f$$

otteniamo che se la frequenza è di 200MHz, la lunghezza d'onda è pari a 15 cm. Se le piste hanno lunghezza circa uguale a lambda quarti, si ha il problema della trasmissione del segnale nell'etere.
Una tecnica usata per aumentare il throughput è stata quella del **double data rate**, che prevede il trasferimento di informazioni sia sul fronte di salita che su quello di discesa del clock.
Per misurare la velocità del bus scollegandola dal parallelismo e dal timing, utilizziamo la misurazione in Gt/s (Giga transfer per seconds), e in Mt/s (Mega transfer per seconds).
Esistono 4 tipologie di ciclo di bus:

- ciclo di **READ**;
- ciclo di **WRITE**;
- ciclo di **IDLE**;
- ciclo di **WAIT;**

Un ciclo di READ è formato da 4 periodi, durante il periodo T1 viene posto l'indirizzo sull'address bus, nel periodo T2 si attende che la circuiteria per selezionare l'indirizzo scelto venga attivata, e nel data bus viene messo il valore Z.
Nei periodi T3 e T4, infine, la memoria pone il dato letto nel data bus.
Anche un ciclo di WRITE è formato da 4 periodi, durante il primo periodo T1 viene posto l'indirizzo sull'address bus, sul successivo periodo T2 viene posto il valore da scrivere sul data bus, e negli ultimi due periodi T3 e T4 la memoria legge il dato dal data bus e lo scrive in memoria.
I cicli di IDLE sono quei cicli in cui non avviene nessuna transazione di bus, ciò accade quando la CPU non necessita di nuovi dati e contemporaneamente la coda delle istruzioni è piena.
Essendo il bus sincrono, deve essere trovato un modo per permettere alle periferiche lente di comunicare con la CPU. Infatti se le periferiche sono troppo lente e non riescono ad effettuare i trasferimenti nei tempi definiti dal clock del bus, sarebbe necessario usare un clock più lento con un conseguente degrado prestazionale.
Per risolvere questo problema, la frequenza del clock viene mantenuta, ma vengono inseriti dei cicli di WAIT tra T3 e T4, per permetere al dispositivo lento di ultimare il trasferimento.

# Esempi

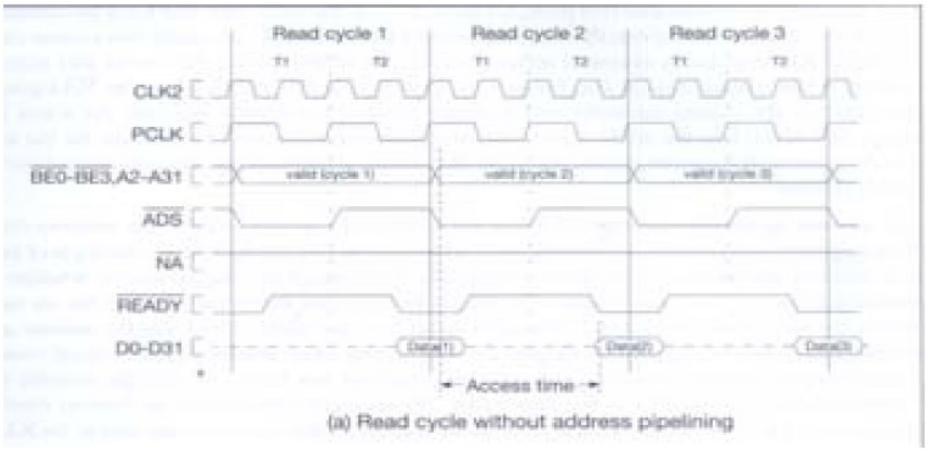
### ADD AX,[BX]

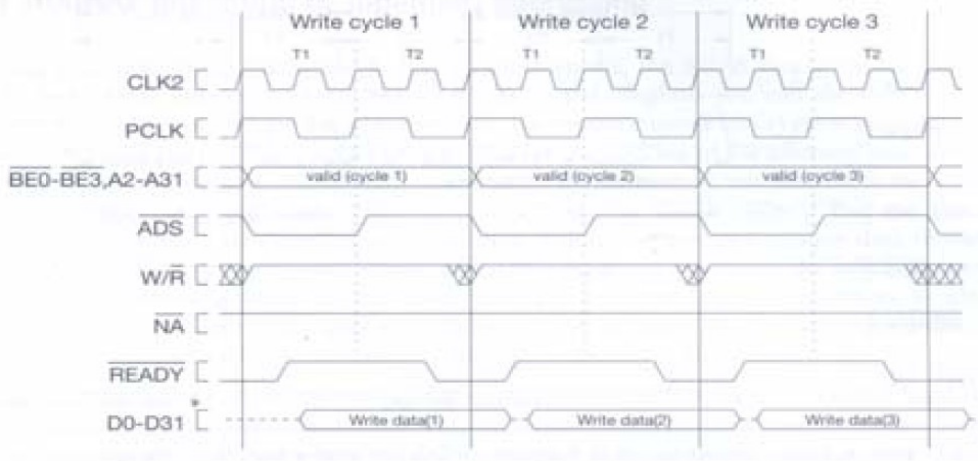
Supponendo che l'istruzione sia lunga 2 byte e che il bus dati sia ampio almeno 16 bit, occorre una transazione per il fetch dell'istruzione. Occorre anche un'altra transazione di bus per il fetch dell'operando.

Quando un dispositivo desidera ottenere il controllo del bus, HOLD viene posto a 1, dopo di che, terminato il corrente ciclo di bus, pone in alta impedenza i segnali di ABUS, e i segnali di controllo. Quando il dispositivo rilascia il bus, riporta a 0 il segnale di HOLD.
Per la gestione degli interrupt, esistono altri tre segnali, essi sono INTR, INTA, NMI.
Il segnale **INTR** è un segnale di input per il processore e determina una richiesta di interrupt da parte di un dispositivo esterno.
Il segnale **INTA**, che invece è un segnale di output, determina l'accettazione di tale richiesta e il trasferimento del codice di interrupt.
Infine, **NMI**, è un segnale di input che se attivo determina l'arrivo di una richiesta di interrupt non mascherabile.
Esiste anche un segnale che indica il fatto che un'istruzione sia atomica e quindi che non possa essere fermata in fase di esecuzione.
Questo segnale è il segnale **LOCK** e viene usato per realizzare strutture come i semafori e l'operazione di test and set, per garantire l'accesso esclusivo alle sezioni critiche.

## Pentium bus cycle

Nei pentium esistono due tipologie fondamentali di cicli di bus: il **single transfer** e il **bus cicle**.
Un ciclo di bus di tipo single transfer dura due cicli di clock (del bus), e viene usato per effettuare singoli transferimenti in lettura/scrittura.




Il tempo di accesso alla memoria deve essere inferiore o uguale al tempo di acquisizione della CPU, se la memoria è troppo lenta devono essere introdotti un determinato numero n di cicli di WAIT, affinchè sia soddisfatta la seguente relazione:

$$ta(memoria) <= ta_{cpu} + n * tw$$

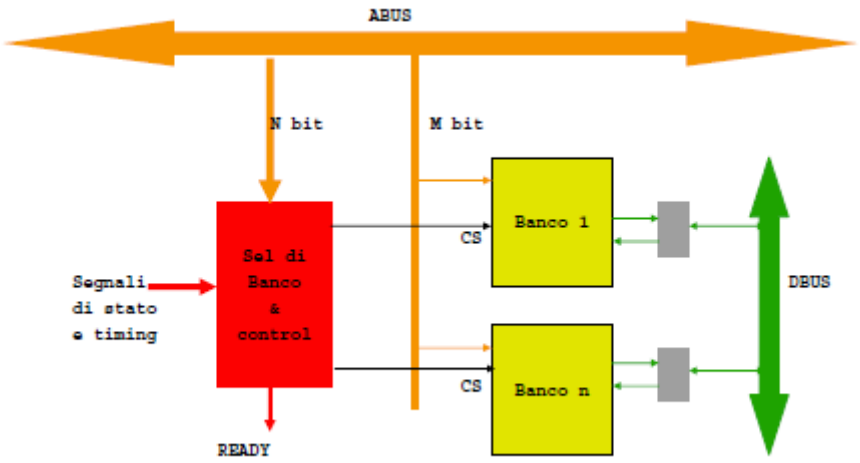
# Capitolo 4

## Introduzione al sistema di memorie

Le memorie lavorano generalmente ad una frequenza inferiore a quella dell'host bus. Questa discrasia fa in modo che le prestazioni del sistema ne risentano negativamente, pertanto sono state trovate delle soluzioni **tecnologiche** (con le memorie fast operative) e **architetturali** (con l'interleaving) che riducano il degrado prestazionale.
Due problematiche fondamentali, proprie delle memorie dinamiche, sono:

- **il refresh** della memoria, che consiste nell'aggiornamento delle informazioni contenute nei condensatori;
- **la rilevazione e/o correzione degli errori**, che consiste nell'aggiungere dei codici che permettano di rilevare o correggere gli errori presenti nei dati memorizzati.

La struttura del sistema di memoria è quella illustrata nella figura, come si può vedere la memoria è suddivisa in banchi, tutti direttamente collegati al data bus, in maniera bidirezionale, per permettere sia la scrittura che la lettura.




Una porzione dei bit dell'address bus sono utilizzati per selezionare il banco su cui si vuole effettuare l'accesso in memoria, tale selezione viene effettuato tramite opportuni segnali di banco e di controllo.
La restante porzione viene utilizzata per individuare lo spiazzamento interno al banco.

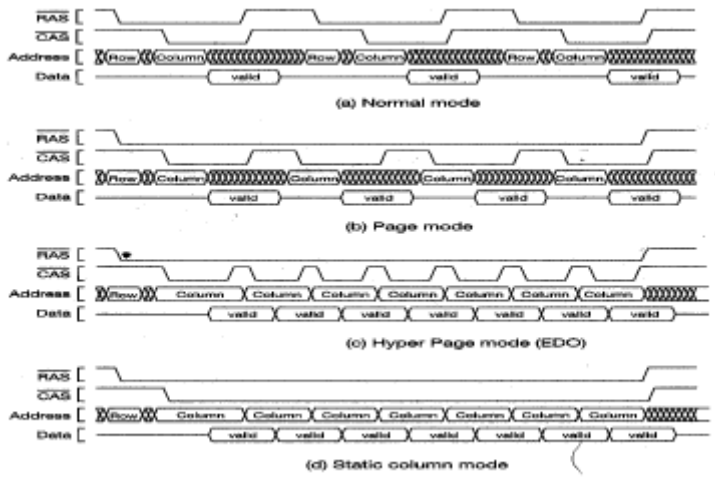
## Funzionamento delle memorie DRAM

Il funzionamento delle DRAM prevede alcuni cicli: il ciclo di **read**, relativo alla lettura di un dato; il ciclo di **write**, corrispondente ad una scrittura; il ciclo di **refresh**, necessario per evitare di perdere i dati contenuti in memoria a causa della diminuzione della carica presente all'interno dei condensatori che costituiscono le celle di memoria; e il ciclo **fast operative**, il quale viene utilizzato per ottenere tempi di accesso più piccoli a dati consecutivi presenti in memoria, permettendo l'accesso a tutte le celle associate ad una riga.
Grazie alla disposizione a matrice delle celle di memoria, il numero di linee utilizzate per l'indirizzamento delle celle di memoria è pari alla metà di quello che si avrebbe nel caso in cui queste fossero disposte in modo vettoriale: prima viene inviato sull'address bus l'indirizzo relativo alla riga, successivamente viene inviato sempre sull'address bus l'indirizzo relativo alla colonna.
I segnali **$\overline{\text{CAS}}$** e **$\overline{\text{RAS}}$**, permettono di stabilire a cosa si riferisce l'indirizzo che viene trasmesso in quell'istante, per cui vengono multiplexati sullo stesso bus gli indirizzi di riga e colonna.

# Fast operative

Il sistema fast operative permette di migliorare le prestazioni di una memoria, quando l'operazione di lettura o scrittura coinvolge alcune celle adiacenti, o in generale delle celle associate alla stessa linea.
Queste condizioni sono generalmente vere quando si sta per aggiornare una cache line del processore.
Una memoria è fast operative se i chip sono progettati secondo la tecnologia adatta, che permette di attivare congiuntamente tutti i condensatori disposti su una linea.
Quindi questa soluzione dipende strettamente dalla tecnologia e non da come la memoria viene usata.
Le memorie fast operative prevedono che l'indirizzo della riga selezionata rimanga fisso, pertanto la circuiteria di selezione della riga non va modificata col procedere delle letture.
Il multiplexer di selezione della colonna invece varia rapidamente, in modo da selezionare tutte le posizioni della riga considerata.
In altre parole il segnale $\overline{RAS}$ resta fisso, mentre varia il segnale $\overline{CAS}$.
Così procedendo, solamente il primo accesso impiega tras per completarsi, mentre i successivi impiegano un tempo sensibilmente inferiore, in quanto non è necessario di volta in volta specificare l'indirizzo di colonna.
Questo sistema offre miglioramenti significativi solo se i dati sono disposti sequenzialmente in memoria, tuttavia, dato che la maggior parte delle letture in memoria sono fatte per aggiornare la cache, la maggior parte dei dati a cui si accede, sono proprio disposti sequenzialmente.
Esistono tre tipi di fast operative mode:

- sincrono;
- asincrono;
- protocol based.




Adesso descriviamo gli esempi mostrati in figura. Il primo caso (b) mostra come il $\overline{RAS}$ rimanga fisso e il $\overline{CAS}$ vari sistematicamente, il caso (c) mostra come si possano usare sia i segnali di salita che quelli di discesa per il segnale $\overline{CAS}$, infine, il caso (d) mostra come la scansione delle colonne possa avvenire tramite un clock interno.

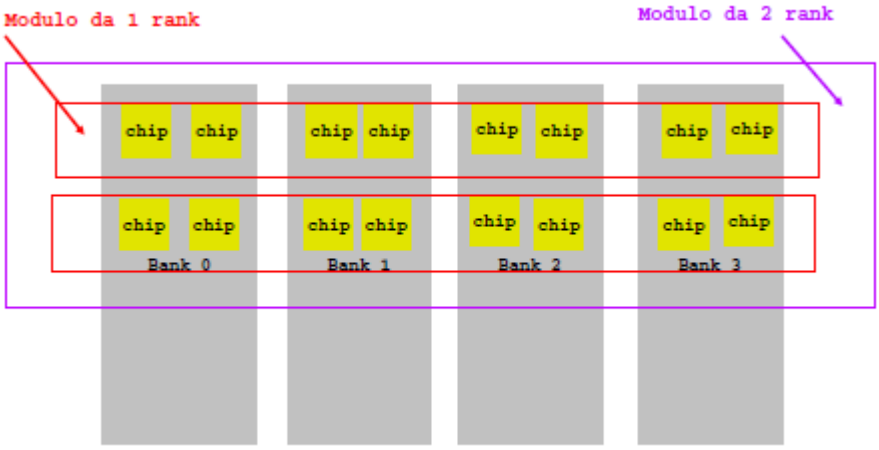
# La famiglia delle DRAM

Vi sono stati due salti generazionali nell'evoluzione delle memorie RAM, prima vi è stato quello relativo ai meccanismi di interleaving e di fast operative (relativi alle memorie **EDO** e **BEDO RAM**), quindi il salto generazionale successivo si è avuto con le **SDRAM**.
Le SDRAM ottengono migliori prestazioni tramite **un memory controller più efficiente (burst control)**, e **la sincronizzazione tra il clock del bus e quello della CPU**.

In generale i compilatori allineano i dati agli indirizzi di memoria più convenienti, per minimizzare i tempi di lettura, quindi multipli del parallelismo sul bus dati. Esistono delle parole chiave nei compilatori, per forzare gli allineamenti a indirizzo multipli di 8 o 16 byte.

## Organizzazione fisica

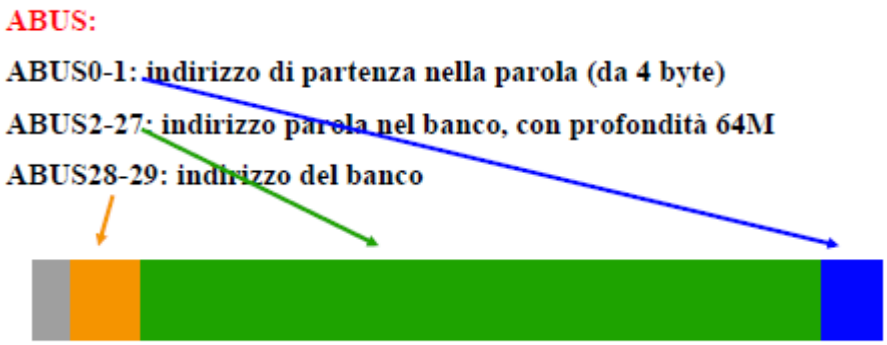
Introduciamo dei termini che descrivono le componenti fisiche che fanno parte di un'organizzazione fisica di una memoria.




Un **banco** è un blocco fisico logico con parallelismo di un byte, tale byte viene selezionato tramite i segnali BEi (bank enable).
Un **device** è un singolo chip caratterizzato da uno spazio di indirizzamento e da un parallelismo dati.
Un **modulo** è un insieme di device con parallelismo pari al data bus, realizzato su un'unica piastrina.
Un **rank** è una porzione di modulo che ha un parallelismo pari al data bus e uno spazio di indirizzamento con profondità pari a quella dei device.

## Progetto di una memoria

Nella figura seguente descriviamo un esempio di un modulo di memoria da 1GB.
Consideriamo di avere a disposizione un data bus e un address bus da 32 bit, dei device da 64MB, di non considerare la rilevazione e la correzione degli errori.
Lo spazio di indirizzamento necessario è di $2^{30}$ byte, per cui basta utilizzare solamente 30 dei 32 bit presenti nell'address bus.

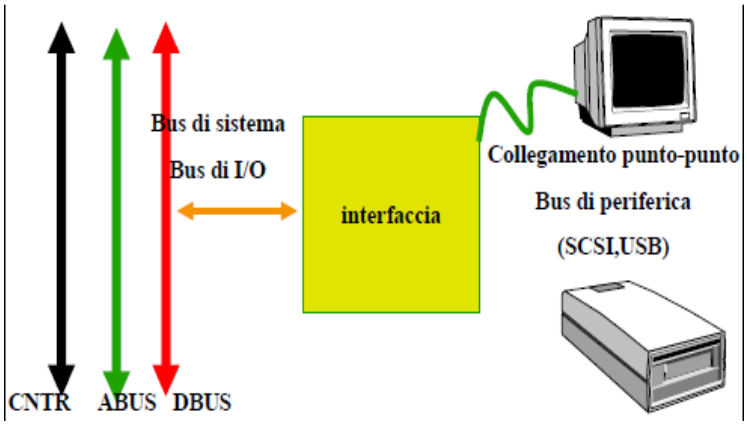



I primi due bit dell'address bus, codificano i 4 segnali di BE, i successivi 26 bit permettono di individuare il byte all'interno dei 64 MB del device, e infine gli ultimi due bit individuano l'indirizzo del banco, permettendo di selezionarne uno tra i 4 disponibili.
I segnali BEi vengono inviati ai 4 rank, a ciascun rank sono associati i device che restituiscono l'(i+1)-esimo byte meno significativo sul data bus.
Le due linee più significative dell'address bus vengono utilizzate per la selezione del

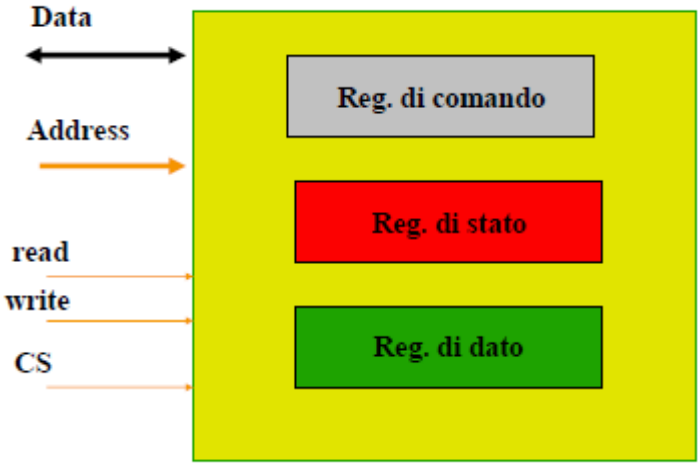
# Capitolo 5

## Il sottosistema di I/O

I dispositivi periferici vengono connessi al microprocessore ad un determinato livello di bus, a tal proposito è importante che il bus sia coerente con le caratteristiche del dispositivo, ad esempio se un dispositivo usa meccanismi di comunicazione basati su interrupt o DMA, allora il bus deve supportare tali caratteristiche.
Ad esempio all'host bus non è possibile collegare periferici, innanzitutto perchè non gestisce gli interrupt.
Per individuare i dispositivi periferici all'interno del sistema, viene assegnato un indirizzo ad ogni periferica, la totalità di tali indirizzi determina lo spazio di I/O.

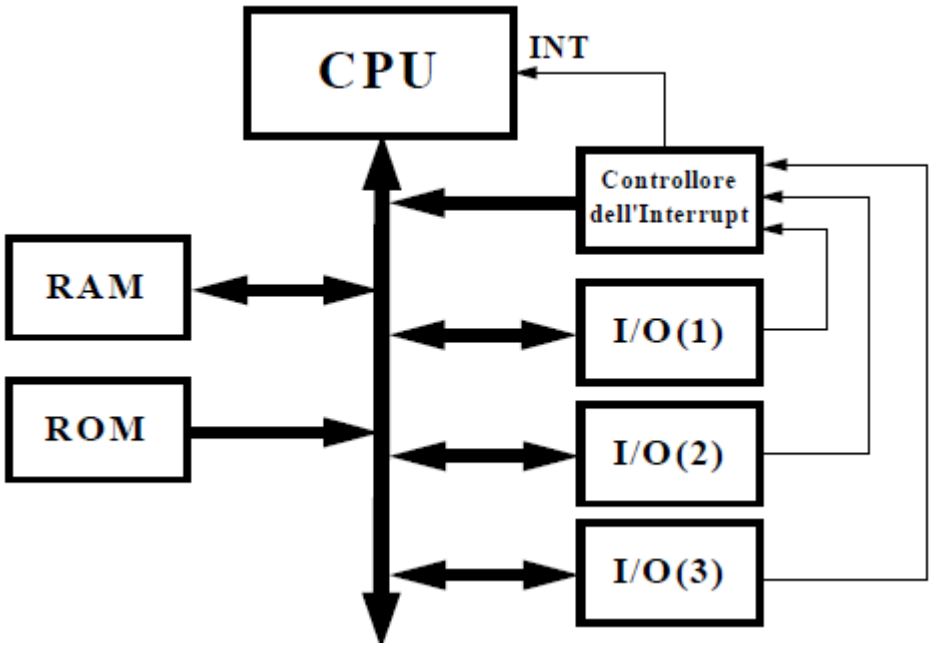



Ogni dispositivo si interfaccia col bus tramite un'interfaccia, che possiede internamente un determinato numero di registri, ciascuno dei quali è accessibile tramite un differente indirizzo.




Esistono almeno tre registri nei dispositivi odierni: un registro di comando, per inserire i comandi da fare eseguire al periferico, un registro di stato, che descrive lo stato del periferico, e un registro dato che permette lo scambio di dati tra il periferico e il sistema, e viceversa.
Nella figura seguente possiamo notare come parte dei bit dell'address bus vengano direttamente collegati al dispositivo per la selezione del registro interno, mentre gli altri vengono inviati ad un decoder che poi effettua la selezione dell'interfaccia.

Il polling è il sistema di gestione delle richieste di servizio più semplice, la CPU interroga ciclicamente i dispositivi tramite un ciclo software, nel quale viene verificato se i dispositivi hanno dei trasferimenti da effettuare: se un dispositivo ha dei dati in uscita viene servito.
La maggior parte del tempo viene impiegata per la gestione del ciclo di polling, che ha il merito di risultare **molto semplice da implementare**.
Di contro, ogni dispositivo deve attendere il suo turno prima di vedere soddisfatta la sua richiesta, e ciò genera un **elevato tempo di latenza**.
Nel sistema di gestione basato su interrupt, la CPU non interroga i dispositivi di I/O, ma è il dispositivo stesso che quando ha bisogno di essere servito attiva il segnale di interrupt: pertanto deve essere presente nell'interfaccia un segnale di dato pronto/dato trasmesso.
La richiesta verrà soddisfatta dalla CPU, al termine dell'esecuzione dell'istruzione corrente.
La gestione dell'interrupt inizia eseguendo una routine di gestione specifica per il tipo di interrupt, chiamata **interrupt handler**.
Il circuito che gestisce le richieste di interrupt viene chiamato **interrupt controller**.




Anche quando un dispositivo è gestito con interrupt il tempo di latenza e il throughput rivestono un ruolo importante per le sue prestazioni.
Nell'analizzare gli interrupt si utilizza spesso il concetto di tempo di latenza (definito come tempo fra la richiesta dell'interrupt generato dall'interfaccia, e l'esecuzione dell'operazione di I/O).
Esso è definito come **Tlat = Ti + Tc + Tr**, dove:

- **Ti, tempo necessario a completare l'istruzione corrente**, gli interrupt non sono sincronizzati con il clock della CPU e quindi possono arrivare in un qualsiasi momento, se si analizza la situazione considerando il caso peggiore bisogna considerare questo tempo pari a quello di esecuzione di un'istruzione.
- **Tc, tempo di attivazione della routine gi gestione dell'interrupt**, include il salvataggio dello stato corrente e l'aggiornamento del program counter alla locazione della procedura di gestione dell'interrupt. Queste operazioni richiedono l'accesso alla memoria e quindi aumentano il tempo di latenza.
- **Tr, tempo fra l'inizio della routine e l'istruzione di trasferimento dati**. Non è necessario che la procedura di gestione dell'interrupt legga subito il dato dall'interfaccia della periferica ma potrebbe prima eseguire una mangiata di istruzioni di inizializzazione o di preparazione per la lettura del dato.

Il **transfer rate**, o TR, è la frequenza con cui arrivano i blocchi di dati provenienti dalla

Sebbene questa tabella sia differente tra la modalità di funzionamento real e protected della CPU, i concetti alla base del suo funzionamento sono gli stessi. L'IVT contiene 256 elementi che rappresentano i 256 diversi interrupt gestibili dal processore 8086, ogni elemento occupa 32 bit (4 byte), 16 (2 byte) per il registro di segmento e 16 (2 byte) per l'indirizzo a cui saltare nel segmento. In modalità reale l'IVT occupa lo spazio di memoria compreso tra gli indirizzi 0x0000 e 0x03FE.
La posizione dei puntatori associati ai vari interrupt nella IVT, ne indica la priorità.
L'indirizzo del descrittore associato all'interrupt n-esimo si ottiene semplicemente moltiplicando n per 4, cioè per il numero di byte occupati da un singolo descrittore.
All'avvio del PC il BIOS provvede subito a caricare i puntatori nell'IVT alle principali e fondamentali procedure di gestione degli interrupt (che si trovano per ora nella ROM del BIOS).
Una volta caricate le funzioni di gestione degli interrupt più importanti, il bios provvede a inserire ad alcuni indici specifici i puntatori alle funzioni offerte (come la gestione del disco e dello schermo).
Quando il bios ha terminato le attività di inizializzazione procede al caricamento del loader del sistema operativo (letto dal boot sector del disco) così da potergli passare il controllo.
Visto che spesso l'implementazioni delle varie funzioni, offerte dal bios, non sono sufficientemente elaborate, sarà il loader a sostituirle con delle routine di gestione interne al sistema operativo stesso.
La caratteristica più importante della IVT risulta essere la **flessibilità**, in quanto è possibile cambiare le routine di gestione delle varie funzionalità senza ricompilare i programmi che le utilizzano.

## Abilitazione e disabilitazione degli interrupt

Il flag IF permette di abilitare e disabilitare gli interrupt esterni, tale bit può essere settato a 1 tramite l'istruzione STI, e può essere impostato a 0 tramite l'istruzione CLI.
Quando viene invocata una routine di gestione dell'interrupt, il flag IF viene automaticamente resettato per impedire che altri interrupt mascherabili possano interrompere la routine di gestione in esecuzione.

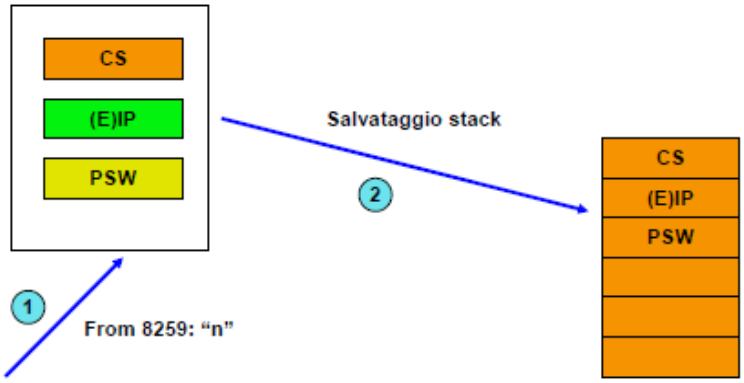
## Interrupt per il debugging

L'operazione di debug viene ottenuta attraverso l'utilizzo di interrupt sia software che hardware.
Nei microprocessori recenti, in cui viene gestita la memoria virtuale, la gestione del debug risulta essere difficoltosa e richiede la presenza di hardware apposito, di seguito viene analizzato solo il caso di un sistema in real mode.
Le funzioni principali di debug risultano essere:

- **inserimento di breakpoint**: durante l'esecuzione del programma da debuggare, nel caso venga raggiunta una ben definita istruzione, il controllo passa al debugger;
- **esecuzione step by step**: il controllo del programma passa istruzione per istruzione al debugger.

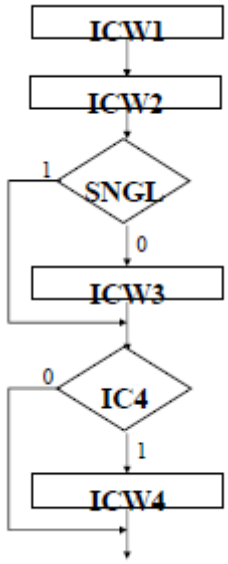
La funzione di step by step è attivata tramite il flag TF: l'unità di controllo del processore, al termine di ogni istruzione ne controlla lo stato, nel caso in cui sia attivato viene generata una trap (corrispondente ad un INT 4); il debugger deve aver precedentemente posizionato nella IVT, alla posizione 4, una routine del debugger stesso per gestire tale modalità, inoltre lo stesso deve anche farsi carico di salvare l'indirizzo delll'istruzione sospesa, così che sia possibile riprendere l'esecuzione da tale riga di codice.
Per quanto riguarda la realizzazione della funzione di breakpoint, si potrebbe inserire all'interno del debugger una tabella contenente tutti gli indirizzi di breakpoint. Mentre il programma viene eseguito, il debugger dovrebbe confrontare il valore del program counter

accedere alla routine di gestione dell'interruzione associata al dispositivo, il cui descrittore sarà presente alla posizione N*4 della IVT.
Nel momento in cui la CPU riceve il valore N, viene salvato nello stack il valore del registro dei flag PSW, e dell'indirizzo di ritorno (cioè il valore registri CS e IP).




Vengono azzerati i flag IF e TF per disabilitare gli interrupt hardware esterni ed il trap mode, dopo di che si accede all'elemento (4*N)-esimo della IVT e viene attivata la procedura di gestione dell'interrupt corrispondente.

L'ordine delle ICWs è fisso e normalmente vengono inviate una volta sola in fase di inizializzazione, mentre le OCWs possono essere inviate singolarmente in qualunque fase del programma.
La sequenza di inizializzazione viene riconosciuta dal dispositivo in quanto, in corrispondenza della prima parola ICW1 si ha che A0 è uguale a 0 e D4 è uguale a 1.
Quando viene ricevuta una sequenza di inizializzazione si azzera il registro IMR per il mascheramento degli interrupt.
La seguente figura riporta l'ordine della sequenza delle parole di inizializzazione.




La parola **ICW1** configura tre aspetti fondamentali:
- i segnali IR0-7 sono sensibili ai fronti o ai livelli (**LTIM level triggered mode**);
- se il dispositivo funziona singolarmente o in cascata con altri 8259 (**single mode o cascade mode**);
- se oltre alle tre parole di inizializzazione ne invio una quarta (**IC4**).

| A0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| 0 | X | X | X | 1 | LTIM | X | SNGL | IC4 |

La parola **ICW2** stabilisce l'indirizzo di partenza nella IVT delle routine di gestione degli interrupt associati al dispositivo (nell'8086 è 08H l'indirizzo associato al primo dispositivo, 40H quello associato al secondo).

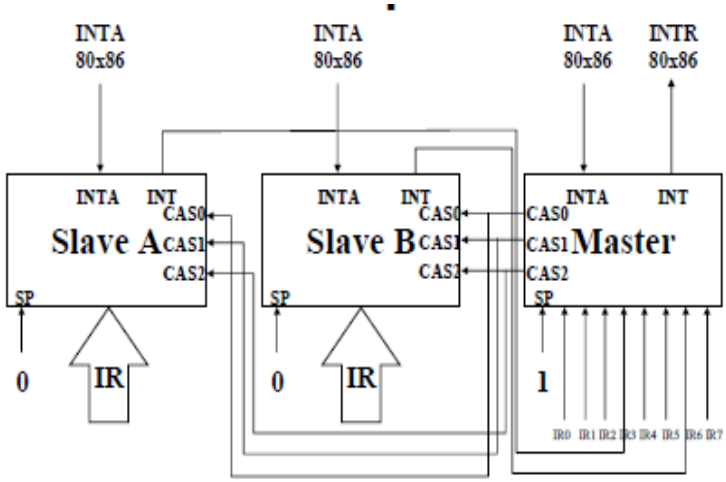
| A0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| 1 | T7 | T6 | T5 | T4 | T3 | X | X | X |

La terza parola **ICW3** stabilisce i parametri di connessione tra master e slave, se il dispositivo è un master, allora contiene 8 bit S0-7, che indicano se a ciascuna delle 8 linee IR0-7 è connesso un altro 8259 o un dispositivo periferico; se il dispositivo è uno slave la parola contiene il valore ID della linea a cui è connesso l'8259.

| | A0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|
| **Dispositivo Master** | 1 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

| | A0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|
| **Dispositivo Slave** | 1 | 0 | 0 | 0 | 0 | 0 | ID2 | ID1 | ID0 |

Se la richiesta non proviene da uno slave, il master invia sul data bus l'indice associato alla routine di gestione dell'interrupt, leggendolo da ICW2, tramite cui è possibile effettuare l'accesso alla entry corretta nella IVT.
Se la richiesta arriva da uno slave, il master piazza sui pin CAS, il valore del livello di interruzione IR a cui è connesso lo slave; a questo punto tutti gli 8259 ricevono il segnale INTA, ma solo quello il cui ID è uguale al valore contenuto nei segnali CAS, risponderà a tale segnale
Il dispositivo selezionato setta il bit opportuno di ISR, pulisce il bit relativo di IRR e invia l'indice associato alla routine di gestione dell'interrupt, leggendolo dal suo ICW2.
Se vi sono più 8259 coinvolti nella comunicazione, bisogna inviare ad entrambi i segnali di EOI per la chiusura.

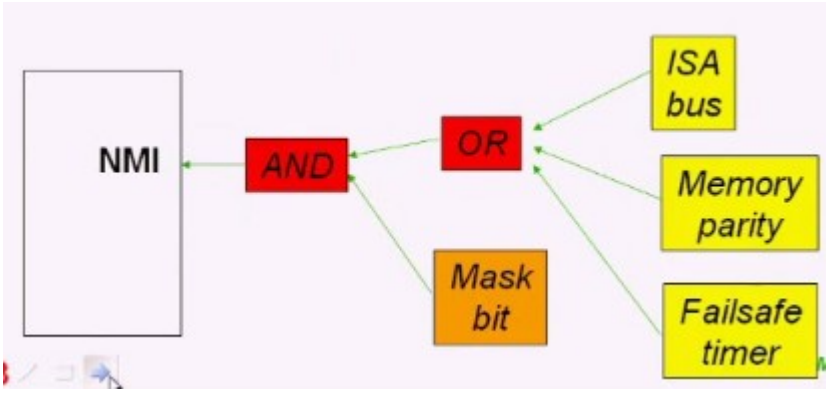



## Interruzioni nei PC

Nei personal computer, le interruzioni sono gestite da due dispositivi 8259A (o compatibili inseriti nel chipset), collegati in modalità master-slave.
Il PC è in grado di gestire **15 tipi di interruzione mascherabili**:

- **7 sul master** (l'ottava connessione viene usata per gestire la comunicazione con lo slave);
- **8 sullo slave**.

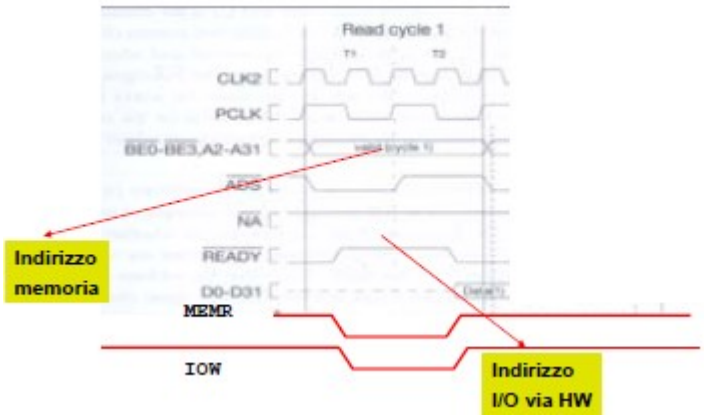
Inoltre vi è una fonte ulteriore di interruzioni, che sono **non mascherabili,** collegata alla logica di rilevamento degli errori di parità della memoria dinamica, ad timer-counter programmabile (una porta dell'8253), e ad un segnale del bus ISA.




Vediamo dallo schema semplificato in figura, che **in alcune fasi è necessario mascherare gli interrupt non mascherabili**, ad esempio nella fase di test della memoria, e a tal fine esistono degli opportuni mask bit impostabili dall'esterno.

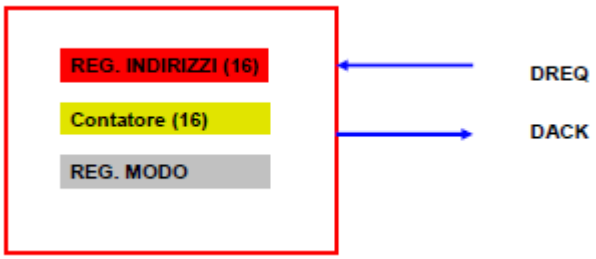
CS e IP.
**Vanno salvati sullo stack e ripristinati alla fine tutti i registri** a cui si fa accesso durante l'esecuzione della routine.
**Non bisogna abusare dello stack**, in quanto la ISR usa lo stesso stack del programma chiamante: se è necessario salvare più informazioni sullo stack, bisogna dichiarare un nuovo stack segment.
**Non si deve richiamare codice BIOS o DOS**, in quanto questii tipi di codice non sono rientranti.
**Se una ISR ha un tempo di esecuzione troppo elevato dovrebbe decidere di lasciarsi interrompere da tutti gli interrupt con priorità più elevata**, ciò può essere realizzato semplicemente con un'istruzione STI.
**La comunicazione col programma chiamante può avvenire tramite variabili globali**, ma bisogna occuparsi di aggiornare opportunamente il registro DS per accedere a tali variabili, in quanto la ISR si trova su un segmento diverso.
Infine, **la ISR si occuperà della comunicazione col dispositivo** associato tramite opportune operazioni che verranno rese necessarie per gestire il dispositivo stesso.

Mentre il trasferimento basato su **interrupt è basato sul concetto di istruzione** (inseriamo le istruzioni della ISR dopo l'esecuzione dell'istruzione corrente), qua questo concetto non esiste, **il DMA possiamo dire sia basato sul concetto di ciclo di bus**, cioè vengono richiesti dei cicli di bus al termine di quello corrente.
L'istruzione, essendo associata a più cicli di bus, non viene terminata ma viene, in generale, sospesa.

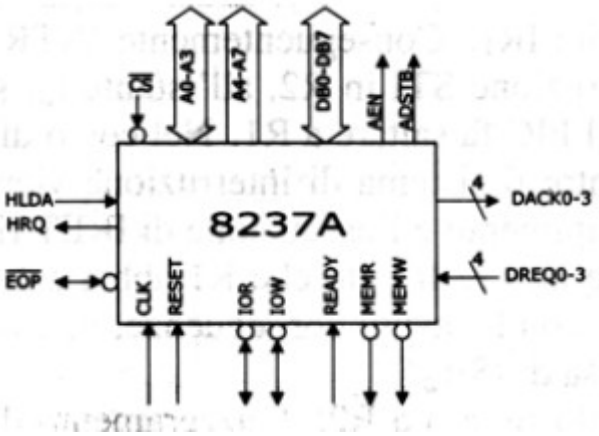



In un ciclo di bus DMA, devono essere introdotti due meccanismi diversi per indirizzare il dispositivo di I/O che verrà attivato tramite il suo chip select, e la locazione di memoria su cui si vuole effettuare il trasferimento.
A tal proposito, **la memoria viene selezionata mediante ABUS e MEMR/MEMW**, mentre **il periferico viene selezionato mediante un altro circuito hardware e da IOR/IOW**.
In tal modo si passa da due cicli di memoria (memoria-cpu,cpu- I/O), ad un unico ciclo.
Un canale di comunicazione DMA deve essere formato almeno da:

- **un registro indirizzi**, che contiene l'indirizzo iniziale da cui parte il trasferimento;
- **un registro contatore**, che indica il numero di bytes (N-1) da trasferire (fino ad un massimo di 64KB);
- **un registro modo**, che stabilisce le proprietà e le modalità del trasferimento (single o block mode, ciclo di lettura o scrittura).




Nell'8086, il DMA viene implementato mediante un dispositivo che viene chiamato 8237, inizialmente tale dispositivo era un chip singolo, attualmente viene integrato nel chipset.

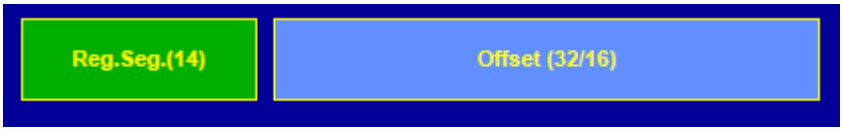



Il dispositivo ha 4 canali DMA, gestiti con priorità fissa (configurazione di default) o rotante,

# Capitolo 8

## Gestione modo protetto nell'architettura x86

In modo reale gli indirizzi, determinati a livello di istruzioni, indirizzano direttamente la memoria fisica, seppur mediante un modello di segmentazione, tramite 20 bit, ottenendo uno spazio di indirizzamento di 1MB.
Nel seguito descriviamo il modello x86-32, con registri di offset da 32 e 16 bit, nonostante le ultime generazioni di processori implementino anche un modello a 64 bit.
In modo protetto vengono introdotti i concetti di **protezione della memoria** e di **paginazione**, gli indirizzi di memoria logica sono lunghi 46 bit, permettendo di indirizzare 64TB di memoria complessivamente.
La memoria logica è lo spazio di memoria visibile direttamente dai registri dell'architettura, secondo cui la memoria è scomposta in $2^{14}$ segmenti, ciascuno formato da un massimo di $2^{32}$ byte =4GB di memoria.
A livello di memoria logica, gli indirizzi sono formati nel seguente modo:




I 14 bit del registro di segmento, sono utilizzati per effettuare l'accesso alla tabella dei descrittori di segmento, che permette di accedere a varie informazioni riguardanti i segmenti.
Esistono due tabelle, la **GDT** (**Global descriptor table**) e la **LDT** (**Local descriptor table**), la prima contiene i descrittori dei segmenti a cui tutti i processi possono accedere, mentre la seconda contiene i descrittori dei segmenti accessibili solo localmente da determinati processi; esse contengono 8k di descrittori ciascuna, ogni descrittore è formato da 8 bytes.
I descrittori di segmento contengono le seguenti informazioni:

- **indirizzo di base (32 bit)**, che corrisponde all'indirizzo di testa del segmento nella memoria lineare;
- **limite (20 bit)**, che indica il numero totale di bytes occupati dal segmento;
- **attributi (10 bit)**;
- **privilegi (2 bit)**.

Gli attributi contenuti nel descrittore sono i seguenti:

- **presenza**, indica se il segmento è presente in memoria lineare o meno;
- **system**, indica se il segmento è di dati/codice, o è impiegato per memorizzare una struttura di sistema;
- **execute**, indica se il segmento è di codice o di dato;
- **R/W**, indica se il segmento può essere letto o scritto;
- **accessed**, indica se il segmento è stato letto o scritto.

I livelli di privilegio sono 4 (da 0 a 3) e a valori numericamente più bassi corrispondono valori più alti di privilegio.
Esistono due regole di cui bisogna tenere conto relativamente ai privilegi: per quanto riguarda la **qualità dei dati**, un segmento dato può accedere ad un altro segmento dato con privilegio minore o uguale; per quanto riguarda **l'affidabilità del codice**, un segmento codice può accedere ad un altro segmento codice con livello di privilegio maggiore o uguale.
Il primo passaggio nella conversione dell'indirizzo da logico a fisico, ci permette di ottenere un indirizzo lineare, nella modalità specificata in figura, cioè aggiungendo all'indirizzo di

Si hanno un totale di 1024 tabelle di secondo livello, ciascuna di esse occupa 4 * 1024 byte, per cui complessivamente l'occupazione di memoria è data da 4 * 1024 * 1024 byte = 4MB.
Le conversioni degli indirizzi da logici a fisici, devono essere effettuate con dei tempi compatibili a quelli della pipeline, pertanto i descrittori dei segmenti attivi e gli elementi delle tabelle delle pagine più frequentemente usati, vanno memorizzati in delle strutture con una alta velocità di accesso.
La struttura implementata viene chiamata **TLB (Traslation Lookaside Buffer)**, e viene gestita direttamente tramite una cache on chip.

## Confronto tra segmentazione e paginazione

Una prima differenza tra segmentazione e paginazione è rappresentata dal fatto che **la segmentazione è visibile al compilatore e al programmatore**, mentre la paginazione non lo è.
Nella memoria segmentata, le informazioni che sono correlate vengono inserite nello stesso segmento, facendo in modo, ad esempio, che vengano distinti i segmenti dati dai segmenti codice.
Inoltre, è possibile fornite diversi attributi ad ogni segmento, quali ad esempio il livello di privilegio necessario per accedervi.
**La segmentazione semplifica anche la gestione della grandezza delle porzioni di memoria assegnate ad un processo**, infatti sarà il sistema operativo ad allargare o ridurre la dimensione dei segmenti assegnati, qualora il processo abbia bisogno di più o di meno memoria.
Grazie alla segmentazione, non è necessario ricompilare e rilinkare tutti i programmi di un calcolatore in seguito allo spostamento delle informazioni in memoria, in quanto i programmi faranno riferimento al segmento relativo e non accederanno alla memoria in maniera assoluta.
Infine, **la segmentazione favorisce la protezione e la condivisione dei segmenti**, in quanto un singolo segmento può essere acceduto da più processi e ha un determinato livello di privilegio necessario per accedervi.

## Passaggio al modo protetto

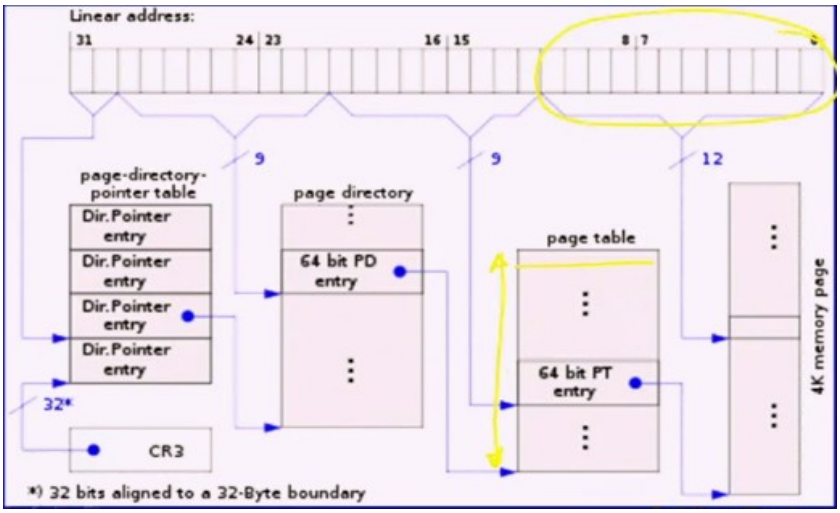
Il bios, dopo aver effettuato il bootstrap, termina l'uso della modalità reale, e il loader del sistema operativo (windows o linux) effettua il passaggio al modo protetto.
Intanto è necessario inizializzare la GDT, con almeno tre segmenti: uno per i dati, uno per lo stack e uno per il codice.
Successivamente deve essere riinizializzata la IVT per configurare le routine di gestione delle interruzioni scelte dal sistema operativo, ogny entry della IVT in modo protetto occuperà il descrittore del segmento a cui saltare, che è formato da 8 byte, mentre in modo reale ogny entry è formata solamente da 4 byte (indirizzo segmento + offset).
A questo punto devono essere inizializzati i registri IDTR e GDTR che contengono gli indirizzi iniziali di tali tabelle.
Viene impostato il bit PE del registro CR0, e viene eseguita una JMP FAR per svuotare la coda delle istruzioni, in quanto il calcolo degli indirizzi nelle istruzioni della pipeline era stato calcolato in modo reale.
A questo punto si possono inizializzare i registri di segmento e se si vuole abilitare la paginazione è necessario settare un bit opportuno del registro CR0.

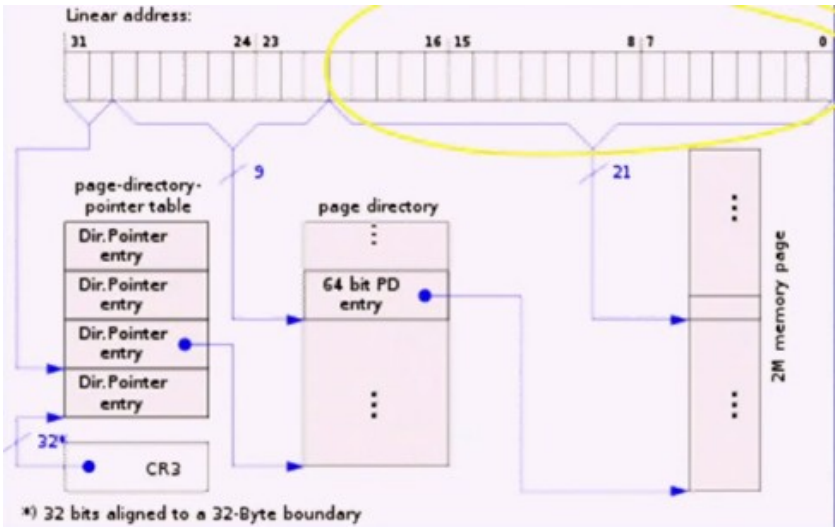
## O.S. moderni: considerazioni

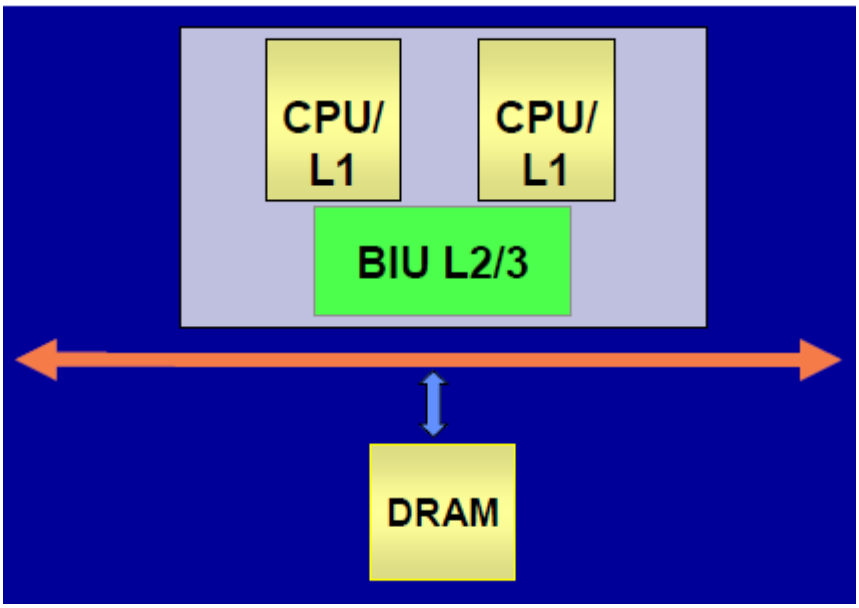
Nei sistemi operativi odierni, la segmentazione è in un certo modo disabilitata.
Al momento dell’attivazione dei segmenti, si dichiarano segmenti di grandi dimensioni (4

# Evoluzione architettura pentium

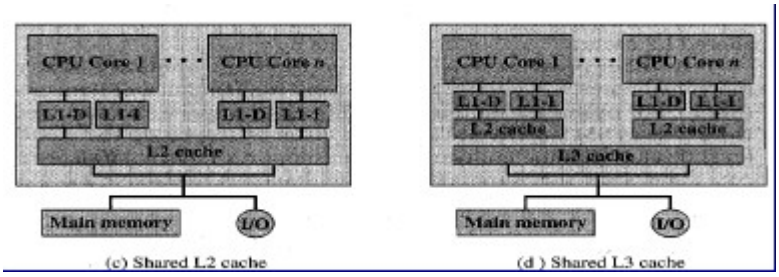
Uno dei principali concetti su cui si stanno sviluppando le architetture pentium, è quello dell'introduzione di tecniche che gestiscano efficientemente il **parallelismo tra i thread**, sfruttando la presenza di più core.
Un'altra notevole novità è stata l'introduzione, nel Pentium 2, di un ABUS a 36 bit **(PAE Physical Address Extension)**, mantenendo sempre il DBUS a 64 bit.
L'ABUS a 36 bit permette di indirizzare 64GB di memoria fisica, tuttavia gli indirizzi rimangono comunque a 32 bit, per cui per rendere effettivo l'utilizzo dei 36 bit, bisogna introdurre un sistema diverso della gestione delle pagine.
I primi due bit dell'indirizzo lineare individuano una entry della nuova tabella introdotta, la **page directory pointer table**, che contiene 4 entry diverse a 4 diverse **page directory**, e il cui indirizzo iniziale è contenuto nel registro CR3.
I nove bit successivi dell'indirizzo lineare, individuano una entry della page directory precedentemente selezionata.
Tale entry sarà formata da 64 bit, e l'intera page directory conterrà $2^9$ = 512 entry, occupando la stessa memoria che occupava la tabella corrispondente in x86-32.
L'entry della tabella individua l'indirizzo della **page table** selezionata, che avrà la stessa dimensione della page directory, e permetterà di ottenere l'indirizzo di 24 bit della pagina selezionata.
Concatenando i 24 bit ottenuti ai 12 bit dell'offset si otterrà l'indirizzo della pagina a cui fa riferimento l'indirizzo lineare, considerando che le pagine abbiano dimensione pari a 4KB.






Il modello della figura precedente è detto **hyper threading**, e prevede l'uso di almeno una cache comune ai due processori e due cache L1 parcellizzate.
In questo modello la BIU è condivisa dalle due CPU.




In generale, i modelli architetturali utilizzati in contesti SMP sono quelli in figura, in cui la cache più esterna è condivisa.
Il primo dei due modelli viene implementato nel core duo, mentre il secondo viene implementato nel core i7.

# Capitolo 9

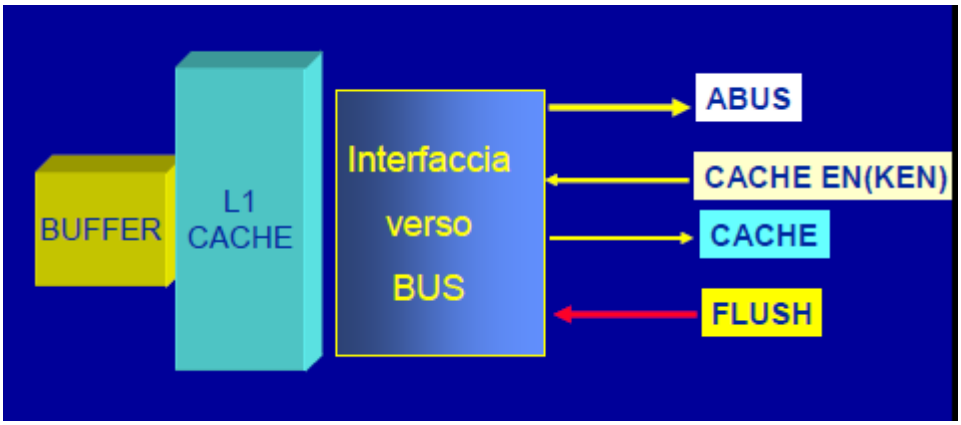
## Architetture SMP

Le architetture SMP (**symmetrical multiprocessing**) prevedono l'esistenza di due o più identici processori che condividono la stessa memoria e sono gestiti dalla stessa istanza di sistema operativo.
Esistono tre principali problematiche che bisogna gestire in architetture del genere:

- gestione dell'**allocazione del bus** tra i vari processori;
- selezione del processore a cui inviare i vari **interrupt**;
- gestione della **coerenza delle cache**.

## Coerenza delle cache

Analizziamo i segnali necessari per la gestione della cache in un sistema monoprocessore.



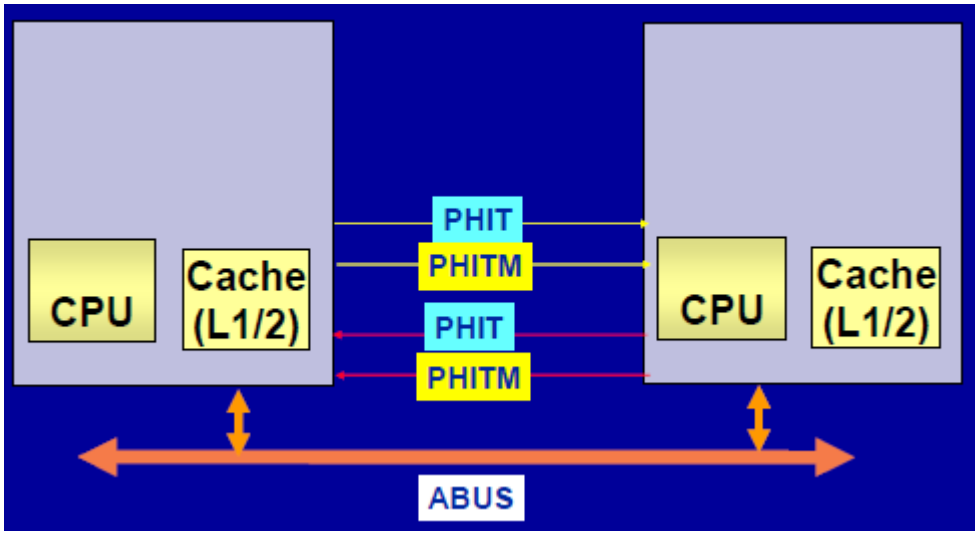

Il segnale **CACHE** viene generato internamente a seconda della configurazione del processore (vi sono due bit in CR0, oppure si può considerare la configurazione a livello di pagina), e definisce se l'area di memoria è cachable, facendo in modo che venga stabilito se il ciclo di bus relativo ad una lettura deve essere un normale ciclo di lettura o un ciclo di burst per l'aggiornamento della cache line.
Il segnale di cache enable (**KEN**) viene generato esternamente dal controllore della memoria per stabilire se il blocco di memoria è cachable o meno (ad esempio non è cachable la memoria video).
Il segnale di **FLUSH** viene usato semplicemente per effettuare la cancellazione completa dei dati in cache, se è attiva la politica di write back, vengono aggiornati i dati in memoria.
Quando un dato viene scritto in cache, si possono adottare tre politiche:

- **write through**;
- **write back**;
- **write allocate**.

La politica di write through consiste nell'effettuare le scritture sia in cache che in memoria, garantisce pertanto, in maniera semplice, che la coerenza della cache sia mantenuta.
Lo svantaggio principale consiste nel fatto che ad ogni scrittura bisogna scrivere il dato anche in memoria.
La politica di write back consiste nell'effettuare le scritture solo sulla cache, la copia dei dati in memoria viene effettuata solamente:

- **quando si ha un cache miss**, e una cache line non allineata alla memoria va sostituita;
- **quando viene eseguita** un'istruzione software di nome **WBINVD**;

Il singolo processore che avvia lo snooping, invia all'altro processore (quello che detiene il bus) un segnale di PHIT, e uno di PHITM; questi segnali comunicano al processore che sta eseguendo l'istruzione se il dato a cui si sta accedendo è presente nella cache dell'altro processore (PHIT), e se è presente e non allineato con la sua cache (PHITM).
Il processore che detiene il bus, legge i segnali PHIT e PHITM, e se entrambi sono alti, sospende il suo ciclo di bus e lo prosegue dopo che l'altro processore abbia effettuato l'allineamento della cache.




Quando all'interno di una cache devono essere introdotti dei dati, allora una delle cache line presenti deve essere sostituita.
Esistono varie politiche in base a cui scegliere tale cache line, tra cui:

- LRU;
- pseudo LRU;
- casuale.

## Protocollo Mesi

E' il protocollo utilizzato per la coerenza delle cache tra i processori.
Il protocollo specifica le azioni che ogni controllore della cache deve effettuare nel momento in cui:

- un'operazione viene effettuata su una variabile locale;
- un'operazione viene effettuata da un altro processore su una variabile che è anche nella cache locale;
- un'interrogazione su una variabile presente nella cache locale viene effettuata da un altro processore.

Per valutare la coerenza di una cache line, il protocollo mesi usa dei meccanismi di passaggio di stato associati ad ogni cache line.
Ciascuna cache line può trovarsi nei seguenti stati:

- **Modified**, il dato presente nella cache è stato modificato, pertanto ha un valore diverso rispetto a quello contenuto nella memoria centrale ed in futuro dovrà essere effettuato l'aggiornamento;
- **Exclusive**, il dato presente nella cache è coerente con la memoria ed è presente solo nella cache corrente;
- **Shared**, indica che è il dato è presente anche in altre cache, ed è coerente con la memoria;
- **Invalid**, indica che la cache line non è valida.