

Appunti del corso

# **SISTEMI A MICROPROCESSORE**

Prof. Mezzalama M.

2008/2009

# Introduzione ai Microprocessori

Un microprocessore è un'implementazione fisica della CPU in un singolo circuito integrato, chiamato Die.

Esso è utilizzato, oltre che in quasi tutti i computer, in molti dispositivi digitali quali ad esempio telefoni cellulari e stampanti.

Esistono 3 grandi gruppi di sistemi di microprocessori:

- General Purpose: sono processori ad alte prestazioni e versatili, non limitati ad un solo possibile utilizzo. Fanno parte di questo gruppo i processori dei personal computer.
- Special Purpose: sono processori realizzati per compiti specifici. Ne fanno parte i microprocessori delle schede grafiche
- Embedded: sono sistemi a microprocessore progettati per una determinata applicazione, spesso con una piattaforma hardware ad hoc. Sono quindi integrati completamente nel sistema, di cui ne gestiscono tutte le funzionalità o quasi.

Ad oggi il mercato è costituito per il 98% da microprocessori per applicazioni embedded come elettrodomestici, periferiche computer e altro. I microprocessori per personal computer, pur rappresentando solo lo 0.2% del numero totale di pezzi prodotti, assorbono però la metà del fatturato totale.

## Sistemi, prestazioni e classificazioni

I vari microprocessori sono classificati in base a differenti parametri che possono essere più o meno incidenti a seconda dell'applicazione per cui vengono usati.

	<b>Desktop</b>	<b>Server</b>	<b>Embedded</b>
<b>Prezzo</b>	500\$-5000\$	5000\$-5M\$	10\$-100K\$
<b>Prezzo/μP</b>	50\$-500\$	200\$-10K\$	0,01\$-100\$
<b>Aspetti Critici</b>	<ul style="list-style-type: none"> <li>• Rapporto prezzo/prestazioni</li> <li>• Prezzo</li> <li>• Prestazioni Grafiche</li> </ul>	<ul style="list-style-type: none"> <li>• Throughput</li> <li>• Dependability (MTBF)</li> <li>• Scalabilità</li> </ul>	<ul style="list-style-type: none"> <li>• Consumo Energetico</li> <li>• Prestazioni in Task Specifici</li> </ul>

Nella tabella precedente si può vedere una classificazione in base al prezzo medio sul mercato, che può influire molto nelle applicazioni embedded le quali richiedono un elevato numero di microprocessori dalle basse prestazioni ma altamente specializzati nelle loro funzioni.

Per i server, invece, un parametro che ne guida il progetto architetturale è il MTBF (Mean Time Between Failures), da cui si può determinare un Costo orario di Disservizio, cioè la perdita di denaro che si subisce fintanto che il server non è nuovamente operativo.

Più in particolare  $MTBF = MTTF$  (Mean Time To Failure, tempo di individuazione del guasto) +  $MTTR$  (Mean Time To Repair, tempo di riparazione del guasto).

## Prestazioni

I principali parametri per la valutazione dello sviluppo e delle prestazioni sono:

- Throughput (ampiezza di banda) = (Operazioni, Azioni, Compiti) / Tempo, corrisponde alla quantità di operazioni svolte in un'unità di tempo.
- Latenza, ovvero il tempo medio impiegato per la completa esecuzione di un'operazione.

Nella valutazione dello sviluppo nel campo della microelettronica è molto importante la Potenza dissipata.

Essa è definita come:

$$P_{Dissipata} = \frac{1}{2} \cdot C_{Carico} \cdot Tensione^2 \cdot frequenza = \frac{Energia}{Tempo}$$

Per esempio, quindi, basterebbe diminuire la tensione di alimentazione per avere una notevole riduzione del consumo.

Inoltre, l'energia dissipata (potenza per tempo) è un parametro essenziale per tutti i sistemi portatili alimentati a batteria (ne determina la durata).

A parità di microelettronica sono stati apportati miglioramenti anche grazie all'utilizzo del parallelismo (numero di pipeline per gestire le istruzioni in parallelo), lo sfruttamento del principio di localizzazione spaziale (inserimento nella cache delle istruzioni adiacenti in memoria a quella in esecuzione) e temporale (inserimento nella cache delle istruzioni chiamate più frequentemente).

Un errore comune è però considerare la CPU, in quanto a prestazioni, l'elemento più influente dell'intera architettura di un elaboratore. In realtà le prestazioni di un sistema sono determinate anche e soprattutto da:

- Architettura del sistema (BUS, gerarchia delle memorie, I/O, interrupt)
- Prestazioni e tipologia delle periferiche (rete, memorie di massa)
- Software di base

Per esempio in un sistema per applicazioni web il 60% del tempo è occupato dalle operazioni di I/O, mentre solo il restante 40% dall'esecuzione delle istruzioni.

Quindi, sostituendo il processore di questo sistema con uno 10 volte più veloce, lo speed-up, ovvero l'aumento di prestazioni, sarebbe:

$$SpeedUp = \frac{Prestazioni_{MIGLIORAMENTO}}{Prestazioni_{ORIGINALI}} = \frac{T \cdot (\%Tempo_{I/O} + \%Tempo_{Elaborazione})}{T \cdot (\%Tempo_{I/O} + \frac{\%Tempo_{Elaborazione}}{Fattore_{MIGLIORAMENTO}})}$$

Ecco perchè in qualunque progetto di server il costo del processore non è mai dominante, mentre è molto rilevante quello dei dischi.

## Architettura dei microprocessori

Un microprocessore è composto da più unità:

- Unità di Controllo + ALU (Algebrical Logic Unit)
- Unità di Decodifica Istruzioni (determina la tempistica dovuta ai tempi di Fetch, Decode ed Execute. Nelle architetture CISC questi tre tempi sono variabili per la diversa lunghezza delle istruzioni, dal numero di microistruzioni necessarie per la loro implementazione e dal tipo di istruzione (1 o più argomenti, ecc...))
- Unità di Gestione Indirizzi
- Registri
- Unità di Gestione BUS

Il chipset, un'appendice della CPU posta di solita sulla scheda madre, si occupa della gestione periferiche, del DMA, etc...

Per migliorare le prestazioni si sono introdotti dei miglioramenti architetturali: si è scelto di disaccoppiare il bus di sistema da quello della CPU demandando la fase di prelevamento istruzioni alla BIU (Bus Interface Unit) ed inserendo una coda di prefetch, che si occupa di precaricare l'istruzione successiva del programma in esecuzione; è anche possibile utilizzare una memoria cache L1 apposta per il codice.

Si è inoltre scelto di aumentare il parallelismo di esecuzione introducendo una o più pipeline o utilizzando un'architettura superscalare, cioè un'architettura che permette di eseguire più istruzioni contemporaneamente.

Per quanto riguarda i sistemi multi-core il miglioramento delle prestazioni è ottenuto grazie ad un più ampio utilizzo del parallelismo tra thread (strategia Thread Level Parallelism) piuttosto che ad uno tra istruzioni (Instruction Level Parallelism): quest'ultima strategia infatti ha bisogno di più complessità hardware oltre che di nuovi criteri di programmazione.

### Differenze architetturali

	<b>IA-16 (x86-16)</b>	<b>IA-32 (x86-32)</b>	<b>X64 (x86-64)</b>
Indirizzamenti	16	16,32	16,32,64
Registri	8,16	8,16,32	8,16,32,64
Natività	8086 e 286	386,486,Pentium	Pentium

L'IA-64 non è presente nella precedente tabella in quanto è una architettura Intel completamente differente, chiamata Itanium, basata su tecnologia VLIW (Very Long Instruction Word). Le VLI sono macroistruzioni già ottimizzate per essere eseguite con il massimo parallelismo.

## Performance Evaluation ed Evoluzione dei microprocessori

In questo paragrafo vengono considerati, in termini quantitativi, i miglioramenti prestazionali introdotti da alcune evoluzioni tecnologiche ed architetturali sui microprocessori. In particolare si valuteranno i seguenti elementi:

- velocità del bus esterno
- parallelismo(prefetch, ...)
- cache L1

Per valutare le prestazioni si farà riferimento alla gestione della memoria video dato che è uno degli elementi critici nell'architettura del sistema sia in profondità nella memoria sia nell'aggiornamento della memoria video.

Di seguito ci saranno degli esempi riguardo la gestione della memoria video,utilizzando diversi sistemi architetturali e tecnologie.

### Caso di Studio

Memoria<sub>Video Richiesta</sub> = 600 pixel · 400 pixel · 3 Byte = 0,72 MB

Refresh = 50 Hz

Transfer Rate richiesto = 0,72 MB · 50 Hz = 36 MB/s

Tempo aggiornamento richiesto = 20 ms

Come tempo di esecuzione si considera il tempo necessario a fare una scrittura in memoria

### Sistema I

#### Caratteristiche

Sistema sequenziale, Monobus

Microprocessore M24 (8086/88  $f_p=8\text{MHz}$ )

BUS pre-ISA

Ampiezza<sub>BUS</sub> = 16bit

$f_{BUS}=8\text{MHz}$

$T_{ck} = 1/f_p = 125 \text{ ns}$

$T_{\text{Accesso Memoria}} = 4 \cdot T_{ck} = 500 \text{ ns}$

#### Elaborazione

Calcolo del Tempo medio di istruzione,  $T_{\text{Medio Istruzione}}$

Lunghezza media istruzioni = 2,5 Byte

(2,5 Byte / Ampiezza<sub>BUS</sub> = 2 accessi in memoria per la fase di fetch)

$T_{\text{Medio Istruzione}} = T_{\text{Fetch}} + T_{\text{Execution}} = 2 \cdot 500 + 500 \text{ ns} = 1,5 \mu\text{s}$

Milioni di operazioni in un secondo =  $1 \text{ s} / 1,5 \mu\text{s} = 0.7 \text{ Mips}$

Transfer Rate =  $1 / T_{\text{Accesso Memoria}} \cdot \text{Ampiezza}_{BUS} = 4 \text{ MB/s}$

CASO: Refresh video memory

Per l'aggiornamento della memoria video mediante istruzioni è richiesto un tempo pari a:

$\text{Tempo}_{\text{Aggiornamento Medio}} = \text{Memoria}_{\text{Video Richiesta}} \cdot T_{\text{Medio Istruzione}} / \text{Ampiezza}_{BUS} =$   
 $0,72 \text{ MB} \cdot 1,5 \mu\text{s} / 2 \text{ Byte} = 540 \text{ ms} \gg 20 \text{ ms}$

Il sistema NON risulta essere sufficiente.

## Evoluzione o

Prefetch Buffer  
Faster BUS ( $2 \cdot T_{ck}$ )

## Sistema II

### Caratteristiche

Sistema sequenziale, Monobus  
Microprocessore M24 (8086/88  $f_p=8\text{MHz}$ )  
BUS pre-ISA  
Ampiezza<sub>BUS</sub> = 16bit  
 $f_{BUS}=8\text{MHz}$   
 $T_{ck} = 1/f_p = 125 \text{ ns}$   
 $T_{\text{Accesso Memoria}} = 2 \cdot T_{ck} = 250 \text{ ns}$

### Elaborazione

Calcolo del Tempo medio di istruzione,  $T_{\text{Medio Istruzione}}$   
Lunghezza media istruzioni = 2,5 byte (2,5 Byte / Ampiezza<sub>BUS</sub> = 2 accessi per il Fetch)  
 $T_{\text{Medio Istruzione}} = T_{\text{Fetch}} + T_{\text{Execution}} = 2 \cdot 500 + 250 = 250 \text{ ns}$   
Non viene considerato il  $T_{\text{Fetch}}$ , siccome è attivo il Prefetching  
Milioni di operazioni in un secondo =  $1 \text{ s} / 250 \text{ ns} = 4 \text{ Mips}$   
Transfer Rate =  $1 / T_{\text{Accesso Memoria}} \cdot \text{Ampiezza}_{BUS} = 8 \text{ MB/s}$   
CASO: Refresh video memory  
Per l'aggiornamento della memoria video mediante istruzioni è richiesto un tempo pari a:  
 $\text{Tempo}_{\text{Aggiornamento Medio}} = \text{Memoria}_{\text{Video Richiesta}} \cdot T_{\text{Medio Istruzione}} / \text{Ampiezza}_{BUS} =$   
 $0,72 \text{ MB} \cdot 250 \text{ ns} / 2 \text{ Byte} = 90 \text{ ms} > 20 \text{ ms}$

Il sistema NON risulta ancora essere sufficiente.

## Sistema III

### Caratteristiche

Sistema parallelo, Monobus + veloce  
Pentium I (30 MHz)  
BUS  
Ampiezza<sub>BUS</sub> = 32bit  
 $f_{BUS}=30\text{MHz}$   
 $T_{ck} = 1/f_p = 33 \text{ ns}$   
 $T_{\text{Accesso Memoria}} = 2 \cdot T_{ck} = 66 \text{ ns}$

### Elaborazione

Calcolo del Tempo medio di istruzione,  $T_{\text{Medio Istruzione}}$   
Lunghezza media istruzioni = 2,5 Byte (2,5 / Ampiezza<sub>BUS</sub> = 1 accesso in memoria per il Fetch)  
Tempo medio istruzione =  $T_{\text{Fetch}} + T_{\text{Execution}} = 0 \text{ (Prefetching)} + 66 = 66 \text{ ns}$   
Milioni di operazioni in un secondo =  $1 \text{ s} / 66 \text{ ns} = 15,2 \text{ Mips}$   
Transfer Rate =  $1 / T_{\text{Accesso Memoria}} \cdot \text{Ampiezza}_{BUS} = 60 \text{ MB/s}$   
 $\text{Tempo}_{\text{Aggiornamento Medio}} = \text{Memoria}_{\text{Video Richiesta}} \cdot T_{\text{Medio Istruzione}} / \text{Ampiezza}_{BUS} =$   
 $0,72 \text{ MB} \cdot 66 \text{ ns} / 4 \text{ Byte} = 11,88 \text{ ms}$

Il sistema risulta essere sufficiente.

## Evoluzione 1

Multi Level Bus (AGP, PCI, ISA)  
Cache (multilevel)

## Sistema IV

### Caratteristiche

Il tempo di istruzione è una media tra i vari tempi di CPU, memoria e I/O.

Calcolo del Tempo medio di istruzione,  $T_{\text{Medio Istruzione}}$

$$T_{\text{Medio Istruzione}} = \text{Media}\{T_{\text{CPU}}, T_{\text{I/O}}, T_{\text{Memoria}}\}$$

$T_{\text{CPU}}$  è il tempo dovuto a istruzioni che impiegano solamente la CPU

$T_{\text{I/O}}$  è il tempo dovuto a istruzioni che coinvolgono dispositivi di I/O

$T_{\text{Memoria}}$  è il tempo dovuto a istruzioni che effettuano l'accesso in memoria

### Ipotesi base

Il calcolatore esegue istruzioni ripartite nel seguente modo:

25% tempo dedicato alle istruzioni della CPU

70% tempo dedicato alle istruzioni per l'accesso in memoria;

5% tempo dedicato alle istruzioni per i dispositivi di I/O

Calcoliamo i singoli tempi:

$$T_{\text{CPU}} = 5 \text{ ns (200 MiPS, 200 MHz)}$$

$T_{\text{Memoria}}$ :

Tempo Cache (Hit/Miss = 90%) = 10 ns

Tempo DRAM = 60 ns

### Ipotesi 1

$T_{\text{I/O}}$ :

Tempo ISA (10 MHz,  $2 \cdot T_{\text{ck}}$ ) = 200 ns (5 MiPS)

Otteniamo

$$\text{Tempo totale ip. 1} = 25\% \cdot 5 \text{ ns} + 70\% \cdot (90\% \cdot 10 \text{ ns} + 10\% \cdot 60 \text{ ns}) + 5\% \cdot 200 \text{ ns} = 21,75 \text{ ns}$$

### Ipotesi 2

$T_{\text{I/O}}$ :

Tempo PCI (66 MHz,  $2 \cdot T_{\text{ck}}$ ) = 33 ns (30 MiPS)

Otteniamo

$$\text{Tempo totale ip. 2} = 25\% \cdot 5 \text{ ns} + 70\% \cdot (90\% \cdot 10 \text{ ns} + 10\% \cdot 60 \text{ ns}) + 5\% \cdot 33 \text{ ns} = 13,4 \text{ ns}$$

Si può notare come anche un aumento della frequenza di lavoro della CPU influisce in modo non superiore a pochi punti percentuali sul tempo totale.

Ad esempio un miglioramento alla sola CPU del 100% ( $T_{\text{CPU}}$  si dimezza) riduce solo di circa il 5% il Tempo totale.



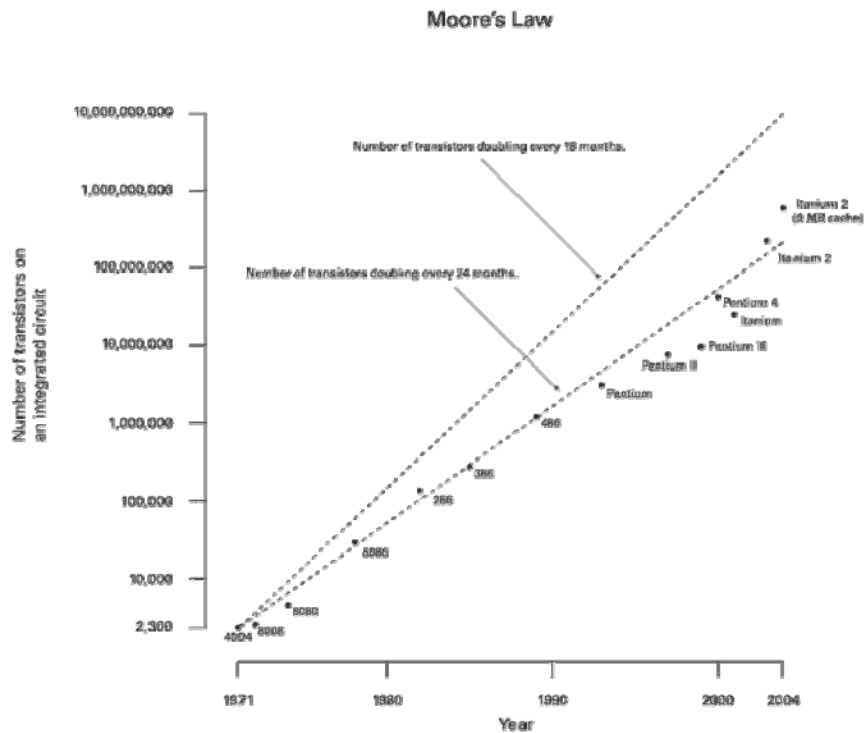
## Evoluzione 2

Dual/Four Core Architecture

Dual Level instruction set

Parallelismo implicito a livello hardware (Pentium) o compilatore (Itanium-EPIC)

In base alla [Legge di Moore](#), le prestazioni dei processori ed il rispettivo numero di circuiti integrati raddoppia ogni 18 mesi.



Nell'ultimo decennio, si è verificata la tendenza a raddoppiare il numero di core per microprocessore.

# Architettura x86-32

## Modalità di funzionamento

L'architettura IA-32 ha 3 diverse modalità di funzionamento. Ogni modalità determina quali istruzioni e features del processore sono utilizzabili.

Esse sono:

- Real Mode (Real Address Mode)

E' attiva al momento del Power-On o Reset. In questa modalità si ha uno spazio di allocazione di memoria a 20bit, accesso diretto alle routine del BIOS.

La quantità massima di memoria indirizzabile è 1MB ( $2^{20}$  Byte).

Non è presente (o attivo) alcun sistema di protezione memoria e multitasking a livello hardware.

- Protected Mode (Protected Virtual Address Mode)

E' lo stato nativo del processore. Modo base a 32bit (Windows e Linux).

La quantità massima di memoria indirizzabile è 4GB ( $2^{32}$  Byte).

In questa modalità è possibile utilizzare memoria virtuale, paging, multitasking (safe).

Si passa in questa modalità modificando alcuni registri di controllo ed abilitando opportuni flag.

- VM86 (Virtual Mode)

La VM86 (Virtual Mode) permette l'esecuzione di software Real Mode che non possono essere eseguite all'interno di ambienti in Protected Mode (ad esempio, applicazioni DOS su O.S. recenti).

Essa usa una segmentazione identica alla Real Mode.

Vi è comunque il meccanismo di paginazione, ma è trasparente al programmatore. E' attiva la protezione della memoria e l'isolamento dello spazio indirizzi.

Nei processori attuali è in realtà una sotto-modalità della Protected Mode.

Esempio:

*Richiesta di Accesso a disco*

- Viene richiamato l'O.S (Protected Mode)
  - Viene coinvolto il Filesystem (Protected Mode)
    - Viene richiamato il Driver Logico (BIOS) (Protected Mode)
      - INT xx (Real Mode, non rientrante)

Solitamente, in questa fase, l'O.S. bypassa il BIOS.

## Registri e Modes

A seconda della modalità in cui si opera, si ha una diversa visibilità di registri e modello di memoria.

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

## Registri, segmenti e principali funzioni

I registri possono essere classificati a seconda del loro utilizzo.

### Data Registers

Sono utilizzati per memorizzare operandi ed i risultati delle operazioni.

Registro	Descrizione	Utilizzato ANCHE	Note per l'Assembler
EAX / AX / AH AL	Accumulator Register		Moltiplicazione, Divisione, I/O, Shift Veloce
EBX / BX / BH BL	Base Register	Per calcolo indirizzi	
ECX / CX / CH CL	Count Register	Come contatore	Cicli, Rotazioni, Shift
EDX / DX / DH DL	Data Register	Per operazioni di I/O come puntatore	Moltiplicazione, Divisione, I/O

## Pointer Registers

Sono utilizzati come puntatori.

Registro	Descrizione	Funzione	Note per l'Assembler
EIP / IP	Instruction Pointer	Punta alla successiva istruzione da eseguire	Non modificabile dall'utente
ESI / SI	Source Index	Registri indice per la memoria	Da trattare come operandi di tipo mem
EDI / DI	Destination Index	Registri indice per la memoria	Da trattare come operandi di tipo mem

## Stack Registers

Sono utilizzati per l'implementazione dello stack.

Registro	Descrizione	Funzione	Note per l'Assembler
ESP / SP	Stack Pointer	Punta alla testa dello stack	L'indirizzo viene decrementato ad ogni PUSH e incrementato ad ogni POP
EBP / BP	Base Pointer	Base address dello stack	

## Test Registers

Registri utilizzati nella fase di Self-Test

Registro	Descrizione
TR7	Test Data
TR6	Test Command

## Segment Registers

Registro	Registri OFFSET validi	Descrizione	Funzione	Note per l'Assembler
DS,ES	EBX / ESI / EDI / BX / SI / DI	Data Segment	Indirizzo base del segmento dati	
CS	EIP / IP	Code Segment	Indirizzo base del segmento codice	Per modificare tale registro è necessaria una chiamata a <i>procedura far</i> , oppure una <i>far jump</i> oppure un <i>interrupt</i> . In Protected Mode viene verificato se il nuovo segmento può essere utilizzato dal task
SS	ESP / EBP / SP / BP	Stack Segment	Indirizzo del segmento stack	

## Modelli di memoria

Le applicazioni non accedono direttamente alla memoria fisica. La memoria può essere vista dal programma in 3 diversi modi:

- Flat memory

La memoria appare come un singolo blocco (Linear Address Space) di dimensioni  $(2^{32}-1)$  Byte.

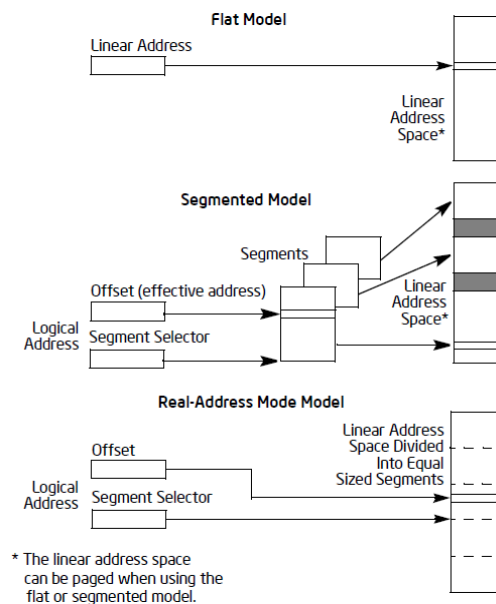
- Segmented Memory Model

La memoria appare come un gruppo di spazi d'indirizzamento indipendenti, chiamati segmenti. Ogni applicazione può indirizzare fino a  $(2^{16}-1)$  segmenti di differenti dimensioni e tipologie. Ogni segmento può essere di dimensioni massime  $2^{32}$  Byte.

Il programma, per accedere ad un Byte, si riferisce ad un indirizzo logico. Esso è definito mediante un Segment Selector e un Offset.

- Real-address mode Memory Model

E' il modello di memoria per i processori Intel 8086 compatibili. Andremo ad analizzare in modo approfondito solo quest'ultima modalità. Maggiori informazioni al capitolo "8086 Emulation" de [Intel® 64 and IA-32 Architectures Software Developer's Manual – Volume 3A](#).

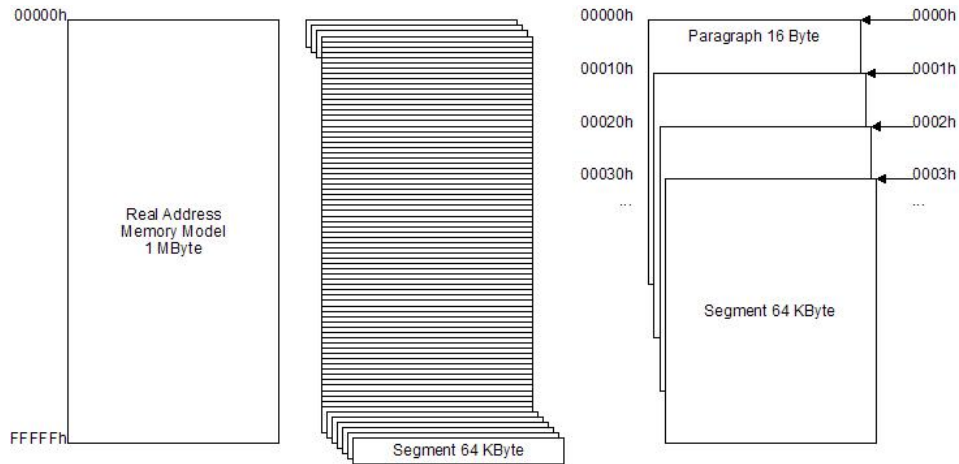


## Real-address mode Memory Model

I segmenti sono visibili al programmatore a livello istruzione.

Essi 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 un paragrafo (16 Byte) ognuno.



Il *Physical Address* (20bit) viene ottenuto fornendo un indirizzo di segmento (*Segment Address*, 16bit) e un *Offset* (16bit).

Il *Segment Address* verrà shiftato di 4 bit (equivale ad una moltiplicazione per 16, segnata nell'esempio in **grassetto**) e sommato all'*Offset*.

<i>Segment Address</i>	0100	0100	1100	0011	<b>0000</b>	b	+
<i>Offset</i>		0000	0000	0001	0011	b	=
<i>Physical Address</i>	0100	0100	1100	0100	0011	b	

Tale tipo di segmentazione minimizza lo spreco di memoria. Inoltre, si ottiene uno spazio di allocazione (virtuale) a 20bit chiuso su se stesso.

Ad esempio, posizionandoci all'ultimo segmento (che fisicamente sarà di lunghezza pari a 16 Byte)

*Segment Address*    FFFFh

E imponendo un offset superiore a 16 Byte

*Offset*                0011h

Ci stiamo riferendo all'indirizzo

*Physical Address*    00001h

Fisicamente, l'indirizzo punta al 2° Byte dello spazio di indirizzamento.

In alcune architetture non è possibile manipolare i registri di segmento, ma solo gli offset. In questi casi è il Loader dell'O.S. ad occuparsi del caricamento di tali registri.

## Process Status Word

La Process Status Word (PSW) è il "registro" FLAG. E' composto da 16bit.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
o	NT	IOPL	IOPL	OF	DF	IF	TF	SF	ZF	o	AF	o	PF	1	CF

E' possibile individuare 2 gruppi principali:

- Bit di Flag
  - OF - Overflow Flag, attivo se l'ultima istruzione ha generato overflow
  - SF - Sign Flag, attivo se il MSB del risultato dell'ultima operazione aritmetica
  - ZF - Zero Flag, attivo se il risultato è nullo
  - AF - Aux Carry Flag, attivo se è avvenuto (BCD) riporto o prestito in operazioni di somma o sottrazione
  - PF - Parity Flag, attivo se il numero di bit a 1 degli 8 bit meno significativi è pari
  - CF - Carry Flag, attivo se è avvenuto riporto o prestito in operazioni di somma o sottrazione (altre istruzioni ne fanno uso speciali)
- Bit di Controllo
  - DF - Direction Flag
    - 0 se le stringhe vanno manipolate a partire da indirizzi minori
    - 1 se le stringhe vanno manipolate a partire da indirizzi maggiori
  - IF - Interrupt Flag
    - 0 se gli interrupt mascherabili sono ignorati dalla CPU
    - 1 se gli interrupt sono gestiti normalmente dalla CPU
  - TF - Trap Flag
    - 1 se viene generata una Trap al termine di ogni istruzione (debugging)
    - 0 se l'esecuzione è normale

## Memoria: parti riservate e dedicate

La scrittura (e lettura) in memoria è in little endian. Il Byte meno significativo va nella locazione con indirizzo inferiore.

Esempio:

Memorizzazione WORD di valore FA10h

00000h	.	.
...	.	.
0000Fh	.	.
00010h	0	1
00011h	A	F
00012h	.	.
...	.	.
FFFFFh	.	.

La memoria è suddivisa, a grandi linee, in:

00000h	Interrupt Vector Table
003FFh	
	Liberi
FFFF0h	Reset Bootstrap
FFFFCh	Riservati

La IVT contiene:

- in Real Mode: 256 puntatori di 4 Byte [Segmento 16bit , Offset 16bit]. I primi 32 vettori sono riservati a eccezioni interne del processore.
- in Protected Mode: 256 puntatori di 8 Byte [Offset H 16bit, Flags (Segment Present, DPL o Kernel Ring, Type, Padding), Segment Selector, Offset L 16bit]. Essi possono essere Interrupt Gates, Trap Gates o Task Gates. I primi 32 vettori sono riservati a eccezioni interne del processore.

Il Reset Bootstrap contiene una *JMP* alla procedura di Bootstrap.

La vera e propria procedura è solitamente posizionata nel BIOS (in sistemi PC) ed è mappata logicamente al di sopra della Reset Bootstrap.

Reset Bootstrap, procedure base (BIOS) e parte Riservata (Versione BIOS) sono fisicamente in una ROM.

Esempio:

RAM	00000h	Interrupt Vector Table
	003FFh	Liberi
ROM	FF800h	BIOS - POST (Power on Self Test)
	FF810h	BIOS - Procedure base e software
	FFFF0h	Reset Bootstrap
	FFFFCh	Riservati



Il POST è una procedura molto importante: durante questa fase il sistema realizza la memoria presente e controlla il suo funzionamento.

La CPU vede fisicamente i BUS. Le altre interpretazioni ("legge il dato dalla memoria", "legge un byte da una periferica di I/O", ...) sono solo interpretazioni logiche.

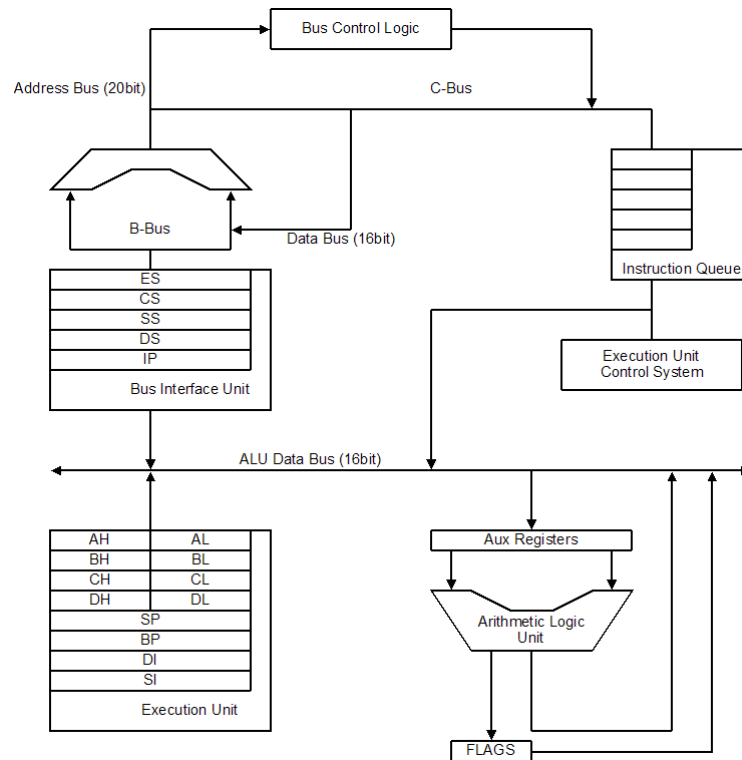
Esempio:

La CPU, a fronte dell'istruzione

*MOV AX, [indirizzo che fa riferimento ad una locazione non fisicamente presente]*

Continua ad eseguire il programma, anche se in AX sarà presente il valore FFFFh (dovuto al fatto che solitamente i BUS sono a  $V_{cc}$  quando scollegati).

## Architettura interna 8086



### La Bus Interface Unit:

- esegue il Fetching delle Istruzioni, accodandole sulla Instruction Queue
- legge e scrive gli operandi e risultati delle operazioni
- genera gli indirizzi

### La Execution Unit:

- decodifica Istruzioni (può richiedere alla BIU la risoluzione di indirizzi, se necessario)
- esegue Istruzioni
- testa ed aggiorna i FLAG

La *Instruction Queue* è una struttura FIFO di dimensioni pari a 6 Byte. Essa:

- viene caricata dalla BIU ogni qual volta vi è una Word libera ed il BUS è libero (lettura dal CS)
- in caso di istruzioni JMP, la IQ è azzerata e la EU deve attendere la BIU

## Pipelining

La velocità di esecuzione dei programmi è influenzata da diversi fattori. Oltre alla tecnologia utilizzata per l'implementazione dei circuiti (elettronica utilizzata, ecc.), è importante l'organizzazione hardware.

L'esecuzione simultanea di più operazioni aumenta il throughput, nonostante il tempo necessario per eseguire una singola operazione non vari.

Un'istruzione può essere suddivisa in 2 fasi principali:

- Fetch, prelievo che necessita di un  $T_f$
- Execution, esecuzione che necessita di un  $T_e$

Bisogna innanzitutto tenere in considerazione che le 2 fasi non hanno la stessa durata.

Ma non sono  $T_f$  e  $T_e$  i valori da considerare.

Il fattore da tenere in considerazione nella scelta di implementare o meno una sistema di pipelining è la varianza di tempo necessario per lo "svolgimento" delle varie fasi.

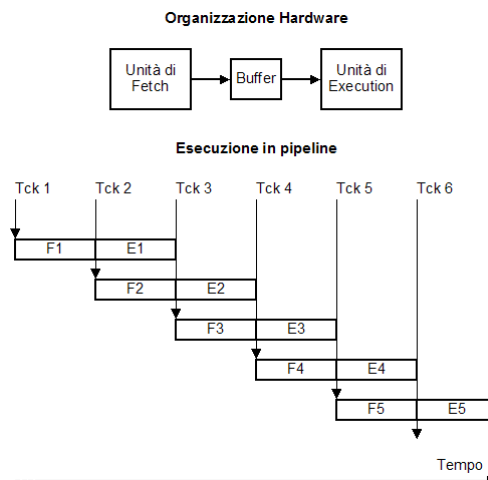
Stiamo considerando il valore di  $|T_e - T_f|$

Se esso risulta essere elevato, la pipeline comporta minor vantaggio.

Dobbiamo focalizzare l'attenzione sul rendimento a régime, ovvero il throughput.

Il pipelining può essere di tipo:

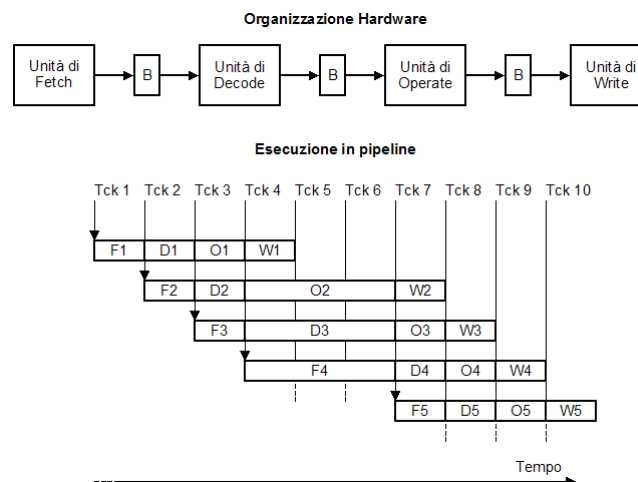
- ideale, se ogni stadio o fase ha egual durata: in questo caso si ha una riduzione del tempo medio di esecuzione istruzioni di un fattore pari al numero di stadi della pipeline (il throughput è proporzionale al numero di stadi)
- imperfetto, se gli stadi non sono di eguale durata



In realtà un'istruzione è suddivisa in 4 fasi:

- Fetch, prelievo
- Decode, decodifica
- Operate, elaborazione
- Write, scrittura

Andiamo ora a considerare il caso in cui la fase di Operate impieghi più di un ciclo di clock per essere eseguita.



Al 4° ciclo di clock:

- l'unità di Fetch può iniziare il fetch dell'istruzione successiva
- l'unità di Decode effettuerà il decoding dell'istruzione 3, ma dovrà attendere l'unità di Operate (che lavora sull'istruzione 2)

Al 5° ciclo di clock:

- l'unità di Fetch non può procedere oltre perchè il Buffer precedente all'unità di Decode è già occupato dall'istruzione 3
- l'unità di Decode non può procedere oltre

Al 7° ciclo di clock:

- l'unità di Operate ha concluso l'elaborazione dell'istruzione 2 ed inizia l'elaborazione della istruzione 3
- l'unità di Decode può decodificare l'istruzione 4
- l'unità di Fetch può passare all'istruzione 5

Quando si verificano queste situazioni, la pipeline entra in *stallo*.

Può essere causata da operazioni aritmetiche particolarmente lunghe.

Lo stallo comporta un degrado di prestazioni.

Le istruzioni critiche possono essere:

- istruzioni che coinvolgono il fetch da memoria (Cache MISS, ad esempio)
- istruzioni di JMP condizionato, a cui si è avviato utilizzando
  - Architetture speculative
  - Branch prediction

Esempio:

ADD AX, [BX]

Modo Reale

1. CS:IP punta all'istruzione
  - a. l'indirizzo viene posto sulla BIU, il blocco di Fetch ottiene l'istruzione dalla gerarchia delle memorie (possibilità di ricorrere alla memoria) o dall'IQ
2. il blocco D1 si occupa di interpretare l'istruzione e settati i segnali di controllo per la ALU in modo che possa essere eseguita una "ADD registro, memoria"
3. il blocco D2 risolve il dato relativo all'indirizzo DS:BX
  - a. conversione a indirizzo segmentato
  - b. conversione per Address BUS
  - c. fetch del dato (2 Byte) in un registro temporaneo: possibile stallo (in caso di Cache MISS)
4. il blocco di EX ha AX ed il registro temporaneo: la ADD può essere eseguita via microcodice o moltiplicazione hardware
5. il WB non implica nessuna scrittura in memoria ed il risultato è posto su AX ed aggiornati i FLAG della PSW

Il tempo reale è dato, in generale, da

$$T_p = 1/f_p = \tau_{MAX}$$

$$T_{ADD\ AX,[BX]} = \tau_F + \tau_{D1} + \tau_{D2} + \tau_{EX} + \tau_{WB}$$

Best Case

$$T_{ADD\ AX,[BX]} = \tau_{MAX} + \tau_{MAX} + \tau_{MAX} + \tau_{MAX} + \tau_{MAX} = 5 \cdot \tau_{MAX}$$

Worst Case

$$T_{ADD\ AX,[BX]} = m \cdot \tau_{MAX} + \tau_{MAX} + m \cdot \tau_{MAX} + \tau_{MAX} + \tau_{MAX} = (3+2 \cdot m) \cdot \tau_{MAX}$$

m: fattore per la lettura da DRAM

Da questi punti, è possibile notare che:

- la lunghezza del ciclo di clock deve essere sufficiente per completare le attività svolte da ogni stadio, ovvero  $T_{ck} \geq \max\{\tau_F, \tau_D, \tau_O, \tau_W\}$
- il tempo di accesso alla memoria principale può essere 10 volte maggiore di quello necessario per operazioni interne al processore: da qui l'introduzione della memoria cache
- è stato necessario dividere la memoria cache in 2 parti indipendenti (istruzioni e dati), in modo da eliminare i problemi di concorrenza
- è importante considerare i vincoli di dipendenza delle istruzioni: la destinazione di un'istruzione j viene utilizzata come sorgente di un'istruzione j+1
- con il Pentium, si è considerato il fattore di avere una "densità di lavoro" più o meno simile per tutti gli stadi

Nell'era pre-Pentium, le pipeline erano imperfette.

Dal Pentium si è passati a pipeline ideale (a meno di alcuni casi in cui vengono introdotti dei cicli di clock tampone per istruzioni più esose in termini di tempo).

Il Pentium ha 2 pipeline “integer”(u, v) ed una Floating Point. Sotto alcune condizioni, le u e v possono operare in parallelo.

Ogni pipeline “integer” ha 5 stadi, ognuna con durata 1 ciclo di clock (nel Best Case):

- IF, è lo stadio instruction prefetch stage: vengono caricate 2 istruzioni successive, una per pipeline e con l’aiuto del sistema di Branch Prediction, vengono evitati alcuni casi di flush completo di pipeline
- D1, decodifica istruzione e controlli di privilegio e attributi in modo protetto
- D2, calcolo degli indirizzi degli operandi (calcolo indirizzi reali tramite segmentazione e paginazione)
- EX, esecuzione istruzione (utilizzo ALU)
- WB, write-back in registro, cache o memoria e settaggio definitivo della PSW

Dopo un lungo periodo di evoluzione che ha portato a 30 stadi di pipeline negli ultimi anni, vi è ora una regressione volta al TLP (Thread Level Parallelism).

E' possibile calcolare un "fattore di accelerazione", pari a:

$$PipelineAcceleration = \frac{T_{ESECUZIONE,NOPIPELINE}}{T_{ESECUZIONE,PIPELINE}} = \frac{N \cdot SP \cdot T_{MAX}}{((SP \cdot T_{MAX}) + (N - 1) \cdot T_{MAX})} = \frac{N \cdot SP}{SP + N - 1}$$

$$\lim_{N \rightarrow \infty} \frac{N \cdot SP}{SP + N - 1} = SP$$

N: Istruzioni

SP: Stadi di Pipeline

$T_{MAX}$ :  $\max\{T_{Stadio}(1), T_{Stadio}(2), \dots T_{Stadio}(SP)\}$

## I/O

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 d'indirizzamento per la memoria è "occupato" dalle periferiche. A seconda del sistema, tale occupazione può essere temporanea o permanente.
- In Isolated I/O: l'accesso avviene attraverso speciali istruzioni (IN e OUT). Le periferiche hanno indirizzi dedicati.

Lo spazio d'indirizzamento è al più di 64KByte, 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, quali:

- La FPU (Floating Point Unit): permette di eseguire operazioni via hardware. Esegue sia operazioni fixed, floating point e calcolo di funzioni trascendentali e trigonometriche. E' un processore con architettura a stack. Oltre all'hardware necessario, sono disponibili registri e istruzioni ad hoc.
  - Registri:
    - FPU Registers Stack (80 bit)
    - FPU Instruction Pointer (48 bit)
    - FPU Data Pointer (48 bit)
    - FPU Control, Status, Tag Registers (16 bit)
    - FPU Opcode Register (11 bit)
  - Istruzioni: FCOS, FSIN, FADD...
- La MMX (MultiMedia eXtension) è una tecnologia che permette di elaborare dati organizzati a blocchi da 64 bit. E' divenuta una necessità a causa del crescente utilizzo di interfacce grafiche a finestra. Si hanno a disposizione registri e istruzioni dedicate.
  - Registri:
    - 8 MMX Registers (64 bit)
  - Istruzioni: possono essere utilizzati come normali registri, maneggiabili in diversi formati (8·Byte = 64-bit packed Byte integers, 4·Word = 64-bit packed Word integers, 2·DWord = 64-bit packed DWord integers). Il set di istruzioni comprende le comuni operazioni di trasferimento, aritmetiche, logiche, ecc. ... Le istruzioni SIMD possono operare al massimo su 4 Word con una singola istruzione.
- 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 registri ed istruzioni.
  - Registri:
    - 8 XMM Registers (128 bit) in non-64-bit modes oppure 16 XMM Registers (128 bit) in 64-bit mode
    - MXCSR Register (32 bit) per controllo e stato delle operazioni eseguite sui XMM Registers
  - Istruzioni: viene reso disponibile un nuovo formato (4 IEEE single-precision floating point value). Sono disponibili istruzioni che eseguono operazioni SIMD su floating point e su interi (anche se i dati sono memorizzati in registri MMX) e che permettono il prefetch esplicito dei dati, controllo della cache e memorizzazione.

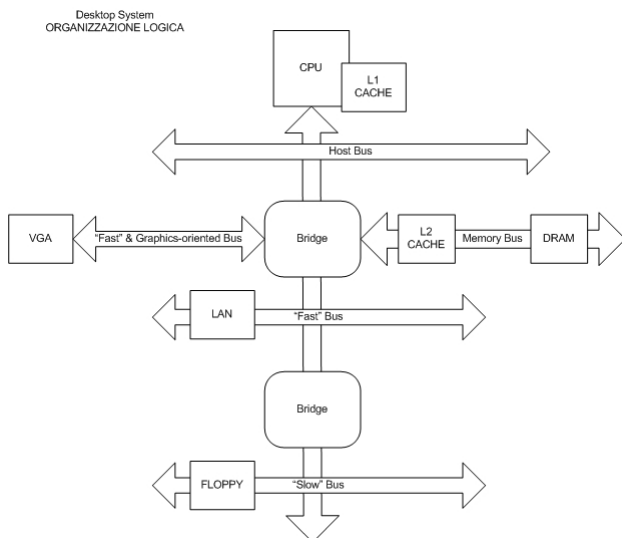


# BUS e segnali di controllo

## I livelli di BUS

A seconda del sistema in analisi (Desktop o Server), abbiamo diverse scelte di organizzazione dei BUS.

Di base, vi è una organizzazione logica comune per tutte le implementazioni.



Andiamo ad analizzare alcuni esempi.

L'organizzazione logica di un sistema Desktop prevede una suddivisione su più bus, i quali sono interconnessi da dei Bridge, che effettuano una sorta di routing dei segnali (da notare che i BUS possono avere diverse frequenze di funzionamento). Il Bridge interposto fra Host e "Fast" BUS ha tra le sue principali funzioni, quella di gestire la memoria.

Scendendo nei livelli si nota come i vari Bridge dovranno effettuare più "conversioni" per far comunicare la CPU con un device periferico, aumentando il ritardo.

Osservando una organizzazione reale di un sistema Desktop, possiamo notare che sono presenti 2 Bridge:

- Il **NorthBridge**, si occupa della gestione della memoria
- Il **SouthBridge** comunica con il NorthBridge ed attualmente ingloba un numero elevato di Controller (IDE, Serial ATA, USB, ...), oltre a gestire il BUS PCI ed eventuali Graphics Controller integrate (o BUS veloci).

Il **Super I/O** è un dispositivo che gestisce tutte le interfacce più datate (Porta seriale e parallela, porte PS/2, ...).

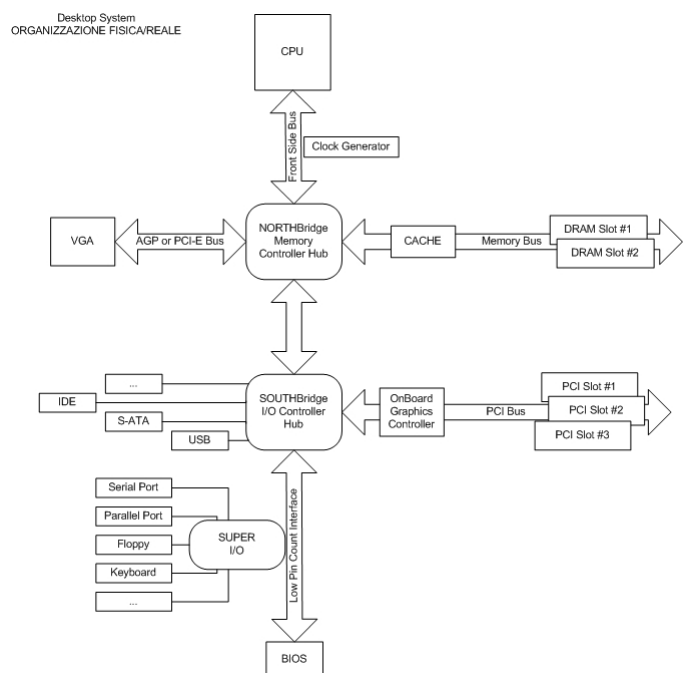
Comunica con il SouthBridge attraverso un'interfaccia chiamata LPC, la quale è andata a sostituire in un certo modo I/O ISA based.

La **Low Pin Count** interface permette di rendere l'implementazione hardware dei dispositivi di I/O meno esosa in termini di numero di linee di comunicazione.

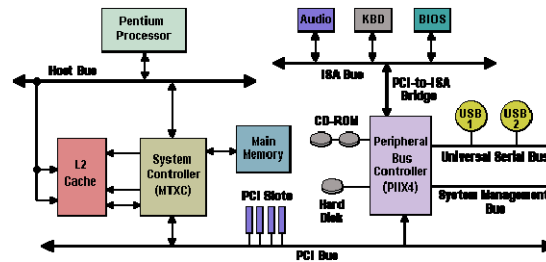
Un possibile esempio per evidenziare le differenze velocistiche ai vari livelli di BUS può essere quella del Fetch di un'istruzione memorizzata nel BIOS.

Il BIOS lavora a 8MHz: escludendo i ritardi dovuti ai vari Bridge presenti fra CPU e BIOS, il Fetch da BIOS avrà una durata di 250 ns, contro 2 ns di un Fetch da Cache.

Da qui l'idea di implementare la **Shadow RAM**. All'avvio, o su richiesta, il contenuto della ROM viene copiato in una sezione della RAM. Gli indirizzi occupati vengono rimappati in modo da occupare gli indirizzi utilizzati originariamente dal BIOS.



Analizziamo ora una implementazione reale: il chipset Triton 430TX.



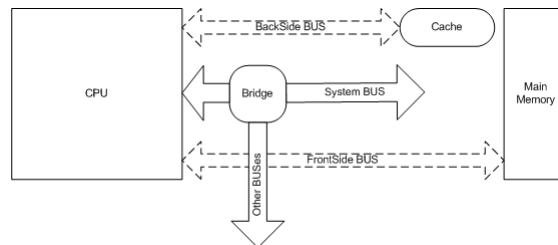
La scelta di far affacciare sullo stesso BUS sincrono sia Main Memory che la Cache comporta lo svantaggio di non poter trasferire direttamente dati tra di esse.

Un passo evolutivo è dato dal *Dual Independent Bus*.

Nei sistemi con architettura DIB, il BUS singolo è sostituito da:

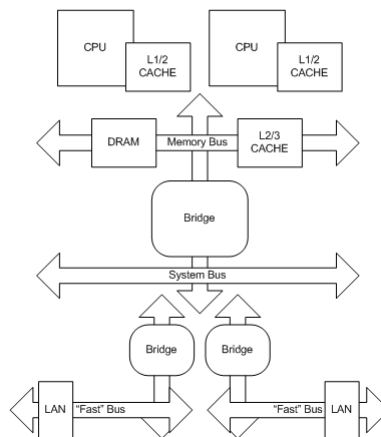
- un *Front Side BUS* per il trasferimento dei dati tra CPU e Main Memory
- un *BackSide BUS* per l'accesso alla Cache

Ciò permette di effettuare trasferimenti da memoria e cache contemporaneamente e quindi implementare il pipelining.



L'immagine successiva rappresenta una schematizzazione logica di un sistema Server. I sistemi server sono solitamente SMP e dispongono di più BUS ad alta velocità.

Server System  
ORGANIZZAZIONE LOGICA



## Cicli di BUS

Un ciclo di BUS è una sequenza di eventi attraverso i quali il  $\mu P$  comunica con la memoria, dispositivi di I/O o l'interrupt controller.

In funzione della CPU, un ciclo di BUS può essere di durata pari a 2 o 4 cicli di clock ( $T_{CLK}$ ).

In ogni caso, il ciclo di BUS è composto sempre da 4 periodi ( $T_1, T_2, T_3, T_4$ ).

Nell'8086, il ciclo del BUS on-chip coincide con il ciclo di BUS esterno ed è di  $4 \cdot T_{CLK}$ .

Nei Pentium (o superiori) e nei BUS di sistema (PCI, ad esempio), il ciclo di BUS è di  $2 \cdot T_{CLK}$ .

Esempio:

*Pentium con Host BUS pari a 100MHz. Data BUS con parallelismo 64bit.*

Un trasferimento avviene in  $2 \cdot T_{CLK}$ , ovvero 20 ns.

$$Bandwidth = \frac{\#Bit_{DataBUS}}{8bit} \cdot f_{BUS} \cdot \frac{1}{\#T_{CLK}}$$

Nel nostro caso, il Bandwidth è pari a  $8 \text{ Byte} \cdot 100 \text{ MHz} \cdot \frac{1}{2} = 400 \text{ MBps}$

Nelle architetture attuali si parla di MT/s o GT/s, che indicano il numero di trasferimenti al secondo, indipendentemente dal parallelismo e timings.

Esempio:

*DDR, 100MHz: vengono sfruttati sia il fronte  $L \rightarrow H$  che  $H \rightarrow L$ .*

Il Bandwidth è di 200 MT/s.

Perché non aumentare l'FSB? Principalmente per motivi di disturbi elettromagnetici e/o trasmissioni di linea. Il segnale e gli stessi dati viaggiano su linee fisiche. Al crescere delle frequenze abbiamo una diminuzione della lunghezza d'onda.

a)  $100 \text{ MHz} \rightarrow \lambda = c/100\text{MHz} = 3 \text{ m}$

b)  $200 \text{ MHz} \rightarrow \lambda = c/200\text{MHz} = 1,5 \text{ m}$

Le linee si comportano come antenne: se sono di lunghezza pari a  $\lambda/4$ , sono in condizione di massima irradiazione. Calcoliamo una lunghezza  $L_{NON \text{ TX}}$  tale per cui non si presentino problemi, ovvero discostandoci del 90% dalla  $L_{CRITICA}$ .

a)  $L_{CRITICA} = \lambda/4 = 75 \text{ cm} \rightarrow L_{CRITICA} = 75 \cdot 10\% = 7,5 \text{ cm}$

b)  $L_{CRITICA} = \lambda/4 = 37,5 \text{ cm} \rightarrow L_{CRITICA} = 37,5 \cdot 10\% = 3,75 \text{ cm}$

La lunghezza fisica del bus è fortemente vincolata dall'FSB.

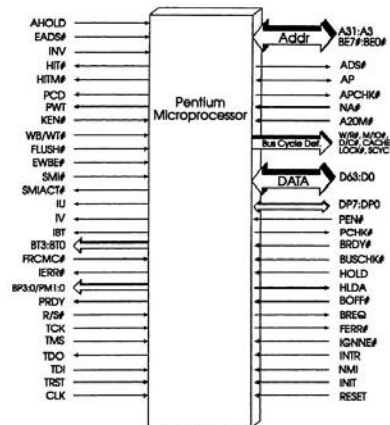
## Tipologie di Ciclo di BUS

Vi sono 4 principali tipologie di ciclo di BUS:

- Ciclo di READ
  - $T_1$ : Address su Address BUS
  - $T_2$ : CPU setta il Data BUS in Z-state
  - $T_3, T_4$ : la Memoria scrive Data su Data BUS
- Ciclo di WRITE
  - $T_1$ : Address su Address BUS
  - $T_2$ : CPU scrive Data su Data BUS
  - $T_3, T_4$ : la Memoria legge Data su Data BUS
- Ciclo di IDLE: introdotto dalla CPU quando essa non necessita di Data e non può essere eseguito Prefetch
- Ciclo di WAIT: necessario per operazioni lente; viene introdotto effettuando una segnalazione fra  $T_3$  e  $T_4$ . Il dispositivo “lento” informerà della conclusione dell’operazione con un segnale (nell’8086, con READY)

Verranno analizzati i cicli di BUS specifici per le memorie DRAM più avanti.

## Segnali e temporizzazioni del Pentium



I segnali più importanti del Bus Cycle Description sono:

- **CACHE#**: se attivo insieme a **KEN#**, un read/fetching cycle viene esteso a CACHE Line Fill cycle altrimenti verrà solo avviato un Burst Mode cycle
- **KEN#**: segnala se il ciclo corrente è cacheable (converte quindi un Burst Mode cycle in un CACHE Line Fill); gli I/O cycle non vengono mai cacheati
- **BRDY#**: segnala se il device indirizzato ha completato l'accesso richiesto in Burst Mode
- **R/W#**: indica se il ciclo di BUS è di Read o Write
- **C/D#**: indica se le informazioni coinvolte nel ciclo di BUS sono di tipo Code o Data
- **IO/M#**: indica se il ciclo di BUS fa riferimento a dispositivi di I/O o a Memoria
- **LOCK#**: indica che nel corrente ciclo di BUS, il BUS non può essere rilasciato

I segnali necessari per la comunicazione al DMA Controller sono:

- **HOLD**: viene attivato se un dispositivo vuole utilizzare il bus in DMA
- **HLDA**: viene attivato dalla CPU dopo aver messo in Z-state l'interfaccia all'Address BUS e i segnali di controllo

I segnali necessari per l'implementazione dell'Interrupt Controller sono:

- **INTR(in)**: Interrupt Request da un dispositivo esterno
- **INTA(out)**: Interrupt Acknowledgement da parte della CPU e temporizzazione del trasferimento del codice di Interrupt
- **NMI(in)**: Not-Maskable Interrupt

I segnali necessari per l'Address BUS sono:

- **ADS#**: Address Strobe, segnala durante  $T_1$  che un indirizzo è pronto sull'Address BUS
- **A31÷A2**: 29 linee che identificano una QuadWord (nei Pentium II e successivi, 36 linee)
- **BE<sub>n</sub>÷BE<sub>0</sub>#**: Byte Enable, dove n è il numero di Byte di parallelismo del Data Bus (per il Pentium è pari a 7, dato che il BUS ha parallelismo 64bit)

Instruction	Address h	BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	Transfer Type
MOV AL,[0100]	00000100	1	1	1	1	1	1	1	0	8-bit transfer
MOV AL,[0101]	00000100	1	1	1	1	1	1	0	1	8-bit transfer
MOV BL,[0102]	00000100	1	1	1	1	1	0	1	1	8-bit transfer
MOV AH,[0103]	00000100	1	1	1	1	0	1	1	1	8-bit transfer
MOV AL,[0104]	00000100	1	1	1	0	1	1	1	1	8-bit transfer
MOV AL,[0105]	00000100	1	1	0	1	1	1	1	1	8-bit transfer
MOV BL,[0106]	00000100	1	0	1	1	1	1	1	1	8-bit transfer
MOV AH,[0107]	00000100	0	1	1	1	1	1	1	1	8-bit transfer
MOV AX,[0100]	00000100	1	1	1	1	1	1	0	0	16-bit transfer
MOV AX,[0102]	00000100	1	1	1	1	0	0	1	1	16-bit transfer

I segnali necessari per il Data BUS sono:

- **READY**: segnale di sync con l'esterno; all'istruzione WAIT, la CPU testa tale segnale: se attivo, introduce cicli di IDLE, se inattivo, esegue l'istruzione successiva alla WAIT
- **D63÷D0**: 64 linee di Data BUS (dopo il Pentium II, 128 o 256 linee)

- DP7÷DP0: 8 linee di Parity per il Data BUS (1 bit ogni Byte)

I segnali per segnalazioni di CACHE:

- HIT#: viene attivato se si verifica un HIT in una CACHE Line
- HITM#: viene attivato se si verifica una HIT in una CACHE Line modificata
- PHIT#: private HIT#, per SMP
- PHITM#: private HITM#, per SMP



## Pentium Burst Cycle

I Burst Cycle sono principalmente utilizzati nei casi di CACHE Line Fill o Write Back da CACHE a Memoria. Più informazioni riguardo la politica di CACHING [qui](#).

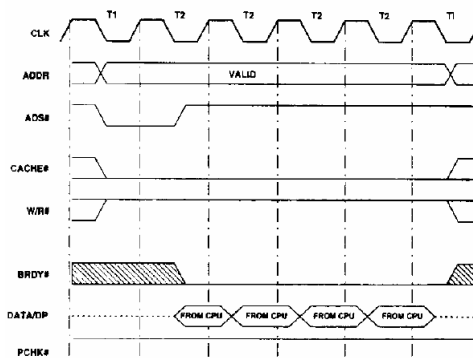
Dal punto di vista progettuale vi sono 2 principali parametri:

- In # di Byte di parallelismo della CACHE Line: più è alto, più è efficace il principio di località, ma aumenta anche il tempo di inattività della CPU
- Il  $T_{\text{CACHE LINE FILL}} = \#_{\text{CYCLE}} \cdot T_{\text{BUS CYCLE}}$  dove  $\#_{\text{CYCLE}}$  è stato fissato a 4, considerando però di aumentare il parallelismo del Data BUS

La CACHE effettua una copia di Byte ADIACENTI dalla Memoria: ciò presuppone una scansione ove gli indirizzi sono contigui e facilmente generabili dalla BIU.

$4 \cdot T_{\text{BUS CYCLE}}$			
$2 \cdot T_{\text{CLK}}$	$T_{\text{CLK}}$	$T_{\text{CLK}}$	$T_{\text{CLK}}$
I	II	III	IV
2	1	1	1

A  $\frac{3}{4}$  del I Ciclo di BUS, il Burst viene segnalato via BRDY#



Per tali tempistiche, sono necessarie memorie Fast Operative.

Il Burst Cycle viene avviato quando i segnali Bus Cycle Description sono in determinate configurazioni (CACHE# è attivo) ed il segnale KEN# è attivo.

I Write-Back possono avvenire per 3 motivi:

- Preservare la consistenza dei dati in CACHE
  - Uno SNOOPING esterno rileva un HIT in una Line modificata nella CACHE interna
  - Uno SNOOPING interno rileva un HIT in una Line modificata nella CACHE interna
- Per non perdere le modifiche effettuate sui dati
  - Una Line modificata deve essere rimossa per fare spazio ad una nuova CACHE Line



# Sottosistema di Memoria

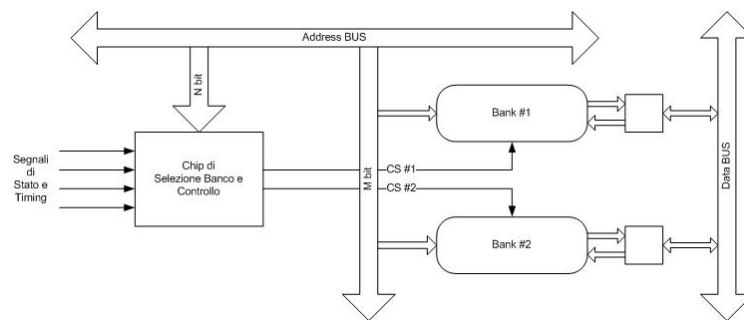
## Introduzione al sistema di memorie e problematiche

Le memorie lavorano solitamente ad una frequenza differente da quella dell'Host BUS. Per far sì che gli accessi in memoria non rallentino il sistema, sono state adottate, nel tempo, soluzioni:

- Tecnologiche, con le memorie Fast Operative Mode
- Architetture, con l'Interleaving

Inoltre, vi sono altre problematiche, proprie delle memorie dinamiche, quali:

- Il Refresh
- La Rilevazione e/o Correzione degli errori



## Tipologie di DRAM cycle

Vi sono 4 tipi di cicli DRAM:

- Read
  - Con Address Pipelining
  - Senza Address Pipelining
- Write
  - Con Address Pipelining
  - Senza Address Pipelining
- Refresh: necessari a causa della tecnologia delle memorie, perché le celle di memoria mantengono le informazioni sotto forma di cariche elettriche all'interno di "capacità parassite" e sono soggette a deteriorare nel tempo, a meno di non essere "rinfrescate"
  - RAS-only: il CAS# è inattivo, il RAS# è attivo, l'Address BUS contiene un indirizzo di riga; tale lettura "dummy" (non viene riversata sul Data BUS), va a ri-amplificare il contenuto delle celle
  - CAS-before-RAS: le memorie devono integrare una circuiteria di refresh e un Address Counter interno (per la generazione di indirizzi); viene attivato prima il CAS# e poi il RAS#, l'Address BUS è indeterminato/non valido
  - Hidden: è un ciclo di Refresh nascosto perché avviene successivamente ad una Read, lasciando il CAS# attivo
- Fast Operative
  - Page Mode: il RAS# rimane attivo, viene switchato solo il CAS# (in concomitanza con l'aggiornamento dell'Address BUS (la sola Column))
  - Hyper Page Mode (EDO): sostanzialmente molto simile al Page Mode, ma il CAS# cambia stato più velocemente (e conseguentemente anche l'indirizzo di Colonna), aumentando il throughput
  - Static-Column Mode: simile all'Hyper Page Mode, ma il CAS# rimane attivato tutto il tempo necessario per scansionare la riga, vengono aggiornati solo gli indirizzi di Colonna (la memoria ha un circuito di rilevazione di cambiamento Column Address)

L'Interleaving consiste nella scelta di realizzare le memorie suddividendole in banchi di eguale parallelismo e mappando gli indirizzi ADIACENTI sequenzialmente tra i banchi. Ad esempio, scegliendo di realizzare la memoria con 2 banchi, è possibile assegnare al Bank #0 tutti gli indirizzi pari e al Bank #1 i dispari. In questo modo, il RAS precharge time di un banco si sovrappone al tempo di accesso dell'altro banco (un banco viene precaricato mentre l'altro è acceduto e vice-versa).

## Esempio di progettazione di una memoria

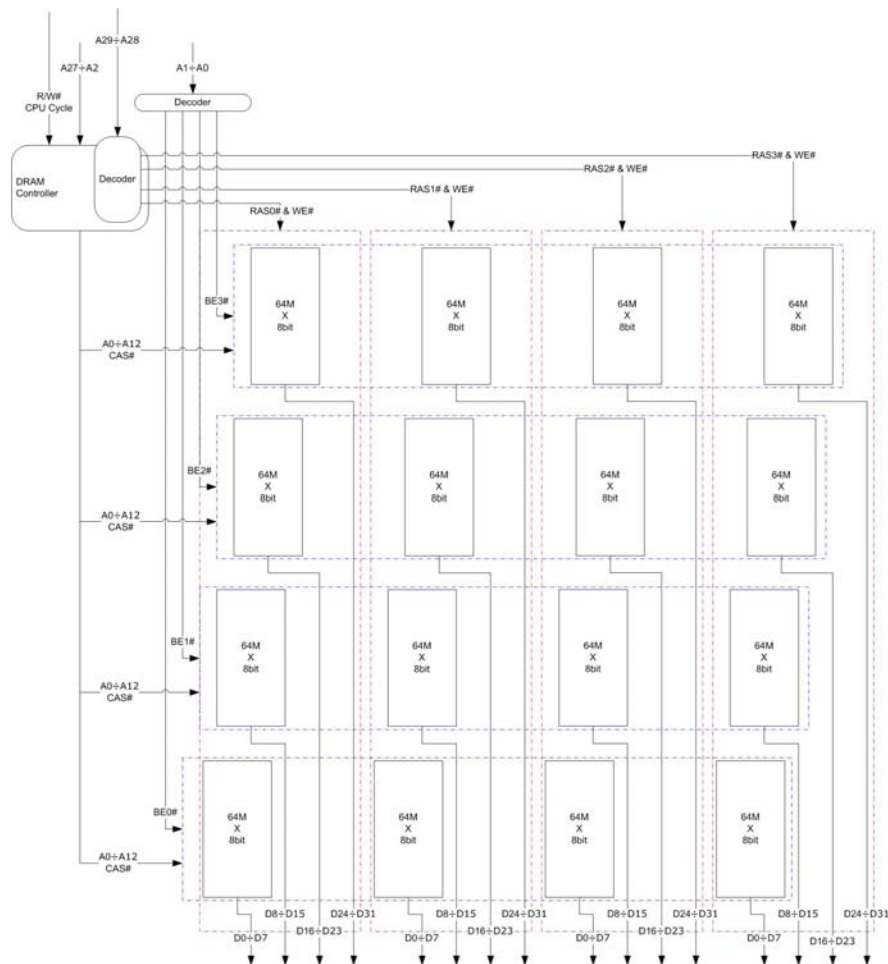
Data BUS a 32bit

Address BUS a 32bit +  $BE0 \div 3$

Chip da 512Mbit ( $64Mword \cdot 8bit/word$ ,  $2^{26} linee \cdot 1Byte$ )

Non considerare la rilevazione errori

Memoria totale: 1GByte



Il Pentium dell'esempio ha un Data BUS a 32bit e 4 linee per il Byte Enable ( $BE3\# \div BE0\#$ ).

I BE vengono codificati su 2 bit dell'Address BUS ( $A1 \div A0$ ):

- $BE0\#$  viene collegato a tutti e 4 i chip contenenti il Byte meno significativo ( $D0 \div D7$ ): riquadrati in blu, in basso
- ...
- $BE3\#$  viene collegato a tutti e 4 i chip contenenti il Byte più significativo ( $D24 \div D31$ ): riquadrati in blu, in alto

Le linee  $A29 \div A28$  vengono utilizzate per la selezione del banco (ogni banco è composto da 4 Chip da  $64Mword \cdot 8bit/word$ , per un totale di 256MByte) e quindi decodificati in 4 segnali di RAS:  $RAS3\# \div RAS0\#$ .

Le linee  $A27 \div A2$  (26bit) vengono multiplexate dai segnali  $RASx\#$  e  $CAS\#$  e quindi tutti i chip  $64M \cdot 8bit$  verranno collegati a un sotto-BUS a 13linee.

## Controlli di errore su Memoria

Gli errori possono essere rilevati/corretti a seconda della tecnologia utilizzata:

- Parity Error Check: 1bit/Byte di memorizzazione aggiuntiva richiesta, rileva gli errori di parità; necessita di un circuito per il calcolo della parità e di una porta XOR per il controllo al riversamento sul Data BUS
- ECC (Error Correcting Code, basato su codice di Huffman): 8bit/8Byte di memorizzazione aggiuntiva richiesta, per la correzione di errori singoli, rilevazione di doppi e alcuni multipli; comporta una diminuzione delle prestazioni quantificabile al 2% e necessita di circuiti di calcolo e porta XOR

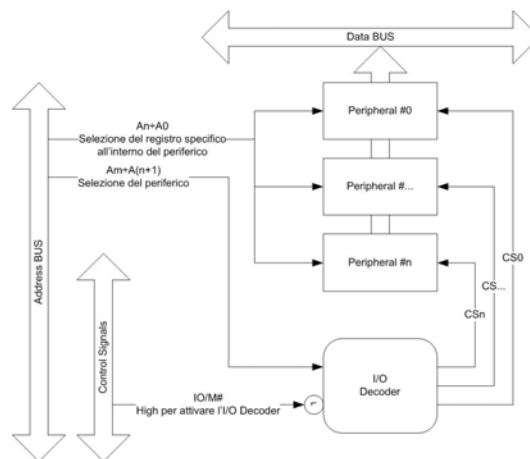
## Sottosistema di I/O

Esistono 2 principali tipologie di accesso ai periferici da parte della CPU:

- Memory Mapped
- Isolated I/O

Nel caso di Isolated I/O, i dispositivi hanno un loro spazio d'indirizzamento, che non può essere però acceduto attraverso le normali istruzioni di accesso a memoria.

La CPU va a "contatto" diretto con i registri di tali dispositivi. Le istruzioni assembly IN e OUT permettono di eseguire rispettivamente lettura e scrittura dei registri del periferico, accedendo attraverso delle "porte". Lo spazio d'indirizzamento della memoria non viene quindi ridotto dai periferici.



Nel caso Memory Mapped, gli indirizzi dei periferici sono mappati all'interno dello spazio d'indirizzamento della memoria (perciò vi è una riduzione dello spazio d'indirizzamento assegnabile alla memoria fisica).

I registri dei periferici sono accessibili attraverso le normali istruzioni per l'accesso in memoria. Gli indirizzi occupati dai devices devono essere conosciuti a priori dal sistema di indirizzamento.

## Gestione degli I/O in interrupt

A seconda del tipo di trasferimento dati, i periferici possono essere classificati in:

- Periferici a carattere (predomina il fattore  $T_{\text{ACCESSO}}$ )
- Periferici a blocco (i fattori principali sono il  $T_{\text{ACCESSO}} + T_{\text{TRASFERIMENTO}}$ )

Una richiesta di interrupt può essere accolta in:

- Polling: un ciclo software scansiona tutti i dispositivi, interrogandoli; è semplice ma vi è una occupazione della CPU per il solo ciclo di polling, inoltre la latenza è elevata
- Interrupt: un periferico che necessita di un servizio attiva la richiesta; la CPU, terminata l'esecuzione dell'istruzione corrente serve il periferico. In questo caso un circuito apposito accoglie le richieste (l'interrupt controller) e la procedura di servizio viene lanciata dall'interrupt handler

Focalizzeremo l'attenzione sull'interrupt centralizzato.

Dal punto di vista delle tempistiche, vi sono 2 principali fattori da considerare per l'I/O in interrupt:

- $T_{\text{LATENZA}}$ : è il tempo che intercorre fra la richiesta di interruzione e il trasferimento del dato da/per l'I/O device. Considerando il worst case, il  $T_{\text{LATENZA MAX}}$  è dato da
  - $T_{\text{IC}}$ : tempo per il completamento dell'istruzione corrente
  - $T_{\text{AR}}$ : tempo di attivazione della routine di interrupt, determinata dall'hardware
  - $T_{\text{AT}}$ : tempo di avvio trasferimento; è il tempo che intercorre fra l'inizio della routine e l'istruzione di trasferimento dati
- Transfer Rate (TR): deve risultare compatibile con il  $T_{\text{LATENZA}}$ , ovvero  $1/\text{TR} \geq T_{\text{LATENZA}}$

Le priorità di richiesta per un I/O gestito in interrupt viene attribuita di norma al periferico con TR più alto e, conseguentemente, con minima latenza. In questo modo i periferici più lenti possono essere interrotti da quelli più veloci.

L'80x86 implementa 256 tipi di interrupt, suddivisi in 4 gruppi, elencati in ordine decrescente per priorità:

- Interrupt interni (Trap): eccezioni interne del processore; il numero a loro assegnato non è modificabile (priorità massima, parte bassa della IVT)
- Interrupt non mascherabili (NMI): il numero a loro assegnato non è modificabile
- Interrupt software: per la gestione di servizi dell'O.S.
- Interrupt hardware esterno: dipendenti dall'architettura, hanno priorità minima (e quindi numero assegnato è nella parte alta della IVT)

Gli interrupt con priorità elevata possono interrompere quelli di priorità inferiore.

Questo tipo di gestione di interrupt fa affidamento all'Interrupt Vector Table: essa contiene 256 entry (da 4 Byte in Real mode, 8 Byte in Protected mode).

Essa è situata nella parte bassa della memoria (indirizzi 00000h÷003FEh, riposizionabile in Protected mode).

In Real mode, le entry contengono:

- 2 Byte che contengono l'offset (IP) della procedura di interrupt
- 2 Byte che contengono il segmento di codice (CS) all'interno del quale è situata la procedura

In Protected mode, le entry contengono:

- 2 Byte di offset (parte H)
- Flags (Segment Present, DPL o Kernel Ring, Type, Padding)
- Segment Selector
- 2 Byte di offset (parte L)

Le Trap sono interrupt riservati per situazioni anomale nella gestione dei processi interni del processore. Sono differenziati in base alle modalità. Alcuni di questi sono:

- Divisione per Zero (0)
- Single step (1): utilizzato per l'implementazione della modalità di Debug Step by Step; in Protected mode, se TF è TRUE, ad ogni istruzione viene lanciato tale interrupt
- NMI (2)
- Breakpoint (3): utilizzato per l'implementazione della modalità di Debug Breakpoint; viene utilizzata l'istruzione INT 3, perché viene codificata su un solo Byte (questo per fare sì che le istruzioni successive non vengano sovrascritte)
- Overflow (4): l'istruzione INTO, se OF è TRUE, causa l'attivazione della routine all'IVT (indirizzo 00010h)
- Riservati (5÷31): vettori per Protected mode o riservati
  - Invalid OpCode (6)
  - Segment not present (11)
  - Page fault (14)
- Bios e O.S. (32÷255)

Gli interrupt non mascherabili (NMI) sono positive-edge-triggered (NMI deve essere attivo per 2  $T_{CLK}$ ) e ad essi è assegnato il vettore di interrupt 2.

Le istruzioni assembly STI e CLI rispettivamente abilitano e disabilitano gli interrupt mascherabili. Il flag IF viene automaticamente resettato all'attivazione della procedura di interrupt.

Il protocollo di interrupt prevede le seguenti fasi:

- i. Un dispositivo invia una richiesta su INTR
- ii. La CPU rileva, durante l'ultimo TCLK dell'istruzione in corso di esecuzione, l'INTR attivo
- iii. La CPU invia un primo acknowledgement su INTA
- iv. La CPU invia un secondo INTA, richiedendo che il numero di INT richiesto sia riversato sul Data BUS
- v. La CPU legge 1 Byte dal Data BUS, ottenendo così il numero di INT richiesto
- vi. La CPU esegue il salvataggio in stack della PSW, IP e CS
- vii. IF e TF vengono disabilitati
- viii. La CPU accede all'IVT all'entry pari al numero di INT (indirizzo #INT·4)
- ix. Ad IP e CS vengono assegnati i valori contenuti nella entry della IVT

## PC Interrupts e APIC

### 8259A

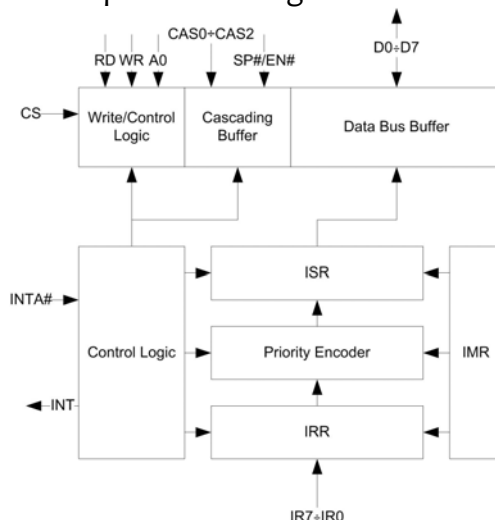
La gestione degli interrupt è affiancata in hardware da un dispositivo, l'8259. Esso è stato progettato per:

- Minimizzare il codice
- Minimizzare i tempi di risposta

Per la gestione di livelli di interrupt multipli con priorità.

Nei sistemi attuali, l'8259 è integrato nel SouthBridge, mentre in origine era un chip.

Ogni 8259 è capace di gestire 8 livelli di interrupt vettorizzati. E' possibile ampliare il numero di interrupt gestibili collegando più 8259 in cascata (1 Master ed 1 o più Slave) fino ad un massimo di 64 livelli. Il Master controlla gli Slave attraverso dei segnali di CAS che operano da CS codificato per gli Slave durante la sequenza di interrupt acknowledgement.



Un periferico effettua un Interrupt Request sulla linea ad esso assegnata (IR7-IR0). Il bit dell'IRR (Interrupt Request Register) associato viene settato. Viene inviata un segnale di INT alla CPU.

Si avvia la Interrupt Acknowledgement sequence: la CPU riceve la richiesta e al termine dell'istruzione corrente invia un INTA all'8259. Il Priority Encoder, rispettando l'IMR (Interrupt Mask Register), passa all'ISR il bit a più alta priorità e resetta il bit corrispondente dell'IRR.

La CPU invia un secondo INTA, richiedendo all'8259 di trasferire il numero corrispondente al bit settato dell'ISR sul Data BUS Buffer: il numero andrà ad occupare i 3 LSB del Byte, i bit restanti possono essere programmati.

La CPU legge il Byte dal Data BUS.

A questo punto, se l'8259 opera:

- In AEOI (Automatic End of Interrupt), l'acknowledgement sequence è terminata ed il bit dell'ISR viene resettato
- In EOI, è responsabilità dell'interrupt handler far inviare dalla CPU un EOI esplicito: l'ISR verrà resettato manualmente



## Programmazione dell'8259A

Il chip 8259A deve essere programmato per operare nella modalità voluta.

In particolare, esso necessita di:

- 4 Initialization Command Words (ICW) per l'inizializzazione
- 3 Operation Command Words (OCW) per selezionare la modalità e le opzioni di funzionamento volute

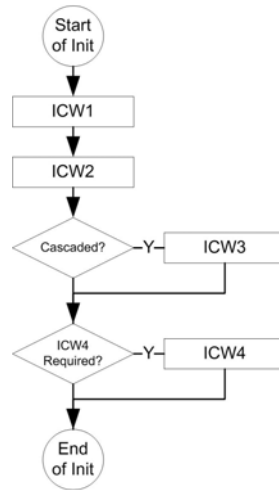
### Initialization Command Words

	A0	D7	D6	D5	D4	D3	D2	D1	D0
ICW1	0	0	0	0	1	LTIM	0	SNGL	IC4
ICW2	1	Off7	Off6	Off5	Off4	Off3	0	0	0
ICW3 <sub>MASTER</sub>	1	S7	S6	S5	S4	S3	S2	S1	S0
ICW3 <sub>SLAVE</sub>	1	0	0	0	0	0	ID2	ID1	ID0
ICW4	1	0	0	0	SFNL	BUF	M/S	AEOL	μPM

Analizziamo i bit di inizializzazione delle ICWs:

- LTIM: Level Triggered Mode
  - 0 per Edge Triggering
  - 1 per Level Triggering
- SNGL: Single Mode
  - 0 per Cascaded
  - 1 per Master Only
- IC4: ICW4 needed
  - 0 se non presente
  - 1 se presente
- Off7÷Off3: Programmable Interrupt Vector Offset
- S7÷S0: Slave o Peripheral
  - Sn a 0 per periferico connesso su IRn
  - Sn a 1 per 8259 Slave connesso su IRn
- ID2÷ID0: Slave 8259 Identification Number
- SFNL: Special Fully Nested Level Mode
  - 0 per No SFNL Mode
  - 1 per SFNL Mode
- BUF: Buffered Mode, se attiva SP#/EN# diventa un pin di output che abilita i buffer transreceivers quando l'8259A effettua operazioni di scrittura su Data BUS
  - 0 per No Buffered Mode
  - 1 per Buffered Mode
- M/S: Buffered Master or Slave 8259 Flag
  - 0 per Buffered Mode Slave (solo se Buffered Mode)
  - 1 per Buffered Master (solo se Buffered Mode)
- AEOL: Automatic EOI
  - 0 per manual EOI
  - 1 per automatic EOI
- μPM: Microprocessor Mode
  - 0 per MCS-80/85
  - 1 per 8086/88

La sequenza di inizializzazione è composta dalle seguenti fasi:



## Operation Command Words

	A0	D7	D6	D5	D4	D3	D2	D1	D0
<b>OCW1</b>	1	M7	M6	M5	M4	M3	M2	M1	M0
<b>OCW2</b>	0	R	SL	EOI	0	0	L2	L1	L0
<b>OCW3</b>	0	0	ESMM	SMM	0	1	P	RR	RIS

Analizziamo i bit delle OCWs:

- M7÷M0: Masked IR# (La OCW1 è una “copia” della IMR)
  - Mn a 0 per mascherare l'IRn
  - Mn a 1 per mascherare l'IRn
- R, SL, EOI agiscono a seconda della loro configurazione complessiva:
  - 000: Rotate in AEOI Mode
  - 001: Non-Specific EOI
  - 010: NOP
  - 011: Specific EOI (richiede L2÷L0 per la specifica del canale)
  - 100: Rotate in AEOI Set Mode
  - 101: Rotate for Non-specific EOI
  - 110: Set Priority Command
  - 111: Rotate for Specific EOI
- ESMM, SMM operano a seconda della loro configurazione:
  - 00: NOP
  - 01: NOP
  - 10: Clear Specific Mask
  - 11: Set Specific Mask
- RR, RIS operano a seconda della loro configurazione:
  - 00: NOP
  - 01: NOP
  - 10: Read IRR
  - 11: Read ISR
- P: Polling Mode
  - 0 per No Polling
  - 1 per Polling Mode

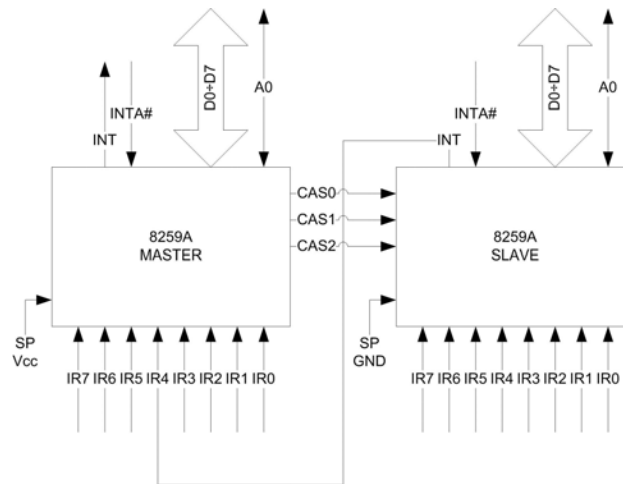
## Modalità di funzionamento

E' possibile elencare una serie di modalità di funzionamento:

- Fully Nested Mode: è la modalità di default per l'8259A. Le richieste di interruzione sono ordinate da 0 a 7 (0 ha priorità maggiore). Il Priority Encoder, ricevuto il primo INTA, calcola la richiesta a priorità più alta e setta il corrispondente bit del registro ISR. Fintanto che il bit rimane settato, tutte le richieste a priorità inferiore sono disabilitate.
  - AEOI: il bit in ISR è resettato automaticamente dopo il fronte di salita del secondo INTA inviato dalla CPU
  - EOI: il bit in ISR rimane settato finchè la CPU invia un comando EOI esplicito (lanciato prima di ritornare dalla routine di servizio della interruzione)
    - Specific EOI: (in modalità diverse da Fully Nested Mode) necessario in tutti i casi nei quali non c'è modo di riconoscere qual è l'ultimo livello di interruzione abilitato
    - Non-Specific EOI: (solo in priorità statiche) viene resettato il bit a priorità più alta del registro ISR (ultimo livello di int abilitato e servito)
- Rotazione di priorità: viene utilizzato nei casi in cui i dispositivi di I/O hanno eguale priorità. In pratica, il dispositivo n appena servito riceve la priorità più bassa ( $\text{Priorità}(\text{IR}[n]) = 7$ ,  $\text{Priorità}(\text{IR}[n+1]) = 0$ )
  - Rotazione di priorità Specifica: è possibile cambiare i livelli di priorità specificando il livello di priorità più basso mediante la OCW2
- Special Mask Mode: in questa modalità, i bit settati di IMR disabilitano il livello  $\text{IR}_n$  corrispondente e abilitano tutti gli altri livelli. Tale modalità è utile quando in modo EOI una richiesta è stata abilitata e il comando esplicito EOI non ha disabilitato IS
- Special Fully Nested Mode: è la modalità utilizzata quando gli 8259A sono connessi in cascata e si vuole conservare la priorità all'interno di ciascuno slave. Uscendo dalla routine di servizio, prima di inviare i comandi EOI al Master, occorre inviare un EOI non specifico allo Slave, leggere l'ISR dello Slave: se ISR è zero, allora si può inviare il secondo EOI al Master.

## Cascaded 8259As

Andiamo a descrivere il protocollo utilizzato nel caso di più 8259A collegati in cascata. A titolo d'esempio, abbiamo la seguente connessione fra chip.



Uno dei 2 8259A riceve un Interrupt Request:

- Se si tratta di uno Slave, l'IRn dello Slave (se a massima priorità locale e non mascherato) e conseguentemente il Priority Encoder genera il segnale di INT, il quale è girato all'IRn del Master (IR4)
- Se l'IRn è a massima priorità e non è mascherata, viene inviato un INTR alla CPU dal Master
- La CPU risponde con il primo INTA.
- L'8259A Master setta il bit della ISR e resetta il bit della IRR e controlla se la richiesta proviene da uno Slave
  - Se non proviene da uno Slave, il Master agisce come in modalità singola
  - Se proviene da uno Slave, il numero corrispondente al bit di interrupt della ISR viene codificato e inviato su CAS2÷CAS0
    - Il secondo INTA viene ricevuto da tutti gli Slave
    - Ogni Slave controlla se il suo ID è pari al valore su CAS2÷CAS0
    - Lo Slave selezionato setta l'opportuno bit su ISR e resetta il bit su IRR, inviando poi il numero del bit di interrupt sul Data BUS

In un sistema Master/Slave è necessario inviare 2 EOI (uno per il Master e uno per lo Slave).

E' possibile leggere il contenuto dei registri:

- IRR, inviando una OCW3 (RR,RIS pari a 10b) ed effettuando la lettura
- ISR, inviando una OCW3 (RR,RIS pari a 11b) ed effettuando la lettura
- IMR effettuando la lettura con A0 pari ad 1, senza alcuna OCW3

## Interrupt nei sistemi PC

Nei sistemi PC gli interrupt mascherabili sono gestiti da 2 8259 (o compatibili) sotto forma di chip o integrati nel chipset.

Sono quindi gestibili 15 livelli di interrupt (7 sul Master e 8 sullo Slave).

Vi sono poi gli interrupt non mascherabili che sono collegati direttamente alla CPU via NMI. Su tale pin sono raccolti in OR gli interrupt scatenati da:

- Parity Check della memoria dinamica
- BUS ISA
- Il chip 8253

Tali interrupt sono in AND con una bit mask.

Gli 8259 non devono essere programmati (sono programmati all'avvio dal BIOS).

Durante il normale funzionamento, si “accede” a questi dispositivi per:

- Abilitare/Disabilitare un determinato canale di interrupt
- Inviare un Non-Specific EOI

Nei sistemi multiprocessore è necessario che ogni singola CPU abbia una APIC Locale. Tutte le CPU comunicheranno con un I/O APIC che si prende carico di gestire le priorità e sincronizzare le varie richieste di interrupt, distribuendole verso le CPU meno occupate.

# Chip 8253 – Temporizzatore di Intervalli

## Generalità

L'[8253](#) svolge funzioni di temporizzazione e conteggio. In particolare, esso può:

- Generare onde quadre o impulsi ad una frequenza programmabile
- Comportarsi da contatore (binario o BCD)
- Comportarsi da divisore di frequenza
- Comportarsi da misuratore di intervalli di tempo
- Generare ritardi sotto controllo software

Al suo interno ci sono 3 contatori da 16 bit che possono, ognuno indipendentemente dall'altro:

- Essere letti o caricati in modo software
- Conteggiare a ritroso
- Comportarsi da contatori binari o BCD

La programmazione è effettuata mediante la parola di controllo, passata al Registro di Controllo. Tale parola definisce:

- Le modalità di funzionamento
- Modalità di caricamento del valore iniziale
- Tipologia di conteggio

SC1	SC0	RL1	RL0	M2	M1	M0	BCD
<b>Selected Counter</b>		<b>Read/Load</b>		<b>Mode</b>			<b>Binary/BCD</b>
00 per Counter0		00 per Counter Latch		000, Modo 0			0 per binario
01 per Counter1		01 per Read/Load LSB-only		001, Modo 1			1 per BCD
10 per Counter2		10 per Read/Load MSB-only		010, Modo 2			
		11 per Read/Load LSB then MSB		x11, Modo 3			
				100, Modo 4			
				101, Modo 5			

Le modalità di funzionamento sono:

- Modo 0: Interrupt a termine conteggio
- Modo 1: One-Shot programmable
- Modo 2: Generatore di frequenza
- Modo 3: Generatore di onde quadre
- Modo 4: Software Triggered Strobe
- Modo 5: Hardware Triggered Strobe

Focalizziamo l'attenzione sulle modalità 0 e 2.

**Mode 0: Interrupt a termine conteggio**

In questa modalità, una volta caricato il valore iniziale, il conteggio parte.

Durante il conteggio, il pin OUT resta basso. Al termine passa al livello alto e permane in quello stato.

Il pin GATE abilita il conteggio se a livello alto.

**Mode 2: Generatore di frequenza**

In questa modalità, il 8253 opera come divisore di frequenza.

Una volta caricato il valore  $n$ , ogni  $n$  conteggi il segnale OUT passa al livello basso per un ciclo. Il conteggio parte al momento del caricamento.

GATE, se a livello basso, fissa OUT a livello alto. Il successivo fronte fa ripartire il conteggio.



## Chip 8255 – Interfaccia parallela

### Generalità

L'[8255](#) è un dispositivo hardware programmabile che permette di effettuare trasferimenti su interfacce parallele di bit, nibble e Byte.

Offre 3 porte di I/O indipendenti da 1 Byte ciascuna.

Le porte sono divise principalmente in 2 Gruppi:

- Group A
  - Port A, 8bit
  - Port C (upper), 4bit
- Group B
  - Port C (lower), 4bit
  - Port B, 8bit

Dal punto di vista logico, l'8255 mette a disposizione:

- 3 Registri di Porta (A,B e C)
- 1 Registro di Controllo, write-only per definire i modi di funzionamento od effettuare operazioni su singoli bit della porta C

Tali registri sono accessibili attraverso D7÷D0 e selezionabili via A1÷A0.

Il Registro di Controllo è strutturato come segue:

1	GAM1	GAM0	Port A	Port C High	GAMB	Port B	Port C Low
	<b>Group A Mode</b> 00 per Mode 0 01 per Mode 1 1x per Mode 2		1 per Input 0 per Output	1 per Input 0 per Output	<b>Group B Mode</b> 0 per Mode 0 1 per Mode 1	0 per Output 1 per Input	0 per Output 1 per Input

## Modalità di funzionamento

Sono disponibili 3 modalità di funzionamento:

- Mode 0: Basic I/O (default, in Input); non è disponibile alcun supporto per l'handshake
- Mode 1: Strobed I/O; sono disponibili segnali di handshake e l'I/O è interrupt driven
- Mode 2: Bidirectional BUS; permette di utilizzare una porta in modo bidirezionale con segnali di handshake e interrupt

### Mode 0

In questa modalità sono disponibili 3 porte da 1 Byte ciascuna, programmabili indipendentemente in Input o Output.

### Mode 1

In questa modalità vi è supporto per segnali di interrupt ed handshake.

Le porte disponibili sono 2, ognuna da 8bit, supportata da 4 bit di controllo. Input ed Output sono latched.

In Input:

- $STB_A\#$  e  $STB_B\#$  (Strobe Input) caricano un dato in buffer quando passano a livello basso, attivando  $IBF_A$  e  $IBF_B$  (Interrupt Buffer Full).
- $IBF_A$  e  $IBF_B$  vengono resettati dal fronte di salita di RD
- $INTR_A$  e  $INTR_B$  (Interrupt Request) si attivano quando  $IBF$ ,  $INTE$  (Interrupt Enable) ed  $STB$  sono alti
- $INTR_A$  e  $INTR_B$  vengono resettati dal fronte di discesa di RD

In Output:

- $OBF_A\#$  e  $OBF_B\#$  passano a livello basso quando la CPU ha scritto sulla rispettiva porta (attivato dal fronte di salita di  $WR\#$ )
- $INTR_A$  e  $INTR_B$  (Interrupt Request) si attivano quando  $ACK$ ,  $INTE$  (Interrupt Enable) ed  $OBF$  sono alti e resettati sul fronte di discesa di  $WR\#$
- $ACK_A\#$  e  $ACK_B\#$  (Acknowledgement Input) passano a livello basso quando un dato è stato ricevuto dalla periferica

In tale modalità alcuni bit della Porta C sono utilizzati per l'invio di segnali di Interrupt alla CPU. Possono essere abilitati o disabilitati agendo con un'operazione di bit set/reset sulla porta C.

### Mode 2

La Mode 2 rende disponibile una porta di parallelismo 1 Byte con supporto handshake e interrupt. E' disponibile solo per il Gruppo A: la porta dati è la porta A, la porta di controllo è la porta C (a 5 bit). Input ed Output sono latched.

In tale modalità alcuni bit della Porta C sono utilizzati per l'invio di segnali di Interrupt alla CPU. Possono essere abilitati o disabilitati agendo con un'operazione di bit set/reset sulla porta C.

# Chip 8250– Interfaccia seriale

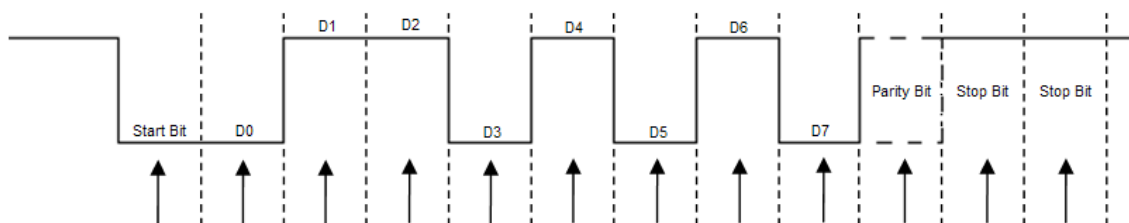
## Comunicazioni seriali

Le comunicazioni seriali vengono adottate quando la distanza fra trasmettitore e ricevitore è elevata. La trasmissione può essere misurata in bps (bit per second) o baud (transizioni al secondo). I dati trasmessi sono caratteri da n bit.

All'interno del dispositivo, il singolo carattere è memorizzato in forma parallela (solitamente in un buffer). Il dispositivo di controllo di trasmissione seriale deve:

- Eseguire una conversione parallelo/seriale in trasmissione
- Eseguire una conversione seriale/parallelo in ricezione
- Gestire la sincronizzazione (o fornire segnali per poterla gestire)
- Controllare eventuali errori di sincronizzazione o trasmissione

Il Frame di comunicazione è strutturato come nell'immagine sottostante.



Le frecce indicano gli istanti di campionamento.

## Funzionamento

Durante la programmazione si comunica il fattore di scalamento tra il clock interno e il clock di ricezione.

Per quanto concerne la trasmissione:

- Dalla prima transizione H→L, il ricevitore lascia passare K/2 colpi di clock per poi iniziare a campionare il segnale
- Se il valore campionato è L, esso è considerato bit di Start; in caso contrario vi è un errore di trasmissione
- Il ricevitore inizia a campionare ogni K colpi di clock
- Al termine dei dati carattere, devono essere letti anche il bit di Parità (se presente) e il/gli Stop Bit (se il numero di Stop Bit non è quello previsto, viene segnalato un errore di frame)

Solitamente si utilizzano 5÷8 bit per carattere. E' possibile prevedere un bit aggiuntivo per ciascun carattere per il controllo di parità.

Il tasso di trasmissione è dato da:

$$tasso_{TX} = \frac{bps_{CANALE}}{1(StopBit) + n(Bit_{CARATTERE}) + [1 \div 2](StopBit)}$$

## 8250

Il Chip [8250](#) è un dispositivo in grado di effettuare comunicazione seriale asincrona UART (Universal Asynchronous Receiver and Transmitter). Esso permette di generare internamente il bitrate nell'intervallo di valori previsto dallo standard RS-232.

Il fattore di scalamento base è 16. Supporta il riconoscimento degli errori.

La piedinatura prevede:

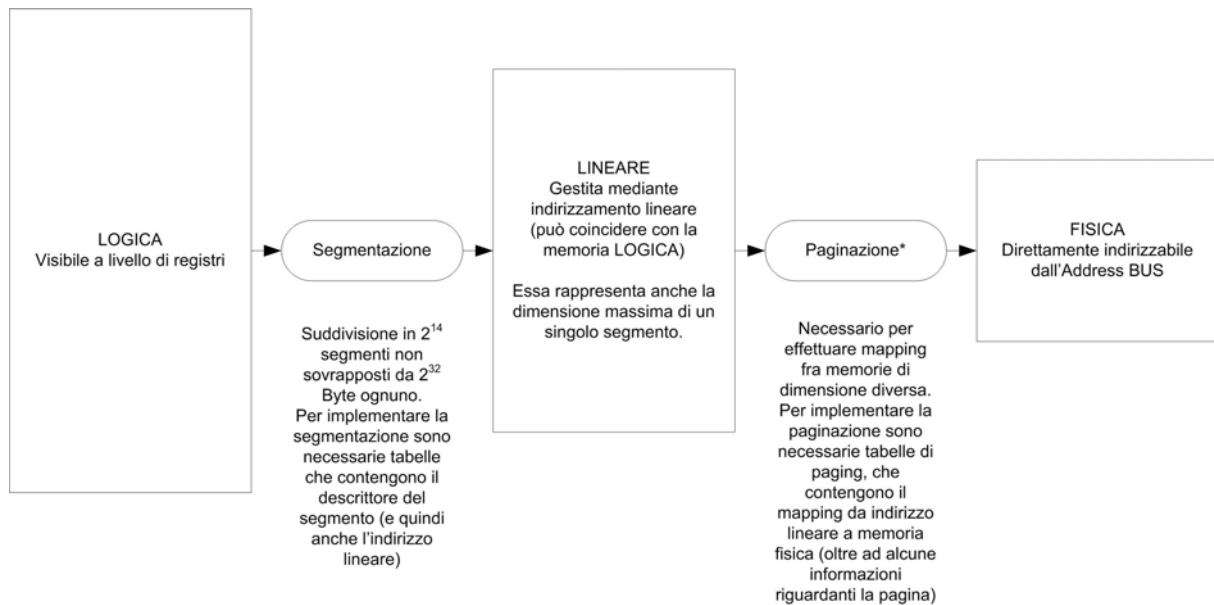
- A7÷A0 di Address BUS, che insieme a DLAB (un bit implementato nel Line Control Register) seleziona un singolo registro per accessi di lettura e/o scrittura
- D7÷D0 di Data BUS
- Pin di controllo (RD, WR, Select, Interrupt Request...)
- SIN, SOUT che stanno rispettivamente per Serial IN e Serial OUT
- Una serie di linee per la segnalazione di eventi asincroni

I registri interni del dispositivo sono 9. Ecco i più importanti:

- RBR (Receiver Buffer Register): contiene il dato ricevuto dalla Serial IN; il bit meno significativo è quello ricevuto per primo
- THR (Transmitting Holding Register): contiene il dato che deve essere trasmesso su Serial OUT; il bit meno significativo è quello che deve essere trasmesso per primo
- LSR (Line Status Register): contiene i bit di stato dell'interfaccia
  - FI: Full Input, RBR contiene il dato ricevuto
  - OE: Overrun Error, la CPU non ha letto il dato in RBR prima dell'arrivo del carattere seguente
  - PE: Parity Error
  - FE: Frame Error, generato dai Bit di Stop incorretti
  - B: Break, la linea è mantenuta a 0 per più tempo del necessario
  - EO: Empty Output, il THR deve essere caricato con un dato da trasmettere
- LCR (Line Control Register): permette la configurazione dell'interfaccia seriale
  - L2÷L1 per la selezione del numero di bit per carattere (5÷8 bit)
  - STOP per la selezione fra 1 e 1,5/2 bit di Stop (a seconda del numero di bit per carattere)
  - P, PS, SP: P abilita il bit di Parità, PS e SP indicano che tipo di parità o se il bit di Parità è fixed
  - BE: Break Enable, per abilitare eventi Break
  - DLAB: per l'accesso ai DLR
- DLR (Divisor Latch Register): contiene la “costante di tempo” per la definizione del bitrate di comunicazione. Il bitrate è pari a  $1.8432 \text{ MHz} / 16 \cdot \text{“costante di tempo”}$
- IER (Interrupt Enable Register): per la segnalazione di eventi in Interrupt, quali la ricezione di un dato in RBR, errori, richieste di caricamento dati in THR. Supportato da IIR (Interrupt Identification Register per identificare l'evento).

# Protected Mode (Segmentazione e Paginazione)

## Overview sulla memoria



Lo schema sovrastante esemplifica come viene eseguito il mapping attraverso i vari livelli di memoria. Andiamo ad analizzare le due tecniche di mapping in modalità protetta. Gli esempi e le informazioni si riferiscono ad un sistema a 32 bit.

## Segmentazione

La segmentazione in modo protetto prevede la suddivisione della memoria in  $2^{14}$  segmenti non sovrapposti da  $2^{32}$  Byte (4 GB) ognuno.

La segmentazione:

- Permette la gestione più facile di strutture dati anche complesse (magari di dimensioni non conosciute a priori): l'O.S. andrà a dimensionare il segmento a seconda della memoria necessaria
- Permette di modificare e ricompilare un singolo modulo di un programma senza dover re-linkare tutti i moduli del programma
- Permette di condividere un segmento
- Permette di implementare sistemi di protezione grazie ai privilegi

Per cui, ogni indirizzo è espresso su 46bit (spazio di indirizzamento pari a 64 TB):

- 16bit di Selettore di Segmento
  - 1 bit per il Table Indicator (GDT o LDT)
  - 2 bit per Requested Privilege Level
  - 13 bit per la selezione del segmento
- 32bit di Offset (all'interno del segmento)

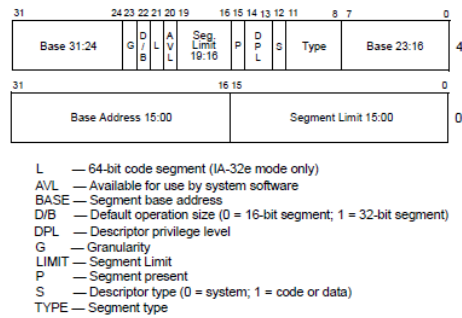
Tale mapping fra segmenti e memoria lineare è memorizzato in delle Descriptor Table:

- La GDT (Global Descriptor Table), di 8 KByte, è "riservata" all'O.S.
- La LDT (Local Descriptor Table), di 8 KByte, è disponibile all'utente; da notare che vi sono più local descriptor tables, ma non allo stesso tempo

Ogni entry della Descriptor Table contiene un Descrittore.

Il Descrittore di Segmento ha dimensione 8 Byte. Esso contiene:

- Base Address: indirizzo base del segmento nella memoria lineare (32 bit)
- Segment Limit: Byte effettivamente utilizzati, arrotondato ai 20 bit più significativi
- Gli attributi: informazioni riguardanti il segmento
  - Present: al momento dell'esecuzione di un programma, ogni segmento viene trasferito in un segmento della memoria. A seconda del tipo di programma, il descrittore verrà posizionato nella GDT o nella LDT. Alla prima istruzione, l'attributo present non è attivo e viene schedulato (arriva una TRAP dell'O.S.) l'allocatore che trasferisce il segmento di programma e lo posiziona nella memoria fisica.
  - DPL: viene utilizzato al momento della creazione dei descrittori. L'O.S. attua delle politiche per la gestione dei privilegi.
  - System: indica se il segmento contiene dati, codice o strutture di sistema
  - Execute: indica se si tratta di codice o dati
  - R/W: indica se il segmento può essere letto o scritto
  - Accessed: indica se il segmento è stato letto o scritto. Tale valore è modificato direttamente dalla ALU. E' necessario per implementare le tecniche di Swapping.
- Livello di privilegio: implementato con 2 bit
  - massimo (00b) ÷ minimo (11b)



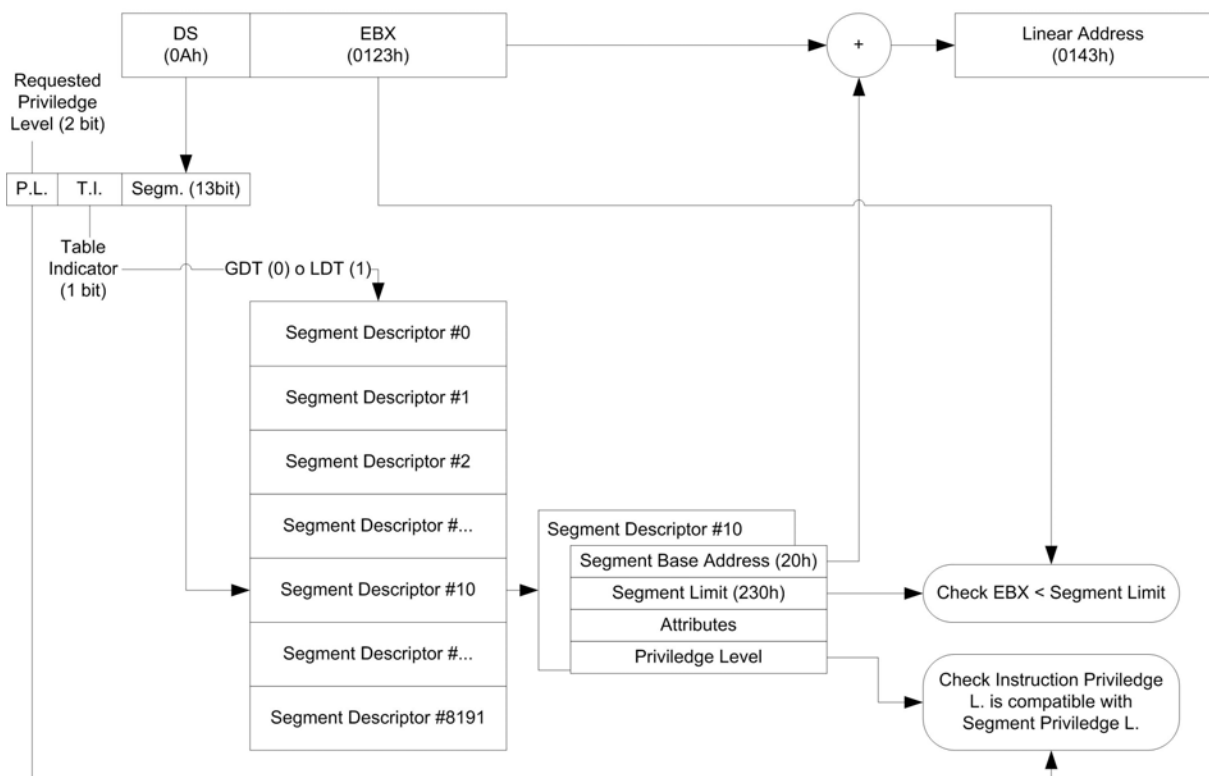
Tali informazioni vanno modificate in esecuzione: non è possibile riferirsi alla memoria, né alla cache. Per questo, ogni registro di Segmento ha un suo descrittore attivo nella cache.

Vi sono delle regole fondamentali per quanto riguarda i privilegi:

- Qualità dei dati: un processo può accedere a dati (segmenti) dello stesso livello o inferiori
- Affidabilità del codice: un segmento di codice può accedere ad un altro segmento dello stesso livello o superiori

Esempio:

MOV AX, [EBX]



## Focus sulla memorizzazione dei file eseguibili in memoria

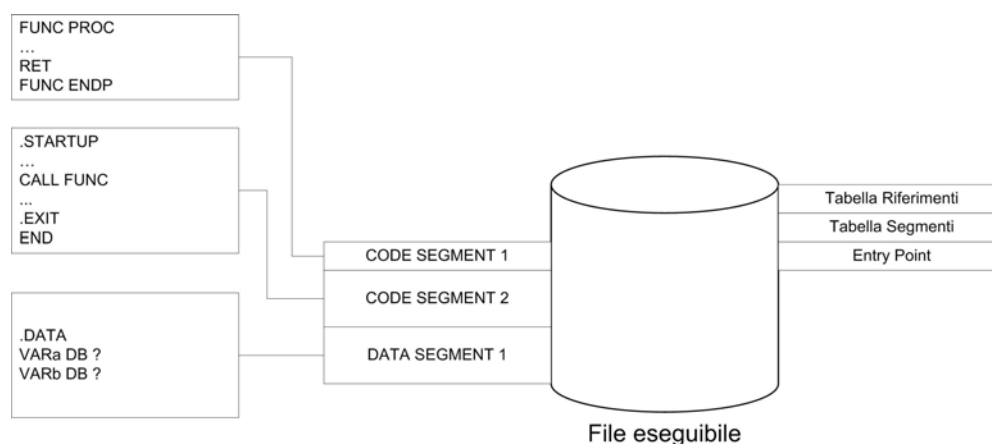
In fase di assembling, alcune informazioni contengono informazioni incomplete.

Ciò è dovuto al fatto che tali istruzioni si riferiscono ad indirizzi che per forza di cose non possono essere statici, perché i segmenti (codice o dati) che li contengono verranno caricati in memoria al momento dell'esecuzione di volta in volta in posizioni diverse.

Questi indirizzi o label verranno sostituiti da dei riferimenti. L'elenco dei riferimenti diverrà una tabella, scritta in una parte del file eseguibile.

Tutti i segmenti (dati, codice, stack...) vengono registrati anch'essi in una tabella chiamata tabella dei riferimenti.

L'entry point è invece l'indirizzo della prima istruzione del file eseguibile.



L'eseguibile contiene:

- Tabella Segmenti
- Tabella Riferimenti
- Entry Point
- Segmenti (Codice, Dati, Stack...)

Al momento dell'esecuzione, l'allocatore dell'O.S.:

- i. Individua uno spazio di memoria libera
- ii. Sposta tutti i segmenti in memoria (anche non sequenzialmente)
- iii. Risolve il codice utilizzando la Tabella Segmenti e Tabella Riferimenti
- iv. Carica il PC con l'entry point



## Paginazione

La paginazione permette il mapping fra memorie di dimensione diversa.

Quando abilitata, la CPU suddivide lo spazio d'indirizzamento lineare in pagine di dimensione fissa (da 4 KByte, 2MByte o 4 MByte) che possono essere mappate sia in memoria fisica sia sugli hard drive.

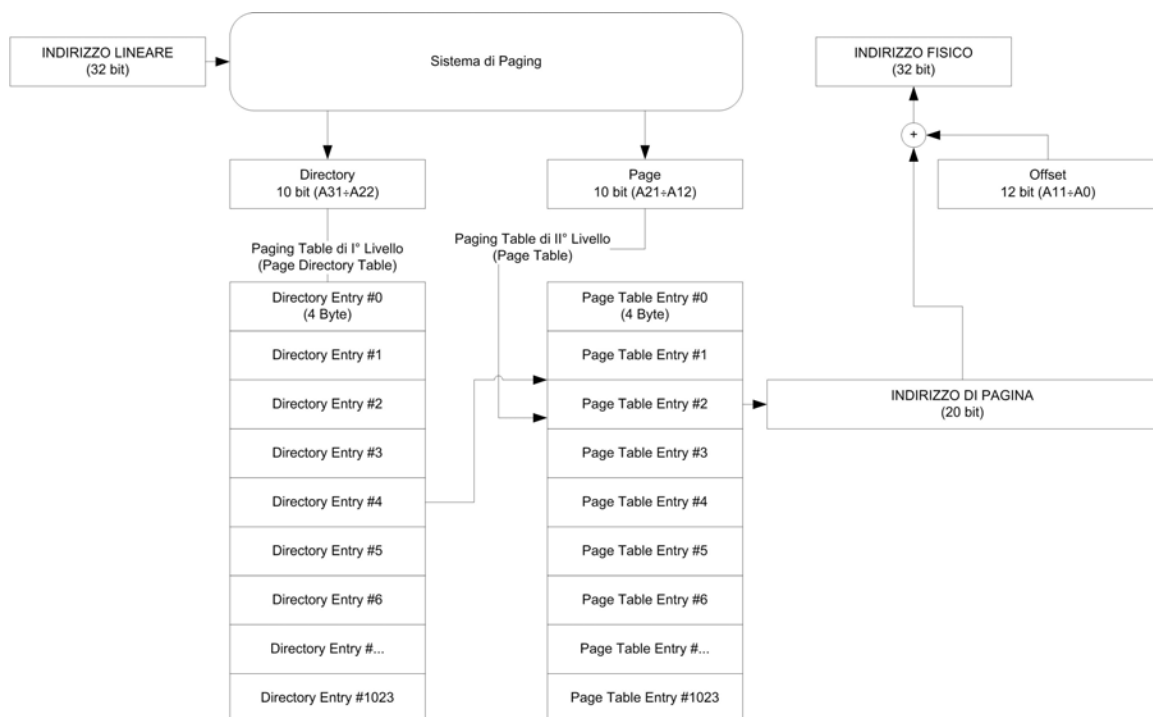
L'indirizzo della pagina è calcolato dal sistema di paging in più fasi e avvalendosi delle tabelle di Paginazione. Se la pagina non è nella memoria fisica, viene scatenato un Page Fault.

Nei sistemi a 32 bit i livelli di Paginazione sono 2, nei sistemi a 64 bit sono 4.

In un sistema a 32 bit, ogni tabella ha  $2^{10}$  elementi. Ogni elemento è di 4 Byte (con pagine da 4 KB).

In totale le 2 tabelle occupano 4MByte.

L'indirizzo della Tabella di Paginazione di 1° Livello è scritto nel registro CR3.



Le entry delle Tabelle di Paginazione, oltre a contenere l'indirizzo di Pagina, forniscono diverse informazioni:

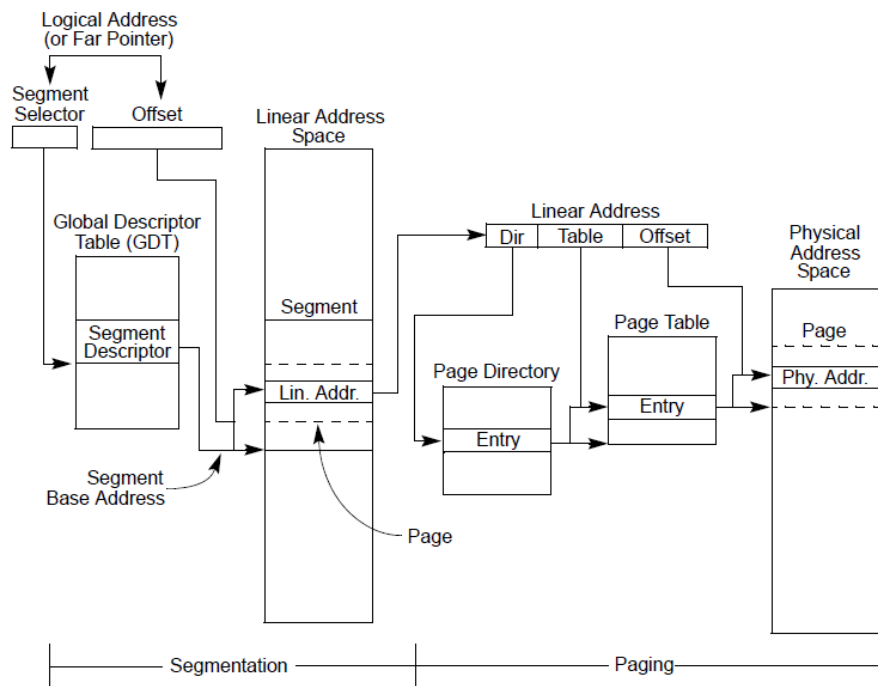
- User/Supervisor: permette l'implementazione della gestione dei permessi
- Access
- Read/Write: informazioni riguardo possibilità di lettura/scrittura della pagina

Tali attributi sono stati introdotti per una sorta di "similitudine" con altre architetture, che sono magari sprovviste di segmentazione.

Dato che la risoluzione degli indirizzi deve essere eseguita spesso, le entry delle tabelle di Paginazione utilizzate più recentemente vengono inserite nella TLB (Translation LookAside Buffer).

La TLB è una cache on chip (sulla CPU) che contiene una copia dell'ultima Page-Directory entry e delle ultime Page-Table entries.

## Visione complessiva



## O.S. moderni: considerazioni

Se da una parte la paginazione può essere disabilitata, la segmentazione è sempre attiva.

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 GB) e sovrapposti, al fine di far puntare tutti i descrittori ad un solo enorme segmento.

La gestione dei privilegi e dei processi viene quindi affidata, seppur in modo meno preciso, alla paginazione.

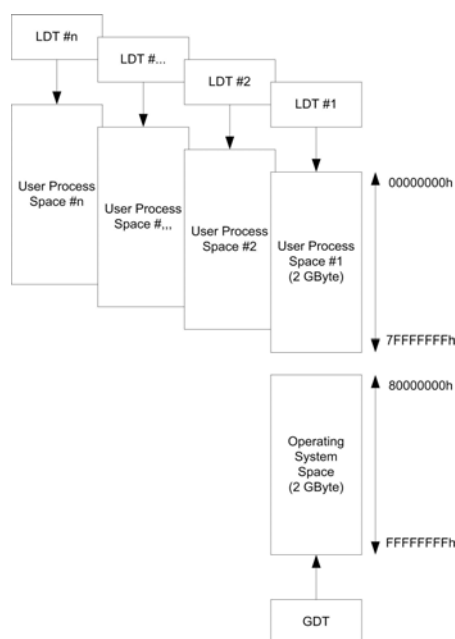
Al bootstrap, tutti i registri di segmento ed i relativi descrittori verranno inizializzati al fine di realizzare una memoria paginata lineare.

Esempio:

*Modello di memoria avanzato*

Peculiarità di tale modello:

- Più "User Process Spaces": ognuno ha la propria LDT
- Il tutto è gestibile via hardware
- Process Task Switching via hardware, con il supporto del TSS (Task State Segment) che contiene le tabelle, PSW, registri, IVT...



## Evoluzione (x86-64)

L'architettura x86 si evolve al fine di riuscire a gestire in hardware i processi e/o thread.

Tra le diverse tecnologie che sono implementate vi sono:

- Multiprocessor Chip: nuovo modello computazionale basato sul parallelismo dei thread
- SMP on chip: sincronizzazione dei processori e cache on chip, senza il supporto di circuiteria esterna
- Address BUS esteso a 36 bit ([PAE](#)): ciò permette di indirizzare 64 GByte di memoria fisica; poiché gli indirizzi lineari sono a 32 bit, vi è una gestione particolare delle tabelle di paginazione, che hanno entry da 8 Byte ognuna
- Registri RAX di Offset a 64 bit
- Abolizione della segmentazione e gestione completa via paginazione a 4 livelli
- Supporto alla virtualizzazione: possibilità di lanciare più O.S. e dedicare ad ognuno un certo numero di CPU fisiche e/o logiche
- [Hyper-Threading](#): possibilità di eseguire più thread simultaneamente, anche se si ha a disposizione un solo processore
- FSB dual o quad pumped: vengono sfruttati fronti tutti i fronti d'onda

## SMP (a BUS comune)

I seguenti esempi si riferiscono ad un sistema con 2 processori con identica priorità e a BUS comune. All'avvio, uno dei due processori viene dichiarato come Master e l'altro come Slave (per la contesa del BUS).

L'arbitraggio prevede che le CPU si possano scambiare i ruoli di M/S al termine di un ciclo di BUS.

### Hardware per SMP (segnali)

Dal punto di vista di piedinatura e segnali, ogni CPU deve disporre:

- per la gestione della CACHE ([MESI](#))
  - di PHIT: lo Slave segnala all'altra CPU che c'è stato un HIT nella propria CACHE
  - di PHITM: lo Slave segnala all'altra CPU che c'è stato un HIT nella propria CACHE in una Line modified
- per la gestione del BUS (arbitraggio distribuito)
  - di BUSRQ
  - di BUSACK

### Snooping e coerenza CACHES

La gestione della coerenza delle CACHE è supportata da una tecnica chiamata Snooping: ogni CACHE controlla gli indirizzi e la tipologia di operazione che è in corso attraverso l'Address BUS ed i segnali di controllo. Tale tecnica necessita del raddoppio dell'Address BUS.

Se la CACHE individua una operazione di Write su un indirizzo che ha caricato, il CACHE controller invalida tale Line.

I cicli di Read/Write coinvolgeranno quindi l'Address BUS (0) e il segnale KEN (Cache Enable).

I cicli di Snooping coinvolgono sia l'Address BUS (1) e il segnale SNOOP.

Se è stata eseguita una Write su una CACHE Line, possono essere adottate 3 tecniche:

- Write-Back: la scrittura è effettuata solo in CACHE. I dati in CACHE verranno ricopiati in memoria solo quando:
  - La CACHE Line deve essere sostituita per fare spazio ad una nuova Line
  - E' stata lanciata una istruzione di Write-Back esplicita
  - Il segnale di FLUSH è stato attivato
- Write-Through: la scrittura è effettuata sia in CACHE che in memoria
- Write-Allocate: il dato non è ancora in CACHE e viene quindi caricato e aggiornato

Nei casi di CACHE Miss, se la cache è Set-Associative, vi sono diverse politiche per la scelta delle Line da sostituire: LRU, Pseudo-LRU, Random...

## MESI

Per l'implementazione della tecnologia MESI, sono necessari lo Snooping e delle CACHE Line che contengono informazioni riguardo il loro "stato".

MESI (Modified Exclusive Shared Invalid) è un protocollo per la gestione della coerenza delle CACHE.

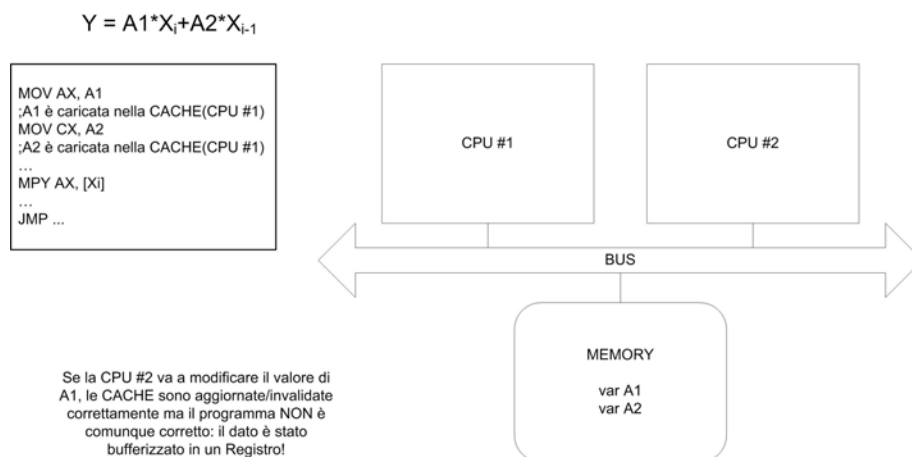
Gli stati di una Line possono essere riassunti con la seguente tabella.

Stato della CACHE Line	Modified	Exclusive	Shared	Invalid
Line valida?	Sì	Sì	Sì	No
La copia in memoria è ...	Non aggiornata	Valida	Valida	-
Esiste una copia anche nella CACHE di altre CPU?	No	No	Forse	Forse
Una Read su questa Line ...	Nessuna attività su BUS di sistema	Nessuna attività su BUS di sistema	Nessuna attività su BUS di sistema	Si ottiene un MISS: diventa E o S se il dato è cacheable, I se non è possibile effettuare caching. Conseguente CACHE Line Fill
Uno Snooping su questa Line ...	Un ciclo Inquiry ha prodotto un HIT: diventa S o I. Write-Back su BUS di sistema	Un ciclo Inquiry ha prodotto un HIT: diventa S o I.	Può diventare I: il controller esterno che ha eseguito l'Inquiry sa che la copia è stata aggiornata Nessuna attività su BUS di sistema	Nessuna attività su BUS di sistema
Una Write su questa Line ...	Nessun Write-Back su BUS di sistema	Nessun Write-Back su BUS di sistema. Diventa M	La CPU che la contiene otterrà proprietà esclusiva della linea. Diventa E	Andrà direttamente sul BUS di sistema

Il MESI è utilizzabile in caso di CACHE Write-Back senza Write-Allocation. Può essere estesa a CACHE Write-Through, ma in questo caso gli stati M ed E non esistono.

Un algoritmo più avanzato di MESI è MOESI (dove O sta per Owned).

Esempio:



## Bibliografia

La dispensa è stata scritta utilizzando come supporto le seguenti risorse:

- [Intel® 64 and IA-32 Architectures Software Developer's Manual](#)
- Messmer – The Indispensable PC HARDWARE BOOK 3rd Ed.
- Slide del Prof. Mezzalama Marco, Matteo Sonza Reorda, Paolo Bernardi, M. Rebaudengo
- [Wikipedia](#)
- Don Anderson, Tom Shanley, MindShare, Inc - Pentium Processor System Architecture

## Riferimenti

Per consigli, domande e suggerimenti  
Luca Belluccini - [lucabelluccini@gmail.com](mailto:lucabelluccini@gmail.com)

## Licenza



Quest'opera è stata rilasciata sotto la licenza Creative Commons Attribuzione-Non commerciale-Non opere derivate 2.5 Italia. Per leggere una copia della licenza visita il sito web <http://creativecommons.org/licenses/by-nc-nd/2.5/it/> o spediisci una lettera a Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.