

Modalità di funzionamento IA-32

Slide 62

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).

Nessun 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 può passare 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

Slide 63→65

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

Slide 66→72

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 <i>mem</i>
EDI / DI	Destination Index	Registri indice per la memoria	Da trattare come operandi di tipo <i>mem</i>

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 <i>PUSH</i> e incrementato ad ogni <i>POP</i>
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

Slide 73→78

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.

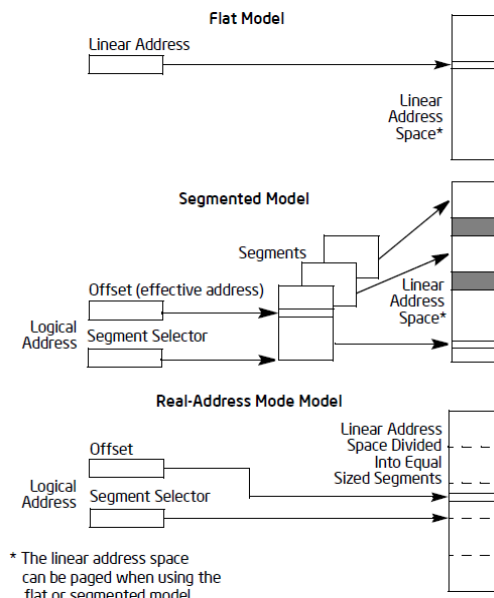
- Segmented Memory Model

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](#).

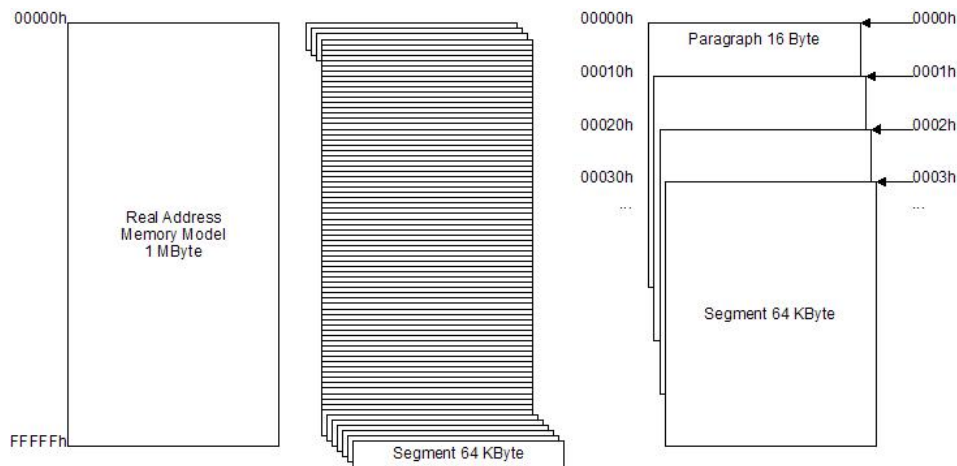


Real-address mode Memory Model

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	0000	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.