



Le virtual machine e la memoria virtuale

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento Patterson: 5.6, 5.7, 2.12.



Sommario

Le virtual machine

La memoria virtuale

La traduzione degli indirizzi



Virtual machines (VM)



La VM è un software che fornisce un **ambiente completo a livello di sistema**, compatibile con una certa ISA (e.g. MS-DOS): VMware, ESX Server...

Sviluppate negli anni 60 e riprese recentemente per:

- Sicurezza e isolamento di sotto-sistemi.
- Condivisione dello stesso sistema da parte di molti utenti (e.g. cloud computing)
- Riproduzione di SO vintage.

Velocità dei processori rende il costo della virtualizzazione accettabile.

Lo stesso calcolatore può **supportare più VM** e può quindi supportare più SO e ambienti, anche obsoleti quali ad esempio il DOS e quindi programmi non più eseguibili direttamente. Esempio:

- MS-DOS
- Linux Ubuntu
- Beta-release di un SO (per testing)

Può consentire l'esecuzione di un'applicazione nel suo ambiente nativo (versione corretta del SO: e.g. Windows 7, Windows XP).



Caratteristiche



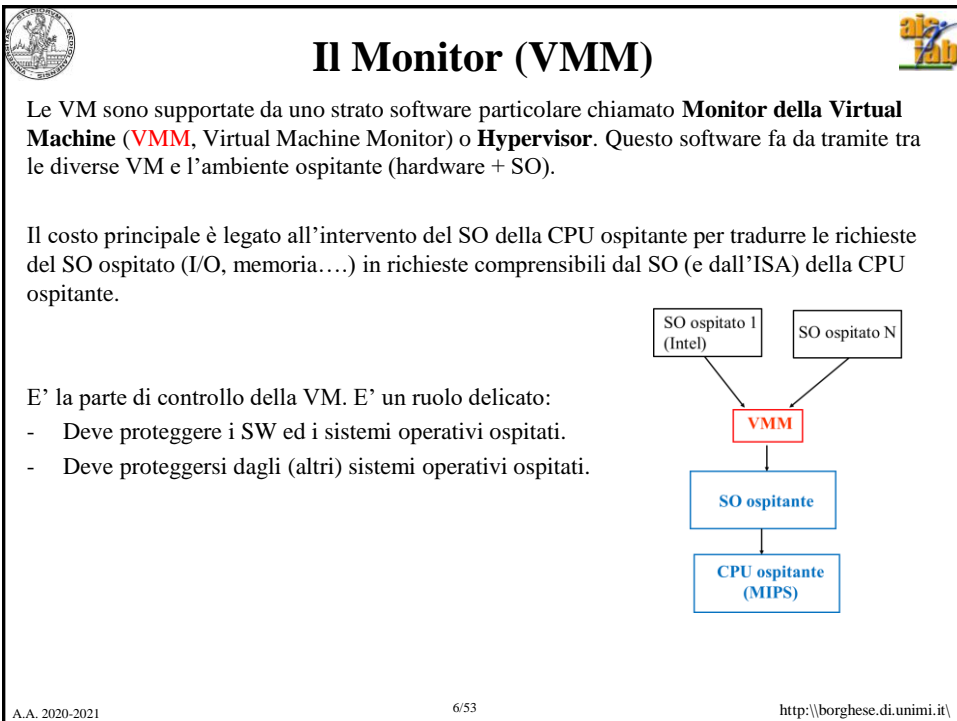
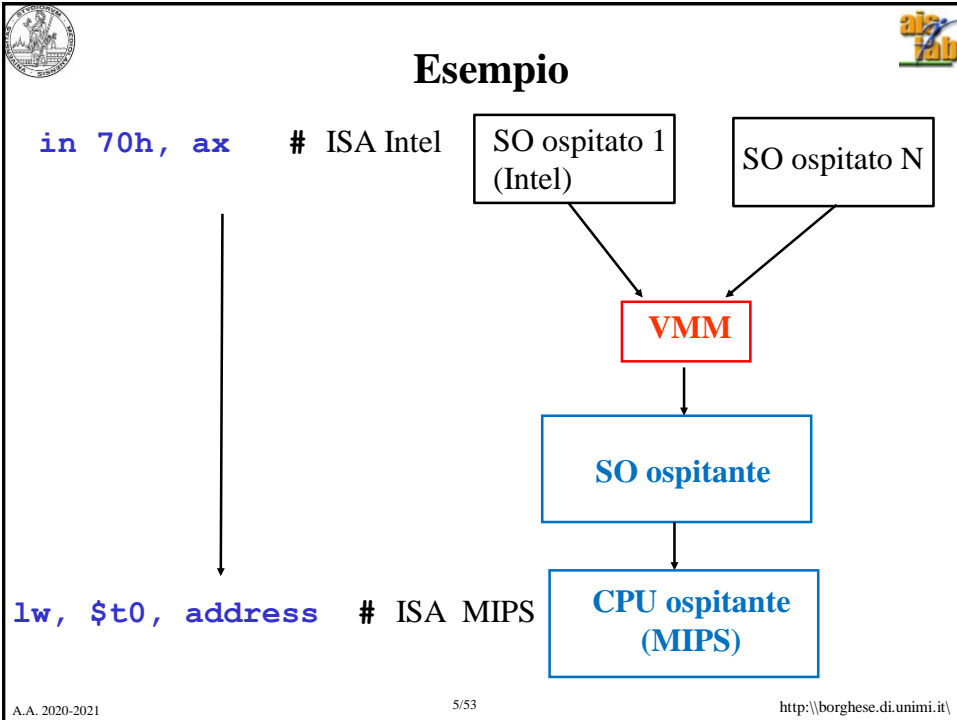
SW di emulazione. (e.g. Java Virtual Machine): non è una macchina fisica!

VM di un'ISA: ambiente di sistema congruente con una certa ISA (elementi di stato, funzionalità messe a disposizione dall'HW).

Le VM creano l'illusione, ad esempio, di avere a disposizione una macchina MS-DOS su una macchina il cui SO è Linux.

Essendo SW, diverse VM possono essere eseguite sulla stessa macchina ospitante, emulando diversi SO. Vengono eseguite in modo segregato. Ciascuna macchina, quando è in esecuzione, utilizza l'hardware a disposizione come fosse suo.

Il software eseguito da una VM (software ospitato) deve essere eseguito esattamente **come se fosse eseguito sull'ISA nativa** (a parte le prestazioni).





Il funzionamento del VMM



Solitamente:

- un **VMM ha privilegi di sistema** (deve interfacciarsi con la CPU ospitante).
- una **VM ha i privilegi di utente** (è equivalente a un comune programma per la CPU ospitante).

Il software eseguito da una VM **non deve modificare nulla delle risorse della macchina ospitante direttamente**, ma deve passare attraverso il SO della macchina ospitante (per cui le richieste di risorse vengono dettate (trapped) dal VMM).

Un VMM mappa le risorse virtuali (della VM) nelle risorse fisiche della CPU ospitante:

- Time-sharing
- Partizionamento
- Emulazione.
- Multi-thread.

Per **esempio**, nel caso in cui un processo ospitato (guest) sia basato su un timer (eccezione di sistema), l'eccezione dovrà essere «tradotta» dal VMM. La «traduzione» potrebbe utilizzare il timer di sistema della CPU ospitante, se disponibile o essere tradotta in un task del SO.

Il timer del processo ospitato non accede direttamente al timer, ma ci accede attraverso il VMM che si interfaccia con il SO della macchina ospitante.



Il VMM e le istruzioni privilegiate



Il costo della virtualizzazione dipende dal tipo di processo eseguito. Tanto **maggiore è il ricorso al SO**, tanto **maggiore il costo della virtualizzazione** (e.g. processi con un I/O intensivo hanno un overhead elevato, processi di calcolo intensivi hanno poco over-head).

L'overhead dipende

- Numero di chiamate al SO (e di istruzioni che devono essere emulate)
- Aumento del tempo di esecuzione di ciascuna di queste istruzioni (e.g. timer HW ha un costo molto inferiore di una “traduzione” SW in un task del SO).



Richiesta al SO



Cosa succede quando un SO ospitato fa una richiesta privilegiata (e.g. lancia un interrupt o un'eccezione?).

Queste devono essere intercettate (**trap**) dal VMM che le gestirà nel modo più appropriato tenendo conto dello stato della CPU ospitante e delle altre VM.

Questo processo può rilevarsi molto costoso soprattutto quando l'applicazione ospitata fa molto uso di I/O ma anche di altre funzionalità.

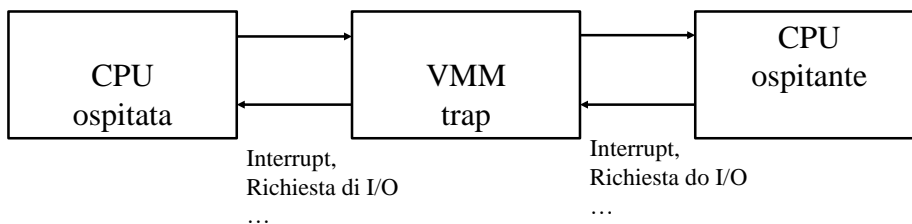
Applicazioni che eseguono principalmente funzioni di calcolo o fanno poco utilizzo della virtualizzazione, sono quelle eseguite più velocemente dalle VM.



Funzionamento del VMM



Comunicazione a 2 vie





Amazon Web Services (AWS)



Utilizza le VM per il cloud computing:

- I diversi utenti vengono protetti uno dall'altro.
- Il SO e la sua configurazione, richiesto da un cliente può venire installato su più macchine del cloud.
- AWS può eliminare le VM che non servono più risparmiando risorse.
- Si possono utilizzare macchine eterogenee, non della stessa generazione, aggiornandole via via. Le prestazioni vengono specificate "Equivalent CPU capacity".
- Il VMM può essere configurato per misurare:
 - Lo spazio su disco
 - L'utilizzo della rete
 - L'utilizzo del processorePer poi fatturare al cliente.



Sommario



Le virtual machine

La memoria virtuale

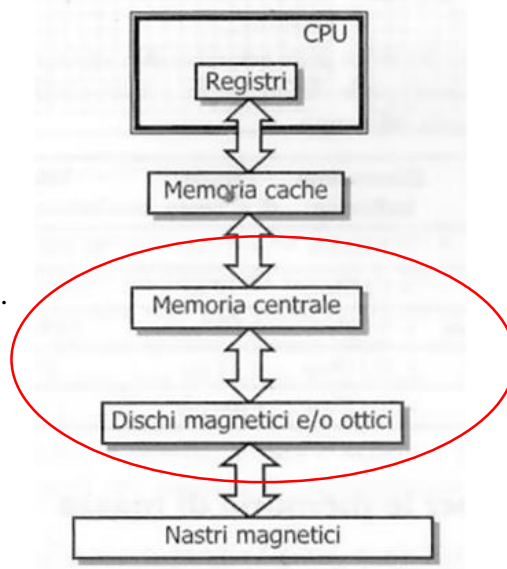
La traduzione degli indirizzi



Gerarchia di memoria



Memoria centrale
“cache” per la
memoria su disco.



Le origini della Memoria Virtuale



1) L'insieme di dati e di memoria (**working set**) di un task può essere maggiore di quanto si può caricare in memoria principale.

Il working set si può suddividere in moduli che possono risiedere alternativamente in memoria. Questi moduli condividono lo stesso spazio di indirizzi fisico e non potranno coesistere nella memoria.

Il meccanismo che gestisce il caricamento / scaricamento dei moduli di uno stesso task si chiama(va) **overlay**. Era il programmatore a specificare quali moduli potevano essere in overlay, poi ha iniziato a pensarci il linker. **Moduli bilanciati**.

2) Necessità di protezione dello spazio di memoria (principale) di un processo. Nessun altro processo deve potere scrivere nel suo spazio di lavoro (**protezione della memoria**).



La memoria virtuale



Principio: distinzione tra memoria logica e memoria fisica.

Meccanismo. Viene **assegnata** la memoria principale **logica** a ogni processo in una **quantità sufficientemente grande** (anche maggiore della memoria fisica presente sulla CPU).

La memoria principale **fisica** è la **stessa** per **diversi programmi in esecuzione**, i quali **condividono lo stesso spazio di indirizzamento fisico** e quindi devono risiedere in zone diverse della MM.

I programmi sono inizialmente residenti su disco.

La memoria virtuale consente di mappare lo spazio di memoria logico (software) di ciascun processo sulla memoria fisica (reale).

Lo spazio virtuale viene mappato dalla MMU sulla MM, memoria fisica, **evitando sovrapposizioni** e consente **una gestione sicura della memoria**.

La memoria virtuale consente anche di nascondere al programmatore la **dimensione limitata della memoria principale fisica** rispetto alla memoria logica richiesta dal processo.



Terminologia



Un blocco di memoria principale: virtuale o fisica (della stessa dimensione) viene chiamato **pagina** (equivalente alla linea della cache).

L'accesso alla memoria principale avviene mediante l'indirizzo virtuale (che è quello specificato dall'ISA, `lw $t0, address. Address` fa quindi riferimento alla memoria virtuale – indirizzo virtuale).

L'indirizzo virtuale del processo deve essere trasformato in un indirizzo fisico dalla MMU.

- Il processo può essere mappato ovunque in MM.
- L'indirizzo fisico viene assegnato all'atto del caricamento (**loading**) del processo da disco a MM, viene caricato in un certo numero di pagine della memoria fisica.

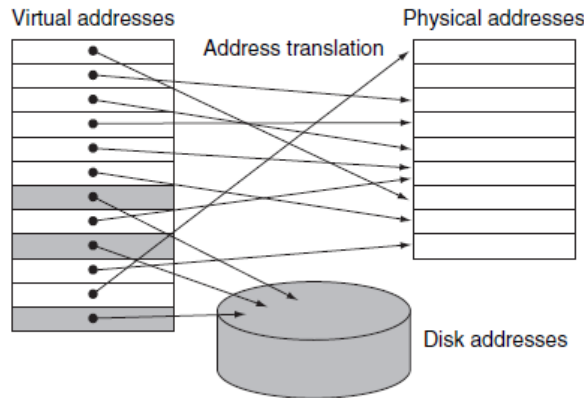
Una miss dalla memoria principale in giù viene chiamata **page fault**.

Se si verifica una hit nella memoria principale, il micro-blocco viene trasferito dalla memoria principale alla linea di cache, se si verifica un page fault, occorre caricare la pagina da disco in memoria principale e poi trasferire il micro-blocco contenuto in quella pagina nella opportuna linea di cache.



Indirizzi fisici e indirizzi virtuali

Tabella delle pagine (mappatura)



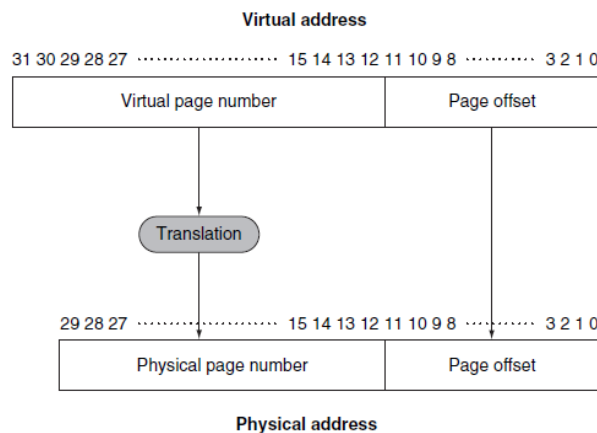
Ciascun programma ha il suo spazio di indirizzamento (virtuale) che viene mappato su disco e sulla memoria principale (indirizzi fisici).

Il processore genera indirizzi virtuali mentre alla memoria si accede mediante indirizzi fisici.



Indirizzamento della MM

Indirizzamento della MM: **indirizzo base** (indirizzo della pagina) + **offset** di pagina



Altro esempio di base (numero di pagina) + offset

- Il numero di pagina fisico è determinato dalla quantità di memoria fisica.
- Il numero di pagine virtuali è virtualmente infinito: dipende dal numero di bit degli indirizzi virtuali. In questo caso 30 bit di indirizzamento della memoria fisica -> 1 Gbyte di memoria fisica e 32 bit di indirizzamento della memoria virtuale -> 4 Gbyte di memoria logica (virtuale)



Criteri di progettazione della memoria virtuale



Strategie diverse da quelle utilizzate nelle cache. Obiettivo principale è **nascondere la penalità di page fault**.

Memory technology	Typical access time	\$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	\$500–\$1000
DRAM semiconductor memory	50–70 ns	\$10–\$20
Flash semiconductor memory	5,000–50,000 ns	\$0.75–\$1.00
Magnetic disk	5,000,000–20,000,000 ns	\$0.05–\$0.10

L'accesso a disco (magnetico) può essere quasi un milione di volte più lento della MM. 100 volte più lento nel caso ottimale di disco (a stato solido).

NB Questa penalità è dovuta largamente al tempo per accedere alla prima parola della pagina (latenza). Con il trasferimento a burst il resto dei dati viene trasferito più velocemente.

Occorre nascondere questa latenza! E minimizzare i page fault.



Criteri di organizzazione della memoria principale



Le pagine devono essere sufficientemente ampie per ammortizzare i tempi di accesso (anche a seguito di una hit). Pagine di 4 KByte sono tipiche.

- Server -> 32/64 Kbyte
- Sistemi embedded -> 1 KByte

La mappatura delle pagine virtuali da disco sulla memoria principale è **completamente associativa** per massimizzare il riempimento della MM fisica ed evitare i page fault => Non c'è necessariamente contiguità tra le diverse pagine di uno stesso processo all'interno della MM fisica.

La gestione dei page fault può essere software (SO) visto il lungo tempo a disposizione per gestirli.

Ottimizzazione del posizionamento delle pagine (scelta oculata di quale pagine scaricare su disco per fare posto a una nuova pagina).

Lo scaricamento su disco **di una pagina** (scrittura) viene gestita in modalità *write-back* (con write buffer).



Posizionamento delle pagine

Si utilizza la **Tabella delle pagine**, a sua volta residente nella memoria principale, indirizzata dal *registro della tabella delle pagine*.

Non si può cercare una pagina virtuale, controllando tutte le pagine contenute nella memoria fisica => non si può utilizzare una tabella delle pagine associativa.

La tabella delle pagine è **indicizzata** con il numero della pagina virtuale (è una memoria ad accesso diretto).

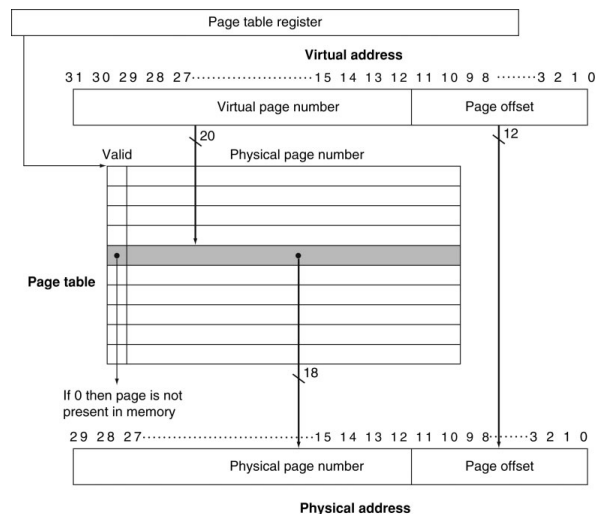
La tabella delle pagine contiene la traduzione del numero di pagina virtuale in numero di pagina fisica. Controlla anche se la pagina fisica sia già presente nella memoria fisica (hit o miss).

Tabella delle traduzioni o mappatura. Ogni processo ha la sua PT.

E.g. Catalogo di una libreria (mappatura tra numero sequenziale del libro – indirizzo virtuale; e posizione nella libreria – indirizzo fisico). Alcuni libri possono non essere contenuti nella libreria locale ma in una libreria più grande (il disco).



Tabella delle pagine



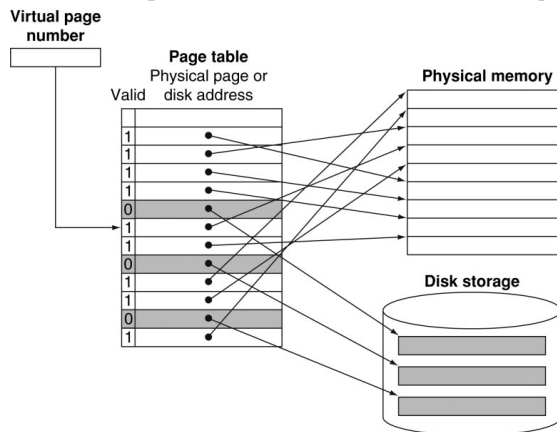
L'altezza della memoria è teoricamente pari al numero di pagine virtuali. Per ogni pagina virtuale è riportata la traduzione. Bit di validità come nelle cache.



Gestione di un page fault - I

All'avviamento di un processo il SO crea tutte le pagine del processo su disco (**swap space**). La pagina richiesta all'inizio non sarà contenuta in memoria, ma sarà contenuta su disco nello **swap space** di ogni processo (page fault da cold-start). Anche successivamente la pagina potrebbe non essere presente in memoria principale.

La posizione in MM e su disco può essere data dalla stessa tabella delle pagine.



A.A. 2020-2021

<http://borghese.di.unimi.it/>



Gestione di un page fault - II

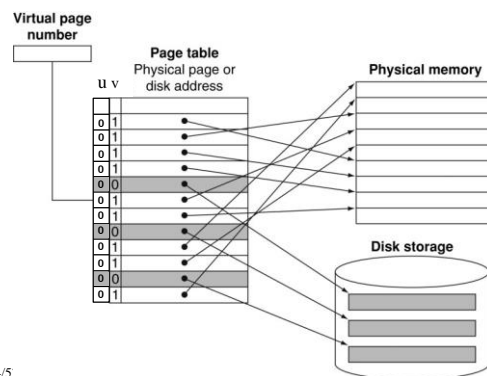
Se la MMU cercando nella tabella delle pagine identifica che la pagina cercata ha il bit di validità = 0 (pagina non ancora caricata), genera un page fault.

Un page fault genera un'eccezione che deve essere gestita via hardware/software (dal SO). Come?

LRU approssimato. **Use bit (Reference bit)**. Quando la MMU accede a quella pagina imposta lo use bit a 1. Periodicamente tutti gli use bit vengono azzerati.

In caso di page fault, la pagina virtuale viene trasferita in una pagina il cui Use bit è = 0.

NB Lo use bit è differente dal bit di validità.



A.A. 2020-2021

24/5



Dimensionamento di una tabella delle pagine



Indirizzi virtuali e fisici su 32 bit

Dimensione della pagina di memoria di 32 Kbyte (2^{15} Byte)

Numero di elementi *massimo* della tabella delle pagine (numero di linee massimo):

$$2^{32} / 2^{15} = 2^{17} \text{ linee (ciascuna di 4 Byte} \rightarrow \text{indirizzi fisici su 4 Byte)}$$

Tabella di ampiezza 4 Byte e altezza massimo 128 K linee \Rightarrow 512 Kbyte (upper bound)

Abbiamo bisogno di 0.5 Mbyte **per ogni processo attivo**.

Cosa succede se abbiamo centinaia di processi?

Cosa succede per indirizzi a 64 bit?

$$2^{64} / 2^{15} = 2^{49} \text{ linee (ciascuna di 4 Byte)} \Rightarrow 2\,000\,000\,000\,000\,000 \text{ di Byte!}$$



Limitazione della tabella delle pagine



- **Limitazione monodirezionale dell'altezza.** Tabella virtuale di altezza limitata (numero max di linee – cioè di pagine virtuali). Possibilità di espandere in senso crescente.
- **Limitazione bidirezionale dell'altezza.** Memoria dati suddivisa in stack e heap. Crescono in direzioni opposte \Rightarrow 2 tabelle delle pagine, una che ha indirizzi che crescono verso il basso e una che ha indirizzi che crescono verso l'alto. Entrambe con numero di linee limitato (segmenti).
- **Accesso indiretto.** Altezza pari all'altezza della memoria fisica. Funzione di hashing per mappare l'indirizzo virtuale in indirizzo fisico. L'indirizzo virtuale non è più un indice ma diventa una chiave di accesso alla tabella.
- **Tabella delle pagine gerarchica.**
 - Primo livello. Indicizza segmenti costituiti da un certo numero di pagine virtuali contigue (**segment table**).
 - A) Il segmento contiene almeno una pagina caricata in MM.
 - B) Il segmento non contiene pagine caricate in MM
 - Nel caso A) la linea della segment table punta alla sotto-tabella delle pagine per quel segmento.
- **Virtualizzazione della tabella delle pagine** che viene inserita in memoria virtuale.



IL TLB – Translation Lookaside buffer



Con le PT (Page Table - Tabelle delle pagine) ogni accesso alla memoria principale richiede in realtà **2 accessi** (1 accesso alla PT + 1 accesso alla pagina).

Si sfrutta il principio di località: quando si accede al dato di una certa pagina, facilmente si avrà bisogno di un altro dato della stessa pagina. Si può evitare di dovere ritradurre una seconda volta l'indirizzo virtuale. Come?

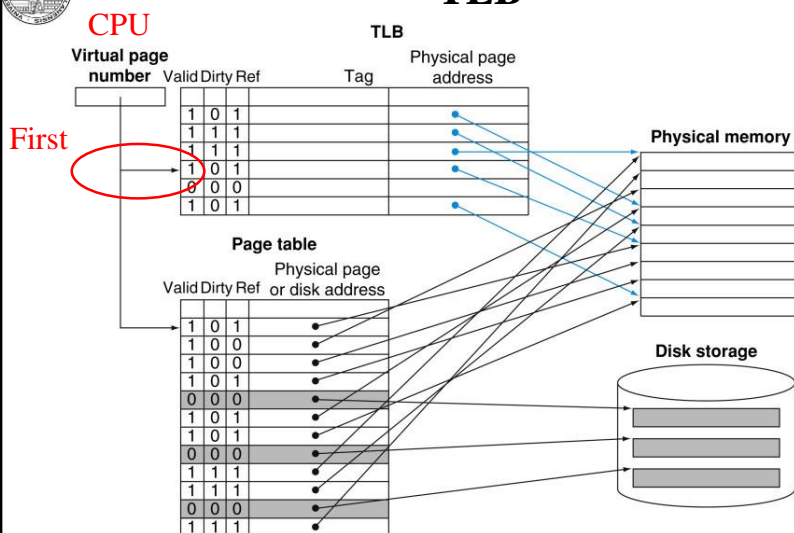
Si introduce un buffer HW: **Translation Lookaside Buffer** (TLB) che memorizza le traduzioni (cache delle traduzioni):

- Dimensioni: 16-512 elementi (linee)
- Dimensione della linea (1-2 pagine, 4-8 byte ciascuna)
- Tempo di hit: 0,5-1 ciclo di clock
- Penalità di miss: 10-100 cicli di clock (per caricare la traduzione)
- Frequenza di miss: 0,01%-1%

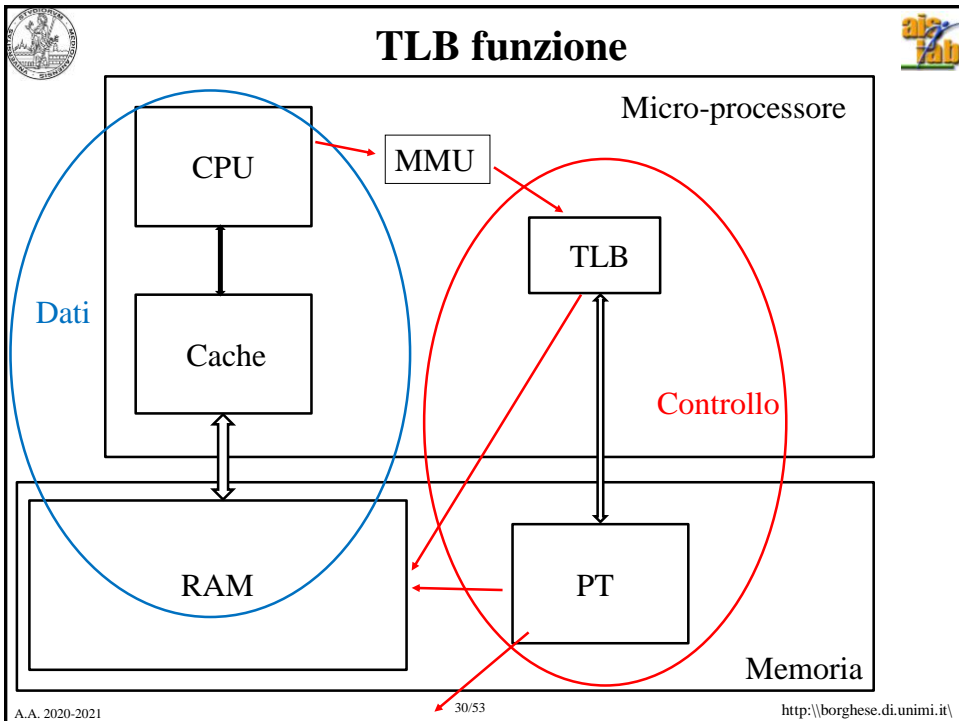
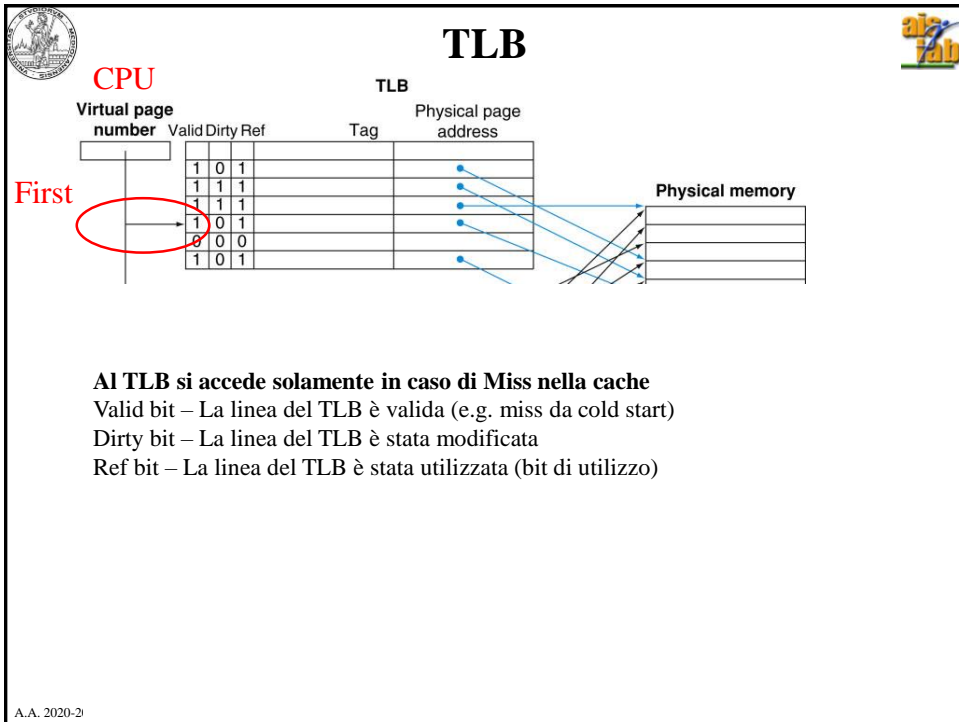
E.g. Nella libreria è come mantenere su un fogliettino la corrispondenza tra il numero d'ordine dei libri appena consultati e la loro posizione sugli scaffali della libreria.



TLB



Il TLB è una cache a tutti gli effetti, associativa con un numero di elementi molto più ridotto del numero delle pagine virtuali. Si trova nella MMU (all'interno della CPU).





Accesso alla memoria principale (hit in cache)



1) Invio al TLB dell'indirizzo virtuale

Ho due situazioni:

- 1a) Indirizzo della pagina fisica è presente nel TLB (**hit del TLB** - TAG = #pagina e valid = 1)
Reference bit = 0/1
Dirty bit = 1 (se accesso in scrittura).
- 1b) Miss -> Indirizzo della pagina fisica non è presente (**miss del TLB**).

Se ho una hit -> indirizzo fisico della (pagina della) memoria principale
Lettura del micro-blocco e trasferimento in cache.



Accesso alla memoria principale (miss in cache, ma hit in MM)



1) Invio al TLB dell'indirizzo virtuale

1b) Miss -> Indirizzo della pagina fisica non è presente (**miss del TLB**).

2) Leggo la linea della tabella delle pagine usando come indice l'indirizzo virtuale e controllo il bit di validità.

2a) La traduzione è nella PT -> indirizzo fisico.

2b) La traduzione non è nella PT -> (**page fault**) -> indirizzo su disco.

2a) La traduzione è in MM

Carico nel TLB la traduzione dell'indirizzo virtuale se miss nel TLB

Trasferisco da MM a cache la linea richiesta.



Accesso alla memoria principale (miss in cache e page fault)



- 1) Invio al TLB dell'indirizzo virtuale
- 1b) Miss -> Indirizzo della pagina fisica non è presente (**miss del TLB**).
- 2b) La traduzione non è nella PT -> (**page fault**).

Nel caso di **page fault**:

- Trasferisco da disco a MM la pagina richiesta
- Carico nella TB la traduzione dell'indirizzo della pagina virtuale.

Le miss del TLB sono più frequenti dei page fault.



Gestione dei fallimenti



Miss nel TLB. Il TLB è una cache associativa. Sostituzione random. I bit di stato (valid, dirty, reference) vengono utilizzati per la politica di sostituzione delle pagine => quando scarico una linea del TLB nella PT devo aggiornare lo stato della pagina fisica. Se ho utilizzato di recente i dati di quella pagina quando erano in cache o li ho modificati deve risultare.

Page fault. Cercare la pagina virtuale nella tabella delle pagine e localizzare la pagina fisica corrispondente (su disco).

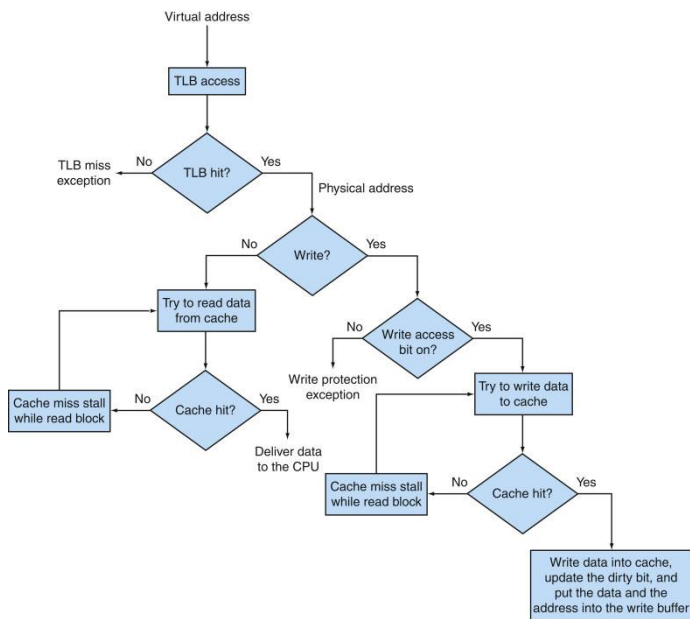
Scegliere una pagina fisica da sostituire; se la pagina fisica identificata ha il bit dirty = 1 (ci sono delle parole modificate), deve essere scritta su disco prima di caricare la nuova pagina nella memoria principale.

Iniziare il trasferimento della pagina da disco a memoria principale.

Aggiornare la traduzione della pagina virtuale nella pagina fisica all'interno della PT e reset del bit dirty, set del bit di validità e di reference.



Gestione della richiesta di un dato



Possibili situazioni in Cache, PT e TLB



TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.



Implementazione della protezione della MM



Page Table page pointers are stored inside the OS space. They can be accessed only if OS is in Kernel mode.

We have to avoid that a **user process** (a virus) changes its own PT, **mapping one of its virtual pages into a physical page of a different process**. When accessing a physical page, a program should own that page.

This can be obtained by a personalized PT that can be addressed and modified by writing only by the OS. **Only reading the PT is granted to any user process.**

The write access bit can be used to protect the page from writing.

A **process cannot read a page of another process** because the mapping is contained inside its PT that has been written and is modified by the OS.

If a process wants to share a page with another process, the two virtual pages should point to the same physical page.

All these mechanisms are mediated by the OS in kernel mode.



Changing process



SO changes running process from P1 to P2 (**context switch**)

- Save the state of the CPU (registers, PC)
- Change the address of the Page Table from that of P1 to that of P2 (kernel mode operation)
- **Invalidate the TLB** (that is HW, to avoid to work on the physical pages of P1).

Upon restart of P1, we will have a lot of cold-start misses in the TLB.

NB We have 1 TLB but many PTs.

Per evitare lo svuotamento del TLB inutilmente:

- **Process identifier or task identifier** (field on 8 bits Intrinsic FASTMath MMU) added to the TAG field.
- Check for a hit is carried out on both the TAG and the process identifier fields (both in caches and in TLB).



Sommario



- Le virtual machine
- La memoria virtuale
- La traduzione degli indirizzi



Le istruzioni in linguaggio macchina



- Linguaggio di programmazione direttamente comprensibile dalla macchina
 - Le parole di memoria sono interpretate come *istruzioni*
 - Vocabolario è *l'insieme delle istruzioni (instruction set)*

**Programma in
linguaggio ad alto
livello (C)**

**Programma in
linguaggio
macchina**

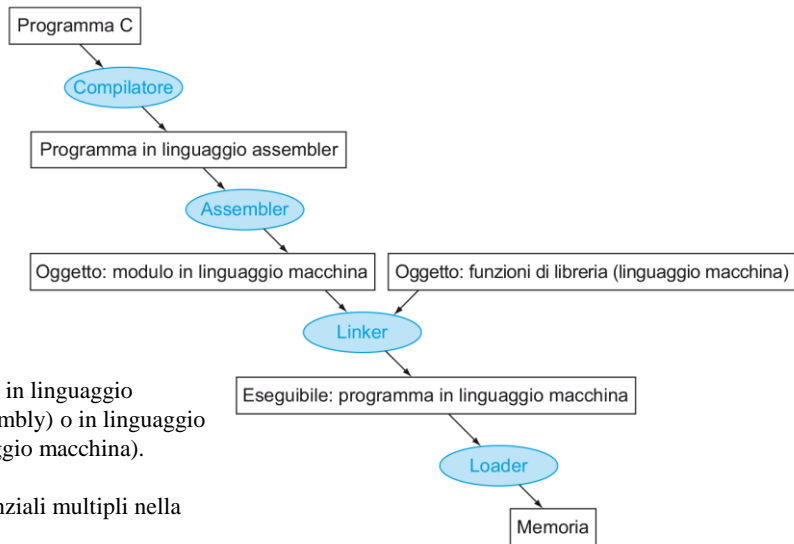
```
a = a + c  
b = b + a  
var = m [a]
```



```
011100010101010  
000110101000111  
000010000010000  
001000100010000
```



Dai simboli ai numeri binari



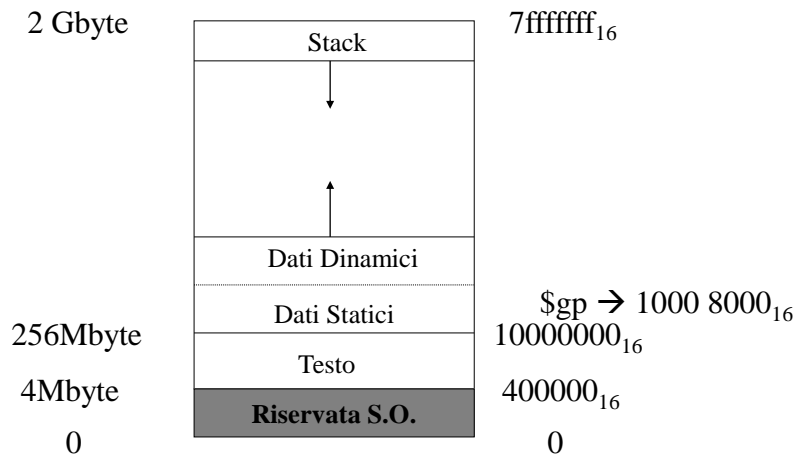
ISA esprimibile in linguaggio simbolico (assembly) o in linguaggio binario (linguaggio macchina).

Passaggi sequenziali multipli nella traduzione.

Compilazione parallela dei diversi moduli.



Organizzazione logica della memoria





Dal codice sorgente al codice Assembler



Compilatore: dal codice sorgente al codice Assembler.

Assemblatore: adatta il codice Assembler all'ISA (adattamento al «dialetto») dell'Architettura.

Assmblatore: codifica in linguaggio macchina (creazione del [file oggetto](#)), codifica binaria del codice.

Compilatore ed assemblatore possono essere uniti in un'unica fase.



L'assemblatore: i file oggetto



L'assemblaggio produce:

- L'insieme delle istruzioni in linguaggio macchina
- I dati statici
- Le informazioni necessarie per inserire le istruzioni in memoria correttamente.
- Le informazioni necessarie per inserire I dati in memoria

Un file oggetto è così costituito:

- Header. Posizione e dimensione dei vari pezzi che costituiscono il file oggetto.
- Segmento testo. Contiene le istruzioni.
- Segmento dati statici. Contiene i dati relativi al file oggetto.
- Informazione di rilocazione. Identifica istruzioni, etichette e dati che dipendono dalla posizione del programma in memoria.
- La **tabella dei simboli**. Contiene le etichette che non sono definite (ad esempio riferimenti esterni, di altri moduli oggetto o librerie).
- Informazioni di debug. Consente di associare ai costrutti Assembler ai costrutti del linguaggio ad alto livello (la traduzione non è uno a uno).



Il linker (link editor)



Consente di fare ricompilare ed assemblare solo i moduli che vengono modificati (*Rebuild*).

Il linker è costituito da 3 step:

1. Disporre in memoria i moduli di codice ed i dati (statici).
2. Identificare gli indirizzi dei dati e delle etichette delle istruzioni di salto.
3. Risolvere le etichette interne ai moduli ed esterne (trovare la corrispondenza). Questo passo è equivalente a compilare una **tabella di rilocalizzazione**.

Nei passi 2 e 3, il linker utilizza le informazioni di rilocalizzazione degli oggetti e le tabelle dei simboli.

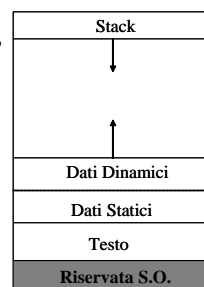
Quali sono i **simboli** da risolvere (da trasformare in indirizzi di memoria)?

- Etichette di salto (branch o jump)
- Indirizzo dei dati (e.g. A[0]).

Dopo avere risolto tutte le etichette (sostituito le corrispondenze), occorre trovare gli indirizzi assoluti associati alle etichette.

Vengono cioè **rilocati** (ripizionati) gli oggetti al loro indirizzo finale.

Viene creato il file eseguibile. Ha lo stesso formato del file oggetto ma non ha riferimenti non risolti e gli indirizzi sono assoluti (virtuali -> rilocalazione).



Esempio



Analizziamo due procedure:

Procedure A

```

0: Proc_A:  lw $a0, 8000($gp)
4:         jal Proc_B
8:         add $t2, $t1, $t0
...

```

Procedure B

```

0: Proc_B:  sw $a1, 8000($gp)
4:         jal Proc_A
...

```

NB In questo caso \$gp punta a metà del primo segmento di 64Kbyte dell'area dati (256 Mbyte + 32Kbyte)

La tabella dei simboli contiene le etichette in grassetto.

Header Procedure A

Text Size 100_{hex} (=256byte)

Data Size 20_{hex} (=32 byte)

...

Header Procedure B

Text Size 200_{hex} (=512byte)

Data Size 30_{hex} (=48byte)

Object file header			
	Name	Procedure A	
	Text Size	100 _{hex} (=256byte)	
	Data Size	20 _{hex} (=32 byte)	
Text Segment	Address	Instruction	
	0	lw \$a0, X	
	4	jal Proc_B add \$t2, \$t1, \$t0	
	
Data Segment	0	(X)	
	
Relocation info	Address	Instruction type	Dependency
	0	lw	X
	4	jal	Proc_B
Symbol table	Label	Address	
	X	--	
	Proc_B	--	

Oggetto proc A



Object file header			
	Name	Procedure B	
	Text Size	200 _{hex} (=512byte)	
	Data Size	30 _{hex} (=48 byte)	
Text Segment	Address	Instruction	
	0	sw \$a1, Y	
	4	jal Proc_A	
	
Data Segment	0	(Y)	
	
Relocation info	Address	Instruction type	Dependency
	0	sw	Y
	4	jal	Proc_A
Symbol table	Label	Address	
	Y	--	
	Proc_A	--	

Oggetto proc B

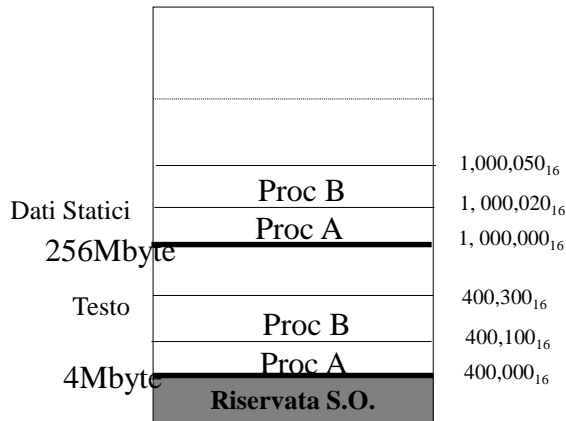


Nella programmazione modulare, il programmatore di A non sa quello che ha scritto il programmatore di B.



Il funzionamento del linker

Per prima cosa vengono posizionate nella posizione desiderata le 2 procedure (A sotto e B sopra): copia delle istruzioni e dei dati. Vengono considerate in modo assoluto -> indirizzi virtuali.



Calcoli degli indirizzi testo

Segmento testo:

Inizia dopo il segmento riservato al S.O., indirizzo 0x400 000 =
0100 0000 0000 0000 0000 0000 binario = $1 \times 2^{22} = 4$ Mbyte.

Procedura A. Inizia subito dopo. Indirizzo 0 della procedura è l'indirizzo 0x400 000.

Procedura B. Inizia dopo la procedura A. Indirizzo 0 della procedura B è: 0x400 000 +
0x100 (dimensione della procedura A) = 0x400 100.

Queste osservazioni consentono di sostituire le etichette di salto a procedura (jal).

Analizziamo due procedure:

Procedura A

```
0: Proc_A: lw $a0, 8000($gp)
4:        jal Proc_B
8:        add $t2, $t1, $t0
```

Procedura B

```
0: Proc_B: sw $a1, 8000($gp)
4:        jal Proc_A
...
```



Risoluzione delle etichette sul codice

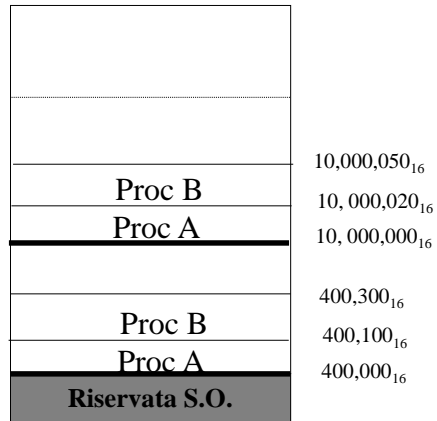


$$\text{Text size} = 100_{\text{hex}} + 200_{\text{hex}} = 300_{\text{hex}}$$

$$\text{Data size} = 20_{\text{hex}} + 30_{\text{hex}} = 50_{\text{hex}}$$

Executable file header		
	Text Size	300 _{hex} (=768byte)
	Data Size	50 _{hex} (=82 byte)
Text Segment	Address	Instruction
Proc A	400,000	lw \$a0, 8000(\$gp)
	400,004	jal 400,100
	400,008	add \$t2, \$t1, \$t0

Proc B	400,100	sw \$a1, 8000(\$gp)
	400,104	jal 400,000



Risoluzione delle etichette sui dati

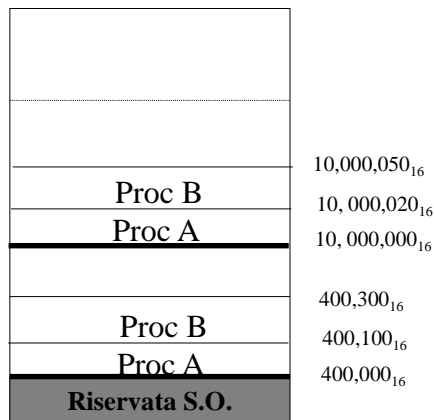


Executable file header		
	Text Size	300 _{hex} (=768byte)
	Data Size	50 _{hex} (=80 byte)
Text Segment	Address	Instruction
Proc A	400,000H	lw \$a0, 8000(\$gp)
	400,004H	jal 400,100
	400,008H	add \$t2, \$t1, \$t0

Proc B	400,100H	sw \$a1, 8020(\$gp)
	400,104H	jal 400,000

Data Segment	Address	
	10,000,000H	(X)
	10,000,020H	(Y)

$\$gp = 10,008,000_{\text{hex}} = 256\text{Mbyte} + 32\text{Kbyte}$
 $\$gp = 0001\ 0000\ 0000\ 0000\ 0100\ 0000\ 0000\ 0000$
 Il valore di \$gp è comune a tutti i moduli.





Sommario



Le virtual machine

La memoria virtuale

La traduzione degli indirizzi