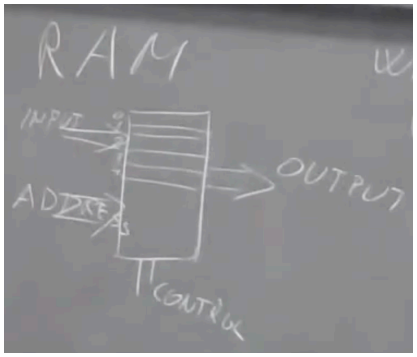
1 Memoria RAM

La memoria RAM può essere visualizzata come un dispositivo (o scatola nera) che contiene dei fili che la mettono in comunicazione con il mondo esterno e questi fili di comunicazione possono essere:

• **di indirizzamento**: quali servono per individuare una tra le n celle di memoria
• **di input**: servono per portare la codifica dell'informazione da memorizzare all'interno delle celle di memoria
• **di output**: servono per portare al di fuori del dispositivo di memoria il contenuto letto da una cella di memoria

Esistono anche dei segnali di controllo che servono per dire quali sono le operazioni da fare (scrittura o lettura). La memoria RAM è organizzata come un vettore, quindi per l'operazione di scrittura avremo: RAM[address] ← value, mentre per l'operazione di lettura avremo: output ← RAM[address].



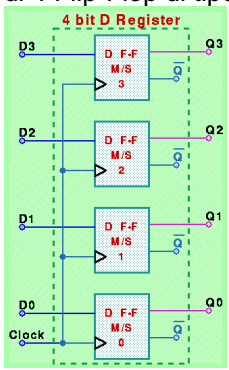
Operazione di scrittura
WRITE RAM[ADDR]←VAL (codifica binaria di un certo valore)
Operazione di lettura
READ OUTPUT←RAM[ADDR] (estrarre sui fili di uscita il contenuto di una cella di memoria)

Con la RAM si ha la possibilità di andare a memorizzare dei numeri specificando la posizione all'interno dell'array dove bisogna salvare questi numeri e di estrarli da quella determinata posizione

RAM STATICHE

Organizzazione di una memoria RAM di tipo statico

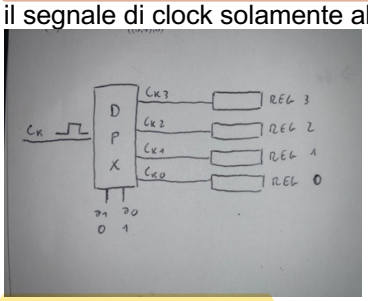
Una RAM viene realizzata con registri di tipo D che consentono di memorizzare per un tempo indefinito una combinazione di valori binari e rappresentano il tipo di semplice di circuito sequenziale sincrono.
I registri di tipo D si realizzano connettendo uno o più flip-flop di tipo D e tutti i loro ingressi CK sono comandati dalle variazioni di una sola variabile di ingresso chiamata Clock.
Nella seguente figura viene riportato lo schema di realizzazione di un registro "tipo D" a 4 bit, basato sull'uso di 4 Flip-Flop di tipo D M/S:



Nella RAM ci sono i fili di input negli ingressi D e i fili di output sull'uscita Q
Ogni flip flop memorizza un singolo bit e tutti i bit salvati nei flip flop di un registro vengono considerati come appartenenti ad una stessa configurazione binaria.
L'operazione di scrittura avviene all'interno di un registro ponendo la configurazione binaria in ingresso e poi applicando un impulso al clock in grado di memorizzare questo valore all'interno del registro che sarà sempre disponibile sulle uscite Q.

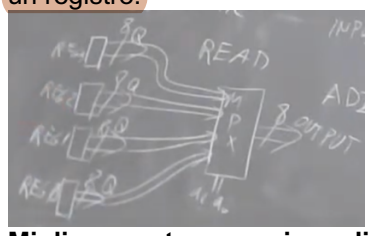
Operazione di scrittura

Il dispositivo di indirizzamento si occupa di decodificare l'indirizzo del registro che deve ricevere il segnale di clock per memorizzare il valore. Per farlo si utilizza un demultiplexer (ad esempio ricevendo in input 01 invia il segnale di clock solamente al registro 1)



Operazione di lettura

Si effettua collegando le uscite dei registri ad un multiplexer. A seconda della configurazione binaria in ingresso ricevuta attraverso i fili di indirizzamento connette l'uscita della memoria RAM statica alle uscite di un registro.



Miglioramento operazione di lettura e scrittura

Con la soluzione precedente si hanno 2 insiemi di fili di indirizzamento (uno per le operazioni di lettura ed uno per le operazioni di scrittura) e si potrebbe fare contemporaneamente un'operazione di lettura ed una di scrittura, cosa non utile.

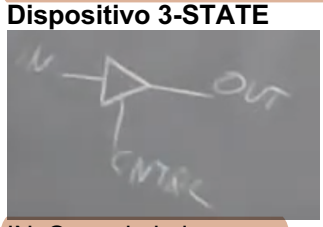
Si unificano il multiplexer ed il demultiplexer per poter economizzare siccome si supporta un unico indirizzo valido per le operazioni di lettura e scrittura siccome nella realizzazione della RAM la parte più costosa sono i fili di interconnessione.
I segnali di controllo scelgono quale delle due operazioni effettuare o se non fare nulla (quando il processore esegue operazioni che non coinvolgono l'uso della RAM)

I fili di controllo vengono standardizzati, uno si chiama CS (Chip Select) e l'altro R/ $\overline{W}$

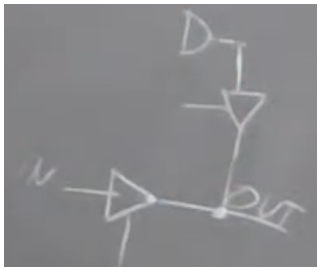
CS	R/ $\overline{W}$	
0	0	NOOP
0	1	NOOP
1	0	WRITE
1	1	READ

Riassumendo in una RAM ci sono una certa quantità di fili di indirizzamento (sempre e solo di ingresso), 2 fili per i segnali di controlli e un unico insieme di fili per il trasferimento di dati sia in input che in output.

Dispositivo 3-STATE



IN: Segnale in ingresso
OUT: Segnale in uscita
CNTRL: Segnale di controllo
In questo dispositivo l'uscita funziona come uscita solo quando CNTRL ha come valore 1, quando ha valore 0 è come se l'uscita non è collegata con l'ingresso permettendo di collegare attraverso altri dispositivi a 3 stati altre uscite senza creare interferenze (basta non dare il valore 1 a CTRL a più dispositivi contemporaneamente)



Caratteristiche della RAM

- Ottima velocità di funzionamento (memorizzare e recuperare informazioni in pochi nano secondi)
- La memorizzazione di ogni singolo bit è costosa dal punto di vista della complessità circuitale (ha implicazioni sul consumo di energia elettrica, sul riscaldamento)
- Hanno una dimensione massima limitata dal numero di transistor che implementano la memoria
- I dispositivi possono essere implementati con tecnologie diverse (cambiando la velocità ed il consumo energetico e quindi la produzione di calore)
- I dispositivi tendono a diventare sempre più piccoli, il calore sempre più concentrato e quindi sempre più difficile da estrarre
- I dati rimangono finchè non viene spento il dispositivo

Le memorie RAM statiche implementate con una tecnologia per renderle veloci tendono ad essere molto piccole

LEGGE DI MOORE

Esprime la velocità dell'avanzamento tecnologico, dice che ci può essere un raddoppio delle potenzialità di un dispositivo in termini di velocità e di dimensione ogni 3 anni. Non è stata seguita dalla velocità dai processori e dai tempi di accesso delle memorie, ma dalla dimensione si.

RAM DINAMICHE

Le memorie RAM dinamiche funzionano in maniera totalmente di versa dalle RAM statiche. I singoli bit non sono più memorizzati all'interno di un circuito sequenziale sincrono all'interno di un flip-flop ma sono memorizzati su dei dispositivi elettronici sfruttando delle caratteristiche elettromagnetiche.
La realizzazione classica è quella di usare un effetto capacitivo all'interno di un transistor ad effetto di campo. L'idea è quella di immagazzinare una certa quantità di carica elettrica e per leggere il valore inserito indirettamente misurando la quantità di corrente che può passare all'interno di questo transistor. Grazie a questo posso ridurre il numero di transistor necessari ad immagazzinare un bit.
Un altro aspetto di queste memorie RAM è che funzionano ad una velocità minore rispetto a quelle statiche, ovvero i tempi di scrittura e lettura sono più lenti. Grazie a questo possiamo ridurre anche la potenza elettrica impegnata per l'alimentazione dei dispositivi e quindi anche il calore prodotto è minore e questo permette di aumentare il numero dei dispositivi. Questi fattori fanno si che con la tecnologia dinamica si possano realizzare memorie molto superiori rispetto alla tecnologia statica. L'unico svantaggio sono i tempi di accesso. Usando la tecnologia dinamica la memorizzazione delle informazioni NON è permanente. La memoria RAM è una memoria che "tende a dimenticare".
Per risolvere questo problema si effettua un'operazione di refresh

Operazione di refresh

Per poterla effettuare bisogna conoscere per quanto tempo il dispositivo è in grado di mantenere la memoria.
Consiste nel fare un'operazione di lettura del valore prima che venga perso e successivamente di scrittura per reintegrare la carica iniziale.
Viene svolto autonomamente da un dispositivo presente nel dispositivo senza interagire con il processore.
Siccome deve essere fatta su ogni singolo bit da parte della memoria stessa richiede una complicazione dal punto di vista circuitale, infatti bisogna aggiungere un registro di tipo contatore ed un generatore di clock per scandire tutti gli indirizzi delle celle di memoria.
Durante il refresh il processore non può accedere alla RAM perché vengono utilizzati i fili di indirizzamento.

Operazione di scrittura

Viene effettuata inserendo una certa quantità di carica elettrica che rimane all'interno del dispositivo.

Operazione di lettura

Si alimenta il transistor per misurare la quantità di cariche elettriche che scorrono da una parte all'altra del transistor.

Caratteristiche

- Si riesce a ridurre la quantità di transistor necessari per memorizzare un singolo bit

- Funzionano ad una velocità inferiore rispetto alle RAM statiche consentendo di ridurre la potenza elettrica impegnata per alimentare i dispositivi
- Siccome si consuma meno corrente si possono utilizzare più transistor in un'area ristretta
- Si possono realizzare memorie molto più grandi, ma molto più lente
- La memorizzazione delle informazioni non è permanente, ma richiede di effettuare un'operazione di refresh.

RAM STATICHE	RAM DINAMICHE
Possono essere quasi veloci quanto il processore (usando una tecnologia simile a quella usata per i registri all'interno del processore) Sono fortemente limitate dal punto di vista della dimensione (troppo piccole per soddisfare le esigenze)	Sono molto più lente delle RAM statiche Hanno dimensioni molto più grandi
Si tratta di un circuito sequenziale di tipo sincrono perché salva i bit attraverso dei flip flop.	Si tratta di un circuito sequenziale di tipo sincrono perché è presente il circuito di refresh

2 Gerarchia di memoria

Consente di combinare le RAM statiche e le RAM dinamiche.
Si basa sul concetto di località nel tempo e nello spazio che rendono possibile ed efficiente l'implementazione di una gerarchia di memoria basata sull'idea di memoria cache

Principio di località nel tempo

Se in un certo momento un processore ha generato un certo indirizzo i per effettuare un'operazione di lettura o di scrittura è probabile che nel giro di pochi cicli di clock il processore generi nuovamente lo stesso indirizzo i per poter accedere di nuovo alla stessa cella di memoria

Principio di località nello spazio

Se in un certo momento il processore sta generando l'indirizzo i mi posso aspettare che successivamente venga generato l'indirizzo i+1 (quindi la cella di memoria adiacente).

MEMORIA ASSOCIATIVA

A differenza di una RAM che è organizzata come un array di celle di memoria ed ogni cella è individuata con un indirizzo, una memoria associativa è più sofisticata.
Nella RAM è presente un'associazione tra indirizzo che è costante e valore, in una memoria associativa si possono usare delle variabili per identificare una particolare cella di memoria ed accedere al contenuto.

Realizzazione

Si utilizza un dispositivo di memorie associato ad un circuito combinatorio di tipo comparatore che permette di confrontare due configurazioni binarie e stabilire se sono uguali o diverse consentendo di confrontare contemporaneamente tutti i valori presenti nelle varie celle con un valore di riferimento e capire se il valore è presente e dove si trova la cella che lo contiene (con il processore non si potrebbe fare simultaneamente su tutta la memoria, ma bisognerebbe fare una scansione sequenziale di ogni cella fino a quando non si trova il valore oppure non si ha finito la lista). Ogni cella di memoria ha bisogno di un circuito comparatore.
Utilizzando una memoria associativa il processore manda solamente un valore e l'operazione di accesso alla memoria associativa è una risposta affermativa, in caso il valore sia presente e quindi conoscendo anche la posizione, oppure una risposta negativa comunicando che il valore non è presente.
Siccome lo scopo di una memoria associativa è di velocizzare le operazioni di ricerca si utilizza una tecnologia di tipo RAM statico (e quindi ha le sue stesse caratteristiche)

COLLEGAMENTI TRA LE MEMORIE

Il processore viene collegato ad una memoria associativa statica così ha tempi di accesso brevi.
La memoria associativa statica viene collegata ad una RAM dinamica attraverso l'introduzione di un controller integrato con la memoria associativa che si comporta come il processore (fa cicli di lettura e di scrittura sulla RAM in base alle esigenze del processore).
Il processore **NON** accede direttamente alla RAM dinamica.
Il processore "crede" di essere collegato direttamente alla RAM, mentre in realtà è collegato alla cache. Il processore manda un indirizzo che appartiene alla RAM che viene preso in carico e interpretato dalla memoria cache e viene considerato come un valore che deve essere confrontato con altri valori (indirizzi della RAM) che vengono replicati all'interno della memoria cache. Nella memoria cache vengono salvati gli indirizzi corrispondenti alle celle della RAM e i valori corrispondenti a quegli indirizzi.

MEMORIA CACHE

In una memoria cache si memorizza l'indirizzo di una cella di memoria (tag) ed il suo contenuto.
Quando si trova la corrispondenza di un indirizzo con quello cercato dal processore si restituisce il valore corrispondente a quell'indirizzo.
Quando l'indirizzo cercato dal processore non è presente nella cache viene aggiunta la coppia indirizzo valore alla cache effettuando un'operazione di lettura nella RAM dinamica ed una volta completata l'operazione di lettura viene inviato il valore al processore.
Siccome il valore viene salvato nella memoria cache, se dovesse venire letto di nuovo quel valore la velocità di lettura sarebbe molto più veloce.
Il caso in cui il valore è già presente al momento della ricerca si chiama hit (in quel caso si ha un grande aumento della velocità)
Il caso in cui il valore non è presente viene chiamato miss (si accede alla RAM dinamica, quindi tempi maggiori)

Operazione di rimpiazzamento

Siccome la cache ha una dimensione limitata, lo spazio potrebbe esaurire. Se si dovessero esaurire tutte le celle a disposizione si effettua un'operazione di rimpiazzamento, ossia si prende una cella di memoria, si cancella il contenuto e si sovrascrive.
L'algoritmo cerca di sostituire le celle di memoria che probabilmente non saranno più utilizzate togliendo l'elemento utilizzato meno recentemente (algoritmo LRU – list recently used).
Questo algoritmo funziona bene con programmi che sfruttano i principi di località nel tempo e nello spazio e che usano una quantità di memoria limitata in un intervallo di tempo sufficientemente largo.

Organizzazione della cache in linee

Il principio di località nello spazio porta ad una organizzazione delle memorie cache chiamata in linee.
Una linea di cache è un insieme di indirizzi di memoria consecutivi: nel momento in cui il dispositivo di controllo della cache accede alla RAM, non accede ad una singola cella di memoria ma a tutte quelle corrispondenti a una linea di cache. (es. supponiamo di organizzare la nostra cache in linee di due celle di memoria consecutive: quando accedo ad un indirizzo di memoria se l'indirizzo è dispari accedo anche all'indirizzo pari immediatamente successivo; se l'indirizzo è pari, accedo all'indirizzo immediatamente precedente).
Le linee di cache solitamente hanno come dimensione una potenza di 2 per agevolare la gestione dell'organizzazione della cache.
Un modo semplice per implementare questa cosa consiste nel prendere gli indirizzi di memoria espressi sotto forma binaria e suddividere i bit di indirizzamento della memoria in due sottoinsiemi, gli ultimi bit di indirizzamento indicano la posizione della cella all'interno della linea di cache e i bit rimanenti indicano a quale linea di cache si fa riferimento.
Ad esempio con una cache con le linee composte da 4 indirizzi consecutivi, gli ultimi due bit di indirizzamento indicano la posizione della cella all'interno della linea di cache ed i bit rimanenti indicano a quale linea di cache si fa riferimento.

Conseguenze dell'organizzazione in linea e scelta di una dimensione opportuna

Con l'aumentare della dimensione della linea di cache aumenta la quantità di dati da spostare quando si effettua una lettura nella RAM.
Siccome per effettuare lo spostamento si usa un bus, con l'aumentare della dimensione della linea di cache bisogna aumentare la dimensione del bus (con un numero maggiore di fili) in modo da spostare i valori contenuti in più celle di memoria oppure bisogna sequenzializzare l'operazione utilizzando più cicli di clock (uno per ogni elemento contenuto nella linea, allungando quindi i tempi di accesso alla RAM).
Anche se con programmi fortemente caratterizzati dalla località nello spazio il numero di accessi alla RAM diminuiscono con l'aumentare della dimensione della linea di cache, si tende ad utilizzare linee di cache con una dimensione limitata.
Se si dovesse avere un sistema che necessita di prestazioni elevate si aumenta la dimensione del bus aggiungendo abbastanza fili per spostare con un solo ciclo di clock tutti i dati presenti in una linea.

Realizzazione pratica di un dispositivo di tipo cache

Il metodo più semplice è quello di utilizzare una organizzazione di tipo completamente associativo, ovvero ciascuna linea di cache viene tratta come un elemento separato e quindi le viene associato un dispositivo comparatore. Il campo tag corrisponde ad un sottoinsieme n-k bit a seconda della dimensione delle linee di cache. A ciascun tag vengono associati dei dati che sono l'unione del contenuto di tutte le celle di memoria RAM di indirizzi consecutivi appartenenti a quella linea di cache.

Un altro tipo di realizzazione (che consente di ridurre il numero di circuiti comparatori per semplificare la parte combinatoria del dispositivo) consiste di mettere dei vincoli sulla possibilità di inserire linee di cache nelle varie posizioni all'interno della memoria cache realizzando un dispositivo a corrispondenza diretta.

Organizzazione della cache a corrispondenza diretta

Si tratta di una memoria cache dove è presente un solo circuito comparatore per gestire tutte le linee di cache.
Continua ad essere una memoria associativa siccome si fa sempre una ricerca sul campo tag, ma questa ricerca viene effettuata solo sulla linea dove è possibile inserire l'indirizzo che si sta cercando.
Da un punto di vista pratico l'organizzazione è banale e corrisponde ad utilizzare un altro sottoinsieme di bit dell'indirizzo della memoria RAM per identificare non solo una posizione all'interno della memoria RAM, ma anche una posizione all'interno della memoria cache, quindi l'indirizzo della RAM viene diviso in 2 sottoinsiemi, uno (situato nella parte dei bit meno significativi) indica la posizione della cella di memoria all'interno della linea di cache, un altro indica in quale delle linee disponibili nella cache può essere inserita quella linea ed i restanti bit indicheranno il campo tag.
In questo tipo di organizzazione è più difficile trovare spazio libero per l'indirizzo, in quanto ogni indirizzo può finire solo in una certa linea, quindi se queste linee è occupata dovrò eseguire l'algoritmo di rimpiazzamento anche nel caso alcune linee siano vuote.

Esempio
Indirizzo: 10110110
Cache con 4 linee, ogni linea contiene 2 celle di memoria adiacenti
Siccome l'indirizzo ha uno 0 finale, corrisponde al primo elemento presente nella linea di cache
Si utilizzano altri 2 bit per trovare la linea di cache (in questo caso 11)

TAG	00	Contenuto indirizzo 0	Contenuto indirizzo 1
TAG	01	Contenuto indirizzo 0	Contenuto indirizzo 1
TAG	10	Contenuto indirizzo 0	Contenuto indirizzo 1
TAG	11	Contenuto indirizzo 0	Contenuto indirizzo 1

Nella cella rossa viene salvato l'indirizzo.
Utilizzando un solo circuito comparatore si riesce a capire se i primi 5 bit sono presenti nel campo tag.

La cache a corrispondenza diretta quindi è più facile da realizzare, costa meno, ma funziona peggio siccome utilizza in modo molto meno efficiente lo spazio (e quindi beneficerà meno delle caratteristiche di località nel tempo e nello spazio)

Memoria associativa ad insiemi

Consiste nella via di mezzo tra una memoria associativa ed una a corrispondenza diretta.
Richiede di un bit in più nel campo tag, si ha bisogno di un circuito comparatore in più per confrontare contemporaneamente due campi tag e per inserire una cella basta che una delle due possibili linee sia libera, ma se entrambe dovessero essere occupate bisogna effettuare un'operazione di rimpiazzamento.
Basandosi sull'esempio precedente si avranno due possibilità dove inserire il valore

TAG	0	Contenuto indirizzo 0	Contenuto indirizzo 1
TAG	1	Contenuto indirizzo 0	Contenuto indirizzo 1
TAG	0	Contenuto indirizzo 0	Contenuto indirizzo 1
TAG	1	Contenuto indirizzo 0	Contenuto indirizzo 1

Livelli della cache

Per poter ottenere il massimo delle prestazioni con il minimo costo si utilizzano più memorie cache.
Nel primo livello il processore è collegato ad una cache molto veloce e di piccole dimensioni. Nel secondo livello la cache è connessa con un'altra cache più grande e più lenta. Nel primo livello il processore crede di comunicare con la RAM, nel secondo livello la cache di primo livello di comporta come il "processore" mentre la cache di secondo livello come la RAM.
Attraverso i diversi livelli si arriva gradualmente dal processore alla ram. La cache di secondo livello consente di fornire un hit ratio sufficientemente elevato alla cache di primo livello per velocizzare le operazioni.
La cache di primo livello siccome è più piccola ed ha un livello di associatività superiore tende ad utilizzare il protocollo write through per minimizzare la complessità circuitale.
La cache di secondo livello può avere un hit ratio inferiore siccome non è collegata direttamente al processore, può avere un livello di associatività inferiore per aumentare la dimensione e tende ad utilizzare il protocollo write back per ridurre i tempi di accesso in scrittura.
Con l'organizzazione a più livelli si possono quindi combinare le caratteristiche dei vari protocolli di consistenza.

PROTOCOLLI DI CONSISTENZA

Mentre durante le operazioni di lettura non dovrebbero insorgere problemi riguardanti la consistenza dei dati, potrebbero insorgere durante l'operazione di scrittura siccome si tende ad effettuare l'operazione di scrittura sulla cache creando un'inconsistenza dei dati presenti nella ram.

Protocollo write-through
Attraverso questo protocollo la consistenza viene sempre mantenuta siccome quando viene effettuata un'operazione di scrittura, questa viene propagata anche alla RAM.
Utilizzando questo protocollo l'operazione di scrittura non viene velocizzata dalla cache siccome dipende dal tempo di scrittura della ram.
Siccome generalmente gli accessi in scrittura sono minori rispetto a quelli in lettura, una cache con un protocollo write-through comunque porta vantaggi all'interno del sistema siccome velocizza le operazioni di lettura.
Protocollo write-back
Viene utilizzato quando sono presenti programmi che devono scrivere frequentemente valori all'interno della memoria o se si vuole avere un fattore di velocizzazione maggiore.
L'operazione di scrittura viene eseguita soltanto all'interno della cache ed è il controllore della cache che prima o poi propagherà il cambiamento. Il controllore ripristina la consistenza quando è costretto ad eliminare quel dato dalla cache o quando termina il programma. Questa operazione di ripristino sarà costosa e dipende dai tempi di accesso in scrittura della RAM. È possibile che la CPU debba leggere un dato modificato nella memoria cache che non è ancora stato aggiornato nella RAM. Wright back garantisce un incremento di velocità quando è presente una forte caratteristica di località nel tempo, ad esempio quando è presente un contatore, infatti ritardando l'operazione di scrittura nella ram si evitano le operazioni di scrittura dei valori intermedi.
Per ottimizzare questo protocollo si inserisce un bit per ogni linea di cache che dice se il contenuto è stato modificato rispetto alla copia presente nella ram.
Utilizzando il protocollo write-back il controllore deve essere più complicato siccome deve aggiornare il bit per indicare se la linea di cache è stata modificata o meno e quindi è più costoso da realizzare.

3 Il processore e le memorie
MACCHINE CISC E RISC
CISC (Complex Instruction Set Computer) – Hanno istruzioni più complicate e quindi la realizzazione dei processori è più complicata.

RISC (Reduced Instruction Set Computer) – Hanno un numero di istruzioni ridotto, in quanto non tutte le istruzioni implementate nell'architettura CISC venivano usate. Prevedono la realizzazione di processori più performanti.

I processori RISC sono un'ottimizzazione dei processori CISC siccome durante la loro progettazione si è tenuto conto di tutta la pila della stratificazione e non soltanto del singolo livello architetturale.

Le macchine RISC conservano le stesse caratteristiche fondamentali delle macchine più antiche (ad esempio la possibilità di supportare i dati su 8/16/32/64 bit)
Effetti sulla gerarchia di memoria
Quando parliamo di indirizzi di memoria RAM, in realtà, siamo costretti ad indirizzare i singoli byte. Anche in un'architettura da 32bit o 64bit, che usano parole di nbit, queste parole vengono suddivise in byte e gli indirizzi vanno ad identificare singoli gruppi di 8bit all'interno della memoria. L'indirizzamento della memoria è sempre organizzato sui singoli byte. Questo fa si che quando parliamo del numero di bit di indirizzamento, dobbiamo fare riferimento alla dimensione della memoria non espressa in parole ma in byte. Per realizzare le operazioni di lettura e scrittura avrò bisogno di tanti bus quanto il numero di bit dell'architettura.
Ovviamente non verranno sempre utilizzati tutti, in quanto il processore ha l'opportunità di indirizzare anche le mezze parole (16 bit), un byte (8 bit), 32 bit e i long (64 bit). Per accedere a un long il processore effettuerà due accessi consecutivi a due celle di memoria consecutive, raddoppiando il tempo. Se la memoria cache è organizzata, per esempio, da linee di due parole un modo efficiente per realizzare questo sarà quello di avere il bus tra processore e cache da 32 fili di indirizzamenti, mentre il bus tra cache e RAM saranno presenti 64 fili di indirizzamenti (la differenza del numero di fili è possibile grazie alla presenza della cache perché la RAM non deve supportare il funzionamento del processore, ma quello del controller della cache).
I fili di dati sono utilizzati in modo bidirezionale, i fili di indirizzamento sono monodirezionali (dal dispositivo master verso il dispositivo slave).

4 Stratificazione in livelli dei sistemi
I sistemi si possono dividere in diversi livelli. La prima divisione si può fare tra hardware e software.
Livello hardware
1. Descrizione fisica dei dispositivi (semiconduttori, proprietà fisiche della materia)
2. Dispositivi elettronici (transistor)

3. Logica circuitale (funzioni combinatorie): impossibile dare la descrizione completa di un sistema di calcolo
4. Micro architettura (composizione con i dispositivi sequenziali e le varie porte logiche): serve per spiegare la relazione di oggetti molto complicati con altri oggetti
5. Architettura del sistema: la spiegazione del processore e della sua interazione con gli altri dispositivi

Livello software

1. Nucleo del sistema operativo: si tratta di una parte del sistema operativo ed è dove si scrivono i programmi per gestire i dispositivi fisici
2. Librerie: sono un'altra componente del sistema operativo, non dipendono dall'hardware e servono a far vedere alle applicazioni una macchina virtuale più complessa e sofisticata della macchina fisica
3. Linguaggi di programmazione
4. Applicazioni

Applicazioni	Software
Linguaggi programmazione	
Librerie	
Nucleo del sistema operativo	
Architettura	Hardware
Micro architettura	
Logica circuitale	
Transistor	
Fisica	

I vari livelli si possono dividere in altri sottolivelli siccome c'è un'arbitrarietà della strutturazione a livelli.
I livelli dipendono sia da quelli superiori, sia da quelli inferiori (ad esempio quando si realizza un nuovo processore bisogna fornire un'implementazione efficiente delle cose considerate standard ai livelli superiori.

5 Processore

Anche il processore è stratificato in livelli

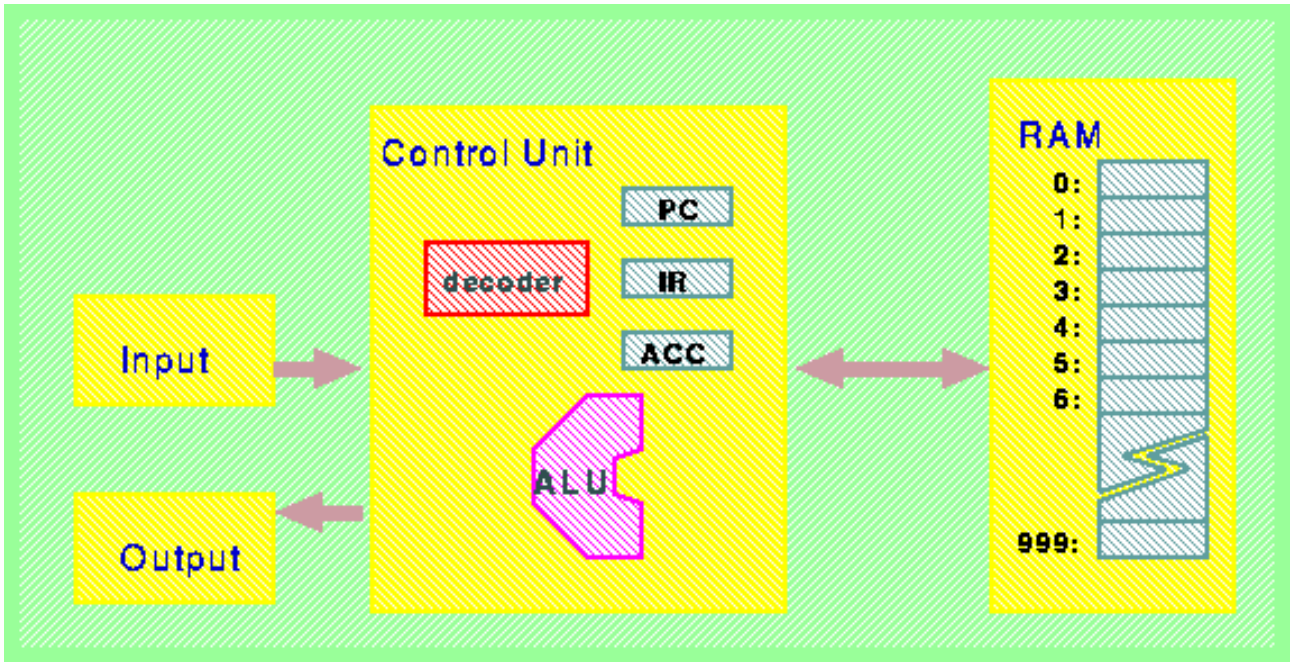
0. Logica circuitale
1. Micro architettura e trasferimento tra registri
2. Macchina convenzionale
3. Nucleo del sistema operativo
4. Assembler e librerie
5. Linguaggi di alto livello

MACCHINA DI VON NEUMANN

Facciamo qui riferimento ad una organizzazione di un sistema di calcolo effettivamente realizzato e reso operativo nell'ambito di un progetto dell'Università di Princeton sotto la guida del matematico John Von Neumann. Prescinderemo ovviamente da alcuni dettagli operativi derivanti dai limiti della tecnologia elettronica degli anni '40, oggi ampiamente superati, e ci concentreremo sulle intuizioni di base, che ancor oggi stanno dietro la progettazione e la realizzazione dei sistemi di calcolo.

Struttura della macchina

La macchina é realizzata mediante l'interconnessione di quattro dispositivi complessi: memoria, unità operativa, ingresso e uscita, come illustrato in figura:



RAM (Random Access Memory, memoria ad accesso arbitrario mediante indirizzo)
Realizza la memorizzazione di un vettore di numeri interi. Sia la dimensione del vettore (numero di elementi componenti) che il massimo valore memorizzabile in ogni elemento del vettore sono predeterminati al momento della costruzione e/o assemblaggio del dispositivo. Inteso come macchina virtuale, supporta l'esecuzione di due istruzioni fondamentali: memorizza V nella cella N; recupera il valore precedentemente memorizzato nella cella N.

Input (unità di ingresso)
Permette all'utente di interagire con la macchine (per esempio attraverso l'uso di una tastiera numerica) per l'introduzione di valori di tipo intero, uno alla volta.

Output (unità di uscita)
Permette alla macchina di stampare in un formato numerico leggibile dall'utente un valore intero per volta.

Control Unit (unità di controllo)
realizza il funzionamento della macchina secondo le modalità spiegate nel seguito. Contiene due **registri** in grado di memorizzare valori numerici interi, esattamente come le celle della RAM: un registro **accumulatore** ed un registro **contatore di programma** (Program Counter, abbreviato PC).

Per fissare le idee possiamo ipotizzare che l'unità RAM disponga di mille elementi (o celle, o parole) di memoria, e che ciascuna cella di memoria sia quindi univocamente individuata da un numero compreso tra 0 e 999. Potremmo quindi indicare con la notazione RAM[5] la sesta parola di memoria.

L'unità di controllo é realizzata in modo da poter eseguire (direttamente oppure comandando parte dell'esecuzione alle altre unità della macchina) un insieme finito di istruzioni. Sempre a titolo esemplificativo potremmo individuare un insieme di nove istruzioni base.

Una cosa irrealistica è che si prevede di usare base10 invece che base2 per la rappresentazione dei numeri (per semplificare la descrizione dell'insieme delle istruzioni e per utilizzare meno cifre)
Usa una codifica delle istruzioni basata su cifre decimali. Una cifra decimale viene usata per rappresentare una tra le istruzioni possibili e le altre 3 per rappresentare gli indirizzi delle celle di memoria (un processore con solo 1000 celle di memoria è irrealistico)
Come avviene l'esecuzione da parte della CPU delle istruzioni
L'esecuzione è divisa in 3 parti:

- Prima fase: viene indirizzata la memoria usando il program counter, si effettua un'operazione di lettura e il valore letto viene inserito nel registro delle istruzioni. Il PC viene incrementato di uno, quindi in questa fase si riporta nell'instruction register il codice operativo della prossima istruzione da eseguire (Fase di **Fetch**)
- Seconda fase: confronta il valore contenuto nel registro IR con la tabella delle istruzioni della macchina per individuare quale istruzione è codificata da tale numero. (Fase di **Decodifica**)

- Terza fase: realizza l'istruzione codificata dal valore numerico contenuto nel registro IR; alla fine torna ad eseguire il passo fetch, a meno che l'istruzione eseguita non fosse la 8. (Fase di **Esecuzione**).

Esempio di un programma

Se nell'instruction register arriva il numero 1053 questo significa che 1000 sarà l'operazione di differenza e l'operazione che farò sarà: prendo il contenuto dell'accumulatore, gli sottraggo il numero contenuto nell'indirizzo di memoria 53 e salvo il nuovo valore nell'accumulatore.

Insieme delle operazioni

- istruzione 0: somma di due numeri interi; in particolare, somma il valore memorizzato nel registro accumulatore col valore memorizzato in una cella di memoria, individuata dall'indirizzo N; Il risultato della somma viene memorizzato nel registro accumulatore al posto del primo addendo.
- istruzione 1: differenza tra due numeri interi; in particolare, sottrae il valore contenuto nella cella RAM[N] al valore contenuto nel registro accumulatore; Il risultato della differenza viene memorizzato nel registro accumulatore al posto del primo operando.
- istruzione 2: lettura di un valore numerico dal dispositivo di ingresso; Il valore letto viene memorizzato nel registro accumulatore (perdendo memoria del valore precedentemente memorizzato nell'accumulatore stesso).
- istruzione 3: scrittura di un valore numerico sul dispositivo di uscita; Il valore memorizzato nel registro accumulatore viene trasferito per la stampa sul dispositivo di uscita (senza perdere memoria del valore nel registro accumulatore).
- istruzione 4: memorizza il valore contenuto nel registro accumulatore anche all'interno della cella RAM[N] (senza perdita di memoria da parte del registro accumulatore).
- istruzione 5: copia il valore memorizzato nella cella RAM[N] anche nel registro accumulatore; il registro accumulatore perde traccia del valore in esso precedentemente memorizzato, mentre la cella di RAM[N] mantiene invariata la sua memoria.
- istruzione 6: memorizza il valore numerico N nel registro PC, perdendo quindi memoria del valore precedente;
- istruzione 7: controlla il valore numerico contenuto nel registro accumulatore; se il valore é zero allora esegui le stesse operazioni definite per l'istruzione 6, altrimenti non fare niente;
- istruzione 8: ferma l'interpretazione del programma.

Uso della memoria

Le celle di memoria possono essere usate sia per contenere il codice operativo dell'istruzione da eseguire, sia i dati che sono stati elaborati rendendo la macchina molto flessibile. Le celle tra 0 e 5 sono utilizzate per le istruzioni, mentre la cella di indirizzo 50 che viene utilizzata per contenere un dato. Con quale criterio abbiamo scelto le celle di memoria per contenere dati o istruzioni da eseguire? L'indirizzo zero, "siamo stati costretti ad utilizzarlo", in quanto è necessario per il reset del registro PC. Se non usassimo salti, l'esecuzione del programma sarebbe puramente sequenziale. Il perché abbiamo scelto la cella 50 per contenere un dato "non esiste", avremmo potuto scegliere un indirizzo qualsiasi tra 6 e 1999. Questo perché una volta deciso che il nostro programma si trovi nelle celle dalla 0 alla 5, l'unico vincolo che abbiamo è quello che il programma non venga modificato.

Il processore per capire se si tratta di un'istruzione o di un dato semplicemente segue il flusso che viene specificato dal programma stesso attuando il ciclo di fetch, codifica e decodifica. Questo perché il processore esegue tutte le istruzioni basandosi sul valore del program counter, quando il PC arriverà alla fine (in questo caso alla cella 5), ci sarà il codice di terminazione e finirà l'esecuzione del programma.

Un motivo per cui la macchina di Von Neumann è efficiente è che non bisogna per forza tenere in RAM tutto il programma, ma una volta che viene eseguita un'istruzione che si è certi che non verrà più eseguita si può sostituire quel valore con un altro.

Programma automodificante

Si tratta di un programma che dopo che viene inserito in memoria e che viene mandato in esecuzione calcola alcune delle prossime istruzioni da eseguire. Un programma automodificante può imparare da solo nuovi comportamenti inizialmente non realizzati, l'importante è che il programmatore faccia un algoritmo che sappia imparare dalla propria esperienza.

I programmi automodificanti nella macchina di Von Neumann servono per realizzare gran parte degli algoritmi necessari in un computer.

Grazie alla possibilità di scrivere programmi automodificanti si può utilizzare un insieme di istruzioni ridotto per realizzare qualsiasi algoritmo.

Un programma che può essere scritto solo con programmi automodificanti è il programma di bootstrap

Programma di bootstrap

Si tratta di un programma che fa parte del sistema operativo e permette di caricare in memoria il programma che si vuole eseguire.

Con la macchina di Von Neumann si possono programmare applicazioni, ma per inserirle in memoria si hanno due possibilità:

- Utilizzare un dispositivo hardware che permette di inserire nella RAM i valori che si vuole
- Il processore si occupa di caricarle in memoria utilizzando un programma (Quella normalmente applicata nei sistemi di calcolo)

Da un punto di vista pratico non ci sarebbero vincoli tecnologici ad includere tutto quello che abbiamo visto nella macchina di Von Neumann in un unico chip. Il motivo per cui non si fa è quello di rendere indipendenti alcuni componenti. Per fare questo si utilizza il bus di sistema, con il quale il processore comunica con gli altri componenti. L'utilizzo del bus permette di avere una macchina modulare.
Per avere un salto qualitativo del processore possiamo aggiungere dei registri. Aggiungendo registri all'interno della CPU questi si possono comportare come memoria cache di LV.0, indipendenti l'uno dall'altra

MACCHINE MODERNE

Differenza tra la macchina di Von Neumann e le macchine moderne

Sebbene le macchine moderne cambino tantissimo dal punto di vista tecnologico siccome sono estremamente complicate, restano invariate alcune cose concettuali ed è per questo motivo che si può usare la macchina di Von Neumann per comprendere alcuni aspetti delle architetture moderne

Differenze:

- Sebbene non ci siano vincoli dal punto di vista pratico e realizzativo di includere tutto quello presente nella macchina di Von Neumann in un singolo chip, si cerca di estrarre alcuni componenti dal processore per renderli indipendenti ed interscambiabili (per realizzare dispositivi con caratteristiche diverse)
- In un'architettura moderna il processore ha bisogno di altri componenti con i quali comunica attraverso l'uso del BUS, invece nella macchina di Von Neumann il processore è il sistema di calcolo
- In una macchina moderna sono presenti tanti registri all'interno del processore (influenzando l'ottimizzazione che può fare il compilatore siccome con solo un registro il lavoro di ottimizzazione fatto dal compilatore sarebbe molto limitato, invece con molti registri si possono usare come ulteriori livelli di cache)

Estendere le capacità di una macchina

L'utilizzo dei bus permette di costruire processori con una frequenza di clock maggiore ed inserirli all'interno di un sistema togliendo il vecchio processore senza cambiare il resto.
Le macchine inoltre si possono estendere diversificando i dispositivi di input/output e aggiungendo componenti che permettono a macchine diverse di interagire tra loro.
Un'altra estensione consiste nell'introdurre dispositivi di memoria di massa per avere della memoria non volatile

Sistemi embedded

Sono dispositivi che possono contenere un sistema di calcolo per aggiungere un livello di sofisticazione per rendere la macchina più efficiente (elettrodomestici, automobili, ecc...)
Sono un esempio dove si slega il processore dagli altri componenti per far si che si adatti al meglio ad una determinata necessità

Registri interni al processore

I registri interni del processore sono delle unità di memoria in grado di memorizzare un singolo valore e possono essere usati in modo indipendente l'uno dall'altro.
Il tempo di accesso è quello del clock interno del processore.
Sono realizzati con una tecnologia di tipo statico, sono indirizzabili individualmente ed in modo indipendente e per questo motivo è come avere una cache interna al processore.
In questi registri non serve usare una memoria associativa perché il programma sa quali valori sono presenti senza dover assegnare indirizzi.
Nei registri si potrebbero salvare alcune variabili come una variabile contatore e il valore di N (nel caso di for(i=0;i<N;i++)) per evitare di accedere più volte alla ram per gestire il ciclo.
I valori presenti nei registri vengono gestiti a livello software e l'algoritmo di rimpiazzamento non è l'algoritmo LRU, ma viene deciso dal compilatore.

Miglioramento macchine RISC

Già le macchine CISC avevano più registri rispetto alla macchina di Von Neumann, ma con le macchine RISC c'è stato un ulteriore miglioramento, infatti risparmiando sull'esecuzione delle istruzioni inutili si è potuto ridurre il numero di componenti elementari del processore consentendo di aggiungere un numero maggiore di registri.

Modi di indirizzamento

Siccome nel processore sono presenti diversi registri è stato necessario introdurre diversi modi di indirizzamento per poter gestire in un modo più libero ed efficiente la memoria

- Registro: il dato o operando dell'operazione è il contenuto di un registro interno al processore e quindi non viene coinvolta la gerarchia delle memorie e questo non comporta rallentamenti del processore.
- Diretto: si fa riferimento ad una cella di memoria RAM il cui indirizzo viene indicato attraverso una costante. Si può accedere ad una sola cella di memoria.
  RAM[COSTANTE]
- Indicizzato: si può realizzare solo con più di un registro e consiste nell'utilizzare come primo operando il contenuto di un registro e come secondo operando una costante. Si può usare ad esempio per gestire un array. Per accedere ad esempio a V[10] si mette 10 come valore costante e si utilizza un registro che contiene l'indirizzo del primo elemento. La somma potrebbe andare in overflow ed in quel caso si andrebbero ad indirizzare le celle iniziali della RAM (siccome la somma è in modulo con la dimensione).
  Questo modo è fondamentale siccome nei linguaggi ad alto livello sono sempre presenti gli array e quindi consente di accedervi il modo efficiente.
  RAM[REGISTRO+COSTANTE]
- Indiretto mediante registro: si accede ad una cella di memoria il cui indirizzo è contenuto in un registro. Nelle macchine RISC è inutile siccome si realizza con il metodo indicizzato usando 0 come costante.
  RAM[REGISTRO]
- Indiretto mediante cella di memoria: l'indirizzo della cella di memoria a cui si vuole accedere è contenuto in un'altra cella di memoria della ram, quindi si tratta di un metodo di indirizzamento diretto che utilizza il valore presente in un'altra cella della ram. Nelle architetture RISC si evita di utilizzarlo perché altrimenti viene meno un'altra delle condizioni delle macchine RISC, ossia avere istruzioni eseguite tutte nello stesso tempo. In un'architettura RISC si fanno due operazioni di lettura, con la prima si porta il risultato all'interno del registro e con la seconda si usa il contenuto di quel registro per indirizzare un'altra cella di memoria della RAM.
  RAM[RAM[COSTANTE]]
- Autoincremento: è simile al metodo indicizzato, ma comporta il cambiamento del valore presente nel registro (può essere anche di autodecremento). Si utilizza quando si vuole fare una scansione degli elementi di un array siccome è sufficiente caricare in un registro l'indirizzo della prima cella.
  Si utilizza anche per accedere ad uno stack (con registro++ si aggiungono i valori, con registro-- si rimuovono)
  RAM[REGISTRO++]
  RAM[REGISTRO--]

Architettura load/store

Si tratta di un'architettura introdotta con i processori RISC.
Siccome si ha a disposizione una grande quantità di registri all'interno del processore si cerca di riportare quanti più dati possibili all'interno di uno dei registri, farci sopra le operazioni necessarie e riscaricarli in RAM solo quando le operazioni sono terminate. Quindi si utilizza solamente l'indirizzamento di tipo registro. Le uniche operazioni diverse sono quelle di load, per caricare i dati, e quella di store per salvare i dati.
La maggior parte delle istruzioni quindi usa l'indirizzamento di tipo registro e successivamente vengono utilizzate le operazioni load e store per trasferire i dati dai registri alla ram e viceversa.
Questo tipo di architettura ha portato ad una semplificazione delle istruzioni introducendo un ulteriore livello all'interno della gerarchia delle memorie composto dai registri.
Le istruzioni sono quindi divise in due sottoinsiemi distinti, le istruzioni che si occupano soltanto di spostare i dati dalla memoria verso i registri e viceversa e le istruzioni che eseguono tutti i calcoli implementati dal processore all'interno dei singoli registri.

6 Memoria virtuale

La memoria virtuale è una visione della memoria RAM, da parte del processore, diversa rispetto a come questa è realmente implementata. Si tratta del primo passo verso la virtualizzazione di una macchina necessaria per poter semplificare il modo in cui viene organizzato un sistema di calcolo per far finta che ci sia solo un programma in esecuzione. Questo procedimento serve per poter usare la memoria in maniera più efficiente senza perdere la possibilità di scrivere programmi relativamente comprensibili. Nella macchina di Von Neumann un pregio che abbiamo è che possiamo sfruttare in maniera più efficiente le celle della memoria RAM, mentre un difetto è l'organizzazione della memoria dove ci farebbe comodo avere memoria separate per non mischiare dati e codice. La memoria virtuale ci permette proprio di fare questo, ovvero ci permetti di far vedere al processore tante memorie RAM separate.

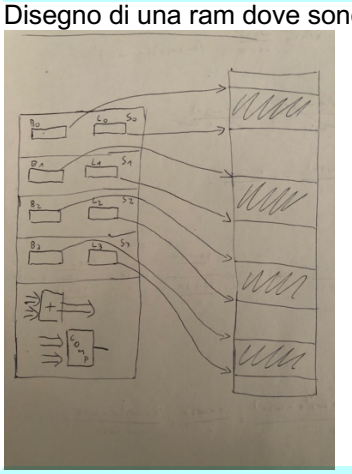
SEGMENTAZIONE

La segmentazione consiste nella ripartizione della ram disponibile in diversi pezzi separati ciascuno dedicato alla memorizzazione di un certo tipo di dato.
Generalmente viene divisa in 4 pezzi:

- Codice
- Dati statici: ossia i dati allocati dentro la memoria e che rimangono per tutta l'esecuzione del programma
- Dati dinamici nello stack: ossia i dati necessari solo in una certa porzione del programma. Si possono allocare attraverso le chiamate a delle procedure o a delle funzione siccome i parametri vengono allocati nello stack
- Dati dinamici nello heap: quando il programmatore si accorge di aver bisogno di altra memoria e quindi utilizza una chiamata di sistema per ottenerne

Funzionamento

Il processore per accedere alle celle che gli interessano utilizza un indirizzo virtuale, questo viene tradotto dall'MMU nell'indirizzo fisico corrispondente.
Il modo più facile per effettuare questa traduzione è quella di usare due registri per ogni segmento: un registro base e un registro limite.
Il registro base contiene l'indirizzo di una cella di memoria RAM e identifica la prima cella appartenente ad un certo segmento, mentre il registro limite (che può essere implementato in diversi modi) identifica la prima cella non appartenente a quel segmento.
Per ottenere l'indirizzo fisico basta prendere l'indirizzo virtuale e sommarci il registro base. Se l'indirizzo fisico non appartiene al segmento al quale voglio accedere l'MMU prende il controllo della situazione, non traduce l'indirizzo e genera una situazione di errore. Il dispositivo MMU quindi limita le capacità di indirizzamento della memoria da parte del processore evitando che eventuali errori di programmazione vadano a sovrascrivere i dati di altri programmi.
L'MMU dovrà essere dotato di un circuito sommatore e un circuito comparatore.
Disegno di una ram dove sono stati allocati 4 segmenti



Vantaggio della segmentazione

La grande semplicità a livello hardware del dispositivo di traduzione

Svantaggio della segmentazione

Sorgono degli svantaggi per quanto riguarda la gestione dinamica della memoria, infatti possono verificarsi due situazioni quando si cerca di aumentare la memoria assegnata:

- Le celle di memoria sono occupate e quindi l'MMU termina l'esecuzione del programma perché non è in grado di fare nulla
- Lo spazio ci sarebbe ma sono presenti altri dati che non si possono sovrascrivere. Se nella gestione della RAM sono stati lasciati degli spazi liberi tra i vari segmenti si può riuscire ad accontentare la richiesta agganciando altre celle di memoria libere modificando il valore nel registro limite, ma se le celle successive sono occupate bisogna spostare i segmenti per compattarli (cambiando i valori nei registri e copiando quello che è presente nelle celle di memoria). Questa operazione si chiama di riallocazione del segmento, è un'operazione costosa e che è poco efficiente dal punto di vista del sistema operativo. Per questo motivo non si usa questo tipo di segmentazione

Permessi di accesso ad un segmento

Per evitare che un segmento non acceda ad un'area non corretta, i segmenti vengono dotati di permessi di accesso dal programma in fase di esecuzione.
Attraverso i permessi di accesso è possibile restringere il comportamento di un programma impedendogli di accedere a certe aree di memoria ritenute non valide. Ad esempio il segmento codice deve poter essere

utilizzato per poter portare a termine la fase di fetch, quindi deve supportare un accesso in lettura, ma non accessi in scrittura.
Fisicamente viene realizzato aggiungendo dei bit di informazione ad ogni segmento.
Ad esempio si può utilizzare un bit per il permesso in lettura, uno per il permesso in scrittura ed uno per l'esecuzione del codice.
L'MMU durante la fase di traduzione degli indirizzi verifica anche il tipo di operazione richiesta.

PAGINAZIONE

La paginazione consente di risolvere il problema della riallocazione e consiste nel suddividere la memoria RAM in pagine, ossia in insiemi di celle di memoria consecutive. Le pagine sono tutte della stessa dimensione e solitamente la dimensione è una potenza di 2 per poter avere un numero esatto di pagine.
Quando si utilizza la paginazione l'indirizzo di memoria viene suddiviso in due parti, la prima contenente le cifre più significative indica il numero di pagina e la seconda parte indica la posizione della cella all'interno della pagina (offset).
Si avrà sempre un MMU per tradurre gli indirizzi e anche l'indirizzo virtuale mandato dal processore avrà un numero di pagina e un offset e la ripartizione dei bit sarà la stessa presente nell'indirizzo fisico.
L'MMU traduce questo indirizzo mantenendo l'offset uguale, per tradurre il numero delle pagine è presente una tabella chiamata tabella di paginazione. Il numero di pagina virtuale sarà l'indice della cella della tabella dove è presente il numero della pagina fisica.
La page table sarà salvata all'interno della ram, per velocizzare le operazioni si mette una memoria associativa (chiamata TLB, table lookaside buffer) all'interno dell'MMU che funzioni come una cache per la tabella di paginazione. Nel campo tag del TLB si inserisce il numero di segmento ed il numero di pagina, nella parte dati ci saranno i bit di permesso ed il numero della pagina fisica. All'interno del TLB si salvno aolo gli indirizzi corretti in modo da effettuare la traduzione in un singolo ciclo di clock. Si utilizza l'algoritmo LRU.
Questo sistema ci permette di allocare le pagine fisiche in modo arbitrario mantenendo l'ordinamento crescente degli indirizzi. L'impaginazione, da sola, non è ottimale ed è per questo che viene combinata con la segmentazione.

Dimensione delle pagine

La dimensione delle pagine è una via di mezzo tra due casi, infatti se si dovessero fare poche pagine di grandi dimensioni si riuscirebbe a diminuire la dimensione della tabella delle pagine (vantaggio dal punto di vista realizzativo), ma siccome il sistema operativo usa la divisione in pagine per ottimizzare l'uso della memoria conviene usare pagine piccole (siccome la dimensione della pagina è la quantità di memoria più piccola che si può allocare).

UNIONE DI PAGINAZIONE E SEGMENTAZIONE

Un limite della segmentazione è la necessità di rimuovere segmenti in caso di allocazione dinamica della memoria perché all'interno di un segmento è necessario che le celle di memoria abbiano indirizzi crescenti, in modo da poter essere ordinate come un array. Questo vincolo può essere rimosso attraverso l'utilizzo della paginazione: possiamo pensare ad una traduzione in due passi, in cui prima pensiamo alla traduzione dei segmenti e dopo passiamo alla traduzione delle pagine. Potremmo suddividere gli indirizzi logici in tre parti: la prima parte contente il numero di segmento, la seconda parte potrebbe identificare il numero di pagine e la terza parte saranno bit di offset. Sarà presente una tabella dei segmenti (segment table), con dimensione fissata, che corrisponderà al numero di combinazioni che potrò inserire nella prima parte dell'indirizzo. Per inserire le pagine all'interno dei segmenti abbiamo bisogno di tante tabelle quante il numero dei segmenti presenti. Dentro la tabella dei segmenti bisogna inserire la dimensione delle tabelle di paginazione per sapere quanto è grande quel segmento (ovvero un multiplo della grandezza della singola pagina), questo porta una limitazione ovvero l'allocazione non sarà più fatta a livello di singole celle di memoria ma a livello di pagine. La tabella dei segmenti quindi conterrà due registri dove viene salvato l'indirizzo dove inizia la page table e la dimensione di questa tabella (il numero di pagine), inoltre sono presenti sempre i bit di permesso. L'indirizzo fisico sarà composto da un numero di pagine, che sarà il numero di pagina fisica, e dall'offset. Viene rimosso il numero di segmento. Lo scopo dell'MMU è quello di prendere la coppia (numero di segmento, numero di pagina virtuale) e di trasformarlo nel numero di pagina fisica.
La segmentazione può essere implicita od esplicita. Nella rappresentazione esplicita il numero di segmento viene rappresentato da un certo numero di bit e quindi viene posto un vincolo sulla dimensione massima di un segmento. Nella segmentazione implicita, invece, scompare il numero di segmento dall'indirizzo virtuale consentendo di ampliare la dimensione massima dei segmenti che diventerà pari al numero di bit di indirizzamento utilizzati (si potrebbe utilizzare tutta la ram per un unico segmento). Nella segmentazione implicita si associa un segmento a ciascuna istruzione, ad esempio l'istruzione pop (necessaria per estrarre un elemento dallo stack) può accedere solo al segmento stack.
Supponendo di avere una segmentazione implicita, abbiamo un maggior numero di pagine disponibili. Più il numero di pagine disponibili è alto minore è la dimensione del campo offset.

7 Interrupt e trap

Interrupt e trap consentono di gestire gli errori e di virtualizzare la macchina.
Attraverso interrupt e trap si riesce a far interagire il processore con ciò che accade all'esterno.

INTERRUPT

L'interruzione è divisa in 3 fasi:

1. segnalazione: viene fatta a livello hardware e consente a qualsiasi dispositivo esterno al processore di mandare una segnalazione di errore
2. risposta: viene fatta a livello software e consiste nel capire cosa ha causato l'interruzione
3. gestione: viene fatta a livello software e ne esistono di due tipi:
    - immediata: si interrompe l'attività precedente e ne comincio un'altra per rispondere all'evento segnalato e solo dopo aver terminato questa gestione posso riprendere l'attività che stavo eseguendo.
    - differita: si continua l'attività che si stava facendo, ma prima o poi bisogna gestire l'interruzione

Implementazione

Per implementare l'interruzione posso pensare di aggiungere un filo sul bus che possa essere utilizzato per portare l'interruzione al processore. In questo modo posso dare la possibilità a qualsiasi dispositivo connesso al bus di sistema di mandare un'interruzione. Il processore, quando arriva un'interruzione, dovrà sospendere l'esecuzione del programma attuale e cominciare l'esecuzione del programma per gestire le interruzioni. Per fare ciò potremmo cambiare il contenuto del PC attraverso un dispositivo hardware che modifica il valore del pc simulando un jump alla prima istruzione del gestore delle interruzioni (interrupt handler), a questo punto si termina l'esecuzione del programma in corso e si comincia con l'esecuzione del gestore delle interruzioni. Le prime istruzioni consistono nel capire quale dispositivo ha mandato la richiesta di interruzione ed il motivo (per capire il motivo si accede ai registri presenti nel dispositivo che possono essere interrogati dal processore attraverso un ciclo di lettura).
Una volta capito il motivo viene deciso se gestire l'interruzione immediatamente o in maniera differita.
Durante la gestione immediata si eseguono dentro al gestore delle interruzioni tutte le istruzioni per risolvere l'interruzione (il processore non può riconoscere altre richieste di interruzione allo stesso livello di priorità).
Con la gestione differita il gestore delle interruzioni si conclude rapidamente (deve solamente scrivere in memoria cosa bisogna fare per risolvere l'errore) e durante la gestione differita delle interruzioni saranno abilitate le richieste di interruzione.

TRAP

Il meccanismo Trap è del tutto analogo a quello delle interruzioni, la differenza è legata al fatto di chi è che manda la richiesta dell'interruzione. Nell'interruzione è un dispositivo esterno che manda la richiesta, mentre, per il maccanismo della trap la richiesta viene mandata o dal processore stesso oppure da hardware che segnalano un funzionamento inaspettato, quindi serve per gestire gli errori (ad esempio segmentation fault, dove l'MMU riconosce un tentativo di uso della memoria anomalo). Nel caso di una trap il gestore può decidere se interrompere o meno il programma.
La segnalazione di una trap può avvenire su un solo bit, ma bisogna definire i vari tipi di trap per consentire al gestore delle trap di riconoscere la causa e decidere come gestirla.
Un caso particolare si presenta quando avviene una violazione nel segmento stack o heap, siccome potrebbe avvenire quando viene assegnato un segmento troppo piccolo e quindi il gestore delle trap assegna un segmento di dimensioni maggiori.
Tramite le trap quindi si può implementare la sicurezza del sistema sia dal punto di vista delle violazioni dovute ad errori di programmazione sia migliorare l'efficienza del sistema operativo consentendogli di allocare al programma una quantità di memoria ridotta.

IMPLEMENTAZIONE INTERRUPT E TRAP

Con l'aggiunta delle interruzioni viene aggiunto un altro passaggio al ciclo di esecuzione delle istruzioni, ovvero, viene aggiunto un controllo sul bit di presenza di interruzioni. Se questo bit è uguale a 1, allora è presente un'interruzione e il processore comincerà ad eseguire l'InterruptHandler.
Ovviamente l'esecuzione dell'InterruptHandler non può essere interrotta, così a livello hardware viene disabilitata la possibilità di mandare interruzioni. Per fare questo viene implementata la cosiddetta maschera di interruzione. Questa maschera consiste in un certo numero di bit pari al numero di bit utilizzati per la richiesta di interruzioni, se la richiesta è fatta su un solo bit, la maschera sarà di un solo bit. Questo bit sarà registrato all'interno di un registro del processore, questo bit viene messo in AND con il filo che porta la richiesta di interruzione. Quando il processore gestisce un'interruzione metterà il bit della maschera a 0 per evitare di essere interrotto. Grazie a questo abbiamo la possibilità di creare in modo funzionante il macchinismo di risposta alle interruzioni. All'interno dei sistemi vengono implementati più bit per l'interruzione assegnando quindi un grado di importanza alle varie segnalazioni, in questo modo se

un’interruzione dovesse essere molto lunga, posso interromperla e rispondere ad un’altra di importanza maggiore (si associa ai dispositivi che hanno bisogno di una risposta più rapida un grado di importanza maggiore).

Vettore di interruzioni

Consiste nell’insieme di tutti gli interrupt handler e di tutti i contenuti delle maschere di interruzione che possono essere usate in un processore per gestire livelli multipli di priorità

8 Suddivisione istruzioni del processore

L’insieme delle istruzioni è diviso in due insiemi: istruzioni privilegiate e istruzioni non privilegiate. Esistono almeno due modi di esecuzione. Questi metodi sono realizzati introducendo, oltre al bit di maschera, almeno un bit in più (se si volessero realizzare più modi di funzionamento si usano più bit). Abbiamo il bit di stato che può essere zero o uno. Se il bit di stato è zero il processore può eseguire le istruzioni non privilegiate. Le istruzioni privilegiate vengono eseguite solo se il bit di stato è uguale a uno. Il controllo sul tipo di istruzione avviene durante la fase di fetch. Per dire che non posso eseguire una certa istruzione genero una Trap. Le istruzioni privilegiate possono avere accesso diretto alla memoria, mentre le istruzioni non privilegiate sono costrette a mandare indirizzi virtuali in modo che questo venga filtrato dall’MMU (in caso di errore manda un segnale di trap al processore).

Quando viene generata una trap il processore inserisce il valore 1 nel registro di stato in modo da consentire al processore di eseguire anche le istruzioni privilegiate.

Fase di boot

Al momento dell’accensione il processore ha accesso alle istruzioni privilegiate, il programmatore deve tenere conto che non può utilizzare le istruzioni di accesso alla memoria virtuale e quindi deve preparare le tabelle di paginazione e di segmentazione. Dopo aver fatto questo, i programmi di interruzione di interrupt e trap vanno allocati nella memoria. Soltanto dopo aver inizializzato il meccanismo di traduzione degli indirizzi si può fare in modo che il processore azzeri il bit di privilegio. Da qui in avanti il processore non potrà più accedere alla memoria direttamente. Questo fa in modo che in caso un programma volesse utilizzare istruzioni privilegiate, venga bloccato dalla generazione di una trap.

9 Gestione dei segmenti

SEGMENTO STACK

Il segmento Stack serve per poter contenere dei dati. Viene implementato all’interno del processore grazie a due registri: lo stackPointer, che serve per identificare l’estremo superiore dello Stack e il framePointer che determina la fine dello Stack. Bisogna specificare per SP (e anche FP) se è l’ultima cella prima del segmento Stack o la prima cella di esso (per FP se è l’ultima cella dello Stack o la prima cella dopo il segmento). Generalmente si utilizza una convenzione diversa per i due registri: all’interno dell’SP si mette l’indirizzo più piccolo meno 1, mentre all’interno del FP si mette l’indirizzo più grande. All’interno dello Stack inseriamo una copia dei valori di alcuni registri del processore, i registri che possano far riprendere l’esecuzione del programma dopo un’interruzione. Per questo servirà, sicuramente, salvare il valore del PC. Una volta fatto questo posso modificare lo Stack, creando uno Stack adiacente, negli indirizzi superiori. Quando si crea uno stack adiacente bisogna memorizzare il valore del registro FP. Una volta fatta questa operazione bisogna cambiare i valori di SP e FP. Per inserire dei valori all’interno dello Stack utilizzo l’operazione push che si basa su un indirizzamento di tipo autoincremento di tipo valore del registro SP. Quindi si può fare un push di un valore e successivamente viene decrementato il valore di SP. Per accedere al contenuto dello Stack si utilizza un metodo indicizzato. Si potrebbe utilizzare anche la funzione pop.

Un programma può venire interrotto quando si riceve un interrupt o una trap. In entrambi i casi i dati vengono comunque salvati, siccome il meccanosimo hardware che implementa le trap non può sapere se il programma verrà ripreso o meno.

Lo Stack viene anche utilizzato anche per contenere altri tipi di interruzioni, ad esempio la chiamata di una procedura: è un’istruzione particolare che il programma stesso si dà per dire quale parte di codice deve essere mandata in esecuzione. In questo caso non viene cambiato il bit privilegio ed è il programma stesso a gestirla. Anche con questa funzione, chiamata call, vengono salvati tutti i registri necessari per fare riprendere l’esecuzione del programma.

Per motivi di efficienza i frame successivi dello stack sono considerati appartenenti allo stesso segmento stack per evitare di manipolare troppo spesso le tabelle di paginazione e di segmentazione e quindi non sono sotto il controllo dell’MMU causando dei problemi di sicurezza legati allo stack (attacco chiamato buffer overflow o stack overflow).

Attraverso questi attacchi si potrebbe modificare il valore che deve essere ripristinato all’interno del program counter causando un cambiamento nella normale esecuzione del codice.

INCREMENTO SEGMENTO STACK E SEGMENTO HEAP
Il segmento heap viene gestito in modo dinamico aumentando gli indirizzi a disposizione.
Mentre lo stack viene organizzato partendo dal minimo indirizzo disponibile ed incrementando gli indirizzi, per aumentare la dimensione dello heap avviene il contrario.
Si possono aumentare le dimensioni di stack ed heap finchè il valore dello SP rimane minore dell'ultimo indirizzo presente nello heap (e viceversa).
Lo stack quindi viene gestito interamente dal sistema attraverso l'MMU che determina le condizioni di errore e come comportarsi, nello heap si segnala la condizione di errore all'applicazione e sarà poi l'applicazione a decidere come comportarsi.

10 Sistema operativo

Il SO è un software che è in grado di utilizzare direttamente l'hardware e di interfacciarsi con l'utente.
L'innovazione più grande è stato il sistema operativo Unix, con concetto fondamentale di semplificare sé stesso, infatti è stato il primo scritto con un linguaggio ad alto livello. Un modo intelligente per considerare il SO è quello di considerarlo come un qualsiasi programma. Quando si costruisce il meccanismo di virtualizzazione bisogna tenere conto che all'inizio questo non funziona. Per uscire da questo impasse, devo essere in grado di riuscire a virtualizzare una macchina, se riesco a fare questo posso scrivere l'SO per la macchina virtuale attraverso un meccanismo di emulazione.

11 Virtualizzazione con unità disco

È un dispositivo abbastanza semplice, costituito da un materiale che è possibile magnetizzare. Oltre a questo, è dotato di una testina di lettura/scrittura che era in grado di leggere le varie magnetizzazioni. I dischi magnetici sono divisi in cerchi concentrici in modo da avere la possibilità di immagazzinare più dati. L'unità disco consente di immagazzinare una quantità di dati molto superiore rispetto alla ram. Grazie all'introduzione delle memorie di massa è nata la possibilità di estendere la memoria virtuale all'unità disco consentendo di far vedere una quantità di ram maggiore rispetto a quella realmente disponibile salvando parte dei dati nella ram ed i restanti nell'unità disco e sfruttando le caratteristiche di località nel tempo e nello spazio.
Implementazione
L'unità base della gestione della memoria virtuale attraverso l'unità disco è la pagina, quindi la quantità minima di dati spostati è una singola pagina.
Durante l'allocazione dinamica invece di guardare la disponibilità di un'ulteriore pagina all'interno della ram, si guarda la disponibilità all'interno dell'unità disco.
Per implementare questo tipo di paginazione, chiamata paginazione a richiesta, possiamo partire da un'allocazione di segmenti vuoti tranne quelli del codice e del programma.
Se il programma accede solo ai dati statici funziona il normale meccanismo di caching accedendo alla ram attraverso l'MMU.
L'unica differenza è che se il processore accede per la prima volta ad una pagina, quella pagina si trova all'interno dell'unità disco, quindi viene segnalata una trap e se il gestore della trap trova quella pagina sul disco la copia in una pagina libera della ram e la associa al segmento corretto.
Se il processore dovesse accedere ad un segmento dinamico si verifica una trap, il sistema operativo guarda se c'è spazio nella memoria virtuale (se ci sono pagine libere nella parte dell'unità disco dedicata alla memoria virtuale) ed in caso aggiunge una nuova pagina al segmento corretto e la aggiunge anche all'interno della ram.
Per mostrare una memoria ram di dimensioni maggiori si utilizza l'algoritmo LRU per rimpiazzare le varie pagine (si potrebbe rimuovere dalla ram anche la pagina di un altro programma in esecuzione).
Virtualizzazione macchine moderne
La maggior parte delle applicazioni non tollera più questo tipo di virtualizzazione a causa dei lunghi tempi di accesso all'unità disco, ma tendono a fare un uso intelligente della memoria salvando i dati su file.
La virtualizzazione viene mantenuta per rendere la memoria persistente per poter ripristinare lo stato della macchina dopo lo spegnimento aumentando l'affidabilità della macchina a discapito della sicurezza (si possono recuperare i dati presenti nella ram allo spegnimento)

SCOPO DELLA VIRTUALIZZAZIONE
1. Eseguire un programma senza rischiare di fare danni all'interno del sistema
2. Mandare in esecuzione più programmi contemporaneamente suddividendo le risorse disponibili

DIFFERENZA TRA EMULAZIONE E VIRTUALIZZAZIONE
Virtualizzazione: si ha una macchina fisica e si virtualizza quella stessa macchina fisica usando memoria virtuale, segmentazione, il meccanismo di interrupt e trap e la partizione delle istruzioni in privilegiate e non privilegiate.

Emulazione/simulazione: si parte da un sistema virtualizzato e ci si chiede se si dovesse virtualizzare una macchina fisica diversa (ad esempio con istruzioni in più) cosa bisognerebbe fare, se è possibile partire da un processore fisico ed arrivare ad un processore virtuale che usando quel processore fisico faccia finta di avere un processore fisico con più operazioni.

VEDERE FILE STORIA VIRTUALIZZAZIONE

12 Interconnessione di tipo bus

Il BUS è la componente che permette di far interagire tra di loro i vari componenti fisici consentendo lo spostamento delle informazioni. Il BUS dal punto di vista fisico può essere realizzato con un insieme di fili di interconnessione e di connettori. L'interazione richiede un coordinamento a livello logico, per questo si utilizza l'interazione master/slave: il dispositivo master è quello principale, considerato come attivo, questo dispositivo ne identifica un altro chiamato slave con il quale si coordinerà. Il dispositivo master sceglie anche la direzione dai dati. Nelle architetture moderne il componente master è la CPU mentre il componente slave è la memoria RAM. Il trasferimento dei dati non è immediato a causa del tempo di propagazione del segnale ed al tempo di funzionamento dei dispositivi elettronici.
I BUS adottano protocolli che possono essere sincroni o asincroni:

- Asincrono: possiamo pensare di avere i due dispositivi, master e slave. Questo protocollo segue l'idea di poter permettere al dispositivo master di mandare dei comandi allo slave e di ricevere risposta da quest'ultimo, nel minor tempo possibile data la dimensione del mio dispositivo. Il modo più facile per implementare questo protocollo è quello di aggiungere due variabili: una scritta dal master e letto dallo slave e una scritta dallo slave e letta dal master. Questo implica che per realizzare questo protocollo servano due fili, uno che porta le informazioni dal master allo slave e viceversa. All'inizio il M (master) non ha bisogno dello S (slave) e quindi ci sarà una variabile MA = 0 e una variabile SL = 0. Quando il M ha bisogno dello S cambia il valore della variabile MA da 0 a 1. Quando lo S termina l'operazione richiesta cambia la variabile SL da 0 a 1 in questo modo il M sa che lo S ha finito. A questo punto il M cambia la variabile MA da 1 a 0, quando lo S vede che MA = 0 allora pone SL = 0.
  Le variabili sono salvate all'interno di un flip flop.
  Il master può dare solo un ordine alla volta allo slave.
- Sincrono: Lo scopo di questo protocollo è quello di limitare al minimo i tempi di propagazione (un ciclo di lettura potrebbe impiegare meno tempo rispetto a quello di lettura). L'idea di questo protocollo si basa sul non usare il tempo di propagazione sul basa, bensì conoscere il tempo di propagazione e basarsi su questo per creare un sistema il più veloce possibile basato su un clock (segnale periodico di durata predefinita che permette di scandire il tempo). Questo protocollo viene creato conoscendo l'effettiva velocità dei dispositivi, quindi è un protocollo più limitato. Il clock dei BUS è diverso da quello della memoria o quello del processore.
  I bus sincroni richiedono lo stesso numero di fili siccome il clock sostituisce il filo ACK. È più veloce perché si riducono i tempi di propagazione sul bus.
  Il problema di questo tipo di bus è che il numero di cicli di clock deve essere il massimo tra tutti i tempi di ritardo possibili, per risolvere il problema si reintroduce il segnale di risposta da parte dello slave
- Sincrono con segnale di risposta: il clock viene messo in aggiunta e non sostituisce il filo di ACK che prende il nome di WAIT. Con il filo di WAIT si da la possibilità allo slave di rispondere al master per segnalare se ha finito o se ha bisogno di ulteriori cicli di clock consentendo di dimensionare la frequenza di clock per ottimizzare il dispositivo più veloce.

Nei sistemi di calcolo moderni i bus sono di tipo sincrono
Guardare il file con gli appunti sui bus per esempi

13 Dispositivi input/output

Il problema principale per quanto riguardo l'inserimento di questi dispositivi in un sistema è quello di inserirli in questo contesto in una comunicazione di tipo master/slave. Supponiamo di avere una tastiera e dobbiamo leggere un carattere: un modo potrebbe essere che la CPU interroghi la tastiera in attesa di un carattere, ma il carattere deve essere inviato dall'utente siccome non è salvato in un registro. Spesso capita che la CPU richieda il carattere ancora prima che l'utente l'abbia schiacciato, per questo ci sono due tipi di attese che vengono implementare per poter mantenere la sincronizzazione:

1. Attesa attiva: si presuppone che il programma possa andare in esecuzione molto più velocemente rispetto ad esempio all'utente che schiaccia i pulsanti e quindi è normale che il programma perda del tempo rimanendo in attesa. Il programma continua a fare richieste al dispositivo di input finché non riceve il valore.

2. Interruzioni: durante l'attesa il programma non è in esecuzione. Quando l'utente schiaccia il pulsante il dispositivo manda una segnalazione di interruzione al processore, il programma in esecuzione in quel momento viene interrotto e si manda in esecuzione il gestore delle interruzioni. Il gestore delle interruzioni interroga il dispositivo di input e salva il valore ottenuto all'interno di una zona di memoria accessibile dal programma. Finché non si hanno tutti gli input richiesti il programma resta in attesa e si eseguono altri programmi. Quando si ricevono tutti gli input viene mandato in esecuzione il vecchio programma che completa la sua esecuzione. Il vantaggio di questo tipo di attesa è che non si tiene impegnato il processore in attese inutili.

Solitamente si utilizza il meccanismo con le interruzioni.
L'attesa attiva si usa solo quando la velocità dei dispositivi è talmente elevata che una soluzione basata sull'interruzione rischia di essere troppo lenta.

Implementazione a livello hardware

L'interconnessione dei dispositivi di input si può realizzare in due modi:

1. È possibile aggiungere all'interno del processore delle istruzioni dedicate per i dispositivi in/out. Queste istruzioni devono essere privilegiate per far si che se ci dovessero essere 2 programmi che hanno bisogno dello stesso dispositivo sia il sistema operativo ad occuparsene e a decidere come distribuire gli input e consente di non virtualizzare i dispositivi di input. Questo metodo non viene utilizzato perché troppo complesso
2. Un'altra soluzione sarebbe quella di utilizzare dei registri all'interno del dispositivo di input il quale contenuto può essere letto (per ricevere i risultati) e scritto (per dare ordini) dal processore. Per fare questo possiamo mappare questi registri in memoria: non occupiamo tutta la RAM con celle RAM, ma mettiamo all'interno delle celle mancanti i registri dei dispositivi in/out. In questo modo il processore vede questi registri come celle di memoria e può fare le operazioni di lettura e scrittura in memoria (non sono privilegiate). Se il sistema operativo volesse non virtualizzare il dispositivo mappa il registro di quel dispositivo all'interno di un segmento accessibile in lettura/scrittura dal programma. La decisione se virtualizzare o meno viene presa dal dispositivo.

Nei sistemi moderni si usano i registri mappati in memoria.

Esempio di funzionamento

Un programma deve ricevere in input dei tasti schiacciati sulla tastiera da un utente
Quando il programma deve analizzare un carattere in input accede ad un buffer nella memoria attraverso un'operazione di lettura.
Se all'interno del buffer c'è un valore il programma va avanti, altrimenti scrive all'interno del buffer un valore per indicare che non è stato schiacciato nessun tasto ed esegue una trap. In questo modo il programma entra in attesa e si eseguono altri programmi finché non viene schiacciato un pulsante. Quando il dispositivo di I/O riceve un input manda una segnalazione di interruzione (usando dei registri mappati in memoria ed associati al dispositivo di I/O durante la fase di bootstrap). Una volta ricevuta l'interruzione il processore cerca il dispositivo che ha mandato la richiesta ed inserisce il carattere ottenuto in un buffer che virtualizza il dispositivo. L'eventuale carattere fine dell'input si va a scrivere in una allocazione di memoria consecutiva all'interno del buffer. Il dispositivo di input virtuale viene letto nel momento in cui è presente il valore, quindi la virtualizzazione di questo dispositivo comporta la sincronizzazione del programma applicativo con il funzionamento del dispositivo fisico sostituendo l'attesa attiva.

Ottimizzazioni

Se si dovesse ricevere un solo carattere non vengono fatte ottimizzazioni.
Se si dovessero ricevere messaggi di dimensioni molto più grandi potrebbe essere necessario effettuare tante operazioni di lettura del registro (siccome i registri possono contenere una quantità limitata di dati). Questa operazione va fatta nel minor tempo possibile per non perdere informazioni (finché non si svuota il buffer non si possono ricevere altri messaggi). Per evitare che il processore rimanga impegnato per troppo tempo nella gestione dell'interruzione si introduce a livello hardware il DMA (direct memory access).
Il DMA è un microprocessore che può eseguire una parte delle istruzioni del processore. Il suo compito è di effettuare le operazioni di trasferimento dei dati da un'allocazione di memoria ad un'altra. Per realizzarlo si introducono dei registri nel DMA per indirizzare la memoria RAM (contengono l'indirizzo iniziale e l'indirizzo finale del buffer, l'indirizzo del registro del dispositivo di input/output). Il DMA viene programmato dal processore quando riceve la richiesta di interruzione inserendo i valori corretti nei registri (mappati in memoria). Durante la programmazione il processore funziona come master ed il DMA come slave, ma quando effettua le operazioni di lettura e scrittura diventa master, prende il controllo del bus e la RAM diventa slave. Quando termina la fase di programmazione si conclude la gestione dell'interruzione. Quando il DMA termina la copia invia una richiesta di interruzione al processore.
Il problema di questa soluzione è che si hanno due dispositivi (processore e dma) che possono diventare master sul bus. Per risolvere questo problema si utilizza la cache siccome quando il processore deve accedere alla cache non è in competizione con il DMA. Solo il controllore della cache può entrare in competizione con il DMA in caso di miss.

Miglioramento del DMA

Siccome ogni trasferimento corrisponde a due cicli sul bus, si potrebbero evitare i cicli di lettura inserendo il DMA all'interno del dispositivo di input/output.
In questo modo il dispositivo di input/output può comportarsi come master e come slave.
Dal punto di vista del processore cambia solo l'indirizzo mappato in memoria.
Questa struttura si chiama DMA BUS Mastering. Il problema di questa struttura è che se si dovessero avere tanti dispositivi di input/output bisognerebbe gestire la possibilità di avere tanti master che competono per l'uso del bus. Si tende ad utilizzare un numero limitato di dispositivi bus mastering e siccome inizialmente erano previsti 2/4 canali DMA, non si complica la situazione dal punto di vista circuitale.

Ulteriore miglioramento del DMA

Per programmare il DMA il prima possibile, questo viene fatto nella fase di bootstrap. In questo modo si elimina la prima richiesta di interruzione (quella necessaria per programmare il DMA). A livello di interconnessione non cambia nulla, ma bisogna dividere la RAM in sistema e user. Nell'area di sistema il processore alloca un certo numero di buffer ed una struttura dati condivisa (quindi può essere letta e scritta sia dal processore che dal dma) chiamata ring. Nel ring si inseriscono due indicatori di posizione per indicare il primo (corrisponde al primo elemento libero) e l'ultimo (corrisponde all'ultimo elemento usato) elemento inserito all'interno dell'anello. Nell'anello sono inseriti degli elementi che puntano ai vari buffer. Oltre alla creazione del ring e all'allocazione dei buffer, durante la fase di bootstrap il processore mappa i registri nel dispositivo di input/output inserendo dove si trova la struttura dati ring. Durante la fase di bootstrap il dispositivo di input funziona come slave, una volta terminata funziona sempre come master.
Quando il DMA riceve un messaggio autonomamente va a cercare il buffer puntato dal primo elemento libero dell'anello e copia il messaggio dentro il buffer. Aggiorna il flag dell'anello e invia un'interruzione alla cpu, il processore riceve l'interruzione e dopo aver verificato il dispositivo che l'ha mandata estrae il contenuto del buffer di ricezione togliendolo dall'anello, lo utilizza e successivamente lo rende di nuovo disponibile per la scrittura.
La base del funzionamento di questo meccanismo è la memoria condivisa che consente l'interazione tra processore e dispositivo di input.
Questa struttura si chiama RING-BASED DMA.
Il problema di questa soluzione è che aumentano i costi del dispositivo siccome deve contenere un processore ed un firmware.
Per inviare i dati dal processore al dispositivo (in questo caso probabilmente di output) il meccanismo è lo stesso, il processore riempie il buffer di trasmissione e lo inserisce nell'anello. Il DMA quando vede un nuovo buffer nell'anello inizia a fare la copia dei dati.
Questa soluzione si può usare anche con le unità disco.
Nella fase di bootstrap nel programmare il DMA dovremo usare gli indirizzi fisici per dire dove si trova il Ring e non gli indirizzi virtuali.
Il vantaggio di questa soluzione è che rende indipendente la velocità del dispositivo dalla velocità del processore in risposta alle interruzioni.
Per terminare l'esecuzione di copia dei valori ricevuti in input bisogna trasferire i dati presenti in una zona accessibile solo dal sistema operativo in una zona accessibile dalle applicazioni. L'esistenza della memoria virtuale quindi è fondamentale e permette di virtualizzare i dispositivi. Per questo motivo è necessario avere l'MMU tra la cpu ed il bus (il processore vede gli indirizzi virtuali, il DMA quelli fisici).
Un altro problema è mantenere la consistenza dei dati, infatti solo la prima volta i dati verrebbero presi dal buffer, ma successivamente sarebbero presenti nella cache. Il protocollo write-through (e quello write-back) mantiene la consistenza quando solo il processore scrive all'interno della RAM.
La soluzione più semplice è assegnare ad ogni pagina un bit per indicare se è condivisa da più processori. Una pagina condivisa non viene inserita nella cache (non cacheable) e quindi ogni volta che si prova ad accedervi si verifica un miss.
Un'alternativa più complicata da realizzare e più costosa) sarebbe inserire nella cache una copia dei dati provenienti dal dispositivo di input

14 Ottimizzazione del processore (pipelining)

Consente di migliorare la velocità di esecuzione del codice
È complicato portare a termine tutte le 4 fasi di un'istruzione in un singolo ciclo di clock. La soluzione è la tecnica di pipelining.
Se noi pensiamo di suddividere le nostre istruzioni, supponiamo una sequenza di tre istruzioni, ciascuna di queste è divisa in quattro fasi. Possiamo suddividere la CPU in 4 "pezzi" in modo che ognuno di esso faccia un'operazione diversa e che ognuno lavori autonomamente.
Suddividendo il processore in questo modo possiamo eseguire le istruzioni non più in maniere sequenziale, ma un'esecuzione in pipelining. Questa organizzazione ci fa guadagnare un compattamento, se prima

avevamo bisogno di 12 cicli di clock adesso ne bastano solamente 6. Questo perché, nel primo ciclo di clock eseguo la fase 1 di I1, mentre nel secondo ciclo eseguo la fase 2 di I1 e la fase 1 di I2, e così via.
La singola istruzione richiede sempre 4 cicli di clock, ma si riesce ad aumentare la frequenza di completamento delle istruzioni (se ogni istruzione richiede 4 cicli di clock, a partire dalla quinta istruzione ogni ciclo di clock si completa un'istruzione).
Il problema più grave a cui si può andare in contro è legato all'esecuzione di programmi non sequenziali: in caso fosse presente un'istruzione di salto devo abortire il flusso della pipeline e aspettare il prossimo ciclo di clock per eseguire l'istruzione corretta. Se non faccio in questo modo potrebbero avvenire errori gravi.
Quindi tutte le volte che trovo istruzioni di salto devo svuotare tutta la pipeline per poter eseguire l'istruzione giusta.
Per avere una migliore efficienza della struttura pipeline è stata modificata la semantica di salto, ovvero è stato introdotto il concetto di salto ritardato: il salto avviene dopo un certo numero di clock costringendo il compilatore ad anticipare le istruzioni di salto. Per convertire un programma con i salti normali in uno con i salti ritardati basta scambiare l'istruzione del salto con l'istruzione precedente. L'istruzione che dovrebbe essere eseguita prima si chiama delay slot.
Se l'istruzione è di salto condizionale le cose cambiano in quanto sino all'ultimo non si sa se bisogna fare il salto (si scopre se bisogna fare il salto al termine dell'esecuzione dell'istruzione, i salti non condizionali vengono riconosciuti alla fine della fase di decodifica). Se il salto è condizionale non è più sufficiente il delay jump. A questo punto si introduce la tecnica della predizione dei salti (branch prediction). Questa tecnica si basa sul valutare in termini di probabilità se il salto deve venire eseguito oppure no. Il processore scommette dei cicli di clock, se vince l'esecuzione continua senza ritardi, se perde viene svuotata tutta la pipeline.
Per prevedere l'esito di un salto si può tenere conto di ciò che è successo in passato (complicando la struttura interna della cpu, siccome deve mantenere in memoria quello che è successo le volte precedenti) o facendo prevedere se bisogna fare il salto o meno al compilatore (che riesce a tenere conto della struttura del linguaggio ad alto livello). Il compilatore inserisce due tipi di salti diversi per indicare se bisogna fare il salto o meno.

15 Laboratorio

Makefile
Si tratta di un file che viene letto e interpretato dall'applicazione make.
Consente di creare delle dipendenze tra i vari file, ad esempio:

```
string_master: string_master.c
        gcc -o string_master -DMY_SOURCE_FILE='"string_master"' string_master.c
```

Se esiste il file string_master.c viene fatta l'istruzione all'interno creando il file string_master.
Potrebbero esserci anche dei casi in cui il file non viene creato, ad esempio:

```
test_master : string_master gc_string_master
        ./$(SCRPT) $(CMP) $(M) m
        ./$(SCRPT) $(CMP) $(M) f
        ./$(SCRPT) $(CMP) $(M) p
        ./$(SCRPT) $(CMP) $(M) fp
```

In questo caso test_master non viene creato.
Per utilizzare test_master bisogna fare il comando make test_master (per essere eseguito devono esistere i file string_master e gc_string_master.
test_master prende il nome di target.
Se tutti i file sono presenti l'applicazione make guarda la data di creazione o ultima modifica dei file.

Durante la compilazione se si fa -D stringa è possibile dire al compilatore il valore di alcune costanti. Serve per configurare un programma senza modificare il sorgente. Consente anche di eseguire parti di codice diverse:

```
#ifdef A32
codiceA
#else
codiceB
#endif
```

Se la costante A32 è definite viene eseguito il codiceA altrimenti viene eseguito il codiceB
Little endian
Si inizia con la cifra meno significativa, quindi si inserisce nella prima posizione il byte numero 0, poi il byte numero 1, ecc... char a={'c','i','a','o',0}

0	1	2	3
'c'	'i'	'a'	'o'
0	?	?	?
4	5	6	7

Gli indici crescono da sinistra verso destra

Big endian

Si mette il byte meno significativo a destra, il byte più significativo a sinistra ed in memoria produce una struttura del seguente tipo:

char a={‘c’,’i’,’a’,’o’,0}

3	2	1	0
‘o’	‘a’	‘i’	‘c’
?	?	?	0
7	6	5	4

La stringa quindi va letta da destra verso sinistra.

Gli indici crescono da destra verso sinistra

Variabili di ambiente (environment)

Le variabili di ambiente sono variabili presenti nella shell che consentono di comunicare ulteriori informazioni ai programmi mandati in esecuzione.

Per passarle si usa envp

int main(int argc, char** argv, char ** envp)

L’ultimo valore deve puntare a null per riconoscere la fine dell’array.

Allocazione dinamica

Non è possibile allocare un solo byte, ma viene allocata una parola per poter mantenere allineati gli indirizzi.

Quando viene allocata una quantità di memoria, si restituisce il puntatore alla prima cella di memoria e viene salvato in una tabella nel sistema operativo la dimensione ed il puntatore alla prima cella (motivo per cui quando si fa la free non si possono passare puntatori diversi dall’originale).

L’alternativa all’uso della tabella è di salvare la dimensione nella prima parola disponibile e restituire il puntatore alla parola successiva.