ARCHITETTURA DEGLI ELABORATORI

Appunti ed Esercizi
corso tenuto da ALESSANDRO BOGLIOLO

appunti di
Stefano Zizzi

Università degli Studi di Urbino
Informatica Applicata

A.A. 2021-2022

Indice

Capitolo 1

Appunti di Teoria

1.1 Introduzione

Un'ALU è un componente che prende due ingressi e calcola una uscita, oltre a prendere dei segnali di controllo che di volta in volta dicono che operazione svolgere sugli operandi.

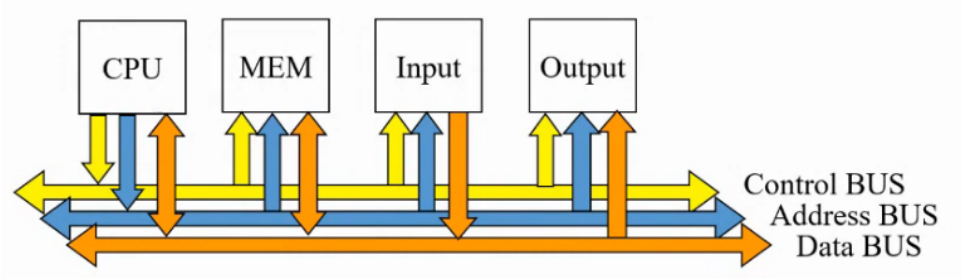
1.2 Computer systems

1.2.1 Macchina di Von Neumann

Una Macchina di Von Neumann è una macchina che legge una alla volta da una memoria delle istruzioni provenienti da un'instruction set finito e le esegue. Le due fasi sono dette:

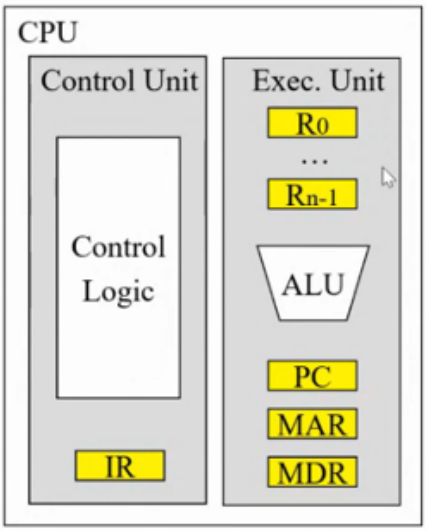
- Instruction Fetch
- Instruction Execute

Entrambe le fasi sono controllate dalla Control Unit. Per collegare fra loro questi componenti il sistema prevede un canale di comunicazione condiviso detto BUS.



Il canale è da immaginare come un fascio di interconnessioni metalliche. L'utilità di avere 3 BUS è data dal fatto che sul Data BUS possono passare i dati ma evitando i conflitti in quanto la CPU tramite l'Address BUS può comandare un componente alla volta e tramite il Control BUS gli dirà se dovrà scrivere o leggere.

1.2.1.1 Micro-Architettura CPU



Il Program Counter (PC) è il puntatore alla locazione di memoria da cui leggere la prossima istruzione.
L'Instruction Register (IR) è quello che contiene l'ultima istruzione letta, dunque la prossima istruzione da eseguire. Viene rappresentata nella parte sinistra in quanto la decodifica di questa istruzione prima di essere eseguita viene fatta dalla Control Unit.
Memory Address Register e Memory Data Register (MAR)(MDR) sono i registri con cui il processore si affaccia sul BUS degli indirizzi e sul BUS dei dati.

1.2.1.2 Loop fase di Fetch:

1. **MAR** ← **PC** Scrive su MAR l'indirizzo dell'istruzione da leggere;
2. **MDR** ← **MEM[MAR]** Leggo nel MDR il contenuto del BUS in cui la memoria ha reso disponibile l'istruzione;
3. **IR** ← **MDR** Porto nell'IR il contenuto dell'MDR
4. **Decode** Decodifico l'istruzione
5. **PC** ← **PC** + **1** Aggiorno il PC
6. **Execute** Eseguo l'istruzione
7. **Go back to 1** Ripeto

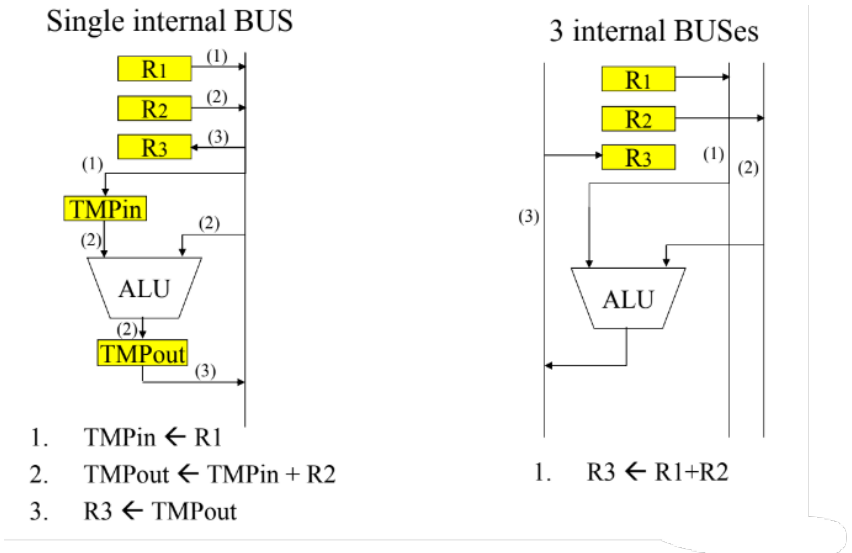
In un'architettura con un'interfaccia dedicata alla memoria le fasi 1,2,3,4 si riassumerebbero in:

1. **IR** ← **MEM[PC]**
2. **PC** ← **PC** + **1** Aggiorno il PC
3. **Execute** Eseguo l'istruzione
4. **Go back to 1** Ripeto

1.2.1.3 Fase di Esecuzione:

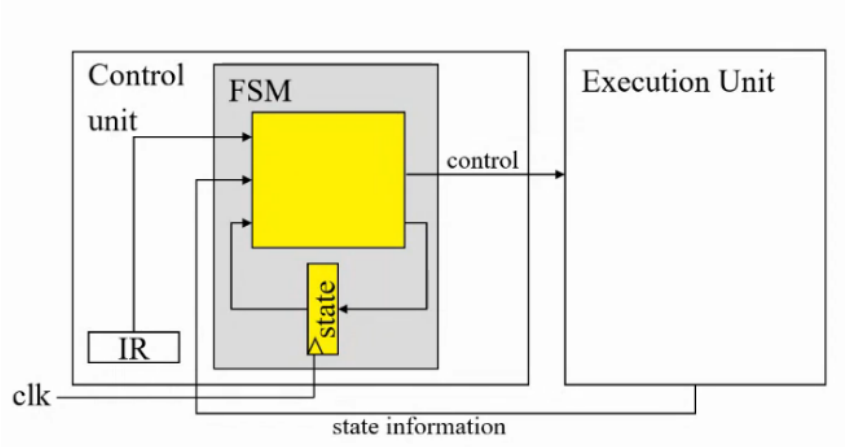
1. **Operand Fetch:** porta gli operandi alla ALU;
2. **Execute:** Esegue l'operazione richiesta;
3. **Write Back:** Riporta il risultato nei registri di destinazione.

In base al numero di BUS la fase di Instruction Fetch può avvenire in modi diversi:

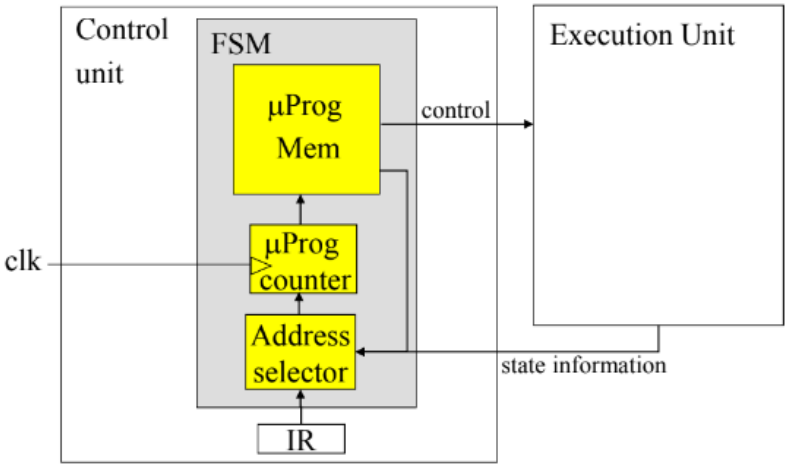


1.2.1.4 Control Unit

Si tratta di una macchina a stati finiti che ha come input l'IR, e lo stato in cui la macchina si trovava e come output i segnali di controllo per l'unità di esecuzione.



1.2.1.5 Control Unit Microprogrammata



1.2.2 Criteri Classificazione Microprocessori

I criteri di classificazione dei Microprocessori in base al repertorio di Istruzioni sono i seguenti:

- Istruction Set
  - Grandezza: repertorio di istruzioni più o meno ristretto.
  - Formato delle Istruzioni: le istruzioni possono essere più o meno lunghe.

  Esempi:

  - **Instruction Set Computer Ridotte** RISC:
    - Tutte della stessa dimensione
    - Pochi formati diversi
    - OPCODE con dimensione e posizione fissa
  - **Instruction Set Computer Complesse** CISC:
    - Istruzioni con dimensioni differenti
    - Formati differenti
    - OPCODE con dimensione e posizione variabile

- Modello di Esecuzione
  - MEM-MEM
  - MEM-REG
  - REG-REG
  - Stack
- Modalità di Indirizzamento
  - Oridnamento
  - Allineamento
  - Computazione dell'indirizzo
    * Diretto
      LD R1,var
    * Indiretto
      LD, R1,(R2)
    * Relativo ai Registri
      LD R1,var(R2) dove var è un'offset rispetto a R2
    * Relativo ai Registri, indicizzato e scalato
      LD R1,var(R2)(RX) RX è un'metodo di indicizzazione

1.3 CPU

1.3.1 Fasi del'esecuzione:

1. IF (Instruction Fetch): Carica un'istruzione dalla memoria;
2. ID (Instruction Decode): Decodifica l'istruzione;
3. EX (Execute): Esegui i calcoli;
4. MA (Memory Access): Leggi o scrivi i dati in memoria;
5. WB (Write Back): Riscrivi i risultati nei registri.

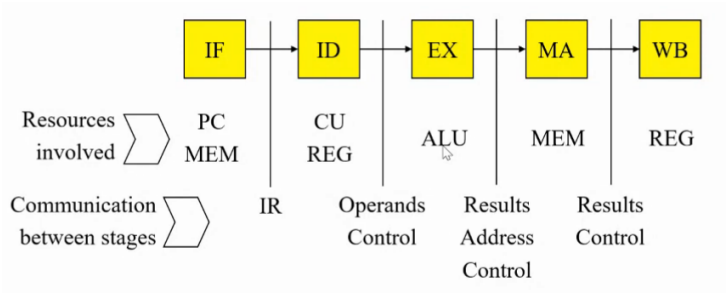


Figura 1.1: Risorse utilizzate e comunicazione tra stadi

1.3.2 Metriche di Performance

- CPUT [sec/program]
- CPUC [clocks/program]
- Tclk [sec/clock]
- CPI [clocks/instruction]
- IC [instruction/program]
- MIPS [minstruction/sec] (milioni di istruzioni al secondo)
- RMIPS = CPUT(ref)/CPUT*MIPS(ref)
- Latency [Clocks]
- Throughput [instruction/sec]

1.3.3 Data Hazards

1.3.3.1 Problema Strutturale Architettura Von Neumann

Con l'architettura Von Neumann persiste un grave problema di dipendenza dagli stessi registri, in quanto, come riportato nell'esempio sotto, utilizzando un solo registro saranno causati stalli in quanto più istruzioni richiedono lo stesso registro.

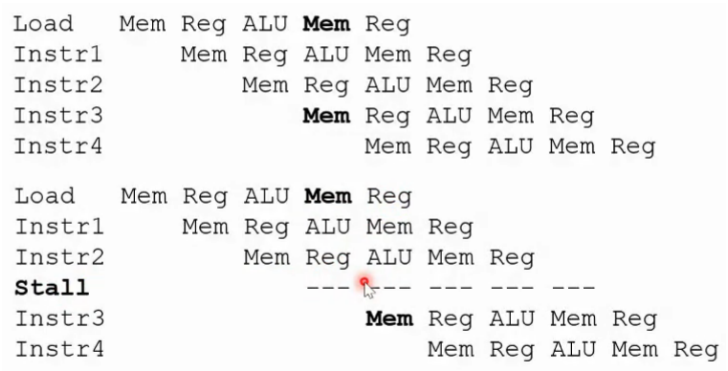


Figura 1.2: Architettura Von Neumann a un solo registro

A questo problema dell'architettura Von Neumann è stata proposta una soluzione:

Architettura Harvard: Utilizzo di memorie separate per dati e istruzioni.

1.3.3.2 Dipendenze di Dati

Se immaginiamo che più istruzioni vengano eseguite, una dopo l'altra, è facile pensare che prima o poi in due istruzioni avranno bisogno dello stesso registro nello stesso ciclo di clock, causando una serie di stalli. Quì sono riportate le categorizzazioni degli stalli:

- **RAW - Read After Write**:
  La dipendenza RAW si riferisce al caso in cui un'istruzione richieda un dato non ancora calcolato. Per esempio:

  $$i1.\ R2 \leftarrow R1 + R3$$
  $$i2.\ R4 \leftarrow R2 + R3$$

  La prima istruzione calcola la somma di R1 ed R3 e pone il risultato in R2. la seconda istruzione invece calcola la somma di R2 e R3 e pone il sisultato in R4, ma ovviamente non può eseguire la somma fino a quando la prima istruzione non è completa. Quindi si ha una dipendenza dai dati. Un'eventuale esecuzione fuori ordine non è possibile.

- **WAR - Write After Read**:
  La dipendenza WAR si verifica nel momento in cui un'istruzione legge un dato che si trova in una locazione in cui un'istruzione successiva sta per salvare un altro dato. Per esempio:

  $$i1.\ R1 \leftarrow R2 + R3$$
  $$i2.\ R3 \leftarrow R4 \cdot R5$$

  La prima istruzione somma R2 a R3 e pone il risultato in R1, mentre la seconda istruzione moltiplica R4 con R5 e pone il risultato in R3. Per ottenere l'esecuzione corretta del programma bisogna garantire che la prima istruzione legga il valore da R3 prima che la seconda istruzione aggiorni il valore in R3.

- **WAW - Write After Write**:
  La dipendenza WAW si riferisce al caso in cui più istruzioni utilizzino simultaneamente gli stessi registri. Per esempio:

  $$i1.\ R2 \leftarrow R1 + R3$$
  $$i2.\ R2 \leftarrow R4 \cdot R7$$

Entrambe le istruzioni vogliono salvare il loro risultato nel registro R2 e quindi bisogna garantire che la prima istruzioni salvi il risultato prima della seconda. Nota: il risultato della prima istruzione viene cancellato dalla seconda istruzione. Se le due istruzioni sono consecutive, la prima istruzione è totalmente inutile. Anche se potrebbe nascondere un errore di programmazione, l'operazione non causa errori.

- **Structural Stalls**:
  Lo stallo strutturale è il tentativo di usare la stessa risorsa hardware da parte di diverse istruzioni in modi diversi nello stesso ciclo di clock

- **Branch Taken Stalls**:
  Quando si esegue un branch (vai a un'istruzione diversa dalla successiva), l'istruzione sull'indirizzo di destinazione non è ancora stata decodificata. Quindi deve essere prima decodificato, e quindi l'esecuzione di questa istruzione richiede due clock.

Per alleggerire i problemi causati da queste dipendenze sono state adottate due soluzioni:

- Data Forwarding:
  al posto di rendere disponibili i dati solo a seguito della fase di WB, li rende disponibili già a seguito dell'EX, in quanto collega l'uscita della ALU al MUX che si trova ai suoi input.

- Double Rate Registers:
  al posto di rendere disponibili i dati solo a seguito della fase di WB, li rende disponibili già a seguito del MA, in quanto rende possibile lettura e scrittura allo stesso momento.

1.3.3.3 Controllo di Flusso

Un'altro problema che potremmo affrontare è quello del controllo del flusso in quanto se incontrassimo un'istruzione di Branch avremmo il problema che:

- Il branch è riconosciuto in fase di ID.
- La condizione è testata durante l'EX.
- Il PC è aggiornato durante il MA.

Aggiungendo risorse possiamo trovare diverse soluzioni:

- Prima Soluzione:
  - Il branch è riconosciuto in fase di ID.
  - La condizione è testata durante l'ID.
  - Il PC è aggiornato durante l'EX.
- Seconda Soluzione:
  - Il branch è riconosciuto in fase di ID.
  - La condizione è testata durante l'ID.
  - Il PC è aggiornato durante l'ID.
- Predict Untaken:
  Se la condizione risulta falsa non dobbiamo fare l'abort della condizione successiva di cui abbiamo fatto il fetch.
- Delayed Branch:
  Prendendo atto del fatto che quando deve essere effettuato un salto me ne accorgo in ritardo al posto di mettere a seguito dell'istruzione successiva al salto, un'istruzione che realmente sarebbe dovuta essere eseguita dopo, ne metto una che in realtà era precedente al salto, e che quindi realmente doveva essere eseguita(ritardandone l'esecuzione).

1.3.3.4 Esecuzioni a cicli multipli

Come ultima considerazione bisogna ragionare sul fatto che l'esecuzione di un'istruzione possa richiedere più di un Ciclo di Clock, in particolare succede con istruzioni che richiedono un calcolo, come addizioni e moltiplicazioni.
Se le istruzioni sono eseguite con pipeline serve solo preoccuparsi che più istruzioni non si ritrovino nella stessa fase dell'EX, in caso contrario per evitare stalli bisogna fare in modo di avere le unità di elaborazione necessarie.

1.4 Ottimizzazioni delle Performance

Nei Microprocessori Superscalari l'unico modo per far scendere il CPI ad un numero inferiore ad 1 è di dotare il processore di più Pipeline che agiscono in parallelo, in particolare se n è il numero di pipeline, il CPI ideale sarà $\frac{1}{n}$.

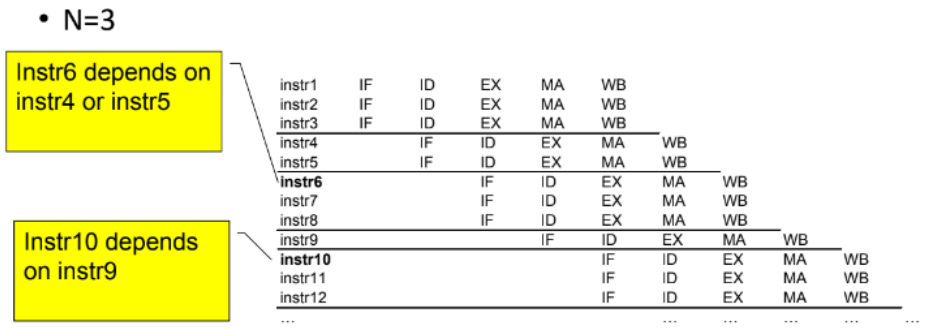


Figura 1.3: Esecuzione istruzioni in processore Superscalare

Repetition time è il tempo necessario per mandare in esecuzione un'istruzione dello stesso tipo.
Nei processori superscalari le diverse Pipeline potrebbero essere assegnate a diversi tipi distruzioni (i.e. una pipeline per gli interi ed una per i float). Nel momento in cui c'è uno stallo, tutte le pipeline vanno in stallo, in più diverse pipeline potrebbero avere diversa latenza, ma l'importante è che abbiano stesso repetition time.
Tramite la seguente formula:

$$PIC = Numero\ di\ Istruzioni\ Parallele$$
$$CPUT = (PIC + SC)Tclk > \frac{IC}{N \cdot Tclk}$$

possiamo effettuare una valutazione delle performance del processore.

1.4.1 Processori VLIW

L'acronimo VLIW sta per Very Long Instruction Word, e come da nome, ha la particolarità di avere istruzioni molto lunghe in quanto sono la concatenazione di più istruzioni che poi vengono spezzate e lette.
A differenza di altri modelli lo scheduling e l'issuing sono statici, in quanto non solo l'ordine di esecuzione delle istruzioni è già deciso ma lo è anche il raggruppamento tra esse.
Il chiaro vantaggio di questa architettura è che le decisioni di Issuing sono già state prese quindi il processore avrà meno lavoro da compiere.
Assumendo che ogni istruizione abbia CPI = 1, la stima delle performance sarà pari a:

$$VLIWC = Numero\ di\ Istruzioni\ VLIW$$
$$CPUT = (VLIWC + SC)Tclk > \frac{IC}{N \cdot Tclk}$$

1.4.2 Ottimizzazione Dinamica

Affronteremo 3 tipi di ottimizzazioni dinamiche:

- Esecuzione Fuori Ordine (OOO):
  Sheduling Dinamico (Algoritmo di Tomasulo);
- Speculazione:
  Decisioni a tentativi su branch condizionali con condizioni sconosciute;
- Operazioni sotto il ciclo di clock:
  Più di una istruzione processata ad ogni ciclo di clock.

1.4.2.1 Esecuzione Fuori Ordine

La OOO fornisce le seguenti opportunità:

- Dare priorità alle istruzioni pronte ad essere eseguite;
- Evitare false dipendenze di dati;
- Inviare i risultati parziali a tutte le unità di esecuzione;
- Mantenere la coerenza sequenziale alla scrittura nel register file.

Per supportare questo tipo di esecuzione vi è il bisogno di implementare del supporto hardware:

- ID e Register Fetch devono avvenire in diverse fasi;
- Coda delle istruzioni;
- Reservation Station;
- Common Data Bus (CDB);
- Reorder Buffer (ROB);

I componenti elencati forniscono l'implementazione hardware dell'algoritmo di Tomasulo.
Fasi Esecuzione Fuori Ordine:

1. **Issue**: Prende un'istruzione dalla instruction queue e la inserisce nella reservation station;
2. **Execute**: Quando la risorsa è disponibile e gli operandi sono disponibili nella reservation station, esegue l'istruzione;
3. **Write results**: Invia il risultato tramite il Common Data Bus al Re-order buffer e a tutte le reservation station che la stanno aspettando;
4. **Commit**: Sposta i risultati dalla testa del Re-order buffer al file di registro o alla memoria.

ROB Il Re-order Buffer ROB é una coda circolare di tipo FIFO, tiene traccia dell'ordine reale dell'esecuzione delle istruzioni e, a mano a mano che le istruzioni sono eseguite dalle unità funzionali, preleva i dati elaborati e li memorizza nei registri del processore seguendo l'ordine logico del programma. Dopo aver memorizzato nei registri il risultato dell'istruzione il sistema effettua il commit dell'istruzione cancellandola dal buffer ROB.

RS Le reservation station RS facilitano l'esecuzione di istruzioni in parallelo tenendo traccia di quali istruzioni sono in attesa di un risultato da un'altra istruzione e quali invece sono pronte ad eseguire. Un bus detto "Common Data Bus" collega le stazioni e permette ai risultati di giungere alle istruzioni in attesa.

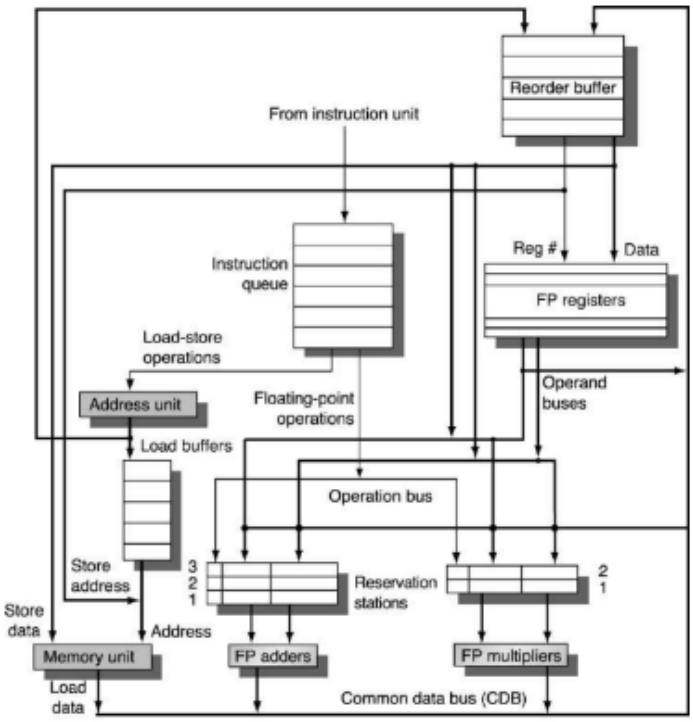


Figura 1.4: Architettura con supporto per l'Esecuzione Fuori Ordine

1.4.2.2 Speculazione

Immaginando che il codice abbia un salto condizionato e siamo in un architettura che supporta l'esecuzione fuori ordine (considerando che questo tipo di esecuzione serve a dare precedenza alle istruzioni che possono essere eseguite).Se immaginiamo che il salto condizionato sia in attesa del valore della condizione nasce un problema: se ci sono istruzioni che possono già essere eseguite e il salto non ha ancora deciso se effettivamente le istruzioni debbano essere eseguite o meno, potrebbero essere eseguite istruzioni che in realtà non dovrebbero essere eseguite.
Allora senza effettivamete calcolare se il salto debba essere eseguito o meno possiamo provare a "indovinare" se la condizione sarà vera o falsa.
Questo è possibile nell'esecuzione fuori ordine grazie all'esistenza del ROB, perchè nel momento in cui proviamo ad indovinare l'esito del Branch, le soluzioni delle eventuali istruzioni eseguite vengono inserite nel ROB, in caso avessimo indovinato l'esito del Branch avverrebbe il Commit, in caso contrario verrebbe semplicemente svuotato il ROB.
Per la predizione esistono 2 diverse strategie:

- **Branch Prediction**: la predizione della condizione di salto;
- **Target Prediciton**: la predizione del target.

Branch Prediction ha la possibilità di avere predittori di diversi bit:

- 1-bit Si ricorda se l'ultimo salto incontrato è stato fatto o meno, e applica la stessa scelta al nuovo salto;
- n-bit Più bit si dedicano alla memoria, più accurata sarà la predizione.

Considerando il grande numero di predizioni da effettuare è logico pensare che serva più di un predittore, quindi avere un predittore per ogni branch. Avere un Branch Prediction Buffer (BPB) formato da un solo bit per branch è già abbastanza in quanto la divisione dei branch assicura già una precisione migliore. Utilizzando però il BPB è necessario avere un metodo di indicizzazione nel buffer, per questo si sfrutta estraendo l'Index dall'indirizzo puntato dal PC.

Branch Target Buffer Il BTB fornisce direttamente il target del PC, funziona come una cache completamente associativa. Se un salto deve essere effettuato, viene inserito nella struttura dato, in caso contrario viene cancellato nella struttura o non viene scritto.

1.5 Memoria

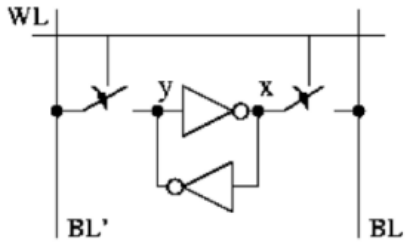
1.5.1 Classificazione

I dispositivi di memoria possono essere classificati in base a diversi criteri:

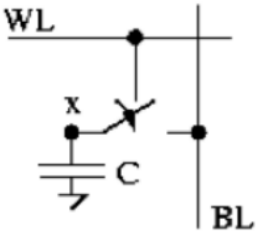
- **Volatilità**:
  - Volatile: perde i dati quando perde l'alimentazione.
  - Non Volatile: mantiene i dati anche senza alimentazione.
- **Bilanciamento lettura-scrittura**:
  - ROM (Read Only Memories).
  - PROM (Programmable ROM).
  - EPROM (Erasable PROM).
  - EEPROM (Electrically-Erasabe PROM).
  - FLASH: Memoria in lettura non volatile.
  - RAM (Random Access Memory): Memoria in lettura e scrittura di tipo volatile.

1.5.1.1 Memoria Volatile

SRAM RAM statica, formata da un bistabile utilizzato per memorizzare un bit.



DRAM RAM Dinamica, formata da un semplice condensatore, che sarebbe un componente passivo con 2 terminali che produce una differenza di potenziale.



SRAM vs DRAM Comparando i due tipi di memoria, si possono fare diverse considerazioni:

- **Performance**:
  i dispostivi SRAM sono più veloci in quanto le connessioni statiche tra Vdd e Ground rendono la lettura molto più veloce di quella della DRAM.
- **Densità**:
  La DRAM ha densità minore della SRAM in quanto è composta solamente da un transistor e un condensatore mentre la SRAM è composta da 6 transistor. In ogni caso l'utilizzo di un condesatore trenched o stacked porta l'area della DRAM ad essere molto vicina a quella della SRAM.
- **Costo**:
  La DRAM è molto meno costosa in termini di costo per bit, a causa della densità più alta.
- **Capacità**:
  La DRAM può contenere più bit a causa della densità maggiore.

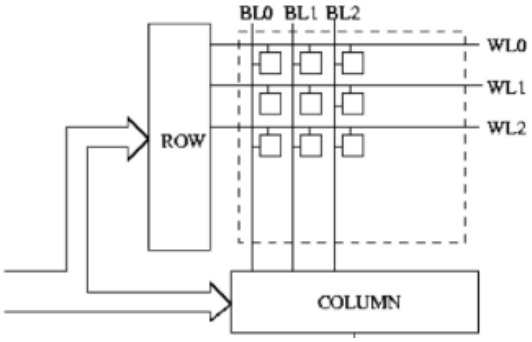


Figura 1.5: Organizzazione Memoria

Organizzazione Memoria Le celle di memoria sono posizionate in una matrice quadrata. Gli indirizzi sono divisi in indirizzo colonna e riga, questi ultimi sono processati da un decodificatore di riga, mentre gli indirizzi delle colonne sono selezionati tramite la Bit Line.
Ciclo di lettura memoria:

- Decodifica indirizzo riga;
- Asserzione di una Wordline;
- Attivazione di tutte le celle della riga scelta;
- Pilotaggio delle Bitline;
- Decodifica dell'indirizzo colonna;
- Selezione della colonna;
- Pilotaggio dell'output nel Data Bus.

Con questo sistema possono essere utilizzati molteplici banchi di memoria per creare sistemi con capacità aumentata. Se consideriamo che un dispositivo di memoria contenga N parole di n bit, 2 dispositivi di memoria possono essere utilizzati per realizzare una memoria da $2N$ parole da n bit.

1.5.1.2 Memoria Non Volatile

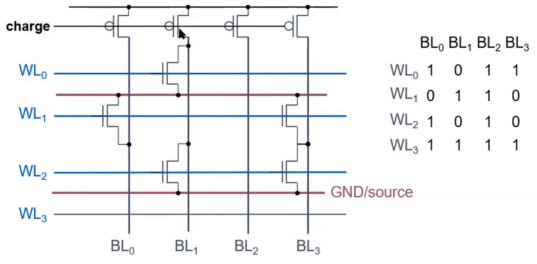


Figura 1.6: ROM basata su NOR

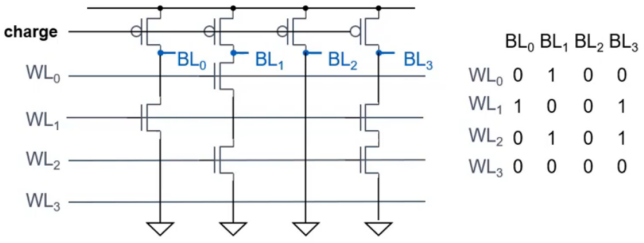


Figura 1.7: ROM basata NAND

Floating Gate Transistor:
Si tratta di un convenzionale transistore MOS con l'aggiunta di un ulteriore terminale di gate, detto Floating Gate, situato tra il substrato ed il Control Gate e separato da essi dal biossido di silicio. Questa struttura è un condensatore, ed è capace di contenere carica elettrica per periodi molto lunghi. Alcune tra le numerose applicazioni del FGMOS sono le memorie EPROM, EEPROM e flash.

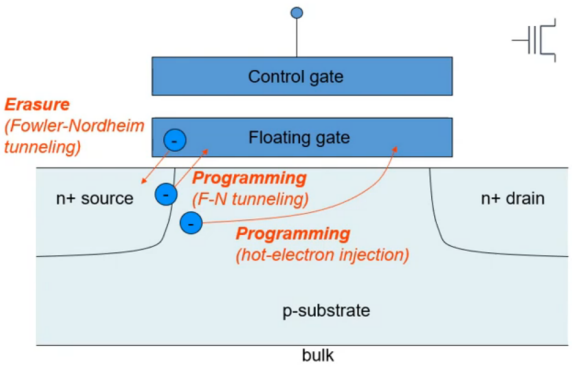


Figura 1.8: Floating Gate Transistor

Memorie Flash vs HDD

- I meccanismi di lettura e scrittura funzionano in modo diverso nella flash, quindi vi è un numero limitato di clicli cancella-scrivi;
- La memoria flash consuma meno energia;
- Costo superiore per bit rispetto agli Hard Disk.

1.5.2 Gerarchie di Memoria

Il principio di funzionamento della gerarchia di Memoria parte dall'esigenza di soddisfare contemporaneamente i requisiti di dimensione, velocità e costo dei dispositivi di memoria, e dal fatto che le due soluzioni tecnologiche di cui disponevamo non le soddisfano tutte contemporaneamente. Il dispositivo ideale che vorremmo realizzare dovrebbe essere veloce quanto la SRAM ma grande ed economico quanto la DRAM.

Località spaziale e temporale gli indirizzi di memoria che il processore genera nel tempo non sono distribuiti in modo casuale, ma sono circoscritti nel tempo e nello spazio, quindi se un processore ha richiesto un determinato indirizzo di memoria in un determinato istante, con buona probabilità chiederà gli indirizzi di memoria che gli stanno vicini.

1.5.2.1 Cache Fisica e Virtuale

Una cache è una memoria piccola e veloce che duplica parte della memoria principale. Gli accessi alla cache sono molto più veloci rispetto a quelli alla memoria principale, finchè il processore trova in cache i dati di cui ha bisogno, la performance registrata è quella della cache. Per decidere cosa spostare in cache viene sfruttato il principio di località, dunque copia n celle di memoria adiacenti a quella selezionata.
La cache può essere categorizzata in 3 tipi diversi, in base al modo in cui viene indirizzata:

- Mappatura Diretta:

Block Address					
Tag		Index		Offset	
0	1	0	0	1	1

- Completamente Associativa:

Block Address					
Tag				Offset	
0	1	0	0	1	1

- Parzialmente Associativa (Set-associativa):

Block Address					
Tag			Index	Offset	
0	1	0	0	1	1

Se la cache è piena, bisogna utilizzare un criterio per decidere in che ordine rimpiazzare la memoria in cache:

- **LRU** Least Recently Used;
- **RND** Random;
- **FIFO** First In First Out.

Esisistono più livelli di cache:

1. **Cache Livello 1**(L1): è la cache più alta nella gerarchia dei livelli di cache di una CPU nonchè la cache più veloce nella gerarchia. Ha una dimensione minore e un ritardo di lettura minore (stato di attesa zero) perché di solito è integrato nel processore. Come memoria viene utilizzata la SRAM.
2. **Cache Livello 2**(L2): è la cache che si trova accanto a L1 nella gerarchia della cache. L2 è tipicamente implementato utilizzando una DRAM. La maggior parte delle volte, L2 è saldato sulla scheda madre molto vicino al chip (ma non sul chip stesso).

Il funzionamento della Cache in un sistema è molto semplice, il processore richiede dei dati, prima li cerca nella L1, in caso non sia presente effettua un cache miss, il dato viene quindi ricercato nella L2, se non presente avviene un nuovo cache miss e viene finalmente cercato nella memoria principale.

1.5.2.2 Memoria Virtuale

La memoria virtuale è un architettura capace di simulare uno spazio di memoria principale piu grande di quelo effetivo usando una memoria secondaria per aumentare lo spazio di indirizamento disponibile, per fare ciò la CPU deve produrre degli indirizzi virtuali in modo da accedere allo spazio di indirizzamento ampliato che devono essere tradotti in indirizzi fisici per poter accedere alle locazioni fisiche della memoria fisica.

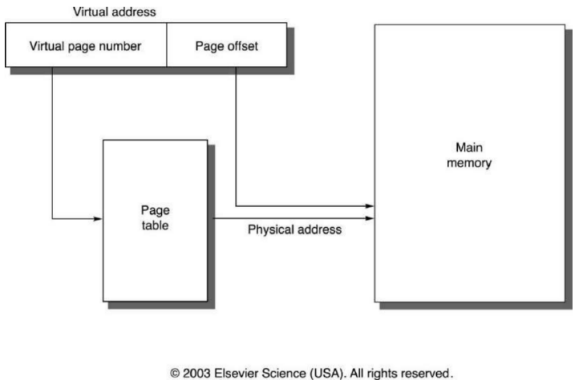


Figura 1.9: Conversione Indirizzo

MMU La Memory Management Unit MMU è una classe di componenti con il compito di eseguire la traduzione degli indirizzi virtuali in indirizzi fisici, necessaria per la gestione della memoria virtuale.

TLB Il Translation Lookaside Buffer TLB è una memoria cache che l'MMU usa per velocizzare la traduzione degli indirizzi virtuali. Il TLB possiede un numero fisso di elementi della tabella delle pagine, la quale viene usata per mappare gli indirizzi virtuali in indirizzi fisici.

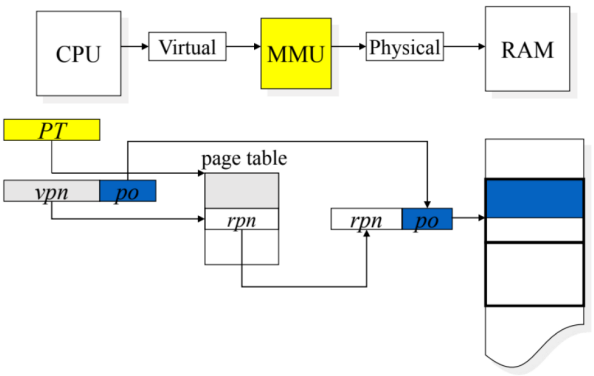


Figura 1.10: Memoria Virtuale

1.5.2.3 Cache Fisica e Virtuale

Nella **cache fisica** si traduce l'indirizzo virtuale in indirizzo fisico tramite l'MMU, in caso di cache miss si effettua la ricerca all'interno della RAM.

Figura 1.11: Cache Fisica

Si parla di **cache virtuale** quando alla cache ci si interfaccia prima della MMU e si utilizzano direttamente i VPN, quindi l'indirizzo virtuale sta nella cache in caso di cache miss, si ricerca in RAM.

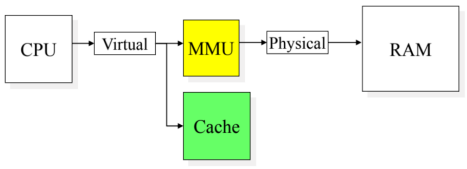


Figura 1.12: Cache Virtuale

1.6 Comunicazione

1.6.1 Periferiche

Interfaccia:

- **DREG** Registro che acquisice i dati dal BUS;
- **SREG** Registro che scrive i dati sul BUS
- **CREG** Registro che acquisice i bit di controllo del BUS;
- **Address Comparator** Controlla che l'indirizzo sia della periferica;
- **Address Decoder** Decoder interno che seleziona i registri chiamati in causa.

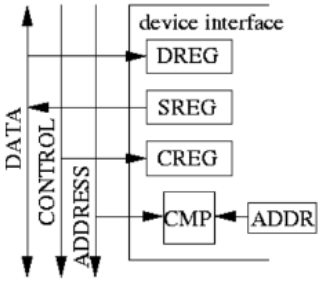


Figura 1.13: Architettura della periferica

La comunicazione tra periferiche e dispositivo avviene tramite l'utilizzo del BUS, interfacce specifiche per il dispositivo fanno utilizzo di driver che sfruttano istruzioni di lettura e scrittura di basso livello per fornire funzionalità di alto livello.
Ci sono due modalità di gestione delle periferiche dal punto di vista di lettura e scrittura:

- Memory-Mapped:
  Sono periferiche mappate in memoria, dunque tutto lo spazio di indirizzamento è in parte assegnato alle periferiche.

- I/O-Mapped:
  Istruzioni diverse per la gestione delle periferiche, quindi lo stesso registro può essere usato sia per la memoria che per le periferiche.

1.6.2 Problemi di Sincronizzazione

Accedere ad una periferica può richiedere migliaia di cicli di clock, il tempo di servizio ne può richiedere milioni mentre il tempo di arrivo di input esterno è imprevedibile. Le Pipeline sono progettate considerando che gli accessi alla memoria siano effettuati in pochi cicli di clock. A causa di tutto ciò il sistema mette da parte i processi di interfacciamento delle periferiche per preferire processi che siano pronti ad essere svolti.
A questo problema sono state trovate 3 diverse soluzioni:

- **Program-Controlled I/O**:
  Il processo consiste in un un polling[1] che serve a mantenere impegnati sia il processore che il BUS delle periferiche.

- **Interrupt-Controlled I/O**:
  è un componente che funge da interfaccia fra CPU e periferiche. Invia un segnale alla CPU nel caso in cui le periferiche mandino un segnale di interruzione[2] all'Interrupt controller. Se la CPU concede l'interruzione, e sono presenti più dispositivi che la richiedono, si avvia un meccanismo di priorità.

- **Direct Memory Access**(DMA):

[1] **Polling** è una tecnica che tratta di interrogare continuamente un bus di comunicazione delle periferiche, controllando lo stato del registro fin quando non sia pronto, fino a quando non è pronto per svolgere una determinata operazione.
[2] **Interrupt** è un segnale che comunica alla CPU il verificarsi di un evento per via dell'Interrupt Controller, nel caso delle periferiche, la richiesta di scambio dati con una quest'ultima.

con questo metodo il **DMA controller** assume il controllo del bus degli indirizzi. I dati da trasferire vengono memorizzati o letti sequenzialmente nella memoria dell'elaboratore a partire da una posizione prefissata. Il dispositivo DMA comunicherà alla CPU la fine del trasferimento per mezzo di un preciso segnale.

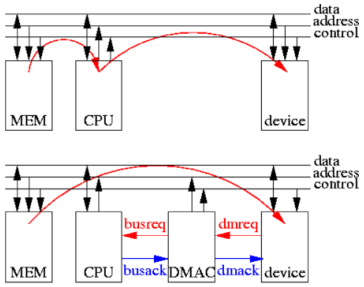


Figura 1.14: Rappresentazione DMA

ISR I risultati delle periferiche sono inseriti in un OR ed esiste bus di comunicazione, l'ISR **Interrupt Status Register** è un registro che campiona i segnali delle periferiche e gestisce la comunicazione con la CPU.

Capitolo 2

Esercizi

2.1 Espressione in Diverse Architetture

Espressione: $A + (B \cdot C)$

Mem - Mem	Mem - Reg	Mem - Reg (1 Reg)	Reg - Reg	Stack

MUL D,B C
ADD R,D A
	LOAD R1,C
	MUL R1,R1,B
	ADD R1,R1,A
	STORE R,R1
		LOAD C
		MUL B
		ADD A
		STORE R
			LOAD R1,C
			LOAD R2,B
			LOAD R2,A
			MUL R4,R1,R2
			ADD R5,R3,R4
			STORE R,R5
				PUSH B
				PUSH C
				MUL
				PUSH A
				ADD
				POP

2.2 Latenza Pipeline

Se il quesito richiede quando latenza e repetition time sono uguali:
$add = Numero\ di\ Addizionatori$

- Pipeline $\rightarrow add = Latenza$
- No Pipeline $\rightarrow \frac{Latenza}{add} = Latenza$

Esercizio 1:

Con riferimento ad un microprocessore pipelined, in quali delle seguenti soluzioni architetturali la somma floating point ha repetition time minore della latenza?

a. Un solo addizionatore FP pipelined con latenza 3.

b. 3 addizionatori FP non pipelined con latenza 3.

c. Un solo addizionatore FP non pipelined con latenza 4.

d. Un solo addizionatore FP non pipelined con latenza 1.

Rappresentazioni:

Soluzione 1:

a:

IF	ID	EX	EX	EX	MA	WB	
	IF	ID	EX	EX	EX	MA	WB

Repetition Time = 1, Latenza = 3. Corretto

b:

IF	ID	EX	EX	EX	MA	WB		
	IF	ID	EX	EX	EX	MA	WB	
		IF	ID	EX	EX	EX	MA	WB

Repetition Time = 1, Latenza = 3. Corretto

c:

IF	ID	EX	EX	EX	EX	MA	WB				
	IF	ID	ID	ID	ID	EX	EX	EX	EX	MA	WB

Repetition Time = 4, Latenza = 4. Sbagliato

d:

IF	ID	EX	MA	WB	
	IF	ID	EX	MA	WB

Repetition Time = 1, Latenza = 1. Sbagliato

2.3 Stalli

IF	ID	EX	MA	WB

Il Data Forwarding fornisce i dati a seguito dell'Execute. Il Double Rate Registers fornisce i dati a seguito del Memory Access.

Esercizio 2:

Con riferimento alla pipeline del DLX, quanti cicli di stallo comporta l'esecuzione dell'istruzione LOADD f0, 0(R1) seguita dall'istruzione ADDD f0, f0, f6 in presenza di data forwarding (assumendo che il primo parametro di ogni istruzione sia la destinazione e che la somma abbia latenza 3)?

Soluzione 2:

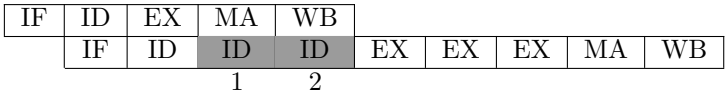
IF	ID	EX	MA	WB				
	IF	ID	ID	EX	EX	EX	MA	WB

1

Esercizio 3:

Con riferimento alla pipeline del DLX, quanti cicli di stallo comporta l'esecuzione dell'istruzione LOADD f0, 0(R1) seguita dall'istruzione ADDD f0, f0, f6 in assenza di data forwarding e double-rate registers (assumendo che il primo parametro di ogni istruzione sia la destinazione e che la somma abbia latenza 3)?

Soluzione 3:



2.4 Fase Esecuzione Fuori Ordine

Esercizio 4:
Con riferimento ad un microprocessore con supporto per esecuzione fuori ordine, dire in che fase le istruzioni vengono assegnate ad una entry del ROB.

Soluzione 4:
Soluzione: Write Results

2.5 CPI ideale

Esercizio 5:
Qual è il CPI ideale di un'architettura superscalare con 4 pipeline in parallelo a 5 stadi?

Soluzione 5:
CPI = $\frac{1}{n\_pipeline} = \frac{1}{4}$

Esercizio 6:
Qual è il CPI ideale di un'architettura VLIW con 3 pipeline in parallelo a 5 stadi?

Soluzione 6:
CPI = $\frac{1}{3}$

2.6 Disegni

Cliccare sulla traccia per essere riportati al disegno.

Esercizio 7:
Disegnare lo schema circuitale di un dispositivo di RAM Statica.

Soluzione 7:
Immagine.

Esercizio 8:
Disegnare lo schema circuitale di un dispositivo di RAM Dinamica.

Soluzione 8:
Immagine.

Esercizio 9:
Disegnare l'architettura interna di una memoria.

Soluzione 9:
Immagine.

Esercizio 10:
Disegnare lo schema circuitale di un dispositivo ROM basato su NAND.

Soluzione 10:
Immagine.

Esercizio 11:
Disegnare lo schema circuitale di un dispositivo ROM basato su NOR.

Soluzione 11:
Immagine.

Esercizio 12:
Rappresentare il collegamento al BUS di memoria e a quello degli indirizzi di due banchi di memoria con parallelismo doppio.

Soluzione 12:

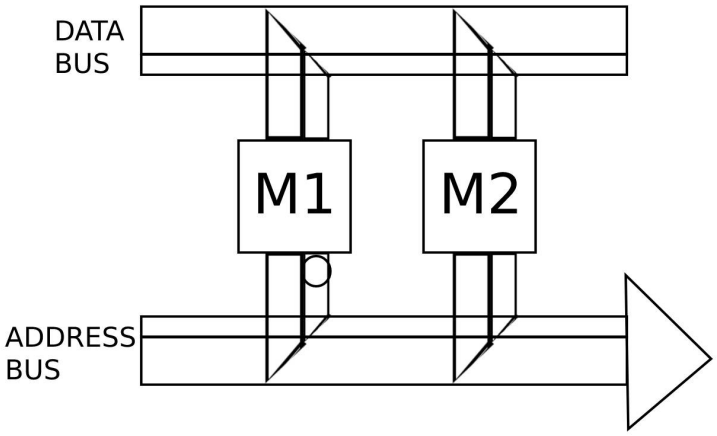


Figura 2.1: Archiettura esterna memorie parallele

2.7 Tempo di Accesso Medio Memoria

$$\begin{aligned} Miss\ Penalty &= Miss\ Time - Hit\ Time \\ T_{avg} &= Hit\ Rate \cdot Hit\ Time + Miss\ Rate \cdot Miss\ Time \\ &= Hit\ Time + Miss\ Rate \cdot Miss\ Penalty \end{aligned}$$

Esercizio 13:
Qual è il tempo medio di accesso ad un sistema di memoria dotato di un solo livello di cache sapendo che: il tempo di accesso alla cache è pari a 1 ciclo di clock, il miss time è pari a 10 cicli di clock e l'hit rate è del 90%?

Soluzione 13:
$T_{avg} = 0.9 \cdot 1 + 0.1 \cdot 10 = 1.9Tclk$

Esercizio 14:
Qual è il tempo medio di accesso ad un sistema di memoria dotato di un solo livello di cache sapendo che: il tempo di accesso alla cache è pari a 1 ciclo di clock, il miss penalty è pari a 99 cicli di clock e l'hit rate è del 98%?

Soluzione 14:

$T_{avg} = 1 + 0.02 \cdot 99 = 2.98Tclk$

Esercizio 15:
Qual è il tempo medio di accesso ad un sistema di memoria dotato di un solo livello di cache sapendo che: il tempo di accesso alla cache è pari a 2 cicli di clock, il tempo in caso di miss è pari a 50 cicli di clock e l'hit rate è del 99%?

Soluzione 15:
$T_{avg} = 0.99 \cdot 2 + 0.01 \cdot 50 = 2.48Tclk$

2.8 Determina Grandezza Tag

Esercizio 16:

In un sistema di memoria con indirizzi fisici a 32 bit, si consideri una cache a mappatura diretta con 1024 Blocchi complessivi di 16 Byte ciascuno. Determinare il numero di bit di TAG

Soluzione 16:

Offset = $\log_2(16byte) = 4bit$
Index = $\log_2(1024byte) = 10bit$
Tag = Bit Totali - Offset - Index = $32 - 4 - 10 = 18bit$

Se fosse stata completamente associativa:
Offset = $\log_2(16byte) = 4bit$
Tag = Bit Totali - Offset = $32 - 4 = 28bit$

Se fosse stata a parzialmente associativa a 2 vie con 1024 blocchi ciascuno, 16 byte ciascuno:
Offset = $\log_2(16byte) = 4bit$
Index = $\log_2(1024byte) = 10bit$
Tag = Bit Totali - Offset - Index = $32 - 4 - 10 = 18bit$
Se fosse stata parzialmente associativa a 4 vie con 1024 blocchi complessivi, 16 byte ciascuno:

Offset = $\log_2(16byte) = 4bit$
Index = $\log_2(\frac{1024}{4}byte) = 8bit$
Tag = Bit Totali - Offset - Index = $32 - 4 - 8 = 20bit$

Esercizio 17:

In un sistema di memoria con indirizzi fisici a 32 bit, si consideri una cache fisica setassociativa a 4 vie con 2048 blocchi complessivi di 256 byte ciascuno. Determinare il numero di bit di TAG

Soluzione 17:

Offset = $\log_2(256byte) = 8bit$
Index = $\log_2(\frac{2048}{4}bytebyte) = 9bit$
Tag = Bit Totali - Offset - Index = $32 - 8 - 9 = 15bit$

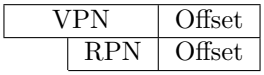
Esercizio 18:

In un sistema di memoria con indirizzi fisici a 32 bit, si consideri una cache fisica setassociativa a 8 vie con 1024 blocchi complessivi di 64 byte ciascuno. Determinare il numero di bit di TAG

Soluzione 18:

Offset = $\log_2(64byte) = 6bit$
Index = $\log_2(\frac{1024}{8}byte) = 7bit$
Tag = Bit Totali - Offset - Index = $32 - 6 - 7 = 19bit$

2.9 Determina Grandezza Page Table



Esercizio 19:

Si consideri un sistema di memoria paginato con indirizzi logici a 32 bit, indirizzi fisici a 28 bit e pagine di 64kbyte. Determinare le dimensioni della tabella delle pagine assumendo che non sia gerarchica e che i bit di controllo (Prot) associati ad ogni pagina siano 4.

Soluzione 19:

Offset $= \log_2(64kbyte) = 16bit$
VPN $= 32 - 16 = 16bit$
RPN $= 28 - 16 = 12bit$
Page Table $= 2^{16} \cdot (12 + 4)bit$
Se fossero stati indirizzi fisici a 29 bit e 3 bit di protezione:

Offset $= \log_2(64kbyte) = 16bit$
VPN $= 32 - 16 = 16bit$
RPN $= 29 - 16 = 13bit$
Page Table $= 2^{16} \cdot (13 + 3)bit$

Esercizio 20:

Si consideri un sistema di memoria paginato con indirizzi logici a 32 bit, indirizzi fisici a 28 bit e pagine di 8kbyte. Determinare le dimensioni della tabella delle pagine assumendo che non sia gerarchica e che i bit di controllo (Prot) associati ad ogni pagina siano 1.

Soluzione 20:

Offset $= \log_2(8kbyte) = 13bit$
VPN $= 32 - 13 = 19bit$
RPN $= 28 - 13 = 15bit$
Page Table $= 2^{19} \cdot (15 + 1)bit$

Esercizio 21:

Si consideri un sistema di memoria paginato con indirizzi logici a 32 bit, indirizzi fisici a 30 bit e pagine di 4kbyte. Determinare le dimensioni della tabella delle pagine assumendo che non sia gerarchica e che i bit di controllo (Prot) associati ad ogni pagina siano 6.

Soluzione 21:

Offset $= \log_2(4kbyte) = 12bit$
VPN $= 32 - 12 = 20bit$
RPN $= 30 - 12 = 18bit$
Page Table $= 2^{20} \cdot (18 + 6)bit$

2.10 Lettura Parole Cache

Esercizio 22:

In un sistema con cache L1 associativa di 1024 blocchi di 8 parole si assuma di dover leggere una volta, un array di 1.000 parole. Quale sarà il miss rate medio sperimentato con politica LRU?

Soluzione 22:

Dato che ogni 8 parole ci sarà un Cache Miss, e in 1024 blocchi in tutto ci vanno ben più di 1000 parole, il Miss Rate sarà:

$$MissRate = \frac{\frac{1000}{8}}{1000} = \frac{1}{8}$$

Se invece leggessimo l'array due volte:
Dato che alla seconda lettura nella cache ci saranno già caricate tutte le parole, il Miss Rate sarebbe 0:

$$MissRate = \frac{\frac{1}{8} + 0}{2} = \frac{1}{16}$$

Se lo leggessimo 2 volte e fossero 10000 parole:
Stavolta dato che tutte e 10000 le parole non entrano dentro 1024 blocchi, nella seconda lettura non si trovano le parole dunque:

$$MissRate = \frac{\frac{1}{8} + \frac{1}{8}}{2} = \frac{1}{8}$$

Mentre la politica di rimpiazzamento LRU, sia quella FIFO rimpiazzano quello che sta per servirci, la politica Random non lo fa, quindi in questi casi è quella più efficiente.

Esercizio 23:

In un sistema con cache L1 associativa di 1024 blocchi di 8 parole si assuma di dover leggere due volte, nello stesso verso, un array di 5000 parole. Quale sarà il miss rate medio sperimentato con politica RND?

Soluzione 23:

Grazie ai benefici della politica RND possiamo dedurre che sia semplicmente:

$$Missrate = \frac{1}{8}$$

2.11 Domande di Teoria

Tralasciando le definizioni questi sono esempi di possibili domande: **Esercizio 24:**

Cosa succederebbe se un'architettura non facesse utilizzo del DMA controller?

Soluzione 24:

Senza un controller DMA le periferiche non possono accedere alla memoria principale. Ciò significa che la CPU dovrebbe assumersi la responsabilità di tutti i trasferimenti di dati sul BUS di memoria, che è essenzialmente ciò che fa il controller DMA. Funziona come un semaforo per vari dispositivi che vogliono accedere alla memoria principale tramite il BUS, oltre a specificare cosa viene copiato e dove viene copiato.

Esercizio 25:

Spiegare il ruolo del Floating Gate Transistor e dove viene utilizzato.

Soluzione 25:

Si tratta di un convenzionale transistore MOS con l'aggiunta di un ulteriore terminale di gate, detto Floating Gate, situato tra il substrato ed il Control Gate e separato da essi dal biossido di silicio. Questa struttura 'e un condensatore, ed 'e capace di contenere carica elettrica per periodi molto lunghi. Alcune tra le numerose applicazioni del FGMOS sono le memorie EPROM, EEPROM e flash.

Queste dispense sono state preparate con LaTeX, esse costituiscono soltanto un ausilio per lo studente , quindi non sono in nessun modo sostitutive dei testi consigliati. Il PDF è dotato di hyperlinking quindi è possibile cliccare sui nomi delle sezioni nell'indice per spostarsi tra le pagine, allo stesso modo, per tornare all'indice è sufficiente cliccare sul numero della pagina.