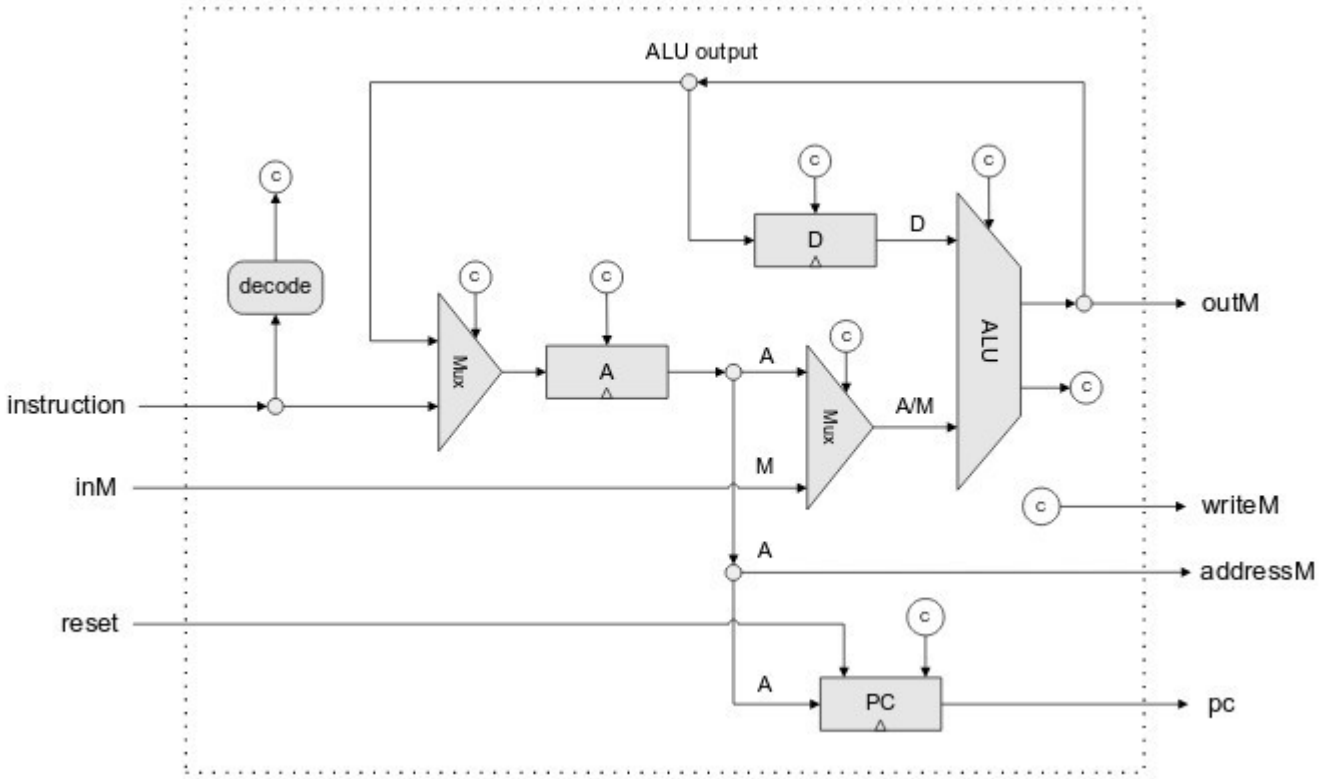
Microarchitettura



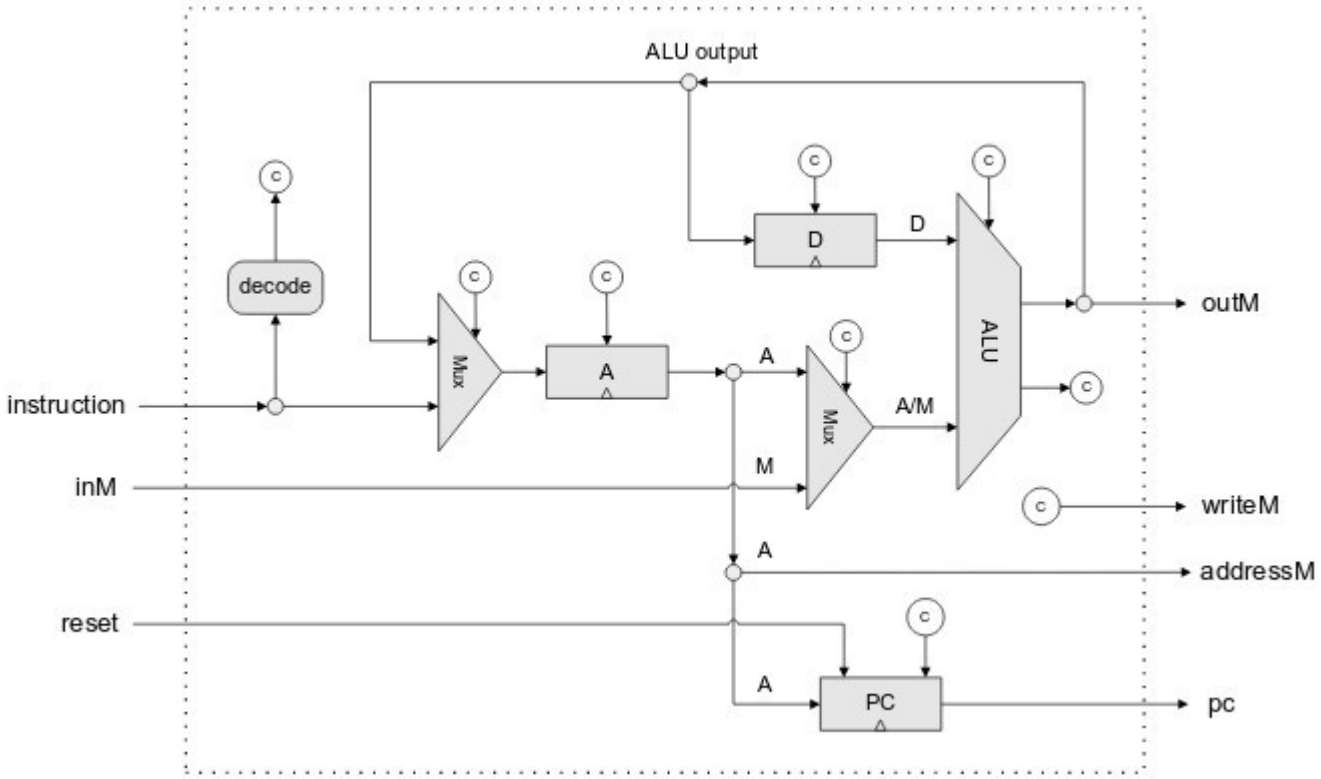
Prof. Ivan Lanese

Microarchitettura

- Il livello di microarchitettura si occupa di utilizzare i componenti del livello logico digitale per realizzare il linguaggio macchina (ISA: Instruction Set Architecture)
- Esistono microarchitetture estremamente sofisticate
  - Iniziamo da una molto semplice (quella del nostro processore Hack)
  - Poi studieremo aspetti generali di microarchitetture più complicate
    - Cache
    - Pre-fetch istruzioni
    - Pipeline

Microarchitettura del processore Hack

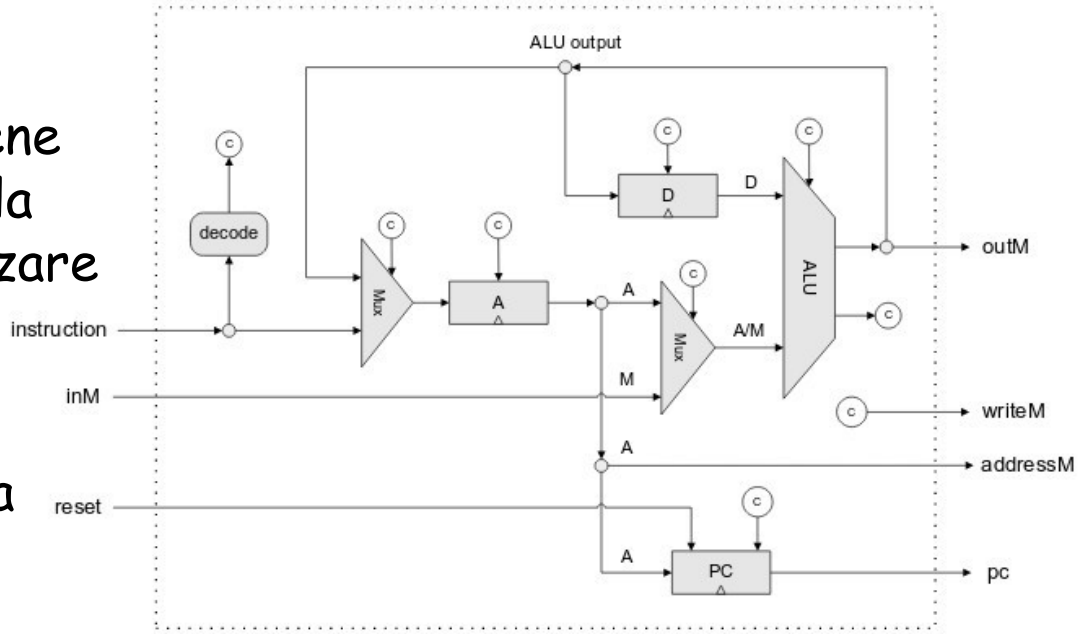
- Il processore Hack è progettato secondo il seguente schema:



L'istruzione viene letta, ed i suoi bit vengono usati come control bit (vedi i simboli "c") per tutti gli altri componenti

Microarchitettura del processore Hack (continua)

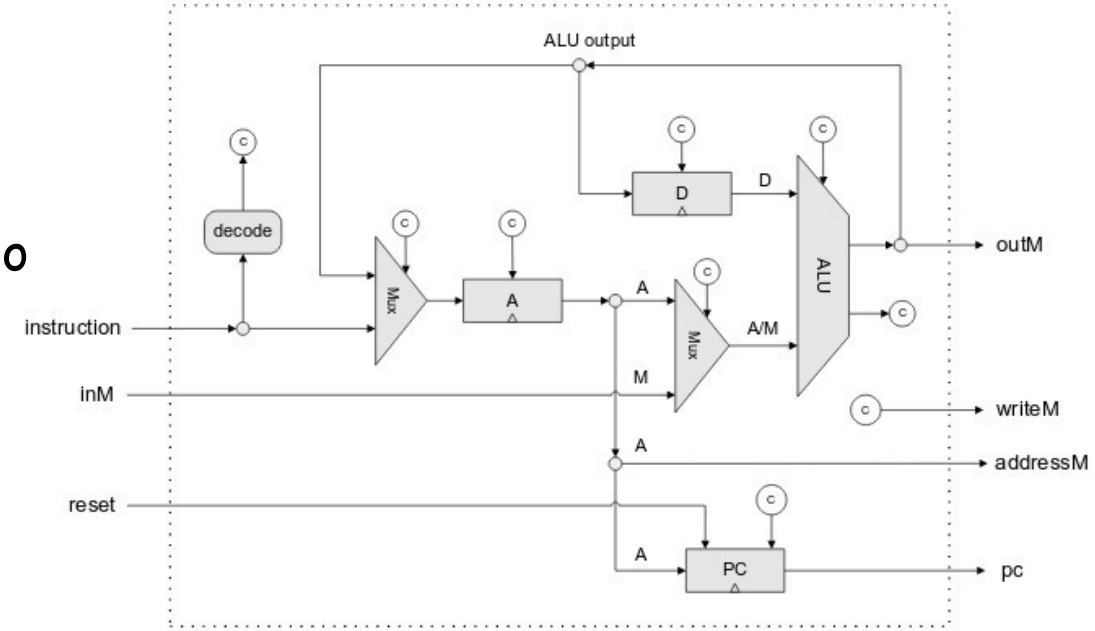
- E' presente la ALU
- Un registro D che contiene uno dei due operandi della ALU, e che può memorizzare un precedente output
- Un registro A che può contenere un dato che fa parte delle istruzioni o un precedente output



- Il secondo input della ALU può essere o il contenuto del registro A oppure un dato proveniente dalla memoria
- E' presente anche il Program Counter che, per quanto riguarda i salti, può essere impostato tramite il registro A
- Il registro A può essere anche usato come puntatore alla memoria (per operazioni di lettura/scrittura)

Microarchitettura del processore Hack (continua)

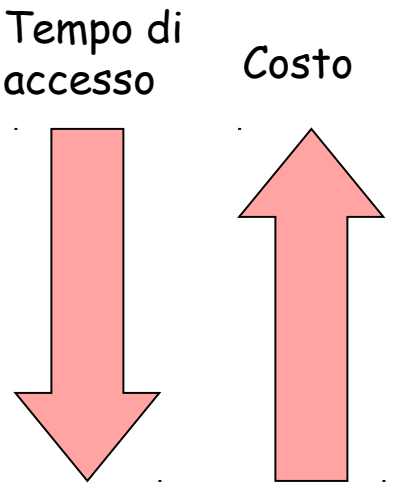
- Il flusso dei dati fra i vari componenti viene controllato tramite Mux
- I Mux ed i bit di controllo dei registri, vengono gestiti da una micro-architettura composta da semplici circuiti combinatori



- Infatti, l'intero ciclo Fetch-Decode-Execute del processore Hack viene eseguito in un solo ciclo di clock, ed i segnali di controllo sono funzione dell'istruzione corrente
  - In altre parole, una istruzione in ingresso al tempo t, viene completamente eseguita entro il tempo t+1
  - Al tempo t+1 viene considerata l'istruzione successiva

Accesso ai dati in memoria

- Le memorie realizzate usando flip-flop come in Hack sono chiamate SRAM (sono tra le più veloci, ma anche fra quelle più costose per quanto riguardo il costo per bit)
- Sono troppo costose per realizzare una intera memoria con tale tecnologia, per questo motivo la memoria viene organizzata secondo una gerarchia (eccone un tipico esempio)
  - SRAM ("static"), usate per le cache
  - DRAM ("dynamic"), usate per la memoria centrale
  - Dischi



- Si usano sofisticati algoritmi di caching e paginazione per gestire il passaggio dei dati tra i vari livelli della gerarchia

SRAM e DRAM

- Le SRAM (Static RAM) sono realizzate tramite flip-flop come le memorie viste in precedenza
  - Veloci (ordine del nanosecondo)
  - Usate principalmente per le cache
- Le DRAM (Dynamic RAM) o SDRAM (Synchronous DRAM), usate per le memorie centrali, hanno un solo transistor ed un condensatore che mantiene (tramite carica elettrica) un singolo bit
  - Visto che il condensatore perde la propria carica, deve essere ricaricato per evitare di perdere la propria informazione
  - Si rendono necessarie periodiche fasi di "refresh" (ad intervalli dell'ordine del millisecondo)
    - A causa del refresh sono più lente (ordine della decina di nanosecondi)
    - Richiedendo un solo transistor costano meno e possono essere maggiormente miniaturizzate

Cache

- Le cache sono largamente utilizzate, ed organizzate in modo sofisticato. Riportiamo un tipico esempio di cache a tre livelli:
  - Una prima piccola cache (livello 1: L1) è direttamente nel chip della CPU separata fra istruzioni e dati (dimensioni fra 16-64 KB)
  - Una seconda cache (livello 2: L2) nel medesimo "involucro" della CPU "unificata" fra dati e istruzioni (fra 512 KB ed 1 MB)
  - Una terza cache (livello 3: L3) esterna alla CPU (alcuni MB)

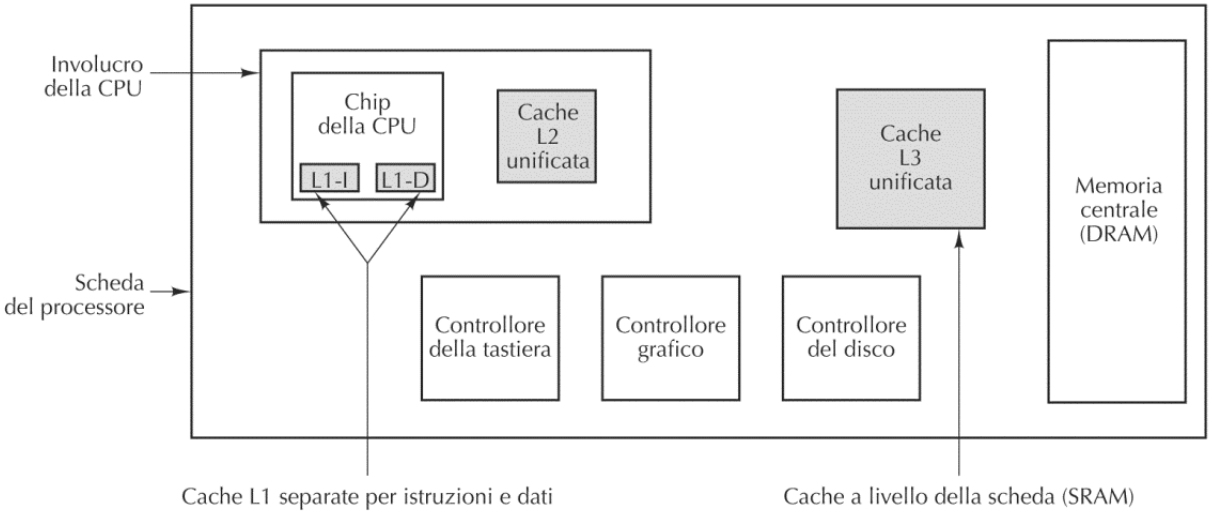


Figura 4.37 Sistema con tre livelli di cache.

Località temporale e località spaziale

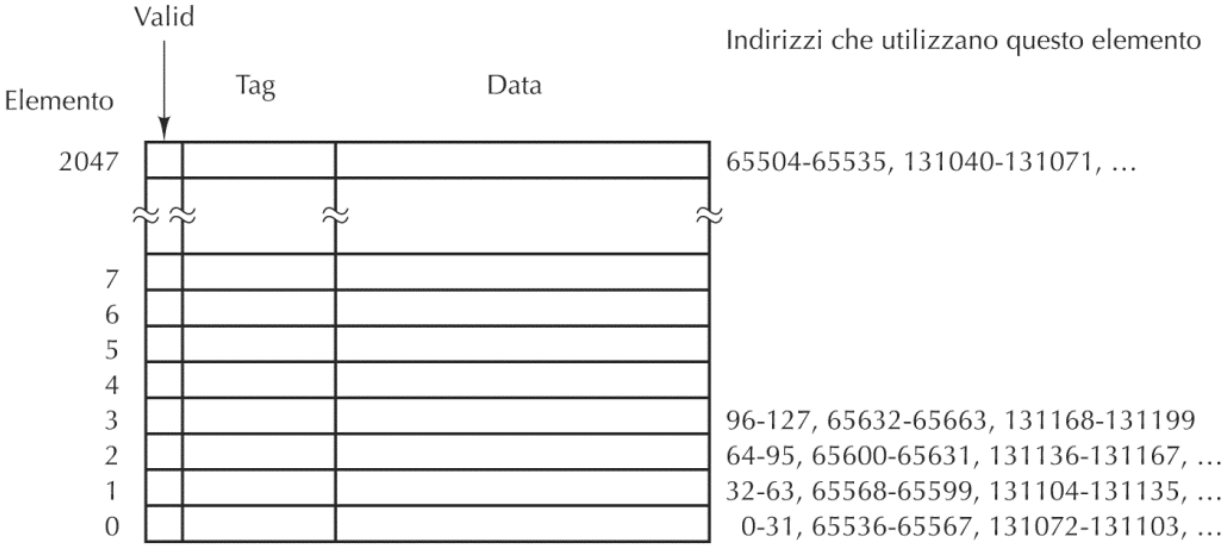
- Per località temporale intendiamo l'alta probabilità che la stessa cella di memoria venga acceduta più volte a breve distanza di tempo
  - Mantenere in cache una cella acceduta di recente rende altamente probabile che una delle prossime operazioni trovi in cache la cella di cui necessita, senza dover andare in memoria centrale
  - Località temporale avviene ad esempio in esecuzioni basate su stack (last-in-first-out) che limitano l'accesso ai soli dati in cima allo stack, o nei loop
- Per località spaziale intendiamo l'alta probabilità che celle di memoria vicine possano essere accedute a breve distanza di tempo
  - Se inseriamo in cache blocchi contigui di celle di memoria è altamente probabile che una cella che verrà a breve acceduta verrà trovata in cache, senza dover andare in memoria centrale
  - Località spaziale dovuta ad esempio a esecuzione sequenziale delle istruzioni e accesso sequenziale ad array

Organizzazione della cache

- A livello di microarchitettura si definisce il modo di organizzare la cache. Esistono diverse politiche di gestione: vediamo come esempio la "cache a corrispondenza diretta" (direct mapped cache)
  - Si suddivide la memoria centrale in blocchi di dimensione m
  - La cache è organizzata in un certo numero, diciamo n, di linee di cache di medesima dimensione m
    - Le linee di cache sono indicizzate da 0 a n-1
  - I blocchi della memoria principale possono essere inseriti in cache secondo uno schema "ad orologio"
    - Il k-esimo blocco in memoria centrale può essere inserito nella linea di cache di indice "k mod n"
    - In cache viene tenuta traccia di quale specifico blocco è attualmente presente in ogni linea di cache
  - Ad ogni accesso in memoria, in base all'indirizzo della cella di memoria, si capisce in quale linea di cache andare a cercarlo

Direct Mapped Cache: esempio

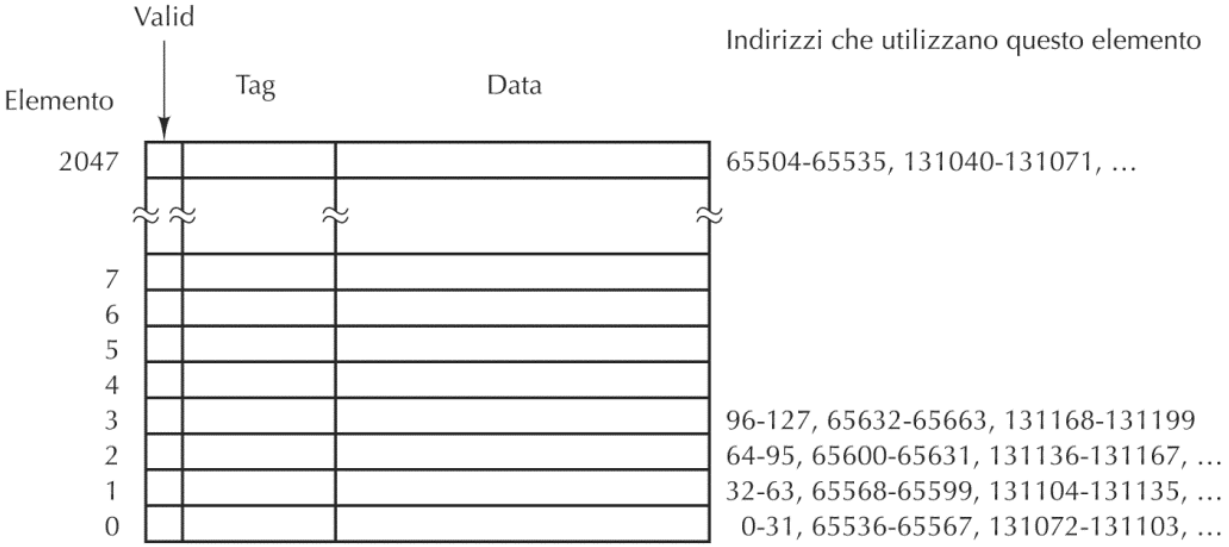
- Immaginiamo ora una cache con n=2048 linee di dimensione m=32 byte



- "Valid": indica se la linea di cache contiene un blocco
  "Data": contiene i 32 byte del blocco
  "Tag": indica esattamente quale blocco è contenuto

Direct Mapped Cache: come accedere alla cache

- Immaginiamo ora una cache con n=2048 linee di dimensione m=32 byte



- Immaginiamo di avere indirizzi di memoria a 32 bit
- I 5 bit meno significativi indirizzano il byte dentro data
- Gli 11 bit successivi indicano quale linea di cache usare
- Gli altri 16 bit vanno confrontati con tag: se coincidono (e se valid è vero) allora il byte cercato è in cache, altrimenti è da cercare in memoria principale

Gestione della cache

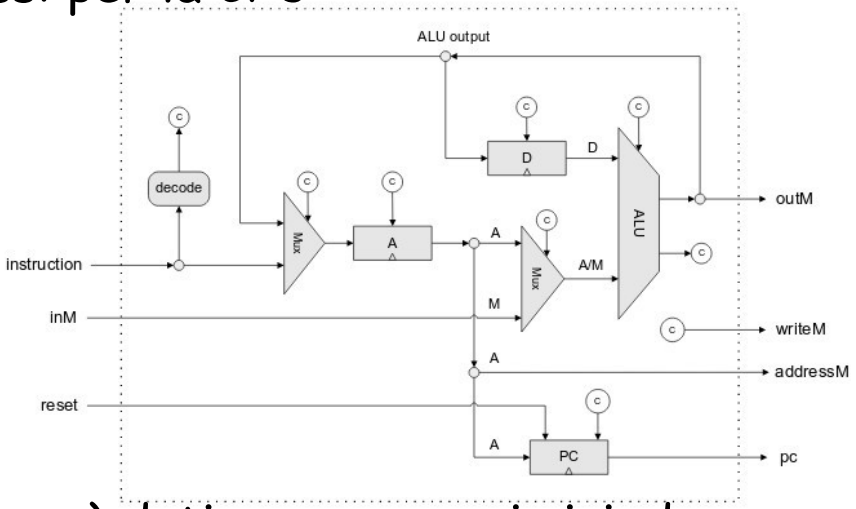
- Quando l'accesso alla cache ha successo si dice che è avvenuta una "cache hit" (successo della cache)
- Quando l'accesso alla cache non ha successo si dice che è avvenuta una "cache miss"
  - In questo caso il blocco di interesse deve essere portato in cache, ma prima il contenuto della corrispondente linea di cache deve essere riportato in memoria
  - Solo in questa fase, una modifica ad una cella di memoria presente nella linea di cache, diventa visibile anche in memoria centrale
- Esiste quindi un momento in cui non c'è coincidenza fra i dati in memoria e i corrispondenti dati in cache
  - Questo può generare problemi quando più processori o più dispositivi accedono alla memoria centrale!
  - Bisogna quindi gestire questi casi di conflitto (ad esempio, vietando l'accesso per i blocchi attualmente in cache alla loro versione in memoria)

Paginazione

- Anche fra memoria centrale e memoria di massa (dischi) esistono politiche simili di spostamento dei dati
  - Solitamente si usa la tecnica della paginazione, gestita dal sistema operativo
  - Blocchi di dati (solitamente chiamate "pagine") sono mantenute in memoria di massa, e vengono spostate in memoria centrale quando tali dati servono
  - Quando i dati non servono più vengono riportati in memoria di massa
  - Questi spostamenti sono gestiti da algoritmi, detti algoritmi di "paginazione", eseguiti dal sistema operativo
- Studieremo queste cose durante l'analisi del livello "sistema operativo"

Pre-fetch istruzioni

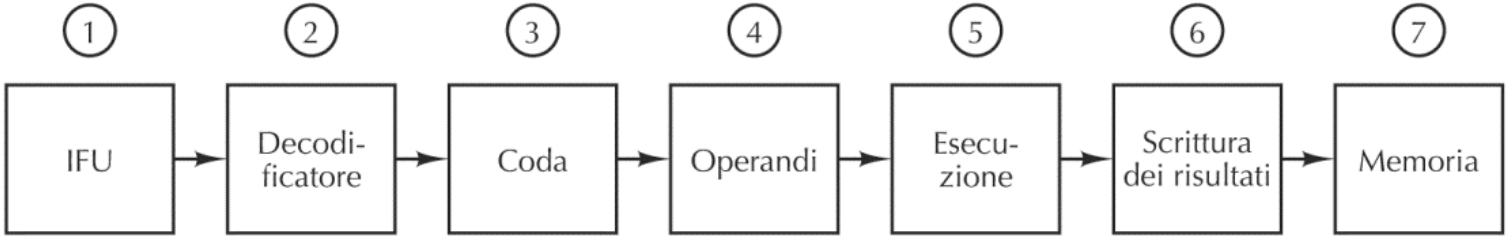
- Nell'architettura del nostro calcolatore Hack, ci siamo semplificati la vita considerando due distinti ingressi per la CPU:
  - "instruction": carica l'istruzione da eseguire da una specifica memoria programma
  - "inM": carica i dati necessari da una distinta memoria dati



- Nelle architetture usuali (Von Neumann) dati e programmi risiedono nella stessa memoria
- In ogni caso caricare un'istruzione e poi i suoi operandi richiede un ciclo di clock molto lungo
- Una tecnica largamente usata prevede di pre-caricare la prossima istruzione mentre la precedente è in esecuzione
  - Si usa hardware dedicato chiamato IFU: Instruction Fetch Unit

Pipeline

- L'aggiunta di una unità indipendente di pre-fetch dell'istruzione è un primo esempio di pipeline
  - Primo stadio: pre-fetch istruzione
  - Secondo stadio: decodifica ed esecuzione
  - In un dato istante ci sono due istruzioni contemporaneamente in elaborazione nel processore, una istruzione nel primo stadio ed una al secondo stadio
- Esistono pipeline più complesse, ad esempio la seguente a 7 stadi:



Problema dei salti durante l'esecuzione in pipeline

- Le istruzioni vengono inserite nel pipeline in modo sequenziale
  - Istruzione i-esima, (i+1)-esima, (i+2)-esima,…
- Cosa succede quando l'istruzione i-esima è un salto (da istruzione i-esima a istruzione k-esima)? I salti possono essere molto frequenti!

```
if (i == 0)
      k = 1;
else
      k = 2;
```

(a)

```
        CMP i,0      ; confronta i con 0
        BNE Else     ; salta a Else se diversi
Then:   MOV k,1      ; sposta 1 in k
        BR  Next     ; salta a Next
Else:   MOV k,2      ; sposta 2 in k
Next:
```

(b)

Figura 4.40 (a) Frammento di programma. (b) Il frammento tradotto in un linguaggio assemblativo.

- Se ci accorgiamo del salto allo stadio v della pipeline (ad esempio dopo l'"esecuzione", stadio 5 nella pipeline della slide precedente) le istruzioni successive già negli stadi 1..(v-1) devono essere scartate

Predizione dei salti

- Le microarchitetture moderne usano meccanismi sofisticati per cercare di predire i salti già dai primi stadi
- Per i salti incondizionati, si riescono a intercettare in fase di decode (solitamente stadio 2)
  - per evitare di fare lavoro inutile, si può mettere sempre una istruzione nulla "nop" (no operation) dopo i salti incondizionati
- Per predire salti condizionati si usano "euristiche"
  - i salti all'indietro è più probabile che vengano eseguiti rispetto a salti in avanti
    - i salti indietro sono usati alla fine dei cicli per tornare ad inizio ciclo (vedi "for" in C)
  - se una istruzione ha generato un salto nelle sue ultime due esecuzioni, mi aspetto che anche la prossima volta salti;
    se invece non ha saltato nelle ultime due esecuzioni mi aspetto che non salti nemmeno la prossima volta

Problema dell'accesso concorrente ai registri

- Può succedere che istruzioni in stadi diversi accedano ai medesimi registri (bisogna fare attenzione all'ordine in cui eseguire tali operazioni "concorrenti")

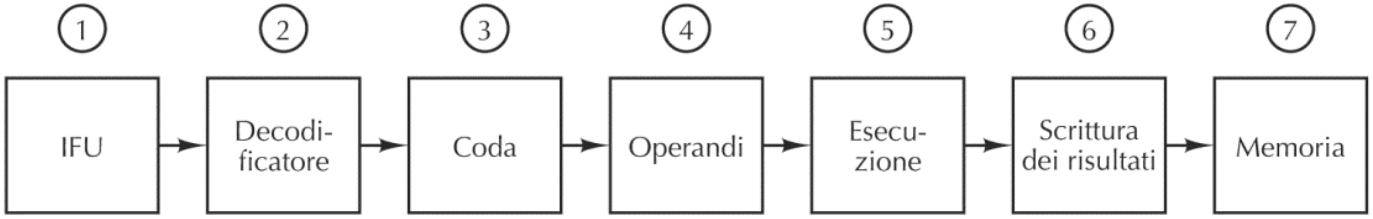


Figura 4.36 Pipeline di Mic-4.

  - Esempio di ordine sbagliato: istruzione in stadio 6 modifica i registri che una istruzione allo stadio 4 sta leggendo per caricare i propri operandi (dipendenza RAW=Read After Write)
- In questi casi l'istruzione allo stadio 4 deve aspettare, bloccando anche quelle dietro (stall)
- Si possono usare tecniche di riordino delle istruzioni per limitare queste situazioni
- Se le istruzioni vengono riordinate dalla CPU la situazione si complica ulteriormente

Conclusione

- Concludiamo riportando una microarchitettura che rappresenta l'attuale stato dell'arte

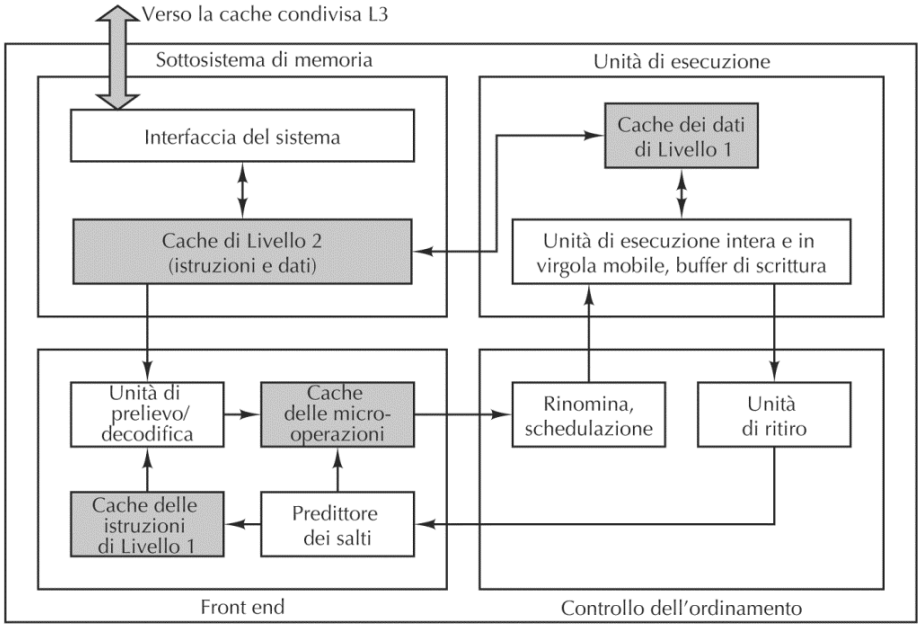


Figura 4.46 Diagramma a blocchi della microarchitettura Sandy Bridge del Core i7.

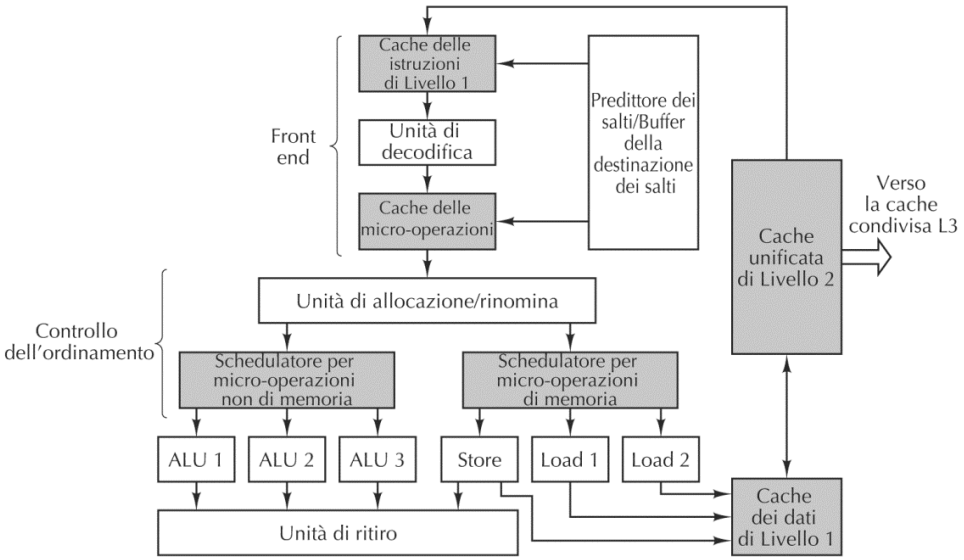


Figura 4.47 Vista semplificata del percorso dati del Corei7.

- L'unità di "ritiro" ha lo scopo di rendere definitivi i risultati delle istruzioni (memorizzati temporaneamente in registri "invisibili") nell'ordine giusto
  - un dato in un registro "invisibile" non viene spostato fino a che non sono state ritirate tutte le istruzioni precedenti

Ecco un'immagine reale dell'Intel Core i7

- La microarchitettura descritta si replica per vari core disponibili all'interno del processore

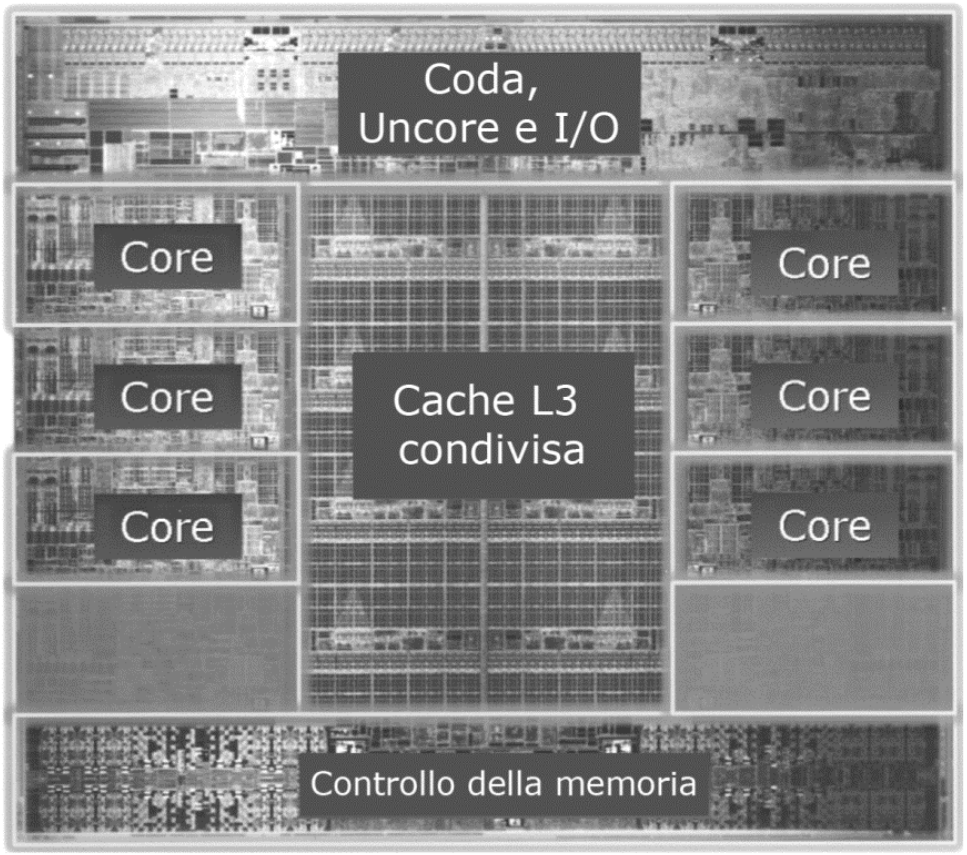


Figura 1.12 Il microprocessore Intel Core i7-3960X, © 2011 Intel Corporation. Il chip misura 21×21 mm^2 e contiene 2,27 miliardi di transistor (con "Uncore" si intendono le funzionalità esterne al core).