UNIVERSITÀ
degli STUDI
di CATANIA

Il Processore o CPU

Architettura degli elaboratori AA 2023/24

Corso di Laurea Triennale in Informatica

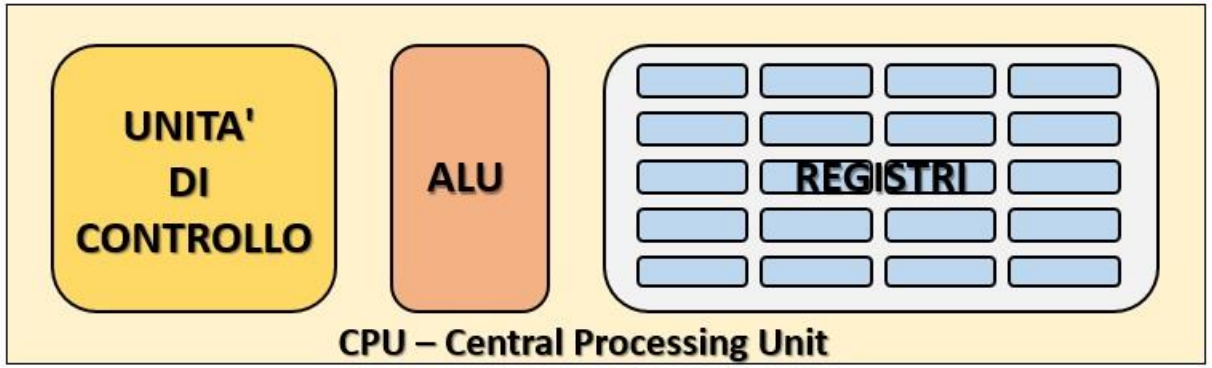
Massimo Orazio Spata

massimo.spata@unict.it

Dipartimento di Matematica e Informatica

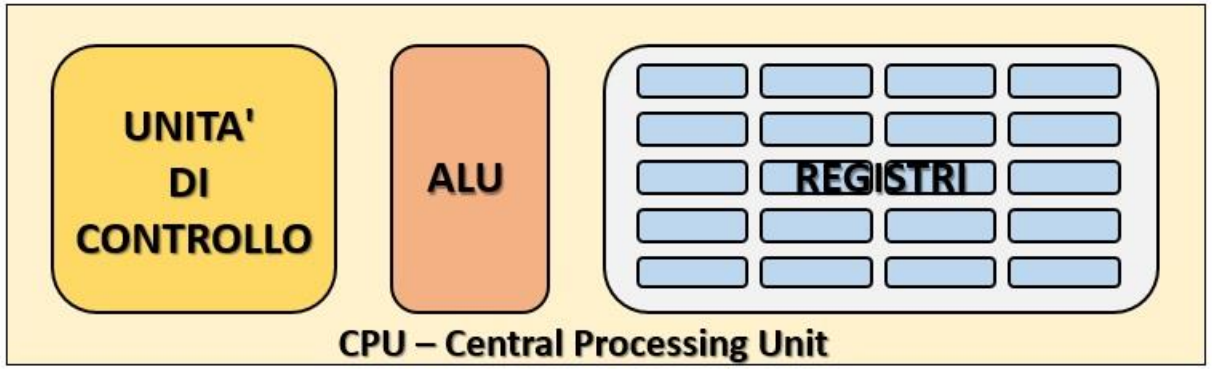
Componenti di un Processore

- Un processore è un singolo circuito integrato in grado di effettuare operazioni decisionali, di calcolo o di elaborazione dell'informazione; spesso il processore è indicato con la sigla CPU (Central Processor Unit).
- Il processore può essere visto come suddiviso in tre unità funzionali, l'CU (unità di controllo), l'area dei registri e l'ALU (unità aritmetico-logica).La CU si affaccia sul bus, lo arbitra impostando i valori sulle linee ABus, DBus e CBus, legge il DBus e il CBus, legge dalla memoria (e dall'I/O) i dati o li aggiorna in memoria (o nell'I/O) dopo aver compiuto operazioni.



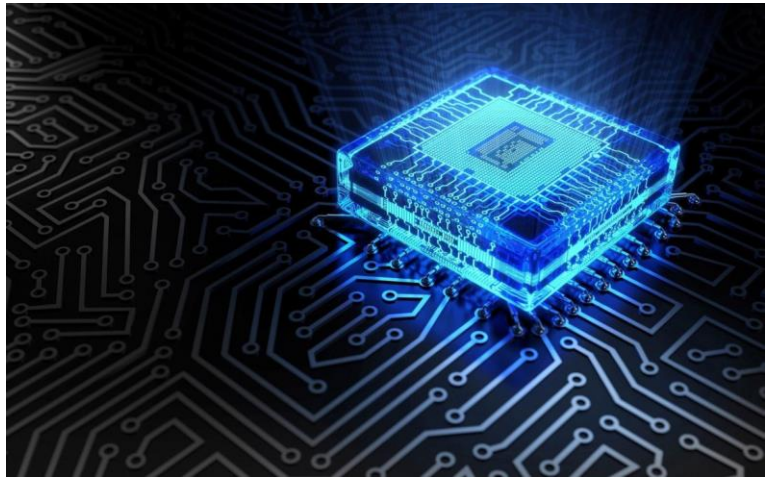
Registri e ALU

- I registri contengono i dati letti dall'UC sul bus per predisporli all'esecuzione delle istruzioni che avverranno nell'ALU; oppure contengono i risultati delle operazioni compiute dall'ALU in attesa di essere passati allUC e quindi sul bus.
- L’ALU è l'unità di esecuzione effettiva del processore, all'interno della quale si trovano microprogrammi cablati direttamente in hardware, scritti nel cosiddetto microcodice con relative microistruzioni.



Instruction Set

- Ogni processore viene progettato con un set di istruzioni specifico denominato ISA (Instruction Set Architecture o Instruction Set), in corrispondenza di ognuna delle quali è implementato un preciso microprogramma in ALU.
- Ogni istruzione dell'ISA è contraddistinta da un numero specifico, denominato Op. Code (Operation Code o op.cod.) e ogni istruzione dotata di Op.Code necessita di un numero preciso e definito di parametri (operandi) che, assieme all'Op. Code, determinano la lunghezza dell'istruzione (in byte).
- Ad ogni Op. Code si associa anche una breve descrizione in lingua naturale che ne ricorda la funzione, detta codice mnemonico. Un registro speciale del processore, detto PC (Program Counter), si incrementa automaticamente della lunghezza dell'istruzione appena eseguita.



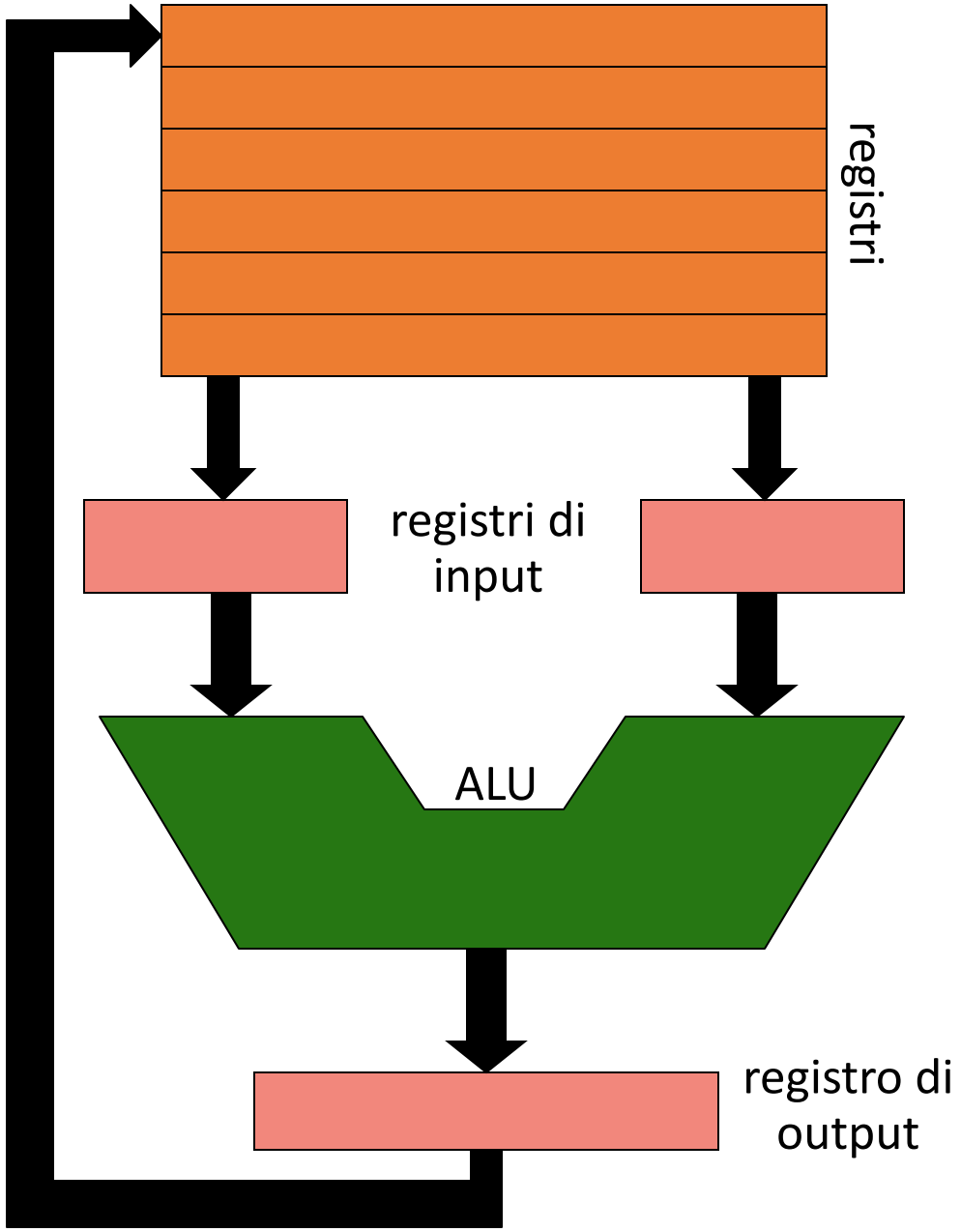
Il data path

L’ALU, i registri e molti bus costituiscono il data path.

L’ALU puo’ avere 2 registri di input e 1 di output.

I registri sono collegati ai registri di input.

Il registro di output e’ collegato ai registri.

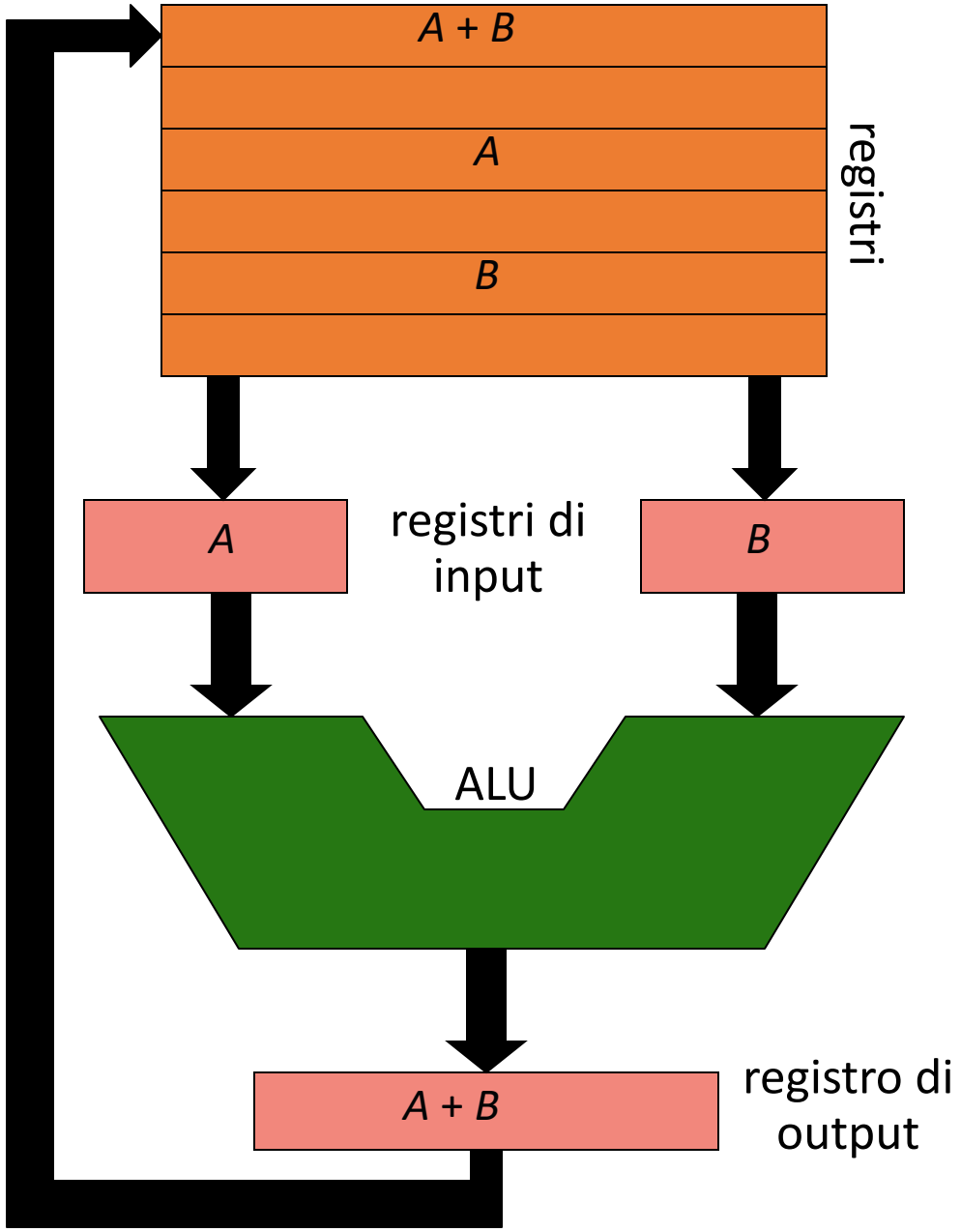


Le istruzioni

registro-memoria: trasferisce un dato dalla memoria (esterna) ai registri, o viceversa

registro-registro: trasferisce un dato da un registro a un altro

Ciclo del data path: passaggio di due operandi attraverso la ALU e memorizzazione del risultato nei registri.

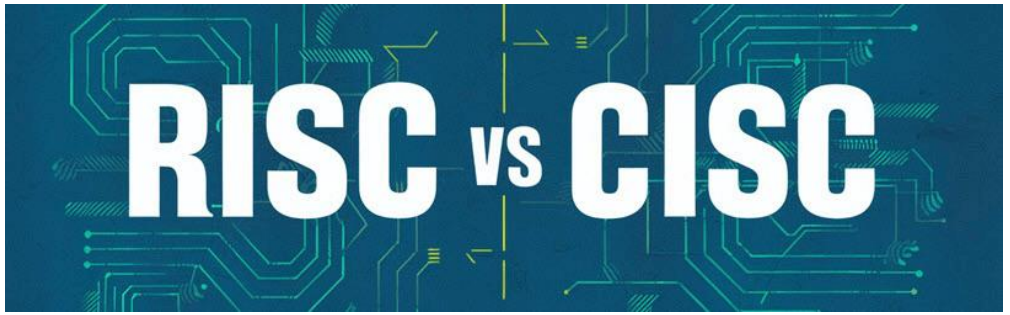


Implementazione delle istruzioni

1. via hardware, nella CPU
2. via software, ossia con un interprete implementato secondo (1)

- Esistevano le architetture *CISC* (*Complex Instruction Set Computer*), sostenute dai colossi IBM, VAX, Intel, … basate su **molte** e **complicate** istruzioni, **interpretate**
- Nascono negli anni ’80 le architetture *RISC* (*Reduced Instruction Set Computer*), come il DEC Alpha, basate su **poche** e **semplici** istruzioni, **non interpretate**

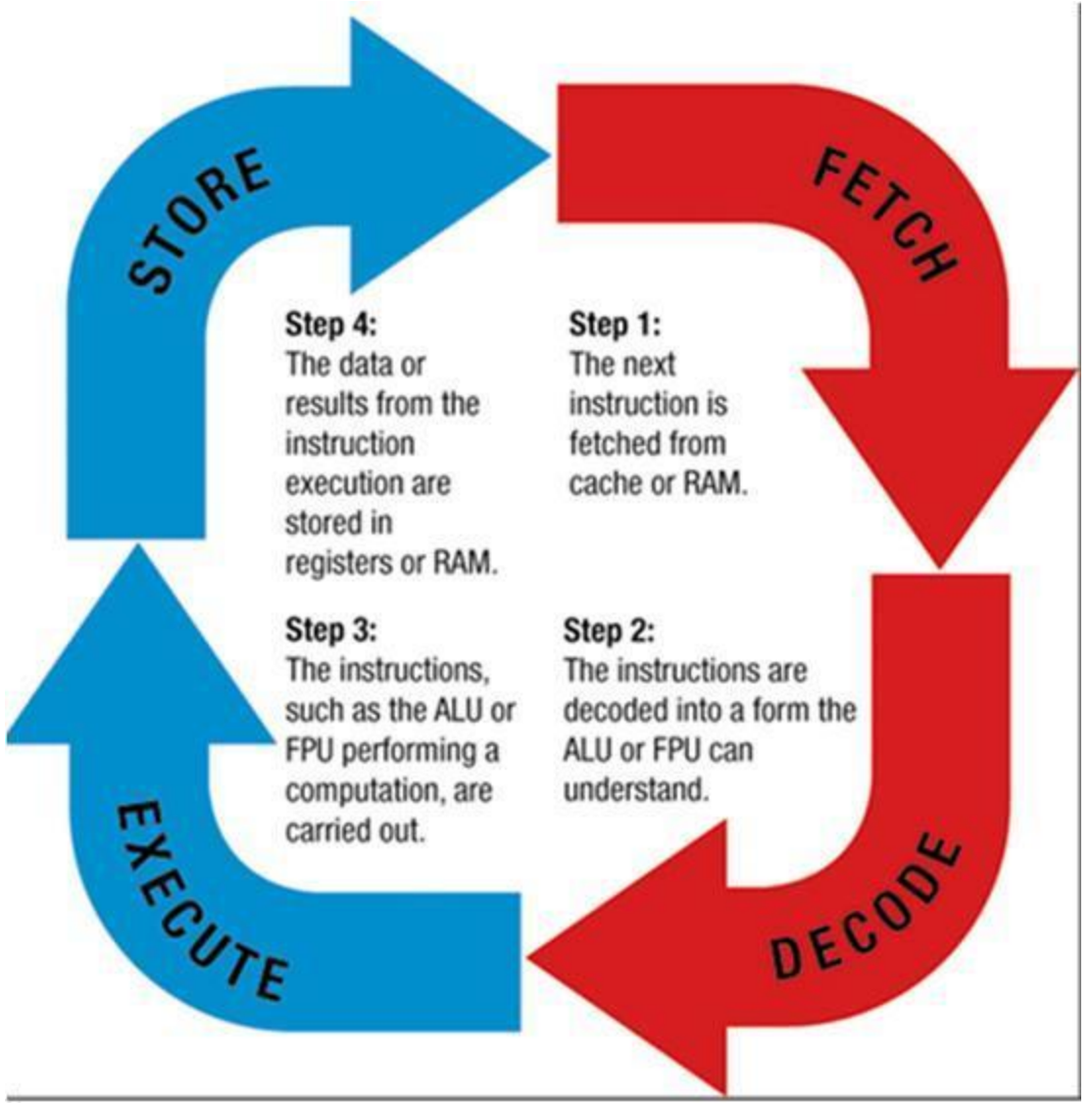
Oggi, anche il Pentium e’ ibridato con tecnologia *RISC*.



Fetch-Decode-Execute-Store

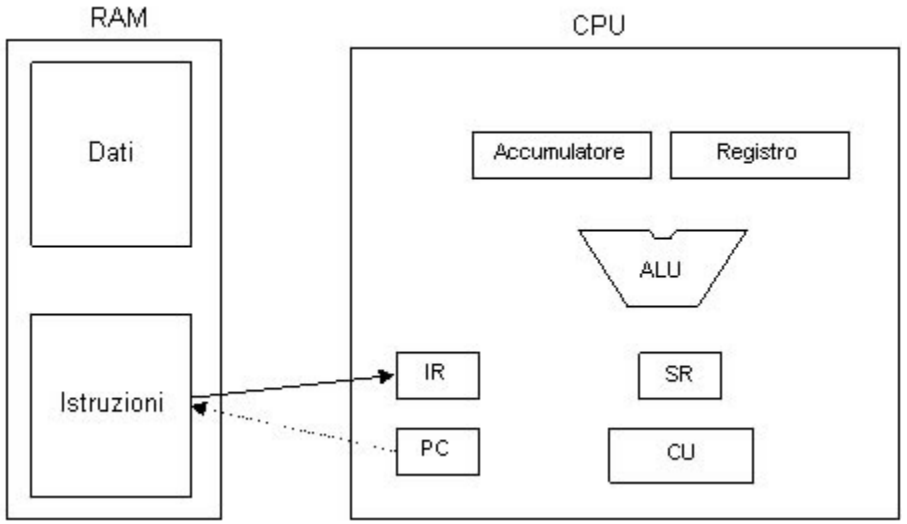
Il processore tipico quindi agisce secondo una rigida sequenza di passi che si ripetono fino all'arresto della macchina:

- **Fetch**: l'UC pone sull'ABus il valore del registro Program Counter, imposta lettura da memoria e carica l'Op. Code ivi memorizzato
- **Decode**: l'UC, a partire dall'Op.Code appena letto, determina la lunghezza dell'istruzione, cioè la quantità di parametri di cui necessita, quindi attiva una fase intermedia di caricamento degli operandi (Operand Fetch) che si trovano necessariamente e in modo ordinato agli indirizzi adiacenti a quello dell'Op. Code. Gli operandi caricati andranno a depositarsi nei registri.
- **Execute**: viene avviato il microprogramma relativo all'Op. Code attuale, che usa i propri parametri correttamente memorizzati nei registri. La frequenza in base alla quale vengono eseguiti i microprogrammi è regolata dal clock di CPU (frequenza del microprocessore).
- **Store**: al termine della fase di Execute gli eventuali risultati, posti nei registri, vengono scritti attraverso il bus dall'UC, o verso la memoria, o verso IO.



Terminazione del ciclo di CPU

- Il ciclo del processore qui descritto termina, in effetti, consultando il segnale INTR per capire se il controller di una periferica ha richiesto una interruzione, ovvero la sospensione temporanea dell'esecuzione per servire il codice associato alla interruzione pendente.

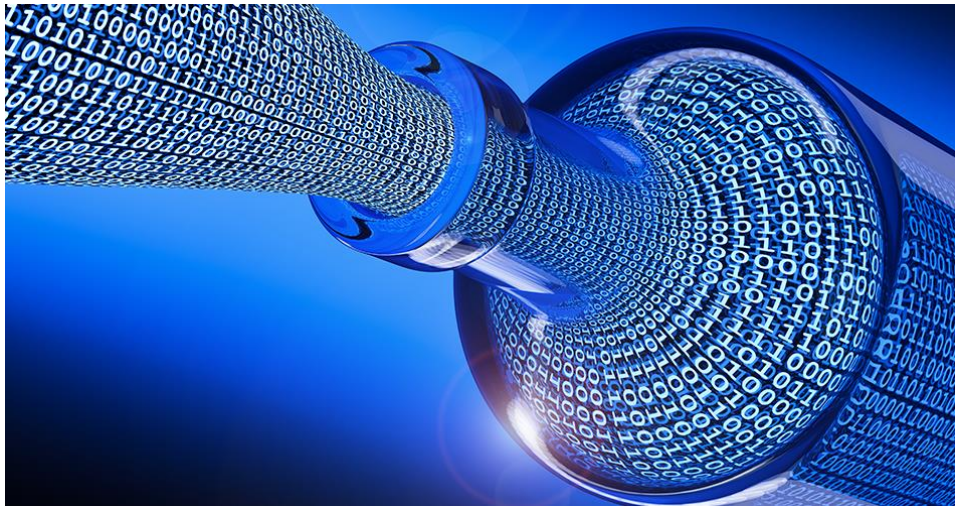


Tempo di esecuzione di un microprogramma

- Ogni singola istruzione dell'ISA di un processore è contraddistinta da un proprio Op. Code, una determinata lunghezza (in base al numero di operandi che utilizza) e un preciso numero di cicli di bus per il suo completamento (compresi tra il fetch, il decode e lo store).
- Il tempo di effettiva esecuzione del microprogramma influisce relativamente sulla durata dell'istruzione, essendo il clock di CPU di almeno un ordine di grandezza superiore al clock di bus.

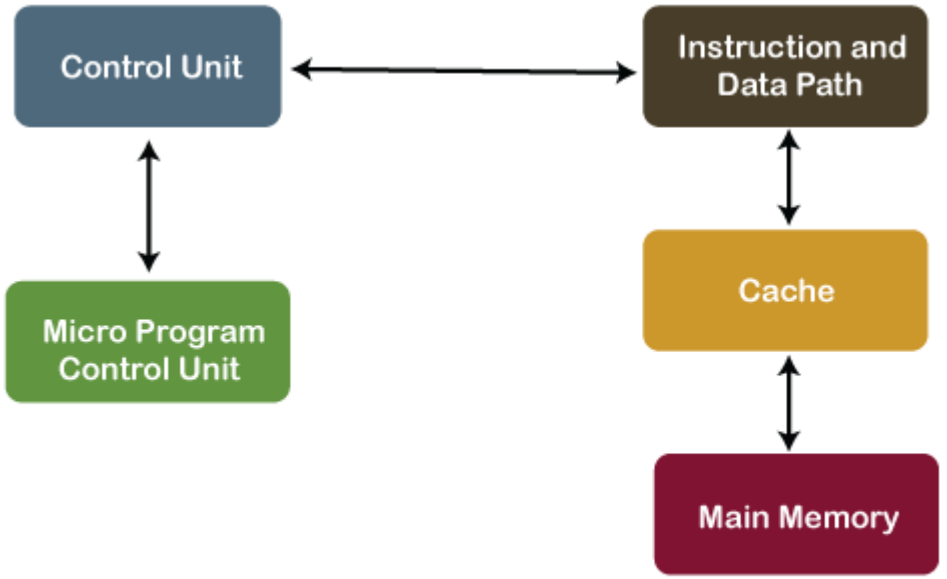
Bottleneck

- Questa considerazione ricorda di nuovo il già citato collo di bottiglia dell'architettura CISC, causato da una fase di decode molto lunga e onerosa a causa della quantità di istruzioni dell'ISA e della loro complessità e varietà in termini di numero di operandi.
- Ovviamente lo schema è semplificato. Per esempio un microprocessore reale contiene, oltre all'ALU, una FPU (Float Point Unit) per i calcoli in virgola mobile e varie unità di calcolo per istruzioni complesse (per esempio per calcoli vettoriali), ma il modello di riferimento rimane ancora concettualmente significativo.



CISC

- Il modello di processore appena descritto contiene microprogrammi e microcodice, cioè si dice che è un microprocessore «a interprete».
- In altre parole, ogni istruzione dell'ISA deve essere decodificata (decode), quindi dotata degli operandi che richiede (operand fetch) e, infine, avviata alla fase di esecuzione (execute) che richiede l'avvio di uno specifico microprogramma.
- Tutto questo significa che ogni singola istruzione di un'ISA del genere ha uno svolgimento (o data path) a più cicli.
- Il data path è il percorso dei dati all'interno del processore, e i suoi cicli sono scanditi dal clock della CPU.



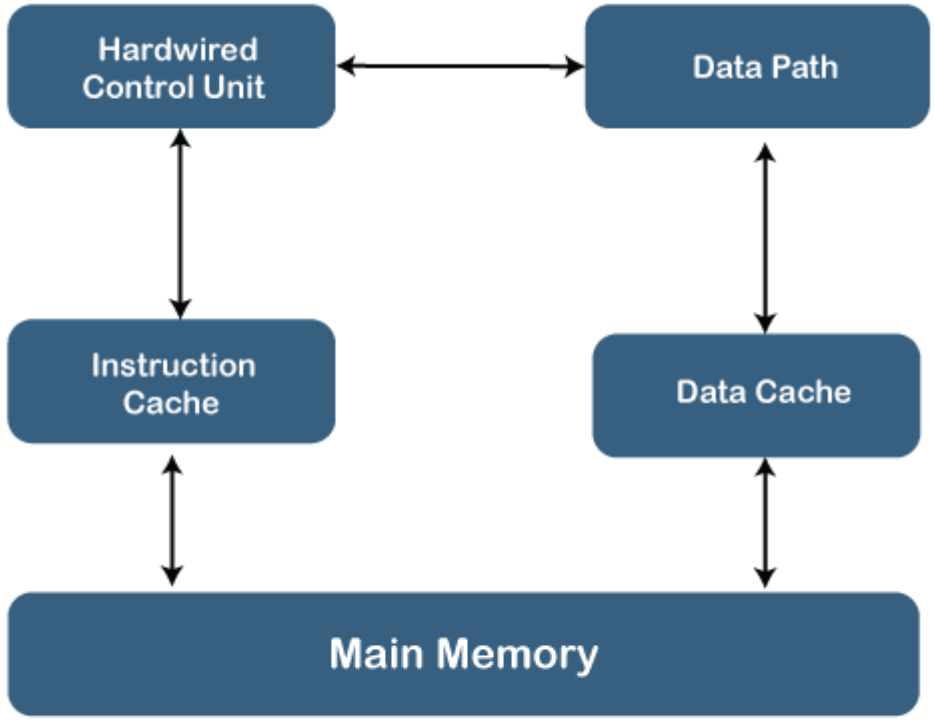
CISC Architecture

CISC

- Questi tipi di ISA sono denominati **CISC** (*Complex Instruction Set Code*).
- Le architetture **CISC** possiedono un set di istruzioni numeroso; le ampiezze delle istruzioni sono molto variabili, con corrispondente fase di depier complessa, da svolgersi con un data path a più passi.
- Si tratta di architetture che facilitano la portabilità del software, dato che l'insieme dei microprogrammi (o interprete del processore) può essere trasportato su processori più recenti, e quindi sono processori adatti per essere programmati anche in Assembly.

RISC

- Al contrario, un'architettura **RISC** (*Reducted Instruction Set Code*), possiede un data path a singolo passo.
- Il set di istruzioni di una architettura **RISC** è limitato, contiene istruzioni di lunghezza costante (con un numero di operandi fisso), con fase di decode breve e senza microprogrammi da eseguire nel processore: ogni istruzione è eseguita direttamente in hardware con pochi cicli di clock.
- In questo modo una elaborazione **RISC** appare nettamente più veloce (almeno di un ordine 10).
- In sostanza un'istruzione **CISC** - con molti passi nel data path - equivale a numerose istruzioni **RISC** con data path singolo.
- Per questo i programmi per **ISA RISC** sono molto più lunghi di un analogo programma per **ISA CISC**.



RISC Architecture

CRISC

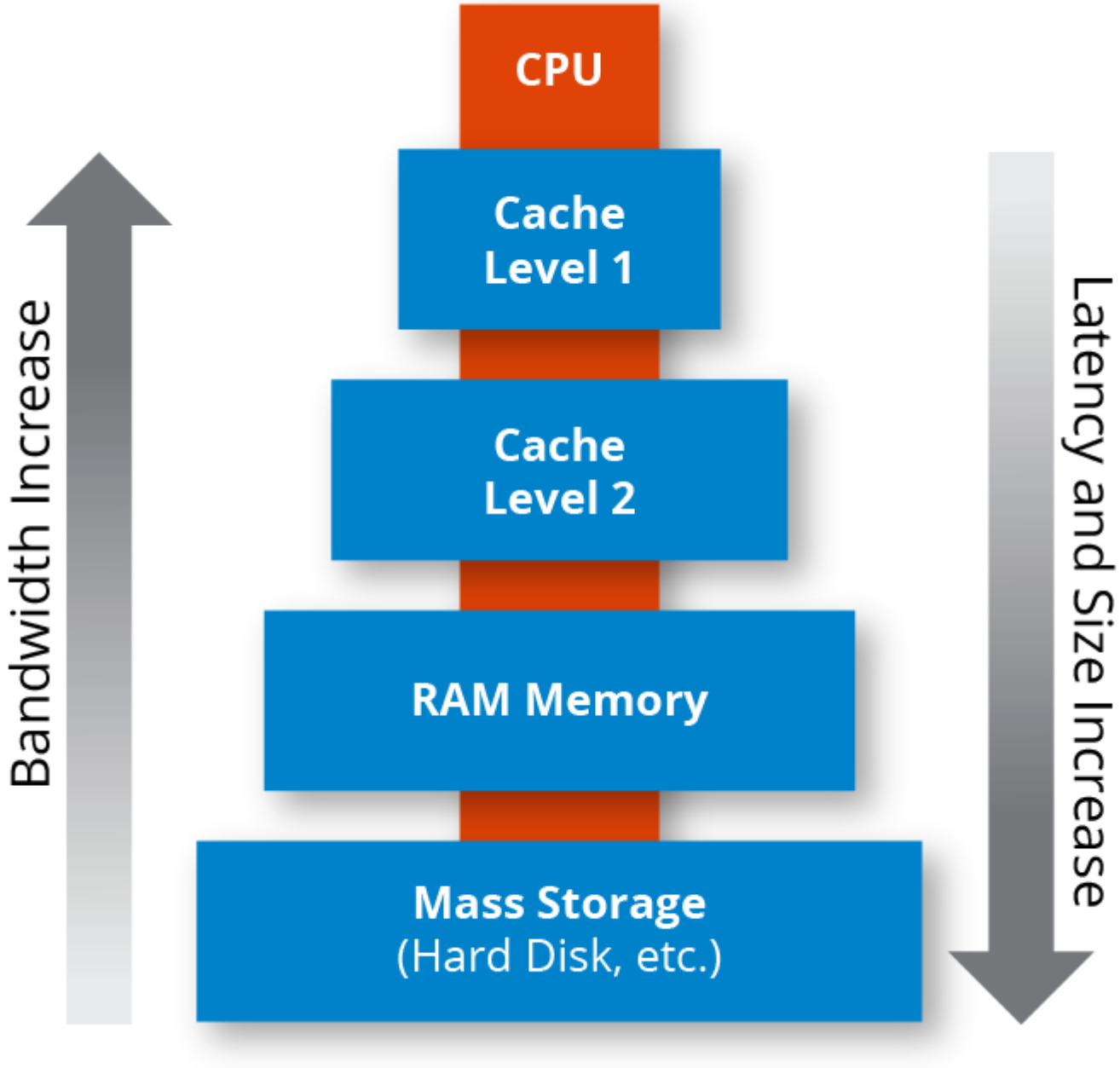
- Tutto ciò implica maggiori difficoltà per la portabilità del software e maggiore complessità dei compilatori.
- I calcolatori RISC sono difficilmente programmabili in Assembly, causa la mancanza di istruzioni ISA di alto livello.
- Nessuna delle due architetture è ideale.
- Piuttosto la tendenza attuale è l'implementazione di processori su base CISC, come descrive il modello di von Neumann, dotati di sottosistemi interni basati su RISC, soprattutto dedicati alla computazione delle istruzioni semplici (e più comuni) dell'ISA adottata. In questo caso si parla di architetture **CRISC** (*Complex Reducted Instruction Set Code*).

SISD/CISC

- Il modello SISD/CISC dell'architettura di von Neumann si scontra con un paio di problemi che ne limitano, strutturalmente, la performance.
- Una singola istruzione su singolo dato eseguita nell'unità di tempo è un limite da tempo inaccettabile. Molte delle soluzioni per incrementare le prestazioni di un calcolatore tendono a parallelizzare l'esecuzione. Questo è il limite di un'architettura SISD.
- Così come il cronico ritardo con cui si accede alla memoria (clock del bus, sull'ordine dei MHz), rispetto alla velocità di esecuzione del processore (clock della CPU, sull'ordine dei GHz), penalizza enormemente il modello CISC con una fase di decode troppo lenta e complessa.
- Molte delle soluzioni per incrementare le prestazioni di un calcolatore tendono a fornire istruzioni e operandi dalla memoria alla stessa velocità che impiega il processore per eseguirle. Questo è il limite di un'architettura CISC.

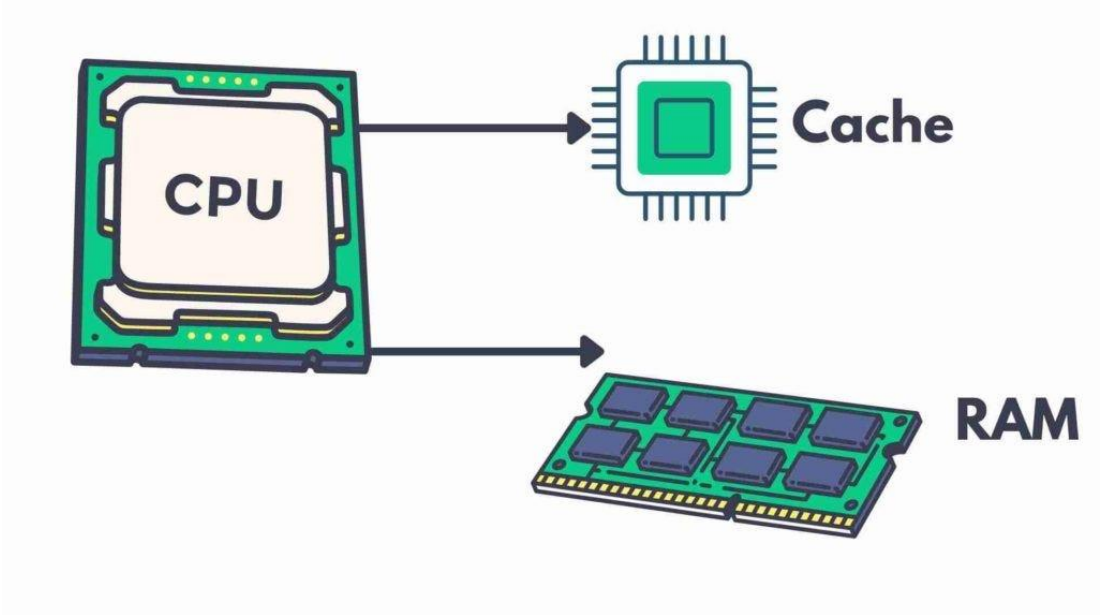
Cache

- Un modo ingegnoso per diminuire gli accessi al bus e alla memoria, e quindi di contrastare i limiti di un'architettura CISC, è quello di dotare il calcolatore di una memoria «tampone» (Cache Memory) tra il processore e il bus.



Cache

- Man mano che il processore legge dalla memoria, ad un determinato in-dirizzo, molte locazioni di memoria con indirizzi prossimi a quello, vengono spostati nella memoria cache (nello stesso tempo di bus).
- Questi gruppi di valori che vengono portati nella cache sono detti linee di cache.



Cache

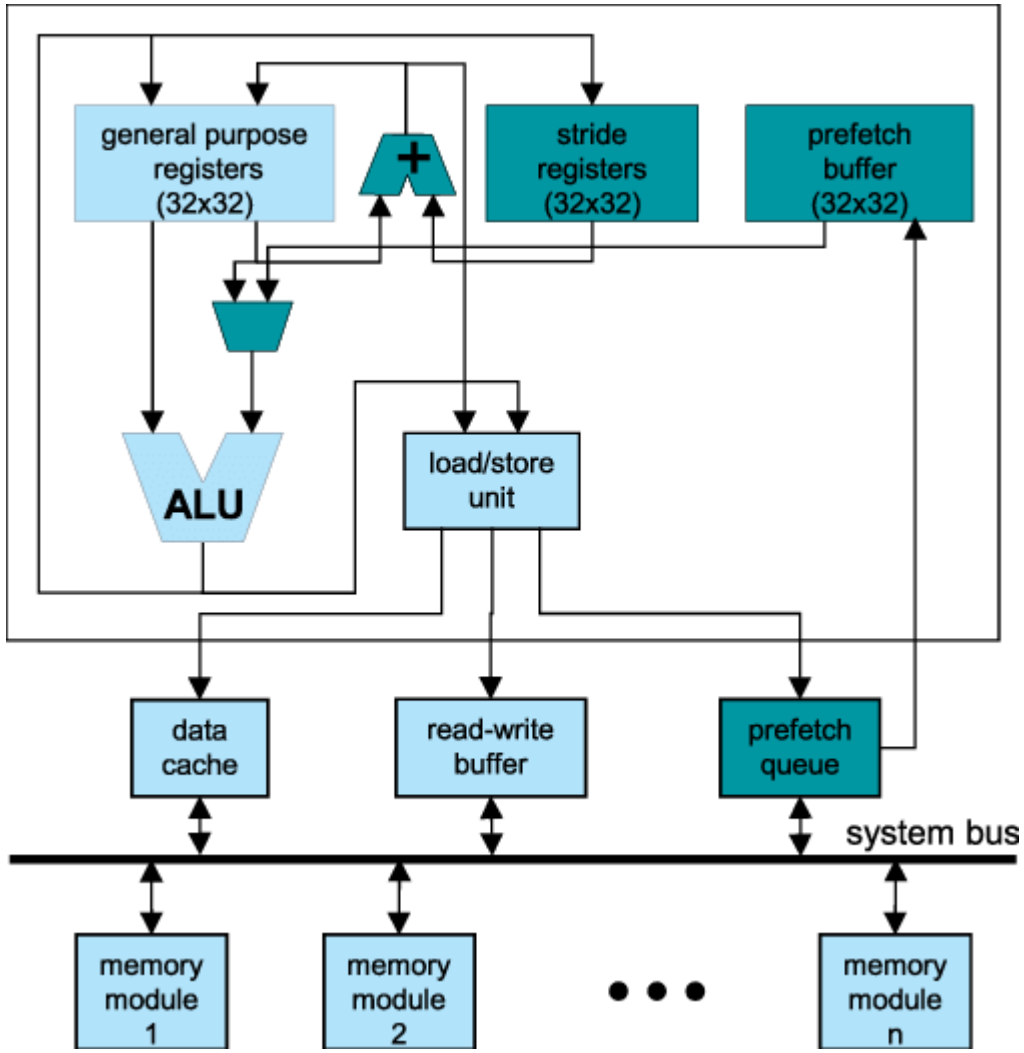
- Il motivo è che, secondo il principio della località spazio-temporale, un programma utilizza, entro un breve intervallo di tempo, solo una piccola parte del suo spazio degli indirizzi, una parte composta da indirizzi numericamente prossimi tra di loro.
- Cosicché la memoria cache (realizzata in SRAM), che è molto più veloce di una memoria DRAM, può fornire, nell'immediato futuro (per esempio, la prossima istruzione da eseguire), i valori desiderati senza dover accedere al bus.

Cache

- Quindi, ad ogni operazione di lettura (del processore dalla memoria), l'informazione viene cercata prima di tutto nella cache; se è presente (hit), non è necessario accedere al bus; se non è presente (miss) si accede alla memoria e, oltre all'informazione richiesta, si carica una nuova linea in cache, sovrascrivendo la linea di cache meno usata di recente.
- Nei calcolatori sono montate almeno tre tipi di memorie cache (livelli):
  - **Livello 1**, all'interno del processore;
  - **Livello 2**, collegato al processore;
  - **Livello 3** sulla piastra madre.

Prefetch

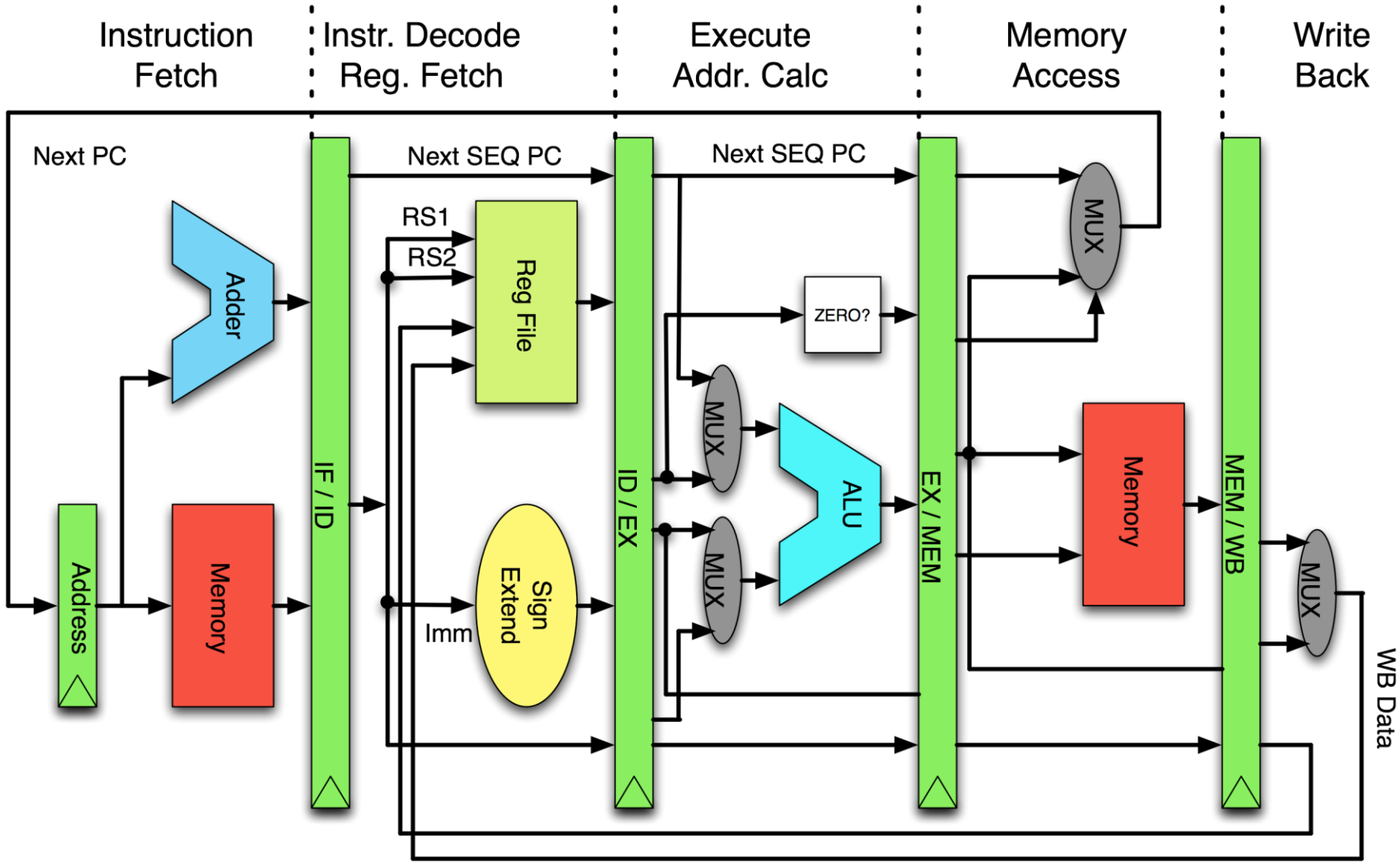
- Un modo per aumentare il parallelismo d'esecuzione, e quindi di superare i limiti di un'architettura SISD, fu quello di caricare nel processore più istruzioni oltre a quella richiesta.
- Fin dagli esordi, per esempio, i processori erano dotati di una coda di **Prefetch**, ovvero di un buffer interno in cui il processore memorizzava i successivi 6 o 8 byte consecutivi a quello appena letto dalla memoria. In questo modo, con un solo accesso alla memoria, si aveva a disposizione una serie di valori che potevano essere usati successivamente (come istruzioni od operandi) senza dover accedere di nuovo al bus.



Pipeline

L'elaborazione di un'istruzione da parte di un processore si compone di cinque passaggi fondamentali:

- IF (instruction fetch): lettura dell'istruzione da memoria;
- ID (instruction decode): decodifica istruzione e lettura operandi da registri;
- EX (execution): esecuzione dell'istruzione;
- MEM (memory): attivazione della memoria (solo per certe istruzioni);
- WB (write back): scrittura del risultato nel registro opportuno;



Pipeline

- Ben presto, alla coda di **Prefetch**, fu affiancato un sistema a **Pipeline** che ha lo scopo di sfruttare il concetto di catena di montaggio: invece di eseguire un'istruzione completamente e solo al termine la sua successiva, si può avviare l'istruzione subito dopo che la precedente è stata inserita nel data path.
- Per esempio, basta che la prima istruzione si trovi in fase di decode, e la successiva può essere posta in stato di fetch. Così come in una catena di montaggio, un nuovo pezzo può essere lavorato anche se il precedente non è ancora stato completato: basta che le fasi (dette anche stadi della pipeline) non si sovrappongano.

Pipeline

- Così, una **Pipeline** a 5 stadi trasporta cinque istruzioni in catena di montaggio.
- La **Pipeline** sopperisce così alle attese di CPU veloci nei confronti di memorie lente (*collo di bottiglia di von Neumann*).
- Una volta dotato di **Pipeline**, in un processore si è notato che lo stadio di esecuzione è il più lento: lo stadio precedente fornisce più valori di quanto lo stadio di esecuzione, implementato nell'ALU, può elaborare.
- Ecco allora che sui processori sono montate più ALU in modo da servire velocemente

Superscalarità

- Ecco allora che sui processori sono montate più ALU in modo da servire velocemente ogni istruzione che arriva allo stadio di execute. In questo caso il processoreè detto **Superscalare**.
- In questo modo è possibile dotare i processori anche di due o quattro pipeline differenti.

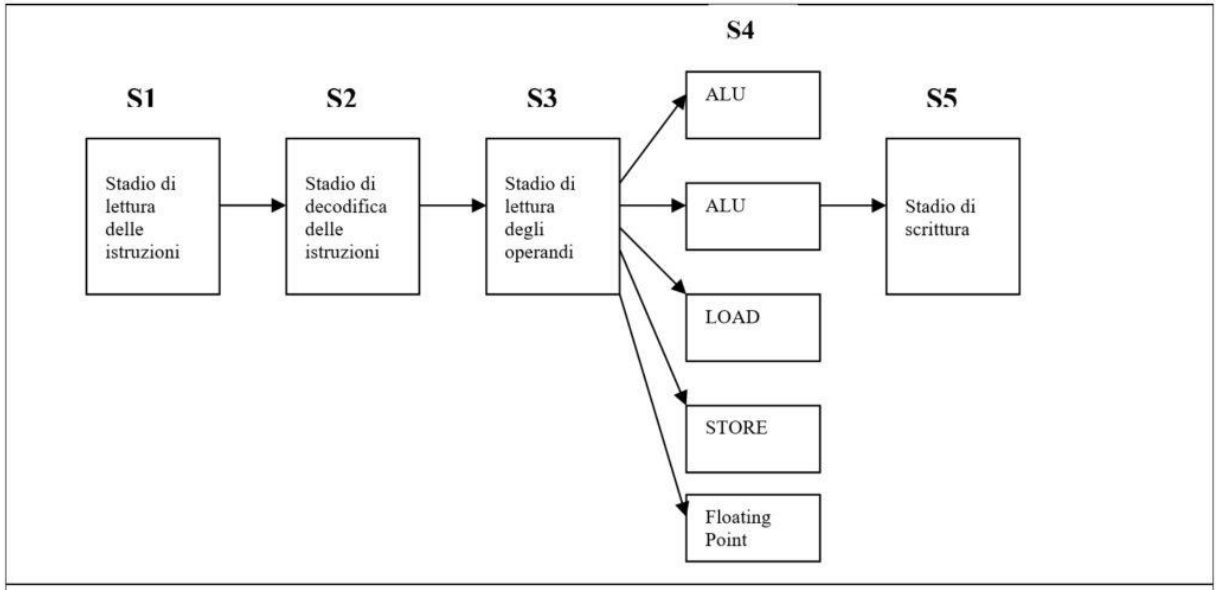


Figura 11. Processore superscalare ad una sola pipeline ma con cinque unità funzionali.

Problema!

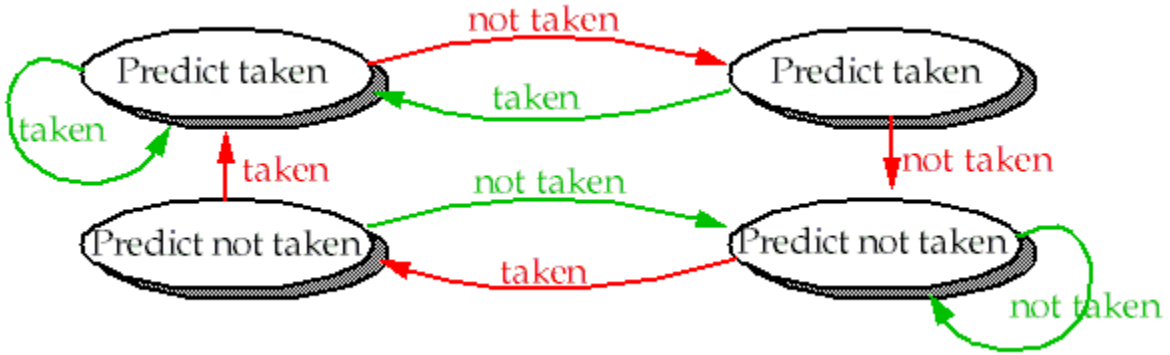
- Tutte queste metodologie, però, vengono vanificate da due ovvie situazioni:
  - Istruzioni di salto;
  - Dipendenza dei dati tra le istruzioni.
- Nel primo caso la pipeline viene del tutto persa se l'istruzione corrente è un salto (branch) distante dall'istruzione successiva che è attualmente in pipeline.
- In questo caso viene persa anche la cache.
- Nel secondo caso una pipeline deve essere interrotta se l'istruzione successiva necessita, come operando, del risultato finale dell'istruzione precedente.

Esempio

- Esempio:
  - Istruzione1: A = B + C;
  - Istruzione2: D = A + 1.
- La pipeline che sta servendo l'Istruzione 2 deve interrompersi allo stadio di operand fetch, dato che A non è disponibile se non quando l'Istruzione 1 non è del tutto terminata.

Esecuzione predicativa

- **L'esecuzione predicativa**, implementata in moduli del processore denominati unità di previsione dei salti (*Dynamic Branch Prediction*), cerca di prevenire la perdita delle pipeline a causa delle istruzioni di salto.
- In questo caso le unità cercano, con vari algoritmi che usano tabelle simili a memorie cache, di capire se una istruzione di salto avverrà o meno (cosa del tutto non deterministica ma solo statistica o «storica», dato che il fatto viene computato solo durante l'esecuzione).
- Per esempio, alcuni criteri considerano sempre «presi» (taken) i salti all'indietro, tipici dei cicli (in un ciclo il salto avviene molto più spesso all'indietro).

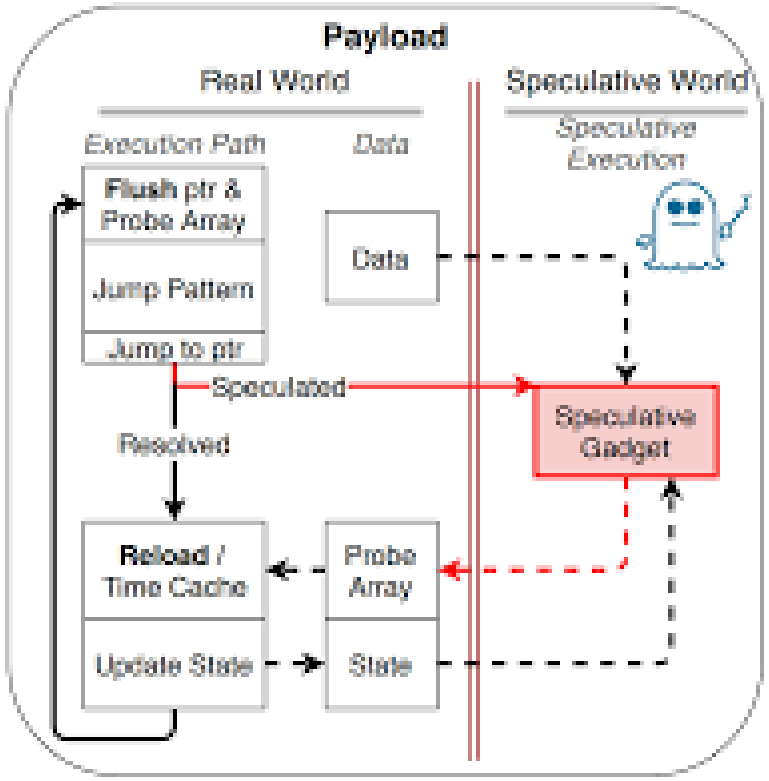


Esecuzione speculativa

- Il **problema** di questa tecnica, che in realtà è molto efficiente, si ha quando la previsione è sbagliata: le istruzioni eseguite inutilmente devono essere gettate e lo stato della macchina ripristinato.
- **L'esecuzione predicativa** è anche nota come **esecuzione speculativa**, intendendosi quella elaborazione che computa anche il codice che potrebbe non essere mai utilizzato.
- Con l'esecuzione fuori ordine (out of order execution) si cerca di prevenire lo svuotamento delle pipeline a causa di dipendenze tra le istruzioni.

Esecuzione speculativa

- Quando il processore individua una dipendenza in una pipeline, invece «di buttarla» e attendere l'operando mancante, la salta e prosegue con istruzioni "future" che, in teoria, dovrebbero essere eseguite solo dopo quella interrotta.
- In questo caso la pipeline viene quasi del tutto conservata (un solo stadio rimane bloccato, fino all'arrivo dell'operando mancante) ma, una volta risolta la dipendenza, saranno già state eseguite altre istruzioni (quelle che avevamo chiamato istruzioni «future»), con conseguente risalita delle prestazioni.



Condizioni critiche

- Naturalmente le istruzioni "future" possono essere eseguite solo se non hanno, a loro volta, dipendenze con istruzioni in corso.
- Non appena le istruzioni con dipendenze terminano, il processore continuerà l'esecuzione in ordine (in order execution).
- La condizione più critica per questa tecnica si presenta quando il processore deve essere interrotto a causa di un interrupt; se il processore si trova in fase di fuori ordine, lo stato del sistema potrebbe non essere coerente. In questi casi il processore deve ripristinare lo stato della CPU ritirando «in ordine» tutte le fasi fuori ordine.

Dipendenze RAW

- Gran parte del lavoro delle unità che gestiscono l'esecuzione fuori ordine è dovuta all'individuazione delle dipendenze nelle istruzioni.
- La dipendenza classica e più complicata (RAW, Read After Write) è proprio quella descritta nell'esempio precedente (
  - Istruzione1: A = B + C;
  - Istruzione2: D = A + 1
    - l'Istruzione2 contiene una dipendenza RAW.

Register renaming

- Per poter riordinare il giusto flusso di esecuzione dopo aver saltato e ricalcolato una istruzione con dipendenza, i processori utilizzano una serie di registri d'appoggio (interni e invisibili al programmatore) su cui memorizzare i calcoli temporanei delle istruzioni fuori ordine.
- All'atto del riordinamento, per evitare di spostare i valori dai registri interni a quelli effettivamente usati nel data path, i processori sono in grado di rinominare i registri interni nei nomi dei registri effettivi, risparmiando il tempo del trasferimento (**register renaming**).

VLIW

- Anche se non esplicitamente, tutte queste innovazioni (pipeline, super-scalarità, predicazione, esecuzione fuori ordine) cercano di implementare un modello di esecuzione parallelo molto studiato nei centri di calcolo, e denominato **VLIW** (*Very Long Instruction Word*).
- In questo modello, oltre alla parallelizzazione dell'esecuzione ottenuta in hardware, si presuppone che lo stesso codice esecutivo generato dai compilatori sia «pre-cucinato» per essere parallelizzato ottimamente dalle CPU.