Chapter 1

Seconda Parte

1.0.1 Input/Output

Un computer è collegato a innumerevoli apparecchi i quali

- gestiscono quantità di dati differenti
- velocità diverse
- formati diverse

È certo che siano più lenti della CPU e della RAM, per ottenere il massimo delle prestazioni della propria CPU è necessario che le periferiche esterne siano controllate da dei moduli comuni: **moduli di input output**.

Vari tipi di dispositivi esterni:

- Comprensibili dall'uomo
  Monitor, stampanti, ...
- Comprensibili dalla macchina
- Comunicazione
  Modem, router

I componenti di un dipositivo esterno:

- Logica di controllo
  Comunica con il modulo di I/O
- Buffer di memoria
  immaggazzina memoria, esempio (logistica container)
- Transducer
  trasforma dati fisici (analogici) in bits o viceversa

Componenti di un modulo I/O:

-

1.0.2 Tecniche di gestione input/output

I/O da programma

Attesa attiva della CPU e si occupa del trasferimento dei dati I/O memory mapped o separato

I/O guidato da interrupt

Non c'è l'attesa della CPU, essa viene interrota quando il dispositivo è pronto

Direct Memory Access

Hardware aggiuntivo
Modulo DMA:

- Data Lines
  - Data Count
  - Data Register
  - Address Register
- Address Lines
  Address Register
- Control Logic

Operazioni DMA:

- CPU comunica al controllore DMA:
  - lettura scrittura
  -
-
-

Il trasferimento dei dati DMA occupa il bus e lo sottrae alla CPU, si possono utilizzare due metodi differenti per accedere al canale:

- Una parola alla volta, cycle stealing, sottrae di tanto in tanto per un solo ciclo
- Per blocchi, burst mode, è il metodo più efficace in quanto l'occupazione del bus è un'operazione molto onerosa.

a

1.1 Rappresentazione binari numeri reali

1.2 Linguaggio Macchina

Il linguaggio macchina è costituito da:

- Insieme istruzioni eseguibili dalla CPU
- Tipologia dei dati manipolabili
- Tipi di operandi

Nessun linguaggio macchina è superiore ad un altro, ed esso dipende dall'ambito applicativo e dalle funzioni che si devono far svolgere ad esso.
Gli elementi dell'istruzione della macchina:

- Codice operativo, specifica l'operazione da eseguire
- Riferimento all'operando sorgente, specifica l'operando che rappresenta l'input dell'operazione
- Riferimento all'operando risultato, dove va messo il risultato ottenuto
- Riferimento all'istruzione successiva

Gli operandi si possono trovare:

- Memoria centrale, o (virtuale)
- Registri della CPU, ognuno ha il proprio numero identificativo
- Dato immediato con l'istruzione
- Dipsositivi di I/O

L'istruzione è una sequenza di bit divisa in campi, viene usata una rappresentazione simbolica delle configurazioni di bit. Pure gli operandi hanno la propria rappresentazione simbolica
Il formato di un'istruzione di 16 bit è:

- 4 bits, Codice operativo (opcode)
- 6 bits, Indirizzo operando, (operand reference)
- 6 bits, Indirizzo operando, (operand reference)

I vari tipi di istruzioni sono di:

- elaborazione dati, istruzioni aritmetico logiche, nei registri della CPU
- Immagazinamento dei dati in Memoria o recupero dati dalla Memoria
- Trasferimento dati (I/O)
- Controllo del flusso del programma

Gli indirizzi necessari per un'istruzione possono essere:

- un indirizzo per ogni operando (1 o 2)
- uno per il risultato
- indirizzo dell'istruzione successiva

Quindi possono essere al massimo 4, (cosa molto rara e dispendiosa), ma in genere sono 1, 2 o 3 per gli operandi/risultati
Quando si riferisce ad un solo registro vuol dire che il secondo è implicito ed è memorizzato in un registro, per esempio nell'accumulatore.
Quando non si riferisce a nesssun indirizzo vuol dire che tutti gli indirizzi sono impliciti e si sta utilizzando una pila in cui si accumulano i vari indirizzi utilizzati.

- Se si utilizzano **meno indirizzi** di conseguenza si eseguiranno delle istruzioni più elementari e corte, se si hanno più istruzioni per uni stesso programma porterà ad un tempo di esecuzione più lungo. L'archiettura RISC utilizza questa filosofia, significa infatti **Reduced Instruction Set Computer** e si basa sul velocizzare le istruzioni più frequenti.
- se si utilizzano **più indirizzi** le istruzioni diventeranno più complesse quindi impiegherà meno tempo per eseguire istruzioni più complesse ma userà più potenza per istruzioni semplici.
  L'architettura RISC si basa su questo principio **Complex Instruction Set Computer**

Cosa comporta progettare un insieme di istruzioni:

- Repertorio:
  quante e quali operazioni
- Tipo di dato:
  su quali dati
- Formato:
  lunghezza, numero indirizzi, dimensione dei campi
- Registri:
  numero di registri della CPU indirizzabili dalle istruzioni
- Indirizzamento:
  modo di specificare gli indirizzi degli operandi

Tipi degli operandi

- Indirizzi, rappresentati come interi senza segno
- Numeri
  limite al modulo
  limite alla precisione
- Caratteri (stringhe)
- Dati logici, variabili booleane per il controllo del flusso dell'esecuzione.

I numeri vengono rappresentati:

- Interi (con la virgola fissa)
- Virgola Mobile (Floating point, IEEE754)
- Decimali impaccati, nelle operazioni di I/O, non efficienti in quanto per rappresentare una cifra decimale si utilizzano 4 bit, che vuol dire solo 10 delle 16 configurazioni disponibili vengono utilizzate, si utilizza per evitare la conversione.
  es: 246 viene rappresentato come:

$$246 = \underset{200}{0010}\ \underset{+40}{0100}\ \underset{+6}{0110} \tag{1.1}$$

I caratteri vengono rappresentati in ASCII (American Standard Code for Infromation Exchange) da 7 bit, quindi si hanno in totale 128 configurazioni disponibili.
Si ha di solito 1 bit per i caratteri di controllo.
C'è inoltre una versione estesa da 8 bit in cui si possono rappresentare 256 configurazioni.
Dati Logici:

- n bit, invece che un singolo dato
- manipolare bit separatamente

Tipi di dati Intel x86

- 8 (byte), 16 (word), 32 (doppia parola), 64 (quadword), 128 (double quadword) bits
- L'indirizzamento è per unità di 8 bit (1 byte)
- Una double word da 32 bit inizia da un indirizzo divisibile per 4
- Non si ha la necessità di allineare gli indirizzi per le strutture dati in memoria
- Bisogna allineare i dati per i trasferimenti dati nel bus

Ci sono vari tipi di operazioni:

- Trasferimento Dati,
  comporta specificare: la sorgente (dove si trova il dato), la destinazione (dove andrà messo il d.), la lunghezza del dato da trasferire
- Aritmetiche,
  somma, sottrazione, moltiplicazione, divisione.
  I numeri interi hanno sempre il segno, viene utilizzata anche per numeri con la virgola mobile e inoltre possono esserci le operazioni di:
  incremento (+1), decremento(-1), negazione (inversione del segno, trovare il numero opposto), calcolo valore assoluto
- Logiche,
  sono operazioni dirette sugli specifici bit, operazioni di: AND, OR, NOT, XOR, EQUAL, possono anche essere eseguite parallelamente su tutti i bit di un registro
- Conversione
- I/O
- Sistema
- Trasferimento del controllo,
  Salto condizionato (branch), per esempio BRE R1, R2, X (salta a X se R1 equivale a R2), si ha la necessità di saltare se si deve eseguire più volte una stessa operazione come in un 'for, while' loop, il che dà spazio alla programmazione modulare.
  Salto incondizionato, salta alla prossima istruzione, non ha operandi in quanto non si devono verificare delle condizioni.
  Chiamata di procedura: una proc. è un pezzo di programma a cui si può dare un nome il che permetto di eseguirlo indicandolo con il nome. Questo permette di risparmiare di scrivere codice e di poter affidare a qualcun'altro la scrittura di questo. Dobbiamo avere due istruzioni: la chiamata e il ritorno.
  Al fine di memorizzare l'indirizzo di ritorno ci sono 3 luoghi di memorizzazione differenti:

- Registro, non funzionale quando sono presenti dei cicli ricorsivi (ovvero che si 'richiamano')
- All'inzio delle procedura chiamata, non funzionali sempre in presenza di cicli ricorsivi
- Cima della pila, ovvero si utilizza una porzione di M dove le scritture e le letture avvengono sempre in cima. In questo modo si richiamano gli indirizzi che sono stati per ultimi scritti nella pila e che si trovano appunto in cima, evitando il problema dei loop ricorsivi.

1.2.1 Linguaggio Assembly

Questo linguaggio è ad un livello più alto rispetto al linguaggio macchina ed è più comprensbile dall'uomo.
Gli indirizzi numerici (binario) vengono interpretati come indirizzi simbolici (A, B, ...,Z),
I codici operativi diventano simboli (SUB, ADD, BRE).
L'assemblatore è un programma che traduce dal linguaggio assembly al linguaggio macchina.

Big / Little Endian

I bit nella memoria vengono memorizzati in maniera differente in base all'architettura del calcolatore, ovvero:

- Big Endian,
  I bit più significativi vengono memorizzati prima negli indirizzi, da sinistra a destra
- Little Endian,
  I bit meno significativi vengono memorizzati negli ultiimi indirizzi, da destra a sinistra.

1.2.2 Modi di indirizzamento

Esistono diversi tipi per specificare l'indirizzo degli operandi:

- Immediato
- Diretto
- Indiretto
- Registro
- Registro indiretto
- Spiazzamento
- Pila

Immediato

L'operando è specificato nell'istruzione stessa, (nella parte del campo indirizzo)
VANTAGGIO: non si esegue nessun accesso in Memoria (operazione molto onerosa, in quanto implica l'occupazione del bus)
SVANTAGGIO: limitato dalla dimensione del campo indirizzo.
Se abbiamo per esempio 6 bit destinati al campo indirizzo, avremo 2^6 valori diversi.

Diretto

Il campo indirizzo contiene l'indirizzo dell'operando in Memoria.
SVANTAGGIO: un accesso in Memoria (operazione onerosa) e, spazio di indirizzamento limitato legato alla grandezza della memoria.

Indiretto

Il campo indirizzo contiene l'indirizzo di una cella di Memoria che contiene l'indirizzo dell'operando.
VANTAGGIO: Con parole di lungehzza N si possono indirizzare 2^n entità diverse, n deve essere comunque uguale o inferiore alla grandezza del campo indirizzo.
SVANTAGGIO: è che si avrà bisogno di 2 accessi alla memoria.

Registro

L'operando si trova in un registro indicato nel campo indirizzo
VANTAGGIO: pochi bit per l'indirizzamento.
SVANTAGGIO: limitato dal numero di registri disponibili dal tipo di architettura.

Registro Indiretto

Si basa sullo stesso principio dell'indirizzamento a registro, l'operando si trovan in una cellla di Memoria puntata dal contenuto del registro.
VANTAGGIO: si ha solo 1 accesso in memoria a differenza dell'indirizzamento indiretto, si ha anche un grande spazio di indirizzamento (che dipende dal numero di registri e dalla lunghezza del campo indirizzo)

Spiazzamento

È la combinazione dell'indirizzamento diretto e a registro indiretto. Il campo indirizzo è suddiviso in due parti:
il valore di base (A)
il registro che contiene l'indirizzo di un valore da sommare ad A
Una versione di questo tipo è l'indirizzamento **relativo**, in cui non si ha un registro casuale ma il Program Counter.
L'indirizzamento **registro-base**, in esso A contiene lo spiazzamento, R contiene il puntatore all'indirizzo base.
L'**indicizzazione** ha in 'A' l'indirizzo base e nel campo registro lo 'spiazzamento', per indicare gli operandi da un certo punto della memoria in poi indicheremo con 'A' questo punto di partenza e per accedere a tutti i dati successivi basta incrementare il contenuto del campo registro di 1.

1.2.3 Stack/Pila

La pila è una sequenza lineare di locazioni riservate della Memoria, c'è un puntatore (che si trova nel registro SP, stack pointer, ha come indirizzo la cima della pila). L'operando si trova nella cima della pila, si può considerare come un'evoluzione dell'indirizzamento a registro indiretto.

1.2.4 Formato delle istruzioni

Il formato delle istruzioni è come sono scritte le istruzioni date alla macchina, influsice la struttura dei campi dell'istruzione, include il codice operativo, include uno o più operandi e

generalmente si ha più di un formato per linguaggio macchina.

Lunghezza delle istruzioni

È strettamente correlata al formato delle istruzioni, è influenzata e influenza:

- La dimensione della Memoria
- L'organizzazione della Memoria
- Struttura del bus
- La complessità della CPU
- La velocità della CPU

Per sfruttare al meglio tutte le risorse di un calcolatore deve essere un giusto compromesso fra un repertorio delle istruzioni potente e la necessità di risparmiare spazio (fisicamente nella parte hardware della macchina).

Esiston due tipi di formati delle istruzioni:

- Lungehzza fissa (fixed length),
  esempio: PDP-8, PDP-10
- Formati a lunghezza variabile o ibrida (hybrid/variable length),
  esempio: PDP-11, VAX, Intel x86, questo tipo aggiunge complessità al realizzazione della macchina, ma può renderla più potente per certi utilizzi specifici.

Allocazione dei bit

Come i bit vengono allocati dipende dai diversi tipi di indrizzamnento usati da una determinata architettura, dal numero variabile degli operandi (0, 1, 2), il numero dei registri, dei banchi registri (tipi di 'buffer' di memoria). Dipende anche dall'intervallo degli indirizzi (ogni quanto viene ripetuto un indirizzo) e dalla loro granularità (se sono a byte o parola), l'indrizzamento a byte è più oneroso ma utile per la manipolazione dei caratteri.

1.2.5 Approfondimento sul funzionamento della CPU

La CPU, essendo il componente hardware più potente della macchina è la parte più importante, essa ha i compiti di:

- Prelevare le istruzioni (Instruction Fetch)
- Interpretare le istruzioni (Instruction Decode)
- Prelevare Dati (Operand Fetch)
- Elaborare Dati (Execute)
- Scrivere Dati (Write Back)

Le componenti principali della CPU sono:

- ALU, l'unità aritmetico logica, il 'cervello' della CPU

- Registri, la memoria della CPU
- Unità di Controllo, le parti che verificano il corretto funzionamento del processore

La CPU è inoltre collegata al bus di sistema per poter interagire con gli altri componenti dell'elaboratore

Registri

I registri sono uno 'spazio' di lavoro in cui la CPU può memorizzare i dati che ha elaborato senza dover interagire con la Memoria Principale, in quanto questa è un'operazione molto onerosa e che si deve cercare di limitare al più possibile.
I registri sono al vertice della gerarchia di memoria. Essi possono avere funzioni diverse determinate dall'impianto progettuale della CPU.

I tipi di Registri sono:

- Utente, vengono utilizzati dal 'programmatore' per memorizzare internamente i dati alla CPU e successivamente da elaborare.
- di Controllo e di Stato, utilizzati dall'unità di controllo per monitorare le operazioni svolte dalla CPU, sono anche utilizzati dai programmi del sistema operativo per controllare l'esecuzione dei programmi.

Con 'programmatore' ci si riferisce a:

- L'umano che programma in assembly, (che poi viene trasformato in codice macchina dall'assemblatore)
- Il compilatore che produce un codice in assembly da un programma in HLL (Linguaggio ad Alto Livello)

I registri che sono visibili all'utente sono:

- Uso generale (general purpose),
  possono essere ad uso generale o dedicati a particolari funzioni, possono memorizzare indirizzi, dati
- memorizzazione dei dati
- memorizzazione di indirizzi
- memorizzazione dei codici di condizione

Inoltre la memoria principale può essere organizzata logicamente come un insieme di 'segmenti' (spazi di indirizzamento multipli).
Un segmento al suo interno contiene locazioni di memoria indirizzabili, inoltre si può indicare all'interno della memoria fisica la 'base' del segmento (dove comincia) e la sua 'lunghezza' (quanto è lungo effettivamente)

Registri ad uso generale:
Questi registri si possono dividere in due sottocategorie:

- Effettivamente ad uso generale,
  aumentano la flessibilità e le opzioni disponibili al 'programmatore',
  aumentano le dimensioni dell'istruzione e della sua complessità, in quanto necessitano di uno spazio separato nel formato dell istruzioni che può variare la lunghezza in base al numero di registri.

- Specializzati,
  le loro istruzioni sono più piccole e veloci,
  a discapito di un'inferiore flessibilità

In genere il numero dei registri generali varia da 8 a 32, se ce ne fossero meno di 8 si avrebbe un maggior numero di accessi in memoria principale (operazione onerosa), se fossero più di 32 non si limiterebbe l'accesso alla memoria e aggiungerebbe molta complessità alla struttura della CPU.
Tuttavia l'architettura RISC arriva ad utilizzare fino a centinaia di registri.

Lunghezza dei registri:
Generalmente un registro è lungo abbastanza da poter contenere un indirizzo della memoria principale e da contenere una 'full word'

Registri per memorizzazione di Codici di Condizione:
Si tratta dell'insieme di bit individuali che possono essere letti implicitamente da un programma non possono essere impostati da un programma

Registri di Controllo e di Stato:
Sono registri che abbiamo già incontrato:

- Progrma Counter (PC), il registro ceh tiene il 'conto' delle istruzioni da svolgere
- Instruction Register (IR), il registro che contiene le istruzioni da eseguire
- Memory address register (MAR), il registro degli indirizzi di memoria
- Memory Buffer Register (MBR), un registro di buffer, ovvero una memoria temporanea interna al processore

Program Status Word:
La PSW è un insieme di bit che include codici di condizione fra cui:
il segno dell'ultimo risultato, zero, riporto, uguale, overflow, abilitazione/disabilitazione interrupt, supervisore

Modo Supervisore:
Si tratta di una modalità che permette al Sistema Operativo di utilizzare le procedure del Kernel, che agiscono su componenti critiche del sistema, ovvero l'esecuzione di istruzioni privilegiate. Essa è disponibile SOLAMENTE al sistema operativo e non all'utente o al programmatore in assembler.

Ciclo di esecuzione CPU con indirettezza:

Al ciclo di esecuzione della CPU nella fase di fetch e di execute hanno un sottociclo che è l'indirettezza.
Al fine di recuperare gli operandi indicati in un'istruzione può essere necessario accedere più volte in memoria secondo la modalità di indirzzamento indiretto.

Flusso dei dati
L'instruction Fetch generalmente si suddivide in queste operazioni:

- L'indirizzo dell'istruzione successiva 'x' è contenuta nel PC;
- L'indirizzo dell'istruzione 'x' viene spostato nel MAR;
- L'indirizzo 'x' viene emesso nel bus indirizzi;
- L'unità di controllo richiede una lettura nella memoria principale;
- Viene letto l'indirizzo nella memoria principale;
- L'indirizzo ottenuto viene inviato tramite il bus dati e viene ricevuto e copiato dal MBR;
- L'indirizzo viene spostato nell'IR
- Viene incrementato il PC con l'indirizzo dell'istruzione 'x+1'

Data Fetch
Il ciclo del data fetch comprende:

- Viene esaminato l'IR;
- Se il codice operativo (opcode) dell'istruzione richiede un indirizzamento indiretto si esegue il ciclo di indirettezza:
  - N bit più a destra del MBR vengono trasferiti nel MAR;
  - L'unità di controllo richiede la lettura dalla memoria principale;
  - Il risultato della lettura, l'indirizzo dell'operando, viene trasferito nel MBR;

Execute
Questa parte del ciclo può variare molto in base al tipo di architettura, dipende dal tipo di istruzioni che bisogna eseguire e può includere le operazioni di:

- Lettura/scrittura della Memoria;
- Input/Output;
- Trasferimento di dati fra registri e/o nei registri;
- Operazioni della ALU;

Interrupt
Esso è semplice e prevedibile si svolge come segue:

- Viene salvato il contenuto del PC, al fine di recuperare il ripristino dell'esecuzione dopo la gestione dell'interruzione;
- Il PC viene caricato con l'indirizzo della prima istruzione della routine di gestione dell'interruzione;
- Il fetch dell'istruzione viene puntato al PC;

Prefetch

È la fase di prelievo dell'istruzione e accede alla memoria principale, quest'istruzione viene di solito eseguita durante la fase di esecuzione dell'istruzione corrente (durante questa fase non si deve accedere in memoria).
Questa procedura può diventare inutile se sono in esecuzione delle istruzioni di 'jump' o 'branch' che vanno a modificare il contenuto delle istruzioni da eseguire.

1.3 Pipeline

Il concetto di **pipeline** è quello di suddividere il lavoro e:

- eseguire più attività contemporaneamente
- eseguire il lavoro in un tempo minore

Tuttavia non si può raggiungere il parallelismo totale come conseguenza della **dipendenza funzionale**, ovvero quando si necessità che l'attività precedente sia terminata per poter accedere ai dati che ha elaborato.
Nella pipeline i lavori sono affidati a degli **esecutori** che possono di diversi tipi:

- generici, ogni esecutore può eseguire un lavoro completo, ogni esecutore ha le stesse risore, ha lo stesso throughput del parallelismo totale
- specializzati, ogni esecutore svolge sempre la stessa fase, ogni esecutore è limitato alla propria fase, ogni lavoro passa da un'esecutore all'altro, questo tipo utilizza meno risorse.

Il secondo metodo è quello più diffuso in quanto:

- Ogni lavoro viene suddiviso in certo numero di fasi (i)
- Ogni fase è svolta da operatori diversi
- In ogni istante sono eseguite fasi diverse
- In ogni fase successiva ogni operatore riesegue la stessa fase.

Le fasi del ciclo esecutivo di un'istruzione sono:

- prelevare istruzione
- interpretare istruzione
- prelevare dati

- elaborare dati
- memorizzare dati

Ognuna di queste fasi è eseguita da una diversa unità funzionale della CPU, per esempio l'ALU esegui la fase di elaborazione dati.
Al fine di non perdere nessuna informazione e senza dover utilizzare la memoria principale si utilizzano dei buffer (registri temporanei), su cui si scrivono i dati utili alla fase successiva. Ne viene posto uno fra ogni fase.
Per aumentare le prestazioni bisogna:

- decomporre maggiormente il lavoro
- rendere le fasi più indipendenti fra loro e con una durata simile

Tuttavia aggiungere più fasi alla pipeline può essere una fonte di criticità perchè si creano più dipendenze e più possiblità di malfunzionamenti.

1.3.1 Pipeline hazards

Una situazione di criticità della pipeline in cui:
l'istruzione successiva non può essere eseguita nel ciclo di clock immediatamente successivo (stallo).

1. sbilanciamento delle fasi
2. problemi strutturali
3. dipendenza dai dati
4. dipendenza dal controllo

Sbilanciamento delle fasi
Questa criticità si ha in quanto non tutte le fasi richiedono lo stesso tempo di esecuzione.
per esempio: la lettura di un operando tramite registro o mediante indirizzamento indiretto.
Per questo motivo la suddivisione in fasi va misurata in base alla durate dell'istruzione più onerosa.
Inoltre non tutte le istruzioni richiedono le stesse fasi e le stesse risorse.
Il rischio che si corre con questo tipo di problema è che se un esecutore ha svolto un lavoro e deve passare un dato al successivo esecutore finchè quest'altro sta ancora svolgendo la propria mansione rischia di sovrascrivere i dati di quest'ultimo.
Come soluzione a questo tipo di problema ci sono due possibli soluzioni:

- decomporre le fasi più onerose in più sottofasi,
  ha però un elevato costo e una scarsa utilizzazione.
- duplicare gli esecutori delle fasi più onerose e farli operare in parallelo,
  per esempio due ALU, una per l'aritmetica intera e una per l'aritmetica a virgola mobile.

Problemi strutturali
Accadono quando due o più istruzioni che sono già nella pipeline richiedono di accedere ad una stessa risorsa nello stesso ciclo di clock. In questo caso gli accessi devono essere sequenziali e non paralleli.
Per risolvere questo tipo di problema si attuano queste strategie:

- Introduzione fasi non operative (nop)
- suddivisione delle memorie in modo tale che permettano gli accessi paralleli:
  per esempio una cache per le istruzioni e una per i dati.

Dipendenza dai dati
Accade quando una fase non può essere eseguita in un certo ciclo di clock perchè i dati di cui ha bisogno non sono ancora disponibili, ovvero deve attendere il termine dell'elaboraione di una fase precedente.
Avendo due istruzioni: istruzione i, istruzione i+1, si possono avere questi tipi di criticità:

- **read after write**: lettura dopo scrittura,
  i+1 legge prima che i abbia scritto,

  1. Si introducono fasi non operative (nop) quindi fasi di stallo
  2. si propagano i dati appena sono stati calcolati, (dataforwarding),
     se l'operando viene calcolato alla fine della fase *EI* lo si comunica subito doppo questa fase (e non nella fase successiva *WO*) e si riduce di uno le fasi di stallo.
  3. Si riordinano le istruzioni in maniera più efficiente dal compilatore.

- **Write After Write** : scrittura dopo scrittura,
  i+1 scrive prima che i abbia scritto
- **Write After Read** : scrittura dopo lettura,
  i+1 scrive prima che i abbia letto (caso raro in pipeline)

Dipendenza dal controllo
Accade quando entra nella pipeline un'istruzione di salto condizionato o incodizionato.
Nel caso del salto condizionato:

- Si mette in stallo la pipeline finchè non si conosce l'esito della condizione del salto
- Si individuano le istruzioni critiche e si aggiunge un'apposita logica di controllo, nel farlo si complica il compilatore e l'hardware specifico
- Si aggiungono flussi multipli, ovvero si caricano le due possibili istruzioni che sono specificate nell'istruzione di salto:
  l'istruzione *n* o l'istruzione *i+1* .
- Si esegue il prefetch dell'istruzione target del salto.
- Buffer circolare (*loop buffer*),
  ha una capienza di 256 bytes, viene indirizzato al byte, dato un indirizzo di target controllo se c'è nel buffer: 8 bit meno significativi (come indice buffer), gli altri bit più significativi si utilizzano per verificare se la destinazione del salto sta già nello sticker

  - Piccola e veloce memoria che ricorda tutte le ultime n instruzioni prelevate
  - In caso di salto si controlla se l'istruzione è già dentro il buffer
  - Utile in caso di loop, se il buffer contiene tutte le istruzioni da eseguire nel loop esse devono essere caricate una sola volta nella memoria (scenario molto frequente con i loop)
  - Accopiato al pre-fetch

- Predizione dei salti:
  Si possono avere due tipi di approcci:

  - **Statico** ,
    prevedo di saltare *sempre* ,
    prevedo di *non* saltare *mai* ,
    prevedo di saltare in base al *codice operativo*
  - **dinamico** ,
    bit take/not taken,
    tabella dell storia dei salti,
    Questo approccio cerca di migliorare la qualità della predizione del salto memorizzando la *cronologia delle istruzioni di salto condizionato* di un certo programma.
    Per questo ad **ogni** istruzione di salto condizionato associo **1 o 2 bit** per ricordae l'andamento delle ultime istruzioni. I bit vengono memorizzati in una locazione temporanea ad accesso **molto veloce** .

    * 1 bit,
      ricorda come è andata l'ultima volta quindi predice di comportarsi in maniera uguale:
      se **1** predico di saltare,
      se **0** predico di non saltare,
      se **sbaglio** predizione inverto il bit.
    * 2 bit,
      Ricorda come è andata la predizione degli ultimi due salti, per invertire la predizione si ha bisogno di due errori consecutivi

- Salto ritardato,
  finchè non si è a conoscenza dell'esito dell'istruzione di salto, al posto di rimanere in stallo si esegue un'istruzione che non dipende dal salto, il compilatore alloca un'istruzione 'opportuna' subito dopo l'istruzione di salto quindi la CPU esegue sempre PRIMA l'istruzione presente nel *branch delay slot* e dopo altera l'ordine di esecuzione delle istruzioni.

1.4 CISC e RISC

Nell'evoluzione dei calcolatori si investono molti più capitali nel reparto **software** piuttosto che in quello **hardware**. Con l'avvenire dei *linguaggi ad alto livello* (HLL è più semplice esprimere algoritmi complessi in maniera coincisa e delegano al compilatore il compito di tradurre questi in *linguaggio macchina* . Questi HLL supportano costrutti di programmazione strutturata, ovvero i diversi paradigmi.
Per poter ridurre il *gap semantico* ovvero ciò che sta fra le istruzioni in HLL a quelle in linguaggio macchina, una soluzione dei progettisti hardware è stata quella di:

- Ampliare il set delle istruzioni

- Aggiunta di diversi modi di indirizzamento
- Implementazione hardware di costrutti di linguaggi ad alto livello

Così si semplifica il lavoro del compilatore, l'esecuzione diventa più efficiente, si possono supportare HLL più complessi.
Un approccio diverso a questa soluzione può essere:

- Individuazione delle caratteristiche e e dei pattern di esecuzione delle istruzioni macchina generate dai programmi in HLL
- Semplificazione dell'architettura di base piuttosto che la sua complicazione

1.4.1 RISC

Il processo di semplificazione coinvolge:

- Le operazioni eseguite,
  semplificare le funzionalità del processore e la sua interazione con la memoria
- operandi,
  tipo e frequenza d'uso degli operandi sono alla base dell'organizzazione della memoria e dei modi di indirizzamento
- Flusso dell'esecuzione,
  organizzazione della pipeline e del controllo

Per rendere possibile questa semplificazione è necessario analizzare le istruzioni macchina generate dai programmi HLL, e analizzare le misure dinamiche che si raccolgono con l'esecuzione del programma e contano il numero di occorrenze di una certa proprietà o di una certa caratteristica.
Per esempio nei linguaggi ad alto livello si hanno molte occorenze delle istruzioni di assegnamento e delle istruzioni condizionali. Oltre alla frequenza di un'istruzione è importante tenere in conto il tempo d'esecuzione di un'istruzione.

Esito Ricerca
Il risultato della ricerca eseguita in particolare Hennessy e Patterson negli anni '80 porta alla conclusione che la strategia migliore per supportare i linguaggi ad alto livello è di:

- **NON** rendere le istruzioni macchina simili a quella di HLL
- **OTTIMIZZAZIONE** delle performance e dei pattern più usati e più time-consuming
- **ampio numero di registri** e il compilatore li utilizza in maniera ottimizzzata
- **pipeline** accuratamente progettata
- **set di istruzioni semplificato, RISC** ed efficientemente implementato

Uso dei regsitri
Un registro è una memoria interna alla CPU ad accesso molto rapido (+ veloce della cache), quindi hanno indirizzi più brevi di quelli della cache e della memoria principale, è importante che gli *operandi* utilizzati siano conservati più a lungo nei registri cosicchè da ridurre i trasferimenti memoria-registro.

- **Hardware** ,
  aumenta il numero di registri,
  più variabili sono mantenute per più tempo

- **Software** ,
  il compilatore massimizza l'uso dei regsitri,
  le variabili più usate per ogni intervallo di tempo sono allocate nei registri,
  richiede una sofisticata analisi dell'utilizzo dei programmi

Per poter utilizzare i molti registri general purpose un'idea è quella di suddividere i registri in molti piccoli gruppi (finestre) e per ogni procedura viene utilizzata un gruppo diverso.
Ogni gruppo è composto da:

- Registro parametri,
  contiene i parametri che vengono passati quando la procedura viene chiamata e contiene il valore da restituire al chiamante al termine della procedura

- Registri locali,
  memorizza il contenuto delle variabili locali

- Registri temporanei,
  Scambia parametri e il valore di ritorno con un'eventuale procedura chiamata,
  molto spesso il registro temporaneo di una procedura i è il registro dei parametri della procedura successiva $i + 1$ a livello fisico, quindi non c'è nemmeno il bisogno di trasferire i dati

Buffer Circolare

I registri sono spesso organizzati in un regitro circolare, in esso è presente il **CWP** , current window pointer, come suggerisce il nome esso indica la finestra corrente,
mentre il **SWP** , saved window pointer salva l'indrizzo della finestra in cui si deve ritornare al termine della procedura in corso.

Variabili Globali

esse sono variabili che sono accessibili da qualunque procedura e da più di esse. Il compilatore le allocca in memoria, questa cosa è però poco efficiente in quanto vengono usate spesso e l'utilizzo della memoria è un'operazione molto onerosa. La **soluzione** a questo problema è quello di usare un **gruppo di registri ad hoc** che sono disponibili a tutte le procedure.

Otimizzazione dei registri

Lo scopo di quest'operazione è quella di trovare gli operandi il più possibile nei registri e minimizzare le operazioni di *load/store* . L'implementazione *software* avviene attraverso l'ottimizzazione del compilatore e mediante l'utilizzo di **registri simbolici** .

Ogni variabile viene mappata ad un registro reale e *se* due variabili vengono utilizzate in momenti diversi possono essere mappate sullo *stesso registro* , il che può rendere il numero dei regsitri quasi infinito.
Se il numero dei registri non è sufficiente per contenere *tutte* le variabili esse vengono memorizzate nella *memoria principale* .
Il numero di registri che vengono generalmente utilizzati nell'architettura *RISC* sono fra i 32 e i 64.

1.4.2 CISC

Il tipo di architettura **CISC** *Complex Instruction Set Computer* a differenza dell'architettura RISC ha un'*ampio* insieme di istruzioni, e ha istruzioni **più** complesse al fine di *semplificare* il lavoro del compilatore e migliorarne le performance.
Tuttavia, non semplifica le istruzioni ma le rende più complesse e più simili ai linguaggi ad alto livello, inoltre con set di istruzioni più complesso è difficile *ottimizzare il codice macchina* per ridurlo e riorganizzarlo.
Le istruzioni più complesse possono essere eseguite più *velocemente* **ma** al costo di

- unità di controllo più complessa
- controllo microprogrammato necessita più spazio
- rallentamento dell'esecuzione delle istruioni più semplici, che rimangono *le più frequenti*

1.4.3 Confronto fra CISC e RISC

Istruzinoi per ciclo di clock

RISC : il ciclo esecutivo di quest'architettura dura un solo machine cycle, *se la pipeline è piena* si termina un'istruione ad **ogni** ciclo di clock
CISC : le istruzioni di questo tipo impiegano più di un ciclo di clock per essere terminate.

Memorie usate per le operazioni

RISC : sono tutte fra registri tranne che per la **load/store** che sono fra registri e memoria.
CISC : hanno anche operazioni *memory-memory* e *register-memory* . Dato che vengono spesso utilizzati *scalari locali* aumentando e/o otttimizzando i registri si ha un incremento delle prestazioni.

Modi di indirizzamento

RISC : si usano pochi e semplici modi di indirizzamento (si semplifica l'istruzione)
CISC : si utilizzano molti modi di indirizzamento e alcuni complessi

Caratteristiche architetturali

RISC: pochi e semplici formati *fissi* per le istruzioni, decodifica opcode e accesso ai registri può essere simultaneo

Verdetto

Tuttavia non si può notare quale architettura sia la *migliore* ma ciò dipende dai campi applicativi in cui queste vengnono utilizzate, è difficile capire quanta influenza viene data dal *compilatore* , (per esempio un buon compilatore CISC può rendere quest'architettura migliore nel confronto se si ha uno scarso compilatore RISC). Inoltre, la maggior parte dei confronti eseguiti sono stati fatti su prototipi semplificati e non su macchine ad uso commerciale. Al giorno d'oggi alcune architetture utilizzano aspetti caratteristici delle altre, per esempio architetture RISC con elementi CISC o viceversa.
Altre componenti cominciano a dover essere tenute in considerazione come:

- GPU, graphic processing unit
- TPU, tensore processing unit, reti neurali e processori per machine learning

1.4.4 Banco registri vs Cache

- Banco Registri,
  - contiene **tutti** gli scalari locali
  - variabili individuabili
  - variabili globabli assegnate a registri specifici
  - save/restore basato sulla profondità di annidamento
  - indirizzamento a registro
- Cache,
  - *solo* gli scalari usati di recente
  - blocchi di memoria
  - *solo* le variabili globali usate di recente
  - save/restore basato sull'algoritmo di sostituzione della cache
  - indirizzamento a memoria

Per riferirsi agli scalari locali i registri sono più veloci della cache.

1.5 MIPS

MIPS = microprocessor without interlocked pipeline stages,
Questa architettura fu progettata da Hennesy e Patterson con lo scopo di implementare una **pipeline efficiente**.
Tutte le istruzioni sono a *32 bit* , le operazioni avvengono fra *registri* , le uniche operazioni sulla memoria sono *load e store* e servono per trasferire dati fra la memoria e i regsitri. I dati che vengono caricati sui registri possono essere sottoforma di: *byte, mezze parole, parole*

Modi di indirizzamento

- Immediato
- Displacement
- Alternative
  - Indiretta registro (displacement a 0)
  - Assoluta (Registro 0 come base)

Formato Istruzioni
Esistono 3 tipi di formati nell'architettura MIPS:

- Formato R(register):
  - codop, 6bit,
    sono istruzioni aritmetico-logiche
  - rs, 5 bit,
    registro con il primo argomento
  - rt, 5 bit,
    registro con il secondo argomento
  - rd, 5 bit,
    registro destinazione che riceve il risultato
  - shamt, 5 bit,
    shift amount, nelle istruzioni di shift dice di quanti bit fare lo shift
  - funct, 6 bit,
    identifica la variante operativa
- Formato I(mmediato):
  - codop, 6 bit,
    sono le istruzioni di load/store, immediate e salto condizionato
  - rs, 5 bit,
    primo argomento
  - rt, 5 bit,
    secondo argomento
  - address/const, 16 bit,
    spiazzamento o costante
- Formato J(ump):
  - codop, 6 bit,
    salto incondizionato
  - target address, 26 bit

Ciclo esecutivo MIPS
Il ciclo esecutivo si suddivide in 5 fasi:

- IF, instruction fetch
- ID, instruction decode
- EX, execution,
  tutte le istruzioni usano l'alu fuorchè quella di salto incondizionato
- MEM, Memory access/ branch completion
- WB, write back, scrittura del risultato nei registri.

1.6 Processori Multi-Core