

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 dispositivo esterno:

- Logica di controllo
Comunica con il modulo di I/O
- Buffer di memoria
immagazzina 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 interrotta 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
- Dispositivi 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
- Immagazzinamento 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 nessun 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 un stesso programma porterà ad un tempo di esecuzione più lungo. L'architettura 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:

$$246 = \underset{200}{0010} \underset{+40}{0100} \underset{+6}{0110} \quad (1.1)$$

I caratteri vengono rappresentati in ASCII (American Standard Code for Information 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 permette 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'inizio 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 comprensibile 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 ultimi 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 lunghezza 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 trova in una cella 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, influisce 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).

Esistono due tipi di formati delle istruzioni:

- Lunghezza 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 alla 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 indirizzamento 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'indirizzamento 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:

- Program Counter (PC), il registro che 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 indiretanza:
 - 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 necessita 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 risorse, 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 possibilità 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 durata 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 possibili 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'elaborazione 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 dopo 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 incondizionato.

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 istruzioni 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)
 - Accoppiato 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 associa **1 o 2 bit** per ricordare 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 inverte 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.