

1. **Architettura di Von neuman** : prima calcolatore avente dati ed istruzioni memorizzate in un'unica memoria accessibile tramite indirizzo
2. **Cos'è un programma** : insieme di istruzioni eseguite sequenzialmente
3. **Programma Hardware** : calcolatore predisposto a ristrette operazioni, poca flessibilità e molta manualità
4. **Programma software** : astrazione dell'hardware maggiore, permette una maggior flessibilità nell'esecuzione dei vari compiti
5. **Ciclo di esecuzione cpu** : si denotano 2 principali fasi : FETCH, ovvero il prelievo di dati ed istruzioni, e l'EXECUTE, cioè l'esecuzione delle istruzioni prelevate attraverso il fetch
6. **Interrupt** : meccanismo in grado di interrompere il normale flusso di esecuzione dati potendo migliorare l'efficienza dell'elaborazione (i/o più lenti della cpu)
7. **Interrupt multipli o annidati** : possibilità di interrompere più volte la cpu, ma non solo, possibilità di interrompere l'interrupt precedente annidandone un altro al suo interno
8. **Bus** : mezzo di trasmissione composto da linee parallele , condiviso tra due o più dispositivi del calcolatore in grado di trasmettere dati, istruzioni ed indirizzi tra le varie componenti attraverso dei voltaggi specifici (denotanti informazioni binarie)
9. **Bus di sistema** : mezzo che connette cpu con l'i/o e la memoria principale avente una molteplicità di linee al suo interno denotanti tre funzioni principali di trasferimento : ISTRUZIONI , cioè tutte quelle istruzioni o dati necessarie per l'esecuzione delle operazioni richieste, INDIRIZZO , cioè gli indirizzi delle fonti o destinazioni di tali operazioni, ed infine CONTROLLO, ovvero il controllo dell'accesso alle due precedenti linee (BUS REQUEST BUS GRANT)
10. **Bus singoli e multipli** : multipli adottati in caso di congestione delle linee; tali possono essere configurati in maniera diversa a seconda della scelta architetturale eseguita
11. **Blocco** : unità minima indivisibile della memoria centrale
12. **Perché della suddivisione in blocchi** : la memoria viene suddivisa in blocchi dato che in un calcolatore si ha una cosiddetta organizzazione gerarchica che vede all'estremo memorie di capienza assai minore di quelle al loro di sotto (pensasi alla piramide), quindi sarebbe impossibile mappare tutta la memoria centrale (HARD DISK) in una cache ad esempio
13. **Organizzazione cache e politiche di rimpiazzo (...)** :
 - I. **Direct mapping** : metodo che vede la possibilità di allocare il blocco solo in una determinata posizione della cache. Per trovare i parametri seguire le istruzioni :
 - i. **PAROLA** è uguale al n° di bit necessari per raggiungere la grandezza di parola del blocco (es avendo una sola parola in blocco significherebbe $2^0 = 1$ cioè 0 bit di parola)
 - ii. **LINEA** è uguale al n° di bit necessari ad indirizzare i blocchi della cache (es avendo 16 blocchi di cache saranno necessari $2^4 = 16$ cioè 4 bit di linea)
 - iii. **TAG o ETICHETTA** sarà dato dalla differenza dei bit totali di indirizzo (o blocco) meno parola e linea

ETICHETTA	LINEA	PAROLA
-----------	-------	--------

- II. **Full associative** : tecnica che permette di allocare qualsiasi blocco in qualsiasi posizione di cache. Per trovare i parametri seguire le istruzioni:
 - i. **PAROLA** è riconducibile all'esempio precedente
 - ii. **TAG o ETICHETTA** uguale semplicemente alla differenza tra lunghezza indirizzo memoria meno il campo parola

ETICHETTA	PAROLA
-----------	--------

- III. **Set associative** : tecnica che prevede la suddivisione della cache in gruppi e per ogni gruppo un tot di linee (metodo ibrido accostante i vantaggi delle precedenti metodologie di allocazione). Per trovare i parametri si seguano le istruzioni :

*“Si consideri una cache di **4KB** con associazione a gruppi a **8** vie in congiunzione con una memoria centrale di **1MB**. Supponendo che un blocco sia di dimensione **64B** si dica a quanto ammonta la dimensione di ogni campo”*

I dati necessari allo svolgimento sono quelli evidenziati, quindi possiamo iniziare:

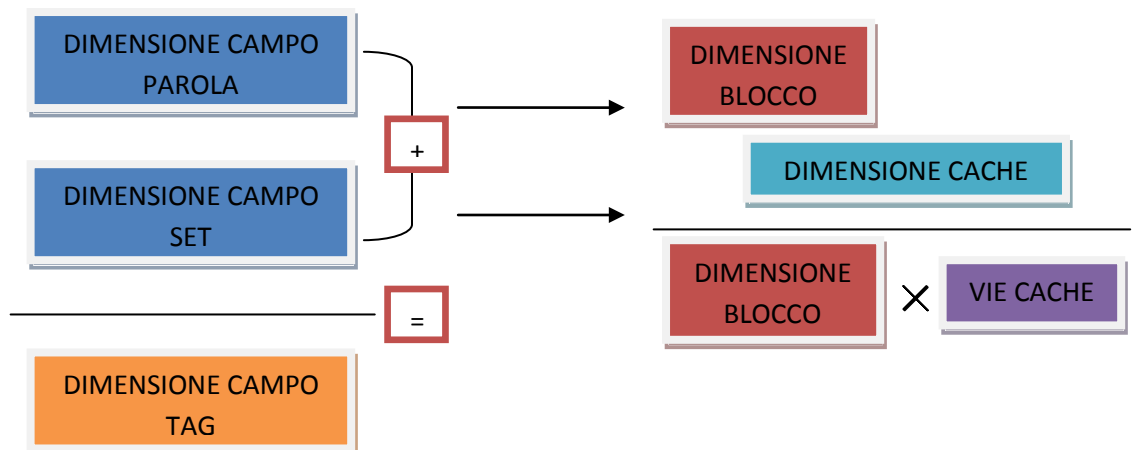
1. Troviamo la grandezza dell'indirizzo di memoria che è uguale al n° di bit necessari per poter definire la nostra memoria potendone indirizzare anche il singolo bit :

$$1\text{MB} = 2^{20} \text{ cioè l'indirizzo è composto di 20 bit}$$

2. Ora dovremo trovare le misure rispettive iniziando da quella del campo parola individuato univocamente dalla dimensione del blocco :

$$64\text{B} = 2^6 \text{ cioè il campo parola sarà composto di 6 bit}$$

3. Ora dato che la cache è di 4KB possiede 2^{12} byte, ovvero ogni linea deve contenere un blocco quindi facendo $2^{12} / 2^6 = 2^6$ linee di cache. Ora queste linee dovranno essere divise nelle vie di cache (2^3) ottenendo così 2^3 denotanti i 3 bit per il campo set
4. Di conseguenza dovremo solamente fare la differenza tra l'ammontare dell'indirizzo (20) e la somma del campo parola (6) e di quello di set (3) ottenendo 11 bit dedicati al campo tag



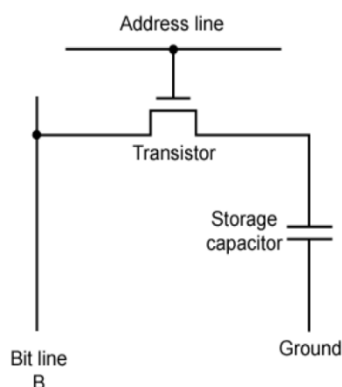
14. **Politica di rimpiazzo** : è un'insieme di politiche che possono essere usate nel momento in cui la cache satura necessita di alcuni blocchi della memoria centrale mancanti in essa :
 - I. **Fifo** : il blocco più vecchio se ne va
 - II. **LFU** : il blocco meno usato se ne va
 - III. **LRU** : il blocco meno recente se ne va, preservando la località temporale
15. **Politiche di scrittura** : avendo compiuto delle modifiche su linee di cache è necessario aggiornare il rispettivo blocco nella memoria centrale potendo evitare delle incoerenze, dunque è necessario scrivere le modifiche o nel medesimo momento in cui le modifiche vengono apportate in cache (

WRITE TROUGH aumentando però di assai il traffico sul bus) oppure scrivere in memoria in maniera differita (WRITE BACK dovendo però ricordare i blocchi subenti delle modifiche)

16. **Il problema dei miss** : i miss possono avvenire per **capacità insufficiente** della nostra cache, **conflitto** nel momento in cui vengono mappati su più gruppi, o senza possibilità di rimedio per il **primo accesso** . Dovendo rispondere con delle soluzioni ci è possibile pensare ad una **maggior dimensione di blocco** fruendo di ottima località spaziale rischiando di aumentare i conflitti dovuti al numero di blocchi inferiore, oppure godendo di una **maggior associatività** penalizzando però i tempi di ricerca del blocco in cache; oppure con le tecniche adottate negli ultimi anni : **cache multilivello, separazione cache dati ed istruzioni, ottimizzazione degli accessi mediante compilatori**

17. **Memorie** : esistono due tipologie di memorie, Dinamiche e Statiche (DRAM o SRAM)

- I. **Dinamiche** : memorie aventi i dati memorizzati all'interno di condensatori con cariche attive le quali necessitano di un refresh per il mantenimento dei dati; tali sono assai più lente delle rispettive SRAM anche se con costi inferiori dovuti alla semplicità architetturale



FUNZIONAMENTO:

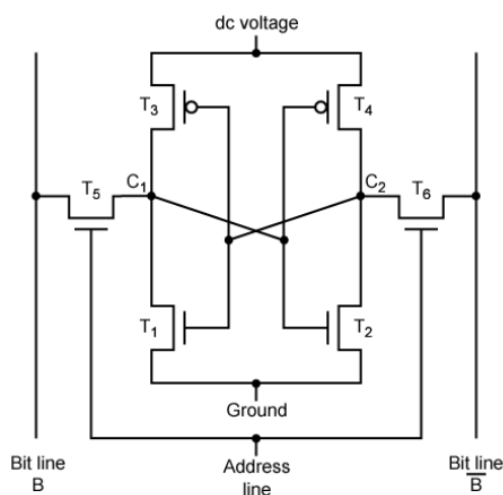
Write

- Viene applicata tensione alla bit line (tensione alta bit 1 e viceversa)
- Si applica un segnale alla address line trasferendo la carica al condensatore

Read:

- Si seleziona la linea indirizzo
- La carica si convoglia verso la address line venendo amplificata
- Si applica un refresh per ripristinare la carica

- II. **Statiche** : a differenza delle DRAM le informazioni vengono memorizzate tramite porte logiche potendo così non perdere alcun dato al decadimento dell'alimentazione. Tali memorie sono assai più costose delle precedenti, costi dovuti ad un'architettura assai ben più complessa ed alla velocità d'uso assai maggiore (vedesi ad esempio l'uso nelle cache)



FUNZIONAMENTO

Write

- Si applica il valore da scrivere alla linea B ed il complemento a B segnato

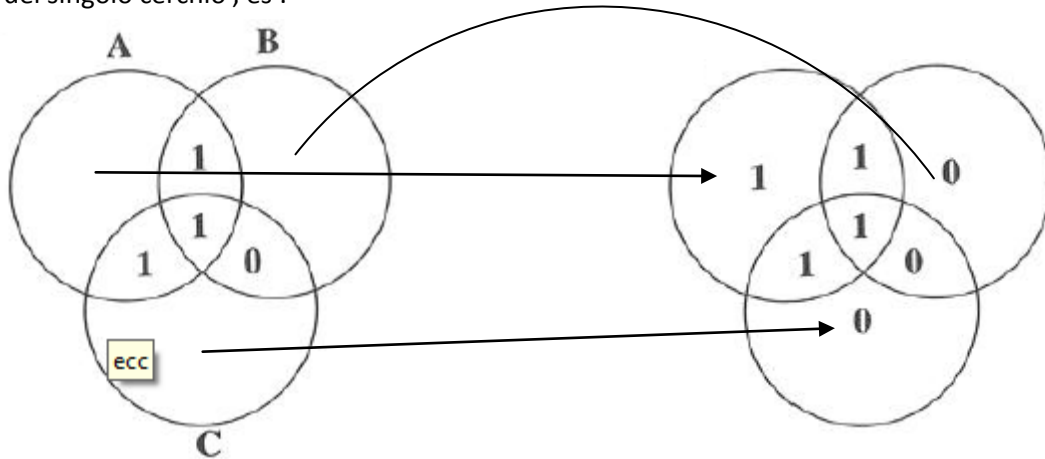
Read

- Viene letto il valore in B

18. **Rom** : (*read only memory*) memoria di sola lettura non volatile implementata in programmi di sistema (BIOS) o per qualsiasi altra architettura in cui si necessita di avere una memoria

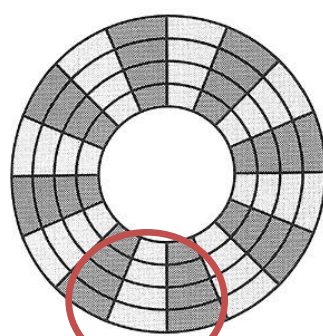
permanente non riscrivibile (per non riscrivibile s'intende non ad accesso casuale) (*PROM, EPROM, EEPROM, FLASH MEMORY*)

19. **Errori** : Gli errori in un calcolatore possono essere causati da **Guasti hardware** i quali difficilmente sono recuperabili oppure da **Errori software** non permanenti; tali comunque sia possono essere corretti attraverso più modi di cui uno dei quali è il codice di Hamming il quale vede l'iscrizione all'interno di più cerchi intersecati i valori espressi in binario. Ora per controparte nella porzione di cerchio di provenienza verrà iscritto il bit a complemento a due, calcolato sui presenti all'interno del singolo cerchio , es :

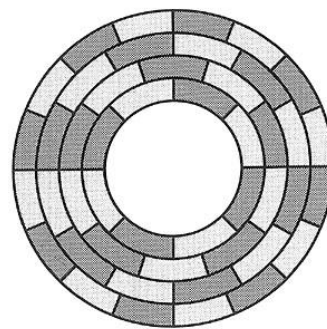


Nel caso in cui ad esempio uno dei valori all'interno delle intersezioni venga perso, basterà ricorrere al bit di parità per poter ripristinare l'informazione perduta

20. **Organizzazione memorie ottiche** : supporti magnetici rivestiti di ossido di ferro e vetro. Le informazioni al suo interno sono organizzate in tracce, ovvero cerchi concentrici separati, suddivisi al loro interno per settori, i quali possono essere posizionati nello stesso modo (**velocità angolare - costante**) oppure in maniera differente (**registrazione a più zone**), ed il tutto può a sua volta essere distribuito su più piatti (**cilindro**)



(a) Velocità angolare costante



(b) Registrazione a più zone

21. **Differenza tra le varie organizzazioni dei dati** : Nell'organizzazione a velocità angolare costante avremo uno spreco di spazio dato che essendo i settori più all'interno di dimensione inferiore e dovendo conservare però una velocità di lettura / scrittura costante, si ricorre a delle informazioni nulle via via verso gli estremi delle tracce esterne; nella seconda invece non avremo spreco di spazio.



Il settore interno è più piccolo di quello esterno, di conseguenza i settori esterni seppur più grandi potranno fruire solo di un tot di informazioni presenti nei propri settore, occupando il restante spazio con informazioni "nulle"

22. **Prestazioni del disco** : si tratta dell'analisi delle tempistiche necessarie per il trasferimento di un tot di informazioni. I tempi in questione sono :

- I. **SEEK TIME** : tempo di posizionamento della testina sulla giusta traccia
- II. **LATENZA** : la latenza denota il tempo necessario affinché il settore venga letto dal laser
- III. **ACCESS TIME** : tempo di accesso denotato dalla somma dei due precedenti
- IV. **TRANSFER TIME** : tempo necessario al trasferimento dei dati

$$T = \frac{b}{r * N}$$

b → Byte da trasferire
 N → Numero di byte per traccia
 r → Velocità rotazionale espressa per secondi
 quindi nel caso in cui vengano dati le rotazioni al minuto, esse dovranno essere divise per 60

V. **TEMPO TOTALE** : dato dalla somma di seek time più il tempo di latenza calcolato calcolato in rapporto alla velocità di rotazioni in secondi trasformato poi in millisecondi

$$L = \frac{t_l}{2 * \frac{rpm}{60}} * 1000$$

Più il tempo di trasferimento anch'esso trasferito in millisecondi costituendo da seguente formula :

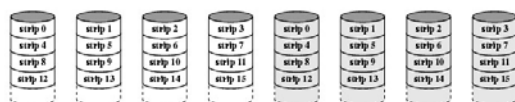
$$T_{tot} = T_{seek} + T_{latency} + T_{trasferimento}$$

23. **RAID** : Redondant array of independent disks, ovvero insieme di dischi organizzati in maniera tale da essere visti dal sistema operativo come un unico disco, avente la capacità di memorizzare informazioni per parità. Queste soluzioni organizzate per livelli hanno diverse configurazioni, numero giustificabile (7 livelli) dalle diverse esigenze degli utilizzatori :



(a) RAID 0 (non-redundant)

Nessuna ridondanza, ma dati distribuiti su più dischi tramite strisce di informazioni diminuendo il carico di lavoro per ogni disco



(b) RAID 1 (mirrored)

Informazioni replicate su altrettanti dischi → costoso



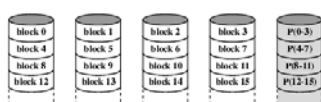
(c) RAID 2 (redundancy through Hamming code)

Ridondanza sviluppata attraverso codici correttori di Hamming → costoso → non utilizzato



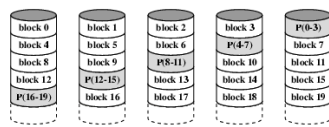
(d) RAID 3 (bit-level parity)

Unico disco di parità indipendentemente dal numero di dischi, con informazioni calcolate tramite bit di parità per ogni insieme di bit



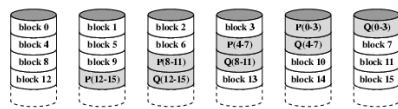
(e) RAID 4 (block-level parity)

Informazioni distribuite tramite blocchi su più dischi → soddisfacimento alti ritmi, unità di parità suddivisa in blocchi nei quali risiedono informazioni calcolate tramite parità per ogni blocco presente sulle linea attuale



(f) RAID 5 (block-level distributed parity)

Idealmente come il 4, ovvero parità calcolata su un tot di blocchi distribuiti per tutti i dischi, ed i blocchi di parità a loro volta distribuiti per i vari dischi



(g) RAID 6 (dual redundancy)

Come il 5 solo che viene adottata un'altra metodologia di calcolo di parità → ottimo per prevenire perdite di dati accidentali

24. **Esercizio prestazioni disco** : “si supponga di sapere che per trasferire 64KB di dati da un dato disco rigido occorra un tempo totale di circa 9,728571 ms (non contando attesa che il canale sia libero), sapendo che : il disco possiede 524288 tracce, ogni settore memorizza 512B, il tempo medio di posizionamento della testina è 0,8 ms e la velocità di rotazione del disco è di 4200 rpm, si calcoli il numero totale di byte che il disco può memorizzare”

I. siamo messi in questa situazione :

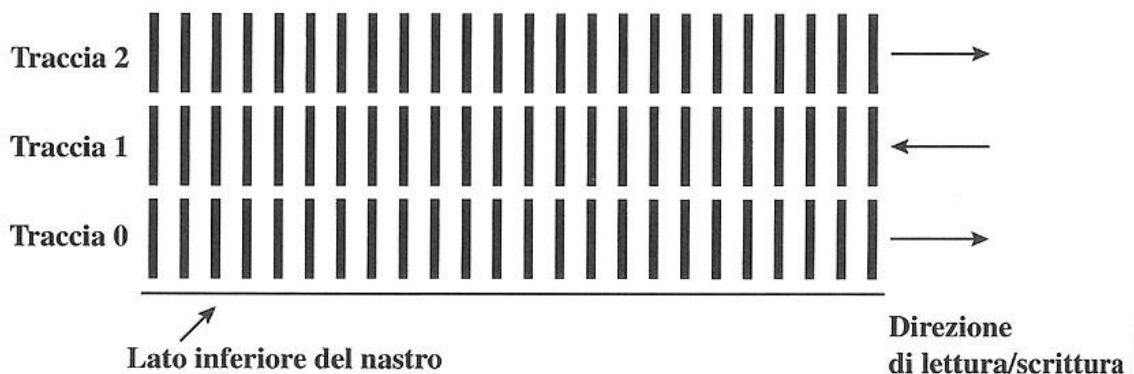
$$9,728571 = 0.8 + \frac{1000}{2 * 4200/60} + T_{\text{trasferimento}}$$

esplicitando per il tempo di trasferimento ci sarà possibile esplicitare per N, ovvero il numero di byte per traccia, conoscendo così l'ammontare complessivo di byte nel disco :

$$N = \frac{b}{rT_r} \times 1000 = 524288$$

avendo ottenuto il numero di byte per traccia basterà moltiplicare tale per il numero di tracce ottenendo il numero complessivo di byte pari a 256 GB

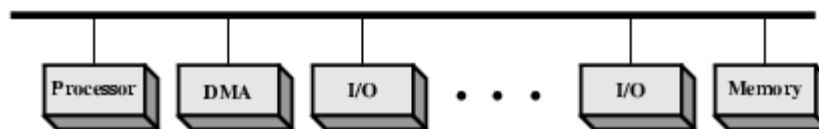
25. **Organizzazione informazioni Nastro Magnetico** : le informazioni al suo interno sono organizzate in maniera seriale che implica una velocità assai ridotta di lettura anche se in controparte gode di economici costi di produzione. Il suo uso principale è quello di ospitare backup o archiviazioni generiche



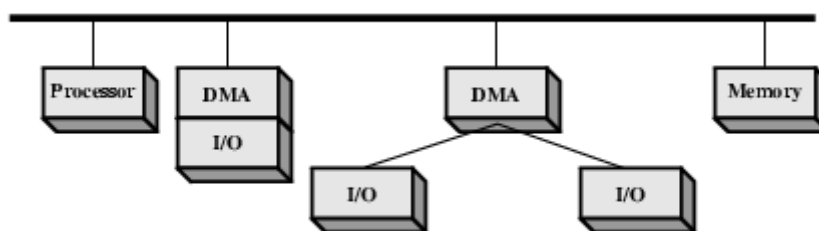
26. **Tecniche di gestione Input / Output** : tali tecniche riguardano la gestione da parte della cpu sui dispositivi di i/o (ricordiamo che tali dispositivi sono assai più lenti dei dispositivi interni) :

- I. **I/O da programma** : in questa scelta di gestione il trasferimento dati è simile all'accesso alla memoria da parte della cpu che dovrà solamente conoscere l'indirizzo univoco assegnato ad ogni dispositivo. Di conseguenza la cpu ha il controllo diretto sull'i/o potendone alterare gli stati o comandi dovendo però in controparte attendere il completamento → SPRECO CPU. Esistono però due metodologie di agire da programma :
 - i. **Memory mapped** : la memoria condivide lo spazio di indirizzamento coi dispositivi non necessitando quindi di comandi speciali dedicati all'i/o
 - ii. **Isolated** : memoria e dispositivi non condividono lo stesso spazio di indirizzamento necessitando quindi da parte della cpu dei comandi speciali e linee di selezione apposite fra I/O e memoria
- II. **I/O guidato da interrupt** : prevede l'evitare spreco di cpu evitando quindi i controlli continui sullo stato di pronto del dispositivo attraverso richieste di interruzione da parte del modulo I/O
- III. **Accesso diretto alla memoria (DMA)** : a differenza delle precedenti, questa soluzione non richiede l'intervento attivo della cpu nella gestione ma bensì delega il lavoro di gestione ad un modulo hardware ad hoc chiamato DMA posto sullo stesso bus di comunicazione. Ovviamente tale configurazione limita l'uso del bus a seconda di quale sia l'architettura adottata per il bus di sistema :

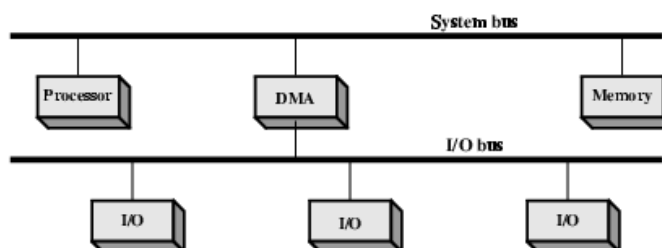
27.



Bus Singolo – necessita di 2 accessi per ogni trasferimento : (I/O → DMA , DMA → MEM)



Bus Singolo i/o annidato – necessita di 1 accessi per ogni trasferimento : (I/O → DMA(0) , DMA → MEM (1))



Bus i/o separato con collegamento diretto a DMA – necessita di 1 accessi per ogni trasferimento : (I/O → DMA(0) , DMA → MEM (1))

28. **Come vengono rappresentati i numeri all'interno di un calcolatore** : all'interno del calcolatore i numeri, cioè le informazioni vengono rappresentate secondo diverse rappresentazioni :

- I. **Modulo e segno** : il bit più a sinistra rappresenta il segno, ed il segue il numero per esteso

$$\square +18 = 00010010$$

$$\square -18 = 10010010$$

- II. **Complemento a due** : il bit a sinistra rappresenta il segno \rightarrow avendo a disposizione n bit ci è possibile rappresentare al max -2^{n-1} a $+2^{n-1} - 1$

Decodifica :

- 1- Si prende per segno il bit più a sinistra se 0 = positivo se 1 = negativo
- 2- Se positivo si legge normalmente
- 3- Se negativo si riscrivono gli stessi bit da dx a sx sino al primo 1, da li in poi si effettua il complemento a due :

$1010 = \text{negativo, riscrivendo e completando trovo } 0110 \text{ trovando } 6 \text{ ed aggiungendoci il segno precedente diventa } -6$

Benefici : a differenza della rappresentazione con modulo e segno avente due zeri, qui si ha la possibilità di aver un'unica rappresentazione dello zero rendendo così le operazioni più facili ed altrettanto facile il calcolo dell'opposto

Operazioni : per la somma basta effettuare una normale somma binaria controllando di non cadere in un overflow, per la sottrazione basterà avere i circuiti per somma e complemento, per la moltiplicazione invece bisognerà usare la rappresentazione in complemento a due per i prodotti parziali (oppure usare direttamente l'algoritmo di Booth)

- III. **Floating point** : metodologia che vede il numero in questione rappresentato tramite una stringa suddivisa in 3 campi :

Bit di segno	Esponente Polarizzato	Significando o Mantissa
--------------	-----------------------	-------------------------

Che vede il numero come $\pm 1.\text{mantissa} \times 2^{\text{esponente}}$ con l'esponente in forma **polarizzata**, ovvero ad un valore fisso viene sottratto l'esponente in questione ottenendo il vero esponente

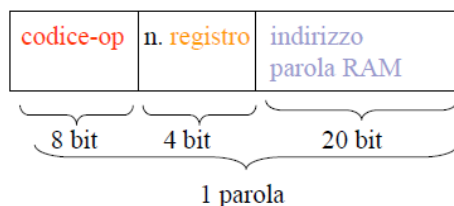
Schema di traduzione da a virgola mobile :

- i. Decimale \rightarrow Binario \rightarrow V.M.
 - a. Sottointenso il bit di segno, lo pongo ad 1 o 0 a seconda che il decimale sia rispettivamente negativo o positivo
 - b. Converto le unità in decimale così come i decimali e li riscrivo (unità dall'ultimo valore trovato al primo e per i decimali l'incontrario) separandoli da una virgola

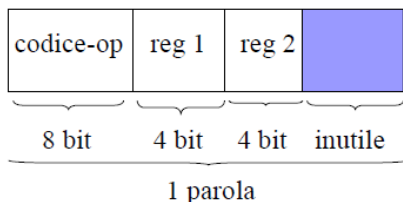
- c. Normalizzo la mantissa spostando la virgola verso sinistra tante volte quante servono per ottenere un numero nella forma 1, Ottenendo un numero X volte
 - d. Calcolo l'esponente sommandolo al BIAS, cioè $2^{(\text{bit esp} - 1)} - 1$ ottenendo un numero
 - e. Converto il numero appena trovato, trovando così l'esponente da inserire come esponente in VM
 - f. Riscrivo il bit di segno seguito dai bit esponente e mantissa
- j. V.M. → Binario → Decimale
- a. Dato il numero in VM scompongo le varie parti procedendo col calcolo dell'esponente
 - b. Calcolo l'esponente sottraendo dall'esponente dato convertito in decimale il BIAS ottenendo il numero di spostamenti da effettuare sulla mantissa de normalizzata (cioè alla quale è stata posta con virgola tuta a sinistra ed un'unità).
 - c. Fatto ciò sposto la virgola di tante posizioni quante il risultato della precedente sottrazione e calcolo le parti intere con le potenze di due.

29. **Operazioni macchina e tipo operazioni** : non sono altro che delle stringhe di istruzioni che un calcolatore può eseguire. Quelle a noi conosciute sono 6 aventi suddivisioni e parametri diversi l'una dall'altra :

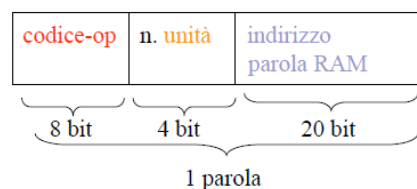
- I. **Trasferimento** : istruzione riguardante registro e ram (LOAD, STORE)



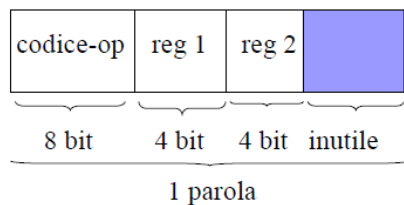
- II. **Aritmetica** : istruzioni riguardanti le unità di calcolo (es ALU) (ADD, SUB, MULT, DIV, MOD, FADD, FSUB, FMULT, FDIV)



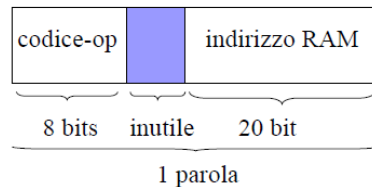
- III. **I/O** : istruzioni di READ, WRITE, STINP, STOUT



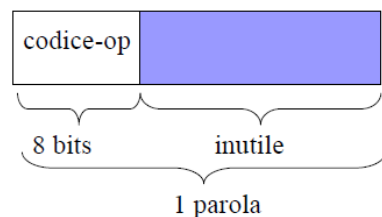
IV. **Confronto** : idealmente avvengono tra due registri (COMP , FCOMP)



V. **Salto** : salto da un'istruzione all'altra a seconda del risultato del confronto (BRLT, BRLE , BREQ, BRNE, BRGE, BRGT, BRANCH)



VI. **Stop** : stoppa l'esecuzione del programma e ve ne esce



NB : É lecito ricordare che meno indirizzi avrò a disposizione più avrò operazioni elementari con conseguente cpu meno complessa → più istruzioni da eseguire per singolo programma

30. **Tipi di operandi istruzioni** : gli operandi utilizzati nelle istruzioni precedentemente viste sono :

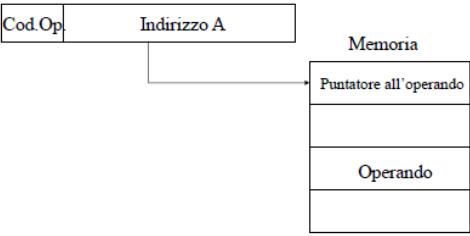
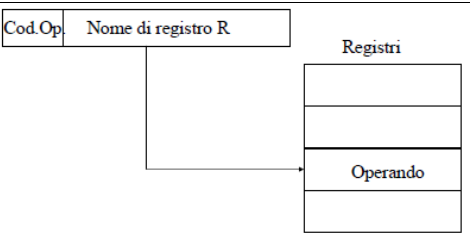
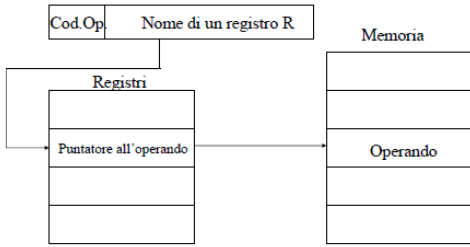
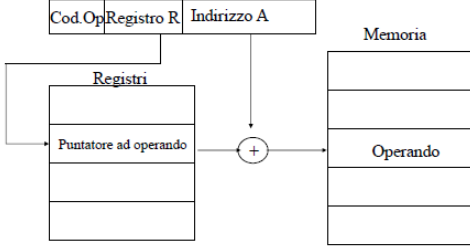
- I. Indirizzi
- II. Numeri
- III. Caratteri
- IV. Dati logici

31. **Tipo di operazioni** : le operazioni possibili da eseguire avendo i precedenti formati di istruzioni sono **trasferimento dati, aritmetiche, logiche, di conversione, i/o, di sistema, e di trasferimento del controllo** (esempio salti condizionati o incondizionati)

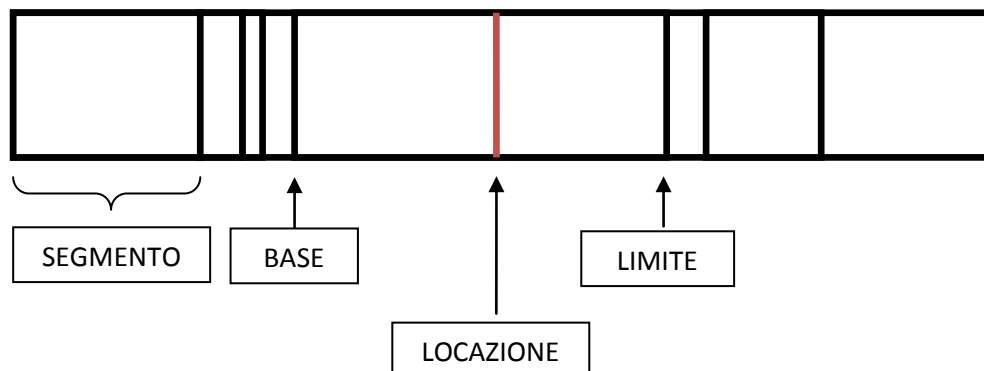
32. **Procedura** : stesura di codice identificata da un alias, potendola così richiamare quando richiesto

33. **Modi di indirizzamento** : consiste nelle modalità di esprimere un operando in un'istruzione; di essi ne esistono molti come vedesi in tabella :

IMMEDIATO		Il campo operando denota l'effettivo indirizzo dell'operando richiesto
DIRETTO		Il campo indirizzo denota l'indirizzo di memoria centrale dentro al quale trovare l'operando, implica quindi un accesso in memoria centrale per ogni operando richiesto

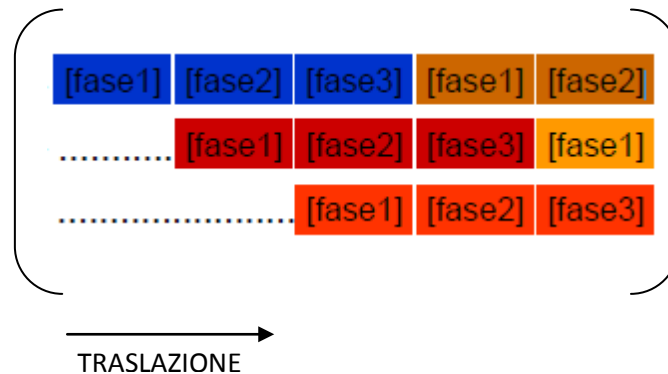
INDIRETTO	 <p>Il campo indirizzo denota l'indirizzo di memoria centrale dentro al quale trovare l'indirizzo a cui far riferimento per trovare l'operando in questione; tale implica due accessi in memoria per ogni operando richiesto</p>
REGISTRO	 <p>L'operando è semplicemente denotato dal numero del registro che lo contiene</p>
REGISTRO INDIRETTO	 <p>L'operando è reperibile tramite il campo d'istruzione che denota un registro all'interno del quale è fornito l'indirizzo di memoria centrale dove trovare l'operando implicando quindi 1 accesso per ogni operando richiesto</p>
SPIAZZAMENTO	 <p>L'operando è reperito tramite la somma di un puntatore ad operando presente nel registro fornito in uno dei due campi dell'istruzione e dall'indirizzo diretto fornito sempre all'interno dell'istruzione, componendo così l'indirizzo dell'operando in memoria centrale con conseguente accesso in memoria per ogni operando richiesto</p>

34. **Organizzazione della memoria per segmenti** : la memoria può essere organizzata logicamente come insieme di segmenti o spazi di indirizzamento riferiti dal programmatore tramite segmento e posizione della locazione all'interno del segmento (seg 4 loc 1009) necessitando quindi di conoscere la posizione di inizio del segmento e al sua estensione tramite una base e limite (base 00049 limite 2gb)

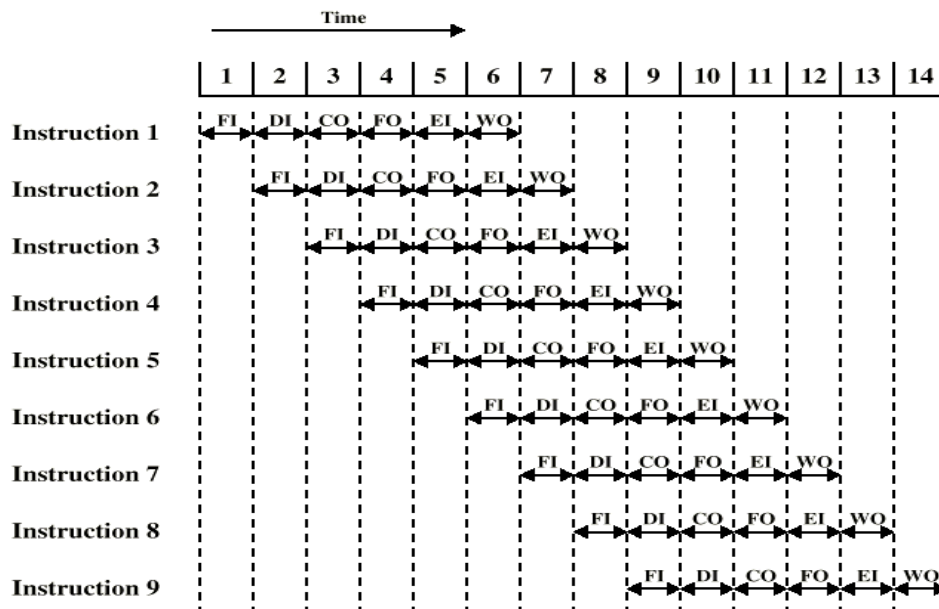


35. **Registri** : spazio di lavoro della cpu i quali vengono suddivisi per competenza: **registri utente** ovvero registri all'interno dei quali l'utente può svolgere compiti di uso **generale** o **particolare**, oppure i **registri di stato e controllo**, cioè registri usati direttamente dalla cpu per memorizzare ad esempio istruzioni prossime, indirizzi o dati a sua volta (PC, MAR, IR , MBR)

36. **Prefetch** : operazione che vede il prelievo dell'istruzione prossima in un tempo precedente , ovvero nel contempo che l'istruzione precedente viene eseguita. Tale scelta non sempre può comportare prestazioni dell'ordine $2N$, per il semplice fatto che l'operazione precedente non sempre può richiedere l'operazione immediatamente successiva (vedesi esempio i salti condizionati)
37. **Cos'è il Pipelining** : il metodo denotato con il termine del pipeline è una metodologia di esecuzione di una molteplicità di operazioni in maniera coordinata, affinché la cpu sia costantemente impegnata potendo così evitare attese inutili; tale metodologia di lavoro viene concretizzata dalla suddivisione delle varie istruzioni in più fasi, le quali vengono svolte traslandole.



Quindi è necessario suddividere le istruzioni in fasi di **fetch (FI)** nella quale si ha la semplice lettura dell'istruzione, seguita poi dalla fase di **decodifica (DI)**, ovvero il capire come è struttura l'indirizzo, seguito dal **calcolo dell'indirizzo operando (CO)**, prelievo operandi (**fetch operandi – FO**), terminando quindi con l'**esecuzione (EI)** dell'istruzione stessa e la **scrittura (WO)** in memoria.



Esempio di evoluzione di una pipeline con 9 istruzioni

38. **Fattore velocizzante pipeline a k stadi** : lo speedup della pipeline lo si ricava da 3 calcoli principali :

$$\tau = \max_i [\tau_i] + d = \tau_m + d \quad 1 \leq i \leq k$$

- τ_m = massimo ritardo di stadio (ritardo dello stadio più oneroso)
- k = numero di stadi nella pipeline
- d = ritardo di commutazione di un registro, richiesto per l'avanzamento di segnali e dati da uno stadio al successivo

conoscendo così il tempo necessario per l'esecuzione di un'operazione, dopodiché si calcola il tempo necessario per eseguire k stadi :

$$T_k = [k + (n-1)]\tau$$

ed infine ci si calcola il fattore velocizzante della pipeline :

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k + (n-1)]\tau} = \frac{nk}{[k + (n-1)]}$$

39. **Problematiche pipeline** : le principali problematiche riguardanti l'uso delle pipeline sono : lo **sbilanciamento delle fasi** ovvero i diversi tempi di durata delle fasi, i **problemi strutturali** legati all'accesso a risorse limitate e condivise, la **dipendenza dai dati** cioè quando l'istruzione eseguita in sincrono necessita dell'elaborazione di un'altra ancora in esecuzione; ed infine la **dipendenza dai controlli** quindi ad esempio quando la normale esecuzione sequenziale viene alterata (ad esempio in casi di salto condizionato) svuotando quindi la pipeline.

40. **Risoluzione problematiche pipeline** : per quanto riguarda lo

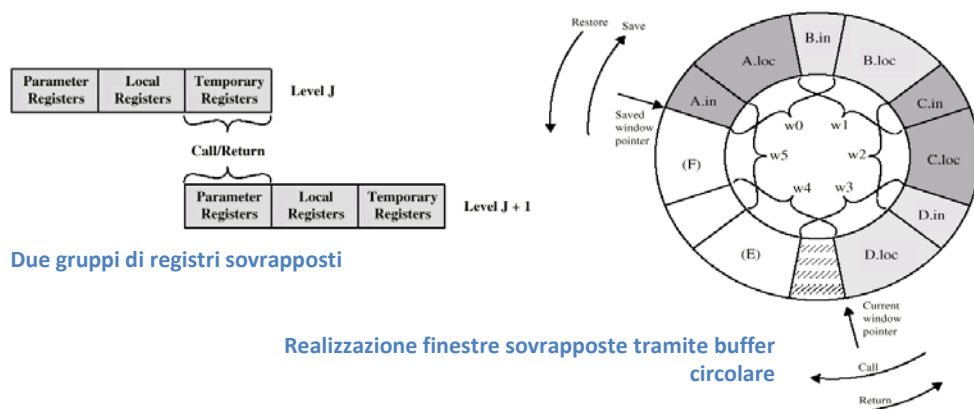
- I. Sbilanciamento delle fasi è possibile introdurre o dei tempi di attesa forzato potendo compensare le durate differenti, decomporre le fasi più durature in più sottofasi (costoso) oppure delegare l'esecuzione delle fasi più lunghe a più esecutori;
- II. Problemi strutturali : per tali problemi, provocati dal sempre aumentare delle risorse internet, creando così colli di bottiglia della memoria ecc... è necessario suddividere le memorie presenti all'interno del calcolatore potendo eseguire degli accessi paralleli (es memoria dedicata ai dati, e memoria dedicata alle istruzioni), introducendo inoltre delle nop (*not operative process*)
- III. Dipendenza dai dati : le dipendenze dai dati possono essere combattute in maniera assai rudimentale adottando delle semplici fasi nop, oppure individuando in sede il rischio del prelievo del dato (**data forwarding**) oppure riorganizzando le istruzioni della pipeline
- IV. Dipendenza dai controlli : le possibili soluzioni con pro e contro sono :
 - i. **Flussi multipli**, ovvero il replicare le parti contenenti le istruzioni successive a quella di salto e la destinazione del salto → il problema di questi sta nel conflitto all'accesso alle risorse da parte delle due pipeline (*essendoci due pipeline suddivise ed accedenti alla stessa risorsa si ha un problema di conflitto d'accesso*)

- ii. **Prelievo anticipato della destinazione** : viene semplicemente prelevata anticipatamente l'istruzione di destinazione → ovviamente essa in caso di svuotamento della pipeline non sanerebbe la situazione
- iii. **Buffer circolare** cioè una memoria di minime dimensioni contenente n dimensioni prelevate che in caso di salto vengono prelevate (*se pertinenti*), evitando così il fetch della stessa
- iv. **Predizione del salto** ovvero il prevedere se il salto sarà intrapreso oppure no, per tale è possibile avere due tipologie di approcci :
 - 1. **DINAMICI**, e cioè memorizzando la storia delle istruzioni di salto condizionato di uno specifico programma
 - 2. **STATICI**, cioè l'assumere come sempre vere o no le condizioni di salto
- v. **Salto ritardato**, metodo che vede l'allocazione in un'area opportuna di memoria (*branch delay slot*) dell'istruzione opportuna

41. **Architettura RISC** : l'architettura RISC (*reduced instruction set computer*) è un'architettura avente la proprietà chiave di godere di un set d'istruzioni alquanto semplice e limitato potendo così ottimizzare la pipeline, oltre ad adottare un elevato numero di registri ad uso generale.

(**STRATEGIE**) Le modalità con cui RISC permette tali si rifanno principalmente a due soluzioni spazianti dall'**hardware**, cioè attraverso l'uso di un elevato numero di registri → potendo quindi trattenere più variabili nei registri, al **software**, facendo allocare i registri dal calcolatore a seconda delle variabili più usate (*località temporale*)

(**HARDWARE**) La molteplicità di registri viene organizzata in finestre, cioè gruppi di registri i quali a loro volta vengono suddivisi singolarmente in tre parti per poi essere sovrapposti l'un l'altro, permettendo così un passaggio dei parametri senza lo spostamento dei dati fisici



42. **Architettura CISC** : le architetture CISC (*complex instruction set computer*) hanno avuto un'iniziativa del tutto opposta alle opposte RISC dato che esse si fondano sull'adozione di un set d'istruzioni assai ampio oltre ai svariati metodi d'indirizzamento questo per poter (**PRO**)migliorare l'efficienza dell'esecuzione e poter quindi sfruttare linguaggi ad alto livello, a differenza delle RISC le quali adottando set molto ristretti potranno supportare solo linguaggi di basso livello (*assembly*).

(**CONTRO**) Come conseguenza però si ha la necessità di hardware complesso e purtroppo in tali architetture il costo del software sarebbe maggiore di quello dell'hardware.

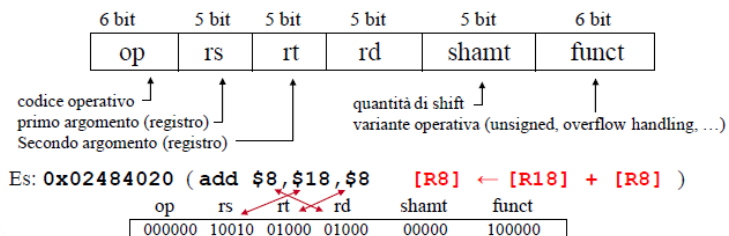
(**PERCHÉ CISC**) Il motivo che ha spinto alla creazione di tali architetture è stata semplicemente l'evoluzione dei linguaggi di programmazione ad alto livello diventati sempre più complessi al

calcolatore (per controparte maggior un linguaggio di programmazione è ad alto livello più sarà semplice per il programmazione interagire con tale)

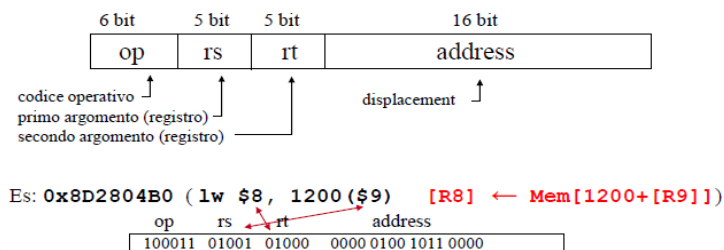
43. **MIPS** : È un esempio d'architettura RISC adottante una codifica omogenea delle istruzioni a 32 bit, istruzioni operanti su un set di 32 registri da 32 bit, accettante esclusivamente come modi di indirizzamento quello immediato, quello a spiazzamento, assoluto o di registro indiretto.

(**FORMATI**) I formati delle mips sono tre, e sono :

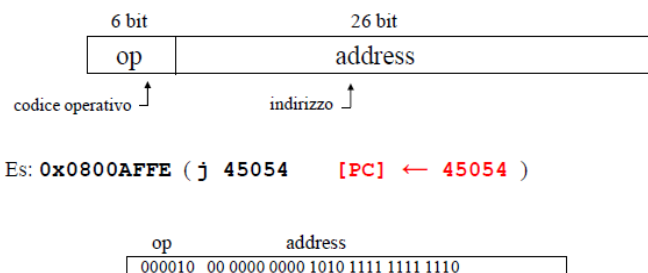
Formato R (registro)



Formato I (istruzioni load / store)



Formato J (istruzioni jump)



44. **Pipeline MIPS** : Nella stesura di una pipeline si dovrà far attenzione a due cose principali : la dipendenza dei dati nella fase ID (*instruction decoding*) che produrrà uno stallo dell'istruzione stessa, ed al data forwarding da adottare per evitare lo stallo dell'istruzione richiedente il dato ancora in uso o ancora da calcolare.
- Dobbiamo ricordare però che le fasi in questione (ovvero in cui suddividere la pipeline) sono : IF (*instruction fetch*), ID (*instruction decoding*), EX (*execute*), MEM (*eseguente* $PC \leftarrow NPC$), WB (*write back operation*)

Situazione	Esempio di codice	Azione
Nessuna dipendenza	LD \$1, 45(\$2) DADD \$5, \$6, \$7 DSUB \$8, \$6, \$7 OR \$9, \$6, \$7	Non occorre fare nulla perché non c'è dipendenza rispetto alle 3 istruzioni successive
Dipendenza che richiede uno stallo	LD \$1, 45(\$2) DADD \$5, \$1, \$7 DSUB \$8, \$6, \$7 OR \$9, \$6, \$7	Opportuni comparatori rilevano l'uso di \$1 in DADD ed evitano il rilascio di DADD
Dipendenza risolvibile con un forwarding	LD \$1, 45(\$2) DADD \$5, \$6, \$7 DSUB \$8, \$1, \$7 OR \$9, \$6, \$7	Opportuni comparatori rilevano l'uso di \$1 in DSUB e inoltrano il risultato della load alla ALU in tempo per la fase EX di DSUB
Dipendenza con accessi in ordine	LD \$1, 45(\$2) DADD \$5, \$6, \$7 DSUB \$8, \$6, \$7 OR \$9, \$1, \$7	Non occorre fare nulla perché la lettura di \$1 in OR avviene dopo la scrittura del dato caricato

Nella prima non sussiste alcuna dipendenza dato dal fatto che il registro 1 (\$1) non è richiesto da nessun'altra operazione. Nel secondo invece è possibile vedere uno stallo dato che LD rientra nella fase ID mentre DADD rientra nella sua prima fase IF, cioè il primo sta operando la decodifica dell'istruzione, mentre il secondo tenta di prelevare i valori che però sono attualmente in uso → **STALLO**.

Nel terzo esempio si ha la possibilità di operare un **forwarding** della dipendenza dato che il registro richiesto si trova nella fase EX dell'istruzione , ed arriverà per tempo nella fase EX di DSUB

	FASE 1	FASE 2	FASE 3	FASE 4	FASE 5	FASE 6	FASE 7	FASE 8
LD \$1, 45(\$2)	IF	ID	EX	MEM	WB			
DADD \$5, \$1, \$7		IF	ID	ID*	MEM	WB		
DSUB \$8, \$1, \$7			IF	ID				
OR \$9, \$6, \$7								