Importanti Architetture Teoria Esonerato

Disco magnetico

Costituito da un insieme di piatti rotanti rivestiti da una superficie magnetica, con una **testina** (bobina) per ogni faccia del piatto.
La superficie del disco è suddivisa in anelli concentrici detti **tracce**, ciascuna divisa in **settori**.
Settore: la più piccola unità che può essere trasferita.
Nei dischi più vecchi ogni traccia conteneva lo steso numero di settori, invece nei dischi più recenti per aumentare le prestazioni, si utilizzano maggiormante le tracce esterne.

RAID

Redundant Array of Inexpensive (Independent) Disks: Insieme di dischi a basso costo ma coordinati in a
Raid 0: Nessuna ridondanza dei dati, solo striping dei dati.
Striping: allocazione di blocchi logicamente sequenziali (memorizzanti p.e. lo stesso file, che quindi è suddiviso in più blocchi) su dischi diversi per aumentare le prestazioni rispetto a quelle di un singolo disco
dNon è un vero RAID perché non c'è nessuna ridondanza, è la migliore soluzione in scrittura, perché non ci sono overhead per la gestione della ridondanza, ma non in lettura.
Raid 1: Mirroring (o shadowing); Ciascun disco è completamente replicato su un disco ridondante avendo così sempre una copia.
Ottime prestazioni in lettura (leggere due file contemporamente su dischi gemelli), ma una scrittura logica richiedere due scritture fisiche.
Raid 2: Rivelazione e correzione degli errori (codice di Hamming)
Striping a livello di parola o di byte, ad ogni scrittura bisogna aggiornare i dischi di "parità"
anche per la modifica di un singolo bit di informazione (ormai in disuso)
Raid 3: Un bit di parità orizzontale ed uno verticale, resiste ad un guasto (transiente o permanente) alla volta. (Ciascuna operazione coinvolte tutti i dischi)
Raid 4: Evoluzione di Raid 3 con striping a blocchi.
La stripe nell'ultimo disco contiene i bit di parità dell'insieme di bit omologhi di tutte le altre stripe
Raid 5: Blocchi di parità distribuita, le stripe di parità sono distribuite su più dischi in modalità round-robin (circolare)
Raid 6: Ridondanza P+Q (si aumenta la distanza di Hamming)
Anziché la parità, si usa uno schema che consente di ripristinare anche un secondo guasto, la singola parità consente di recuperare un solo guasto. Overhead di memorizzazione doppio rispetto a RAID 5

DMAC = Direct Memory Access Control
Il DMAC si interpone tra la CPU e i dispositivi di I/O per scambiare dati tra di essi.
Può agire in due modi:
- **Burst**: La modalità più veloce per trasferire dati, appena ottiene il possesso dell'address bus e del data bus dalla cpu, copia l'intero blocco di memoria e lo invia, per tutto il tempo del trasferimento la cpu rimane in attesa e il DMAC e la periferica hanno il pieno controllo sul bus di sistema.
- **Bus Stealing**: I dati vengono trasferiti un byte alla volta, per poi ridare controllo alla CPU per poi mandare nuovamente richiesta ad essa per il controllo del bus e inviare nuovamente un altro byte, cosi fino a quando non finisce tutti i dati da inviare.
La velocità di trasferimento è inferiore ma impedisce alla CPU di rimanere in attesa senza fare nulla per un lungo periodo di tempo.

Dispositivi di I/O

Un dispositivo di I/O è costituito da due componenti:

– Il dispositivo fisico effettivo (disco, stampante, mouse, video, ...)

– Il device controller (o interfaccia) che gestisce tutte le operazioni che il dispositivo è in grado di svolgere.

Il device controller è collegato attraverso il bus di sistema con CPU e memoria principale

– Fornisce eventuali registri dove possono essere appoggiati i dati del trasferimento ed i comandi al dispositivo

Affidabilità e disponibilità

•**Affidabilità** - reliability: probabilità che il sistema funzioni secondo le specifiche di progetto continuamente dall'istante in cui viene attivato all'istante di "osservazione" – R(t)

•**Disponibilità** (availability) all'istante t: probabilità che il sistema funzioni secondo le specifiche di progetto quando gli si chiede un servizio – A(t)

•**Disponibilità** (a regime permanente) – availability: disponibilità quando t -> infinito

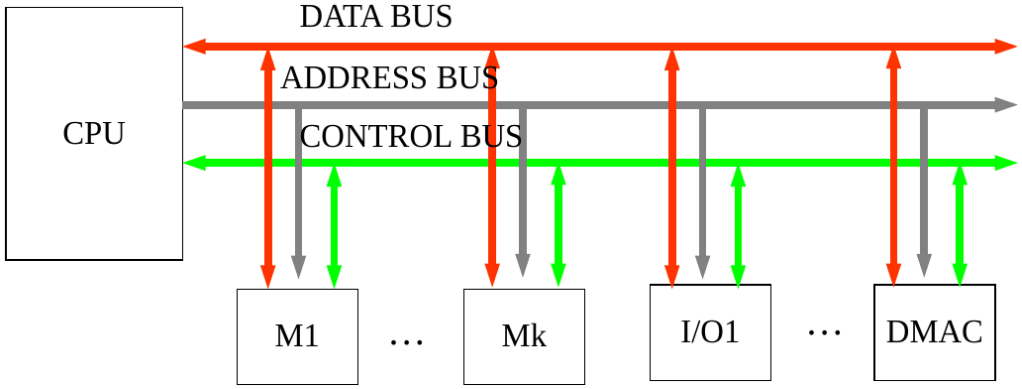
•**Tempo medio di fallimento** (mean time to failure o **MTTF**)

–Tempo medio che intercorre tra l'istante in cui il servizio è ripristinato ed il fallimento successivo

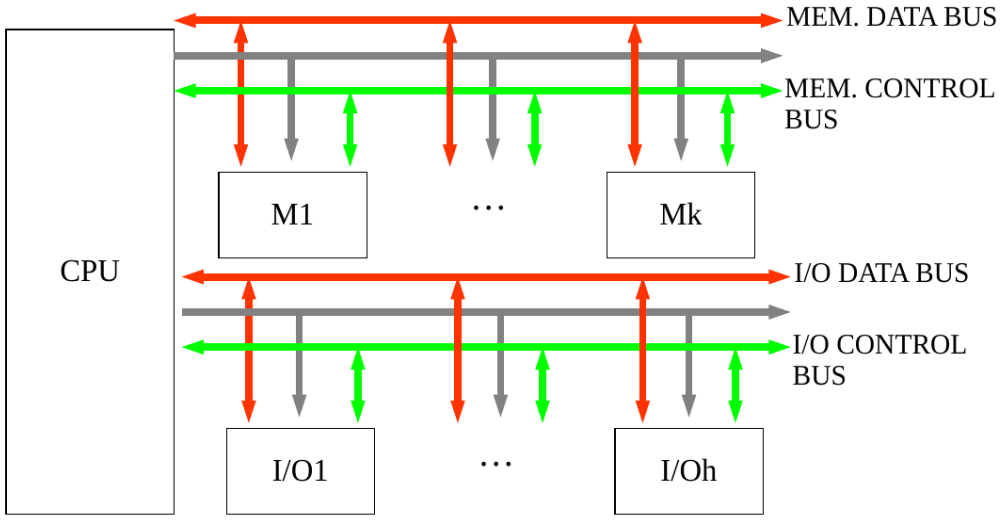
•**Tempo medio di riparazione** (mean time to repair o **MTTR**): Tempo medio necessario per ripristinare il servizio

Possibili organizzazioni dei bus

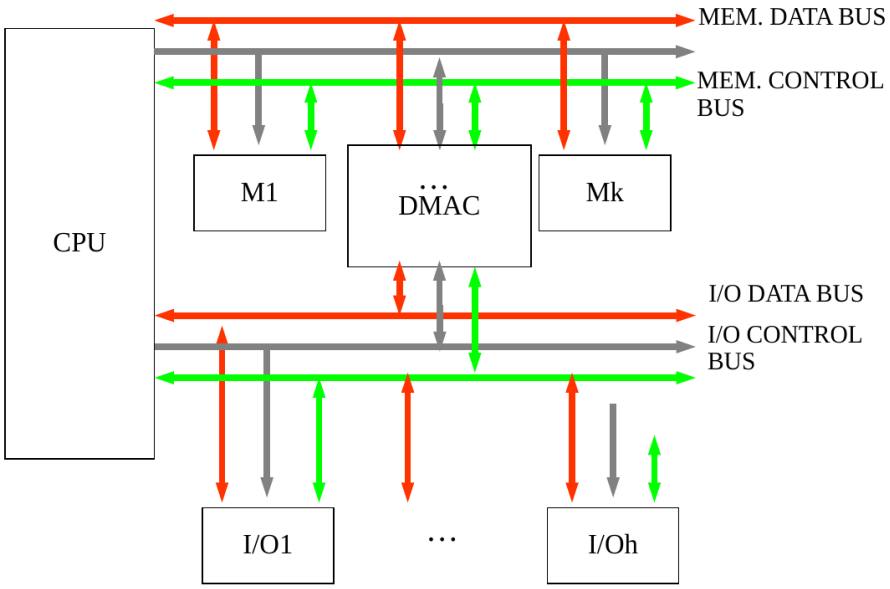
Possibile organizzazione di un calcolatore ad un bus



Possibile organizzazione di un calcolatore a due bus



Variante con DMAC



Il **Bus** rappresenta il canale di comunicazione tra le varie componenti del calcolatore
Potenziale collo di bottiglia essendo le sue prestazioni limitate dalla **Lunghezza** e dal **Numero di dispositivi connessi**
Bus composto da:

- **Linee dati (e indirizzi)**
- **Informazioni**: dati, indirizzi (anche comandi complessi)
- **Ampiezza**: numero di linee dati, possibile condividere le linee per dati e indirizzi (multiplexing)
- **Linee di controllo** per controllare l'accesso e l'uso delle linee dati ed indirizzi, richieste ed ack, tipo di informazione sulle linee dati

I bus possono essere proprietari o a specifica pubblica. Normalmente i bus processori/memoria sono proprietari perché essendo sincroni devono lavorare alle velocità proprie del processore.
I bus a specifica pubblica sono stati introdotti per abbattere i costi di realizzazione dei sistemi di elaborazione.

Tipologie di bus

Bus processore-memoria

- Lunghezza ridotta, alta velocità
- In generale proprietario
- Progettato per massimizzare la banda di trasferimento processore-memoria

Bus di I/O

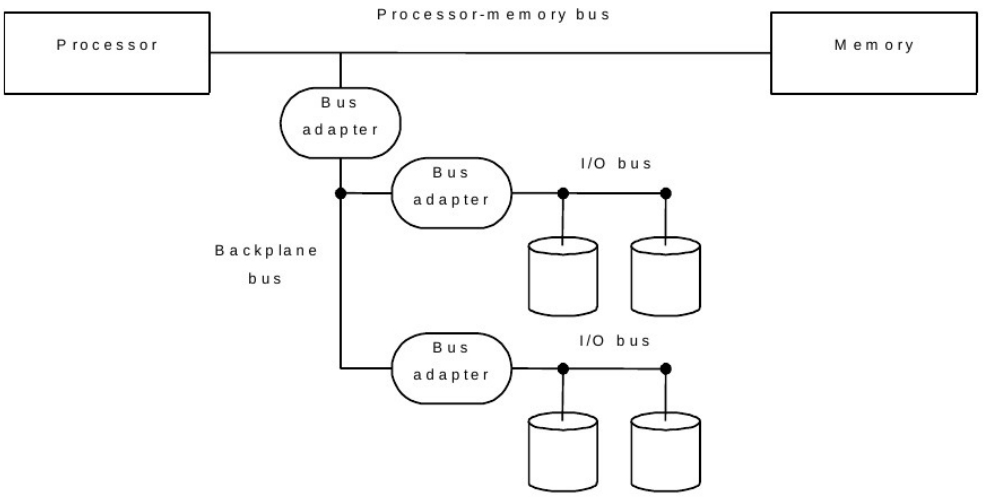
- Tipicamente di lunghezza maggiore e più lenti
- Una gran varietà di dispositivi di I/O connessi
- Standard, ad es. Firewire (IEEE 1394), USB, SCSI

Bus backplane

- Struttura di interconnessione all'interno dello chassis
- Usati spesso come struttura intermedia tra i bus di I/O ed il bus processore-memoria

Esempio

Bus backplane connesso al bus processore-memoria
Bus di I/O connessi al bus backplane



La comunicazione sul bus deve essere regolata attraverso un protocollo di comunicazione

Bus sincrono

Le linee di controllo del bus includono un segnale di sincronizzazione (clock)
Il protocollo di comunicazione è scandito dai cicli di clock
Ogni ciclo del bus per lettura/scrittura richiede più cicli di clock
Vantaggi: Molto veloce e non richiede molta logica, perché tutti gli eventi sono sincroni con il clock
Svantaggi: Ogni dispositivo deve essere sincronizzato con il clock e non può avere lunghezza elevata (problemi di clock skew: il segnale di clock arriva a dispositivi differenti in tempi diversi)

Bus asincrono

Non è dotato di clock per questo la comunicazione tra le due parti avviene tramite un **protocollo di handshaking**
Vantaggi: Può avere lunghezza elevata e connettere molti dispositivi e il tempo impiegato dalle singole operazioni sul bus è legato esclusivamente alla velocità delle parti coinvolte
Svantaggi: Più lento dei bus sincroni

Protocollo di HandShaking

Lo schema asincrono visto è incentrato sulla seguente procedura:

ReadReq viene inviato
Ack viene inviato in risposta a ReadReq
ReadReq viene resettato in risposta ad Ack
Ack viene resettato in risposta a ReadReq
DataRdy viene inviato
Ack viene inviato in risposta a DataRdy
DataRdy viene resettato in risposta ad Ack
Ack viene resettato in risposta a DataRdy

Accesso al bus

Accesso regolato tramite ruoli: **master e slave**

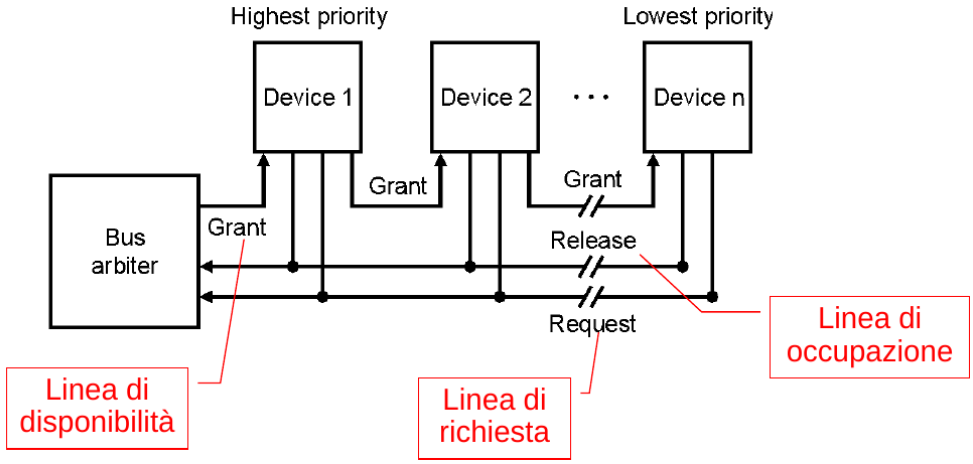
- Unità master: può iniziare attivamente una transazione di lettura o scrittura
- Il processore è sempre un master, la memoria uno slave
- Un bus può avere molteplici master

Architettura più semplice: un solo bus master (un processore), che media tutte le comunicazioni
Svantaggio: il processore deve prendere parte ad ogni transazione sul bus
Alternativa: avere più master e seguire un protocollo per coordinare le richieste dei master

Schemi di arbitraggio centralizzati:
Un controllore decide a chi assegnare il bus
Daisy chain e livelli multipli di priorità
Schemi di arbitraggio distribuiti (decentralizzati):
Nessun controllore centralizzato: i dispositivi seguono un algoritmo per il controllo d'accesso e cooperano nella condivisione del bus
Round-robin e rilevamento della collisione

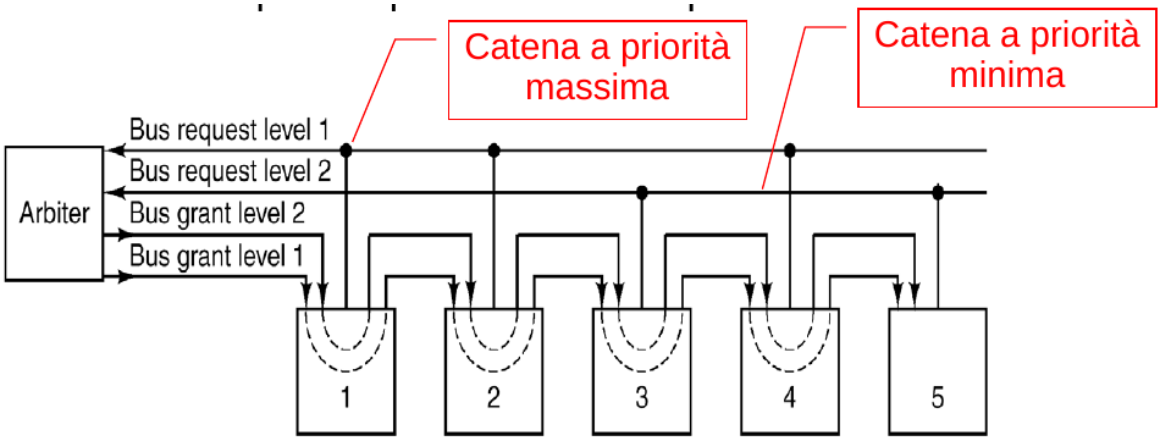
Daisy chain

Ad ogni dispositivo è assegnata una priorità
Sceglie il dispositivo che richiede l'accesso al bus e possiede priorità maggiore (più vicino all'arbitro)
Problema: non garantisce la fairness, favorisce alcuni dispositivi rispetto ad altri



Livelli multipli di priorità

Anche detto parallelo centralizzato
Diverse linee di richiesta associate a diversi livelli di priorità
In caso di conflitto favorite le catene a priorità più alta
All'interno di ciascuna catena vale la posizione (daisy chain)
In genere, se c'è un solo bus con anche la memoria, il processore ha priorità più bassa dei dispositivi di I/O



Schemi di arbitraggio distribuiti

Round-robin

Disciplina circolare

Scambio ciclico di un segnale di disponibilità tra le unità utilizzatrici del bus

Rilevamento delle collisioni

Esiste un'unica linea su cui è segnalato lo stato del bus (libero/occupato)

Più unità contemporaneamente possono occupare il bus: situazione di collisione

Occorre rilevare la collisione ed annullare la trasmissione

La trasmissione sarà ripetuta dopo un intervallo di tempo (il cui valore è generato in modo casuale)

Simile a rete Ethernet

Bus interni ed esterni

• I bus in un calcolatore si possono anche distinguere in bus interni ed esterni

• **Bus interni (o locali)**

– Confinati all'interno di una singolo chip (tra processore e cache) o tra processore e memoria

– Elevata velocità per massimizzare la banda passante

– Tecnologia proprietaria

• **Bus esterni**

– Collegano dispositivi diversi

– Maggiore lunghezza

– Velocità inferiore

Banda passante di un bus

• Un bus trasmette sequenze di dati: la rapidità con cui si passa da un dato al successivo è detta **ciclo di bus**

• Più alta è la frequenza, maggiori sono le prestazioni del bus (**bandwidth o banda passante**)

• Per ricavare la massima banda passante teorica:

max banda = frequenza * numero di linee [MB/sec]

• Le fasi di inattività e di scambio comandi riducono la banda passante reale

• I limiti fisici all'aumento della frequenza sono:

– Alte frequenze creano disturbi

– Ritardi del segnale

– Bus skew (segnali su linee diverse che viaggiano a velocità diverse)

Tecniche per aumentare la banda passante

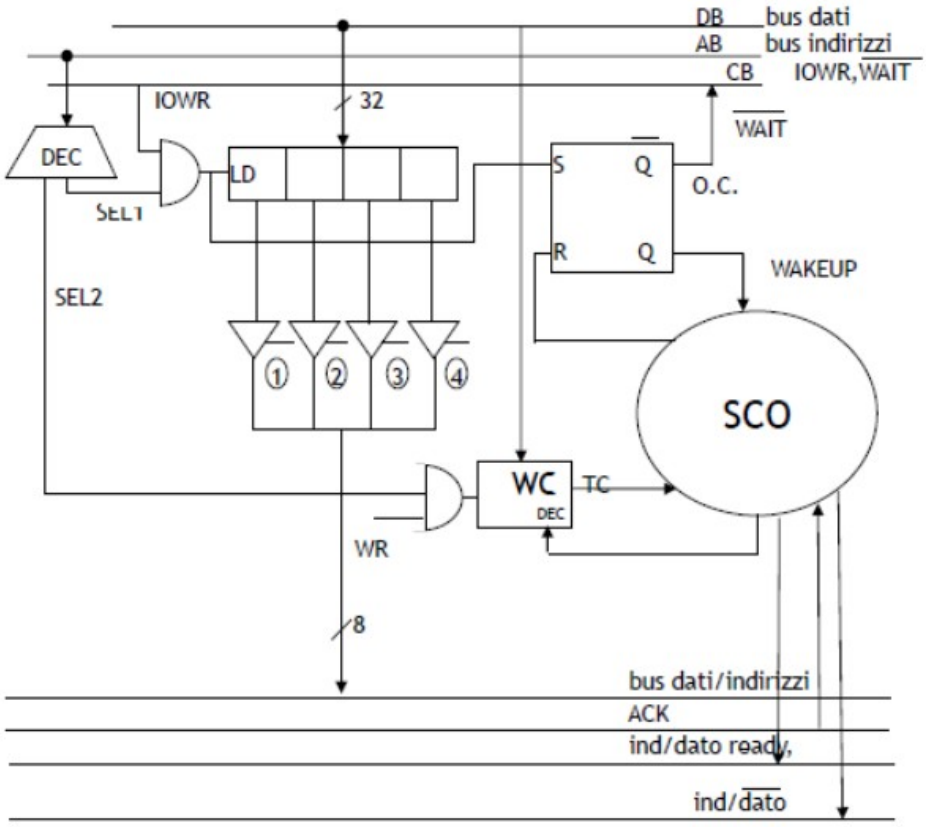
• **Parallelismo delle linee dati**: Aumento del numero di linee
• **Linee dati ed indirizzi separate**: Aumento del numero di linee
• **Trasferimento di dati a blocchi**: Riduzione del tempo di risposta
• Protocollo **split transaction:**
La transazione sul bus viene divisa in due parti: transazione di richiesta e transazione di risposta
Al termine della transazione di richiesta viene rilasciato il bus; per la transazione di risposta occorre nuovamente competere per l'accesso al bus
Vantaggio: si evitano tempi di non utilizzo del bus, sfruttando meglio la banda del bus
Svantaggio: tempi di transazione più lunghi
– Usato nei sistemi multiprocessore che condividono il bus di memoria

Bus paralleli e seriali

• **Bus paralleli**
– Più bit alla volta: i bit vengono inviati contemporaneamente su più linee
• **Bus seriali**
– Un bit alla volta: i bit vengono inviati in tempi diversi su un'unica linea
– Un bus seriale può avere un clock con frequenza superiore rispetto ad un bus parallelo
Necessità di avere a disposizione una velocità di trasferimento dei dati sempre più elevata: maggiore attenzione verso bus seriali e collegamenti punto-punto

Progetto di un adattatore di bus

Gli adattatori di bus devono consentire la trasmissione dei dati da un bus ad un altro, Come visto negli esempi precedenti ogni bus usa un proprio protocollo di comunicazione (sincrono od asincrono), le funzioni di un adattatore sono quindi simili a quelle di un interprete umano che per consentire a due persone con linguaggio differente di poter comunicare deve conoscere almeno entrambe le lingue. Quindi gli adattatori devono essere progettati in modo da poter prelevare informazioni utilizzando un protocollo da un bus e di inviarle con un altro protocollo sull'altro bus. Di seguito faremo vedere come progettare un adattatore che deve prelevare informazioni con un protocollo sincrono e trasmetterle con un protocollo asincrono. Per semplicità di presentazione si utilizzeranno protocolli già introdotti precedentemente, ma la progettazione che si presenta può essere adattata a qualunque protocollo. Nel caso specifico si progetterà un adattatore in grado di trasmettere informazioni alla velocità più lenta tra i due bus (normalmente quello asincrono). Nell'ipotesi, invece, che il protocollo sincrono, come avviene normalmente, fosse il bus con maggiore banda passante e non si volesse rallentare il produttore delle informazioni, sarebbe necessario modificare l'adattatore che dovrebbe essere in grado di acquisire informazioni alla velocità di produzione del produttore, filtrandogli la lentezza del consumatore finale. In questo caso si potrebbe utilizzare una memoria tampone in cui prima si memorizzano le informazioni e poi le si ritrasmettono al consumatore. Questa è una modalità tipica dei sistemi di comunicazione tra i calcolatori, utilizzata dai router, e denominata **store and forward**.



Lo spazio di indirizzamento dell'I/O appartiene allo stesso spazio di indirizzamento della memoria
– I registri, compresi i flip/flop, dei vari device controller sono considerati logicamente come locazioni di memoria, pur essendo fisicamente localizzati all'interno del device controller

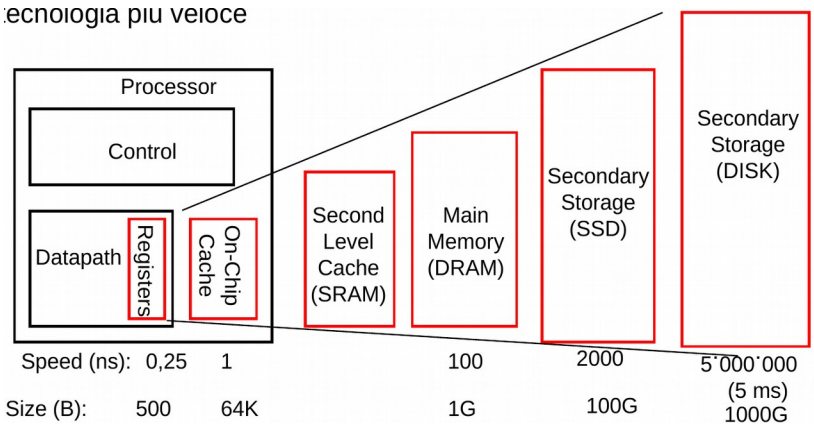
Per consentire al processore di accedere ai registri dei controller delle periferiche vengono inserite delle istruzioni specifiche nell'insieme delle istruzioni, dedicate alla gestione dell'I/O

Queste istruzioni dedicate fanno riferimento esplicitamente al dispositivo interessato all'operazione di I/O

Sul control bus oltre al segnale RD e WR è previsto un altro segnale di controllo: I/O

Se I/O = 1 si interagisce con una periferica altrimenti con la memoria

Gerarchie di memoria



Principio di località

Località temporale (nel tempo):
– Se un elemento di memoria (dato o istruzione) è stato acceduto tenderà ad essere acceduto nuovamente entro breve tempo
– Caso tipico: le istruzioni ed i dati dentro un ciclo saranno acceduti ripetutamente
Località spaziale (nello spazio):
– Se un elemento di memoria (dato o istruzione) è stato acceduto, gli elementi i cui indirizzi sono vicini tenderanno ad essere acceduti entro breve tempo
Livelli di memoria inclusivi
– Un livello superiore della gerarchia (più vicino al processore) contiene un sottoinsieme di informazioni dei livelli inferiori
– Tutte le informazioni sono memorizzate nel livello più basso
– Solo il livello massimo di cache (L1 cache) è acceduto direttamente dal processore
Migrazione delle informazioni fra livelli della gerarchia
– Le informazioni vengono di volta in volta copiate solo tra livelli adiacenti
Blocco: la minima unità di informazione che può essere trasferita tra due livelli adiacenti della gerarchia
– La dimensione del blocco influenza direttamente la larghezza (banda) del bus
Hit (successo): l'informazione richiesta è presente nel livello acceduto
Miss (fallimento): l'informazione richiesta non è presente nel livello acceduto
Processore
– Deve essere acceduto il livello inferiore della gerarchia per recuperare il blocco contenente l'informazione richiesta

Strategia di utilizzo della cache

• **Cache strutturata in linee**
– Ogni linea contiene un blocco (più parole: da 4 a 64 byte)
La prima volta che il processore richiede un dato in memoria si ha un cache miss
– Il blocco contenente il dato viene trasferito dal livello inferiore di memoria e viene copiato anche nella cache
Le volte successive, quando il processore richiede l'accesso alla memoria
– Se il dato è presente in un blocco contenuto nella cache, la richiesta ha successo ed il dato viene passato direttamente al processore
• Si verifica un **cache hit**
– Altrimenti la richiesta fallisce ed il blocco contenente il dato viene caricato anche nella cache e passato al processore
• **Si verifica un cache miss**

Le decisioni per la gerarchia di memorie
Quattro decisioni da prendere:
1. Dove si può mettere un blocco nel livello gerarchico più alto (**posizionamento del blocco** o block placement)
2. Come si trova un blocco nel livello gerarchico più alto (**identificazione del blocco** o block identification)
3. Quale blocco nel livello gerarchico più alto si deve sostituire in caso di miss (**algoritmo di sostituzione** o block replacement)
4. Come si gestiscono le scritture (**strategia di aggiornamento** o write strategy)

Posizionamento del blocco

Tre categorie di organizzazione della cache in base alla restrizioni sul posizionamento del blocco in cache

• In una sola posizione della cache:

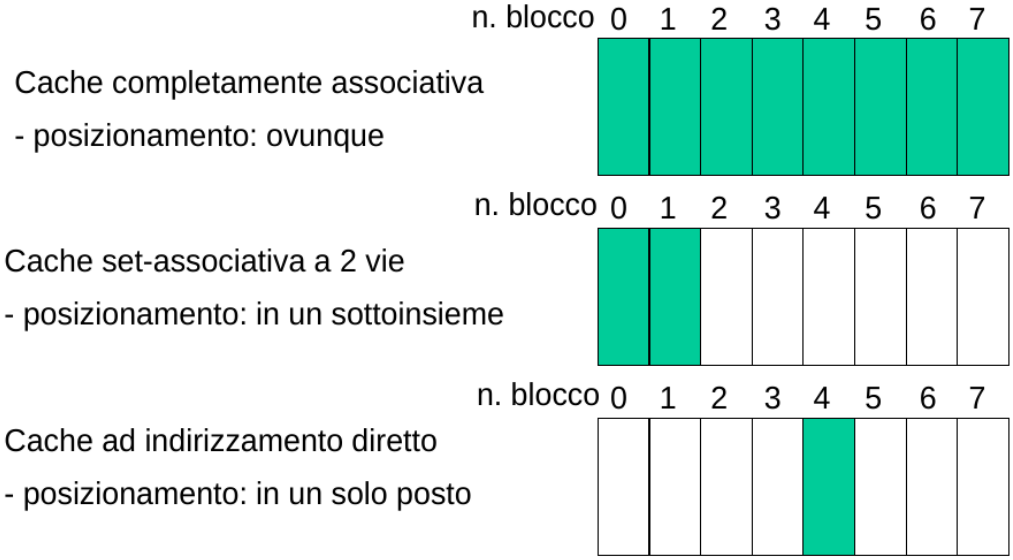
– **cache ad indirizzamento diretto** (a mappatura diretta o direct mapped cache)

• In una qualunque posizione della cache:

– **cache completamente associativa** (fully-associative cache)

• In un sottoinsieme di posizioni della cache:

– **cache set-associativa a N vie** (set-associative cache): soluzione intermedia alle due estreme



Cache ad indirizzamento diretto

• Ogni blocco nello spazio degli indirizzi trova il suo corrispondente in uno e un solo blocco in cache

N_B: numero di blocchi in cache

B_{AC}: indirizzo del blocco in cache

B_{AM}: indirizzo del blocco in memoria

$B_{AC} = B_{AM}$ modulo N_B

•L'indirizzo del blocco in cache (detto indice della cache) si ottiene usando i $\log_2(N_B)$ bit meno significativi dell'indirizzo del blocco in memoria

– La definizione si modifica opportunamente se il blocco contiene più parole (vediamo come tra breve)

•Tutti i blocchi della memoria che hanno i $\log_2(N_B)$ bit meno significativi dell'indirizzo uguali vengono "mappati" sullo stesso blocco di cache

In una cache ad indirizzamento diretto ogni linea di cache include:

– **Il bit di validità**: indica se i dati nella linea di cache sono validi

• All'avvio, tutte le linee sono non valide (compulsory miss)

– **Il tag** (etichetta): consente di individuare in modo univoco il blocco in memoria che è stato mappato nella linea di cache

– Il blocco di dati vero e proprio, formato da una o più parole (2r, se r = 2, blocco di solo 4 byte, i.e doppia parola, caso z64, o parola, caso MIPS)

Gestione di cache hit e cache miss

• In caso di **hit** (il processore continua il processamento):
– Accesso al dato dalla cache dati
– Accesso all'istruzione dalla cache istruzioni
• In caso di **miss**:
– Stallo del processore in attesa di ricevere l'elemento dalla memoria
– Invio dell'indirizzo al controller della memoria (simile ad un DMAC, identificato anche come MMU - Memory Management Unit)
– Reperimento dell'elemento dalla memoria
– Caricamento dell'elemento in cache
– Ripresa dell'esecuzione

Sostituzione nelle cache ad indirizzamento diretto

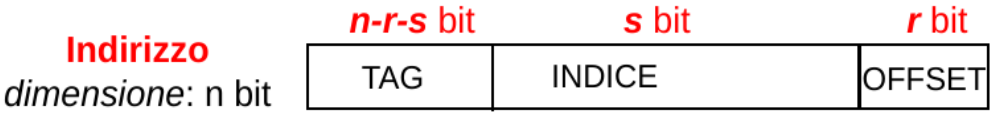
• Banale: se il blocco di memoria è mappato in una linea di cache già occupata, si elimina il contenuto precedente della linea e si rimpiazza con il nuovo blocco
– I miss sono dovuti a conflitti sull'indice di cache (conflict miss)
• La sostituzione non tiene conto della località temporale!
– Il blocco sostituito avrebbe potuto essere stato usato molto di recente
– Facile il fenomeno di thrashing
• **Vantaggi** della cache ad indirizzamento diretto
– Implementazione facile
– Richiede poca area
– E' veloce
• **Svantaggi**
– Non molto efficiente per quanto riguarda la politica di sostituzione

Cache completamente associativa

• E' l'altro estremo per il posizionamento del blocco in cache: nessuna restrizione sul posizionamento
• Ogni blocco di memoria può essere mappato in una qualsiasi linea di cache
– Non ci sono conflict miss, ma i miss sono generati soltanto dalla capacità insufficiente della cache (capacity miss)
• Il contenuto di un blocco in cache è identificato mediante l'indirizzo completo di memoria
– Il tag è costituito dall'indirizzo completo della parola
– L'accesso è indipendente dall'indice di cache
• La ricerca viene effettuata mediante confronto in parallelo dell'indirizzo cercato con tutti i tag
• Problemi
– Hardware molto complesso
– Praticamente realizzabile solo con un piccolo numero di blocchi

Cache set-associativa a N vie

• Compromesso tra soluzione ad indirizzamento diretto e completamente associativa
• La cache è organizzata come insieme di set, ognuno dei quali contiene N blocchi (N: grado di associatività)
• Anche la memoria è vista come organizzata in set
– Ogni set della memoria viene correlato ad uno e un solo set della cache con una filosofia ad indirizzamento diretto
• Ogni indirizzo di memoria corrisponde ad un unico set della cache (individuato tramite l'indice) e può essere ospitato in un blocco qualunque appartenente a quel set
– Stabilito il set, per determinare se un certo indirizzo è presente in un blocco del set è necessario confrontare in parallelo tutti i tag dei blocchi nel set
• Si attenua il problema della collisione di più blocchi sulla stessa linea di cache
• L'indirizzo di memoria ha la stessa struttura dell'indirizzo per la cache ad indirizzamento diretto
– L'indice identifica il set



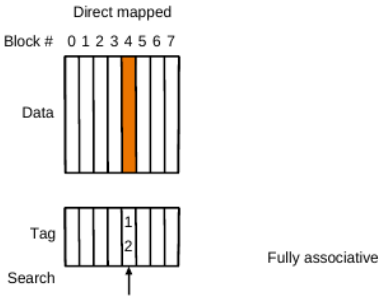
• A parità di dimensioni della cache, aumentando per esempio l'associatività di un fattore 2
– raddoppia il numero di blocchi in un set e si dimezza il numero di set
– l'indice è più corto di un bit, il tag aumenta di un bit
– il numero dei comparatori raddoppia (i confronti sono in parallelo)
Cache set-associativa a N vie: N comparatori

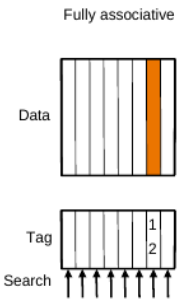
Identificazione del blocco e associatività

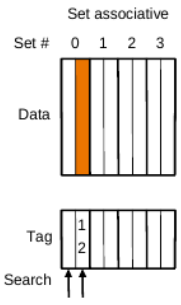
Cache a mappatura diretta
- Calcolo posizione del blocco in cache
- Verifica del tag
- Verifica del bit di validità

Cache completamente associativa
- Verifica dei tag di tutti blocchi in cache
- Verifica del bit di validità

Cache set-associativa a N vie
- Identificazione dell'insieme in cache
- Verifica di N tag dei blocchi nel set
- Verifica del bit di validità







Incremento dell’associatività

• Principale vantaggio
– Diminuzione del miss rate
• Principali svantaggi
– Maggior costo implementativo
– Incremento dell’hit time
• La scelta tra cache ad indirizzamento diretto, set-associativa e completamente associativa dipende dal costo dell’associatività rispetto alla riduzione del miss rate

Sostituzione nelle cache completamente associative e set-associative

• Quale blocco sostituire in caso di miss (capacity miss)?
– In caso di cache completamente associativa: ogni blocco è un potenziale candidato per la sostituzione
– In caso di cache set-associativa a N vie: bisogna scegliere tra gli N blocchi del set
• Politica di sostituzione **Random**
– Scelta casuale
• Politica di sostituzione **Least Recently Used** (LRU)
– Sfruttando la località temporale, il blocco sostituito è quello che non si utilizza da più tempo
– Ad ogni blocco si associa un contatore all’indietro, che viene portato al valore massimo in caso di accesso e decrementato di 1 ogni volta che si accede ad un altro blocco
• Politica di sostituzione **First In First Out** (FIFO)
– Si approssima la strategia LRU selezionando il blocco più vecchio anziché quello non usato da più tempo

Problema della strategia di scrittura

• Le scritture sono molto meno frequenti delle letture
• Le prestazioni sono migliori per le letture
– La lettura può iniziare non appena è disponibile l’indirizzo del blocco, prima che sia completata la verifica del tag
– La scrittura deve aspettare la verifica del tag
• In conseguenza di un’operazione di scrittura effettuata su un blocco presente in cache, i contenuti di quest’ultima saranno diversi da quelli della memoria di livello inferiore
– Occorre definire una strategia per la gestione delle scritture

Strategia write-through

• Scrittura immediata: il dato viene scritto simultaneamente sia nel blocco della cache sia nel blocco contenuto nella memoria di livello inferiore
• Vantaggi
– E’ la soluzione più semplice da implementare
– Si mantiene la coerenza delle informazioni nella gerarchia di memorie
• Svantaggi
– Le operazioni di scrittura vengono effettuate alla velocità della memoria di livello inferiore → diminuiscono le prestazioni
– Aumenta il traffico sul bus di sistema

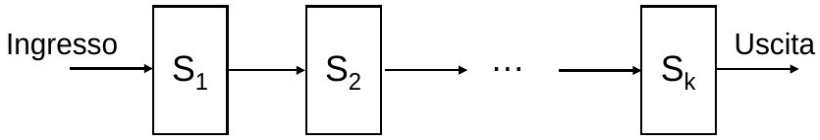
Strategia write-back

• **Scrittura differita**: i dati sono scritti solo nel blocco presente in cache; il blocco modificato viene trascritto nella memoria di livello inferiore solo quando viene sostituito
– Subito dopo la scrittura, cache e memoria di livello inferiore sono inconsistenti (mancanza di coerenza)
– Il blocco in cache può essere in due stati (dirty bit):
• clean: non modificato
• dirty: modificato

• Vantaggi
– Le scritture avvengono alla velocità della cache
– Scritture successive sullo stesso blocco alterano solo la cache e richiedono una sola scrittura nel livello inferiore di memoria
• Svantaggi
– Ogni sostituzione del blocco (ad es. dovuto a read miss) può provocare un trasferimento in memoria più lungo (prima scrittura del blocco modificato e poi lettura del nuovo blocco)

Architetture RISC

• Il **pipelining** è una tecnica per migliorare le prestazioni del processore basata sulla sovrapposizione dell'esecuzione di più istruzioni appartenenti ad un flusso di esecuzione sequenziale.
• Il lavoro svolto da un processore con pipelining per eseguire un'istruzione è diviso in passi (stadi della pipeline), che richiedono una frazione del tempo necessario all'esecuzione dell'intera istruzione
• Gli stadi sono connessi in maniera seriale per formare la pipeline; le istruzioni:
– entrano da un'estremità della pipeline
– vengono elaborate dai vari stadi secondo l'ordine previsto
– escono dall'altra estremità della pipeline



Istruzioni logiche/aritmetiche (istruzioni di tipo **L/A)**

Istruzione	**Sintassi**	**Semantica**
Somma	add regsorg1, regsorg2, regdest	(regdest) = (regsorg1) + (regsorg2)
Sottrazione	sub regsorg1, regsorg2, regdest	(regdest) = (regsorg1) - (regsorg2)
Prod. Logico	and regsorg1, regsorg2, regdest	(regdest) = (regsorg1) and (regsorg2)
Som. Logica	or regsorg1, regsorg2, regdest	(regdest) = (regsorg1) or (regsorg2)
Neg. Logica	not regsorg1, regdest	(regdest) = not (regsorg1)

Formato istruzioni

31-26	25-21	20-16	15-11	10-0
opcode	regsorg1	regsorg2	regdestL/A	non utilizzati

Esempi di codice

add 2, 3, 1 -- somma il contenuto dei registri 2 e 3, il risultato mettilo nel registro 1

000001	00010	00011	00001	----------

sub 5, 6, 4 -- sottrai il contenuto del registro 5 con quello di 6, il risultato mettilo nel registro 4

000010	00101	00110	00100	----------

and 2, 3, 1 -- fai l'and tra il contenuto dei registri 2 e 3, il risultato mettilo nel registro 1

000011	00010	00011	00001	----------

Istruzioni caricamento/memorizzazione (istruzioni di tipo **C/M)**

Istruzione	**Sintassi**	**Semantica**
Caricamento di parola	load regdest, offset(regbase)	(regdest) = memoria[offset(regbase)]
Memorizzazione di parola	store regsorgM, offset(regbase)	memoria[offset+(regbase)] = (regsorg)

Formato istruzioni load ed esempio di codice

31-26	25-21	20-16	15-0
opcode	regbase	regdestC	Offset

Load 1, 32(3) --
trasferisci il contenuto della locazione di memoria il cui indirizzo è dato dalla somma di 32 con il contenuto del registro.

000110	00101	00001	0000000000100000

Formato istruzioni store ed esempio di codice

31-26	25-21	20-16	15-0
opcode	regbase	regsorgM	Offset

Store 7, 16(4) --
trasferisci il contenuto del registro 7 nella locazione di memoria il cui indirizzo è dato dalla somma di 16 con il contenuto del registro 4.

000111	00100	00111	0000000000010000

Istruzioni di salto condizionato (istruzioni di tipo S)

Istruzione	**Sintassi**	**Semantica**
salta se flag X = =1 (dove X può essere uno dei flag del registro di stato)	jumpX indirizzo	se flag X = =1 allora PC=PC+S+indirizzo (dove S è un intero che dipende da come è organizzata la memoria, per esempio in un processore a 32 bit con il banco di memoria della cache costituita con moduli di memoria di 8 bit il suo valore è pari a 4)

Formato ed esempio di codice

31-26	25-23	22-0
opcode	flag	indirizzo

jumpC 1026 --
se il bit di CARRY = = 1 allora metti il PC ad un valore pari a (PC)+S+1026.

001000	001	00000000000001000000010

Istruzione non operativa (NOP)

Istruzione	Sintassi	Semantica
Nulla	Nop	Non modificare nulla

Formato e esempio di codice

nop -- non fare nulla

000000	0000	0000	0000000000000000

Considerazioni sul formato

• Il campo codice-operativo è sempre contenuto nei bit 31-26;
• I due registri **sorgente** delle operazioni logiche/aritmetiche sono specificati, rispettivamente, nei bit 25-21 e 20-16, nella stessa posizione troviamo specificato anche il registro base e registro sorgente dell'istruzione store, mentre il registro base dell'istruzione load è specificato nei bit 25-21; questa disposizione semplificherà la progettazione del data path, in quanto, come si vedrà, sarà possibile accedere al banco dei registri per prelevare gli operandi indipendentemente dal tipo di istruzione;
• Il registro **destinazione** delle operazioni logiche/aritmetiche è specificato nei bit 15-11, mentre quello dell'istruzione load nei bit 20-16; questo disallineamento purtroppo complicherà un po' l'architettura del data path;
• L'offset sia nelle istruzioni **load** che **store** è memorizzato nei bit 15-0.

Fasi esecuzione istruzioni logiche/aritmetiche

• Prelevare l'istruzione dalla memoria ed incrementare di S il PC (**FETCH**);
• Decodificare l'istruzione e leggere il contenuto dei registri sorgente (**DECODE**);
• Effettuare l'operazione logica o aritmetica specificata (**EXECUTE**);
• Memorizzare nel registro destinazione il risultato dell'operazione (**WRITE BACK**).

Fasi esecuzione istruzione di load

• Prelevare l'istruzione dalla memoria ed incrementare di S il PC (**FETCH**);
• Decodificare l'istruzione e leggere il contenuto del registro base (**DECODE**);
• Calcolare l'indirizzo della locazione di memoria da cui prelevare il dato (**EXECUTE**);
• Leggere il contenuto della locazione di memoria in cui è memorizzato il dato (**MEMORY**);
• Memorizzare nel registro destinazione ciò che è stato letto dalla memoria (**WRITE BACK**).

Fasi esecuzione istruzione di store

• Prelevare l'istruzione dalla memoria ed incrementare di S il PC (**FETCH**);
• Decodificare l'istruzione e leggere il contenuto del registro sorgente del dato da memorizzare e il contenuto del registro base (**DECODE**);
• Calcolare l'indirizzo della locazione di memoria in cui scrivere il dato letto dal registro sorgente (**MEMORY**);
• Memorizzare nella locazione di memoria ciò che è stato letto dal registro sorgente (**WRITE BACK**).

Fasi di esecuzione di una istruzione di salto condizionato

• Prelevare l'istruzione dalla memoria ed incrementare di S il PC (**FETCH**);
• Decodificare l'istruzione (**DECODE**);

• Calcolare l'indirizzo della locazione di memoria da cui prelevare l'istruzione successiva nel caso in cui il flag di stato selezionato è pari ad 1 (**EXECUTE**);
• Dipendendo dal valore del flag selezionato aggiornare il PC o lasciarlo invariato.

Fasi di esecuzione di una NOP

• Prelevare l'istruzione dalla memoria ed incrementare di S il PC (**FETCH**);
• Decodificare l'istruzione (**DECODE**).

Esecuzione delle istruzioni nel processore con pipeline

F Instruction Fetch	D Instruction Decode	E Execute	M Memory access	WB Write-Back

• Istruzioni logico-aritmetiche

F Prel. istr. e incr. PC	D Lettura reg. sorgente	E Op. ALU su dati letti		WB Scrittura reg. dest.

• Istruzioni di load

I Prel. istr. e incr. PC	D Lettura reg. base	E Somma	M Prelievo dato da M	WB Scrittura reg. dest.

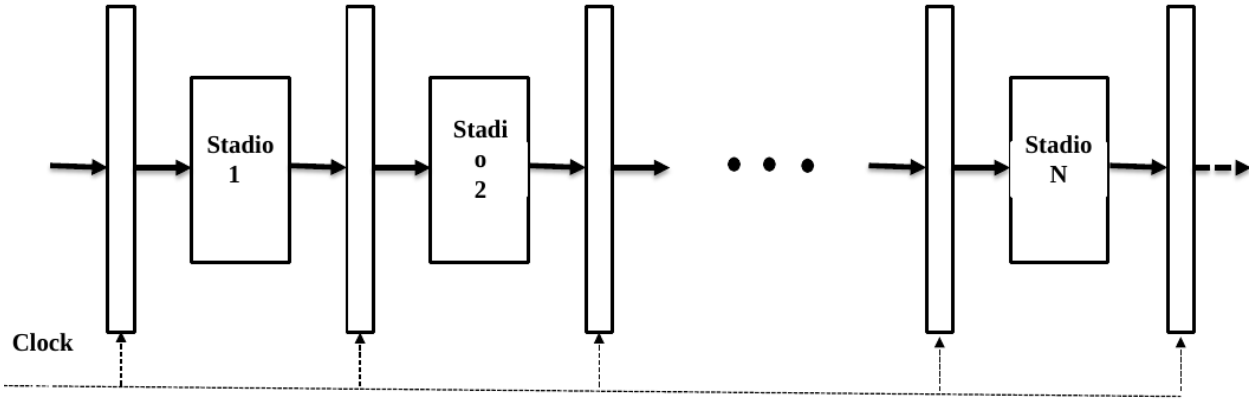
• Istruzioni di store

F Prel. istr. e incr. PC	D Lettura reg. base e reg sorg M	E Somma	M Scrittura dato in M	

• Istruzioni di jump

F Prel. istr. e incr. PC	D Decodifica	E Somma	M Scrittura PC	

Schema di principio delle architetture pipeline

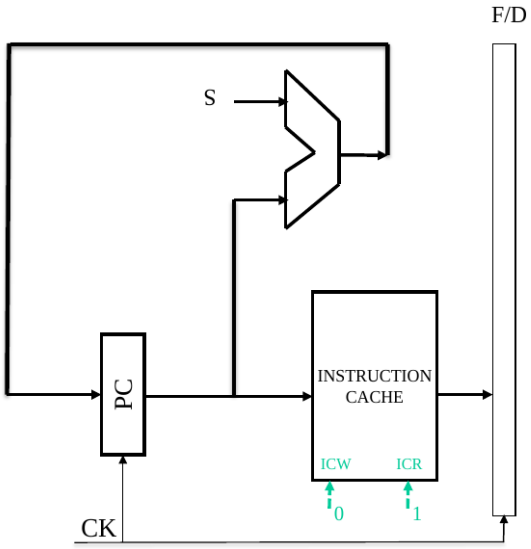


L'unità di elaborazione con pipeline

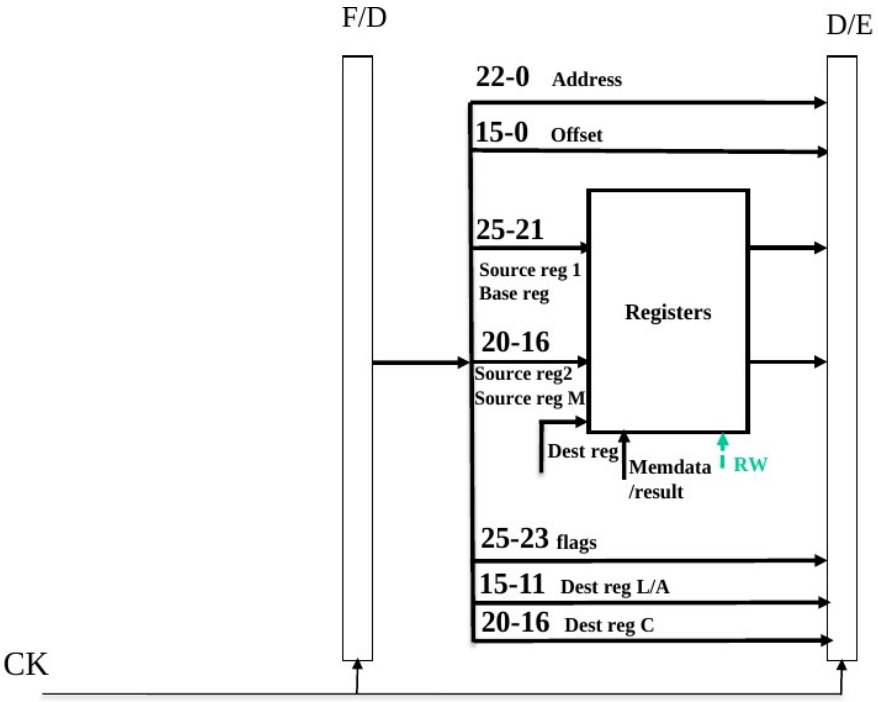
• Principio guida:
– Permettere il riuso delle componenti per l'istruzione successiva
• Introduzione di **registri di pipeline** (registri interstadio)
– Ad ogni ciclo di clock le informazioni procedono da un registro di pipeline a quello successivo
– Il nome del registro è dato dal nome dei due stadi che separa

• **Registro F/D** (Instruction Fetch / Instruction Decode)
• **Registro D/E** (Instruction Decode / Execute)
• **Registro E/M** (Execute / Memory access)
• **Registro M/WB** (Memory access / Write Back)
– Il PC può essere considerato come un registro di pipeline per lo stadio FETCH
• I registri commutano sul fronte positivo dei segnali di abilitazione Flip/flop positive edge triggered

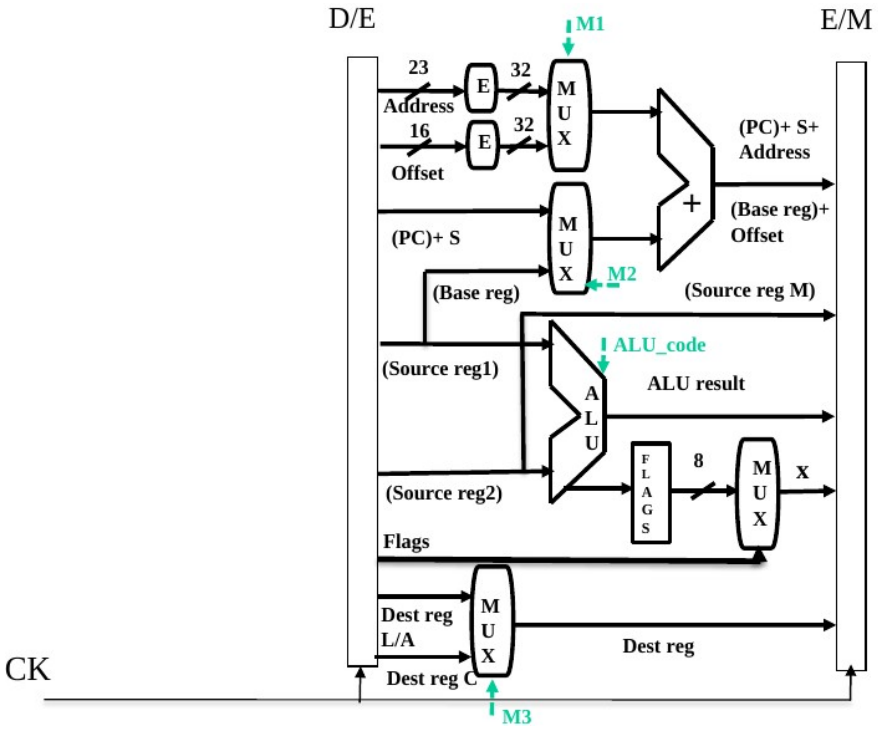
Istruzioni di tipo L/A: Data path dello stadio di fetch



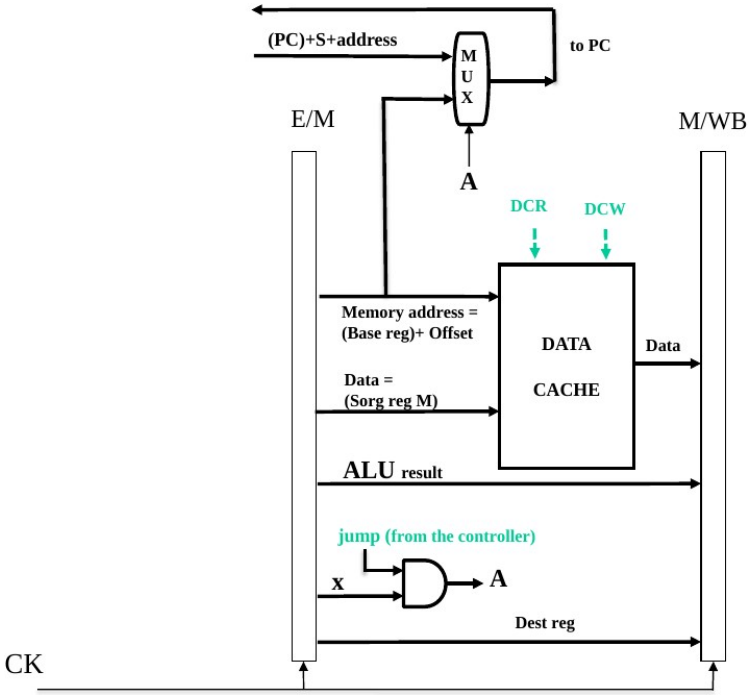
Data path dello stadio Instruction Decode



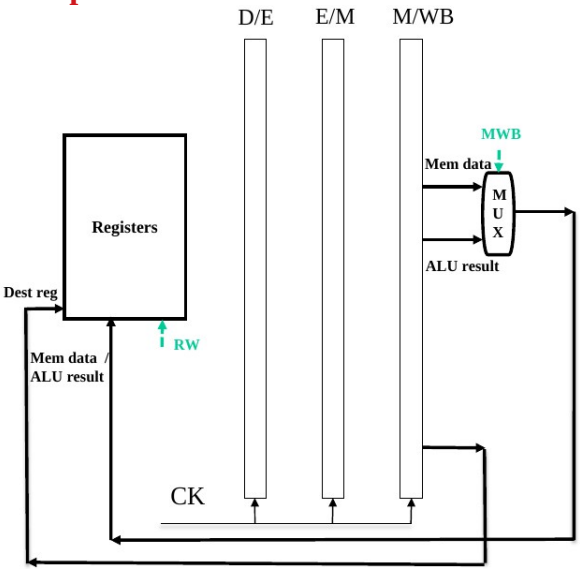
Data path dello stadio di Execute



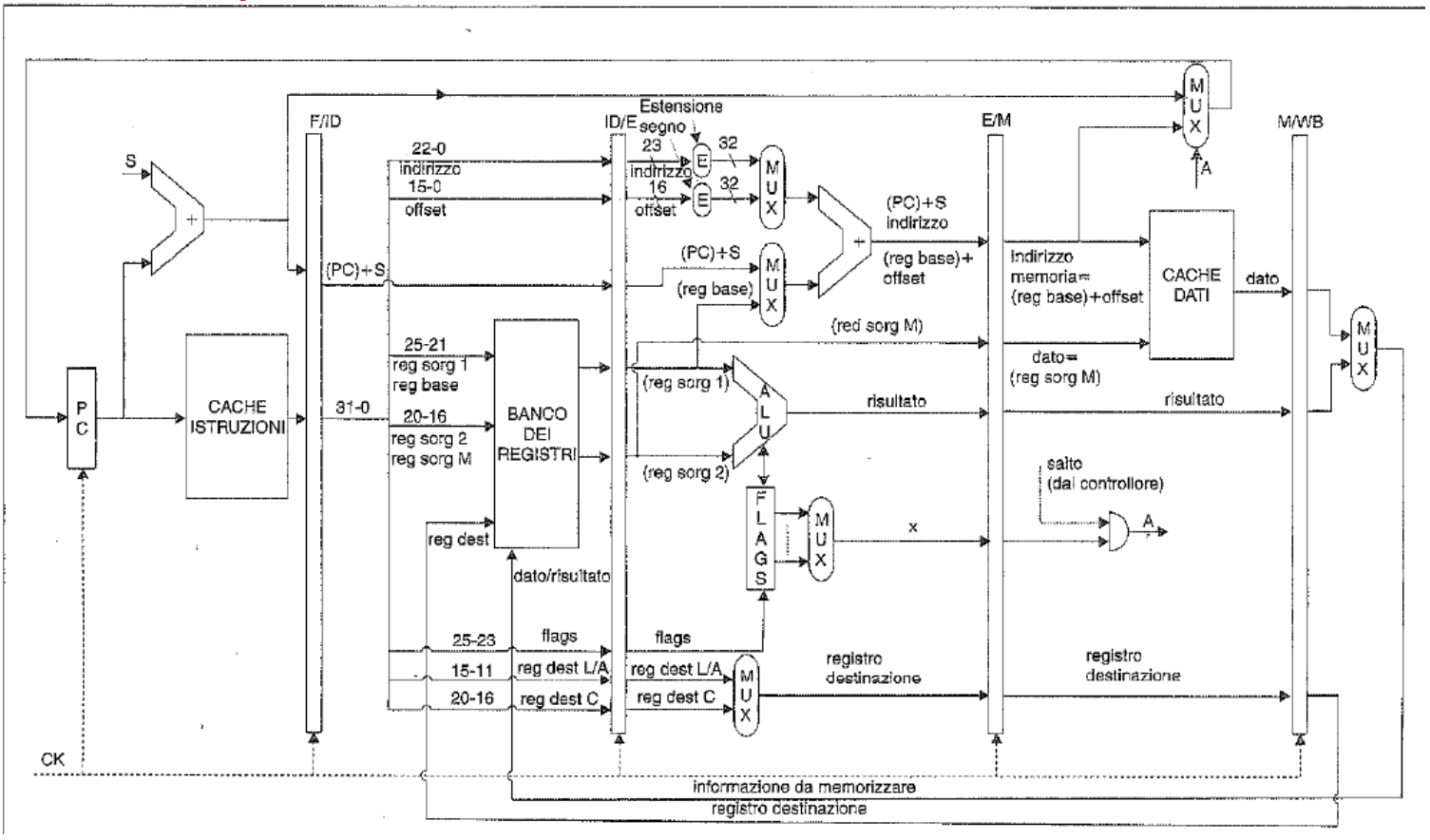
Data path dello stadio di Memory



Data path dello stadio di Write Back



Architettura complessiva



I segnali di controllo

• Nello stadio di **EXECUTE** sono previsti:
– **M1** per pilotare il primo multiplexer nella selezione del primo operando da mandare all'addizionatore tra l'indirizzo di memoria e l'Offset;
– **M2** per pilotare il secondo multiplexer nella selezione del secondo operando da mandare all'addizionatore tra il valore di (PC)+ e (Base_reg):;
– **M3** per selezionare l'appropriato registro destinazione tra Dest_reg_L/A e Dest_reg_C
– **ALU_OPCODE** per comandare l'ALU ad effettuare l'operazione appropriata in funzione del tipo di istruzione da eseguire.
• Nello stadio **MEMORY** sono previsti:
– **DCR** per abilitare la lettura del dato dalla memoria
– **DCW** per abilitare la scrittura del dato in memoria
– **JMP** per consentire il salto di sequenza nell'esecuzione di un programma
• Nello stadio di **WRITE BACK** sono previsti:
– **MWB** per pilotare il multiplexer nella selezione tra dato e risultato
– **RW** per abilitare la scrittura nel banco dei registri di quanto selezionato dal multiplexer

Prestazioni del pipelining

• La presenza della pipeline aumenta il numero di istruzioni contemporaneamente in esecuzione
• Quindi, introducendo il pipelining nel processore, **aumenta il throughput** ...
– Throughput: numero di istruzioni eseguite nell'unità di tempo
• ... ma **non si riduce la latenza** della singola istruzione
– Latenza: tempo di esecuzione della singola istruzione, dal suo inizio fino al suo completamento
– Un'istruzione che richiede 5 passi, continua a richiedere 5 cicli di clock per la sua esecuzione con pipelining, mentre una che ne richiederebbe 4 necessita di 5 cicli di clock

Stadi della pipeline

• Il tempo necessario per fare avanzare un'istruzione di uno stadio lungo la pipeline corrisponde ad un ciclo di clock di pipeline
• Poiché gli stadi della pipeline sono collegati in sequenza, devono operare in modo sincrono
– avanzamento nella pipeline sincronizzato dal clock
– durata del ciclo di clock del processore con pipeline determinata dalla durata dello stadio più lento della pipeline
• Es.: 200 ps per l'esecuzione dell'operazione più lenta
– per alcune istruzioni, alcuni stadi sono cicli sprecati
• Obiettivo dei progettisti: bilanciare la durata degli stadi
• Se gli stadi sono perfettamente **bilanciati** e non ci sono istruz. con cicli sprecati, lo **speedup ideale** dovuto al pipelining è pari al numero di stadi della pipeline

$$\text{Speedup ideale}_{\text{pipeline}} = \frac{\text{tempo tra istruzioni}_{\text{no pipeline}}}{\text{tempo tra istruzioni}_{\text{pipeline}}} = \text{num. stadi pipeline}$$

• Ma, in generale, gli stadi della pipeline non sono perfettamente bilanciati
• L'introduzione del pipelining comporta quindi costi aggiuntivi
– L'intervallo di tempo per il completamento di un'istruzione è superiore al minimo valore possibile
– Lo **speedup reale** sarà minore del numero di stadi di pipeline introdotto
• In genere una pipeline a 5 stadi non riesce a quintuplicare le prestazioni

Miglioramento delle prestazioni

• In generale: partendo dalla pipeline vuota con k stadi, per completare n istruzioni occorrono k + (n-1) cicli di clock:
– k cicli per riempire la pipeline e completare l'esecuzione della prima istruzione
– n-1 cicli per completare le rimanenti n-1 istruzioni
• All'aumentare del numero di istruzioni n, il rapporto tra i tempi totali di esecuzione su macchine senza e con pipeline si avvicina al limite ideale
• Il tempo per riempire/svuotare la pipeline diventa trascurabile rispetto al tempo totale per completare le istruzioni
• Nel caso asintotico ($n \to \infty$)
– La latenza di alcune istruzioni potrebbe peggiorare, per esempio le istruzioni logiche/aritmetiche nel caso di processore multiciclo necessitano
• passa da 800 ps (senza pipelining) a 1000 ps (con pipelining)
– Il throughput però migliora di 4 volte
• Passa da 1 completata ogni 800 ps (senza pipelining) ad 1 istruzione completata ogni 200 ps (con pipelining)

CASO IDEALE

– Se consideriamo un processore multiciclo con un clock da 200 ps ed un processore con pipelining (con 5 stadi da 200 ps ciascuno) nel caso asintotico
– La latenza della singola istruzione rimane invariata e pari a 1000 ps
– Il throughput migliora di 5 volte
• Passa da 1 istruzione completata ogni 1000 ps (senza pipelining) ad 1 istruzione completata ogni 200 ps (con pipelining)
• Quindi il pipelining potrebbe **incrementare il throughput del processore** (numero di istruzioni completate nell'unità di tempo), ma sicuramente non riduce il tempo di esecuzione (latenza) della singola istruzione
• Anzi, in generale il pipelining aumenta il tempo di esecuzione della singola istruzione, a causa di sbilanciamenti tra gli stadi della pipeline e overhead di controllo della pipeline
– Lo sbilanciamento tra gli stadi della pipeline riduce le prestazioni
• Il clock non può essere minore del tempo necessario per lo stadio più lento della pipeline
– L'overhead della pipeline è causato
• dai ritardi dei registri di pipeline e dal clock skew (ritardo di propagazione del segnale di clock sui fili)
• dalla presenza di **criticità**

Le criticità

• Le criticità (o conflitti o alee) sorgono nelle architetture con pipelining quando non è possibile eseguire un'istruzione nel ciclo immediatamente successivo senza cambiare la semantica del programma eseguito in una struttura multiciclo

Tre tipi di criticità

• **Criticità strutturali**:
  – Tentativo di usare la stessa risorsa hardware da parte di diverse istruzioni in modi diversi nello stesso ciclo di clock, come nella lettura e scrittura contemporanea di un registro del banco dei registri da una istruzione in fase di decode ed una in fase di write back.

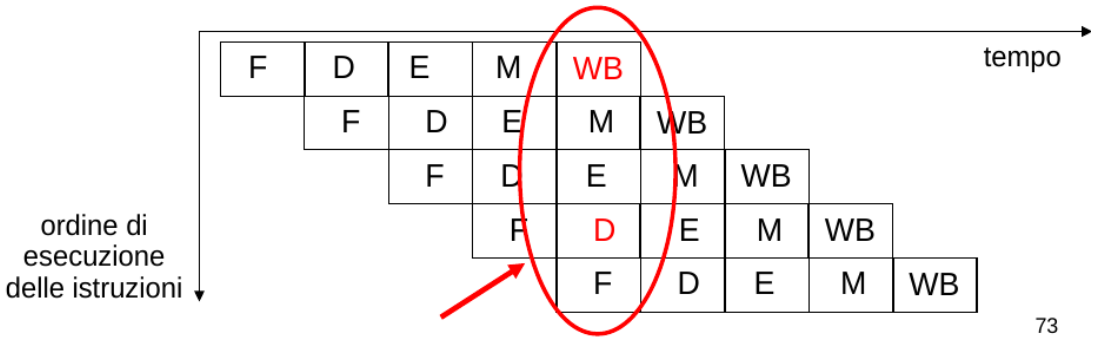
– Da notare che si avrebbe un'altra criticità strutturale se nel processore didattico avessimo un'unica memoria per le istruzioni e i dati

• **Criticità sui dati:**
  – Tentativo di usare un risultato prima che sia disponibile
  – Es.: istruzione che dipende dal risultato di un'istruzione precedente che è ancora nella pipeline

• **Criticità sul controllo**:
  – Nel caso di salti condizionati, decidere quale prossima istruzione sia da eseguire prima che la condizione sia valutata

Criticità strutturali

• Nel processore didattico c'è un conflitto strutturale sul banco dei registri
• Si potrebbe evitare se il banco dei registri venisse progettato per evitare conflitti tra la lettura e la scrittura nello stesso ciclo
• **Soluzione**
– Scrittura del banco dei registri nella prima metà del ciclo di clock
– Lettura del banco dei registri nella seconda metà del ciclo di clock



73

Criticità sui dati

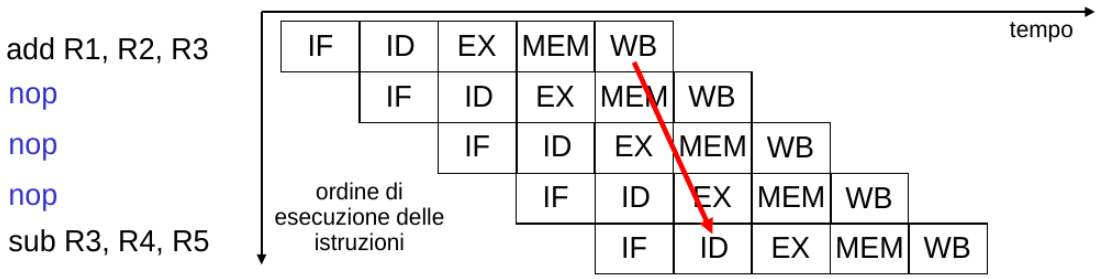
• Un'istruzione dipende dal risultato di un'istruzione precedente che è ancora nella pipeline
• Esempio 1:
add R3, R4, **R5**
sub R3, **R5**, R6
– uno degli operandi sorgente di sub (R5) è prodotto da add, che è ancora nella pipeline
– Criticità sui dati di tipo **define-use**
• Esempio 2:
load R3, 122(R1)
sub R5, **R3**, R6
– uno degli operandi sorgente di sub (R3) è prodotto da load, che è ancora nella pipeline
– Criticità sui dati di tipo **load-use**

Soluzioni per criticità sui dati

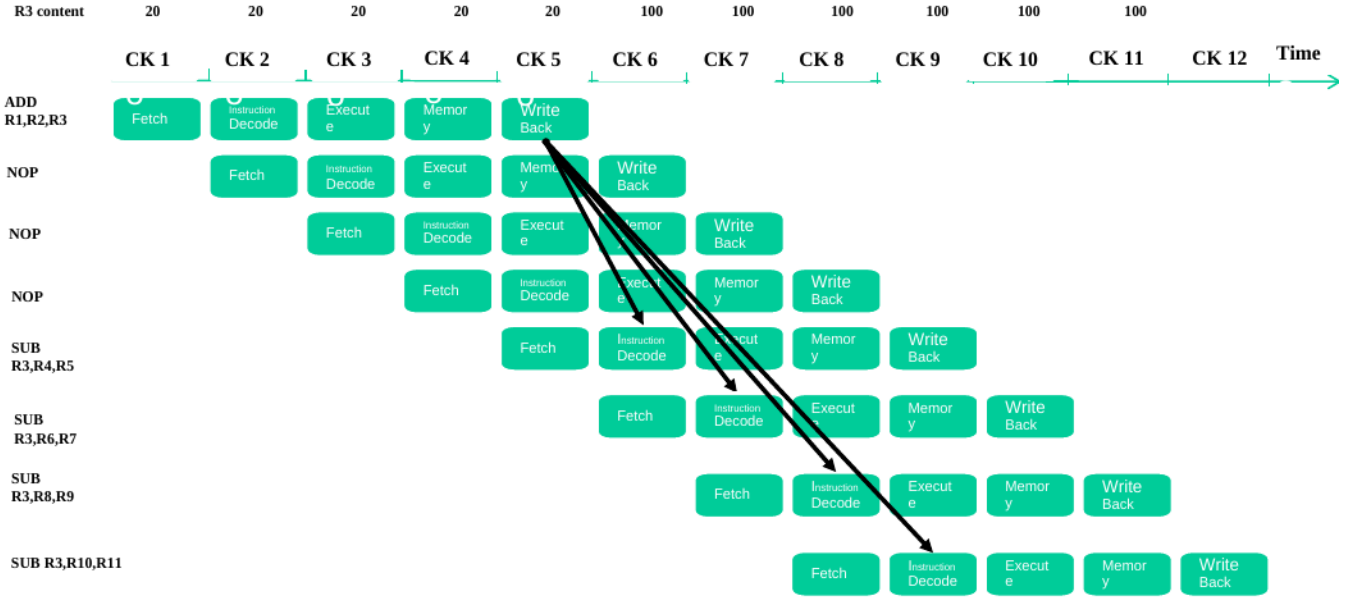
• Soluzioni di tipo software
– inserimento di istruzioni **nop** (no operation), ma peggiora il throughput
– riordino delle istruzioni: **spostare** istruzioni "innocue" in modo che esse eliminino la criticità
• Soluzioni di tipo hardware
– inserimento di bolle (**bubble**) o stalli nella pipeline
• si inseriscono dei tempi morti
• peggiora il throughput
– propagazione o scavalcamento (**forwarding** o bypassing)
• si propagano i dati in avanti appena sono disponibili verso le unità che li richiedono

Define use (soluzione SW) Inserimento di nop

• Esempio 1: l'assemblatore deve inserire tra le istruzioni add e sub tre istruzioni nop, facendo così scomparire il conflitto (caso di presenza di criticità strutturale sul banco registri)
– L'istruzione nop è l'equivalente software dello stallo



Nel caso di assenza di criticità strutturale sul banco dei registri sarebbero sufficienti solo **due nop**



Define use(soluzione SW): riordino delle istruzioni

• L'assemblatore o il programmatore riordina le istruzioni in modo da impedire che istruzioni correlate siano troppo vicine

• L'assemblatore o il programmatore cerca di inserire tra le istruzioni correlate (che presentano dei conflitti) delle istruzioni **indipendenti** dal risultato delle istruzioni precedenti

– Quando l'assemblatore non riesce a trovare istruzioni indipendenti deve inserire istruzioni nop

• Esempio

Codice con criticità	Codice senza criticità
criticità: add R1, R2, R3	add R1,R2, R3
criticità: sub R3, R6, R7	sub R8, R9, R10
sub R8, R9, R10	add R11, R12, R13
add R11, R12, R13	sub R14, R15, R16
sub R14, R15, R15	sub R3, R6, R7

Criticità sul controllo

• Per alimentare la pipeline occorre inserire un'istruzione ad ogni ciclo di clock

• Tuttavia, nel processore didattico la decisione sul salto condizionato non viene presa fino al quarto passo (MEMORY) dell'istruzione **jumpX**

• Comportamento desiderato del salto:

– se il bit selezionato è pari a 0 continuare l'esecuzione con l'istruzione successiva a **jumpx**

– se il bit selezionato è pari a 1 non eseguire le istruzioni successive alla **jumpx** e saltare all'indirizzo specificato

Soluzioni per criticità sul controllo

• **Approccio pessimistico**: non si eseguono istruzioni significative fino al completamento della scelta
• **Approccio ottimistico**: si prospetta che non venga effettuato il salto
•**Vantaggi/svantaggi:**
– **pessimistico**: attesa della decisione (perdita di tempo)
– **ottimistico**: necessità di annullare gli effetti delle istruzioni eseguite nel caso di salto

Soluzioni Pessimistiche

• **Software**: inserimento di 3 nop
• **Hardware**: inserimento di 3 bolle
– si blocca la pipeline finché non è noto il risultato del confronto della jumpX e si sa quale è la prossima istruzione da eseguire

Soluzioni Ottimistiche (Hardware)

Approccio ottimistico: si prospetta che non venga effettuato il salto
Quindi si eseguono le istruzioni in sequenza all'istruzione di salto:
- se il salto non doveva essere effettuato OK
- se il salto doveva essere effettuato è necessario annullare gli effetti dell'esecuzione delle tre istruzioni eseguite (**RECOVERY** dello stato antecedente all'esecuzione delle tre istruzioni erroneamente eseguite)

Ottimizzazione delle prestazioni: predizione del salto

• Tecniche di predizione dinamica:
– Locali
– Globali
• Esempio di tecnica di predizione locale: **BIMODALE**
- utile nel caso di salti collegati a cicli (loop)
- migliora le prestazioni pesantemente in funzione del numero di iterazioni da effettuare prima di uscire dal loop

Tecnica bimodale

• Uso di una tabella di contatori, a cui si accede utilizzando i bit meno significativi dell'indirizzo delle istruzioni di salto
• Tali contatori, di due bit, possono assumere quattro valori:
– 00, fortemente non scelto;
– 01, debolmente non scelto;
– 10, debolmente scelto;
– 11, fortemente scelto.
• Ad ogni predizione di salto, se la condizione è verificata si passa da un valore a quello crescente (e quindi se si è già nello stato 11 si rimane in tale stato), mentre se la condizione non è verificata si passa al valore inferiore (anche in questo caso se si è già nello stato 00 si rimane in tale stato).

Altri problemi della pipeline

• FPU: L'FPU ha tempi di elaborazione molto lunghi rispetto a quelli del singolo stadio visto fino ad ora, necessità quindi di mettere in stallo per lunghi periodi il processore
• Gestione interruzioni/eccezioni: In un calcolatore con pipelining è difficile associare sempre in modo corretto un'eccezione all'istruzione che l'ha provocata (ci sono in esecuzione un numero di istruzioni pari al numero degli stadi) o quale istruzione è in esecuzione al momento dell'arrivo di una interruzione
- **Eccezione/Interruzione imprecisa**: non è associata alcuna istruzione in modo esatto, come nel multiciclo che ha causato l'eccezione
- **Eccezione/Interruzione precisa**: è sempre associata all'istruzione esatta che ha causato l'eccezione o all'istruzione in fase di execute durante l'arrivo dell'interruzione.

		EXECUTE						MEMORY			WRITE BACK	
Instruction	OPCODE	M1	M2	M3	OP_3	$0P_2$	$0P_1$	DCW	DCR	JMP	MWB	RW
ADD	000001	-	-	0	0	0	1	0	-	0	1	1
SUB	000010	-	-	0	0	1	0	0	-	0	1	1
OR	000011	-	-	0	0	1	1	0	-	0	1	1
AND	000100	-	-	0	1	0	0	0	-	0	1	1
NOT	000101	-	-	0	1	0	1	0	-	0	1	1
LOAD	000110	1	1	1	-	-	-	0	1	0	0	1
STORE	000111	1	1	-	-	-	-	1	0	0	-	0
JMPC	001000	0	0	-	-	-	-	0	0	1	-	0
NOP	000000	-	-	-	0	0	0	0	-	0	-	0