

Appunti Architettura degli Elaboratori

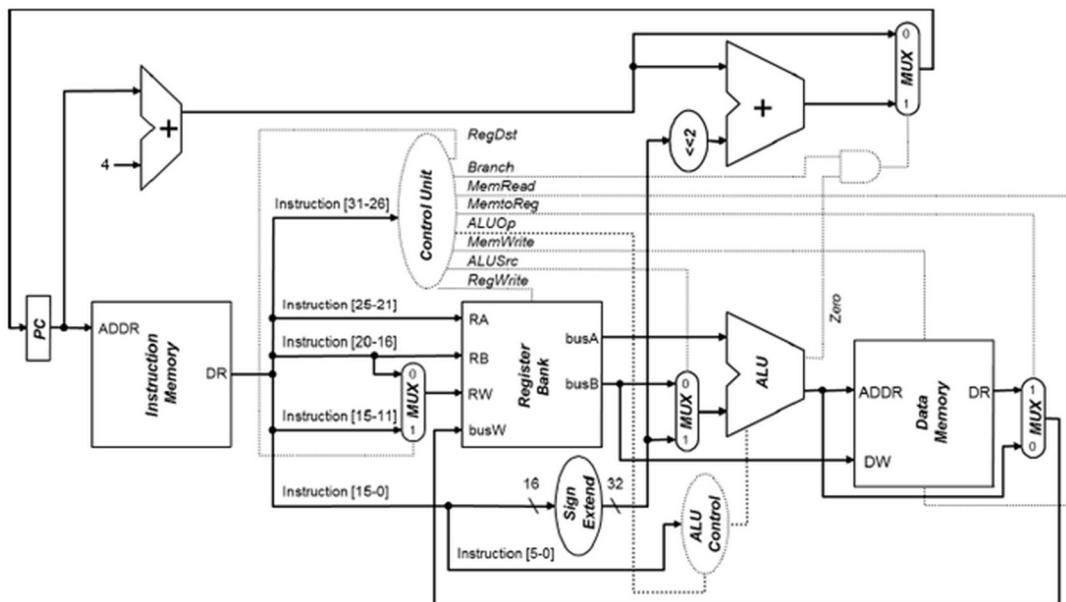
Domande e Risposte

Capitolo 5 - Datapath

1) Datapath: definizione

Il **datapath** è un insieme di unità di calcolo, come ad esempio le unità di elaborazione, i registri e i moltiplicatori necessari per l'esecuzione delle istruzioni nella **CPU**.

Il passaggio di due operandi attraverso la **ALU** e la memorizzazione del risultato in un nuovo registro viene detto **ciclo di datapath**.



2) Come viene realizzato un datapath?

Per realizzare un datapath è necessario:

- stabilire il set di istruzioni da implementare;
 - identificare i componenti del **datapath** (**ALU, register file, ecc**);
 - stabilire la metodologia di **clocking**;
 - assemblare il **datapath** identificando i segnali di controllo;
 - analizzare l'implementazione di ogni istruzione per determinare il setting dei segnali di controllo;
 - assemblare la logica di controllo.

3) Quali sono i passi per l'esecuzione di un'istruzione?

I passi sono:

- **Fetch**: la quale legge l'istruzione dalla memoria e la salva in un registro dedicato (**Instruction Register**) e l'indirizzo di memoria che indica l'istruzione da leggere si trova nel registro **Program Counter (PC)**. Dopo la lettura

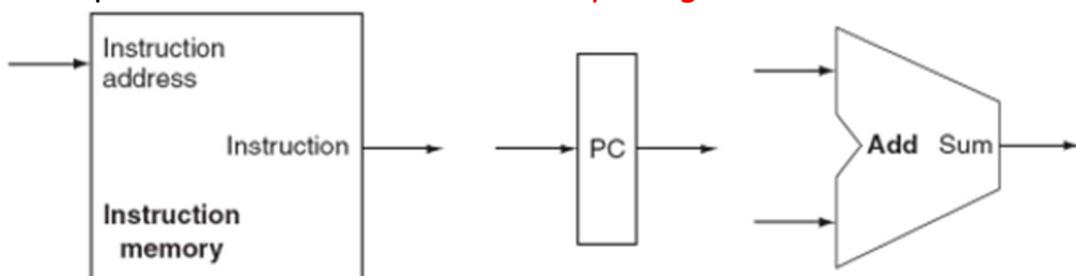
dell'istruzione in PC viene incrementato di 4 la prossima istruzione da leggere (usando l'**ALU**).

→ **Decode**: decodifica i vari campi dell'istruzione per decidere quali sono i passi necessari per la sua esecuzione;

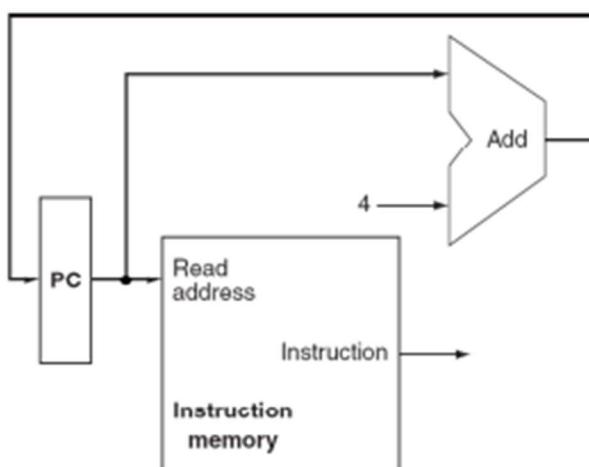
→ **Execute**: svolge i passi necessari per eseguire l'istruzione.

4) Quali sono i componenti necessari per implementare la fase di fetch?

I componenti sono **Instruction memory**, **Program Counter** e **Adder**.



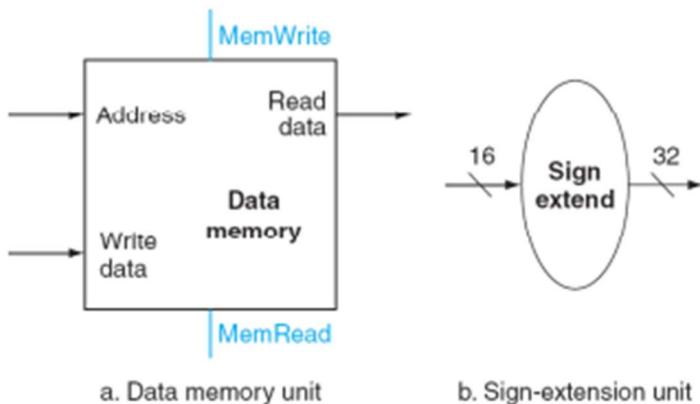
Calcolo: PC+4



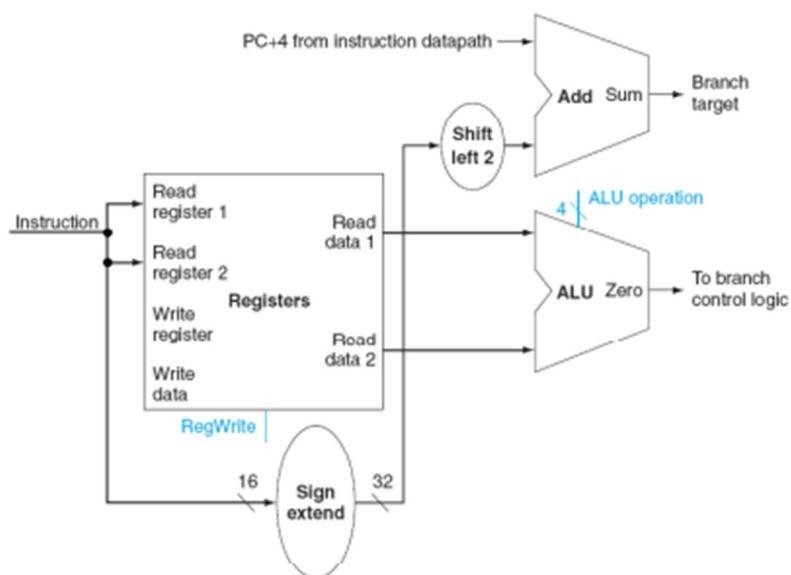
Execute: R-Type



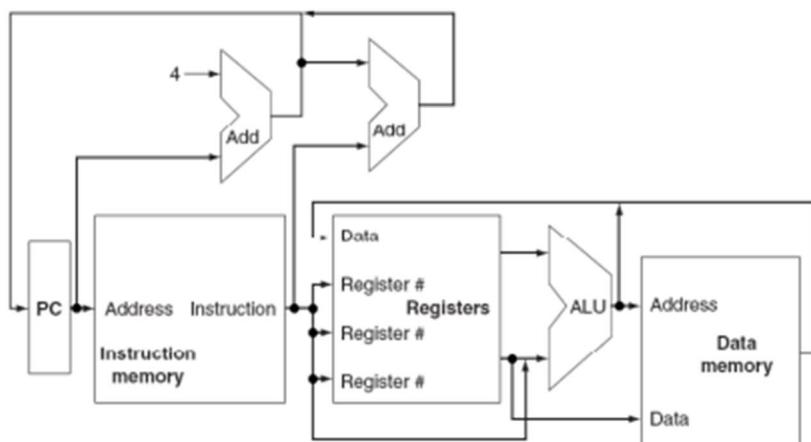
Execute: load & store

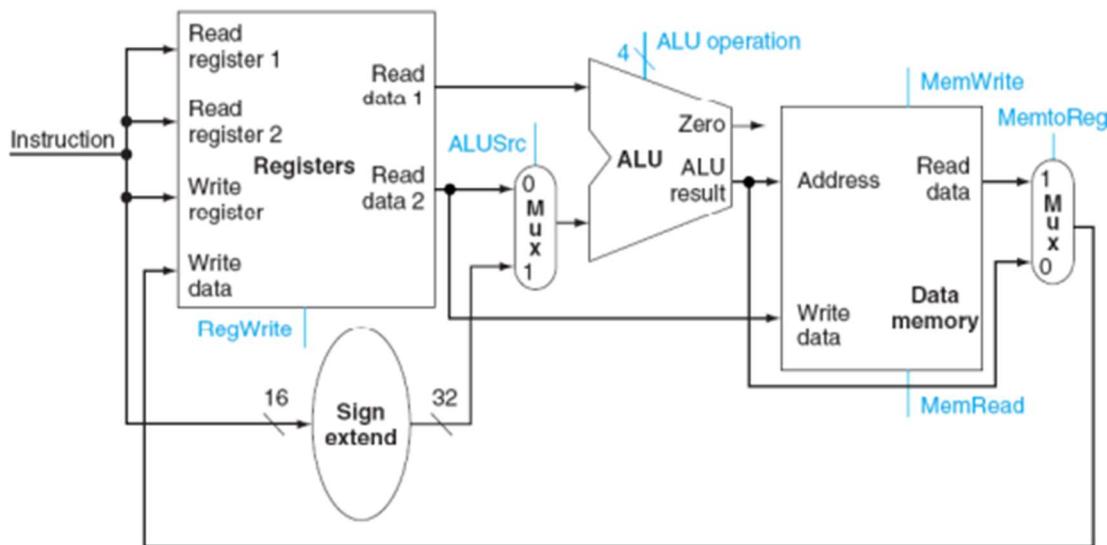


Execute: beq



Una vista astratta del datapath





5) Quali sono le metodologie di clocking?

Le metodologie di clocking sono:

a) **Single clock**: in cui si ha:

→ ciclo singolo di lunghezza fissa uguale al tempo necessario per eseguire l'istruzione più lunga;

→ ogni istruzione viene eseguita in un ciclo di clock;

→ spreco di tempo (svantaggio);

b) **Multi clock**: in cui si ha:

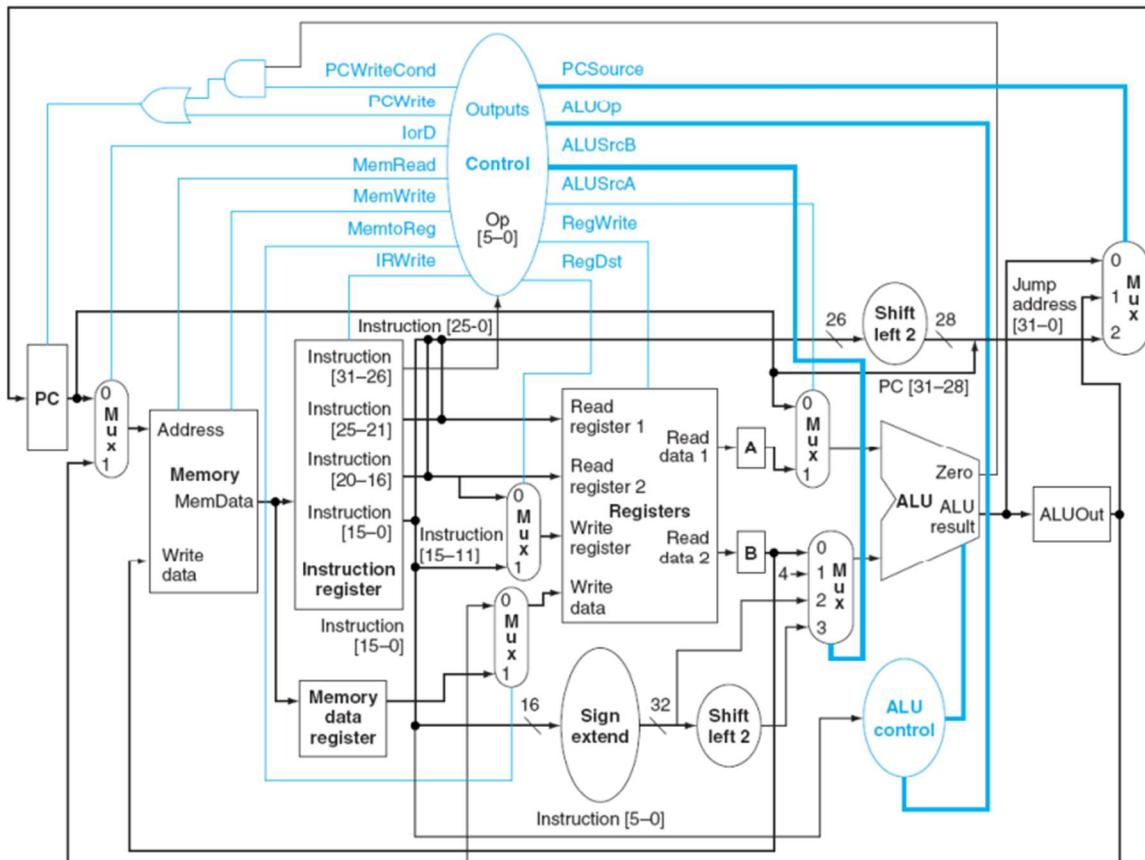
→ ciclo di lunghezza fissa più corto;

→ ogni istruzione viene eseguita in più cicli di clock;

→ istruzioni di tipo diverso - eseguite in un numero di cicli di clock diverso.

6) Datapath: automa per il controllo

Step name	Azione per istruzione R - type	Azione per riferimenti ad istruzioni
Istruzione fetch	/	$IR \leftarrow \text{Memory}[PC]$ $PC \leftarrow PC + 4$
Istruzione decode/register fetch	/	$A \leftarrow \text{Reg}[IR[25:21]]$ $B \leftarrow \text{Reg}[IR[20:16]]$ $ALUOut \leftarrow PC + (s_ext \ll 2)$
Execution, address computation, branch/jump completion	$ALUOut \leftarrow A \text{ op } B$	$ALUOut \leftarrow A + \text{sign - extend}(IR[15:0])$
Memory access or R type completion	$\text{Reg}[IR[15:11]] \leftarrow ALUOut$	Load: $MDR \leftarrow \text{Memory}[ALUOut]$ Store: $\text{Memory}[ALUOut] \leftarrow B$
Memory read completion	/	Load: $\text{Reg}[IR[20:16]] \leftarrow MDR$



Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None.	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None.	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
lOrD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of the memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by PCSource.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

Actions of the 2-bit control signals

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC + 4[31:28]) is sent to the PC for writing.

7) Datapath multiciclo: quali sono e quale funzione svolge ciascuna delle unità funzionali?

I componenti del **datapath multiciclo** sono:

- **Memoria**: contiene istruzioni e dati; sulla base dell'indirizzo in input, restituisce in output l'istruzione o il dato letto;
- **Register file**: 32 registri che contengono i dati utilizzati nel corso dell'esecuzione delle istruzioni; restituisce in output il contenuto dei due registri letti (indicati dai due input - **ReadRegister1** e **ReadRegister2**) e scrive, se il segnale di scrittura è attivo, il dato in input **WriteData** nel registro indicato dall'indirizzo in input (**WriteRegister**);
- **ALU**: il quale effettua operazioni a seconda del tipo di istruzioni:
 - a) **Istruzioni R - Type**: esegue operazioni aritmetico-logiche su due operandi in funzione del **function_code** indicato dai 6 bit meno significativi dell'istruzione;
 - b) **Istruzioni di accesso a memoria**: calcola l'indirizzo di memoria cui accedere;
 - c) **Istruzioni branch**: confronta i due registri in input;
 - d) **Istruzioni restanti**: incrementa il **PC** (**PC+4**) e calcola il valore di **branch** (che sarà usato solo nel caso di istruzioni di **branch**).
- **Sign Extended**: opera l'estensione di segno dai 16 bit in input ai 32 bit in output.
- **Shift Left 2**: opera uno shift a sinistra dei bit in input (con l'effetto di moltiplicare per 4 l'input).
- **PC**: contiene i 32 bit che indicano l'indirizzo dell'istruzione da eseguire.
- **Instruction register**: contiene i 32 bit che corrispondono alla codifica dell'istruzione prelevata da memoria per l'esecuzione.
- **Memory Data Register**: contiene il dato letto da memoria prima della sua scrittura nel registro destinazione (e.g. nel caso di esecuzione di istruzione **load**).
- **A e B**: contengono i valori letti dai registri del **Register File**;
- **ALUOut**: contiene l'output dell'**ALU**.

8) Qual è il ruolo di ciascuno dei multiplexer?

Il ruolo di ciascun **multiplexer** è il seguente:

- **A**: seleziona l'indirizzo di memoria a cui accedere tra **PC** (nel caso si voglia leggere un'istruzione) e output della **ALU** (nel caso di esecuzione dell'istruzione **load**).
- **B**: seleziona il gruppo bit dell'**IR** che indica il registro in cui scrivere per i due tipi di istruzione **I - type** (**IR[20:16]**) e **R - type** (**IR[15:11]**).

→ **C**: seleziona la sorgente per il dato da scrivere in memoria tra **ALUOut** (per istruzioni **R-type**) e **MemoryDataRegister** (per istruzioni **load**).

→ **D**: seleziona il primo operando dell'operazione aritmetico - logica eseguita dalla **ALU** tra **PC** (per incremento +4) e **registro A** (per istruzioni **R-type**);

→ **E**: seleziona il secondo operando dell'operazione aritmetico - logica eseguita dalla **ALU**, tra i quali:

a) **B** (per istruzioni **R - type**);

b) **4** (per aggiornamento del **PC**);

c) sign - extended **IR[15:0]**;

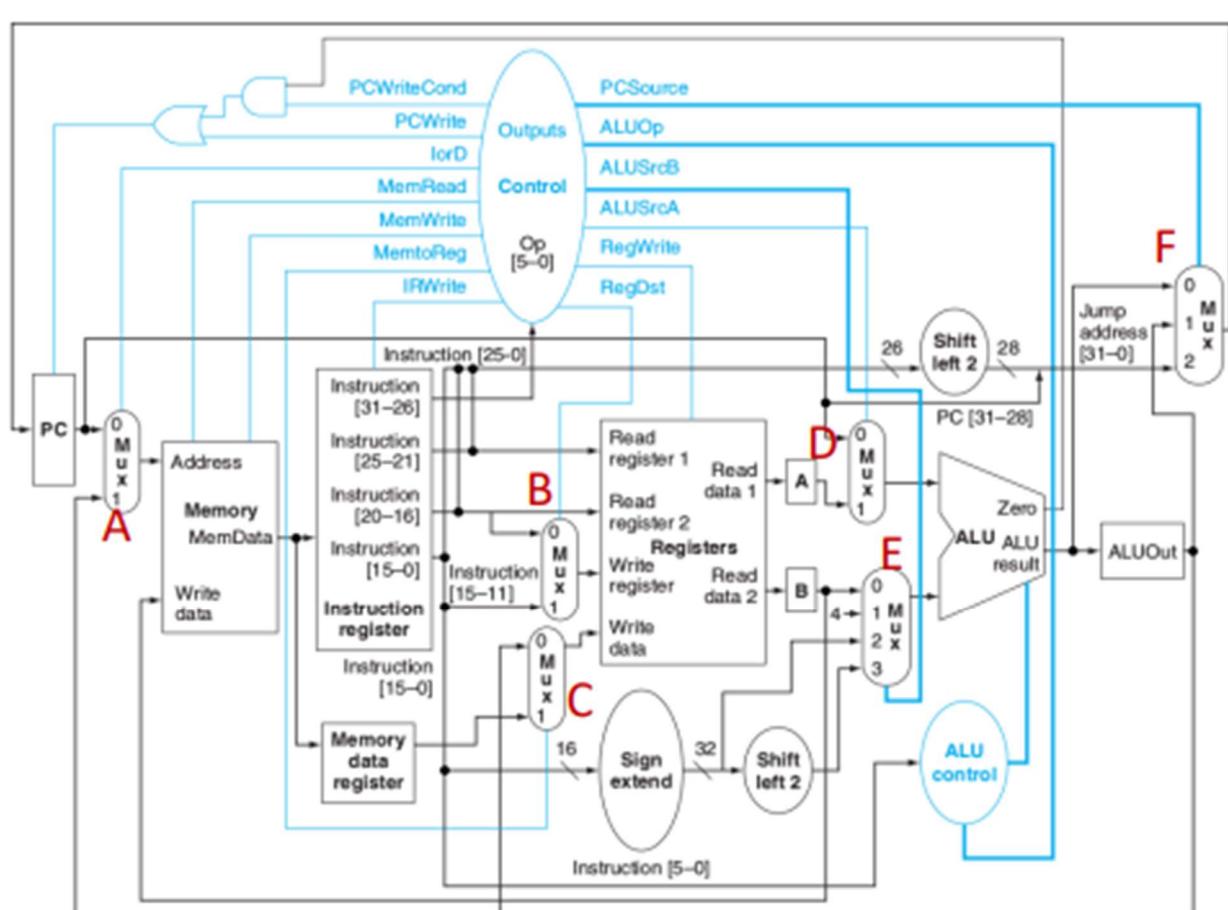
d) sign - extended 2 - shiftees (**j**).

→ **F**: seleziona il valore per l'aggiornamento del **PC**

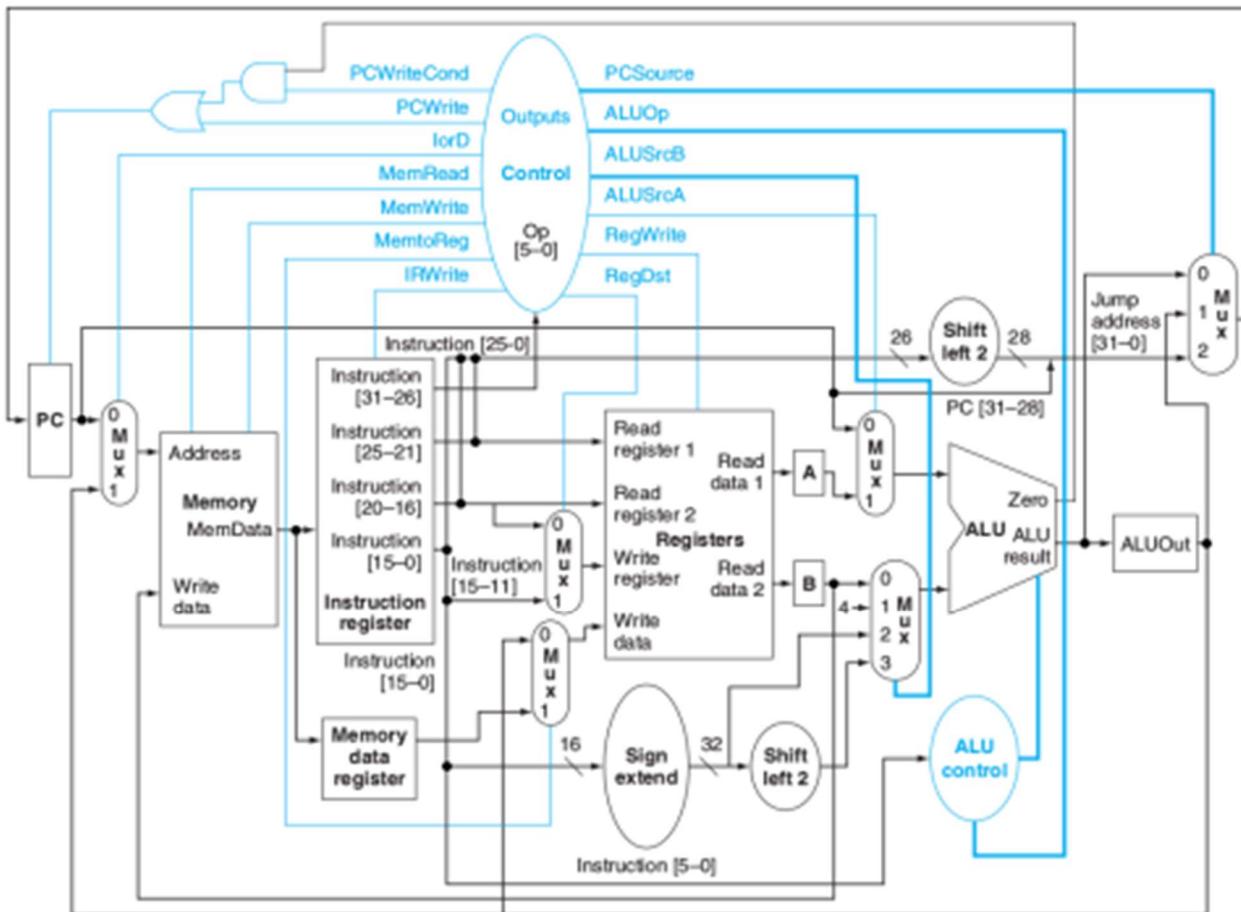
- output dell'**ALU**: **PC** + 4;

- contenuto di **ALUOut**: 2 shifted 26-bit **beq** field;

- jump target address (**IR[25:0]** shifted left 2 bits e concatenato con **PC** + **4[31:28]**).



9) In riferimento allo schema completo del datapath multiciclo (fig. 5.28), in che fase (fetch, decode, execute), e come, sono impostati i segnali di controllo per le diverse classi di istruzioni?



→ L'attivazione dei segnali di controllo per un **datapath multiciclo** può essere descritta da un automa a stati finiti e i segnali di controllo (`next_state` per l'automa) sono definiti **in funzione dello stato corrente e dell'Opcode**.

→ Per tutti i tipi di istruzioni, nella fase di **fetch** (**stato 0**) sono impostati i seguenti **segnali**:

- IorD** = 0 per selezionare PC come sorgente per l'indirizzo di memoria;
- MemRead**: per leggere un'istruzione da memoria;
- IRWrite**: per scrivere l'istruzione letta nell'Instruction register;
- ALUSrcA**, **ALUSrcB**, **ALUOp**, **PCWrite**, e **PCSource** sono impostati per calcolare **PC + 4** e memorizzare il nuovo valore nel **PC**.

→ Per tutti i tipi di istruzioni, nella fase di **decode** (stato 1) sono impostati i segnali **ALUSrcA**, **ALUSrcB** e **ALUOp** per calcolare il valore di **branch**.

→ Nella fase di **execute** sono impostati i segnali di controllo, in funzione del tipo di istruzione in esecuzione.

→ Le istruzioni di **jump** o **branch**, dopo lo stato 1, richiedono un ciclo di **clock**

per il loro completamento.

→ Le istruzioni di tipo **R-type** e le istruzioni di memorizzazione (store), richiedono due cicli di clock per l'esecuzione di una sequenza di due stati d'impostazione dei segnali.

→ Le istruzioni di lettura da **memoria** richiedono tre cicli di clock per l'esecuzione di una sequenza di tre stati d'impostazione dei segnali.

10) Per quali classi di istruzioni, e come, è utilizzata la ALU (nella fase di execute)?

La **ALU** (nella fase di **execute**) è utilizzata per:

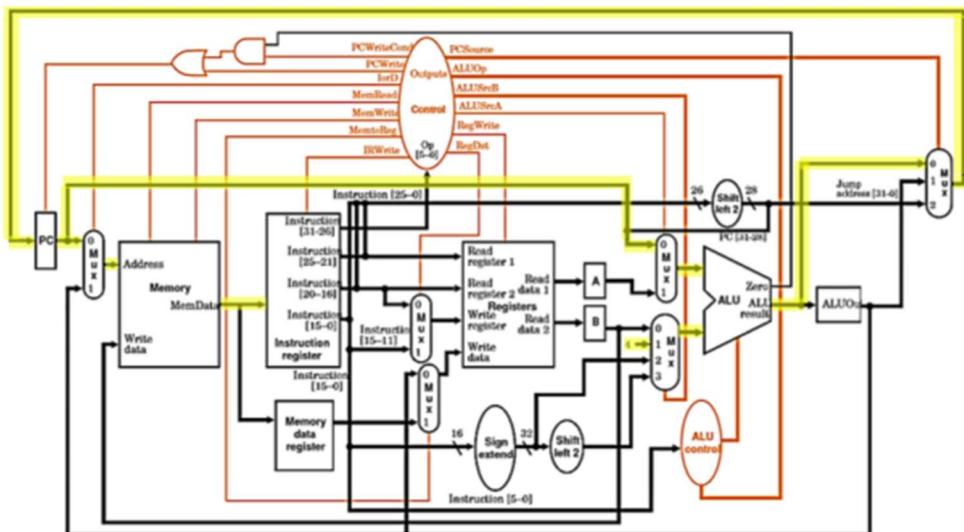
- **istruzioni R-type**: per esecuzione dell'operazione indicata tra i due operandi;
- **istruzioni di accesso a memoria (lw/sw)**: per calcolare la somma tra l'offset e il contenuto del registro base;
- **istruzioni di salto condizionato (branch)**: per il confronto dei valori contenuti nei due registri che compaiono nella condizione. L'indirizzo di salto è già in **ALUout** perché è stato calcolato nella fase di decode;
- non viene usata nella fase di execute di istruzioni **jump**.

11) Quanti cicli sono necessari per l'esecuzione dell'istruzione add \$s0, \$s1, \$s2?

L'istruzione **add \$s0, \$s1, \$s2** è di tipo R-type e richiede quindi 4 cicli per essere completata.

1 ciclo

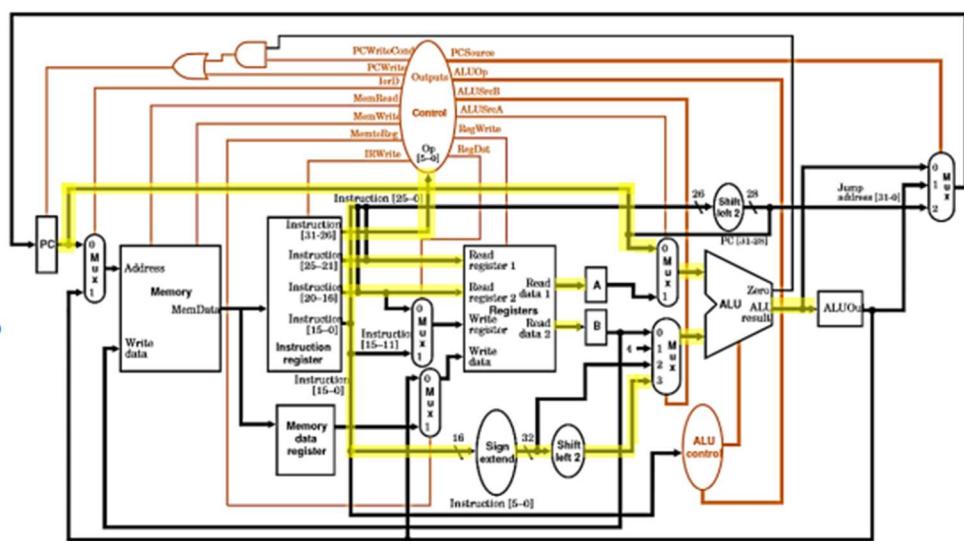
Signal	Value
PCWrite	1
lOrD	0
MemRead	1
MemWrite	0
IRWrite	1
PCSource	00
ALUOp	00
ALUSrcB	01
ALUSrcA	0
RegWrite	0



2 ciclo

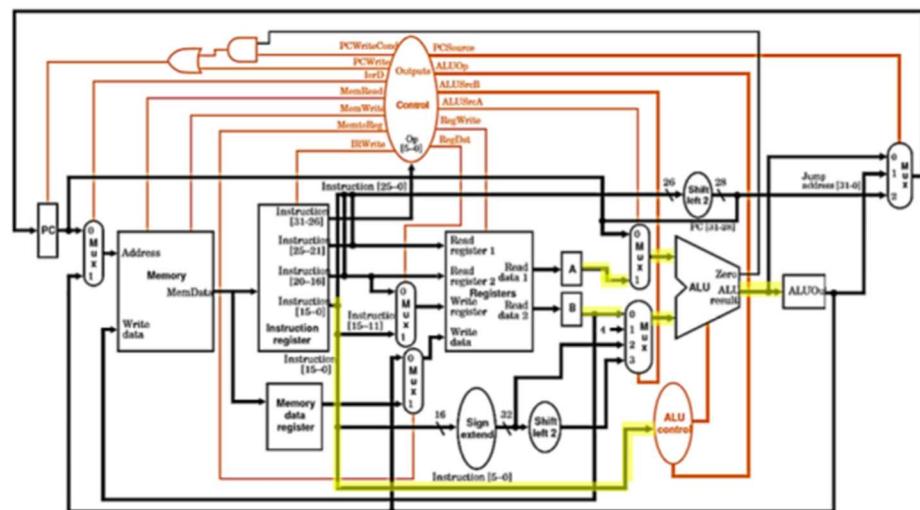
Signal	Value
ALUOp	00
ALUSrcB	11
ALUSrcA	0

N.B. anche se in questo caso (istruzione R-type) non sarà necessario, è comunque calcolato il valore di branch che viene utilizzato per l'aggiornamento del PC nel caso di istruzioni di salto condizionato



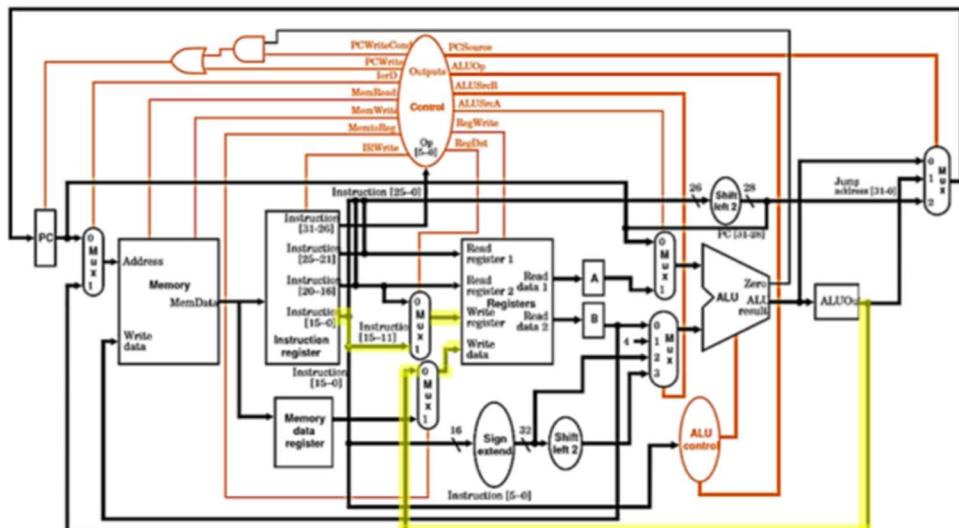
3 ciclo

Signal	Value
ALUOp	10
ALUSrcB	00
ALUSrcA	1



4 ciclo

Signal	Value
MemtoReg	0
RegWrite	1
RegDst	1

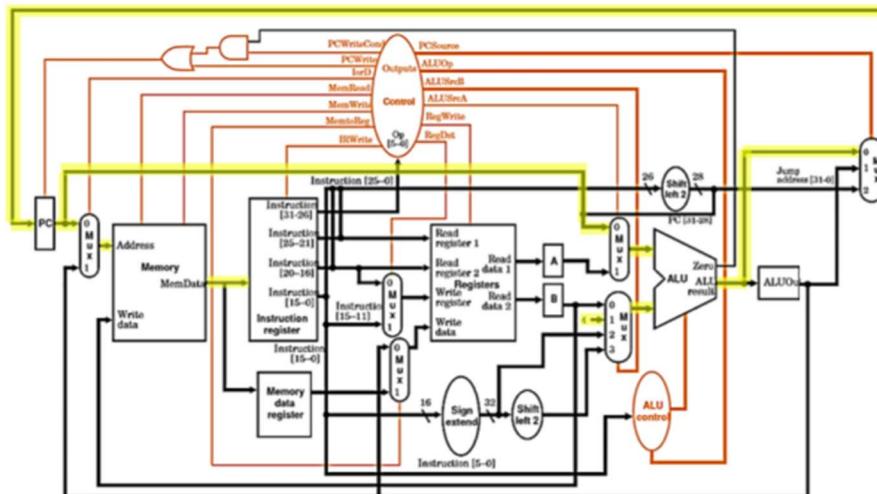


12) Quanti cicli sono necessari per l'esecuzione dell'istruzione `beq $s0, $s1, imm`?

L'istruzione `beq $s0, $s1, imm` richiede 4 cicli per essere completata.

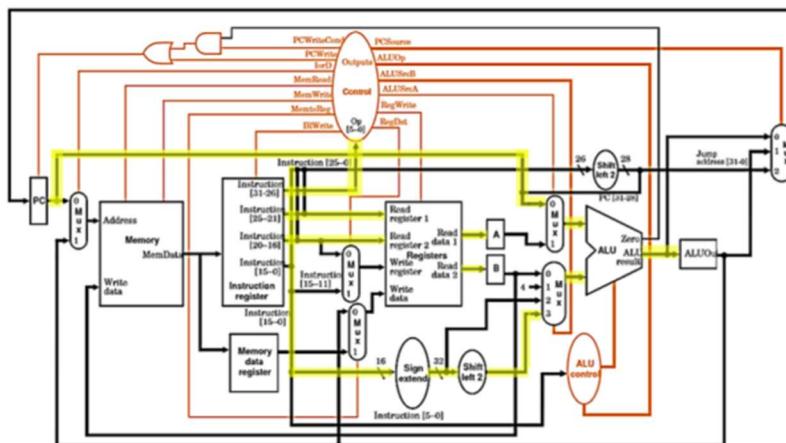
1 ciclo

Signal	Value
PCWrite	1
lrd	0
MemRead	1
MemWrite	0
IRWrite	1
PCSource	00
ALUOp	00
ALUSrcB	01
ALUSrcA	0
RegWrite	0



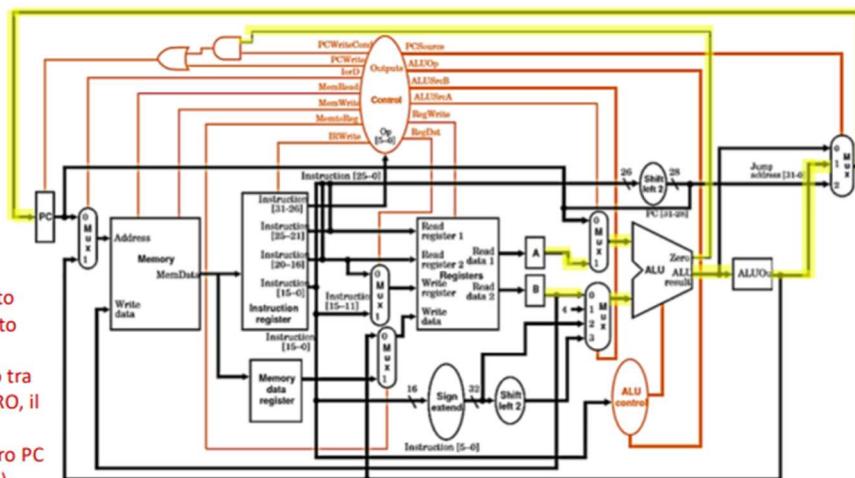
2 ciclo

Signal	Value
ALUOp	00
ALUSrcB	11
ALUSrcA	0



3 ciclo

Signal	Value
ALUOp	01
ALUSrcB	00
ALUSrcA	1
PCSource	01
PCWriteCond	1



N.B. il valore di branch calcolato nel ciclo precedente viene usato per l'aggiornamento del PC

La ALU è usata per il confronto tra A e B e solo se il risultato è ZERO, il segnale di output della ALU consente la scrittura del registro PC (v. porta AND in alto a sinistra!)

13) Come implementare l'istruzione jr nel datapath multiciclo?

L'istruzione **jr** salta all'istruzione contenuta nel registro rs. Dopo la fase di fetch/decode, avremo il contenuto di rs in A, e il contenuto di rt in B. rt nel segmento di un'istruzione jr vale 0. Si può quindi effettuare una somma tra A e B, e impostare **PCSource** in modo tale da usare l'uscita della **ALU** come ingresso del PC.

Segnali di controllo attivati:

ALUSrcA = 1 // il registro con l'indirizzo a cui dobbiamo saltare

ALUSrcB = 0 // vale 0 nella jr

ALUOp = 00 / /somma

PCSource = 00 // l'output della ALU

PCWrite

14) Modificare l'implementazione seguente (immagine del datapath multi ciclo) in modo che sia possibile eseguire l'operazione jal senza l'utilizzo dell'ALU

Si crea un collegamento diretto tra il **MUX** del data-write e il **PC** ed infine si aggiunge al **MUX** prima del **register-file** un ingresso per il registro 31.

15) Spiegare per che cosa vengono utilizzati i multiplexer nel datapath

I **multiplexer** nel **datapath** vengono utilizzati per rendere possibili le diverse operazioni effettuabili nell'architettura mips, i segnali di controllo provengono dalla control unit che decide in base al tipo di istruzione quali segnali abilitare per poterla eseguire.

Capitolo 6 - Eccezioni/Interruzioni

16) Differenza tra eccezione e interruzione

L'**eccezione** è un evento sincrono generato all'interno del processore e provocato da problemi nell'esecuzione di un'istruzione (overflow, istruzione non valida, errori, pagine non presenti in memoria), mentre l'**interruzione** è un evento asincrono che giunge dall'esterno del processore (terminazione di un'operazione I/O la cui esecuzione era richiesta dalla **CPU**).

17) Caratteristica importante di eccezione e interruzione

Eccezione: se la condizione di eccezione è risolubile, allora il programma può riprendere l'esecuzione; altrimenti, il programma termina prima della sua fine.

Interruzione: si sospende l'esecuzione del programma utente, si gestisce l'interruzione e poi si riprende l'esecuzione del programma utente.

18) Come gestire le eccezioni?

Tutti i **processori** devono eseguire i seguenti passi per gestire un'eccezione o interruzione:

- a - interruzione dell'esecuzione del programma corrente;
- b - salvataggio parziale dello stato di esecuzione corrente (**PC**) - per riprendere eventualmente l'esecuzione dell'esecuzione del programma corrente, se possibile;
- c - salto a una routine del **sistema operativo (OS)** per gestire l'**eccezione/interruzione**;
- d - esecuzione della routine del **Sistema Operativo (OS)**;
- e - ripristino dello stato di esecuzione del **programma** e continuazione dell'esecuzione del programma.

19) Come vengono rilevate le eccezioni?

È possibile rilevare le eccezioni attraverso due modalità:

→ **Indirizzo fisso - Registro fisso (Cause)**: utilizzata in **MIPS** ed è una modalità in cui viene salvato l'identificativo numerico dell'eccezione nel registro dedicato chiamato **Cause** e il gestore accede al registro per determinare la causa dell'eccezione

→ **Interruzioni vettorizzate**: viene predisposto un vettore di indirizzi, uno per ogni tipo di eccezioni/interruzioni, da indirizzare tramite il codice numerico dell'eccezione/interruzione.

Exception Program Counter (EPC): registro in cui viene salvato l'indirizzo dell'istruzione corrente che ha causato l'eccezione.

Esempi di eccezione:

1 - **istruzione non valida** (in **MIPS** l'unità di controllo rileva tale eccezione sulla base del di controllo rileva tale eccezione sulla base del **OPCODE** di un'istruzione);

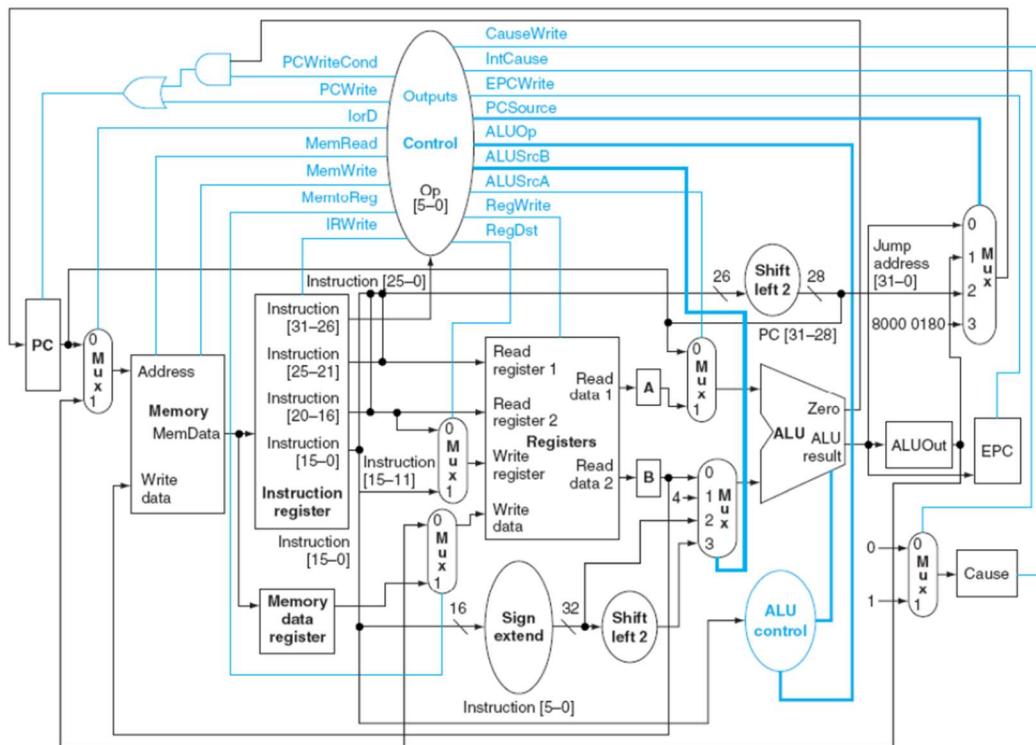
2 - **overflow** (segna che arriva all'unità di controllo dalla **ALU**).

20) Come vengono gestite le eccezioni in **MIPS**?

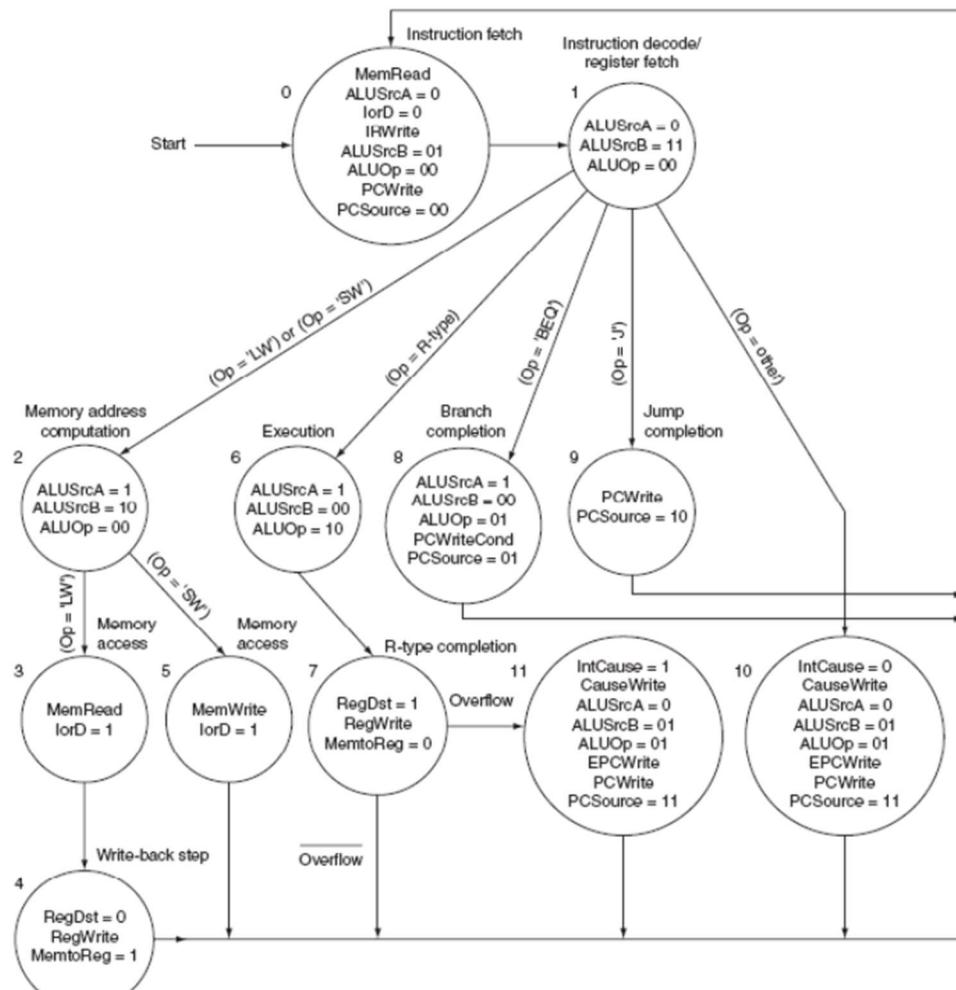
In **MIPS**, le eccezioni vengono gestite nella seguente maniera:

- viene salvato l'indirizzo dell'istruzione che ha causato l'eccezione (in **EPC**);
- viene aggiornato il **PC** con l'indirizzo **0x80000180 (global pointer)**;
- viene eseguita la procedura di gestore delle eccezioni - **exception handler** (collocata nello spazio **kernel text** a partire dalla locazione **0x80000180**);
- viene ripristinato lo stato al momento dell'eccezione.

Datapath modificato con aggiunta di EPC e Cause:



FSM



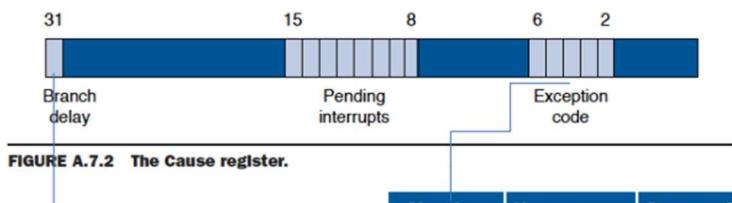


FIGURE A.7.2 The Cause register.

1 se l'eccezione è avvenuta durante esecuzione di un'istruzione del branch delay slot

Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

Register name	Register number	Usage
BadVAddr	8	memory address at which an offending memory reference occurred
Count	9	timer
Compare	11	value compared against timer that causes interrupt when they match
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception
Config	16	configuration of machine

21) Si descriva l'Exception Handler

Porzione di codice assembly che contiene il codice che gestisce le eccezioni situato alla locazione fissa di **0x80000180** in area **ktext** riservata al sistema operativo. Dal momento che viene eseguito dopo l'interruzione di un programma utente in un punto qualsiasi, è tenuto a preservare tutti i registri macchina e ad esso sono riservati due registri **\$k0, \$k1** che può utilizzare senza doverne preservare i contenuti. Al termine del trattamento dell'eccezione, **l'exception handler** ritorna il controllo al programma interrotto. "In un file apposta verranno illustrati esempi exception handler".

Capitolo 7 - Tecniche di gestione input/output

22) Dispositivi I/O: definizione e caratteristiche

Insieme di architetture e dispositivi per il trasferimento di informazioni da e verso l'elaboratore. Questi dispositivi sono eterogenei per:

→ velocità di trasferimento;

→ latenze;

- sincronizzazione;
- modalità di interazione.

23) Cosa si intende per bus di sistema?

Il **bus di sistema** è il canale di comunicazione, che permette la comunicazione tra la **CPU** e le **periferiche I/O** del sistema attraverso il trasferimento dei dati da una parte all'altra dell'elaboratore.

Ci sono tre tipologie di bus:

→ **Bus dati**: è il bus sul quale transitano le informazioni ed è usufruibile da tutti i componenti del sistema, sia in lettura che in scrittura ed è bidirezionale, nel senso che permette il passaggio dei dati in più direzioni contemporaneamente.

→ **Bus degli indirizzi**: è il bus unidirezionale attraverso il quale la CPU decide in quale indirizzo andare a scrivere o a leggere informazioni; sia le celle di memoria (**RAM**) sia le periferiche di **I/O (Input/Output)** sono infatti divise in zone e porte, ognuna delle quali ha un dato indirizzo.

→ **Bus di controllo**: è un insieme di collegamenti il cui scopo è coordinare le attività del sistema; tramite esso, la **CPU** può decidere quale componente deve scrivere sul bus dati in un determinato momento, quale indirizzo leggere sul bus indirizzi, quali celle di memoria devono essere scritte e quali invece lette.

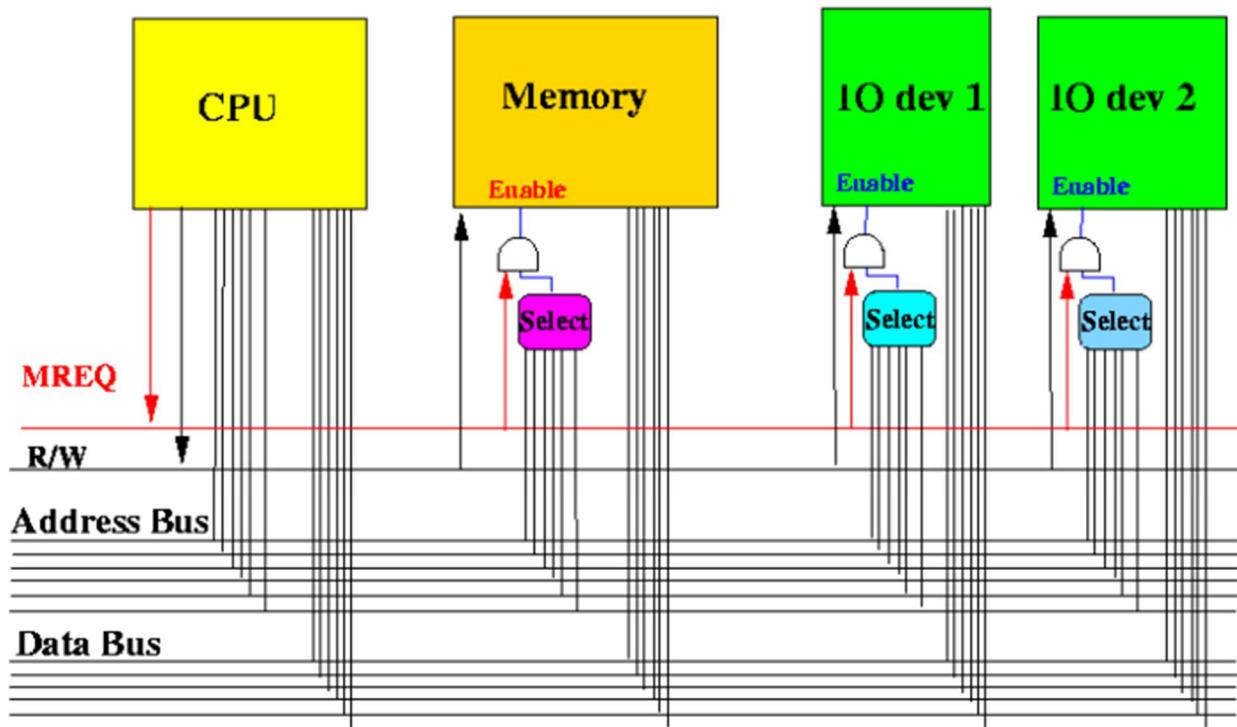
I vantaggi del **bus di sistema** sono elevata flessibilità, semplicità, basso costo e lo svantaggio è la gestione complessa del canale condiviso.

24) Periferica: definizione e caratteristiche

Le **periferiche** sono dispositivi per **I/O** di informazioni collegate alle **CPU** tramite il bus di sistema e/o **interfacce**. Le **interfacce** sono standardizzate per la comunicazione e hanno una componente **hardware** (controllore della periferica) e una componente **software** e contiene registri di dati con un registro di stato associato.

25) Periferica mappata in memoria: definizione e caratteristiche

Una parte della **memoria** riservata al **sistema** viene usata per la comunicazione con le periferiche e ogni periferica ha uno spazio dedicato nella memoria assegnandole un identificatore unico (**indirizzo unico**). I **registri** dell'interfaccia sono mappati in memoria e non sono accessibili da un programma utente. L'**accesso alla periferica** è simile all'accesso alla memoria dal punto di vista della **CPU**.



26) Registri di interfaccia della periferica: definizione e caratteristiche

Il registro di interfaccia della periferica si compone di due registri:

- **registro di stato della periferica**: il quale rappresenta lo stato della periferica e viene letto dalla CPU;
- **registro di dati della periferica**: rappresenta i dati di input o di output in base al tipo della periferica.

27) Descrizione processo di interfaccia I/O

Il processo di **interfaccia I/O** è costituito dalle seguenti fasi:

- la **CPU** interroga lo stato della periferica;
- la periferica risponde il suo stato;
- se la periferica è pronta a trasmettere/ricevere dati, la **CPU** richiede il trasferimento dei dati;
- la **CPU** invia o riceve dati.

Le tecniche di **gestione I/O** sono:

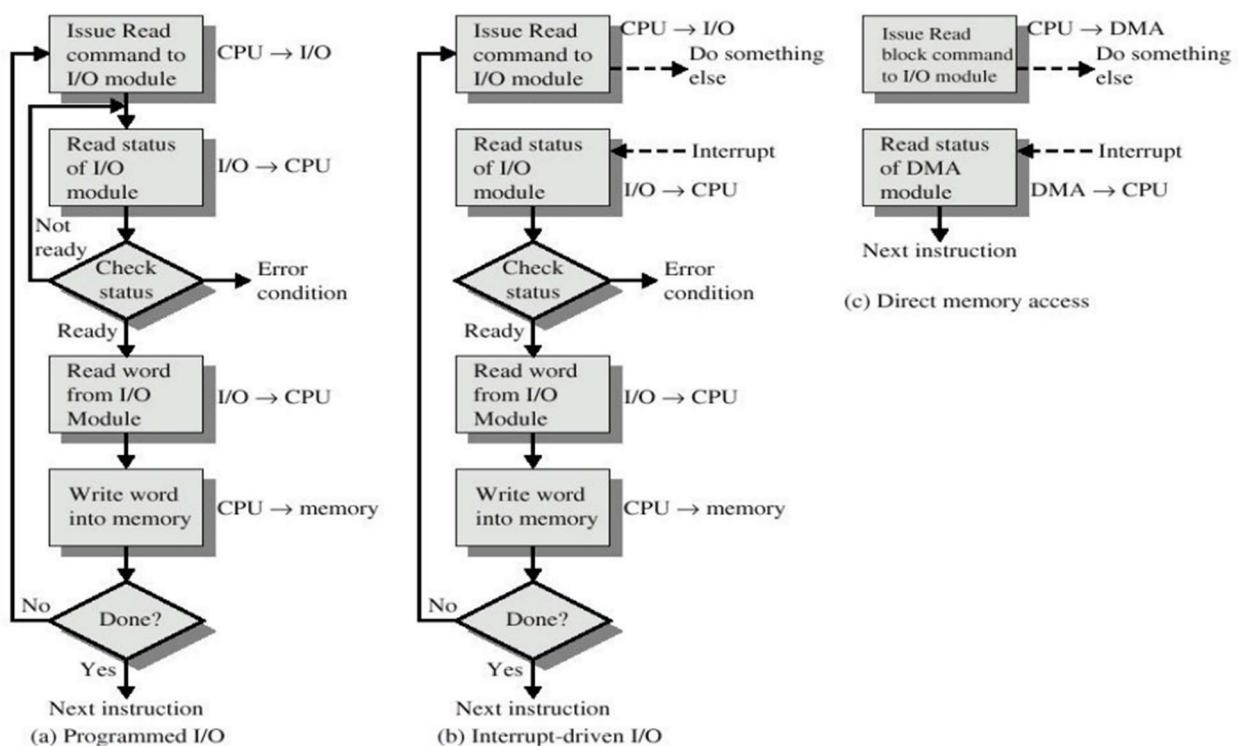
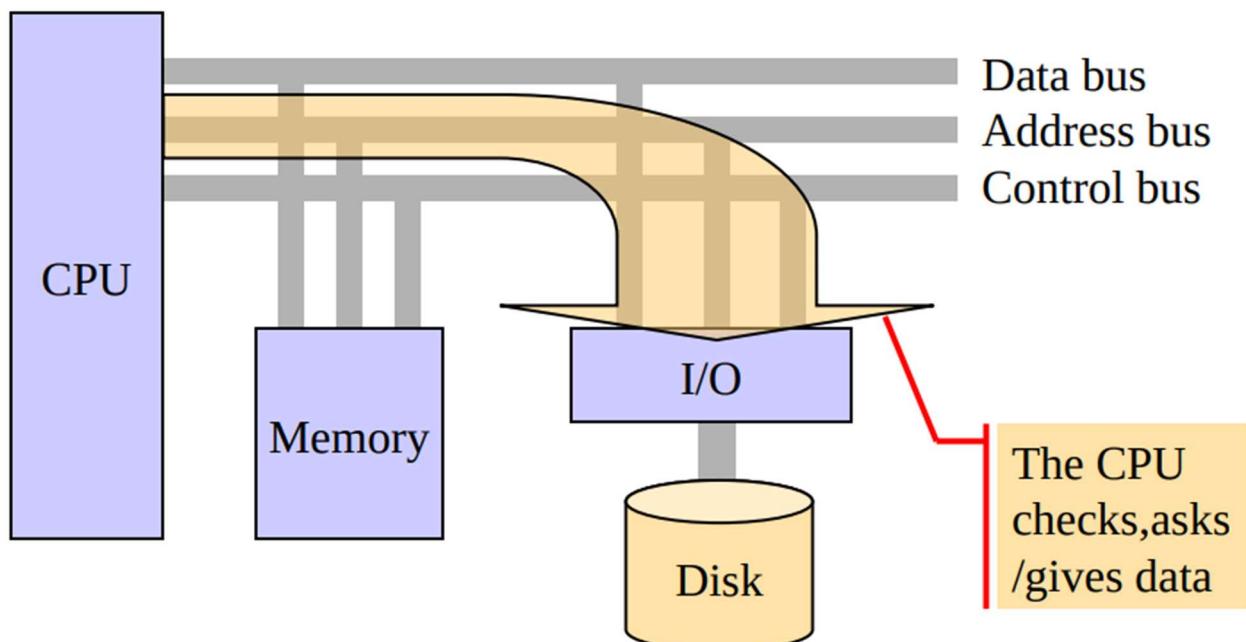
- **I/O gestito da programma (Programmed I/O)**;
- **I/O guidato da interrupt (Interrupt Driven I/O)**;
- **Accesso Diretto alla Memoria (DMA - Direct Memory Access)**.

28) I/O gestito da programma

La tecnica di **I/O gestito da programma** ha le seguenti caratteristiche:

- **I/O gestito dal controllo di programma**;

- la **periferica** ha un ruolo passivo;
- la **CPU** si occupa sia del controllo sia del trasferimento dati;
- la **CPU** predispone il controllore della periferica all'esecuzione dell'I/O;
- la **CPU** si ferma e interroga il registro di stato della periferica in attesa che sia pronta (e.g., ready bit assume un determinato valore);
- **vantaggio**: risposta veloce al ready bit;
- **svantaggio**: la **CPU** bloccata in stato di **busy waiting**.



29) Banda passante e latenza

Banda passante: rappresenta la quantità dei dati che si può trasferire per unità di tempo; rappresenta una misura di flusso;

Latenza: rappresenta il tempo che intercorre tra l'istante in cui una periferica è pronta per il trasferimento e l'istante in cui il dato viene trasferito.

30) I/O guidato da interrupt

La tecnica di **I/O guidato da interrupt** viene gestita nella seguente maniera:

→ La **periferica** segnala alla **CPU** di aver bisogno di attenzione mediante un segnale sul bus di controllo;

→ la **periferica** avvisa la **CPU** attivando un segnale **interrupt request**; quando il processore se ne accorge (in una fase di **fetch**) informa la **periferica** con un segnale **interrupt acknowledge**;

→ la **CPU** interrompe la **CPU** interrompe l'esecuzione del programma corrente (salvando il contesto dell'esecuzione del programma per poter riprendere la sua esecuzione) ed esegue la procedura di risposta all'interrupt;

→ terminata l'esecuzione della procedura di interrupt, la **CPU** riprende l'esecuzione del programma interrotto;

→ il programma utente continua la sua esecuzione.

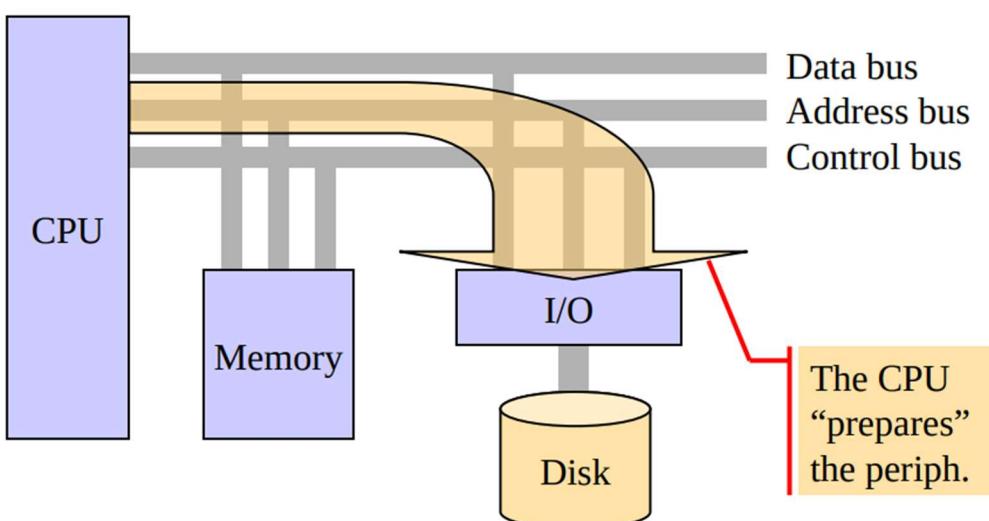
31) I/O guidato da interrupt: vantaggi e svantaggi

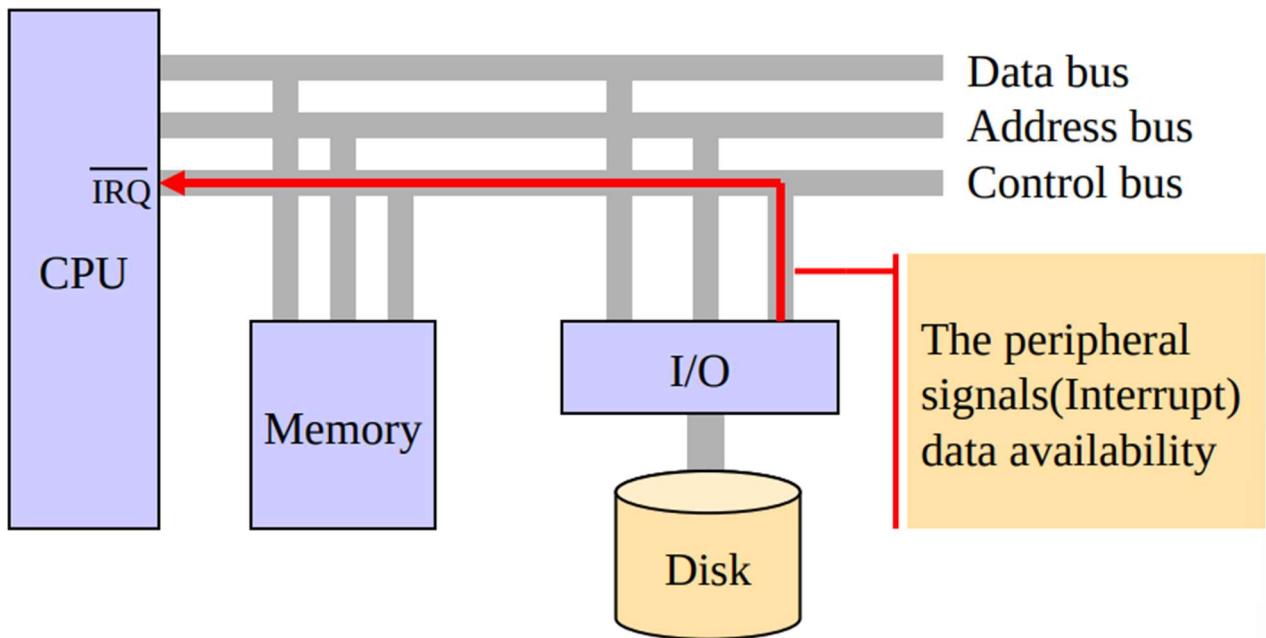
I vantaggi e svantaggi di **I/O guidato da interrupt** sono:

vantaggio: la **CPU** non fa più uso di busy wait;

svantaggio: la **CPU** deve comunque gestire le operazioni di trasferimento.

Per evitare l'intervento della **CPU** durante il trasferimento dei dati è stato ideato un nuovo protocollo chiamato **Direct Memory Access (DMA)**.





32) DMA: Direct Access Memory

La **DMA** è una tecnica utilizzata quando si trasferiscono velocemente grandi quantità di dati e la periferica diventa autonoma nell'accesso alla memoria. In questa tecnica, la periferica gestisce i trasferimenti e la **CPU** non interviene nel trasferimento dei dati.

Le caratteristica principale della **DMA** è che essa necessita di due registri in più per ogni periferica oltre altri registri di stato e registro dei dati:

- un registro che indichi l'indirizzo di memoria da/dove trasferire i dati;
- un registro che indichi la quantità dei dati da trasferire.

Questi due registri sono mappati in memoria e alla fine del trasferimento la periferica invia un interrupt alla **CPU** per segnalare il completamento del trasferimento. Nella **DMA** la banda passante è massima perché la **CPU** non deve eseguire nessuna istruzione e la latenza è minima dato che nessuna istruzione è eseguita dalla **CPU**.

Capitolo 8 - Cache

33) Distinzione tipi di memoria e gerarchie

Esistono altri tipi di **memorie** distinte in base a diversi parametri:

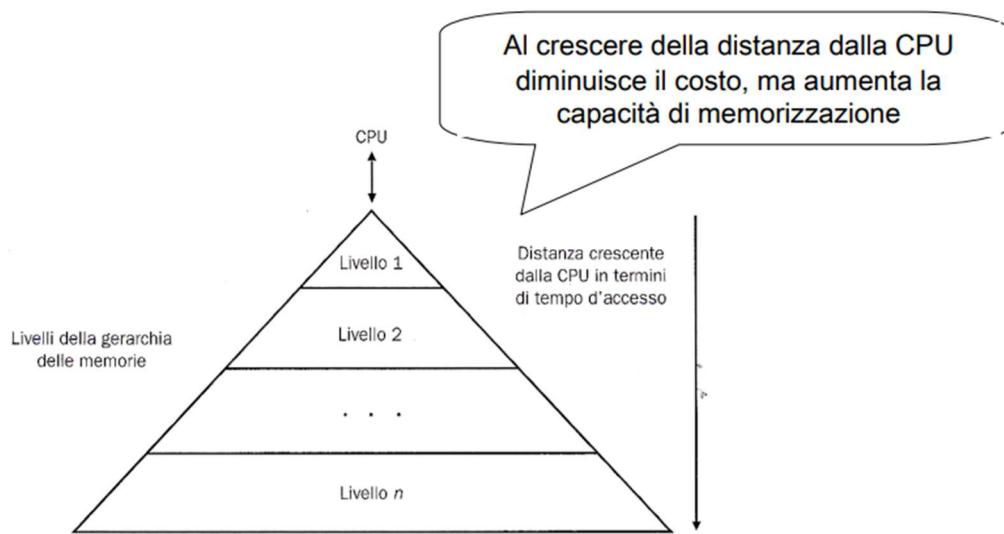
- 1) **dimensione**: quantità di dati memorizzabili;
- 2) **velocità**: intervallo di tempo tra la richiesta del dato e il momento in cui è disponibile;
- 3) **consumo**: potenza assorbita;

4) **costo**: costo per bit.

La gerarchia di memoria è la seguente:

→ memorie piccole, più veloci (e costose) sono poste ai livelli alti;

→ memorie ampie, più lente (e meno costose) sono poste ai livelli più bassi.



→ La **memoria interna** alla **CPU** è costituita dai registri ed è caratterizzata da alta velocità e limitate dimensioni.

→ La **memoria centrale** è caratterizzata da dimensioni molto maggiori della memoria interna alla CPU, ma con tempi di accesso più elevati. È accessibile in modo diretto tramite indirizzi.

→ Le **memorie secondarie** sono ad alta capacità, bassi costi e non volatili.

34) Principio di Località

Il **principio di località** rappresenta la base del comportamento dei programmi in un calcolatore, dato che esso accede soltanto a una porzione relativamente piccola del suo spazio di indirizzamento. Il principio di località viene sfruttato strutturando la memoria in modo gerarchico, ovvero più è veloce, più è vicina al processore, più è costosa e più è grande, più è lontana dal processore, meno costosa. Un livello di memoria più vicino al processore contiene un sottoinsieme di dati memorizzati in ogni livello sottostante e tutti i dati si trovano nel livello più basso.

35) Località spaziale e temporale

Località temporale: afferma che gli elementi ai quali si è fatto riferimento di recente saranno usati ancora nel prossimo futuro;

Località spaziale: afferma invece che gli elementi i cui indirizzi sono vicini ad un dato indirizzo di riferimento tendono ad essere referenziati in tempi molto più brevi.

36) Blocco/Linea: definizione

La più piccola quantità di informazione che può essere presente/assente in una gerarchia di memoria

37) Hit: definizione

L'informazione richiesta dal processore si trova in uno dei blocchi nel livello superiore di memoria

38) Miss: definizione

Il dato non è presente nel livello immediatamente superiore ed occorre accedere al livello più distante.

39) Hit rate: definizione

Frazione degli accessi alla memoria nei quali l'informazione richiesta è stata trovata nel livello superiore di memoria

40) Miss rate: definizione

Frazione degli accessi alla memoria nei quali l'informazione richiesta NON è stata trovata nel livello superiore di memoria ($1 - \text{HitRate}$)

41) Timing hit: definizione

Tempo di accesso al livello superiore della memoria che comprende anche il tempo necessario a stabilire se il tempo di accesso si risolva in un successo o in un fallimento

42) Timing miss: definizione

Il tempo necessario a sostituire un blocco del livello superiore con un nuovo blocco dal livello inferiore della gerarchia, e trasferire i dati di questo blocco al processore

43) Frequenza di hit

Accessi risolti con successo / Numero totale di accessi

44) Frequenza di miss

Accessi risolti senza successo / Numero totale di accessi

45) Cache: definizione

La **cache** è un componente hardware che rappresenta il livello della memoria gerarchica che si trova tra il processore e la memoria principale. L'algoritmo di caching si basa sui principi di località spaziale e temporale e mantiene i dati richiesti recentemente "vicino" alla **CPU (località temporale)** e muove blocchi contigui di memoria che contendono il dato richiesto (**località spaziale**).

46) Tipologie di cache: quali sono i tipi di cache

Le tipologie di cache sono:

- **Direct Mapped**: ciascun blocco della memoria corrisponde una specifica locazione nella cache;
- **Fully Associative**: ogni blocco può essere collocato in qualsiasi locazione della cache e per ricercare un blocco nella cache è necessario cercarlo in tutte le linee della cache (troppo tempo per la ricerca sequenziale).
- **Set Associative**: soluzione intremedia tra direct mapped e fully associative e ciascun blocco della memoria ha a disposizione un numero fisso di locazioni in cache.

47) Direct Mapped Cache

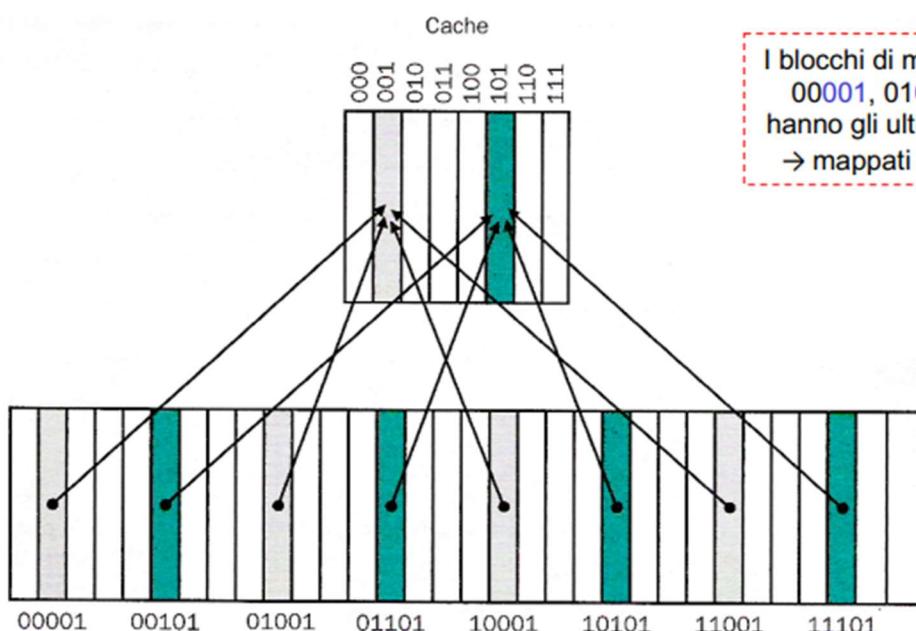
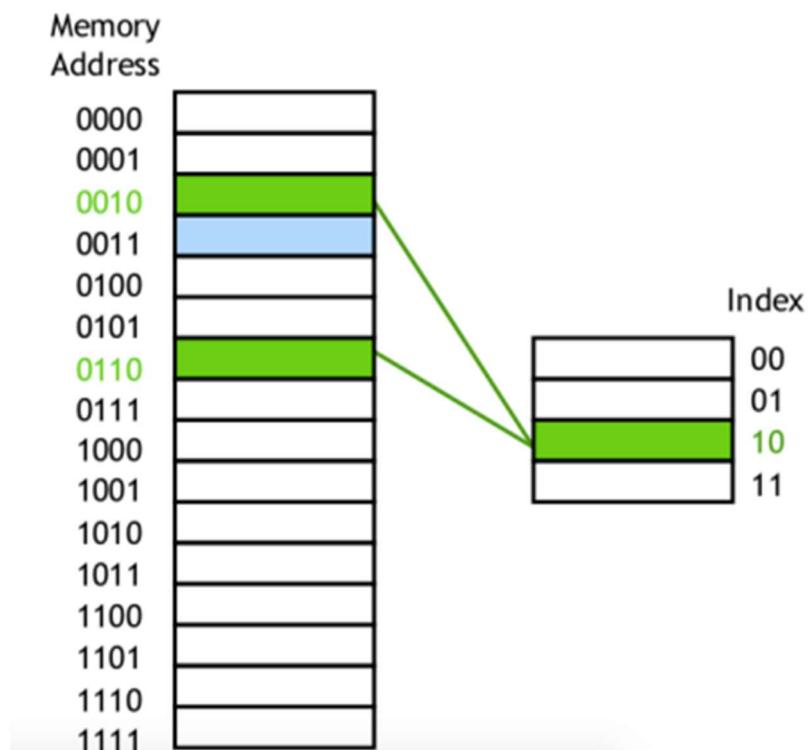
Associa una sola locazione della cache a ogni parola della memoria definendo una corrispondenza tra l'indirizzo in memoria della parola e la locazione nella cache. L'indirizzo è costituito da:

- **Tag (etichetta)**: contiene informazioni necessarie a verificare se una parola della cache corrisponde o meno alla parola cercata;
- **Indice**: utilizzato per selezionare il blocco della cache;
- **Offset**: bit necessari per selezionare il byte richiesto nella parola.

Dato che il numero di blocchi nella cache è una potenza di 2, la posizione corrispondente della parola in cache è data dai $\log_2(\text{numero elementi nella cache})$ bits meno significativi dell'indirizzo in memoria principale.

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

- **bit di validità**: indica se i dati nella linea di cache sono validi;
- **tag**: consente di individuare in modo univoco il blocco in memoria che è stato mappato nella linea di cache;
- **blocco di dati**: formato da una o più word.



Memory (16 parole)

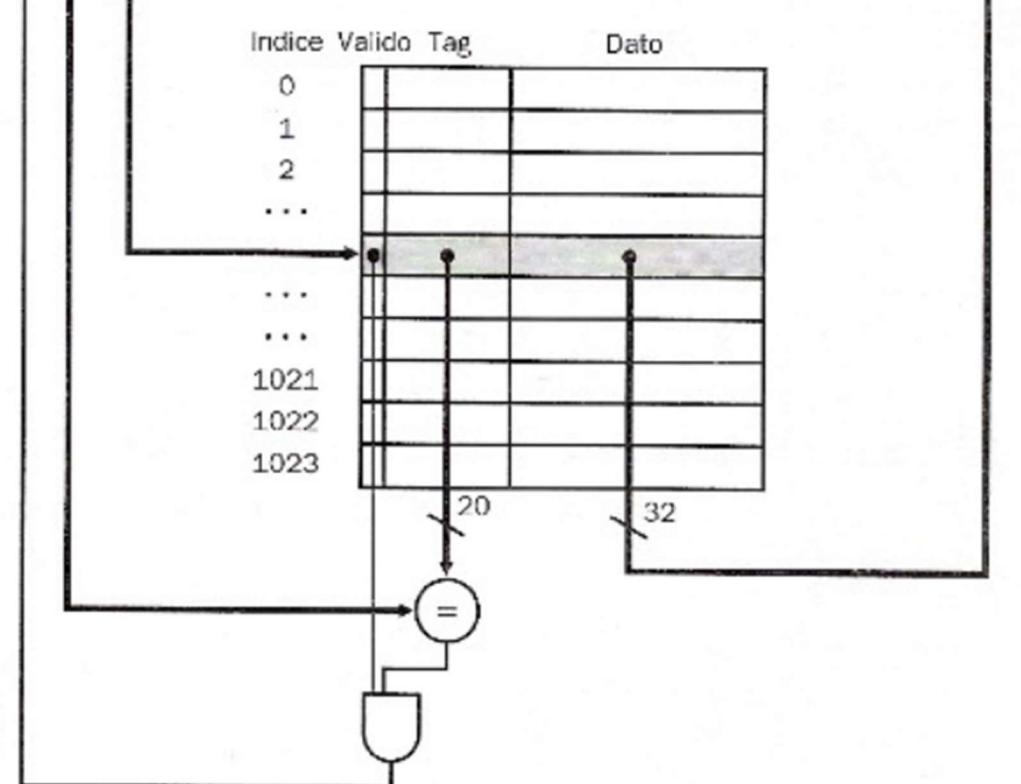
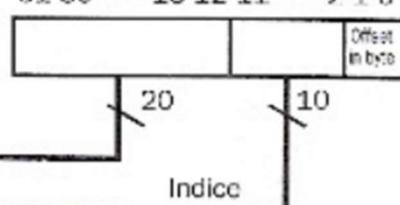
0 (0000)
1 (0001)
2 (0010)
3 (0011)
ccc
4 (0100)
5 (0101)
6 (0110)
7 (0111)
xyz
8 (1000)
9 (1001)
A (1010)
bbb
B (1011)
C (1100)
D (1101)
E (1110)
F (1111)

Direct Mapped Cache (4 parole)

Valid	Tag	Data
1	10	xyz
1	11	aaa
1	10	bbb
1	00	ccc

Indirizzo (con le posizioni dei bit)

31 30 ... 13 12 11 ... 2 1 0



48) Come vengono gestiti in cache hit e miss?

In caso di hit si ha accesso al dato dalla cache dati e accesso all'istruzione dalla cache istruzioni e in caso di miss si ha stallo del processore in attesa di ricevere l'elemento dalla memoria, invio dell'indirizzo al controller della memoria, reperimento dell'elemento dalla memoria, caricamento dell'elemento in cache e ripresa dell'esecuzione.

49) Come riconoscere l'indirizzo?

L'indirizzo viene identificato nella seguente maniera:

→ bit meno significativi dell'indirizzo del dato richiesto indicano la **posizione del dato nella cache**;

→ bit più significativi dell'indirizzo del dato richiesto rappresentano il **tag**.

Indirizzo: 10110

Tag = 10

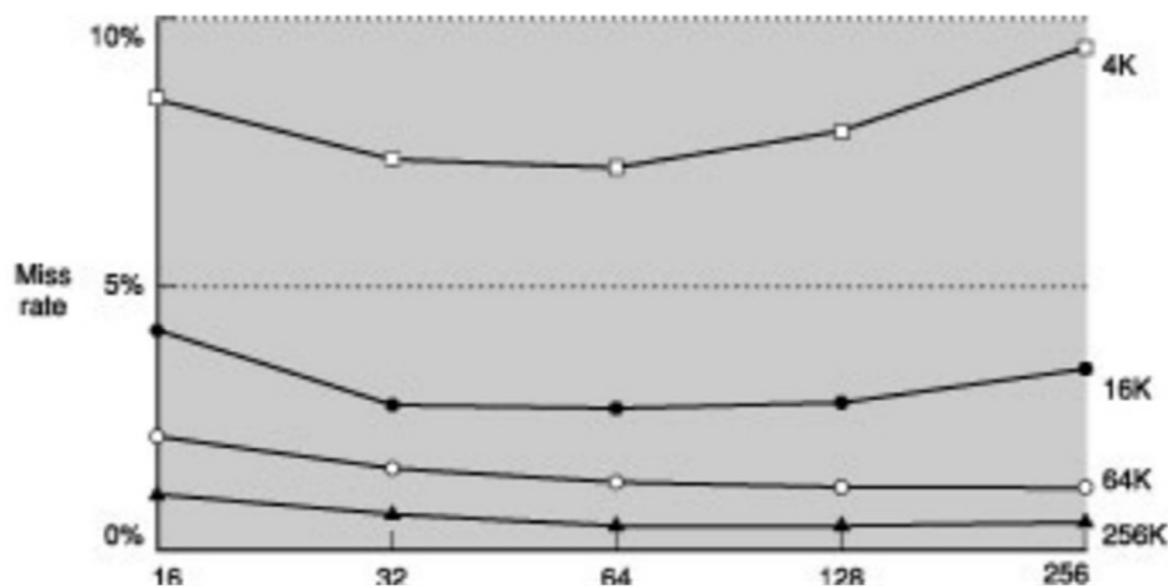
Indice = 110

50) Come viene ricavato l'indirizzo del blocco?

Mediante la seguente formula:

$$\text{indirizzo del blocco} = \frac{\text{Indirizzo del dato in byte}}{\text{Byte per blocco}}$$

La dimensione del blocco di cache è in stretta relazione con la frequenza miss



51) Miss penalty: definizione

La **miss penalty** è la differenza tra il tempo di accesso al livello inferiore e il tempo di accesso alla cache. Un'ampia dimensione per il blocco permette di

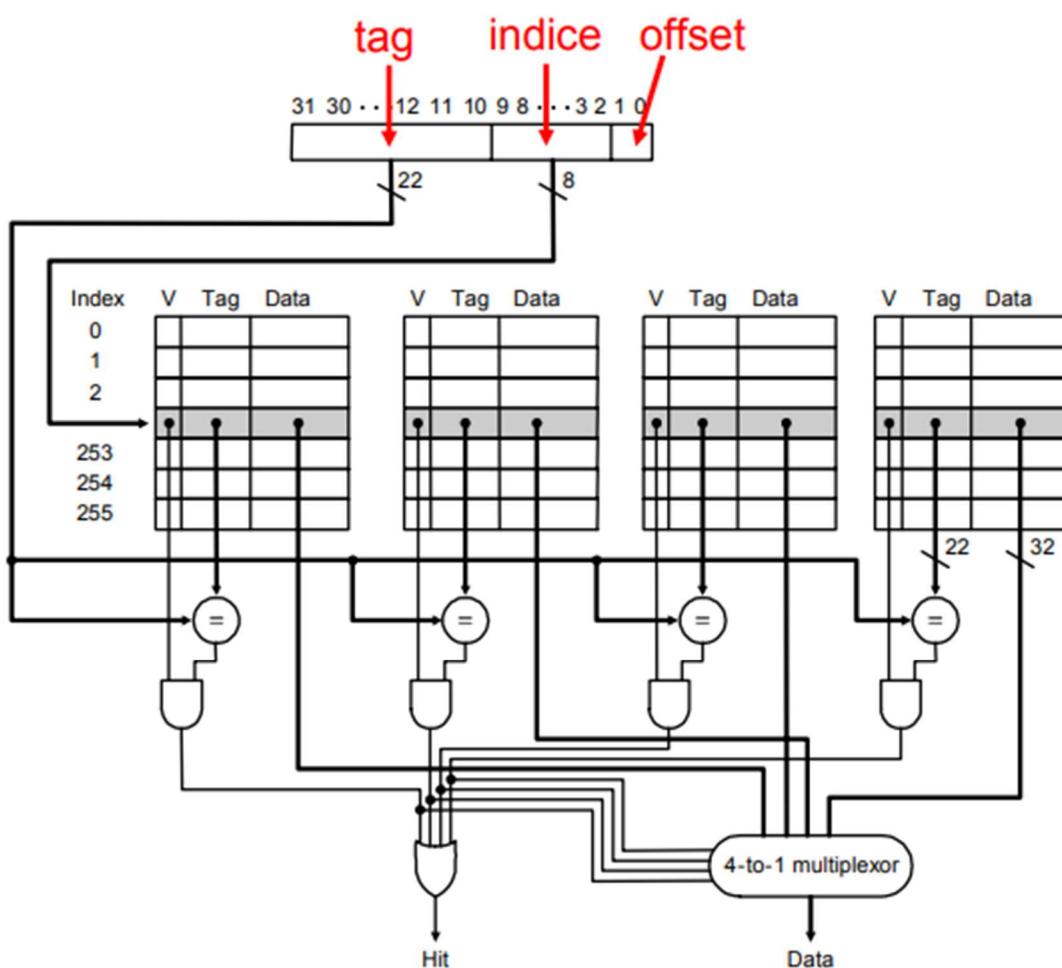
sfruttare la località spaziale, ma blocchi di grossa dimensione comportano maggiori miss penalty ed è necessario più tempo per trasferire il blocco. Se la dimensione del blocco è troppo grande rispetto alla dimensione della cache, il miss rate aumenta.

51) Caratteristiche: Cache set associative

Le caratteristiche della cache set associative sono:

- i blocchi appartenenti alla cache sono raggruppati in set;
- in una cache set associative ad **N vie** (N-way set associative) ogni set raggruppa N blocchi;
- ogni indirizzo di memoria corrisponde ad un unico set della cache (accesso diretto tramite 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 i tag di tutti i blocchi.

Cache a 4 vie



Gli svantaggi della **cache set associative** sono:

- possibili ritardo fornito dal **multiplexer**;
- il blocco è disponibile dopo la decisione **Hit/Miss** e la selezione del **set** rispetto alla cache ad accesso diretto;
- maggior costo implementativo.

Il vantaggio dell'utilizzo della cache set associative è la diminuzione della **miss rate**.

La scelta tra cache ad indirizzamento diretto, set - associative e completamente associativa dipende dal costo dell'associatività rispetto alla riduzione del **miss rate**.

52) Quale blocco sostituire in caso di (capacity) miss nelle cache set- o fully-associative?

Nella cache ad accesso diretto: se il blocco di memoria è mappato in una linea di cache già occupata (conflict miss), si elimina il contenuto precedente della linea e si rimpiazza con il nuovo blocco. In caso di cache completamente associativa, ogni blocco è un potenziale candidato per la sostituzione e in caso di cache set-associativa a **N vie**, bisogna scegliere tra gli **N blocchi** del set.

52) Algoritmi di sostituzione a blocchi

Si hanno tre tipi di algoritmi a sostituzione a blocchi:

- Politica di sostituzione **Random**: scelta casuale;
- **Least Recently Used (LRU)**: si sfrutta la località temporale, il blocco sostituito è quello che non si utilizza da più tempo, e 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;
- **First In First Out (FIFO)**: si approssima la strategia LRU selezionando il blocco più vecchio anziché quello non usato da più tempo.

53) Come vengono gestite le miss in lettura?

Le miss in lettura vengono gestite nella seguente maniera:

- se un blocco non è presente nella cache bisogna mettere in stallo l'intera **CPU**;
- al verificarsi dei miss, vengono eseguiti i seguenti passi:

1. inviare PC alla memoria;
2. lettura dalla memoria;
3. scrittura nella cache;
4. riavvio l'esecuzione dell'istruzione che ha causato il miss.

54) Accesso in scrittura: problema

La scrittura di un dato nella cache significa creare un'incoerenza, se non si aggiornano i livelli inferiori della gerarchia di memorie. Si hanno due tecniche risolutive:

- write - through;
- write - back;
- utilizzo di un write buffer.

Index	V	Tag	Data
...
110	1	11010	42803
...

Mem[1101 0110] = 21763

55) Write - through

Tecnica risolutiva in cui i dati sono scritti nel blocco della cache e nel blocco del livello inferiore.

I vantaggi del write through sono:

- soluzione più semplice da implementare;
- si mantiene la corenza delle informazioni nella gerarchia di memorie.

Gli svantaggi sono:

- le operazioni di scrittura vengono effettuate alla velocità della memoria di livello inferiore;
- aumenta il traffico sul bus di sistema.

56) Write - back

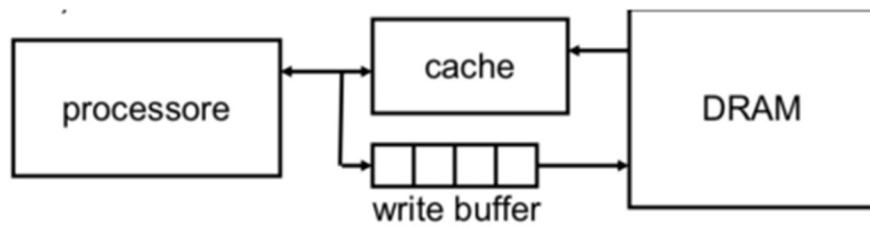
Tecnica risolutiva in cui i dati sono scritti nel blocco della cache. Il blocco modificato viene scritto nel livello inferiore della gerarchia solo quando deve essere sostituito (subito dopo la scrittura, cache e memoria di livello inferiore sono inconsistenti e viene mantenuto il dirty bit).

I vantaggi sono:

- le scritture avvengono alla velocità della cache;
- scritture successive sullo stesso blocco alterano solo la cache;

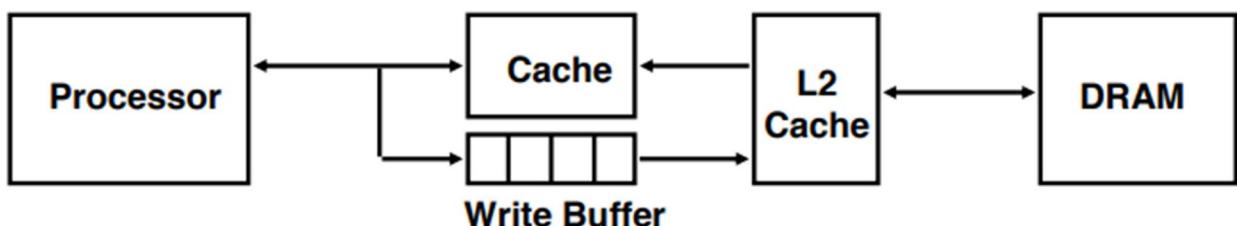
Gli svantaggi sono:

- ogni sostituzione del blocco può provocare un trasferimento in memoria.



57) Write - buffer

Interposto tra la memoria di livello inferiore e la cache, in cui il processore scrive i dati nella cache e nel write buffer e il controller della memoria scrive il contenuto del write buffer in memoria. Esso è gestito in modalità FIFO:



58) Write miss

La **write miss** consiste nella scrittura di un indirizzo che non è contenuto in cache. Le scritture possono indurre write miss e le soluzioni possibili sono:

- **Write Allocate**: il blocco viene caricato in cache e si effettua la scrittura;
- **No - write allocate**: il blocco viene scritto direttamente nella memoria di livello inferiore, senza essere trasferito in cache. Si hanno le seguenti combinazioni:
 - **write back con write allocate**;
 - **write through con write not allocate**.