

211

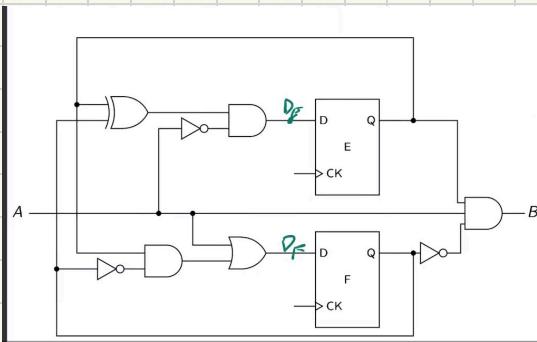
SINTESI >

ES: (continua)

(3) Ricavo esp. booleane per i componenti

$$\begin{aligned}B &= EFA \quad \rightarrow \text{output} \\D_E &= \overline{EFA} + EF\bar{A} = \\&= (\overline{E}F + E\overline{F})\bar{A} \\&= (E \oplus F)\bar{A} \quad \rightarrow \text{Input flip flop 1} \\D_F &= \overline{E}\overline{F}A + \overline{E}FA + E\overline{F}\bar{A} + E\overline{F}A + EFA \\&= \overline{E}\overline{F}A + \overline{E}FA + E\overline{F}A + EFA + E\overline{F}\bar{A} \\&= (\overline{E}\overline{F} + \overline{E}F + E\overline{F} + EF)A + E\overline{F}\bar{A} \\&= A + E\overline{F} \\&= (A + E\overline{F})(A + \overline{A}) \\&= A + E\overline{F} \quad \rightarrow \text{Input flip flop 2}\end{aligned}$$

4) Realizzazione circuito



## REALIZZAZIONE CPU

Per poter consentire il data path (Processo di elab. del dato attraverso l'ALU) bisogna realizzare un circuito che realizzi le operazioni di Fetch, Decode ed Execute.

• **FETCH**

- Si ricava dal Program Counter l'indirizzo della prossima istr.
- Usando l'indirizzo, si ricava l'istruzione vera e propria dalla memoria primaria
- Si salva l'istruzione nell'Instruction Registry

Data Path RISC-V (Semplificato)

- (fetch)
- Il PC fornisce un indirizzo dalla memoria delle istruzioni
  - Per semplificare, assumiamo che la "memoria delle istruzioni" sia "read-only"
  - Assumiamo anche che ci siano 2 memorie diverse:
  - Una per le istruzioni ed un'altra per i dati.



SP → 0000 0000 1EEE FFFF  
Dati dinamici  
Dati statici  
Riservato

• **DECODE**

- La control unit decodifica il tipo di istruzione
- I dati necessari vengono prelevati dalla memoria degli indirizzi.
- Si attiva l'ALU con gli operandi

Data Path RISC-V (Semplificato)

- (decode)
- La ALU deve "capire" il tipo di istruzione
  - Per esempio, il numero d'ordine dei registratori contenenti gli operandi può essere letto nei campi opportuni dell'istruzione stessa
  - L'unità di controllo (non mostrata nella figura) è la parte che effettivamente decodifica le istruzioni

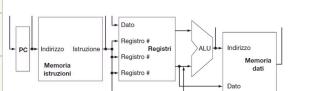
• **EXECUTE**

In base al tipo di istruzione si può:

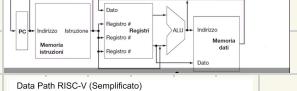
- Eseguire un'operazione logico-aritmetica → il risultato va salvato nel registro
- Eseguire una load: il risultato serve l'indirizzo da dove prendere i dati
- Eseguire una store: i dati di 1 dei 2 operandi verrà messo come dato

Data Path RISC-V (Semplificato)

- (execute)
- Una volta caricati gli operandi, si può:
  - eseguire il calcolo effettivo (operazioni aritmetico-logiche su interi)
  - elaborare il risultato per determinare un indirizzo di memoria (caso di load o store)
  - eseguire un confronto (salto condizionato)



Data Path RISC-V (Semplificato)



L12

## Data Path

2

# REALIZZAZIONE CPU > PROGRAM

Nel frattempo che l'istruzione viene eseguita, all'indirizzo corrente viene sommato un **Full Adder**

Questo per fare in modo che,  
al prossimo ciclo di clock ci  
sia la prossima istruzione.

**DATAPATH > ISTRUZIONI TIPO "R"**

Il dottor-path delle istruzioni di tipo R prevede di

- **FETCE**: istruzione + source 2e
  - **DECODE**: Si prendono i registri su cui eseguire l'op.
  - **EXECUTE**: L' ALU processa il risultato su un nuovo registro o sugli esistenti

The diagram illustrates the internal structure of a processor. It starts with a PC (Program Counter) at the bottom left, which feeds into an 'Indirizzo' (Address) register. This address is used to access a 'Memoria Istruzioni' (Instruction Memory). The 'Istruzione' (Instruction) is read from memory and sent to a 'DECODE' block. Simultaneously, the 'Indirizzo' is also sent to a 'Registratore #2' (Register #2). The 'DECODE' block outputs control signals to three 'Registri' (Registers) labeled 'X1', 'X2', and 'X3'. These registers receive their data from 'Registratore #1', 'Registratore #2', and 'Registratore #3' respectively. The outputs of 'X1', 'X2', and 'X3' are fed into an 'ALU' (Arithmetic Logic Unit). The 'ALU' has two outputs: one goes to a 'Registratore #4' (Register #4), and the other is labeled 'X2+X3'. A feedback loop from 'Registratore #4' connects back to 'Registratore #2'. A blue box highlights the 'ALU' and its connections. A handwritten note next to the ALU says 'Lleg Write=1' with an asterisk.

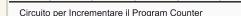
\* Il valore per scrivere nei registri, in queste istruzioni il valore di esso si stabilizzerà dalla control unit

DATA PATH ➤ ISTRUZIONI TIPO "I" (LOAD) e "S" (store)

## • LOAD

Letta l'istruzione si legge  
l'indirizzo su cui mem.in RS2  
al quale si sottoscrive l'immediata  
Il dato trovato viene poi ri-salvato  
in RS3.

## COUNTER

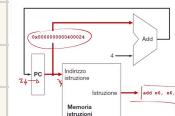


- Istruzione



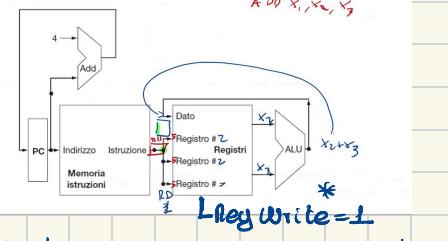
Memoria Istruzioni

  - Circuito per prelevare le istruzioni incrementare il program counter
  - L'istruzione viene utilizzata dall'unità di elaborazione

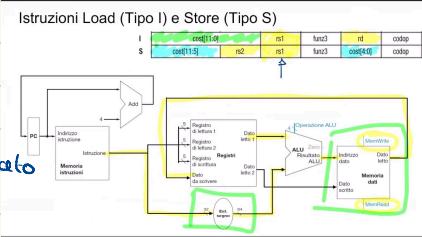


888000000020 888 888 888 88  
888000000024 888 888 888 88

## Datapath per le Istruzioni di Tipo F



DATA PATH ➤ ISTRUZIONI TIPO "I" (LOAD) e "S" (store)

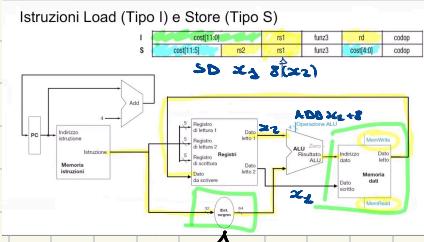


## DATA PATH > ISTRUZIONI TIPO "I" (LOAD) e "S" (store)

- STORE

Letta l'istruzione si legge dal 1° registro di lettura e va nel Data letto 2 che andrà in scrittura sulla memoria

Ne vengono selezionati 32 per considerarne le possibili posizioni

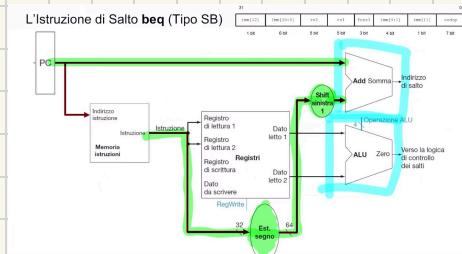


L'estensione del segno serve per estendere il bit per l'ALU  
(l'offset può essere positivo o negativo)

## DATAPATH > ISTRUZIONI TIPO "SB"

- Si prendono i dati dei 2 registri [Eq - a-b] se = 0 Allora si fa il salto

- Per calcolare l'offset:
  - Si estende il segno
  - Si fa lo shift di 1 bit perché puri
  - Si somma al PC



## DATAPATH > ARCHITETTURA UNIFICATA

Per gestire tutte e 3 le tipologie si unifica il circuito.



## DATA PATH > REALIZZAZIONE CONTROL UNIT

La control unit nell'unità di elaborazione.  
dovrà valorizzare i bit di controllo  
a seconda dell'istruzione letta.

### • FASE DI DECODE

Letta l'istruzione, i bit di

- cod op
- funz 3 se c'è
- funz 7 se c'è

Vengono letti dalla control unit per identificare il tipo di istruzione.

→ Per parallelizzare

L'implementazione effettiva viene gestita con

- I ALU Principale che valorizzerà i bit di controllo e l'ALU secondaria
- I ALU Secondaria che valorizzerà il selettore di operazione per l'ALU.

## CONTROL UNIT > ARCHITETTURA

### • C.U. PRINCIPALE

- Inputs: cod op → Dall'istruzione

- Output:

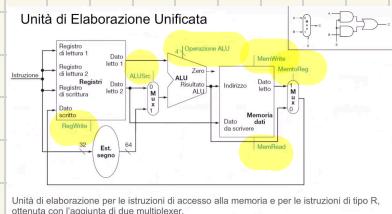
- I bit di controllo

- L'ALU op, ovvero il tipo di operazione da dare in input all'ALU secondaria.

00 : per load e store

01 : per salti condizionati

10 : per istruzioni R



Unità di elaborazione per le istruzioni di accesso alla memoria e per le istruzioni di tipo R, ottenuta con l'aggiunta di due multiplexer.

## CONTROL UNIT &gt; ARCHITETTURA

## • C.U. SECONDARIA

- Inputs:

- + ALUop : Da C.U. principale

- + funz3 / Funz7 : Da istruzione

- Outputs

- + I bit di controllo per l'ALU principale

Perché non c'è  $\perp$ ?

Perché le operazioni definite in macro-categorie:

- 00  $\rightarrow$  Sia che load che store fanno add

- 01  $\rightarrow$  Il branch deve sottrarre

- 10  $\rightarrow$  Il codop è lo stesso ma variano le funz.



## CONTROL UNIT &gt; BIT DI CONTROLLO

## • REGWRITE :

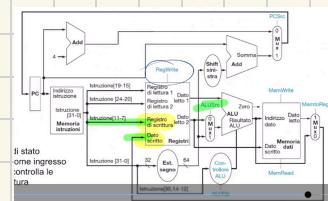
- ASERITO: Abilita la scrittura nellregistro specificato nel blocco di registri del dato proveniente dal  $\perp$

- NON ASERITO: Non scrive nei reggisti

## • ALU SRC

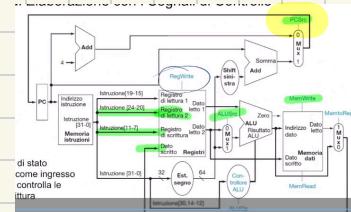
- ASERITO: Imposta il MUX a 1 ovvero imposta come  $2^{\text{a}}$  parametro dell'ALU l'offset (usato da beq)

- NON ASERITO: Imposta il MUX a 0 ovvero imposta come  $2^{\text{a}}$  parametro dell'ALU il valore preso dal 2<sup>o</sup> registro



## CONTROL UNIT ➤ BIT DI CONTROLLO

- PC SRC  $\rightarrow$  se beq  $\wedge x_1 - x_2 = 0$ 
  - ASERITO: MUX=1 (beq)
  - Somma al Program Counter l'offset
  - NON ASERITO: MUX=0
  - Somma il valore standard al P.C.



## • MEMSTORE

- ASERITO: Abilita della scrittura in memoria (store)

- N. ASERITO: Non abilita scrittura

## • MEM READ:

- ASERITO: Abilita della lettura in memoria (load)

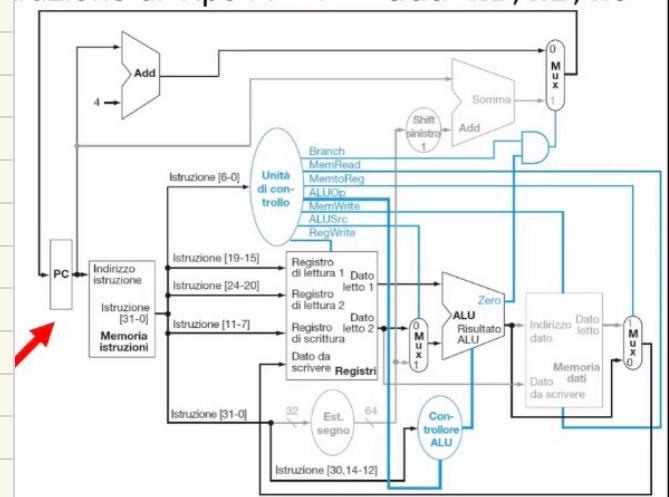
- N. ASERITO: Non abilita scrittura

## • MEM TO REG

- 1** - ASERITO: Salva il dato dalla mem. nel registro  
**0** - N. ASERITO: // // dato dall'ALU // //

## CONTROL UNIT ➤

Schema di implementazione:



## DEFINIZIONE ➤

È un sistema di comunicazione tra più elementi.

Viene definito da un protocollo di comunicazione

## BUS ➤ TIPOLOGIE DI DISPOSITIVI

Nella comunicazione nei bus, i dispositivi possono essere:

## • MASTER

Se fanno **partire** la comunicazione. Di solito hanno un chip chiamato Bus Driver

## • SLAVE

Se **subiscono** la comunicazione. Hanno il chip Bus Receiver

## • ENTRAMBI

Se sono sia master che slave. Hanno il chip Bus Transceiver.

Master	Slave	Example
CPU	Memory	Fetching instructions and data
CPU	I/O device	Initiating data transfer
CPU	Coprocessor	CPU handing instruction off to coprocessor
I/O	Memory	DMA (Direct Memory Access)
Coprocessor	CPU	Coprocessor fetching operands from CPU

Architettura degli Elaboratori

La CPU è **sempre master**

8

## BUS ➤ PONTE CUIANE DI PROTEZIONE

- LARGHEZZA: La capacità del Bus → Aumenta velocità
- ARBITRAZIO: Delle strategie per orchestrare le comunicazioni senza contaminare
- FUNZIONAMENTO: L'implementazione.

# L 14 • Bus

2

## BUS > TIPOLOGIE

### • SINCRONI

Usano un clock per regolare le operazioni.

Il bus avrà cicli pari al clock.

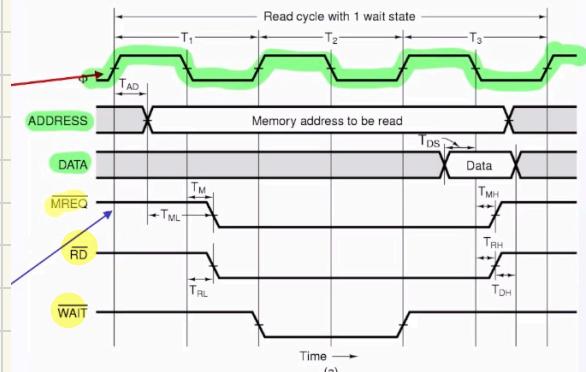
### • ASINCRONI

Basano i cicli del bus in funzione degli elementi coinvolti.  
Quindi varieranno per coppie diverse.

## BUS > SINCRONO : LETTURA DATI

I dati vengono letti usando il seguente schema:

- $\phi$ : Segnale di clock
- ADDRESS: Indica l'area di memoria pronta per essere trasferita (bianco) contenente l'indirizzo
- DATA: I dati da leggere.
- MREQ: Indica che si interagisce con la memoria
- RD: Indica se si fa lettura / scrittura
- WAIT: Indica di aspettare prima di proseguire perché è in corso un accesso alla memoria.



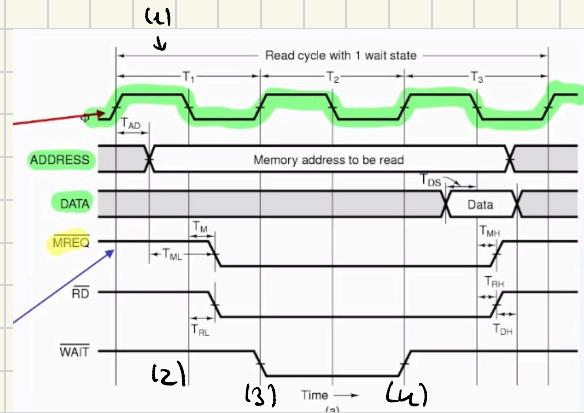
## BUS > Sincrono : lettura dati

1. Quando il clock è asserito (i<sub>ad</sub>) viene preso l'indirizzo del dato.

- 2o Quando il fronte  
di discesa di T1  
è assestito, si  
altera memreq ed  
rd. (lettura in menu)

- 3o Nel fronte di salita di T2 il WAIT si asserisce indicando l'operazione di accesso

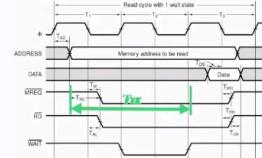
4. Nel fronte di Scritta di T<sub>3</sub> viene segnalata la fine di accesso della memoria **nonostante** i dati non siano ancora pronti.  
Va bene comunque perché i flip flop li rendevano pronti nel prossimo fronte.



### Bus sincrono – Ciclo di lettura

Qual è il tempo massimo che la memoria ha a disposizione per produrre i dati dal momento in cui appare l'indirizzo?

$$T_{max} = T_1 + T_2 + T_3 / 2 - T_{DS} - T_{AD}$$

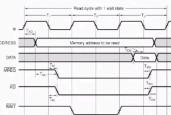


Simbolo	Parametro	Min	Max	Unità
T <sub>AT</sub>	Ritardo dell'indirizzo rispetto al clock	4	8	ns
T <sub>MEQ</sub>	Indirizzo stanco prima di MREQ	2	8	ns
T <sub>RD</sub>	Ritardo di MREQ dal fronte di discesa di φ	3	8	ns
T <sub>RD2</sub>	Ritardo di RD dal fronte di discesa di φ	3	8	ns
T <sub>ASD</sub>	Tempo di assestamento dati	2	8	ns
T <sub>MEQ2</sub>	Ritardo di MREQ dal fronte di discesa di φ	3	8	ns
T <sub>RD3</sub>	Ritardo di RD dal fronte di discesa di φ o di T <sub>3</sub>	3	8	ns
T <sub>TCM</sub>	Tempo di hold dei dati dalla Neozazione di RD	0	8	ns

*Architettura degli Elaboratori*

## Bus sincrono – Ciclo di lettura

Nella specifica di temporizzazione occorre tenere conto di alcuni parametri temporali



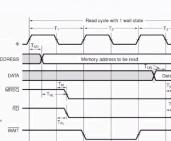
**T<sub>AD</sub>**: intervallo di tempo tra il fronte di salita del clock e l'istante in cui sono valide le linee degli indirizzi (max)

validare le linee degli indirizzi prima dell'effettiva segnalazione di MREQ (min)

**W<sub>RL</sub>**: intervallo di tempo tra il fronte di discesa del clock e il fronte di discesa

**T<sub>PS</sub>**: tempo di setup per le linee dati prima di

Con un ciclo di bus di 30 nsec. quanto tempo ha la RAM per inserire il dato sul bus *dopo che il segnale MREQ ha assunto il valore 0?*



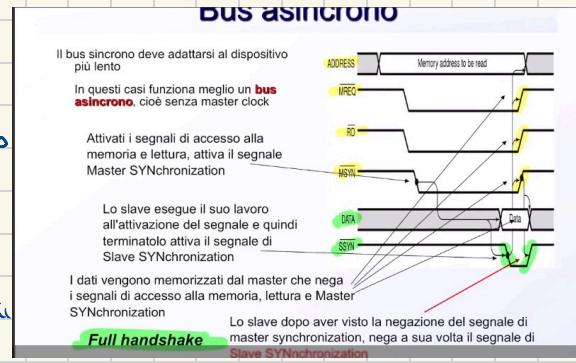
Simbolo	Parametro	Min	Max
T <sub>AD</sub>	Ritardo dell'indirizzo rispetto ai clock	4	10
T <sub>AM</sub>	Indirizzo stabile prima di MREQ	2	4
T <sub>M</sub>	Ritardo di MREQ dal fronte di discesa di $\phi$	3	5
T <sub>RD</sub>	Ritardo di RD dal fronte di discesa di $\phi$	3	5
T <sub>TS</sub>	Tempo di assestamento dati	2	4
T <sub>MH</sub>	Ritardo di MREQ dal fronte di discesa di $\phi$	3	5

## BUS &gt; ASINCRONO: LETTURA DATI

Al posto del clock sono presenti

- **MSYN:** Impostato dal master stesso quando i dati sono stabili
- **SSYN:** Impostato dallo slave quando i dati sono disponibili

fai da notifica al master.



## Sincrono vs Asincrono

## Bus sincrono

- Vantaggi
  - Realizzazione slave semplice
  - Se la durata di un'operazione è fissa non occorre una linea di wait
- Svantaggi
  - Durata di una operazione di comunicazione dove necessariamente avrà una durata pari ad un numero intero di cicli

## Bus asincrono

- Vantaggi
  - È possibile: la durata di una operazione è determinata unicamente dalla velocità della coppia master/slave
- Svantaggi
  - Per completare una operazione di comunicazione sono sempre necessarie 4 azioni
  - Occorre inserire negli slave i circuiti necessari a rispondere opportunamente al protocollo

## BUS &gt; ARBITRAGGIO DEL BUS

## • CENTRALIZZATO

Un arbitro esterno

- riceve: in 1 solo linea (Bus req) se 2 o più elementi richiedono il Bus
- restituisce: Bus grant. Viene fornito un grant, ovvero un segnale di OK agli elementi

## • DECENTRALIZZATO

Si usa un segnale di Busy per indicare che si è occupati



Arbitro centralizzato quando l'arbitro "vede" una richiesta attiva la linea di grant del bus

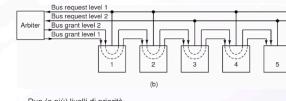
Daisy chaining: il grant viene trasmesso lungo la linea bus grant finché un dispositivo non accetta l'assegnamento

• Vince il dispositivo più vicino

**S può potenzialmente non essere selezionato**

**Fiss:** Coda con priorità Diversa

**Arbitraggio del bus**

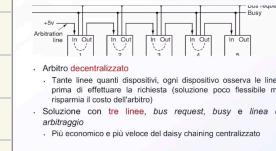


- Due (o più) livelli di priorità
- L'assegnamento segue lo stesso meccanismo del daisy chaining ma di dispositivi con priorità più alta hanno precedenza nell'assegnamento del grant da parte dell'arbitro

• Linea di acknowledge

Linea di acknowledge

## Arbitraggio del bus



- Arbitro controlla direttamente gli elementi
- Tutte le linee su cui i dispositivi osservano le linee prima di effettuare la richiesta (soluzione poco flessibile ma risparmia il costo dell'arbitro)
- Soluzione con tre linee, bus request, busy e linea di arbitraggio
- Più economico e più veloce del daisy chaining centralizzato

## GERARCHEIA DELLE MEMORIE ➤

In base alle loro performance / costo le tipologie di memorie sono così organizzate:

- Più è in alto:

- maggiore è il costo.
- minore è la quantità

- Più è in basso:

- minore è il costo
- maggiore è la quantità



## MEMORIA CACHE ➤ INTRO

È una memoria (solitamente inserita all'interno della CPU) che fa da buffer tra CPU e memoria.

Essa si basa sui principi di

- LOCALITÀ TEMPORALE

Se si accede ad un'area di memoria è probabile che dopo si dovrà di nuovo accedervi

- ES Istruzioni di salto: accedono alla mem. di indirizzi

- // di accesso: salvo dato (es. controllore)

- LOCALITÀ SPAZIALE

Se si accede ad un dato, è probabile che si accede a dati "vicini"

```

addi sp,sp,-10
sd ra,0(sp)
sd a0,8(sp)
addi a0,a0,-1
jal fib
ld t0,8(sp)
sd a0,8(sp)
addi a0,t0,-2
jal fib
ld t0,8(sp)
add a0,a0,t0

```

## CACHE ➤ FUNZIONAMENTO

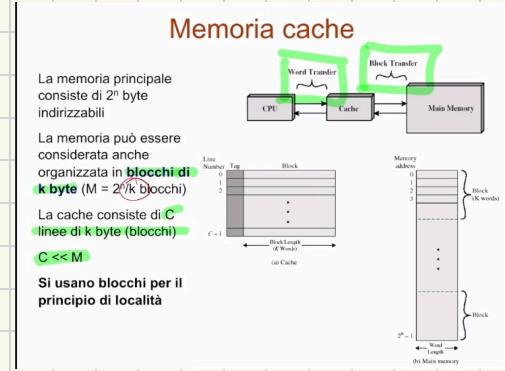
La memoria è organizzata in  $2^n$  righe.

Esse sono partizionabili in blocchi

La cache sarà organizzata in righe, in cui ogni riga contiene un blocco

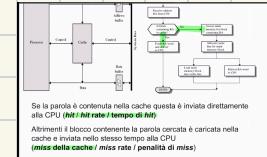
La comunicazione servirà quindi

- CPU  $\leftrightarrow$  CACHE: Tramite word  
Se il dato cercato esiste:
  - si parla di Hit
  - si parla di Miss
- CACHE  $\leftrightarrow$  MEMORIA: Tramite blocchi



Se scelgo cache

- troppo piccola  
Troppi accessi
- troppo grande  
Non dinamica,



## CACHE ➤ MAPPATURA: DIRETTA

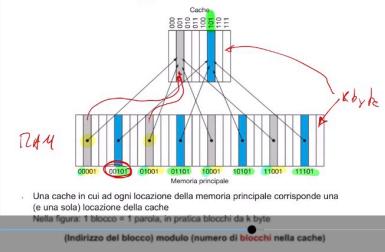
Prevede che gli indirizzi della cache guidino quelli indirizzi venuti in un blocco e non altri.

In particolare, guida il modulo ovvero gli  $N$  bit finali identificano l'appartenenza al blocco

I primi bit contraranno invece il Tag ovvero dei bit per identificare l'elemento nel blocco.

Infine viene aggiunto un bit di validità per capire se il dato è valido o no

## Mappatura diretta



## CACHE &gt; MAPPATURA: DIRETTA (CASO DI USO)

## Mappatura diretta

Un esempio

Cache a mappatura diretta di otto blocchi

I tre bit meno significativi indicano il numero della linea

Elenco di nove richieste

Indirizzo decimale dell'indirizzo memoria principale	Indirizzo binario del dato nella memoria principale	Hit o miss	Blocco della cache corrispondente a trovare o scrivere il dato
22	10110 <sub>base</sub>	Miss (5.9b)	(10110 <sub>base</sub> mod 8) = 110 <sub>base</sub>
26	11010 <sub>base</sub>	Miss (5.9c)	(11010 <sub>base</sub> mod 8) = 010 <sub>base</sub>
22	10110 <sub>base</sub>	Hit	(10110 <sub>base</sub> mod 8) = 110 <sub>base</sub>
26	11010 <sub>base</sub>	Hit	(11010 <sub>base</sub> mod 8) = 010 <sub>base</sub>
16	10000 <sub>base</sub>	Miss (5.9d)	(10000 <sub>base</sub> mod 8) = 000 <sub>base</sub>
3	00011 <sub>base</sub>	Miss (5.9e)	(00011 <sub>base</sub> mod 8) = 011 <sub>base</sub>
16	10000 <sub>base</sub>	Hit	(10000 <sub>base</sub> mod 8) = 000 <sub>base</sub>
18	10110 <sub>base</sub>	Miss (5.9f)	(10110 <sub>base</sub> mod 8) = 010 <sub>base</sub> →
16	10000 <sub>base</sub>	Hit	(10000 <sub>base</sub> mod 8) = 000 <sub>base</sub>

Se si prendono indirizzi con stesso modulo ma tag diverso si fa miss e si sovrascrive.

## CACHE &gt; COMPLETAMENTE ASSOCIAUTIVA

Segue policy che si può salvare in cache ovunque si voglia. Se piú si sceglie la politica

Anche in questo caso si usano Tag, validità

Sorgono problemi nelle ricerche fattibili come

- ricerca lineare  $\Rightarrow$  + latenza
- parallelizzazione  $\Rightarrow$  + costi

## CACHE &gt; SET-ASSOCIAUTIVA

Approccio ibrido. Si fa una mappatura diretta per i blocchi. Ma poi dentro il blocco ci sono piú contenitori in mapp. completamente associativa.

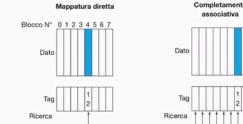
## Completemetnente associativa

Un blocco della memoria principale può essere scritto in una qualsiasi locazione della cache

La ricerca va effettuata su tutti gli elementi della cache

Svolta in parallelo attraverso un comparatore (aumentando i costi)

Adatta solo per un numero ridotto di blocchi



## Cache set-associativa

Cache set-associativa a una via  
(a mappatura diretta)

Blocco	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Cache set-associativa a due vie

Linea	Tag	Data	Tag	Data
0				
1				
2				
3				

Cache set-associativa a quattro vie

Linea	Tag	Data	Tag	Data	Tag	Data
0						
1						

Tag: l'informazione per capire se il blocco contiene la parola richiesta

Utilizzo meno recente (LRU): il blocco sostituito è quello che è rimasto inutilizzato per più tempo

Esempio di prestazione:

Associatività	Frequenza di miss
1	10,3%
2	8,6%
4	8,3%
8	8,1%

Possiede un numero prestabilito di locazioni alternative (almeno due) nelle quali può essere scritto o letto ogni blocco della memoria principale

(Numero del blocco) modulo (Numero delle linee della cache)

## CACHE ➤ GESTIONE DI SCRITTURA

La gestione delle scritture può seguire le policies di

- write through

Sincronizza i dati scritti tra memoria e cache.

Lenta

- write-back

se il dato non è sottoscritto, scrive in cache  
se viene sottoscritto scrive anche su memoria,

### Gestione della scrittura

#### Write-through

Ad ogni scrittura in cache si modifica il valore anche in memoria

Se il dato da scrivere non è presente in cache, viene prima caricato, poi modificato in cache e in memoria

Prestazioni non buone (uso di un buffer di scrittura)

#### Write-back

Il dato viene scritto solo in cache

La scrittura al livello inferiore avviene solo quando il blocco nella cache deve essere rimpiazzato

Più complesso da implementare

\*approfondimento in 6.8, non incluso nel programma del corso

## INTRODUZIONE >

Generalmente sono i dispositivi quali mouse, monitor.

In linea di principio, comunicano con la CPU usando un Bus.

In questo caso, il Bus viene detto controllore.

## CONTROLLORE > circuito

Si occupa di

- CPU → I/O : Trasforma le istruzioni / dati in segnali
- I/O → CPU : Trasforma i segnali in dati

Al suo interno contiene dei registri per contenere lo stato dei dispositivi

## CONTROLLORE > REGISTRI

I registri possono essere referenziali tramite 2 tipologie di "mapping"

### - MEMORY MAPPED I/O

Ovvero i registri vengono mappati in alcune aree della memoria primaria.

### - ISOLATED I/O

Si usano degli indirizzi / nomi dedicati per dispositivo.

All'interno è molto possibile vedere

- Dati
- Comandi
- Info di stato

## I/O > MODALITÀ BUSY WAITING

È un tipo di I/O in cui, per fare in modo che la CPU capisca se il dispositivo è usato o meno tramite richieste continue.

Si fa quindi il Polling verso l'I/O (non sempre).

Si ha lo scambio che delle risorse della CPU sono comunque utilizzate.

Il funzionamento si gestisce tramite i registri uscendo un Bit di stato.

- 1) Quando qualcosa viene fatto, si imposta a 1
- 2) La CPU legge il Bit e lo re-imposta a 0. Infine legge il dato.

Per l'output è diverso.

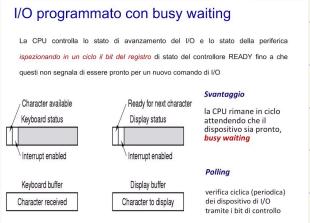
## I/O > MODALITÀ INTERRUPT

L'I/O, quando pronto, notificherà la CPU inviando un segnale di Interrupt.

Come prima, si gestisce uscendo più bit nella CPU che però dovrà dedicare bit destinati per ogni azione.

Dopo che c'è un gestore di interrupt che sarà un software. Si interrompe modificando il PC.

Interrupt hardware
• Gli interrupt hardware sono cambiamenti nel flusso di controllo legati alla gestione dei periferici e non al programma stesso
• Gli interrupt vanno sempre dal dispositivo alla CPU
• Il dispositivo che alza un'interruzione assegna una linea del bus che è direttamente connessa alla CPU (per esempio a uno o più bit di un registro)
• L'interrupt interrompe il programma in atto e trasfere il controllo ad un gestore di interrupt che eseguirà le azioni appropriate
• Una volta terminata la gestione dell'interrupt il controllo torna al programma interrotto



## I/O > MODALITÀ INTERRUPT (GESTIONE DI FUN.)

- Quando si usano le gestioni dell'Interrupt si maschera il segnale per non re-interrupper

### AZIONI SW:

Il codice ISA di gestione delle interruzioni salva tutti i registri (stato del programma che è stato interrotto)

Si possono leggere tutte le informazioni dal buffer del dispositivo

Se c'è stato un errore di I/O lo si gestisce

Le variabili **ptr** e **count** vengono aggiornate, se ci sono altri caratteri da visualizzare si copia il carattere puntato da **ptr** nel buffer del controller del dispositivo

Eventuali altri ack di fine trattamento interruzione

Ripristino registri

Esecuzione di una apposita istruzione di "return from interrupt - RETI" che ripristina modalità e stato della CPU.

### Interrupt

• Esempio: scrittura sullo schermo di una riga di "count" caratteri puntata da "ptr", gestita con interruzione a livello del singolo carattere (il controller del dispositivo è in grado di visualizzare un solo carattere alla volta)

#### AZIONI HW:

Il controller attiva una linea di interrupt del bus di sistema

Quando la CPU è pronta a gestire l'interruzione manda un ack sul bus

Il controller invia un intero sulle linee dati (vettore di interrupt)

La CPU preleva il vettore di interrupt dal bus e lo salva

La CPU impila PC ed altri registri di stato sullo stack

Il vettore di interrupt è usato come indice di memoria per trovare il valore di PC a cui si trova il gestore di quel tipo di interruzione

Si modifica PC, e a volte si evita che un'altra interruzione possa interrompere la gestione dell'interruzione in corso di esecuzione

## INTERRUPT > TRAP

La Trap è un tipo di interrupt sincrono.

Sono generati da condizioni eccezionali  
(overflow, div. per 0)

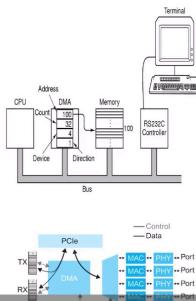
Da cui la CPU :

- Salva lo stato
- Esegue il codice dell'interrupt
- Continua l'esecuzione.

## I/O > MODALITÀ DMA (Direct Memory Access)

Predice di scaricare del lavoro di I/O ad un dura controller

### DMA (Direct Memory Access)



I/O programmato ma svolto da un'apposita componente con accesso diretto al bus

Il DMA è impostato direttamente dal software o il sistema operativo inizializzando opportuni registri

La CPU e il DMA si contendono l'uso del bus

### Comportamento del calcolatore

- Il programma in esecuzione deve essere **sospeso** e poi riattivato
  - Il calcolatore salva il punto del codice (Program Counter) in cui si è verificata l'eccezione (1)
- Il controllo passa quindi al "Gestore dell'Eccezione" (Exception Handler)
  - Il gestore dell'eccezione deve rimediare alla situazione
  - A seconda del tipo di eccezione, il gestore esegue azioni diverse
  - Il calcolatore identifica e salva la causa dell'eccezione (2)
- RISC-V: registri speciali **SEPC** e **SCAUSE**
  - (1) **SEPC** - l'indirizzo del codice in cui si verifica l'eccezione
  - (2) **SCAUSE** - la causa dell'eccezione

## GESTIONE ECCEZIONI ➤

Ve ne sono gestite come:

(00) - Salto diretto all'indirizzo della routine di gestione

(01) - Vettore di interruzioni

Per determinare il tipo si usa il registro STVEC (salvando 00101 come bit meno significativo)

## GESTIONE ECCEZIONI ➤

- SALTO DIRETTO

In STVEC si troverà l'indirizzo di base del gestore di tutte le interruzioni. Da lì si userà SCAUSE per direzionare.

- VETTORE DI INTERRUZIONE

Dentro il base ci sarà l'indirizzo di base di più gestori. Quello giusto viene preso tramite i SCAUSE.

Tabella con gli indirizzi delle handler per ogni causa

• PC - MEM [base + cause\*4]

• RISC-V, ma anche 370, 68000, Vax, 80x86, ...



## Salvataggio dello stato

- Salvataggio sullo stack (Push)
  - Vax, 68k, 80x86 (intero set dei registri)
  - RISC-V, MIPS (solo i registri necessari, potenzialmente zero)
- Registri ausiliari (Shadow Registers)
  - M88k, ARM
- Salvataggio in registri speciali
  - RISC-V, MIPS
  - Registri Speciali: EPC, CAUSE, STATUS, TVAL (o BadVaddr), ...