Il livello della microarchitettura

Aniello Minutolo

Introduzione

Sommario

1. Introduzione
2. Il data path
3. Operazioni della memoria

Livello di microarchitettura e ISA

- Il livello di microarchitettura si trova al di sopra del livello logico digitale e ha il compito di implementare il livello **ISA** (Instruction Set Architecture).
- La progettazione del livello di microarchitettura dipende sia dall'**ISA** che si vuole implementare sia dagli obiettivi di costo e prestazioni del calcolatore.

Complessità degli ISA

Molti **ISA** moderni, specialmente nelle architetture **RISC**, sono costituiti da **istruzioni semplici** che generalmente è possibile eseguire in un unico ciclo di clock.

ISA più complessi, come quello del **Core i7**, possono richiedere più cicli di clock per l'esecuzione di una singola istruzione.

- l'esecuzione di un'istruzione complessa può includere più operazioni come il recupero degli operandi in memoria, la loro lettura e la memorizzazione del risultato;
- l'ordinamento delle operazioni in un'istruzione complessa porta a strategie di controllo diverse rispetto agli **ISA** più semplici.

IJVM come modello di studio

Per studiare il comportamento di una microarchitettura considereremo un sottoinsieme della **Java Virtual Machine** che contiene solo istruzioni su interi, chiamato **IJVM**

- l'**IJVM**, pur essendo un insieme di piccole dimensioni, rappresenta tuttavia un buon punto di partenza per descrivere il controllo e l'ordinamento delle istruzioni.

Molte architetture che, come IJVM, contengono istruzioni complesse sono implementate tramite la **microprogrammazione.**

Microprogrammazione

La nostra microarchitettura conterrà un microprogramma, memorizzato in una **ROM**, il cui compito sarà quello di **prelevare**, **decodificare** ed **eseguire** le istruzioni IJVM.

Dato che abbiamo bisogno di un piccolo microprogramma che guidi in modo efficiente le singole porte logiche non possiamo utilizzare l'interprete Oracle JVM

- questo interprete è stato infatti scritto in C per essere portabile e non può controllare l'hardware.

Il ciclo fetch-decodifica-esecuzione

Un modo convenzionale per progettare una microarchitettura consiste nel concepirla come un problema di programmazione

- ogni istruzione del livello ISA è una funzione che deve essere richiamata dal programma principale;
- il programma principale è un semplice ciclo infinito che determina la funzione da invocare, la richiama e poi ricomincia la propria esecuzione.

Il ciclo fetch-decodifica-esecuzione

Il microprogramma ha delle variabili che costituiscono lo **stato del calcolatore**, e ogni funzione invocata cambia il valore di almeno una delle variabili, modificando di conseguenza lo stato del calcolatore

- il Program Counter (PC, “contatore d’istruzioni”) è una delle variabili che fanno parte dello stato e indica la locazione di memoria contenente la successiva funzione (cioè la successiva istruzione ISA) da eseguire;
- durante l’esecuzione di un’istruzione il PC viene fatto avanzare in modo da farlo puntare all’istruzione successiva.

Il ciclo fetch-decodifica-esecuzione

Le istruzioni IJVM sono corte e dall'aspetto semplice, e ogni istruzione è composta da alcuni campi, di solito uno o due, con uno scopo predefinito

- il primo campo di ogni istruzione è il codice operativo (**opcode**), che identifica il tipo d'istruzione, indicando se è di tipo ADD, di tipo BRANCH o altro;
- in molte istruzioni è presente un campo aggiuntivo che specifica l'**operando.**

Il ciclo fetch-decodifica-esecuzione

- Questo modello di esecuzione, chiamato **fetch-decodifica-esecuzione**, è utile a livello astratto e può anche costituire la base per l'implementazione di ISA complessi come IJVM.
- L'insieme delle microistruzioni compone il microprogramma e ciascuna di loro ha il controllo del **data path** durante un ciclo **fetch-decodifica-esecuzione.**

Il data path

Definizione e componenti del data path

- Il **data path** è la parte della **CPU** che contiene la **ALU**, i suoi input e i suoi output, ed è ottimizzato per interpretare programmi **IJVM.**
- Il **data path** include dei registri a cui è possibile accedere solamente a livello di microarchitettura (cioè dal microprogramma).
- I registri del **data path** generalmente memorizzano il valore di una variabile omonima appartenenti all'architettura del livello ISA.

Definizione e componenti del data path

Il **data path** della microarchitettura utilizzata nel nostro esempio, è stato attentamente ottimizzato per interpretare i programmi IJVM

- è tuttavia molto simile ai percorsi dati della maggior parte delle macchine.

La maggior parte dei registri può inviare il proprio contenuto sul **bus B**, collegato in **input** alla **ALU.**

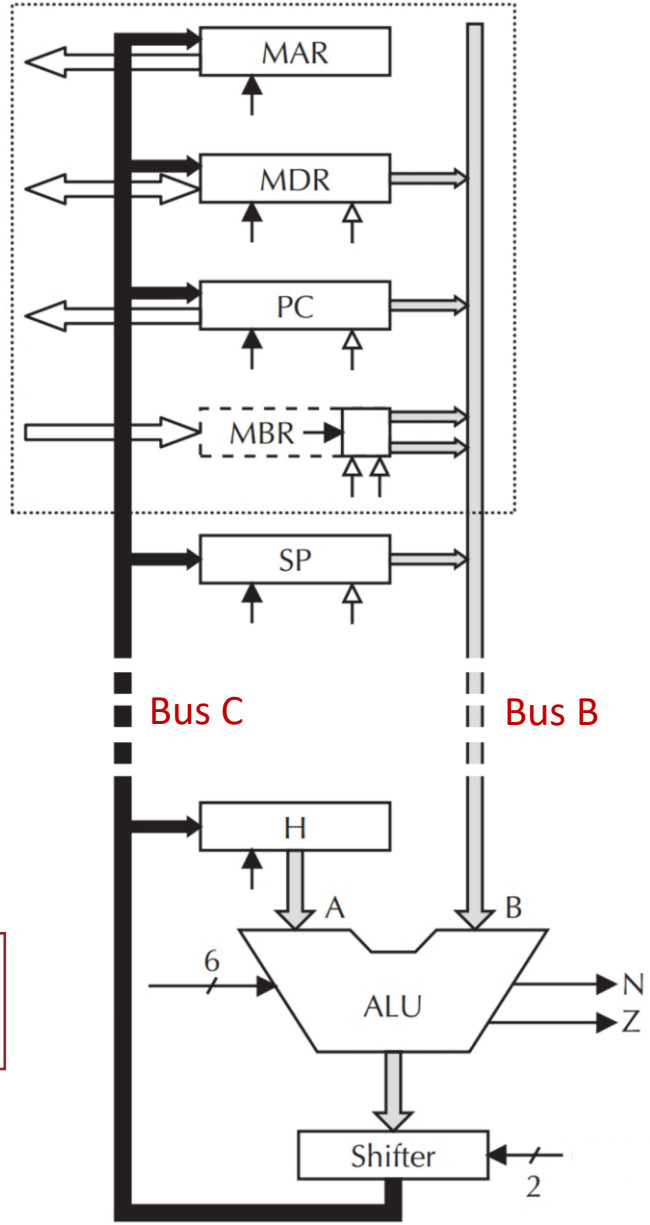
L'**output** della **ALU** guida invece lo **shifter**, che a sua volta invia il proprio risultato sul bus C i cui valori possono essere scritti allo stesso tempo in uno o più registri.

Il data path

Il **data path** della microarchitettura utilizzata, è stato attentamente ottimizzato per interpretare i programmi IJVM

- è tuttavia molto simile ai data path della maggior parte delle macchine.

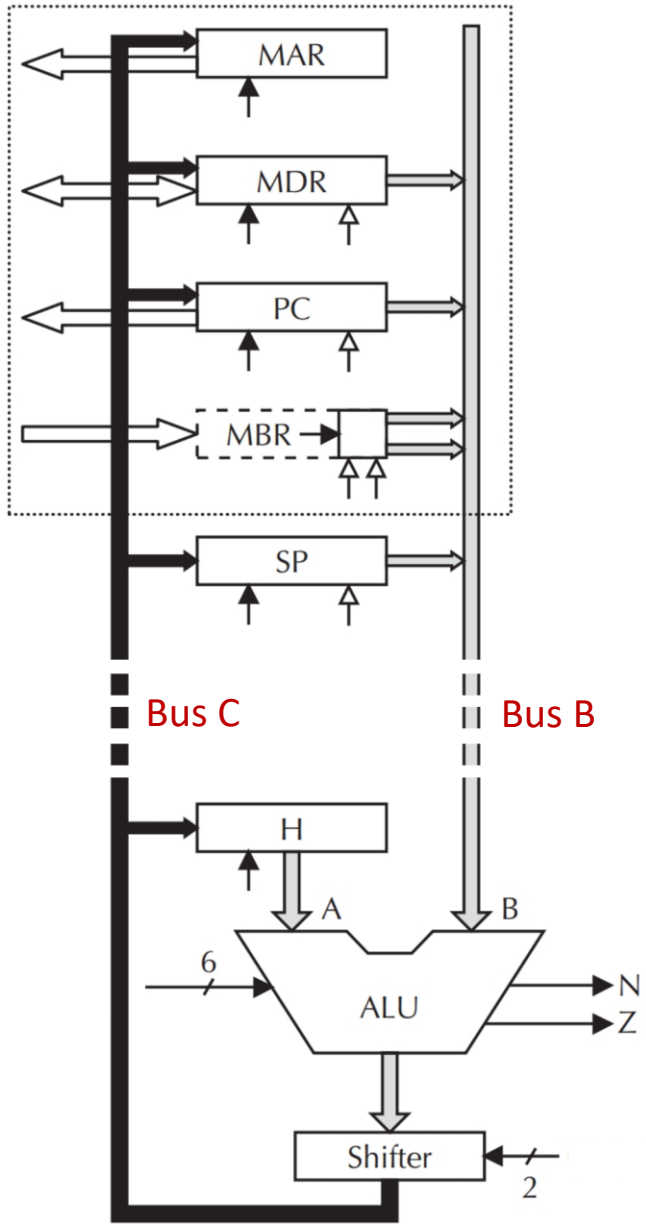
il trattino diagonale con a fianco un numero indica le linee per il controllo



Il data path

- La maggior parte dei registri può inviare il proprio contenuto sul **bus B**, collegato in **input** alla **ALU.**
- L'**output** della **ALU** guida invece lo **shifter**, che a sua volta invia il proprio risultato sul **bus C** i cui valori possono essere scritti allo stesso tempo in uno o più registri.

il trattino diagonale con a fianco un numero indica le linee per il controllo



Funzionamento della ALU

La **ALU** è controllata da sei linee di controllo che determinano il suo funzionamento

- **F0** e **F1** determinano l'operazione della ALU.
- **ENA** e **ENB** abilitano individualmente i due input.
- **INVA** inverte l'input di sinistra.
- **INC** forza la presenza di un riporto nel bit meno significativo, sommando quindi 1 al risultato.

Funzionamento della ALU

F_0	F_1	ENA	ENB	INVA	INC	Funzione
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	A
1	0	1	1	0	0	B
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

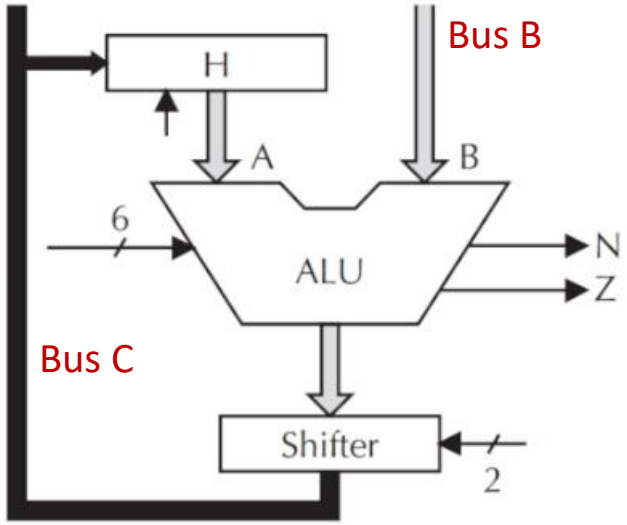
Le combinazioni delle linee di controllo della **ALU** definiscono operazioni come somma, sottrazione e complemento a due

Input e output della ALU

La **ALU** richiede due input: un input sinistro (**A**) collegato al registro **H** e un input destro (**B**) collegato al bus **B.**

L'output della **ALU** passa attraverso uno **shifter** che invia il risultato sul bus **C**, dove può essere scritto in uno o più registri.

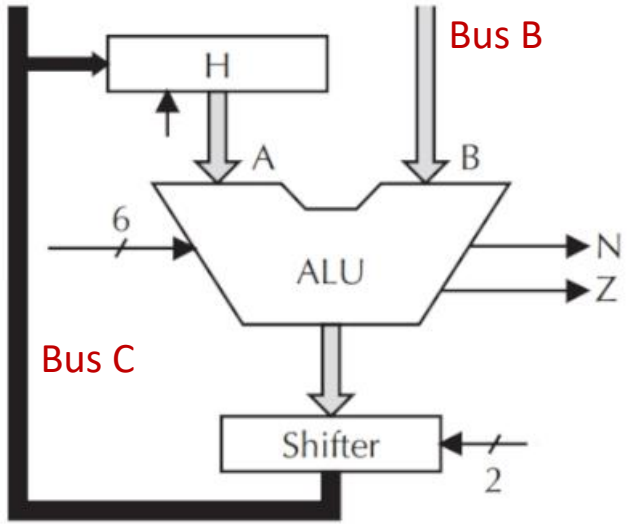
Per memorizzare un valore in **H**, è possibile far passare l'input destro della **ALU** senza modifiche attraverso lo **shifter.**



Linee di controllo dello shifter

Lo **shifter** può eseguire operazioni come **SLL8** (scorrimento logico a sinistra di un byte) e **SRA1** (scorrimento aritmetico a destra di 1 bit)

- **SLL8** trasla il valore a sinistra impostando gli 8 bit meno significativi a 0, mentre **SRA1** lascia inalterato il bit più significativo;
- queste operazioni sono controllate da linee di controllo specifiche che gestiscono l'output della **ALU.**

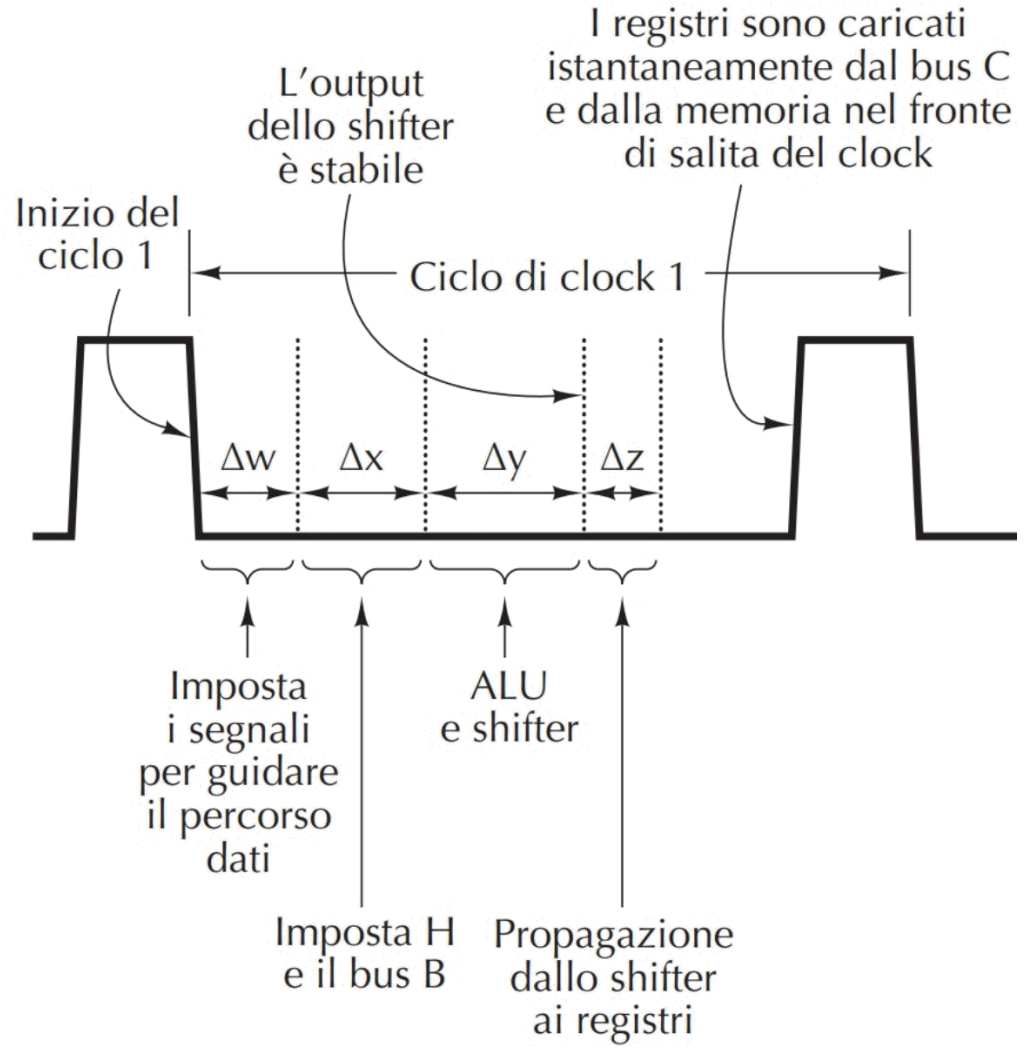


Lettura e scrittura dei registri

- È possibile leggere e scrivere lo stesso registro nello stesso ciclo grazie alla temporizzazione precisa del data path.
- La lettura di un registro avviene all'inizio del ciclo, mentre la scrittura avviene verso la fine, quando gli output della **ALU** sono stabili.
- Questa modalità evita incoerenze nei dati sfruttando i ritardi di propagazione dei segnali nel data path.

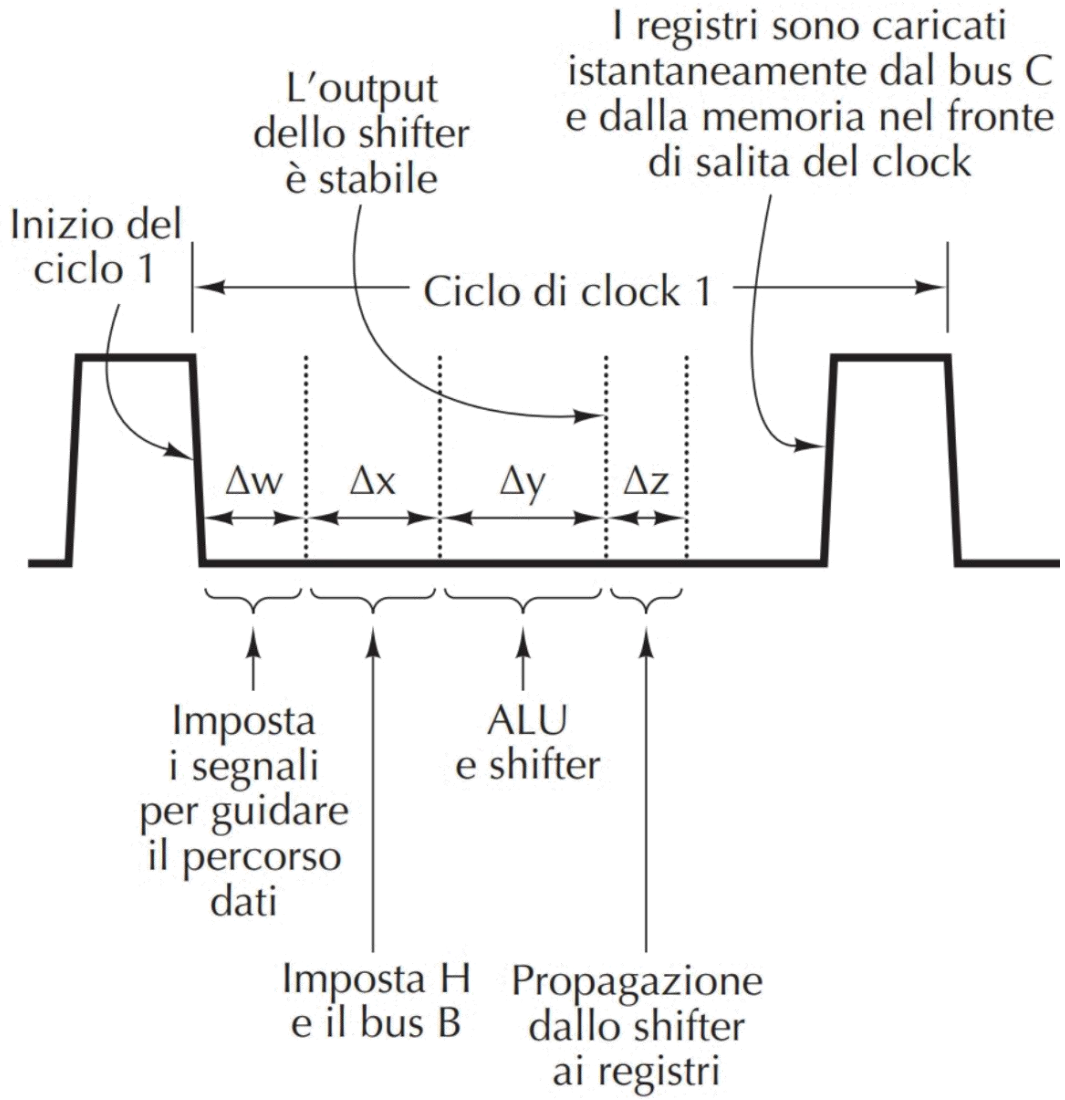
Temporizzazione degli eventi

La temporizzazione degli eventi nel data path è divisa in sottocicli impliciti determinati dai ritardi di propagazione dei circuiti, e l'inizio del sottociclo 1 segue il fronte di discesa del clock.



Temporizzazione degli eventi

È responsabilità dell' ingegnere progettista assicurarsi che il tempo Δw + Δx + Δy + Δz giunga sufficientemente in anticipo rispetto al fronte di salita del clock, in modo che il caricamento dei registri possa funzionare in ogni momento

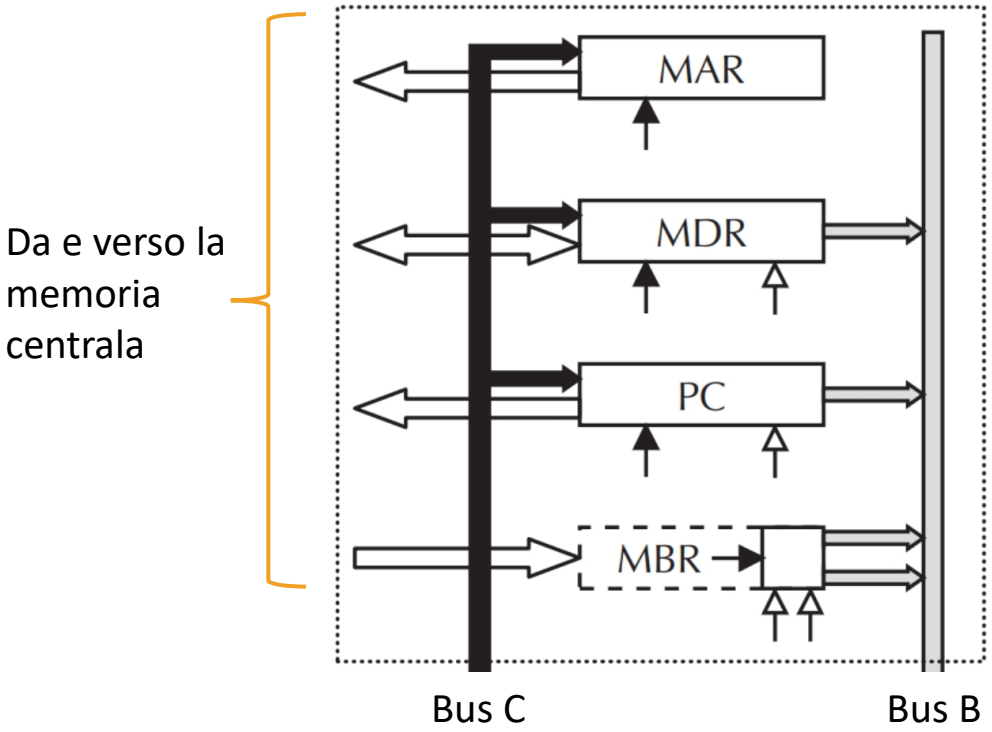


Operazioni della memoria

Modalità di accesso alla memoria

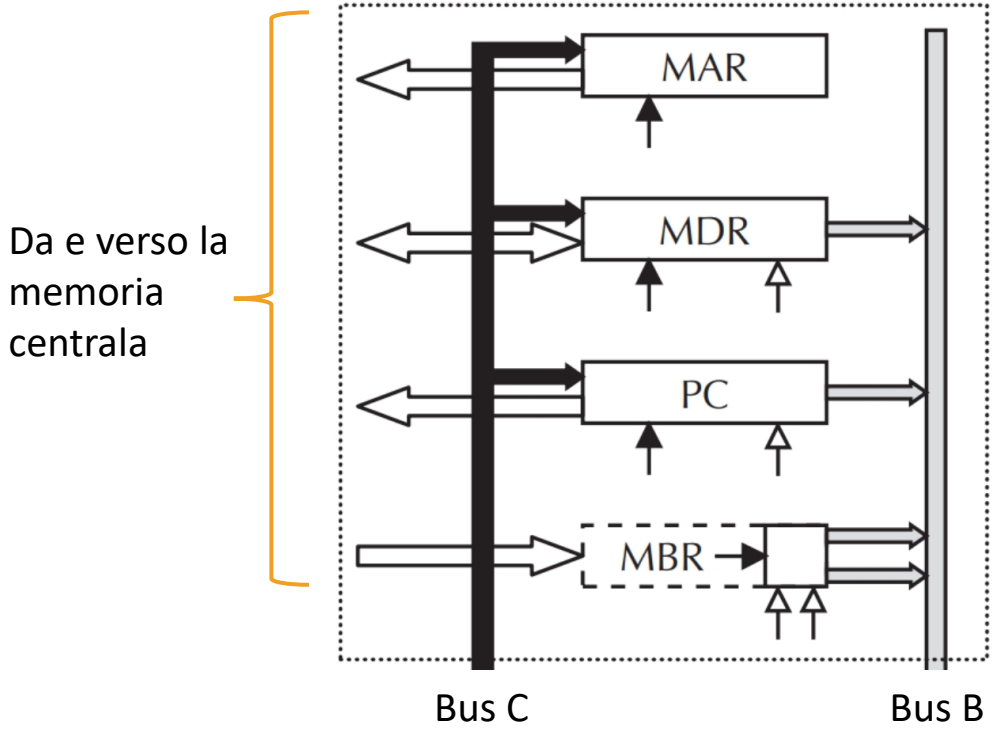
La macchina gestisce due porte di memoria

- una porta a 32 bit controllata dai registri **MAR** (Memory Address Register) e **MDR** (Memory Data Register);
- una porta a 8 bit che può soltanto leggere i dati dalla memoria, ed è controllata da un unico registro, **PC**, che legge 1 byte negli 8 bit meno significativi di **MBR.**



Modalità di accesso alla memoria

- Ciascuno dei registri è comandato da uno o due segnali di controllo;
- una freccia bianca sotto un registro indica un segnale di controllo che abilita l'output del registro verso il bus B;
- una freccia nera sotto un registro indica un segnale di controllo che scrive nel registro (cioè "carica") un valore proveniente dal bus C.



Modalità di accesso alla memoria

Per iniziare una lettura o una scrittura occorre caricare il registro di memoria appropriato e successivamente inviare alla memoria un segnale di scrittura.

MAR e PC sono utilizzati per far riferimento a due parti diverse della memoria

- la combinazione MAR/MDR è utilizzata per leggere e scrivere parole di dati del livello ISA;
- la combinazione PC/MBR è utilizzata per leggere il programma eseguibile del livello ISA, che consiste in un flusso di byte.

Confronto tra MAR e PC

- Il registro **MAR** contiene indirizzi di memoria espressi in parole, i suoi valori quindi si riferiscono a parole consecutive in memoria.
- Il registro **PC** contiene invece indirizzi espressi in byte e quindi i suoi valori fanno riferimento a byte consecutivi in memoria.

Registro	Unità di indirizzamento	Esempio (valore = 2)	Cosa legge in memoria	Dove va il dato
MAR	**Parole** (4 byte)	riferimento alla **parola 2**	byte **8-11**	MDR (32 bit)
PC	**Byte** (1 byte)	riferimento al **byte 2**	byte **2**	MBR (solo 8 bit)

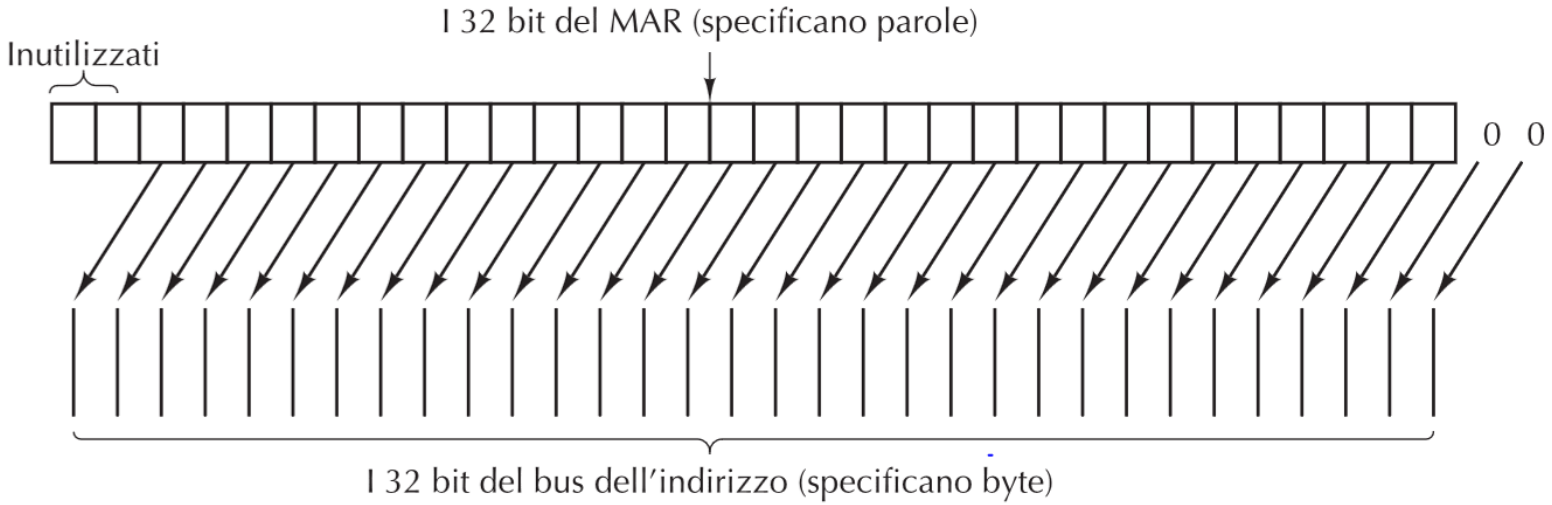
Conversione degli indirizzi di MAR

Nelle reali implementazioni è presente un'unica memoria, orientata al **byte**

- MAR continua a fare riferimento a parole consecutive in memoria (cosa necessaria per il modo in cui JVM è definita) anche se gli indirizzi della memoria fisica sono espressi in byte;
- quando MAR viene portato sul bus degli indirizzi, il suo valore dev'essere convertito ad un valore che faccia riferimento a byte consecutivi in memoria;
- poiché ogni parola è composta da 4 byte, per convertire un valore di MAR (riferimento alla i-esima parola) in un valore basato invece sui byte di memoria, è sufficiente moltiplicarlo per 4.

Esempio di valore in MAR	Riferimento a parole consecutive	Valore portato sul bus degli indirizzi	Riferimento a byte consecutivi
2	riferimento alla **parola 2**	8	riferimento al **byte 8**
3	riferimento alla **parola 3**	12	riferimento al **byte 8**

Conversione degli indirizzi di MAR



- La moltiplicazione per 4 del valore di MAR avviene tramite un **espediente** che effettua uno **shift a sinistra di 2 bit** di MAR;
- viene realizzato collegando i bit del registro MAR al bus indirizzi in maniera sfasata di 2 bit, in modo che sul bus degli indirizzi venga sempre caricato il valore di MAR moltiplicato per 4.

Bibliografia

Libro di testo

- Andrew S. Tanenbaum e Todd Austin. Architettura dei calcolatori. Un approccio strutturale. 6/ED. Anno 2013. Pearson Italia spa. (Disponibile nella sezione “Biblioteca”)

Fonte argomenti e immagini

- Capitolo 4, Paragrafo 4.1.1, pp. 249-257