
Riduzione della lunghezza del percorso di esecuzione

Aniello Minutolo




Velocità e costi





Sommario

1. Velocità e costi
 2. Ottimizzazione del microcodice
 3. Architettura a tre bus
- 

Compromessi nella progettazione

La progettazione del livello di microarchitettura, come tutto ciò che riguarda i calcolatori, è caratterizzata da compromessi e bilanciamenti.

Sono molte le caratteristiche di un calcolatore che si vorrebbero ottimizzare:

- velocità, costi, affidabilità, facilità di utilizzo, consumo energetico e dimensioni fisiche.

Strategie per aumentare la velocità

Esistono principalmente tre approcci tramite i quali è possibile aumentare la velocità di esecuzione

1. ridurre il numero di cicli di clock (path di esecuzione) necessari per eseguire un'istruzione per implementare le istruzioni ISA;
2. semplificare l'organizzazione così che il ciclo di clock possa abbreviarsi;
3. sovrapporre l'esecuzione delle istruzioni (architetture superscalari e parallelismo).

Quale tecnica usare?

- tante progettazioni possono influire in modo drastico sul numero di cicli di clock, sul periodo di clock o, molto spesso, su entrambi i fattori.

Lunghezza del percorso di esecuzione

La **lunghezza del percorso** è definita come il numero di cicli di clock necessari per eseguire un insieme di operazioni

- in alcuni casi la lunghezza del percorso può essere accorciata aggiungendo dei componenti hardware specializzati.

Lunghezza del percorso di esecuzione

Ad esempio:

- se si aggiunge al **PC** un circuito incrementatore (ovvero un sommatore in cui un addendo è sempre impostato a 1) è possibile eliminare alcuni cicli, dato che non è più necessario utilizzare la ALU per far avanzare il suo valore;
- il prezzo da pagare è rappresentato dall'hardware aggiuntivo;
- tuttavia, per la maggior parte delle istruzioni, durante gli stessi cicli spesi per incrementare **PC**, viene svolta un'operazione di lettura; di conseguenza l'esecuzione dell'istruzione successiva non può essere anticipata perché dipende dal dato che deve giungere dalla memoria.

Lunghezza del percorso di esecuzione

- Se si intende ridurre il numero di cicli necessari per prelevare le istruzioni occorre qualcosa di più rispetto al solo circuito che incrementa il PC.
- La tecnica più efficace per una significativa accelerazione del prelievo delle istruzioni è la sovrapposizione dell'esecuzione delle istruzioni.

Lunghezza del percorso di esecuzione

La separazione delle componenti dei circuiti per il prelievo dell'istruzione (la porta a 8 bit della memoria e i registri MBR e PC) è più efficace se si rende quest'unità indipendente, dal punto di vista funzionale, dal percorso dati:

- in questo modo l'unità può prelevare autonomamente il successivo codice operativo o operando, anche in modo asincrono rispetto al resto della CPU;
- permette di prelevare in anticipo una o più istruzioni.

Proprietà terriera di un chip

La velocità costituisce soltanto metà del problema; l'altra è rappresentata dai costi:

- anche il costo può essere misurato in vari modi, ma è difficile darne una precisa definizione.

Spesso i progettisti parlano di costi in termini di “proprietà terriera”, riferendosi in questo modo all'area richiesta dal circuito (presumibilmente misurata in pico-acri):

- anche se è possibile contare i singoli componenti, come i transistor, le porte logiche e le unità funzionali, spesso questo valore non è importante quanto invece lo è l'area occupata sul circuito integrato.

Costi e complessità hardware

Uno dei circuiti che nella storia è stato studiato in modo più accurato è il sommatore:

- sono stati realizzati migliaia di progetti e quelli più veloci, oltre a essere molto complessi, hanno prestazioni decisamente migliori rispetto a quelli più lenti.

Il progettista di sistema deve decidere se l'aumento di velocità giustifica l'aumento della "proprietà terriera".

Impatto dei ritardi nella decodifica

Per determinare a quale velocità può arrivare il clock, uno dei fattori chiave è la quantità di lavoro da eseguire durante ogni ciclo:

- l'hardware è in grado di eseguire più compiti in parallelo
- la lunghezza del ciclo di clock dipende dalla sequenza di operazioni che devono essere eseguite **serialmente** durante ogni singolo ciclo

Un aspetto che può essere controllato è la quantità di operazioni di decodifica da eseguire:

- ad esempio, all'interno della parola della microistruzione avevamo bisogno solamente di 4 bit per indicare il registro selezionato;
- purtroppo questo risparmio introduce un ritardo nel percorso dati dovuto al circuito di decodifica.

Impatto dei ritardi nella decodifica

Spesso il ritardo dovuto ai circuiti di decodifica determina quanto deve essere lungo il ciclo di clock

- ciò potrebbe limitare la frequenza del ciclo di clock, costringendo quindi l'intero calcolatore a funzionare a una velocità leggermente inferiore.

Bilanciamento tra velocità e costi

- la riduzione della memoria di controllo di 5 bit per parola avviene a spese di un rallentamento del clock;
- se si vuole implementare un sistema ad alte prestazioni, l'utilizzo di un decodificatore non è probabilmente una buona idea.

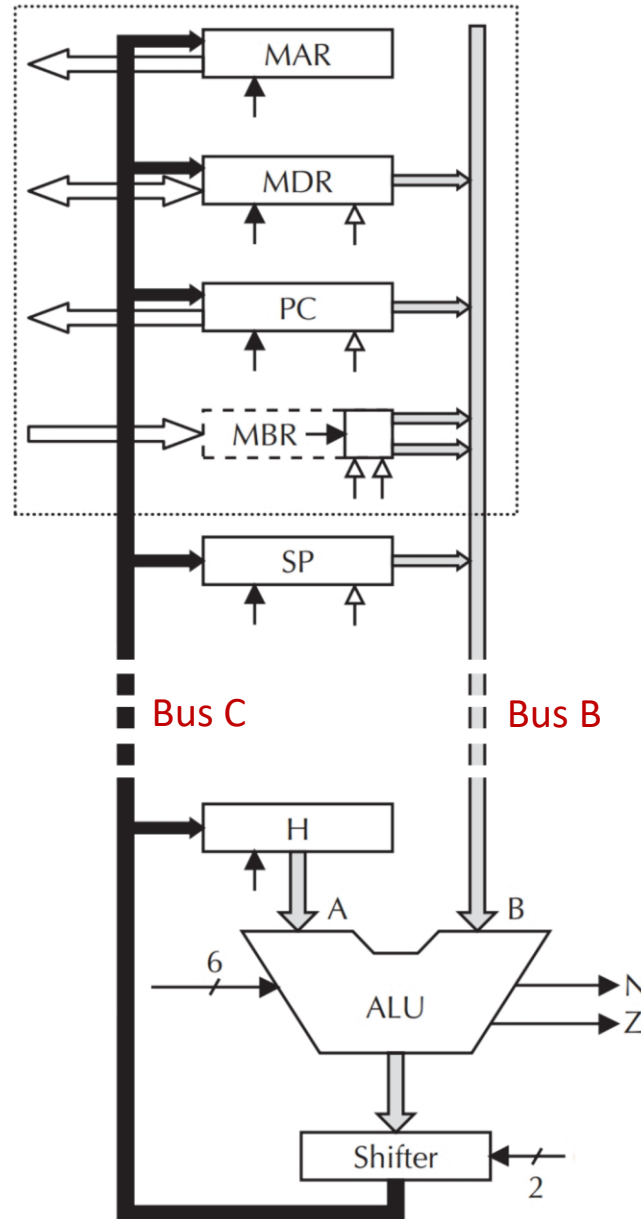


Ottimizzazione del microcodice

Progettazione della Mic-1

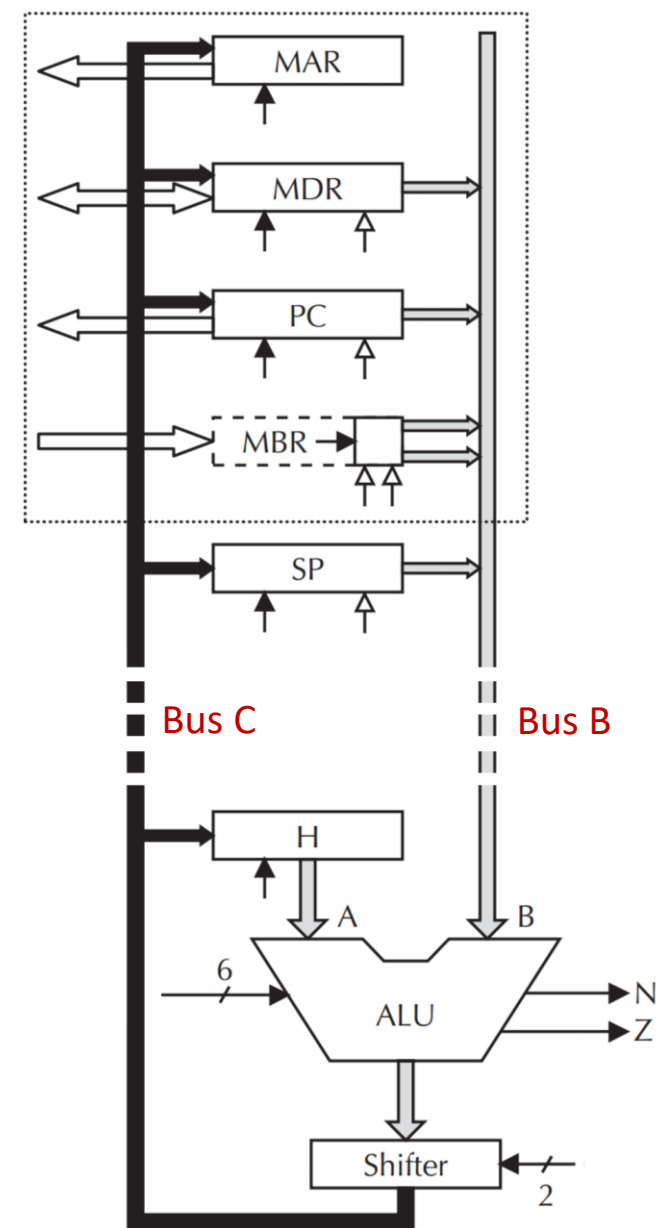
La macchina Mic-1 è stata progettata per essere allo stesso tempo moderatamente semplice e moderatamente veloce

- la CPU utilizza una quantità minima di hardware, tra cui 10 registri, una semplice ALU replicata 32 volte, uno shifter, un decodificatore, una memoria di controllo e alcuni bit qua e là, necessari per tenere insieme i diversi componenti



Progettazione della Mic-1

- La macchina Mic-1 è stata progettata per essere allo stesso tempo moderatamente semplice e moderatamente veloce;
- la CPU utilizza una quantità minima di hardware, tra cui 10 registri, una semplice ALU replicata 32 volte, uno shifter, un decodificatore, una memoria di controllo e alcuni bit qua e là, necessari per tenere insieme i diversi componenti.



Progettazione della Mic-1

Abbiamo visto come implementare IJVM tramite microcodice e con poco hardware, ma esistono alternative più veloci:

- vedremo quelle basate sulla riduzione del numero di microistruzioni delle istruzioni ISA (cioè, basate sulla riduzione della lunghezza del percorso di esecuzione).

Riduzione della lunghezza del percorso di esecuzione

Per ridurre la lunghezza del percorso di esecuzione è spesso necessario complicare la microarchitettura introducendo:

- registri interni;
- bus interni;
- unità funzionali specializzate (ad esempio, dedicate al fetch).

In alcuni casi, un'ottimizzazione mirata del microcodice può portare a miglioramenti significativi.

Esempio di ottimizzazione del microcodice

In Mic-1 il ciclo principale consiste in una microistruzione *Main1* da eseguire all'inizio di ogni istruzione IJVM.

In alcuni casi è possibile sovrapporre *Main1* all'istruzione precedente e in realtà in parte questo è già stato fatto:

- infatti, quando *Main1* viene eseguito il codice operativo da interpretare è già presente all'interno di *MBR*, in quanto è già stato prelevato o dal precedente ciclo principale oppure durante l'esecuzione dell'istruzione precedente.

Esempio di ottimizzazione del microcodice

Consideriamo le sequenze di microistruzioni che terminano con un salto a *Main1*:

- *Main1* può essere spostata alla fine della sequenza (piuttosto che all'inizio della sequenza successiva), replicando così in più punti del microcodice la diramazione a più destinazioni;
- in alcuni casi, *Main1* può essere unita alle microistruzioni precedenti, se queste non vengono utilizzate completamente.

Esempio di ottimizzazione del microcodice

Ad esempio, consideriamo la sequenza dinamica originale di microistruzioni per l'esecuzione dell'istruzione POP:

- questa istruzione richiede quattro cicli di clock: tre per le microistruzioni specifiche di POP e uno per il ciclo principale;
- si noti che nel ciclo di clock *pop2* la ALU non viene utilizzata.

Etichetta	Operazioni	Commenti
pop1	$MAR = SP = SP - 1; rd$	Legge la parola sotto la cima dello stack.
pop2		Attende che il nuovo TOS sia letto dalla memoria.
pop3	$TOS = MDR; goto Main1$	Copia la nuova parola in TOS.
Main1	$PC = PC + 1; fetch; goto (MBR)$	MBR contiene il codice operativo; prelievo del byte successivo, diramazione

Esempio di ottimizzazione del microcodice

- Nel ciclo di clock *pop2* in cui la ALU non veniva utilizzata, è possibile eseguire alcune delle operazioni di *Main1*, al prezzo di un piccolo incremento della memoria di controllo;
- in *pop2* si possono anticipare incremento PC e fetch, e in *pop3* si può saltare direttamente alla prossima istruzione invece che a *Main1*;
- questa piccola modifica riduce di un ciclo il tempo di esecuzione della successiva microistruzione, e riduce a 3 microistruzioni il microprogramma per l'esecuzione dell'istruzione POP.

Etichetta	Operazioni	Commenti
pop1	MAR = SP = SP - 1; rd	Legge la parola sotto la cima dello stack.
Main1.pop	PC = PC + 1; fetch	MBR contiene il codice operativo; prelievo del byte successivo.
pop3	TOS = MDR; goto (MBR)	Copia la nuova parola in TOS; diramazione a seconda del codice operativo.

Esempio di ottimizzazione del microcodice

È possibile formulare in questo modo la prima tecnica che ci permette di ridurre la lunghezza del percorso:

- unire il ciclo d'interpretazione alla fine delle sequenze di microcodice.

Etichetta	Operazioni	Commenti
pop1	MAR = SP = SP – 1; rd	Legge la parola sotto la cima dello stack.
pop2		Attende che il nuovo TOS sia letto dalla memoria.
pop3	TOS = MDR; goto Main1	Copia la nuova parola in TOS.
Main1	PC = PC + 1; fetch; goto (MDR)	MDR contiene il codice operativo; prelievo del byte successivo, diramazione



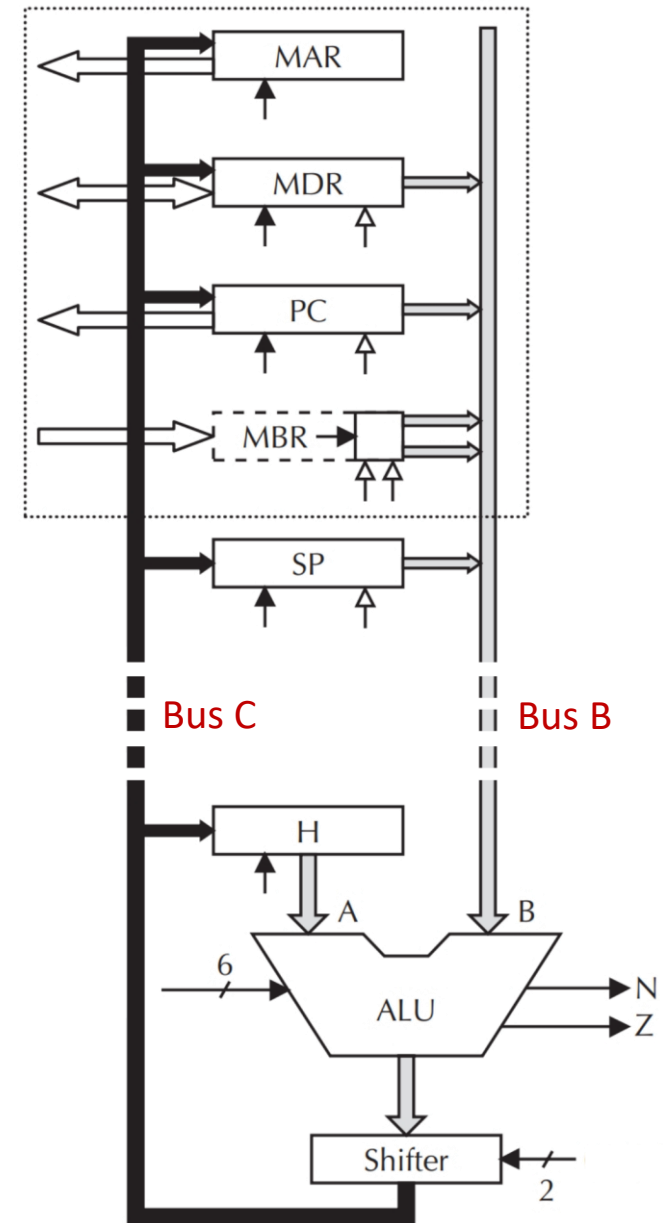
Etichetta	Operazioni	Commenti
pop1	MAR = SP = SP – 1; rd	Legge la parola sotto la cima dello stack.
Main1.pop	PC = PC + 1; fetch	MDR contiene il codice operativo; prelievo del byte successivo.
pop3	TOS = MDR; goto (MDR)	Copia la nuova parola in TOS; diramazione a seconda del codice operativo.



Architettura a 3 bus

Architettura a 3 bus

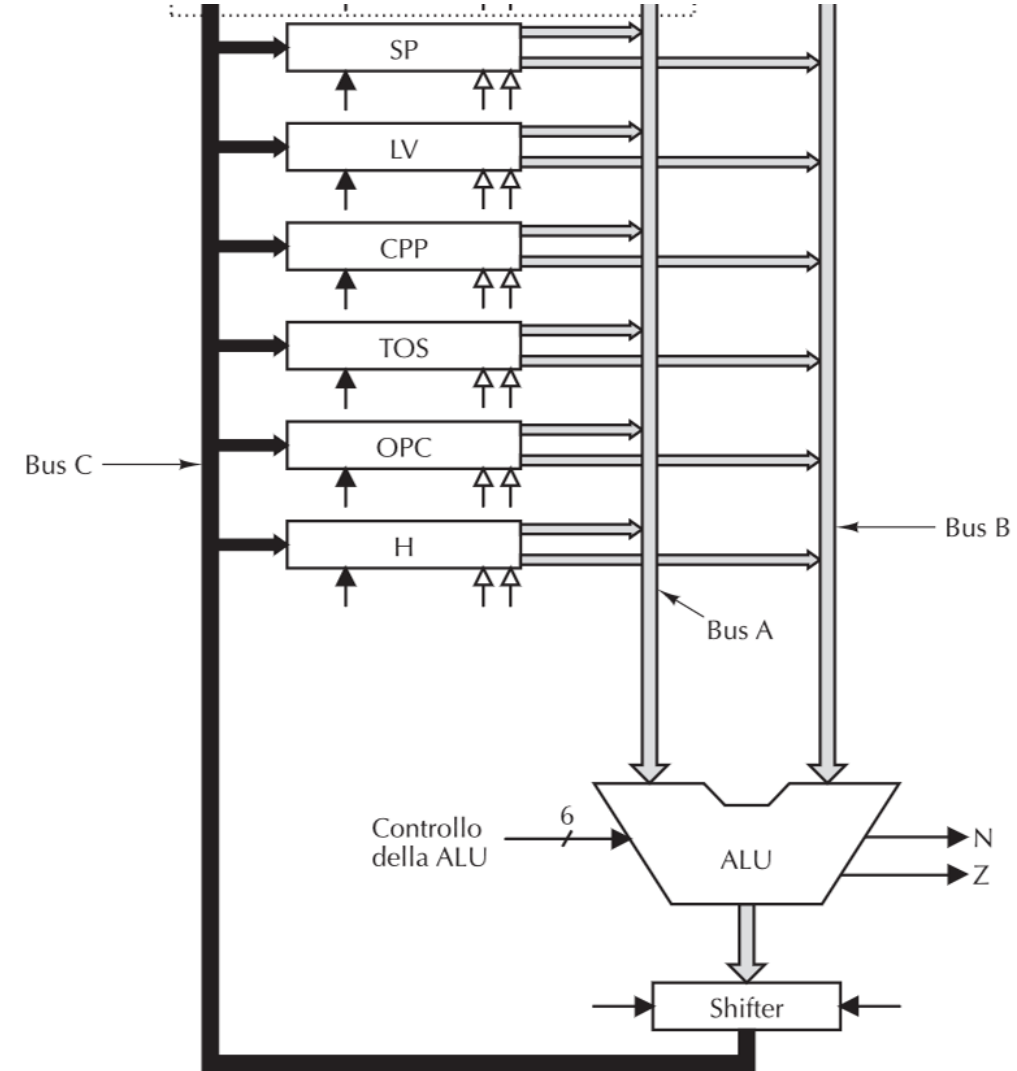
- Cos'altro possiamo fare per ridurre la lunghezza del percorso di esecuzione?
- Nella architettura originale a due bus di Mic-1 molti cicli di clock sono sprecati;
- è possibile utilizzare l'input sinistro della ALU soltanto con il registro H;
- non esiste alcun modo per sommare direttamente due registri arbitrari;
- l'unica soluzione consiste nello sprecare inizialmente un ciclo per copiare il valore di un registro in H.



Architettura a 3 bus

Un'altra semplice modifica per ridurre la lunghezza del percorso di esecuzione consiste nell'utilizzo di due bus di input, A e B, per la ALU:

- tutti (almeno la maggior parte) dei registri avrebbero accesso a entrambi i bus di input;
- il vantaggio di avere due bus di input è che in questo modo diventa possibile sommare fra loro, in unico ciclo, due registri qualsiasi.



Architettura a 3 bus

Ad esempio, per comprendere il valore di un'architettura a 3 bus, consideriamo l'implementazione originale di ILOAD per Mic-1

- durante *iload1* LV è copiato all'interno di H per l'unico motivo di poterlo successivamente sommare a MBRU durante *iload2*.

Etichetta	Operazioni	Commenti
iload1	$H = LV$	MBR contiene l'indice; copia LV in H.
iload2	$MAR = MBRU + H$; rd	MAR = indirizzo della variabile locale da inserire nello stack.
iload3	$MAR = SP = SP + 1$	SP punta alla nuova cima dello stack; prepara la scrittura.
iload4	$PC = PC + 1$; fetch; wr	Incrementa PC; prelievo del successivo codice operativo; scrive in cima allo stack.
iload5	$TOS = MDR$; goto Main1	Aggiorna TOS.
Main1	$PC = PC + 1$; fetch goto (MBR)	MBR contiene il codice operativo; prelievo del byte successivo; diramazione.

Architettura a 3 bus

In un'architettura a tre bus possiamo invece risparmiare un ciclo, sommando il ciclo d'interpretazione a *ILOAD*

- l'esecuzione di *ILOAD* passa da sei a cinque cicli.

Etichetta	Operazioni	Commenti
iload1	MAR = MBRU + LV; rd	MAR = indirizzo della variabile locale da inserire sullo stack.
iload2	MAR = SP = SP + 1	SP punta alla nuova cima dello stack; prepara la scrittura.
iload3	PC = PC + 1 ; fetch; wr	Incrementa PC; prelievo del successivo codice operativo; scrive in cima allo stack.
iload4	TOS = MDR	Aggiorna TOS.
iload5	PC = PC + 1; fetch goto (MBR)	MBR contiene già il codice operativo; prelievo del byte dell'indice.

Architettura a 3 bus

L'aggiunta del terzo bus non ha modificato la lunghezza del percorso di esecuzione, ma ha ridotto il tempo totale di esecuzione di *ILOAD*, portandolo da sei a cinque cicli.

Architettura a 3 bus

A questo punto abbiamo a disposizione una seconda tecnica che ci permette di ridurre la lunghezza del percorso:

- passare da un'architettura a due bus a una a tre bus-

Etichetta	Operazioni	Commenti
iload1	$MAR = MBRU + LV$; rd	MAR = indirizzo della variabile locale da inserire sullo stack.
iload2	$MAR = SP = SP + 1$	SP punta alla nuova cima dello stack; prepara la scrittura.
iload3	$PC = PC + 1$; fetch; wr	Incrementa PC; prelievo del successivo codice operativo; scrive in cima allo stack.
iload4	$TOS = MDR$	Aggiorna TOS.
iload5	$PC = PC + 1$; fetch goto (MBR)	MBR contiene già il codice operativo; prelievo del byte dell'indice.

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, Paragrafi da 4.4 a 4.4.2, pp. 291-297.