2 IL CICLO MACCHINA

La CPU ha il compito di eseguire una sequenza di istruzioni (un programma). Prima preleva l'istruzione insieme a eventuali dati dalla memoria esterna (la RAM) e carica tutto negli opportuni registri. Poi l'ALU elabora l'istruzione e mette a disposizione il risultato in un registro dedicato o in memoria.
Tutto questo rappresenta un **ciclo macchina** (FIGURA 4) o **fetch-execute cycle**.

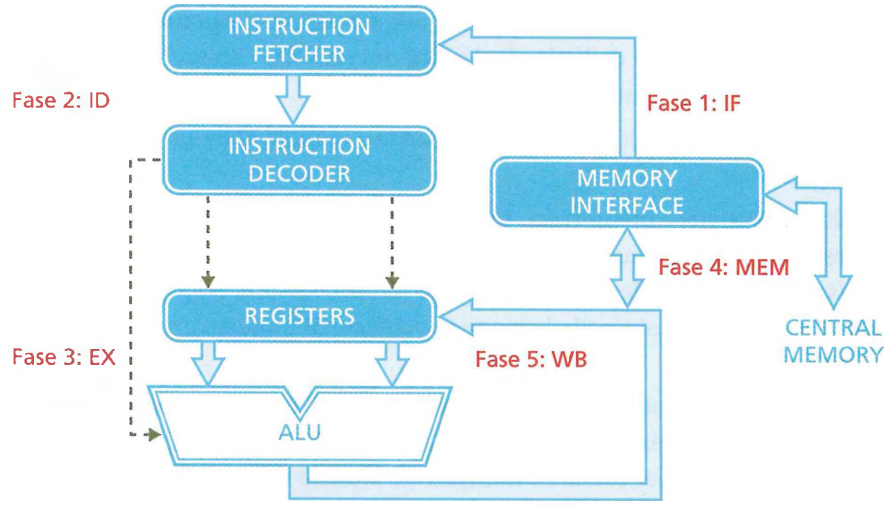


FIGURA 4 Schema a blocchi del ciclo macchina

Le fasi durante le quali la CPU compie un intero ciclo sono cinque e si ripetono in sequenza fino al termine del programma:

1. **IF** (Instruction Fetch): lettura dell'istruzione da memoria;
2. **ID** (Instruction Decode): decodifica istruzione e lettura dati dai registri;
3. **EX** (Execution): esecuzione dell'istruzione;
4. **MEM** (Memory): scrittura del risultato in memoria (solo per certe istruzioni);
5. **WB** (Write Back): scrittura del risultato nel registro opportuno e aggiornamento dello stato.

Gli attuali microprocessori superano la suddivisione delle istruzioni in cinque fasi. È possibile frazionare le fasi del ciclo in sequenze di **micro-operazioni**, ognuna di complessità comparabile alle altre, che sono ripetute identicamente su flussi continui di dati.
Ottimizzando i microprocessori, ogni micro-operazione è eseguita in un ciclo di clock.

Un ciclo macchina è così costituito da una sequenza di fasi, ognuna a sua volta costituita da micro-operazioni eseguite dalla CPU. La durata di una micro-operazione prende il nome di **cycle time** ed è comunemente indicata con t_{CPU}.

esempio

Frequenza di clock = 2 GHz
t_{CPU} = 0,5 ns (nanosecondi).

Il valore $\mathbf{1/t_{CPU}}$ è la **frequenza di clock** della CPU, misurata in gigahertz (GHz).
Nella FIGURA 5 sono mostrate tre istruzioni eseguite in sequenza, in cui ogni intervallo dell'asse del tempo rappresenta un t_{CPU}, che per semplicità supponiamo corrispondere alla durata di una fase (anche se sarebbe più realistico che corrispondesse alla durata di una delle micro-operazioni in cui è suddivisa ogni fase).
Se ogni fase ha un t_{CPU} di 0,5 ns, occorreranno 7,5 ns (3 istruzioni × 5 fasi × 0,5 ns) per eseguirle.

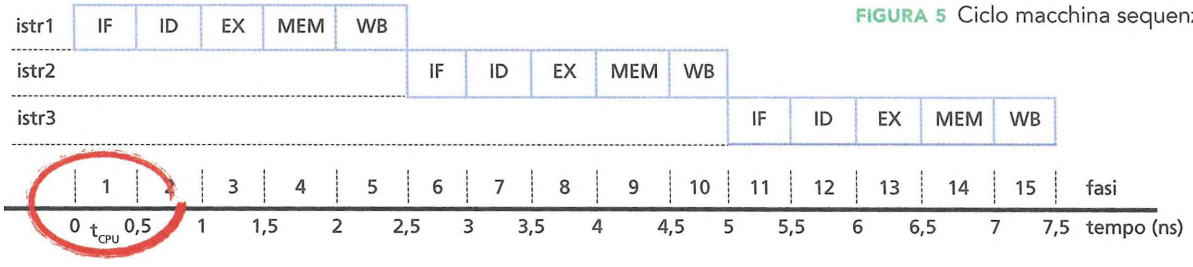


FIGURA 5 Ciclo macchina sequenziale

Il flusso di esecuzione dei cicli macchina coinvolge tutti i **registri** che abbiamo visto nella precedente lezione.
Il flowchart nella FIGURA 6 a pagina seguente mostra i passi principali del ciclo e i registri coinvolti (i blocchi tratteggiati indicano azioni non obbligatoriamente realizzate a ogni ciclo).

FASE 1: IF

- L'indirizzo della prima istruzione viene caricato nel Program Counter e poi da questo all'Address Register.
- L'istruzione è prelevata (fetch) dalla memoria e caricata nell'Instruction Register (anche eventuali dati vengono caricati nel Data Register).
- Il Program Counter viene incrementato al fine di puntare alla successiva istruzione da eseguire. La posizione dell'istruzione successiva è (salvo nelle istruzioni di salto) implicita; il programma è eseguito in sequenza e quindi l'indirizzo dell'istruzione successiva corrisponde all'indirizzo della prima istruzione successiva all'istruzione corrente.

FASE 2: ID

L'opcode dell'istruzione è decodificato e in base a esso si caricano nei registri di lavoro (R0, ..., Rn) gli operandi necessari all'istruzione e si attivano le microcircuiterie necessarie a svolgere l'operazione richiesta.

FASE 3: EX

Le elaborazioni previste dall'opcode dell'istruzione sono eseguite nell'ALU, sfruttando i registri di lavoro (R0, ..., Rn) per eventuali risultati parziali.

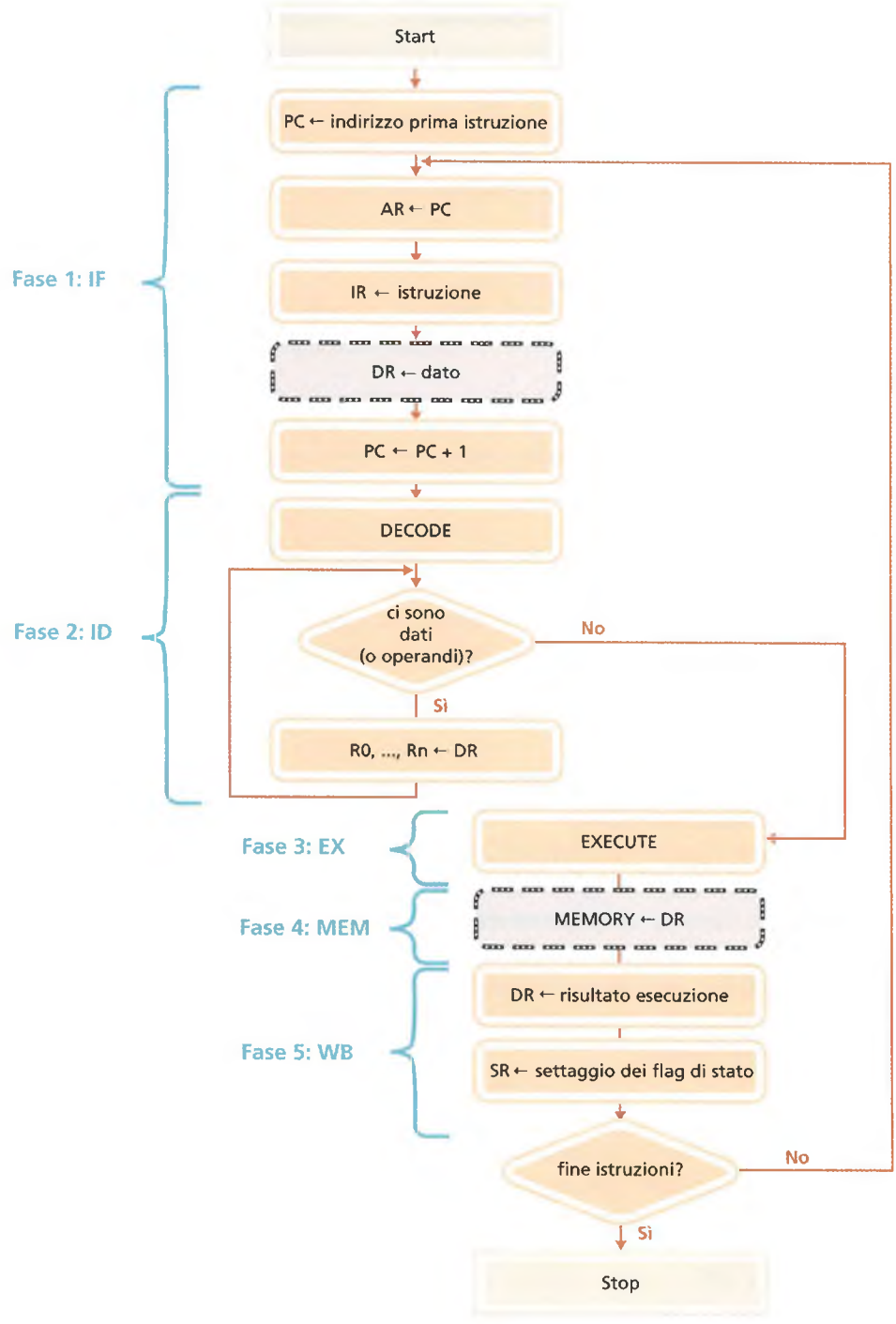
FASE 4: MEM

Consiste nell'eventuale scrittura del risultato di un'operazione in memoria (per esempio in caso di assegnamento di valori a variabili del programma) o verso le periferiche.

FASE 5: WB

Il risultato dell'esecuzione dell'istruzione viene scritto nel Data Register e lo stato del processore a seguito dell'esecuzione viene settato nello Status Register.

FIGURA 6 Flowchart del ciclo macchina



FISSA LE CONOSCENZE

- Quali sono le cinque fasi di un ciclo macchina?
- Che cos'è il cycle time (indicato con t_{CPU})?
- Che cos'è la frequenza di clock della CPU e con quale unità di misura è misurata?
- Descrivi nel dettaglio che cosa succede nella fase IF (*Instruction Fetch*).

3 LA TECNICA PIPELINING

3.1 La pipeline

Il **pipelining** è una tecnica che consente di elaborare in parallelo più istruzioni.
Come abbiamo visto nella Lezione precedente, l'esecuzione delle istruzioni segue il ciclo macchina costituito da una sequenza di fasi, a loro volta costituite da diverse micro-operazioni, eseguite dalla CPU ognuna in un ciclo di clock.

Con la tecnica del pipelining, le micro-operazioni sono raggruppate in **unità funzionali** indipendenti le une dalle altre. Le unità funzionali sono dei blocchi (microcircuiti) specializzati nell'eseguire velocemente determinate micro-operazioni. Questo implica che nella stessa CPU si possa elaborare **contemporaneamente** più micro-operazioni appartenenti a unità funzionali diverse. Ogni sequenza di elaborazioni indipendente forma una "conduttura", detta **pipeline**.

In pratica più unità eseguono le loro operazioni (eventualmente identiche) in **parallelo** su diversi dati.

Supponiamo per semplicità di essere nella situazione ottimale in cui un'unità funzionale sia in grado di elaborare in modo indipendente un'intera fase del ciclo macchina di un'istruzione.
Abbiamo quindi 5 unità funzionali indipendenti (si può arrivare quindi a 5 pipeline) e supponiamo di dover elaborare un programma di 8 istruzioni.
Nella FIGURA 7 si vede come dall'intervallo di tempo 5 all'intervallo 8, le 5 unità funzionali lavorino contemporaneamente realizzando delle pipeline a pieno regime di funzionamento.

#prendinota

Un parrucchiere per signora può organizzare il suo salone con 5 dipendenti, ognuno dei quali specializzato in un servizio diverso: tinta, shampoo, taglio, piega e manicure.
Se ha molte clienti da servire (così come una CPU ha molte istruzioni da eseguire), può servirle meglio e in minor tempo avendo dipendenti specializzati che possono lavorare in parallelo ognuno su un servizio diverso.

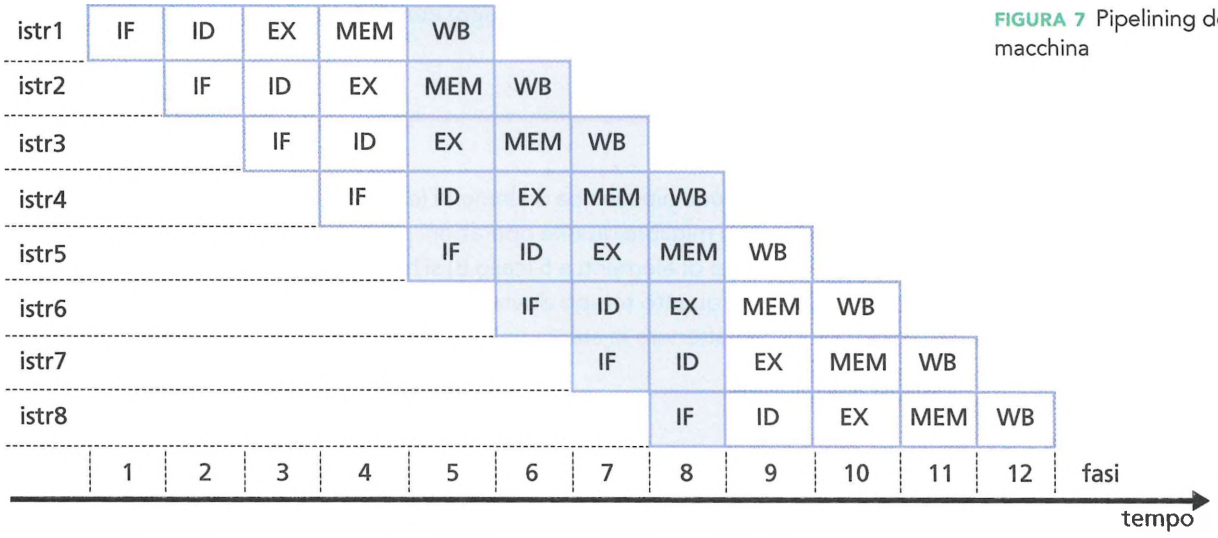


FIGURA 7 Pipelining del ciclo macchina

Quando la prima istruzione è nella fase 5, la seconda si trova già nella fase 4 e può immediatamente passare alla fase 5. Come si può osservare dalla Figura 7, 8 istruzioni sono eseguite in 12 fasi complessive. Non utilizzando la tecnica delle pipeline, occorre invece aspettare 40 fasi (5 fasi × 8 istruzioni). Nei sistemi di elaborazione privi di pipelining, ogni fase può iniziare solo dopo che la precedente è terminata.

Il pipelining si realizza quindi sostituendo all'unità funzionale "monolitica" una cascata di unità più semplici, una per ciascun gruppo di micro-operazioni progettate, dette **stadi della pipeline**.

3.2 Problemi di gestione delle pipeline

Questa tecnica funziona molto bene se non vi sono legami troppo stretti tra due istruzioni; in particolare, se a un'istruzione serve il risultato di un'istruzione precedente, essa non potrà entrare nella fase EX fino a quando l'istruzione precedente non è giunta alla fase WB.
Un altro problema che si presenta è quello conseguente ai cosiddetti **salti di esecuzione**: blocchi di istruzioni che non vengono eseguite se non sono verificate determinate condizioni; in questo caso il microprocessore passa a eseguire istruzioni memorizzate "distanti" dalle precedenti, con la necessità di svuotare la pipeline prima di eseguire il salto, con conseguente rallentamento dell'esecuzione.

Per ovviare a questi problemi si sono adottate diverse soluzioni:

- utilizzare i cosiddetti registri a doppia porta che mettono a disposizione il risultato per le istruzioni che ne hanno bisogno al termine della fase EX, evitando che tali istruzioni debbano aspettare altre due fasi (MEM e WB) per disporre del dato;
- utilizzare più pipeline autonome in parallelo (tecnica superscalare) consentendo di elaborare molte istruzioni in parallelo;
- introdurre dei circuiti che si occupano di analizzare i possibili salti (unità di predizione delle diramazioni) e di attivare pipeline dedicate all'esecuzione delle istruzioni dopo il salto;
- suddividere l'esecuzione di un'operazione in fasi elementari (20-30) che possono essere eseguite molto rapidamente aumentando la frequenza del clock.

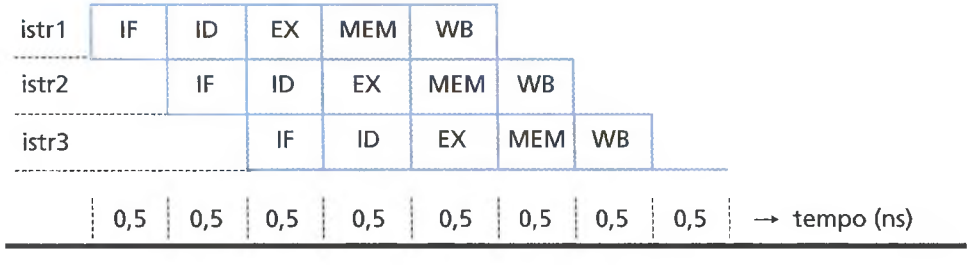
L'introduzione delle CPU multicore nelle architetture hardware ha ridotto i problemi di gestione delle pipeline, perché ogni core lavora autonomamente.

esercizio

→ PROBLEMA

Supponendo di avere una pipeline da 3 elementi (caso A) e che ciascuna fase sia eseguita in 0,5 ns, nell'ipotesi migliore quante operazioni possono essere eseguite al secondo? Aumentando il numero di elementi a 5 (caso B) si hanno dei miglioramenti nella velocità di elaborazione? In quanto tempo dovrebbe essere eseguita una fase della pipeline più lenta tra le 2 per ottenere la stessa velocità di elaborazione?

→ SVOLGIMENTO



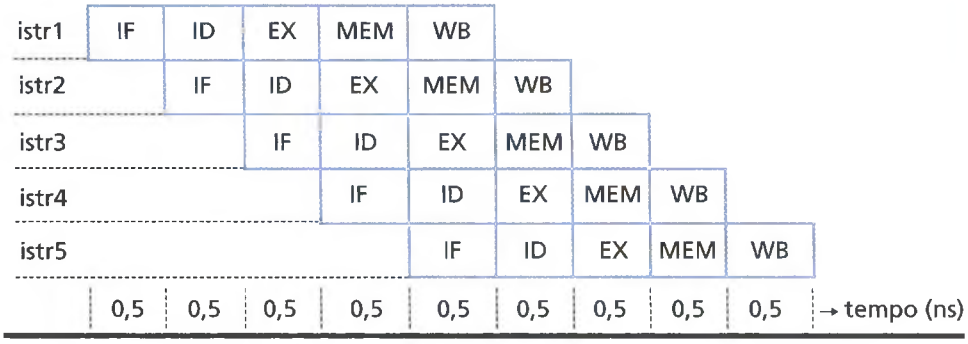
CASO A: Schema pipeline da 3 elementi

Dallo schema si vede che 3 operazioni sono eseguite in 7 fasi quindi in
$T = 7 \cdot 0{,}5 \text{ ns} = 3{,}5 \text{ ns}$.

Quindi la proporzione è:

$3 : 3{,}5 \text{ ns} = x : 1 \text{ s}$

$$x = \frac{3 \cdot 1}{3{,}5 \cdot 10^{-9}} = 857.142.857 = 857{,}1 \text{ \#MIPS}$$



#techwords

Il **MIPS (Million Instructions Per Second)** è un'unità di misura della frequenza di esecuzione delle istruzioni effettuate da un microprocessore.

CASO B: Schema di pipeline da 5 elementi
Dallo schema si vede che 5 operazioni sono eseguite in 9 fasi quindi in:
$T = 9 \cdot 0{,}5 \text{ ns} = 4{,}5 \text{ ns}$

Quindi la proporzione è:

$5 : 4{,}5 \text{ ns} = x : 1 \text{ s}$

$$x = \frac{5 \cdot 1}{4{,}5 \cdot 10^{-9}} = 1.111.111.111 = 1.111 \text{ MIPS}$$

Il miglioramento è di $n = 1.111 - 857{,}1 =$ **253,9 MIPS**

Per ottenere le stesse prestazioni, 1.111 MIPS, con la pipeline da 3 elementi si dovrà ridurre la durata delle fasi.

Calcoliamo il tempo complessivo in cui la pipeline da 3 elementi deve eseguire le istruzioni:

$3 : T = 1.111 \text{ MIPS} : 1 \text{ s}$

$$T = \frac{3 \cdot 1}{1.111 \cdot 10^{-6}} = 0{,}0027 \cdot 10^{-6} = 2{,}7 \text{ ns}$$

Una pipeline da 3 elementi utilizza 7 fasi quindi ogni fase dovrà essere eseguita in
$t_f = 2{,}7 : 7 =$ **0,39 ns** anziché 0,5 ns.

FISSA LE CONOSCENZE

- Che cos'è e come si realizza il pipelining?
- Perché i salti nell'esecuzione delle istruzioni creano problemi alle pipeline?
- Com'è possibile ovviare ai problemi creati dai salti nel codice?

Esercizio commentato
Pipeline

4 I SET DI ISTRUZIONI MACCHINA: CISC E RISC

4.1 Il linguaggio macchina e le architetture

Il **linguaggio macchina** rappresenta l'insieme delle istruzioni macchina che la CPU è in grado di comprendere ed eseguire.
La forma con cui viene rappresentata ogni istruzione macchina è una sequenza di bit, **opportunamente formattata in campi**. A ogni campo sono assegnati un certo numero di bit destinati a contenere dati diversi.
Un esempio di istruzione macchina (o **codice macchina**) in un'architettura a 16 bit con 5 campi è mostrato nella FIGURA 8.

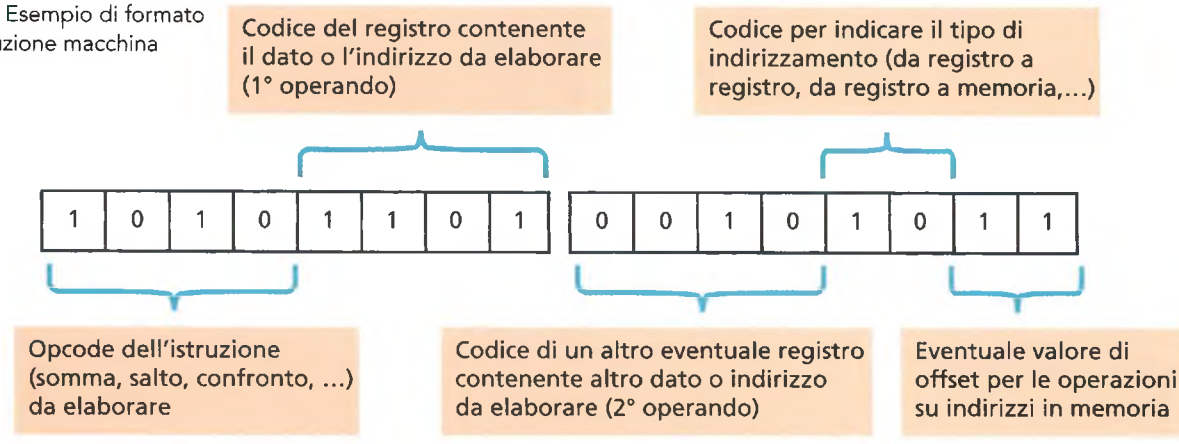


FIGURA 8 Esempio di formato di un'istruzione macchina

#prendinota

Le prestazioni e le potenzialità di una CPU dipendono anche dal numero di operandi per istruzione.
La maggior parte delle istruzioni coinvolge due operandi e produce un risultato.

Il formato dei campi può variare a seconda dell'architettura, ma anche da istruzione a istruzione.
In generale, ogni istruzione macchina deve avere i campi per specificare:

- l'operazione da svolgere (opcode);
- gli operandi coinvolti (o il loro indirizzo).

In genere ogni costruttore di microprocessori (Intel, AMD, AIM) mantiene la **retrocompatibilità**: le istruzioni scritte per i microprocessori meno recenti sono perfettamente comprensibili dai nuovi e da essi eseguite in minor tempo, mentre le istruzioni nuove servono per sfruttare al meglio le nuove potenzialità, ma non possono essere comprese dai microprocessori precedenti.

In base al numero (**massimo**) di operandi che una macchina può elaborare in un'unica istruzione, esistono diverse possibili architetture:

- architettura a un operando;
- architettura a due operandi;
- architettura a tre operandi.

ARCHITETTURA A UN OPERANDO

Opcode	Operando

Il campo **Operando** contiene direttamente l'operando o l'indirizzo dell'operando (registro o memoria). Poiché un'operazione (aritmetica, logica) deve solitamente avere un secondo operando, occorre che l'altro operando sia stato precedentemente caricato in un registro di lavoro dedicato della CPU, che al termine conterrà il risultato.

- **Vantaggi di questa architettura**: molti bit a disposizione per l'operando o per l'indirizzo dell'operando.
- **Svantaggi di questa architettura**: occorre predisporre inizialmente un operando in un registro di lavoro e trasferire poi il risultato alla sua destinazione finale (molto usato in CPU "vecchie" con parola di memoria a 8-16 bit).

ARCHITETTURA A DUE OPERANDI

Opcode	Operando1/Risultato	Operando2

Il campo **Operando1/Risultato** contiene inizialmente il primo operando (o l'indirizzo del primo operando) e alla fine il risultato (o l'indirizzo del risultato) dell'operazione.

- **Vantaggi di questa architettura**: ci sono abbastanza bit a disposizione per gli indirizzi. Inoltre, consente l'immediata elaborazione di entrambi gli operandi senza registri d'appoggio.
- **Svantaggi di questa architettura**: occorre "salvare" il primo operando, se si vuole riutilizzarlo in futuro (alla fine dell'operazione, non è più disponibile nel registro in cui si trovava inizialmente).

ARCHITETTURA A TRE OPERANDI

Opcode	Risultato	Operando1	Operando2

Il campo **Risultato** diventa il terzo operando destinato a contenere il risultato dell'operazione tra gli altri due operandi.

- **Vantaggi di questa architettura**: la programmazione è semplice, perché gli operandi (o i loro indirizzi) e il risultato (o il suo indirizzo) sono **espliciti**.
- **Svantaggi di questa architettura**: il numero di bit per codificare gli indirizzi di ciascun operando e del risultato è limitato.

La scelta dell'architettura rappresenta un punto critico per le prestazioni definitive del microprocessore. L'architettura hardware del microprocessore non può prescindere dall'**Instruction Set Architecture** (**ISA**), cioè da come è stato **strutturato l'insieme delle istruzioni** macchina, e dalle **modalità di indirizzamento** di dati e istruzioni.

#prendinota

L'**ISA** è la parte di architettura che è visibile al programmatore e al compilatore.

4.2 Architetture CISC e RISC

Fin dai primi microprocessori si sono sviluppate due linee di progetto.

- **CISC** (Complex Instruction Set Computer, cioè computer con un insieme di istruzioni complesse): la filosofia è quella di creare tante istruzioni diverse, anche molto complesse, una per ogni possibile operazione che la CPU deve compiere.
  In questa architettura il microprocessore è in grado di eseguire istruzioni complesse, che però richiedono tempi di esecuzione lunghi (la frequenza del clock deve essere piuttosto bassa, il t_{CPU} di un'istruzione risulta alto).

#techwords

Il **Thermal Design Power** (TDP), chiamato anche *Thermal Design Point*, rappresenta un'indicazione del calore (energia) dissipato da un processore, che il sistema di raffreddamento deve smaltire per mantenere la temperatura del processore stesso entro una soglia limite. La sua unità di misura è il watt.

#prendinota

Non conviene puntare sull'ottimizzazione di istruzioni complesse che si usano raramente.
È meglio rendere veloci le istruzioni che si usano più spesso.

La fase di decodifica delle istruzioni è complessa (come sono complesse le istruzioni) e avviene attraverso un programma residente (firmware) nella control unit della CPU, che occupa spazio e aumenta il TDP (#Thermal Design Power).
L'insieme di istruzioni macchina CISC è molto ampio e riduce il divario tra linguaggio macchina e linguaggio ad alto livello. Le architetture CISC sono prevalentemente a due operandi.

- **RISC** (Reduced Instruction Set Computer, cioè computer con un insieme di istruzioni ridotte): la filosofia è quella di ridurre al minimo l'insieme delle istruzioni, selezionando quelle indispensabili e di uso più frequente nei programmi.
  Il microprocessore può eseguire una tipologia abbastanza ridotta di istruzioni base, ma in modo estremamente veloce. Il firmware è molto più semplice rispetto a quello delle architetture CISC e in parte è costituito da circuiti elettronici estremamente veloci (la frequenza del clock può essere piuttosto alta, il t_{CPU} risulta basso).
  La presenza di operazioni inevitabilmente complesse è risolta scomponendole in una sequenza di operazioni più semplici.
  Le architetture RISC sono prevalentemente a tre operandi.

Esiste un limite tecnologico alla dimensione dell'area di silicio su cui è possibile realizzare la microcircuiteria integrata che svolge le funzioni della CPU.

Occorre scegliere tra una elevata complessità della control unit (tipico delle architetture CISC), che riduce lo spazio per i registri, e una ridotta complessità della control unit (tipico delle architetture RISC), che consente di progettare un maggior numero di registri.
Attualmente molti microprocessori elaborano istruzioni di tipo CISC, trasformandole internamente in istruzioni di tipo RISC che poi vengono eseguite.

Per la misura delle prestazioni di un'architettura si usano i **benchmarks**, programmi appositamente studiati per misurare la velocità di elaborazione (per esempio in MIPS, milioni di istruzioni al secondo) nell'eseguire operazioni di vario tipo (grafico, gestionale ecc.).

RISC E PIPELINING

Le architetture RISC, essendo progettate per un numero ridotto di istruzioni semplici che sono più facilmente scomponibili in fasi standardizzate (microoperazioni della durata di un clock), risultano particolarmente adatte per le tecniche di pipelining.

FISSA LE CONOSCENZE

- Che cosa deve sempre contenere ogni istruzione macchina?
- Che cos'è l'ISA (*Instruction Set Architecture*)?
- Descrivi le caratteristiche delle architetture CISC (*Complex Instruction Set Computer*).
- Descrivi le caratteristiche delle architetture RISC (*Reduced Instruction Set Computer*).

5 EVOLUZIONE E CONFRONTO TRA MICROPROCESSORI

5.1 Evoluzione

Il primo a elaborare la struttura di un microprocessore fu il fisico italiano **Federico Faggin** nel 1968. Egli contribuì anche allo sviluppo di vari tipi di microprocessori che hanno caratterizzato la storia dell'informatica; i microprocessori attuali derivano direttamente da quelli inventati da Faggin.

La TABELLA 1 mostra l'evoluzione negli anni dei principali microprocessori e ne riassume le caratteristiche fondamentali.

TABELLA 1 Cronologia dello sviluppo dei microprocessori

ANNO	NOME	NUMERO BIT	CLOCK	NUMERO REGISTRI DATI/NUMERO BIT	NOTE
1970	Intel 4004	4	740 kHz	16/4	Primo microprocessore integrato
1972	Intel 8008	8	800 kHz	16/8	Primo microprocessore a 8 bit
1975	MOS 6502	8	2 MHz	1 + memoria interna da 256 celle/8	Primi home computer Commodore VIC20/Apple II
1976	Zilog Z80	8	8 MHz	7(+7)/8	Registri raddoppiati internamente
1976	TMS9900	16	3,3 MHz	16/16 nella RAM esterna	Home computer TI-99
1978	Intel 8086	16	10 MHz	8/16	Primi microcomputer (personal computer M24 Olivetti)
1982	Intel 80286	16	20 MHz	8/16	PC IBM
1982	Motorola 68000	32	10 MHz	8/32	Macintosh
1986	Intel 80386	32	40 MHz	8/32	Gestione Sistemi Operativi con memoria virtuale
1989	Intel 80486	32	100 MHz	8/32	Miglioramento del 386
1991	AMD386	32	100 MHz	8/32	Compatibile con Intel 386
1993 1997 1999	Pentium 1, 2, 3	32	60 MHz - 80 MHz	8/32	Introdotte tecniche per la pre-elaborazione delle istruzioni e cache L1 e L2 integrate.
1995 1997 1999	AMD K5 - K6 - Athlon	32	133 MHz - 1,4 GHz	8/32	Prestazioni comparabili con i Pentium, ma architettura diversa
2000 2002	Pentium 4 Power PC G5	32/64	1,3 - 3,8 GHz 2 GHz	8/32	Migliorata la pre-elaborazione cache integrata fino a livello L3
2003	Athlon64	64	800 MHz - 1 GHz	8/64	Prestazioni comparabili con i Pentium 4, ma architettura diversa e frequenza di clock più bassa
2005	Pentium D (Intel) Core I3 Opteron (AMD)	64	3,6 GHz 3,2 GHz	8/64	Dual-core. Cache L1 e L2 Core i3 con L1, L2 e L3

ANNO	NOME	NUMERO BIT	CLOCK	NUMERO REGISTRI DATI/NUMERO BIT	NOTE
2006	Core 2 Quad Core i5 (Intel) Phenom (AMD)	64	2,66 GHz	8/64	Quad-core. Cache L1, L2, L3
2010	Core i7 Extreme Phenom 2 X6	64	3,3 GHz	8/64	Esa-core. Cache L1, L2, L3
2014	Core i7 - 5960X Extreme	64	3,3 GHz	8/64	Octa-core. Cache L3 shared
2019	Core i9-9900KS	64	4,0 GHz	8/64	Octa-core. Cache L3 Smart Cache
2020	Core i9-10980HK	64	5,3 GHz	8/64	Octa-core. Cache L3 Smart Cache

Le principali specifiche tecniche da controllare nella scelta di un microprocessore sono le seguenti.

- **Frequenza**: misurata in gigahertz (GHz), è la velocità alla quale opera il chip, quindi più è alta e più il chip opera velocemente. Le CPU moderne aumentano o riducono la loro frequenza in base all'operazione e alla loro temperatura; viene indicata perciò una frequenza base (minima) e una turbo (massima).
- **#Core**: le CPU moderne offrono da 2 a 64 core, la maggior parte ne contiene da 4 a 8. Ogni core è in grado di gestire le proprie operazioni.
- **Thread**: è il numero di processi indipendenti che un chip può gestire alla volta e in teoria dovrebbe essere lo stesso numero dei core. Tuttavia molti processori hanno capacità multithreading, che consentono a un singolo core di creare due thread. Intel chiama questa capacità Hyper-Threading, mentre AMD la chiama SMT (Simultaneous Multithreading). Più thread significa miglior multitasking e prestazioni migliorate.
- **TDP** (Thermal Design Power): è la quantità massima di calore che un chip genera, ed è misurata in watt. Sapendo, per esempio, che il Core i7-8700K ha un TDP di 95 W, potete assicurarvi di avere un dissipatore (ventola di raffreddamento) capace di gestire quella quantità di calore dissipato e anche l'alimentatore in grado di fornire l'alimentazione adeguata. Un TDP elevato di solito coincide anche con maggiori prestazioni.
- **Cache**: la cache a bordo del processore è usata per velocizzare l'accesso ai dati e le istruzioni tra la CPU e la RAM. Ci sono tre tipi di cache: la cache L1 è la più veloce ma è limitata, la L2 è maggiore ma è più lenta, la L3 è ancora maggiore ma comparativamente lenta. Quando il dato di cui ha bisogno una CPU non è disponibile in nessuna di queste cache, allora lo si preleva dalla RAM, che però è molto più lenta.
- **IPC** (Instructions per Clock): CPU che hanno la stessa frequenza e lo stesso numero di thread, se arrivano da aziende diverse o sono costruite su diverse architetture della medesima azienda, produrranno un IPC diverso. L'IPC è pesantemente dipendente dall'architettura della CPU, di conseguenza i chip di più recente generazione sono migliori.

La grande diffusione di computer portatili (notebook, netbook, laptop) ha portato le aziende produttrici di microprocessori a puntare su chip adatti a questi dispositivi.

#techwords

Core è un termine hardware che descrive il numero di CPU indipendenti, presenti in un unico dispositivo integrato, ovvero il numero di nuclei di processori montati sullo stesso package.

#prendinota

Negli ultimi anni sia AMD sia Intel hanno spinto molto sul numero di core, arrivando a 8-16 core per le CPU mainstream (Core i9-9900K, Ryzen 9 3950X) e a ben 32-64 core per le soluzioni destinate alle workstation (Intel Xeon 7210, Threadripper 3970X, Threadripper 3990X).

Nella FIGURA 9 è mostrato il die-mapping degli Intel 4C/8T (4 core e 8 thread) di decima generazione (Ice Like) progettato per i notebook con processori i3, i5 e i7.

Un **die**, in elettronica digitale, è la sottile piastrina di materiale semiconduttore sulla quale è stato realizzato il circuito integrato. Il die viene sigillato in un contenitore, chiamato col termine inglese **package**.

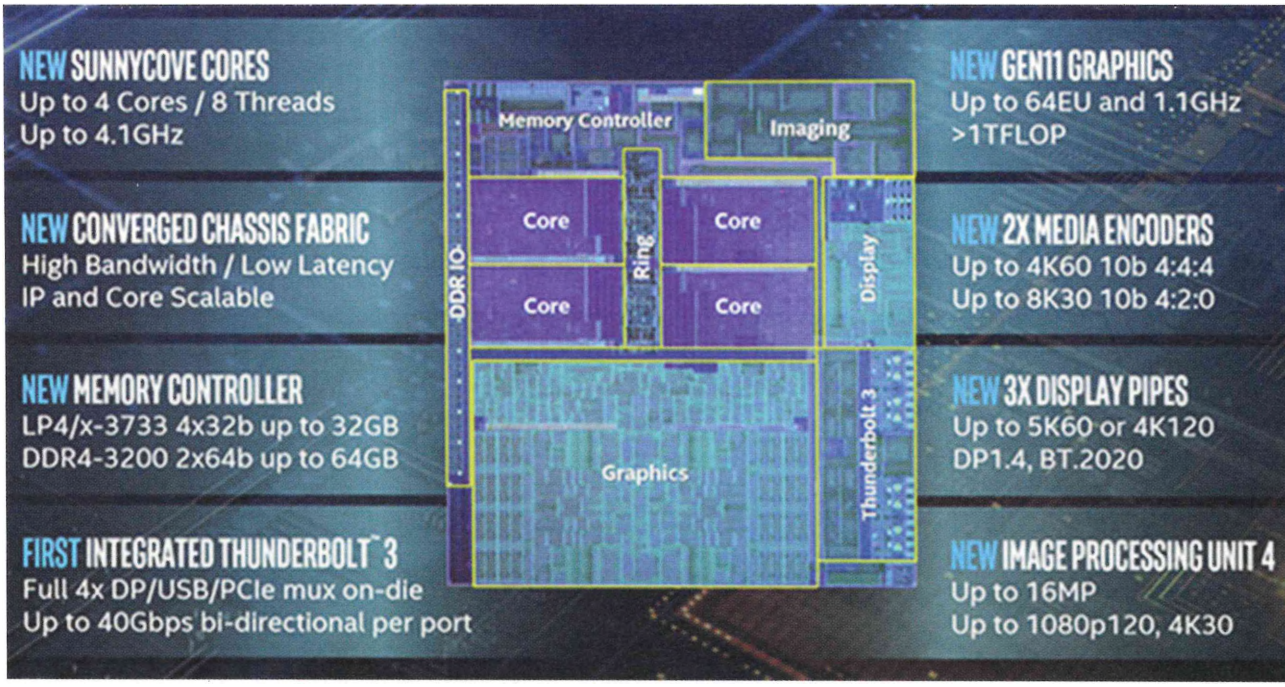


FIGURA 9 Die-mapping Intel i3, i5 e i7 4C/8T Ice Like

SunnyCove è il nome scelto per indicare i nuovi core, mentre *Ice Lake* è il nome in codice che identifica la prima famiglia di processori della gamma Core che sono basati su questi core. Le novità implementate in questi processori sono significative e hanno permesso di ottenere un aumento dell'IPC dal 15% al 18% rispetto alla precedente generazione.

Il grafico nella FIGURA 10, presentato da Intel, mostra gli incrementi delle prestazioni in single thread dei processori Intel Core passando da Sky Lake (Intel Core di

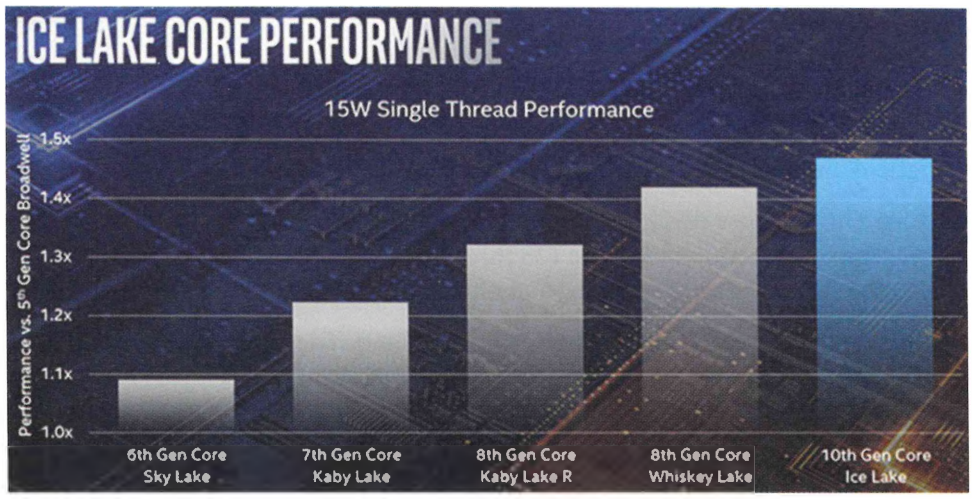


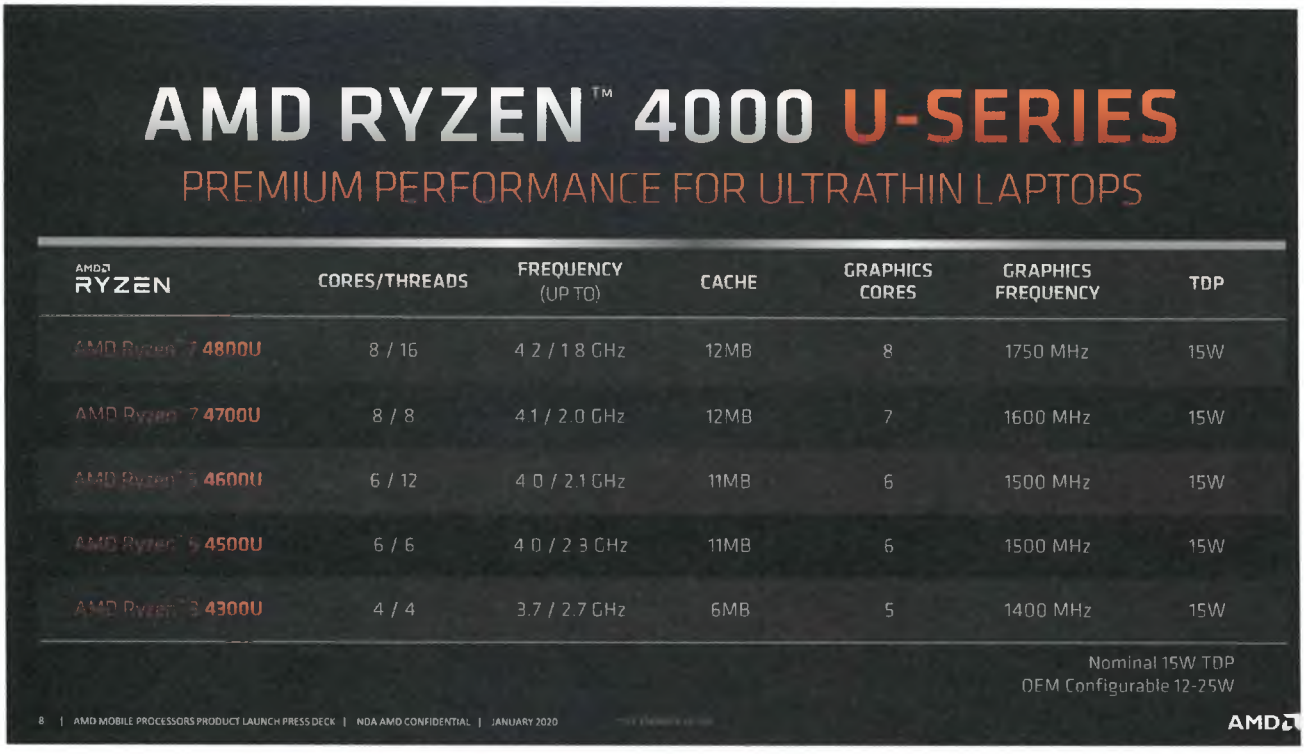
FIGURA 10 Prestazioni in single thread dei processori Intel Core

sesta generazione) sino a Ice Lake (Intel Core di decima generazione): cinque generazioni durante le quali sono stati registrati interessanti incrementi delle performance, giustificate sia dalle novità architetturali sia dall'incremento nelle frequenze di clock massime in single thread.
Tutto questo a parità di consumo, con un TDP di 15 W che è quello tipico dei processori installati in sistemi notebook.

L'AMD, azienda che con Intel si spartisce la maggior parte del mercato dei microprocessori per desktop e laptop, produce gli AMD Ryzen 3000 con le serie 3, 5, 7, 9 e di recente ha presentato la gamma di processori AMD Ryzen 4000 Mobile.

Le specifiche di consumo dei modelli appartenenti alla serie 4000 ben si adattano ai notebook: le CPU caratterizzate dalla sigla H hanno un TDP di 45 W, mentre quelle contrassegnate con la sigla HS hanno un TDP di 35 W. La serie U, invece, ha un TDP ridotto da 15 W.
Nella FIGURA 11 sono evidenziate le specifiche della serie 4000 U.

FIGURA 11 Specifiche della serie AMD RYZEN 4000 U



#techwords

I **chiplet** sono piccoli blocchi base personalizzabili e utilizzabili più volte (analogamente ai mattoncini Lego) che possono diventare processori, ricetrasmettitori, memoria o altri tipi di componenti.

AMD realizza anche microprocessori per workstation con la serie **Ryzen Threadripper**. Si tratta di microprocessori dotati di un alto numero di core e di thread.

Il Ryzen Threadripper 3990X, punta di diamante dell'ultima generazione di processori Threadripper, è dotato di 64 core e 128 thread. All'interno di questa CPU troviamo, infatti, **8 #chiplet attivi**, ognuno dei quali comprende 8 core e 32 MB di cache L3.

Il mercato dei dispositivi mobili (smartphone, tablet) vede leader del mercato ARM (Advanced RISC Machine). ARM non si basa sulla vendita di chip ma sulla progettazione di CPU e GPU dove i progetti vengono poi venduti, con diversi tipi di licenza, a chi deve produrre il SoC, System on Chip, il chip fisico. A differenza delle CPU Intel che sono CISC, i SoC ARM sono di tipo RISC.

esercizio

→ PROBLEMA

Osservando la seguente tabella, quale dei due microprocessori è più adatto per un dispositivo portatile? Motivare la scelta.

Nome prodotto	Intel® Core™2 Quad Processor (12 MB Cache, 3,00 GHz, 1333 MHz FSB)	Intel® Core™2 Quad Processor (12 MB Cache, 2,26 GHz, 1066 MHz FSB)
Numero del processore	Q9650	Q9100
Numero di core	4	4
Velocità di clock	3 GHz	2,26 GHz
Cache	12 MB L2 cache	12 MB L2 cache
Bus di sistema	1.333 MHz	1.066 MHz
Set di istruzioni	64-bit	64-bit
Litografia	45 nm	45 nm
TDP massimo	95 W	45 W
Dimensione package	37,5 mm × 37,5 mm	35 mm × 35 mm
Dimensione die di elaborazione	214 mm^2	214 mm^2
Numero di transistor su die di elaborazione	820 million	820 million
Socket supportati	LGA775	PGA478

→ SVOLGIMENTO

Una delle esigenze più pressanti per un dispositivo portatile è la durata della batteria, che si ottiene ottimizzando e limitando i consumi di energia elettrica. Il parametro che misura il consumo di energia elettrica è il TDP (Thermal Dissipation Power). Quindi il più adatto è quello a frequenza minore ma con un TDP di soli 45 W.

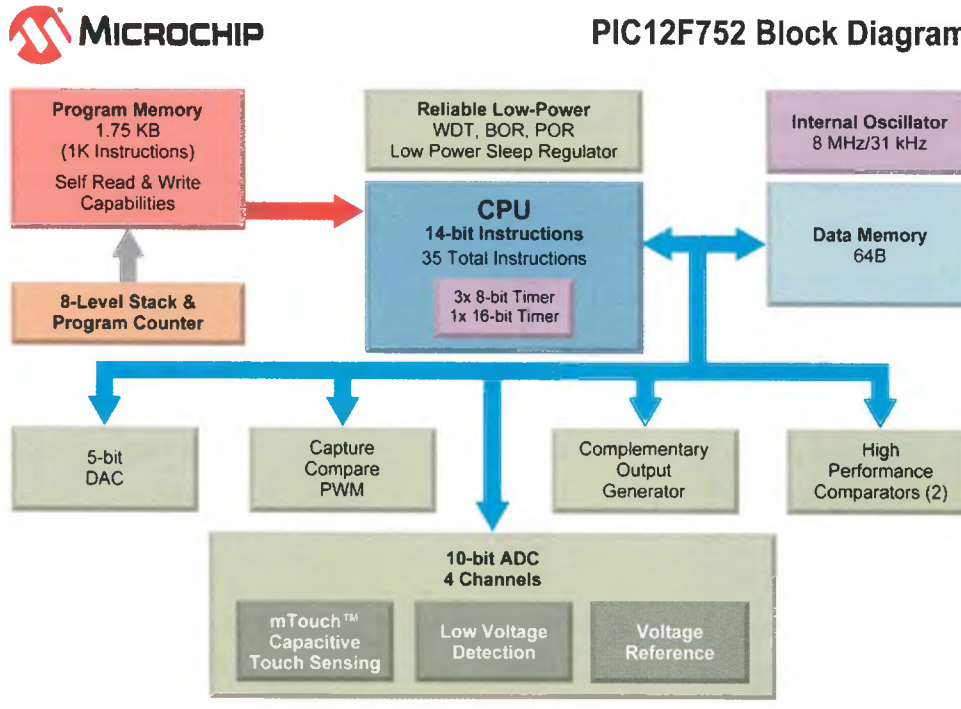
5.2 I microcontrollori

Negli ultimi anni si è diffuso l'uso di microcontrollori all'interno di dispositivi ed elettrodomestici di uso comune (lavatrici, macchine per il caffè, navigatori satellitari, antifurti, robot aspirapolvere, smartphone, apparecchiature mediche, periferiche per videogiochi, ...), consentendo così di realizzare dispositivi "intelligenti", in grado cioè di avere un funzionamento molto variabile in base alle esigenze dell'utente.

I microcontrollori sono microprocessori con prestazioni ridotte, a basso costo e con bassi consumi, che al loro interno integrano circuiti elettronici in grado di semplificare il controllo dei dispositivi che formano, per esempio, l'elettrodomestico (pulsanti, motori, display ecc.).

Tali microcontrollori sono normalmente programmati in un linguaggio assembly specifico: le istruzioni sono simili a quelle dei microprocessori, ma sono in quantità estremamente ridotta e sono disponibili anche istruzioni specifiche che consentono un più semplice controllo dei segnali in ingresso e in uscita per il funzionamento degli apparati (FIGURA 12).

FIGURA 12 Schema a blocchi di un microcontrollore



esempio

Caratteristiche di un microcontrollore (PIC12F752)

ISA: 35 istruzioni
Memoria flash da 1,75 kB
Clock: 8 MHz
64 registri da 8 bit
1 generatore di forme d'onda
4 generatori a impulso variabile (PWM)
2 comparatori analogici
Modulo indicatore di temperatura
4 convertitori analogico-digitali
1 convertitore digitale-analogico
3 temporizzatori a 8 bit
1 temporizzatore a 16 bit

FISSA LE CONOSCENZE

- Che cosa misura il TDP?
- Che cosa misura l'IPC?
- Che cosa si intende per *core*?
- Che cos'è un thread?
- Che cosa sono i microcontrollori?

6 IL LINGUAGGIO ASSEMBLY INTEL X86

6.1 Il linguaggio assembly

Il codice macchina può essere visto in una forma equivalente attraverso un **linguaggio mnemonico**, dunque comprensibile all'uomo, al contrario delle sequenze di bit che sono comprensibili solo al microprocessore.

esempio

codice macchina: 1010 0101 0011 1010 (linguaggio macchina)
codice mnemonico: MOV R0, R1 (linguaggio assembly)

L'insieme dei codici mnemonici di tutte le istruzioni prende il nome di **linguaggio assembly**.

Il linguaggio assembly prevede l'uso di parole chiave: MOV per assegnare valori a registri, ADD per sommare i contenuti di registri o aree di memoria, SUB per sottrarre e così via.
Il processo che traduce il codice assembly nel corrispondente codice macchina eseguibile direttamente dal microprocessore si compone di due passaggi:

- **ASSEMBLER**: è un processo di traduzione che non richiede alcuna intelligenza; esiste il processo inverso, detto disassemblaggio o disassembly;
- **LINKER**: serve a collegare moduli e librerie di cui si compone il programma e a distribuire il codice oggetto nello spazio di indirizzi di memoria centrale assegnato al programma.

#prendinota

Il linguaggio macchina e il linguaggio assembly stanno in rapporto 1 : 1 (a una istruzione macchina corrisponde una istruzione assembly).
Il linguaggio macchina e i linguaggi ad alto livello stanno in rapporto N : 1 (ci vogliono N istruzioni macchina per fare una istruzione evoluta).

Il loro compito è mostrato nella FIGURA 13.

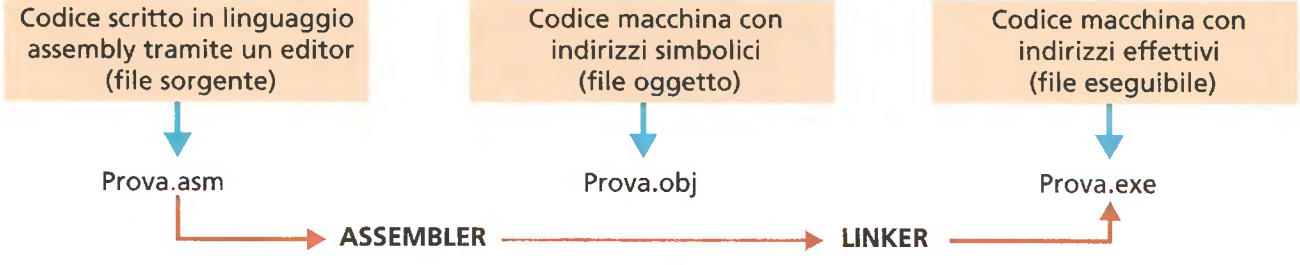


FIGURA 13 Passaggio da linguaggio assembly a codice macchina eseguibile

Il legame tra le istruzioni assembly e il codice macchina è dunque strettissimo e poiché, come sappiamo, le architetture variano a seconda degli operandi (uno, due o tre), il linguaggio assembly prevede istruzioni a uno, due o tre operandi (ma anche a zero).

Un'istruzione assembly (FIGURA 14) di un generico processore Intel x86 è formata da:

- un'etichetta opzionale (**label**), che assegna un nome simbolico all'indirizzo di memoria in cui si troverà l'istruzione;
- un codice operativo (**opcode**) mnemonico obbligatorio, che specifica l'operazione da svolgere;
- zero, uno, due o tre **operandi separati da una virgola**, che specificano i dati su

memoria riservata nello stack segment per le subroutine. I registri BP e SP lavorano quindi in coppia col registro SS.

#prendinota

L'Intel 8086 ha 5 coppie celebri, cioè registri che lavorano in coppia:
- CS:IP
- SS:BP
- DS:DI
- SS:SP
- DS:SI

Ogni coppia concorre alla generazione dell'indirizzo di memoria delle istruzioni macchina o dei dati o delle subroutine.

6.3 Il set di istruzioni base dell'assembly dei processori Intel x86

Le istruzioni possono essere classificate in base al loro uso. Elenchiamo le principali istruzioni per ogni tipo, con alcuni esempi di istruzioni usate nella prossima Unità per scrivere semplici algoritmi.

1. Istruzioni di utilizzo generale
- **MOV**: per spostare dati da memoria a CPU e viceversa;
- **PUSH**: per inserire un dato in cima allo stack;
- **POP**: per prelevare un dato dalla cima dello stack;
- **XCHG**: per scambiare il contenuto di due registri o di una locazione di memoria e un registro.

#prendinota

Tutte le istruzioni mettono il risultato nel primo operando: MOV destination, source.

esempio

MOV AX,BX: sposta (copia) il contenuto di BX in AX.
XCHG AX,BX: scambia il contenuto tra AX e BX.

2. Istruzioni di Input/Output (I/O)
- **IN**: trasferisce un byte o una #word da una porta di I/O ad AL (se si tratta di un byte) o ad AX (se si tratta di una word);
- **OUT**: trasferisce un byte o una word da AL (se si tratta di un byte) o da AX (se si tratta di una word) a una porta di I/O.

#techwords

Con **word** (parola) si intende un tipo di dato a 16 bit. Viene usato insieme ad altri tipi di dato: nibble (4 bit), byte (8 bit), dword o double word (32 bit), qword o quad word (64 bit).

3. Istruzioni per operazioni aritmetiche
- **ADD**: addizione;
- **ADC**: addizione con riporto (carry);
- **SUB**: sottrazione;
- **SBB**: sottrazione con prestito (borrow);
- **MUL**: moltiplicazione;
- **DIV**: divisione;
- **INC**: incremento di uno;
- **DEC**: decremento di uno;
- **NEG**: negazione (complemento a 2);
- **CMP**: confronta i due operandi con una sottrazione e setta i flag di stato ZF e SF.

#prendinota

Tutte le istruzioni provocano la modifica dei flag del registro di stato oltre alla modifica degli operandi coinvolti. La **CMP** (*compare*) è l'unica istruzione che modifica **solo** i flag e non gli operandi.

esempio

ADD AX,BX: somma il contenuto di AX e BX e mette il risultato in AX.
INC AX: incrementa di 1 il contenuto di AX (istruzione a un operando).
CMP AX,BX: confronta il contenuto di AX con quello di BX con la sottrazione AX–BX e setta i flag.

4. Istruzioni per operazioni logiche
- **AND**: and logico;
- **NOT**: not logico (complemento a 1);

- **OR**: or logico;
- **XOR**: esclusive-or logico.

5. Istruzioni per operazioni di rotazione e shift

- **ROL**: per la rotazione a sinistra;
- **ROR**: per la rotazione a destra;
- **SHL**: per lo shift a sinistra;
- **SHR**: per lo shift a destra.

6. Istruzioni per operazioni di trasferimento non condizionato

- **CALL**: per la chiamata di una subroutine (istruzione a un operando);
- **RET**: per il ritorno da una subroutine (istruzione a zero operandi);
- **JMP**: per salti (jump) incondizionati (istruzione a un operando, che è una label).

esempio

CALL Calcola: chiama l'esecuzione della subroutine Calcola.
RET: se è posta come ultima istruzione di una subroutine, la chiude e ritorna al main.
JMP inizio: obbliga a saltare all'istruzione etichettata con la label "inizio".

#prendinota

Tutte le istruzioni di salto **condizionato** devono sempre essere **immediatamente precedute** dall'istruzione **CMP**, che effettua la sottrazione tra i due operandi e setta i flag **ZF** e **SF** del registro di stato.
Le varie **jump** testano i due flag e, in base al valore letto (1 o 0), decidono se saltare (bit a 1) o proseguire (bit a 0) in sequenza.

7. Istruzioni per operazioni di trasferimento (salto) condizionato

- **JG** (Jump if Greater): salta se il primo operando è maggiore del secondo;
- **JGE** (Jump if Greater or Equal): salta se il primo operando è maggiore o uguale al secondo;
- **JL** (Jump if Less): salta se il primo operando è minore del secondo;
- **JLE** (Jump if Less or Equal): salta se il primo operando è minore o uguale al secondo;
- **JE** (Jump if Equal): salta se il primo operando è uguale al secondo;
- **JNE** (Jump if Not Equal): salta se il primo operando è diverso dal secondo.

esempio

CMP AX,BX: confronta il contenuto dei due registri e setta i flag.
JE fine-ciclo: salta alla label "fine-ciclo" se il contenuto dei due registri è uguale.

#prendinota

Le istruzioni di tipo **LOOP** permettono il controllo dei cicli con l'appoggio del registro **CX** (*Count Register*), che viene **automaticamente decrementato di uno** a ogni ciclo. Occorre inizializzare CX al numero di cicli desiderati e, quando CX = 0, il ciclo finisce. Con queste istruzioni si possono realizzare i costrutti iterativi come il **for**.

8. Istruzioni per il controllo dei cicli

- **LOOP**: esegue il ciclo se CX è diverso da 0;
- **LOOPE**: preceduta dalla CMP, esegue il ciclo se CX è diverso da 0 e il flag Z = 0;
- **LOOPNE**: preceduta dalla CMP, esegue il ciclo se CX è diverso da 0 e il flag Z = 1.

9. Istruzione per caricare gli indirizzi

- **LEA** (Load Effective Address): opera sugli indirizzi e consente quindi di caricare in un registro l'indirizzo di una variabile di memoria (e non il valore della variabile). Il suo utilizzo è tipico nelle stringhe per caricare l'indirizzo iniziale.

esempio

LEA DI,stringa1: carica l'indirizzo iniziale della stringa1 in DI.

6.4 Gli interrupt dei processori Intel x86

I programmi assembly, come i programmi scritti in linguaggio ad alto livello, consentono l'input/output dei dati mediante tastiera e video.
L'I/O coinvolge le periferiche, che comunicano con la CPU (chiedono la sua attenzione) mediante degli interrupt.

Un **interrupt** è un evento che interrompe il normale funzionamento della CPU. In risposta a un interrupt, la CPU sospende l'esecuzione del programma corrente ed esegue la routine di gestione dell'interrupt: **interrupt handler**. Al termine dell'interrupt handler, la CPU riprende l'esecuzione del programma interrotto. La presenza di un eventuale interrupt è identificata alla fine di ogni ciclo macchina.

I processori x86 gestiscono fino a 256 interrupt identificabili con un numero N da 0 a 255 in decimale (o da 00h a FEh in esadecimale) e richiamabili via software mediante l'istruzione assembly **INT** N.
Molti interrupt sono in grado di eseguire più servizi, cioè di far eseguire interrupt handler diversi, a seconda del contenuto precedentemente caricato nel registro di lavoro **AH** (parte alta di AX).
Nella TABELLA 2 è riportato l'elenco dei principali interrupt attualmente supportati dall'emulatore del DOS nei sistemi operativi Windows.

TABELLA 2 Servizi resi da principali interrupt

INTERRUPT	AH	AZIONE (ED EVENTUALI ALTRI REGISTRI DA CARICARE)	RETURN
INT 10h	03h	*Get* della posizione del cursore	DH = row DL = column
INT 10h	08h	*Read* di un carattere da tastiera (con echo)	AL = ASCII character
INT 10h	06h	*Scroll up* window con AL = numero linee di scroll (AL = 00h pulisce l'intera finestra DOS)	
INT 10h	07h	*Scroll down* window con AL = numero linee di scroll (AL = 00h pulisce l'intera finestra DOS)	
INT 10h	0Ch	*Set* del colore di un singolo pixel con: AL = pixel color CX = column DX = row	
INT 10h	0Dh	*Get* del colore di un singolo pixel con: CX = column DX = row	AL = pixel color
INT 10h	0Ah	*Write* a video di un carattere a video con AL = ASCII character alla posizione del cursore	
INT 10h	13h	*Write* a video di una stringa a video a partire dalla posizione: DL = column DH = row	

→

INTERRUPT	AH	AZIONE (ED EVENTUALI ALTRI REGISTRI DA CARICARE)	RETURN
INT 16h	00h	*Read* di un carattere da tastiera (senza echo), dopo aver pulito il buffer della tastiera	AL = ASCII character
INT 20h	----------	*Exit* dal programma e restituisce il controllo al Sistema Operativo	
INT 21h	01h	*Read* di un carattere da tastiera (con echo), aspettando finché non viene inserito	AL = ASCII character
INT 21h	02h	*Write* a video di un carattere con DL = ASCII character da scrivere	AL = ASCII character (copia DL in AL)
INT 21h	09h	*Write* a video di una stringa in memoria terminante con "$". Occorre caricare l'indirizzo di memoria della stringa in DX mediante l'istruzione LEA DX,stringa	
INT 21h	2Ah	*Get* system date	CX = year (1980-2099) DH = month DL = day AL = day of week (00h = Sunday)
INT 21h	2Ch	*Get* system time	CH = hour CL = minute DH = second DL = 1/100 seconds
INT 21h	4Ch	*Exit* dal programma e restituisce il controllo al Sistema Operativo	
INT 33h	AX=0000h	*Initialization* del mouse	
INT 33h	AX=0003h	*Get* della posizione del mouse e dello stato dei button	if left button is down: BX = 1 if right button is down: BX = 2 if both buttons are down: BX = 3 CX = row DX = column

FISSA LE CONOSCENZE

- È mnemonico il linguaggio macchina o quello assembly? Perché?
- Da quali campi è formata un'istruzione assembly?
- Quali sono e che cosa indicano i tre flag di controllo del registro di stato?
- Perché le istruzioni *jump* condizionate devono essere precedute dall'istruzione *compare*?
- A quale registro si appoggia l'istruzione *loop* per eseguire il numero di cicli previsti?
- Che cosa fa la CPU dopo aver ricevuto un interrupt?
- Che cos'è uno stack?

7 I METODI DI INDIRIZZAMENTO INTEL X86

7.1 L'indirizzamento

Un altro dei punti cruciali nella progettazione di un'architettura hardware è costituito dalle modalità con cui il microprocessore consente di accedere a un dato o a un'istruzione, contenuti in un registro o in memoria centrale.
In base ai metodi di indirizzamento previsti, le istruzioni del linguaggio macchina possono assumere formattazioni diverse.

Per comprendere le modalità di indirizzamento, occorre avere chiari tre concetti che sono indispensabili alla CPU per dialogare con la memoria centrale: base address, offset e displacement.

- Con **base address** si intende l'indirizzo della cella iniziale, a partire dalla quale è stato caricato in memoria centrale il blocco di codice (dati e istruzioni) da elaborare. La maggior parte delle architetture prevede la presenza di un **registro Base** (per esempio **BX**) per contenere il base address.
- Con **offset** si intende la distanza da sommare al base address per puntare l'istruzione o il dato da elaborare. La maggior parte delle architetture prevede la presenza di uno o più **registri Index** (per esempio **SI** e **DI**) per contenere l'offset.
- Con **displacement** si intende l'ulteriore spostamento (detto anche "spiazzamento") rispetto alla base e all'offset per puntare l'istruzione o il dato da elaborare. La maggior parte delle architetture prevede che il displacement sia esplicitato direttamente nell'istruzione macchina sotto forma di valore numerico.

L'operazione seguente:

base address + offset + displacement = effective address

permette di calcolare l'indirizzo di memoria centrale reale (effective address), dove è possibile reperire il dato o l'istruzione da elaborare.
Le istruzioni più complesse possono prevedere la presenza di tutti e tre i valori.

Il caso è mostrato nella FIGURA 18, dove nella memoria centrale ogni riga di codice macchina è costituita da 16 bit pari a 4 cifre esadecimali.

Per leggere il codice macchina indirizzato, la CPU deve partire dall'indirizzo base del blocco di memoria riservato al programma, saltare all'indirizzo calcolato tramite l'offset e, infine, spostarsi di tante celle di memoria quante sono indicate nel displacement.

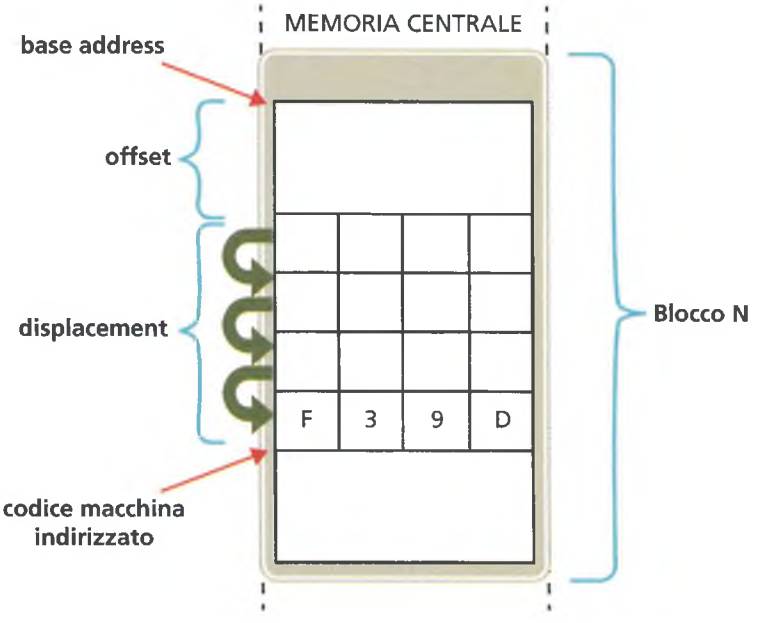


FIGURA 18 Indirizzamento con base, offset e displacement

Per esemplificare ogni metodo, faremo uso dell'istruzione assembly **MOV** (in un'architettura a due operandi e a 16 bit), che opera il trasferimento di un dato da una sorgente a una destinazione, la cui struttura sintattica di base è:

MOV destination,source

Si noti che i due operandi, separati da una virgola, rappresentano registri o indirizzi di memoria e, quindi, il significato dell'istruzione è:

"sposta (copia) in **destination** il dato che trovi in **source**".

I due operandi possono quindi indifferentemente essere registri della CPU o locazioni della memoria centrale (con qualche eccezione che vedremo).
I metodi di indirizzamento si suddividono in tre tipi, in base al modo in cui l'istruzione tratta il dato da elaborare:

- **immediato**: nell'istruzione c'è il dato;
- **diretto**: nell'istruzione c'è l'indirizzo di memoria o il registro in cui si trova il dato;
- **indiretto**: nell'istruzione c'è l'indirizzo di memoria o il registro dove si trova l'indirizzo di memoria in cui c'è il dato. In questo tipo di indirizzamento il dato finale può essere solo in memoria e non in un registro o nell'istruzione.

A loro volta, inoltre, i metodi di indirizzamento possono presentarsi in diverse modalità. La FIGURA 19 riassume le sette modalità fondamentali realizzate da Intel per i processori x86.

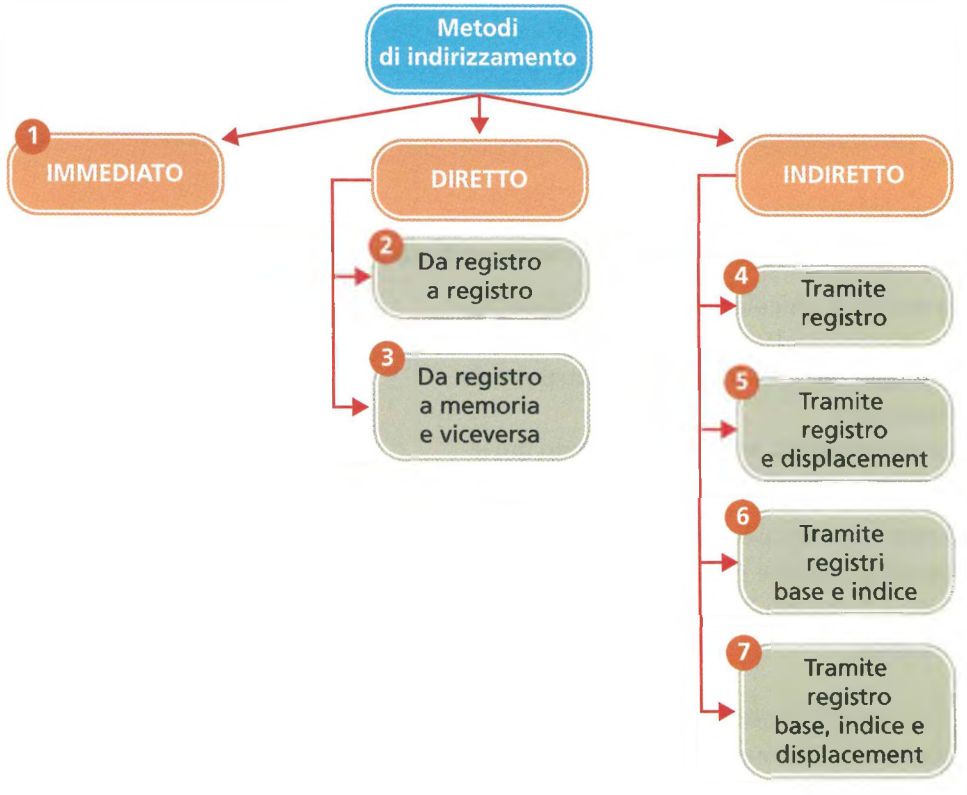


FIGURA 19 I sette principali metodi di indirizzamento dell'Intel x86

Esaminiamo ora ciascuno dei sette metodi di indirizzamento; ci sarà utile per capire la differenza tra un dato e un indirizzo e per comprendere come si muovono sui bus i dati e gli indirizzi, all'interno di un'architettura hardware complessa, passando dalla memoria centrale ai registri della CPU e viceversa.

7.2 Indirizzamento immediato

#prendinota

A differenza di tutte le altre modalità, questa non consente l'inversione degli operandi, poiché non avrebbe senso.

Il dato da scrivere è direttamente specificato nel secondo operando dell'istruzione:

MOV AL,4Dh

L'esecuzione di questa istruzione determina, in un unico passo, la scrittura del valore "4D", espresso in esadecimale (h), in AL, ovvero nella parte bassa del registro Accumulator AX (FIGURA 20).
È quindi un'istruzione che non usa né base, né offset, né displacement.

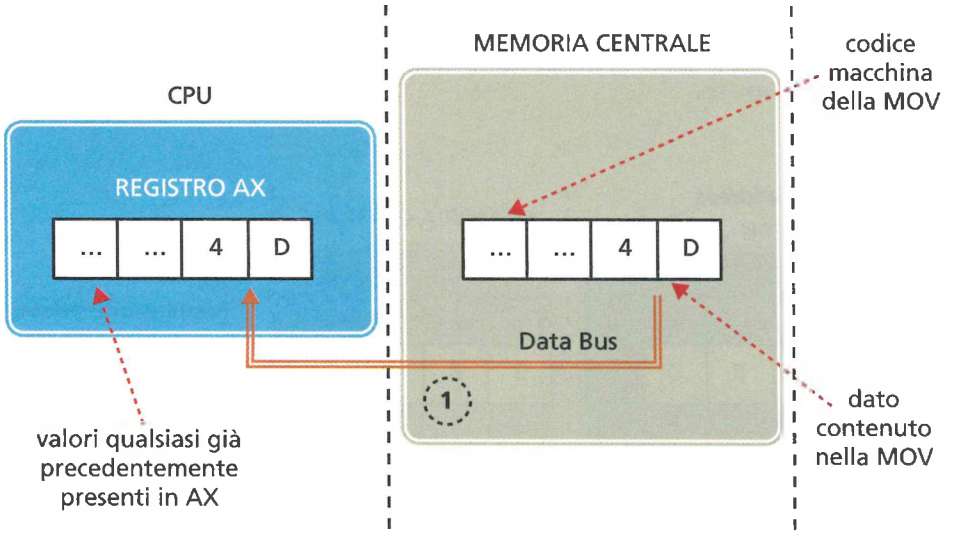


FIGURA 20 Indirizzamento immediato con istruzione MOV AL,4Dh

I registri di lavoro come AX possono, in molte architetture, essere utilizzati anche solo nella parte alta o nella parte bassa (nella Figura 20 è usata solo la parte bassa AL).

7.3 Indirizzamento diretto da registro a registro

Entrambi gli operandi sono registri, rispettivamente destination e source:

MOV AX,BX

L'esecuzione di questa istruzione provoca, in un unico passo, la copiatura del contenuto del registro base BX nel registro accumulatore AX (FIGURA 21).

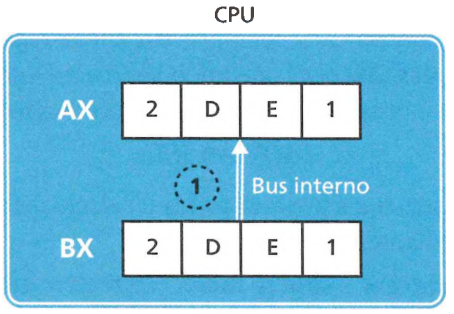


FIGURA 21 Indirizzamento diretto con istruzione MOV AX,BX

Invertendo gli operandi, cioè scrivendo **MOV BX,AX**, il contenuto di AX verrebbe copiato in BX. Questo metodo di indirizzamento non coinvolge la memoria centrale e i bus esterni; l'esecuzione avviene tutta nella CPU dove si trovano i registri. È quindi un'istruzione che non usa né base, né offset, né displacement.

#prendinota

La presenza delle **parentesi quadre** indica che il valore scritto al loro interno è un indirizzo e che il dato che serve all'ALU si trova a quell'indirizzo. In altre parole, [6A] è un indirizzo e non un dato.
In particolare, rappresenta l'**offset** che compone l'indirizzo per individuare il dato in memoria centrale.

7.4 Indirizzamento diretto da registro a memoria e viceversa

Un operando è un registro, mentre l'altro è una locazione di memoria centrale specificata direttamente dall'istruzione mediante l'offset (tra parentesi quadre):

MOV AX,[6Ah]

L'esecuzione di questa istruzione determina prima il calcolo dell'indirizzo in cui si trova il dato realmente, mediante l'operazione:

base address + offset = effective address

Il secondo passo è la copiatura del valore che si trova in memoria (all'offset "6A") nel registro AX (FIGURA 22).

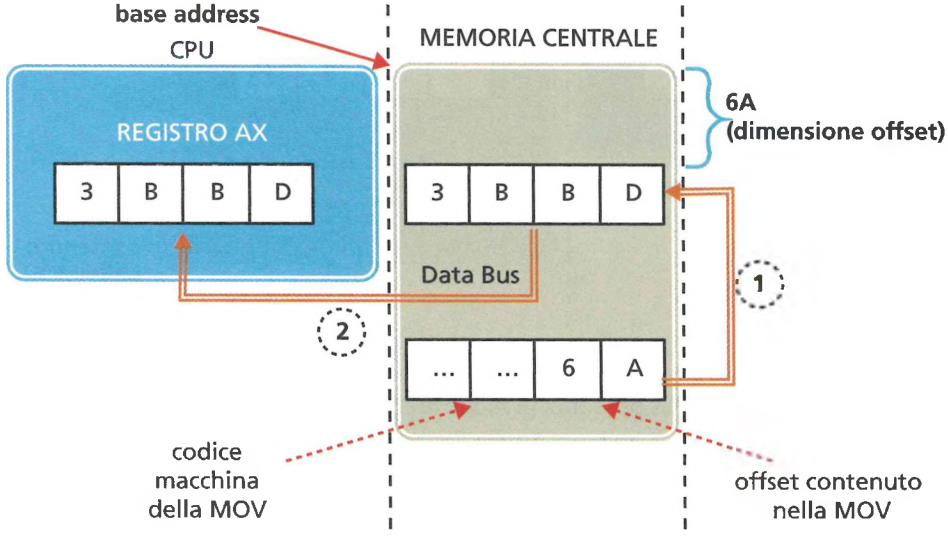


FIGURA 22 Indirizzamento diretto da memoria a registro con istruzione MOV AX,[6Ah]

Invertendo gli operandi e cioè scrivendo l'istruzione **MOV [6Ah],AX**, il contenuto di AX verrebbe copiato in memoria all'offset "6A".

7.5 Indirizzamento indiretto tramite registro

Quando nell'architettura della CPU sono previsti anche registri puntatore o indice (per esempio Source Index, **SI**, per l'operando source e Destination Index, **DI**, per l'operando destination), è possibile indirizzare tramite registro:

MOV AX,[SI]

L'esecuzione di questa istruzione provoca prima il calcolo dell'indirizzo in cui si trova il dato realmente mediante l'operazione:

base address + source index = effective address

dove però l'offset non è specificato nell'istruzione come nella modalità precedente, ma è in SI.
Il secondo passo è la copiatura del valore che si trova in memoria (all'offset contenuto in SI) nel registro AX (FIGURA 23).

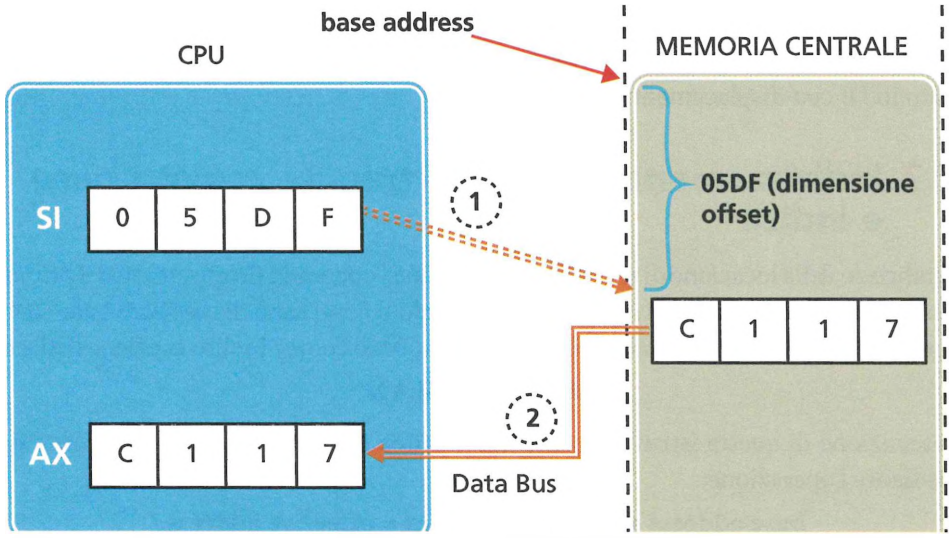


FIGURA 23 Indirizzamento indiretto tramite registro con istruzione MOV AX,[SI]

Invertendo gli operandi e usando DI al posto di SI (essendo il primo operando una destinazione), cioè scrivendo l'istruzione **MOV [DI],AX**, il contenuto di AX verrebbe copiato in memoria all'offset contenuto in DI.

7.6 Indirizzamento indiretto tramite registro e displacement

L'indirizzo della locazione di memoria che contiene l'operando si ottiene spostandosi del valore di displacement specificato nell'istruzione rispetto all'indirizzo ottenuto con l'offset:

MOV AX,03[SI]

L'esecuzione di questa istruzione prevede il calcolo dell'indirizzo effettivo mediante l'operazione:

base address + source index + displacement = effective address

dove l'offset è in SI (primo passo) e lo spiazzamento "03" nell'istruzione (secondo passo). Infine (terzo passo) il valore che si trova in memoria all'offset e al displacement calcolati è copiato nel registro AX (FIGURA 24).

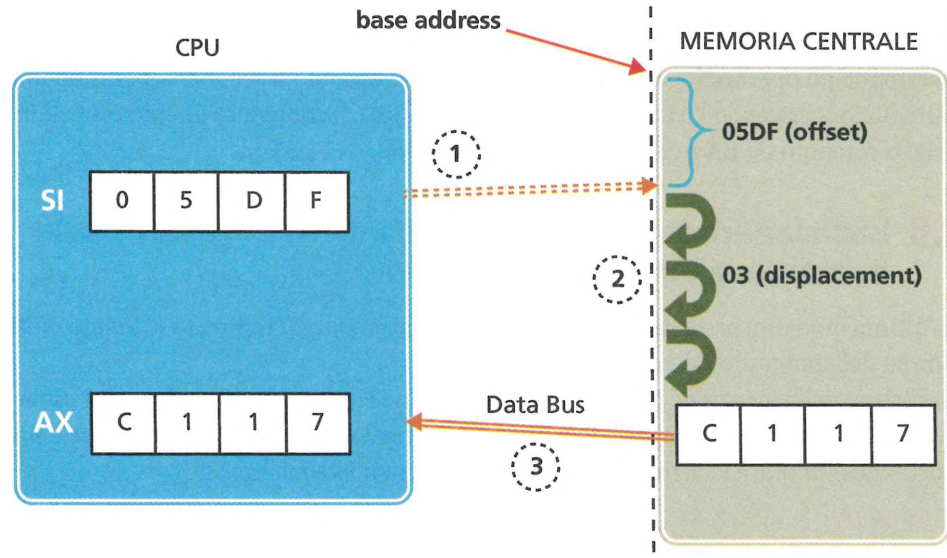


FIGURA 24 Indirizzamento indiretto tramite registro e displacement con istruzione MOV AX,03[SI]

Invertendo gli operandi e usando DI al posto di SI, cioè scrivendo l'istruzione **MOV 03[DI],AX**, il contenuto di AX verrebbe copiato in memoria all'offset contenuto in DI con displacement "03".

7.7 Indirizzamento indiretto tramite registri base e indice

#prendinota

Il numero di operandi dell'istruzione è sempre due (separati dalla virgola), però può aumentare il numero di registri coinvolti e la complessità dell'algoritmo: il ciclo macchina deve eseguire più micro-operazioni e dunque impiegherà più tempo di CPU per il calcolo dell'indirizzo.

L'indirizzo della locazione di memoria che contiene l'operando (in questo caso il primo, cioè quello di destinazione) si ottiene sommando il contenuto del registro base (BX) con il contenuto del registro indice (DI). Quest'ultimo come al solito contiene l'offset.

MOV [BX][DI],AX

L'esecuzione di questa istruzione determina prima il calcolo dell'indirizzo effettivo mediante l'operazione:

base address + destination index = effective address.

Quindi, secondo passo, il valore che si trova nel registro AX viene copiato all'indirizzo di memoria calcolato (FIGURA 25).

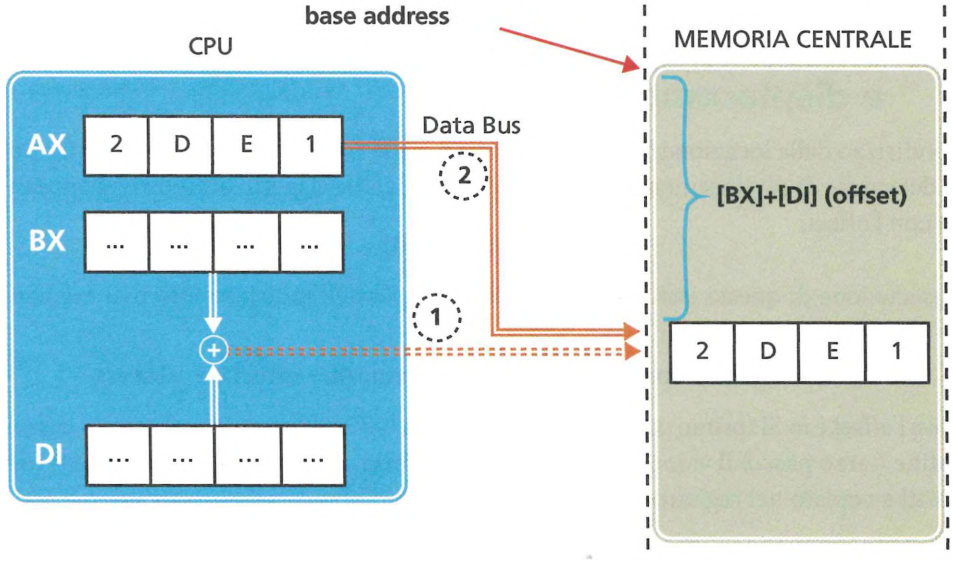


FIGURA 25 Indirizzamento indiretto tramite registri base e indice con istruzione MOV [BX][DI],AX

Invertendo gli operandi e usando SI al posto di DI (essendo il secondo operando una source), cioè scrivendo l'istruzione **MOV AX,[BX][SI]**, il contenuto di memoria all'indirizzo ottenuto da BX e SI verrebbe copiato in AX.

7.8 Indirizzamento indiretto tramite registri base e indice e con displacement

Un'ultima modalità prevede di aggiungere il displacement all'indirizzo ottenuto dalla somma del contenuto dei registri base e indice.
L'indirizzo della locazione di memoria che contiene l'operando da copiare, per esempio in AX, si ottiene sommando il contenuto del registro base e di quello indice e poi spostandosi di tante locazioni quante specificate nell'istruzione:

MOV AX,02[BX][SI]

L'esecuzione di questa istruzione determina prima il calcolo dell'indirizzo effettivo mediante l'operazione:

base address + source index + displacement = effective address

Il secondo passo prevede che all'indirizzo sia sommato lo spiazzamento "02" che si trova nell'istruzione. Infine, come terzo passo, il valore che si trova in memoria (all'offset e al displacement calcolati) è copiato nel registro AX (FIGURA 26).

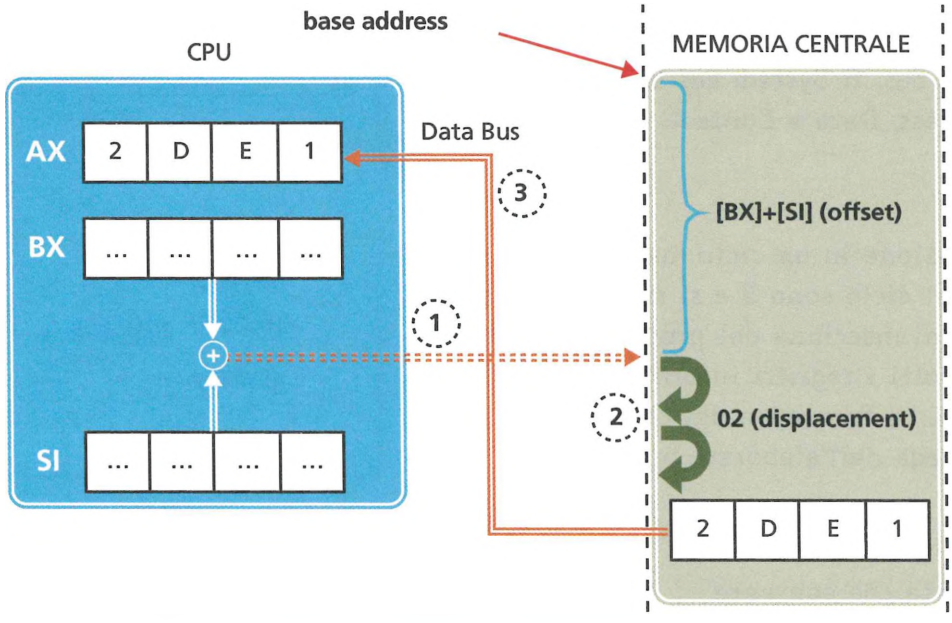


FIGURA 26 Indirizzamento indiretto tramite registri base e indice e con displacement con istruzione MOV AX,02[BX][SI]

Invertendo gli operandi e usando DI al posto di SI, cioè scrivendo l'istruzione **MOV 02[BX][DI],AX**, il contenuto di AX verrebbe copiato in memoria all'indirizzo ottenuto da BX e SI e dal displacement "02".

FISSA LE CONOSCENZE

- Perché non ha senso scrivere MOV 55CCh,AX?
- Che tipo di indirizzamento utilizza l'istruzione MOV AX,[BX][SI]?
- Che tipo di indirizzamento utilizza l'istruzione MOV 03[DI],AX?
- Che cos'è il base address?
- Che cos'è l'offset?
- Che cos'è il displacement?
- Che cos'è e come si calcola l'effective address?
- Descrivi l'indirizzamento diretto da memoria a registro.
- Descrivi l'indirizzamento indiretto tramite registri base e indice e con displacement.