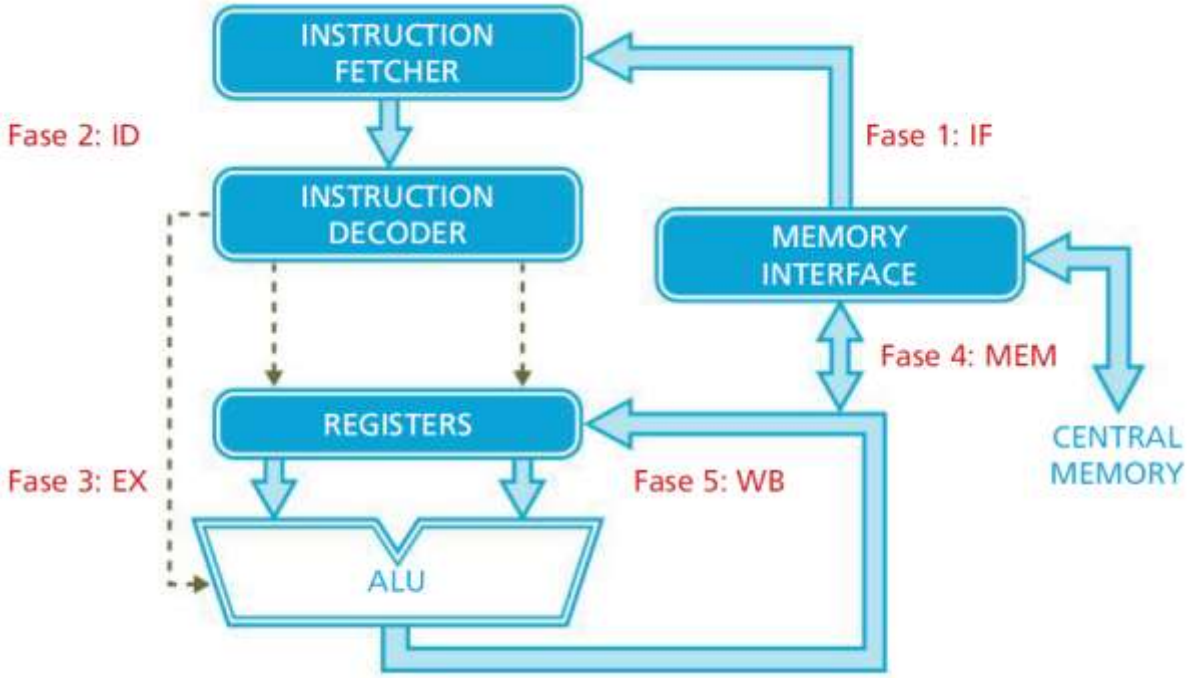
Unità 3

Il microprocessore

Il ciclo macchina (1)

La CPU esegue ogni istruzione in un **ciclo macchina** (*o fetch-execute cycle*).

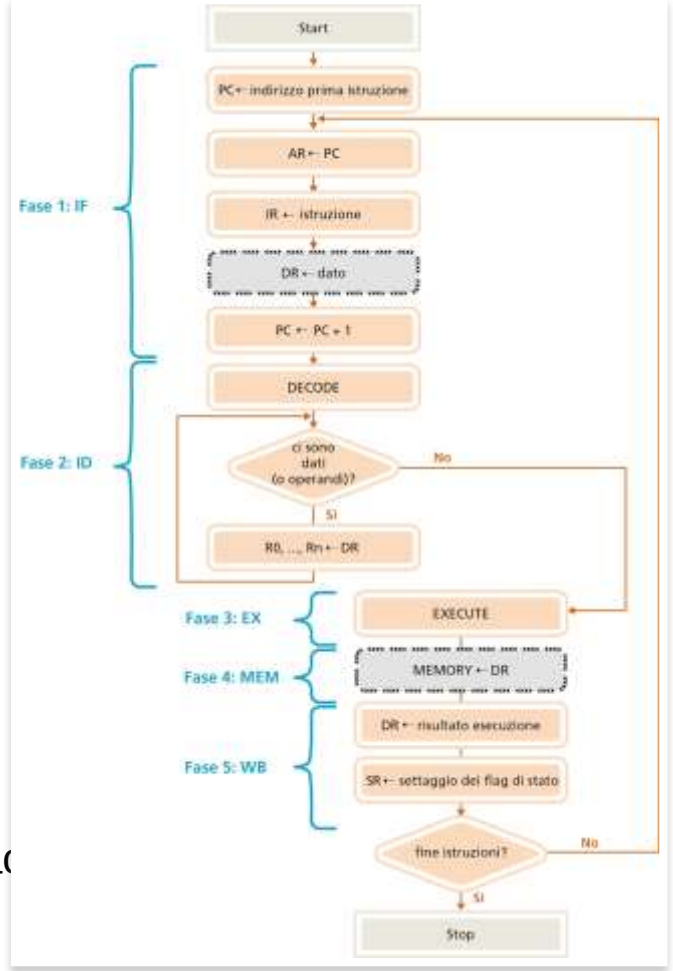


Schema a blocchi del ciclo macchina

Il ciclo macchina (2)

Le **fasi del ciclo sono 5** e si ripetono fino al termine delle istruzioni macchina del programma:

- IF (*Instruction Fetch*);
- ID (*Instruction Decode*);
- EX (*Execution*);
- MEM (*Memory*);
- WB (*Write Back*).

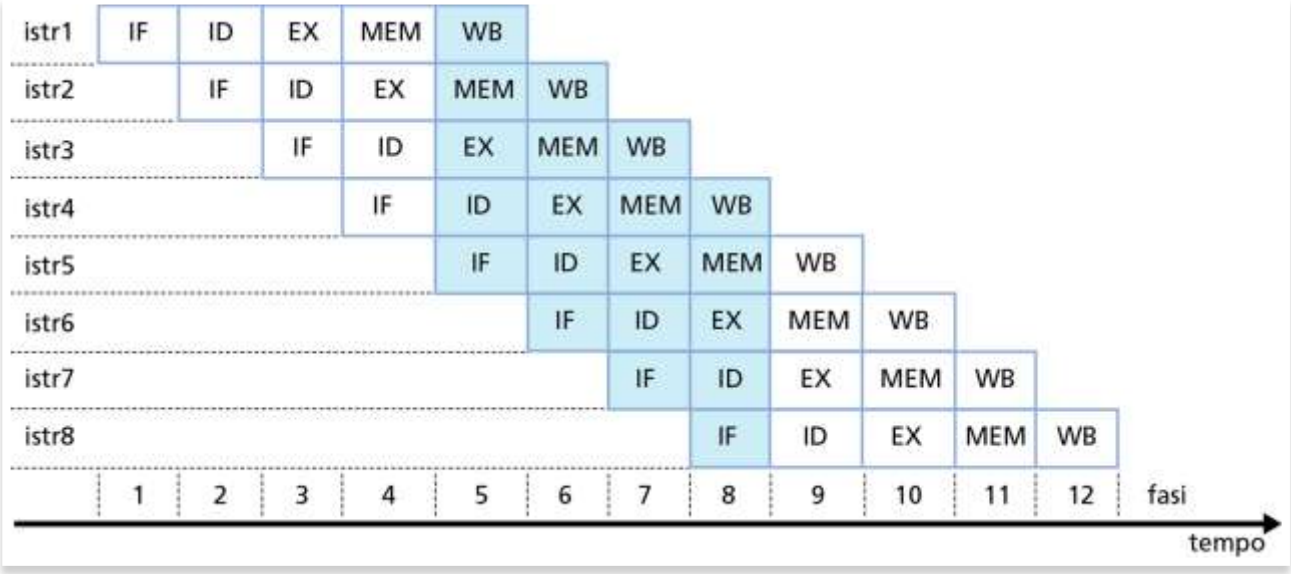


Flowchart del ciclo mac

La tecnica pipelining (1)

Il **pipelining** è una tecnica che consente di elaborare in parallelo più istruzioni, misurato in MIPS (Million Instruction per Second)

Con la tecnica pipelining più unità funzionali sono usate per eseguire un'istruzione macchina, formando una conduttura o pipeline.



La tecnica pipelining (2)

La tecnica pipelining funziona molto bene se non vi sono legami troppo stretti tra due istruzioni.

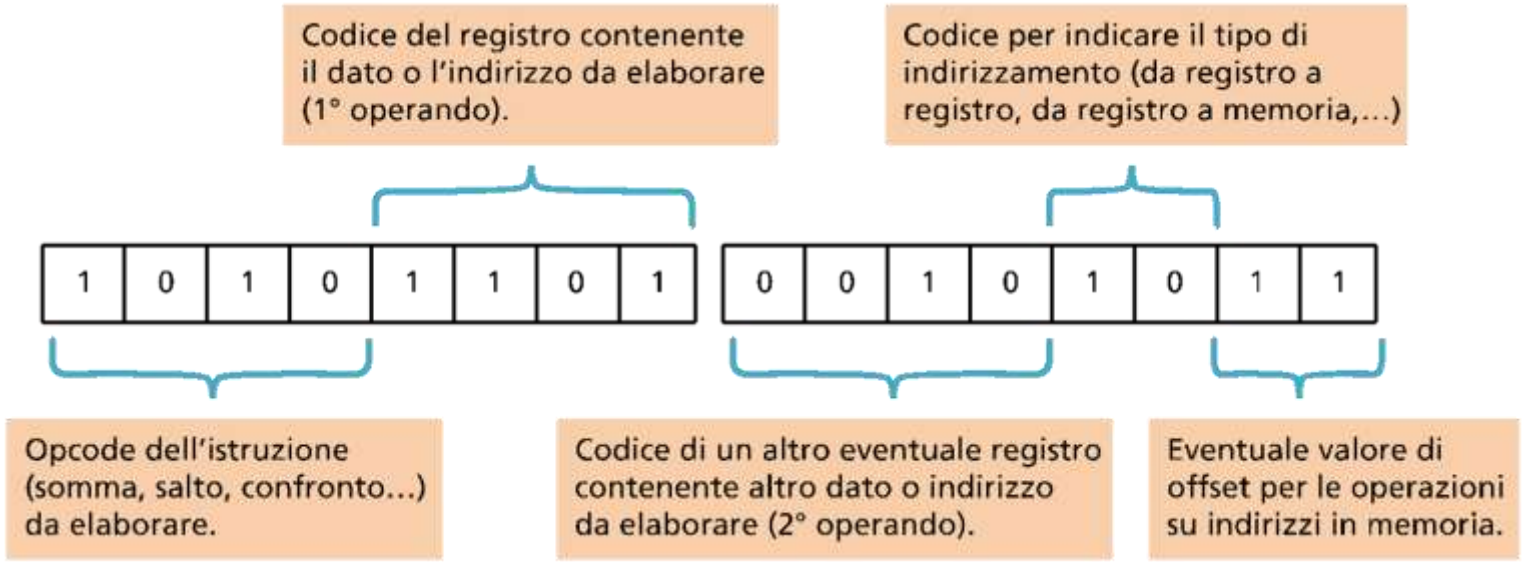
Un altro problema è dovuto ai **salti di esecuzione**: blocchi di istruzioni che non sono eseguite se non sono verificate determinate condizioni.

Per ovviare a questi problemi sono adottate diverse soluzioni:

- utilizzare i cosiddetti registri a doppia porta;
- utilizzare più pipeline autonome in parallelo (**tecnica superscalare**);
- introdurre dei circuiti che si occupano di analizzare i possibili salti (**unità di predizione delle diramazioni**);
- suddividere l'esecuzione di un'operazione in fasi elementari (20-30) che possono essere eseguite molto rapidamente aumentando la frequenza del clock.

I set di istruzioni macchina (1)

Il **linguaggio macchina** rappresenta l'insieme delle istruzioni macchina che la CPU è in grado di comprendere ed eseguire.



Esempio di formato di un'istruzione macchina

I set di istruzioni macchina (2)

- Architettura a due operandi

Opcode | Operando 1 | Operando 2

Pro: Abbastanza bit a disposizione per gli indirizzi

Contro: Occorre salvare il primo operando se lo si vuole usare in futuro

- Architettura a tre operandi

Opcode | Risultato | Operando 1 | Operando 2

Pro: Programmazione semplice, gli indirizzi sono espliciti (risultato)

Contro: Numero di bit per codificare gli indirizzi è limitato

I set di istruzioni macchina (3)

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

CISC (*Complex instruction Set Computer*, 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.

RISC (*Reduced Instruction Set Computer*, 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.

I set di istruzioni macchina (4)

RISC

Reduced Instruction Set Computing

Caratteristiche:

- Istruzioni semplici e uniformi
- Esecuzione in un ciclo di clock
- Molti registri general-purpose
- Load/Store architecture

Vantaggi:

- Pipeline più efficiente
- Minore consumo energetico
- Design più semplice

Esempi:

- ARM
- MIPS
- RISC-V

CISC

Complex Instruction Set Computing

Caratteristiche:

- Istruzioni complesse e specializzate
- Cicli di clock multipli per istruzione
- Pochi registri specializzati
- Accesso diretto alla memoria

Vantaggi:

- Codice più compatto
- Supporto legacy
- Istruzioni potenti

Esempi:

- x86 (Intel)
- x86-64 (AMD64)
- VAX

Evoluzione e confronto tra microprocessori (1)

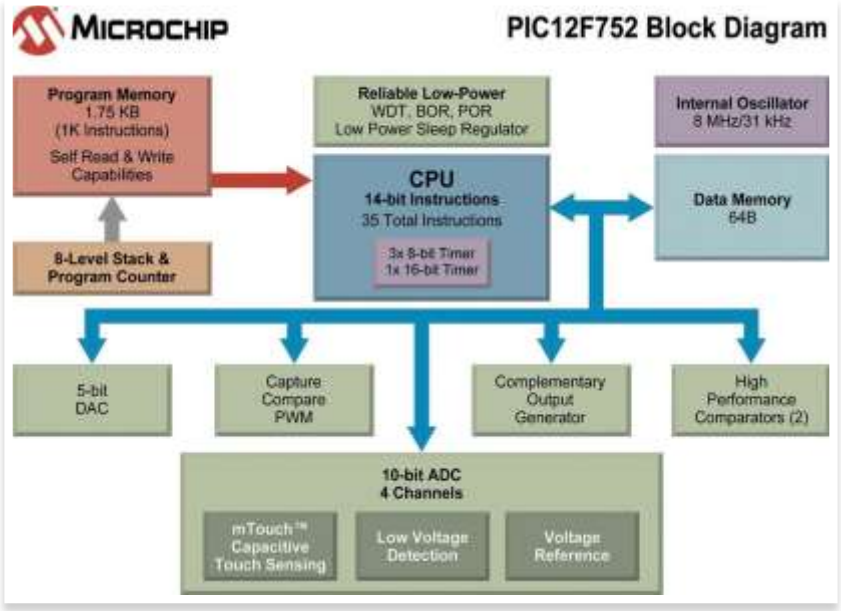
Il primo a elaborare la struttura di un **microprocessore** fu il fisico italiano **Federico Faggin**, nel 1968.

Faggin contribuì anche allo sviluppo di vari tipi di microprocessori che hanno caratterizzato la storia dell'informatica.

Dal 1968 a oggi le caratteristiche e le prestazioni dei microprocessori sono cambiate e oggi disponiamo di microprocessori in grado di garantire performance impensabili fino a due decenni fa.

Evoluzione e confronto tra microprocessori (2)

Negli ultimi anni si è diffuso l'uso di **microcontrollori** all'interno di dispositivi ed elettrodomestici di uso comune, consentendo così di realizzare dispositivi "intelligenti", in grado cioè di avere un funzionamento molto variabile in base alle esigenze dell'utente.



Il linguaggio assembly (1)

Il codice macchina può essere visto in una forma equivalente attraverso un **linguaggio mnemonico**.

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

Esse operano sui **registri** (in esadecimale: 0-9, A-F).

Un'istruzione ha questa sintassi:

- Label = nome simbolico istruzione
- Opcode = codice operando istruzione
- Zero/Uno/Due operandi = registri su cui operare

Esempio:

SUM 0X, 0D -> Somma dal registro 0x e metti in 0D

Il linguaggio assembly (2)

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 chdisassemblaggio e non richiede alcuna intelligenza;
- **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.

Assembly X86: info

Parliamo della famiglia di processori di Intel 8086, che possiedono vari tipi di registro, con diverse funzioni:

- Stato
- Lavoro
- Indice
- Segmento
- Puntatore

Essi vengono utilizzati comunemente con la stessa logica nei processori odierni.

Assembly X86: set di istruzioni

Trasferimento Dati

MOV dest, src - Copia dati da src a dest
PUSH src - Inserisce src nello stack
POP dest - Preleva dallo stack in dest
LEA dest, src - Carica indirizzo effettivo
XCHG op1, op2 - Scambia i valori tra op1 e op2

Aritmetiche

ADD dest, src - Addizione (dest = dest + src)
SUB dest, src - Sottrazione (dest = dest - src)
MUL src - Moltiplicazione unsigned (AX = AL × src)
DIV src - Divisione unsigned (AL = AX ÷ src)
INC dest - Incremento (dest = dest + 1)
DEC dest - Decremento (dest = dest - 1)
NEG dest - Negazione (dest = -dest)

Logiche

AND dest, src - AND bit a bit
OR dest, src - OR bit a bit
XOR dest, src - XOR bit a bit
NOT dest - Inversione bit a bit
TEST op1, op2 - AND bit a bit senza salvare il risultato

Controllo del Flusso

JMP label - Salto incondizionato
JE/JZ label - Salta se uguale/zero
JNE/JNZ label - Salta se non uguale/non zero
CALL label - Chiamata a procedura
RET - Ritorno da procedura
LOOP label - Decrementa CX e salta se non zero

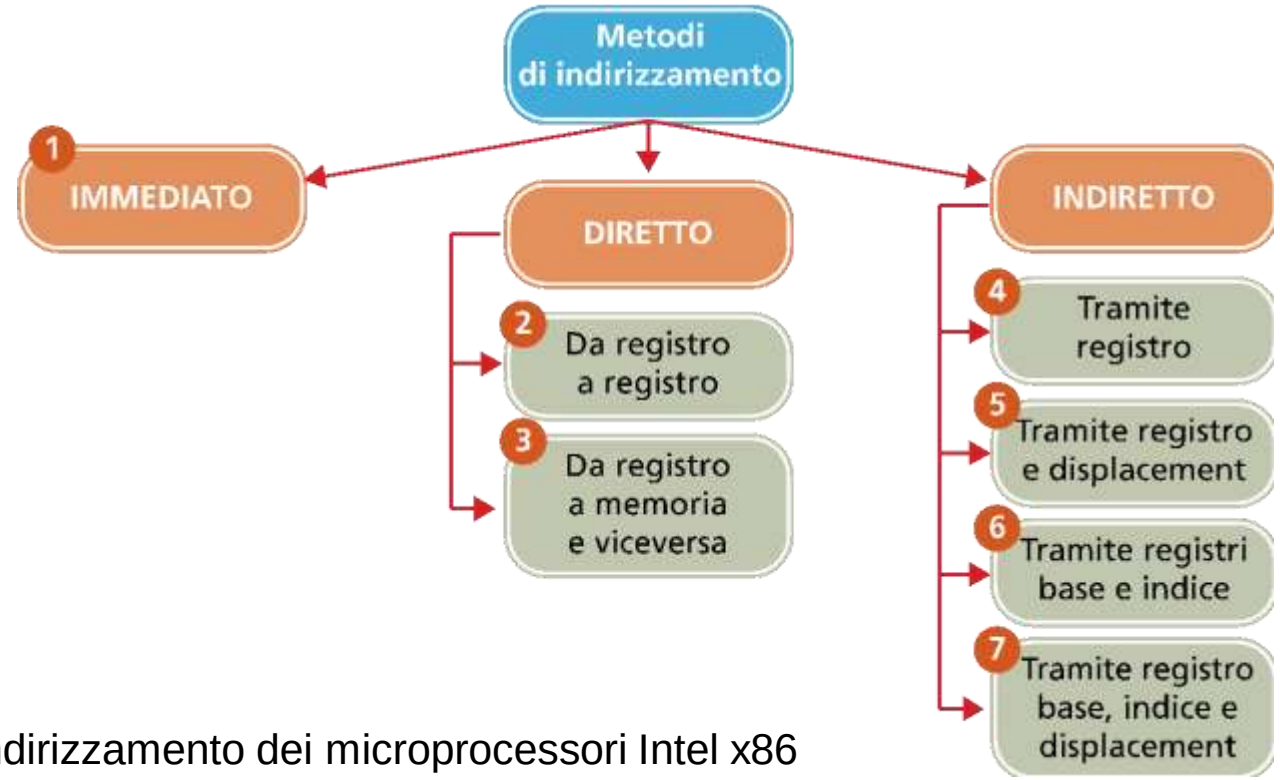
Confronto e Rotazione

CMP op1, op2 - Confronta op1 con op2
SHL/SAL dest, count - Shift logico/aritmetico a sinistra
SHR dest, count - Shift logico a destra
ROL dest, count - Rotazione a sinistra
ROR dest, count - Rotazione a destra

I metodi di indirizzamento

I **metodi indirizzamento** si suddividono in tre tipi, in base al modo in cui l'istruzione tratta il dato da elaborare:

- immediato;
- diretto;
- indiretto.



I principali metodi di indirizzamento dei microprocessori Intel x86

I metodi di indirizzamento (2)

Esso sfrutta tre parti:

- Base address (o cella iniziale)
- Offset (distanza da sommare per puntare al dato)
- Displacement (spostamento per puntare l'istruzione o il dato)

Base address + Offset + Displacemente = Effective address

I metodi di indirizzamento (3)

1. Indirizzamento Immediato

- Il dato è contenuto direttamente nell'istruzione
- Esempio: MOV AX, 1234h ; Carica il valore 1234h nel registro AX
- Vantaggio: accesso immediato, Svantaggio: valore non modificabile

2. Indirizzamento Diretto

2.1 Registro a Registro:

- Trasferimento diretto tra registri
- Esempio: MOV AX, BX ; Copia il contenuto di BX in AX

2.2 Registro a Memoria (e viceversa):

- Accesso diretto alla memoria tramite indirizzo
- Esempio: MOV [1000h], AX ; Memorizza AX all'indirizzo 1000h
- Esempio: MOV BX, [2000h] ; Carica in BX il contenuto dell'indirizzo 2000h

3. Indirizzamento Indiretto

3.1 Tramite Registro:

- Usa un registro come puntatore alla memoria
- Esempio: MOV AX, [BX] ; Carica in AX il contenuto dell'indirizzo contenuto in BX

3.2 Base + Displacement:

- Indirizzo = Registro Base + Spiazzamento
- Esempio: MOV AX, [BX + 4] ; Carica in AX il contenuto di (BX + 4)

3.3 Indice + Base + Displacement:

- Indirizzo = Base + Indice * Scala + Displacement
- Esempio: MOV AX, [BX + SI * 2 + 100h] ; Accesso array con offset
- Utile per accedere ad array e strutture dati