



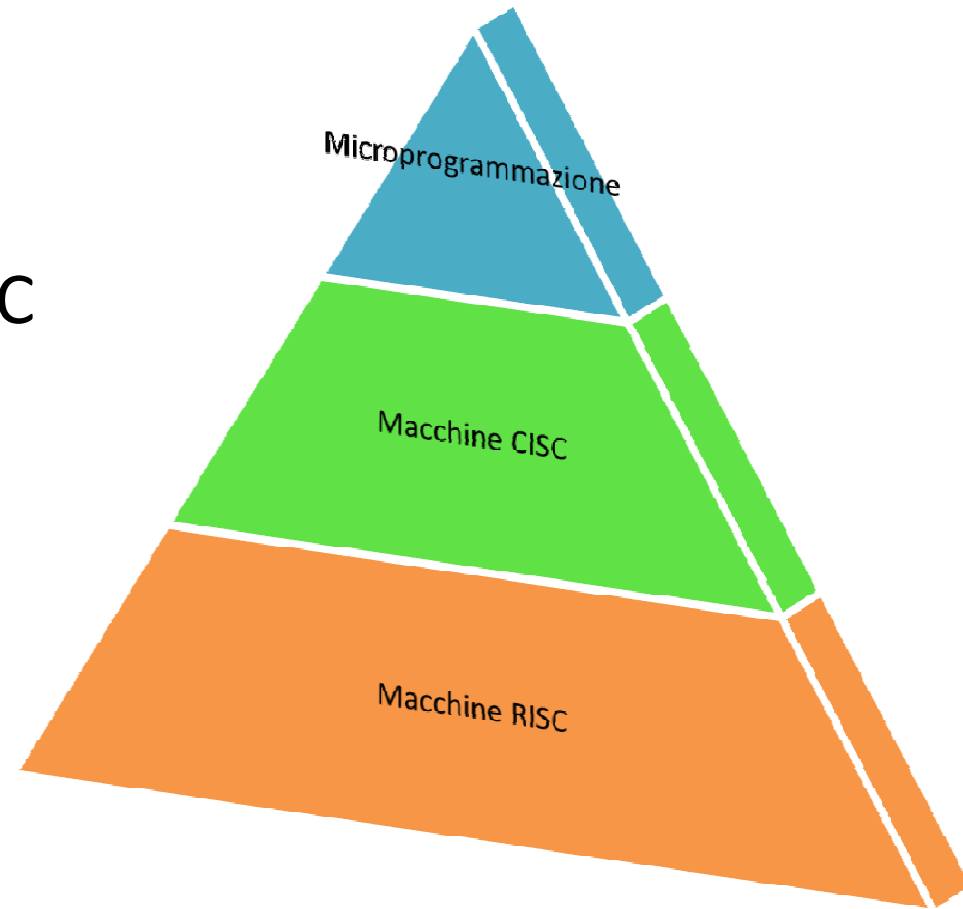
Architettura degli elaboratori

Microprogrammazione

Dott. Franco Liberati

ARGOMENTI DELLA LEZIONE

- ❑ Macchine RISC e CISC
- ❑ Microprogrammazione
- ❑ Processori ibridi RISC-CISC





Architettura degli elaboratori

Macchine CISC e RISC



MACCHINE CISC

Generalità

- ❑ Lo sviluppo tecnologico e la scarsa diffusione dei compilatori portò alla fine degli anni '70 alla realizzazione di architetture in grado di eseguire istruzioni complesse per minimizzare i costi di produzione. Nacquero, cioè, le **macchine CICS** (*Complex Instruction Set Computing*)
- ❑ Le **macchine CICS** hanno un set, di istruzioni numeroso (es.: Intel x86, dalle 200 alle 300 istruzioni) a cui appartengono svariati modi di indirizzamento
- ❑ Vantaggio: codice compatto (miglior debug)

Istruzioni RISC	Istruzione CISC
lw \$t0,0x100 lw \$t1,0x200 add \$t2,\$t0,\$t1 sw \$t2,0x300	ADD 0x100,0x200,0x300
lb \$t0,0x100 sb \$t0,0x500 lb \$t0,0x101 sb \$t0,0x501 lb \$t0,0x199 sb \$t0,0x599	MVC 15 (100, R5), 50 (R8) <i>Sposta 100byte consecutivi a partire dall'indirizzo in memoria dato da 50 + [R8] all'indirizzo 15 + [R5]</i>



MACCHINE CISC

Le istruzioni complesse

- ❑ Le **istruzioni complesse** richiedono **più cicli di clock** per essere eseguite, molti accessi in memoria ed un tempo di decodifica variabile
- ❑ Per eseguire un'istruzione complessa è spesso necessario effettuare degli **accessi ed operazioni** in maniera sequenziale e **nel giusto ordine**
- ❑ Il processore, può elaborare le istruzioni in due modi, per **interpretazione**, scomponendo l'istruzione in tante istruzioni più semplici, o in **esecuzione diretta**, in cui non avviene una decomposizione dell'istruzione in istruzioni più semplici
- ❑ Vista la complessità ed il grande numero delle istruzioni, eseguirle direttamente richiederebbe per l'implementazione un **numero improponibile di componenti fisiche** (passaggio da Unità di controllo da rete combinatoria a rete sequenziale)
- ❑ La soluzione a questo problema, è quella di avere all'interno del processore una **ROM** (memoria in sola lettura) **che contiene la traduzione delle istruzioni complesse in sequenze di istruzioni semplici**. Praticamente, non è il compilatore a scomporre le istruzioni complesse in tante istruzioni semplici, ma è il processore che si preoccupa di scomporle in istruzioni direttamente eseguibili in hardware
- ❑ L'operazione di decodifica è un punto fondamentale dei processori CISC, dato che, più sono complesse le istruzioni, maggiore sarà il tempo per decodificarle



MICROPROGRAMMAZIONE

Generalità

- ❑ Una alternativa alla comune realizzazione di un transcodificatore combinatorio fu proposta negli anni Cinquanta da **Maurice Vincent Wilkes** che dotò la CU di una memoria di controllo in cui archiviare le sequenze di comandi che venivano trasformati da circuiterie dedicate in segnali di controllo: **l'unità di controllo microprogrammata (UC-M)**
 - ❑ La presenza di un transcodificatore richiede un alto numero di componenti ed interconnessioni che risultano essere costosi e scarsamente flessibili ad aggiunte e modifiche
 - ❑ La fase di decodifica delle istruzioni complesse in un UC-M è demandata ad un micromacchina



MICROPROGRAMMAZIONE

Microprogramma e Micro-macchina

- ❑ L'idea è quella di far corrispondere ad una singola istruzione macchina l'**esecuzione di un microprogramma 'scritto' in una memoria non modificabile** (*read only memory*, ROM)
- ❑ Il processore in questo caso ha una struttura logica che ha la funzione di interpretare programmi scritti in linguaggio macchina, in programmi scritti in **linguaggio micro-programmato**
- ❑ Ad ogni istruzione macchina corrisponde una serie di attività che interessano la CPU

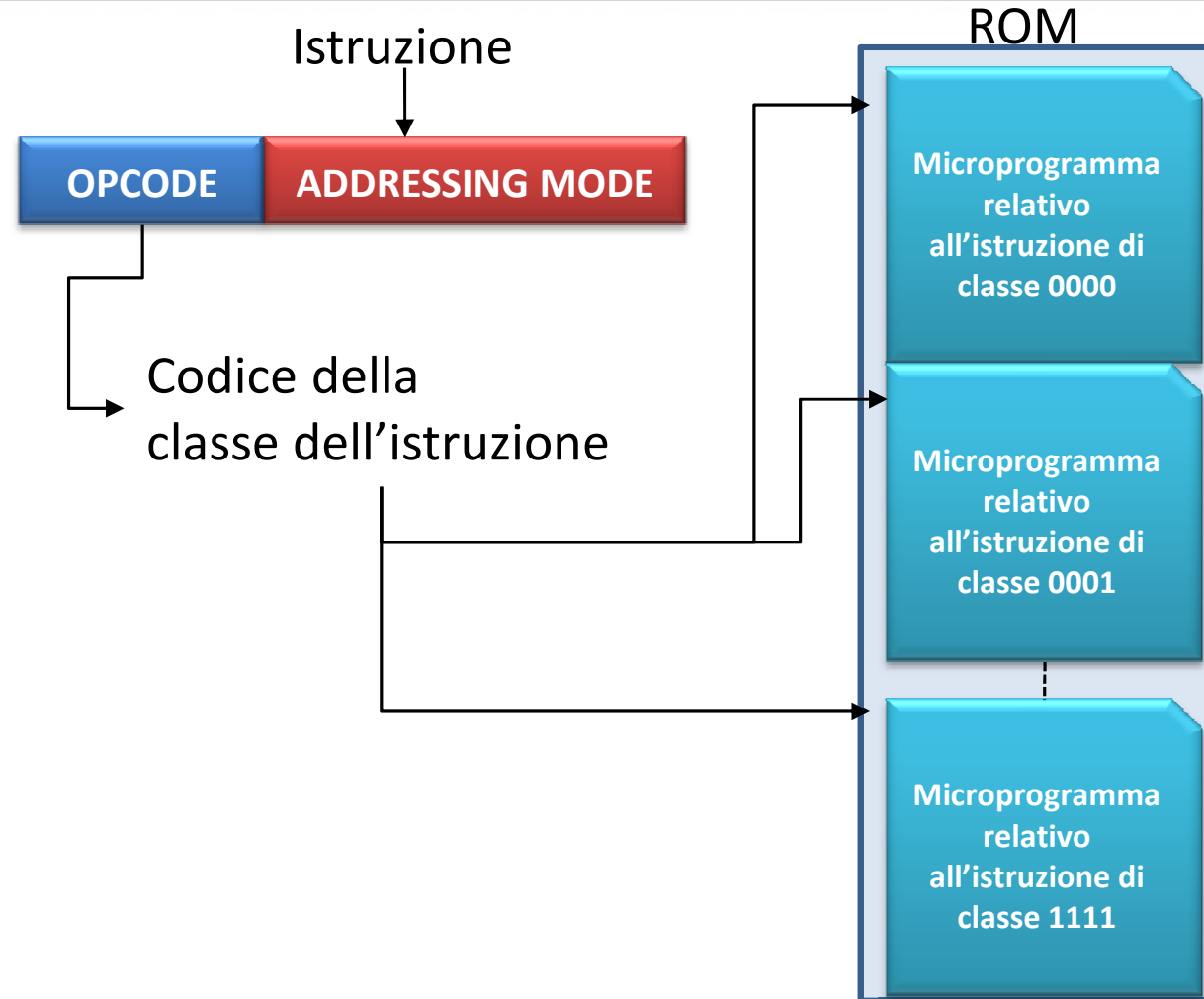
Osservazione. Il contenuto della ROM è fisso e non può essere modificato; esso è determinato e reso permanente in fase di produzione.

In fase progettuale invece, per eseguire test, sono usate **PROM** (ROM programmabili via hardware) o **WCM** (memorie di controllo riscrivibili) (microprogrammazione dinamica) dal programmatore del microprogramma

MICROPROGRAMMAZIONE

Microprogramma

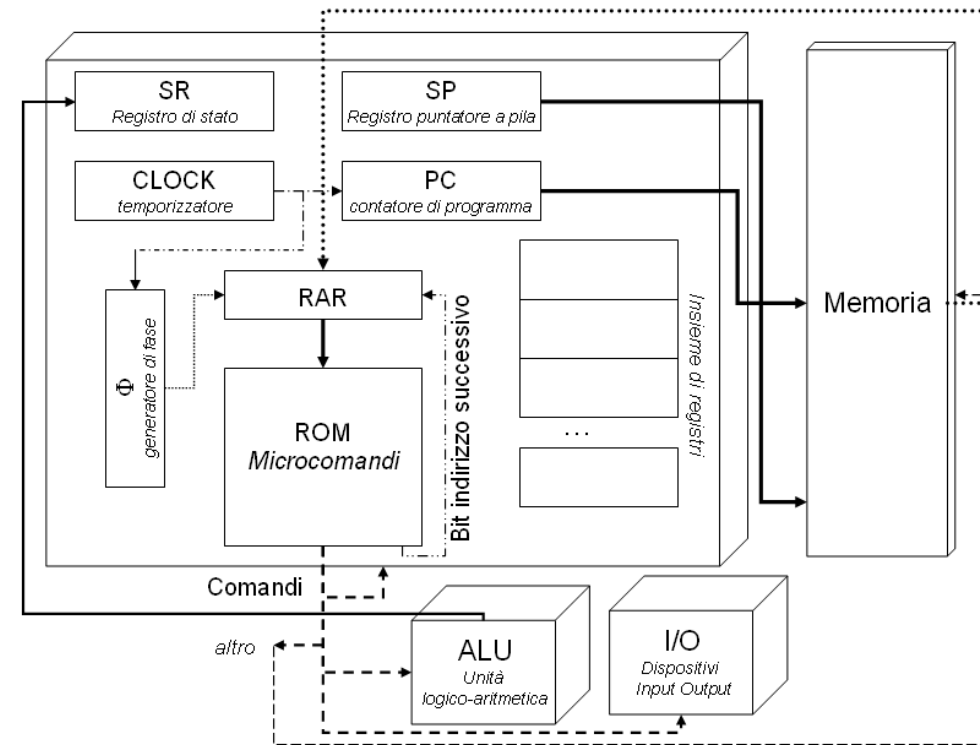
- ❑ Per ogni istruzione che la CU deve interpretare e da cui genera un serie di comandi, la stessa CU-M deve fare in modo di eseguire un **microprogramma** che è costituito da un insieme di **microistruzioni** il cui numero dipende dal numero di **microoperazioni** che debbono essere effettuate e dal loro sequenziamento per generare i comandi (che comunemente dovrebbero essere generati dal transcodificatore)
- ❑ Per accedere correttamente ed in maniera univoca ai diversi microprogrammi si può utilizzare l'OPCODE di ogni istruzione (in questo caso l'opcode ha il ruolo di indirizzo iniziale al microprogramma da eseguire)



MICROMACCHINA

Componenti basi

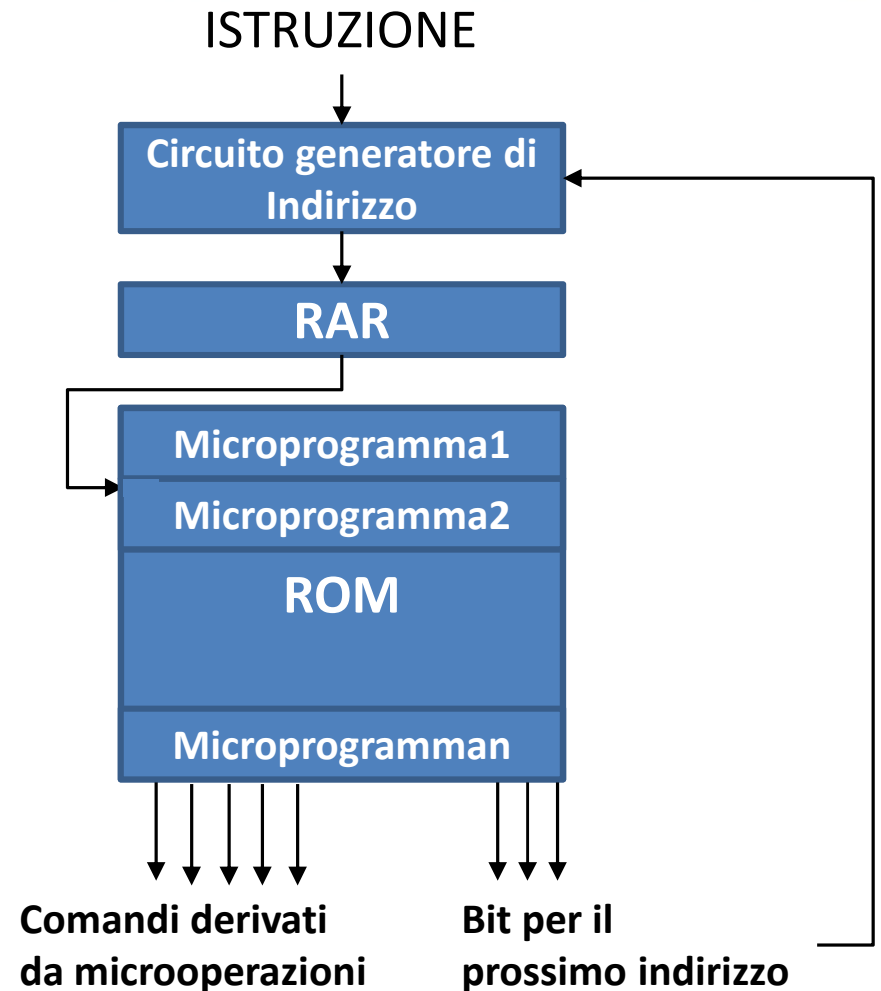
- ❑ Tra le componenti essenziali della micromacchina ci sono:
 - ❑ Il **generatore di indirizzo** cioè il circuito predisposto al calcolo dell'indirizzo della successiva microistruzione
 - ❑ Il **registro RAR** (ROM address register) un registro in cui all'interno è presente l'indirizzo alla microistruzione da elaborare (è una sorta di **program counter per la macchina microprogrammata**)
 - ❑ Il contenuto del RAR ha il valore all'indirizzo della ROM (input) che risponde con una parola di controllo (output)
- ❑ La parola letta dalla ROM rappresenta una microistruzione costituita da tante microoperazioni che possono coinvolgere ogni componente della macchina (ALU, memoria, registri, bus,...)
- ❑ Una volta eseguita una microistruzione bisogna individuare la successiva o predisponendo che alcuni bit della parola di controllo siano dedicati ad assolvere questo compito oppure che alcuni bit specificano esplicitamente l'indirizzo della microistruzione successiva
- ❑ I microprogrammi possono usare anche delle **microsubroutine** (per svolgere compiti ripetitivi) e quindi necessita di un registro di ritorno (Es: SBR)



MICROPROGRAMMAZIONE

Microprogramma: micromacchina e esecuzione

- ❑ Un indirizzo, pertanto, può essere raggiunto con:
 1. Mapping dell'OPCODE dell'istruzione ai bit che rappresentano il primo indirizzo della microistruzione della routine opportuna (cioè l'OPCODE dell'istruzione è copiato nel RAR indicando la prima microistruzione)
 2. Incremento del RAR
 3. Salto incondizionato all'indirizzo specificato da un campo presente nella microistruzione
 4. Salto condizionato dipendente dai **bit di stato nei registri** della micro-macchina
 5. Un indirizzo contenuto in un **registro di ritorno (SBR) nel caso di salto a subroutine**



MICROPROGRAMMAZIONE

Microprogramma: esempio

❑ Formato istruzione

❑ ADDM 40, 44, 42

Prende l'operando sito all'indirizzo 42
e lo mette nel primo accumulatore

Prende l'operando sito all'indirizzo 44
e lo somma con il valore
nell'accumulatore

Il risultato riportato
nell'accumulatore è spostato alla
locazione 40

OPCODE

ADDRESSING MODE

000000

101000

101100

101010

PRIMA FASE

$MAR \leftarrow PC$ e $PC \leftarrow PC + 1$

$MDR \leftarrow M(\text{lettura istruzione})$

SECONDA FASE

$RAR \leftarrow MDR(\text{OPCODE})$

$MAR \leftarrow MDR(\text{ADDRESSING MODE})$



MICROPROGRAMMAZIONE

Microprogramma: esempio

☐ Microistruzione nella ROM

Bit	Microps	Descrizione
1	$MDR \leftarrow M$	Lettura dalla memoria (nel MAR indirizzo sorgente)
2	$M \leftarrow MDR$	Scrittura in memoria (nel MAR indirizzo destinazione)
3	$MAR \leftarrow PC$	Indirizzo dell'istruzione (fetch)
4	$PC \leftarrow PC + 1$	Incremento program counter
5	$TMP \leftarrow MAR$	Trasferimento del MAR in un registro temporaneo
6	$MAR \leftarrow TMP$	Trasferimento da un registro temporaneo al MAR
7	$MDR \leftarrow AC$	Trasferimento del valore nell'accumulatore in memoria
8	$AC \leftarrow MDR$	Trasferimento di un operando nell'accumulatore
9	$AC \leftarrow AC + MDR$	Somma dell'accumulatore con l'operando in MDR
10	$MAR \leftarrow MAR \gg 6$	Shift a destra di 6 bit del MAR
11	$MAR \leftarrow MAR \& 63$	Maschera per estrarre 6 bit meno significativi

22	11	10	9	8	7	6	0
MICROPS				CD	BR	Indirizzo di salto ADF	

Condizioni

Bit	Microps	Descrizione
00	J	Salto incondizionato
01	I=1	Salto
10	AC=1	Salta se operando nell'accumulatore è negativo
11	AC=0	Salta se operando nell'accumulatore è zero

Salto

Bit	Microps	Descrizione
00	Se CD=1 $RAR \leftarrow ADF$	Salto incondizionato
01	$RAR \leftarrow ADF$ e $SBR \leftarrow RAR + 1$	Chiamata a subroutine
10	$RAR \leftarrow SBR$	Ritorno da subroutine
11	$RAR \leftarrow MDR(OP)$	Caricamento prossima Microoperazione

MICROPROGRAMMAZIONE

Microprogramma	Indirizzo	Microistruzione	Micro operazione
ADDM	0000000 (0)	0000000000 01 01 1000011	Salto a subroutine $RAR \leftarrow ADF$ e $SBR \leftarrow RAR+1$
	0000001 (1)	0001000000 01 00 0000011	$AC \leftarrow MDR$ e salto a istr. succ.
	0000010 (2)	0000000000 01 01 1000011	Salto a subroutine $RAR \leftarrow ADF$ e $SBR \leftarrow RAR+1$
	0000011 (3)	0010000000 01 00 0000100	$AC \leftarrow AC + MDR$ e salto a istr. succ.
	0000100 (4)	0000100000 01 00 0000101	$MDR \leftarrow AC$ e salto a istr. succ.
	0000101 (5)	0000000010 01 00 0000110	$M \leftarrow MDR$ e salto a istr. succ.
	0000110 (6)	0000000000 01 00 1100000	Salto incondizionato a FETCH
...
SBR_ADD_DIR <i>subroutine per prelievo operando e ritorno all'indirizzo</i>	1000011	0000001000 00 00 1000100	$TMP \leftarrow MAR$ e salto a istr. succ.
	1000100	1000000000 00 00 1000101	$MAR \leftarrow MAR \& 63$ e salto a istr. succ.
	1000101	0000000001 00 00 1000110	$MDR \leftarrow M$ e salto a istr. succ.
	1000110	0000010000 00 00 1000111	$MAR \leftarrow TMP$ e salto a istr. succ.
	1000111	0100000000 00 10 0000000	$MAR \leftarrow MAR \gg 6$ e salto a $RAR \leftarrow SBR$
...
FETCH <i>carica istruzione successiva</i>	1100000	00000000100 00 00 1100001	$MAR \leftarrow PC$ e salto a istr. succ.
	1100001	00000001001 00 00 1100010	$MDR \leftarrow M$ e $PC \leftarrow PC+1$ e salto a <i>routine specificata in AD</i>

MICROPROGRAMMAZIONE

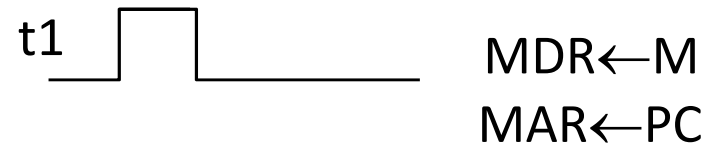
Microprogramma

- ❑ L'esecuzione del microprogramma relativo ad una istruzione è detto **ciclo istruzione**
- ❑ Ogni ciclo istruzione è costituito da una o più fasi elementari (cicli macchina) che comporteranno l'attivazione di microcomandi o microoperazioni relativi ad unità interne o esterne alla CPU
- ❑ Ogni istruzione è caratterizzata dal relativo numero dei cicli macchina

Microlistruzione

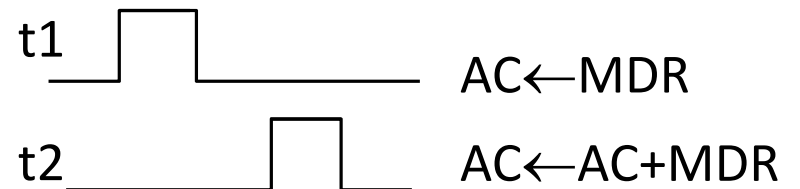
Micro-operazione

00000001001 00 00 000000



00010000000 00 00 000000

00100000000 00 00 000000





MICROPROGRAMMAZIONE

Microcodice orizzontale, verticale o diagonale

- ❑ È proprio il numero delle microoperazioni possibili ed il numero massimo di microoperazioni che debbono essere eseguite in parallelo che determina la possibilità di optare per una organizzazione **orizzontale**, **verticale** o **diagonale** del microcodice
- ❑ La struttura del microcodice, infatti, è solitamente una soluzione di compromesso tra due esigenze diverse: buona flessibilità e potenza operativa contro la contenuta occupazione di memoria dei microprogrammi. Se si privilegia la compattezza del microcodice allora le istruzioni sono fortemente codificate e limitate nella capacità di diramazione (**struttura verticale**), se si massimizza la potenza del microcodice consentendo la massima parallelizzazione delle microoperazioni e massima flessibilità nel sequenziamento degli indirizzi allora si parla di **struttura orizzontale**. Soluzioni intermedie sono dette a **struttura diagonale**
- ❑ Quindi, per poter interpretare un qualsiasi programma la CU-M deve essere strutturata in modo tale da fare corrispondere ad ogni istruzione macchina una o più microprogrammi distinti (procedure sequenziali). Per ottenere questo si può utilizzare una ROM considerando i vari microprogrammi come tante sottomacchine in una unica macchina sequenziale



MICROPROGRAMMAZIONE

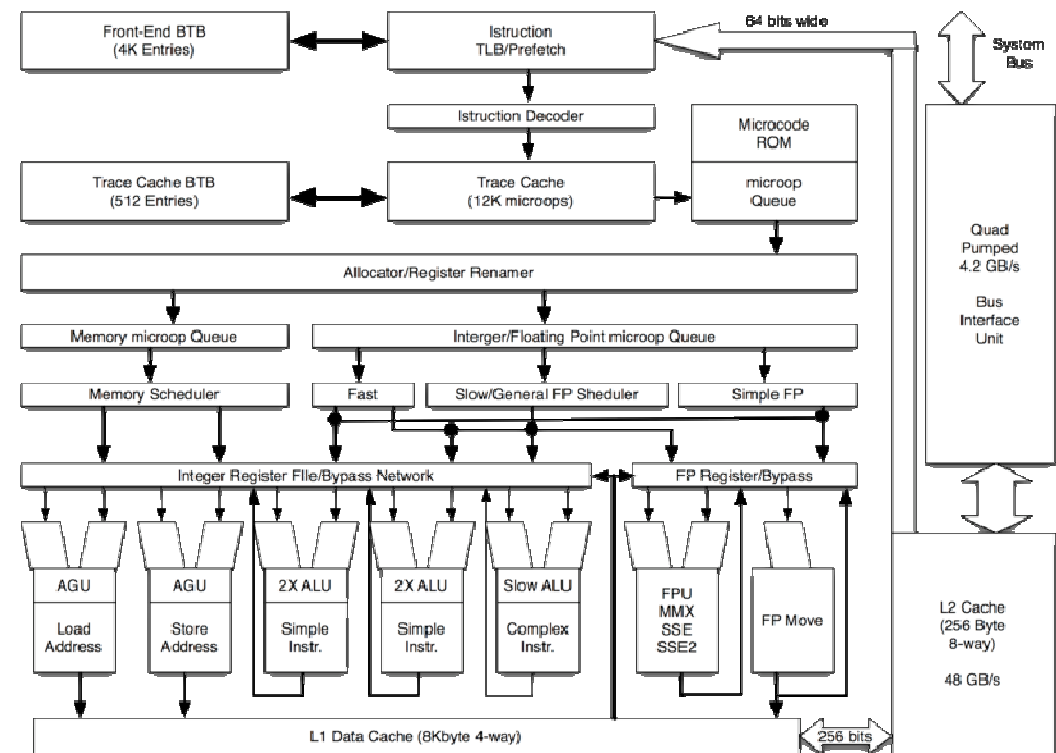
Ottimizzazione del microcodice

- ❑ L'**organizzazione di una macchina microprogrammata** dipende dalle prestazioni che vogliono essere ottenute e dai costi. Se non si vuole minimizzare il costo, allora il dimensionamento della ROM è fatto tenendo conto di tutte le variabili di ingresso (opcode, condition code,...) e di tutti i segnali di controllo (uscita). Ovviamente questa strategia porta a ROM di dimensioni molto elevate.
- ❑ Per questo si cerca di minimizzare il numero degli ingressi (quindi una riduzione del numero delle microistruzioni) e delle uscite (operando sulla dimensione di ogni singola parola della ROM). Vediamo alcune strategie per un risparmio:
 - ❖ **Allocare fisicamente in modo adiacente microistruzioni** in grado di interpretare una istruzione macchina (si usa solo uno spiazamento senza codificare tutto l'indirizzo per il salto alla microistruzione seguente).
 - ❖ **Ridurre le variabili di ingresso** che possono essere esaminate in una altra fase (esempio: se si è nella fase di fetch è inutile considerare i dati provenienti dallo SR).
 - ❖ **Ridurre il numero dei segnali di controllo raggruppandoli logicamente quei segnali di controllo che sono mutuamente esclusivi.** Esempio se la ALU ha bisogno di 10 segnali di controllo per le operazioni che può effettuare dato che queste operazioni possono essere attivate una sola alla volta si potranno pensare di utilizzare 4 linee demandano il compito di identificare l'operazione richiesta alla ALU stessa.
 - ❖ Strutturare il **microprogramma con codice rientrante** se differenti istruzioni macchina hanno parti di codice identico. In questo caso però è necessario prevedere il salvataggio degli indirizzi in una memoria gestita a pila a livello di microcodice

MICROPROGRAMMAZIONE

Processori ibridi

- ❑ L'architettura interna dei processori odierni segue l'approccio CISC solo per offrire la **retrocompatibilità** con i vecchi programmi
- ❑ I processori retrocompatibili decodificano il codice con istruzioni complesse, ma una volta decodificato, esso viene mandato in esecuzione su una serie di stadi di elaborazione di tipo RISC e cioè:
 - ❖ Le istruzioni delle macchine CISC complesse (tante istruzioni di natura diversa), e disomogenee (le istruzioni hanno una lunghezza variabile), sono decodificate attraverso una ROM che contiene la corrispondenza tra le istruzioni complesse e quelle più semplici, e che si adopera alla decodifica delle stesse
 - ❖ Le istruzioni delle macchine CISC sono, quando possibile, sostituite con istruzioni in formato omogeneo e immesse in una memoria tampone in attesa di essere processate (utilizzate efficientemente con la canalizzazione)



Osservazione. I processori Intel Pentium IV hanno un processore RISC che esegue istruzioni semplici e che si affianca alla macchina CISC con la quale vengono interpretate ed elaborate le istruzioni meno comuni. Il vantaggio di questa strategia è quella di una ottimizzazione dei tempi e la possibilità di riutilizzare i programmi esistenti.



MACCHINE CISC e RISC

Generalità

- ❑ Per tutti gli anni '50 e '60 l'uso di istruzioni macchina molto complesse era ampiamente giustificato dalla tecnologia disponibile in quegli anni:
 1. Il **tempo di accesso alla RAM era superiore al tempo di accesso alla ROM che conteneva i microprogrammi e che era posizionata fisicamente più vicino alla CPU** (in quegli anni le CPU non erano circuiti integrati miniaturizzati, ma occupavano lo spazio di più circuiti stampati montati all'interno di armadi della dimensione di qualche metro cubo)
 2. **Non esistevano ancora CPU dotate di cache**, che furono introdotte solo a partire dal 1968 con l'IBM 360/85. Aveva quindi senso, ogni volta che si doveva accedere alla RAM per leggere la successiva istruzione da eseguire cercare di fare in modo che questa istruzione esprimesse una gran quantità di lavoro
 3. Le **tecniche di progettazione dei compilatori erano ancora in fase di sviluppo**, e avere a disposizione istruzioni macchina molto espressive permetteva di semplificare il lavoro del compilatore. In altre parole, il set di istruzione complesso permetteva di ridurre il gap semantico tra il linguaggio ad alto livello e il linguaggio macchina
 4. La **RAM era costosa** e scarseggiava, e i processori di quegli anni non erano in ogni caso in grado di gestire grandi spazi di indirizzamento. Dunque l'uso di istruzioni macchina molto espressive permetteva di generare eseguibili più corti
 5. Infine, l'uso di microprogrammi per descrivere la funzione di controllo rendeva semplice **arricchire il set di istruzioni di una CPU** con nuove istruzioni (bastava sostituire la ROM)



MACCHINE CISC e RISC

Generalità

- ❑ Nel 1980 un gruppo di Berkeley con a capo David Patterson e Carlo Séquin, cominciò a progettare chip VLSI che non usavano l'interpretazione ma avevano un set di istruzioni ridotto ed a quello venivano ricondotte tutte le istruzioni anche le più complesse. Vennero realizzate le **macchine RISC** (*Reduced Instruction Set Computer*).
- ❑ Questi nuovi processori erano molto diversi da quelli commerciali disponibili in quel momento. Poiché queste nuove CPU non avevano bisogno di essere compatibili con prodotti già esistenti, i progettisti furono liberi di scegliere il set di istruzioni nuovo ed in grado di ottimizzare le prestazioni totali del sistema. Mentre l'enfasi iniziale fu posta su istruzioni semplici e di dimensione limitate che si potessero eseguire in poco tempo, ben presto si capì che era più importante progettare istruzioni che venissero messe in esecuzione velocemente ricorrendo alle **canalizzazione o pipeline**
- ❑ Con il passare degli anni, quando i compilatori erano divenuti molto più efficienti, e le memorie meno costose, i progettisti si accorsero dei **diversi vantaggi offerti dalle macchine RISC**:
 1. da test effettuati ci si accorse che per il 90% del tempo il processore utilizza sempre un piccolo sottoinsieme di istruzioni elementari
 2. si poteva ottimizzare i tempi predisponendo e realizzando istruzioni semplici e completabili in un singolo ciclo di clock
 3. Si poteva provvedere a utilizzare semplici istruzioni di trasferimento tra la CU e la Memoria (*store*) e tra Memoria e CU (*load*) e sfruttare (ed aumentare) i registri all'interno della CPU per limitare gli accessi in memoria (e in più l'uso della cache)



MACCHINE CISC e RISC

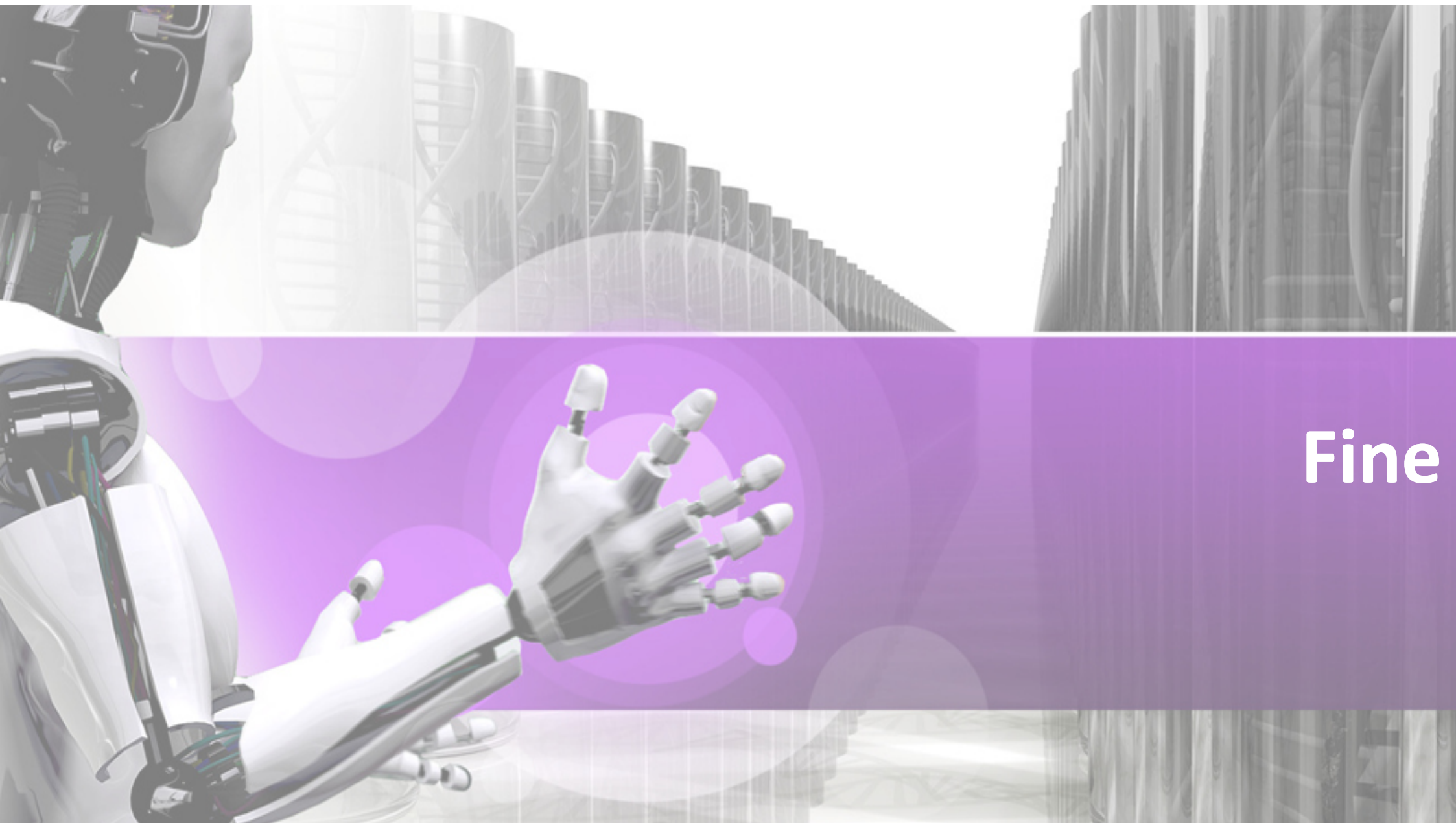
Sintesi

Architettura CISC (Complex Instruction Set Computer)

- ❑ Istruzioni di dimensione variabile
- ❑ Formato variabile
 - ❖ Decodifica complessa
- ❑ Operandi in memoria
 - ❖ Molti accessi alla memoria per istruzione
- ❑ Pochi registri interni
 - ❖ Maggior numero di accessi in memoria
- ❑ Modi di indirizzamento complessi
 - ❖ Maggior numero di accessi in memoria
- ❑ Durata variabile della istruzione
 - ❖ Conflitti tra istruzioni più complicati
- ❑ Istruz. Complesse: più veloci ma pipeline più complicata
- ❑ Codice compatto e facilità di debug

Architettura RISC (Reduced Instruction Set Computer)

- ❑ Istruzioni di dimensione fissa
 - ❖ Fetch della successiva senza decodifica della prec.
- ❑ Istruzioni di formato uniforme
 - ❖ Per semplificare la fase di decodifica
- ❑ Operazioni ALU solo tra registri
 - ❖ Senza accesso a memoria
- ❑ Molti registri interni
 - ❖ Per i risultati parziali senza accessi alla memoria
- ❑ Modi di indirizzamento semplici
 - ❖ Con spiazzamento, 1 solo accesso a memoria
 - ❖ Durata fissa della istruzione
- ❑ Istruz. semplici => pipeline più veloce
- ❑ Codice più complesso, ma facilmente producibile e ottimizzato dal compilatore



Fine