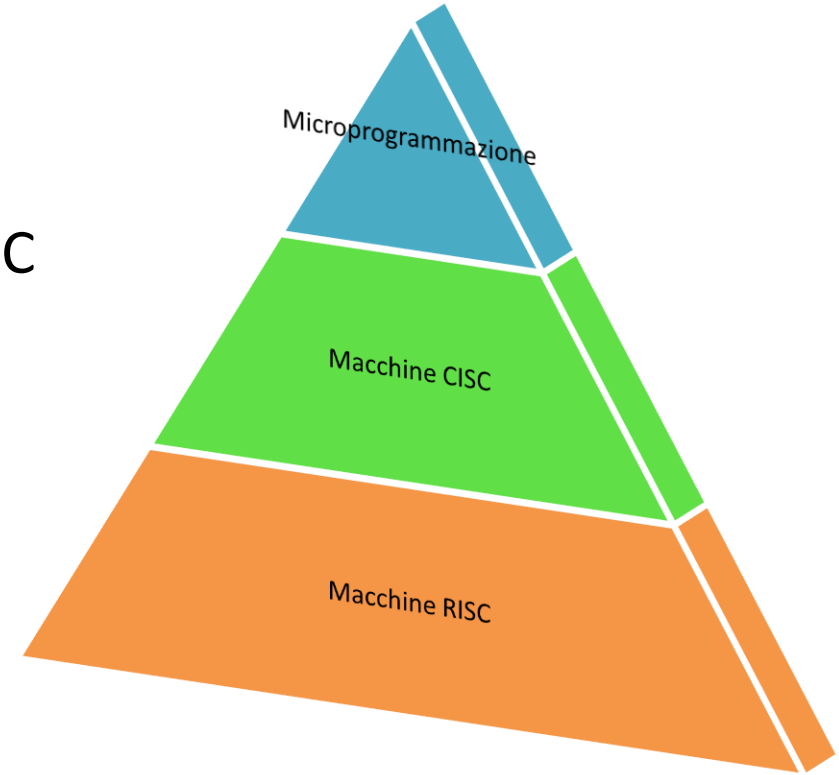
Architettura
degli Elaboratori Elettronici

ARGOMENTI DELLA LEZIONE

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



Macchine CISC

MACCHINE CISC

Generalità

- Lo sviluppo tecnologico e la non ottimizzazione dei compilatori portò alla fine degli anni ’70 alla realizzazione di architetture in grado di eseguire istruzioni complesse. Nacquero, cioè, le **macchine CICS** (*Complex Instruction Set Computing*)
- Le macchine CICS (es.: Intel x86) hanno un **set di istruzioni numeroso** (dalle 200 alle 300 istruzioni), spesso **a lunghezza variabile** alle quali sono associati **svariati modi di indirizzamento**
  - Vantaggio: codice compatto (miglior debug)
  - Svantaggio transcodificatore complesso

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)** *Sposta 100byte consecutivi a partire dall’indirizzo in memoria dato da 50 + [R8] all’indirizzo 15 + [R5]*

MACCHINE CISC

Storia

- Per tutti gli anni ‘50 e ‘60 l’uso di istruzioni macchina molto complesse era ampiamente giustificato dalla tecnologia disponibile in quegli anni:
  1. 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 consentiva di ridurre il gap semantico tra il linguaggio ad alto livello e il linguaggio macchina
  2. La **RAM era costosa** e scarseggiava, e i processori di quegli anni non erano in grado di gestire grandi spazi di indirizzamento. Dunque l’uso di istruzioni macchina molto espressive permetteva di generare eseguibili più corti
  3. **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
  4. 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 una ROM)
     - 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)

MACCHINE CISC

Le istruzioni complesse

- Le **istruzioni complesse richiedono più cicli di clock** (o un clock molto lungo) per essere eseguite, molti accessi in memoria ed un tempo riservato alla fase di load molto 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**, modalità nella quale 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 in hardware un **numero improponibile di componenti**
- 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 più 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 elaborarle

MICROPROGRAMMAZIONE

Generalità

- Una alternativa alla comune realizzazione di un transcodificatore combinatorio fu proposta negli anni Cinquanta da **Maurice Vincent Wilkes** che dotò la CU con un **decodificatore sequenziale** e con **una memoria di controllo di sola lettura in cui archiviare le sequenze di comandi** che venivano trasformati da circuiterie dedicate in segnali di controllo: **l'Unità di Controllo micro-programmata (UC-M)**
  - La presenza di un transcodificatore combinatorio 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 **una micro-macchina**

MICROPROGRAMMAZIONE

Micro-macchina

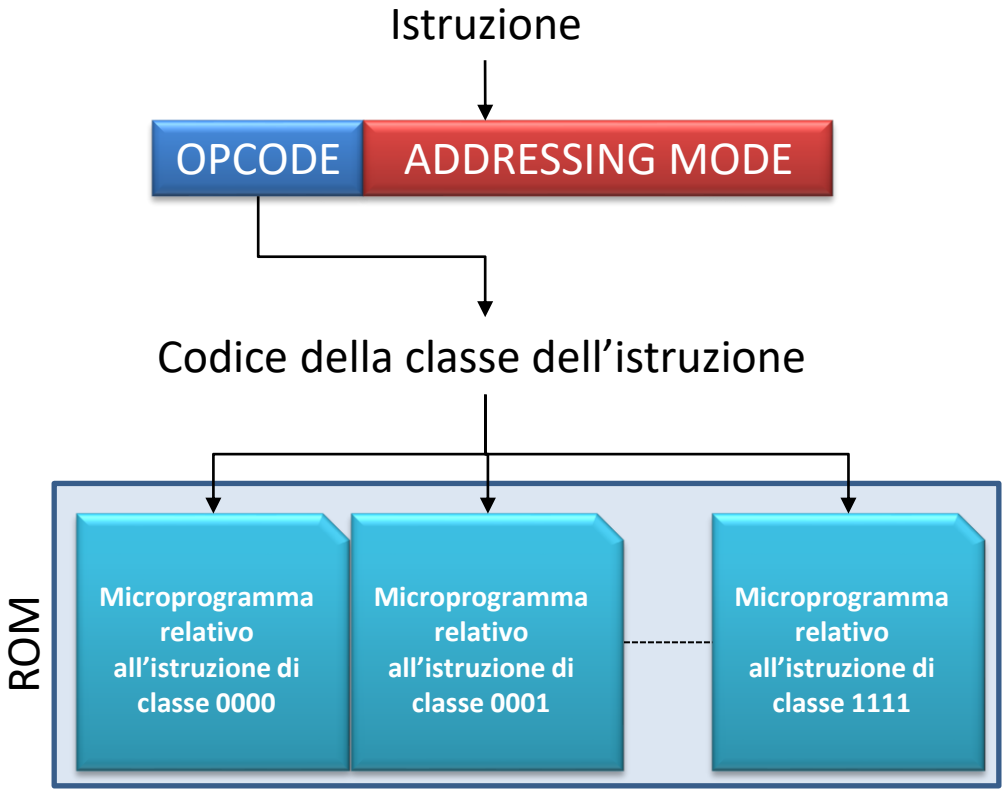
- L'idea è quella di far corrispondere ad una singola istruzione macchina l'**esecuzione di un micro-programma 'scritto' in una memoria non modificabile** (*read only memory,* ROM)
- Il processore in questo caso ha una struttura logica e fisica che ha la funzione di interpretare programmi scritti in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

Osservazione. Il contenuto della ROM è fisso e non può essere modificato; esso è determinato e reso permanente durante la 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

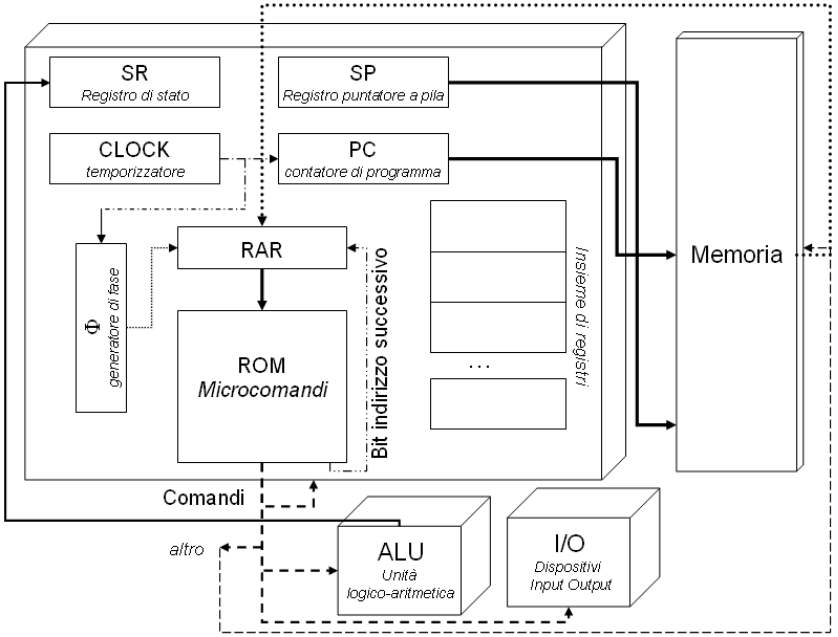
- Per ogni istruzione che la CU deve interpretare e da cui genera un serie di comandi, la stessa CU-M deve fare in modo di eseguire un **micro-programma** che è costituito da un insieme di **micro-istruzioni** il cui numero dipende dal numero di **micro-operazioni** che debbono essere effettuate e dal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



MICROMACCHINA

Componenti basi

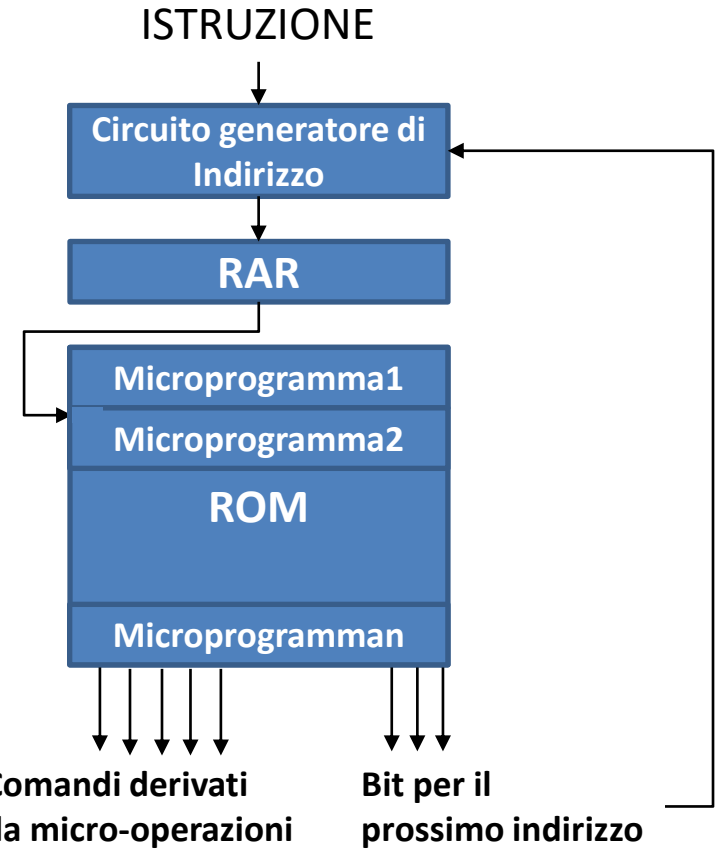
- Tra le componenti essenziali della micro-macchina ci sono:
  - Il **generatore di indirizzo** cioè il circuito predisposto al calcolo dell'indirizzo della successiva microistruzione
  - Il **registro RAR** (ROM address register) un registro al cui all'interno è presente l' indirizzo alla micro-istruzione successiva 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 micro-istruzione costituita da tante micro-operazioni che possono coinvolgere ogni componente della macchina (ALU, memoria)
- Una volta eseguita una micro-istruzione 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 micro-istruzione successiva
- I micro-programmi possono usare anche delle **micro-subroutine** (per svolgere compiti ripetitivi) e quindi necessita di un registro di ritorno (es.: SBR)



MICROPROGRAMMAZIONE

Microprogramma:esecuzione

- Riassumendo un indirizzo può essere raggiunto con:
  1. Mapping dell'OPCODE dell'istruzione ai bit che rappresentano il primo indirizzo della microistruzione della routine opportuna
  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

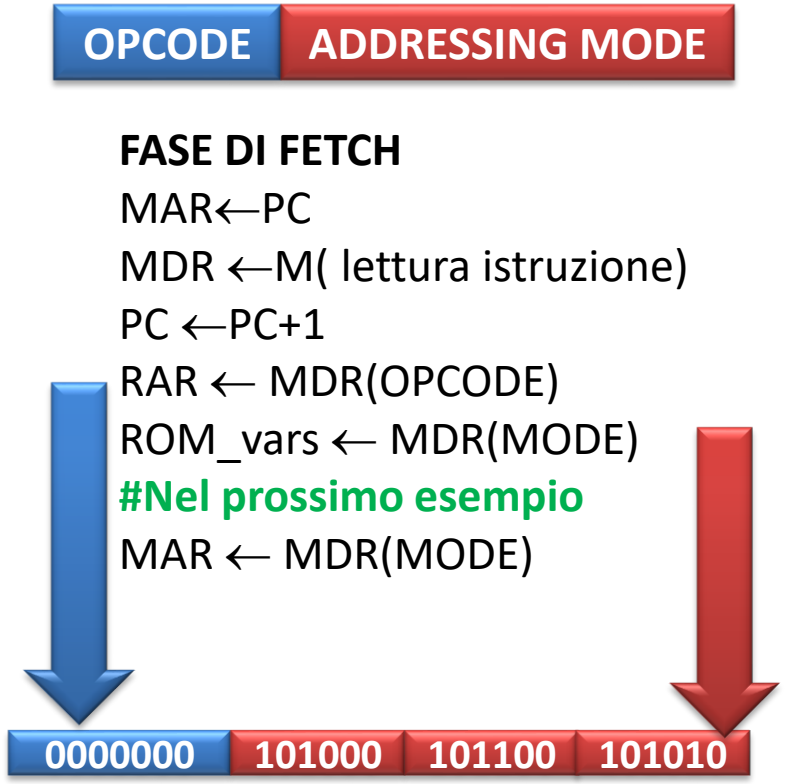
Microprogramma: esempio

- Formato istruzione
  - ADDM 40, 42, 44

    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

Formato in bit
ADDM Md , Ms , Mt
6bit | 6bit 6bit 6bit



MICROPROGRAMMAZIONE

Microprogramma: esempio

- Microistruzione nella ROM

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

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

Salti

Bit	Microps	Descrizione
00	Se CD=1 RAR← ADF	Salto incondizionato
01	RAR←ADF e SBR←RAR+1	Chiamata a subroutine
10	RAR←SBR	Ritorno da subroutine
11	RAR←MDR(OP)	Caricamento prossimo Micro-programma

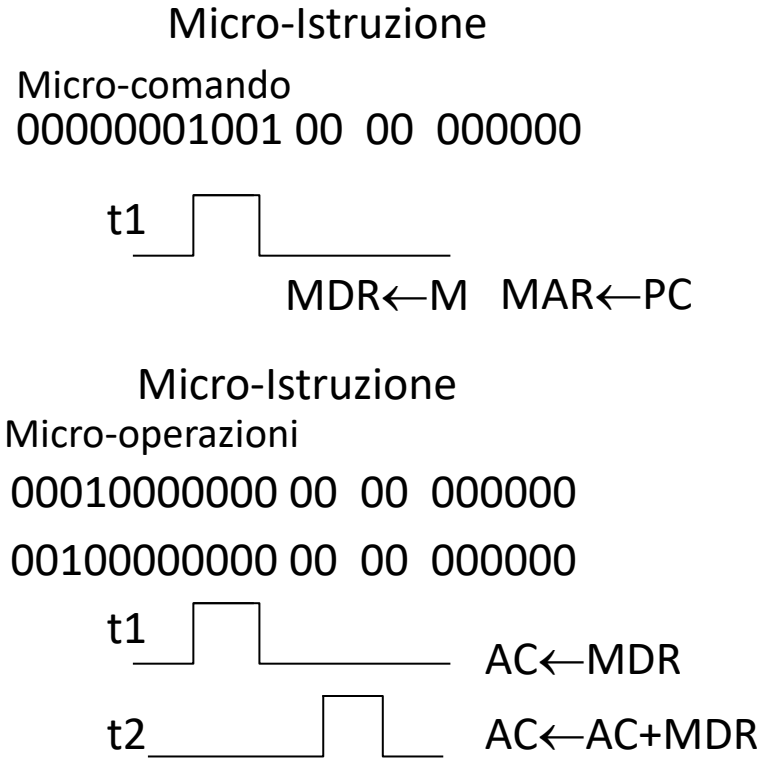
MICROPROGRAMMAZIONE

Microprogramma	Indirizzo	Microistruzione	Micro operazione
ADDM	0000000 (0)	00000000000 01 01 1000011	Salto a subroutine RAR←ADF e SBR←RAR+1
	0000001 (1)	00010000000 01 00 0000010	AC ← MDR e salto a istr. succ.
	0000010 (2)	00000000000 01 01 1000011	Salto a subroutine RAR←ADF e SBR←RAR+1
	0000011 (3)	00100000000 01 00 0000100	AC←AC+MDR e salto a istr. succ.
	0000100 (4)	00001000000 01 00 0000101	MDR ←AC e salto a istr. succ.
	0000101 (5)	00000000010 01 00 0000110	M ←MDR e salto a istr. succ.
	0000110 (6)	00000000000 01 00 1100000	Salto incondizionato a FETCH
…	…	…	…
SBR_ADD_DIR *subroutine per prelievo operando e ritorno all'indirizzo*	1000011	00000010000 00 00 1000100	TMP←MAR e salto a istr. succ.
	1000100	10000000000 00 00 1000101	MAR← MAR&&63 e salto a istr. succ.
	1000101	00000000001 00 00 1000110	MDR← M e salto a istr. succ.
	1000110	00000100000 00 00 1000111	MAR←TMP e salto a istr. succ.
	1000111	01000000000 01 10 0000000	MAR ← MAR>>6 e salto a RAR ←SBR
…	…	…	…
FETCH *carica istruzione successiva*	1100000	00000000100 00 00 1100001	MAR ← PC e salto a istr. succ.
	1100001	00000001001 00 00 1100010	MDR ←M e PC← PC+1 e salto a *routine specificata in AD*

MICROPROGRAMMAZIONE

Microprogramma

- L'esecuzione del micro-programma relativo ad una istruzione è detto **ciclo istruzione**
- Ogni ciclo istruzione è costituito da una o più fasi elementari che comportano l'attivazione di micro-comandi o micro-operazioni relativi ad unità interne o esterne alla CPU
- Ogni istruzione è caratterizzata dal relativo numero di fasi elementari



MICROPROGRAMMAZIONE

Microcodice orizzontale, verticale o diagonale

- È proprio il numero delle micro-operazioni possibili ed il numero massimo di micro-operazioni che debbono essere eseguite in parallelo che determina la possibilità di optare per una organizzazione **orizzontale**, **verticale** o **diagonale** del microcodice
- La struttura del micro-codice, infatti, è solitamente una soluzione di compromesso tra due esigenze diverse: buona flessibilità e potenza operativa contro la contenuta occupazione di memoria dei micro-programmi. 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) micro-programmi distinti (procedure sequenziali). Per ottenere questo si può utilizzare una ROM considerando i vari micro-programmi come tante sottomacchine in una unica macchina sequenziale

MICROPROGRAMMAZIONE

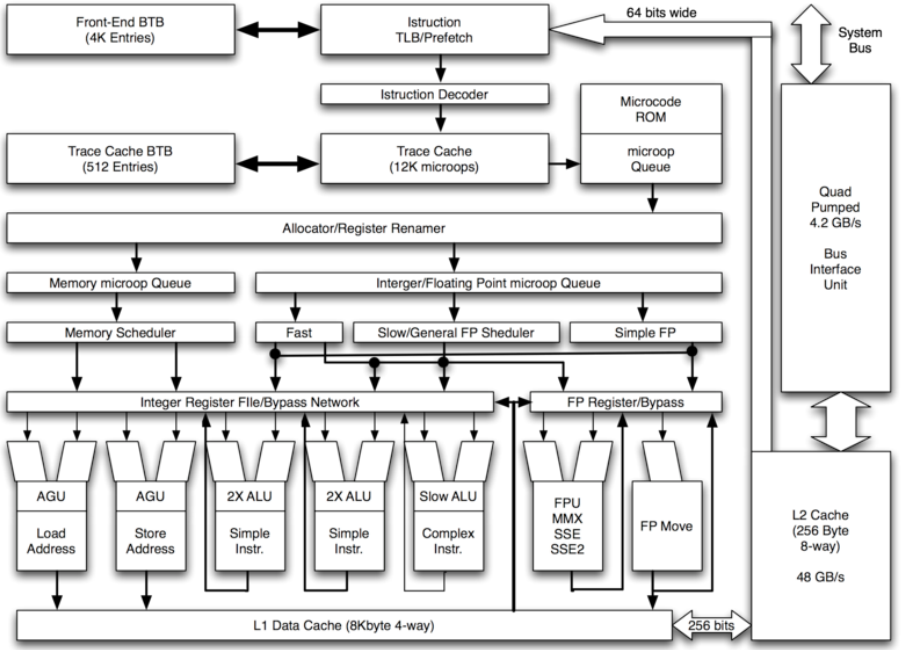
Ottimizzazione del microcodice

- L'**organizzazione di una macchina micro-programmata** 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 spiazzamento 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 quelli 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

- Sono esistiti processori che hanno seguito l'approccio CISC per offrire la **retrocompatibilità** con i vecchi programmi (es.: Pentium IV)
- I processori retrocompatibili decodificano il codice con istruzioni complesse, ma una volta decodificato, esso è 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 (per essere 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

- 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 istruzione 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 dal quelli commerciali disponibili in quel momento. Poiché queste nuove CPU non avevano bisogno di essere compatibili con prodotti già esistenti, i progettisti furano 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 **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 si introdusse la memoria 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 complicate
- Istruzioni 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