EVOLUZIONE DEI MICROPROCESSORI

La nascita del microprocessore

Il microprocessore ha visto la luce nel 1971 a Santa Clara in California, nei laboratori dell'Intel (nel 1968, fu fondata per costruire chip di memoria). Il responsabile del progetto fu Marcian Hoff, che propose come struttura quella di un calcolatore elettronico vero e proprio, ossia lo schema di von Neumann, composta semplicemente da tre soli chip: uno di CPU, uno di RAM, uno di ROM, e con l'utilizzo fondamentale di una struttura a bus.
L'implementazione elettronica dello schema fu attuata con tecnologia MOS da Federico Faggin e portò al **4004**, la prima CPU su un chip o microprocessore.
L'interesse suscitato da questo tipo di prodotto superò ogni attesa, spingendo l'azienda a progettare e immettere sul mercato, nel 1974, una versione con parallelismo ad otto bit, notevolmente più potente della vecchia a soli quattro bit, il famoso **8080**.

Le linee di processori Intel e Motorola

Nell'individuare le tappe dell'evoluzione del microprocessore si può fare riferimento alle linee di prodotto di due aziende di semiconduttori leader mondiali nel settore.

Linea Intel

Dopo l'8080, fu raddoppiato il parallelismo con lo sviluppo dell'**8086** un dispositivo a 16 bit messo sul mercato nel 1978. Va però ricordato l' **8088** uscito un anno dopo non tanto perché segnò un particolare progresso tecnologico, ma perché fu adottato dall'IBM per il suo primo personal computer: PC/XT.
In seguito l'Intel progettò l' **80286**, più avanzato dei suoi predecessori e compatibili con l'8086. L'80286 fu adottato dall'IBM per i suoi PC/AT ed ebbe un enorme successo.

Nome	Anno	N° Transistor	Data Bus	Spazio indirizzamento
4004	15/11/1971	2300	4	1 K
8008	01/04/1972	3500	8	16 K
8080	01/04/1974	6000	8	64 K (PC Altair)
8085	01/03/1976	6500	8	64 K
8086	08/06/1978	29000	16	1 M
8088	01/06/1979	29000	8 (16)	1 M
80186	01/01/1982		16	1 M
80286	01/02/1982	134000	16	16 M (1 G virtuale)
80386 DX	17/10/1985	275000	32	4 G (64 T virtuale)
80386 SX	16/06/1988	275000	16 (32)	4 G (64 T virtuale)
80486 DX	10/04/1989	1200000	32	4 G (64 T virtuale)
80486 SX	22/04/1991	1185000	32	4 G (64 T virtuale)
80486 DX2	03/03/1992	1200000	32	4 G (64 T virtuale)
80486 DX4	07/03/1994	1600000	32	4 G (64 T virtuale)
Pentium 60-66	22/03/1993	3100000	64	4 G (64 T virtuale)
Pentium 75-166	1994-1996	3200000	64	4 G (64 T virtuale)
P6 150-200	01/11/1995	21000000	64	64 G (64 T virtuale)

Linea Motorola

Il primo processore presentato dalla Motorola fu il **6800**, introdotto non molto tempo dopo l'8080. Il 6800 era confrontabile, anche se non compatibile, con il prodotto Intel. Quando, negli anni Settanta, la Motorola introdusse il **68000** si ebbe una sostanziale diversificazione fra le due aziende. Infatti, il 68000 pur avendo un bus a 16 bit, presentava una struttura di registri a 32 bit e uno spazio di memoria indirizzabile di 16 MB. Grazie a queste caratteristiche questo processore fu adottato da costruttori come Apple, Atari e Amiga.
Importante ricordare anche che il successivo **68010** fu utilizzato con UNIX per il concetto di memoria virtuale, caratteristica fondamentale di questo sistema operativo. Il 68010 fu seguito dal **68020**, un trentadue bit completo (registri e bus), con indirizzamento fino a quattro Gbyte. Questo dispositivo divenne il cuore di molte workstation avanzate, come quelle di Sun, Apollo e HP.
In seguito fu presentato sul mercato il **68030** che conteneva nel chip anche l'unità di gestione della memoria (MMU), e poi il **68040** che inglobava nel chip un coprocessore a virgola mobile e una memoria cache.
Analizzando gli attuali prodotti delle linee Intel e Motorola si può affermare che essi sono sostanzialmente confrontabili quanto a prestazioni e complessità.
Ad es. il 68040 contiene 1.2 milioni di transistori, mentre l'80486 ne contiene 1.16 milioni.

Nome	Anno	Size registri	Size data bus	Spazio indirizzamento
68000	1979	32	16	16 M
68008	1982	32	8	4 M
68010	1983	32	16	16 M
68012	1983	32	16	2 G
68020	1984	32	32	4 G
68030	1987	32	32	4 G
68040	1989	32	32	4 G

Microprogrammazione

Il concetto di microprogrammazione non è affatto originale del microprocessore, ma risale alla fine degli anni ’50, introdotto da Wilkes: È l’approccio per cui le funzioni dell’unità di controllo sono suddivise in un numero di operazioni elementari, le microistruzioni, molto più semplici delle normali istruzioni.
Le varie sequenze di microistruzioni sono registrate in una memoria di sola lettura (ROM), molto più veloce della RAM. Il vantaggio principale della microprogrammazione risiede nella sua flessibilità.
Quest’approccio è stato adottato in tutti i microprocessori sia Intel sia Motorola e la ROM è ormai parte integrante del chip.
La microarchitettura di tutte le CPU Intel è simile, perché sono tutte un’evoluzione della prima: l’8086. L’8086 usa un misto di microcodice e logica cablata (circuiti hardware specializzati) per dare una buona funzionalità minimizzando la dimensione del microcodice.

La pipeline

La macchina tradizionale di von Neumann appartiene a tipo **SISD** (Single Instruction Single Data) ha un solo flusso di istruzioni (in pratica, un programma) eseguito da una CPU ed una memoria che contiene i suoi dati.
La prima istruzione è presa dalla memoria e poi eseguita.
Poi è prelevata ed eseguita la seconda istruzione.
Tuttavia, anche entro questo modello sequenziale è possibile avere una quantità limitata di parallelismo, per esempio, prelevando una nuova istruzione e cominciando ad eseguirla prima che quella in esecuzione sia finita.
In altre parole nella stessa CPU si eseguono contemporaneamente più parti di azioni diverse come in una catena di montaggio: allora basta costruire l'hardware con diverse unità funzionali.

Parallelismo temporale: è dato dalla suddivisione del ciclo di esecuzione di un'istruzione in più fasi, ciascuna delle quali è assegnata ad una differente unità funzionale.
Nella tabella si considera una CPU costituita da cinque unità funzionali.

P1	P2	P3	P4	P5
Unità di prelevamento istruzione	Decodifica istruzione	Unità di calcolo degli indirizzi	Unità di prelevamento operandi	Unità di esecuzione istruzione

Durante il primo intervallo di tempo, l'istruzione è presa dalla memoria da P1.
Nel secondo intervallo di tempo, l'istruzione è passata a P2 per essere analizzata, mentre P1 prende un'altra istruzione.
In ognuno degli intervalli successivi, una nuova istruzione è prelevata da P1, mentre le altre istruzioni sono passate ad un'unità successiva lungo il percorso.

P1	Unità di prelevamento istruzione	I1	I2	I3	I4	I5	I6	I7	I8	I9
P2	Decodifica istruzione		I1	I2	I3	I4	I5	I6	I7	I8
P3	Unità di calcolo degli indirizzi			I1	I2	I3	I4	I5	I6	I7
P4	Unità di prelevamento operandi				I1	I2	I3	I4	I5	I6
P5	Unità di esecuzione istruzione					I1	I2	I3	I4	I5
		T1	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**

Questa organizzazione è chiamata **macchina a pipeline**;
due concetti collegati sono:
la *superscalarità*, consiste di più pipeline parallele cui smistare diversi tipi di istruzione;
Parallelismo spaziale: la CPU carica simultaneamente più istruzioni e le esegue in parallelo su un certo numero di unità di esecuzione specializzate identiche. Ciascuna delle unità di esecuzione parallela sfrutta a sua volta il parallelismo temporale utilizzando al suo interno una pipeline (pipeline integer e float).
Il *superpipelining*, consiste nell'aumentare i passi in cui è suddiviso il trattamento dell'istruzione.

Se ciascun passo (intervallo di tempo) è di *n* nsec, ci vogliono *5n* nsec per eseguire un'istruzione. Tuttavia a P5 arriva un'istruzione completa ogni *n* nsec, ottenendo un incremento di velocità di cinque volte.
In condizioni ottimali, si esegue un'istruzione per ogni ciclo, invece di una ogni cinque cicli, dando una velocità media di esecuzione di un'istruzione per ciclo.
Sfortunatamente, alcuni studi hanno dimostrato che il 30% delle istruzioni sono salti e questo riduce l'efficacia della pipeline.
Quando si incontra una salto, l'istruzione successiva da eseguire potrebbe essere quella seguente il salto oppure quella all'indirizzo del salto. Poiché l'unità di prelevamento istruzione non sa quale prendere, finché il salto è stato eseguito, si ferma fino all'esecuzione dello stesso. Di conseguenza la pipeline si svuota.
Il salto ha effettivamente causato la perdita di quattro cicli, con una frequenza di un salto ogni tre istruzioni è chiaro che la caduta di prestazioni è sostanziale.
Per riguadagnare parte di queste prestazioni, la cosa più semplice da fare è sperare che il salto non sia fatto, e continuare a riempire la pipeline come se il salto fosse una semplice istruzione aritmetica. Se risulta che il salto non è effettuato, allora non abbiamo perso nulla; in caso contrario, dovremo spazzare via, con un'operazione chiamata **squashing**, le istruzioni che si trovano attualmente nella pipeline e ripartire.
Anche lo squashing causa però dei problemi. In certe macchine, infatti, come effetto collaterale del calcolo dell'indirizzo, può essere modificato un registro. Se l'istruzione da spazzare via ha modificato uno o più registri, questi devono essere ripristinati, il che significa che deve esserci un meccanismo di registrazione dei loro valori originali.

Che cosa si può fare per migliorare le prestazioni?
Per cominciare se si può prevedere la direzione del salto, potremmo acquisire l'istruzione adeguata ed eliminare la penalità.
Sono possibili due tipi di previsione:
quello statica (in fase di compilazione) e quella dinamica (in fase di esecuzione).
Con la previsione statica, il compilatore fa una stima di tutte le istruzioni che genera. Con il ciclo FOR per es. la previsione di un salto all'indietro all'inizio del ciclo è valida la maggior parte delle volte.
Se si esegue un test su una condizione improbabile, come una chiamata al sistema che restituisca un codice di errore, è probabile che il salto non sia effettuato.
Spesso per questi casi si usano diverse istruzioni e uno sguardo al codice operativo fornisce un sostanziale aiuto.
L'altro metodo di previsione è quello dinamico.
Durante l'esecuzione, il microprogramma costruisce una tabella degli indirizzi che contengono salti, e tiene conto del comportamento di ognuno. Questo metodo, però, tende a rallentare la macchina a causa delle registrazioni.

Memory Management Unit (MMU)

Poiché sistemi operativi come UNIX hanno una visibilità unificata di tutte le risorse di memoria (da quelle di lavoro a quelle di massa) diventa necessario esercitare un controllo centralizzato su tutta la gerarchia. Questa funzione, per la parte hardware, è svolta dal cosiddetto Memory Management Unit, che è direttamente incorporato nei microprocessori più potenti.

La biforcazione CISC-RISC

Le architetture tradizionali di un microprocessore sono chiamate **CISC** (*Complex Instruction Set Computer*) dove ad un aumento delle prestazioni corrisponde un aumento della complessità e del set di istruzioni.

Infatti, tra i numerosi cambiamenti sviluppatisi negli ultimi anni possiamo considerare in particolare una crescita costante della complessità del firmware, con microistruzioni sempre più complicate e numerose, nell'intento di ridurre la distanza ("gap semantico") tra linguaggio macchina e linguaggi di alto livello. Il risultato è che negli elaboratori il numero di istruzioni di base è cresciuto a dismisura (diverse centinaia). Ma se si analizza l'utilizzazione pratica di set di istruzioni così estesi, si trova che statisticamente solo un numero molto ridotto (90/10: il 90% delle istruzioni CISC possono essere sintetizzate in un sottoinsieme del 10%) viene utilizzato. Un'ulteriore osservazione riguarda il fatto che l'utilità di disporre di un ampio set di istruzioni è diminuito nel tempo a fronte dei progressi compiuti dalle tecniche di sviluppo dei compilatori. Nasce così l'idea di realizzare delle architetture di elaborazione con un set ridotto di istruzioni. Questo nuovo orientamento viene appropriatamente chiamato **RISC** (*Reduced Instruction Set Computer*).

L'obbiettivo fondamentale dell'approccio RISC è, in definitiva, di ridurre al minimo il numero dei cicli di macchina (*clock*) necessari per eseguire le istruzioni di base. Tutte le istruzioni RISC fondamentali hanno la stessa durata (un ciclo macchina), la stessa lunghezza e lo stesso formato. In questa accezione il RISC rappresenta un nuovo livello di ottimizzazione tra hardware e software, in cui il primo viene semplificato al massimo per raggiungere la massima velocità operativa, mentre il secondo si assume l'onere di compensare la rigidità introdotta nell'hardware. La semplicità strutturale del RISC (assenza di microcodice) rende disponibile un'ampia area di silicio, che può essere vantaggiosamente utilizzata per realizzare sul chip dei dispositivi di memoria veloce: registri e cache.
Inoltre per rendere efficace l'utilizzo dei registri vengono strutturati secondo uno schema a finestre sovrapposte (*overlapping register window*), che consente, in ogni fase di elaborazione, di vedere solo quel sottoinsieme di registri che serve allo scopo. Un ulteriore vantaggio della minor area di silicio richiesta è la possibilità di utilizzare tecnologie più veloci ma più dissipative basate su semiconduttori come l'arseniuro di gallio. La contropartita è un aumento del progetto software.
Un fattore che invece gioca a sfavore dei RISC è la ampia potenza delle istruzioni di I/O.

Le macchine RISC

I primi calcolatori erano molto semplici, avevano poche istruzioni e uno o due modi d'indirizzamento. Con la serie 360 dell'IBM (1964) fece comparsa la microprogrammazione, basata su un insieme di istruzioni contenute nella ROM (linguaggio macchina). In pochi anni mini calcolatori come il VAX raggiunsero le 200 istruzioni e la dozzina di modi di indirizzamento. Col passare del tempo le CPU, grazie al largo uso di linguaggi ad alto livello, divennero molto complesse. Questi linguaggi contenevano istruzioni del tipo *If, While* e *Case*, che dovevano essere tradotte utilizzando *MOV, ADD* e *JUMP*, creando un **gap semantico** che rese difficile la scrittura dei compilatori. Per ridurre questo gap si decise di aumentare la complessità del "linguaggio macchina", implementando molte nuove funzioni nel micro codice. Un altro fattore che aumentò l'espansione delle macchine CISC fu la differenza di velocità tra memoria centrale e CPU. Un'istruzione implementata nella ROM è, infatti, molto più veloce della stessa richiamata in memoria centrale. Negli anni '70 la RAM divenne più veloce, progettare micro codice era sempre più difficile, inoltre correggere un errore significava riprogettare e sostituire i chip. Inoltre da uno studio sui linguaggi di programmazione emerse che i programmi sono costituiti per la maggior parte (circa 85%) da istruzioni di assegnamento, confronto e chiamate a funzioni. Si comprese quindi l'inutilità di realizzare micro codice sempre più potente (che rallenta inutilmente la CPU) e l'importanza di passare a macchine RISC. Una macchina RISC è quindi un calcolatore con un piccolo numero di istruzioni verticali, con programmi compilati in una sequenza di micro istruzioni eseguite direttamente dall'HW senza interprete.

Il RISC non nasce in ogni caso con il microprocessore, risale, infatti, alla metà degli anni Settanta, quando John Cock progettò con componenti integrati convenzionali il mini computer IBM 801 nei laboratori di Yorktown Heights. In seguito presso l'Università di Berkeley David Patterson e Carlo Séquin progettarono l'architettura SPARC (Scalable Processor Architecture) venduta dalla Sun Microsystems, mentre presso l'Università di Standford John Hennessey progettò l'architettura MIPS (Microprocessor without Interlocked Pipeline Stages) venduta dalla MIPS Computer Systems.

Il grande dibattito su risc e cisc

Per paragonare le due macchine è necessario utilizzare un **benchmark** o **routine di paragone** che testa la velocità di esecuzione di un programma; ma in quale linguaggio deve essere scritto il programma? Molti programmi sono eseguiti molto velocemente dalle macchine RISC ma questo non dipende solo dall'architettura ma soprattutto dal compilatore e dal sistema operativo. Negli anni '70 furono creati programmi di benchmark che misuravano la prestazione in virgola mobile (**Whetstone**) e per gli interi (**Dhrystone**); dal lavoro di quest'ultimo emerse la netta superiorità della macchine RISC. Un motivo per cui le macchine RISC sono più efficienti è forse da attribuire al fatto che è una tecnologia nuova, libera dagli errori del passato; se si progettasse una CPU CISC partendo da zero, probabilmente sarebbe più veloce di quelli attuali. Hitchcock e Sprunt simularono tre calcolatori (VAX, 68000 e RISC) per controllare la differenza tra elaboratori con finestra di registri sovrapposti e senza; da questo studio si concluse che la finestra di registri sovrapposti aumenta notevolmente la velocità della CPU. Ma anche questo confronto non si può ritenere equo poiché i processori RISC utilizzano una grande parte di chip; se anche i CISC avessero a disposizione la stessa porzione di chip da destinare allo stesso scopo, sarebbero molto più veloci. Inizialmente i programmi di benchmark testavano la macchina per un breve periodo (al massimo 7 secondi); tentativi più lunghi e su vecchi linguaggi come COBOL furono scoraggianti e portarono Patterson a progettare i chip **SOAR (Smalltalk On A Risc)** e **SPUR (Symbolic Processing Using Risc)**. Questi chip ebbero maggior successo ma sollevarono il problema se la tecnologia RISC fosse adatta a lavorare su diversi linguaggi e diverse applicazioni.

Quali sono più facili da scrivere: i compilatori RISC o CISC?

Chiaramente i RISC hanno difficoltà che i CISC non hanno, soprattutto per le LOAD e per i salti ritardati i quali, per funzionare correttamente, hanno bisogno di un NO-OP (nessuna operazione: tempo di attesa) seguente, il che rallenta l'esecuzione.
Un altro problema è l'impaccamento dei registri, che non sussiste nei CISC poiché la penalità di riferimenti in memoria è minore.
I compilatori RISC hanno però un pregio: esiste un solo modo per tradurre una costruzione di un linguaggio ad alto livello, mentre per un compilatore CISC la scelta della traduzione può essere ardua.

Classificazione	SPARC RISC	MIPS RISC	80386 CISC	68030 CISC
Lunghezza parole	32	32	32	32
Spazio d'indirizzo dell'utente in byte	2^{32}	2^{31}	2^{46}	2^{32}
Spazio separato indirizzi e dati	No	No	No	Si
Finale	Grande	Entrambi	Piccolo	Grande
Registri utente	32	32	8	16
Finestre sovrapposte	Si	No	No	No
Modalità d'indirizzamento	4	3	8	18
Modalità indiretta di registri	Si	Si	Si	Si
Somma di due modalità con registri	Si	No	Si	Si
Lunghezza costanti immediate in bit	13	16	32	32
Offset di modalità indicizzata	±4K	±32K	±2G	±2G
Indirizzamento diretto	Fondo 4K	Fondo 32K	32 bit	32 bit
Lunghezza istruzione in byte	4	4	1 - 17	2 - 26
Formato istruzione	4	3	Molti	Molti
Operandi per istruzione	3	3	2	2
Moltiplicazione	Parziale	Si	Si	Si
Divisione	No	Si	Si	Si
Salti assoluti	No	No	Si	Si
Salti relativi al PC	Si	No	Si	Si
Interconnesione	Si	No	Si	Si
Stadi di pipeline	4	5	3	3
Memoria virtuale	Si	Si	Si	Si
MMU su chip	No	Si	Si	Si
Cache su chip	No	Controllore	No	Si
Cache separata istruzione e dati	No	Si	No	Si
Dimensione pagina	4K	4K	4K	256 - 32K
Tabelle di pagine	2	0	2	4
Coprocessore in virgola mobile	Si	Si	Si	Si
Altri coprocessori	1	2	0	1

Note: