di Andrea de Prisco

M.I.P.S.: diamo sfogo alla fantasia

Siamo ormai giunti al quarto appuntamento con MIPS, la serie di articoli di Appunti di Informatica riguardanti i processori e, in particolar modo, la velocità operativa di questi. Dopo aver parlato di MIPS, di processor convenzionali, dotati di prefetch, utilizzanti memoria partizionata, questo mese vedremo altre particolari architetture che consentono performance ancora maggiori. E come dice quel proverbio fisico «nulla si crea e nulla si distrugge», vedremo ancora una volta che si ottengono velocità superiori solo a fronte di un aumento di complessità non sempre indifferente

Performance ideale e reale

C'è un altro proverbio che dice: «la legge (fisica) è uguale per tutti». Diffidate dunque da chi dice: «il nostro computer corre a 100 mips e oltre» a meno che non si tratti di un super calcolatore da svariati miliardi di costo, di quelli, per intenderci, che generalmente trovano posto nelle sale macchine dei centri di calcolo della NASA o del dipartimento della difesa americano.

A meno che non vogliamo fare i furbi, confondendo performance ideale con performance reale e drogando il significato di mips a nostro uso e consumo. In questo caso, infatti, anche il sottoscritto sarebbe in grado di fare una scheda con cinquanta 68.000 e cinquanta rom contenenti istruzioni nulle (NOP), dare corrente e affermare «state ammirando un supercomputer che elabora 100 milioni di operazioni al secondo...». Se poi proviamo a sostituire le 50 rom con una rom unica (tanto il codice è uguale per tutti) come succede nei multiprocessor veri (lo spazio di indirizzamento dei vari processori è lo stesso) i 100 e passa mips diventerebbero sicuramente una decina se non meno a causa dei conflitti per l'accesso da parte di 50 processori all'unica memoria. Ma volendo fare gli onesti a tutti i costi, installiamo un embrione di sistema operativo per questo sistema multiprocessor e facciamo girare programmi veri (non delle NOP): scopriremmo che i nostri cinquanta 68.000 sì e no valgono per tre-quattro, con performance reale di pochi, davvero pochi mips. Ma di questo ne parleremo

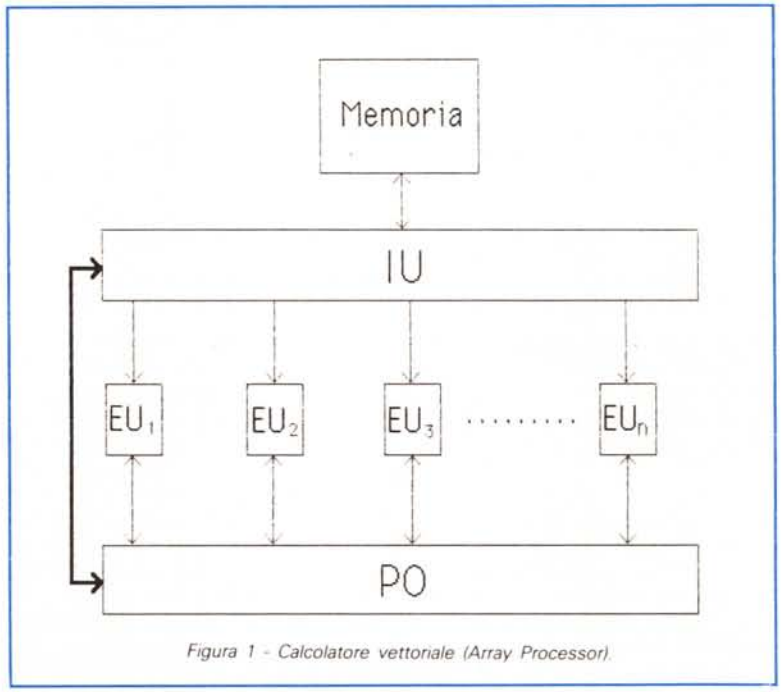

Figura 1 - Calcolatore vettoriale (Array Processor).

più approfonditamente tra qualche numero, quando tratteremo dei sistemi multiprocessore. Si è usato l'esempio dei cinquanta 68.000 solo per ricordarvi il fenomeno della degradazione di performance discusso lo scorso mese in merito alle cosiddette «dipendenze logiche». Ricorderete inoltre che tale fenomeno è congenito nei programmi da eseguire e non è possibile eliminarlo totalmente, ma solo ridurlo al minimo scrivendo i programmi stessi secondo determinati canoni o approntando compilatori ad hoc. Dunque per far «correre» un calcolatore, non basta la super potenza della CPU ma anche i programmi da elaborare devono essere scritti in modo da sfruttare al massimo le particolarità del processore in questione. O viceversa...

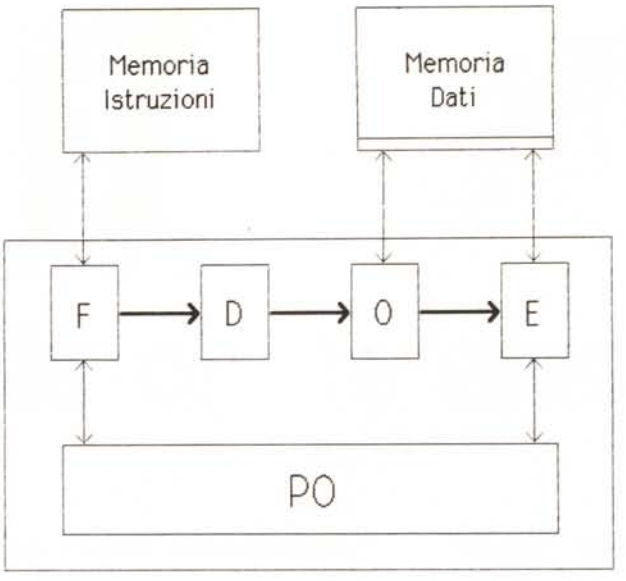

Figura 2 - Soluzione Pipeline. 4 stadi: Fetch, Decodifica, Operandi, Esecuzione.

In che senso?

«Se la montagna (di istruzioni) non va a Maometto, Maometto (il processore) va alla montagna...». Esistono infatti determinate classi di programmi per i quali conviene fare il discorso inverso: non programmi scritti per i processori, ma processori progettati per i programmi. Un esempio tipico sono i calcolatori vettoriali che eseguono velocissimamente (sfruttando al massimo il loro parallelismo interno) i calcoli vettoriali, tipici dei programmi che fanno un massiccio uso di strutture dati di tipo array.

La CPU dei calcolatori vettoriali è detta Array Processor e trovano posto al suo interno non una, ma una bella «sfilza» di unità esecutive (vedi fig. 1) che eseguono in parallelo (una per elemento) le istruzioni vettoriali. Dovendo fare ad esempio la somma di due vettori (che come noto dà come risultato un altro vettore di pari dimensione) si inviano alle unità esecutive in parallelo tutti gli elementi del primo vettore (la EU «i» riceverà l'i-esimo elemento), tutti gli elementi del secondo vettore (discorso analogo), per diffusione il tipo di operazione da eseguire, prelevando da tutte le EU (eventualmente) il risultato dell'operazione. Con questa tecnica si progettano i super calcolatori scientifici che forniscono valori di performance tanto più elevati quante più istruzioni vettoriali vengono usate nei programmi da eseguire (da qui la scientificità dell'utilizzazione). Inutile dire che con programmi normali anche le performance sono «normali» (l'alto grado di parallelismo interno non è affatto sfruttato).

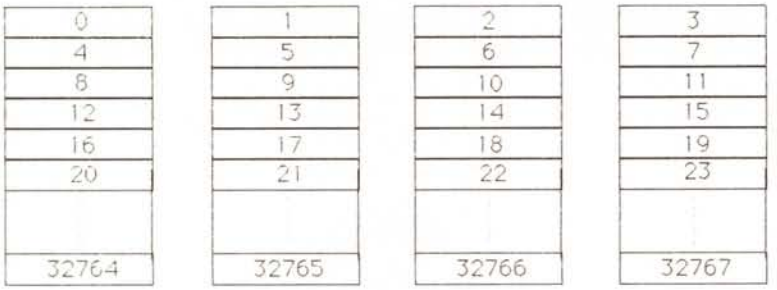

Figura 3 - Organizzazione interlacciata di una memoria da 32K in 4 blocchi.

Soluzioni Pipeline

Negli ultimi due «Appuntamenti» vi abbiamo mostrato i processori dotati di prefetch sottolineandovi il loro funzionamento Pipeline (catena di montaggio). Mentre l'unità istruzioni preleva e decodifica l'istruzione corrente, l'unità esecutiva, o EU, esegue l'istruzione precedente. Il caso dei processori con prefetch si può estendere aumentando il grado di parallelismo e conseguentemente il numero degli stadi del Pipeline. Invece che due sole unità possiamo impiegarne un numero maggiore specializzando ulteriormente le funzionalità di ognuna. Ad esempio potremmo prevedere una prima unità di solo Fetch la quale passa l'istruzione all'unità successiva che si occupa della decodifica. Effettuata anche questa fase, l'istruzione decodificata passa per un'unità atta al prelevamento degli operandi, e infine all'unità di esecuzione vera e propria.

Ovviamente dopo che un'unità ha espletato la sua mansione sull'i-esima istruzione, ceduto il controllo all'unità successiva (che dunque continua a «lavorare» il «pezzo i») inizia immediatamente il medesimo trattamento sull'istruzione i+1.

Dimenticando momentaneamente i problemi dovuti alle dipendenze logiche (argomento già trattato per sommi capi lo scorso mese), un siffatto processore (mostrato schematicamente in figura 2) ha grado di parallelismo 4: in ogni istante vengono eseguite 4 fasi che in un processore convenzionale sarebbero eseguite in 4 istanti successivi. Nel caso ideale, un processore che utilizzi questa tecnica è ben 4 volte più veloce

del suo «fratello» convenzionale. In realtà ciò non avviene a causa delle già citate dipendenze logiche delle operazioni. Pensate ad esempio ad una cella di memoria (o ad un registro) che viene incrementata da un'istruzione e utilizzata dall'istruzione successiva. Scattando un'ideale fotografia sul processore all'opera per queste due istruzioni troveremmo che quando l'unità esecutiva sta eseguendo l'incremento (che nel caso di una cella di memoria richiede ben due accessi, uno in lettura e uno in scrittura) l'unità richiesta operandi non può, parallelamente, prelevare l'operando per l'istruzione successiva, ma deve fermarsi un attimo e aspettare che la EU termini per avere il valore aggiornato del dato e non quello «vecchio». Rallenta oggi, rallenta domani, la performance reale non è quadrupla, ma sì e no doppia. Bella fregatura!

Sistemi Look Ahead

Capirete a questo punto che sforzi per cercare di migliorare il più possibile le prestazioni dei processor ne sono stati fatti molti e in molte direzioni. Sempre sullo scorso numero vi abbiamo mostrato anche l'utilizzazione di memoria partizionata per i programmi e per i dati di questi. In tal modo è possibile, nello stesso istante, accedere a due celle di memoria per prelevare il dato della i-esima istruzione e l'istruzione i+1. Tale partizionamento, unito al meccanismo di funzionamento con prefetch dà sicuramente risultati migliori in quanto a velocità di elaborazione del sistema seppur (come sempre) a fronte di una ulteriore complicazione.

L'estensione più ovvia della memoria partizionata prevede l'utilizzo non di due, ma di un numero maggiore di moduli di memoria distinti. Detto così già si comprende che in tal modo sono possibili molti accessi in memoria contemporaneamente: ma per sfruttare molto tale possibilità è opportuno organizzare in maniera non troppo convenzionale la memoria stessa. Spieghiamoci meglio: avendo parlato di un'unica memoria sottoforma di più moduli distinti, chi non conosce già il problema sicuramente penserà ad una normale suddivisione in blocchi contigui. Ad esempio nel primo blocco le prime mille celle, nel secondo le celle tra 1000 e 1999, nel terzo le celle tra 2000 e 2999 e così via. Così, dovendo accedere alle celle 500, 4000 e 7638, essendo queste in moduli distinti, possiamo prelevarle contemporaneamente inviando le tre richieste ai tre moduli nello stesso ciclo di clock. Coverrete con noi, però, che accessi così «salterellosi» sono sì comuni, ma non certo abituali. Di solito le celle di memoria sono lette «abbastanza» sequenzialmente: ad esempio le istruzioni vengono lette l'una dopo l'altra, finché non bisogna eseguire un salto, per poi riprendere di nuovo in sequenza.

Dal momento che il nostro obiettivo è quello di accedere in un solo ciclo a quante più delle possibili, organizzando la memoria in modo «interlacciato» (non pensate all'Amiga, non c'entra nulla) potremo prelevare N istruzioni in un solo colpo, da dare in pasto al nostro processore velocissimamente, l'una dopo l'altra. In una memoria con organizzazione interlacciata, le celle non si susseguono nello stesso blocco, ma ogni cella presa in considerazione ha la sua cella successiva nel successivo blocco e la sua cella precedente nel precedente blocco, ciclicamente. Ad esempio, disponendo di 4 blocchi, la prima cella della memoria è la prima cella del primo blocco, la seconda cella è la prima cella del secondo blocco, la terza cella è la prima cella del terzo blocco così come la quarta cella è la prima cella del quarto blocco. La quinta cella è la seconda del primo blocco e così via, come mostrato in figura 3. Se utilizziamo un numero di blocchi pari ad una potenza di 2 (4, 8, 16, 32, ecc.) la decodifica indirizzo-posizione (effettiva) è pressoché immediata. Infatti i bit meno significativi indicheranno il blocco, i bit rimanenti la posizione all'interno di questo. Per chiarire meglio «il concetto» facciamo il solito «bell'esempiuccio». Immaginiamo di avere una memoria interlacciata con un numero di moduli pari ad 8. Se la cella da accedere è la numero 465 il nostro problema da risolvere è il seguente: con tale organizzazione, in quale modulo e in quale posizione di questo vado a pescare la cella 465? Facilissimo. Proviamo a trasformare in binario il numero 465 (così ci avviciniamo di più al modo di pensare del computer). In binario, 465 si scrive:

1 1 1 0 1 0 0 0 1

Otto, il numero di moduli di memoria utilizzati, è pari a due elevato a tre.

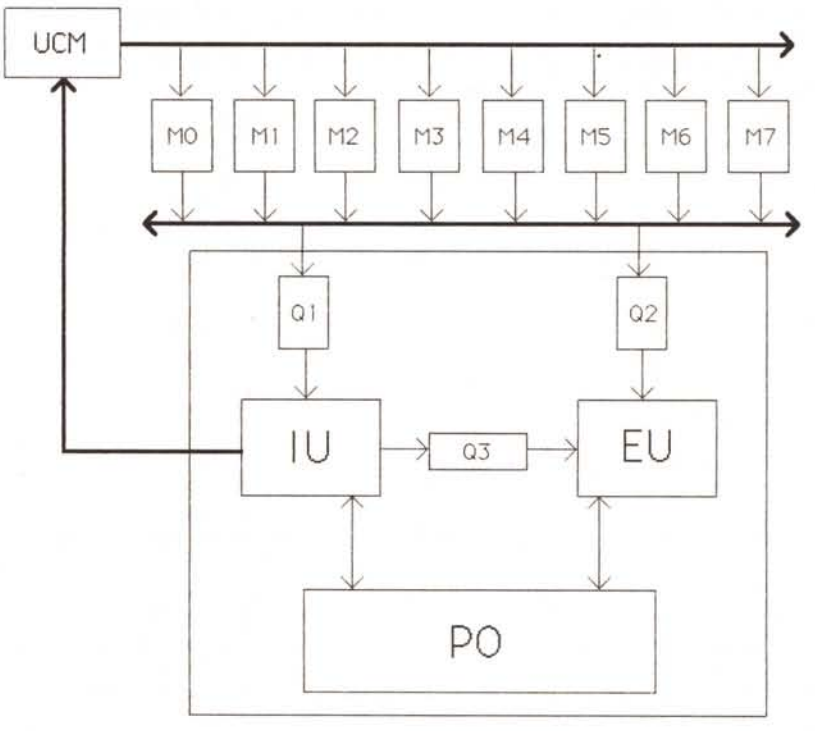

Figura 4 - Sistema Look Ahead. La UCM (unità di controllo memoria) invia le 8 richieste agli 8 moduli di memoria a partire dall'indirizzo ricevuto dalla IU. Nell'ordine, gli 8 dati saranno accodati nella coda Q1 che li passerà, uno alla volta, alla IU.

Spezziamo l'indirizzo di sopra all'altezza del terzo bit:

1 1 1 0 1 0 0 0 1

riconvertiamo questi due numeri in decimale ottenendo 58 e 1. Fatto: la nostra cella 465, e si trova nella posizione 58 del blocco 1 (dunque il secondo, il primo è il blocco 0).

In figura 4 è mostrato un processore interfacciato con una memoria interlacciata.

Tale tipo di sistema, detto Look Ahead (guarda avanti), preleva contemporaneamente un numero di celle di memoria pari al numero di moduli utilizzati e le «schiaffa» (in ordine) nella coda istruzioni dalla quale il processore le preleva l'una dopo l'altra per eseguirle. Si ha un notevole incremento di velocità dato che il tempo di risposta di una coda per cedere il suo primo elemento è ben più breve dei tempi di accesso anche delle memorie più veloci. Naturalmente potrà succedere che una delle istruzioni in coda sia una istruzione di salto, nel qual caso, semplicemente, il processore ignorerà i rimanenti elementi in coda, comunicando il nuovo indirizzo al controller della memoria interlacciata. Semplice, no?

Altre architetture

La nostra carrellata sui processor convenzionali e non (a dire il vero bisognerebbe cominciare a rivedere il significato di «convenzionale», dal momento che oggigiorno sarebbe assurdo progettare un nuovo processore senza nessuna di queste feature) si conclude mostrandovi un tipo di processor parallelo in cui il parallelismo è esplicitato al livello delle operazioni aritmetico-logiche: esse riguardano forse oltre l'ottanta per cento del lavoro che ogni CPU effettua ordinariamente. Abbiamo visto in tutti gli esempi finora mostrati, che per realizzare del parallelismo interno (da qui il nome «processor paralleli») occorre replicare o sdoppiare alcune unità che nell'architettura convenzionale erano presenti in singola copia. Ad esempio i processori funzionanti con Prefetch dispongono della Parte Controllo sdoppiata in IU e EU, nei processor Pipeline si effettuano altre «suddivisioni», gli Array Processor dispongono di molte EU e così via. C'è rimasta una sola cosa che non abbiamo ancora toccato: l'unità aritmetico-logica che come dice il nome è preposta alle operazioni di questo tipo. Di solito funzionano inviando i due operandi, il tipo di operazione da compiere e prelevando il risultato in uscita. Il passo successivo è di prevedere non una, ma diverse ALU, ognuna specializzata per un particolare tipo di operazione. Avremo (figura 5) un'unità per le somme, una per le moltiplicazioni, una per le divisioni, una per i confronti, un'unità di shift e così via. La IU amministrerà queste risorse inviando all'unità competente gli operandi per l'operazione da eseguire. Dato però che si dispongono di più unità, la IU, senza aspettare l'esito della prima operazione, potrà prelevare un'altra istruzione instradando i nuovi operandi verso un'altra unità preposta ad un'altra operazione. Quindi se più operazioni devono essere eseguite da unità diverse possono essere eseguite in parallelo (dipendenze logiche permettendo, naturalmente). Analogamente, se prevediamo che i programmi da eseguire facciano un massiccio uso di addizioni o moltiplicazioni, nulla ci vieta di replicare ancora queste unità in modo da non dover mai fermarci per «unità occupata».

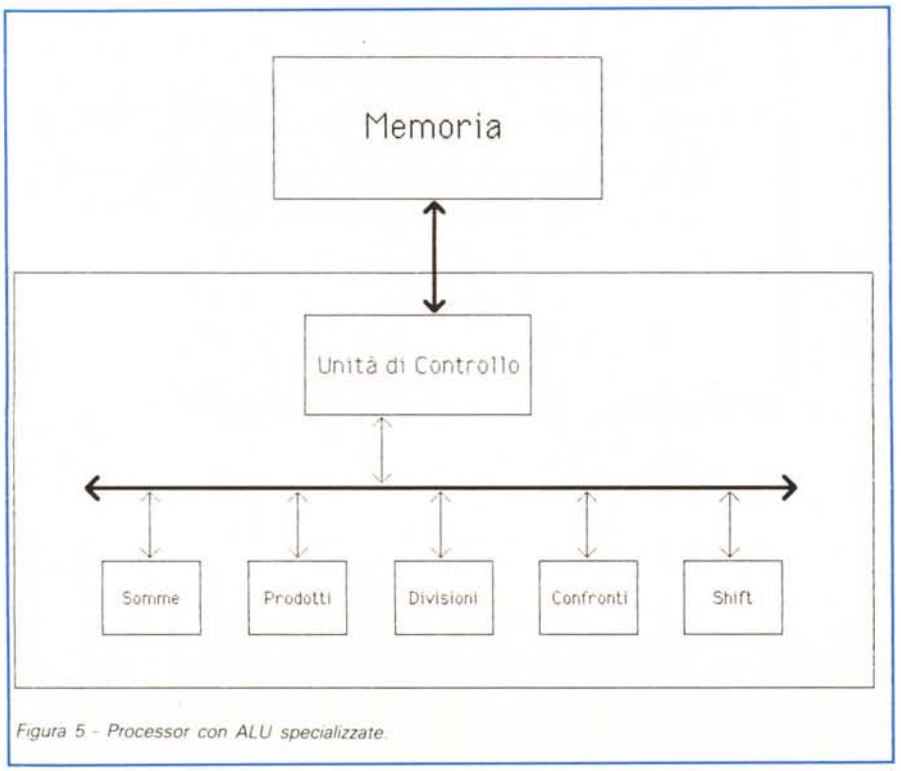

Figura 5 - Processor con ALU specializzate.

Concludendo

Per far correre di più un calcolatore non basta una accurata taratura e qualche goccia d'olio nei punti più critici. Occorre, come visto, rivedere completamente il progetto, partendo innanzitutto dal tipo di programmi che il nostro calcolatore dovrà eseguire. Fissato il tipo di programmi e dunque l'architettura da utilizzare, è necessario che i programmi da eseguire sfruttino al massimo le caratteristiche del processor minimizzando, ad esempio le dipendenze logiche delle istruzioni o sfruttando al massimo il parallelismo fornito. Solo così si riesce ad avere performance di tutto rispetto. E non c'è da meravigliarsi, dunque, se per alcuni programmi «corre» di più un calcolatore che un altro e viceversa: dipende, come detto, dai programmi e dalle architetture.

Da questo (e da altre considerazioni analoghe) l'incomparabilità intrinseca dei calcolatori appartenenti a famiglie diverse.

Allora: è più veloce uno Z80 a 4 MHz o un 6502 a 1 MHz? Ci siamo posti questa domanda tre numeri fa, per anni se la sono posti molti costruttori... ma crediamo che sia proprio come dire «È nato prima l'uovo o la gallina?». Arrivederci! MC

Bibliografia:

Baiardi, Tomasi, Vanneschi: *Architettura dei sistemi di elaborazione*
Ed. Franco Angeli 1987
Andrew S. Tanembaum: *Structured computer organization - Prentice-Hall 1976.*
R. Zaks: *Programmazione del 6502*
G.E. Jackson
G. Kane: *Il Manuale MC680000*
McGraw-Hill 1985