ISA MIPS

Stored Program
I programmi sono sequenze di istruzioni, quindi di dati, e come tutti gli altri dati (semplici “file”) vengono memorizzati nella memoria principale.

Ciclo Fetch Execute
CPU legge (fetch) le istruzioni dalla memoria all'indirizzo segnalatogli dal PC (Program Counter) e le pone nel Register File.
CPU verifica il tipo di istruzioni per verificare quali azioni svolgere e successivamente le esegue (Execute). In seguito incrementa il PC e itera il ciclo.

MIPS
Processore con ISA (Instruction Set Architecture) di tipo RISC (vedi in seguito).
3 operandi.
Stile architettura: registro-registro

Istruzioni e modalità di indirizzamento del MIPS
add, sub
Usano i registri. F.to istr. R-type (Register)

addi
Usa i registri. F.to istr. I-type (Immediate)

lw/sw
Usa i registri e la memoria. F.to istr. I-type (Base)
es.
lw $r1, costante($r2)
Carica nel registro $r1 il valore presente in memoria all'indirizzo ($r2 + costante in Byte)
sw $r1, costante($r2)
Carica in memoria, all'indirizzo ($r2 + costante in Byte), il valore del registro $r1.

beq
Usa registri, memoria e PC. F.to istr. I-type (PC-Relative)

jump
Usa memoria e PC. F.to istr. J-type (Pseudo Direct)

STILI DI ARCHITETTURA
accumulatore (1 solo registro)
Gli operandi sono il dato in registro ed uno in memoria. Non si possono memorizzare dati diversi nel registro perché essendo solo uno viene di volta in volta sovrascritto, quindi si deve ricorrere spesso alla memoria.

registro-memoria
Come l'accumulatore solo esistono più registri. C'è quindi la possibilità di memorizzare dati diversi nei registri, si ricorre meno alla memoria velocizzando i tempi di esecuzione.

registro-registro
Gli operandi sono solo i registri. Questo fa si che le operazioni di dati già salvati nei registri si effettuino velocemente, non dovendo accedere alla memoria. Di contro se si devono effettuare operazioni quando un dato è in memoria bisogna ricorrere alle istr. lw e sw, quindi aumenta il numero di istr. complessivo.

stile linguaggio ad alto livello
Istruzioni molto complesse vengono interpretate direttamente dal processore senza essere compilate. Molto difficile adattare l'hw alle istr. complesse, non era vantaggioso per le istr. comuni.

memoria-memoria
Istruzioni ridotte perché non occorreva caricare i dati nei registri. D'altra parte utilizzare la memoria anziché i registri significa aumentare i tempi di esecuzione.

CISC (Complex Instruction Set Computer) vs RISC (Reduced ...)
CISC trend degli anni '70
RISC evoluzione degli anni '80
CISC ISA set di istr. complesse (linguaggio alto livello), tipo memoria-memoria o registro-memoria
RISC ISA set di istr. semplici e diffuse nei programmi, tipo registro-registro
CISC molti metodi di indirizzamento
RISC pochi metodi di indirizzamento
CISC istr. a f.to variabile
RISC istr. a f.to fisso (32 bit)

MIPS e RISC
Il MIPS è un processore di tipo RISC.
Essendoci pochi istr. a formato ben definito è stato possibile temporizzarle su periodi di esecuzione fissi, permettendo così di essere svolte in parallelo.

PRESTAZIONI (Parte Teorica)

Tempo di CPU e Tempo di risposta
Con tempo di esecuzione intendiamo il tempo di CPU, ovvero il tempo per eseguire il programma speso solo dal processore. Il tempo di risposta è più lungo del tempo di CPU e tiene conto, oltre del tempo speso dal processore, di quello speso per accedere all'I/O e il tempo delle code di attesa nel caso si stiano eseguendo più programmi contemporaneamente.

Tempo di esecuzione e Performance
Performance (Troughput): quanti programmi in un secondo.
Nel valutare un singolo programma è semplicemente 1 / tempo di esecuzione, considerando più programmi contemporaneamente la performance incide sul tempo di risposta. Infatti se in un laboratorio si aggiunge una workstation, il tempo di esecuzione (tempo di CPU) resta uguale, se non abbiamo cambiato le caratteristiche del processore, ma aumenta la performance e, svolgendosi più programmi in un secondo, il proprio dovrà attendere meno tempo in coda, riducendo così il tempo di risposta.

Tempo di esecuzione, frequenza di clock e conseguenza sui tempi di accesso alla memoria
Il tempo di esecuzione si calcola come il prodotto tra il numero dei cicli di clock per T (periodo del ciclo di clock).
Per ridurre il tempo di esecuzione si può
a) ridurre il numero dei cicli di clock
b) ridurre la durata del ciclo di clock (T), sinonimo di aumentare la frequenza (“overclock”)

Risulta più vantaggioso il primo punto perché “overclockando” il processore non si raggiungono risultati proporzionali alla modifica. Infatti se anche aumenta la velocità di elaborazione del processore, questo resta vicolato ai tempi di accesso alla memoria e dovrà aumentare il numero dei cicli di clock, nei tempi di attesa delle operazioni che accedono alla memoria.

Valutazione delle prestazioni poco oggettiva: MIPS
MIPS è acronimo di Milion of Instructons Per Second ed è, oltre che il nome del processore, un indice usato per valutare un programma. Tuttavia non è oggettivo perché il numero di istr. per secondo dipende ovviamente da quale programma si va a misurare. Se una macchina A ha un MIPS superiore ad una macchina B potrebbe essere che A sia più veloce di B per quel particolare programma e non in generale.

Valutazione delle prestazioni più oggettiva: Benchmarks
Programmi scritti in C, C++, Fortran compilati su una macchina per testare la velocità del processore oltre che la qualità del compilatore.

SPEC (System Performance Evaluation Cooperative)
Consorzio tra le principali aziende di hw che prevede una serie di benchmarks riferiti a diverse classi di applicazione: compilatori, sw scientifico, grafica ...

Formula utilizzata dallo SPEC: $\sqrt[n]{\prod_{i=1}^{n} Trif_i/T_i}$

Media geometrica pesata (produttoria) tra *n* programmi di benchmarks basata sullo Speedup tra lo stesso programma *i* eseguito su una vecchia macchina *Trif*. Il processore di riferimento viene aggiornato all'aggiornare dei nuovi standard benchmarks.

Legge di Amdahl
Esiste un limite (*s*) oltre il quale non si può ottimizzare ulteriormente un programma (limite allo Speedup) dovuto ad un tempo che il processore impiega senza di fatto eseguire il programma ma per accedere alla memoria o usare l'I/O.

Stando alla legge di Amdahl è più conveniente velocizzare (*n* più piccolo) le istruzioni per cui il processore impiega la maggior parte del tempo ad eseguire l'istruzione stessa, ovvero le istruzioni più semplici.

PRESTAZIONI (Esercizi)

T (periodo) = durata del clock (sec)

F (frequenza) = 1 / T (1 / sec. = hz)

*Texe (tempo esecuzione) = numero cicli di clock * T (sec)*

Performance (Troughput) = 1 / Texe (programmi / sec)

IC = numero istruzioni del programma

CPI = cicli medi per istruzione

*numero cicli di clock = IC * CPI*

*Texe = IC * CPI * T = (IC * CPI) / F*

*MIPS = IC / (Texe * 10^6) = F / (CPI * 10^6) (milioni si istr. al sec, senza 10^6 è istr. al sec.)*

Legge di Amdahl

*s = Texe / Tnoncomprimibile → Tnoncomprimibile = (1 / s) * Texe*

*Tottimizzabile = Texe – Tnoncomprimibile = (1 – 1 / s) * Texe*

Tottimizzato = Tnoncomprimibile + Tottimizzabile / n (n è il fattore di ottimizzazione)

Tottimizzato = (1/s) * Texe + [(1 – 1 / s) * Texe] / n

*Tottimizzatomin = Tottimizzato per n tendente a infinito = (1*s) * Texe*

Speedupmax = Texe / Tottimizzatomin = s

Esempi esercizi con la legge di Amdahl (del prof. Orlando)
Si migliora una macchina in modo che le operazioni in virgola mobile risultino 5 volte più veloci. Uno dei benchmark viene eseguito in 100 secondi dal vecchio programma, che peso nell'esecuzione del programma devono avere le operazioni in virgola mobile perché lo Speedup sia pari a 3?

Dati:
n = 5 (operazioni 5 volte più veloci)
Texe = 100 sec (vecchio programma in 100 sec)
Speedup = 3

Richieste:
1 – 1 / s ? (peso nell'esecuzione delle operazioni, peso del tempo ottimizzabile)

Svolgimento:
*Tottimizzato = (1/s) * Texe + [(1 – 1 / s) * Texe] / n*

da cui

*1 / s = (n * Tottimizzato – Texe) / (n * Texe – Texe)*

Speedup = Texe / Tottimizzato = 3

Tottimizzato = Texe / Speedup = 100 / 3 sec

Quindi

*1 / s = (5 * 33,3 sec - 100 sec) / (5 * 100 sec - 100 sec) = 1 / 6*

e

1 – 1 / s = 1 – 1 / 6 = 5 / 6

Un programma viene eseguito in 100 sec in una macchina (80 sec per le moltiplicazioni).
Di quanto dobbiamo incrementare la velocità delle moltiplicazioni se vogliamo che il programma sia 4 volte più veloce?
Quale lo Speedupmax raggiungibile?

Dati:
Texe = 100 sec
Tnoncomprimibile = 100 sec – 80 sec = 20 sec (80 sec per le moltiplicazioni)
Tottimizzato = 100 / 4 sec = 25 sec (programma 4 volte più veloce)

Richieste:
n? (fattore ottimizzazione, di quanto incrementare le moltiplicazioni)
Speedupmax? (= s)

Svolgimento:
*Tottimizzato = (1/s) * Texe + [(1 – 1 / s) * Texe] / n*

da cui

*n = [(1 – 1 / s) * Texe] / (Tottimizzato – Texe / s)*

s = Texe / Tnoncomprimibile = 100 sec / 20 sec = 5 (= Speedup max)

Quindi

*n = [(1 – 1 / 5) * 100 sec] / (25 sec – 100 / 5 sec) = 16*