RISC vs CISC

Argomento:

- Architetture CISC e RISC
- Confronto

Materiale di studio:

- Sezioni 15.1, 15.4, 15.5

CPU nel tempo

Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Year developed	1973	1978	1989	1987	1991
Number of instructions	208	303	235	69	94
Instruction size (bytes)	2–6	2–57	1–11	4	4
Addressing modes	4	22	11	1	1
Number of general-purpose registers	16	16	8	40 - 520	32
Control memory size (kbits)	420	480	246	—	—
Cache size (kB)	64	64	8	32	128

CISC → RISC

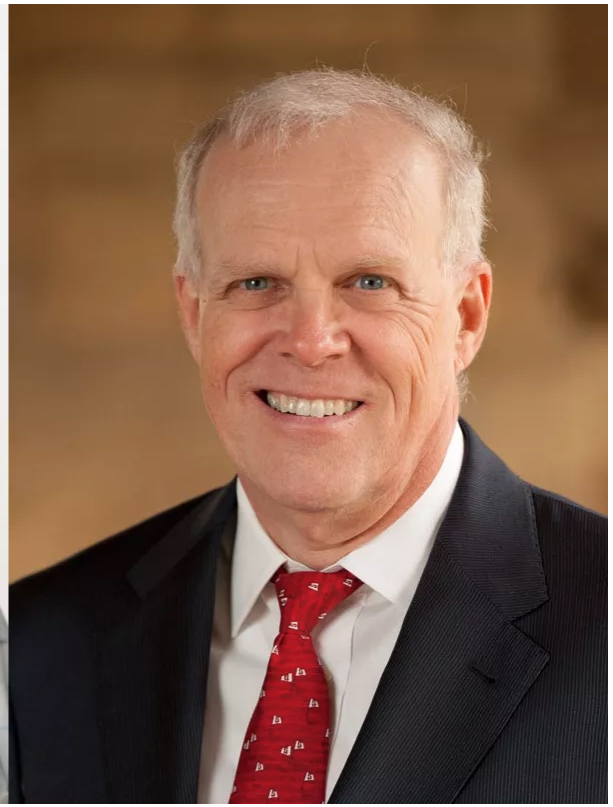
	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer	
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Year developed	1973	1978	1989	1987	1991
Number of instructions	208	303	235	69	94
Instruction size (bytes)	2–6	2–57	1–11	4	4
Addressing modes	4	22	11	1	1
Number of general-purpose registers	16	16	8	40 - 520	32
Control memory size (kbits)	420	480	246	—	—
Cache size (kB)	64	64	8	32	128

Risultati più importanti nell'organizzazione dei processori

- Differenza tra architettura e organizzazione
- Gerarchia di memoria
- Pipeline
- Processori multipli
- Unità di controllo microprogrammata
- **RISC**

Turing award per i RISC

David Patterson e John Hennessy hanno vinto il 2017 ACM Turing Award per l'invenzione delle architetture RISC



CISC e RISC

Due approcci alle architetture:

CISC: Complex Instruction Set Computer

RISC: Reduced Instruction Set Computer

Il tempo di esecuzione T_E di un programma è:

$$T_E = N_I \times C_I \times T_C$$

(NI: num. istruzioni, CI: num. medio di clock per istruzione; TC: periodo clock)

L'obiettivo è ridurre T_E:

- CISC cercando di ridurre N_I
- RISC cercando di ridurre C_I

L'origine dei CISC

- **Linguaggi ad alto livello** (high-level languages, HLLs)
  - Permettono di esprimere algoritmi più in modo conciso
  - Lasciano al compilatore la gestione di dettagli non rilevanti nel design di un algoritmo.
  - Supporta lo sviluppo di programmi strutturali/ad oggetti/funzionali
- **Gap semantico**
  - Differenza nelle operazioni fornite dai HLL e dalle architetture
- **CISC**
  - Chiude il gap semantico usando istruzioni macchina molto complesse

CISC

- Semplifica il lavoro del compilatore
  - Minor gap semantico
- Produce programmi più corti.
  - Minor numero di byte da leggere
- Le istruzioni sono eseguite efficientemente tramite il microcodice
  - Microcodice: l'unità di controllo decompone ogni istruzione in sotto parti.
- Le istruzioni CISC permettono di supportare HLL ancora più complicati e sofisticati.
  - Maggior opportunità per il compilatore

Critica al CISC

- Le architetture CISC punta a ridurre il numero di istruzioni.
- Migliori HLL si ottengono sviluppando architetture che riducono il tempo di esecuzione dei punti critici di tipici programmi HLL.

	Dynamic Occurrence		Machine-Instruction Weighted		Memory-Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

Implicazioni

Vari lavori di ricerca (anni '80) hanno evidenziato che un tipico programma HLL:

1. Effettua molti assegnamenti e tende a riutilizzare dati;

2. Effettua molti branch;

3. Un compilatore riesce ad ottimizzare più facilmente istruzioni corte e di durata fissa, che poche istruzioni complesse.

Nascita del RISC

Le architetture dovrebbero quindi:

- Usare un largo numero di registri
- Avere pipeline più attente alla gestione dei branch
- Usare istruzioni brevi e con un costo facilmente predicibile

Questi sono i principi base del **RISC**!

Caratteristiche del RISC

Le principali caratteristiche del RISC sono:

1. Istruzioni che richiedono 1 ciclo macchina e lunghezza fissa (~ 1 word).
2. Le operazioni lavorano solo su registri e solo semplici operazioni di lettura/scrittura in memoria.
3. Molti registri.
4. Poche e semplici tecniche di indirizzamento.
5. Codifiche semplici delle istruzioni.

Confronto RISC vs CISC

caratteristiche	RISC	CISC
Numero di istruzioni	< 100	> 200
Numero di modi di indirizzamento	1–2	≥ 5
Forme diverse di organizzazione interna delle istruzioni	1–2	> 4
Numero medio di periodi di *clock* per istruzione	~ 1	3–20
Istruzioni che accedono alla memoria	solo LOAD e STORE	quasi tutte
Numero di registri di CPU	≥ 32	8–16
Modulo di controllo	*hardwired*	microprogrammato

Pipeline nel RISC

- Le istruzioni RISC richiedono pipeline più corte e con minor possibilità di stallo.
  - I: Instruction fetch
  - E: execute
  - D: memory access (solo per load e store)

- La semplicità delle istruzioni (stessa lunghezza, durata) permette di usare tecniche avanzate nel compilatore per diminuire gli stalli

Loop unrolling

Loop unrolling: si prende un ciclo e si "srotola"

```
do i=2, n-1
     a[i] = a[i] + a[i-1] * a[i+l]
end do
```

(a) original loop

```
do i=2, n-2, 2
     a[i] = a[i] + a[i-1] * a[i+i]
     a[i+l] = a[i+l] + a[i] * a[i+2]
end do

if (mod(n-2,2) = i) then
     a[n-1] = a[n-1] + a[n-2] * a[n]
end if
```

(b) loop unrolled twice

Loop unrolling (2)

Vantaggi:

- Riduce l'overhead del loop
- Migliora le prestazioni della pipeline: meno branch da eseguire
- Sfrutta la località con registri, data cache e TLB

Svantaggi:

- Aumenta la lunghezza del programma.

Conviene per loop corti.

Esempio di loop unrolling

```
1   int f2(int a){
2   int c = 1;
3   for(int i=1; i<10; i++){
4   c = c+(i>>a);
5   }
6   return c;
7   }
8
```

ARM GCC 8.5.0 (linux) -O1

```
1   f2:
2           mov     r3, #1
3           mov     r2, r3
4   .L2:
5           add     r2, r2, r3, asr r0
6           add     r3, r3, #1
7           cmp     r3, #10
8           bne     .L2
9           mov     r0, r2
10          bx      lr
```

ARM GCC 8.5.0 (linux) (Editor #1)

ARM GCC 8.5.0 (linux) -O3

```
1   f2:
2           mov     r2, #1
3           push    {r4, r5, r6, r7, lr}
4           mov     r7, #2
5           mov     r3, #3
6           mov     r6, #4
7           mov     r5, #5
8           mov     r4, #6
9           mov     lr, #7
10          mov     ip, #8
11          mov     r1, #9
12          asr     r2, r2, r0
13          add     r2, r2, #1
14          add     r2, r2, r7, asr r0
15          add     r3, r2, r3, asr r0
16          add     r3, r3, r6, asr r0
17          add     r3, r3, r5, asr r0
18          add     r3, r3, r4, asr r0
19          add     r3, r3, lr, asr r0
20          add     r3, r3, ip, asr r0
21          add     r0, r3, r1, asr r0
22          pop     {r4, r5, r6, r7, pc}
```

Esempi di processori CISC

- **CISC:**
  - Digital: VAX, PDP-11,
  - Intel: x86 (fino al Pentium),
  - Motorola: 68000.
- **CISC con conversione interna a RISC:**
  - Intel: Pentium II/III/4, Core 2, Atom, Core i7
  - AMD: Athlon, K6, K8.

Esempi di processori RISC

RISC:

- desktop e server:
  - Silicon Graphics: MIPS,
  - Sun Microsystems: SPARC,
  - Digital: Alpha,
  - Hewlett Packard: PA-RISC,
  - IBM e Motorola: PowerPC,
  - Intel: i860, i960.
- embedded:
  - ARM: CORTEX
  - Apple A5, A6
  - Hitachi: SuperH,
  - Mitsubishi: M32R,
  - Silicon Graphics: MIPS16.

x86: CISC o RISC?

- L'architettura x86 è di tipo CISC, ma all'interno dei moderni processori x86 le unità di calcolo eseguono istruzioni RISC
- Le istruzioni di macchina CISC del programma da eseguire vengono convertite dall'hardware in sequenze di istruzioni RISC (chiamate μops dall'Intel e R-ops dall'AMD); sono queste ultime ad essere effettivamente inviate ai pipeline.