CPU Families

Nicola Bombieri
Franco Fummi
Davide Quaglia

University of Verona
Dept. Computer Science
Italy

Contents

History and evolution of CISC Architecture

- The number of instructions increased from 45 on the Intel 4004, to 246 on the 8085, to well over 20.000 variations on the 8086 microprocessor.
- The CISC term came because of the number and complexity of instructions. The 16-bit microprocessor also provided more internal register storage space than the 8-bit microprocessor. The additional register allowed software to be written more efficiently. The 16-bit microprocessor evolved mainly because of the need or larger memory systems.
- The popularity if Intel family was ensured in 1981, when IBM decided to use the 8088 in its personal computer. Application such as spreadsheets, word processors, spelling checkers , were memory intensive and required more than 64K Bytes. The 8088 and 8086 provided 1MegaBytes of memory that was proved limiting for large databases an other applications. This led Intel to introduce the 80286 microprocessor in 1983.

Intel 80286

- Equipped with four independently working units :
  - Bus Unit:
    - Executes all Bus operations of the CPU;
    - When not in use it tries to acquire 6 byte of instructions.
  - Instruction Unit:
    - Decodes the next instruction in advance, containing a maximum of three instructions. Thus the CPU rarely remains inactive, waiting for the next instruction. Increasing performance.

Intel 80286

- Execution Unit:
  - Executes the instructions decoded from the instruction unit.
- Address Unit:
  - Executes all the address calculations

Intel 80386

- Equipped with three principal functional units:
  - BUS Interface unity
  - Central Process Unity (CPU) with 2 levels:
    - Executions Unity:
      - ALU;
      - 8 general purpose 32 bit registers;
      - One 64 Bit Shift register (for multiplication ...)
    - Control Unit for instructions and decoding;
  - Memory Management Unit (MMU) with 2 levels:
    - Segmentation Control
    - Page Control (for 4K byte pages)

Intel 80386/486

- Equipped with four principal functional units:
  - BUS Interface Unit;
  - Central Process Unity (CPU) with 3 levels:
    - Executions Unity: (like 80386)
    - Control Unit of instructions and decoding;
    - Floating Point Unity and registers;
  - Memory Management Unit (MMU) with two levels :
    - Segmentation Control
    - Page Control (for 4K byte pages)
  - Unified Cache 8K byte (for both data and instructions)

Intel 80386

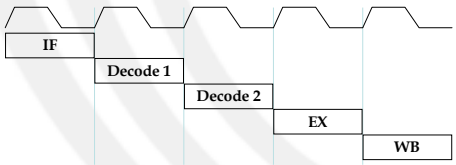
- Clock Frequency :
  - having different version : 16, 20, 25 and 33 MHz;
- Bus Amplitude (Address and data):
  - Data Bus :
    - SX: 16 bit
    - DX: 32 bit
  - Address Bus :
    - SX: 24 bit (16 MB)
    - DX: 32 bit (4 GB)

Intel 80486

- Clock Frequency :
  - having different version : 16, 20, 25, 33 and 50 MHz (improved internal clock in the DX2);
- Bus width (Addresses and data):
  - Data Bus:
    - SX: 16 bit ; DX: 32 bit
  - Address Bus:
    - SX: 24 bit (16 MB) ; DX: 32 bit (4 GB)

Intel 80486

- *Pipelining* Support
  - *single_clock,* 5 periods



Intel Pentium (MMX)

- Features:
  - Clock Frequency: 75-200MHz
  - Internal floating-point co-processor to handle floating-point data at five times the speed of the 486.

Intel Pentium (MMX)
Superscalar Technology

- Superscalar architecture:
  - Three execution units: One executes floating-point instructions, and the other two (U-pipe and V-pipe) execute integer instructions. Thus it is possible to execute three instructions simultaneously (each Clock Cycle), as each instruction is independent of the others.
  - Floating-point unit is also used for MMX instructions, so Pentium MMX can execute two integers and one MMX instruction simultaneously.
  - Software written (i.e., compiler optimized) to take advantage of this superscalar feature can result in up to 40% execution speed improvement.

Intel Pentium (MMX) Branch Prediction

– Branch prediction Unit:
  - The Pentium microprocessor uses branch prediction logic to reduce the time required for a branch caused by internal delays. These delays are minimized because when a branch instruction (short or near only) is encountered, the microprocessor begins pre-fetch instruction at the branch address. The instructions are loaded into the instruction cache, so when the branch occurs, the instructions are present and allow the branch to execute in one clocking period. If for any reason the branch prediction logic errs, the branch requires an extra three clocking periods to execute. In most cases, the branch prediction is correct and no delay ensues.

Intel Pentium (MMX) Cache

– Cache Structure: separate cache for data and instructions:
  - The cache in the Pentium is no longer the same as in the 80486 microprocessor. The Pentium contains two 8KB cache memories instead of one as in the 80486. There is an 8KB data cache and an 8K-byte instruction cache. The instruction cache stores only instructions, while the data cache stores data used by instructions.
  - In the 80486 with its unified cache, a program that was data-intensive, quickly filled the cache, allowing little room for instructions. This slowed the execution speed of the 80486 microprocessor. In the Pentium, this cannot occur because of the separate instruction cache.

Intel Pentium (MMX) MMX instructions

– The MMX (Multi Media eXtensions) technology:
  - adds 57 new instructions to the instruction set of the Pentium. The MMX also has new general purpose instructions which are designed for applications such as motion video, combined graphics with video, image synthesis, speech synthesis and compression, telephony, video conferencing, and 3D graphics.

Intel P6 (1995)

- Introduced in 1995 with PentiumPro.
- Last implementation Pentium IV microprocessor
  - Cache L2 512 KB (half-speed) \ 256 KB (full speed)
  - Cache L1 32 KB (16 KB cache L1 data + 16 KB cache L1 Instructions)
  - Three separate instruction decoders
  - Out of order Execution
  - SIMD (Single Instruction Multiple Data) Instructions
    - MMX
    - SSE (Streaming SIMD Extensions)
  - Bus frequency 100/133 Mhz

From 8086 to Pentium III

Table 2-2. Key Features of Previous Generations of IA-32 Processors

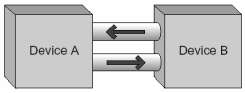
Intel Processor	Date Intro-duced	Max. Clock Frequency at Intro-duction	Tran-sistors	Register Sizes[1]	Ext. Data Bus Size[2]	Max. Extern. Addr. Space	Caches
8086	1978	8 MHz	29 K	16 GP	16	1 MB	None
Intel 286	1982	12.5 MHz	134 K	16 GP	16	16 MB	Note 3
Intel386 DX Processor	1985	20 MHz	275 K	32 GP	32	4 GB	Note 3
Intel486 DX Processor	1989	25 MHz	1.2 M	32 GP 80 FPU	32	4 GB	L1: 8 KB
Pentium Processor	1993	60 MHz	3.1 M	32 GP 80 FPU	64	4 GB	L1:16 KB
Pentium Pro Processor	1995	200 MHz	5.5 M	32 GP 80 FPU	64	64 GB	L1: 16 KB L2: 256 KB or 512 KB
Pentium II Processor	1997	266 MHz	7 M	32 GP 80 FPU 64 MMX	64	64 GB	L1: 32 KB L2: 256 KB or 512 KB
Pentium III Processor	1999	500 MHz	8.2 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32 KB L2: 512 KB
Pentium III and Pentium III Xeon Processors	1999	700 MHz	28 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32 KB L2: 256 KB

Hyper-Threading

- One physical microprocessor contains two distinct logic processors
- Replicated Resources
  – 8 Generic registers, control registers, state registers, pointer to instructions, a mapping table of the registers, return stack predictor
- Partition Resources
  – µops queues after execution trace cache, reorder buffer and load and store buffers.
- Shared Resources
  – They represent the main part of resources in a physic processor: Cache and all executions units. Few shared resources like DTLB (Data Translation Lookaside Buffer), comes with an identification bit.

PCI Express BUS Technology

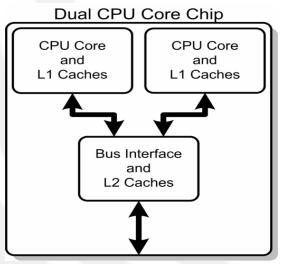
- High-performance interconnect that gives:
  - more bandwidth (High speed equals to 2.5 GB per second)
  - with Low voltage equates to a *differential signaling* environment of between 0.8 and 1.2 volts
  - using approximately one tenth the number of pins.
- Uses a protocol that allows devices to communicate simultaneously by implementing *dual uni-directional paths* between two devices:



The Future of Microprocessor

- Pentium 4 & AMD Opteron Dual-Core briefly explained:
  - many microprocessor will communicate directly with each other, allowing parallel processing without any change to the instruction set or program.
- Currently the superscalar Technology uses many units inside the same microprocessor, but they all share the same register set.
  - This new Technology: Dual core, contain many microprocessor each with its own register set that is linked with the other microprocessor register set.

Pentium & AMD Dual-Core



Pentium & AMD Dual-Core

The main challenge for CPU designer today

- The main challenge for the CPU designer today is the average memory latency the CPU sees.
  - A CPU 3.6 GHz with DDR-400 runs no less than 18 times faster than the base clock of the RAM (200 MHz).
- Every cycle the memory is being accessed, a minimum of 18 cycles pass on the CPU.
  - At the same time, it takes several cycles to even send a request, and it takes a few cycles to send a request back.
  - The result is that wait times of 200 to 300 cycles are not uncommon on the recent CPU's. The goal of CPU cache is to avoid accessing RAM.

The future of Intel Multi Core

- **Yorkfield e Hapertown with 8 Core** .....

Category	Code Name	Core	Cache	Market
Desktop	Wolfdale	Dual core, single die	3 MB shared	2008
Desktop	Ridgefield	Dual core single die	6 MB shared	2008
Desktop	Yorkfield	8 core multi-die	12 MB shared	2008+
Desktop	Bloomfield	Quad core, single die	-	2008+
Desktop/Mobile	Perryville	Single core	2 MB	2008
Mobile	Penryn	Dual core single die	3 MB, 6 MB shared	2008
Mobile	Silverthorne	-	-	2008+
Enterprise	Hapertown	8 core multi-die	12 MB shared	2008

- BUT…….. *The approach of multi core involve some disadvantage. For example the programs have to be optimized to utilize properly the new characteristics of these processors , **otherwise they will use only one of the many cores** , (in case of 4 and 8 cores), and the others will remain not utilized. At the moment there are only very few programs optimized for this new technology.*

Bibliography

- Bibliography:
  - The Intel microprocessors – Architecture, programming and interfacing. Pearson Education Sixt editions. Barry B. Brey
  - http://en.wikipedia.org
  - http://www.anandtech.com/
  - http://www.intel.com
  - http://www.amd.com

Le macchine RISC

- RISC: Reduced Instruction Set Computer
  - Alpha 21964 Digital
  - PowerPC601 IBM/Motorola/Apple
- CISC: Complex Instruction Set Computer
  - 360 IBM (1964)
  - VAX DEC (1978)
  - 80x86 Intel (1982 - 1994)
  - 680x0 Motorola (1979 - 1994)

Le macchine RISC

- Evoluzione dei calcolatori:
  - Esempi come ENIAC, 7094 IBM e CDC 6600
    - hanno poche istruzioni e uno o due modi di indirizzamento;
  - Nel 1964 IBM produce il 360 (mainframe) con una unità microprogrammata (in ROM) che supporta un vasto insieme di microistruzioni;

Le macchine RISC

- Evoluzione dei calcolatori:
  - Dopo pochi anni anche i minicomputer (es: VAX) arrivarono ad avere oltre le 200 istruzioni e 12 modalità di indirizzamento tutti implementati in un microprogramma su ROM;
  - I microprocessori, partiti con una architettura minima, arrivano quasi a superare la complessità dei minicalcolatori e dei mainframe;

Le macchine RISC

- L'evolvere della complessità è incoraggiato dall'uso sempre più elevato dei *linguaggi ad alto livello*;
- L'enorme distanza tra la semantica tra i linguaggi ad alto livello (costrutti come IF, WHILE e CASE) e quella dei "linguaggi macchina" (istruzioni tipo MOVE, ADD e JMP) rende difficile la scrittura dei compilatori.
- La soluzione proposta è *elevare il livello del "linguaggio macchina"*.

Le macchine RISC

- Altro fattore che incoraggia l'espansione delle macchine CISC è la lentezza della memoria rispetto alla CPU (fattore 10 tra ROM e RAM) quindi *ancora più complessità nel microprogramma*.
- Nel 1970 si ha una inversione di tendenza poiché:
  - RAM a semiconduttore più veloci;
  - Microcodice troppo complesso;

Le macchine RISC

- Dal 1970 al 1985 si esamina il reale contenuto, in termini di istruzioni, dei programmi reali:
  - Knuth 1971: programmi FORTRAN;
  - Wortman 1972: programmi di sistema in XPL (simile al PL/1);
  - Tanenbaum 1978: sistema operativo simile in SAL (simile al PASCAL);
  - Patterson 1982: programmi di sistema in C e PASCAL;

Istruzione	SAL	XPL	Fortran	C	Pascal	Media
Assegnazioni	47	55	51	38	45	47
IF	17	17	10	43	29	23
Chiamate	25	17	10	43	29	23
Cicli	6	5	9	3	5	6
GOTO	0	1	9	3	0	3
Altri	5	5	16	1	6	7

85%

Le macchine RISC

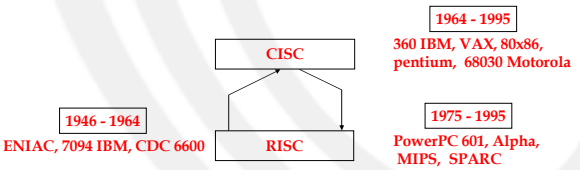
- La conclusione degli studi è la seguente:
  - Teoricamente si possono scrivere programmi molto complessi
  - *Praticamente i programmi scritti sono costituiti da istruzioni semplici*
    - 95% degli assegnamenti coinvolge meno di due operatori (es. var:=valore, var:=val1+val2, var:=val1-val2[i]);
    - 85% delle procedure ha meno di 5 variabili locali;
    - 82% delle procedure ha meno di 5 parametri;

Le macchine RISC

- Quanto più il "linguaggio macchina" diventa esteso e complesso tanto più diventa enorme e lento il microprogramma;
  - più istruzioni significa più tempo per la decodifica dei codici operativi;
  - più modi di indirizzamento implicano chiamate a microprocedure per l'analisi delle modalità.
- Rallentamento non è giustificato poiché:
  - le istruzioni complesse e i vari tipi di modalità di indirizzamento nella pratica non sono quasi mai usate!!

Le macchine RISC

- Sono calcolatori con un piccolo numero di istruzioni semplici che possono essere eseguite con pochissime microistruzioni;
  - il guadagno di un fattore 10 per i casi più comuni



Le macchine RISC

- Radin [IBM] (1975-1982): 801
  - ispiratore del PC/RT IBM
- Patterson e Séquin (1981-1982): RISC I (II)
  - ispiratore della SPARC (Sun Microsystem)
- Hennessy (1984): MIPS
  - ispiratore del MIPS (Computer System) [DEC]

	IBM 370/168	VAX 11/780	IMB 801	RISC I	MIPS
Anno	1973	1978	1980	1981	1983
Istruzioni	208	303	120	3	55
Dim. microcodice	54 k	61 k	0	0	0
Dim. istruzioni	2 - 6	2 - 57	4	4	4
Indirizzamenti	R-R; R-M M-M	R-R; R-M M-M	R-R	R-R	R-R

Le macchine RISC

- Differenze tra RISC e CISC

	RISC	CISC
1	istruzioni semplici da 1 solo ciclo	istruzioni complesse su più cicli
2	solo LOAD e STORE fanno riferimento alla memoria	ogni istruzione può fare riferimento alla memoria
3	PIPELINE	nessuna o poca PIPELINE
4	istruzioni eseguite hardware	istruzioni interpretate (microcod.)
5	istruzioni di lunghezza fissa	istruzioni di lunghezza variabile
6	poche istruzioni e modalità	molte istruzioni e modalità
7	complessità è nel compilatore	complessità è nel microcodice
8	ampio insieme di registri	ridotto insieme di registri

Le macchine RISC

- Punto 1:
  - le operazioni avvengono tra operandi presenti nei registri.
    - le RISC non hanno istruzioni per la moltiplicazione e divisione; quest'ultime sono gestite da procedure di libreria.
    - le operazioni in virgola mobile sono eseguite da un co-processore o da una unità apposita.

Le macchine RISC

- Punto 2:
  - i riferimenti alla memoria richiedono più di un ciclo di clock quindi non esistono istruzioni ordinarie che hanno questa caratteristica; nelle RISC sono introdotte apposite istruzioni di LOAD e STORE.
- Punto 3:
  - le istruzioni sono eseguite direttamente dall'hardware.
    - a vantaggio delle macchine CISC è il risparmio di memoria (meno operazioni ma complesse).

Le macchine RISC

- Punto 4:
  - le istruzioni di lunghezza variabile necessitano di un microprogramma che le spinga fuori dalla coda di prelevamento e ne faccia l'analisi; non va bene per le macchine RISC che hanno, quindi, istruzioni di lunghezza fissa.
- Punto 5:
  - il blocco di decodifica del codice operativo aumenta esponenzialmente con il numero delle istruzioni da decodificare: consumo di area;
  - più modalità di indirizzamento riducono la velocità e aumentano la complessità del chip.

Le macchine RISC

- Punto 6:
  - i compilatori per macchine RISC sono complessi:
    - ottimizzazione nell'uso dei registri;
    - non sono presenti istruzioni ordinarie che fanno diretto riferimento alla memoria;
  - i compilatori per macchine CISC sono semplici poiché le istruzioni complesse (in operazione e modo di indirizzamento) riducono il divario semantico.

Le macchine RISC

- Punto 7:
  - la totale mancanza di microcodice nelle RISC mette a loro disposizione molta area; quest'ultima viene utilizzata per implementare registri riducendo, così, il numero di operazioni di LOAD e STORE.

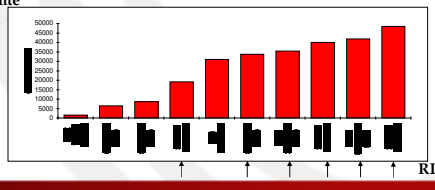
Controversia RISC - CISC

- Quale macchina è "più veloce" nell'esecuzione di programmi scritti in un linguaggio ad alto livello?
  - servono dei benchmark:
    - in che linguaggio devono essere scritti?
      - » le macchine RISC sono ottime per le *chiamate a procedure* ma pessime per i salti;
    - devono includere o no le operazioni di I/O?
      - » le macchine RISC hanno, solitamente, un I/O rudimentale;

Controversia RISC - CISC

- Per confrontare due macchine CISC e RISC si usano (tipicamente) dei benchmark che non fanno riferimento alle operazioni di I/O: misurano le prestazioni della CPU e del compilatore:
  - Whetstones/sec :
    - » numero medio di operazioni svolte in virgola mobile;
  - Dhrystone/sec :
    - » numero medio di operazioni svolte su interi ;

usato tipicamente per i confronti



Controversia RISC - CISC

- Anche se i risultati dei confronti sono a favore delle macchine RISC non và dimenticata la questione relativa alla COMPATIBILITA' che grava pesantemente sulle macchine CISC.
  - Quanto è ampio il vantaggio tratto dall'avere una grande quantità di registri?
    - Usando come metrica il *traffico tra CPU e memoria* ed usando vari benchmark Hitchcock e Sprunt (1985) hanno mostrato come una elevata quantità di registri aumenti la prestazione di una macchina

Controversia RISC - CISC

- Quanto sono valide le macchine RISC?
  - Per applicazioni scritte in linguaggi diversi dal C le macchine RISC hanno portato a risultati poco incoraggianti.
  - Le macchine CISC sono adatte per la generazione di famiglie complete di macchine (i modelli piccoli sono microprogrammati mente i mainframe sono cablati); la macchine RISC non si prestano a questo concetto.

Trend CPU (Previsioni di esperti)

- Al mondo non serviranno più di 5 computer
  - 1958, presidente IBM
- 640Kbyte di memoria sono più che sufficienti per qualsiasi scopo attuale e futuro
  - 1980, Guglielmo Cancelli
- I microprocessori Intel non potranno mai superare il muro dei 100Mhz
  - 1991, Franco Fummi

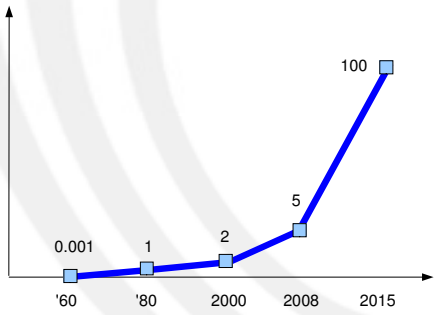
Trend CPU

- Fatturato Top CPU (miliardi):
  - 2000: $32
  - 2001: $23
- Mercato CPU 2001 ($40 miliardi):
  - $ 23: top CPU
  - $ 10: microcontrollori
  - $ 4: DSP
  - $ 3: altro

Trend CPU

- Caratteristiche di innovazione per il mercato:
  - prestazioni
  - affidabilità
  - time-to-market, adattabilità
  - prezzo
- Storia PC 1980-2004
  - CPU???

Trend CPU: nr. CPU/persona



Multiprocessori

Nicola Bombieri
Franco Fummi
Davide Quaglia

University of Verona
Dept. Computer Science
Italy

Contenuto

- Tassonomie
- Calcolatori SIMD
- Calcolatori MIMD
- Rete di interconnessione
- Problemi
  - Cache
  - Sincronizzazione

Tassonomia dei calcolatori

- Classificazione dei calcolatori che analizza le caratteristiche di una macchina attraverso gli *elementi funzionali* che la compongono e le relative interconnessioni.
- Tassonomia di Flynn(1966)

Tassonomia dei calcolatori secondo Flynn

Il parallelismo può apparire a *diversi livelli.*
Nel 1966 Flynn propose un modello di classificazione dei calcolatori basato qui flussi paralleli di istruzioni e di dati:

- Flusso singolo di istruzioni, flusso singolo di dati (SISD, il monoprocessore);
- Flusso singolo di istruzioni, flusso multiplo di dati (SIMD);
- Flusso multiplo di istruzioni, flusso singolo di dati (MISD);
- Flusso multiplo di istruzioni, flusso multiplo di dati (MIMD).

Modello semplice e fornisce una buona prima approssimazione anche se esistono macchine ibride delle precedenti categorie.
Costituisce lo schema più usato per la sua comprensibilità.

Tassonomia dei calcolatori secondo Flynn (cont)

Sorge spontanea la domanda: «singolo o multiplo rispetto a cosa?»

Una macchina che somma dei numeri di 32 bit in un solo ciclo di clock sembra avere un flusso multiplo di dati se viene confrontata con un calcolatore bit-seriale che richiede 32 cicli di clock per effettuare la somma.

Flynn scelse dei calcolatori che a quel tempo erano molto comuni, l'IBM 704 e l'IBM 7090, come modelli delle macchine SISD; oggi, la realizzazione dell'architettura MIPS può essere considerata un punto di riferimento per la classe SISD (Monoprocessori).

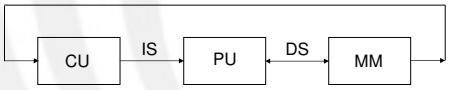
Tassonomia dei calcolatori secondo Flynn (cont)

Gli elementi che costituiscono un calcolatore sono 5:

- *IS - Instruction Stream o Flusso Istruzioni.* E' il flusso di istruzioni da eseguire, ovvero l'insieme di istruzioni che costituisce un programma.
- *DS - Data Stream o Flusso di Dati.* E' il flusso degli operandi e dei risultati sui quali si sta lavorando.
- *CU - Control Unit o Unità di Controllo.* E' l'elemento funzionale che esegue il prelievo e la codifica delle istruzioni.
- *PU - Processing Unit o Unità di Elaborazione.* E' l'unità funzionale, composta da ALU e registri, che esegue le istruzioni.
- *MM - Main Memory o Memoria Principale.* Spazio di memoria nel quale si trovano dati e/o istruzioni.

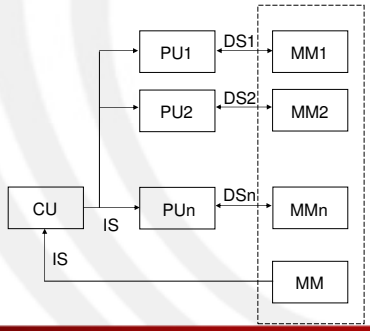
In base al numero degli elementi e alla modalità di connessione degli elementi si differenziano i vari calcolatori.

SISD (Single Instruction, Single Data)



- L'unità di controllo (CU) preleva dalla memoria principale (MM) le istruzioni, mentre l'unità di elaborazione (PU) esegue le istruzioni interagendo con la memoria per modificare i dati.
- Struttura della macchina di Von Neumann, nella quale si ha sempre un singolo programma in esecuzione che si riferisce ad un unico flusso di dati.

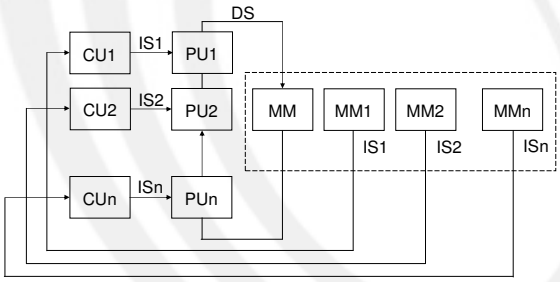
SIMD (Single Instruction, Multiple Data)



SIMD (Single Instruction, Multiple Data)

- L'unità di controllo (CU) preleva dalla memoria principale (MM) l'istruzione, che viene inviata alle diverse unità di elaborazione (PUi) che la eseguono in parallelo su operandi diversi.
- Il flusso di istruzioni è unico: la stessa istruzione viene eseguita in parallelo su più operandi.
- Si ha in esecuzione un singolo programma, ma esiste la possibilità, grazie alla replicazione delle PU, di accedere a diverse zone della memoria.
- Struttura utilizzata dagli Array di Processori.

MISD (Multiple Instruction, Single Data)

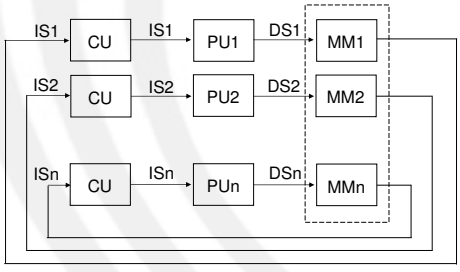


MISD (Multiple Instruction, Single Data)

- Le diverse unità di controllo (CUi) prelevano dalle diverse partizioni di memoria (MMi) le istruzioni (ISi), che vengono inviate alle diverse unità di elaborazione (Pui) che le eseguano in parallelo sullo stesso operando.
- In questa struttura ci sono tanti programmi in esecuzione, che operano su un unico flusso di dati.
- Lo stesso flusso di dati (DS) subisce diverse elaborazioni nelle Pui definite in base alle diverse istruzioni ISi: in pratica esistono tanti flussi di istruzioni che fanno riferimento allo stesso insieme di dati DS.
- Sebbene faccia parte della classificazione di Flynn è difficile immaginare l'architettura MISD:
  - ✓ un flusso singolo di istruzioni è più semplice da gestire di flussi multipli;
  - ✓ flussi multipli di istruzioni si possono immaginare più facilmente se collegati a flussi di dati (MIMD) piuttosto che ad un singolo flusso di dati (MISD).

⇒ Struttura degenere che viene definita per completezza.

MIMD (Multiple Instruction, Multiple Data)



- Si eseguono in parallelo molti programmi che operano ognuno su un diverso flusso di dati

Calcolatori SIMD

Operano su vettori di dati:

- Singola istruzione SIMD di somma di 64 numeri: macchina SID manda 64 flussi di dati a 64 ALU per effettuare le 64 somme nello stesso ciclo di clock.

Caratteristiche:

- Tutte le unità di esecuzione (ognuna con registri di indirizzamento propri) che vanno in parallelo sono *sincronizzate* e rispondono tutte insieme ad una singola istruzione che viene individuata dall'unico registro contatore di programma (PC)
- La programmazione è molto simile a quello delle architetture SISD
- La motivazione originale era la volontà di ammortizzare il costo dell'unità di controllo collegandola a dozzine di unità di esecuzione.

Calcolatori SIMD (cont.)

- Richiedono una *dimensione ridotta della memoria* di programma: *una sola copia del codice* che viene eseguito simultaneamente, mentre un MIMD potrebbe richiedere una copia del codice per ogni processore (Fattore di importanza ridotta al crescere della capacità dei chip di DRAM).
- Utilizzano un misto di istruzioni SISD e SIMD.
- Situazione migliore: istruzioni ad elevato *parallelismo dei dati*. Esempio: gestione di vettori di dati all'interno di *cicli for*.
- Situazione peggiore: istruzioni dove ogni unità di esecuzione deve svolgere una diversa operazione a seconda dei suoi dati. Esempio: gestione di comandi *case* o *switch*. In pratica le unità di esecuzione con dati sbagliati vengono disabilitate.

Struttura dei calcolatori SIMD

- Calcolatore ospitante SISD che esegue le *operazioni sequenziali*, come i salti condizionati o i calcoli di indirizzo.
- Le istruzioni SIMD vengono trasmesse a tutte le unità di esecuzione, ognuna dotata del proprio insieme di registri e della propria memoria.
- Le unità di esecuzione utilizzano una *rete di interconnessione* per potersi scambiare i dati.

Esempio

Somma di 128 000 numeri su di un SIMD con 128 unità di esecuzione (EU):

1. Numeri suddivisi in 128 sottoinsiemi nella memoria locale delle relative EU.
2. Calcolo della somma di ogni sottoinsieme

```
sum = 0;
for (i = 0; i < 1000; i = i + 1) /* ciclo eseguito da tutte le */
                                 /* EU per ogni vettore locale */
     sum = sum + Al[i];          /* somma il vettore locale */
```


Esempio (cont.)

3. Somma dei 128 risultati parziali

```
limit = 128;
half = 128; /* 128 EU nel calcolatore SIMD */
repeat
  half = half/2; /* linea di divisione tra chi spedisce */
                 /* e chi riceve */
  if (Pn >= half && Pn < limit)
      send(Pn % half,sum); /* Pn =# dell'EU
  if (Pn < half)
      sum = sum + receive(); /*limite superiore delle unità */
                             /* che spediscono*/
  limit = half;
until (half == 1); /* uscita con la somma finale */
```


Calcolatori SIMD (cont.)

- Il compromesso durante la progettazione di macchine SIMD consiste nel *bilanciamento* tra le prestazioni dei singolo processore e il numero dei processori.
- Le macchine SIMD presenti sul mercato privilegiano un *alto grado di parallelismo* rispetto alle prestazioni del singolo processore.

Caratteristiche di cinque calcolatori SIMD

Istruzione	Nome	Massimo numero di proc.	Bit per proc.	Freq. di clock del proc. (MHz)	Numero di FPU	Memoria max per sistema (MB)	Banda passante max per sistema (MB/sec)	Anno
U. Illinois	Illiac IV	64	64	5	64	0,125	2 560	1972
ICL	DAP	4 096	1	5	0	2	2 560	1980
Goodyear	MPP	16 384	1	10	0	2	20 480	1982
Thinking Machines	CM-2	65 536	1	7	2048 (opz.)	512	16 384	1987
Maspar	MP-1216	16 384	4	25	0	256 o 1024	23 000	1989

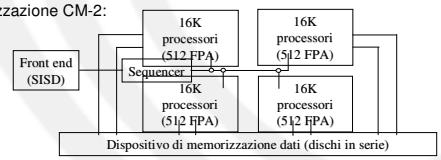
Connection Machine 2 della Thinking Machines

Contiene:
- fino a 65.536 processori (64K), ciascuno dotato di una ALU di un solo bit;
- quattro registri da 1 bit ciascuno;
- 64K bit di memoria (8KB);
- un collegamento alla rete che gli consente di parlare con tutti gli altri processori.
- ad ogni 32 processori può essere aggiunto un acceleratore per i calcoli in virgola mobile (FPA) a 32 bit;
- supporta fino a 512 MB di memoria totale e 2K FPA;
- frequenza di clock 7 MHz.

Ci sono solo quattro tipi di chip:
- fabbricati ad hoc che contengono 16 processori da 1 bit ciascuno (⇒ 64K processori/16 processori per chip = 4K chip)
- normali DRAM (22528)
- interfaccia tra i processori e i FPA (2048)
- FPA (2048).

Connection Machine 2 della Thinking Machines

- Rete d'interconnessione tra processi:
  - ✓dentro il chip da 16 processori: ogni processore ha un collegamento dedicato verso ognuno degli altri processori;
  - ✓interfaccia per la comunicazione con gli altri processori cioè collegamento tra 4K diversi chip: i chip sono collegati attraverso un 12-cubo booleano (4K=2^{12} chip).
- Organizzazione CM-2:



Connection Machine 2 della Thinking Machines

- La CM-2 è composta da una macchina SISD tradizionale (Front end) che esegue il programma.
- Quando il Front end incontra istruzioni SIMD le manda al Sequencer che le trasmette a 64K processori e 2K FPA.
- La CM-2 possiede un canale di I/O per ogni 8K processori (⇒ 8 canali di I/O).

MIPS R2000 vs CM-2

- Somma di due numeri di 32 bit:
  - ✓ CM-2 → 21 µs
  - ✓ R2000 → 0.066 µs (se entrambi gli addendi sono memorizzati in registri), 0.198 µs (se entrambi gli addendi sono in cache)
  - ✓ R2000 effettua una singola addizione a 32 bit da circa 106 (21/0.198) a 318 (21/0.066) volte più velocemente di CM-2
- CM-2 può eseguire 65536 addizioni alla volta
- CM-2 può eseguire da 200 a 600 volte più addizioni al secondo di R2000
  - ✓ (65536*0.066 µs)/(21 µs) ≈ 206
  - ✓ (65536*0.198 µs)/(21 µs) ≈ 618

Calcolatori MIMD

- *Idea base*: collegare fra loro molti calcolatori in modo da crearne uno più potente
  - L'interesse verso i MIMD nasce dalle elevate prestazioni e dal costo ridotto dei microprocessori
  - Chiamiamo *processing unit* (PU) ogni processore del MIMD
- *Scalabilità*: hardware e software progettati in modo da essere venduti con un numero variabile di processori.
- *Flessibilità*: ogni PU può eseguire un programma diverso.
- *Prestazioni*: è ormai assodato che se non si riesce a gestire un carico multiprogrammato con un SISD basato su un solo chip, allora un calcolatore MIMD composto da molti SISD singoli è probabilmente più efficace di un SISD ad alte prestazioni costruito sfruttando una tecnologia più avanzata.
- *Tolleranza ai guasti*: per come sono progettati (scalabilità del software) la funzionalità del sistema non è compromessa dal guasto di un processore: se un processore si guasta in un MIMD con n processori il sistema continua a funzionare con (n-1) processori.

Tendenza per i MIMD

- La miniaturizzazione porta a inserire più PU sullo stesso substrato di semiconduttore dando origine alle seguenti terminologie
  - Multi-core chip
  - System-on-Chip (SoC)
  - Multi-Processor System-on-Chip (MPSoC)

Coordinamento e comunicazione tra PU

- Meccanismi di *sincronizzazione* per accedere a risorse comuni.
- Meccanismi di *spedizione di messaggi* per garantire la comunicazione tra processori.
- I meccanismi sono implementati
  - Attraverso accesso alla memoria
  - Attraverso primitive offerte dal Sistema Operativo

Spazio di indirizzamento

- MIMD con spazio di indirizzamento *unico* condiviso da tutti i processori denominati anche *multiprocessori a memoria condivisa* o *shared memory multiprocessors*.
  - I processori comunicano tra loro attraverso delle variabili condivise in memoria e la gestione *implicita* della comunicazione avviene attraverso operazioni di caricamenti/memorizzazione capaci di accedere a qualsiasi locazione di memoria.
- MIMD con spazi di indirizzamento *multipli* e *privati*.
  - La comunicazione tra processori richiede una gestione esplicita tramite primitive di spedizione e ricezione.

Posizione della memoria fisica

- *Memoria centralizzata*: tempo di accesso ad una locazione fisica di memoria è *uniforme* per tutti i processori.
- *Memoria distribuita*: memoria suddivisa in moduli vicini ai singoli processori ⇒ accesso *non uniforme* alla memoria.
- NOTA BENE
  - I concetti di spazio di indirizzamento (singolo/multiplo) e di posizione fisica della memoria (centralizzata/distribuita) sono *ortogonali* fra loro.
  - In particolare i multiprocessori possono avere spazio di indirizzamento singolo e memoria fisicamente distribuita.

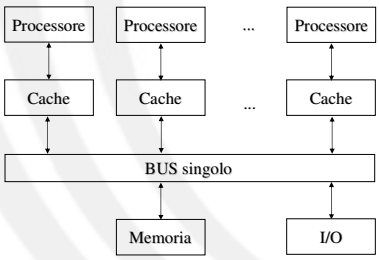
Classificazione UMA/NUMA

- *Multiprocessori UMA* (Uniform Memory Access) o multiprocessori SMP (Symmetric Multiprocessor) cioè macchine che richiedono lo stesso tempo per accedere alla memoria principale a prescindere dal processore che emette la richiesta e dalla parola richiesta.
- *Multiprocessori NUMA* (Non Uniform Memory Access) cioè macchine che richiedono un diverso tempo di accesso a seconda del processore e della parola di memoria richiesta.

Interconnessione tra PU

- Qual è la topologia di interconnessione tra i processori?
  - ✓ A bus singolo
  - ✓ Ad anello
  - ✓ Rete completamente connessa con bus dedicati (*crossbar network*)
  - ✓ Griglia bidimensionale o *mesh*
  - ✓ Rete di collegamento ad *N*-cubo
- La rete di interconnessione sta diventando così complicata che acquisisce funzionamento e problematiche tipiche delle reti "esterne" e si tende a parlare di Network-on-Chip (NoC)
- La topologia di interconnessione può essere diversa per il bus dati e per il bus indirizzi.

Architettura MIMD a bus singolo



MIMD collegati a bus singolo

- Motivazioni dell'architettura a bus singolo:
  - ✓Ogni microprocessore è realizzato su singolo chip ⇒ bus di dimensioni ridotte.
  - ✓La presenza di cache per ogni processore può ridurre il traffico sul bus.
  - ✓Sono stati inventati dei meccanismi per mantenere le cache e la memoria coerenti tra loro nei multiprocessori, proprio come le cache e la memoria vengono tenute coerenti rispetto alle operazioni di I/O.
- Il traffico per processore e la banda passante del bus limitano il numero utile dei processori.
- Le cache replicano i dati sia per ridurre il tempo di latenza per ottenere i dati sia per ridurre il traffico sul bus da e verso la memoria.

Esempi di MIMD commerciali collegati a bus singolo

Casa costruttrice	Nome	Massimo numero di proc.	Bit per proc	Freq. di clock del proc. (MHz)	Numero di FPU	Memoria max per sistema (MB)	Banda passante max per sistema (MB/sec)	Anno
Sequent	Simmetry	30	32	16	30	240	53	1988
Silicon Graphics	4/360	16	32	40	16	512	320	1990
Sun	4/360	4	32	40	4	768	320	1991

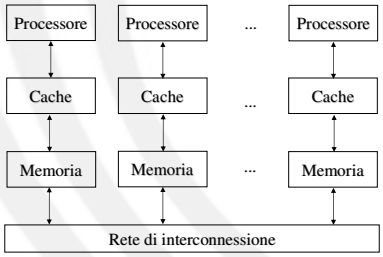
Il numero di FPU indica il numero delle unità dedicate ai calcoli in virgola mobile. Per queste macchine, la banda passante per le comunicazioni corrisponde alla banda passante del bus.

Esempi di MIMD commerciali (1997) collegati a bus singolo

Queste macchine presentano memoria condivisa con tempo di accesso alla memoria uniforme.

	Massimo numero di proc.	Nome del proc.	Freq. di clock del proc. (MHz)	Memoria max per sistema (GB)	Banda passante max per sistema (MB/sec)
Compaq ProLiant 5000	4	Pentium Pro	200	2	540
Digital AlphaServer 8400	12	Alpha 21164	440	28	2150
HP 9000 K460	4	PA-8000	180	4	960
IBM RS/6000 R40	8	PowerPC 604	112	2	1800
SGI Power Challenge	36	MIPS R10000	195	16	1200
Sun Enterprise 6000	30	UltraSPARC 1	167	30	2600

Architettura MIMD a rete di interconnessione



Gestione della coerenza delle cache

- La presenza di processori multipli richiede che diverse copie degli stessi dati siano memorizzate in *più di una cache*.
- In alternativa gli accessi ai dati condivisi dovrebbero aggirare la cache e *andare sempre in memoria* ⇒ soluzione lenta e che richiederebbe una banda passante del bus troppo elevata.
- Possibile uso di *copie multiple* degli stessi dati ⇒ per mantenere la consistenza sono necessari *protocolli per la coerenza delle cache*.

Programmazione dei calcolatori MIMD

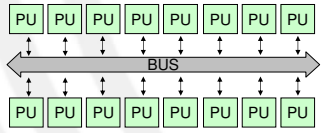
- Problema: solo poche applicazioni tra quelle importanti sono state completamente riscritte in modo da completare i loro compiti *più in fretta* quando sono eseguite da processori paralleli.
- La difficoltà sta nello scrivere *programmi MIMD* che siano veloci al crescere del numero di processori: difficoltà dovuta al carico addizionale richiesto per *gestire la comunicazione tra processori* ⇒ usando *N* processori paralleli, raggiungere un'accelerazione pari a *N* volte diventa sempre meno probabile al crescere di *N*.
- Il programmatore deve avere una *buona conoscenza dell'organizzazione dell'hardware* per scrivere programmi veloci e capaci di essere eseguiti da un numero *variabile* di processori
  - A causa di tale dipendenza dall' hardware, i programmi MIMD non sono portabili semplicemente.
- La parallelizzazione deve tenere conto del dominio applicativo e della semantica dei dati e delle elaborazioni su di essi al fine di una siddivisione efficiente su più PU; in pratica essa non si può demandare ad un compilatore.

SPMD - Single Program Multiple Data o Programma Singolo con Dati Multipli

- Approccio generale: i programmatori di macchine MIMD scrivono un solo programma sorgente e immaginano che questo programma sia eseguito da tutti i processori contemporaneamente.
- La sincronizzazione semplifica la programmazione.
  - A patto di avere memorie indipendenti i processori interagiscono tra loro solo durante la comunicazione.
  - Il processore opera secondo lo stile MIMD mentre non sta comunicando e secondo quello SIMD quando invece comunica.
- L'approccio SPDM offre i vantaggi di programmabilità dei SIMD senza soffrire della loro scarsa flessibilità.

SPMD: esempio

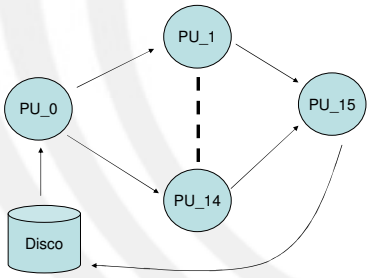
Somma di 128 000 numeri su di un MIMD con un singolo bus con 16 processori e memoria condivisa.



Traccia di soluzione:

1. Numeri suddivisi in 16 sottoinsiemi di 8000 elementi.
2. Calcolo della somma di ogni sottoinsieme
3. Somma dei 16 risultati parziali

SPMD: soluzione A



SPMD: soluzione A (2)

```
main_parallelo(int PU_n)
{
  switch(PU_n) // scelta in base a quale PU sono
  {
    case 0: // emitter
    distribuisci_dati();
    case 15: // collector
    raccogli_dati();
    default:
    elabora();
  }
}
```

SPMD: soluzione A (3)

```
distribuisci_dati()
{
  total_data =leggi_da_disco();
  data[] = suddividi_dati(); // dipende dalla semantica del
                             // problema e dall'architettura
                             // parallela utilizzata
  for(i=1; i<15; i++)
    send(data[i], i); // manda data alla i-esima PU
}

raccogli_dati()
{
   totale = 0;
   for(i=1; i<15; i++)
   {
     receive(tot_parziale, i); // dati dalla i-esima PU
     totale += tot_parziale;
   }
   scrivi_su_disco(totale);
}
```

SPMD: soluzione A (4)

```
elabora()
{
  receive(data, 0); // riceve dati dalla PU n. 0
  tot_parziale = somma(data);
  send(tot_parziale, 15); // manda risultato a PU n. 15
}
```

SPMD: soluzione B

```
main_parallelo(int PU_n)
{
  sum[PU_n] = 0;
  for (i=8000*PU_n; i<8000*(PU_n+1); i=i+1)
  {
    sum[PU_n] = sum[PU_n] + A[i]; /* somma i numeri assegnati */
    notify();
  }

  half =16;     /* 16 processori in un MIMD a bus singolo */
  repeat        /* PU_n numero del processore */
  {
    synch(half); /* attesa che le half somme parziali vengano */
                 /* completate */
    half = half/2; /* linea di separazione, indica chi */
                   /* provvede alla somma */
    if (PU_n<half)
    {
        sum[PU_n] = sum[PU_n]+sum[PU_n+half];
        notify();
    }
  }
  until (half ==1);  // uscita con la somma finale in sum[0]
}
```


SPMD: soluzione B (2)

- Le somme parziali *non vengono spedite o ricevute*; il processore che le utilizza per averne una copia deve solo *caricare* una parola dall'indirizzo corrispondente.
- I processori che collaborano tra loro debbono *sincronizzarsi* per essere sicuri che il risultato sia pronto ⇒ è necessario prevedere una *sincronizzazione* esplicita nei punti chiave del programma.
- I due processori devono sincronizzarsi prima che il processore *consumatore* cerchi di leggere il risultato dalla locazione di memoria che viene scritta dal processore *produttore*, altrimenti il consumatore rischia di leggere il vecchio valore dei dati.
- Questo metodo usa una primitiva di *sincronizzazione alla barriera*, in cui il processore si ferma alla barriera e aspetta finché tutti i processori non lo hanno raggiunto.
- Questa funzione può essere implementata via software con la *primitiva di sincronizzazione* basata su variabili di blocco.
- Nei SIMD non è necessaria una sincronizzazione perché ogni istruzione viene portata avanti di pari passo da tutte le unità ⇒ non c'è bisogno di sincronizzarsi prima di spedire o ricevere un risultato.

Librerie per SPMD

- Nell'esempio in grassetto erano riportate delle chiamate a funzione particolari
- Esse sono fornite da librerie apposite
  - Message-Passing Interface (MPI)
  - Parallel Virtual Machine (PVM)
  - Etc…
- Tali librerie si linkano al programma che viene scritto

Sincronizzazione in SPMD

- Nella soluzione A, la chiamata ad una `receive(data,i)` senza che la PU i-esima abbia fatto una chiamata a `send(data, j)` porta ad una sospensione del processo sul nodo j
  - Queste 2 chiamate sono *sincrone* e il loro uso porta alla *sincronizzazione*
- Nella soluzione B, sono `synch()` e `notify()` che permettono la sincronizzazione tra processi

Sincronizzazione tramite variabili di blocco

Problema: Necessità di coordinare processi che lavorano su un compito comune ⇒ come utilizzare le *variabili di blocco* (note anche come *semafori*) per coordinare e sincronizzare i processi.

L'arbitraggio è semplice per i multiprocessori con un bus singolo, perché c'è un solo percorso che conduce alla memoria: il processore che ottiene il controllo del bus è quello che blocca tutti gli altri fuori dalla memoria.

Soluzione: Il processore e il bus debbono supportare un'*operazione atomica di scambio:* un processore deve poter leggere una locazione di memoria e contemporaneamente assegnarle il valore corrispondente alla situazione di blocco attivo nella stessa operazione di bus, impedendo a tutti gli altri processori e ai dispositivi di I/O di leggere dalla memoria oppure di scriverci finché l'operazione non è conclusa. L'aggettivo *atomica* indica indivisibile.

Procedura per bloccare una variabile tramite un'istruzione atomica di scambio

- Variabile di blocco:
  - ✓ 0 Sbloccata (*unlocked*) VERDE → GO
  - ✓ 1 Bloccata (*locked*) ROSSO → STOP
- Il processore carica la variabile di blocco per verificarne lo stato.
- Il processore continua a leggere e testare la variabile finché il valore indica che è sbloccata (0).
- Il processore si pone in competizione con gli altri processori per cercare di bloccare la variabile utilizzando un'*istruzione atomica di scambio*: il processore legge la variabile e cerca di assegnarle il valore di blocco (1).
- Solo il processore che vince vede la variabile a 0 (sbloccata), mentre i perdenti vedono il valore della variabile a 1 (bloccata) posto dal vincitore.

Procedura per bloccare una variabile tramite un'istruzione atomica di scambio

- I perdenti continuano a scrivere la variabile a 1, ma senza riuscire a cambiare il suo valore.
- Il processore vincente può eseguire il codice che aggiorna i dati condivisi.
- Quando il vincitore esce, assegna il valore 0 alla variabile di blocco lasciando che la competizione tra gli altri processori ricominci.

Passi richiesti per acquistare un blocco