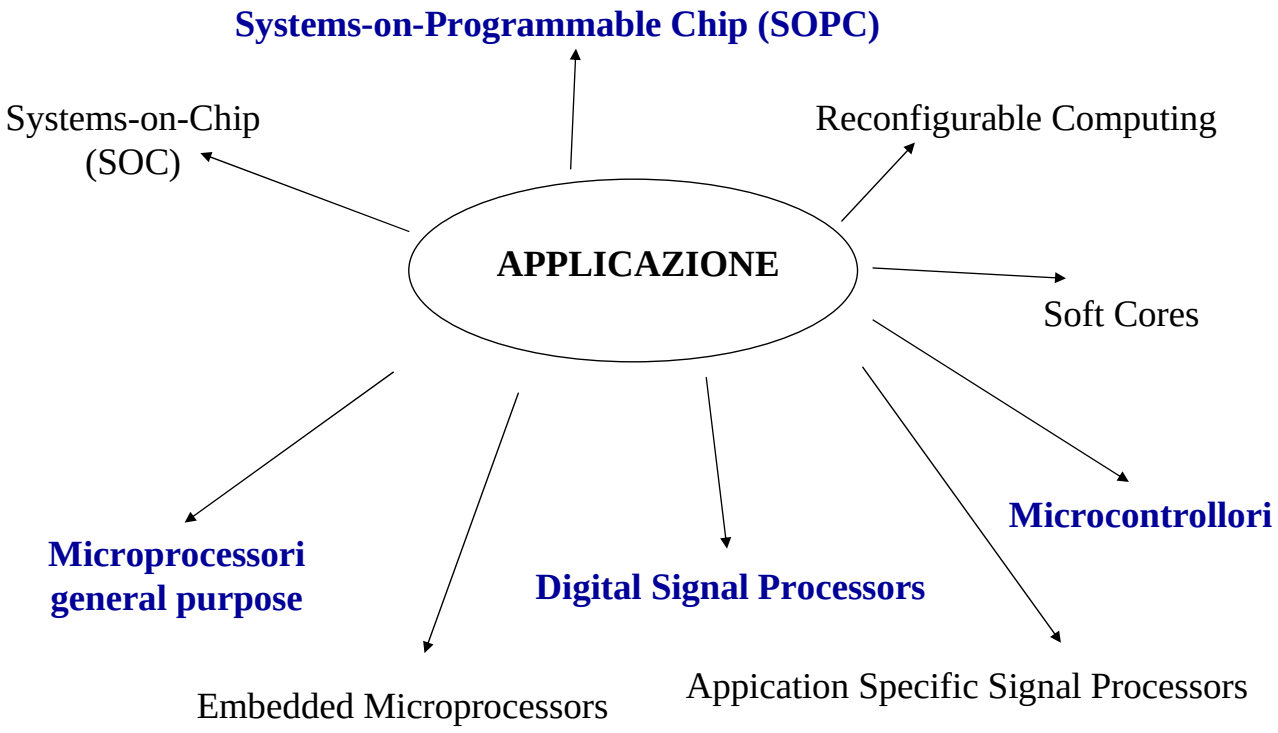
Microprocessori

F.Campi, A.Romani, F.Thei

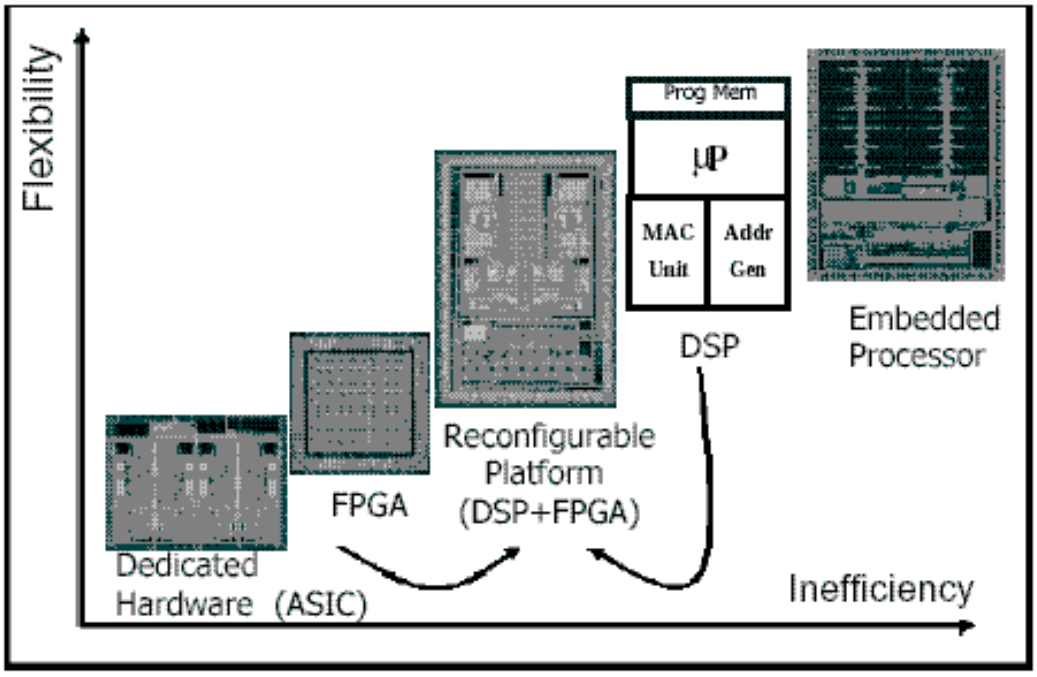
Elettronica dei Sistemi Digitali LA

AA 2009-2010

MACCHINE PROGRAMMABILI - 1



MACCHINE PROGRAMMABILI - 2



APPLICAZIONI PER MICROPROCESSORI

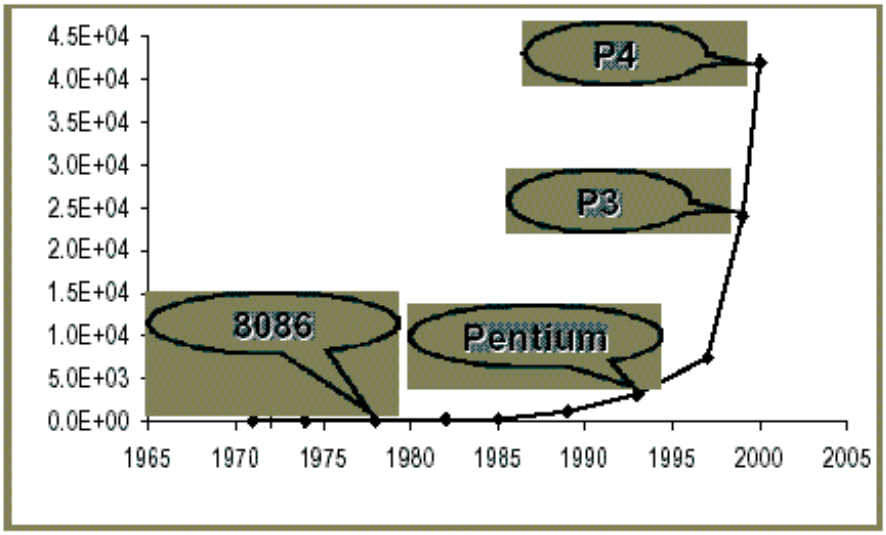
Increasing Cost

- General Purpose - high performance
  - Pentiums, Alpha's, SPARC
  - Used for general purpose software
  - Heavy weight OS - UNIX, NT
  - **Workstations, PC's**
- Embedded processors and processor cores
  - ARM, 486SX, Hitachi SH7000, NEC V800
  - DSP
  - Single program / lightweight, often realtime OS
  - **Cellular phones, consumer electronics (e.g. CD players)**
- Microcontrollers
  - Extremely cost sensitive
  - Small word size - 8 bit common
  - Highest volume processors by far
  - **Automobiles, toasters, thermostats, ...**

Increasing volume

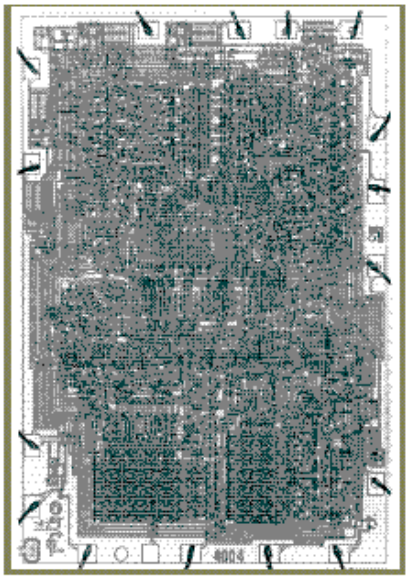
LA LEGGE DI MOORE

"The transistor density available on integrated circuits will double every couple of years"

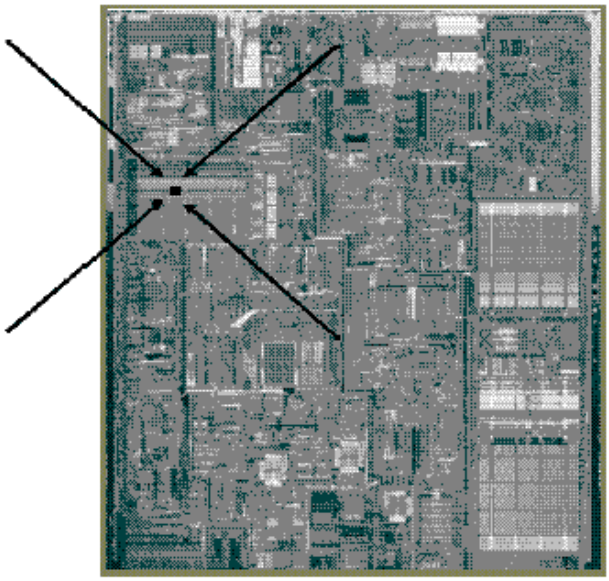


Source: Intel

LA LEGGE DI MOORE – un esempio visivo



Intel 4004 (1971)
2300 transistor / 108 kHz

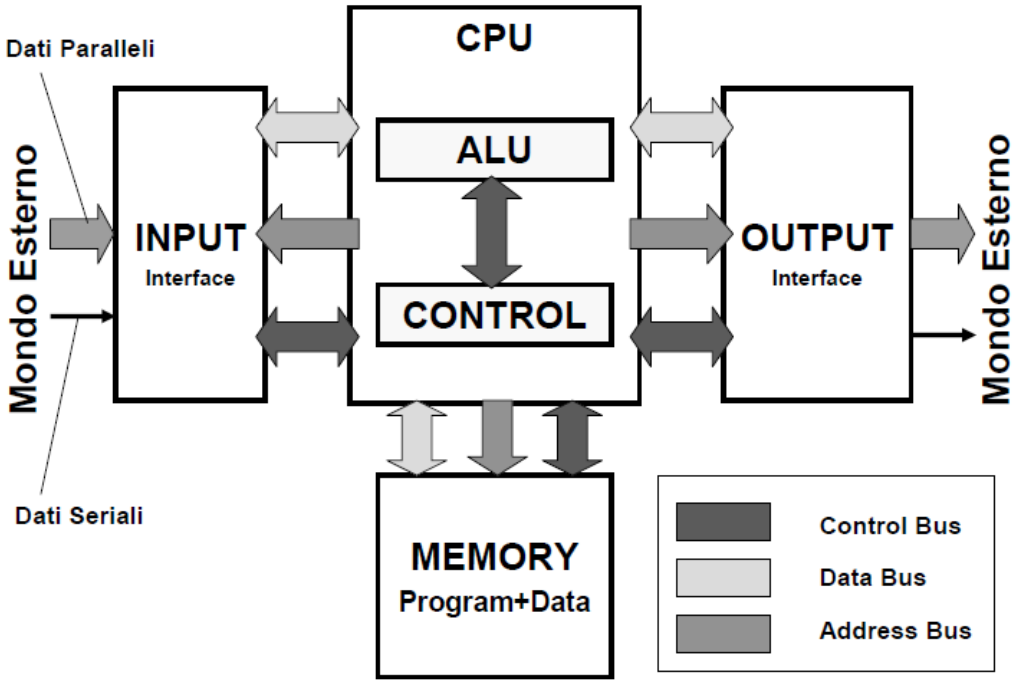


Intel Pentium 4 (2000)
42 M transistor / 1.5 GHz

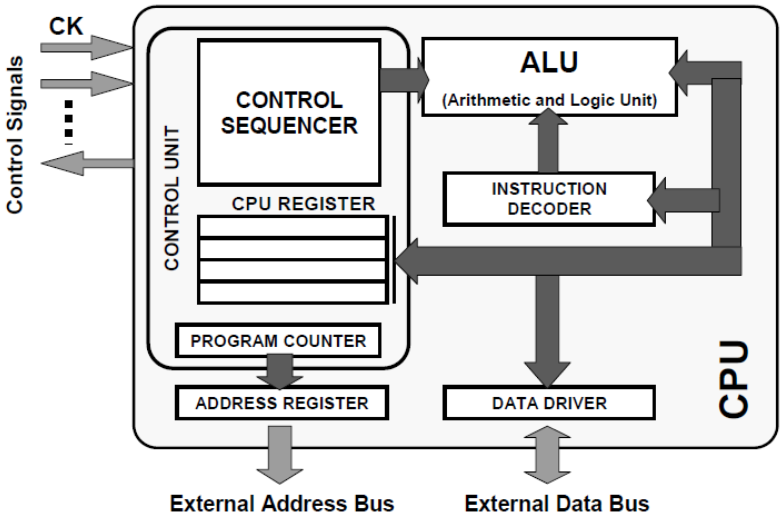
60 years of the transistor: 1947 - 2007

ARCHITETTURA BASE MICROPROCESSORE

Funzionamento base di un microprocessore: preleva un'istruzione, la decodifica, carica gli operandi, esegue l'operazione e salva il risultato.



CPU



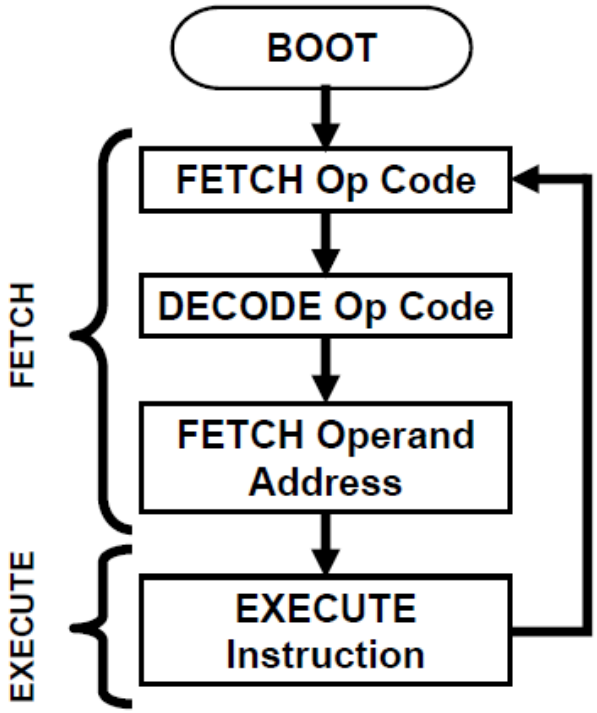
Funzioni base di una CPU:

- Trasferimento dati
- Controllo di flusso
- Operazioni logiche e aritmetiche (ALU)

Ogni CPU ha un array register con almeno:

- Un registro accumulatore (W)
- Il Program Counter (PC)
- L'Instruction Register (IR)
- Lo Stack Pointer (SP)

CPU – esecuzione di un'istruzione

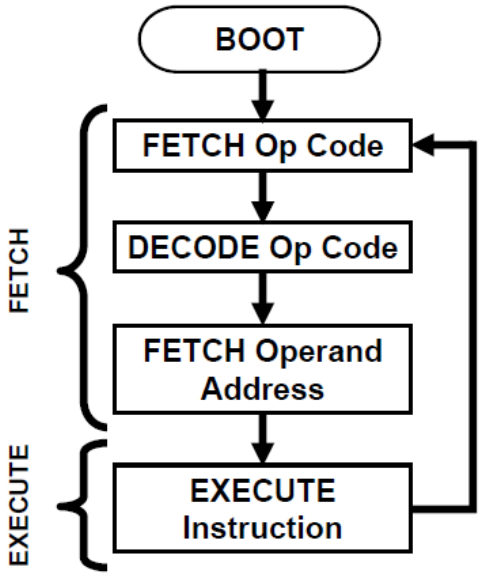


La CPU esegue un'istruzione in due fasi:

Fase di **FETCH**

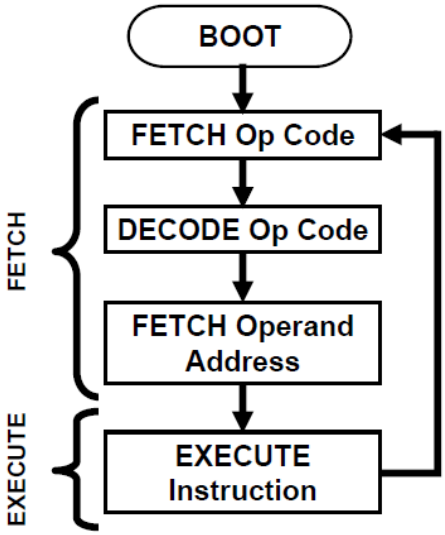
Fase di **EXECUTE**

Fase di FETCH



- La CPU carica sul bus indirizzi l'indirizzo dell'istruzione da eseguire, prelevandolo dal PC;
- Viene letta la locazione di memoria indirizzata dal bus indirizzi tramite i segnali di controllo generati sul control bus;
- Il dato prelevato dalla memoria viene caricato nell' IR;
- Si aggiorna il PC, il quale punterà alla prossima istruzione da eseguire.

Fase di EXECUTE



- Si decodifica l'istruzione presente nell'IR;
- Vengono eseguiti i trasferimenti di dati necessari e le operazioni logiche e/o aritmetiche derivate dalla decodifica dell'op code;
- Il risultato, a seconda del tipo di operazione eseguita, è riscritto in un registro o in una locazione di memoria o su un dispositivo di I/O.

Per essere eseguita, un'istruzione richiede quindi almeno 2 cicli macchina (ovvero 2 accessi in memoria, uno in lettura ed uno in scrittura)

ARCHITETTURE MICROPROCESSORI

L'architettura di un microprocessore riguarda diversi aspetti costitutivi:

- **Set di istruzioni: famiglie CISC e RISC**
- **Organizzazione interna della memoria (es. Von Neumann, Harvard), gestione dei bus (es. AMBA, coreconnect, VCI), unità di calcolo**
- **Implementazione hardware, ovvero lo specifico progetto (es. pipeline, architetture superscalari, VLIW)**

RISC (Reduced Instruction Set Computers)

L' instruction set ideale per una architettura di tipo Risc a 32 bit è stato standardizzato Da J.Hennessy e D.Pattersson attraverso valutazioni QUANTITATIVE.

Si è cercato di individuare quale fosse il set MINIMALE di istruzioni macchina (Assembly) che offrisse le prestazioni migliori SENZA AUSILIO DI MICROCODICE. (Si definisce Microcodice il set di operatori elementari che descrive il comportamento di una determinata istruzione assembly. Nel caso delle macchine Risc non esiste Microcodice ed ad ogni ciclo si esegue una sola istruzione)

Gli i.s. Risc sono più semplici, simmetrici e facilmente modificabili per andare incontro alle esigenze della applicazione specifica. Portano inoltre ad un hardware molto semplificato. In caso di applicazioni general purpose portano però ad una ridotta efficienza, poiché è necessario un numero elevato di istruzioni per eseguire un determinato tipo di calcolo.

Registri:
In assembler si indicano con $0, $1, …. $31
Alcuni registri sono convenzionalmente utilizzati per task specifici (es. $29 stack pointer, $31 return address, $0 zero register)

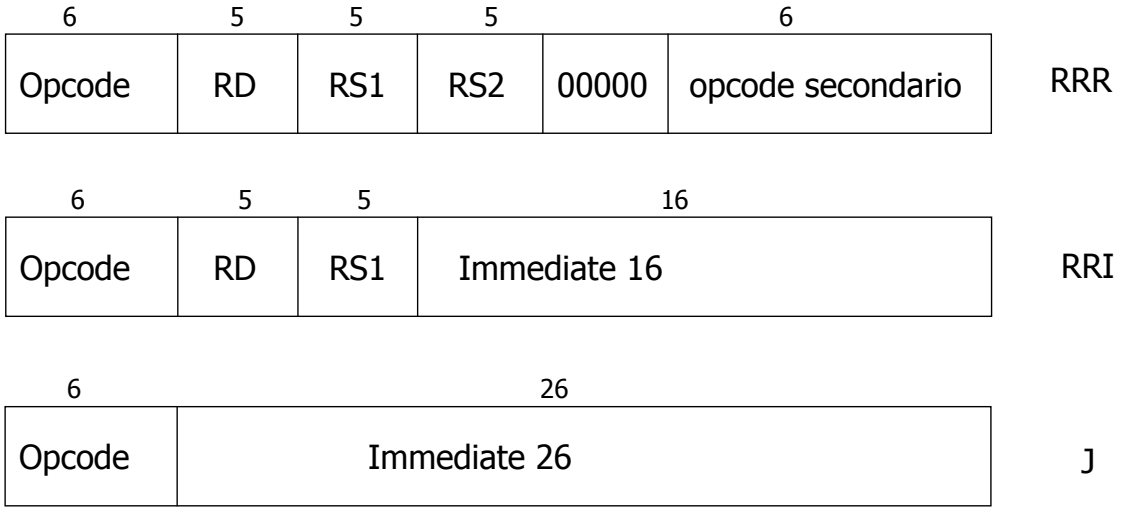
RISC (Reduced Instruction Set Computers)

Progetto nato dall'esigenza di ottimizzare la compilazione automatica, la quale difficilmente prevedeva l'utilizzo di istruzione complesse

- istruzioni in grado di eseguire semplici operazioni in tempi simili
- istruzioni di lunghezza fissa
- ogni istruzione è eseguita in un ciclo di clock
- pochi e semplici metodi di indirizzamento della memoria (solo le istruzioni LOAD e STORE accedono direttamente alla memoria)
- elevato utilizzo dei registri
- per aumentare le prestazioni è necessario eseguire più istruzioni in parallelo → pipeline
- per elaborare più istruzioni ad ogni ciclo di clock si ricorre ad architetture superscalari VLIW

Esempi formato istruzioni RISC

Caso Esemplificativo: ISA RISC standard a 32 bit



Set istruzioni RISC – ISA Standard

Operazioni di Controllo:
`Nop, Li, Lui, Move`
(No operation, Load immediate, load upper immediate, move register to register)

Operazioni Alu:
`Add,Addu, addi, addiu ,sub, subi, and, or, not, xor`
`Seq,sneq,sgt,sge,slt,sle`

Operazioni Shift:
`Sll,srl,sra`

Operazioni Per la gestione della Memoria
`Lw,lh,lhu,lb,lbu`
`Sw,sh,sb`

Operazioni per il controllo del flusso di programmazione
`J,jr,jal,jlr`
`Beqz,bneqz`

Set istruzioni RISC – ISA Standard

Operazioni Moltiplicazione:
`Mul, Mulu` (Moltiplicazione unsigned), `Mad`

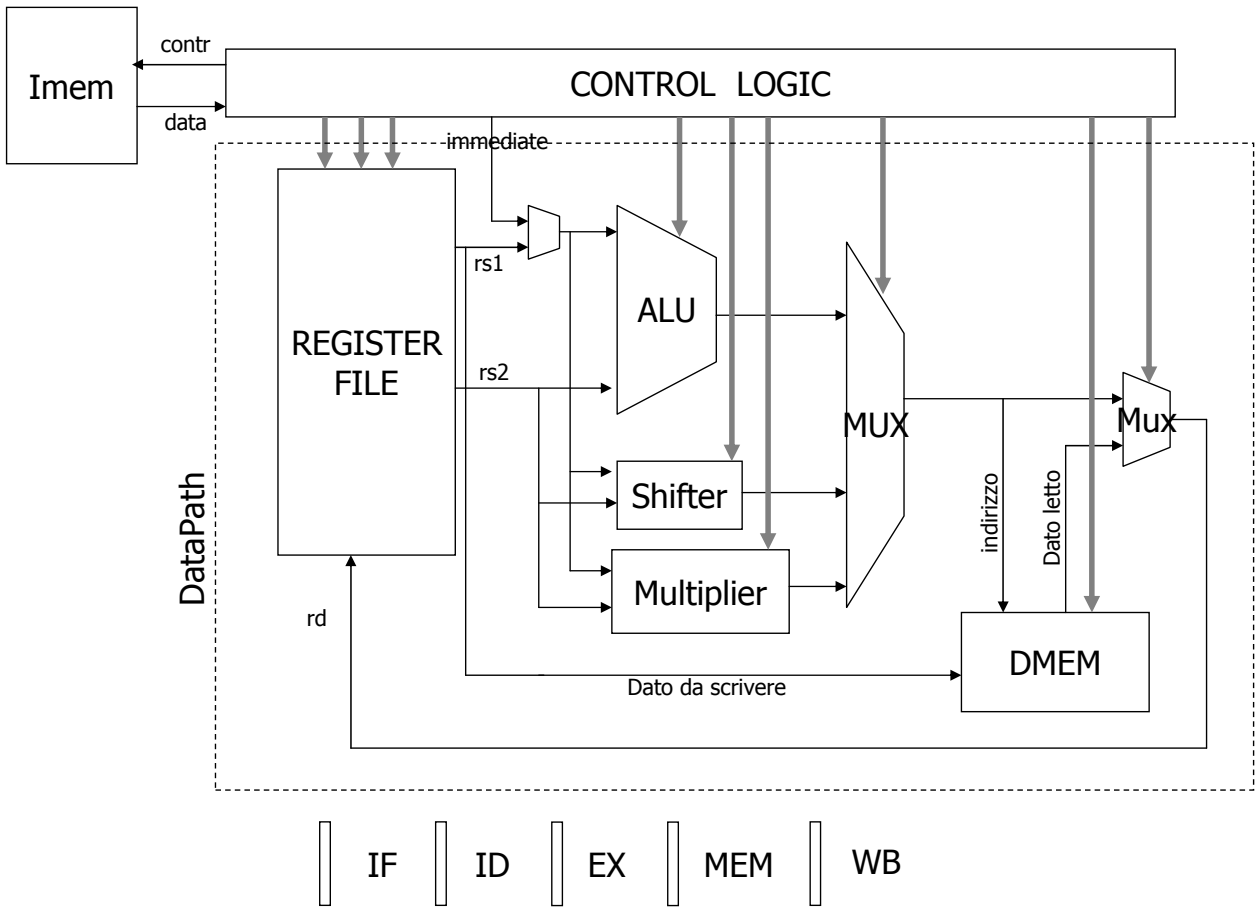
Operazioni Shift:
`Rotl, Rotr`
`Sllv` (shift immediate. V è una variable ed è semplicemente un tipo immediate)
`Srlv,Srav`

Branch con confronto rispetto a zero
`Bgtz,bltz,bgez,blez`

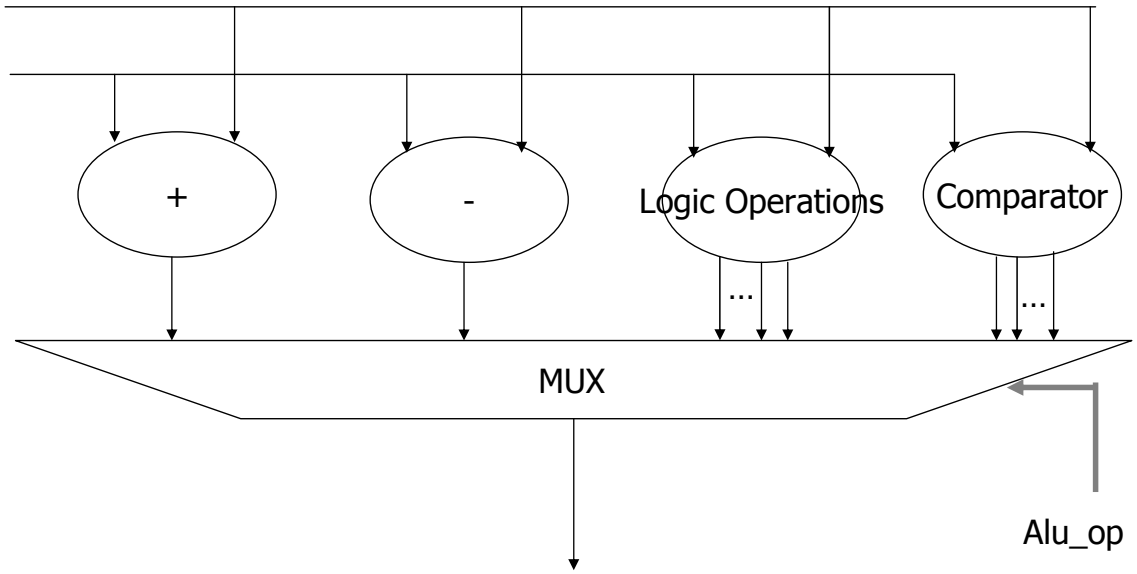
Branch con confronto tra due registri
`Beq,bneq, bgt,bge,blt,ble`

Branch con decremento per implementare cicli for
`Beqzdec, bneqzdec`

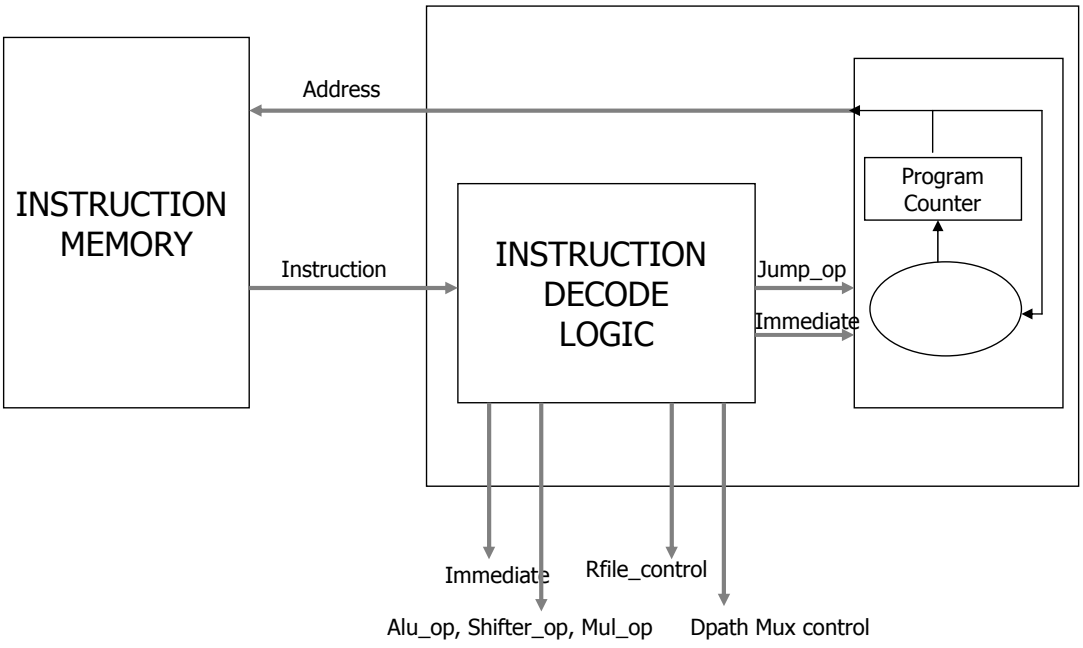
Architettura RISC



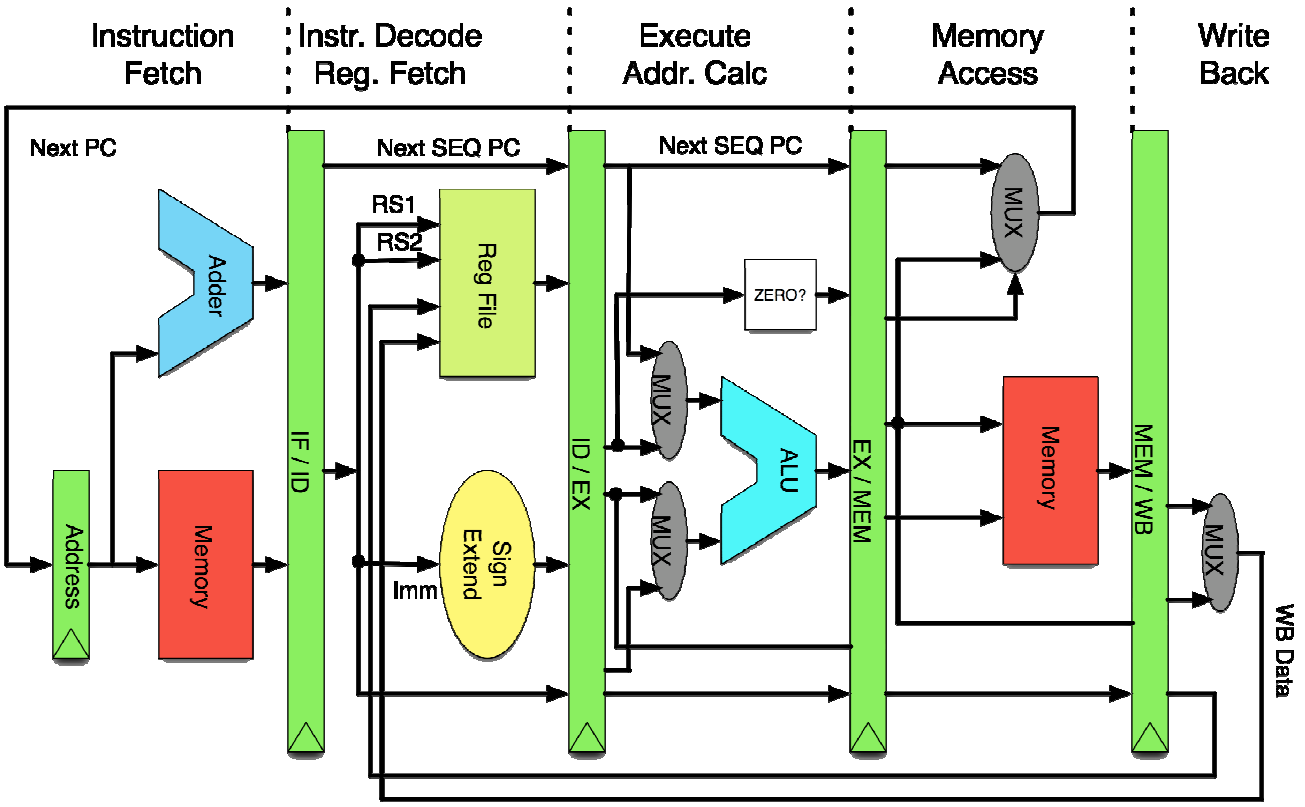
DataPath: Architettura ALU



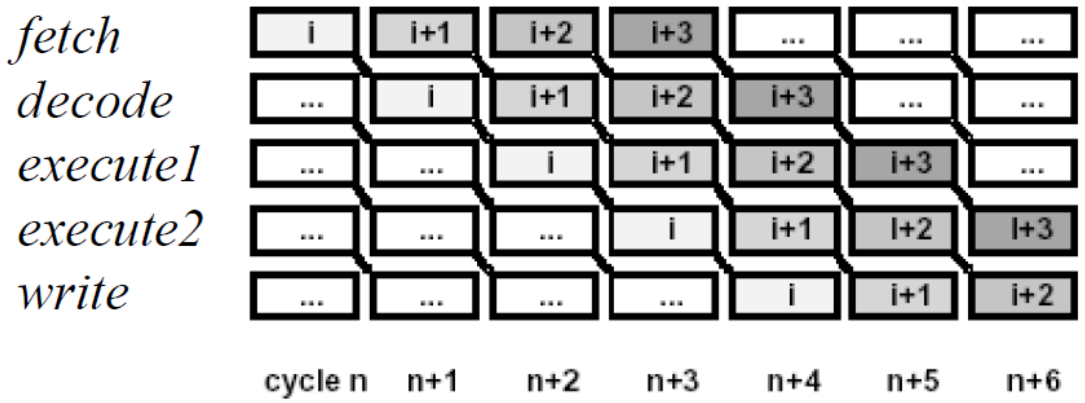
CONTROL LOGIC



Architettura RISC



Flusso istruzioni nella PIPELINE

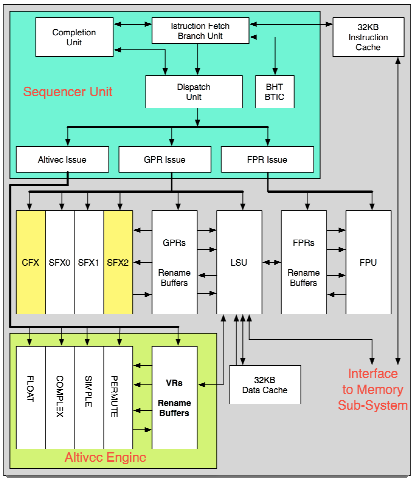


CISC (Complex Instruction Set Computers)

Con il termine CISC si intende un'architettura per microprocessori formata da un set di istruzioni in grado di eseguire operazioni complesse, quali la lettura di un dato, la sua modifica e il suo salvataggio direttamente in memoria tramite una singola istruzione.

- set di istruzioni molto esteso, non sempre sfruttato pienamente
- ogni istruzione è eseguita in un ciclo di clock
- alta densità di codice
- linguaggio macchina più vicino al linguaggio di programmazione

RISC vs CISC

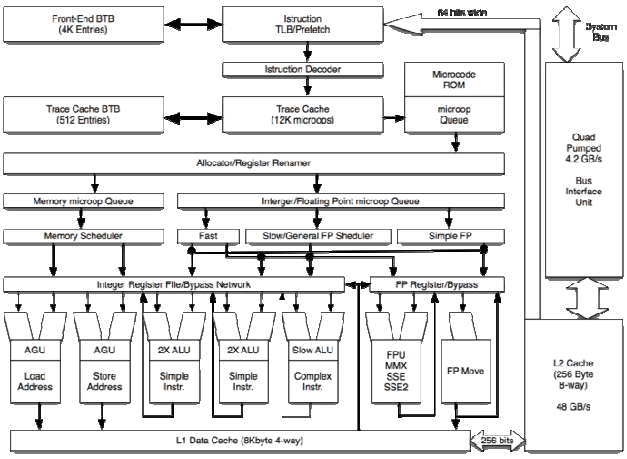


← **Architettura PowerPC G4 (RISC)**

- enfasi sul software
- istruzioni semplici (singolo ciclo)
- Operazioni R-R, trasferimenti M-R
- Scarsa densità del codice
- Numero di transistor usati per realizzare i registri

Architettura Pentium 4 (CISC) →

- enfasi sull'hardware
- istruzioni complesse (multiciclo)
- Trasferimenti M-M
- Codice Compatto
- Numero di transistor usati per memorizzare, decodificare le istruzioni



RISC vs CISC

esempio di salvataggio di un registro su Stack

- **CISC - Stack gestito in hardware**

PUSH AX

- **RISC - Stack gestito via software**

STORE R2, [RSP]

SUB RSP, RSP, 4

Prestazioni Instruction Set

$$\frac{tempo}{programma} = \frac{istruzioni}{programma} \cdot \frac{cicli}{istruzione} \cdot \frac{tempo}{ciclo}$$

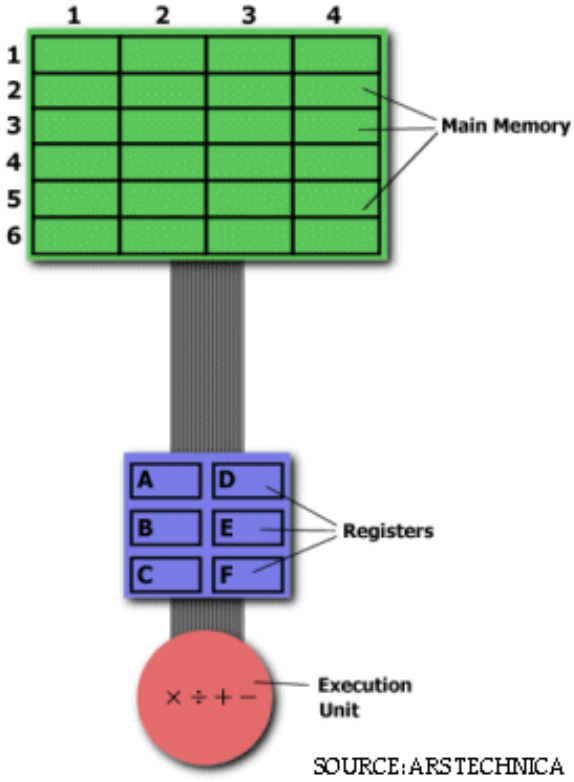
$$T_{exec} = N_{instr} \cdot CPI \cdot T_{ck}$$

CISC: N_{instr} cala RISC: T_{ck} cala

CPI: influenzato da architettura (pipelining, arch. superscalare, esecuzione fuori ordine, etc.)

RISC vs CISC

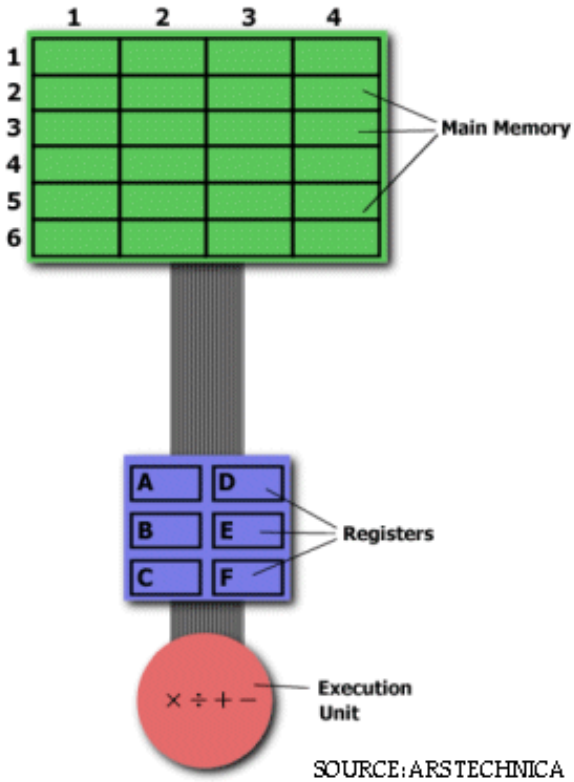
- Esempio.
  - Moltiplicazione di due numeri memorizzati in RAM.
  - Consideriamo due ipotetiche architetture e il seguente schema
  - Memoria indirizzata con x:y
  - Unità di esecuzione può operare solo tra registri
  - (2:3) <= (2:3) * (5:2)



RISC vs CISC

- CISC
  - Istruzioni complesse mappate sull'hardware disponibile
  - Poche istruzioni
  - Unità di controllo complessa, microcodice

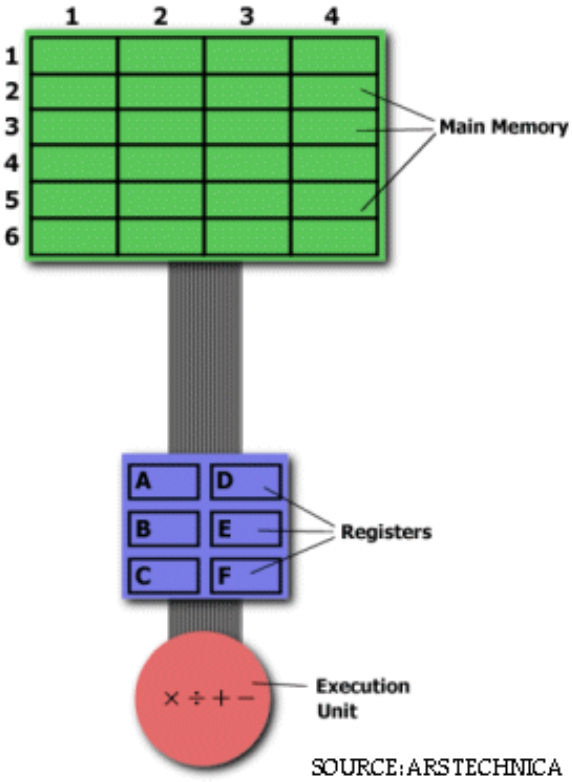
```
MULT  2:3,  5:2
```



RISC vs CISC

- RISC
  - Istruzioni a complessità ridotta
  - Diverse istruzioni
  - Unità di controllo “semplice”

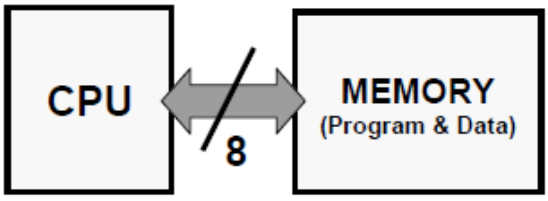
```
LOAD A, 2:3
LOAD B, 5:2
PROD A, B, A
STORE 2:3, A
```



Gestione memoria: architettura Von Neumann

L’architettura Von Neumann utilizza la medesima memoria sia per i dati che per le istruzioni.

- **Utilizzata di solito per processori *general purpose***
- **Prevede un BUS UNICO tra CPU e memoria**
- **RAM (Data Memory) e Program Memory, quindi devono condividere lo stesso bus, per cui devono avere entrambe parole della stessa lunghezza**



COLLO DI BOTTIGLIA: Il fatto di dover condividere un bus unico fa sì che per completare un’istruzione siano necessari 2 accessi in memoria (uno in RAM e uno in Program Memory) per cui si ha:

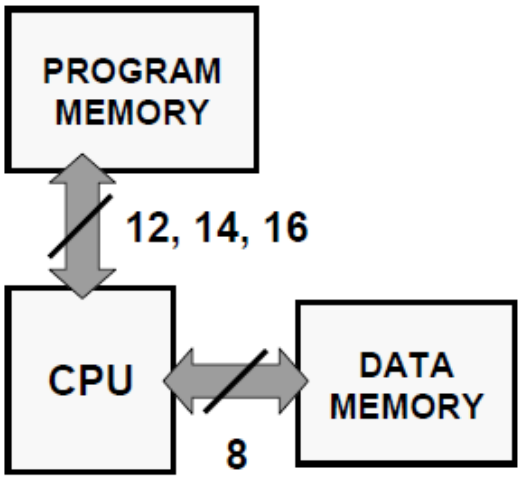
UNA ISTRUZIONE ESEGUITA OGNI 2 CICLI MACCHINA

Gestione memoria: architettura HARVARD

L'architettura Harvard è una tipologia di gestione della memoria che separa i dati dalle istruzioni.

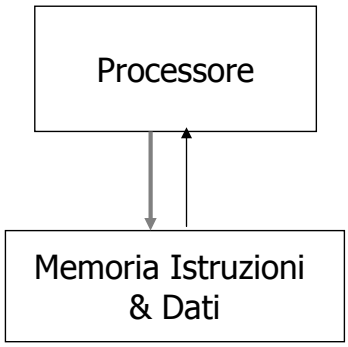
- Utilizzata di solito per processori RISC, come ad es. i PIC (Peripheral Interface Control)
- Prevede 2 BUS SEPARATI tra CPU, program memory e data memory
- RAM (Data Memory) e ROM (Program Memory) possono avere parole di lunghezza DIVERSA

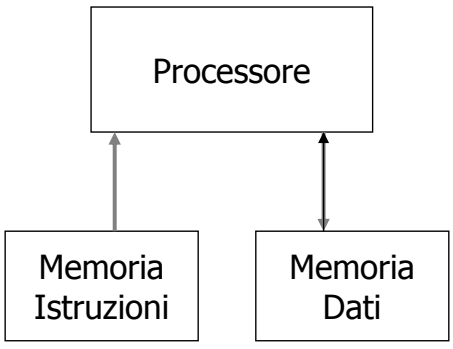
PIC: RAM 8 bit
ROM 12, 14 o 16 bit



La CPU può effettuare un accesso in RAM e uno in ROM contemporaneamente e sfruttando tecniche di *pipeline* si può arrivare ad eseguire 1 ISTRUZIONE OGNI CICLO MACCHINA

Von Neumann vs Harvard



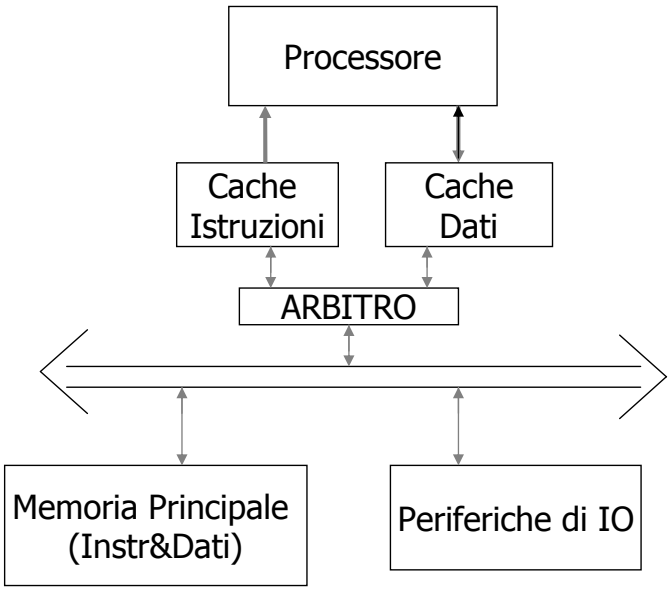


Macchina di Von Neumann (Architettura Princeton):
Vantaggio: Flessibilità
Svantaggio: Lentezza, scarso parallelismo. Adatta per architetture general purpose.

Architettura Harvard:
Vantaggio: Velocità, maggior parallelismo negli accessi. Sicuramente più adatta a macchine RISC e ad applicazioni embedded
Svantaggio: maggiori risorse (mem + canali comunicazione)

Esempio di architettura di memoria

Alternativa piuttosto comune in moderni sistemi embedded:
Core Harvard basato su due cache, che hanno accesso singolo (Von Neumann) al Bus.



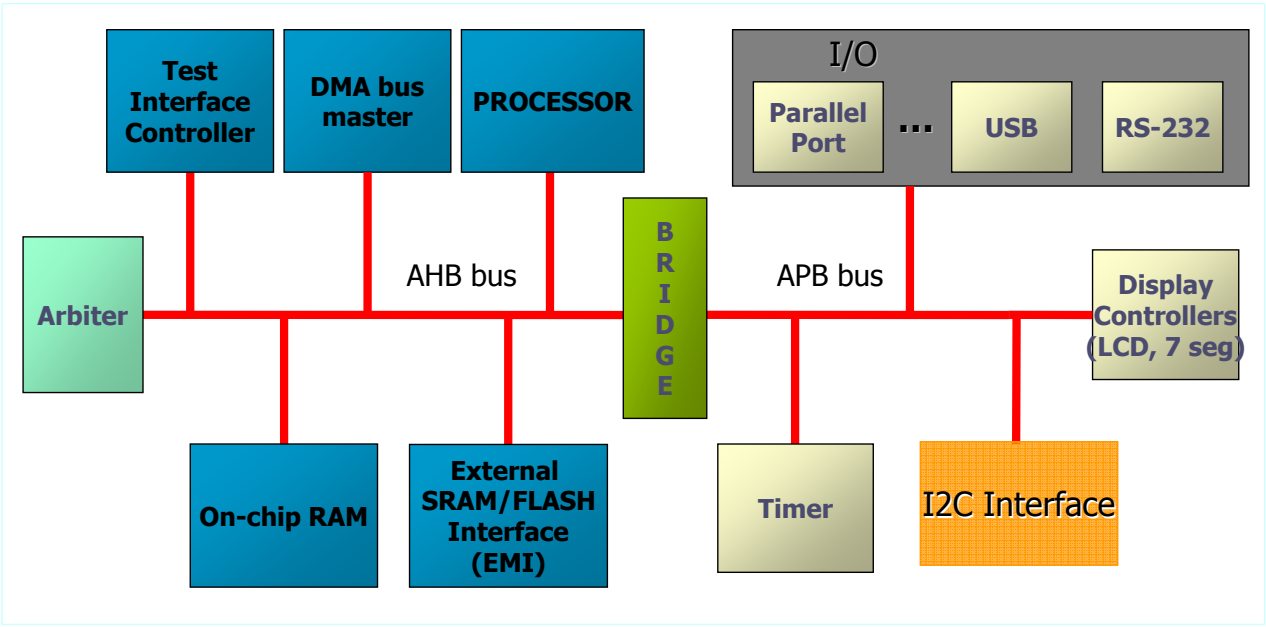
Architetture di BUS

AMBA
Sviluppato da ARM, è ormai lo standard per le architetture embedded

Coreconnect
Leggermente più complesso, e adatto anche a sistemi general purpose, è proprietario di IBM

VCI
(Virtual Component Interface): è uno standard open source, usato per distribuire blocchi di IP-reuse, ma non ha grande diffusione.

Advanced Microcontroller Bus Architecture

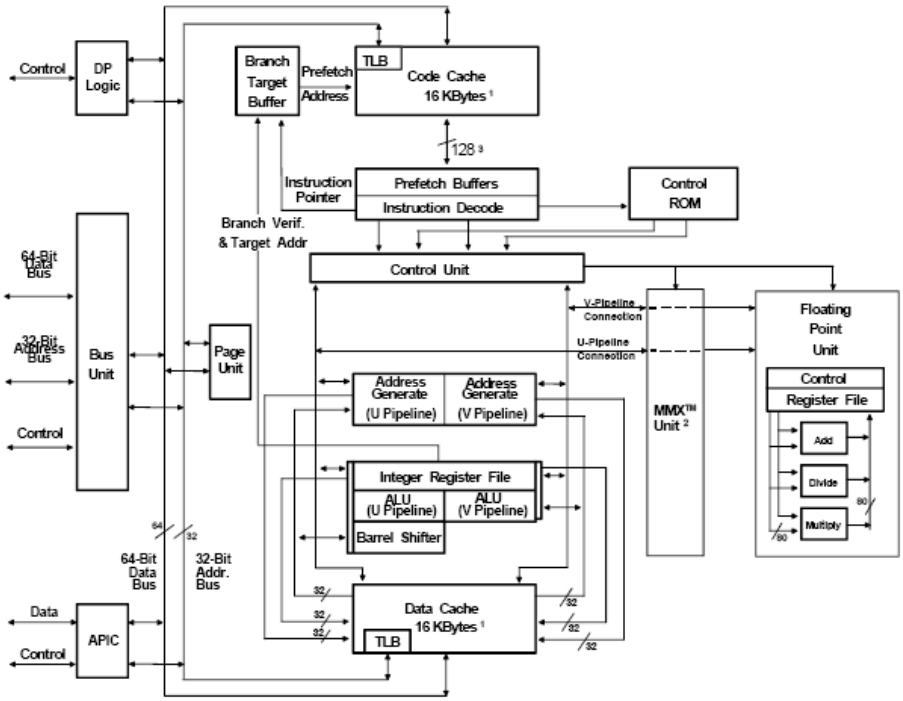


Architetture SUPERSCALARI e VLIW

- Si definiscono SUPERSCALARI processori che eseguono un numero>1 di istruzioni contemporaneamente, determinandone lo scheduling a tempo di esecuzione

- Processori VLIW (Very long Instruction Word), istruzioni che si riferiscono esplicitamente al parallellismo della macchina (1 VLIW = N iw in parallelo). Lo scheduling delle istruzioni avviene a tempo di compilazione: si semplifica la struttura hardware al prezzo di una maggiore complessità del passo di compilazione. (ovvero non serve un controllo hardware delle dipendenze di dato, ad esempio)

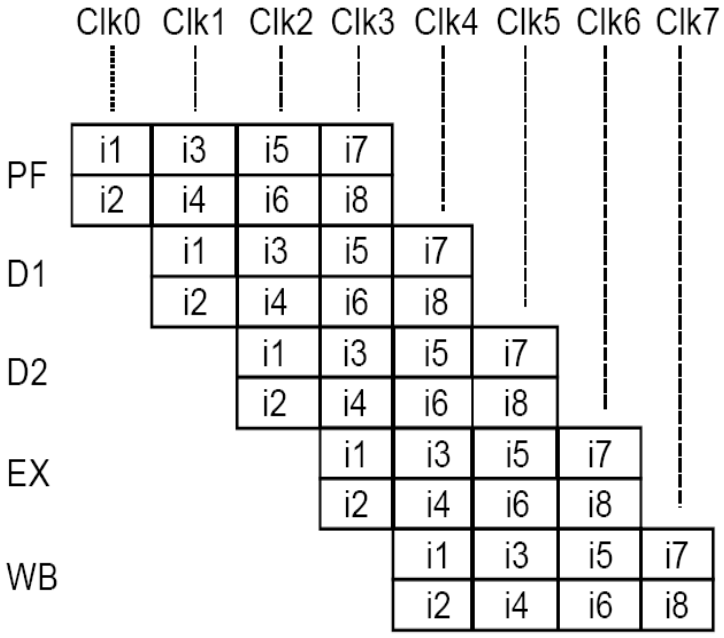
ES. Architettura superscalare: Intel Pentium



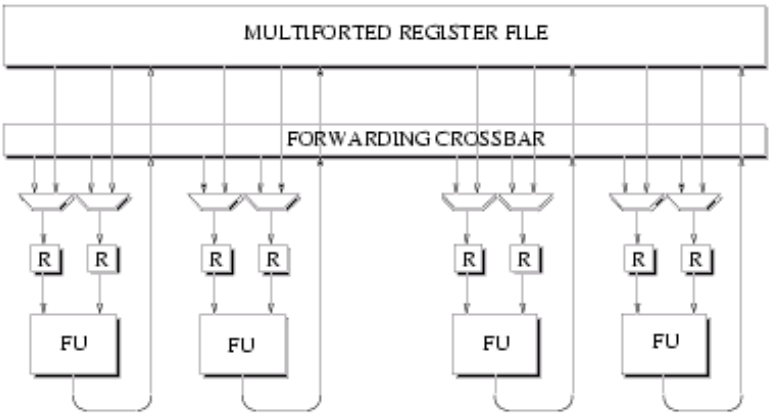
• architettura superscalare (2 pipelines: u, v) Max. 2 istruzioni immesse nelle pipes per ciclo di clock (dipendenze)
• branch prediction (BTB)

Pipeline Intel Pentium

- Due pipelines: u, v
- Struttura delle pipelines:
  PF prefetch (da cache o memoria est.)
  D1 instruction decode (decodifica di 2 istr. e decisione di immetterne 1 o 2 nelle pipes)
  D2 address generate (calcolo degli indirizzi di operandi in memoria)
  EX execute
  WB writeback



Architetture VLIW e superscalari




```
lw  $2,a  || nop
lw  $3,b  || nop
lw  $4,c  || addu $2,$2,$3
lw  $5,d  || mul $2,$2,$2
nop       || addu $4,$4,$5
lw  $5,e  || sll $3,$4,1
addu $3,$3,$4  || nop
subu $2,$2,$3  || nop
addu $2,$2,$5  || nop
j  $31  || addu $29,$29,24
```

Codice VLIW (2 datapath)

Vliw & Superscalar Processors

- ✓ Sfruttamento del parallelismo implicito del codice
- ✓ Presentano problemi di scheduling e hazard handling

ORGANIZZAZIONE DATAPATH

Il data path è un'unità di calcolo che contiene le unità di elaborazione (ALU) ed I registri necessari per eseguire le istruzioni nella CPU.

- **SISD** => *Single Instruction Single Data:* (classica architettura di tipo RISC a cui è associato un singolo percorso dati)
- **SIMD** => *Single Instruction Multiple Data:* Es. Alu multicanale. Architettura in cui la stessa operazione può essere eseguita parallelamente su più dati. Metodologia di calcolo comune a image processing e applicazioni di networking. (es.Intel MMX)
- **MIMD** => *Multiple instruction Multiple Data:* processori che permettono la coesistenza di multipli canali dati indipendenti, come VLIW e superscalari.
- **MISD** => Dsp o unità funzionali complesse sono in grado di eseguire una serie di istruzioni consecutive su un determinato dato.

Architetture SIMD: tecnologia MMX

- Esigenza di elaborazione di dati multimediali
  - Tipi di dato piccoli (pixel o canali video a 8 bit, campioni audio a 16 bit, etc.)
  - Elevata quantità di calcoli
  - Significativo parallelismo
- Intel, Tecnologia MMX (multimedia extension, matricial math extension)

Tecnologia MMX

- Introduzione di un nuovo set di registri (8 reg. a 64 bit)

  (in alias con l'unità FPU)

- Nuovi tipi di dato, mappabili su questi registri:
  - Packed bytes (8x8b)
  - Packed words (4x16b)
  - Packed doublewords (2x32b)
  - Quadword (1x64b)

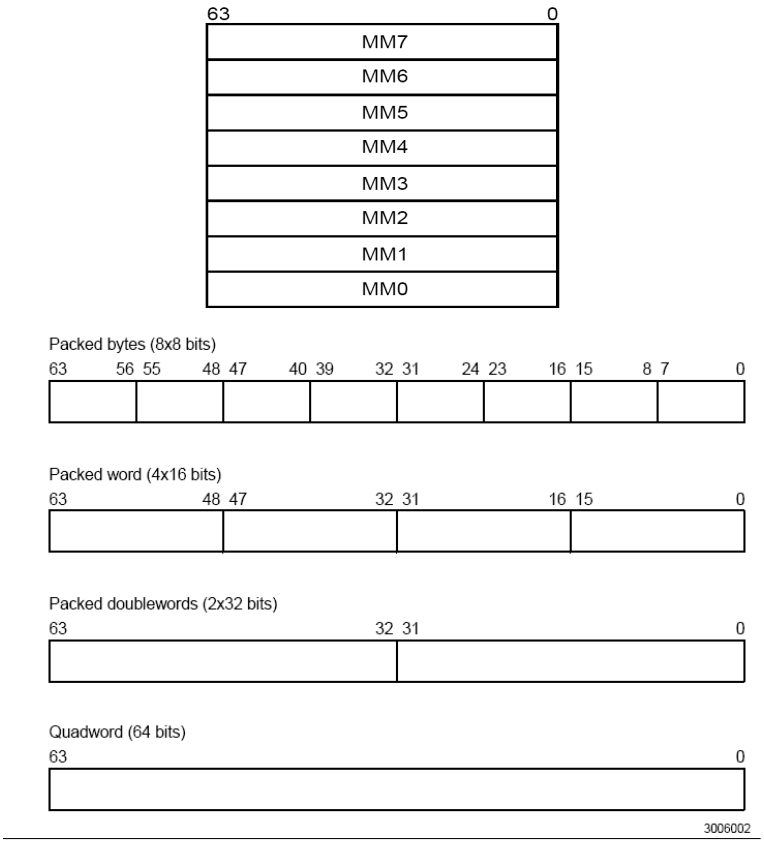
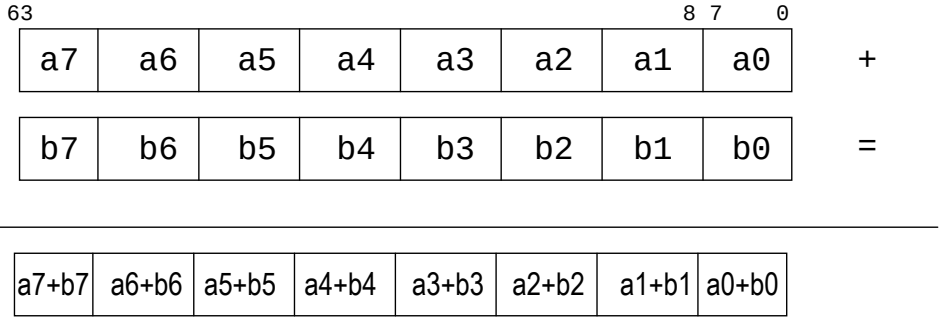


Figure 8-2. MMX™ Data Types

Tecnologia MMX

- Operazioni aritmetiche e logiche
- Aritmetica saturante (es.elab.immagini)
  e non-saturante (tradizionale)
- Operazioni di Pack/Unpack (conversioni tra tipi MMX)
- Operazioni di confronto
- Operazioni di shift

Es.



Tecnologia MMX

Table 8-2. MMX™ Instruction Set Summary

Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	Multiplication	PMULL, PMULH		
	Multiply and Add	PMADD		
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB
	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		

		Packed	Full Quadword
Logical	And		PAND
	And Not		PANDN
	Or		POR
	Exclusive OR		PXOR
Shift	Shift Left Logical	PSLLW, PSLLD	PSLLQ
	Shift Right Logical	PSRLW, PSRLD	PSRLQ
	Shift Right Arithmetic	PSRAW, PSRAD	

		Doubleword Transfers	Quadword Transfers
Data Transfer	Register to Register	MOVD	MOVQ
	Load from Memory	MOVD	MOVQ
	Store to Memory	MOVD	MOVQ
Empty MMX™ State		EMMS	

Esempi di processori commerciali

per ambienti embedded

- Microcontrollori e processori Discreti: Pic (Microchip), Mips, PowerPC, ST 10/20/200, altri prodotti di TI o Motorola, Intel 80x
- Microprocessori Embedded (cores VHDL): Arm, Mips, Tensilica (Processore reconfigurabile a tempo di compilazione VHDL)
- Soft Cores: Nios, MicroBlaze, Leon (Open-Source di Esa), diversi prodotti accademici
- DSP: Ti, Hitachi, Motorola

Funzionalita' Interne

Internal architecture
GP vs. DSP

GP	DSP
• Traditionally comes without parallel multiplier	• Hardware multiplier (usually parallel)
• Not all arithmetic op can be performed in 1 CK cycle	• All key arithmetic operations are performed in 1 CK cycle
• No MAC	• Native MAC support
• Different data format (FP)	• Mostly fixed-point (integer)
• No explicit fixed point support	• Hardware support for granting arithmetic fidelity (normalization, rounding, saturation…)
• Very generic (orthogonal) data path	

Gestione della memoria

Memory architecture
GP vs. DSP

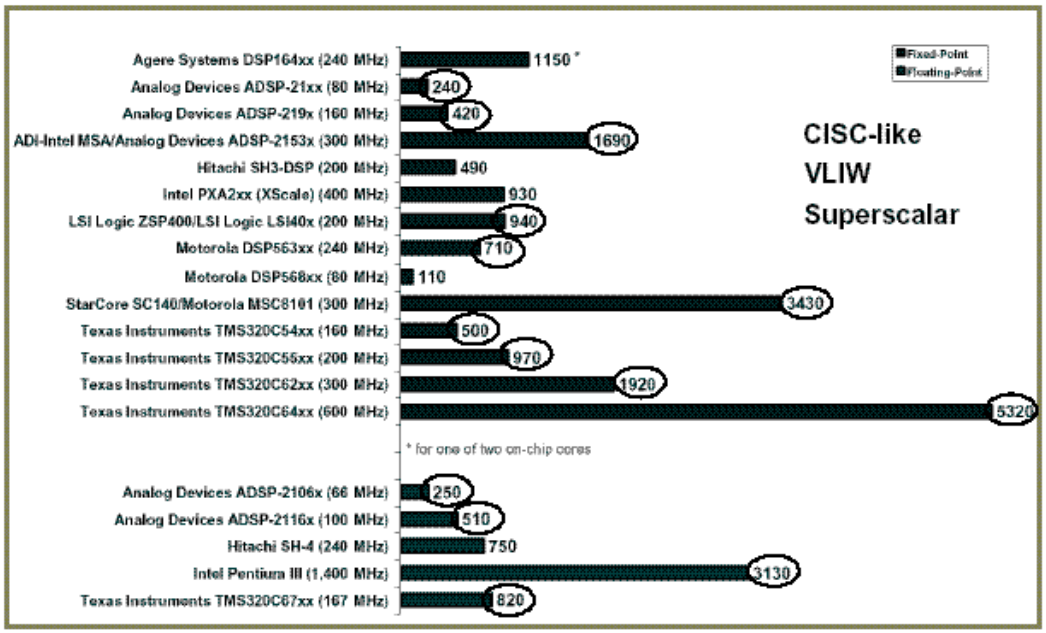
GP

- Von Neumann / *recently* migrated to Harvard
- Harvard achieved through a large use of cache memory
- Cache leads to stochastic execution times (not completely predicable)
- Need for a large addressing space
- Usually one (two) memory access per clock cycle

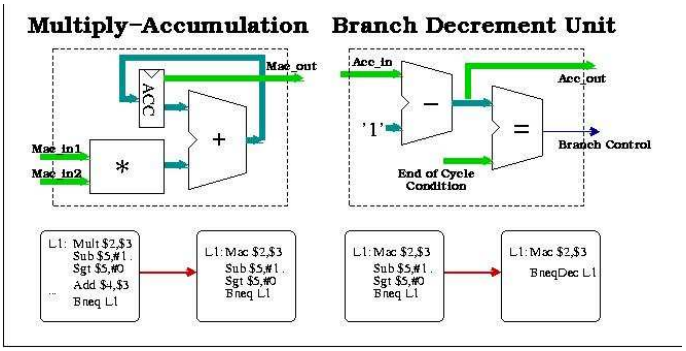
DSP

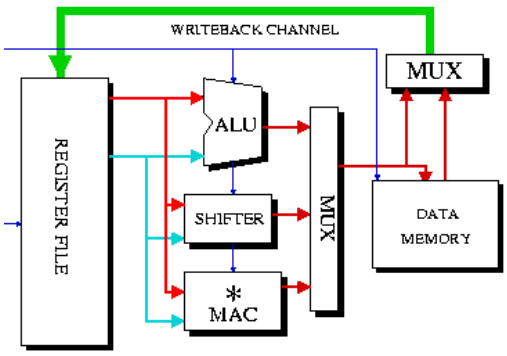
- Harvard since from the beginning (1982)
- Harvard achieved through the use of on-chip RAM banks
- Access time completely predictable
- Limited address space (small program)
- Several memory access per clock cycle (EHA)

Performance comparison



Caratteristiche dei DSP





Processori DSP

- ✓ Unita' funzionali application-specific applicate ad un ambiente software-programmable
- ✓ Difficolta' con i compilatori -> largo uso di assembly

DSP: Applicazioni

• Wireless – Telefonia Mobile (RF Codecs, Voice band Radio)
• Consumer Audio (Stereo A/D, D/A, Audio compression)
• Multimedia (Image Compression Codecs and Filtering)
• DTAD (Segreteria Telefonica Automatica)
(Sintesi e riconoscimento del parlato)
• Automotive
(Active Suspensions,Injection Control etc.)
• HDD (Memorie di Massa)