Von Neumann

- Memoria veloce (cache)
- Memoria fissa (ROM/RAM)
- Unità di calcolo (CPU)

Ciclo macchina

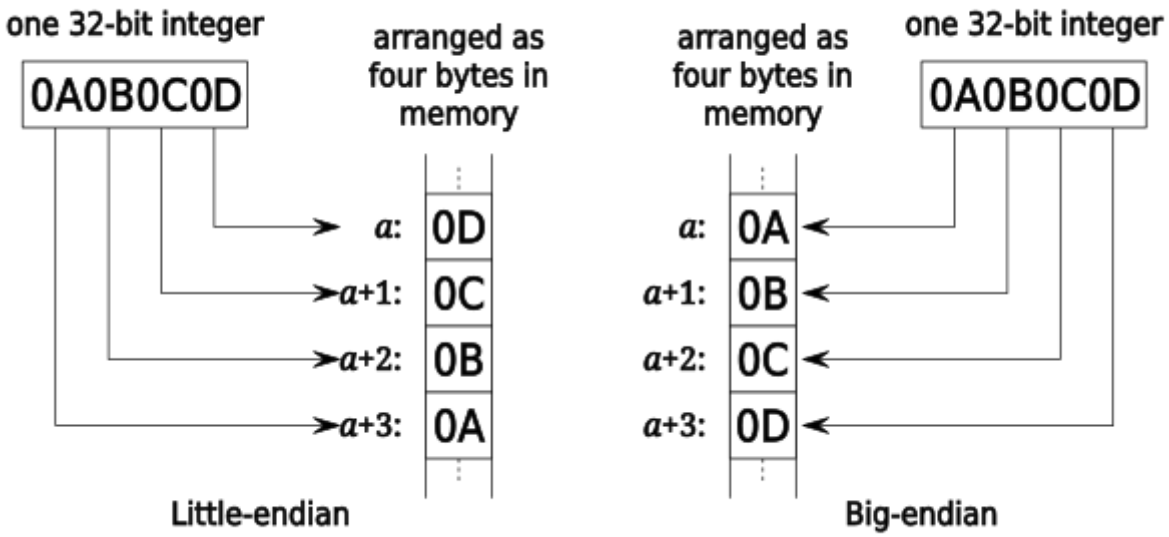
Tendenzialmente eseguita dalla ALU (Unità aritmetico-logica)

- Fetch (Prelievo)
- Decode (Decodifica)
- Execute (Esecuzione)

A livello pratico:

1. Prelievo dalla memoria
    1. Se locale = Non perdo tempo (non andare in ROM e poi salvarlo in cache)
    2. Se non locale = Giro lungo (fisicamente andare a recuperare il dato)
2. Decodifico il formato ma anche l'istruzione
    1. `ADD 0x80, 0x0`
3. Eseguo l'istruzione prelevata nel modo corretto
    1. Dipende dall'architettura

Codifica (dal bit più significativo / dal bit meno significativo):



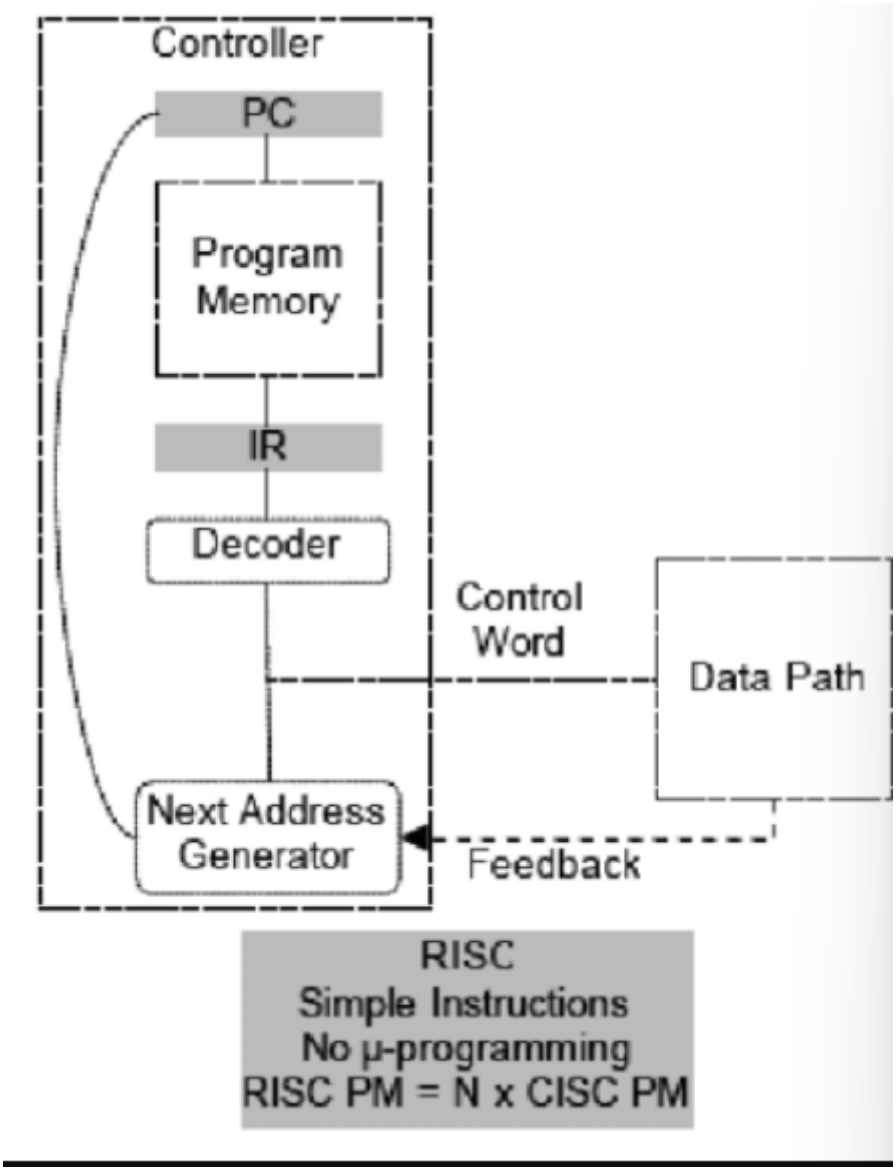
CISC/RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	Register to register: "LOAD" and "STORE" are independent instructions
Small code sizes, high cycles per second	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers

RISC (pipeline)

- - - -

Ciclo che viene fatto in parallelo sequenzialmente (= pezzo per pezzo/uno dopo l'altro) su più core (nuclei di elaborazione della CPU)

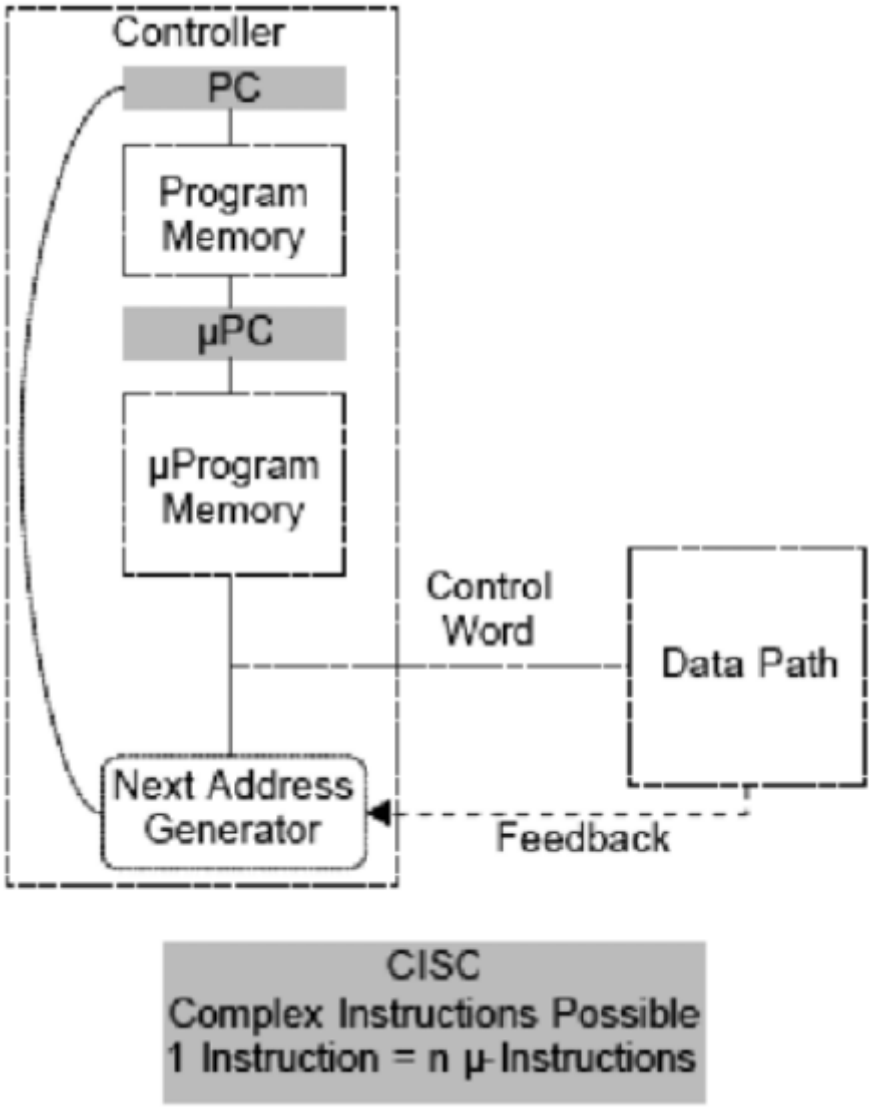


CISC

Caratteristiche generali:

- clock pari ad un Mhz
- lunghezza della parola pari a 16 bit, con la possibilità di indirizzare anche metà parola
- sistema di interconnessione a bus - fu la prima macchina ad implementarlo
- memoria centrale di 32k parole organizzata in 4 pagine per un totale di 256 kbyte di spazio
- una parte di memoria dedicata allo stack
- numeri interi rappresentati in complemento a 2, mentre i naturali in modo standard binario
- interruzioni/eccezioni vettorizzate

	N Z V C	TRAP (1 bit)	PRIORITÀ (3 bit)	MODO DI FUNZIONAMENTO (2 bit)	



CISC
Complex Instructions Possible
1 Instruction = n μ-Instructions

Pipeline

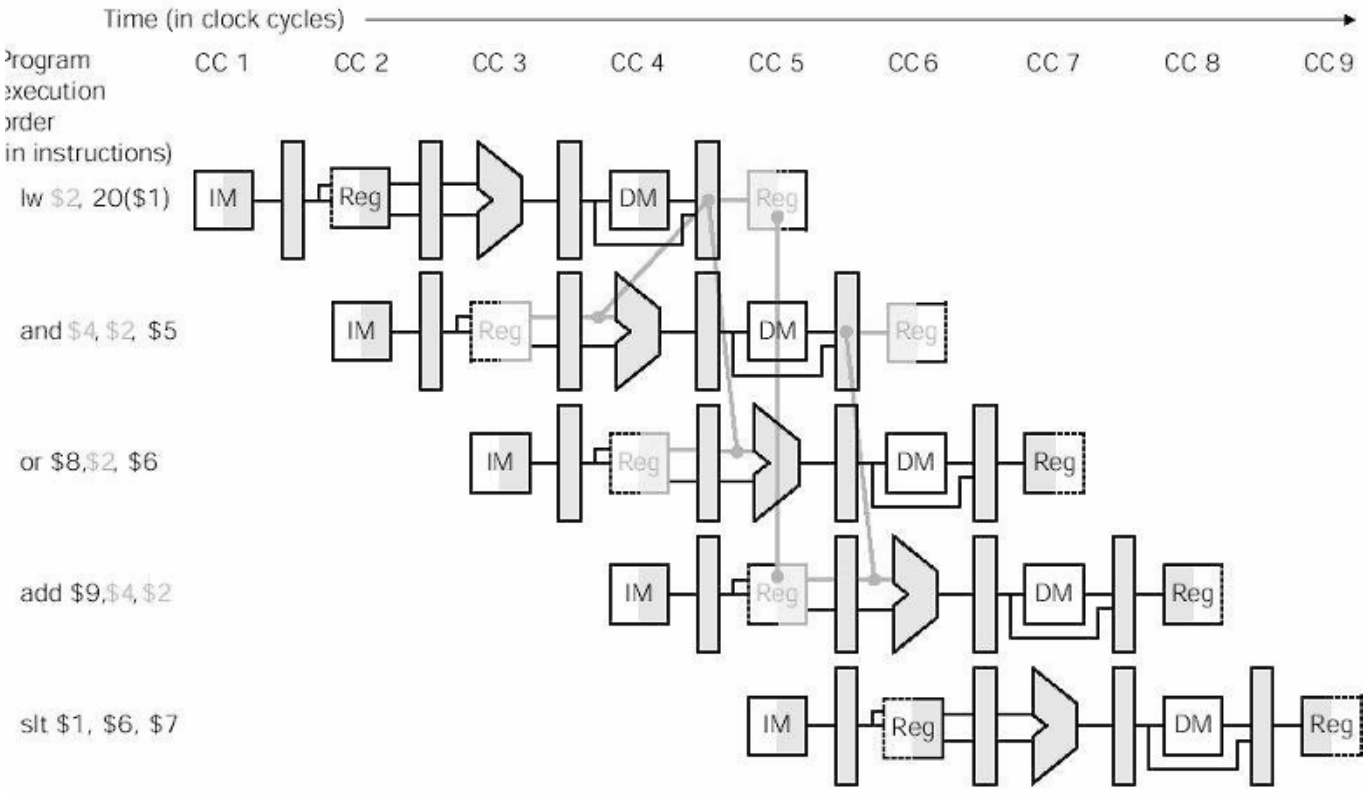
Tendenzialmente si usa RISC usando set di istruzioni apposite:

Possibili casi

Situazione	Esempio di codice	Azione
Nessuna dipendenza	LD **$1**, 45($2) DADD $5, $6, $7 DSUB $8, $6, $7 OR $9, $6, $7	Non occorre fare nulla perché non c'è dipendenza rispetto alle 3 istruzioni successive
Dipendenza che richiede uno stallo	LD **$1**, 45($2) DADD $5, **$1**, $7 DSUB $8, $6, $7 OR $9, $6, $7	Opportuni comparatori rilevano l'uso di $1 in DADD ed evitano il rilascio di DADD
Dipendenza risolvibile con un forwarding	LD **$1**, 45($2) DADD $5, $6, $7 DSUB $8, **$1**, $7 OR $9, $6, $7	Opportuni comparatori rilevano l'uso di $1 in DSUB e inoltrano il risultato della load alla ALU in tempo per la fase EX di DSUB
Dipendenza con accessi in ordine	LD **$1**, 45($2) DADD $5, $6, $7 DSUB $8, $6, $7 OR $9, **$1**, $7	Non occorre fare nulla perché la lettura di $1 in OR avviene dopo la scrittura del dato caricato

Pipeline (MIPS)

Esempio



Assembly

`ADD rax, 0x0` - Esempio di istruzione

X64 = registri (memorie più veloci della CPI a 64 bit)

8-byte register	Bytes 0-3	Bytes 0-1	Byte 0
%rax	%eax	%ax	%al
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rbx	%ebx	%bx	%bl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rsp	%esp	%sp	%spl
%rbp	%ebp	%bp	%bpl
%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b

Opcode = Codice dell'operazione
--> ADD, SUB

Indirizzamenti a parole:

- byte = 8 bit
- word = 16 bit
- doubleword = 32 bit
- quadword = 64 bit

Instruction		Description
		Instructions with one suffix
mov	*S, D*	Move source to destination
push	*S*	Push source onto stack
pop	*D*	Pop top of stack into destination

3.2.1 Unary Operations

Instruction		Description	Page #
`inc`	*D*	Increment by 1	178
`dec`	*D*	Decrement by 1	178
`neg`	*D*	Arithmetic negation	178
`not`	*D*	Bitwise complement	178

3.2.2 Binary Operations

Instruction		Description	Page #
`leaq`	*S, D*	Load effective address of source into destination	178
`add`	*S, D*	Add source to destination	178
`sub`	*S, D*	Subtract source from destination	178
`imul`	*S, D*	Multiply destination by source	178
`xor`	*S, D*	Bitwise XOR destination by source	178
`or`	*S, D*	Bitwise OR destination by source	178
`and`	*S, D*	Bitwise AND destination by source	178

3.4.2 Jump Instructions

Instruction		Description
`jmp`	*Label*	Jump to label
`jmp`	**Operand*	Jump to specified location
`je` / **`jz`**	*Label*	Jump if equal/zero
`jne` / **`jnz`**	*Label*	Jump if not equal/nonzero
`js`	*Label*	Jump if negative
`jns`	*Label*	Jump if nonnegative
`jg` / **`jnle`**	*Label*	Jump if greater (signed)
`jge` / **`jnl`**	*Label*	Jump if greater or equal (signed)
`jl` / **`jnge`**	*Label*	Jump if less (signed)
`jle` / **`jng`**	*Label*	Jump if less or equal
`ja` / **`jnbe`**	*Label*	Jump if above (unsigned)
`jae` / **`jnb`**	*Label*	Jump if above or equal (unsigned)
`jb` / **`jnae`**	*Label*	Jump if below (unsigned)
`jbe` / **`jna`**	*Label*	Jump if below or equal (unsigned)

3.5 Procedure Call Instruction

Procedure call instructions do not have any suffixes.

Instruction		Description
`call`	*Label*	Push return address and jump to label
`call`	**Operand*	Push return address and jump to specified location
`leave`		Set **`%rsp`** to **`%rbp`**, then pop top of stack into **`%rbp`**
`ret`		Pop return address from stack and jump there