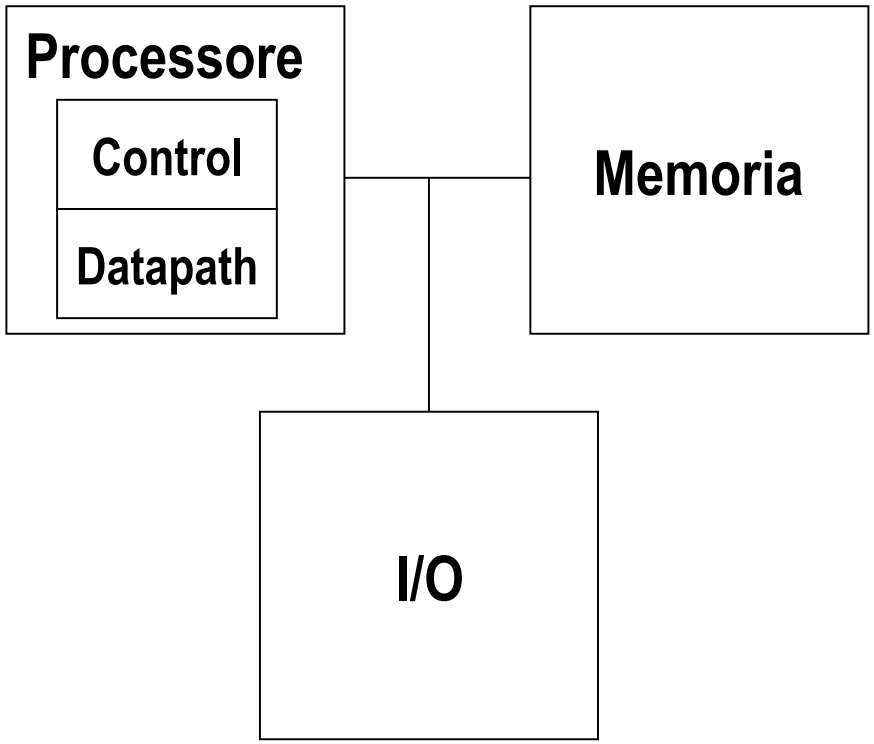
Architettura degli Elaboratori
Modulo 2

Salvatore Orlando

http://www.dsi.unive.it/~architet

Contenuti

- Approfondiremo il progetto e le prestazioni delle varie componenti di un calcolatore convenzionale
  - *processore (CPU)*
    - parte operativa (datapath)
    - parte controllo (control)
  - *Memoria*
    - cache, principale, di massa
  - *Input/Output (I/O)*
    - mouse, tastiera (I), video, stampante (O), dischi (I/O), CD (I/O o I), rete (I/O), bus per i collegamenti



Contenuti

- **Progetti di CPU MIPS-like**
  - *esecuzione di ogni istruzione in un singolo ciclo di clock*
  - *esecuzione di ogni istruzione in un numero variabile di cicli di clock (multi-ciclo)*
  - **esecuzione parallela di più istruzioni**
- **Memoria**
  - **Gerarchie di memoria**
  - **Cache e memoria virtuale**
- **I/O**
  - **Dispositivi fisici e bus**
  - **Tecniche hw / sw per la programmazione dell'I/O**
- **Valutazione delle prestazioni**
  - **Studieremo come i tipi e il numero delle istruzioni eseguite, la frequenza del processore, il parallelismo interno al processore, le gerarchie di memorie e l'I/O influenzano il tempo di esecuzione di un programma**

Contenuti

- Non solo linguaggio macchina
  - Linguaggio assembler - Linguaggio ad alto livello
- Esecuzione dei programmi
  - compilatore, assemblatore, linker, loader
- Programmazione assembler
  - Traduzione assembler (compilazione) delle principali strutture di controllo di un linguaggio ad alto livelli
  - Funzioni e allocazione della memoria
  - Programmazione di I/O
  - Semplici strutture dati
- Uso del simulatore SPIM ed esercitazioni

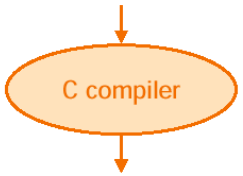
- Tutor per esercitazioni e lab

Linguaggio assembler e linguaggio macchina (caso di studio: processore MIPS)

Livelli di astrazione

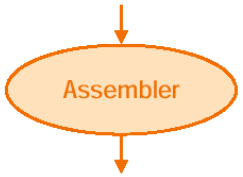
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```



Assembly language program (for MIPS)

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```



Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

- **Scendendo di livello, diventiamo più concreti e scopriamo più informazione**
- **Il livello astratto omette dettagli, ma ci permette di trattare la *complessità***

Quali sono i dettagli che via via scopriamo scendendo di livello?

Livelli di astrazione

- **Il linguaggio di alto livello “astrae” dalla specifica piattaforma, fornendo costrutti più semplici e generali**
- **Cioè, è una “interfaccia generica”, buona per ogni computer (con tutti i vantaggi che questo comporta)**
- **Ma proprio perché è uguale per tutte le macchine, NON può fornire accesso alle funzionalità specifiche di una determinata macchina**
  - **Il linguaggio macchina/assembler permette di fare cose fisicamente impossibili per altri linguaggi (tipo, accedere e manipolare schede grafiche, registri di memoria, ecc.)**

Linguaggio Macchina

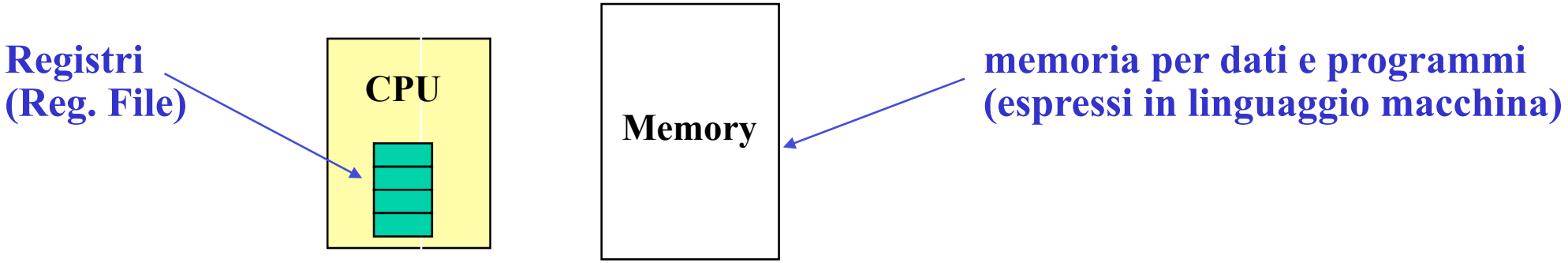
- **Linguaggio della Macchina**
- **Più primitivo dei Linguaggi ad Alto Livello**
  - **es., controllo del flusso poco sofisticato (non ci sono *for, while, if*)**
- **Linguaggio molto restrittivo**
  - **es., istruzioni aritmetiche del MIPS sono solo 3 operandi**

- **Studieremo l'ISA (Instruction Set Architecture) del MIPS**
  - **simile ad altre architetture sviluppate a partire dagli anni '80**

Scopi di progetto dell'ISA: massimizzare le prestazioni - minimizzare i costi, anche riducendo i tempi di progetto

Concetto di “Stored Program”

- **Istruzioni** sono stringhe di bit con un dato formato di rappresentazione
- Programmi sono **sequenze di istruzioni**
- Programmi (come i dati) **caricati in *memoria*** per l’esecuzione
  - La CPU legge le istruzioni dalla memoria (come i dati)



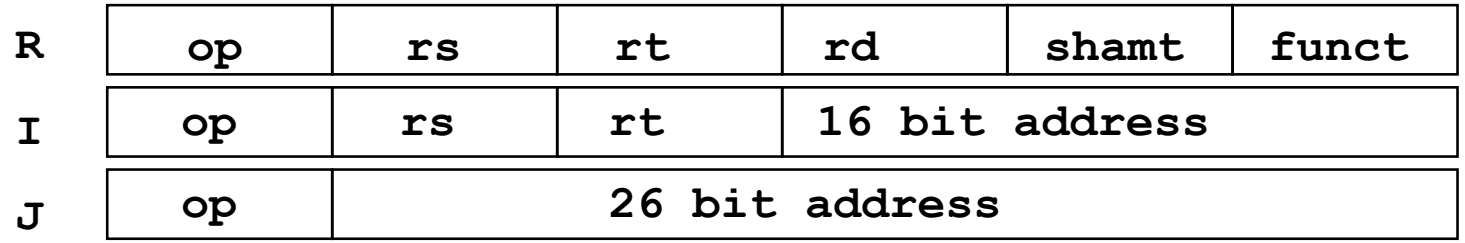
- Ciclo **Fetch & Execute**
  - CPU **legge** (fetch) istruzione corrente (indirizzata dal **PC**=Program Counter), e la pone in un registro speciale interno
  - CPU usa i bit dell’istruzione per "*controllare*" le azioni da svolgere, e su questa base **esegue** l’istruzione
  - CPU determina “*prossima*” istruzione e ripete ciclo

Instruction Set Architecture (ISA) del MIPS

- **Istruzione** **Significato**

```
add $4,$5,$6          $4 = $5 + $6
sub $4,$5,$6          $4 = $5 - $6
lw $4,100($5)         $4 = Memory[$5+100]
sw $4,100($5)         Memory[$5+100] = $4
bne $4,$5,Label       Prossima istr. caricata dall'indirizzo
                      Label, ma solo se $s4 ≠ $s5
beq $4,$5,Label       Prossima istr. caricata dall'indirizzo
                      Label, ma solo se $s4 = $s5
j Label               Prossima istr. caricata dall'indirizzo
                      Label
```

- **Formati:**



Instruction Set Architecture (ISA) del MIPS

• **Istruzione**	**Significato**

`slt $10, $4, $5`	`if $4 < $5 then`
	`  $10 = 1`
	`else`
	`  $10 = 0`

`and  $4, $5, $6`	`$4 = $5 & $6`
`or  $4, $5, $6`	`$4 = $5 | $6`
`addi $4, $5, const`	`$4 = $5 + const`
`slti $4, $5, const`	`if $5 < const then` `  $4=1 else $4=0`
`andi $4, $5, const`	`$4 = $5 & const`
`ori  $4, $5, const`	`$4 = $5 | const`

Instruction Set Architecture (ISA) alternativi

Caratteristiche ISA

- Abbiamo visto le principali istruzioni del MIPS
  - simili a quelle presenti nell'ISA di altri processori

- ISA possono essere categorizzati rispetto a:
  - Modalità di indirizzamento (tipi di operandi)
  - Numero di operandi
  - Stile dell'architettura
  - CISC (Complex Instruction Set Computers)

    vs.

    RISC (Reduced …..)

Modalità di indirizzamento

… descrive gli operandi permessi e come questi sono usati

- **Ogni tipo di istruzione può avere *modalità multiple di indirizzamento***
  - **Esempio, l' `add` del processore SPARC ha una versione a 3-registri, una a 2-registri e una con un operando immediato**
- **I *nomi* dei vari modi di indirizzamenti sono parzialmente standardizzati**

Modi di indirizzamento nel MIPS

Immediate:

Constant & register(s)

`addi`

Register:

Only registers

`add`

Base/displacement:

Memory[Register + Constant]

`lw, sw`

PC-relative:

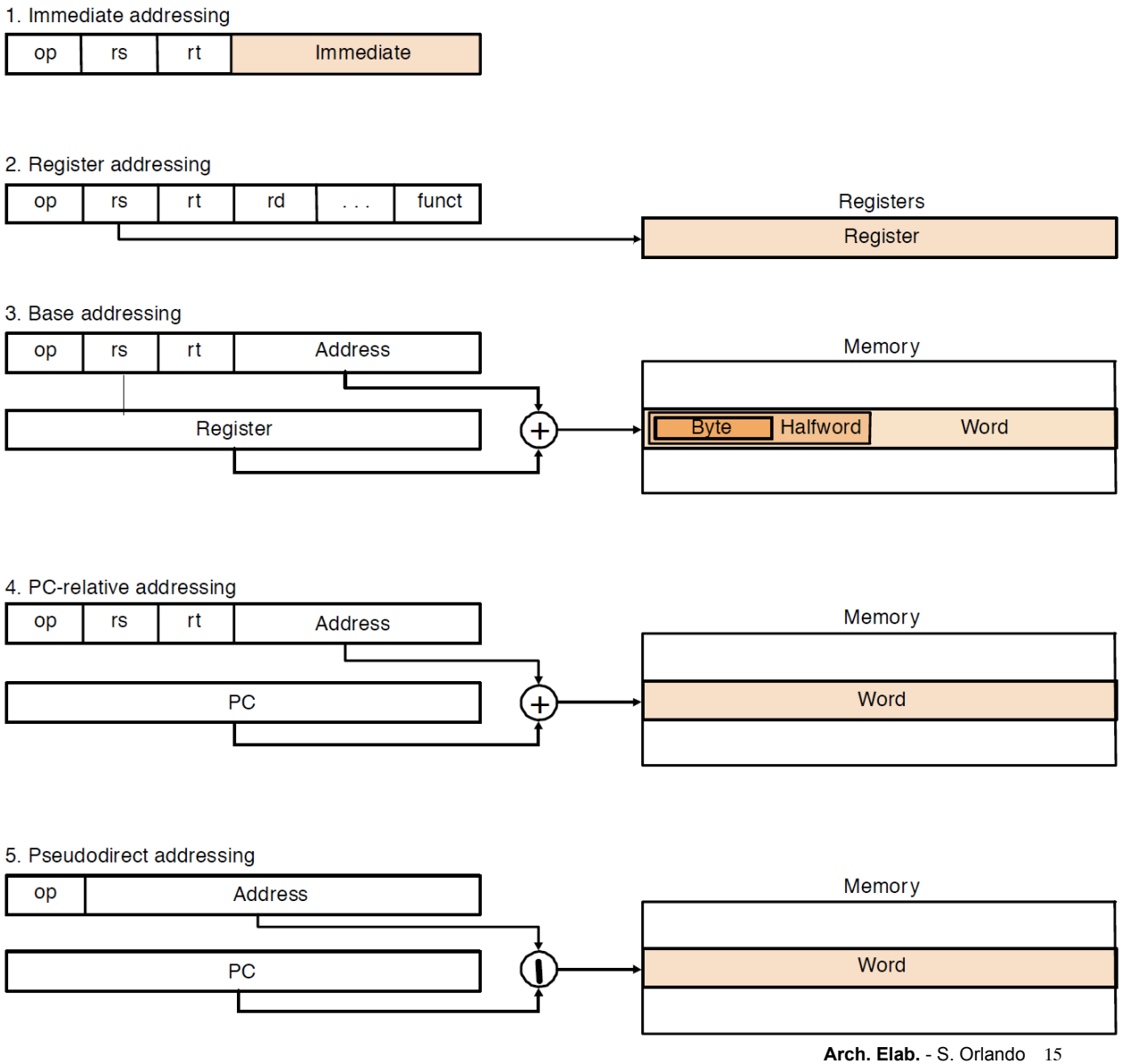
PC + Constant

`beq`

Pseudodirect:

Constant | (PC's upper bits)

`j`



Modi di indirizzamento: SPARC

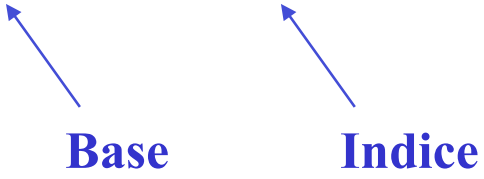
L'architettura Sun (SPARC) è di tipo RISC, come il MIPS.

Ha modalità di indirizzamento simili al MIPS, con in più …..

Indexed:

Memory[Register + Register]

`ld, st`



Modi di indirizzamento: altri ISA di tipo CISC

80x86:

Register indirect:

Memory[Register]

Semplificazione del modo base

Scaled index (diverse versioni):

Memory[Register + Register * Immediate]

Per indicizzare grandi array

Register-

{register,immediate,memory} & Memory{register,immediate}

Non è possibile avere 2 operandi di memoria nella stessa istr.

VAX:

Altri modi, come i seguenti:

Autoincrement & autodecrement:

Memory[Register]

che anche incrementa/ decrementa contestualmente il registro.

Utile per indici di loop.

Le motivazione: comandi C come

`x++, ++x, x--, --x`

Stili architetturali

- **I trend di sviluppo delle architetture sono:**
  - **Hardware meno costoso, più facile da costruire**
    - **Possiamo complicare il progetto**
    - **Aumentare lo spazio di indirizzamento**
    - **Memoria meno costosa e capiente**
  - **Miglioramento della tecnologia dei compilatori**
    - **Miglior uso dell'hardware**
    - **Non è necessario codificare in assembler per ottimizzare il codice**
  - **"Gap" sempre più grande tra**
    - **Velocità dei processori & "lentezza" relativa della memoria**

- **Ma quali sono i possibili stili dell'ISA dei processori? C'è un qualche stile che è risultato vincente nel tempo?**
  - **Ogni classificazione è chiaramente imperfetta, a causa delle innumerevoli variazioni**

Stile Accumulatore

- Solo un registro
  - *Accumulatore* "source & destination" di ogni istruzione. L'altro source in memoria o costante
- Altri registri special-purpose: SP, PC, …

- Esempio per A=B+C:

```
load  B
add   C
store A
```

- Esempio di processori:
  - Intel 8086
  - Zilog Z80

Vantaggi:

- Semplice da progettare e implementare
- Dimensione del codice medio

Svantaggi:

- Relativamente lento

  Molti accessi alla memoria

  Molti movimenti di dati: non ci sono registri temporanei

Evoluzione:

- Istruzioni più potenti

Stile Registro-Memoria

- Un operando in memoria
- Più 1/2 operandi nei registri.
- Ci sono registri general-purpose.

Esempio per A=B+C:

```
load  r1, B
add  r1, r1, C
store  A, r1
```

- Esempio di processori:
  - Intel 80386:
  - Estende l'8086 con istruzioni register-memory

Vantaggi:

- Più veloce
  - Meno accessi alla memoria & meno movimenti di dati
- Dimensione del codice medio
  - Meno istruzioni
  - Istruzioni più lunghe, a formato variabile

Evoluzione:

- Istruzioni più potenti

Stile Memoria-Memoria

- Tutti gli operandi possono essere locazioni di memoria
- Ci sono anche registri general-purpose

Esempio per A=B+C:

add A, B, C

- Esempio di processore:
  - Dec VAX:
  - Uno degli ISA più flessibili

Vantaggi:

- Facile da programmare
- Dimensione del codice piccolo
  - Meno istruzioni
  - Istruzioni più lunghe, a formato variabile

Svantaggi:

- HW complicato, molte opzioni di esecuzione per ogni istr.
- I compilatori spesso sceglievano le traduzioni più semplici, non le più veloci
- I compilatori erano portati a sotto-utilizzare i registri
  - Troppi movimenti di dati con la memoria

Evoluzione:

- Migliorare l'implementazione & i compilatori
- Semplificare il progetto

Stile linguaggio ad alto livello

- **Supporto diretto di linguaggi ad alto livello**
- **Esempio di processori:**
  - **Burroughs 5000: Algol**
  - **Diverse macchine Lisp**

Vantaggi:

- **Facile da programmare**
- **Senza compilatore**
- **Falsa credenza: più veloce ..**

Svantaggi:

- **HW complicato**
- **Economicamente non ammissibile**

  Costoso, poca domanda

Evoluzione:

- **Progetti sperimentali abortiti**

Stile Registro-Registro (Load/Store)

Tutti gli operandi
istr. aritmetiche =
registri o costanti

Molti registri

Istruzioni separate di load & store

Esempio per A=B+C:

```
load  r1, B
load  r2, C
add   r0, r1, r2
store  A, r0
```

Esempio di processori:

CDC 6600

Molto/Troppo innovativo.

Processori RISC: MIPS, SPARC

Vantaggi:

- Più semplice da progettare/ implementare
- Di solito molto veloce

  Più facile ottimizzare l'HW

  Ciclo di clock più corto

Svantaggi:

- Grandi dimensioni del codice

Evoluzione:

…

CISC vs RISC

CISC:

- Molteplici modi di indirizzamento
- Solitamente stili register-memory o memory-memory
- 2-, 3-, o più operandi
- Pochi registri
- Molte istruzioni (set complesso di istr.)
- Tipicamente istruzioni a formato variabile
- Più complessi da implementare

RISC:

- Solo alcuni modi di indirizzamento
- Solitamente, stile register-register
- 2- o 3-operandi
- Molti registri
- Poche istruzioni (set ridotto di istr.), quelle più usate nei programmi
- Tipicamente istruzioni con dimensione 1 word
- Più facile da implementare

Un po' di storia

- Negli anni'70, l'architettura dominante era quella dei cosiddetti computer microprogrammati, con stile CISC
- L'idea era quella di fornire istruzioni molto complesse che rispecchiassero i costrutti dei linguaggi ad alto livello
  - Microprogramma per realizzare tali istruzioni complesse

- Misurazioni di performance effettuate durante la metà degli anni '70 hanno dimostrato che la maggior parte delle applicazioni erano però dominate da poche semplici istruzioni

Un po' di storia

- **"Gli ingegneri hanno pensato che i computer avessero bisogno di numerose istruzioni complesse per operare in modo efficiente. È stata una idea sbagliata.**
- **Quel tipo di design ha prodotto macchine che non erano solo ornate, ma barocche, perfino rococo"**

Joel Birnbaum

(leader progetto RISC 801, Watson Research Center, IBM)

Avvento dei sistemi RISC

- La decade ’80 si apre con due progetti presso due grosse università statunitensi
  - il progetto RISC (Reduced Instruction Set Computer) coordinato dal Prof. David Patterson dell’Università della Californaia a Berkeley
  - il progetto MIPS (Million of Instructions Per Second) coordinato dal Prof. John Hennessy dell’Università di Stanford

L'intuizione dei sistemi RISC

- **L'idea fondamentale del processore RISC è quella di includere nell'ISA solo istruzioni molto semplici, frequentemente impiegate, e implementabili efficientemente**

- **Il progetto del microprocessore diventa più semplice e ottimizzabile**

- **Il task del compilatore diventa più complesso**
  - **è necessario ottimizzare la selezionare delle istruzioni RISC da impiegare per la traduzione**
  - **in ogni caso il costo del compilatore viene pagato una sola volta, e prima dell'esecuzione del programma**

Vantaggi del RISC

- **Questa semplificazione porta molti vantaggi:**
  - **lo spazio risparmiato dall'implementazione di istruzioni complesse può essere usato per memoria (cache) e altri trucchi che velocizzano l'esecuzione delle istruzioni**
  - **l'uniformità delle istruzioni permette di velocizzarne la decodifica**
  - **ma soprattutto. . .**
    - **l'uniformità e prevedibilità dei tempi di esecuzione delle istruzioni permette di eseguire più operazioni in parallelo**

L'eredità dei sistemi RISC

- La semplificazione dell'ISA ha permesso lo sviluppo di tecniche di ottimizzazione molto spinte
- Questo ha portato ad una nuova sensibilità per analisi più quantitative della performance dei sistemi
- Tutti i microprocessori attuali devono molto alla rivoluzione RISC (anche se alcuni sistemi come gli Intel x86 fanno di tutto per nasconderlo)

ISA MIPS oggi

- **Il set di istruzioni MIPS è diventato uno standard. Può essere trovato in:**
  - **chip *embedded (frigoriferi, microonde, lavastoviglie,. . .)***
  - **sistemi di telecomunicazioni (router Cisco, modem ADSL,. . .)**
  - **palmari (Windows CE)**
  - **console per videogame (Playstation, Nintendo 64, Playstation 2, Playstation Portable)**
  - **Sistemi della Silicon Graphics**
  - **ed altro (smartcard, set-top boxes, stampanti, robot,. . .)**

.. in conclusione

MIPS (R2000/R3000 RISC)

- **È la specifica architettura che useremo durante il corso, come esempio di linguaggio macchina/assembly**

- **Usando l'interprete SPIM sarà possibile far girare semplici programmi assembler e lavorarci effettivamente**