



Università degli Studi di Milano
Dipartimento di Informatica "Giovanni Degli Antoni"
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II

Laboratorio

1

Info (questo turno)

- Marco Tarini
- Mi trovate ... su google. Oppure:
- marco.tarini@unimi.it
- <http://tarini.di.unimi.it>
- Ricevimento: Martedì 14:30-17:30
- Ufficio: 4to piano, Dipartimento di Informatica, Via Celoria 18 - 20133 Milano (MI)
o mail

2

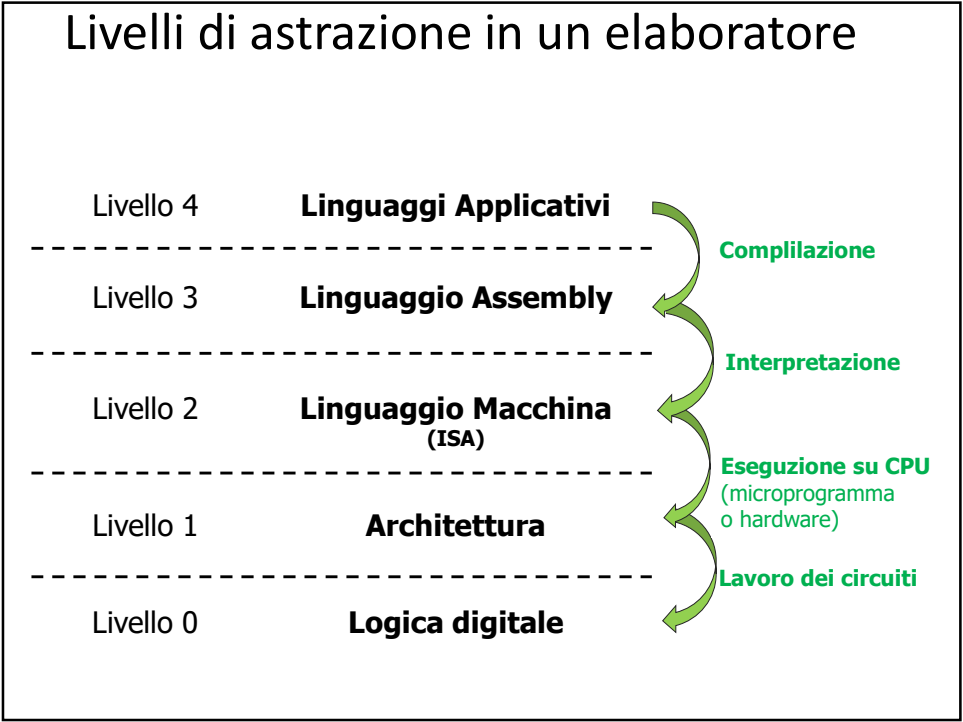
Corso di laboratorio ed esame

- 3x8 = 24 ore di lezione/esercitazione al computer
- Esame di laboratorio al PC
- Voto di Architettura = $\frac{2}{3}TEORIA + \frac{1}{3}LAB$
 - Una volta ottenuto, il voto di laboratorio ha validità di 6 mesi.
 - Una volta ottenuti entrambi i voti, l'esame viene verbalizzato.

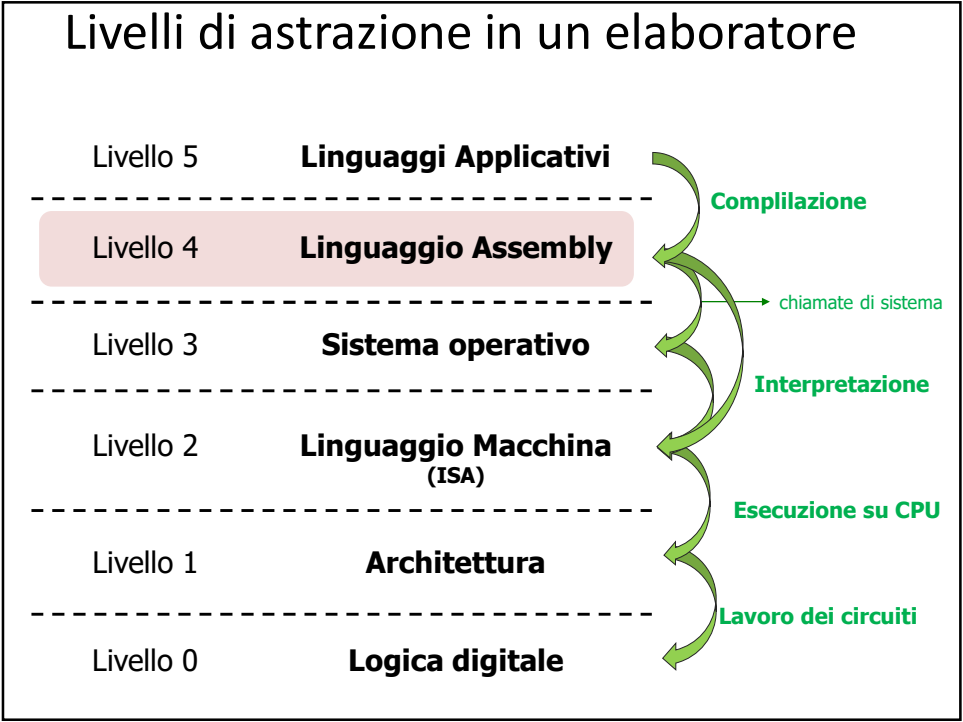
3

Progettare e assemblare software in MIPS

4



5



6

Livelli di astrazione: note

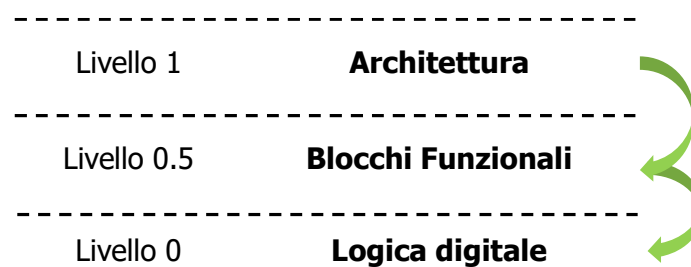
Ciascun livello consiste di:

- Un'interfaccia (verso l'alto)
 - quello che è visibile dall'esterno
 - è usata dal livello superiore
- Un'implementazione
 - come lavora internamente quell livello
 - usa l'interfaccia del livello inferiore

7

Livelli di astrazione intermedi

Livelli intermedi:

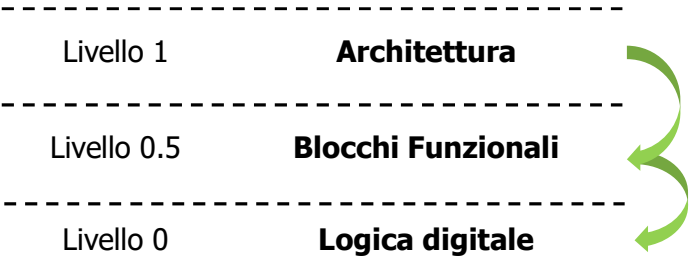


8

Livelli di astrazione intermedi

- Ciascun livello consiste di:
- Un'interfaccia
 - cos'è visibile dall'esterno
 - usata dal livello superiore
 - Un'implementazione
 - come lavora internamente
 - usa l'interfaccia del livello inferiore

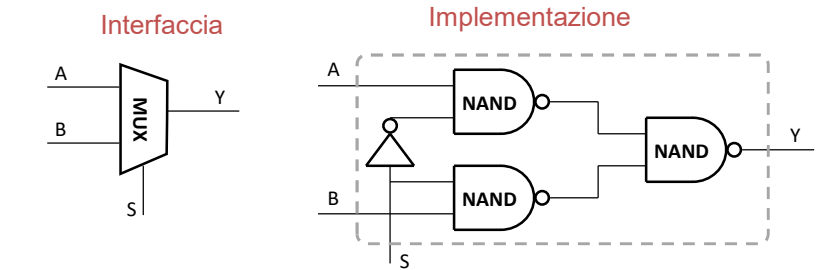
Livelli intermedi:



9

Livelli di astrazione

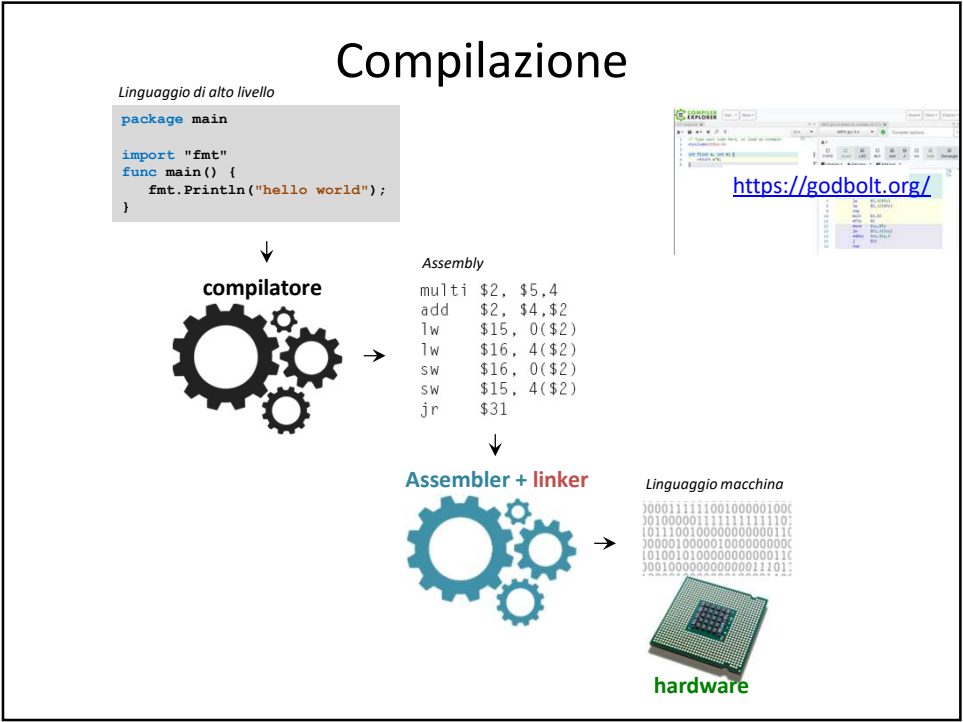
Esempio:
livello dei Blocchi Funzionali



Il livello ISA Architettura degli elaboratori

- 10 -

10



11

Linguaggio Assembly

È la rappresentazione simbolica del linguaggio macchina di un elaboratore.

```
add $t0 $s2 $s3
Binary: 00000010010100110100000000100000
Hex: 0x02534020
```

31	26 25	21 20	16 15	11 10	6 5	0
SPECIAL	\$s2	\$s3	\$t0	0	ADD	
000000	10010	10011	01000	00000	100000	
6	5	5	5	5	6	

[MIPS instruction converter](https://www.eg.bucknell.edu/~csci320/mips_web/)
https://www.eg.bucknell.edu/~csci320/mips_web/

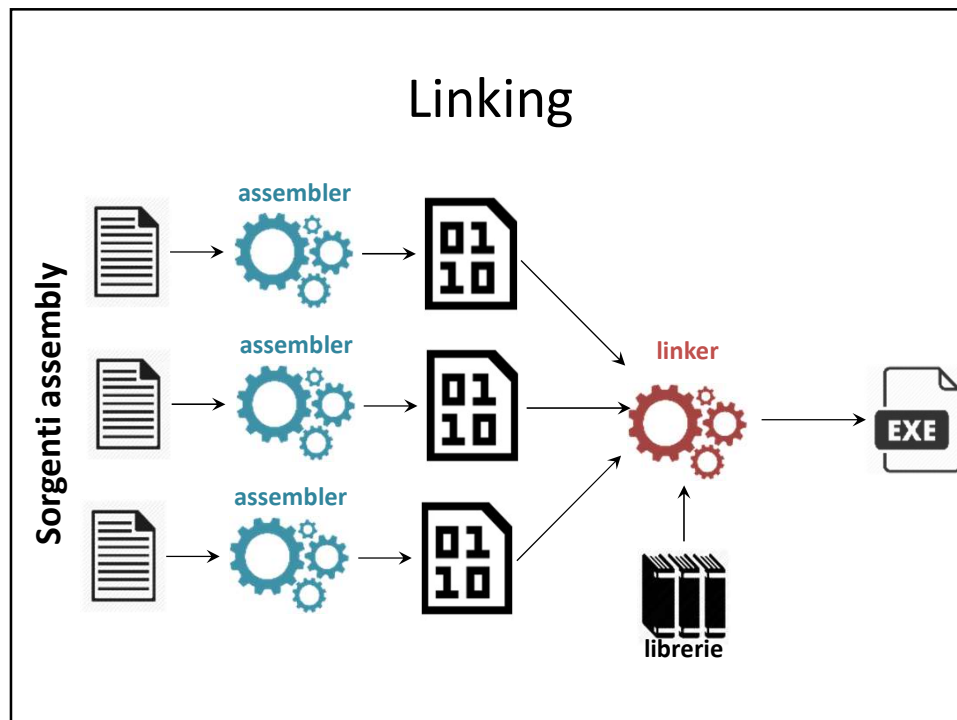
- Dà alle istruzioni una forma *human-readable* e permette di usare **label** per referenziare con un nome parole di memoria che contengono istruzioni o dati.
- Programmi coinvolti:
 - **assembler**: «traduce» le istruzioni assembly (da un **file sorgente**) nelle corrispondenti istruzioni macchina in formato binario (in un **file oggetto**);
 - **linker**: combina i files oggetto e le librerie in un **file eseguibile** dove la «destinazione» di ogni label è determinata.

12

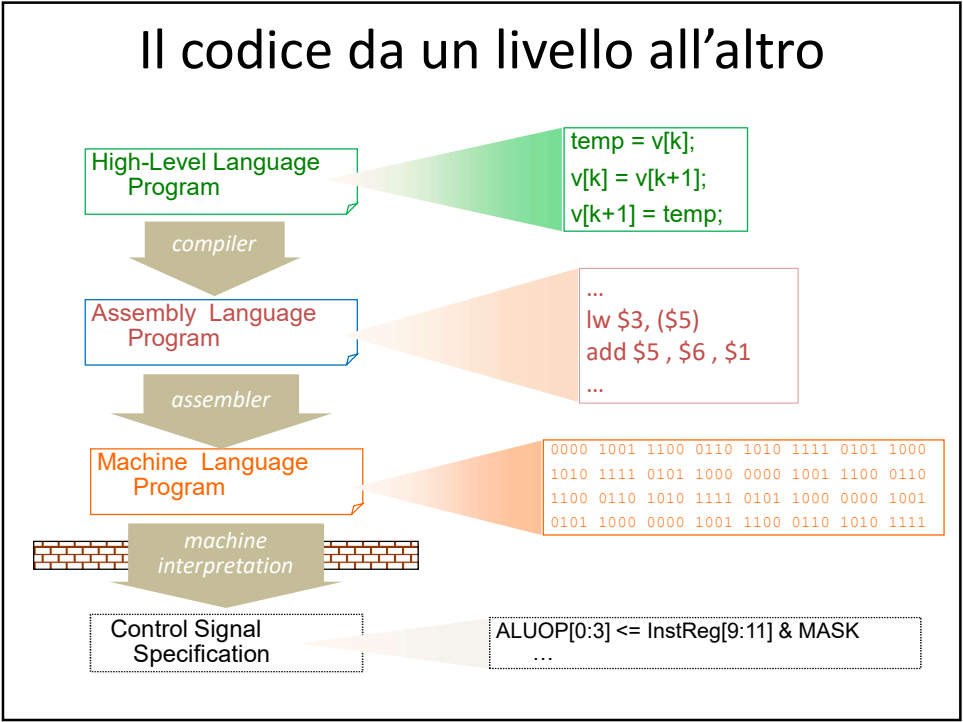
Chi scrive in codice in Assembly?

- Il codice Assembly può essere il risultato di due processi:
 - *target language* del compilatore che traduce un programma in linguaggio di alto livello (C, Pascal, ...) nell'equivalente assembly;
 - *linguaggio di programmazione* usato da un programmatore.
- Assembly è stato l'approccio principale con cui scrivere i programmi per i primi computer.
- Oggi la complessità dei programmi, la disponibilità di compilatori sempre migliori e di memoria rendono conveniente programmare in linguaggi di alto livello.
- Assembly come linguaggio di programmazione è adatto in certi casi particolari:
 - ottimizzare le performance (anche in termini di prevedibilità) e spazio occupato da un programma (ad es., sistemi embedded);
 - eredità di certi sistemi vecchi, ma ancora in uso, dove Assembly rappresenta l'unico modo conveniente per scrivere programmi;
 - rendere più efficienti certe istruzioni che hanno una semantica di basso livello.

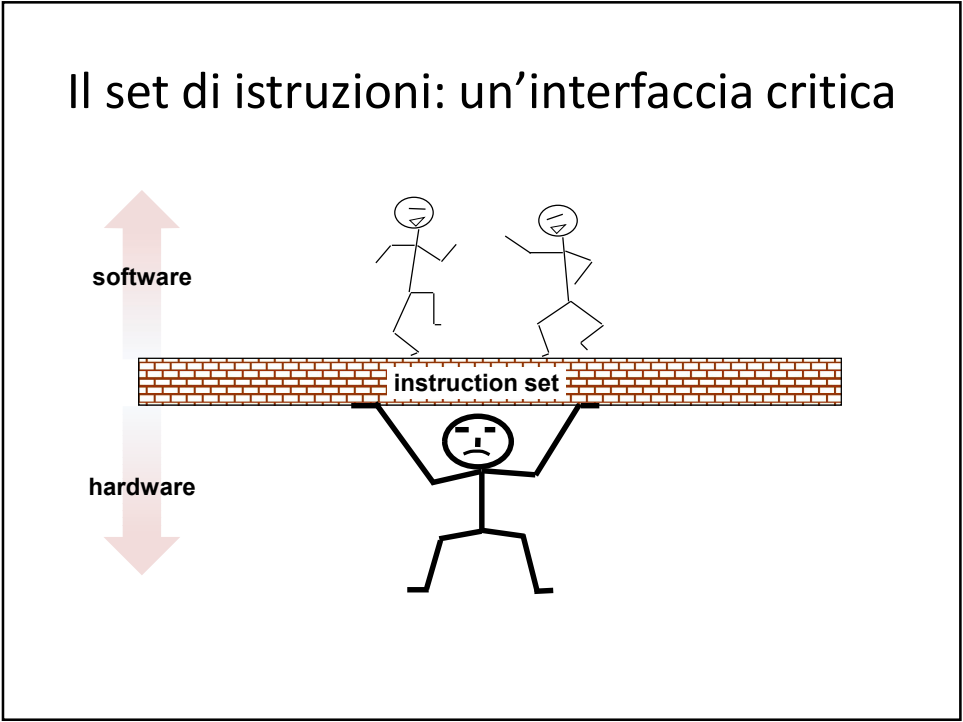
13



15



18



19

ISA: Instruction Set Architecture

- Il livello visto dal programmatore assembly o dal compilatore.
- Comprende:
 - **Instruction Set**
(quali operazioni possono essere eseguite?)
 - **Instruction Format**
(come devono essere scritte queste istruzioni? cioè la loro *sintassi*)
 - **Data Storage**
(dove sono posizionati i dati?)
 - **Addressing Mode**
(come si accede ai dati?)
 - **Exceptions**
(come vengono gestiti i casi eccezionali?)

20

Il set di istruzioni: un'interfaccia critica

- È un difficile compromesso fra:
 - massimizzare le prestazioni
 - massimizzare la semplicità di uso
 - minimizzare i costi di produzione
 - minimizzare i tempi di progettazione
- Definisce la sintassi e la semantica del linguaggio

21

Alcuni Instruction Set popolari oggi



[per i curiosi: cercare sulla Wikipedia...](#)

22

Scelta dell'Instruction Set

MIPS

- 23 -

23

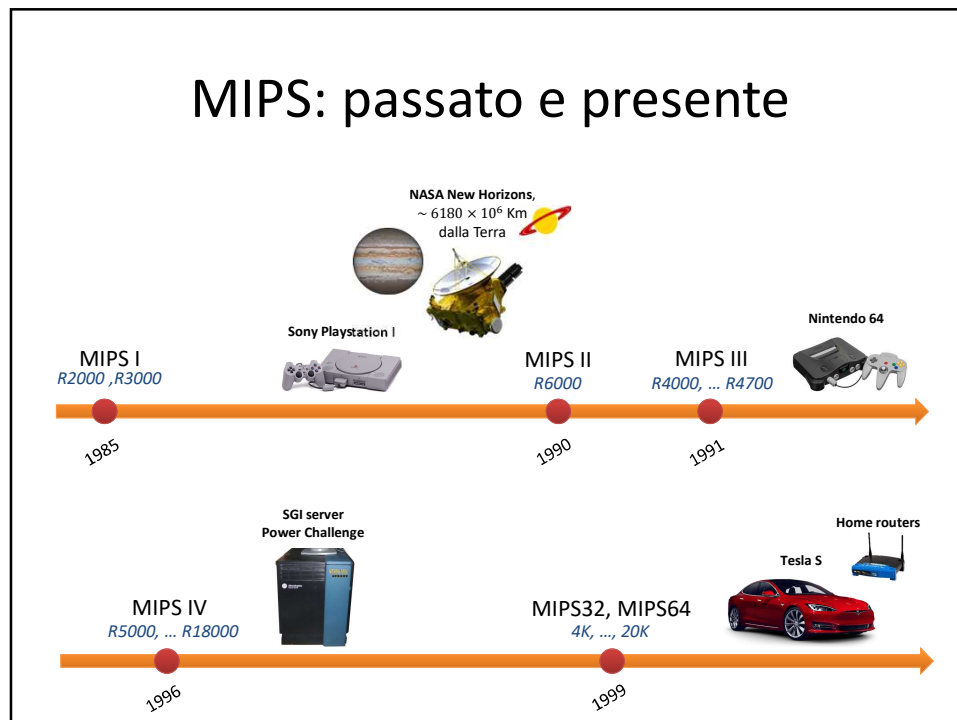
MIPS



- In questo laboratorio lavoreremo con **MIPS**
- MIPS: Multiprocessor without Interlocked Pipeline Stages → un'Instruction Set Architecture (ISA) di tipo RISC
- Nasce a metà anni '80 come architettura *general purpose*;
- Inizialmente è un progetto accademico (Stanford), poco dopo diventa commerciale
- Oggi è impiegata prevalentemente nell'ambito dei *sistemi embedded*

24

MIPS: passato e presente



25

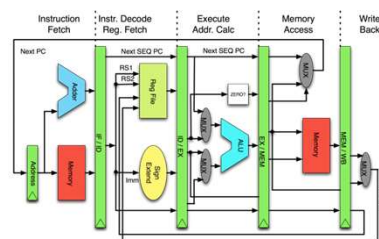
MIPS: passato e presente



26

MIPS

- La maggior parte dei corsi accademici di architetture adotta MIPS, perché?



- È una prima e lineare implementazione del concetto di pipeline
- È costruita su una semplice assunzione: ogni stadio della pipeline deve terminare in un ciclo di clock, ogni stadio non necessita di attendere il completamento degli altri (interlock)
- (Oggi l'assunzione è rilassata per avere istruzioni come moltiplicazione e divisione, ma il nome è rimasto lo stesso)

27

MIPS

- La semplicità dell'ISA emerge anche dalla quantità ridotta di *boilerplate*

“Hello world” in x86 (64 bit)

```
.file "hello_wold.c"
.section .rodata
.LC0:
.string "Hello world!"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $.LC0, %edi
call puts
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609"
.section .note.GNU-stack,"",@progbits
```

“Hello world” in MIPS (32 bit)

```
.data
hello: .asciiz "\nHello, World!\n"
.text
.globl main
main:
li $v0, 4
la $a0, hello
syscall

li $v0, 10
syscall
```

28

Richiamo di
Nozioni base di MIPS

29

Alcune caratteristiche base dell'Instruction Set «MIPS»

- Tutte le istruzioni sono codificate in **32** bit
- I registri sono costituiti da **32** bit
 - Cioé, le *parole* (word) sono di **32** bits
- Si hanno a disposizione **32** registri
 - Il registro R0 è speciale, e vale sempre 0
- Si usano operazioni con (al più) due operandi (di **32** bits)
 - I risultati delle operazioni sono parole di **32** bits
 - (eccezione: pochissime, come la moltiplicazione... producono 2 parole da **32** bits)
- Gli indirizzi di memoria sono di **32** bits
 - E in memoria si leggono o scrivono parole da **32** bits
 - Le parole sono indicizzate a livello di byte: ogni accesso quindi legge 4 bytes ($4 \times 8 = \mathbf{32}$): quello dell'indice fornito e i tre successivi

30

I registri utente in MIPS

- sono 32, numerati da 0 a 31
- denotati con il simbolo **\$**: da **\$0** a **\$31**
- possono essere utilizzati come input o output da tutte le operazioni
 - anche sia input che output di una stessa op
- Il registro **\$0** contiene sempre il valore ZERO!
 - anche se viene sovrascritto, rimane comunque zero
 - è disabilitata la scrittura a livello HW
 - lo si può chiamare anche col nome **\$zero**

33

I registri non utente


- Non sono utilizzabili (ne' lettura ne scrittura) dalle normali operazioni.
- Vengono aggiornati autmaticamente dal sistema
- Sono:
 - PC (Program Counter):
contiene l'indirizzo della prossima istruzione da eseguire
 - Hi & Lo Due registri che contengono il risultato delle operazioni di moltiplicazione e divisione (intera)

34

Richiamo: Istruzioni aritmetiche del MIPS

Comando	Sintassi (es)	Semantica (es)	Commenti
<i>add</i>	<code>add \$1,\$2,\$3</code>	$\$1 = \$2 + \$3$	operandi: 2 registri
<i>subtract</i>	<code>sub \$1,\$2,\$3</code>	$\$1 = \$2 - \$3$	operandi: 2 registri
<i>add immediate</i>	<code>addi \$1,\$2,99</code>	$\$1 = \$2 + 99$	operandi: registro e costante
<i>add unsigned</i>	<code>addu \$1,\$2,\$3</code>	$\$1 = \$2 + \$3$	operandi: 2 registri
<i>subtract unsigned</i>	<code>subu \$1,\$2,\$3</code>	$\$1 = \$2 - \$3$	operandi: 2 registri
<i>add immediate unsigned</i>	<code>addiu \$1,\$2,99</code>	$\$1 = \$2 + 99$	operandi: registro e costante
<i>multiply</i>	<code>mult \$2,\$3</code>	$Hi \mid Lo = \$2 \times \3	prodotto con segno: (risulato in 64 bit)
<i>multiply unsigned</i>	<code>multu \$2,\$3</code>	$Hi \mid Lo = \$2 \times \3	idem, ma senza segno
<i>divide</i>	<code>div \$2,\$3</code>	$Lo = \$2 \div \$3,$ $Hi = \$2 \bmod \3	Lo = quoziente Hi = resto
<i>divide unsigned</i>	<code>divu \$2,\$3</code>	$Lo = \$2 \div \$3,$ $Hi = \$2 \bmod \3	idem, ma senza segno
<i>move from Hi</i>	<code>mfhi \$1</code>	$\$1 = Hi$	Copia Hi in un registro
<i>move from Lo</i>	<code>mflo \$1</code>	$\$1 = Lo$	Copia Lo in un registro

35



Esempio di una istruzione MIPS


0000000001100101111111000001000000

Operazione fra registri	6	11	31	---	SUM
OPCODE	RS	RT	RD	---	FUNCTION

Tradotta in assembly MIPS: `ADD $31, $6, $11`

A parole: « Somma il Registro 11 e il Registro 6 e memorizza il risultato nel Registro 31 »

36



Esempio di una istruzione MIPS


0000000110000000010100000100010

Operazione fra registri	12	0	5	---	SUBTRACT
OPCODE	RS	RT	RD	---	FUNCTION

Tradotta in assembly MIPS: `SUB $5, $12, $0`

A parole: « Sottrai il Registro 0 dal Registro 12 e memorizza la differenza nel Registro 5 »

37



Una istruzione MIPS simile


00000001100100000010100000100100

Operazione fra registri	12	16	5	---	AND
OPCODE	RS	RT	RD	---	FUNCTION

Tradotta in assembly MIPS: AND \$5, \$12, \$16

A parole: « Esegui un AND bit a bit fra il registro 12 e 16. e memorizza il risultato nel registro 5»

38



Un'altra istruzione MIPS simile


000000011001000000101000000000100

Operazione fra Registri	12	16	5	---	Shift a sinistra
OPCODE	RS	RT	RD	---	FUNCTION

Tradotta in assembly MIPS: SLLV \$5, \$12, \$16

A parole: « Fai uno shift a sinistra del Registro 12 di tante cifre quanti ne indica il Registro 16. e memorizza il risultato nel registro 5»

39



Somme e somme (linguaggi a confronto)

- In MIPS (sia il linguaggio Assembly sia il Linguaggio macchina) i comandi “somma un registro ad un valore immediato” e “somma un registro ad un altro registro” sono distinti.
- Tipicamente, non è così, nei linguaggi a più alto livello, come il **Go**.

```
var a,b int
```

...

```
a = a + b
```

```
a = a + 170
```

una variabile

stesso operatore

un «literal»

```
ADD $4, $4, $9
```


```
ADDI $4, $4, 170
```

un registro

comandi diversi

un «immediate»

40



Prossimo esempio di istruzione MIPS

00100100011010110000000011110000

Add register to immediate	3	11	240
OPCODE	RS	RT	IMMEDIATE (16 bits)

Tradotta in assembly MIPS:

ADDI \$11, \$3, 240

A parole: « Somma il valore 240 al Registro 3 e memorizza il risultato nel Registro 11 »

41



Paragone con comando ADD per sommare due REGISTRI

0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Operazione fra Registri	3	11	31	---	SUM
OPCODE	RS	RT	RD	---	FUNCTION

Tradotta in assembly MIPS:


ADD\$31,\$3,\$11

A parole: «

Somma il Registro 3 e il Registro 11

e memorizza il risultato nel Registro 31 »

42



Prossimo esempio di istruzione MIPS

0	0	1	0	0	1	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Add register to immediate	3	11	240
OPCODE	RS	RT	IMMEDIATE (16 bits)

Tradotta in assembly MIPS:


ADDI\$11,\$3,240

A parole: «

Somma il valore 240 al Registro 3

e memorizza il risultato nel Registro 11 »

43



Un'istruzione MIPS simile

0011100001101011000000001110000

XOR between register to immediate	3	11	240
OPCODE	RS	RT	IMMEDIATE (16 bits)

Tradotta in assembly MIPS: XORI \$11, \$3, 240

A parole: « Fai lo XOR bit a bit fra il Registro 3 e 0..01110000 e memorizza il risultato nel Registro 11 »

44

Richiamo: Notazione MIPS

(somma e sottrazione)

- Primo argomento: risultato dell'operazione
 - sempre un registro
 - nota: eccezione per prodotto e divisione (vedi dopo)
- Secondo argomento: il primo operando
 - sempre un registro (scelta del MIPS)
- Terzo argomento: il secondo operando
 - un registro: \$... , oppure...
 - ...un valore immediate (un «literal»): un numero senza \$
 - nota: si tratta di due istruzioni diverse!

Sia in assembly MIPS, che in linguaggio macchina MIPS.
Es: add vs addi , sub vs subi
- Comandi in due versioni
 - «unsigned»: ignora overflow
 - «signed»: riporta overflow

} add vs addu , sub vs subu
dove u sta per «unsigned»
L'unica differenza è l'overflow!

45

Inizializzazione esplicita dei registri

- Come fare a caricare degli indirizzi nei registri?
- Esempio:
 - caricare in `$5` il valore `2`
- L'Instruction Set MIPS è così RISC che non ha un comando per inizializzare i registri!
- Questo perché possiamo utilizzare (fra le altre possibilità) l'istruzione «somma fra registro e valore immediato»

```
addi $5 $zero 2
```



Registro speciale,
il cui valore è sempre 0 (è il registro \$0)
(nota: scrivere su questo registro non ha effetto)

46

Inizializzazione esplicita dei registri

- Come fare a caricare dei valori nei registri?
- Esempio:
 - caricare in `$3` il valore `2`
- L'Instruction Set MIPS è così RISC che non ha un comando per inizializzare i registri! (in linguaggio macchina)
- Questo perché possiamo ri-utilizzare (fra altre possibilità) l'istruzione «somma fra registro e valore immediato»

```
addi $3 $zero 2
```



Sinonimo di registro \$0 :
un Registro speciale il cui valore è sempre 0
(nota: scrivere su questo registro non ha effetto!)

47

Inizializzazione esplicita dei registri

- La soluzione vista è però scomoda o poco leggibile:
- il linguaggio assembly MIPS ci mette quindi a disposizione una pseudoistruzione che ottiene lo stesso effetto:

«Load immediate» (una PSEUDO-istruzione)



```
li $3 2
```



TRADUZIONE AUTOMATICA
effettuata dall'assembler MIPS

```
addi $3 $0 2
```

Istruzione reale, prevista dall'ISA MIPS

48

Inizializzazione esplicita dei registri

- Problema: il campo immediate ha solo 16 bits.
- Come possiamo assegnare ad un registro un valore di 32 bits?
- Ad esempio: come assegnare al registro \$5 il valore 0x12345678 ?
- Soluzione: usare due istruzioni

«Load upper immediate» (assegna i 16 upper bits di \$t2 al campo immediate)



Qui stiamo usando un secondo registro, di appoggio



```
lui $1 0x1234
```

```
ori $5 $1 0x5678
```



«or bit a bit con immediate»

(effettua un or bit a bit fra \$t2 e il campo immediate esteso con 0)

PERCHE' FUNZIONA?

49

Inizializzazione esplicita dei registri

- Anche questa soluzione vista è però poco pratica da usare
- Soluzione: la pseudoistruzione «load immediate» viene tradotta automaticamente nelle due istruzioni, quando il suo operatore immediato supera i 16 bit

«Load immediate» (una PSEUDO-istruzione)

```
li $5 0x12345678
```



TRADUZIONE AUTOMATICA (dell'assembler)

```
lui $5 0x1234  
ori $5 $1 0x5678
```



DUE istruzioni reali

Come registro di appoggio, si usa un registro che per convenzione NON viene mai usato dai nostri programmi. E' il registro detto «ra», «riservato all'assembler», cioè il numero \$1.
(Se lo usasse, si incorre nel rischio che venga sovrascritto da qualche pseudo-istruzione)

50

Emulazione di una macchina astratta MIPS

51

Problema pratico per questo lab

- MIPS = molto adatto alla didattica dei linguaggi assembly
- Vogliamo scrivere ed eseguire programmi in MIPS
- Dove trovare una macchina MIPS?
 - un HW in grado di eseguire un programmal scritto in linguaggio macchina MIPS?

52

Emulazione (= Instruction-Level Simulation)

- Ho una macchina, con Instruction Set a , ma ho programmi eseguibili scritti in diverso Instruction Set b
- Scrivo un interprete (un programma in IS a) per un un IS b
 - Questo interprete è in grado di eseguire programmi scritti in b
- Ora ho una «*macchina virtuale*» per B!
 - posso eseguire programmi per B pur non avendo una macchina fisica costruita per capire B

53

Emulazione (= Instruction-Level Simulation)

Vantaggi:

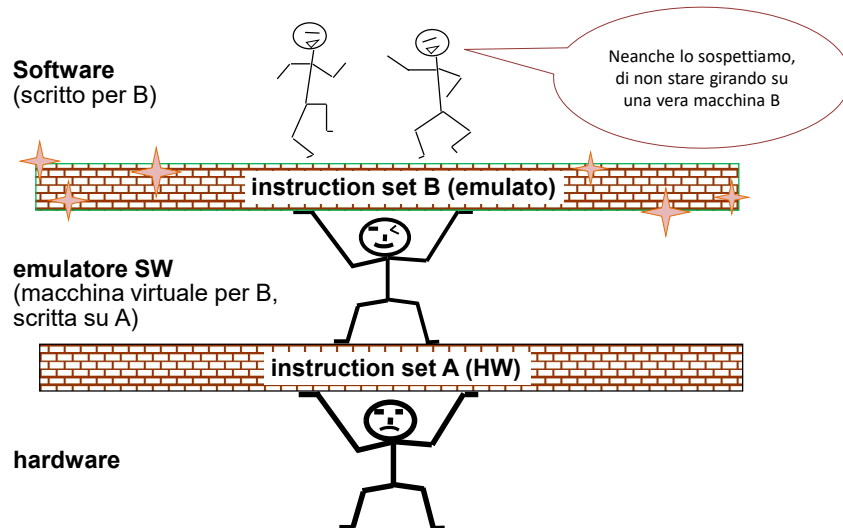
- consente di riutilizzare il programma (così com'è, alcun senza adattamento, o riscrittura) scritto per un'altra architettura B, senza avere l'HW che la esegue
 - (programma = dati + istruzioni)
- Chi scrive (o, ha scritto) il software per B non deve fare nulla di diverso dal solito

Svantaggi:

- La performance può risentirne:
Se le prestazioni di A non sono molto superiori a quelle attese per B,
il programma emulato andrà molto più lento di quello originale che giri su un HW che implementa l'IS B

54

Digressione: le gioie dell'emulazione SW



55

Emulazione: alcuni progetti interessanti

- **DosBox:**
 - emula l'IS X86, e il vecchio Sistema Operativo DOS, su varie piattaforme moderne (Windows, etc)
 - Consente di eseguire vecchi programmi DOS degli anni '80
- **WinE (Windows Emulator):**
 - emula varie piattaforme Windows per MacOS (emula il livello SO, non IS)
 - Consente di eseguire programmi Windows su Mac
- **MAME (Multi Arcade Machine Emulator)**
 - emula migliaia di IS di Arcade Machine (coin-op machine)
 - Consente di eseguire videogames arcade («da sala giochi») fine anni '70, anni '80, '90 e 2000
- **MESS (Multi Emulator Super System) --- ora parte di MAME**
 - emula centinaia di IS di home gaming console (VIC-20, C-64, ...)
 - consente di eseguire videogames da home entertainment

56



multi
arcade-machine
emulator



- Emulazione di migliaia di IS propri delle Architetture HW dedicate al gaming
 - coin-op dagli anni '70 ad oggi
- Software:
 - i programmi originali per questi IS sono recuperati là' dove hanno aspettato per decenni: in chip di ROM
 - ROM dump = scaricare il contenuto di una (qui: vecchia) ROM
- Finalità:
 - recuperare videogames, importanti pezzi della nostra storia culturale
 - è una corsa contro il tempo:
 - ROM è memoria «persistente» sì... ma entro certi limiti temporanei!
 - quasi sempre, HW capace di eseguire quel dato IS non esiste più:
 - Senza emulazione, molti video-games storici sarebbero perduti!

57



multi
arcade-machine
emulator

- I programmi da eseguire in emulazione sono estratti da un insieme (10-50) di *chip di ROM*
 - capacità complessiva: molto poco!
Per es:
pac-man (Midway, 1980): 20 KB
ghost'n'goblin (Taito, 1985): 400 KB
- Come sempre:
Programmi = DATI + ISTRUZIONI
- Nel caso dei videogame
 - DATI: includono i game asset: sprites, effetti sonori, fonts, ...
 - ISTRUZIONI: il programma vero e proprio, scritto nell'IS della rispettiva architettura dedicata



58

MARS



Missouri State
UNIVERSITY



MARS (*MIPS Assembler and Runtime Simulator*)
An IDE for MIPS Assembly Language Programming

MARS is a lightweight interactive development environment (IDE) for programming in MIPS assembly language, intended for educational-level use with Patterson and Hennessy's *Computer Organization and Design*.

- È un emulatore di una CPU che obbedisce alle convenzioni MIPS32
- Perché usare un emulatore e non la macchina vera?
 - Usiamo tutti la stessa ISA indipendentemente dal calcolatore reale.
 - Ci offre una serie di strumenti che rendono la programmazione più comoda.
 - Maschera certi aspetti reali a cui non saremmo interessati (es., delays).
- Disponibile a questo URL <http://courses.missouristate.edu/KenVollmar/MARS/index.htm>

59

MARS (interfaccia)

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui \$1,0x00001001	6: li \$t1 0x10010000
	0x00400004	0x34290000	ori \$9,\$1,0x00000000	
	0x00400008	0xad280001	sw \$9,0x00000001(\$9)	7: sw \$t0 1(\$t1)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0xbadbadad	0x08080808	0x44444444	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Nu...	Coproc 1	Coproc 0	Value
\$zero	0			0x00000000
\$at	1			0x10010000
\$v0	2			0x00000000
\$v1	3			0x00000000
\$a0	4			0x00000000
\$a1	5			0x00000000
\$a2	6			0x00000000
\$a3	7			0x00000000
\$t0	8			0x00000000
\$t1	9			0x10010000
\$t2	10			0x00000000
\$t3	11			0x00000000
\$t4	12			0x00000000
\$t5	13			0x00000000
\$t6	14			0x00000000
\$t7	15			0x00000000
\$a0	16			0x00000000
\$a1	17			0x00000000
\$a2	18			0x00000000
\$a3	19			0x00000000
\$a4	20			0x00000000
\$a5	21			0x00000000
\$a6	22			0x00000000
\$a7	23			0x00000000
\$a8	24			0x00000000
\$t9	25			0x00000000
\$t0	26			0x00000000
\$t1	27			0x00000000
\$gp	28			0x10008000
\$gp	29			0x7ffffefc
\$fp	30			0x00000000
\$sp	31			0x00000000
\$pc				0x0040000c
\$hi				0x00000000
\$lo				0x00000000

Mars Messages

Run I/O

Clear

Assemble: assembling C:\Program Files Mine\mars\mips4.asm
Assemble: operation completed successfully.
C:\mine\mips4.asm

MEMORIA TEXT

MEMORIA DATA

BANCO REGISTRI

LOG e CONSOLE (I/O)

Comproso i registri non utente

nota

nota

nota

61

MARS (interfaccia)

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui \$1,0x00001001	6: li \$t1 0x10010000
	0x00400004	0x34290000	ori \$9,\$1,0x00000000	
	0x00400008	0xad280001	sw \$9,0x00000001(\$9)	7: sw \$t0 1(\$t1)

LIVELLO PIU' BASSO

LIVELLO PIU' ALTO

1. Istruzioni scritte dall'utente, in assembly MIPS, con...

- pseudo istruzioni
- registri indicati dai sinonimi
- etichette [vedi dopo]

2. Istruzioni reali, dopo la «traduzione» di...

- Pseudo istruzioni (in istruzioni)
- Sinonimi dei registri (in numeri dei registri)
- Eitchette (in indirizzi) [vedi dopo]

3. Istruzioni in Linguaggio macchina MIPS

62

Registers			Coproc 1	Coproc 0
Name	Number	Value		
\$zero	0	0		
\$at	1	268500992		
\$v0	2	10		
\$v1	3	0		
\$a0	4	268500992		
\$a1	5	0		
\$a2	6	0		
\$a3	7	0		
\$t0	8	0		
\$t1	9	0		
\$t2	10	0		
\$t3	11	0		
\$t4	12	0		
\$t5	13	0		
\$t6	14	0		
\$t7	15	0		
\$s0	16	0		
\$s1	17	0		
\$s2	18	0		
\$s3	19	0		
\$s4	20	0		
\$s5	21	0		
\$s6	22	0		
\$s7	23	0		
\$t8	24	0		
\$t9	25	0		
\$k0	26	0		
\$k1	27	0		
\$gp	28	268468224		
\$fp	29	2147479548		
\$fp	30	0		
\$ra	31	0		
pc		4194328		
hi		0		
lo		0		

MARS (Registri)

- 32 registri a 32bit per operazioni su interi (**\$0..\$31**).
- 32 registri a 32 bit per operazioni in virgola mobile sul coprocessore 1 (**\$FP0..\$FP31**).
- registri speciali a 32bit:
 - il **Program Counter (PC)** l'indirizzo della prossima istruzione da eseguire;
 - **hi** e **lo** usati nella moltiplicazione e nella divisione;
 - **EPC, Cause, BadVAddr, Status** (coprocessore 0) vengono usati nella gestione delle eccezioni.
- I registri general-purpose sono chiamati col nome dato dalla convenzione MIPS e numerati da 0 a 31
- Il loro valore è ispezionabile nel formato esadecimale o decimale

64

Primo programma in Assembly

65

Un primo programma MIPS

- Scriviamo un programma che calcola la somma di 2+3.
- Idea:
 - Inizializzare un registro con il valore 2
 - Inizializzare un secondo registro con il valore 3
 - Effettuare la somma, ponendo il suo valore in un terzo registro

66

Un primo programma MIPS

```
li $5 2
li $6 3
add $7 $5 $6
```

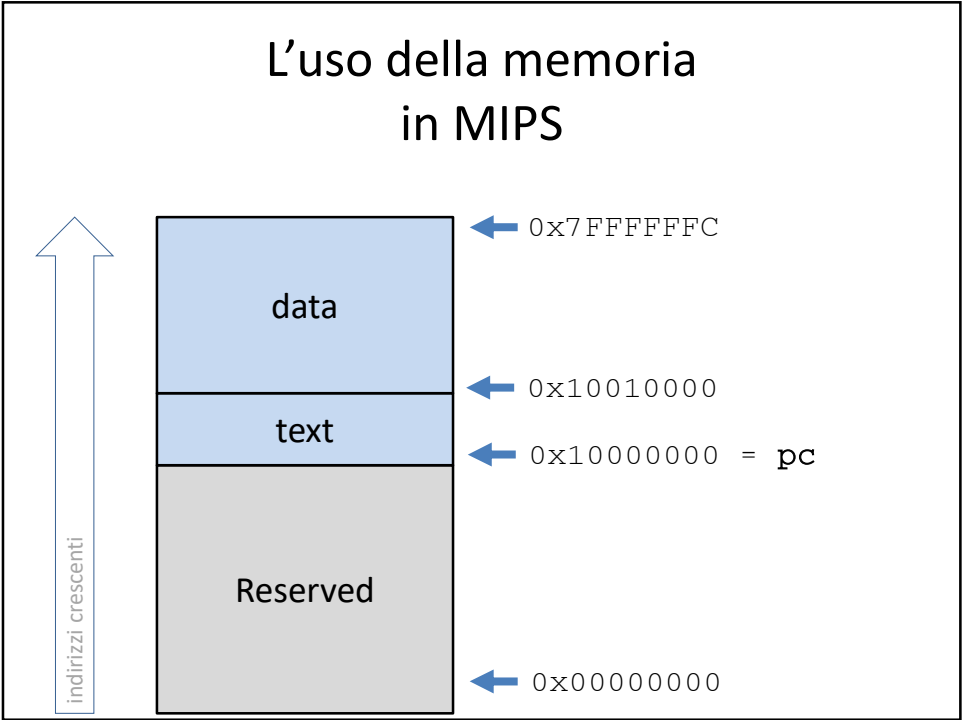
TODO list:

- Scrivere il programma SU MARS
- Compilarlo
- Osservare la traduzione delle pseudo istruzioni e dei sinonimi dei registri
- Osservare il risultato, guardando il banco dei registri.

67

Indirizzamento, lettura
e scrittura della memoria

68



69

L'uso della memoria in MIPS

- Il segmento «riservato» contiene il sistema operativo, etc
- Il segmento «testo» contiene le istruzioni del programma da eseguire
 - Il program counter viene inizializzato di default all'inizio di questo segmento
- Nel segmento «dati» contiene i dati (statici e dinamici – per ora, usiamo quelli statici).

70

Parola (word) in MIPS

e quindi anche nel emulatore MARS

dimensione di 1 parola di RAM = dimensione di 1 registro = dimensione di 1 istruzione = dimensione di 1 indirizzo RAM

1 parola = **32** bit (cioè **4** byte)

(nota: in *altri* Instruction Set, nessuna di queste ugualianze vale neccessariamente)

75

Parola (word) in MIPS

e quindi anche nel emulatore MARS

:	:
12	429
8	131
4	1
0	5000
byte address	Data

Singolo byte: un elemento di memoria spesso utile

Lo spazio degli indirizzi permette di indirizzare ognuno dei 4 bytes che compongono una parola.

Gli indirizzi di due parole consecutive differiscono di 4

76

MIPS: Letture e scritture RAM

Istruzione	Commento
sw \$3, 500(\$4)	Store word
sh \$3, 502(\$2)	Store half
sb \$2, 41(\$3)	Store byte
lw \$1, 30(\$2)	Load word
lh \$1, 40(\$3)	Load halfword
lhu \$1, 40(\$3)	Load halfword unsigned
lb \$1, 40(\$3)	Load byte
lbu \$1, 40(\$3)	Load byte unsigned
lui \$1, 40(\$3)	Load Upper Immediate (16 bits shifted left by 16)

da registro a mem (store)

da mem a registro (load)

77

MIPS: Letture e scritture RAM

- Store: da registri --> a RAM
- Load: da RAM --> a registri
- Primo operando:
il registro da / a quale operare
- Secondo operando, scritto fra parentesi (\$14):
il registro che *contiene l'indirizzo RAM*
 - ricordare: in MIPS 1 registro = 1 indirizzo memoria
- Numero prima della parentesi:
«offset» Viene aggiunto all'indirizzo
 - è opzionale: se non viene specificato, allora vale 0
 - a cosa serve? (vedi dopo)

78

Accesso alla memoria in Assembly

- Lettura dalla memoria: **Load Word**

```
lw $s1, 100($s2) # $s1 <- M[$s2+100]
```

- Scrittura verso la memoria: **Store Word**:

```
sw $s1, 100($s2) # M[$s2+100] <- $s1
```

- La memoria viene indirizzata come un vettore: indirizzo base + offset
identificano la locazione della parola da scrivere o leggere
- L'offset è in byte

79

Direttive Assembler

- Sono «indicazioni» fornite all'Assembler su come trattare il codice Assembly che le segue
- Sintassi: le direttive sono parole precedute da un punto
- In MARS tutte le direttive sono visibili sotto *help* → *directives*

80

Alcune direttive Assembler

`.data` : «quello che segue sono dati (numerici, etc) da memorizzare nel segmento **data** della RAM (nell'ordine specificato)»

- In pratica: precede i dati del nostro software.

`.text` : «quello che segue sono istruzioni da memorizzare nel segmento **text** della RAM (nell'ordine specificato)»

- In pratica: precede il codice del nostro software.

(ricorda: software = codice eseguibile + dati)

81

Esercizio:

- Scriviamo un programma MIPS che:
 - Inizializza questi tre valori word nel settore dati della memoria RAM
0xABBAABBA
0x10003456
«il word che codifica l'intero -5000»
 - Copia il primo di questi tre word nel registro \$t0

82

Soluzione

```
.data
0xABBAABBA
0x10003456
-5000
```

```
.text
li $t7 0x10010000
lw $t0 ($t7)
```

Chiedo
all'assembler
di effettuare la
conversione
dalla base 10
(comodo!)

So che il
segmento di RAM
«data»
comincia
a questo indirizzo

83

Variante:

- Scriviamo un programma MIPS che:
 - Inizializza questi tre valori word nel settore dati della memoria RAM
0xABBAABBA
0x10003456
«il word che codifica l'intero -5000»
 - Copia il **secondo** di questi tre word nel registro \$9

84

Soluzione

```
.data
0xABBAABBA
0x10003456
-5000

.text
li $7 0x10010004
lw $9 ($7)
```

← 4 bytes
dopo
(perché)

85

Soluzione 2

```
.data
0xABBAABBA
0x10003456
-5000

.text
li $7 0x10010000
addi $7 $7 4
lw $9 ($7)
```



Vado al word
successivo

86

Soluzione 3 (best)

```
.data
0xABBAABBA
0x10003456
-5000

.text
li $7 0x10010000
lw $9 4($7)
```



Uso come indirizzo il valore del registro \$7...
incrementato di 4 bytes

87

Vettori

```

.data
4 5 12 4 12 30 12

.text
li $7 0x10010000

lw $9 ($7)

lw $9 4 ($7)
lw $9 8 ($7)
lw $9 24 ($7)
```

In pratica,
un vettore V di
7 elementi,
nel segmento dati statici,
ciascuno codificato
da un word

Metto in \$t7
l'indirizzo del
primo elemento

Leggo in \$t0 il primo
elemento, cioè V[0]

V[1] (l'elemento succ.)

V[2]

V[6] (perché?
Qual'è la regola?)

88

Vettori

- Si consideri un vettore v dove ogni elemento $v[i]$ è una parola di memoria (32 bit).
- Obiettivo: leggere/scrivere $v[i]$ (elemento alla posizione i nell'array).
- Gli array sono memorizzati in modo sequenziale:
 - b : registro base di v , è anche l'indirizzo di $v[0]$;
 - l'elemento i -esimo ha indirizzo $b + 4 \cdot i$.

89

Direttive Assembler per il segmento dati

`.word` : «i valori che seguono vanno memorizzati in un word ciascuno (4 byte)»

`.half` : «i valori che seguono vanno memorizzati in un half-word ciascuno (2 byte)»

`.byte` : «i valori che seguono vanno memorizzati in un singolo byte ciascuno»

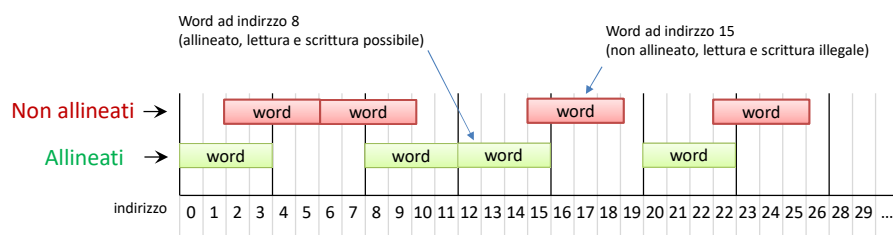
`.space N` : «lascia *N* byte non utilizzati prima del dato successivo»

(ad esempio, il programma scriverà in questo spazio prima di leggervi)

94

Allineamento dati

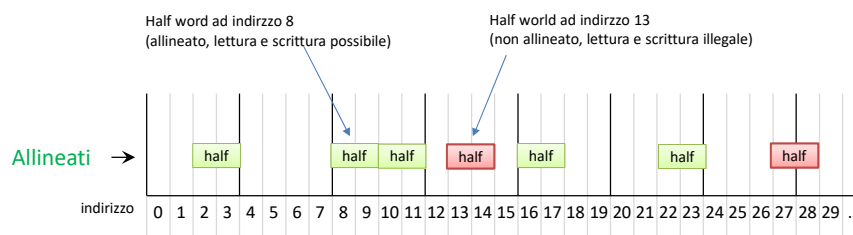
- L'accesso a memoria allineato su n byte se ogni dato di dimensione n byte comincia ad un indirizzo multiplo di n
 - con n potenza di 2 (es 2, 4, 8)
- In MIPS l'accesso a word è allineato a 4:
 - il loro indirizzo deve essere multiplo di 4
 - altrimenti: viene generato un errore a runtime (una «trap»)



95

Allineamento dati

- L'accesso a memoria allineato su n byte se ogni dato di dimensione n byte comincia ad un indirizzo multiplo di n
 - con n potenza di 2 (es 2, 4, 8)
- In MIPS l'accesso agli half-word è allineato a 2:
 - il loro indirizzo deve essere multiplo di 2
 - altrimenti: viene generato un errore a runtime (una «trap»)



96

Direttive Assembler per il segmento dati

`.align n :`

«Lascia qui un certo numero di byte vuoti,
prima del prossimo dato, per rendere la sua
posizione memoria divisibile per 2^n »

- questo lascia da un minimo di 0 fino ad un massimo $2^n - 1$ byte
- n vale 1 o 2
 - per allineare l'word usare $n=2$
 - per allineare l'byte usare $n=1$

97