



ISA e linguaggio assembler

Prof. Alberto Borghese
Dipartimento di Informatica
borgese@di.unimi.it

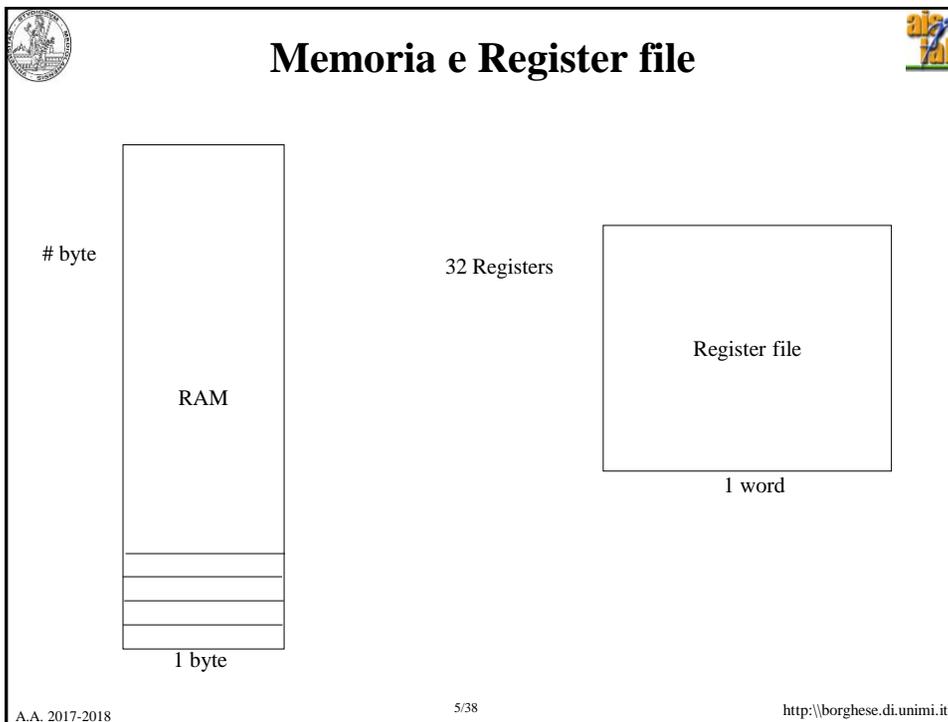
Università degli Studi di Milano

Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.



Introduzione alla CPU

- Istruzioni di accesso alla memoria
- Istruzioni di salto
- I tipi di istruzioni: il formato R



The diagram is titled 'Memoria Principale e parole' and contains a bulleted list of points. The text is as follows:

- In genere, la dimensione della parola di memoria non coincide con la dimensione dei registri contenuti nella *CPU*.
- Per ottimizzare i tempi, ad ogni trasferimento vengono trasferiti contemporaneamente un numero di byte pari o multiplo del numero di byte che costituisce la parola dell'architettura.
 - ⇒ l'operazione di *load/store* di una parola avviene in un singolo ciclo di clock del bus.
- Le parole hanno quindi generalmente indirizzo in memoria che è multiplo di 4.

⇒ Problema dell'allineamento dei dati.

The diagram is framed by a black border with logos in the top corners and text at the bottom.

A.A. 2017-2018 6/38 http://borghese.di.unimi.it/

Indirizzamento della memoria MIPS

byte

	2^{k-4}	2^{k-3}	2^{k-2}	2^{k-1}
12	32 bit			
8	32 bit			
4	32 bit			
0	32 bit	8	9	10
		4	5	6
		0	1	2
				3

A.A. 2017-2018 7/38 http://borghese.di.unimi.it/

Indirizzamento dei byte all'interno della parola

MIPS utilizza un **indirizzamento al byte**, cioè l'indice punta ad un byte di memoria, byte consecutivi hanno indirizzi consecutivi indirizzi di parole consecutive (adiacenti) differiscono di un fattore 4 (8-bit x 4 = 32-bit): ad ogni indirizzo è associato un byte.

32-bit = 1 Word

1 Byte = MSB 8-bit	1 Byte 8-bit	1 Byte 8-bit	1 Byte = LSB 8-bit
2^{24}	2^{16}	2^8	2^0

32 bits

Sign (1 bit)	Exponent 8 bits	Fraction 23 bits
--------------	--------------------	---------------------

IEEE 754 – floating point

A.A. 2017-2018 8/38 http://borghese.di.unimi.it/




Addressing Objects: Endianess

- **Big Endian:** address of most significant byte = word address
(xx00 = Big End of word)
 - IBM 360/370, Motorola 68k, MIPS, Sparc, HP
- **Little Endian:** address of least significant byte = word address
(xx00 = Little End of word)
 - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

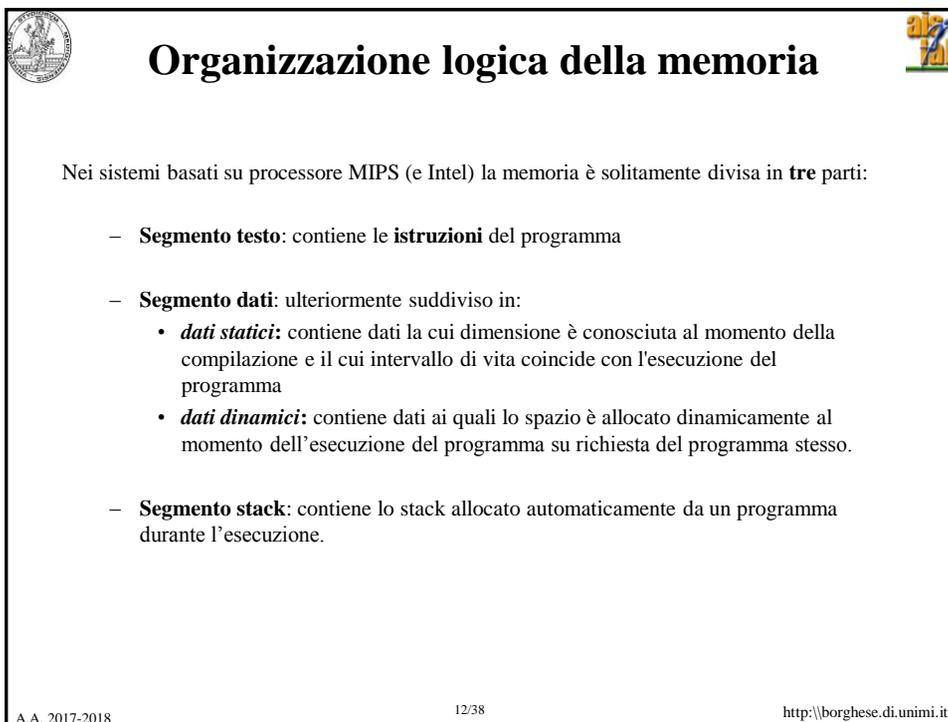
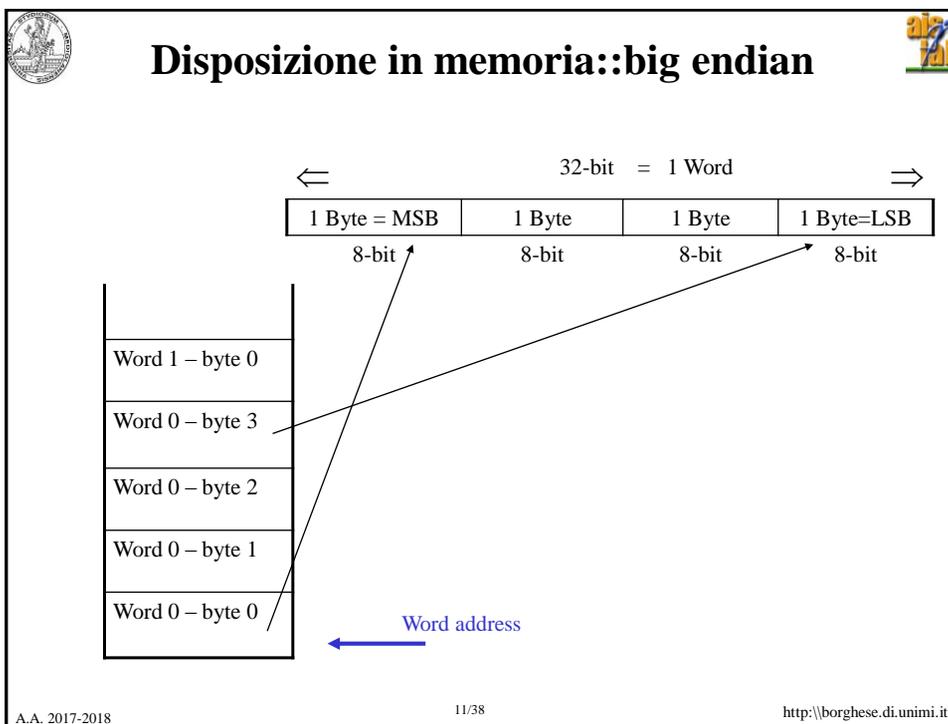
Ispirato dai “I viaggi di Gulliver” di Jonhatan Swift

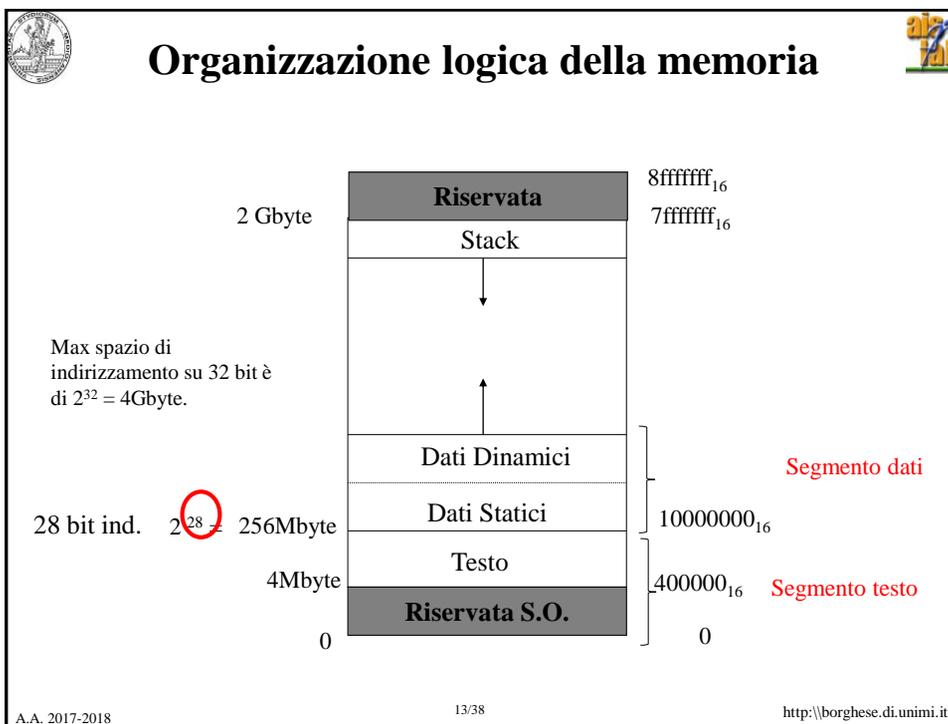
A.A. 2017-2018
9/38
<http://borghese.di.unimi.it/>




Disposizione in memoria::little endian

A.A. 2017-2018
10/38
<http://borghese.di.unimi.it/>





Istruzioni di trasferimento dati

- Gli operandi di una istruzione aritmetica devono risiedere nei registri che sono in numero limitato (32 nel MIPS). I programmi in genere richiedono un numero maggiore di variabili.
- Cosa succede ai programmi i cui dati richiedono più di 32 registri (32 variabili)?

Alcuni dati risiedono in memoria.
- La tecnica di mettere le variabili meno usate (o usate successivamente) in memoria viene chiamata **Register Spilling**.

↓

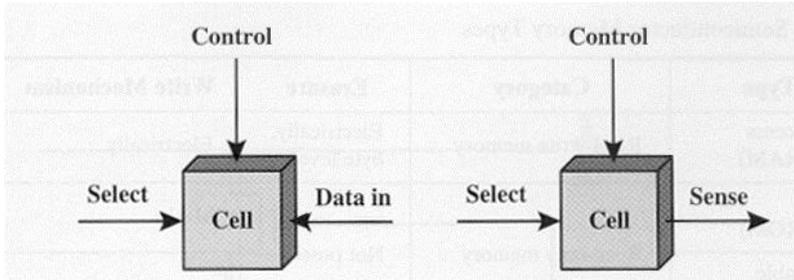
Servono istruzioni apposite per trasferire dati da memoria a registri e viceversa

A.A. 2017-2018 14/38 http://borghese.di.unimi.it/




Cella di memoria

La memoria è suddivisa in celle, ciascuna delle quali assume un valore binario stabile.
 Si può scrivere il valore 0/1 in una cella.
 Si può leggere il valore di ciascuna cella.

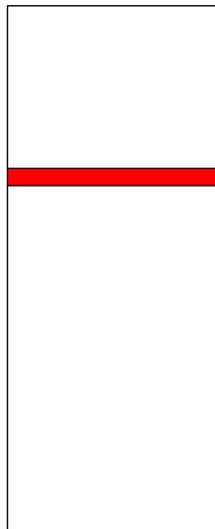


Control (lettura – scrittura)
 Select (selezione)
 Data in oppure Data out (sense)

A.A. 2017-2018
15/38
<http://borghese.di.unimi.it/>




Indirizzamento della memoria dati



Base +

Spiazzamento

MIPS fornisce due operazioni base per il trasferimento dei dati:

- lw (load word)** per trasferire una parola di memoria in un registro della CPU
- sw (store word)** per trasferire il contenuto di un registro della CPU in una parola di memoria

lw e sw richiedono come argomento l'indirizzo della locazione di memoria sulla quale devono operare

A.A. 2017-2018
16/38
<http://borghese.di.unimi.it/>

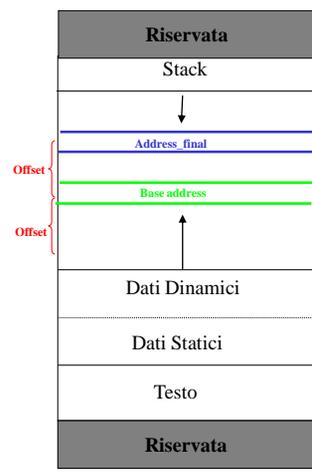


Indirizzamento della memoria dati



Base + spiazamento
Base + Offset

$Address_final = Base_address + Offset$



A.A. 2017-2018

17/38

<http://borghese.di.unimi.it/>



Istruzione *load*



- L'istruzione di *load* trasferisce una copia dei dati/istruzioni contenuti in una specifica locazione di memoria ai registri della *CPU*, lasciando inalterata la parola di memoria:

```
load LOC, r1      # r1 ← [LOC]
```

- La *CPU* invia l'indirizzo della locazione desiderata alla memoria e richiede un'operazione di lettura del suo contenuto.
- La memoria effettua la lettura dei dati memorizzati all'indirizzo specificato e li invia alla *CPU*.

A.A. 2017-2018

18/38

<http://borghese.di.unimi.it/>



Istruzione lw



- Nel MIPS, l'istruzione **lw** ha tre argomenti:
 - il *registro destinazione* in cui caricare la parola letta dalla memoria
 - una costante o *spiazzamento (offset)*
 - un registro base (*base register*) che contiene il valore dell'indirizzo base (*base address*) da sommare alla costante.
- L'indirizzo della parola di memoria da caricare nel registro destinazione è ottenuto dalla somma della costante e del contenuto del registro base.



Istruzione lw: trasferimento da memoria a registro



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



Al registro destinazione *\$s1* è assegnato il valore contenuto all'indirizzo di memoria (*\$s2 + 100*) *in byte*.



Istruzione di *store*



- L'istruzione di *store* trasferisce una parola di informazione dai registri della *CPU* in una specifica locazione di memoria, sovrascrivendo il contenuto precedente di quella locazione:

```
store r2, LOC           # [LOC] ← r2
```

- La *CPU* invia l'indirizzo della locazione desiderata alla memoria, assieme con i dati che vi devono essere scritti e richiede un'operazione di scrittura.
- La memoria effettua la scrittura dei dati all'indirizzo specificato.



Istruzione *sw*: trasferimento da registro a memoria



Possiede argomenti analoghi alla *lw*

Esempio:

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

Alla locazione di memoria di indirizzo ($\$s2 + 100$) è assegnato il valore contenuto nel registro $\$s1$



lw & sw: esempio di compilazione



Codice C: `A[12] = h + A[8];`

- Si suppone che:
 - la variabile **h** sia associata al registro **\$s2**
 - l'indirizzo del primo elemento dell'array (*base address*) sia contenuto nel registro **\$s3 (A[0])**

Codice MIPS:

```

lw $t0, 32($s3)           # $t0 ← M[ [$s3] + 32]
add $t0, $s2, $t0         # $t0 ← $s2 + $t0
sw $t0, 48($s3)          # M[ [$s3] + 48] ← $t0
    
```

A.A. 2017-2018

23/38

<http://borghese.di.unimi.it>



Memorizzazione di un vettore



- L'elemento numero **i-esimo** di un array si troverà nella locazione **br + 4 * i** dove:
 - **br** è il registro base;
 - **i** è l'indice ad alto livello;
 - il fattore **4** dipende dall'indirizzamento al byte della memoria nel MIPS

s3	A[0]	A[0]	0	1	2	3	
s3 + 4	A[1]		4	5	6	7	
s3 + 8	A[2]		Offset (A[2])	8	9	10	11
						



Array: esempio di lettura



- Sia A un array di N word. Realizziamo l'istruzione C: $g = h + A[i]$
- Si suppone che:
 - le variabili **g, h, i** siano associate rispettivamente ai registri **\$s1, \$s2, ed \$s4**
 - l'indirizzo del primo elemento dell'array (*base address*) sia contenuto nel registro **\$s3**
- L'elemento **i-esimo** dell'array si trova nella locazione di memoria di indirizzo **(\$s3 + 4*i)**.
- Caricamento dell'indirizzo di A[i] nel registro temporaneo \$t1:


```
muli $t1, $s4, 4           # $t1 ← 4 * i
add $t1, $t1, $s3         # $t1 ← add. of A[i]
                           # that is ($s3 + 4
                           * i)
```
- Per trasferire A[i] nel registro temporaneo \$t0:


```
lw $t0, 0($t1)           # $t0 ← A[i]
```
- Per sommare h e A[i] e mettere il risultato in g:


```
add $s1, $s2, $t0       # g = h + A[i]
```

A.A. 2017-2018 25/38 http://borghese.di.unimi.it/



Array: aritmetica dei puntatori



```
for (i=0; i<N; i+=2)
  g = h + A[i];
```

–l'indirizzo del primo elemento dell'array (*base address*) sia contenuto nel registro **\$s3**

First iterations:

```
lw $t0, 0($s3)
```

All the other iterations:

```
addi $s3, $s3, 8
lw $t0, 0($s3)
```

- Increment of the address of the location of A[i], inside \$s3, by adding the proper offset.

A.A. 2017-2018 26/38 http://borghese.di.unimi.it/



Istruzioni aritmetiche vs. load/store



- Le istruzioni aritmetiche leggono il contenuto di due registri (operandi) , eseguono una computazione e scrivono il risultato in un terzo registro (destinazione o risultato)
- Le operazioni di trasferimento dati leggono e scrivono un solo operando senza effettuare nessuna computazione



Introduzione alla CPU



- Istruzioni di accesso alla memoria
- **Istruzioni di salto**
- I tipi di istruzioni: il formato R

Istruzioni di salto condizionato

- Salti condizionati relativi:
 - **beq r1, r2, Etichetta** (*branch on equal*)
 - **bne r1, r2, Etichetta** (*branch on not equal*)
- Salti condizionati relativi:
 - Il flusso sequenziale di controllo cambia solo se la condizione è vera. (beq)

A.A. 2017-2018 31/38 http://borghese.di.unimi.it/

I salti incondizionati

Salti incondizionati assoluti (j, jal...) – j Etichetta
 Il salto viene sempre eseguito.
 L'indirizzo di destinazione del salto è un indirizzo assoluto di memoria.
 L'indirizzo di destinazione del salto è un numero sempre positivo.

2 Gbyte

256Mbyte

4Mbyte

0

Stack

↓

↑

Dati Dinamici

Dati Statici

Testo

Riservata S.O.

Segmento dati

Segmento testo

A.A. 2017-2018 32/38 http://borghese.di.unimi.it/



Introduzione alla CPU



- Istruzioni di accesso alla memoria
- Istruzioni di salto
- **I tipi di istruzioni: il formato R**

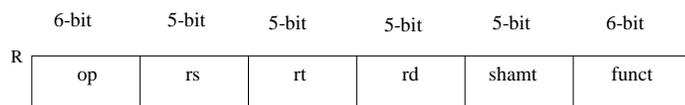


Formato delle istruzioni di tipo R



Contiene:

- Un codice operativo su 6 bit
- Un registro source, rs, su 5 bit
- Un registro target, rt, su 5 bit
- Un registro destinazione, rd, su 5 bit
- Un numero di posizioni di shift (shift amount, shamt), su 5 bit
- Un codice di funzione (cf. selettore ALU), su 6 bit

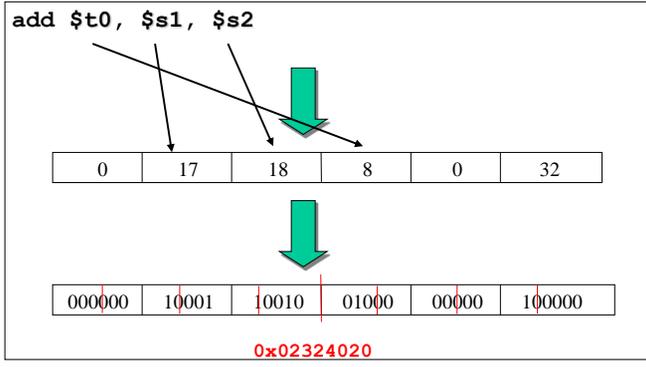




Istruzioni di tipo R: esempio



```
add $t0, $s1, $s2
```



```
000000 10001 10010 01000 00000 100000
```

0x02324020

A.A. 2017-2018
35/38
<http://borghese.di.unimi.it/>



Istruzioni di tipo R: esempi



Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
sub \$t0, \$s1, \$s2	000000	10001	10010	01000	00000	100010

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
and \$s1, \$s2, \$s3	000000	10010	10011	10001	00000	100100

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
sll \$s1, \$s2, 3	000000	X	10010	10001	00011	000000
$s1 = s2 * 2^3$ Se s2 contiene 20 (0000....0010100) => s1 conterrà = 160 (0000....0010100000)						

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
srl \$s1, \$s2, 6	000000	X	10010	10001	00110	000010
$s1 = s2 * 2^{-6}$						

A.A. 2017-2018
36/38
<http://borghese.di.unimi.it/>



Altre istruzioni di tipo R



Funzioni di salto indiretto (ad esempio per accedere alla zona di memoria istruzioni superiore ai 2GByte)



`jr rs`

(jump register con **formato R**)

- Salta all'indirizzo di memoria **assoluto** contenuto nel registro **rs** (spazio di 2^{32} byte = 4 Gbyte > intero spazio di memoria)



Introduzione alla CPU



- Istruzioni di accesso alla memoria
- Istruzioni di salto
- I tipi di istruzioni: il formato R