



# ISA e linguaggio assembler



Prof. Alberto Borghese  
Dipartimento di Informatica  
[borgnese@di.unimi.it](mailto:borgnese@di.unimi.it)

Università degli Studi di Milano  
Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.



## Sommario

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

## La memoria

- La memoria è vista come un unico grande array uni-dimensionale.
- Un **indirizzo di memoria** costituisce un **indice** all'interno dell'array.

Indirizzo  
(Byte)

$2^k-1$

...

$i$

...

1

0

←

n-bit

⇒

Parola (32 bit)  
(4 byte)

Parola  $(2^k-1)/4$

...

Parola  $i/4$

...

Parola 0

Altezza della  
memoria  
(numero di  
elementi della  
memoria)



$b_{n-1} \dots \dots \dots b_1 b_0$

Ampiezza della memoria  
(Solitamente byte)

A.A. 2020-2021

3/55

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

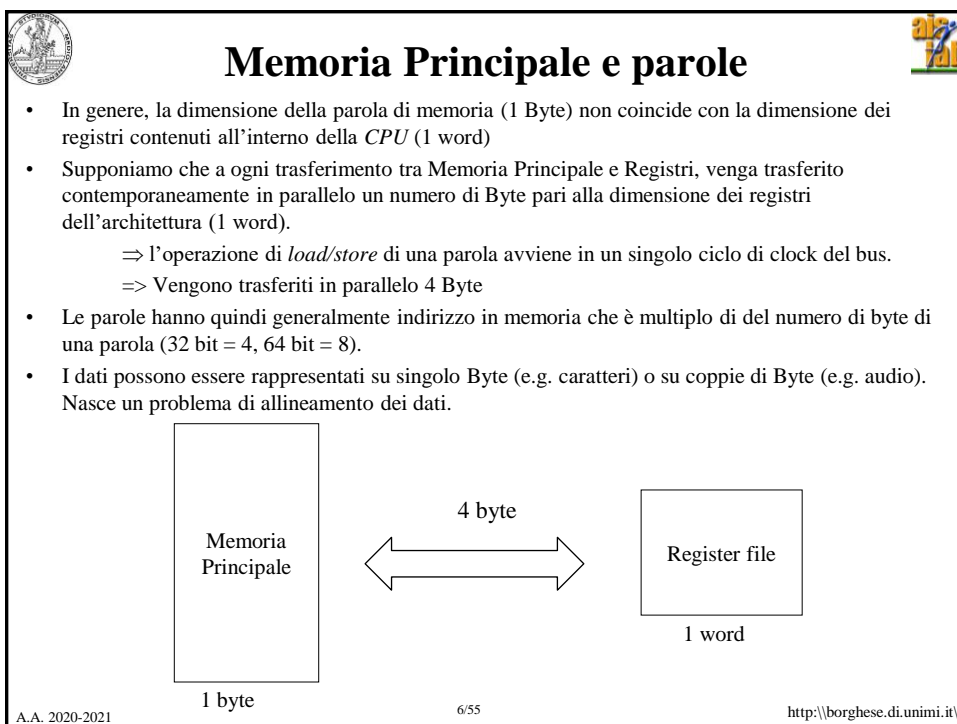
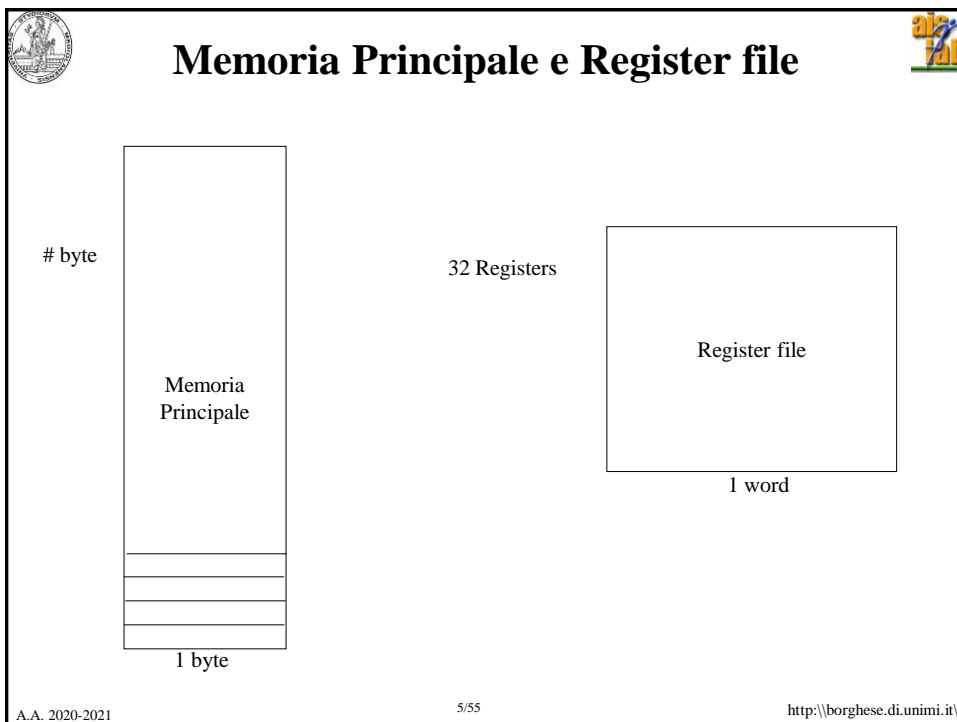
## Indirizzi nella memoria principale

- La memoria è organizzata in *parole* composte da *n-bit* che possono essere indirizzate come un unicum.
- Ogni **parola** di memoria è associata ad un **indirizzo** composto da *k-bit*.
- I  $2^k$  indirizzi costituiscono lo *spazio di indirizzamento* del calcolatore. Ad esempio un indirizzo di memoria composto da *32-bit* genera uno spazio di indirizzamento di  $2^{32}$  Byte o *4Gbyte*.

A.A. 2020-2021

4/55

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



## Organizzazione dei Byte in una parola

MIPS utilizza un **indirizzamento al byte**, cioè l'indice punta ad un byte di memoria, byte consecutivi hanno indirizzi di memoria consecutivi. Ad esempio:

32-bit = 1 Word

1 Byte = MSB (8-bit) | 1 Byte (8-bit) | 1 Byte (8-bit) | 1 Byte = LSB (8-bit)

32 bits

Single precision

8 bits (Exponent) | 23 bits (Fraction)

Sign (1 bit)

IEEE 754 – floating point

indirizzo della parola = indirizzo del LSB

A.A. 2020-2021 7/55 http://borghese.di.unimi.it/

## Addressing Objects: Endianness

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

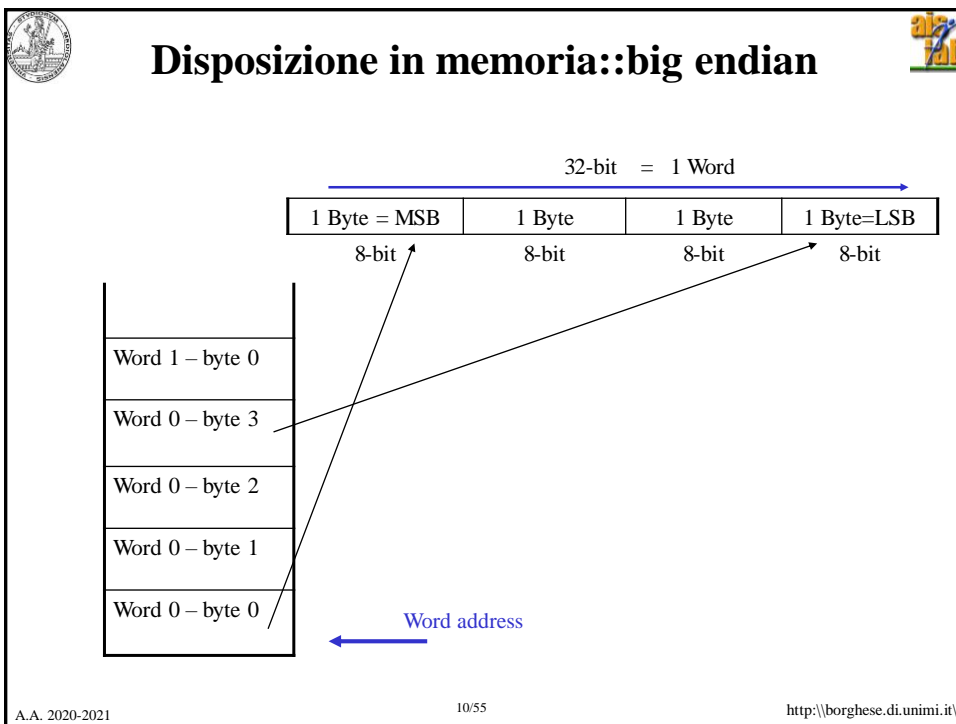
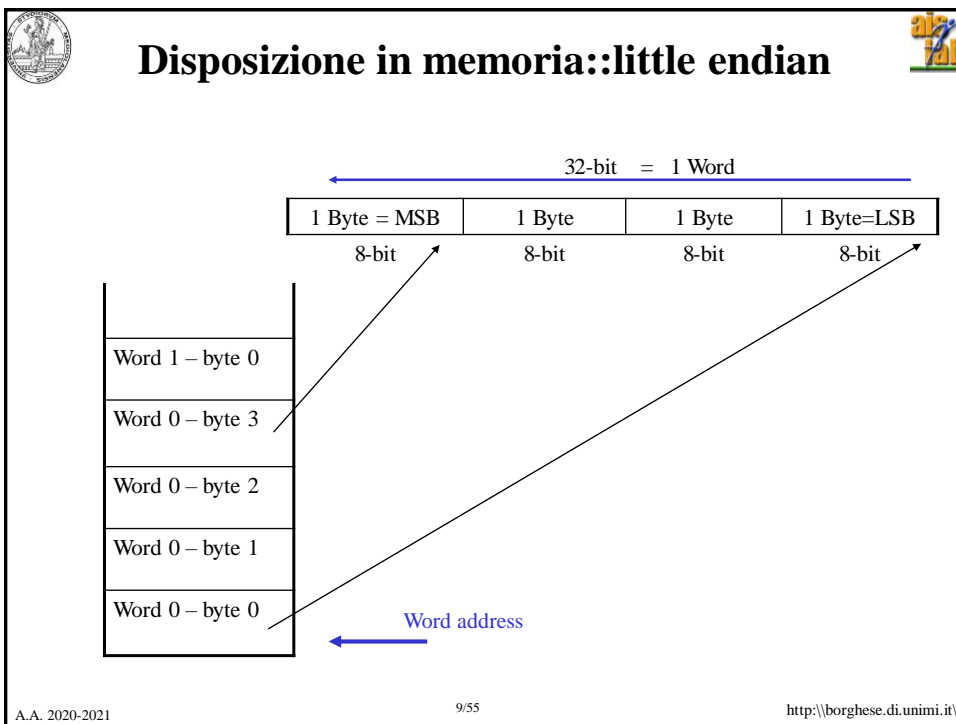
MSB = 3    2    1    0 = LSB    *little endian*

LSB = 0    1    2    3 = MSB    *big endian*

Inspirato dai “I viaggi di Gulliver” di Jonhatan Swift

Indirizzo della parola in memoria principale

A.A. 2020-2021 8/55 http://borghese.di.unimi.it/





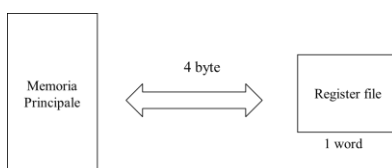
## Istruzioni di trasferimento dati



- Gli operandi di una istruzione aritmetica **devono risiedere nei registri (architettura load/store)** 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 risiederanno in memoria.
- La tecnica di trasferire 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. 2020-2021

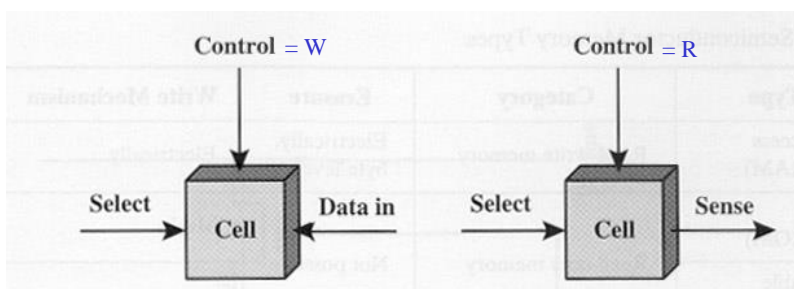
<http://borghese.di.unimi.it/>



## 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)

Base

A.A. 2020-2021

12/55

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

## Organizzazione logica della memoria

Nei sistemi basati su processore MIPS (e Intel) la memoria è solitamente divisa in **tre** parti:

- **Segmento testo:** contiene le **istruzioni** del programma
- **Segmento dati:** ulteriormente suddiviso in:
  - **dati statici:** contiene dati la cui dimensione è conosciuta al momento della compilazione e il cui intervallo di vita coincide con l'esecuzione del programma
  - **dati dinamici:** contiene dati ai quali lo spazio è allocato dinamicamente al momento dell'esecuzione del programma su richiesta del programma stesso.
- **Segmento stack:** contiene lo stack allocato automaticamente da un programma durante l'esecuzione.

4 Gbyte

2 Gbyte

256Mbyte

4Mbyte

0

Riservata S.O.  
(I/O, eccezioni...)

Stack

↓

↑

Dati Dinamici

Dati

Statici

Testo

Riservata S.O.

}

Segmento dati

Segmento testo

**Convenzione di utilizzo!**

A.A. 2020-2021 13/55 http://borghese.di.unimi.it/

## Indirizzamento della memoria dati

Indirizzo Base  +

Spiaziamento  =

← Indirizzo della memoria su cui operare

Indirizzo base su 32 bit  
Spiaziamento con un'ampiezza inferiore (**principio di località**).

Analogo all'indirizzamento degli elementi di un vettore:  
Indirizzo di Vett[i] = indirizzo di Vett[0] + i\*4

Indirizzo =

↗
Base

+

↖
Offset

A.A. 2020-2021 14/55 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. 2020-2021 15/55 http://borghese.di.unimi.it/

## Indirizzamento della memoria dati

Base + spiazamento  
Base + Offset

Address\_final = Base\_address + Offset

Nome	Numero	Utilizzo
\$zero	0	costante zero
\$at	1	riservato per l'assemblatore
\$v0-\$v1	2-3	valori di ritorno di una procedura
\$a0-\$a3	4-7	argomenti di una procedura
\$t0-\$t7	8-15	registri temporanei (non salvati)
\$s0-\$s7	16-23	registri salvati
\$t8-\$t9	24-25	registri temporanei (non salvati)
\$k0-\$k1	26-27	gestione delle eccezioni
\$gp	28	puntatore alla global area (dati)
\$sp	29	stack pointer
\$s8	30	registro salvato (fp)
\$ra	31	indirizzo di ritorno

A volte come base address si considera il registro \$gp (Global Pointer)

A.A. 2020-2021 http://borghese.di.unimi.it/





## Istruzione *load*



- L'istruzione di *load* trasferisce una copia di un dato/istruzione, contenuto in una specifica locazione di memoria, a un registro della *CPU*, lasciando inalterata la parola di memoria:

$$\text{load LOC, reg} \quad \# \text{ reg} \leftarrow [\text{LOC}]$$

- La *CPU* invia l'indirizzo della locazione desiderata alla memoria e richiede un'operazione di lettura del suo contenuto.
- La memoria effettua la lettura del dato memorizzato all'indirizzo specificato e lo invia alla *CPU*.



## Implementazione MIPS



Nel MIPS l'istruzione di caricamento di un dato dalla memoria è: "load word" (*lw*):

- Nel MIPS, l'istruzione *lw* ha tre argomenti:
  - un registro base (*base register*) che contiene il valore dell'indirizzo base (*base address*) da sommare all'offset.
  - una costante o *spiazzamento (offset)*
  - il *registro destinazione* in cui caricare la parola letta dalla memoria

$$\text{lw } \$s1, 100(\$s2) \quad \# \$s1 \leftarrow M[ [\$s2] + 100 ]$$

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



## Istruzione di *sw*



- L'istruzione di *store* trasferisce una parola di dato/istruzione da un registro della *CPU* in una specifica locazione di memoria, sovrascrivendo il contenuto precedente di quella locazione:

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

- La *CPU* invia l'indirizzo della locazione di 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.

L'istruzione MIPS per la scrittura di un registro in memoria è la *sw* (store word). Essa possiede argomenti analoghi alla *lw*

### Esempio:

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

Alla locazione di memoria di indirizzo ( $\$s2 + 100$ ) è caricato il contenuto del registro  $\$s1$

A.



## *lw* & *sw*: esempio



Elaborazione di dati di un vettore A.

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
```

## Memorizzazione di un vettore

- L'elemento **i-esimo** di un array di N elementi, si troverà nella locazione **br + 4 \* i** dove:
  - br** è il registro base;
  - i** è l'indice del vettore (e.g. codice C);
  - il fattore **4** dipende dall'indirizzamento al byte della memoria nel MIPS e si riferisce ad architetture a 32 bit

Assembler  
(puntatori)

s3

s3 + 4

s3 + 8

.....

C

A[0]
A[1]
A[2]
.....
A[N-1]

A[0]	3	2	1	0	0x40000
A[1]	7	6	5	4	0x40004
A[2]	11	10	9	8	0x40008
A[N-1]	$2^{N*4}-1$	$2^{N*4}-2$	$2^{N*4}-3$	$2^{(N-1)*4}$	

A.A. 2020-2021
21/55
<http://borghese.di.unimi.it/>

## Frammento di gestione di un vettore

- 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:
 

```

mulh $t1, $s4, 4      # $t1 ← 4 * i
add $t1, $t1, $s3     # $t1 ← address of A[i]
                     # that is ($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. 2020-2021
22/55
<http://borghese.di.unimi.it/>



## Vettori: aritmetica dei puntatori



### Codice C:

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

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

### Codice Assembler:

First iteration:

```
lw $t0, 0($s3)           # Carico l'indirizzo dell'elemento 0
                          # di A (base address)
```

All the other iterations:

```
addi $s3, $s3, 8         # Carico l'elemento successivo (+=2)
lw $t0, 0($s3)
```

...

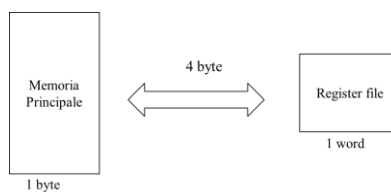
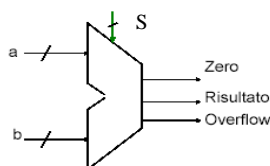
- Increment of the address of the location of A[i], inside \$s3, by adding the proper offset (here 4 Byte \* 2 elements = 8 Byte, as we supposed a 32 bit architecture)



## 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. Tuttavia utilizzano 2 registri, di cui uno viene utilizzato per costruire l'indirizzo.
- Le operazioni di trasferimento dati sono necessarie per eseguire le istruzioni aritmetiche!! (cf. Roof model)





## Sommario



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




## Istruzioni di salto in ciclo for



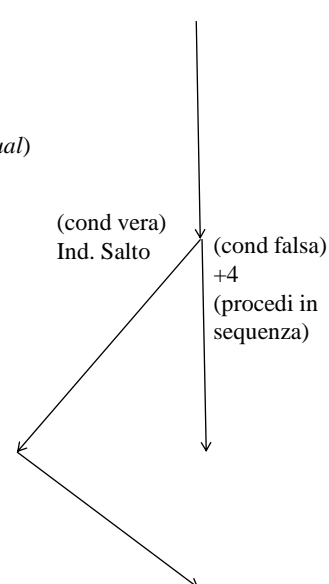
Ciclo a condizione iniziale di uscita (può essere eseguito 0 volte)

```
for (i=0; i<N; i++)           // Istruzioni di controllo
{   elem = i*N + j;           // Istruzioni aritmetico-logiche
    s = v[elem];               // Istruzioni di accesso a memoria
    z[elem] = s;               // Istruzioni di accesso a memoria
}
```

```
inizia: 0x40000 beq $t0, $s0, esci           // $s0 conteggio fine ciclo
        0x40004 ..
        ..
        ..
        0x40068 j inizia                       ; torna in ciclo
esci:
```




## 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`)

```
beq $s1, $s0, esci # if (s1 == s0) esci
bne $s1, $s0, esci # if (s0 ≠ s0) esci
```

A.A. 2020-2021 27/55 <http://borghese.di.unimi.it/>



## Condizioni di minoranza

```
blt $s1, $s0, esci # if (s1 < s0) esci
```

**bgt è una pseudo-istruzione:**

- Non fa parte dell'ISA
- E' un'istruzione molto utilizzata
- Equivale a due o più istruzioni dell'ISA

```
slt $t0, $s1, $s0 # if (s1 < s0) t0 = 1
bne $t0, $zero, esci # if (t0 ≠ 0) esci
```

A.A. 2020-2021 28/55 <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.

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

## Salti più ampi

`j` Etichetta  
 ...

Etichetta: `jr $s1`

Jump Register salta all'indirizzo contenuto in `$s1`, su 32 bit. Copre quindi tutta la memoria.

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



## Sommario



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




## Codifica delle istruzioni




- Tutte le istruzioni MIPS hanno la **stessa** dimensione (**32 bit**) – **Architettura RISC**.
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
  - il tipo di istruzione è riconosciuto in base al valore di alcuni bit (**6 bit**) più significativi (**codice operativo - OPCODE**)
- Le istruzioni MIPS sono di **3** tipi (formati):
  - **Tipo R (register)** – **Lavorano prevalentemente su 3 registri.**
    - Istruzioni aritmetico-logiche.
  - **Tipo I (immediate)** – **Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
    - Istruzioni di accesso alla memoria o operazioni con una costante.
  - **Tipo J (jump)** – **Lavora senza registri: codice operativo + indirizzo di salto.**
    - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	Indirizzo / costante		
J	op	Indirizzo / costante				





## Formato delle istruzioni di tipo R



Contiene:


- Un codice operativo su 6 bit
- Un registro source, rs, su 5 bit (32 registri)
- Un registro target, rt, su 5 bit (32 registri)
- Un registro destinazione, rd, su 5 bit (32 registri)
- Un numero di posizioni di shift (shift amount, shamt), su 5 bit
- Un codice di funzione (cf. selettore ALU), su 6 bit

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
-------	-------	-------	-------	-------	-------


R

op	rs	rt	rd	shamt	funct
----	----	----	----	-------	-------

A.A. 2020-2021
33/55
http://borghese.di.unimi.it/

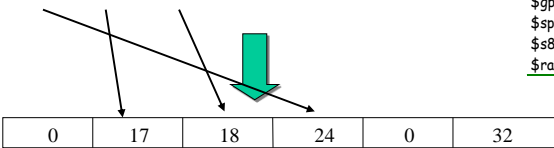


## Istruzioni di tipo R: esempio

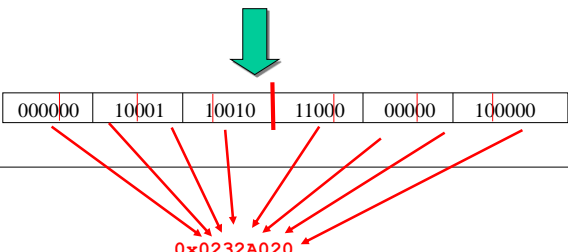


Nome	Numero	Utilizzo
\$zero	0	costante zero
\$at	1	riservato per l'assemblatore
\$v0-\$v1	2-3	valori di ritorno di una procedura
\$a0-\$a3	4-7	argomenti di una procedura
\$t0-\$t7	8-15	registri temporanei (non salvati)
\$s0-\$s7	16-23	registri salvati
\$t8-\$t9	24-25	registri temporanei (non salvati)
\$k0-\$k1	26-27	gestione delle eccezioni
\$gp	28	puntatore alla global area (dati)
\$sp	29	stack pointer
\$s8	30	registro salvato (fp)
\$ra	31	indirizzo di ritorno

**add \$t8, \$s1, \$s2**



0	17	18	24	0	32
---	----	----	----	---	----



0x0232A020

A.A. 2020-2021
34/55
http://borghese.di.unimi.it/



## Istruzioni di tipo R: esempi



Nome campo	Op	Rs	rt	rd	Shamt	Func
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<b>add \$t8, \$s1, \$s2</b>	000000	10001	10010	11000	00000	100010

Nome campo	op	rs	rt	rd	shamt	func
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<b>sub \$t8, \$s1, \$s2</b>	000000	10001	10010	11000	00000	100010

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

Nome campo	op	rs	rt	rd	shamt	func
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<b>sll \$s1, \$s2, 3</b>	000000	X	10010	10001	00011	000000

$s1 = s2 * 2^3$  Se  $s2$  contiene 20 (0000.....0010100) =>  $s1$  conterrà =  $20 * 2^3 = 160$  (0000.....0010100000)

Nome campo	op	rs	rt	rd	shamt	func
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<b>srl \$s1, \$s2, 6</b>	000000	X	10010	10001	00110	000010

$s1 = s2 * 2^{-6}$



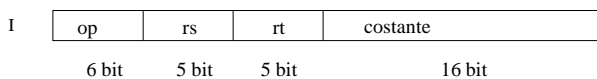
## Sommario



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



## Formato istruzioni di tipo I



- In questo caso, i campi hanno il seguente significato:
  - op** identifica il tipo di istruzione;
  - rs** indica il registro sorgente. Nel caso di una lw contiene il registro base;
  - rt** indica il registro target. Nel caso di una lw, contiene il registro destinazione dell'istruzione di caricamento;
  - costante**. Nel caso di una lw riporta lo spiazzamento (offset).

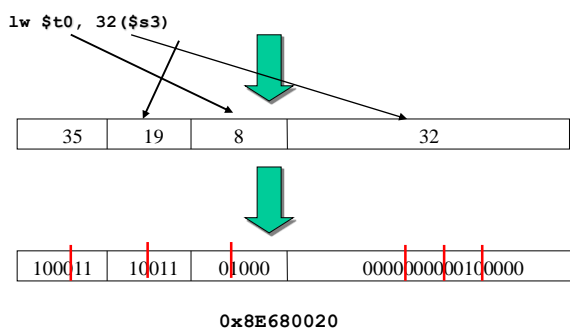


## Istruzioni di tipo I: accesso a memoria



Con questo formato una istruzione **lw** (**sw**) può indirizzare byte nell'intervallo -  $2^{15}$  (-32K) ÷  $+2^{15}-1$  (32K -1) **rispetto all'indirizzo base:**

$$\text{indirizzo} = \text{indirizzo\_base} + \text{offset} (= \text{costante})$$



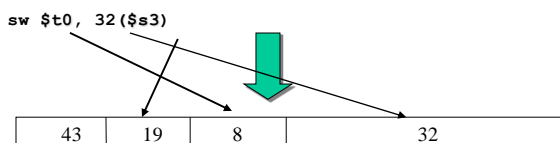


## Istruzioni di tipo I: accesso memoria



Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>lw \$t0, 32 (\$s3)</code>	100011	10011	01000	0000 0000 0010 0000

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>sw \$t0, 32 (\$s3)</code>	101011	10011	01000	0000 0000 0010 0000



Differenza di 1 bit -> cambia la direzione del trasferimento con la memoria.



## Istruzioni di tipo I: aritmetico-logiche




Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>addi \$t0, \$s3, 64</code>	001000	10011	01000	0000 0000 0100 0000

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>andi \$t0, \$s3, 64</code>	001100	10011	01000	0000 0000 0100 0000


Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>slti \$t0, \$s2, 8</code>	001010	10010	01000	0000 0000 0000 1000

# \$t0 = 1 if \$s2 < 8

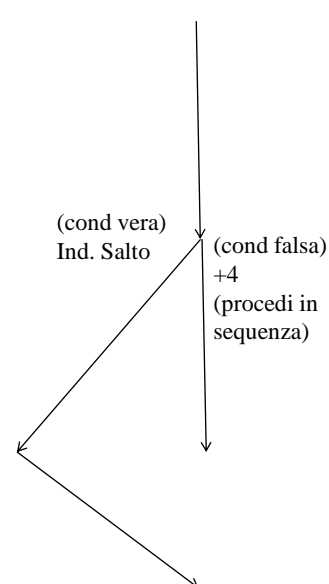
NB ruolo del registro target: nelle istruzioni di `addi` e di `lw` rappresenta **dove nel register file vado a scrivere**. Nelle istruzioni di tipo R e di `sw` contiene **uno dei dati in ingresso**.



## Istruzioni di tipo I: salto condizionato




- Salti condizionati relativi:
  - `beq r1, r2, L1` (*branch on equal*)
  - `bne r1, r2, L1` (*branch on not equal*)
- Salti condizionati relativi:
  - Il flusso sequenziale di controllo cambia solo se la condizione è vera (`beq`)
  - Il calcolo del valore dell'etichetta **L1 (indirizzo di destinazione del salto)** avviene a partire dal Program Counter (PC).
  - Indirizzamento del tipo Base (PC) + Spiazzamento.




A.A. 2020-2021

41/55

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

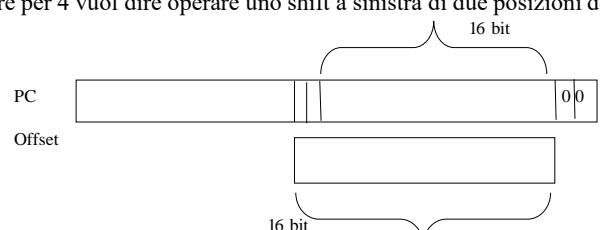


## Allargamento dello spazio di indirizzamento relativo



0000	0	0
0100	1	4
1000	2	8
1100	3	12

Gli offset in Byte avranno sempre **00** come ultimi 2 bit perché gli indirizzi sono multipli di 4.  
**Calcolo lo spiazzamento in numero di parole invece che di Byte.**  
 Considero 64Kword (64K istruzioni) invece di 64Kbyte. Lo spazio indirizzabile all'interno del segmento di testo è di 64Kword \* 4 = 256Kbyte.  
 Moltiplicare per 4 vuol dire operare uno shift a sinistra di due posizioni dell'offset



La costante su 16 bit, contenuta nel codice, rappresenta l'offset in termini di numero di istruzioni

A.A. 2020-2021

42/55

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



## Calcolo dell'indirizzo di salto




Repeat		0x400		addi \$t1, \$zero, 10 # N=10
		0x404		addi \$t0,\$zero,0 # i =0;
{ ....		0x408	loop:	addi \$t0, \$t0,1 # i++
		0x40C		....
		0x420		bne \$t0, \$t1, loop
} until (i == N)				

Quanto vale il campo costante da inserire nella stringa dell'istruzione bne?


A.A. 2020-2021

43/55

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



## Calcolo dell'indirizzo di salto



Ind Dec	Ind Exadec	Istruzione	# istruzione
400	0x400	addi \$t1, \$zero, 10 # N=10	1
404	0x404	addi \$t0,\$zero,0 # i =0;	2
408	0x408	loop: addi \$t0, \$t0,1 # i++	3
412	0x40C	....	4
432	0x424	bne \$t0, \$t1, loop	
436	0x428		9

L'indirizzo di destinazione è 0x408 (indirizzo dell'etichetta loop)

Lo spiazzamento del salto in byte è pari a:  $(0x408 - 0x424) = (408-436) = -28$  Byte

Lo spiazzamento del salto in numero di istruzioni è pari a -7 istruzioni

Prova: Indirizzo di salto = Indirizzo PC+4 + Offset (#istruzioni) \* 4

loop = 408 → costante = 436 + (-7) \* 4

5	8	9	-7 = 1111 1111 1111 1001
---	---	---	--------------------------

A.A. 2020-2021

44/55

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



## Istruzioni di tipo I - Branch



Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<b>beq \$s1, \$s2, L1</b>	000100	10001	10010	0000 0000 0001 1001

L1 = PC+4 + 100 byte Codifica su 18 bit: (00) 000 0000 0001 1001(00) in binario.

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<b>beq \$s1, \$s2, L1</b>	000100	10001	10010	1111 1111 1110 0111(00)

L1 = PC+4 -100 byte Codifica su 18 bit: (11)111 1111 1110 0111(00) in binario.



## Osservazione



Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<b>beq \$s1, \$s2, L1</b>	000100	10001	10010	0000 0000 0001 1000

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<b>lw \$s2, 24(\$s1)</b>	100011	10001	10010	0000 0000 0001 1000

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<b>addi \$s2, \$s1, 24</b>	001000	10001	10010	0000 0000 0001 1000

Istruzioni molto diverse possono distare pochi bit una dall'altra.



## Formato R ed operazioni logico-matematiche



Non tutte le operazioni logico-matematiche, sono di tipo R.

Le operazioni logico-matematiche di tipo R hanno codice operativo 0.

Non tutte le operazioni con codice operativo 0 sono logico-matematiche (ad esempio ci sono le istruzioni di *jr, syscall...*).

Occorre distinguere il funzionamento dell'istruzione elementare dalla sua codifica.

- Codifiche simili (e.g. Tipo R) possono essere condivise da istruzioni di tipo diverso (e.g. aritmetico-logiche, salto).
- Codifiche diverse (e.g. Tipo I e Tipo R) possono essere condivise da istruzioni dello stesso tipo (e.g. *add* ed *addi*)

Non c'è corrispondenza 1 a 1, tra tipi strutturali e tipi funzionali.



## Sommario



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



## Formato istruzioni di tipo J

- E' il formato usato per le istruzioni di salto incondizionato (*jump*):

op	indirizzo
6 bit	26 bit

- In questo caso, i campi hanno il seguente significato:
  - **op** indica il tipo di operazione;
  - **indirizzo** (composto da **26-bit**) riporta una parte (26 bit su 32) dell'indirizzo **assoluto** di destinazione del salto.
- I 26-bit del campo **indirizzo** rappresentano un indirizzo di parola (**word address**)  $2^{26}$  parole = 256 Mbyte (segmento testo).

PC

		00
4 bit (invariati)	26 bit	2 bit

$0 \leq \text{indirizzo} < 2^{28} = 256 \text{ Mbyte (segmento testo!)}$

A.A. 2020-2021 49/55 http://borghese.di.unimi.it/

## Organizzazione logica della memoria

Max spazio di indirizzamento su 32 bit è di  $2^{32} = 4\text{Gbyte}$ .

28 bit ind.  $2^{28} = 256\text{Mbyte}$

Riservata	8fffffff <sub>16</sub>
Stack	7fffffff <sub>16</sub>
↓	
↑	
Dati Dinamici	
Dati Statici	10000000 <sub>16</sub>
Testo	400000 <sub>16</sub>
Riservata S.O.	0

} Segmento dati

} Segmento testo

A.A. 2020-2021 50/55 http://borghese.di.unimi.it/

## Accesso alla memoria

jr \$s1

Stack

↓

↑

PC (31:28)	Indirizzo	Memoria
0000	0	Segmento testo
0001	256 Mbyte	Segmento dati
0010	512 Mbyte	Segmento dati
0011	768 Mbyte	Segmento dati
0100	1 GByte	Segmento dati
0101	1,256 GByte	Segmento dati
0110	1,512 GByte	Segmento dati
0111	1,768 GByte	Segmento dati
1000	2 Gbyte	Segmento Kernel
1001	2,256 Gbyte	Segmento Kernel
.....		

Con una jr \$r1 si può accedere a tutte le celle di memoria direttamente.

A.A. 2020-2021 51/55 http://borghese.di.unimi.it/

## Codifica delle istruzioni

- Tutte le istruzioni MIPS hanno la **stessa** dimensione (**32 bit**) – **Architettura RISC**.
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
  - il tipo di istruzione è riconosciuto in base al valore di alcuni bit (**6 bit**) più significativi (**codice operativo - OPCODE**)
- Le istruzioni MIPS sono di **3** tipi (formati):
  - Tipo R (register)** – **Lavorano su 3 registri**.
    - Istruzioni aritmetico-logiche.
  - Tipo I (immediate)** – **Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante**.
    - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
  - Tipo J (jump)** – **Lavora senza registri: codice operativo + indirizzo di salto**.
    - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	Indirizzo / costante		
J	op	Indirizzo / costante				

A.A. 2020-2021 52/55 http://borghese.di.unimi.it/





## Sommario



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