


ISA e linguaggio assembler

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

Università degli Studi di Milano

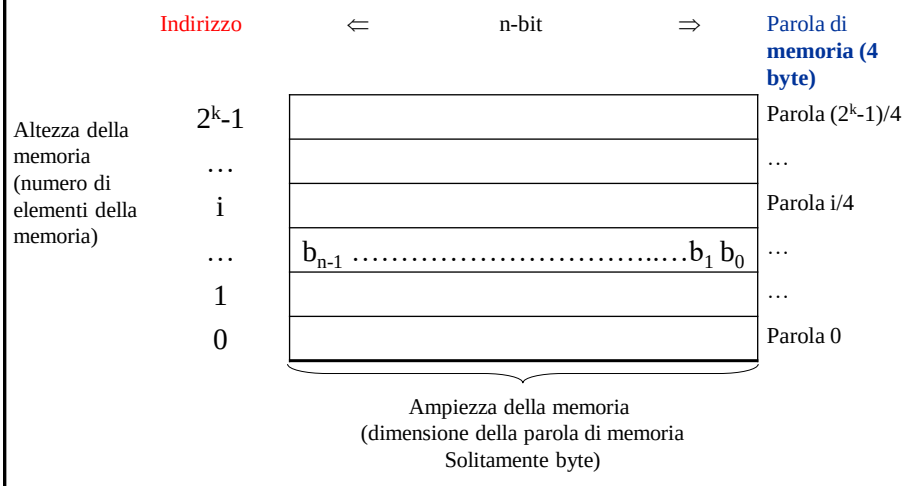
Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.


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

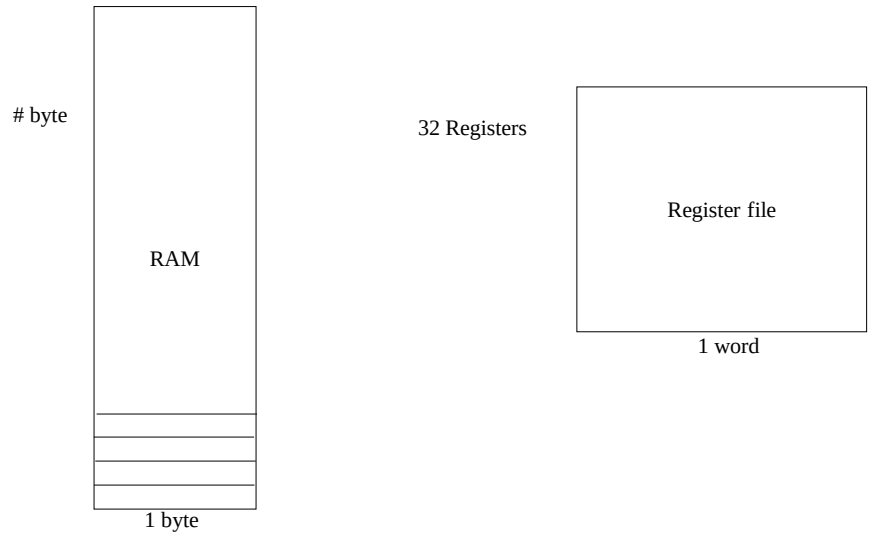
- La memoria è vista come un unico grande array uni-dimensionale.
- Un **indirizzo di memoria** costituisce un indice all'interno dell'array.



Indirizzi nella memoria principale

- La memoria è organizzata in *parole* composte da *n-bit* che possono essere indirizzati separatamente.
- 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 composto da *32-bit* genera uno spazio di indirizzamento di 2^{32} o *4Gbyte*.

Memoria e Register file



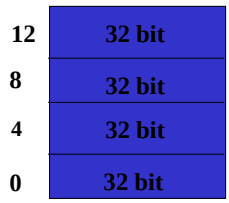
Memoria Principale e parole

- 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.

Indirizzamento della memoria MIPS

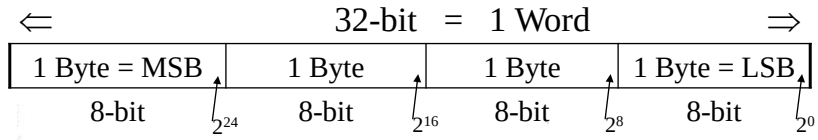


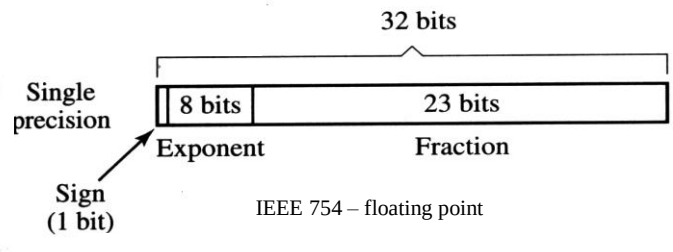
byte

2^k-4	2^k-3	2^k-2	2^k-1
8	9	10	11
4	5	6	7
0	1	2	3

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.

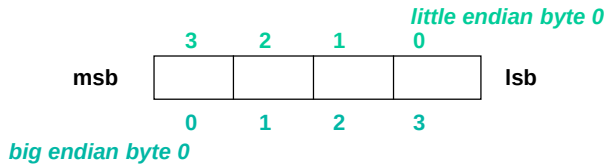




IEEE 754 – floating point

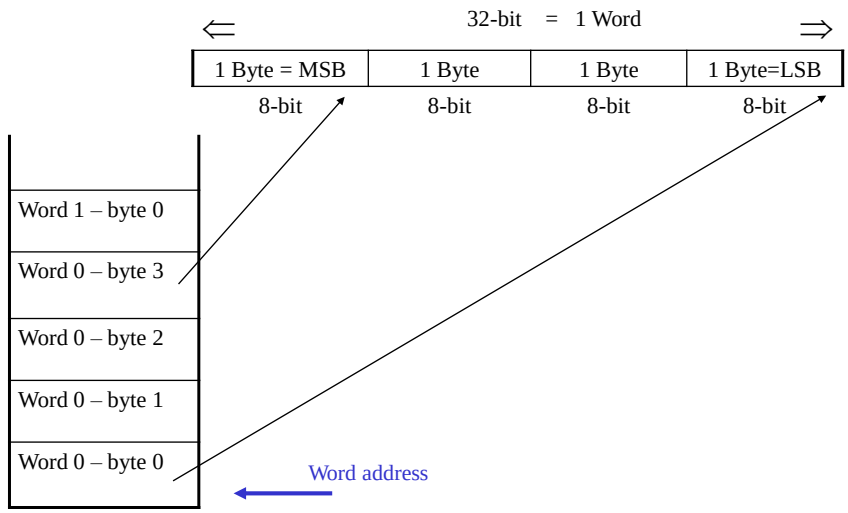
Addressing Objects: Endianess

- 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)

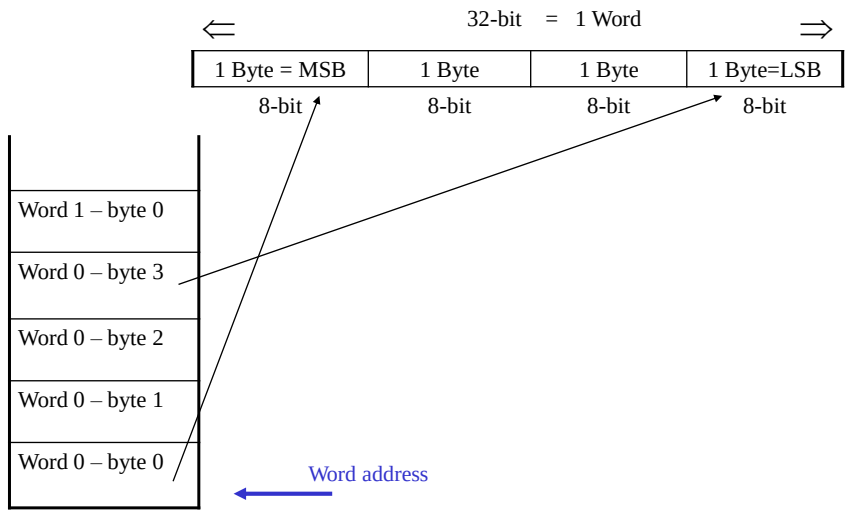


Ispirato dai "I viaggi di Gulliver" di Jonhatan Swift

Disposizione in memoria::little endian



Disposizione in memoria::big endian



Istruzioni di trasferimento dati

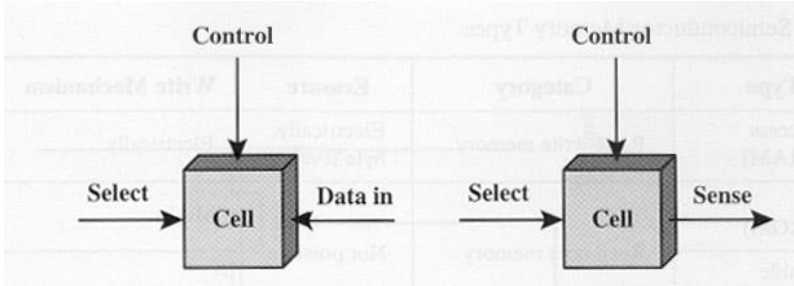
- Gli operandi di una istruzione aritmetica devono risiedere nei registri che sono in numero limitato (32 nel MIPS). I programmi in genere richiedono un numero maggiore di variabili.
- Cosa succede ai programmi i cui dati richiedono più di 32 registri (32 variabili)?

  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

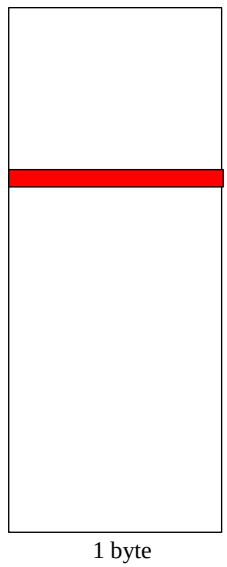
Cella di memoria

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)

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

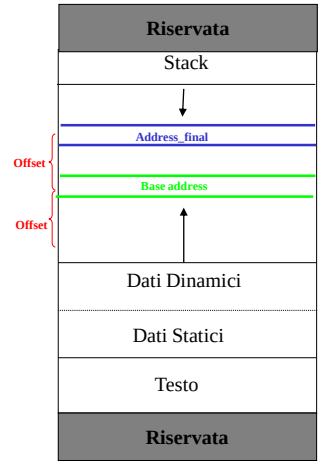
sw (store word) per trasferire il contenuto di un registro della CPU in una parola di memoria

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

Indirizzamento della memoria dati

Base + spiazzamento
Base + Offset

Address_final = Base_address + Offset



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*.

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];`

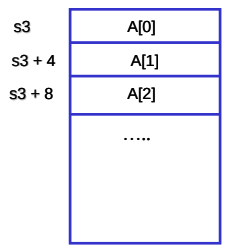
- Si suppone che:
  - la variabile **h** sia associata al registro **$s2**
  - l'indirizzo del primo elemento dell'array *(base address)* sia contenuto nel registro **$s3 (A[0])**

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
```


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



A[0]	0	1	2	3
	4	5	6	7
Offset (A[2])	8	9	10	11
	2^k-4	2^k-3	2^k-2	2^k-1

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]
```


Array: aritmetica dei puntatori

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

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

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.

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

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:   slt $t0, $i, $N                ; t0 = 1 se i < N
          beq $t0, $zero, esci           ; se t0 = 0 (i ≥ N) esci
          ..
          ..
          ..
          j inizia                       ; torna in ciclo
esci:
```


Istruzioni di salto in ciclo repeat

Ciclo a condizione finale di uscita (viene eseguito sempre almeno 1 volta)

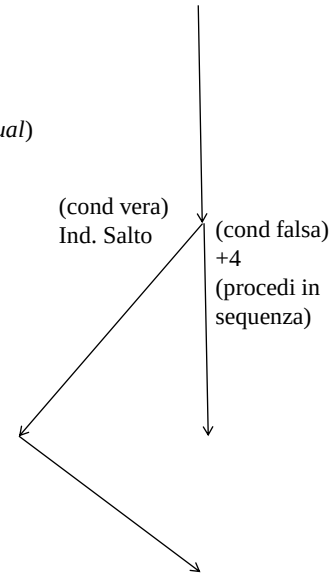
```
i=0;
repeat
{
    elem = i*N + j;          // Istruzioni aritmetico-logiche
    s = v[elem];             // Istruzioni di accesso a memoria
    z[elem] = s;             // Istruzioni di accesso a memoria
    i++;
} until (i >= N)

addi $t0, $t0, 0
inizia:   ..
          ..
          add $t0, $t0, 1
          slt $t0, $i, $N                ; t0 = 1 se i < N
          bne $t0, $zero, inizia         ; se t0 = 1 (i < N) rimani in ciclo
esci:
```

Istruzioni di salto condizionato

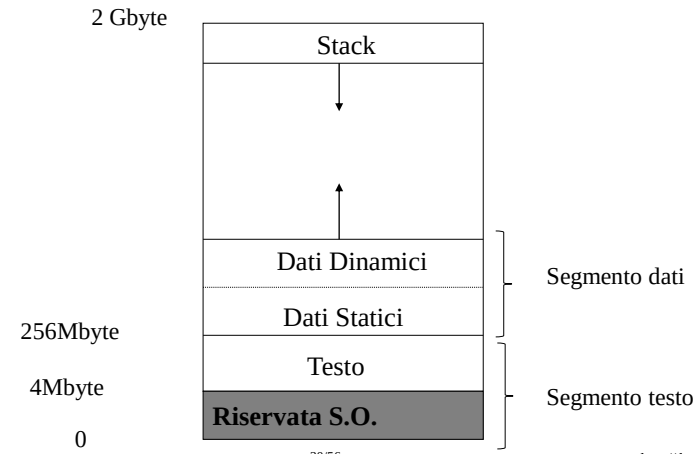
- Salti condizionati relativi:
  - **`beq r1, r2, Etichetta`** (*branch on equal*)
  - **`bne r1, r2, Etichetta`** (*branch on not equal*)

- Salti condizionati relativi:
  - Il flusso sequenziale di controllo cambia solo se la condizione è vera. (beq)



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.



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

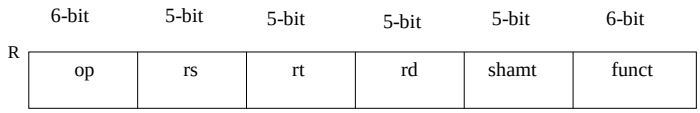
Codifica delle istruzioni

- 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 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.

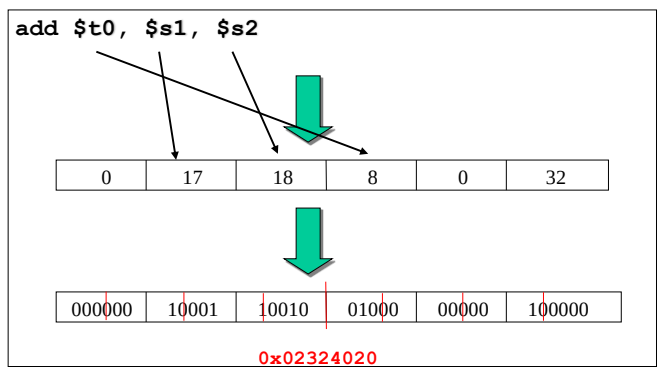
	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	Indirizzo / costante		
J	op	Indirizzo / costante				

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



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

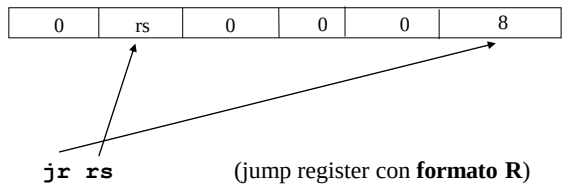
$s1 = s2*2^3$ Se s2 contiene 20 (0000.....0010100) => s1 conterrà = 160 (0000....0010100000)

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

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

Altre istruzioni di tipo R

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



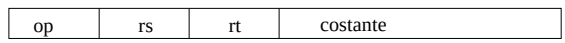
`jr rs` (jump register con **formato R**)

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

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

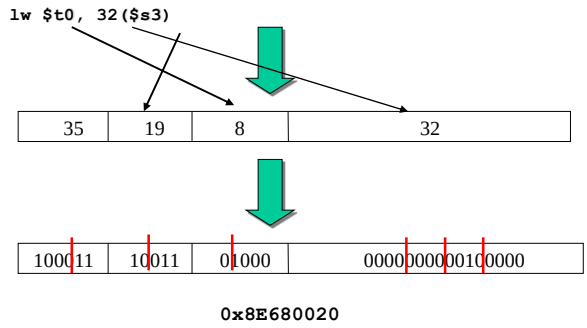


6 bit 5 bit 5 bit 16 bit

- 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: indirizzo = indirizzo_base + offset (= costante)



Istruzioni di tipo I: accesso memoria

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

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

Versione I di istruzioni aritmetico-logiche

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $t0, $s3, 64`	001000	10011	01000	0000 0000 0100 0000

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s1, $s2, 64`	001000	10010	10001	0000 0000 0000 0100

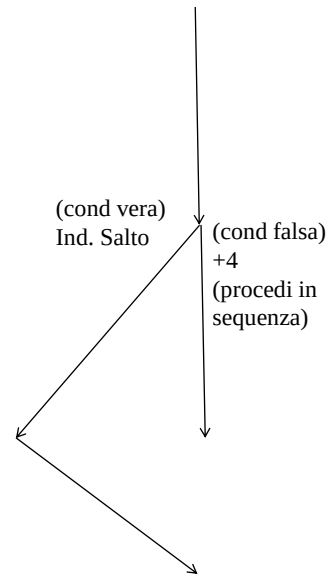
Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
`slti $t0, $s2, 8`	001010	10010	01000	0000 0000 0000 1000

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


Istruzioni di salto condizionato

- Salti condizionati relativi:
  - **`beq r1, r2, L1`** (*branch on equal*)
  - **`bne r1, r2, L1`** (*branch on not equal*)

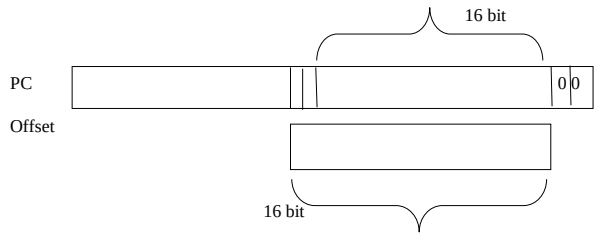
- Salti condizionati relativi:
  - Il flusso sequenziale di controllo cambia solo se la condizione è vera (`beq`)
  - Il calcolo del valore dell'etichetta **`L1`** (**indirizzo di destinazione del salto**) avviene a partire dal Program Counter (PC).
  - Indirizzamento del tipo Base (PC) + Spiazzamento.



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 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 rappresenta l'offset in termini di numero di istruzioni

Calcolo dell'indirizzo di salto

```
                          0x400          addi $t1, $zero, 10   # N=10
for (i=0; i++; i<10)      0x404          addi $t0,$zero,0      # i =0;
{    ....                 0x408    loop:  addi $t0, $t0,1       # i++
                          0x40C          ....

                          0x420          bne $t0, $t1, loop
}
```

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

Calcolo dell'indirizzo di salto

```
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
```

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

Lo spiazzamento del salto in byte è pari a: (0x408 - 0x424) = -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 = 0x408 = 0x424 + -7 * 4

5	8	9	-7 = 1111 1111 1111 1001

Istruzioni di tipo I - Branch

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`beq $s1, $s2, L1`	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
`beq $s1, $s2, L1`	000100	10001	10010	1111 1111 1110 0111

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
`beq $s1, $s2, L1`	000100	10001	10010	0000 0000 0001 1000

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

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s2,$s1,24`	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-matematico, 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)

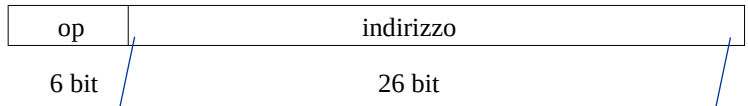
Non c'è corrispondenza 1 a 1, tra tipi strutturali e tipi funzionali.

Sommario

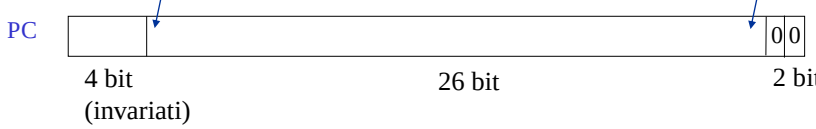
- Istruzioni di accesso alla memoria
- Istruzioni di salto
- I tipi di istruzioni: il formato R
- I tipi di istruzioni: il formato I
- I tipi di istruzioni: il formato J

Formato istruzioni di tipo J

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



- 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).



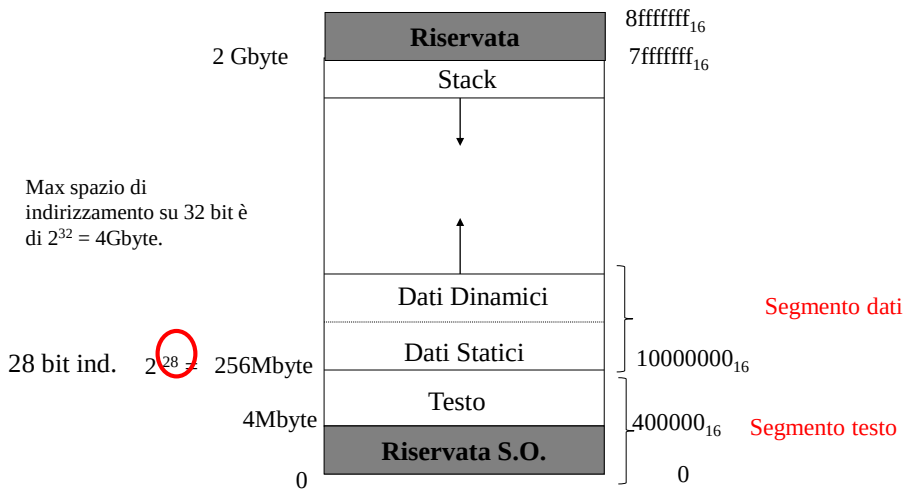
$0 \leq \text{indirizzo} < 2^{28} = 256 \text{ MByte}$

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.

Organizzazione logica della memoria



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.

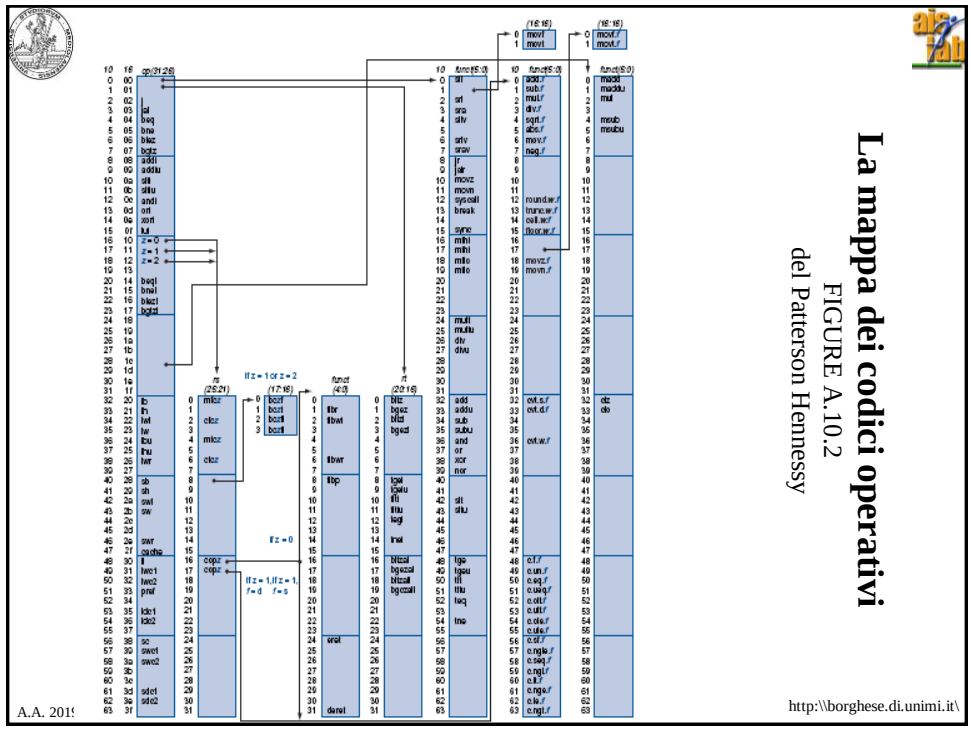
	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	Indirizzo / costante		
J	op	Indirizzo / costante				

Formati e tipi di istruzioni

ISA
- Istruzioni aritmetiche
- Istruzioni di accesso a memoria
- Istruzioni di controllo di flusso
- (Istruzioni di I/O)

Formati
- R - Registro
- I – Immediato
- J - Jump (salto)

Non c'è corrispondenza 1 a 1 tra tipi di istruzioni e formati



La mappa dei codici operativi

FIGURE A.10.2

del Patterson Hennessy

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