Lezione 16

Il linguaggio macchina

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Linguaggio macchina

- *Le istruzioni in Assembly devono essere tradotte in binario (sequenze di 0 e 1) per poter essere eseguite.*
- **Linguaggio macchina:**
  rappresentazione binaria di istruzioni Assembly
- **Linguaggio macchina MIPS:**
  - Le istruzioni sono lunghe **32 bit** (come i registri e le parole di memoria)
- Costruita secondo il suo specifico formato (tipo):
  - **tipo R (register)** – istruzioni utilizzanti i registri
  - **tipo I (immediate)** – istruzioni con operando "immediate"
  - **tipo J (jump)** – istruzioni di salto

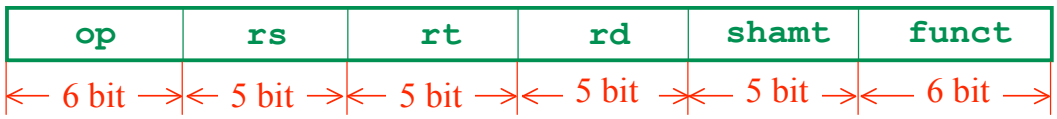
Istruzioni di tipo R

- **Tipo R:** istruzioni che operano **su registri**
  - Generalmente di categoria aritmetico-logica, ma non solo.
  - Aritmetico-logiche, confronto
    - `add, sub, mult, div, and|or|not, ... , slt`
  - Gestione registri speciali: `mflo, mfhi`
  - Salto: jump register (`jr`)

- Le diverse istruzioni aritmetico-logiche si distinguono tra loro in base al campo **`funct (6 bit)`**

Formato istruzioni: ***formato/tipo R***

Formato R (register)



`op:` (**opcode**) identifica il tipo di istruzione

`rs:` registro contenente il primo operando sorgente

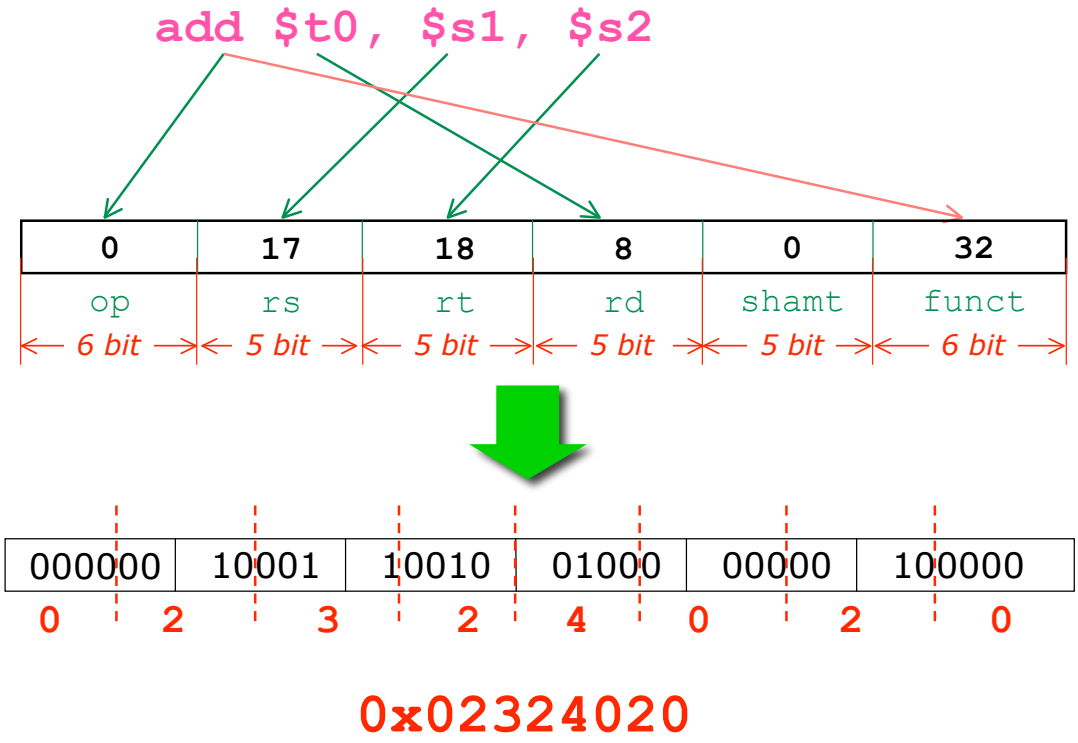
`rt:` registro contenente il secondo operando sorgente

`rd:` registro destinazione contenente il risultato

`shamt:` shift amount (scorrimento)

`funct:` indica la variante specifica dell'operazione

Istruzioni di tipo R: esempio



0x02324020

Istruzioni di tipo R: esempi

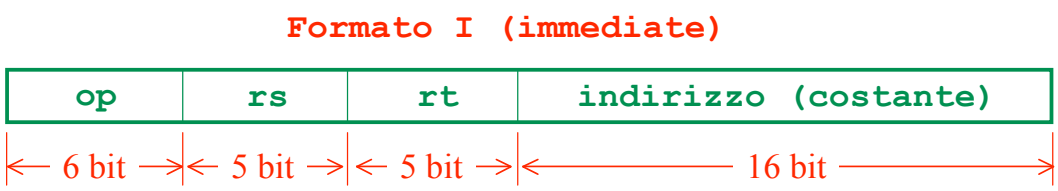
- Le operazioni logico-aritmetiche di tipo **R** hanno **opcode=0**
  - Non tutte le operazioni logico-aritmetiche sono di tipo R
  - Non tutte le operazioni con codice operativo 0 sono logico-aritmetiche

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`add $s1, $s2, $s3`	000000	10010	10011	10001	00000	100000

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`sub $s1, $s2, $s3`	000000	10010	10011	10001	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

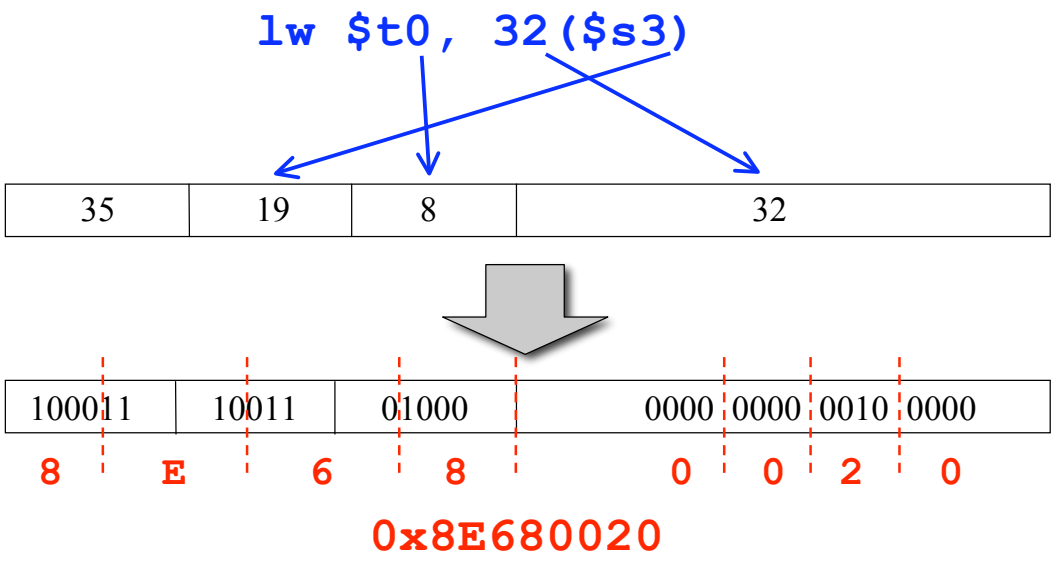
Formato istruzioni di tipo I



- **Es: categoria: load/store**
  - `op` identifica il tipo di istruzione;
  - `rs` indica il registro base / sorgente;
  - `rt` indica il registro destinazione;
  - `Indirizzo` riporta lo spiazzamento (offset)
- **16 bit** di spiazzamento
  - Intervallo: $-2^{15} \div +2^{15}-1$ (±32 kB) rispetto all'indirizzo base.

Istruzioni di tipo I: esempio

- Istruzione `lw`: linguaggio macchina



Istruzioni load/store di tipo I

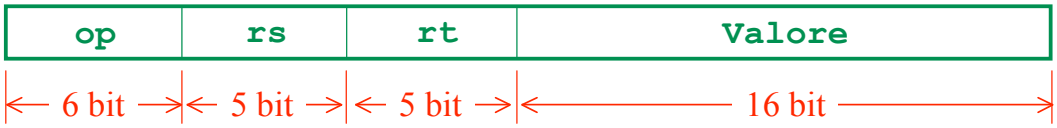
- *Istruzioni:*
  - *Categoria:* ***load/store***
  - *Formato:* ***tipo I:***

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

Formato I – istr. aritmetico-logiche

Formato I (immediate) – aritm.logiche

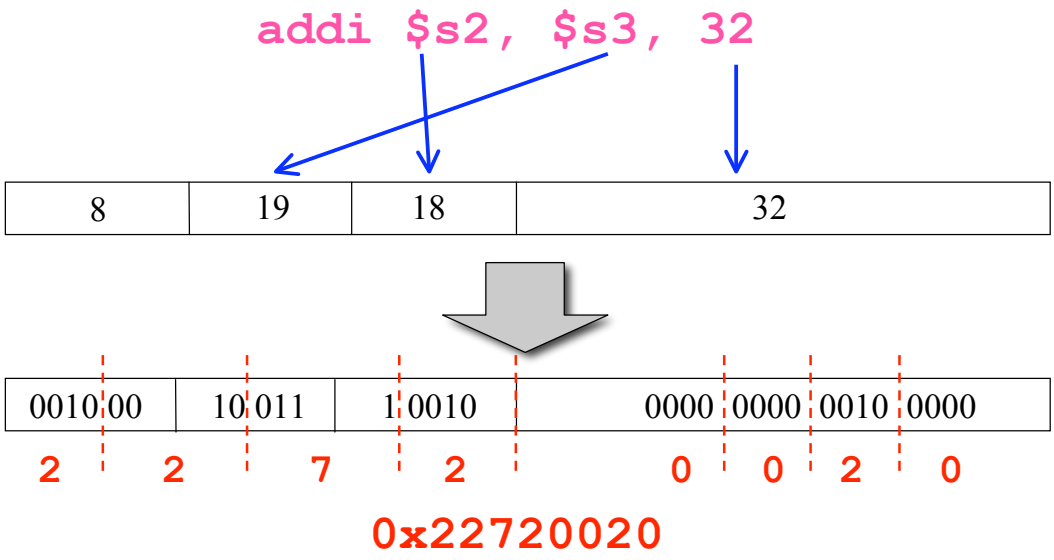


- **Categoria: aritmetico-logiche**
  - `op` identifica il tipo di istruzione;
  - `rs` indica il registro base / sorgente;
  - `rt` indica il registro destinazione;
  - `Valore` riporta l'operando (costante)

- Costante: 16 bit
  - Intervallo: $-2^{15} \div +2^{15}-1$ (± 32 k) con segno;
  - Intervallo: $0 \div +2^{16}-1$ ($0 \div 64$ k) unsigned;

Istruzioni aritmetico-logiche di **tipo I**

- Istruzione: `addi (add immediate)` in linguaggio macchina:



Istruzioni logico-aritmetiche di tipo I

- Il **tipo I** è il formato per le istruzioni con operandi immediati:
  - Esempi:
  - `addi` (*add immediate*)
  - `slti` (*set less than immediate*)

Nome campo	op	rs	rt	"Indirizzo"
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s1, $s1, 4`	001000	10001	10001	0000 0000 0000 0100

Nome campo	op	rs	rt	"indirizzo"
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
```

Esempio: traduzione in linguaggio macchina

C

```
A[300] = h + A[300];
```

dove:

$s2 → h

$t1 → Indirizzo base di A[]

Assembly

```
lw $t0, 1200($t1)
add $t0, $s2, $t0
sw $t0, 1200($t1)
```


Esempio (cont.)

```
lw $t0, 1200($t1)
add $t0, $s2, $t0
sw $t0, 1200($t1)
```

Linguaggio macchina

35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

1000 11	01 001	0 1000	0000 0100 1011 0000		
0000 00	10 010	0 1000	0100 0	000 00	10 0000
1010 11	01 001	0 1000	0000 0100 1011 0000		

```
lw $t0, 1200($t1)      0x8D2804B0
add $t0, $s2, $t0      0x02484020
sw $t0, 1200($t1)      0xAD2804B0
```

Istruzioni di salto

- *Istruzioni di salto:*

  normalmente il Program Counter viene incrementato di 1 word (4 byte) durante l'esecuzione di un'istruzione

- SALTO: aggiornamento del Program Counter
  - Nel PC viene inserito l'indirizzo dell'istruzione a cui saltare.

- Categorie di salto:
  - branch `beq, bne`
  - jump `j`
  - chiamata a procedura `jal`

Salto **condizionato relativo**

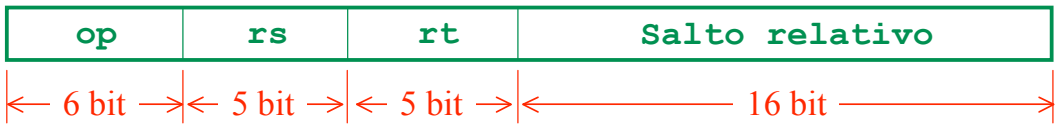
- Branch MIPS:
  salti condizionati relativi:

  `beq r1, r2, L1` (*branch on equal*)

  `bne r1, r2, L1` (*branch on not equal*)

- condizionati:
  - Il flusso sequenziale di controllo cambia solo se la condizione è vera.
- relativi:
  - Il calcolo del valore dell'etichetta `L1` (indirizzo di destinazione del salto) è relativo al Program Counter (PC)

Formato I – Salto condizionato (branch)

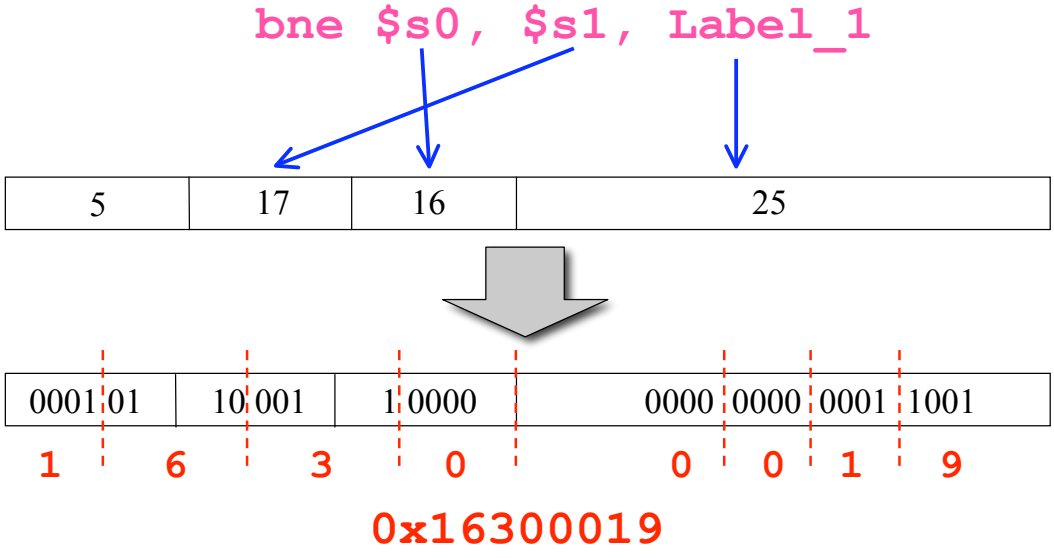


- **Categoria: salto condizionato (branch)**
  - `op` identifica il tipo di istruzione (branch);
  - `rs` indica il primo registro da confrontare;
  - `rt` indica il secondo registro da confrontare;
  - `Salto relativo` riporta l'ampiezza di salto, a partire dal contenuto attuale del Program Counter (PC + 4)

- Offset: **16 bit** del campo `Salto rel.`
  - Rappresentano un indirizzo di parola relativo al PC (*PC-relative word address*)
  - Intervallo: $-2^{15} \div +2^{15}-1$ **parole** ➔ $-2^{17} \div +2^{17}-1$ **Bytes** **(±128 kB)** rispetto all'indirizzo base

Istruzioni di salto condizionato (branch) – tipo I

- Istruzione: `bne` **(branch on not equal)**
  in linguaggio macchina:



Istruzioni di salto condizionato (tipo I)

- L'indirizzamento relativo al Program Counter permette di fare dei salti condizionati ad aree di memoria il cui indirizzo non è esprimibile con 16 bit
- Esempio: `bne $s0, $s1, L1`
  - Per esprimere l'etichetta `L1` si hanno a disposizione 16 bit
  - `L1` è calcolato rispetto al Program Counter in modo da saltare di 2^{15} **parole** avanti o indietro rispetto all'istruzione corrente
- Per il **principio di località** degli indirizzi di memoria è utile calcolare l'indirizzo di destinazione del salto come **offset** rispetto all'istruzione corrente.

Istruzioni di salto condizionato (tipo **I**)

- **Offset** relativo al PC rappresentato in complemento a due per permettere salti in avanti e all'indietro.
  - L'offset varia tra: -2^{15} e $+2^{15}-1$ **parole**
  - Campo `indirizzo` negativo → salti all'indietro
- **Indirizzo di parola**: corrisponde alla divisione per 4 dell'indirizzo di byte
  - I due bit meno significativi sono sempre: "**00**"
  - ottimizzazione possibile grazie alla dimensione fissa (32 bit – 4 bytes) delle istruzioni macchina.
- Si può saltare ad una ***parola*** nell'intervallo:
  -2^{17} ÷ $+2^{17}-1$ bytes
  - Range: 2^{18} = 256 K (± 128 K) bytes

Istruzioni di branch: esempio

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`beq $s1, $s2, 100`	000100	10001	10010	0000 0000 0001 1001

L1: +25 word = +100 byte ➔ Codifica: 0000 0000 0001 1001 (00)

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`beq $s1, $s2, -100`	000100	10001	10010	1111 1111 1110 0111

L1: −25 word = −100 byte ➔ Codifica: 1111 1111 1110 0111 (00)

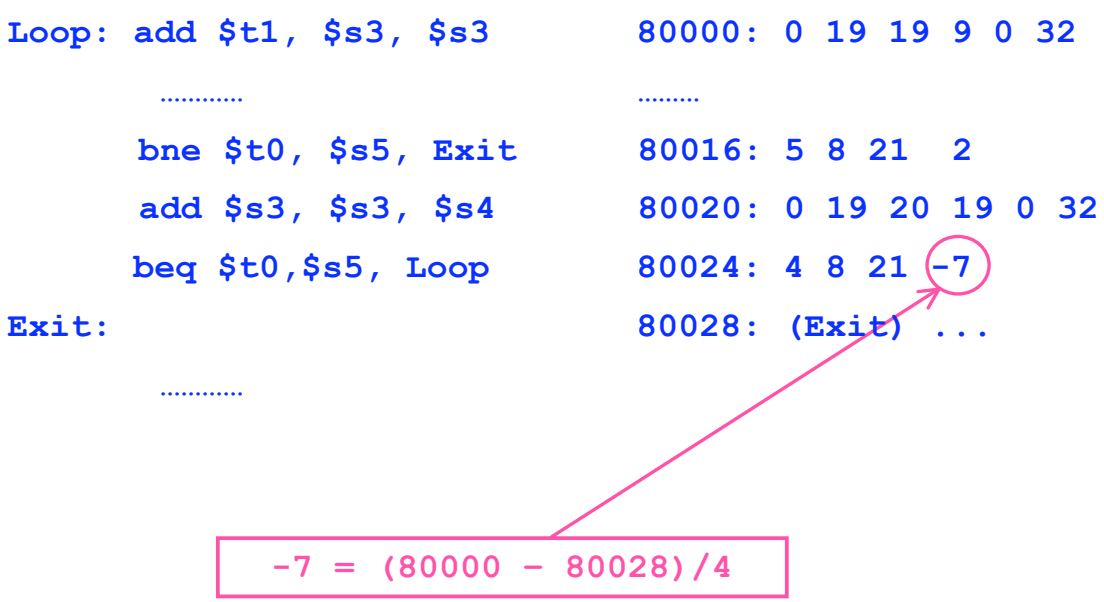
Esempio

```
Loop:  add $t1, $s3, $s3          80000: 0 19 19 9 0 32
         .............
       bne $t0, $s5, Exit         80016: 5  8 21 2
      (bne $t0, $s5, 8)
       add $s3, $s3, $s4
       beq $t0,$s5, Loop
Exit:                             80028: (Exit) ...
         .............
```

2 = (80028 − 80020) / 4

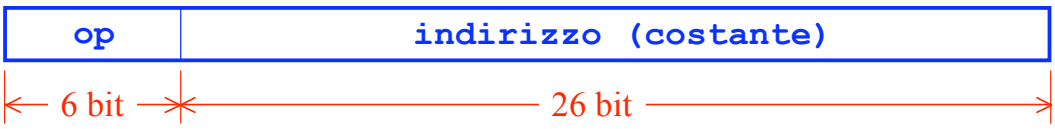
Quando si esegue la **bne**,
PC punta già all'istruzione successiva (**80020**)

Esempio



Istruzioni di tipo J

Formato J (jump)

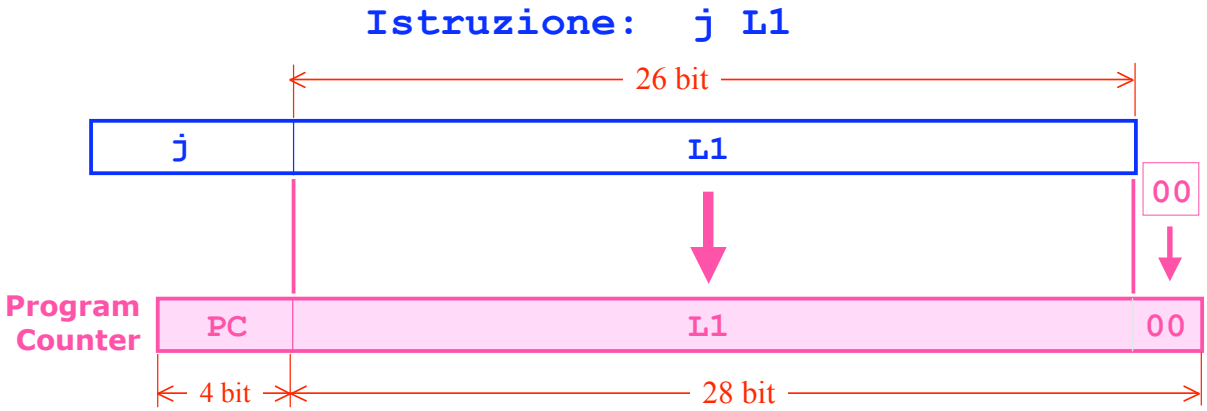


- **Tipo J**: usato per le istruzioni di **salto incondizionato *(jump)***
  - **op (6 bit)**: indica il tipo di operazione
  - **indirizzo (26 bit)**: riporta una parte (**26 bit su 32**) dell' **indirizzo assoluto** di destinazione del salto

- I **26 bit** del campo **indirizzo** rappresentano un indirizzo di parola (word address)
  - Poiché si punta sempre e comunque ad un'istruzione di programma

Jump: modifica del *Program Counter*

- Il campo `L1` di 26 bit rappresenta un indirizzo di parola
  - (indirizzo di byte) = (indirizzo di parola) x 4
  - PC(30:31) ⇐ "00"
  - PC(4:29) ⇐ L1
  - PC(0:3) invariato



Istruzioni di salto incondizionato (tipo J)

- L' Assembler sostituisce l'etichetta `L1` con i **28 bit** meno significativi dell'indirizzo di memoria (byte) traslati a destra di 2, ottenendo **26 bit**
  - I **26 bit** di indirizzo rappresentano un indirizzo di parola (word address)
    → indirizzo di byte (byte address) composto da **28 bit**
  - Si amplia lo spazio di salto: tra 0 e 2^{28} Byte (2^{26} word) = 256 Mbyte
- Registro PC è composto da **32 bit** (spazio: 4 GB) ⇒
  - `jump` rimpiazza solo i 28 bit meno significativi del PC
  - lascia inalterati i 4 bit più significativi
- Per saltare su 32 bit devo utilizzare: `jr $reg`

Esempio : jump (**formato J**)

Assembly	**Linguaggio macchina**
`Loop: add $t1, $s3, $s3`	`80000: 0 19 19 9 0 32`
`.....`	`.....`
`bne $t0, $s5, Exit`	`80016: 5 8 21 2`
`add $s3, $s3, $s4`	`80020: 0 19 20 19 0 32`
`j Loop (j 80000)`	`80024: 2 20000`
`Exit:`	`80028: Exit`

`beq $t0,$s5, Loop` ➔ `j Loop`

Istruzioni di tipo J: esempio

Nome campo	**op**	**indirizzo**				
Dimensione	6-bit	26-bit				
`j 32`	000010	00 0000	0000	0000 0000	0000	1000

Esempio precedente: `j 80000` = 0000 00000001 00111000 10000000

Nome campo	**op**	**indirizzo**				
Dimensione	6-bit	26-bit				
`j 80000`	000010	00 0000	0001	0011 1000	1000	0000

80000 / 4 = 20000

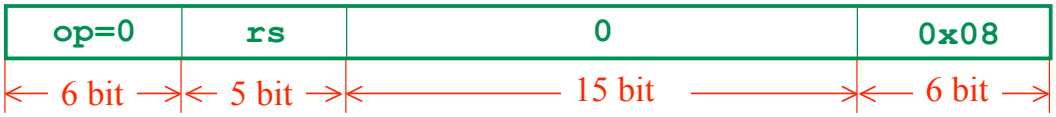
Salti incondizionati: istruzione `jump register`

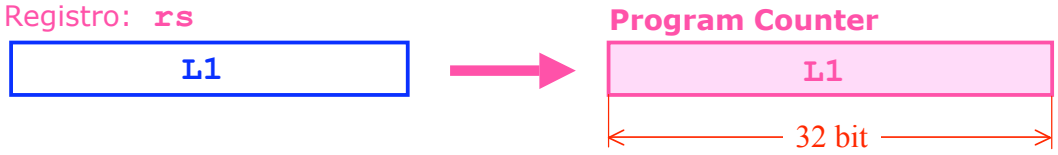
- Per saltare ad indirizzi superiori a 2^{28} Byte si usa l'istruzione:

  `jr rs` `(jump register)`

- Salta all'indirizzo di memoria **assoluto** contenuto nel registro `rs`
  - In questo caso `L1` (contenuto in `rs`) è un **BYTE Address**.

Istruzione: `jr $ra` (formato R)





Istruzioni di salto: sintesi

- Salti condizionati, relativi: Formato **I**
  - Si salta solo se la condizione è vera.
  - L'indirizzo di destinazione del salto è relativo al Program Counter (PC).
  - `beq r1, r2, L1` (*branch on equal*)
  - `bne r1, r2, L1` (*branch on not equal*)

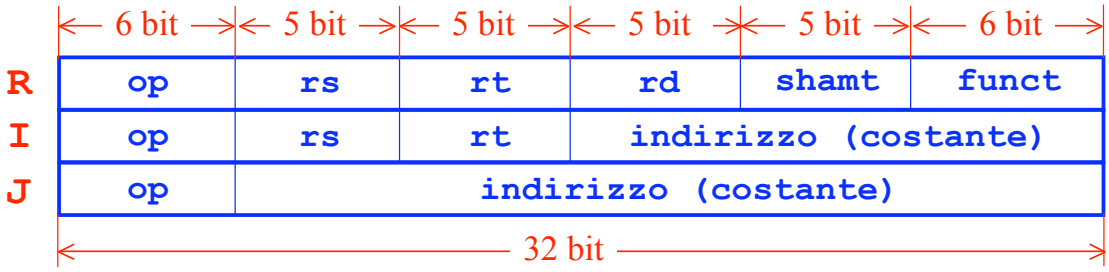
- Salti incondizionati, assoluti: Formato **J**
  - Il salto viene sempre eseguito
  - L'indirizzo di destinazione del salto è un indirizzo assoluto di memoria
  - `j L1` *(jump)*
  - `jal L1` *(jump and link)*

- Salto incondizionato a registro: formato **R**
  - L'indirizzo è contenuto in un registro
  - `jr rs` *(jump register)*

Istruzione	Tipo
`beq`	**I**
`bne`	**I**
`j`	**J**
`jal`	**J**
`jr`	**R**

Riepilogo: Formato istruzioni

- **3 diversi formati: R, I, J**
- Formato riconosciuto dalla CPU tramite il valore del primo campo:
  ***codice operativo* (*opcode*) – (6 bit più significativi)**
  - che indica alla macchina come trattare i rimanenti bit dell'istruzione.



Riepilogo: ***MIPS Instruction Set***