Il linguaggio assembly – Parte II

Architettura
degli Elaboratori e delle Reti

Alberto Borghese
Università degli Studi di Milano
Dipartimento di Scienze dell'Informazione
email: borghese@dsi.unimi.it

Sommario

- Architettura di riferimento dei calcolatori
- Esecuzione delle istruzioni
- Il linguaggio assembly
- Il linguaggio macchina
- Insieme delle istruzioni
- Formato delle istruzioni
- Codifica delle istruzioni in linguaggio assembly
- Modalità di indirizzamento

Linguaggio macchina

- Le istruzioni in linguaggio assembly devono essere tradotte in linguaggio macchina (cioè in sequenze di 0 e 1) per poter essere eseguite.
- Le istruzioni in linguaggio macchina sono lunghe **32 bit** (come i registri e le parole di memoria).

Linguaggio assembly → Linguaggio macchina

Linguaggio macchina

Codifica binaria

- Codifica posizionale
- 2 simboli, base 2.

$01001101 = 0x2^7 + 1x2^6 + 0x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 77$

Codifica esadecimale

- 16 simboli, base 16.

$0x4D = 4x16^1 + 13x16^0 = 77$

Relazione tra codifica binaria ed esadecimale

0100 -> 4 4 -> 0100
1101 -> D D -> 1101

Linguaggio macchina

- E' necessaria una convenzione per rappresentare i registri tramite numeri.
- In MIPS:
  - **$zero → 0,** **0x00**
  - **$at → 1** (registro riservato) **0x01**
  - **$v0, $v1 → 2, 3** **0x02, 0x03**
  - **$a0 - $a3 → 4 – 7** **0x04 – 0x07**
  - **$t0 - $t7 → 8 – 15** **0x08 – 0x0F**
  - **$s0 - $s7 → 16 – 23** **0x10 – 0x17**
  - **$t8, $t9 → 24, 25** **0x18, 0x19**
  - **$k0, $k1 → 26, 27** (registri riservati) **0x1A, 0x1B**
  - **$gp, $sp, $fp, $ra → 28, 29, 30, 31** **0x1C - 0x1F**

Formato istruzioni aritmetiche

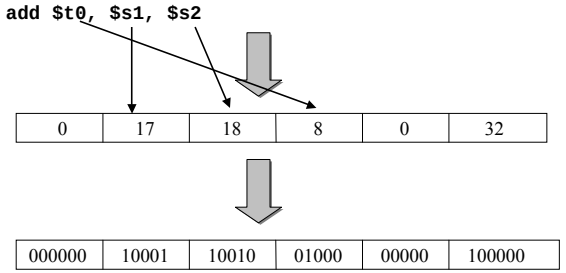
- La rappresentazione binaria di un'istruzione assembly è composta da *32* bit (la stessa dimensione di una parola di memoria) e rispetta il seguente formato (tipo R) nel caso di istruzione aritmetico-logica tra registri.

Formato istruzioni di tipo R

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

- Ai vari campi sono stati assegnati dei nomi mnemonici:
  - **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

- Istruzioni aritmetico-logiche con il tipo di formato visto, vengono chiamate di **tipo R** (registro).
- Esempi:
  - somma, prodotto, divisione
  - shift (scorrimento)
  - AND, OR, NOT
- Le diverse istruzioni aritmetico-logiche si distinguono tra loro in base al campo **`funct`.**

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
`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

Nome campo	op	rs	rt	immediate
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s2, $s3, const`	001000	10010	10011	10001010100111011

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…*).

Linguaggio macchina

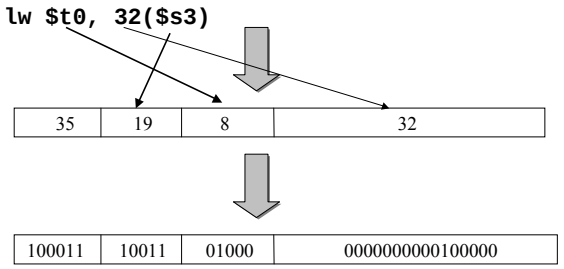
- Il formato delle istruzioni di tipo R non è adatto a rappresentare istruzioni di load/store.
- Alla costante (offset) delle istruzioni **lw** e **sw** sarebbe riservato un campo di **5 bit** (le costanti sarebbero al massimo di dimensione 2^5 = 32)
- Per le istruzioni di load/store si utilizza un formato diverso (**tipo I**) utilizzando sempre 32 bit complessivi

Formato istruzioni di tipo I

op	rs	rt	indirizzo
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 base;
  - **rt** indica il registro destinazione dell'istruzione di caricamento;
  - **indirizzo** riporta lo spiazzamento (offset).
- Con questo formato una istruzione **lw (sw)** può indirizzare parole nell'intervallo -2^{15} + $+2^{15}-1$ rispetto all'indirizzo base.

Istruzioni di tipo I: esempio



0x8E680020

Istruzioni di tipo I: esempi

- Il secondo tipo di formato istruzioni è il formato per le *istruzioni di load/store*:

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

Esempio

A[300] = h + A[300]

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

$s2 ⟶ h

$t1 ⟶ Indirizzo base di *A*

Esempio (cont.)

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

100011	01001	01000	0000010010110000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000010010110000		

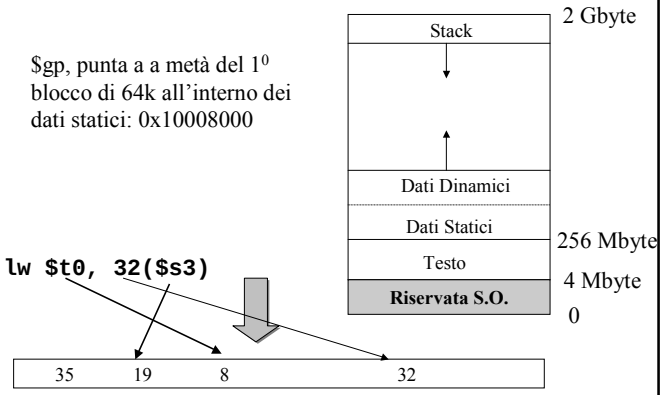
0x8D2804B0

0x02484020

0xAD2804B0

Problema con load/store

$gp, punta a a metà del 1^0 blocco di 64k all'interno dei dati statici: 0x10008000



Problema con lw

Per caricare dalla memoria occorrono due operazioni:

- Caricare il base address del vettore.
- Caricare l'offset.

- lui $s0, 0x1000
- lw $v0, 0x8000($s0)

- lw $s0, 0($gp)

- Sono facilitati gli accessi ai dati compresi tra 4Mbyte e 4,064Mbyte.

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.
  - Il calcolo del valore dell'etichetta **`L1`** (indirizzo di destinazione del salto) è relativo al Program Counter (PC).

Formato istruzioni di salto condizionato (tipo I)

op	rs	rt	indirizzo
6 bit	5 bit	5 bit	16 bit

- Nel caso di salti condizionati, i campi hanno il seguente significato:
  - **op** identifica il tipo di istruzione;
  - **rs** indica il primo registro;
  - **rt** indica il secondo registro;
  - **indirizzo** riporta lo spiazzamento (offset).
- Per l'offset si hanno a disposizione solo 16-bit del campo **indirizzo** ⇒ rappresentano un indirizzo di **parola** relativo al PC **(PC-relative word address)**
- Con questo formato una istruzione **salto** può indirizzare parole nell'intervallo -2^{15} + $+2^{15}-1$ rispetto all'indirizzo base.

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 solo 16-bit ⇒ il valore di **`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)

- I 16-bit del campo indirizzo esprimono **l'offset** rispetto al PC rappresentato in complemento a due per permettere salti in avanti e all'indietro.
- L'offset varia tra -2^{15} e $+2^{15}-1$
- Esempio: **`bne $s0, $s1, L1`**
- L'assemblatore sostituisce l'etichetta **`L1`** con l'indirizzo **di parola** relativo a PC: **(L1- PC)/4**
  - PC contiene già l'indirizzo dell'istruzione successiva al salto
  - La divisione per 4 serve per calcolare l'indirizzo di parola
- Il valore del campo **`indirizzo`** può essere negativo (salti all'indietro)

Istruzioni di salto condizionato (tipo I)

- Indirizzare una parola (4-Byte) corrisponde alla divisione per 4 (ottimizzazione possibile grazie alla memoria indirizzata al byte)
  - I due bit meno significativi sono sempre 00

NB si può saltare ad una ***parola*** nell'intervallo -2^{15} $+2^{15}-1$ rispetto all'indirizzo base dato dal PC.

Istruzioni di tipo I: 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 = 100 in byte Codifica 00000000000011001(00) in binario.

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 = -100 in byte Codifica 1111111111100111(00) in binario.

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

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

Esempio

```
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
       beq $t0,$s5, Loop        80024: 4 8 21 -7
Exit:                           80028: Exit ...
       ……
```

-7 = (80000 – 80028)/4

Istruzioni di tipo I: esempi

- Il tipo I è il formato per le istruzione con operandi immediati: ad esempio `addi` (addition immediate) e `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

Istruzioni di salto

- Salti condizionati relativi:
  - **`beq r1, r2, L1`** (*branch on equal*)
  - **`bne r1, r2, L1`** (*branch on not equal*)
- Salti incondizionati assoluti:
  - **`j L1`** (*jump*)
  - **`jr r`** (*jump register*)
  - **`jal L1`** (*jump and link*) (chiamata a procedura)

Istruzioni di salto

- Salti condizionati relativi:
  - Il flusso sequenziale di controllo cambia solo se la condizione è vera.
  - Il calcolo del valore dell'etichetta **`L1`** (indirizzo di destinazione del salto) è relativo al Program Counter (PC).
- Salti incondizionati assoluti:
  - Il salto viene sempre eseguito.
  - L'indirizzo di destinazione del salto è un indirizzo assoluto di memoria.

Istruzioni di salto

Nome	Formato
beq	I
bne	I
j	J
jr	J
jal	J

Indirizzamento relativo al PC

- Esempio: Operazione di salto condizionato (formato tipo I):

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

Nome campo	op	rs	rt	indirizzo			
Dimensione	6-bit	5-bit	5-bit	16-bit			
`bne $s1, $s2, 100`	000101	10001	10010	0000	0000	0001	1001

Formato istruzioni di tipo J

- Il terzo tipo di formato istruzione (Formato J) è il formato usato per le istruzioni di salto incondizionato *(jump)*:

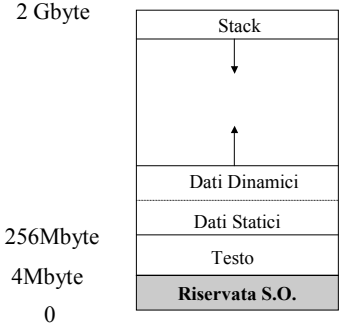
op	indirizzo
6 bit	26 bit

- 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)**

Istruzioni di salto incondizionato (tipo J)

- L'assemblatore sostituisce l'etichetta **L1** con i 28 bit meno significativi traslati a destra di 2 (divisione per 4 per calcolare l'indirizzo di parola) per ottenere 26-bit
  - In pratica elimina i due 0 finali
  - Si amplia lo spazio di salto: si salta tra 0 e 2^{28} Byte (2^{26} word)
- I 26-bit di indirizzo nelle jump rappresentano un indirizzo di parola (word address) ⇒ corrispondono ad un indirizzo di byte (byte address) composto da 28-bit.
- Poiché il registro PC è composto da 32-bit ⇒ l'istruzione jump rimpiazza solo i 28-bit meno significativi del PC, lasciando inalterati i rimanenti 4-bit più significativi.

Organizzazione logica della memoria



Esempio precedente

```
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
       beq $t0,$s5, Loop        80024: 4 8 21 -7
Exit:                           80028: Exit ...
       ……
```

beq $t0,$s5, Loop ➔ j Loop

Esempio

```
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 ...
       ……
```


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 = 00000000 0001001110 0010000000

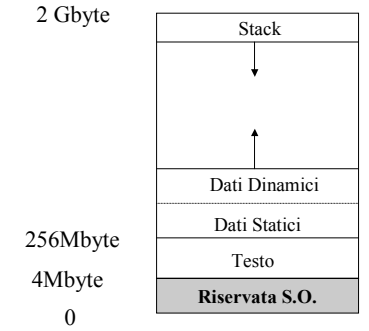
Nome campo	op	indirizzo				
Dimensione	6-bit	26-bit				
`j 80000`	000010	00	00000	0001	0011 1000	1000	0000

Salti incondizionati

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

  `jr rs` (jump register con **formato R**)
- Salta all'indirizzo di memoria **assoluto** contenuto nel registro **rs** (spazio di 2^{32} Word cioè 2^{34} byte : intero spazio di memoria)

Organizzazione logica della memoria



Riepilogo: Assembly vs Linguaggio macchina

Programma in linguaggio ad alto livello (C)

```
a = a + c
b = b + a
var = m [a]
```

Compilatore

Programma in linguaggio assembly (MIPS)

```
add $2, $4, $2
add $3, $3,$2
lw $15, 4($2)
```

Assemblatore

Programma in linguaggio macchina

```
011100010101010
000110101000111
000010000010000
001000100010000
```


Gestione della memoria

- La memoria è indirizzata per byte.
- Le istruzioni sono lunghe 1 parola (4 byte)

- I dati nel segmento dati sono allineati al byte.
- Le istruzioni, contenute nel segmento testo, sono reperibili ogni 4 byte.

Gestione dei salti

Assembly (riferisce al byte)

```
Loop:  add $t1,$s3,$s3
        ……
       bne $t0,$s5,Exit
       (bne $t0,$s5,8)
       add $s3,$s3,$s4
       j Loop
       (j 80000)
Exit:
        ……
```

Linguaggio macchina (riferisce all'implementazione dell'istruzione)

```
80000: 0 19 19 9 0 32
        ……
80016: 5 8 21  2

80020: 0 19 20 19 0 32
80024: 2 20000

80028: Exit ...
        ……
```

PC (80020) : 00000000 00000001 00111000 10010100 +
2 word : 00000000 00000000 00000000 000010(00) =

New PC : 00000000 00000001 00111000 10011100 (80028)

Riepilogo: Assembly vs Linguaggio macchina

Programma in linguaggio ad alto livello (C)

```
a = a + c
b = b + a
var = m [a]
```

Compilatore

Programma in linguaggio assembly (MIPS)

```
add $2, $4, $2
add $3, $3,$2
lw $15, 4($2)
```

Assemblatore

Programma in linguaggio macchina

```
011100010101010
000110101000111
000010000010000
001000100010000
```

Salti condizionati di dimensioni maggiori

```
      beq $s0, $s1, L1
```

```
      bne $s0, $s1, L2
      j L1
L2: ……
```


Formato istruzioni

- I diversi formati (R, I, J) vengono riconosciuti tramite il valore del primo campo, ***codice operativo*** **(*opcode*)**, che indica alla macchina come trattare i rimanenti bit dell'istruzione.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	indirizzo		
J	op	indirizzo				

Sommario

- Architettura di riferimento dei calcolatori
- Esecuzione delle istruzioni
- Il linguaggio assembly
- Il linguaggio macchina
- Insieme delle istruzioni
- Formato delle istruzioni
- Codifica delle istruzioni in linguaggio assembly
- Modalità di indirizzamento

Modalità di indirizzamento

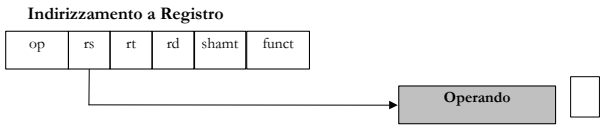
- Le modalità di indirizzamento indicano le diverse modalità attraverso le quali far riferimento agli operandi nelle istruzioni.
- L'esempio più comune di modalità di indirizzamento è l'indirizzamento **a registro** nel quale gli operandi dell'istruzione sono contenuti nei registri:
  ad esempio **`add $s0, $s1, $s2.`**

Modalità di indirizzamento

- MIPS ha solo 5 modalità di indirizzamento:
  - A registro
  - Immediato
  - Con base o spiazzamento
  - Relativo al Program Counter
  - Pseudo-diretto
- Una singola istruzione può usare più di una modalità di indirizzamento.

Indirizzamento a registro

- L'operando (l'indirizzo) è il contenuto di un registro della CPU: il nome del registro è specificato nell'istruzione.



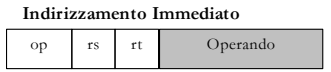
Indirizzamento a registro

- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo R.
- Esempio: istruzione aritmetico-logica:

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

Indirizzamento immediato

- L'operando è una costante il cui valore è contenuto nell'istruzione.
- L'indirizzamento immediato si usa per specificare il valore di un operando sorgente, non ha senso usarlo come destinazione.



- Le istruzioni che usano questo tipo di indirizzamento hanno formato I
  - La costante è memorizzata nel campo a 16-bit

Indirizzamento immediato

- Esempio: operazione aritmetico-logica con operando immediato (formato tipo I):

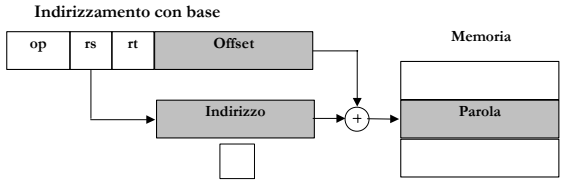
Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s1, $s1, 4`	001000	10001	10001	0000 0000 0000 0100

- Esempio: operazione di confronto con operando immediato (formato tipo I):

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

Indirizzamento con base

- L'operando è in una locazione di memoria il cui indirizzo si ottiene sommando il contenuto di un registro base ad un valore costante (*offset o spiazzamento*) contenuto nell'istruzione.



- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo I.

Indirizzamento con base

- Esempio: istruzione di load
  `lw $t0, 32 ($s3)`
  - L'operando di trova in memoria all'indirizzo **`32+[$s3]`**
- Esempio: istruzione di store
  `sw $t0, 32 ($s3)`
  - L'operando viene copiato in memoria all'indirizzo **`32+[$s3]`**

Indirizzamento con base

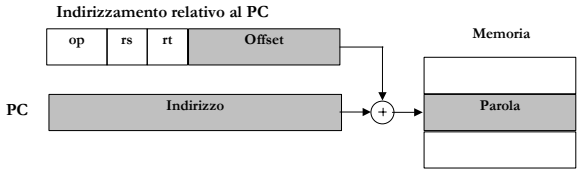
- Esempio: Istruzioni di load/store (formato tipo I):

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

Indirizzamento relativo al PC

- L'operando è in una locazione di memoria il cui indirizzo si ottiene sommando il contenuto del *Program Counter* ad un valore costante (*offset o spiazzamento*) contenuto nell'istruzione:



- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo I.

Indirizzamento relativo al PC

- Esempio: Operazione di salto condizionato (formato tipo I):

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

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`bne $s1, $s2, 100`	000101	10001	10010	0000 0000 0001 1001

Indirizzamento relativo al PC

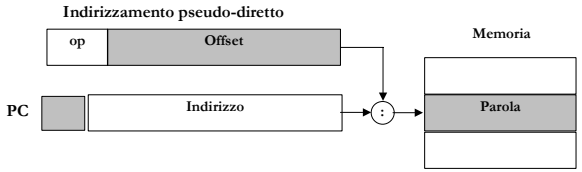
- Si usa l'indirizzamento relativo al PC nei salti condizionati in quanto la destinazione del salto in tali istruzioni è in genere prossima al punto di salto.
- Avendo a disposizione *16* bit di *Offset* $\Rightarrow$ è possibile saltare in un'area tra -2^{15} e $+2^{15}-1$ parole rispetto all'istruzione corrente.

Indirizzamento pseudo-diretto

- Una parte dell'indirizzo è presente come valore costante (offset) nell'istruzione ma deve essere completato.
- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo J.

Indirizzamento pseudo-diretto

- L'indirizzo di salto si calcola facendo uno shift a sinistra di 2 bit dei 26-bit di offset contenuti nell'istruzione (aggiungendo 00 nei bit meno significativi per passare da 26 a 28-bit) e concatenando i 28-bit con i 4-bit più significativi del Program Counter.



Indirizzamento pseudo-diretto

- Esempio: operazione di salto incondizionato (formato J)

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

Utilizzo di costanti

- Spesso le operazioni richiedono l'uso di costanti (ad esempio: somma del valore decimale 4 al contenuto di un registro).
- Possibili **3** opzioni:
  - le costanti risiedono in memoria e sono caricate con **`lw`**
  - utilizzo di registri speciali (es: $zero)
  - utilizzare versioni alternative delle **istruzioni di tipo I** in cui un operando è una **costante**
    - utilizzo di modalità di **indirizzamento immediato**
- La costante è memorizzata nel campo di **16 bit** denominato **`indirizzo.`**

Esempi

```
addi $s0, $s0, 4    # $s0 ← $s0 + 4 (sign-extended)
slti $t0, $s2, 10   # $t0 = 1 if $s2 < 10
andi $s0, $s0, 6    # $s0 ← $s0 and 6 (zero-extended)
ori $s0, $s0, 10    # $s0 ← $s0 or 10 (zero-extended)
li $s0, 20          # $s0 ← 20 (pseudo-instruction)
```

- Le istruzioni di tipo I consentono di rappresentare costanti esprimibili in 16 bit.
- I valori immediati sono espressi in assembly in rappresentazione decimale ma possono anche essere esadecimali o binari

Esempio di indirizzamento a registri & immediato

```
# Somma

.text           #Definizione segmento codice
.globl  main

main:
  li $t1,10  # carica il valore decimale 10 nel reg. $t1
  li $t2,15  # carica il valore decimale 15 nel reg. $t2
  add $a0,$t2,$t1 # $a0 ← $t2 + $t1
print_result:
   li  $v0,1  # stampa risultato
  syscall
```


Esempio di indirizzamento a registri & immediato

```
# L'accesso immediato è usato anche dalle operazioni
# aritmetiche
   .text             #Definizione segmento codice
   .globl  main
main:
   li  $t1,10        # $t1 ← 10
   addi $a0,$t1,15   # $a0 ← $t1 + 15
# stampa risultato
print_result:
   li  $v0,1
   syscall
```


Esempio di indirizzamento a registri & immediato

```
# I valori immediati possono essere dichiarati come
# esadecimali: 10 e 15 espressi in esadecimale con
# 0xa e 0xf.

       .text
       .globl  main
main:
       li $t1,0xA
       addi $a0,$t1,0xF
# stampa risultato
print_result:
       li $v0,1
       syscall
```


Gestione di costanti su 32-bit

- Le istruzioni di **tipo I** consentono di rappresentare costanti esprimibili in 16 bit (valore massimo 65535 unsigned).
- Se 16 bit non sono sufficienti per rappresentare la costante, l'assemblatore (o il compilatore) deve fare due passi:
  - si utilizza l'istruzione **lui (load upper immediate)** per caricare i 16 bit più significativi della costante nei 16-bit più significativi di un registro. I rimanenti 16-bit meno significativi del registro sono posti a 0.
  - una successiva istruzione di **li** specifica i rimanenti 16 bit meno significativi della costante.
- Il registro **$at** è riservato all'assemblatore per creare costanti su 32-bit (costanti 'lunghe').

Istruzione lui: formato tipo I

- L'istruzione **load upper immediate**: **`lui rt, imm`** carica i 16-bit del campo immediato nei 16-bit più significativi del registro **`rt`**. I rimanenti 16-bit meno significativi del registro **`rt`** sono posti a 0.

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`lui $s0, 61`	001111	00000	10000	0000 0000 0011 1101

- Pseudo-istruzione: **`li rdest, imm`** carica il valore imm nel registro rdest.

Caricamento costante di 32 bit Esempio 1

Si consideri la costante a 32 bit:

`0000 0000 0011 1101 0000 1001 0000 0000`

```
lui $s0, 61  (lui $zero,$s0,61)  #61 = 0000 0000 0011 1101
```

valore di **`$s0:`**

```
0000 0000 0011 1101 0000 0000 0000 0000
addiu $s0, $s0, 2304 # 2304 = 0000 1001 0000 0000
```

valore di **`$s0:`**

```
0000 0000 0011 1101 0000 1001 0000 0000
```

400,000 in decimale

Caricamento costante di 32 bit Esempio 2

Si consideri la costante a 32-bit: **118345_{10} (0x1CE49)**

`0000 0000 0000 0001 1100 1110 0100 1001`

16-bit più significativi corrispondenti al valore 1_{10}

16-bit meno significativi corrispondenti al valore 52809_{10}

$$1 \times 2^{16} + 52809 = 118345$$

Si consideri l'istruzione:

```
li $t1, 118345              # $t1 ← 118345
```


Pseudo-istruzione li: esempio

L'assemblatore sostituisce l'istruzione originale con le seguenti istruzioni:

```
lui $at, 1          # 1 = 0000 0000 0000 0001
```

valore di **`$at:`**

```
0000 0000 0000 0001 0000 0000 0000 0000
ori $t1, $at, 52809 # $t1 ← $at or 52809
```

valore di **`$t1:`**

```
0000 0000 0000 0001 1100 1110 0100 1001
```