


La struttura delle istruzioni elementari: il linguaggio Macchina

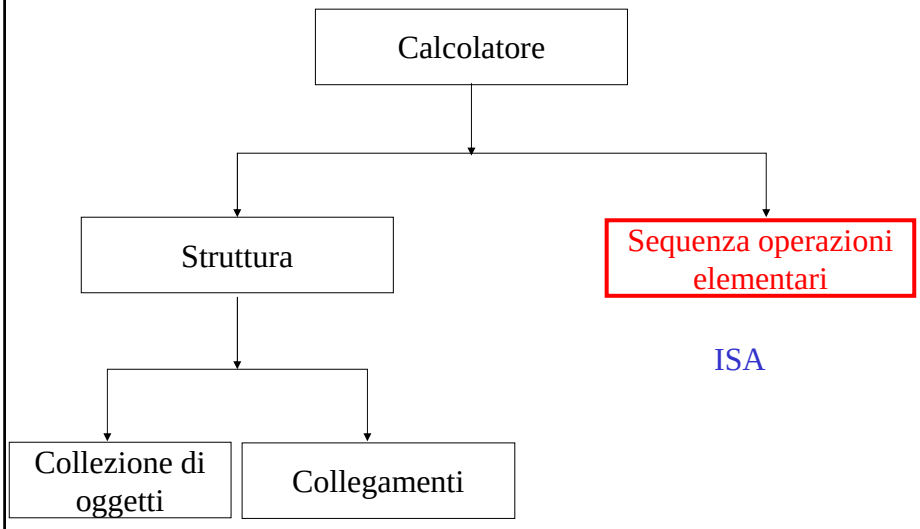
Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: 2.4-2.5-2.6


Sommario

Descrizione di un elaboratore



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).
- Il parsing di un'istruzione in linguaggio macchina viene fatta dalla CPU che ricava le inforamazioni necessarie all'esecuzione dell'istruzione stessa nella fase di decodifica.
- Occorre definire l'architettura delle istruzioni:
  Come vengono raggruppati i bit?
  Cosa viene rappresentato nel singolo bit?
- ***Sono definiti tre tipi di istruzione: tipo R (Register), tipo I (Immediate), tipoJ (Jump).***

Formato istruzioni di tipo R

Campo di un'istruzione: numero di bit consecutivi contenenti un'informazione per l'esecuzione.

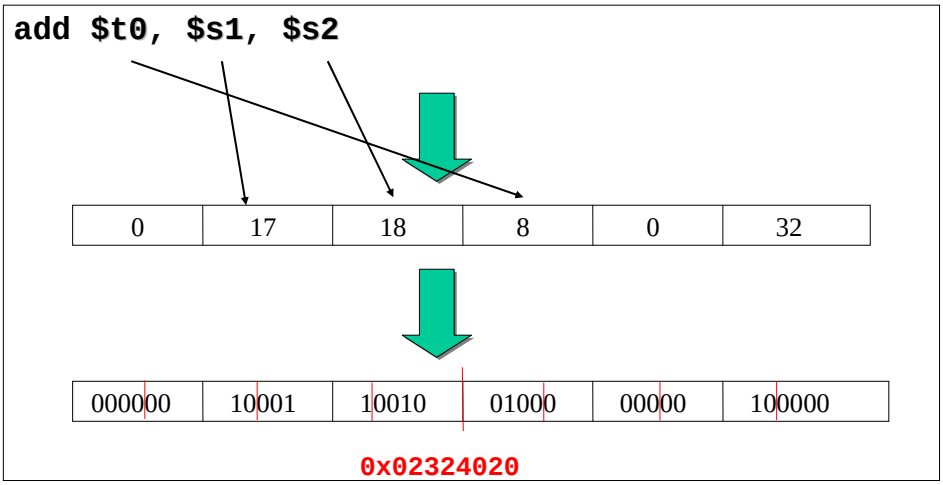
op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

- 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 (target)
  - **`rd:`** registro destinazione contenente il risultato
  - **`shamt:`** shift amount (scorrimento)
  - **`funct:`** indica la variante specifica dell'operazione

MIPS: Software conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer (s8)
31	ra	Return Address (HW)

Istruzioni di tipo R: esempio



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 di tipo R 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
`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	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`sll $s1, $s2, 7`	000000	X	10010	10001	00111 (7)	000000

$s1 = s2*2^7$

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`srl $s1, $s2, 7`	000000	X	10010	10001	00111 (7)	000010

$s1 = s2*2^{-7}$


Altre istruzioni di tipo R (non ALU)

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`jr $ra`	000000	11111	10000	00000	00000	001000

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`syscall`	000000	00000	00000	00000	00000	001100


Sommario

Il linguaggio macchina: le istruzioni di tipo R

Le istruzioni di tipo I

Le istruzioni di tipo J

Modalità di indirizzamento

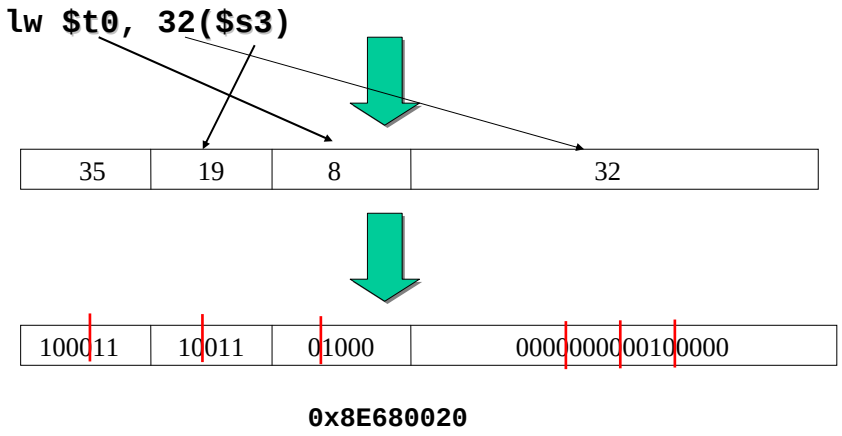
Trattamento delle costanti


Formato istruzioni di tipo I

op	rs	rt	costante
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).
- 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.

Istruzioni di tipo I: esempio



Istruzioni di tipo I: esempi

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

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $t0, $s3, 32`	001000	10011	01000	0000 0000 0010 0000

Dal C al linguaggio macchina

A[300] = h + A[300]

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

$s2 ⟶ h

$t1 ⟶ Indirizzo base di A

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

Versione I di istruzioni aritmetico-logiche

Nome campo	op	rs	rt	costante			
Dimensione	6-bit	5-bit	5-bit	16-bit			
addi $s1, $s2, 4	001000	10001	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 controllo di flusso (salto)

- Il PC viene incrementato di 4 (byte) durante l'esecuzione di un'istruzione.
- Salti condizionati relativi (beq, bne...) – Formato I:
  - 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 (j, jal...) – Formato J:
  - Il salto viene sempre eseguito.
  - L'indirizzo di destinazione del salto è un indirizzo assoluto di memoria.

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).

Esempio

Assembly		Assembly con le etichette risolte
Loop:	**add $t1, $s3, $s3**	**80000: add $t1, $s3, $s3**
	……	**……**
	bne $t0, $s5, Exit	**80016: bne $t0, $s5, 8**
	add $s3, $s3, $s4	**80020: add $s3, $s3, $s4**
	beq $t0,$s5, Loop	**80024: beq $t0,$s5, -28**
Exit:		**80028:**
	……	

Nota: quando si esegue la **bne**, PC punta già all'istruzione successiva (e.g. bne -> PC = 80020)

Analisi dell'offset

- Per il **principio di località** degli indirizzi di memoria è utile calcolare l'indirizzo di destinazione del salto come **offset** rispetto all'istruzione corrente.
- Nelle architetture MIPS tutte le istruzioni hanno 32 bit (RISC) e sono allineate al MSB. I due bit meno significativi dell'indirizzo delle istruzioni sono quindi sempre 00
- Per l'offset si hanno a disposizione solo 16-bit del campo `indirizzo.` Questo indirizzo **è** espresso relativamente al PC. **(PC-relative address).**
- Inoltre sfrutto al massimo i 16 bit $\Rightarrow$ rappresentano un offset di **parola (PC-relative *word* address).**
- Una istruzione di **salto** può indirizzare **parole** nell'intervallo $-2^{15} \div +2^{15}-1$ rispetto all'indirizzo base.
- 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

Esempio: `bne $s0, $s1, L1`

`L1 = (Ind_salto – PC) / 4`

Offset

L'assemblatore sostituisce l'etichetta **`L1`** con l'indirizzo **di parola** relativo a PC: **(L1-PC)/4**

- Il PC contiene già l'indirizzo dell'istruzione successiva al salto
- La divisione per 4 serve per calcolare lo spiazzamento in numero di parole.

Esempio

Assembly		Assembly	Linguaggio Macchina
Loop:	**add $t1, $s3, $s3**	**add $t1, $s3, $s3**	**80000: 0 19 19 9 0 32**
	……	**……**	
	……	**……**	
	……	**……**	
	bne $t0, $s5, Exit	**bne $t0, $s5, 8**	**80016: 5 8 21 2**
	add $s3, $s3, $s4	**add $s3, $s3, $s4**	**80020:**
	beq $t0,$s5, Loop	**beq $t0,$s5, -28**	**80024: 4 8 21 -7**
Exit:			**80028: (Exit) ...**
	……		

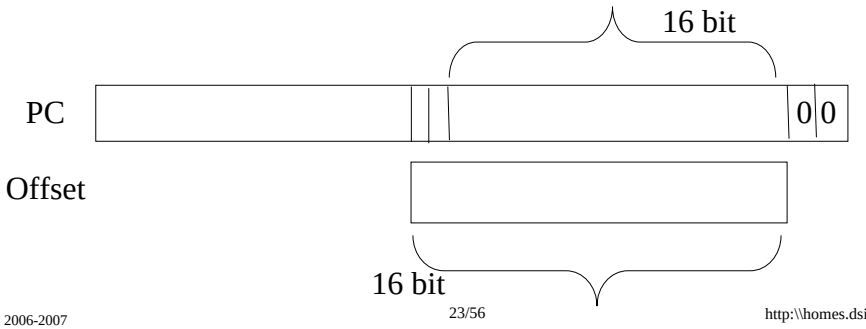
2 = (80028 - 80020) / 4
-7 = (80000 - 80028) / 4

Nota: quando si esegue la **bne**, PC punta già all'istruzione successiva (e.g. bne -> PC = 80020)

Allargamento dello spazio di indirizzamento

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

Considero 64Mword invece di 64Mbyte. Lo spazio indirizzabile all'interno del segmento di testo è di 64Mword * 4 = 256Mbyte.



Formato istruzioni di salto condizionato

op	rs	rt	costante
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;
  - **`costante`** riporta lo spiazzamento (offset). Il valore del campo indirizzo può essere negativo (salti all'indietro)


Branch: esempi

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 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, -100`	000100	10001	10010	1111 1111 1110 0111

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


Sommario

Il linguaggio macchina: le istruzioni di tipo R

Le istruzioni di tipo I

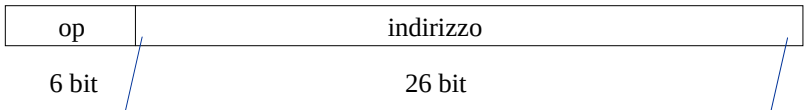
Le istruzioni di tipo J

Modalità di indirizzamento

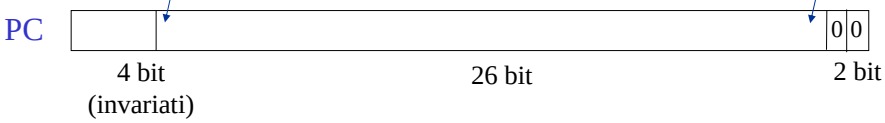
Trattamento delle costanti

Formato istruzioni di tipo J

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



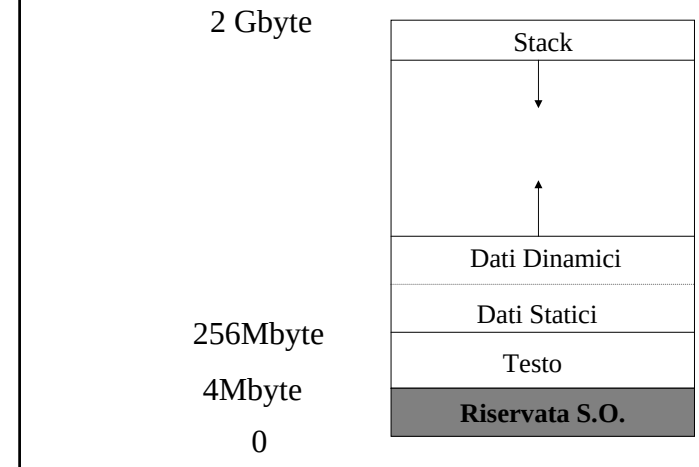
- In questo caso, i campi hanno il seguente significato:
  - **op** indica il tipo di operazione**;**
  - **indirizzo** (composto da **26-bit**) riporta una parte (26 bit su 32) dell'indirizzo **assoluto** di destinazione del salto.
- I 26-bit del campo **indirizzo** rappresentano un indirizzo di parola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



Istruzioni di salto incondizionato

- 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) = 256 Mbyte.
- 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.
- La memoria testo (S.O. + codice) è compresa tra 0 e 256Mbyte (2^{28}), i rimanenti 4 bit saranno 0000.

Organizzazione logica della memoria



Esempio

```
Loop:   add $t1, $s3, $s3      80000: 0 19 19 9 0 32      do {   t1 = s3*2;
        ……                     ……
        ……                     ……
        ……                     ……
        bne $t0, $s5, Exit     80016: 5 8 21  2               if (t1 != s5) break;
        add $s3, $s3, $s4      80020: 0 19 20 19 0 32         s3 +=s4;
        beq $t0, $s5, Loop     80024: 4 8 21 -7           } while (t0 = = s5);
Exit:                          80028: Exit ...
```

```
Loop:   add $t1, $s3, $s3      80000: 0 19 19 9 0 32      Loop:   t1 = s3*2
        ……                     ……
        ……                     ……
        ……                     ……
    bne $t0, $s5, Exit         80016: 5 8 21  2                   if (t1 != s5) break;
        add $s3, $s3, $s4      80020: 0 19 20 19 0 32             s3 +=s4;
        j Loop    (j 80000)    80024: 2 20000                     goto Loop;
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

32 = 100(00)

Nome campo	op			indirizzo			
Dimensione	6-bit			26-bit			
j 80000	000010	00	00000	0001	0100 1110	0010	0000

80000 = 000 0000 0001 0011 1000 1000 00(00)


Salti incondizionati indiretti

`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 = 8 Gbyte > intero spazio di memoria)

0	rs	0	0	0	8

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		
J	op	indirizzo				

Formato R ed operazioni logico-matematiche

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

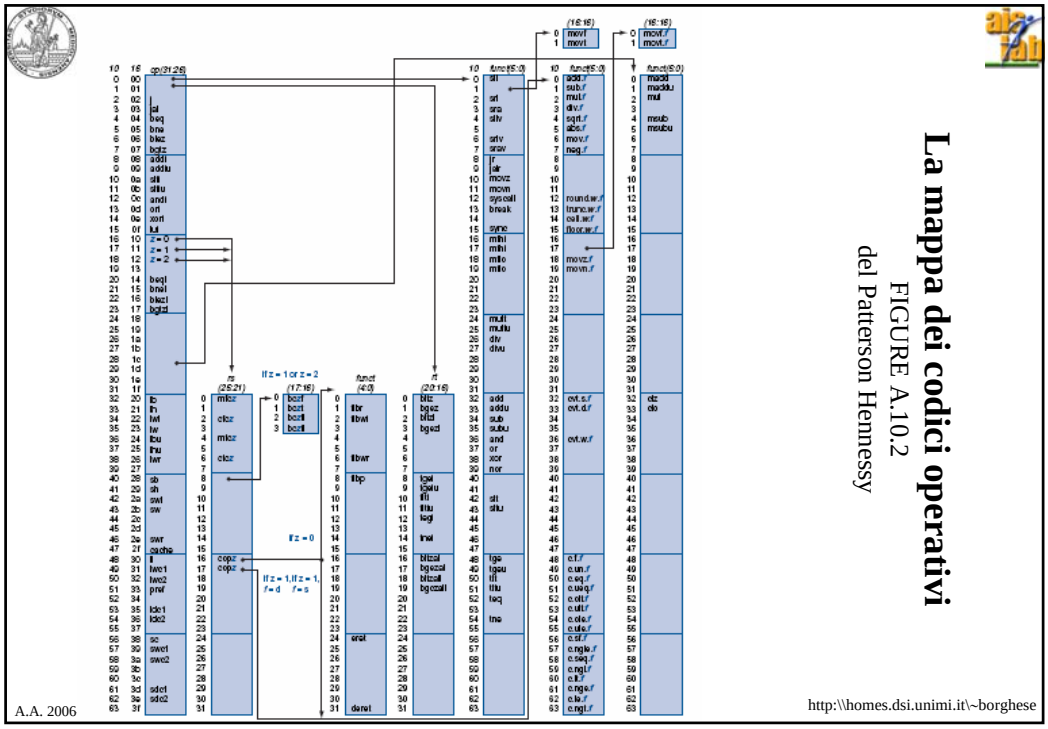
Le operazioni logico-matematiche di tipo R hanno codice operativo 0.

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)



La mappa dei codici operativi

FIGURE A.10.2

del Patterson Hennessy

Sommario

Il linguaggio macchina: le istruzioni di tipo R

Le istruzioni di tipo I

Le istruzioni di tipo J

Modalità di indirizzamento

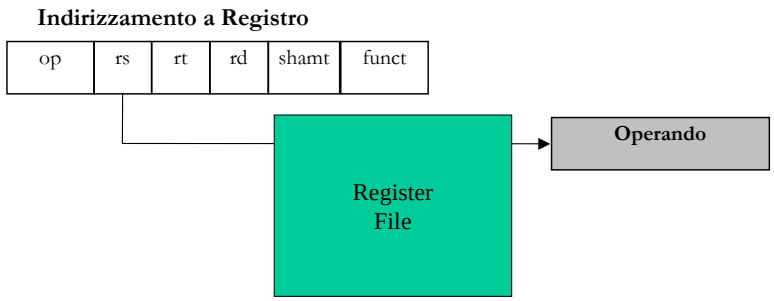
Trattamento delle costanti


Modalità di indirizzamento

- Le modalità di indirizzamento indicano le diverse modalità attraverso le quali far riferimento ai dati ed alle istruzioni in memoria e nel register file.
- 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.`**
- 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 (numero = indirizzo) del registro è specificato nell'istruzione.



Esempio di indirizzamento a registro

- Le istruzioni che usano **solamente** 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):

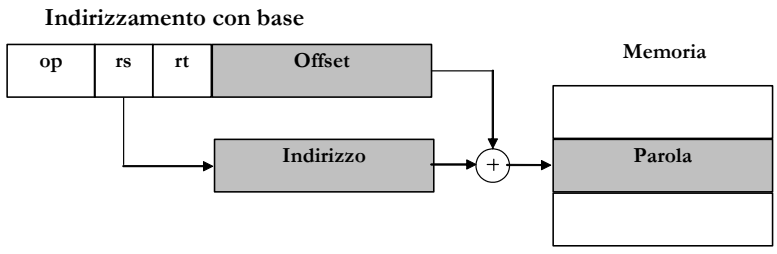
Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s1, $s1, 4`	001000	10001	10001	0000 0000 0000 0100

- 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.

Esempio di 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]`**

L'indirizzo e' espresso in numero di byte.

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

Problema con load/store

Per caricare dalla memoria occorrono due operazioni:
- Caricare il base address del vettore.
- Caricare l'offset.

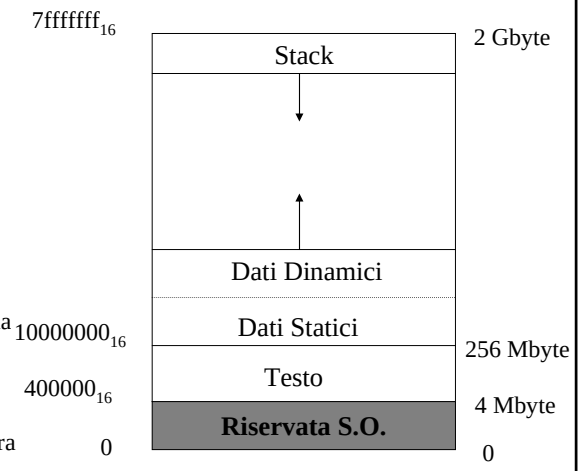
Indirizzo = base_address + offset.

Si può indirizzare una cella di memoria spostando l'indirizzo base o cambiando il valore dell'offset.

Esempio: lw $t0, -32k($g0) punta alla prima locazione dei Dati Statici.

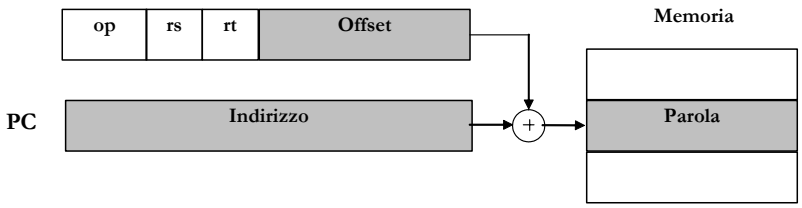
lui $gp, 0x4000

Sono facilitati gli accessi ai dati compresi tra 256Mbyte e 256,064Mbyte.



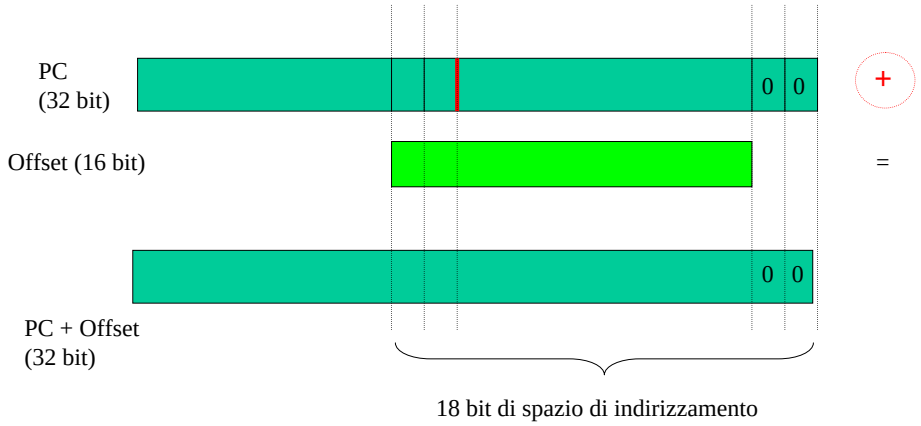
Indirizzamento relativo al PC

- L'**istruzione** 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.

Utilizzo dell'offset



Esempio di indirizzamento relativo al PC

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

•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.

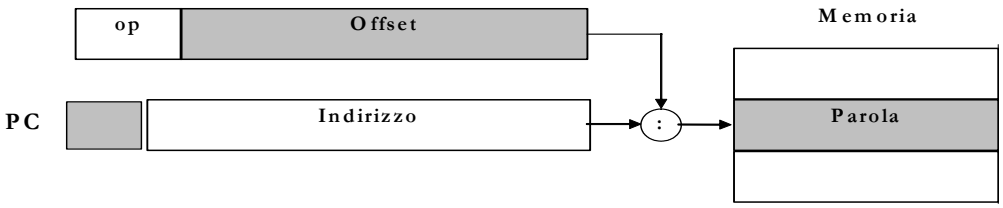
•Avendo a disposizione *16* bit di *Offset* ⇒ è possibile saltare in un'area tra -2^{15} e $+2^{15}-1$ parole rispetto all'istruzione corrente.

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

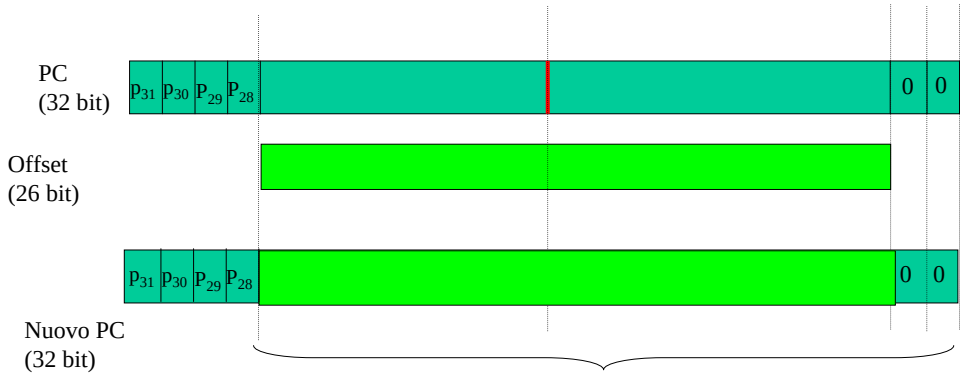
Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`bne $s1, $s2, 100`	000101	10001	10010	0000 0000 0001 1001

Indirizzamento pseudo-diretto

- Una parte dell'indirizzo è presente come valore costante nell'istruzione ma deve essere completato nei suoi bit più significativi. Questa costante si può intendere come offset rispetto alla posizione 0.
- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo J.
- 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.



Utilizzo dell'offset



28 bit di spazio di indirizzamento = 256Mbyte

Esempio di 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

Sommario

Il linguaggio macchina: le istruzioni di tipo R

Le istruzioni di tipo I

Le istruzioni di tipo J

Modalità di indirizzamento

Trattamento delle costanti

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)
  - utilizzo di modalità di **indirizzamento immediato (tipo I).**


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.


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, ad esempio, di **ori, andi,...** specifica i rimanenti 16 bit meno significativi della costante.
- Il registro **`$at`** è riservato all'assemblatore per creare costanti su 32-bit (costanti 'lunghe').

Esempio di caricamento costante di 32 bit

Si consideri la costante su 32 bit: C = 0000 0000 0000 0110 0001 1010 1000 0000 = **400,000 in decimale**

lui $s0, 6 #6 = 0000 0000 0000 0110

valore di **$s0:** 0000 0000 0000 0110 0000 0000 0000 0000 = 6 x 2^{16} = 393,216

addiu $s0, $s0, 6724 # 6724 = 0001 1010 1000 0000

valore di **$s0:** 0000 0000 0011 1101 0001 1010 1000 0000

Si consideri la costante su 32-bit: C = **118345_{10} = 0x1CE49 =**

C = 0000 0000 0000 0001 1100 1110 0100 1001

= 1_{10} = 52809_{10}

C = 1 x 2^{16} + 52809 = 118345

Si può rappresentare con la pseudo-istruzione: **li $t1, 118345** **# $t1 ← 118345**

Sommario

Il linguaggio macchina: le istruzioni di tipo R

Le istruzioni di tipo I

Le istruzioni di tipo J

Modalità di indirizzamento

Trattamento delle costanti