Linguaggio Assembler

Corso di Architettura degli Elaboratori

Autilia Vitiello
Dip. di Informatica ed Applicazioni
Stecca 7, 2° piano, stanza 12
vitiello@dia.unisa.it
http://www.dia.unisa.it/~avitiello

Presentazione

- Corso di Architettura matricole congruo a 0:
  - prof. Alberto Negro
- Parte programmazione assembler
- Libro di testo
  - David A. Patterson, John L. Hennessy, “Struttura, organizzazione e progetto dei calcolatori (Interdipendenza tra hardware e software)”, Jackson Libri, Capitolo 2

Presentazione

- Esercitazioni in aula
- Ricevimento via email
  - vitiello@dia.unisa.it
- Ufficio
  - Dipartimento di Informatica ed Applicazioni
    Sala Dottorandi Dia - Stecca 7, 2° piano, stanza 12

Il linguaggio di un calcolatore

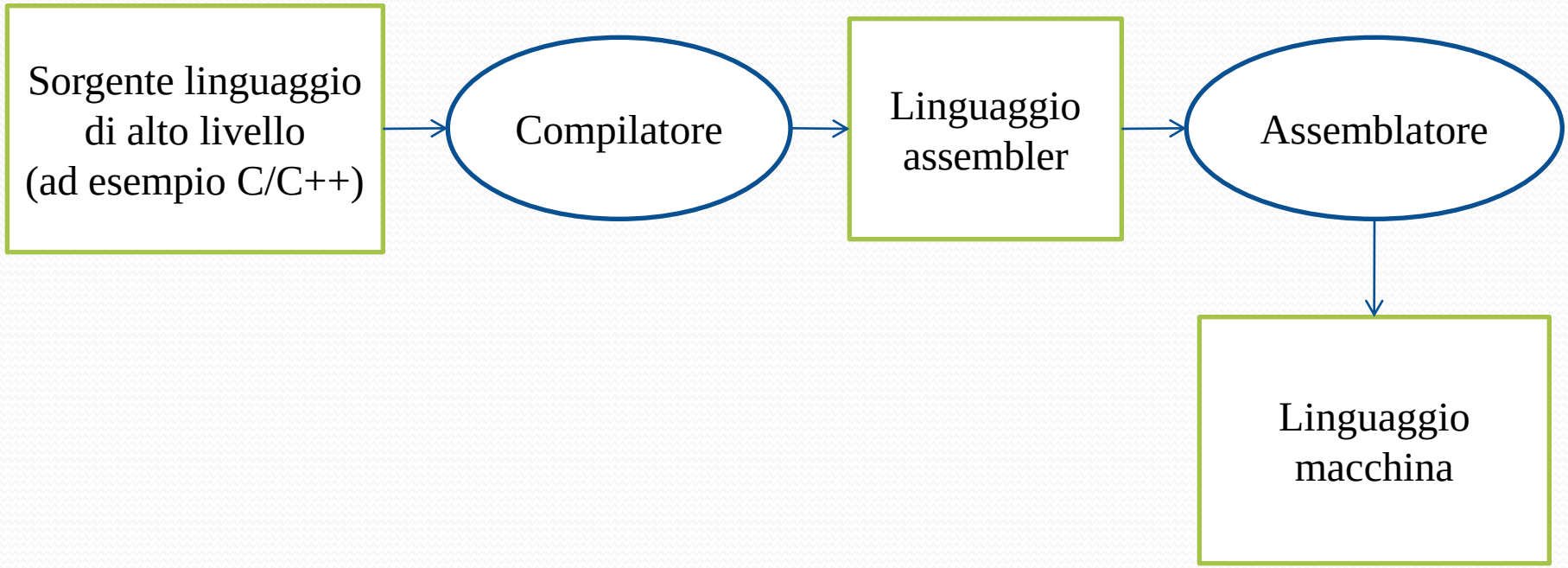
- Prende il nome di *linguaggio macchina* e rappresenta il linguaggio dei programmi eseguibili da un calcolatore;
- E' basato su un alfabeto binario: comprende solo due simboli 0/1 (che corrispondono a segnali elettrici di due livelli);
- La parola di un linguaggio macchina si definisce *istruzione*;
- L'*instruction set* è l'insieme di istruzioni comprese da un processore;
- Il set di istruzione che prendiamo in considerazione è quello del *processore MIPS* progettato durante gli anni '80.

La programmazione dei calcolatori

- Linguaggio macchina: programmazione attraverso numeri binari
  - attività tediosa e lunga
  - facile fare un errore
- Linguaggio Assembler
  - Intermediario;
  - Benefici: dipendenza dal processore che permette la produzioni di programmi più efficienti e (potenzialmente) più compatti;
- Linguaggio di alto livello: Java, C/C++, Fortan, Pascal,…
  - Benefici: Notazione naturale vicina al linguaggio corrente e alla notazione algebrica; Indipendenza dalla architettura (processore); permette il riuso del codice.

Compilatore e Assemblatore

- **Compilatore**: Programma che traduce istruzioni scritte in linguaggio di alto livello in linguaggio assembler
- **Assemblatore**: Programma che traduce la versione simbolica di un istruzione in versione binaria



Un esempio

- Linguaggio C

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

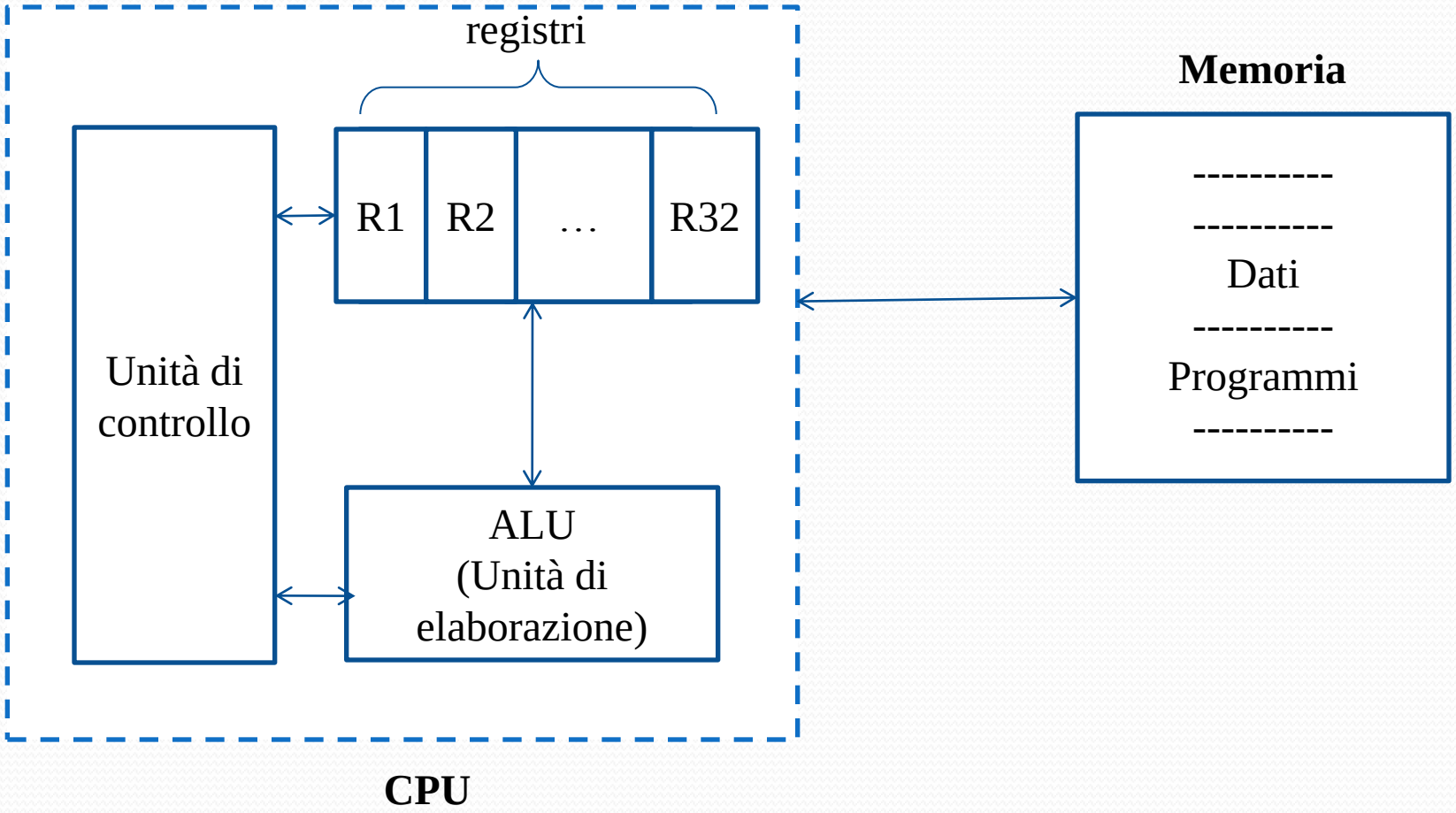
- Linguaggio Assembler

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

- Linguaggio macchina

op	rs	rt	rd	address/shamt	funct
100011	01001	01000		0000 0100 1011 0000	
000000	10010	01000	01000	00000	100000
101011	01001	01000		0000 0100 1011 0000	

Architettura MIPS



Architettura MIPS

- Basata sulla filosofia *Reduced Instruction Set Computer* (RISC)
  - Set di istruzioni semplice
  - Lunghezza fissa delle istruzioni
  - Gli operandi e il risultato sono memorizzati in registri

- Architettura di esempio per questo corso: MIPS32
  - Possiede 32 registri con una lunghezza fissa di 32 bit.

Registri MIPS

#		
0	$0	Sempre uguale a zero
1	$at	Registro riservato
2-3	$v0-$v1	Valore di ritorno di una chiamata a funzione
4-7	$a0-$a3	Primi 4 parametri di una chiamata a funzione (argomenti)
8-15	$t0-$t7	Variabili temporanee; non vanno preservate (salvate)
16-23	$s0-$s7	Variabili di funzione; vanno preservate
24-25	$t8-$t9	Altre 2 variabili temporanee
26-27	$k0-$k1	Registri usati dal kernel
28	$gp	Global pointer
29	$sp	Stack pointer
30	$fp/$s8	Stack frame pointer oppure variabile sub-routine
31	$ra	Indirizzo di ritorno dell’ultima sub-rountine chiamata

Registri MIPS

- Nota
  - Nelle esercitazioni non specificheremo quali registri useremo
  - Indicheremo i registri i maniera generica
    - $1, $2 …
  - $zero = 0

Set di istruzioni

- Istruzioni aritmetico-logiche
  - Operazioni tra dati presenti nei registri
- Istruzioni load/store
  - Sposta dati tra memoria e registri
- Istruzioni decisionali
  - Confronto tra dati nei registri
  - Salto condizionato e incondizionato

MIPS

- Set di istruzioni
  - add, sub
  - and, or
  - slt,
  - lw, sw
  - beq, bne
  - j, jr
  - addi, subi, slti

Rappresentazione in linguaggio macchina

- Istruzioni a 32 bit

- Diversi formati:
  - Tipo R:
    - add, sub, and, or e slt
  - Tipo I:
    - addi, beq, bne, lw, sw
  - Tipo J:
    - j (jump)

Istruzioni Aritmetiche

- Ogni istruzione aritmetica contiene:
  - Esattamente un'operazione;
  - Esattamente tre variabili (istruzioni a tre operandi).
- Ogni istruzione è composta dalla sigla dell'operazione e da tre operandi con un posto ben preciso.
- Gli operandi sono registri
  - Non variabili

Istruzione ADD

- Per eseguire l'operazione di addizione si usa l'istruzione **add**
  - Primo posto assunto dal risultato (op1);
  - Secondo e terzo posto assunti dagli addendi (op2 e op3).

- Sintassi: **add op1, op2, op3**

Esempio

a = b + c

add $t1, $s1, $s2

Assegnazione

$t1 ← a,

$s1 ← b,

$s2 ← c,

Istruzione SUB

- Per eseguire l'operazione di sottrazione si usa l'istruzione **sub**
  - Primo posto assunto dal risultato (op1);
  - Secondo posto assunto dal minuendo (op2);
  - Terzo posto assunto dal sottraendo (op3).

- Sintassi: **sub op1, op2, op3**

Esempio

a = b - c

sub $t1, $s1, $s2

Assegnazione

$t1 ← a,

$s1 ← b,

$s2 ← c,

Esempio

a = (b + c) – (d - f) + g

```
add $t2, $s1, $s2
sub $t3, $s3, $s4
sub $t4, $t2, $t3
add $t1, $t4, $s5
```

Assegnazione

$t1 ← a,

$s1 ← b,

$s2 ← c,

$s3 ← d,

$s4 ← f,

$s5 ← g

Formato R

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

- 6 campi
  - op: codice operativo (*opcode*)
  - rs/rt: primo e secondo operando sorgente
  - rd: registro destinazione
  - shamt: shift amount (usato da op. di shift)
  - funct: codice funzione (seleziona un operazione specifica)

Esempio

a = b + c

add $t1, $s1, $s2

Assegnazione

($9) $t1 ← a,

($17) $s1 ← b,

($18) $s2 ← c,

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
0	17	18	9	0	32

Esempio

a = b - c

add $t1, $s1, $s2

Assegnazione

($9) $t1 ← a,

($17) $s1 ← b,

($18) $s2 ← c,

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
0	17	18	9	0	34

Istruzioni Load/Store

- Trasferimento memoria/registri

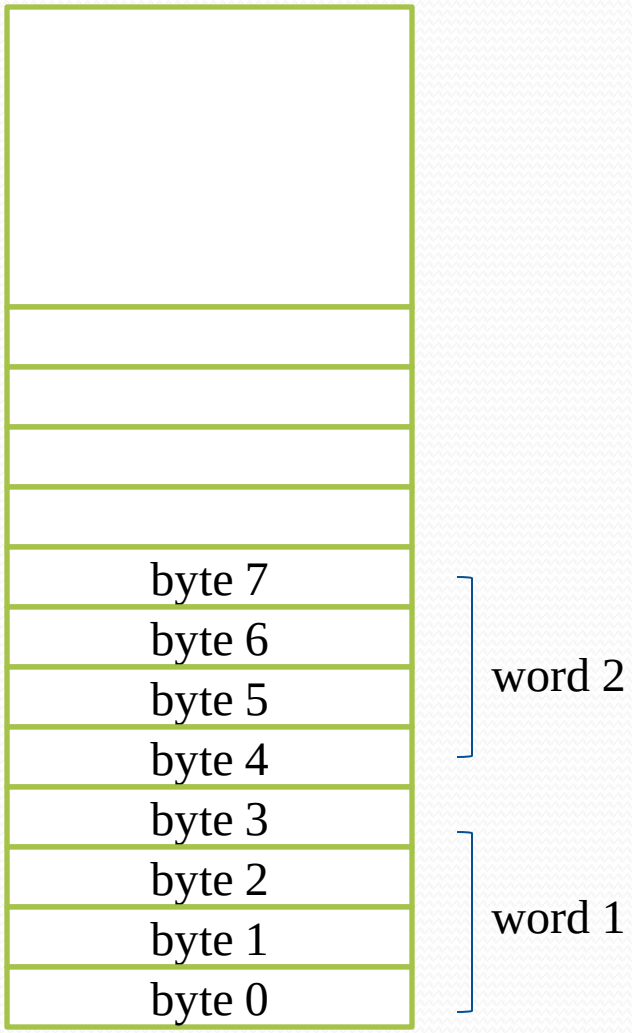
- Load word
  - Trasferimento di un dato dalla memoria ai registri
- Store word
  - Trasferimento di un dato dai registri alla memoria

- L'accesso alla memoria avviene attraverso un *indirizzo*.

Memoria

- Memoria come un vettore;
- Ogni locazione di memoria è di un byte (8 bit);
- Ogni locazione di memoria ha un indirizzo di 32 bit.

- Una parola è composta da 4 byte (pertanto si divide su 4 locazioni);
- L'indirizzo di una parola è l'indirizzo della prima locazione.



Istruzione LW

- Per eseguire l'operazione di trasferimento di un dato contenuto in un registro (ad esempio $s1) nella memoria all'indirizzo che indichiamo con *I* si usa l'istruzione **lw.**
- Si utilizza l'***indirizzamento indicizzato***:
  - *I* = indirizzo base + offset;
    - Indirizzo base è contenuto in un registro (ad esempio $s2)
    - Offset è un numero reale.
- Sintassi: **lw $s1, offset($s2)**

Istruzione SW

- Per eseguire l'operazione di trasferimento di un dato contenuto in memoria all'indirizzo *I* in un registro (ad esempio $s1) si usa l'istruzione **lw.**
- Si utilizza l'***indirizzamento indicizzato***:
  - *I* = indirizzo base + offset;
    - Indirizzo base è contenuto in un registro (ad esempio $s2)
    - Offset è un numero reale.
- Sintassi:

sw $s1, offset($s2)

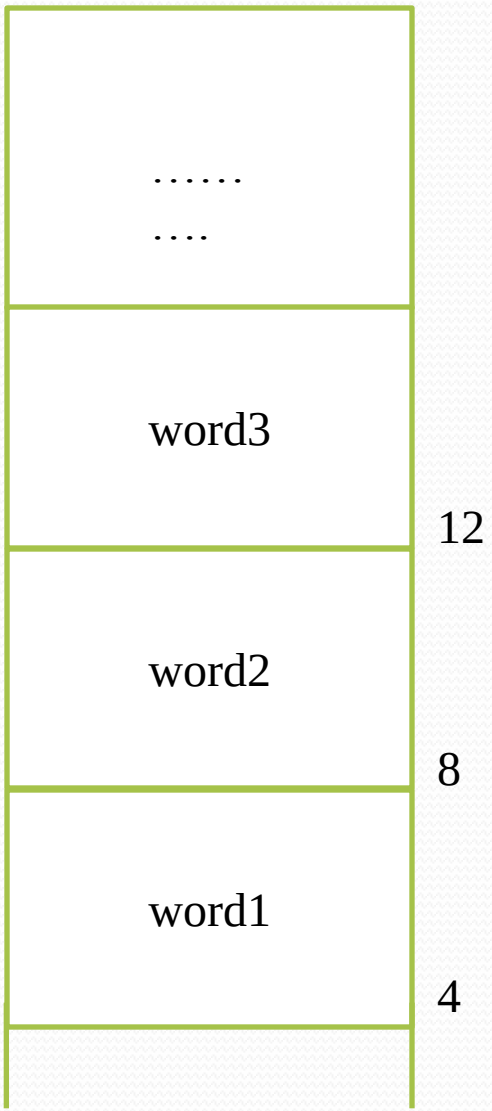
Memoria

- Esempio:
  - Caricare nel registro $s1 il dato contenuto all'indirizzo di memoria 12
  - Consideriamo l'indirizzo base 4 contenuto nel registro $s2
  - 12 = I = 4 + 8

lw $s1, 8($s2)

- Dopo l'esecuzione della lw:

$s1 **word3**



Memoria: gestione di vettori

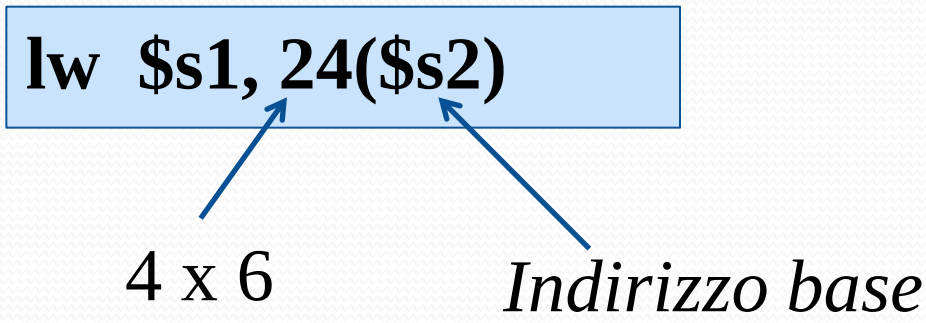
- Consideriamo un vettore A;
- Ogni elemento del vettore è memorizzato in una parola;

- Per calcolare l'indirizzo di memoria ***I*** dell'elemento A[**i**] è sufficiente conoscere l'indirizzo del primo elemento del vettore (A[0]) che denominiamo ***Indirizzo base***.

$$I = Indirizzo\ base + (4 \times i)$$

Memoria: gestione dei vettori

- Esempio
- Caricare nel registro $s1 l'elemento A[6] contenuto in memoria
  - Consideriamo contenuto in $s2 l'indirizzo dell'elemento A[0]



Gestione vettori: esempio

A[14] = b + A[6]

```
lw  $t0, 24($s2)
add $t0, $s1, $t0
sw $t0, 56($s2)
```

Assegnazione

$s1 ← b,

$s2 ← indirizzo di A[0]

Gestione dei vettori: esempio 2

A[i]=j;

Assegnazione
$s1 ← indirizzo di A[0]
$s2 ← i,
$s3 ← j,

- calcolo l'offset

```
add  $t1, $s2, $s2    # $t1 = 2 * i
add  $t1, $t1, $t1    # $t1 = 4 * i
```

- calcolo indirizzo di A[i]

```
add  $t1, $t1, $s1    # indirizzo base + offset
```

- memorizza j in A[i]

```
sw $s3, 0($t1)
```

Formato I

op	rs	rt	address/immediate
6 bit	5 bit	5 bit	16 bit

- 4 campi
  - op: codice operativo (*opcode*)
  - rs: registro base
  - rd: registro in cui scrivere (lw) o da cui prelevare il dato (sw)
  - address: numero intero (con segno) che rappresenta il valore dell'offset

Esempi

$s1 = $17

$s2 = $18

lw $s1, 24($s2)

op	rs	rt	address/immediate
6 bit	5 bit	5 bit	16 bit
35	18	17	24

sw $s1, 24($s2)

op	rs	rt	address/immediate
6 bit	5 bit	5 bit	16 bit
43	18	17	24

Istruzione ADDI

- Per eseguire l'operazione di addizione in presenza di costanti si usa l'istruzione **add immediate (addi)**
  - Si evitano così ulteriori operazioni;
  - 2 operandi nei registri e uno *immediate:*
    - Primo posto assunto dal risultato (op1);
    - Secondo posto assunto da un addendo contenuto in un registro (op2);
    - Terzo posto assunto dall'addendo costante (cost).
- Formato I
- Sintassi: **addi op1, op2, cost**

Istruzione ADDI: Formato I

a = b + 4

addi $t1, $s2, 4

Assegnazione

($9) $t1 ← a,

($18) $s2 ← b

op	rs	rt	address/immediate
6 bit	5 bit	5 bit	16 bit
8	9	18	4

Istruzioni di salto condizionato

- L'ordine di esecuzione di una lista di istruzioni può essere modificata mediante istruzioni di salto.
- Salto condizionato su uguaglianza: **beq**
- Sintassi: **beq $s1, $s2, num_salti**
  - Semantica: si salta all'istruzione con indirizzo *I* se il contenuto di $s1 è uguale al contenuto di $s2.
  - L'indirizzamento è di tipo ***PC-relative***, ossia si calcola *I* a partire dal valore contenuto nel Program Counter.

Indirizzamento PC-relative

- L'indirizzo *I* della locazione dove saltare si calcola nel seguente modo:

$$I = PC + \text{num\_bytes}$$

- Ricordiamo che PC (Program Counter) punta all'indirizzo dell'istruzione successiva alla corrente.
- num_bytes = numero di byte per raggiungere la locazione dove saltare a partire da PC.

$$\text{num\_bytes} = 4 \times \text{num\_salti}$$

- num_salti: numero di istruzioni da saltare per raggiungere l'istruzione da eseguire a partire dall'istruzione successiva alla corrente.

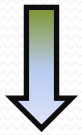
Esempio: costrutto if-then

```
if (i==j) go to L1
i = i+j;
L1:  j = j+4;
```

Assegnazione

$s1 ←i,

$s2 ← j



PC

Indirizzo dove saltare

4	beq $s1, $s2, +1
8	add $s1, $s1, $s2
12	addi $s2, $s2, 4

num_salti

***12 =I = 8* + (4 x 1)**

Istruzione beq: Formato I

beq $s1, $s2, +1

$s1 = $17

$s2 = $18

op	rs	rt	address/immediate
6 bit	5 bit	5 bit	16 bit
4	17	18	1

num_salti

Istruzioni di salto condizionato

- Esiste anche il salto condizionato su disuguaglianza: **bne** (*branch if not equal*)

- Sintassi: **bne $s1, $s2, num_salti**

  - Semantica: si salta all'istruzione con indirizzo *I* se il contenuto di $s1 è diverso dal contenuto di $s2.
  - Il calcolo dell'indirizzo è come la beq.

Istruzione set-on-less-than (slt)

- Sintassi: **slt $s1, $s2, $s3**
  - Confronta i contenuti nei registri $s2 e $s3;
  - assegna al registro $s1:
    - 1 se il contenuto di $s2 è minore del contenuto di $s3,
    - 0 in caso contrario.

- Formato R

op	rs	rt	rd	shamt	funct
6 bit	5 bit – s2	5 bit – s3	5 bit – s1	5 bit	6 bit
0	18	19	17	0	42

Esempio: costrutto if-then

```
if (i<j) go to L1
i = i+j;
L1:  j = j+4;
```

```
104   slt $t1, $s1, $s2
108   bne $t1, $zero,+1
112   add $s1, $s1, $s2
116   addi $s2, $s2, 4
```

Assegnazione

$s1 ←i,

$s2 ← j

Istruzione di salto non condizionato

- L'istruzione **j (jump)** è usata per eseguire un salto senza effettuare un test.
- Sintassi: **j '32'**
- Semantica: vai all'istruzione con indirizzo 32
- Formato J

op	address
6 bit	26 bit
2	32

- 2 campi
  - op: codice operativo (*opcode*)
  - address: indirizzo destinazione

Scorretto: l'indirizzamento è di tipo **Pseudo-direct**

Istruzioni di salto non condizionato

- Jump register (jr)
  - Salta all'indirizzo contenuto in un registro
- Sintassi: **jr $ra**
- Usata nei costrutti break/case e nel ritorno da funzione
- Formato R

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
0	31	0	0	0	8

Esempio: Costrutto if/else

```
if(i==j)  f = g+h;
else f = g-h;
```

Assegnazione
$s1 ←g,
$s2 ← h,
$s3 ←i,
$s4 ← j,
$s0 ←f

```
 4  bne $s3, $s4, +2          # salta all'else
 8  add  $s0, $s1, $s2        # f=g+h
12  j '20'                      # salta alla fine
16  sub $s0, $s1, $s2         # f=g-h
20
```

Istruzioni logiche

- **and**
  - AND logico bit a bit
  - Sintassi: **and $s1, $s2, $s3**
  - $s1 = $s2 & $s3
- **or**
  - OR logico bit a bit
  - Sintassi: **or $s1, $s2, $s3**
  - $s1 = $s2 | $s3
- Formato R

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

Esercizio 1

- Scrivere le istruzioni MIPS necessarie per copiare un valore da un registro ad un altro

Assegnazione
$s1 ←g,
$s2 ← b,

```
4   add $1,$2,$0          #g=b+0
```

Esercizio 2

- Scrivere le istruzioni MIPS necessarie per azzerare un registro

Assegnazione
$s1 ←g,

```
4   add $1,$0,$0          #g=0+0
```

Esercizio 3

- Scrivere un programma che, assumendo di avere dei valori nei registri $s0, $s1 e $s2 scriva nella parola con indirizzo 1000 la somma dei tre registri.

```
 4   add $t1,$s0,$s1          # somma $s0 e $s1
 8   add $t1,$t1,$s2          # somma dei tre registri
12   sw  $t1, 1000($0)        # si memorizza la somma
```

Esercizio 4

- Scrivere un programma che, assumendo di avere nel registro $3 il valore 10, nel registro $1 il valore 4, nel registro $2 il valore 3, scriva nelle parole con indirizzo 1000, 1004, 1008, rispettivamente il valore 10, 13 e 16.

```
 4   sw $3, 1000($0)        # memorizzo 10
 8   add $4,$3,$2           # sommo 10 e 3
12   sw  $4, 1004($0)       # memorizzo 13
16   add $4,$4,$2           # sommo 13 e 3
20   sw  $4, 1008 ($0)      # memorizzo 16
```

Esercizio 5

- Scrivere le istruzioni MIPS per la seguente istruzione C:

A[3] = A[2] + A[0]

Assegnazione
$3 ← indirizzo base di A

```
 4   lw $1, 0($3)          # carico A[0] in $1
 8   lw $2, 8($3)          # carico A[2] in $2
12   add $1,$1,$2          # A[0] + A[2] in $1
16   sw  $1, 12 ($3)       # memorizzo il contenuto di $1 in A[3]
```

Esercizio 6

- Scrivere le istruzioni MIPS per le seguenti istruzioni in C:

```
h = i;
A[h] = 0;
a = A[h+1]
```

Assegnazione
$1 ← i,
$2 ← h,
$3 ← a,
$4 ← indirizzo base di A

```
 4  add $2,$1,$0      # h=i+0;
 8  add $5,$2,$2      # h+h in $5
12  add $5,$5,$5      # 2h+2h in $5
16  add $5,$5,$4      #$5 contiene indirizzo di A[h]
20  sw $0, 0($5)      # memorizzo 0 in A[h]
24  lw $6, 4($5)      # carico A[h+1] in $6
28  add $3,$6,$0      # a = A[h+1] + 0
```

Esercizio 7

```
if (i==h)
    g = B[i] + C[i];
else
    A[i] = g;
h = g;
```

Assegnazione
$7 ← i,
$6 ← h,
$5 ← g,
$1 ← indirizzo base di A,
$2 ← indirizzo base di B,
$3 ← indirizzo base di C

```
 4  add $10,$7,$7          # i+i in $10
 8  add $10,$10,$10        # 2i + 2i in $10
12  bne $7,$6, +6          # i!=h salta all'istruzione 40
16  add $11,$10,$2         # indirizzo di B[i] in $11
20  lw $12, 0($11)         #$12 contiene B[i]
24  add $13, $10, $3       # indirizzo di C[i] in $13
28  lw $14, 0($13)         # carico C[i] in $14
32  add $5,$12,$14         # g = B[i] + C[i]
36  j  48                  # salto all'istruzione 48
40  add $15,$10,$1         # indirizzo di A[i] in $15
44  sw  $5, 0($15)         # memorizzo g in A[i]
48  add $7,$5,$0           # h=g+0
```

Esercizio 8

```
i = h;
if (i!=0)
      A[i] = B[i];
else
      A[0] = h;
h = i;
```

Assegnazione
$1 ← i,
$2 ← h,
$7 ← indirizzo base di A,
$6 ← indirizzo base di B,

```
 4   add $1,$2,$0          # i=h+0
 8   beq $1,$0, +7         # se i==0 salto all'istruzione 40
12   add $4,$1,$1          # i+i in $4
16   add $4,$4,$4          # 2i+2i in $4
20   add $4, $4, $6        # indirizzo di B[i] in $4
24   lw $5, 0($4)          # $5 contiene B[i]
28   add $10,$4,$7         # indirizzo di A[i] in $10
32   sw $5, 0($10)         # memorizzo B[i] in A[i]
36   j  44                 # salto all'istruzione 48
40   sw  $2, 0($7)         # memorizzo h in A[0]
44   add $2,$1,$0          # h=i+0
```

Esercizio 9: ciclo for

```
g=0;
for (i=0; i<100; i++)
        g=g+2;
h = g;
```

Assegnazione
$1 ←i,
$2 ← g,
$3 ← h

```
4   add $2,$0,$0        #g=0;
8   add $1,$0,$0        #i=0;
12  slt $4, $1,100      #in $s4 = 1 se i<100, 0 altrimenti
16  beq $4,$0, +3       # se i>=100 salta all'istruzione 32
20  addi $2,$2,2        #g=g+2
24  addi $1,$1,1        #i++
28  j  12               #torno al test
32  add $3, $2,$0       # istruzione h=g;
36  ……
```

Esercizio 10

- Dato un vettore A di 100 interi memorizzato a partire dalla locazione 2000, scrivere un programma MIPS che lo copi nel vettore B memorizzato a partire dalla locazione 3000.

Esercizio 10

- Dato un vettore A di 100 interi memorizzato a partire dalla locazione 2000, scrivere un programma MIPS che lo copi nel vettore B memorizzato a partire dalla locazione 3000.

```
for(i=0; i<100; i++)
    B[i] = A[i];
```

Esercizio 10

```
for(i=0; i<100; i++)
    B[i] = A[i];
```

Assegnazione

$1 ←i,

```
 4   add $1,$0,$0          #i=0;
 8   slti $2, $1,100       # in $2 c'è 1 se i<100, 0 altrimenti
12   beq $2,$0, +6         # se i>=100 salta all'istruzione 40
16   add $3,$1,1           #i+i in $3
20   add $3,$3,$3          #2i+2i in $3
24   lw $4, 2000($3)       #carico in $4 A[i]
28   sw $4, 3000 ($3)      #memorizzo A[i] in B[i]
32   addi $1, $1,1         #i++
36   j  8                  #torno al test
40   …..
```

Esercizio 10: versione alternativa

```
for(i=0; i<100; i++)
    B[i] = A[i];
```

Assegnazione

$1 ←i,

```
 0   add $3,$0,$0        # spiazzamento in $3 posto a 0
 4   add $1,$0,$0        #i=0;
 8   slti $2, $1,100     # in $2 c'è 1 se i<100, 0 altrimenti
12   beq $2,$0, +5       # se i>=100 salta all'istruzione 36
16   lw $4, 2000($3)     #carico in $4 A[i]
20   sw $4, 3000 ($3)    #memorizzo A[i] in B[i]
24   addi $3,$3,4        #aggiorno spiazzamento
28   addi $1, $1,1       #i++
32   j  8                #torno al test
36   …..
```

Esercizio 11

- Dato un vettore A di 100 interi memorizzato a partire dalla locazione 1000, scrivere un programma che costruisca il vettore B che contiene tutti gli elementi di A ma in ordine inverso.

Esercizio 11

- Dato un vettore A di 100 interi memorizzato a partire dalla locazione 1000, scrivere un programma che costruisca il vettore B che contiene tutti gli elementi di A ma in ordine inverso.

```
for (i=0,k=99; i<100; i++,k--)
    B[i] = A[k];
```

- oppure

```
for(i=0; i<100; i++)
    B[i] = A[99-i];
```

Esercizio 12

- Scrivere un programma in Assembler MIPS che
  - partendo da un array A di 100 elementi, memorizzato a partire dalla locazione 1004
  - somma solamente gli elementi di A minori di 10.

Esercizio 12

- Scrivere un programma in Assembler MIPS che
  - partendo da un array A di 100 elementi, memorizzato a partire dalla locazione 1004
  - somma solamente gli elementi di A minori di 10.

```
sum=0;
for (i=0; i<100; i++)
    if (A[i]<10)
            sum=sum+A[i];
```

Esercizio 12

```
sum=0;
for (i=0; i<100; i++)
      if (A[i]<10)
            sum=sum+A[i];
```

Assegnazione
$1 ← sum,
$2 ← i

```
 0   add $7,$0,$0        # spiazzamento in $7 posto a 0
 4   add $1,$0,$0        #sum=0;
 8   add $2,$0,$0        #i=0;
12   slti $3, $2,100     # in $3 c'è 1 se i<100, 0 altrimenti
16   beq $3,$0, +7       # se i>=100 salta all'istruzione 48
20   lw $6, 1004($7)     #carico in $6 A[i]
24   slti $10,$6,10      # in $10 c'è 1 se A[i]<10, 0 altrimenti
28   beq $10,$0, +1      # se A[i] >=10 salta all'istruzione 36
32   add $1,$1,$6        # sum=sum + A[i]
36   addi $7,$7,4        #aggiorno spiazzamento
40   addi $2, $2,1       #i++
44   j    12             #torno al test del for
48   …..
```

Esercizio 13

- Scrivere un programma Assembler MIPS che cerchi l'elemento massimo nel vettore A (memorizzato a partire dalla locazione 500) di 100 elementi e memorizzi il risultato nella locazione 2000.

Esercizio 13

- Scrivere un programma Assembler MIPS che cerchi l'elemento massimo nel vettore A (memorizzato a partire dalla locazione 500) di 100 elementi e memorizzi il risultato nella locazione 2000.

```
max=A[0];
for (i=1; i<100; i++)
    if (A[i] > max)
        max=A[i];
```

Esercizio 13

```
max=A[0];
for (i=1; i<100; i++)
      if (A[i] > max)
            max=A[i];
```

Assegnazione
$1 ← max,
$2 ← i

```
4   lw  $1, 500 ($0)      #carico A[0] in $1
8   addi $3,$0,4          # spiazzamento in $3 inizializzato a 4
12  addi $2,$0,1          #i=1;
16  slti $4, $2,100       # in $4 c'è 1 se i<100, 0 altrimenti
20  beq $4,$0, +7         # se i>=100 salta all'istruzione 52
24  lw $5, 500($3)        #carico in $5A[i]
28  slt $6,$1,$5          # in $6 c'è 1 se max < A[i], 0 altrimenti
32  beq $6,$0, +1         # se  max >=A[i]  salta all'istruzione 40
36  lw $1, 500($3)        # max=A[i];
40  addi $3,$3,4          #aggiorno spiazzamento
44  addi $2, $2,1         #i++
48  j 16                  #torno al test del for
52  sw  $1, 2000($0)      #memorizzo max nella locazione 2000
```

Esercizio 14

- Dati due vettori A e B di 100 interi memorizzati, rispettivamente, a partire dalle locazioni 2000 e 3000, scrivere un programma in assembler MIPS
  - che crei un vettore C (memorizzato a partire dalla locazione 4000) tale che in ogni posizione C[i] venga memorizzato il massimo tra A[i] e B[i].

Esercizio 14

- Dati due vettori A e B di 100 interi memorizzati, rispettivamente, a partire dalle locazioni 2000 e 3000, scrivere un programma in assembler MIPS
  - che crei un vettore C (memorizzato a partire dalla locazione 4000) tale che in ogni posizione C[i] venga memorizzato il massimo tra A[i] e B[i].

```
for (i=0; i<100; i++){
    if (A[i] < B[i])
            C[i]=B[i];
    else C[i]=A[i];
}
```

Esercizio 14

```
for (i=0; i<100; i++)
      if (A[i] < B[i])
            C[i]=B[i];
      else C[i]=A[i];
```

Assegnazione

$1 ← i

```
4    add  $1, $0, $0          #i=0;
8    add $2,$0,$0             # spiazzamento in $2 inizializzato a 0
12   slti $3, $1,100          # in $3 c'è 1 se i<100, 0 altrimenti
16   beq $3,$0, +10           # se i>=100 salta all'istruzione 60
20   lw $4, 2000($2)          #carico A[i] in $4
24   lw $5, 3000($2)          #carico B[i] in $5
28   slt $3,$4,$5             # in $3 c'è 1 se A[i]<B[i], 0 altrimenti
32   beq $3,$0, +2            # se A[i]>=B[i]  salta all'istruzione 44
36   sw $5, 4000($2)          # memorizzo B[i] in C[i];
40   j 48                     # salto il corpo 2 dell'if/else e vado direttamente all'incremento
44   sw $4, 4000($2)          #memorizzo A[i] in C[i];
48   addi $1,$1,1             #i=i+1;
52   addi $2,$2, 4            #aggiorno spiazzamento
56   j 12                     #torno al test del for
60   ….
```

Fine

- Per ricevimento inviare una e-mail
  - all'indirizzo: vitiello@dia.unisa.it
  - con oggetto "Tutorato di Architettura"