



Il Linguaggio Assembly: Controllo del flusso

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

Università degli Studi di Milano

Riferimento sul Patterson: 2.6, 2.7



Sommario

Istruzioni MIPS di controllo di flusso (condizioni di uguaglianza).

Istruzioni MIPS di controllo di flusso (condizioni di disuguaglianza).

Costrutto switch



Le strutture di controllo



Strutture di controllo:

- cicli (for $i = 0; i < n; i++$)
- condizioni (if then else)
- salto (goto)

- Queste istruzioni (branch & jump):
 - Alterano l'ordine sequenziale di esecuzione delle istruzioni:
 - La prossima istruzione da eseguire non è l'istruzione successiva all'istruzione corrente
 - Permettono di eseguire cicli e condizioni

- In assembly le strutture di controllo sono molto semplici e primitive



Istruzioni di salto condizionato



- Istruzioni di salto: viene caricato un nuovo indirizzo nel contatore di programma (PC) invece dell'indirizzo seguente l'indirizzo di salto secondo l'ordine sequenziale delle istruzioni.
- Istruzioni di *salto condizionato* (*conditional branch*): il salto viene eseguito solo se una certa condizione risulta soddisfatta.
- Esempi: **beq** (*branch on equal*) e **bne** (*branch on not equal*)
 - `beq rs, rt, L1 # go to L1 if (rs == rt)`
 - `bne rs, rt, L1 # go to L1 if (rs != rt)`



Istruzioni di salto incondizionato



- Istruzioni di salto: viene caricato un nuovo indirizzo nel contatore di programma (PC) invece dell'indirizzo seguente l'indirizzo di salto secondo l'ordine sequenziale delle istruzioni.
- Istruzioni di *salto incondizionato* (*unconditional jump*): il salto viene sempre eseguito.

Esempi: **j** (jump) e **jr** (jump register) e **jal** (jump and link)

```
j    L1           # go to L1
jr   r31         # go to add. contained in r31
jal  L1          # go to L1. Save add. of next
                    # instruction in reg. ra (ad
                    # esempio return address).
```



Esempio if ... then



Codice C:

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

- Si suppone che le variabili **f**, **g**, **h**, **i** e **j** siano associate rispettivamente ai registri **\$s0**, **\$s1**, **\$s2**, **\$s3** e **\$s4**

La condizione viene trasformata in codice C implementabile in Assembly:

```
if (i != j)
    goto Etichetta;
f=g+h;
Etichetta:
```

Codice MIPS:

```
bne $s3, $s4, Etichetta    # go to Etichetta if i != j
add $s0, $s1, $s2          # f=g+h is skipped if i != j
Etichetta:
```



Esempio if... then ... else



Codice C:

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

- Si suppone che le variabili **f**, **g**, **h**, **i** e **j** siano associate rispettivamente ai registri **\$s0**, **\$s1**, **\$s2**, **\$s3** e **\$s4**

Codice MIPS:

```
bne $s3, $s4, Else # go to Else if i≠j
add $s0, $s1, $s2 # f=g+h skipped if i ≠ j
j End # go to End
Else: sub $s0, $s1, $s2 # f=g-h skipped if i = j
End:
```

A.A. 2006-2007

7/22

<http://homes.dsi.unimi.it/~borgnese>



Esempio: do ... while (repeate)



Codice C:

```
do
    g = g + A[i];
    i = i + j;
while (i != h)
```

- Si suppone che **g** e **h** siano associate a **\$s1** e **\$s2**, **i** e **j** associate a **\$s3** e **\$s4** e che **\$s5** contenga il *base address* di **A = A[0]**.
- Si noti che il corpo del ciclo modifica la variabile **i**
⇒ devo moltiplicare **i** per **4** ad ogni iterazione del ciclo per indirizzare il vettore **A**.
- Indirizzamento della memoria: indirizzo Base + Offset.
- Strategia utilizzata: sposto l'indirizzo base e considero sempre offset = 0.

A.A. 2006-2007

8/22

<http://homes.dsi.unimi.it/~borgnese>



Esempio: do ... while

Codice C modificato:

```

i = 0;
Ciclo: g = g + A[i];
        i = i + j;
        if (i != h) goto Ciclo;
        g e h -> $s1 e $s2
        i e j -> $s3 e $s4
        A[0] -> $s5

```

Codice MIPS:

```

add $s3, $zero, $zero
Loop: muli $t1, $s3, 4      # $t1 ← 4 * i : offset in byte
      add $t1, $t1, $s5    # $t1 ← address of A[i]
      lw $t0, 0($t1)      # $t0 ← A[i]
      add $s1, $s1, $t0    # g ← g + A[i]
      add $s3, $s3, $s4    # i ← i + j
      bne $s3, $s2, Loop  # go to Loop if i ≠ h

```



Esempio: while

Codice C:

```

while (A[i] == k)
    i = i + j;
Ciclo:  if (A[i] != k) goto Fine;
        i = i + j; goto Ciclo;
        Fine;

```

Si suppone che i, j e k siano associate a $\$s3, \$s4$, e $\$s5$ e che $\$s6$ contenga il *base address* di A

Codice MIPS:

```

Loop: muli $t1, $s3, 4      # $t1 ← 4 * i
      add $t1, $t1, $s6    # $t1 ← address of A[i]
      lw $t0, 0($t1)      # $t0 ← A[i]
      bne $t0, $s5, Exit   # go to Exit if A[i]≠k
      add $s3, $s3, $s4    # i ← i + j
      j Loop              # go to Loop
Exit:

```



Sommario



Istruzioni MIPS di controllo di flusso (condizioni di uguaglianza).

Istruzioni MIPS di controllo di flusso (condizioni di disuguaglianza).

Costrutto switch



Strutture di controllo



- Cosa posso fare se il contenuto di un registro è minore o maggiore del contenuto di un altro?
- MIPS mette a disposizione branch solo nel caso uguale o diverso, non maggiore o minore.
- Spesso è utile condizionare l'esecuzione di una istruzione al fatto che una variabile sia minore di una altra:
 - `slt $s1, $s2, $s3` **# set on less than**
 - Assegna il valore **1** a **\$s1** se **\$s2 < \$s3**; altrimenti assegna il valore **0**
- Con `slt`, `beq` e `bne` si possono implementare tutti i test sui valori di due variabili (`=`, `!=`, `<`, `<=`, `>`, `>=`)



Esempio



```

if (i < j) then
    k = i + j;
else
    k = i - j;

```

##\$s0 ed \$s1 contengono i e j
##\$s2 contiene k

```

if (i < j)
    flag = 1;
If (flag == 0) goto Else;
k = i + j;
goto Exit;
Else: k = i - j;
Exit:

```

→

```

slt $t0, $s0, $s1
beq $t0, $zero, Else
add $s2, $s0, $s1
j Exit
Else: sub $s2, $s0, $s1
Exit:

```



Condizione di disuguaglianza con pseudo-istruzioni (bgt)



```

if (i < j) then
    k = i + j;
else
    k = i - j;

```

##\$s0 ed \$s1 contengono i e j
##\$s2 contiene k

```

if (i < j)
    t = 1;
If (t == 0) goto Else;
k = i + j;
goto Exit;
Else: k = i - j;
Exit:

```

→

```

bgt $s0, $s1, Else
add $s2, $s0, $s1
j Exit
Else: sub $s2, $s0, $s1
Exit:

```



Sommario



Istruzioni MIPS di controllo di flusso (condizioni di uguaglianza).

Istruzioni MIPS di controllo di flusso (condizioni di disuguaglianza).

Costrutto **switch**



Struttura switch/case



- Può essere implementata mediante una serie di *if-then-else*
- Alternativa: uso di una *jump address table* cioè di una tabella che contiene una serie di indirizzi di istruzioni alternative (espressività maggiore che in linguaggio ad alto livello)

```
switch(k)
{
  case 0:  f = i + j; break;
  case 1:  f = g + h; break;
  case 2:  f = g - h; break;
  case 3:  f = i - j; break;
  default: break;
}
```



```
if (k < 0)
    t = 1;
else
    t = 0;
if (t == 1) // k < 0
    goto Exit;
t2 = k;
if (t2 == 0) // k >= 0
    goto L0;
t2--; if (t2 == 0) // k = 1;
    goto L1;
t2--; if (t2 == 0) // k = 2;
    goto L2;
t2--; if (t2 == 0) // k = 3;
    goto L3;
goto Exit; // k > 3;
```

L0: f = i + j; goto Exit;

L1: f = g + h; goto Exit;

L2: f = g - h; goto Exit;

L3: f = i - j; goto Exit;

Exit:

A.A. 2006-2007

17/22

<http://homes.dsi.unimi.it/~borghese>

Struttura switch/case



```
##$s0, ..., $s5 contengono f,...,k, k variabile di test
##$t2 contiene la costante 4
```

```
slt $t3, $s5, $zero
bne $t3, $zero, Exit # if k<0
#case vero e proprio

beq $s5, $zero, L0
addi $s5, $s5, -1
beq $s5, $zero, L1
addi $s5, $s5, -1
beq $s5, $zero, L2
addi $s5, $s5, -1
beq $s5, $zero, L3
j Exit; # if k>3
```

L0: add \$s0, \$s3, \$s4
j Exit

L1: add \$s0, \$s1, \$s2
j Exit

L2: sub \$s0, \$s1, \$s2
j Exit

L3: sub \$s0, \$s3, \$s4

Exit:

A.A. 2006-2007

18/22

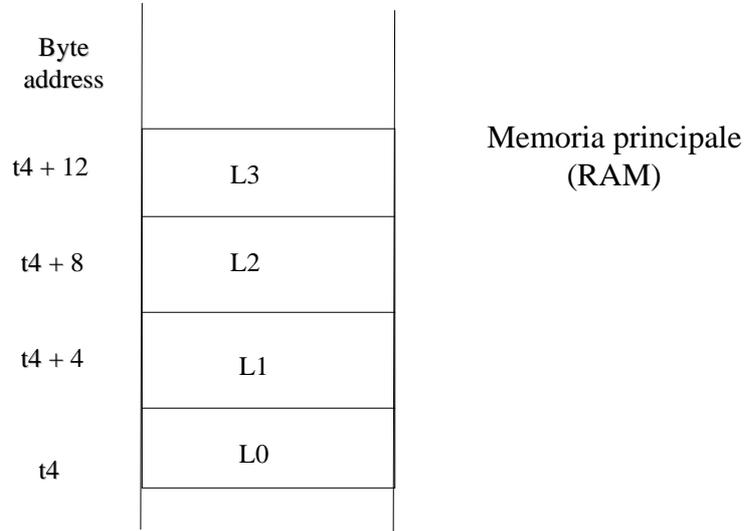
<http://homes.dsi.unimi.it/~borghese>

Struttura switch/case





Jump address table



A.A. 2006-2007

19/22

<http://homes.dsi.unimi.it/~borgnese>



`#$s0, ..., $s5` contengono `f, ..., k`

`#$t4` contiene lo start address della jump address table (che si suppone parta da `k = 0`).

`#verifica prima i limiti (default)`

`slt $t3, $s5, $zero` # if `k < 0` exit

`bne $t3, $zero, Exit`

`slti $t3, $s5, 4`

`beq $t3, $zero, Exit` # if `k >= 4` exit

`#case vero e proprio`

`mulu $t1, $s5, 4` # `t1 = k * 4` offset

`add $t1, $t4, $t1` # `t1 += Table_address`

`lw $t0, 0($t1)`

`jr $t0` # `j A[k]`

`L0: add $s0, $s3, $s4`

`j Exit`

`L1: add $s0, $s1, $s2`

`j Exit`

`L2: sub $s0, $s1, $s2`

`j Exit`

`L3: sub $s0, $s3, $s4`

`Exit:`

**Struttura
switch/case
ottimizzata**

A.A. 2006-2007

20/22

<http://homes.dsi.unimi.it/~borgnese>



Esempio



```

switch(k)
{
  case 0:      f = i + j; break;
  case 1:      f = g + h; break;
  case 2:      f = g - h; break;
  case 3:      f = i - j; break;
  default:     break;
}

.....
muli  $t1, $s5, 4      k      -> $s5
add   $t1, $t4, $t1    550,00016 -> $t1
lw    $t0, 0($t1)
jr    $t0

L0:   add $s0, $s3, $s4      500,01816
      j Exit
L1:   add $s0, $s1, $s2      500,01016
      j Exit
L2:   sub $s0, $s1, $s2      500,00816
      j Exit
L3:   sub $s0, $s3, $s4
Exit:

```

L3 = 500,018 ₁₆
L2 = 500,010 ₁₆
L1 = 500,008 ₁₆
L0 = 500,000 ₁₆
j Exit sub \$s0, \$s1, \$s2
j Exit add \$s0, \$s1, \$s2
j Exit add \$s0, \$s3, \$s4

Jump Address Table (indirizzo iniziale è memorizzato in \$t4)

A.A. 2006-2007
21/22
<http://homes.dsi.unimi.it/~borghese>



Sommarrio



Istruzioni MIPS di controllo di flusso (condizioni di uguaglianza).

Istruzioni MIPS di controllo di flusso (condizioni di disuguaglianza).

Costrutto switch

A.A. 2006-2007
22/22
<http://homes.dsi.unimi.it/~borghese>