



Università degli Studi di Milano
Dipartimento di Informatica "Giovanni Degli Antoni"
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II

Laboratorio

1

Controllo di flusso

2

Controllo di flusso: nei linguaggi ad alto livello

Es: in C / C++ / Java / Go

Costrutti condizionali:

- `if (...) /*then*/ { ... }`
- `if (...) /*then*/ { ... } else { ... }`

Costrutto switch:

- `switch (expr) {
 case 1: ...; case 2: ...; case 3: ... ;
 default: ...;
}`

3

Controllo di flusso: nei linguaggi ad alto livello

Cicli (loops):

- `while (condizione) { fai qualcosa }`
- `do { fai qualcosa } while (condizione)`
- `for (init ; condiz ; passo) {
 fai qualcosa
}`

4

Controllo di flusso: nei linguaggi ad alto livello (nota)

Anche alcuni linguaggi ad “alto” livello prevedono anche un costrutto del tipo:

`goto <locazione>`

Si tratta di un salto.

Evitare! E’ sempre possibile usare al suo posto uno degli altri costrutti disponibili (if then else, loop, switch...)

- Programmazione “ben strutturata”
- Usare `goto` porta invece a programmi “spaghetti”

A basso livello (e.g. MIPS) siamo invece obbligati ad usare salti. Non abbiamo altro, per controllare il flusso.

Vedi anche: <https://xkcd.com/292/>

5

Controllo di flusso: nei linguaggi ad alto livello

Per esempio (in Go)

```
voti := [] int { 28, 21, 30, 18, 18 }  
somma := 0  
for i := 0; i < 5; i++ {  
    somma += voti[ i ]  
}
```

O, ancora più ad alto livello...

```
voti := [] int { 28, 21, 30, 18, 18 }  
somma := 0  
for _ , voto := range voti {  
    somma += voto  
}
```

7

Controllo di flusso: nei linguaggi a **basso** livello

Il controllo di flusso realizzato usando solo due costrutti:

- «Jump» (salto):
modifica l'indirizzo della prossima istruzione.
- «Branch» (bivio, salto condizionato):
come sopra, ma solo se si verifica una data *condizione*

Entrambi hanno l'effetto di modificare il **PC**

- **PC** (program counter) = uno speciale registro che memorizza l'indirizzo della prossima istruzione da eseguire
- Dopo un'istruzione, il PC viene automaticamente incrementato per andare all'istruzione successiva:
 $PC \leftarrow PC+4$ (1 istruzione MIPS = 4 bytes!)
- In caso di salto: il PC viene invece sostituito da un dato *indirizzo target* (specificato nell'istruzione di Jump / Branch)

9

Salti Incondizionati (Jump)

- **Incondizionato** significa che il salto viene **sempre** eseguito
- Istruzioni: **j** (jump), **jr** (jump register)

```
j    INDIRIZZO # salta a un dato indirizzo
```

Esempio:

```
J 0x00400084
```

```
jr $rx      # salta all'indirizzo contenuto in $rx
```

Esempio:

```
la $s1 0x00400084  
jr $s1
```

10

Salti Incondizionati (Jump)

- **Incondizionato** significa che il salto viene **sempre** eseguito
- Istruzioni: `j` (jump), `jr` (jump register)

```
j    INDIRIZZO # salta a un dato indirizzo
```

Esempio:

```
j    0x00400084 ?
```

```
jr   $rx      # salta all'indirizzo contenuto in $rx
```

Esempio:

```
la   $s1 0x00400084 ?  
jr   $s1
```

Ma come facciamo a conoscere l'indirizzo delle istruzioni a cui vogliamo saltare mentre scriviamo il nostro programma? Con le **label**

11

Jump - Salto

- **E' incondizionato** ("unconditional"): il salto viene eseguito in ogni caso
- `j` (jump), `jr` (jump register)

```
j    VALORE    # salta a un dato valore.  
                # PC = Label (invece di PC+4)
```

```
jr   $rx      # salta all'indirizzo contenuto in $rx
```

12

Jump - Salto

| <i>Istruzione</i> | <i>Esempio</i> | <i>Significato</i> |
|----------------------------|----------------------|---|
| <code>jump</code> | <code>j 10000</code> | vai all'istruzione 10000 cioè $PC \leftarrow 10000$ (invece di $PC+4$) |
| <code>jump register</code> | <code>jr \$12</code> | vai all'istruzione di indirizzo "valore di \$12" cioè $PC \leftarrow \$12$ (invece di $PC+4$) |

13


Come trovare l'indirizzo di salto?

- Etichette!
- Nota: le etichette possono essere usate
tanto nel segmento `code`
quanto nel segmento `data`
- In entrambi i casi,
registrano l'indirizzo di memoria
del dato o dell'istruzione
immediatamente seguente

14

Esempio


```
.text:  
...  
... # questa parte viene eseguita  
...  
j qui  
...  
... # codice che NON viene eseguito!  
...  
qui:  
...  
... # questa parte viene eseguita  
...
```



15

Esempio

```
.text:  
li $t0 4  
li $t1 5  
  
j qui  
li $t0 0  
li $t1 0  
  
qui:  
add $t0 $t1 $t0
```

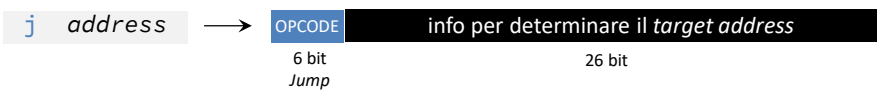


Quanto vale \$t0 alla fine?

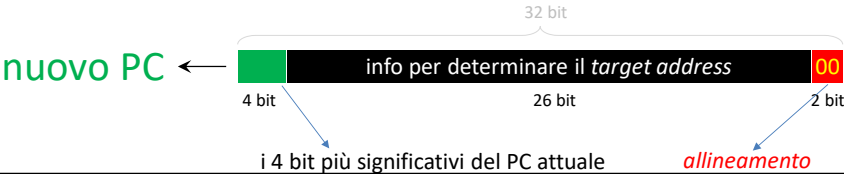
16

Salto in linguaggio macchina MIPS

- Il salto `j` si mappa su un'istruzione di **formato J-type** (J sta per Jump!):



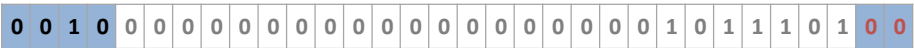
- Il **target address** è di 32 bit (come ogni address MIPS)
- Problema: nell'istruzione ci sono solo 26 bit per specificarli!
- Espediente 1: consideriamo 2 bit sottointesi alla fine: 00 (allineamento)
- Espediente 2: i primi 4 bit dell'address rimangono quelli del PC corrente
- Detta **Modalità di indirizzamento pseudo-diretta**



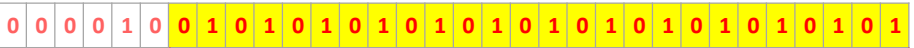
17

Effetti dell'istruzione Jump

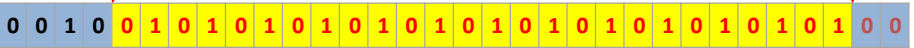
PC corrente:



Istruzione (Jump):



PC dopo il jump:



$$PC = (PC \& 0xf0000000) \mid (target \ll 2);$$

18

Salti in linguaggio macchina MIPS

Conseguenze: il salto **Jump** ...

- non può modificare i primi 4 bits del PC
 - per es, una jump all'indirizzo `0xC-----`
può saltare solo ad un'altra istruzione di indirizzo `0xC-----`
 - non può saltare fuori dal suo «blocco» di istruzioni
- salta sempre ad istruzioni allineate al word
 - ma è ok, tanto sarebbe un errore fare altrimenti.

Per es:
l'insieme delle
word in RAM ad
indirizzi che
iniziano per `0xC`

Nota: **Jump Register** non invece ha alcuna limitazione!

- registro = address = 32 bit

19

Posso saltare “fuori dal blocco”?

Si: con la Jump Register:

```
0xA0000000
0xA0000004 ...
0xA0000008 j foo
0xA000000C ...
0xA0000010
0xA0000014
```

```
...
0xB0000000
0xB0000004 ...
0xB0000008 foo: ...
0xB000000C ...
0xB0000010
```

ERRORE

```
0xA0000000
0xA0000004 ...
0xA0000008 la $at foo
0xA000000C jr $at
0xA0000010
0xA0000014
```

```
...
0xB0000000
0xB0000004 ...
0xB0000008 foo: ...
0xB000000C ...
0xB0000010
```

OK

20

sommario: cosa fa per noi l'assembler nel tradurre un comando jump

1. Traduce da etichetta a *target address* di 32 bit
2. Calcola i **26 bit** dal *target address*
3. Se la jump "salta fuori dal blocco" (raro)...
 - cioè se il suo indirizzo comincia con 4 bit diversi dal target addresslo traduce in sequenza di fino a 3 istruzioni
 - cioè, tratta il comando jump come una pseudoistruzione
 - Usando `jr`, oppure un branch, vedo dopo
 - Tre, perchè la è a sua volta una pseudo istruzione tradotta in due istruzioni (vedi lez precedente)

21

Branch – Bivio , Biforcazione

- Cioè salto **condizionato**:
il salto viene eseguito solo se una certa **condizione** risulta verificata,
altrimenti si passa alla prossima istruzione
(come normale)

- Esempio: **branch on equal**

```
beq $ra $rb destination
```

«Se i registri **\$ra** e **\$rb** hanno lo stesso valore,
allora salta all'etichetta *destination*»

22

Instruzioni di Branch (bivio)

- Con confronto fra due registri

| | | | |
|------------|----------------|----------------------------|-------------|
| beq | \$ra \$rb dest | branch on <i>equal</i> | \$ra = \$rb |
| bne | \$ra \$rb dest | branch on <i>not equal</i> | \$ra ≠ \$rb |
| blt | \$ra \$rb dest | branch on <i>less than</i> | \$ra < \$rb |

- Con confronto fra registro e zero

| | | | |
|-------------|-----------|---|----------|
| bgez | \$ra dest | branch on <i>greater-or-equal zero</i> | \$ra ≥ 0 |
| bgtz | \$ra dest | branch on <i>greater-than zero</i> | \$ra > 0 |
| blez | \$ra dest | branch on <i>less-or-equal to zero</i> | \$ra ≤ 0 |
| bltz | \$ra dest | branch on <i>less-than zero</i> | \$ra < 0 |

23

Instruzioni di Branch (bivio)

- Confronto fra registro e valore

| | | | |
|------------|---------------|------------------------------|------------|
| beq | \$s0 imm addr | branch on <i>equal (imm)</i> | \$ra = imm |
|------------|---------------|------------------------------|------------|

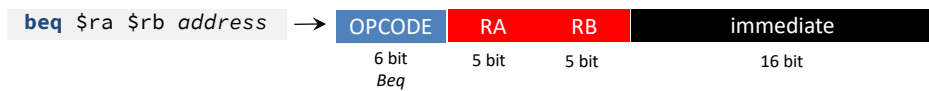
Diventa:

```
addi $at imm  
beq $s0 $at addr
```

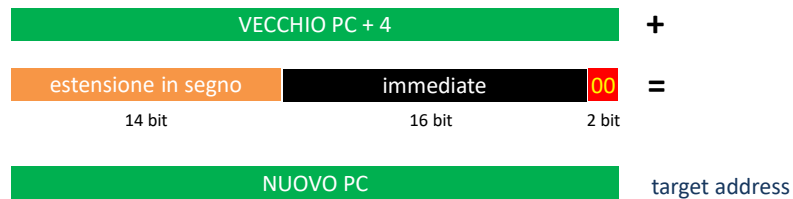
24

Branch in linguaggio macchina MIPS

- I Branch in MIPS hanno **formato** I-type



- Problema: il **target address** (come ogni address in MIPS) è di 32 bit.
- Nell'istruzione I-type abbiamo il campo immediate, di 16 bit, per specificarli
- Soluzione: (detta **modalità di indirizzamento Relativa**):



25

Branch in linguaggio macchina MIPS

Conseguenze: i salti condizionati (**Branch**)

- possono saltare al max a 2^{15} istruzioni sopra o sotto
- possono saltare solo ad indirizzi allineati alla parola (questo è ok)
- (e possono uscire dal «blocco» di istruzioni -
per es: dalla fine di 0xA... posso saltare all'inizio di 0xB...)

Nota:

l'assembler fa per noi il lavoro di ricostruire l'offset di 16 bit

- sottraendo al target address richiesto
l'indirizzo dell'istruzione (successiva al) branch
- se l'indirizzo target è troppo distante ($>2^{15}$),
genera un errore

26

Posso saltare “lontano” condizionalmente?

- Si: cambiando con jump:

```
0xA0000000
0xA0000004 ...
0xA0000008 bgez $t0 far
0xA000000C ...
0xA0000010
0xA0000014
```

...

```
0xA5130000
0xA5130004 ...
0xA5130008 far: ...
0xA513000C ...
0xA5130010
```

ERRORE! too far!

```
0xA0000000
0xA0000004 ...
0xA0000008 bltz $t0 near
0xA000000C j far
0xA0000010 near: ...
0xA0000014
```

...

```
0xA5130000
0xA5130004 ...
0xA5130008 far: ...
0xA513000C ...
0xA5130010
```

OK

27

If – Then: esempio

Codice Go:

```
età := 16
limiteEtà := 18
prezzo := 100
/* sconto per i minorenni */
if età < limiteEtà { prezzo = 80; }
```

```
.data
mieiDati: .word 16 18 100;
.text
la $t0 mieiDati      # ora $t0 punta a «età»
lw $s1 ($t0)         # carico «età» in s1
lw $s2 4($t0)        # carico «limite» in s2
lw $s3 8($t0)        # carico «prezzo» in s3

bge $s1 $s2 end      # se s1 è maggiore o uguale a s2,
                    # vado alla fine e non faccio
li $s3 80             # s3 = 0
end:
```

28

If – Then else: esempio

Codice
C / C++ / Java:

```
int età = 16 ;  
int limiteEtà = 18 ;  
int prezzo;  
  
if (età < limiteEtà) {  
    prezzo = 80 ;  
} else {  
    prezzo = 100 ;  
}
```

oppure, meglio:

```
int età = 16 ;  
int limiteEtà = 18 ;  
  
int prezzo = (età < limiteEtà ) ? 80 : 100 ;
```

29

If – Then else: esempio

Codice Go:

```
età := 16  
limiteEtà := 18  
var prezzo int  
  
if età < limiteEtà { prezzo = 80 }  
else { prezzo = 100 }
```

```
.data  
dati: .word 16 18  
.text  
la $t0 dati    # $t0 punta a «età»  
lw $s1 ($t0)   # carico «età» in s1  
lw $s2 4($t0)  # carico «limite» in s2  
  
ble $s1 $s2 then    # se s1 è < s2 vado a else  
li $s3 80           # ramo else  
j end  
then: li $s3 100  
end:  # resto del programma...
```

30

Do – While: esempio

PseudoCodice di un programma interattivo (legge numeri finchè non ne viene immesso uno neg)

```
do{
    scrivi «inserisci un numero (o negativo per uscire)»;
    leggi numero;
} while (numero >= 0) ;
```

```
.data
msg: .ascii "inserisci un numero (o negativo per uscire)"
.text
loop: li $v0 4
      la $a0 msg
      syscall

      li $v0 5
      syscall

      bgtz $v0 loop

      li $v0 10
      syscall
```

31

While do: esempio

PseudoCodice di un programma interattivo


```
do{
    scrivi «inserisci un numero (o negativo per uscire)»;
    leggi numero;
}
while (numero >= 0) ;
```

```
.data
msg: .ascii "inserisci un numero (o negativo per uscire)"
.text
loop: li $v0 4
      la $a0 msg
      syscall

      li $v0 5
      syscall

      bgtz $v0 loop

      li $v0 10
      syscall
```



32

While do: esempio

data una sequenza di voti in RAM,
trovare la loro somma

Guardia, o terminatore.
Segnala che la lista di
voto è terminata
NB: non è un voto!

```
.data
voti: .word 30 28 27 18 -1
.text
    la $s1, voti
    move $s0, $zero    # s0 = la somma (parziale) dei voti

loop: lw $t0, ($s1)    # t0 = il prossimo voto

    bltz $t0, fine

    addi $s1, $s1, 4    # ora s1 punta al prossimo voto
    addi $s0, $s0, $t0

    j loop
fine: # resto del codice... Ora, $s0 è la somma dei voti
```

33

While do: esempio

- Esercizi:
 - verificare cosa succede se la lista di voti è vuota (cioè se in RAM abbiamo solo la guardia, -1). Funziona?
 - trovare il NUMERO di voti, in \$s2
 - trovare la MEDIA dei voti (arrotondata per difetto), in \$s3, dividendo la somma per il numero di voti
 - evitare la divisione per zero, quando non è presente neanche un voto (in questo caso, la media deve essere 0)
 - stampare la media trovata a schermo

34

Esempi di traduzione in assembly di alcune strutture di controllo di alto livello

35

If - Then

Codice C:

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

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

- Riscriviamo il codice C in una forma equivalente, ma più «vicina» alla sua traduzione Assembly

↓

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

→

Codice Assembly:

```
bne $s3, $s4, L           # if i ≠ j
go to L
add $s0, $s1, $s2
L:
...
```

38

If - Then - Else

Codice C:

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

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

Codice Assembly:

```
bne $s3, $s4, Else
add $s0, $s1, $s2
j End
Else:
sub $s0, $s1, $s2
End:
...
```

39

Do - While

Codice C:

```
i=0;
do{
    g = g + A[i];
    i = i + j;
}while (i!=h);
```

Si supponga che:
g e *h* siano in \$s1, \$s2
i e *j* siano in \$s3, \$s4
A sia in \$s5

- Riscriviamo il codice C:

```
i = 0;
Loop:
g = g + A[i];
i = i + j;
if (i != h)
    goto Loop
```

Codice Assembly:

```
li $s3, 0
Loop:
mul $t1, $s3, 4
add $t1, $t1, $s5
lw $t0, 0($t1)
add $s1, $s1, $t0
add $s3, $s3, $s4
bne $s3, $s2, Loop
```

40

While

Codice C:

```
while (A[i]==k){  
    i=i+j;  
}
```

Si supponga che:
i e j siano in \$s3, \$s4
k sia in \$s5
A sia in \$s6

- Riscriviamo il codice C:

↓

```
Loop:  
If (A[i]!=k)  
    go to End;  
i=i+j;  
go to Loop;
```

→

```
Codice Assembly:  
Loop:  
mul $t1, $s3, 4  
add $t1, $t1, $s6  
lw $t0, 0($t1)  
bne $t0, $s5, End  
add $s3, $s3, $s4  
j Loop  
End:
```

41

Il costrutto switch

- Può essere implementato con una serie di `if-then-else`
- Alternativa: uso di una jump address table*

Codice C:

```
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:
```

42

Marco Tarini

Università degli Studi di Milano

19

Il costrutto switch

- Si supponga che \$s0, ..., \$s5 contengano f,g,h,i,j,k,

Codice Assembly:

| | |
|--|---|
| <pre>slt \$t3, \$s5, \$zero bne \$t3, \$zero, Exit 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</pre> | <pre>j Exit; 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:</pre> |
|--|---|

43

Assegnamenti condizionali

44

Assegnamenti condizionali (linguaggi ad alto livello)

- Due programmi equivalenti (C, Java...)

```
if (x<y) then z = 1; else z = 0;
```

```
z = (x<y) ? 1 : 0 ;
```

In pratica, z è il valore *vero* o *falso* (1 o 0) dell'espressione booleana $(x < y)$

45

Assegnamenti condizionali (linguaggi ad alto livello)

- Due programmi equivalenti (C, Java...)

```
if (x<y) then min = x; else min = y;
```

```
min = (x<y) ? x : y ;
```

46

Assegnamenti condizionali nei linguaggi ad alto livello (note)

- Gli assegnamenti condizionali possono essere un modo per ottenere lo stesso risultato di un if-then-else ma senza controllo di flusso
- Sono disponibili in molti linguaggi ad alto livello (C, Java, C++... ma non ad es Go)
- Possono aumentare la leggibilità / eleganza del codice
- Possono aumentare l'efficienza. Il controllo di flusso infatti è "caro".
le CPU pipelined, richiedono un flushing ad ogni salto
- Questo vale in particolar modo nelle GPU, che hanno più ALU... ma sono meno predisposte al controllo di flusso
(per es, hanno meno circuiteria dedicata ad ottimizzarlo, come il *branch prediction*)

47

In assembly: valutare le condizioni senza controllo di flusso

- Anche molti linguaggi assembly hanno dei costrutti per assegnare dei valori a dei registri in modo *condizionale*, cioè a seconda della valutazione di una condizione (di uguaglianza, disuguaglianza, etc)
- In termini di efficienza, di nuovo, il vantaggio sta nel evitare salti (cambiamenti del PC)
- Il MIPS mette a disposizione **istruzioni** e **pseudoistruzioni** del tipo «Set On (condizione)»
(Set significa «assegnare (un bit) al valore uno 1»)
- Effetto: un dato registro assume il valore intero 1 (0x00000001) se una data condizione è verificata, o il valore 0 (0x00000000) , se la condizione non è verificata
- La condizione è definita su due altri registri
- Questi comandi possono anche essere comodi per valutare condizioni composte (da diverse clausole and, or, not), senza intricate sequenze di salti (vedi esercizi)

48

Assegnamenti condizionati in MIPS

| | | |
|---------------------------------|------------------------|--------------------|
| set on equal | seq \$a \$b \$c | \$a = (\$b = \$c) |
| set on not equal | sne \$a \$b \$c | \$a = (\$b ≠ \$c) |
| set on less then | slt \$a \$b \$c | \$a = (\$b < \$c) |
| set on less then or equal | sle \$a \$b \$c | \$a = (\$b ≤ \$c) |
| set on greater then | sgt \$a \$b \$c | \$a = (\$b > \$c) |
| set on greater then or equal | sge \$a \$b \$c | \$a = (\$b ≥ \$c) |

49

Assegnamenti condizionati in MIPS Esempio di varianti

| | | |
|---|---------------------------------|--------------------|
| set on less-then | slt \$a \$b \$c | \$a = (\$b < \$c) |
| set on less-then <i>unsigned</i> | sltu \$a \$b \$c | \$a = (\$b < \$c) |
| set on less-then <i>immediate</i> | slti \$a \$b <i>imm</i> | \$a = (\$b < imm) |
| set on less-then <i>immediate unsigned</i> | sltiu \$a \$b <i>imm</i> | \$a = (\$b < imm) |

E necessario sapere se i parametri da paragonare siano espressi in *COMPLEMENTO A DUE* o *SENZA SEGNO* (cambia il risultato!)

50