



Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2017-2018

Controllo di flusso



[Homepage del corso](#)

Turno A

Nicola Basilico

Dipartimento di Informatica

Via Comelico 39/41 - 20135 Milano (MI)

Ufficio S242

nicola.basilico@unimi.it

+39 02.503.16294

Turno B

Jacopo Essenziale

Dipartimento di Informatica

Via Celoria 20 - 20133 Milano (MI)

AILab

jacopo.essenziale@unimi.it

+39 02.503.14010

Controllo di flusso

- Nel codice di alto livello il controllo di flusso si attua mediante verifica di condizioni, salti non condizionati o cicli
 - In assembly abbiamo a disposizione le istruzioni di **branch** e **jump** che ci consentono di alterare l'ordine sequenziale di esecuzione delle istruzioni di un programma
 - Operano aggiornando il **program counter** a seconda di una certa logica (ad esempio verifica di un'uguaglianza tra registri)
 - La prossima istruzione da eseguire potrà quindi non essere quella all'indirizzo successivo (PC+4), ma quella ad un indirizzo detto **target address**
 - Il target address è determinato a partire da un operando dell'istruzione seguendo le regole associate prescritte dalla **modalità di indirizzamento**
- Convenzioni di notazione:
 - Identificativo con iniziale maiuscola: deve essere una label (e.g., «Label», «Exit», «Loop»);
 - Identificativo con iniziale minuscola: deve essere un registro o un valore immediato (intero con segno su 16 bit);
 - Identificativo con iniziale «\$»: deve essere un registro.

Salti non condizionati

- **Non condizionato** (unconditional): Il salto viene sempre eseguito
- **j** (jump), **jal** (jump and link), **jr** (jump register)

j Label	# salta a Label
jal Label PC in \$ra	# salta a Label, salva
jr \$rx contenuto in \$rx	# salta all'indirizzo

Salti condizionati

- **Condizionato** (conditional): il salto viene eseguito solo se una certa condizione risulta verificata
- Esempi: **beq** (*branch on equal*) e **bne** (*branch on not equal*)

```
beq $rs, rt, Label          # if (rs ==  
rt) salta a Label
```

```
bne $rs, rt, Label          # if (rs !=  
rt) salta a Label
```

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`

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:
...
```

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`

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:
...
```



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

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
```

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

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:
```

Condizioni di disuguaglianza

- Spesso è utile condizionare l'esecuzione di un'istruzione al fatto che una variabile sia minore di un'altra, istruzione **Set Less Than**.

```
slt $s1, $s2, s3
```

- Assegna il valore 1 (set) 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 (`==`, `!=`, `<`, `<=`, `>`, `>=`).

Condizioni di disuguaglianza

- Si completi la seguente tabella con il corrispettivo codice assembly

Pseudo codice	Assembly
if(\$s1==\$s2) addi \$s3, \$s3, 1	
if(\$s1!=\$s2) addi \$s3, \$s3, 1	
if(\$s1>\$s2) addi \$s3, \$s3, 1	
if(\$s1>=\$s2) addi \$s3, \$s3, 1	
if(\$s1<\$s2) addi \$s3, \$s3, 1	
if(\$s1<=\$s2) addi \$s3, \$s3, 1	

Condizioni di disuguaglianza

- `slt` e `beq` come nell'esempio precedente possono essere usate tramite pseudo-istruzioni, ad esempio **Branch on Greater Than**.

```
bgt $s1, s2, Label
```

- Salta a `Label` se `$s1 > s2`
- Altre pseudo-istruzioni simili:

```
bge, blt, ble
```

Il costrutto switch

- Può essere implementato con una serie di **if-then-else**
- *Alternativa: uso di una jump address table (prossime lezioni)*

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;
```


Il costrutto switch

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

Codice Assembly:

```
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
```

```
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:
```

Ampiezza dei salti non condizionati

- I salti non condizionati con `j` e `jal` si mappano su istruzioni che hanno **formato J-type**



- I 26 bit contengono un valore immediato con cui ottenere il *target address*. Il target address deve avere 32 bit, come ottenerli?
- Modalità di indirizzamento pseudo-diretta:**

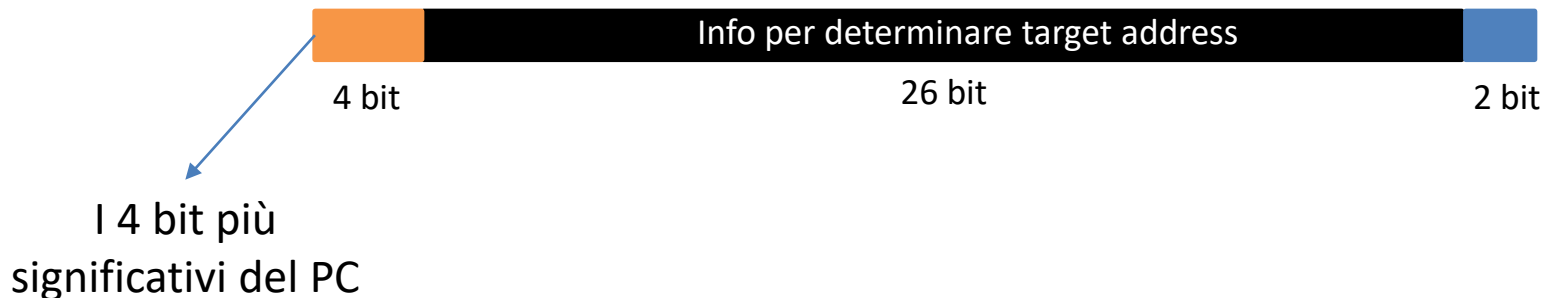


Ampiezza dei salti non condizionati

- I salti non condizionati con `j` e `jal` si mappano su istruzioni che hanno **formato J-type**



- I 26 bit contengono un valore immediato con cui ottenere il *target address*. Come?
- Modalità di indirizzamento pseudo-diretta:**

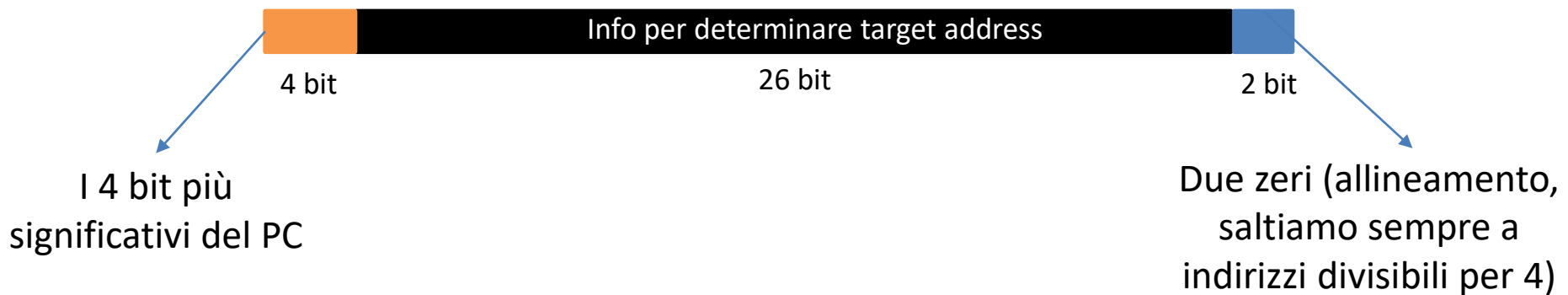


Ampiezza dei salti non condizionati

- I salti non condizionati con `j` e `jal` si mappano su istruzioni che hanno **formato J-type**



- I 26 bit contengono un valore immediato con cui ottenere il *target address*. Come?
- Modalità di indirizzamento pseudo-diretta:**



Ampiezza dei salti condizionati

- Queste istruzioni, oltre al target address, devono specificare altri due operandi per il test di uguaglianza: hanno un altro formato! **I-type**.

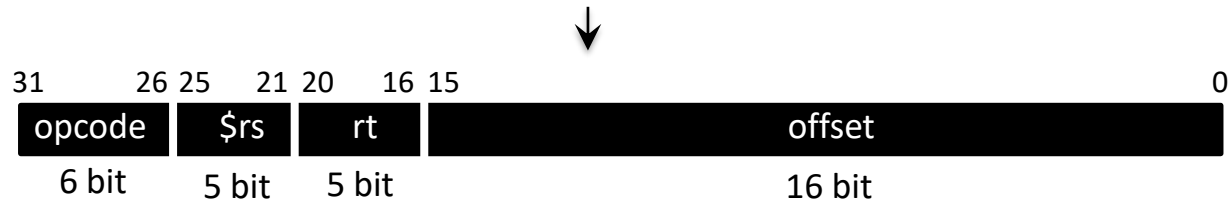
`beq $rs, rt, Label # if (rs == rt) salta a Label`



- Usare i 16 bit come target address (o pseudo direct) sarebbe molto limitante.
- Tuttavia, i salti condizionati (`beq`, `bne`) tipicamente non hanno grandi ampiezze (saltano a istruzioni vicine) perché il loro uso più frequente è nei cicli o nei blocchi con *if*.
- Il concetto di «vicino» è, ovviamente, relativo al valore corrente del PC.
- Modalità di indirizzamento relativa al PC:** i 16 bit rappresentano un offset.

Ampiezza dei salti condizionati

`beq $rs, rt, Label # if (rs == rt) salta a Label`



- **Modalità di indirizzamento relativa al PC** : come si determina il target address?
- Convenzione: l'offset è rispetto al PC che tipicamente punta già all'istruzione **successiva** alla branch.
- Procedimento:
 1. Si aggiungono due zeri in coda all'offset (allineamento, vogliamo sempre un indirizzo multiplo di 4 e rendiamo quindi i due zeri in coda impliciti; dopo questa operazione l'offset si interpreta come numero di istruzioni e non numero di byte);
 2. Si somma l'offset al PC, il risultato è il target address.
- Indirizzamento: da $PC - 2^{17}$ a $PC + 2^{17} - 4$ (senza step 1 sarebbe stato da $PC - 2^{15}$ a $PC + 2^{15} - 1$, si guadagna un fattore 4 nell'ampiezza dei salti)

Ampiezza dei salti condizionati

- Come fare se si necessita di salti **condizionati** molto **ampi**?
- Combiniamo branch e jump. Ad esempio:

```
beq $rs, rt, FarAway
```

Non riusciamo a ottenere questo target address con un offset da 16 bit e l'indirizzamento PC-relative.

- Soluzione:

```
bneq $rs, rt, VeryClose  
J FarAway
```

VeryClose:

...

Ora abbiamo a disposizione 26 bit e indirizzamento pseudo-direct.

Esercizio 4.1

- Nome del file sorgente: *parisuccessivo.asm*
- Si scriva il codice che dato un intero inserito dall'utente restituisca il numero pari successivo.

Esercizio 4.2

- Nome del file sorgente: *checkCondizioni.asm*
- Si scriva il codice assembly che esegua le seguenti istruzioni:

```
a = <intero inserito dall'utente>  
b = <intero inserito dall'utente>  
c = <intero inserito dall'utente>
```

```
If ( (a>=b) && (c!=0) ){  
    z=c(a+b);  
    print z  
}  
else{  
    print «errore»  
}
```

Esercizio 4.3

- Nome del file sorgente: *sommaQuadrati.asm*
- Si scriva il codice che calcola la somma dei primi N-1 numeri elevati al quadrato.
- Nel caso in cui l'i-esimo numero da aggiungere sia multiplo del valore iniziale della somma, si termini il ciclo for.

```
V=<intero inserito dall'utente>;  
N=<intero inserito dall'utente>;
```

```
Sum = V;  
for (i=1; i<N; i++)  
{  
    If ((i*i)%V==0){  
        print «break»;  
        break;  
    }  
  
    Sum+=i*i;  
}  
print Sum
```



Università degli Studi di Milano
Laboratorio di Architettura degli Elaboratori II
Corso di Laurea in Informatica, A.A. 2017-2018

Nicola Basilico

Dipartimento di Informatica

Via Comelico 39/41 - 20135 Milano (MI)

Ufficio S242

nicola.basilico@unimi.it

+39 02.503.16294

Hanno contribuito alla realizzazione di queste slides:

- Iuri Frosio
- Jacopo Essenziale