ARCHITETTURA DEL SET DI ISTRUZIONI

Distanza dei salti (jump e branch)

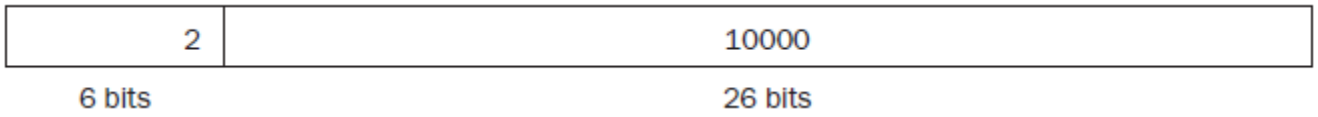
Michele Favalli

Problema dell'indirizzamento

- Le istruzioni MIPS per il «salto incondizionato (jump)» utilizzano il meccanismo di indirizzamento offerto da un nuovo tipo di istruzione:
  - **Istruzioni «J-Type»**.

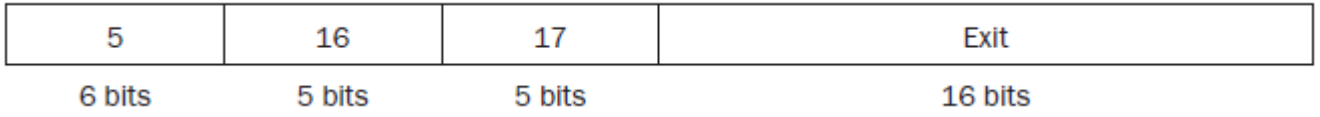
```
j   10000   # go to location 10000
```

E' un indirizzo «assoluto», per esprimere il quale occorre il maggior numero di bit possibile.



- Un salto «condizionato» invece deve specificare, con lo stesso «budget» di bits, due registri ed un indirizzo (branch address):
  - **Istruzioni «I-Type»**.

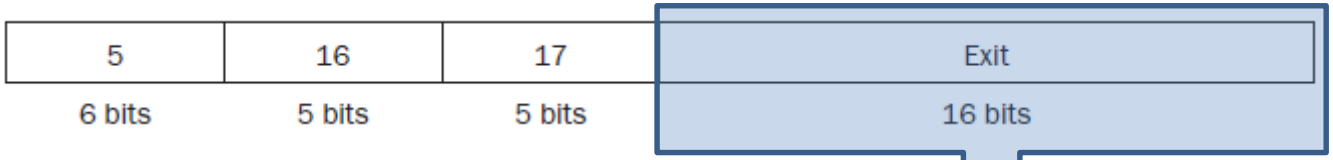
```
bne  $s0,$s1,Exit   # go to Exit if $s0 ≠ $s1
```



Problema:
Un programma non potrebbe essere più grande di 2^16 bytes!

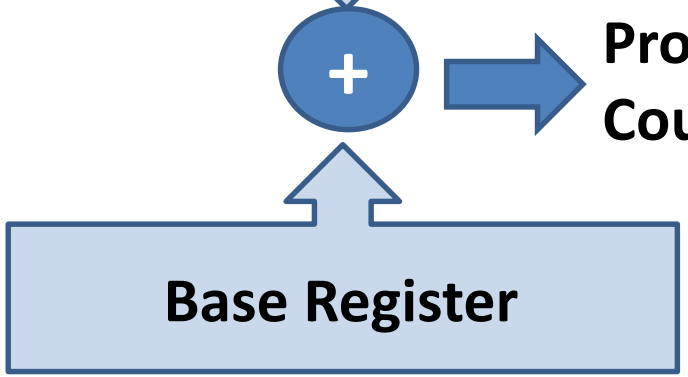
Alternativa

```
bne  $s0,$s1,Exit   # go to Exit if $s0 ≠ $s1
```



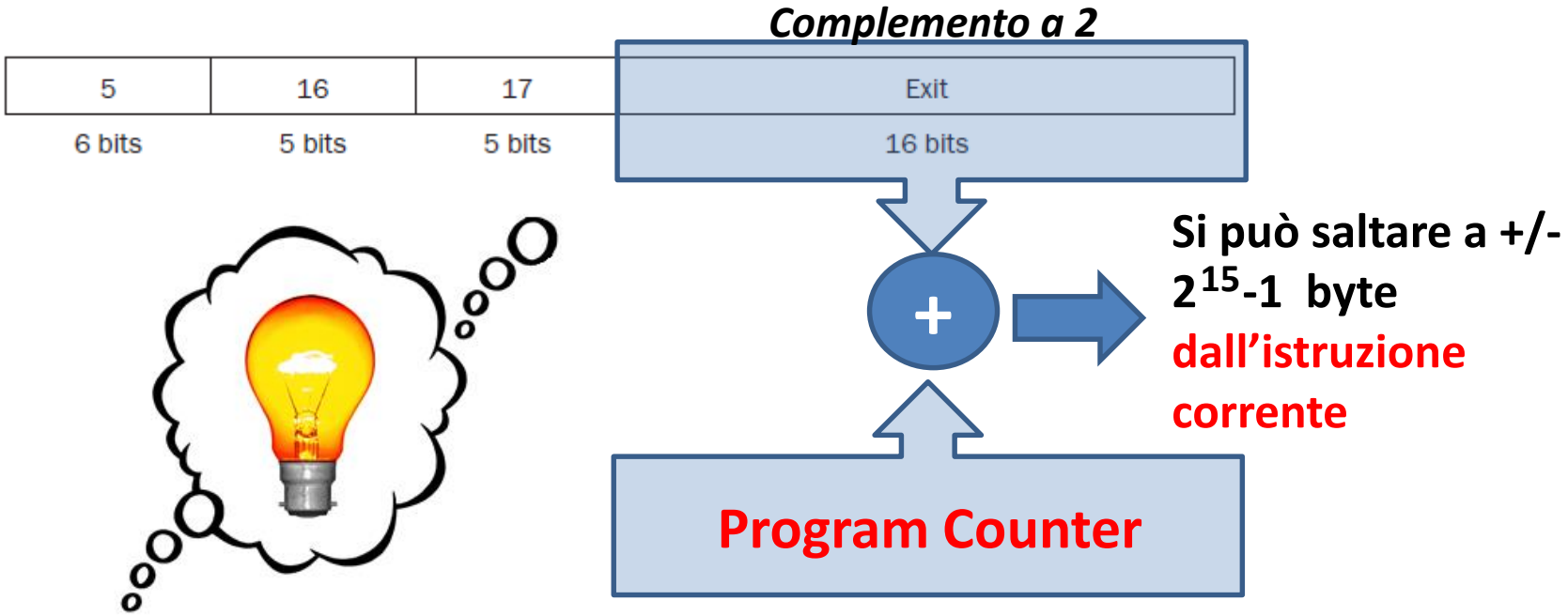
Non è un indirizzo assoluto, ma un offset (con segno)!

- **La macchina sottostante calcolerà l'indirizzo finale come somma di un offset e di un registro base.**



QUALE REGISTRO UTILIZZARE COME «BASE REGISTER»?

PC-Relative Addressing

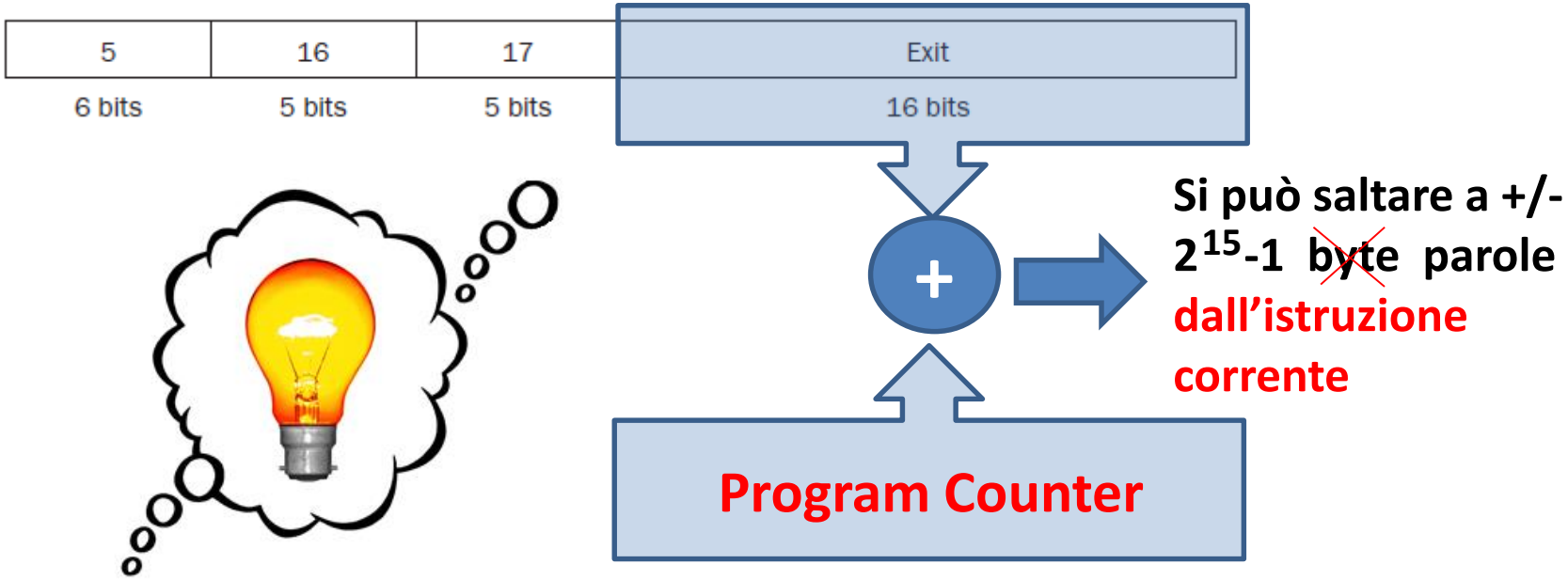


- I Branch condizionali sono utilizzati in *loop* e *if* statements.

  ⇒In generale, essi saltano ad istruzioni vicine!

  ⇒Dunque, usiamo PC come riferimento (in realtà, PC+4)

PC-Relative Addressing



- **Posso migliorare la «branching distance»?**
- **Si, assumendo che l'offset sia riferito a parole (da 4 byte), non ai byte**
- **Posso saltare in un range di istruzioni dal PC pari a:**
  [PC- 2^15 -- PC+2^15-1] (64k istruzioni)
- **Tipicamente, nessun FOR/WHILE loop o IF statement è maggiore di 64k istruzioni. Il PC è una ottima scelta come registro base.**

I Salti Incondizionati

j Etichetta

jal ProcedureAddress

Non c'è motivo di pensare che il salto incondizionato (istruzione «jump») sia in un intorno del PC

Analogamente, quando si *invoca una procedura* attraverso l'istruzione «jump-and-link» (jal), non c'è motivo di pensare che la procedura chiamata sia allocata ad un indirizzo «vicino» a quello del chiamante

- Queste istruzioni codificano un indirizzo assoluto, non un offset!
- Hanno dunque bisogno del maggior numero possibile di bit per codificarlo: **formato J!**

Name	Format	Opcode	Example of Address (26 bits)
j	J	2	2500
jal	J	3	2500

Le istruzioni «jump» e «jump-and-link» sono codificate nel formato J per permettere sezioni di codice da **2^{26} byte**

Estensione della Distanza di Salto

OTTIMIZZAZIONE
Offset ed indirizzi sono da riferirsi alle parole, non ai byte, dal momento che ogni istruzione consiste di 4 bytes. Questo allunga ulteriormente la massima dimensione del codice che può racchiudere una istruzione di «jump».

Name	Format	Example	
j	J	2	2500
jal	J	3	2500

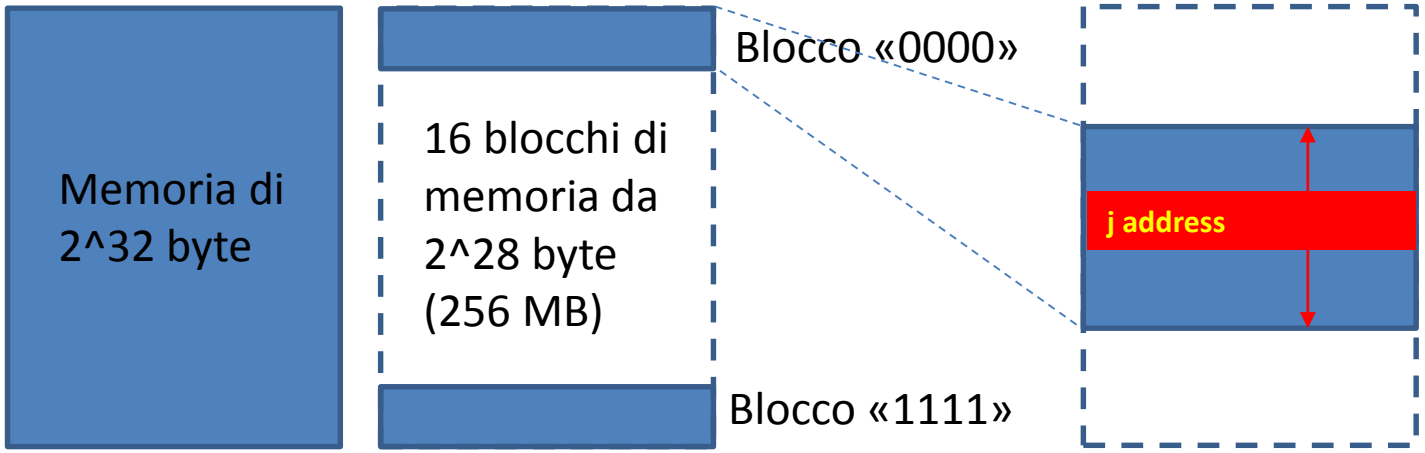
Le istruzioni «jump» e «jump-and-link» sono codificate nel formato J per permettere sezioni di codice da **2^{26} ~~byte~~ parole (64M istruzioni)**

Le istruzioni di salto incondizionato codificano un riferimento assoluto di memoria, non un offset rispetto al Program Counter

Istruzione jump **j address**

Field	000010	address
Bit positions	31:26	25:0

- Occorre comprendere bene l'indirizzamento di questa istruzione:
  - «address» è un indirizzo assoluto con la granularità della parola, non del byte
  - Occorre moltiplicarlo per 4 per ottenere l'indirizzamento a byte, cioè occorre fare lo «shift left» di 2 posizioni, inserendo «00» in posizione meno significativa.
  - Da 26 bit si passa così a 28 bit. Da dove vengono i rimanenti 4 bit?
  - I 4 bit rimanenti provengono dai **bit più significativi di PC+4**



- *Se i primi 4 bit più significativi sono quelli del PC, significa che il salto mi fa rimanere dentro al blocco corrente*
- *«address»*4 consiste di 28 bit, e identifica un byte qualsiasi all'interno del blocco.*

Per jump all'esterno del blocco di memoria, il compilatore deve cambiare tecnica di indirizzamento (cioè usare «jr», vedi prossima slide)

La Soluzione più Estrema

- E se serve andare oltre all'indirizzamento di 64M istruzioni?
- Non resta che ricorrere a:
  - Formazione dell'indirizzo completo a 32 bit a cui saltare (tramite istruzioni assembler)
  - Memorizzazione dell'indirizzo in un registro (es., $s0)
  - **jr $s0 (jump register)**
  - In questo modo, l'intero spazio di indirizzamento di 2^32 byte è disponibile

Ottimizzazioni del compilatore

- Il compilatore viene in soccorso quando il «branch address» non può essere espresso con il campo a 16 bit del formato I:

```
beq    $s0,$s1, L1
```

Posso aumentare la «branching distance» oltre i 16 bit previsti dal formato dell'istruzione ***beq****?*

```
      bne    $s0,$s1, L2
      j      L1
L2:
```

Il compilatore trasforma il salto condizionato in un salto incondizionato, beneficiando così di un «branch address» a 26 bit.

Salti Condizionati e Incondizionati

- Le istruzioni di salto consentono di modificare il flusso di controllo di un programma evitando di eseguire l'istruzione successiva
- Nel caso delle istruzioni di branch si sfrutta il fatto che di solito il salto condizionato avviene verso un istruzione vicina per utilizzare un offset rispetto al PC
- Nel caso delle istruzioni di salto incondizionato, si sfrutta il maggior numero di bit a disposizione per avere un salto verso un istruzione specificata con un indirizzo assoluto
- In casi non risolvibili con i 16 bit a disposizione di