Operandi immediati ampi

- Problema: è possibile caricare in un registro una costante a 32 bit? Supponiamo di voler caricare nel registro x5 il valore 0x12345678
- Soluzione
  - si introduce una nuova istruzione `lui` (load upper immediate, tipo U) che carica i 20 bit più significativi della costante nei bit da 12 a 31 di un registro e pone quelli a sinistra a zero (i 32 bit più significativi hanno lo stesso valore del bit 31)

```
lui x5,0x12345
```

x5 ..... 00010010 00110100 01010000 00000000

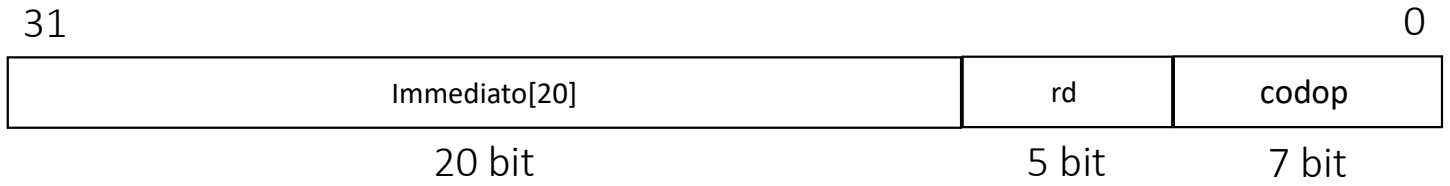
  - Con una operazione di or immediato si impostano i 12 bit meno significativi rimasti

```
ori x5,x5,0x678
```

x5 ..... 00010010 00110100 01010110 01111000

LUI e linguaggio macchina

- Viene introdotto un nuovo tipo: U



es. mio

Operandi immediati ampi

- Potreste realizzare il caricamento con le istruzioni `lui` e `addi` (al posto di `ori`)?
  - Supponete di voler caricare nel registro x5 il valore 0x82345678: succede qualcosa?
  - Supponete di voler caricare nel registro x5 il valore 0x12345878: succede qualcosa?
  - Supponete di voler caricare nel registro x5 il valore 0x82345878: succede qualcosa?

Salti condizionati

- Permettono di variare il flusso del programma (variando il valore del PC) a verificarsi di una condizione

```
beq rs1,rs2,L1
```

Branch if EQual

- Il flusso di programma continua all'istruzione con etichetta `L1` se il valore del registro `rs1` è uguale a quello di `rs2`

```
bne rs1,rs2,L1
```

Branch if Not Equal

- Il flusso di programma continua all'istruzione con etichetta `L1` se il valore del registro `rs1` è diverso a quello di `rs2`

Salti condizionati

- Permettono di variare il flusso del programma (variando il valore del PC) al verificarsi di una condizione

```
blt rs1,rs2,L1
```

Branch if Less Than

- Il flusso di programma continua all'istruzione con etichetta `L1` se il valore del registro `rs1` è minore a quello di `rs2`

```
bge rs1,rs2,L1
```

Branch if Greater than or Equal

- Il flusso di programma continua all'istruzione con etichetta `L1` se il valore del registro `rs1` è maggiore o uguale a quello di `rs2`

Salti condizionati

- Esistono anche le operazioni di salto condizionato che confrontano i due registri `rs1` e `rs2` trattandoli come numeri senza segno

`bltu rs1,rs2,L1`	Branch if Less Than Unsigned
`bgeu rs1,rs2,L1`	Branch if Greater than or Equal Unsigned

Salti condizionati: costrutto if-then-else

- Attraverso le istruzioni `beq` e `bne` è possibile tradurre in assembler il costrutto `if` dei linguaggi di programmazione ad alto livello

La scelta di test per not equal è più conveniente in questo caso

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

Linguaggio C

```
f → x19
g → x20
h → x21
i → x22
j → x23
```

```
        bne x22,x23,ELSE
        add x19,x20,x21
        beq x0,x0,ENDIF
 ELSE:  sub x19,x20,x21
ENDIF:
```

Salto incondizionato

RISC-V assembler

Salti condizionati: ciclo for

- Una possibile implementazione

```
for (i=0;i<100;i++)
{
…
}
```

i → x19

```
        add x19,x0,x0
        addi x20,x0,100
FOR:    bge x19,x20,ENDFOR
        …
        addi x19,x19,1
        beq x0,x0,FOR
ENDFOR:
```

Salti condizionati: ciclo while

```
while (v[i]==k)
{
…
i=i+1
}
```

```
i → x22
k → x24
v → x25
```

```
LOOP:    slli x10,x22,3
         add x10,x10,x25

         ld x9,0(x10)

         bne x9,x24,ENDLOOP
         …
         addi x22,x22,1
         beq x0,x0,LOOP
ENDLOOP:
```

Salva in x10 l'indirizzo della doubleword v[i]

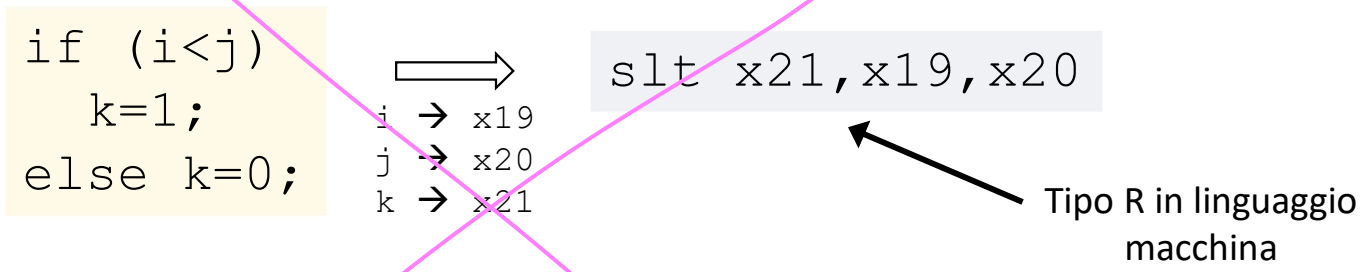
Carica dalla memoria il valore contenuto nella doubleword v[i]

Se v[i] != k, allora il ciclo termina

Incrementa i e torna alla valutazione della condizione del ciclo while

Salti condizionati

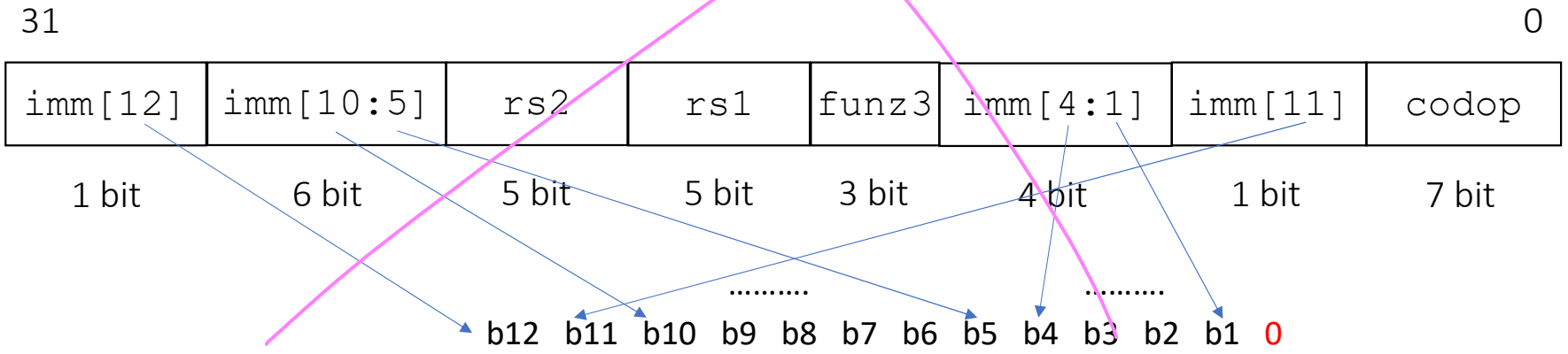
- L'istruzione `slt` permette di costruire strutture di controllo con istruzioni di salto generiche



- Possiamo ad es, inserire dopo `slt` l'istruzione `beq rs1,x0,L1`
  - per il confronto su "'>='' basta invertire la condizione (`bne`)
  - per il confronto su "'>'' basta scambiare gli operandi della `slt`
  - per il confronto su "'<='', inverto condizione e scambio operandi

Salti condizionati e linguaggio macchina

- Le istruzioni di salto condizionato utilizzano il **formato di tipo SB**
- Il formato può rappresentare indirizzi di salto da -4096 a 4094, in multipli di due
- Infatti, abbiamo 13 bit in complemento a 2 con solo i numeri pari



Salti condizionati e linguaggio macchina

- Si utilizza l'indirizzamento relativo al program counter (PC-relative addressing)
- Il campo immediato di 12+1 bit contiene l'offset rispetto al valore di PC (espresso in complemento a due)
- Per ottenere il prossimo valore di PC in caso di salto, viene eseguito il calcolo

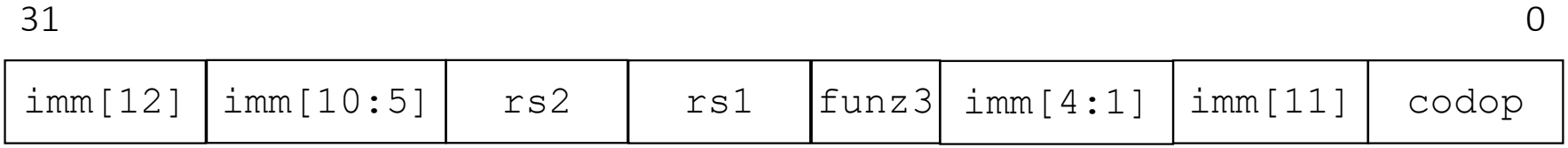
```
PC = PC + immediato
```

Salti condizionati e linguaggio macchina

- Le istruzioni macchina RISC-V hanno una dimensione di 32 bit
- I possibili valori di scostamento sono compresi tra -4096 e + 4094 ([-2048*2,+2047*2])
- Non c'è un vincolo di allineamento a 32 bit, ma questo sistema consente solo di saltare ad indirizzi di memoria pari (perché b0 è implicitamente uguale a 0)
- Perché il moltiplicatore non è 4? I progettisti hanno voluto prevedere la possibilità di rappresentare le istruzioni macchina anche su 16 bit
- Noi, però, continueremo sempre a considerare istruzioni su 32 bit. Il compilatore genererà campi immediati di istruzioni di salto sempre multipli di 4, ovvero con b1 = 0 e con b0 implicitamente a 0.

nulla utile solo binario forma

Salti condizionati e linguaggio macchina

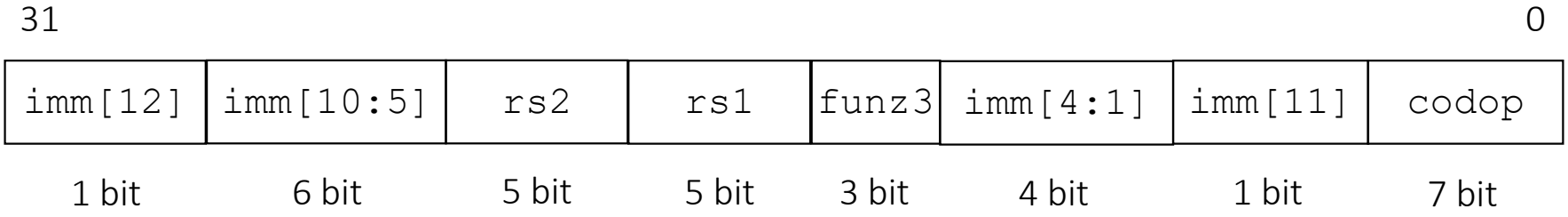



```
LOOP:     slli x10,x22,3
          add x10,x10,x25
          ld x9,0(x10)
          bne x9,x24,ENDLOOP
          addi x22,x22,1
          beq x0,x0,LOOP
ENDLOOP:
```

PC = 80000

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

Salti condizionati e linguaggio macchina

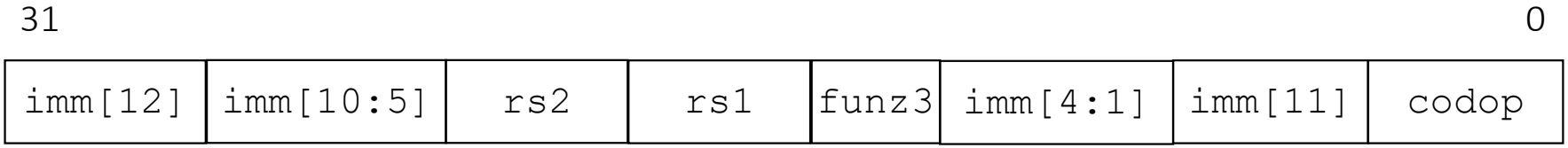



```
LOOP:     slli x10,x22,3
          add x10,x10,x25
          ld x9,0(x10)
          bne x9,x24,ENDLOOP
          addi x22,x22,1
          beq x0,x0,LOOP
ENDLOOP:
```

PC = 80004

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

Salti condizionati e linguaggio macchina

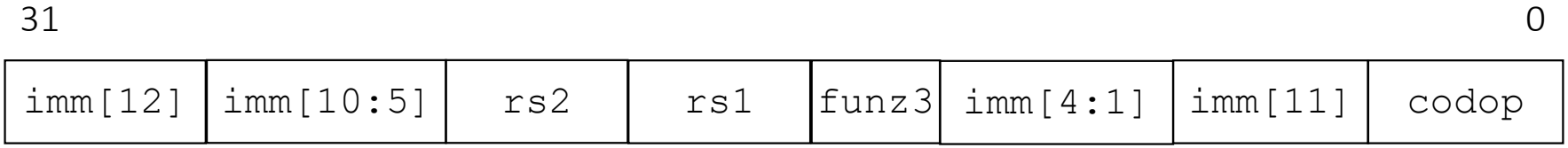



```
LOOP:     slli x10,x22,3
          add x10,x10,x25
          ld x9,0(x10)
          bne x9,x24,ENDLOOP
          addi x22,x22,1
          beq x0,x0,LOOP
ENDLOOP:
```

PC = 80008

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

Salti condizionati e linguaggio macchina

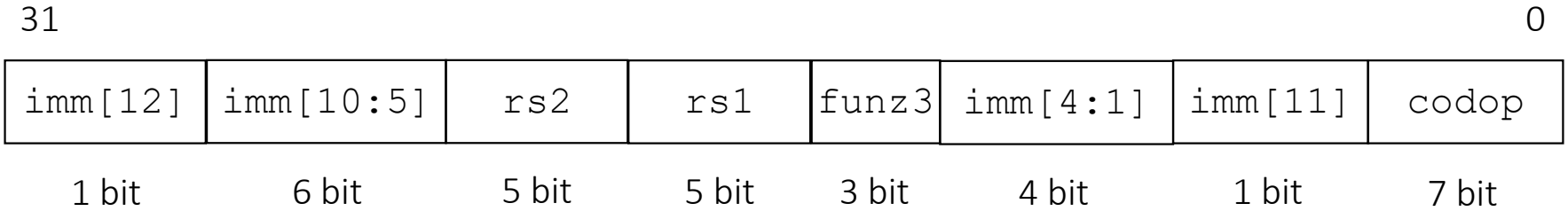



```
LOOP:     slli x10,x22,3
          add x10,x10,x25
          ld x9,0(x10)
          bne x9,x24,ENDLOOP
          addi x22,x22,1
          beq x0,x0,LOOP
ENDLOOP:
```

PC = 80012

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

Salti condizionati e linguaggio macchina




```
LOOP:     slli x10,x22,3
          add x10,x10,x25
          ld x9,0(x10)
          bne x9,x24,ENDLOOP
          addi x22,x22,1
          beq x0,x0,LOOP
ENDLOOP:
```

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

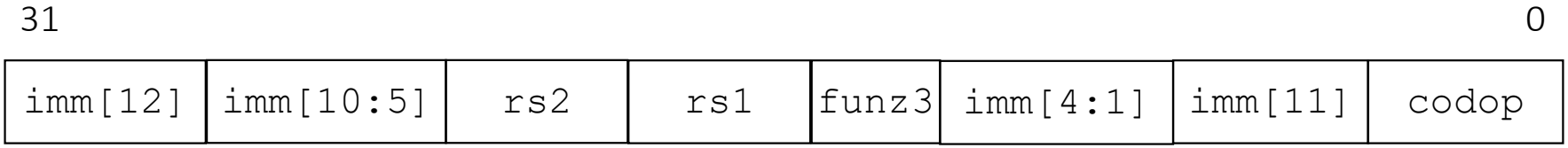
PC = 80012 + 12 = 80024

0 0 0 0 0 0 0 0 0 1 1 0 0 = 12

b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0

Se la condizione di `bne` è vera

Salti condizionati e linguaggio macchina



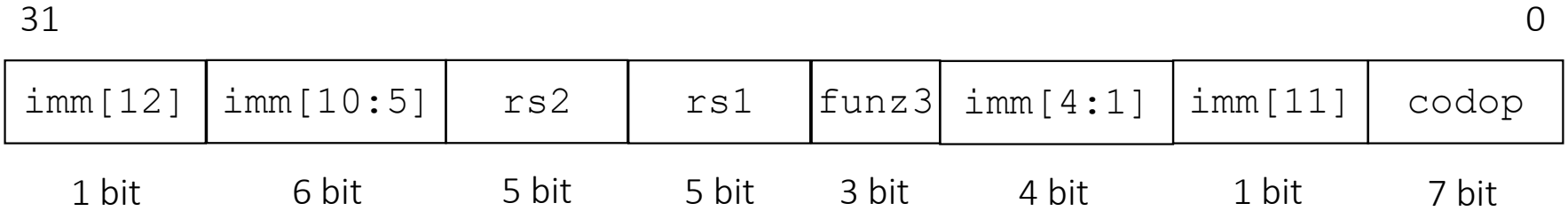

```
LOOP:     slli x10,x22,3
          add x10,x10,x25
          ld x9,0(x10)
          bne x9,x24,ENDLOOP
          addi x22,x22,1
          beq x0,x0,LOOP
ENDLOOP:
```

PC = 80016

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

Se la condizione di `bne` è falsa

Salti condizionati e linguaggio macchina



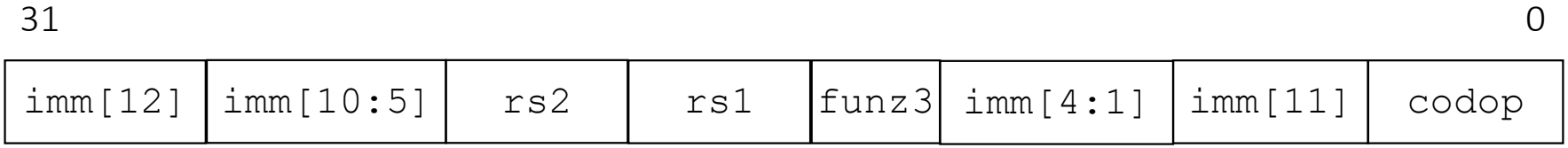

```
LOOP:    slli x10,x22,3
         add x10,x10,x25
         ld x9,0(x10)
         bne x9,x24,ENDLOOP
         addi x22,x22,1
         beq x0,x0,LOOP
ENDLOOP:
```

PC = 80020

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

Se la condizione di `bne` è falsa

Salti condizionati e linguaggio macchina




```
LOOP:     slli x10,x22,3
          add x10,x10,x25
          ld x9,0(x10)
          bne x9,x24,ENDLOOP
          addi x22,x22,1
          beq x0,x0,LOOP
ENDLOOP:
```

Indirizzo	Istruzione					
80000	0000000	00011	10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	0000000	00000	01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	0000000	00001	10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

PC = 80020 - 20 = 80000

1 1 1 1 1 1 1 1 0 1 1 0 0 = -20

b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0

Se la condizione di `beq` è vera (in questo caso lo è sempre)