

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 00110110 01010000 00000000

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

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

x5

.....

00010010 00110110 01010110 01111000

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

Se $rs1 == rs2 \rightarrow L1$

- 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

Se $rs1 \neq rs2 \rightarrow L1$

- 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) a 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

es. cicli

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

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

Linguaggio C

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

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

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

RISC-V assembler

Salto incondizionato

Salti condizionati: ciclo for

- Una possibile implementazione

```
for (i=0; i<100; i++)  
{  
...  
}
```

$i \rightarrow x19$
→

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

incont. (pink arrow from FOR to ENDFOR)

se $i \geq 100$ (pink arrow from bge to ENDFOR)

Salti condizionati: ciclo while

```
while (v[i] == k)  
{  
...  
i = i + 1  
}
```

i → x22
k → x24
v → x25

[i]

```
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:
```

Inc cond

Salva in x10 l'indirizzo della doubleword v[i]
 $x22 \cdot 8 + x25$

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

```
if (i<j)
    k=1;
else k=0;
```

→
i → x19
j → x20
k → x21

`slt x21, x19, x20`

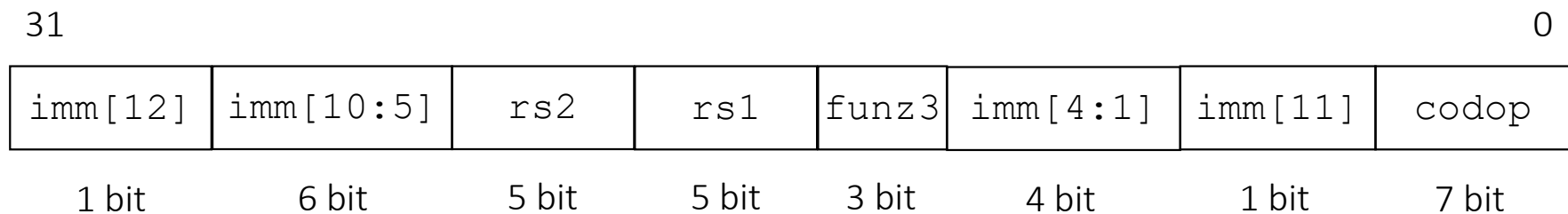
se i < j

← Tipo R in linguaggio
macchina

- 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 "`<=`", inverti condizione e scambio operandi
- Ci sono anche `slti`, `sltu`, e `sltiu`

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 *e distanza*



Salti condizionati e linguaggio macchina

- Si utilizza l'indirizzamento relativo al program counter (PC-relative addressing)
- Il campo immediato di 12 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

$$PC = PC + \text{immediato} * 2$$

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 il moltiplicatore non è 4? I progettisti hanno voluto prevedere la possibilità di rappresentare le istruzioni macchina anche su 16 bit

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:
```

- Codifica in linguaggio macchina del ciclo while precedente

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 incondizionati

- Jump and link register
 - jalr
- Costrutto case/switch

Pseudoistruzioni

- Il linguaggio assembler fornisce “pseudoistruzioni”
 - Una pseudoistruzione esiste solo in linguaggio assembler e non in linguaggio macchina
 - l’assemblatore traduce le pseudoistruzioni nel linguaggio macchina delle corrispondenti istruzioni
 - Quando si effettua per es. il debugging è necessario ricordare quali siano le istruzioni reali
 - L’elenco delle pseudoistruzioni è presente nel manuale di riferimento
- Esempio

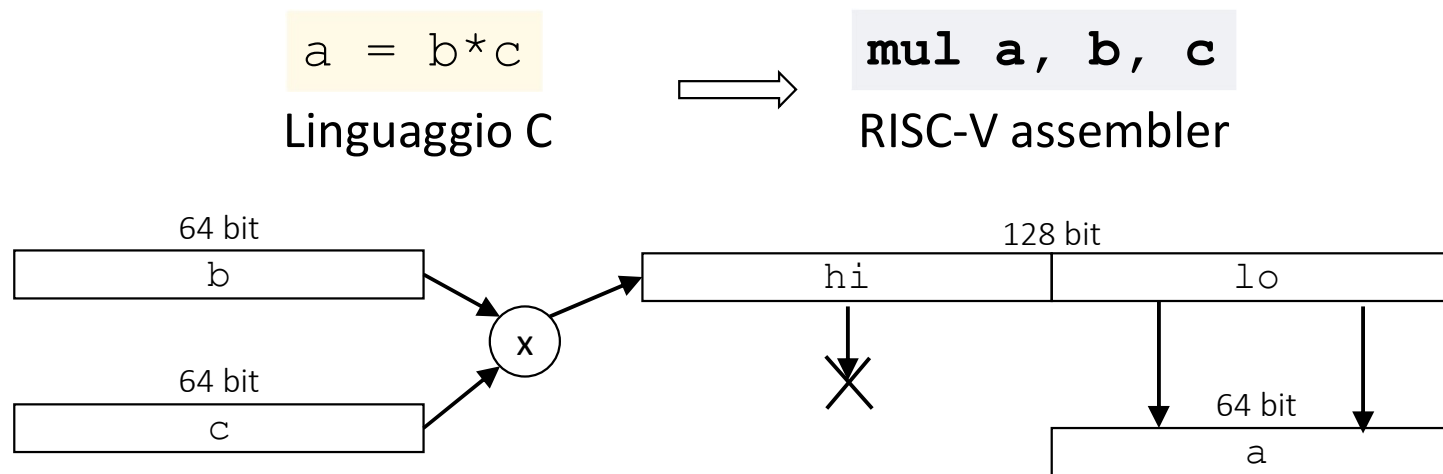
`mv x5, x6` → `addi x5, x6, 0`

`not x5, x6` → `xori x5, x6, -1`

Aritmetica con segno e senza segno

- Le istruzioni aritmetiche fin qui esaminate operano su interi con segno
 - Gli operandi (a 64 bit, nei registri) sono rappresentati in complemento a due: i valori vanno da -2^{63} a $2^{63}-1$
 - Anche i byte-offset di `ld` e `sd` e il numero di istruzioni di cui saltare nella `bne/beq` sono interi con segno da -2^{11} a $2^{11}-1$
- È possibile utilizzare anche operandi senza segno
 - In tal caso i valori vanno da 0 a $2^{64}-1$
 - Invece di `slt`, `slti` si useranno `slt`, `sltiu`
 - `slt` e `sltu` forniranno un risultato diverso se un operando è negativo (es. $-4 < 3$ a 1 con `slt`, ma 0 con `sltu`)
 - Infatti il -4 verrebbe interpretato come un numero molto grande (es. 7 se gli operandi fossero solo di 3 bit), quindi $7 < 3$ risulterebbe falso

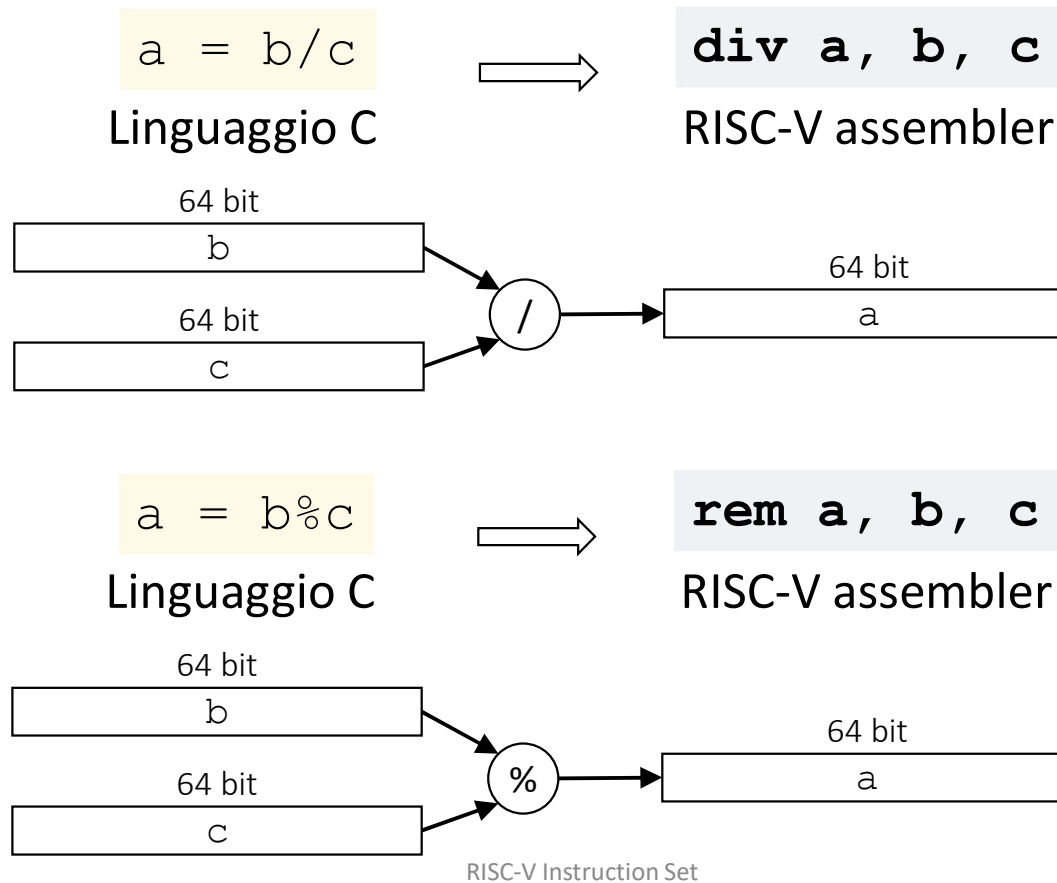
Istruzioni aritmetiche: moltiplicazione



- Viene utilizzato un registro interno nascosto all'utente
- I 64 bit più significativi possono essere ottenuti con l'istruzione

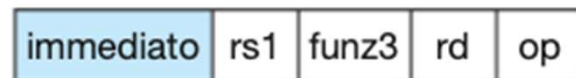
`mulh a, b, c`

Istruzioni aritmetiche: divisione e resto



Modalità di indirizzamento

- Indirizzamento immediato



- Indirizzamento tramite registro



Modalità di indirizzamento

- Indirizzamento tramite base e spiazzamento



- Indirizzamento relativo al program counter

