ISTRUZIONI MARS

Dott. Franco Liberati

Argomenti

01 Struttura di una istruzione MIPS

02 Il set delle istruzioni
- Spostamento
- Logiche aritmentiche
- Confronto e settaggio,
- Salto
- Macchina

03 Pseudoistruzione

Set delle istruzioni del MIPS

Il set delle istruzioni

- Classi di Istruzione
  - istruzione di spostamento dati
  - istruzioni logico ed aritmetiche
  - istruzioni di salto:
    - condizionato
    - non condizionato
    - a funzione (o a subroutine)
    - trap
  - istruzione di controllo macchina

Il set delle istruzioni MIPS

Le istruzioni del linguaggio assembly possono essere divise nelle seguenti categorie:

Istruzioni "Load and Store"
Spostano dati tra la memoria e i registri generali della CPU
(i valori non sono modificati, ma solo spostati)

Istruzioni "Load Immediate"
Caricano, nei registri della CPU, valori costanti

Istruzioni "Data Movement"
Spostano dati tra i registri della CPU

Istruzioni "Aritmetico/logiche"
Effettuano operazioni aritmetico, logiche o di scorrimento sui registri della CPU
(il valore risultante modifica i condition code della ALU)

Istruzioni di "Confronto"
Effettuano il confronto tra i valori contenuti nei registri della CPU

Istruzioni di "Salto condizionato"
Spostano l'esecuzione da un punto ad un altro di un programma se si verifica una condizione

Istruzioni di "Salto non condizionato"
Spostano l'esecuzione da un punto ad un altro di un programma

Istruzioni di "Salto a subroutine"
Spostano l'esecuzione di un programma verso un altro programma (subroutine) e, una volta esaurito, ritornano al processo primordiale

Istruzioni di "Sistema o comando"
Influenzano lo svolgimento del programma (HALT, NOP,BREAK , TRAP …)

Istruzioni di spostamento

Istruzioni di Spostamento

Memoria↔Registro

L'architettura di MIPS è di tipo **Load-and-Store**
In pratica, la maggioranza delle istruzioni di MIPS operano utilizzando i registri interni al processore e non direttamente con i valori presenti nelle celle di memoria

Per questo motivo si usano le istruzioni:

(LOAD) **L** {B|H|W}<reg><address>
il dato è prelevato da una cella di memoria <address>e immesso in un registro<reg>

(STORE) **S**{B|H|W} <reg><address>
il valore di un registro <reg> è salvato in una cella di memoria <address>

Per comodità il programmatore non deve specificare l'indirizzo della Memoria Dati dove risiede il dato, ma può utilizzare una etichetta (definita in .data)

Istruzioni di Spostamento

Memoria↔Registro

lb rdest, address	*Carica un byte sito all'indirizzo address nel registro rdest*
lbu rdest, address	*Carica un byte sito all'indirizzo address nel registro rdest (senza estendere il segno)*
lh rdest, address	*Carica un halfword sito all'indirizzo address nel registro rdest*
lhu rdest, address	*Carica un unsigned-halfword sito all'indirizzo address nel registro rdest (senza estendere il segno)*
lw rdest, address	*Carica una word sita all'indirizzo address nel registro rdest*
la rdest, address	*Carica un indirizzo address nel registro rdest*
sb rsource, address	*Memorizza un byte all'indirizzo address prelevandolo dal registro rsource*
sh rsource, address	*Memorizza un halfword all'indirizzo address prelevandolo dal registro rsource*
sw rsource, address	*Memorizza una word all'indirizzo address prelevandola dal registro rsource*

Istruzioni di Spostamento

Memoria↔Registro

Indirizzo in memoria dell'etichetta *valore*: (262160)	Contenuto: (1471050024)
00000000 00000100 00000000 00010000	01010111 10101110 01110001 00101000

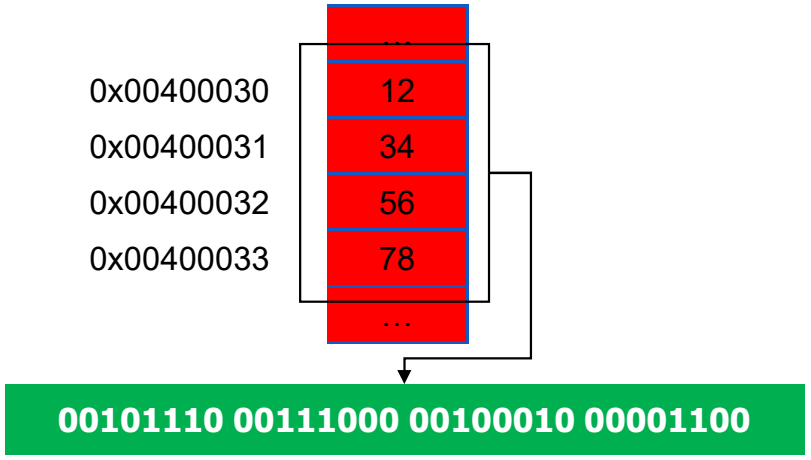
lb rdest, address	lb $t0,valore	00000000 00000000 00000000 00101000
lh rdest, address	lh $t0,valore	00000000 00000000 01110001 00101000
lw rdest, address	lw $t0,valore	01010111 10101110 01110001 10101000
la rdest, address	la $t0,valore	00000000 00000100 00000000 00010000

Istruzioni di Spostamento

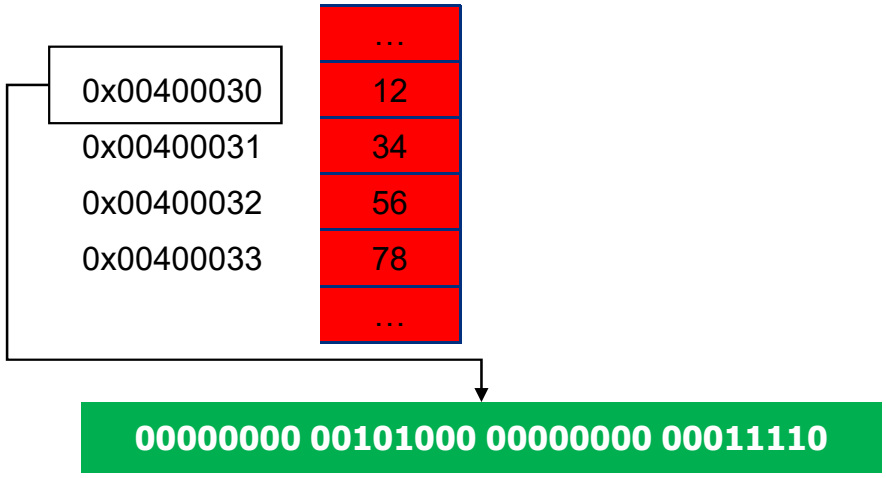
Memoria↔Registro

L'istruzione **lw $t0, address** carica **il contenuto della word** indirizzata dall'etichetta address in memoria

L'istruzione **la $t0, address** carica **l'indirizzo della word** indirizzata dall'etichetta address in memoria (utile quando si vuole caricare in un registro l'indirizzo di una particolare zona di memoria (es.: una variabile))



$t0=0x78563412



$t0=0x00400030

Istruzioni di Spostamento

Memoria↔Registro

L'istruzione **STORE** salva l'operando sito in un registro, in una cella di memoria

Registro	Contenuto
$t0	11010001 10000001 11010101 10101010

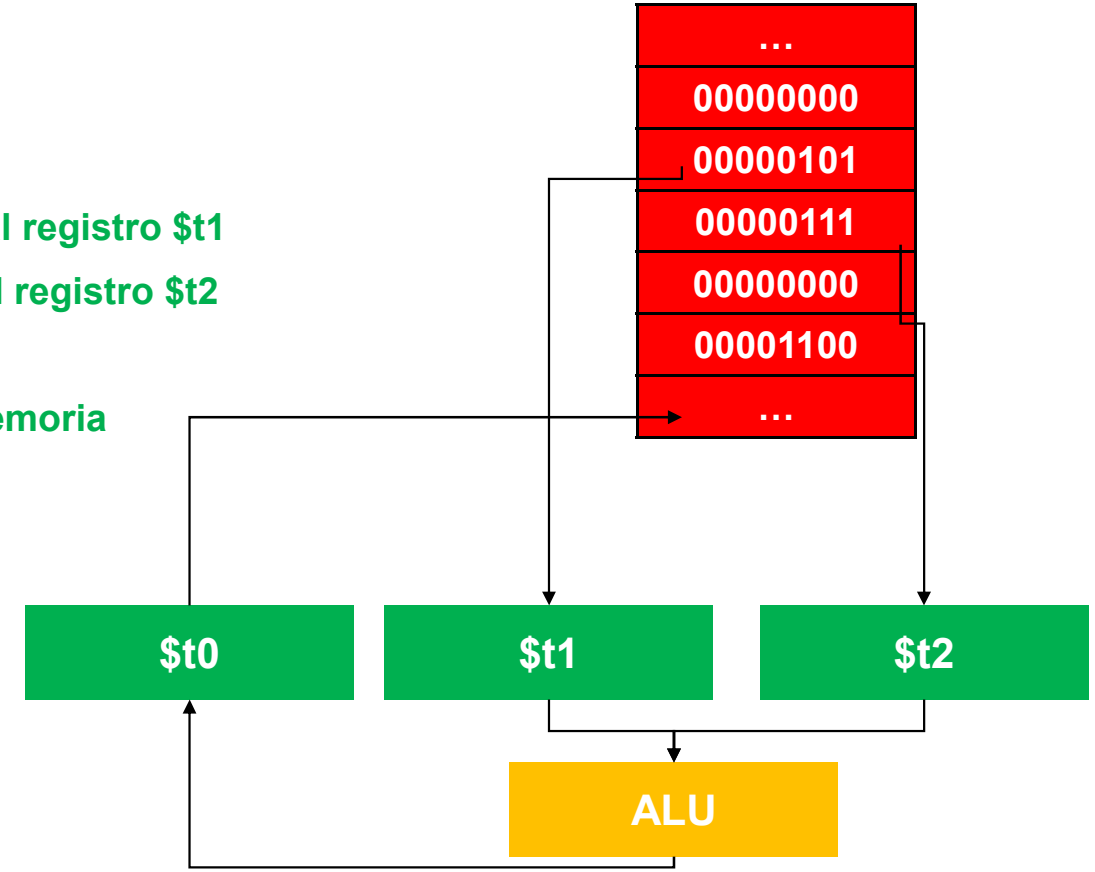
sb rsource, address	sb $t0,variabile1
sh rsource, address	sh $t0,variabile2
sw rsource, address	sw $t0,variabile3

Indirizzo	Contenuto
variabile1	00000000 00000000 00000000 10101010
variabile2	00000000 00000000 11010101 10101010
variabile3	11010001 10000001 11010101 10101010

Istruzioni di Spostamento

ESEMPIO: SOMMA DI DUE OPERANDI SITI IN MEMORIA

```
        .text
        .globl main
main:
        lb $t1, operandA  #carica operandoA dalla memoria al registro $t1
        lb $t2, operandB #carica operandoB dalla memoria al registro $t2
        add $t0, $t1, $t2 #somma gli operandi
        sb $t0, risultato #archivia l'operando sito in $t0 in memoria
        .end

.data
operandA: .byte 5
operandB: .byte 7
risultato: .byte 0
```



DOMANDA: cosa si trova in *risultato* se operandA: .word 256 e operandB:. word 256?

Istruzioni di Spostamento

Memoria↔Registro: Signed - Unsigned

lb $t0, address

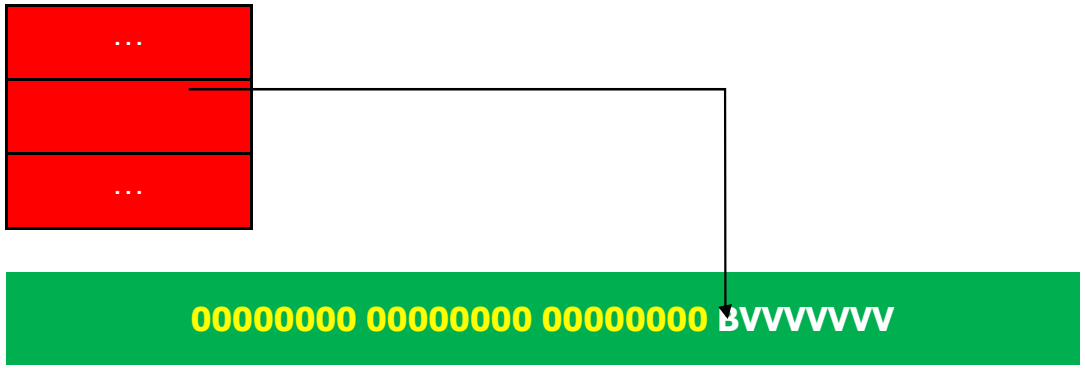
Carica il byte indirizzato da address nel byte meno significativo del registro

Il bit di segno viene esteso

lbu $t0, address

Carica il byte indirizzato da address nel byte meno significativo del registro

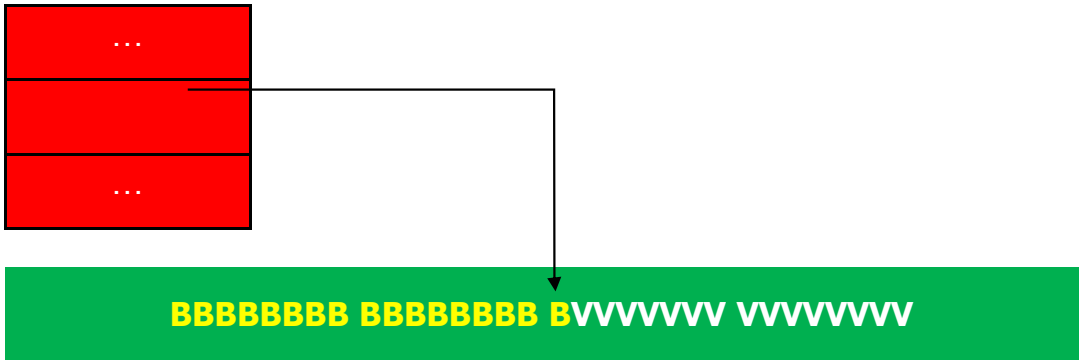
Gli altri tre byte vengono posti a zero



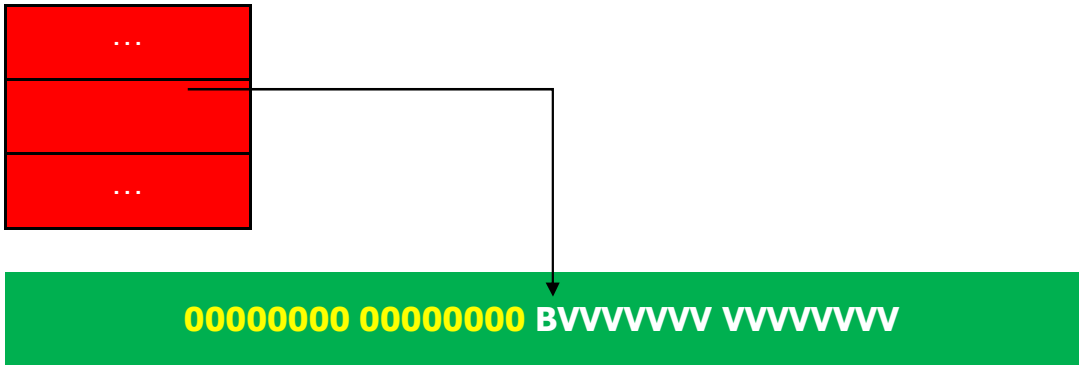
Istruzioni di Spostamento

Memoria↔Registro: Signed - Unsigned

lh $t0, address
Carica 2 byte indirizzati da address nei 2 byte meno significativi del registro
Il bit del segno viene esteso



lhu $t0, address
Carica i 2 byte indirizzati da address nei 2 byte meno significativi del registro
Gli altri due byte vengono posti a zero



Istruzioni di Spostamento

Memoria↔Registro:Signed - Unsigned

lw $t0, address
Carica 4 byte indirizzati da address nel registro
Non c'è alcuna estensione del bit del segno

Istruzioni di Spostamento

Memoria↔Registro:Signed - Unsigned

Valore
01010101 10101100 11110000 10101000

lb $t0,valore	11111111 11111111 11111111 10101000
lbu $t0,valore	00000000 00000000 00000000 10101000
lh $t0,valore	11111111 11111111 11110000 10101000
lhu $t0,valore	00000000 00000000 11110000 10101000
lw $t0,valore	01010101 10101100 11110000 10101000

Istruzioni di Spostamento

Ordinamento dei dati in memoria

Definizione di endianness: disposizione di una parola (word, 32 bit) nei byte di memoria

Little Endian: memorizza prima la “little end” (ovvero i bit meno significativi) della word

Big Endian: memorizza prima la “big end” (ovvero i bit più significativi) della word

Terminologia ripresa da “I viaggi di Gulliver” di Jonathan Swift, in cui due fazioni sono in lotta per decidere se le uova sode devono essere aperte a partire dalla "little end" o dal "big end" dell'uovo

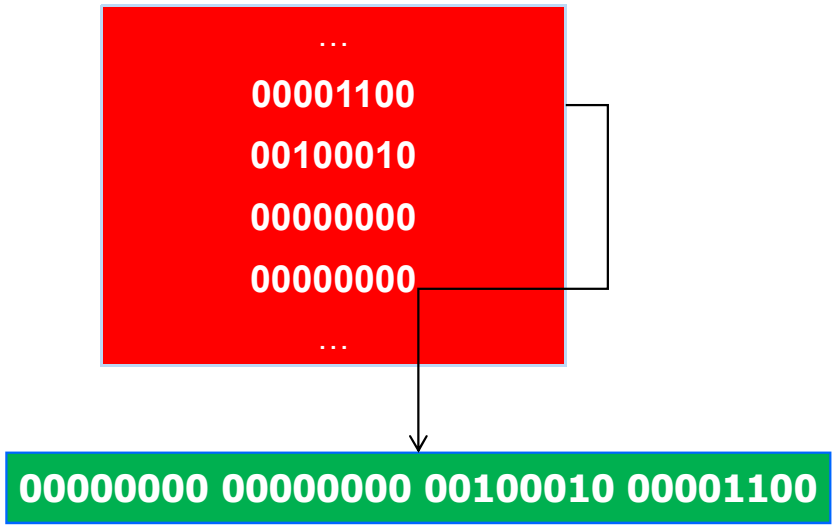
- NB:
  - I processori x86 sono little-endian
  - MIPS può essere little endian o big endian (L’endianness di MARS dipende dal sistema in cui viene eseguito, quindi in generale è little endian)
  - Quando bisogna affrontare i problemi di endianness?
    - Quando si "mischiano" operandi e operazioni ad 8,16 e 32 bit
    - Se si utilizzano operazioni uniformi, non vi sono problemi di sorta

Istruzioni di Spostamento

Ordinamento dei dati in memoria

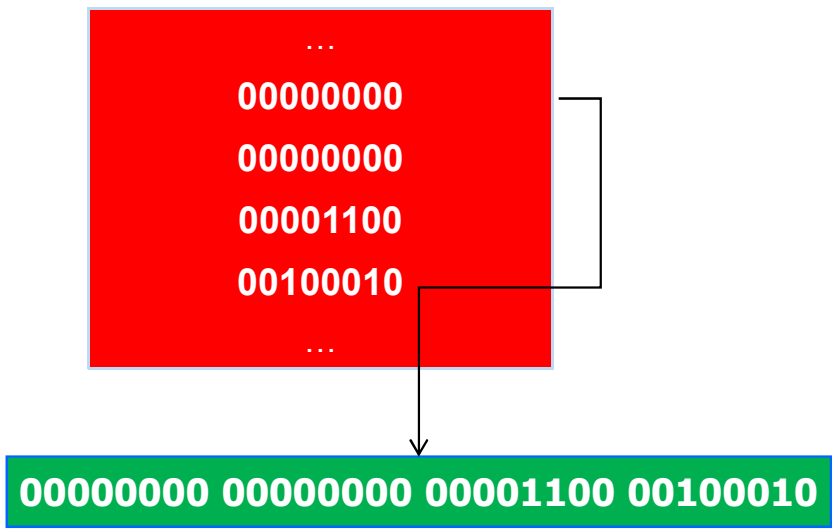
Little Endian:

lh $t0,value



Big Endian:

lh $t0,value



Istruzioni di Spostamento

Inizializzazione (indirizzamento immediato)

Per caricare dei dati numerici nei registri si utilizzano le operazioni di **load immediate**

Si evita di definire una valore in memoria: il valore numerico è nella stessa istruzione

Il valore numerico è anche detto ***IMMEDIATE VALUE***

li rdest, imm	***Muove il valore imm nel registro rdest***
lui rdest, imm	***Muove la halfword imm nella parte più alta dell'halfword del registro rdest***

Esempio:

```
li $t0, 24      #Mette nel registro $t0 il valore 24
                #$t0=00000000 00000000 00000000 00011000
lui $t0,24      #Mette nel registro $t0 il valore 24 nei 16bit più
                #significativi
                #$t0= 00000000 00011000 00000000 00000000
```

Istruzioni di Spostamento

Registro↔Registro

La maggioranza delle istruzioni di MIPS operano tramite i registri interni al processore
Per spostare i valori tra registri si utilizzano le operazioni di **data movement**

move rdest, rsource	*Muove il registro rsource nel registro rdest*
mfhi rdest	*Muove il registro hi nel registro rdest*
mflo rdest	*Muove il registro lo nel registro rdest*
mthi rsource	*Muove il registro rsource nel registro hi*
mtlo rsource	*Muove il registro rsource nel registro lo*

Esempio

```
move $t0, $t1        #Copia il contenuto di $t1 in$t0
```

Istruzioni di Spostamento

ESEMPIO: SCAMBIO DI DUE OPERANDI NEI REGISTRI DI RESIDENZA

Scambio dei valori contenuti in due registri

ESEMPIO:

Prima:

$t0=5 e $t1=3

Dopo

$t0=3 e $t1=5

```
        .text
        .globl main

main:
        li $t0,5          #inizializzazione x a 5
        li $t1,3          #inizializzazione y a 3
        move $t2,$t1      #temp=y
        move $t1,$t0      #y=x
        move $t0,$t2      #x=temp
        xor  $t2,$t2      #pulizia di temp (temp=0)

        li $v0,10     #terminazione del programma
        syscall
```

Istruzioni di Spostamento

ESEMPIO: SCAMBIO DI DUE OPERANDI SITI IN MEMORIA

Definiti due valori in memoria valx,valy
Scambiare la loro posizione in memoria

```
        .text
        .globl main
main:
        lw $t0,valx       #trasferimento di x in $t0
        lw $t1,valy       #trasferimento di y in $t1
        move $t2,$t1      #scambio
        move $t1,$t0
        move $t0,$t2
        xor  $t2,$t2 ,$t2#pulisco $t2
        sw $t0,valx       #archiviazione del valore y
                          #in $t0
        sw $t1,valy       #archiviazione del valore x
                          #in $t1
        li $v0,10         #terminazione programma
        syscall
.data
valx :.word 5
valy :.word 10
```

Istruzioni di Spostamento

ESEMPIO: SCAMBIO DI DUE OPERANDI SITI IN MEMORIA (VARIANTE)

Definiti due valori in memoria valx,valy

Scambiare la loro posizione in memoria

```
        .text
        .globl main

main:
        lw $t0,valx     #trasferimento di x in $t0
        lw $t1,valy     #trasferimento di y in $t1
        sw $t0,valy     #trasferimento di $t1 in x
        sw $t1,valx     #trasferimento di $t0 in y
        li $v0,10       #terminazione programma
        syscall

        .data
valx :.word 5
valy :.word 10
```

Istruzioni di Spostamento

Influenza sui Condition Code

Istruzione	C	N	Z	W	P
LOAD	X	X	X	X	X
MOVE	X	X	X	X	X
STORE	X	X	X	X	X

Istruzioni logiche matematiche

Istruzioni Logiche-Aritmetiche

Gli elaboratori elettronici hanno come principale motivo di esistenza la possibilità di svolgere un gran numero di operazioni logiche ed aritmentiche in tempi rapidissimi

Esempio

```
add $t2, $t0, $t1     #Somma il contenuto di $t1e $t0 e lo mette in $t2
xor $t2,$t2,$t2       #Inizializza il registro $t2 a zero
```

NB: le istruzioni aritmetiche operano sfruttando i registri.
I registri sono usati per tre motivi principali:

- Costruiti con una tecnologia performante
- Sono interni alla CPU
- Consentono di trovare i dati su cui operare utilizzando pochi bit (istruzioni a lunghezza fissa)

Istruzioni Aritmetiche

Sono operazione aritmetiche svolte dall'ALU. Si sfruttano i registri ad uso generale

add rd, rs, rt	rd = rs + rt (con overflow)
addu rd, rs, rt	rd = rs + rt (senza overflow)
addi rd, rs, imm	rd = rs + imm (con overflow)
addiu rd, rs, imm	rd = rs + imm (senza overflow)
sub rd, rs, rt	rd = rs - rt (con overflow)
subu rd, rs, rt	rd = rs - rt (senza overflow)
neg rd, rs	rd = - rs (con overflow)
negu rd, rs	rd = - rs (senza overflow)
abs rd, rs	rd = |rs|

Istruzioni Aritmetiche

Immediate - Unsigned

Versioni immediate (i)

❑Le versioni immediate (i) delle istruzioni precedenti utilizzano un valore costante al posto di un operando in un registro

❑Non sempre è necessario specificare la **i**; l'assemblatore è in grado di riconoscere anche istruzioni.

Esempio:

add $t0, $t0, 1 è riscritto dall'assemblatore come addi $t0,$t0,1

Versioni unsigned (u)

❑ Le versioni unsigned delle istruzioni non gestiscono le problematiche relative all'overflow

Istruzioni Aritmetiche

Sono operazione aritmetiche svolte dall'ALU. Si sfruttano i registri ad uso generale e quelli **hi** e **lo**

mult rs, rt	hi,lo = rs · rt (signed, overflow impossibile)
multu rs, rt	hi,lo = rs · rt (unsigned, overflow impossibile)
mul rd, rs, rt	rd = rs · rt (senza overflow)
mulo rd, rs, rt	rd = rs · rt (con overflow)
mulou rd, rs, rt	rd = rs · rt (con overflow, unsigned))
div rs, rt	hi,lo = resto e quoz. di rs / rt (signed con overflow)
divu rs, rt	hi,lo = resto e quoz. di rs / rt (unsigned, no overflow)
div rd, rs, rt	rd = rs / rt (signed con overflow)
divu rd, rs, rt	rd = rs / rt (unsigned)
rem rd, rs, rt	rd = resto di rs / rt (signed)
remu rd, rs, rt	rd = resto di rs / rt (unsigned)

Istruzioni Aritmetiche

ESEMPIO: PARITA' DI UN OPERANDO

Realizzare un programma che definita una variabile (operando intero a 8bit) residente in memoria archivia in memoria il valore 0 se l'operando della variabile è pari o 1 se l'operando della variabile è dispari

```
        .text
        .globl main

main:
        lb $t0,pippo      #trasferimento operando in $t0
        li $t1,2          #impostazione del valore 2
        rem $t2,$t0,$t1   #calcolo resto della divisione
        sw $t2,parita     #archiviazione del risultato

        li $v0,10         #terminazione del programma
        syscall

        .data
pippo :.byte 5
parita :.word 0
```

Istruzioni di Aritmetiche

ESEMPIO: NUMERI CONTIGUI

Si scriva un programma in linguaggio assembly che definita una variabile (valore intero a 32bit) in memoria riporti in $t0 l'intero precedente, in $t1 il numero corrente e in $t2 il successivo

```
        .text
        .globl main

main:
        lb $t1,pippo     #trasferimento operando in $t1
        subi $t0,$t1,1   #$t0←$t1-1
        addi $t2,$t1,1   #$t2←$t1+1
        li $v0,10         #terminazione del programma
        syscall

        .data
pippo :.word 5
```

Istruzioni di Aritmetiche

ESEMPIO: MEDIA TRA INTERI

Si scriva un programma in linguaggio assembly che definite tre variabili in memoria **TempMarzo** e **TempAprile** e **TempMaggio** riporta la media (fra interi) in $t9

Esempio
TempMarzo=22 TempAprile=29
TemoMaggio=33
media=(22+29+32)/3=61/3=**20**

```
        .text
        .globl main

main:
        lh $t0, TempMarzo #trasferimento operando in $t0
        lh $t1, TempAprile #trasferimento operando in $t1
        lh $t2, TempMaggio #trasferimento operando in $t2
        li $t4,3            #inizializzazione del valore 3
        add  $t3,$t0,$t1            #$t3←$t0+$t1
        add $t3,$t3,$t2             #$t3←$t3+$t2 (somma)
        div $t9, $t3,$t4            # media=(somma/3)
        li $v0,10           #terminazione del programma
        syscall

        .data
TempMarzo:.half 22
TempAprile:.half 29
TempMaggio:.half 33
```

Istruzioni Logiche

Sono operazione logiche svolte dall'ALU. Si sfruttano i registri ad uso generale e si applicano bit per bit

and rd, rs, rt	rd = rs AND rt
andi rd, rs, imm	rd = rs AND imm
or rd, rs, rt	rd = rs OR rt
ori rd, rs, imm	rd = rs OR imm
xor rd, rs, rt	rd = rs XOR rt
xori rd, rs, imm	rd = rs XOR imm
nor rd, rs, rt	rd = NOT (rs OR rt)
not rd, rs	rd = NOT rs

Istruzioni Logiche-Aritmetiche

Lo **Shift** (spostamento) sposta di k posizioni verso destra (o sinistra) i bit di un dato contenuto in un registro.
Lo spostamento perde i bit dalla parte della movimentazione; mentre è colmato con 0 nella direzione opposta: corrisponde ad una moltiplicazione per 2^k (con k numero di posizioni spostate) quando si sposta il valore da destra verso sinistra; mentre corrisponde ad una divisione per 2^k (con k numero di posizioni spostate) quando si sposta il valore da sinistra verso destra

Il **Rotate** (rotazione) mantiene i bit dalla parte della movimentazione riproponendo i bit spostati nella direzione opposta

sll rd, rs, rt	rd = rs è shiftato verso sinistra di rt mod 32 bits
srl rd, rs, rt	rd = rs s è shiftato verso destra di rt mod 32 bits
rol rd, rs, rt	rd = rs è ruotato verso sinistra di rt mod 32 bits
ror rd, rs, rt	rd = rs è ruotato verso destra di rt mod 32 bits

Istruzioni Logiche-Aritmetiche

sll $t2, $t1,3

$t1	**00000000000000000000000010011101**	157
$t2	**00000000000000000000010011101000**	1256

sll $t2, $t1,3

$t1	**00000000000000000000000000000001**	1
$t2	**00000000000000000000000000001000**	8

ror $t2, $t1,3

$t1	**00000000000000000000000010011101**	157
$t2	**10100000000000000000000000010011**	2684354579

Istruzioni di Spostamento

Influenza sui Condition Code

Istruzione	C	N	Z	W	P
ADD	1/0	1/0	1/0	1/0	1/0
CMP	1/0	1/0	1/0	1/0	1/0
NEG	1/0	1/0	1/0	1/0	1/0
SUB	1/0	1/0	1/0	1/0	1/0
AND	0	1/0	1/0	0	1/0
OR	0	1/0	1/0	0	1/0
XOR	0	1/0	1/0	0	1/0
NOT	0	1/0	1/0	0	1/0
SL	1/0	1/0	1/0	1/0	1/0
SR	1/0	1/0	1/0	1/0	1/0
ROL	1/0	0	0	0	1/0
ROR	1/0	0	0	0	1/0

Esercizio proposto per casa

Scrivere un programma che, definite due variabili (interi a 32bit) in memoria **Batman** e **Superman**, determina le seguenti informazioni:

1. $t0=0 se Batman è un numero positivo o $t0=1 se Batman è un numero negativo
2. $t1=0 se Superman è pari $t1=1 se Superman è dispari
3. $t2 riporta Batman+Superman
4. $t3 riporta la somma, in valore assoluto, delle due variabili

Istruzioni di Confronto e Settaggio

Istruzioni di Confronto e Settaggio

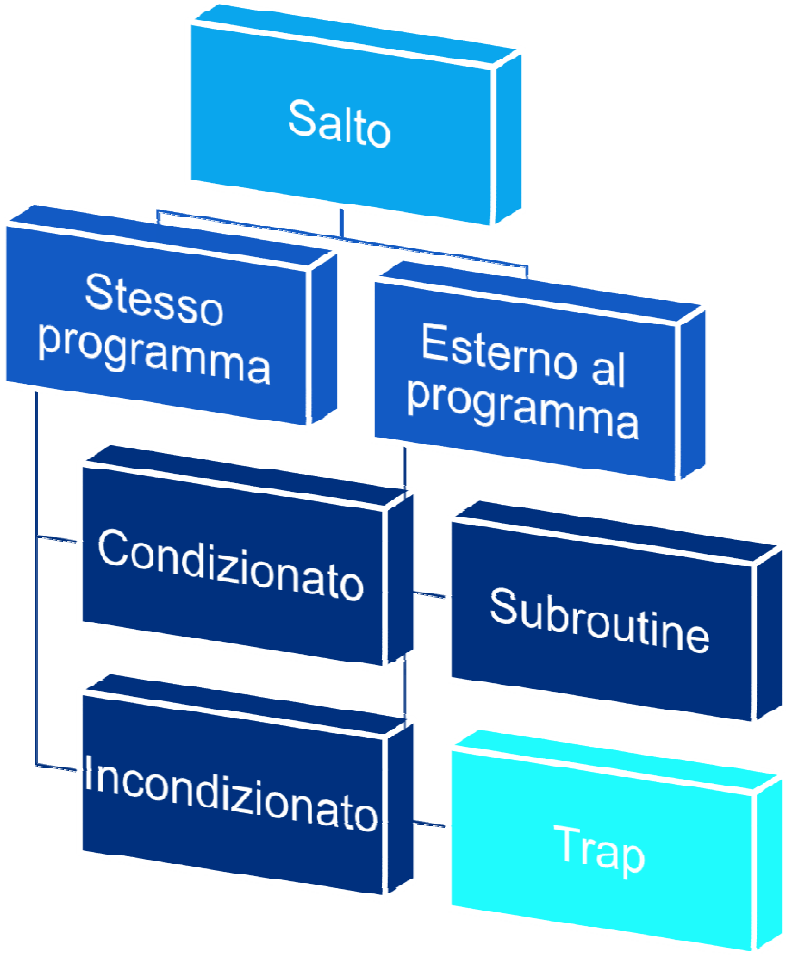
Settano dei registri in base a dei confronti (non sono usate esplicitamente dal programmatore, ma sono sfruttare dall'assemblatore per riscrivere delle pseudoistruzioni)

slt rd, rs, rt	Setta il registro rd a **1** se rs<rt, 0 altrimenti (con overflow)
sltu rd, rs, rt	Setta il registro rd a **1** se rs<rt, 0 altrimenti (no overflow)
slti rd, rs, imm	Setta registro rd a **1** se rs<imm, 0 altrimenti (con overflow)
sltiu rd, rs, imm	Setta registro rd a **1** se rs<imm, 0 altrimenti (no overflow)
sle, sgt, sge, seq, sne	Analoghe per testare $\leq$, >,$\geq$, =, $\neq$

Istruzioni di Salto

Istruzioni di Salto

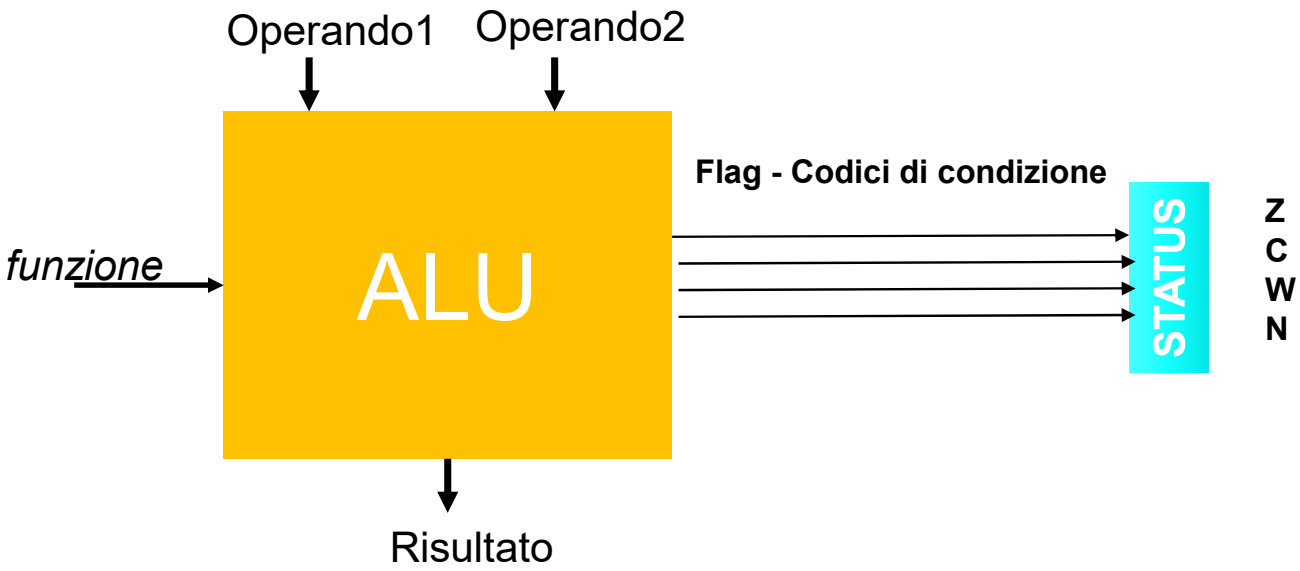
- Le istruzioni di salto si dividono in:
  - salto all'interno dello stesso programma
    - **condizionato**: il salto è eseguito in base ad una certa condizione stabilita dal programmatore (Branch)
    - **incondizionato**: il salto è sempre eseguito (Jump), senza valutare alcuna condizione
  - salto ad un altro programma: **salto a subroutine** (salto a sottoprogramma)
  - ***trap*** (o interruzioni software)



Istruzioni di Salto

SALTO CONDIZIONATO

Le istruzioni condizionate fanno riferimento a i *flag* o codici di condizione presenti nello STATUS REGISTER derivanti da operazione logico-aritmetiche



Istruzioni di Salto

SALTO CONDIZIONATO

Confronto tra due operandi siti in registri ad uso generale

beq rs, rt, target	***Salta all'istruzione con etichetta target se rs=rt***
bne rs, rt, target	***Salta all'istruzione con etichetta target se rs≠rt***
bge rs, rt, target **bgeu rs, rt, target**	***Salta all'istruzione con etichetta target se rs≥rt*** ***Comparazione senza considerare il segno***
bgt rs, rt, target **bgtu rs, rt, target**	***Salta all'istruzione con etichetta target se rs>rt*** ***Comparazione senza considerare il segno***
ble rs, rt, target **bleu rs, rt, target**	***Salta all'istruzione con etichetta target se rs≤rt*** ***Comparazione senza considerare il segno***
blt rs, rt, target **bltu rs, rt, target**	***Salta all'istruzione con etichetta target se rs<rt*** ***Comparazione senza considerare il segno***

Istruzioni di Salto

SALTO CONDIZIONATO

Confronto tra un operando sito in un registro ad uso generale e lo zero

bgez rs, target	***Salta all'istruzione con etichetta target se rs≥0***
bgtz rs, target	***Salta all'istruzione con etichetta target se rs<0***
blez rs, target	***Salta all'istruzione con etichetta target se rs≤0***
bltz rs, target	***Salta all'istruzione con etichetta target se rs<0***
beqz rs, target	***Salta all'istruzione con etichetta target se rs=0***
bnez rs, target	***Salta all'istruzione con etichetta target se rs≠0***

Istruzioni di Salto

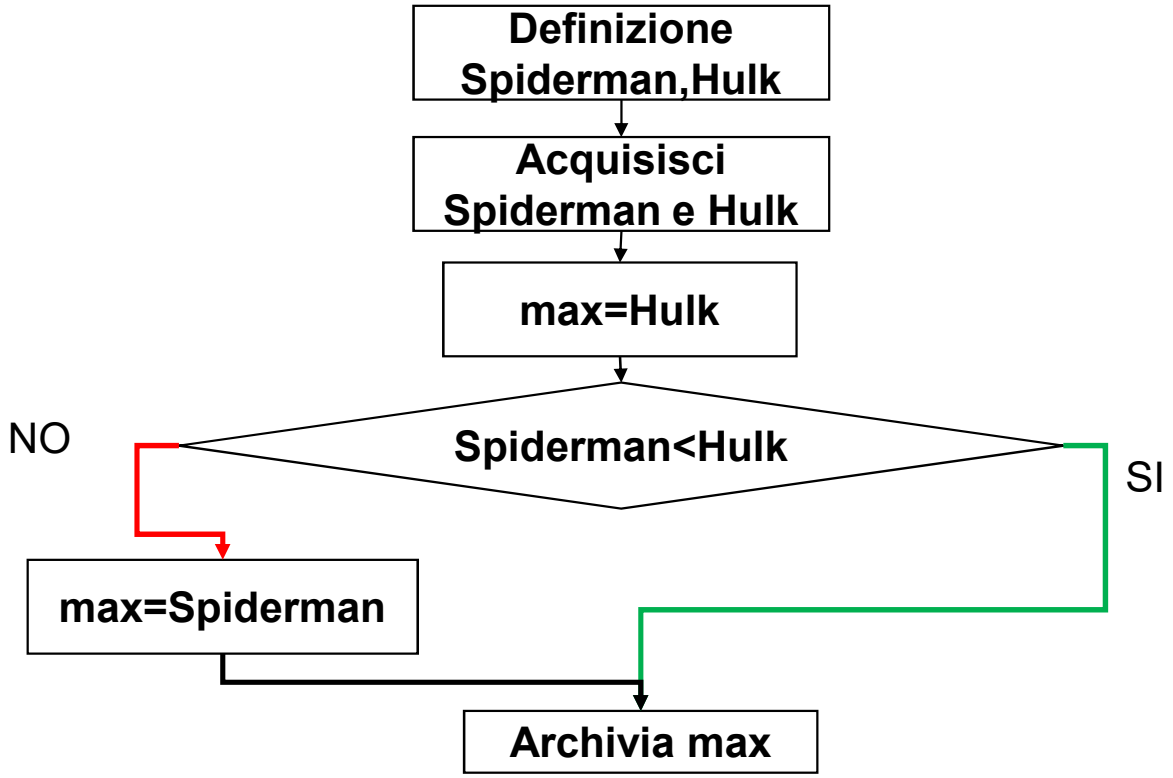
SALTO CONDIZIONATO E FLAG

Mnemonico	Significato	Flag interessato
EQ	=	Z=1
NE	!=	Z=0
CS	> (unsigned)	C=1
	< (unsigned)	C=0
HI	>	C=1 e Z=0
LS	<	C=0 e Z=1
MI	negativo	N=1
PL	positivo	N=0

Istruzioni di Salto

ESEMPIO: MASSIMO TRA DUE NUMERI

Realizzare un programma che valuta il massimo fra due operandi contenuti nelle variabili Spiderman e Hulk (interi a 32bit).
Riportare il massimo in una locazione di memoria definita dall'etichetta Massimo (intero a 32bit)



Istruzioni di Salto

ESEMPIO: MASSIMO TRA DUE NUMERI

Realizzare un programma che valuta il massimo fra due operandi contenuti nelle variabili Spiderman e Hulk (interi a 32 bit). Riportare il massimo in una locazione di memoria definita dall'etichetta Massimo(intero a 32bit)

```
            .text
            .globl main
main:
            lw $t0,Spiderman    #prelievo del primo operando
            lw $t1,Hulk         #prelievo del secondo operando
            move $t2,$t1        #In t2 conservo Hulk come se fosse il
                                #massimo
            blt $t0,$t1,salto   #se Spiderman è minore di Hilk allora salto
                                #perché Hulk è il massimo…
            move  $t2,$t0       #...atrimenti aggiorno il valore massimo
                                #mettendo x in $t2
salto:
            sw $t2, Massimo     #conservo il massimo in memoria

            li $v0,10           #Fine programma
            syscall

            .data
Spiderman: .word 34
Hulk: .word 76
Massimo: .word 0
```

Istruzioni di Salto

SALTO INCONDIZIONATO

Le istruzioni incondizionate spostano l'esecuzione da un punto ad un altro di un programma

j target	***Salto incondizionato all'istruzione con etichetta* target**

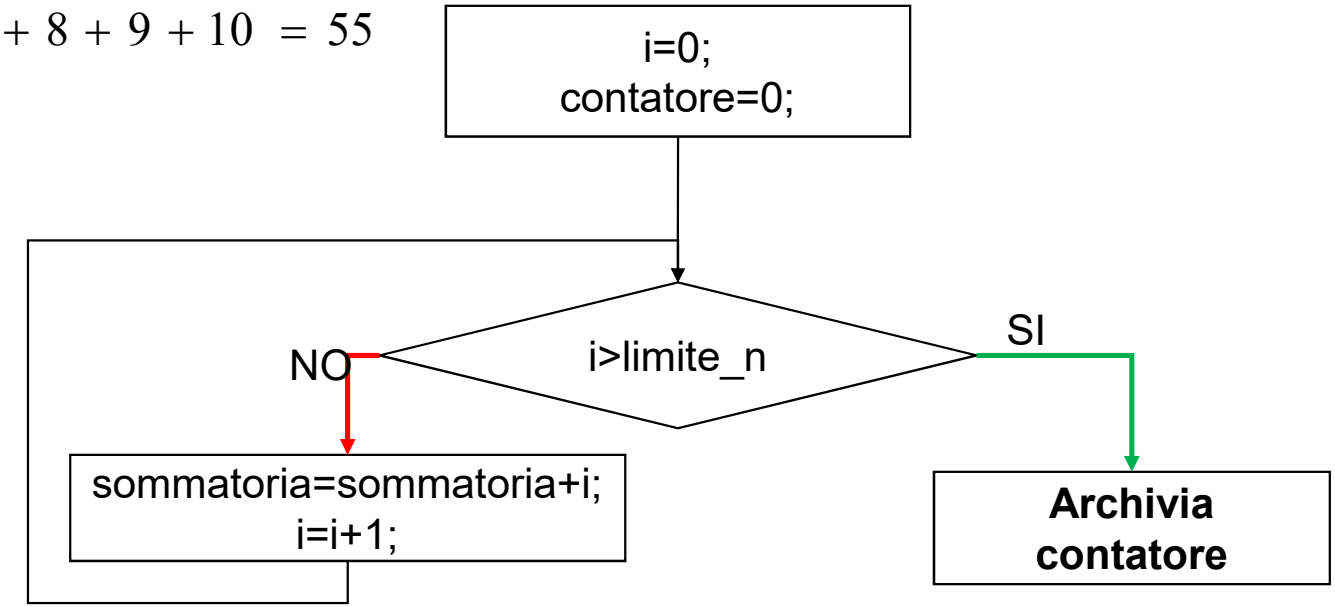
Istruzioni di Salto

ESEMPIO: CONTATORE

Realizzare un programma che svolga la sommatoria dei primi *n* numeri interi positivi a partire da zero

$$ris = \sum_{i=0}^{n} i = 0 + 1 + 2 + ... + n$$

$$ris = \sum_{i=0}^{n=10} i = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55$$



Istruzioni di Salto

ESEMPIO: CONTATORE

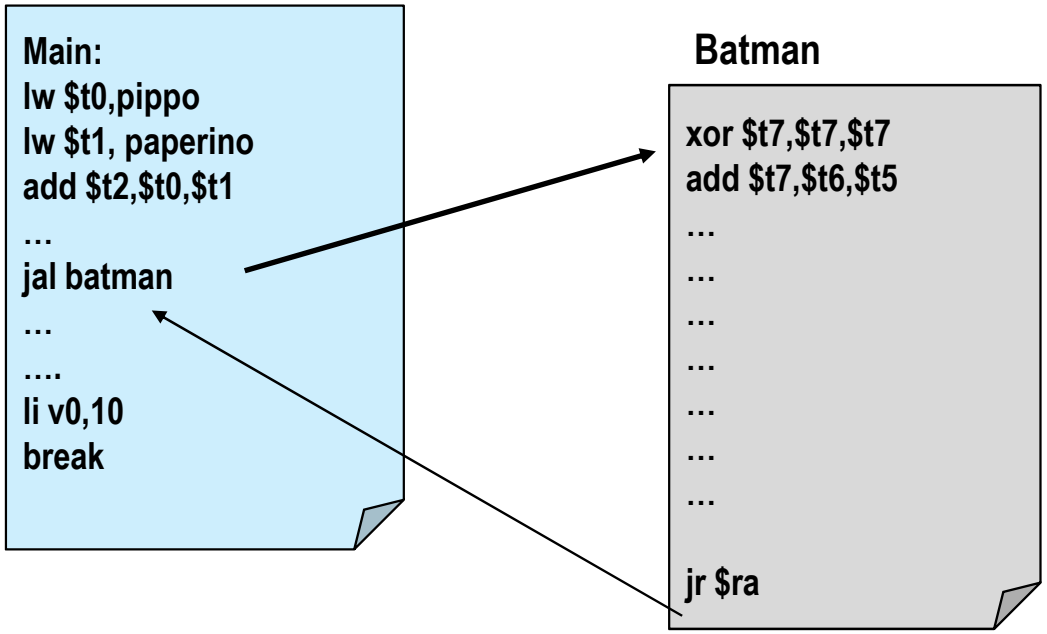
Realizzare un programma che svolga la sommatoria dei primi *n* numeri interi positivi a partire da zero

```
        text
        .globl main

main:
        li $t0,0            #impostazione i
        li $t1,0            #impostazione sommatoria
        lb $t2,limite_n     #lettura limite superiore i
ciclo:
        bgt $t0,$t2,fine    #salto a fine del ciclo iterativo se i>10 (limite di i)
        add $t1,$t1,$t0     #incremento sommatoria
        addi $t0,$t0,1      #incremento i
        j ciclo             #ripetizione del ciclo iterativo
fine:
        sw $t1,sommatoria
        li $v0,10
        syscall

        .data
        sommatoria:.word 0
        limite_n:.byte 10
```


Salto a subrotuine

Istruzioni di Salto a Subroutine

Le **istruzioni di salto a subroutine** (o salto a sottoprogramma, o salto a funzione) spostano l'esecuzione da un programma (**programma chiamante**) ad un altro programma (**programma chiamato** o sottoprogramma), salvando contestualmente l'indirizzo dell'istruzione successiva al salto (indirizzo di ritorno). Il sottoprogramma è eseguito fino alla fine e poi, ripristinando l'indirizzo di ritorno, si continua nell'esecuzione del programma chiamante.



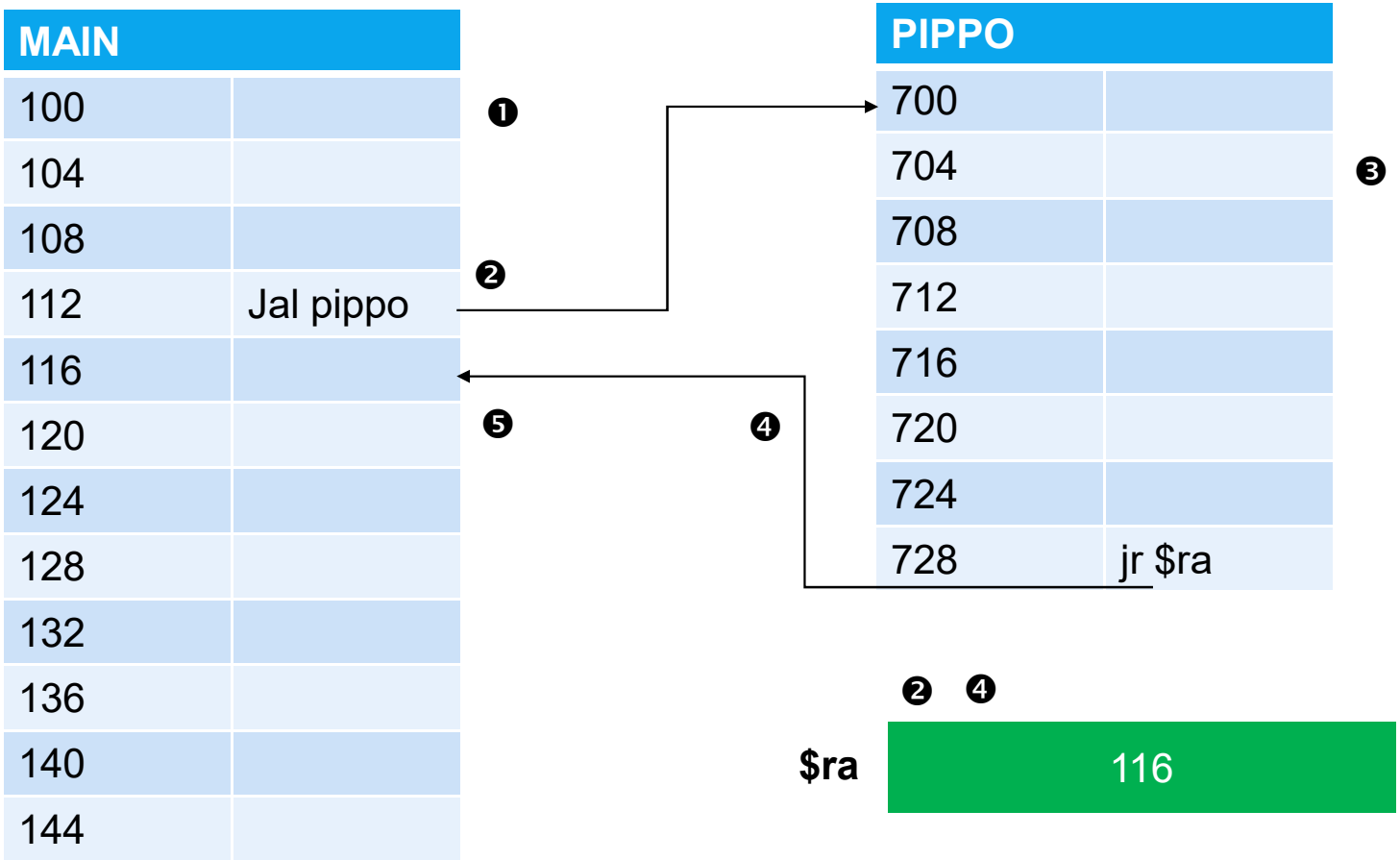
Istruzioni di Salto a Subroutine

Il compito di passaggio da programma chiamante a sottoprogramma è svolto in MIPS dall'istruzione **jal**, sigla derivata dai termini anglosassoni *jump and link* ('salta e collega'). Quando si esegue questa istruzione il MIPS, contestualmente, salva l'indirizzo di ritorno nel registro **$ra**.
Per garantire il passaggio da sottoprogramma a programma chiamante, il programmatore deve inserire una istruzione che recupera l'indirizzo di ritorno conservato in $ra e lo pone nel Contatore di Programma; nel MIPS è jr, cioè *jump and return* ('salta e ritorna').

Istruzioni di salto a subroutine	
Istruzione	**Significato**
jal target	*Salto incondizionato all'istruzione sita all'indirizzo* target *(inizio della subroutine) e salvataggio dell'indirizzo della prossima istruzione in* **$ra**
jr source	*Salto incondizionato all'istruzione che ha indirizzo memorizzato nel registro* source (ritorno da subroutine)
jalr rsource, rdest	*Salto incondizionato all'istruzione che ha indirizzo memorizzato nel registro* rsource *e salvataggio dell'indirizzo della prossima istruzione in* rdest

Istruzioni di Salto a Subroutine

Indirizzo di ritorno



1 Esecuzione del programma MAIN
2 JAL:Salto a subroutine e memorizzazione indirizzo di ritorno nel registro $ra
3 Esecuzione subroutine
4 JR: Ritorno a funzione (contenuto di $ra nel PC)
5) Prosieguo del programma MAIN

Istruzioni di Salto a Subroutine

Parametri ingresso e uscita

Quando si effettua un JAL, di norma tutti i registri non preservanti $t0-$t9 si azzerano, mentre quelli preservanti $s0-$s9, $a0,$a1,$a2,$a3,$v0,$v1 mantengono i loro valori.
Quando si effettua un JAL i parametri di ingresso della subroutine soni posti, **per convenzione**, nei registri preservanti **$a0**,**$a1**,**$a2**,**$a3**; mentre il risultato elaborato dalla subroutine è sito nei registri preservanti **$v0** o **$v1**

ESEMPIO

```
Read(BASE);
Read(ESPONENTE);
ESP=POW(BASE,ESPONENTE);
Print(ESP);
```

```
POW (BASE, ESPONENTE)
        {
        RIS=1;
        for (i=0;i<ESPONENTE;i++)
                RIS=RIS*BASE;
                i=i+1;
        }
```

ESEMPIO

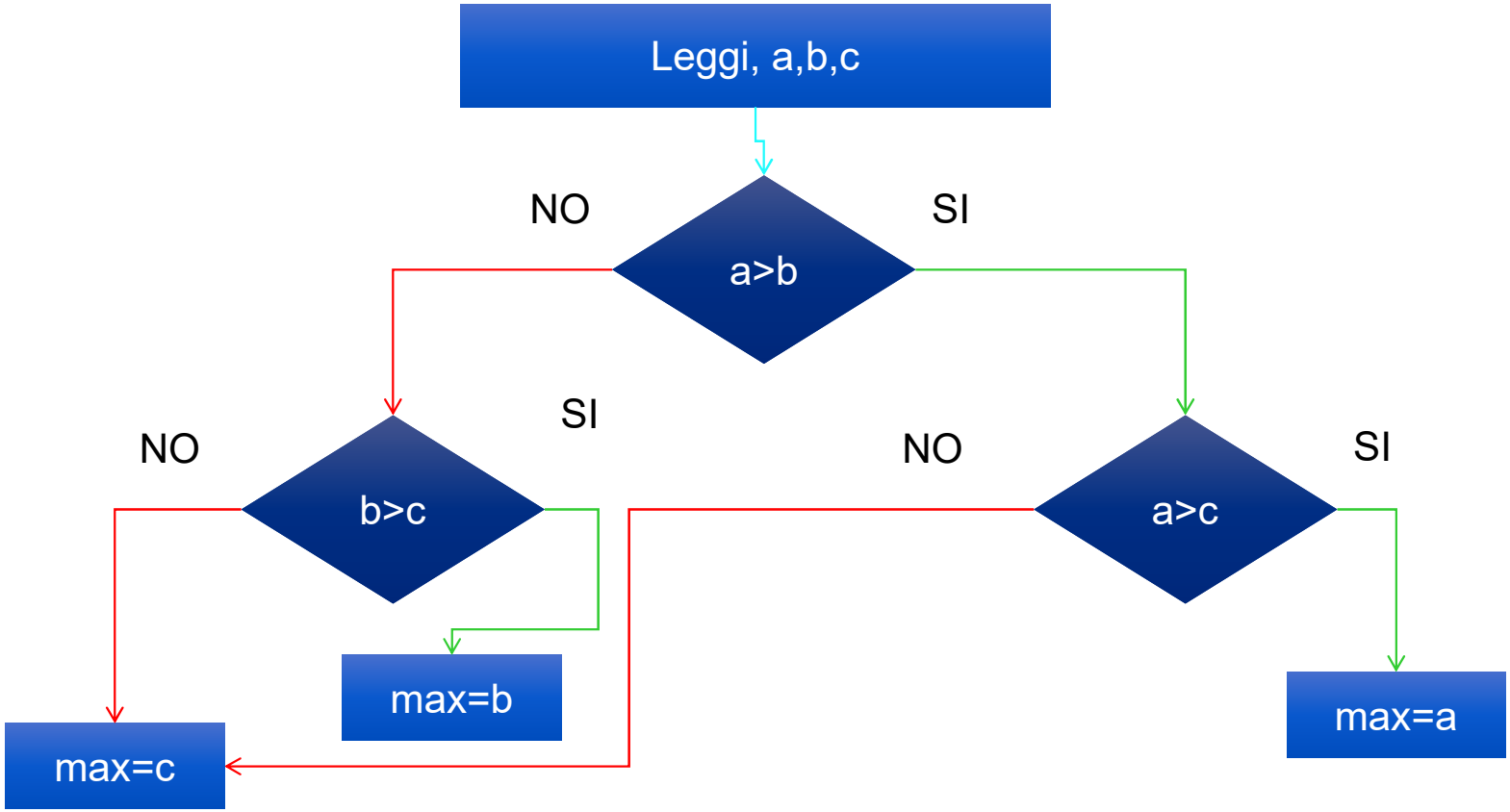
```
lw $t0,BASE
lw $t1,ESPONENTE
move $a0,$t0
move $a1, $t1
jal POW
move  $t0,$v0
sw $t0, ESP
```

```
POW:
        move $t0,$a0 #prelievo parametro base
        move $t1,$a1 #prelievo parametro esponente
        li $t2,1        #ris=1
        li $t3,0        #contatore i
ciclo:
        bge $t3,$t1, fine       #finisco se i>esponente
        mul $t2,$t2,$t1         #calcolo esponente (ris)
        add $t3,$t3,1           #incremento i
        j ciclo
fine:
        move $v0,$t2   #riporto il risultato in $v0
        jr $ra          #ritorno a funzione chiamante
```

Istruzioni di Salto a Subroutine

ESEMPIO: RIUSO SUBROUTINE

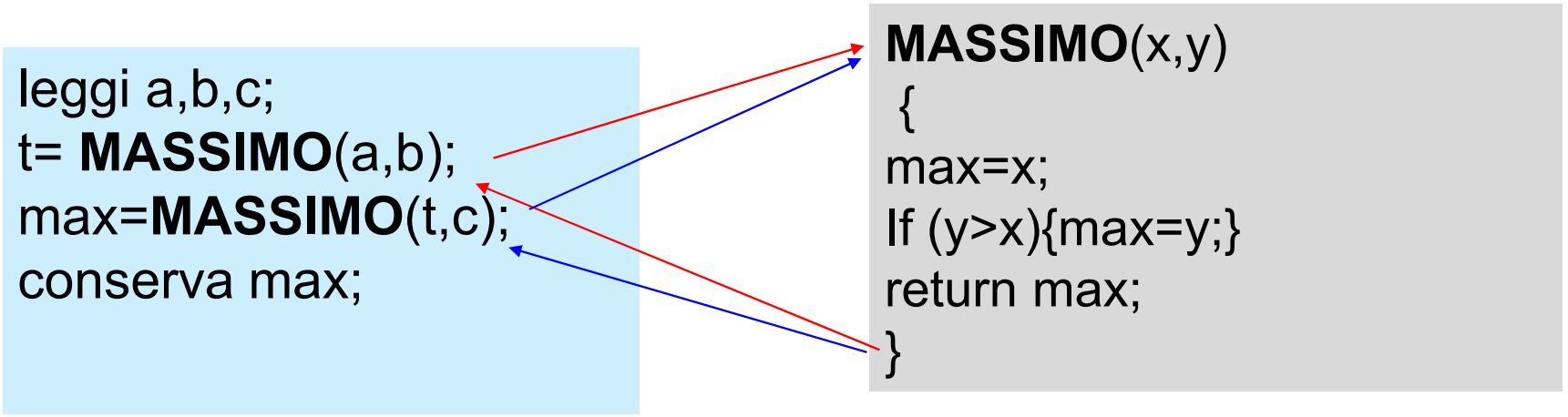
Realizzare un programma che individua il massimo tra tre numeri



Istruzioni di Salto a Subroutine

ESEMPIO: RIUSO SUBROUTINE

Realizzare un programma che individua il massimo tra tre numeri



Istruzioni di Salto a Subroutine

ESEMPIO: RIUSO SUBROUTINE

Realizzare un programma che individua il massimo tra tre numeri

```
.text
.globl main

main:
        lw $a0,x      #lettura primo valore
        lw $a1,y      #lettura secondo valore
        jal MASSIMO #salto a funzione
        move $a0,$v0 #recupero massimo dalla funzione
        lw $a1,z      #lettura terzo valore
        jal MASSIMO #salto a funzione
        move $a0,$v0 #recupero massimo dalla funzione
        move $t0,$a0 #massimo in T0

        li $v0,10
        syscall

.data
x:.word 45
y:.word 100
z:.word 77
```

```
MASSIMO:
# PARAMETRI INGRESSO: A0 e A1 valori interi
# PARAMETRO USCITA:  V0 massimo tra A0 e A1
        move $v0,$a0     #Imposto A0 come massimo
        bgt $a0,$a1,fine   #Se A0>A1 allora finisco
        move $v0,$a1     #Imposto A1 come massimo
fine:
        jr $ra
```

Istruzioni Macchina

Istruzioni Macchina

Le **istruzioni macchina** sono istanze tipiche dell'elaboratore.

Istruzioni Macchina	
Istruzione	**Significato**
NOP	*Nessuna operazione*
HALT	*Interruzione del sistema*
BREAK	*Interruzione del programma*

NB:Non tutte queste sono eseguibili dal simulatore MARS


Pseudoistruzioni

Pseduolstruzioni

Una istruzione in linguaggio assemblativo corrisponde ad una istruzione in linguaggio macchina; eccetto per una **pseudoistruzione** (macro-istruzione o istruzione composita).
Una pseudoistruzione è una opzione offerta al programmatore per consentirgli di usare una singola, e di solito semplice ed immediata, istruzione che, durante la fase di assemblaggio, è riscritta dall'assemblatore in una con un formato più complesso o scissa in due istruzioni

Pseudoistruzioni	
Pseudo istruzione	**Istruzione**
move $t0,$t1	
	addi $t0,$zero,$t1
la $t0,etichetta	
	lui $at,etichetta$_{bit31-16}$
	ori $t0,$at,etichetta$_{bit15-0}$
bnez $t0,etichetta	
	bne $0,$zero,etichetta
rem $t0,$t1,$t2	
	div $t1,$t2
	mfhi $t0

Pseudo istruzioni

LW (Tipo I)

Questo formato, inoltre, realizza l'istruzione di caricamento di un operando in un registro (*load immediate*, li); cioè una **inizializzazione**. In altre parole si inserisce un valore in un registro senza esplicitare un indirizzo di memoria (o evitando una modalità più complessa).
L'istruzione di **caricamento di un valore immediato**, che ha una sintassi del tipo **li $reg,immediate**, ovvero la cancellazione del contenuto del registro *$reg* con il numero *immediate*, è riscritta dall'assemblatore sfruttando l'istruzione di addizione senza l'estensione del segno, ***addiu***

lw $t3,pippo #$t3←MEM(4097) dove pippo individua la locazione 4097

Istruzione di tipo I nel MIPS (inizializzazione) e sua traduzione in linguaggio macchina			
Istruzione in linguaggio asemblativo canonico			
lw	$t3,		pippo
Istruzione in linguaggio assemblativo nativo			
lui	$at	4097	
lw	$t3	0($at)	In $t3 contenuto della variabile pippo

Pseduolstruzioni

Moltiplicazione e Divisione

La moltiplicazione e la divisione nel MIPS nella loro forma di istruzioni native usano dei registri speciali che sono **hi**, che conserva i 32 bit più significativi del risultato di una moltiplicazione o il resto nel caso di una divisione; e **lo**, che memorizza i 32bit meno significativi del risultato di una moltiplicazione o il quoziente se si effettua una divisione
Il prodotto tra due operandi a 32bit restituisce un valore la cui lunghezza è, al più, 64bit.

Istruzioni di prodotto e divisione	
Pseudo istruzione	**Istruzione**
mult $t0,$t1	
	hi #memorizza i 32bit più significativi della moltiplicazione tra $t0 e $t1
	lo #memorizza i 32bit meno significativi della moltiplicazione tra $t0 e $t1
div $t0,$t1	
	hi #memorizza il resto della divisione tra $t0 e $t1
	lo #memorizza il quoziente della divisione tra $t0 e $t1

Pseduolstruzioni

Moltiplicazione e Divisione

Di norma il programmatore utilizza le pseudistruzioni di divisione **div** e di prodotto **mul** con operandi tra registri che si concretizzano con le istruzioni native (mult e div) e gli spostamenti che richiedono le istruzioni speciali mfhi e mflo

Istruzioni di prodotto e divisione	
Pseudo istruzione	**Istruzione**
div $t0,$t1,$t2	
	div $t1,$t2 #Riporta il resto della divisione nel registro *high* e il quoziente della divisione in *low*
	mflo $t0 #Sposta il quoziente nel registro destinazione della pseudo istruzione
mul $t0,$t1,$t2	
	mult $t1,$t2 **#Effettua il prodotto e mette i 32 bit significativi in *high* (che sono scartati) e gli altri in *low***
	mflo $t0 **#Sposta i 32 bit meno significati del prodotto nel registro destinazione della pseudoistruzione**

FINE