UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Instruction Set Architecture (ISA)

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

Trieste, 13 marzo 2025

RECAP

00000001000010010101000000100000

X + Y = Z

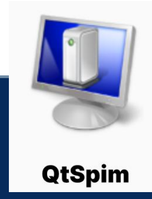
HARDWARE ⟺ **SOFTWARE**

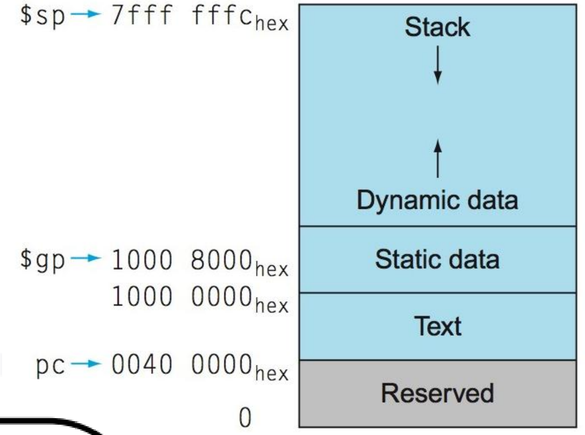
…a patto di star condividendo la stessa architettura e la relativa **Instruction Set Architecture (ISA)**

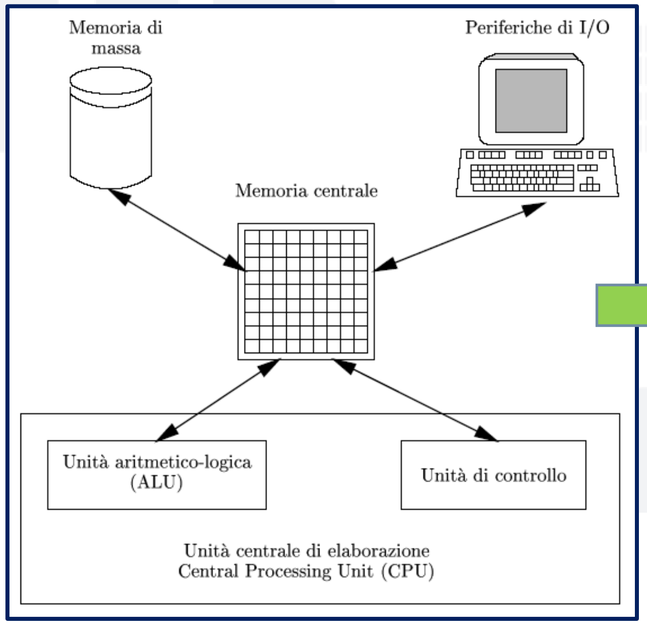
Architettura MIPS32

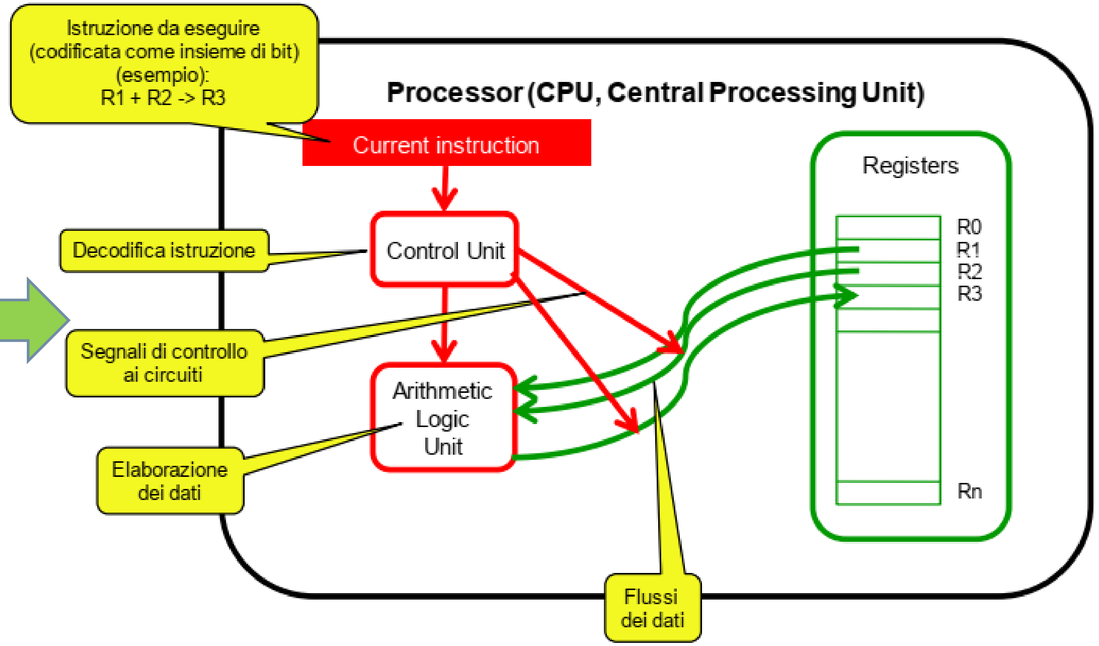
32 registri, 32 bit per ogni cella di memoria, ecc.

add $10, $8, $9 ➡ 00000001000010010101000000100000









UNIVERSITÀ DEGLI STUDI DI TRIESTE

REGOLE FONDAMENTALI DELL'ARCHITETTURA MIPS32 (RISC)

- 32 registri da 32 bit
- Istruzioni da 32 bit
- Manipolazioni di dati solo sui registri
- Trasferimento di dati tra memoria e registri
- Alterazione del flusso di controllo ("salti")

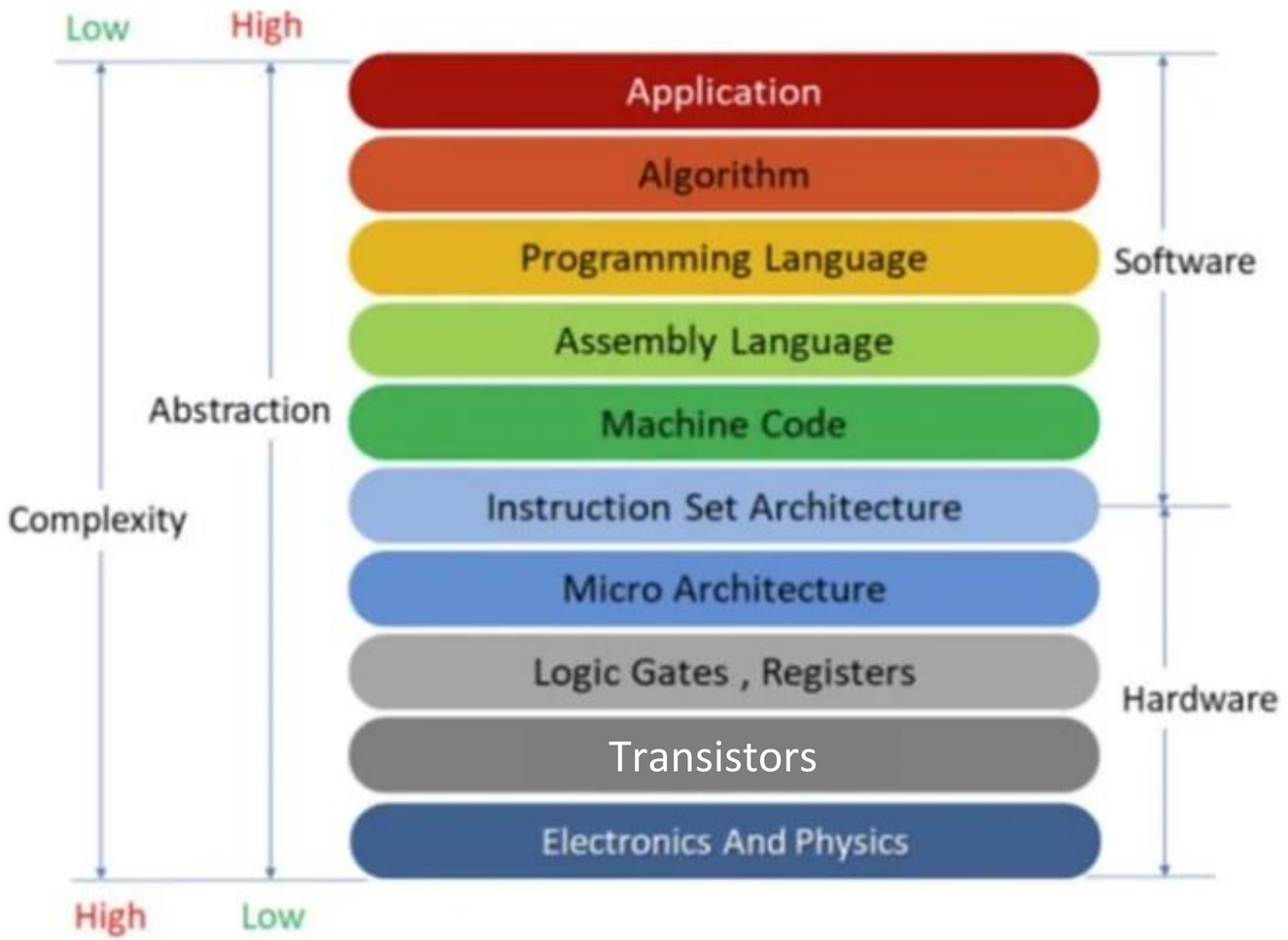
Problema architetturale:

- *come esprimere tutti i tipi di operazioni in 32 bit??* …mantenendo il più possibile omogenea la struttura delle istruzioni

Tre formati delle istruzioni: R-type, I-type, J-type

Attenzione: i tre formati non coincidono esattamente con le tre tipologie di operazioni

INSTRUCTION SET ARCHITECTURE



dipendenza

UNIVERSITÀ DEGLI STUDI DI TRIESTE

INSTRUCTION SET ARCHITECTURE

00000001000010010101000000100000

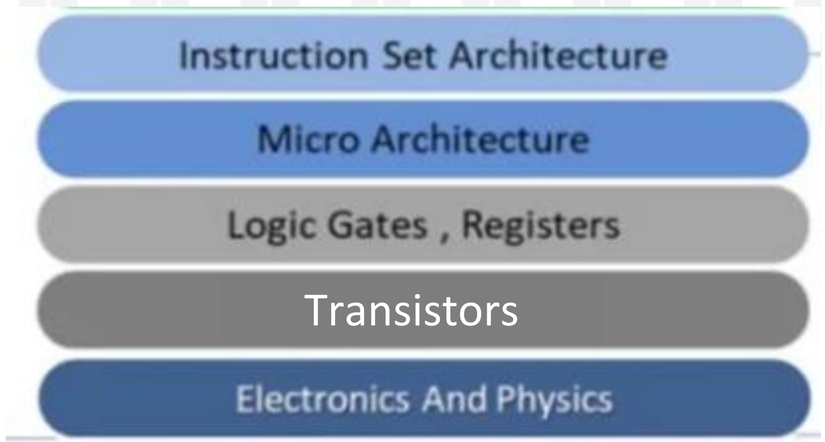
I primi 6 bit definiscono il cosiddetto OPCODE, ovvero il codice dell'operazione che va eseguita.

IL/I successivo/i *blocchi di bit* determinano i registri (indirizzo o valore contenuto) da cui prendere gli operandi.

Il *blocco di bit* seguente determina dove andrà messo il risultato (un altro registro).

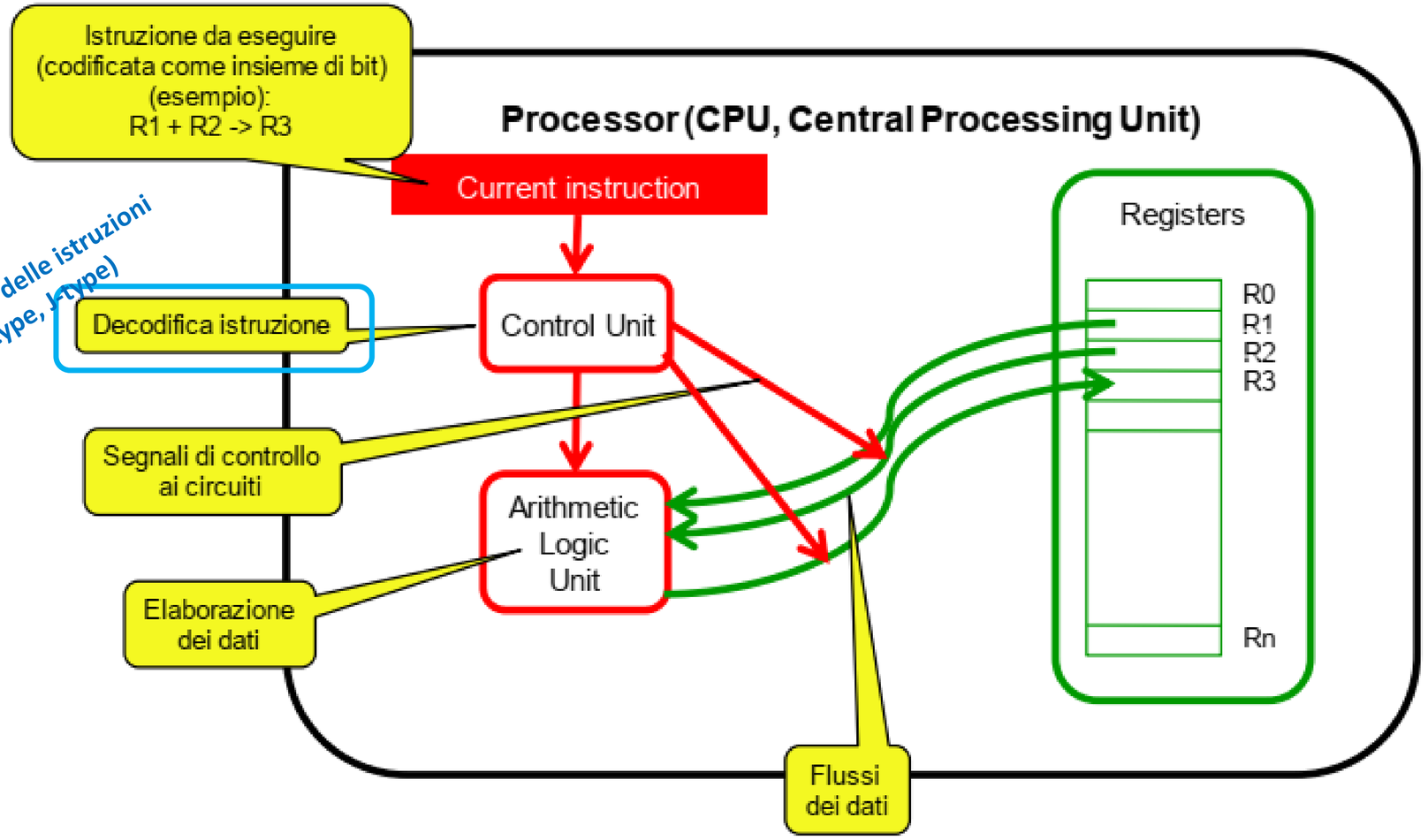
Poi ci sono altri bit che danno informazioni necessarie per implementare altre operazioni o varianti delle stesse.

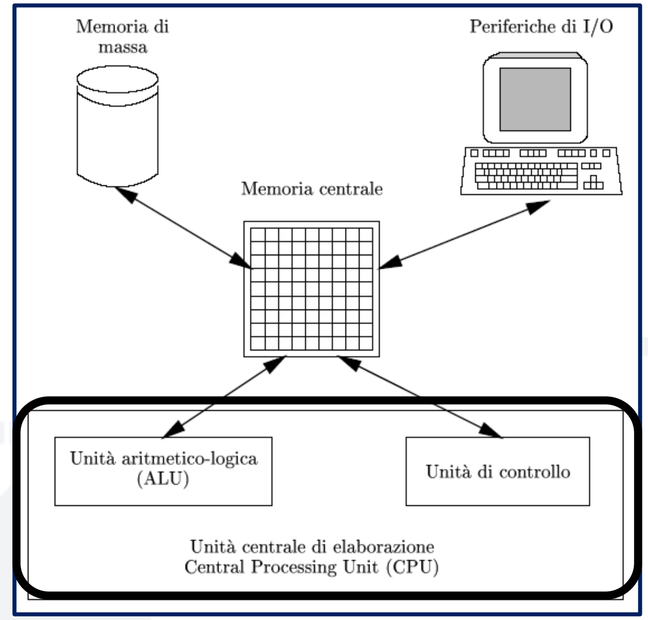
*Per esempio, ISA deve sapere che **ci sono solo 32 registri nel processore** e che le operazioni da poter fare sono solo con **un range di valori ottenibili con 32 bit**..*



ISTRUZIONE → CODIFICA → ESECUZIONE (SU MIPS32)

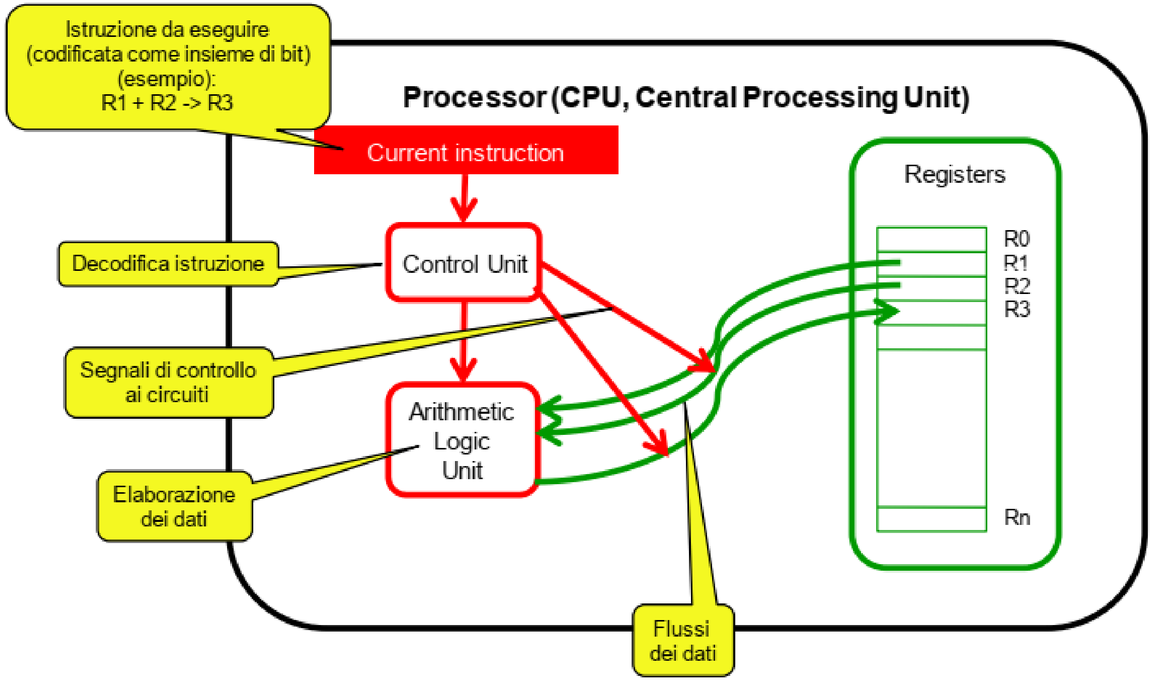
add $10, $8, $9 → 000000 01000 01001 01010 00000 100000



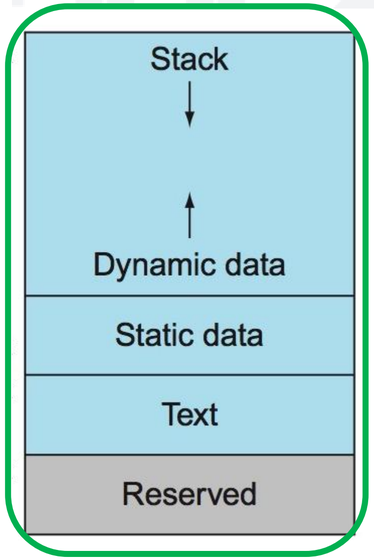


UNIVERSITÀ DEGLI STUDI DI TRIESTE

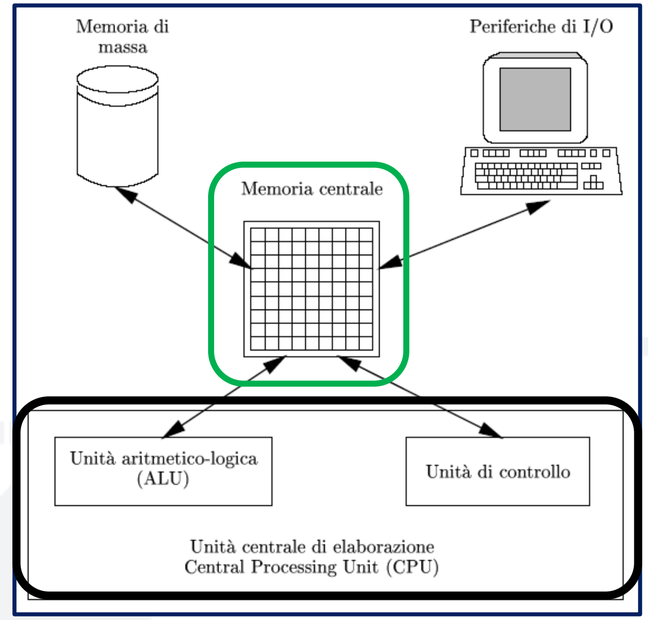
ISTRUZIONE → CODIFICA → ESECUZIONE (SU MIPS32)



Indirizzi codificati
a 32 bit



heap



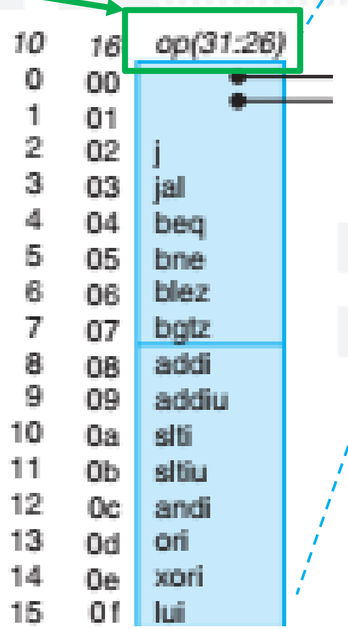
*Nota: **8 cifre HEX**, ovvero 32bit!*

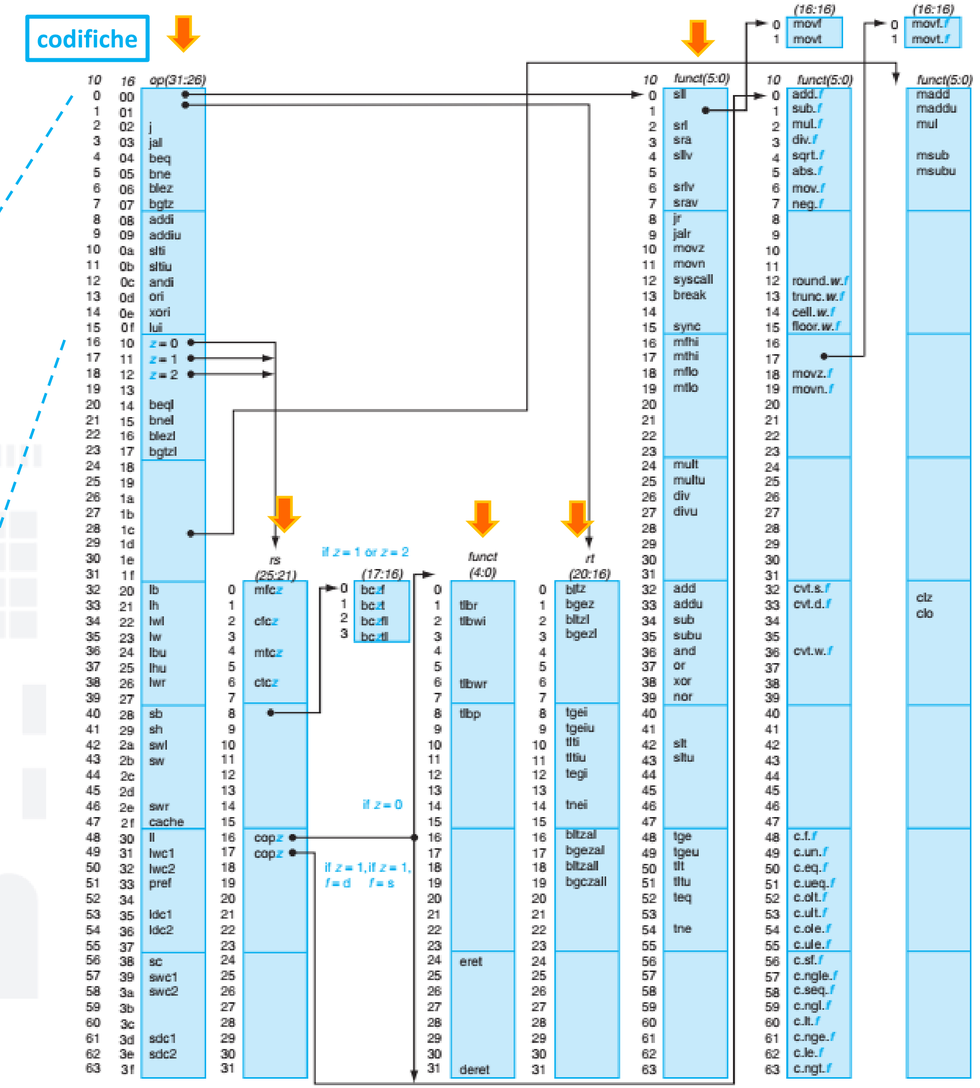
UNIVERSITÀ DEGLI STUDI DI TRIESTE

INSTRUCTION SET ARCHITECTURE: OPCODE PER MIPS32

In MIPS32, si possono realizzare «solo» le operazioni codificabili tramite questa **MAPPA DEGLI OPCODE**.

00000001000010010101000000100000





Appendice del libro di testo (pagina A-50)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Tre formati delle istruzioni: R-type, I-type, J-type

Attenzione: i tre formati non coincidono esattamente con le tre tipologie di operazioni

FORMATO R-TYPE

Per codificare operazioni aritmetico-logiche tra registri.

R-type = register-type

opcode	rs	rt	rd	shamt	funct

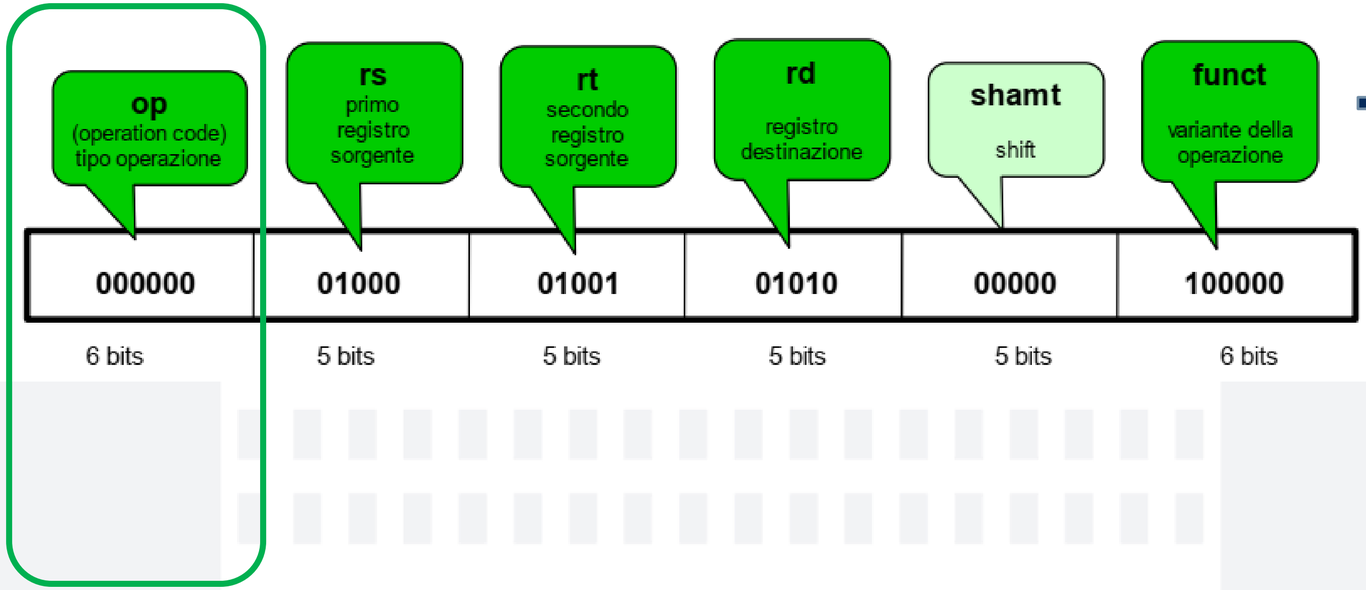
opcode = 000000 → indica che è un'**istruzione aritmetico-logica**.
rs = registro sorgente 1.
rt = registro sorgente 2.
rd = registro di destinazione.
shamt = valore di shift (solo per istruzioni di shift, altrimenti è 00000).
funct = specifica l'operazione esatta.

Operazioni aritmetiche con valori dai registri
Operazioni logiche
Operazioni di confronto (es. minore o uguale)

FORMATO R-TYPE: ESEMPIO

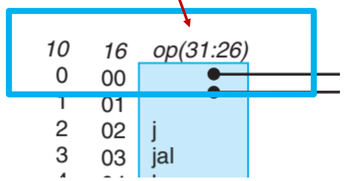
00000001000010010101000000100000

"somma i contenuti del registro 8 e del registro 9 e metti il risultato nel registro 10"



L'Unità di Controllo "sa" come interpretare i singoli campi

Nota: bit dell'opcde sono dal 26 al 31 (MSB) della sequenza. Il primo bit ha indice 0.



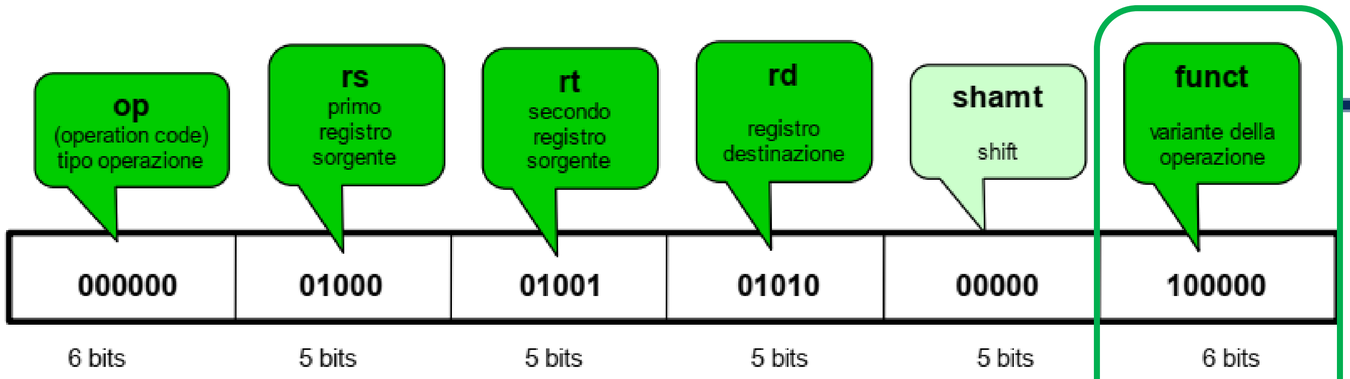
Appendice A p.50

opcode = 000000 → indica che è un'**istruzione aritmetico-logica**

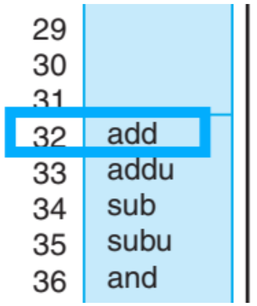
FORMATO R-TYPE: ESEMPIO

00000001000010010101000000100000

"somma i contenuti del registro 8 e del registro 9 e metti il risultato nel registro 10"



L'Unità di Controllo "sa" come interpretare i singoli campi



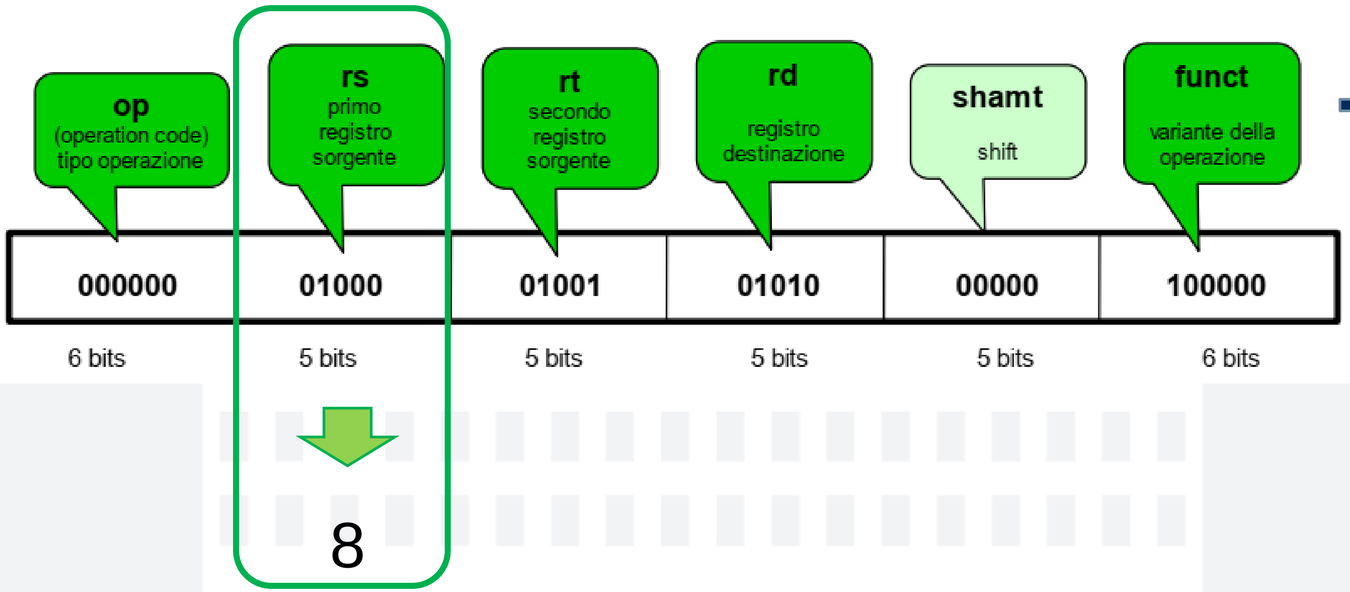
opcode = 000000 → indica che è un'**istruzione aritmetico-logica**

Leggere e decodificare il campo **funct** per determinare l'operazione esatta da eseguire

FORMATO R-TYPE: ESEMPIO

00000001000010010101000000100000

“somma i contenuti del registro 8 e del registro 9 e metti il risultato nel registro 10”



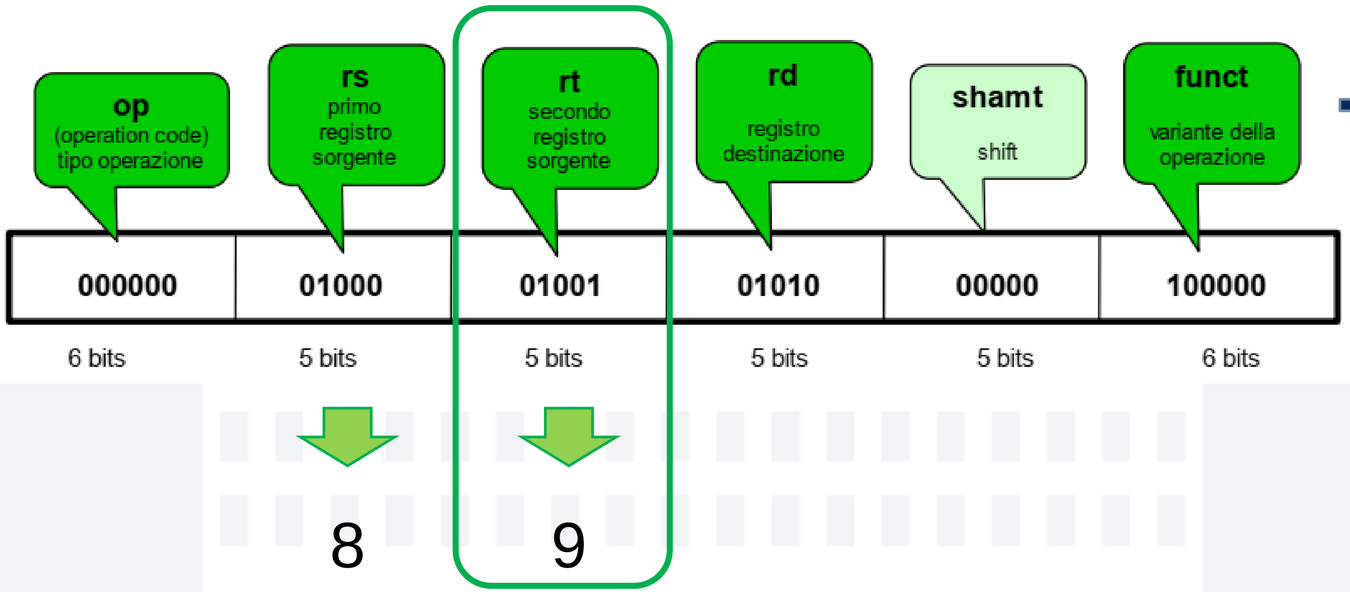
L’Unità di Controllo “sa” come interpretare i singoli campi

UNIVERSITÀ DEGLI STUDI DI TRIESTE

FORMATO R-TYPE: ESEMPIO

00000001000**01001**01010000000100000

“somma i contenuti del registro 8 e del registro 9 e metti il risultato nel registro 10”



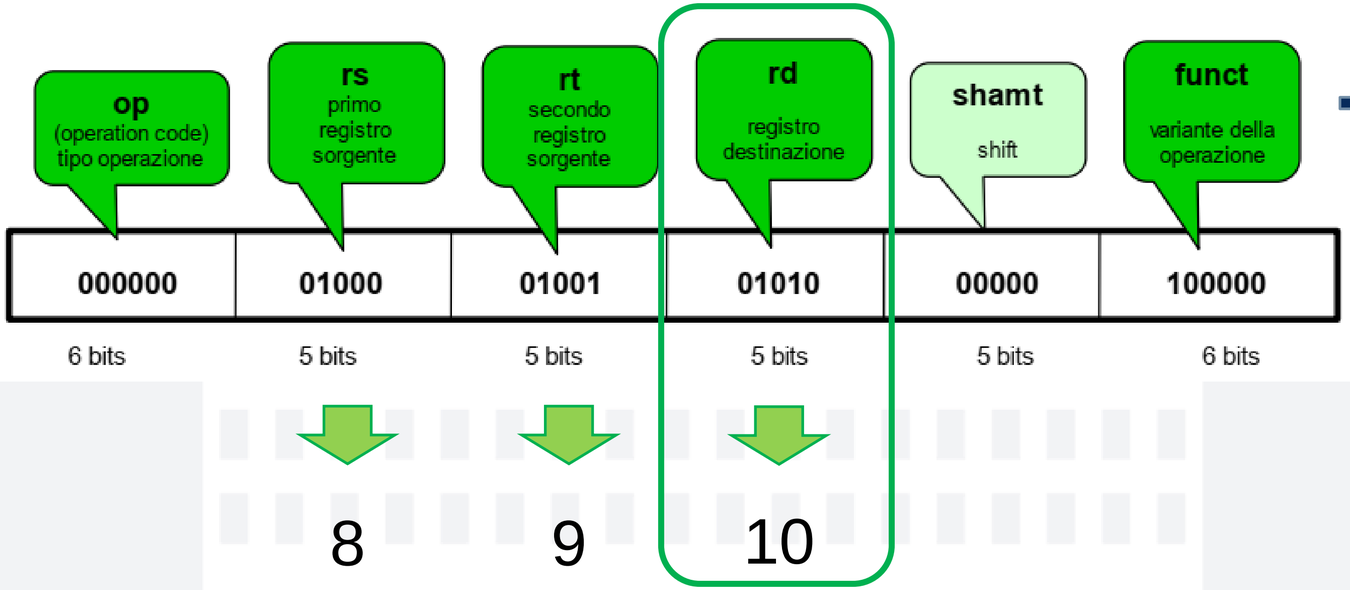
L’Unità di Controllo “sa” come interpretare i singoli campi

UNIVERSITÀ DEGLI STUDI DI TRIESTE

FORMATO R-TYPE: ESEMPIO

00000001000010010101000000100000

“somma i contenuti del registro 8 e del registro 9 e metti il risultato nel registro 10”



L’Unità di Controllo “sa” come interpretare i singoli campi

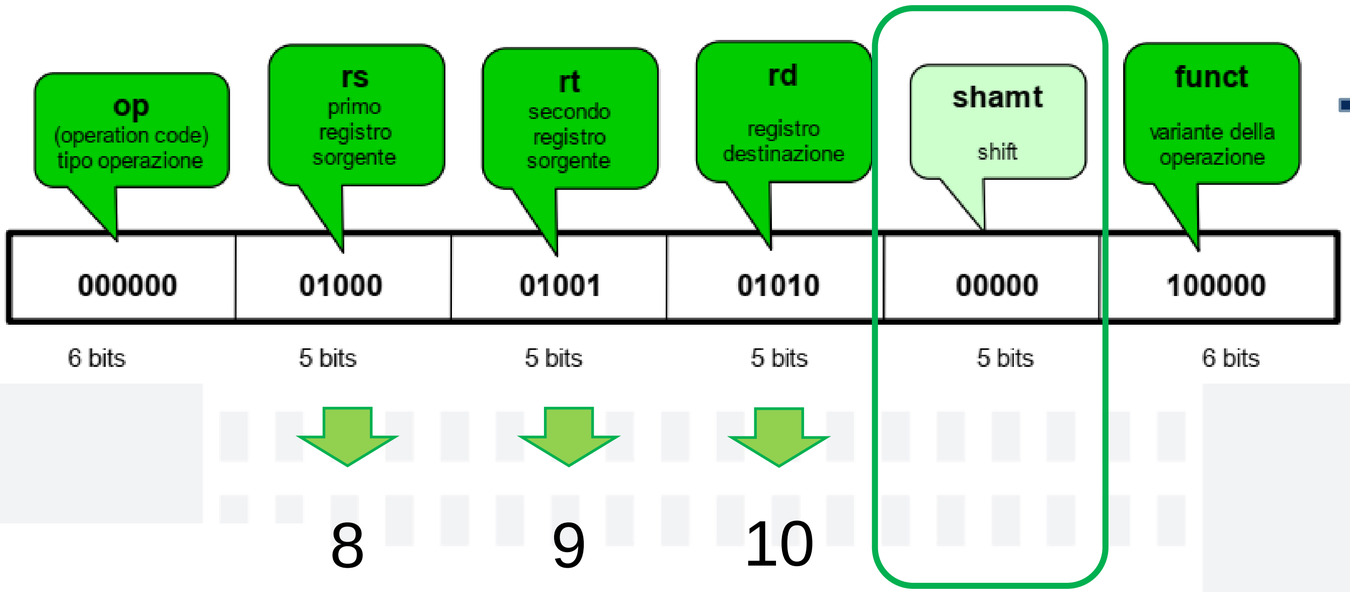
UNIVERSITÀ DEGLI STUDI DI TRIESTE

FORMATO R-TYPE: ESEMPIO

Nota. L'istruzione potrebbe essere anche scritta, in modo più compatto, in esadecimale: 0x1095020

00000001000010010101000000100000

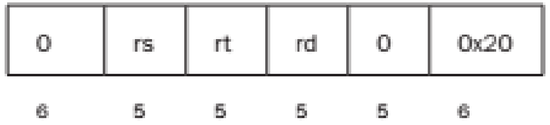
"somma i contenuti del registro 8 e del registro 9 e metti il risultato nel registro 10"



L'Unità di Controllo "sa" come interpretare i singoli campi

Addition (with overflow)

add rd, rs, rt



Appendice A p.51

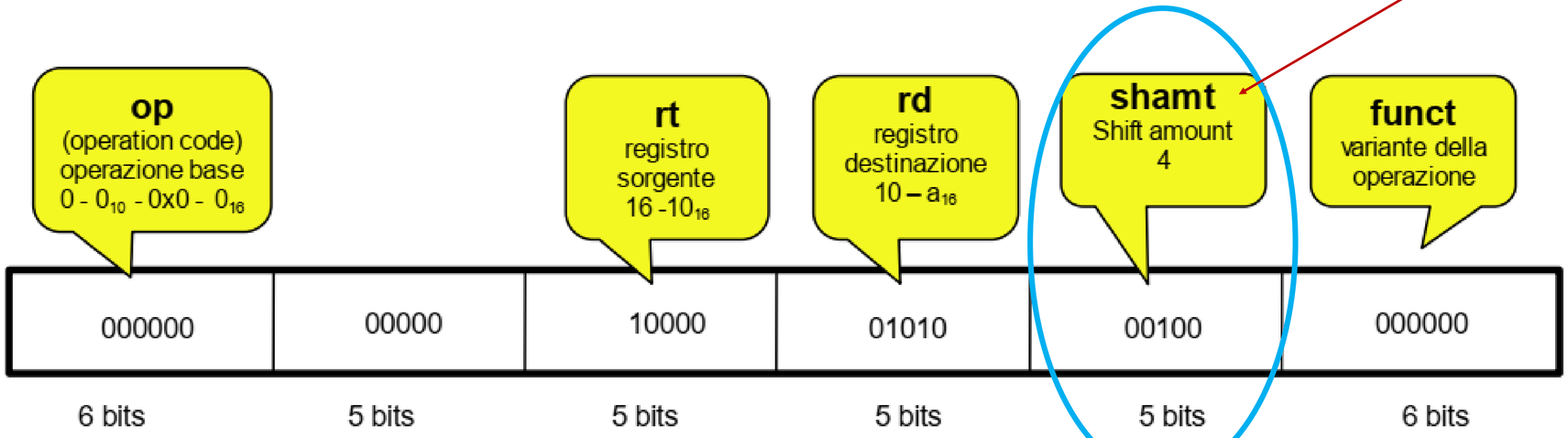
add $10, $8, $9

UNIVERSITÀ DEGLI STUDI DI TRIESTE

FORMATO R-TYPE: SHIFT LEFT

Shift di 4 bit a sinistra il contenuto del registro 16 e metti il risultato nel registro 10.

Shamt: shift amount



Shift left logical

`sll rd, rt, shamt`

0	rs	rt	rd	shamt	0
6	5	5	5	5	6

Appendice A p.55

Shift Left Logical

```
sll $10, $16, 4
```

Ci torneremo la prossima settimana (v. lezione 20/03)

FORMATO J-TYPE

Per codificare operazioni di salto (assoluto) a indirizzi specifici

J-type = jump-type

opcode	Target address

opcode (6 bit) → Indica che è istruzione di salto

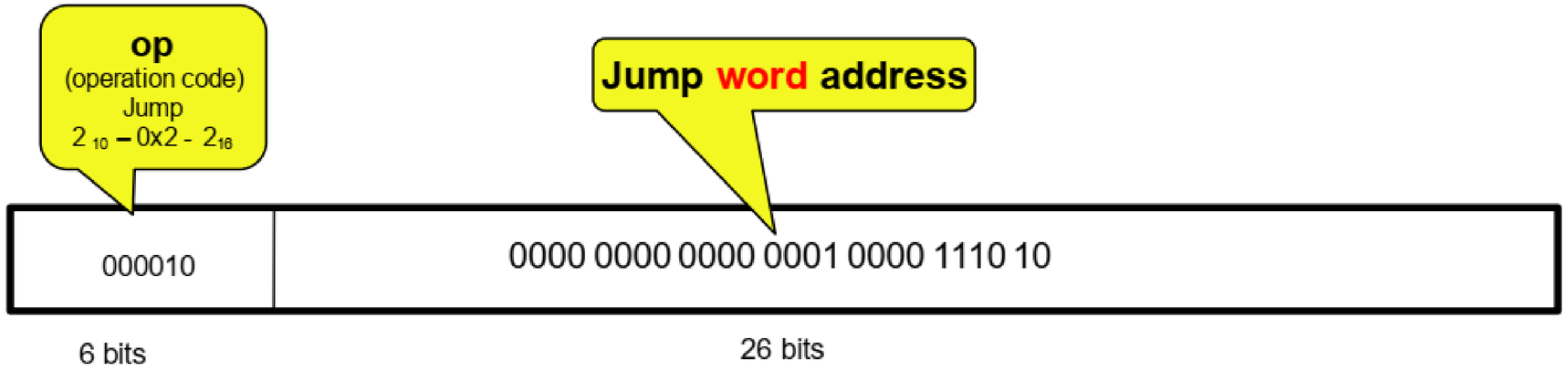
Target address (26 bit) → Specifica l'indirizzo di destinazione

Salti assoluti.

Ci torneremo la prossima settimana (v. lezione 20/03)

FORMATO J-TYPE: ESEMPIO

00001000000000000000010000111010



jump

Jump

j target	2	target
	6	26

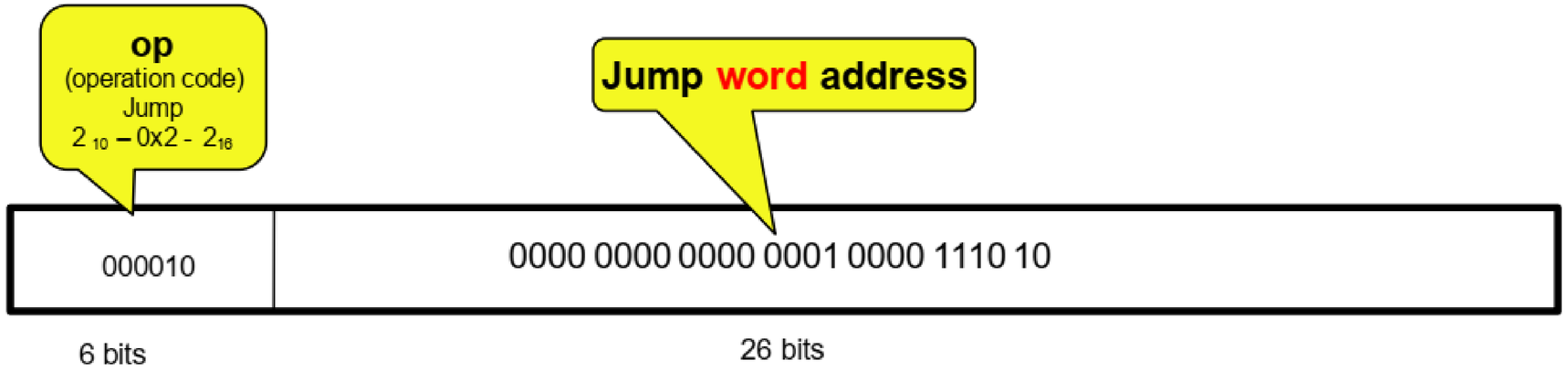
Unconditionally jump to the instruction at target.

L'**indirizzo target** in un'istruzione J-type non è l'indirizzo completo, ma solo i **26 bit più significativi**.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ci torneremo la prossima settimana (v. lezione 20/03)

FORMATO J-TYPE: ESEMPIO



L'**indirizzo target** in un'istruzione J-type non è l'indirizzo completo, ma solo i **26 bit più significativi**.

Per ottenere l'indirizzo effettivo bisogna **shiftare il valore a sinistra di 2 bit** (perché gli indirizzi MIPS sono allineati a 4 byte).

FORMATO I-TYPE

Per codificare operazioni con costanti o accesso alla memoria.

I-type = immediate*-type
***costante**

opcode	rs	rt	Immediate (offset)

opcode (6 bit) → Indica il tipo di istruzione.

rs (5 bit) → Registro sorgente.

rt (5 bit) → Registro di destinazione.

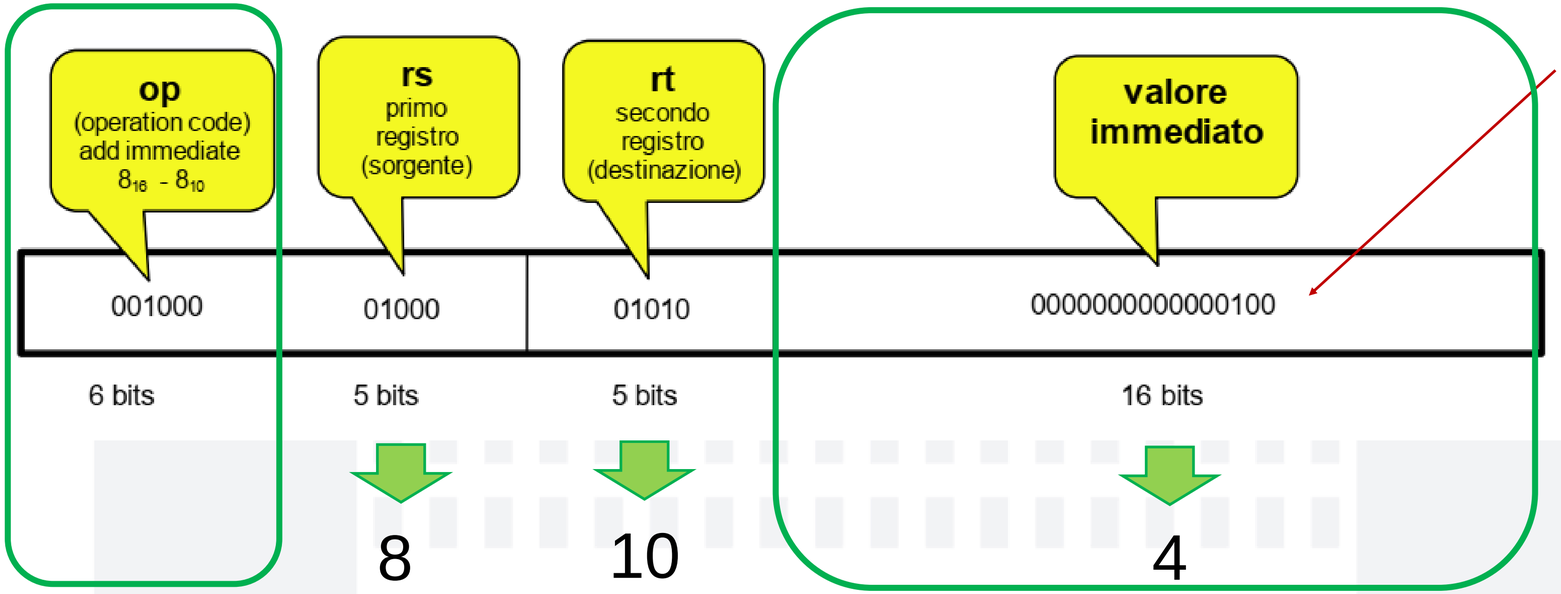
immediate (16 bit) → Valore immediato (costante o offset).

Operazioni aritmetiche con **valori immediati**.
Accesso alla memoria (load/store).
Salti condizionati.

FORMATO I-TYPE: ESEMPIO

001000010000101000000000000000100

“somma il valore 4 al contenuto del registro 8 e metti il risultato nel registro 10”



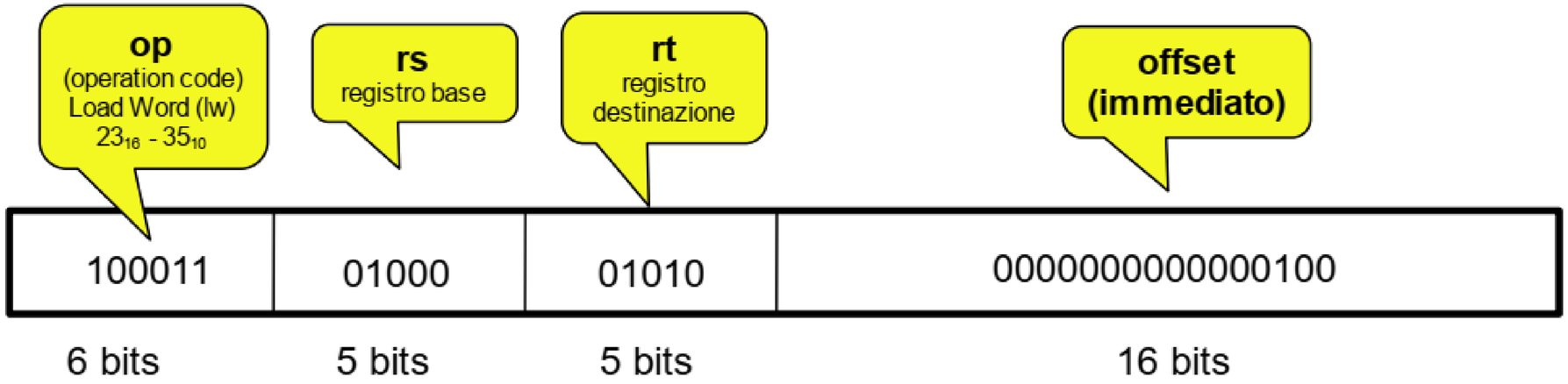
Nota. Qual è il **range di valori immediati** che si può esprimere con 16 bit in complemento a 2?
(-32768 <= valore <= 32767).

addi $10, $8, 4

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ci torneremo la prossima settimana (v. lezione 20/03)

FORMATO I-TYPE: LOAD WORD



Carica nel registro 10 il contenuto della parola (32 bit) che è **all'indirizzo di memoria** ottenuto come somma del contenuto del registro 8 e dell'offset immediato 4

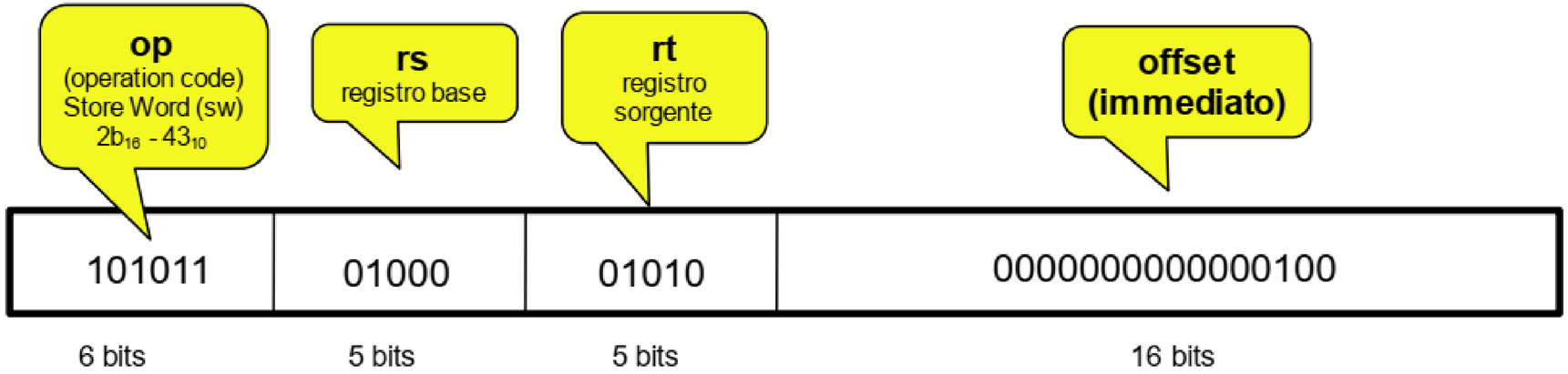
```
lw $10, 4($8)
```

Indirizzo di memoria di PROVENIENZA!

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ci torneremo la prossima settimana (v. lezione 20/03)

FORMATO I-TYPE: STORE WORD



Memorizza il contenuto del registro 10 (32 bit) all'indirizzo di memoria ottenuto come somma del contenuto del registro 8 e dell'offset immediato 4

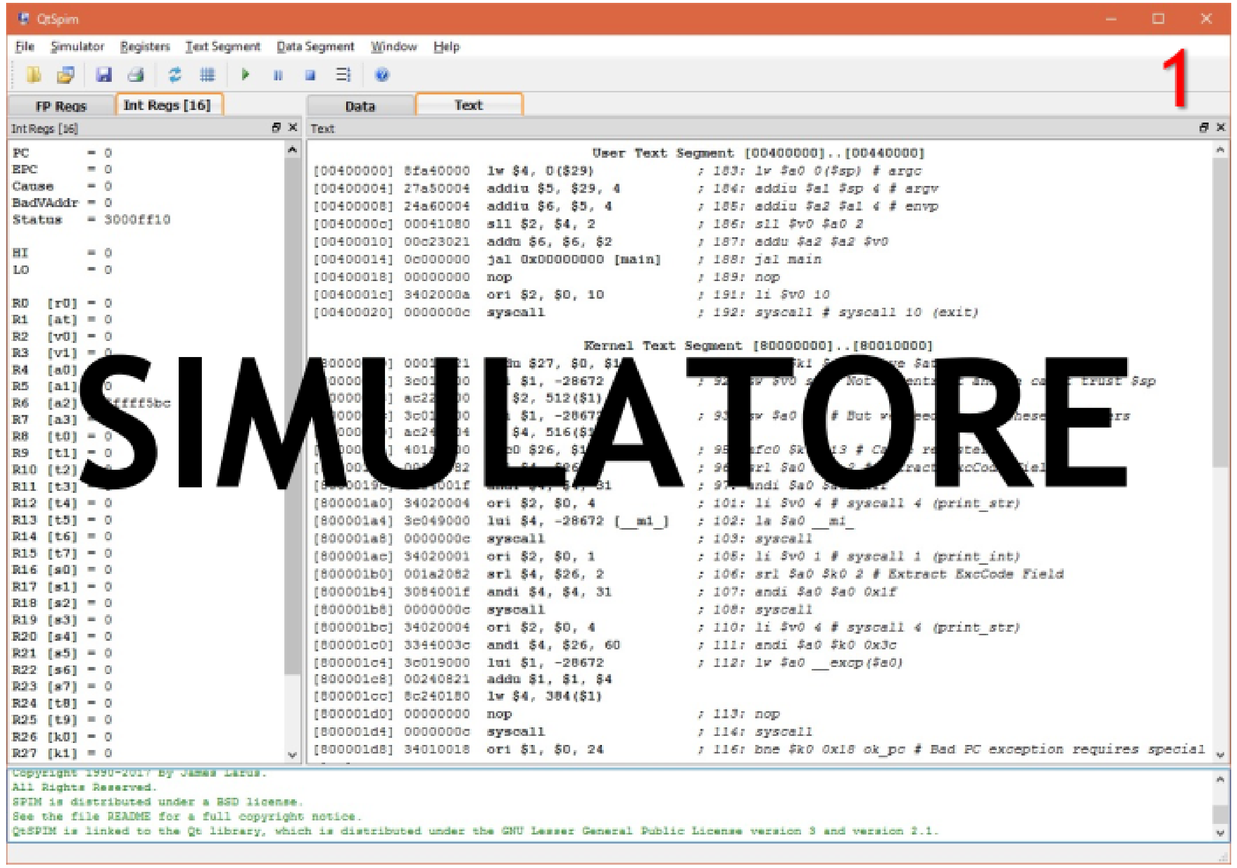
```
sw $10, 4($8)
```

Indirizzo di memoria di DESTINAZIONE!

QtSpim

https://spimsimulator.sourceforge.net/

L'interfaccia del simulatore QtSpim è composta da due parti principali



INTERFACCIA DEL SIMULATORE QtSpim

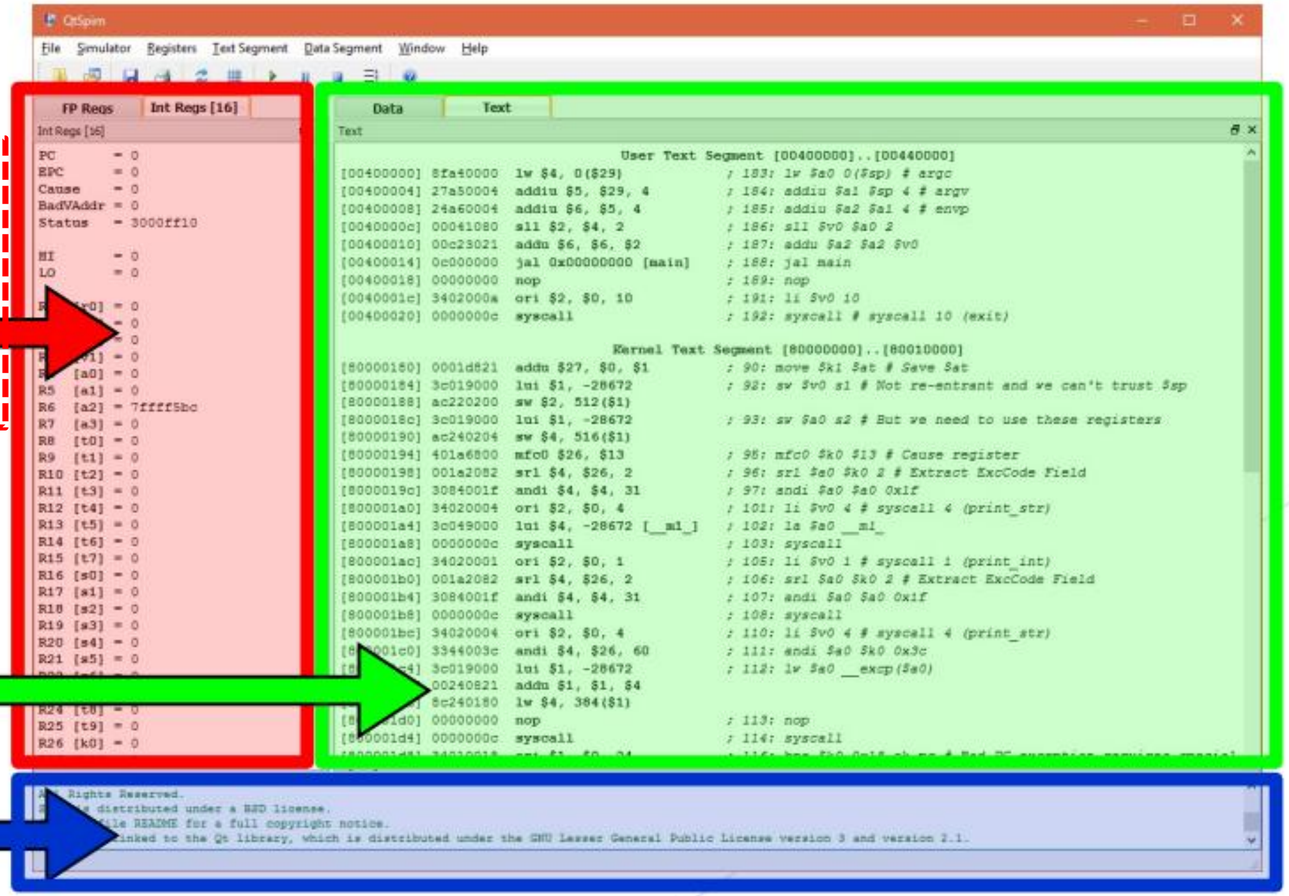
Abbiamo a disposizione **32 registri INTERI e 32 registri FLOATING POINT** (codifica IEEE 754)

LA FINESTRA PRINCIPALE È SUDDIVISA IN TRE PARTI

IL PANNELLO SULLA **SINISTRA** VISUALIZZA I REGISTRI FLOATING POINT oppure I REGISTRI INTERI, A SECONDA DELLA SCHEDA IN PRIMO PIANO

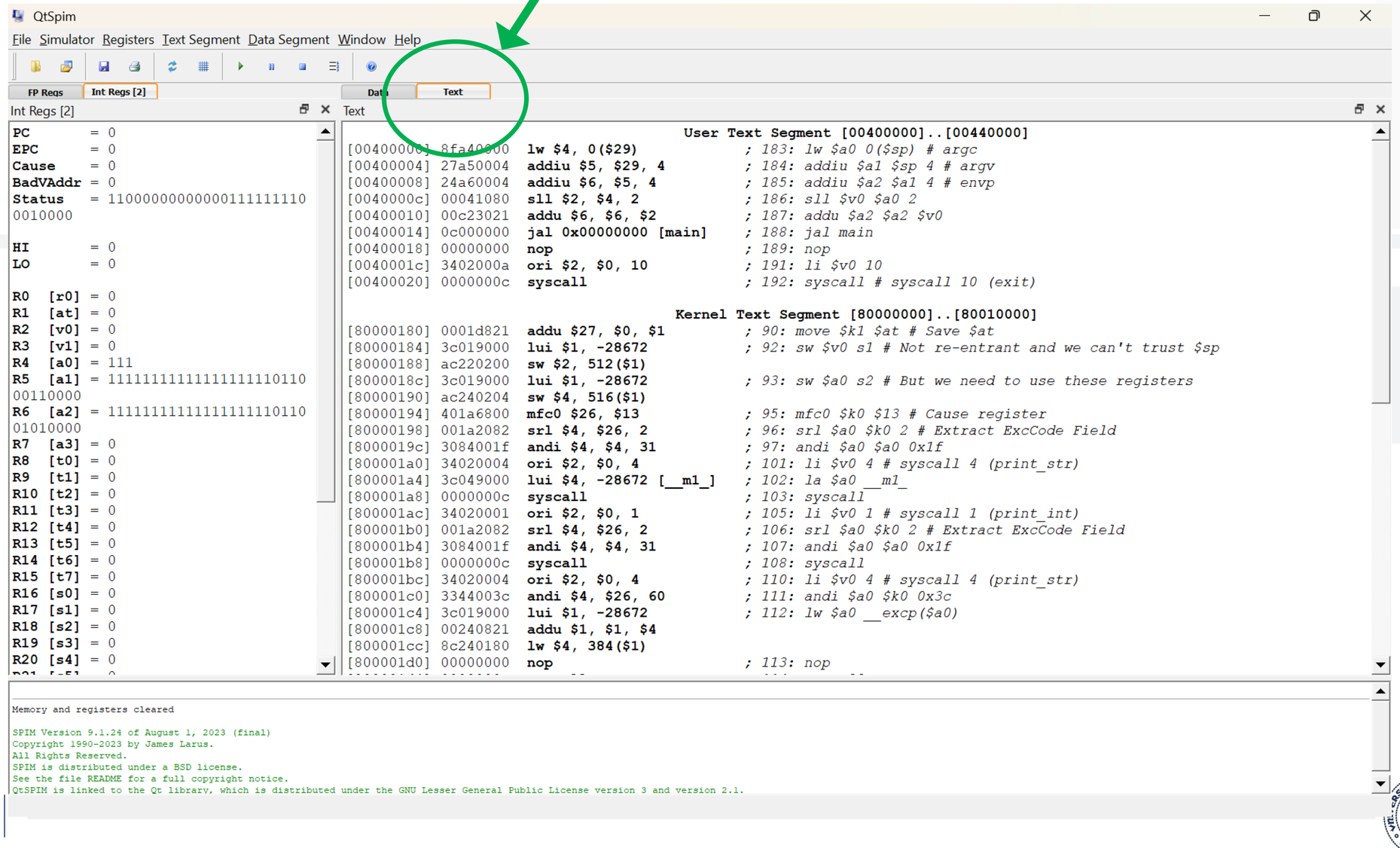
IL PANNELLO SULLA **DESTRA** VISUALIZZA IL TEXT SEGMENT (CONTIENE LE ISTRUZIONI MACCHINA) ED IL DATA SEGMENT (CONTIENE I DATI IN MEMORIA), A SECONDA DELLA SCHEDA IN PRIMO PIANO

IL PANNELLO INFERIORE CONTIENE I MESSAGGI DEL SIMULATORE

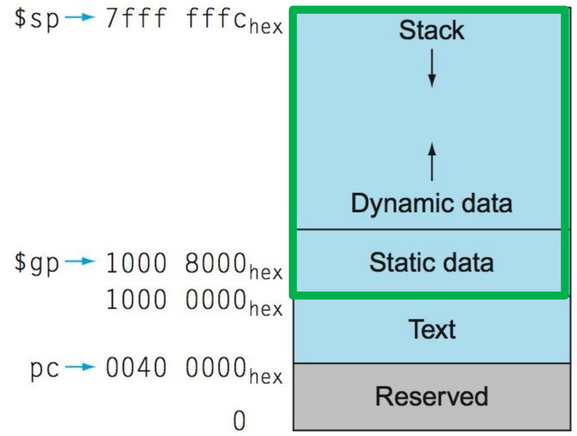


QtSpim

In questa sezione della memoria verranno memorizzate le **istruzioni associate al programma dell'utente** (che si carica, all'avvio di QtSpim, per poterlo eseguire)



QtSpim




```
Data | Text
Text
                                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)            ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4         ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4          ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2            ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2          ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]    ; 188: jal main
[00400018] 00000000  nop                      ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10           ; 191: li $v0 10
[00400020] 0000000c  syscall                  ; 192: syscall # syscall 10 (exit)

                                    Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1         ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672           ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672           ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13            ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2           ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31          ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4            ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]   ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                  ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1            ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2           ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31          ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                  ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4            ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60         ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672           ; 112: lw $a0 __excp($a0)
[800001c8] 00240821  addu $1, $1, $4
[800001cc] 8c240180  lw $4, 384($1)
[800001d0] 00000000  nop                      ; 113: nop
```

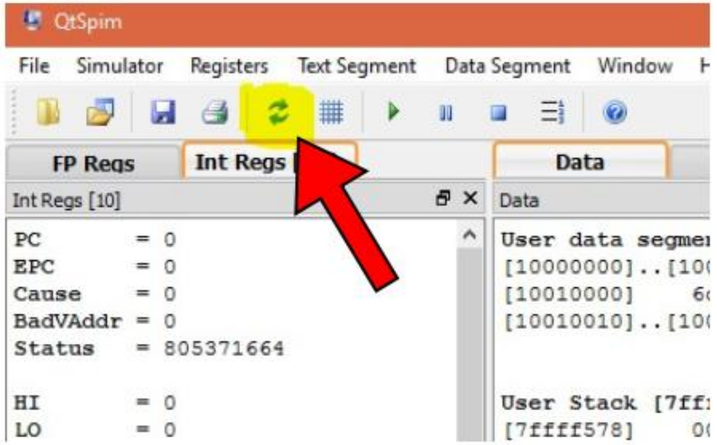
```
Data | Text
Data
User data segment [10000000]..[10040000]
[10000000]..[1003ffff]  00000000

User Stack [7ffff62c]..[80000000]
[7ffff62c]  00000007                                  . . . .
[7ffff630]  7ffff740  7ffff730  7ffff72e  7ffff727    @ . . . 0 . . . . . . . ' . . .
[7ffff640]  7ffff71e  7ffff71a  7ffff701  00000000    . . . . . . . . . . . . . . . .
[7ffff650]  7fffffe1  7fffffba  7fffff89  7fffff4d    . . . . . . . . . . . . . M . . .
[7ffff660]  7fffff1c  7ffffeff  7ffffedb  7ffffea9    . . . . . . . . . . . . . . . .
[7ffff670]  7ffffe9d  7ffffe6c  7ffffe44  7ffffe37    . . . . l . . . D . . . 7 . . .
[7ffff680]  7ffffe21  7ffffc75  7ffffc4b  7ffffc2d    ! . . . u . . . K . . . - . . .
[7ffff690]  7ffffc15  7ffffbf4  7ffffbe6  7ffffa66    . . . . . . . . . . . . . f . . .
[7ffff6a0]  7ffffa28  7ffffa0b  7ffff9c2  7ffff9b0    ( . . . . . . . . . . . . . . . .
[7ffff6b0]  7ffff998  7ffff97d  7ffff95f  7ffff936    . . . . } . . . _ . . . 6 . . .
[7ffff6c0]  7ffff918  7ffff8ad  7ffff896  7ffff882    . . . . . . . . . . . . . . . .
[7ffff6d0]  7ffff873  7ffff85d  7ffff836  7ffff810    s . . . ] . . . 6 . . . . . . .
[7ffff6e0]  7ffff7f5  7ffff7cb  7ffff7bc  7ffff7a1    . . . . . . . . . . . . . . . .
[7ffff6f0]  7ffff78f  7ffff77b  00000000  00000000    . . . . { . . . . . . . . . . .
[7ffff700]  6e696c00  2f72656b  6e61656d  62696c5f    . l i n k e r / m e a n _ l i b
[7ffff710]  726f775f  73612e64  6f63006d  7341006e    _ w o r d . a s m . c o n . A s
[7ffff720]  626d6573  5100796c  69705374  002d006d    s e m b l y . Q t S p i m . - .
[7ffff730]  33323032  3230322d  614c2f34  00342362    2 0 2 3 - 2 0 2 4 / L a b # 4 .
[7ffff740]  552f3a43  73726573  7569672f  4f2f696c    C : / U s e r s / g i u l i / O
[7ffff750]  7244656e  2f657669  75636f44  746e656d    n e D r i v e / D o c u m e n t
[7ffff760]  49442f69  54544144  2f414349  6d696e55    i / D I D A T T I C A / U n i m
[7ffff770]  412f6269  2f686372  5a006465  455f5345    i b / A r c h / e d . Z E S _ E
[7ffff780]  4c42414e  59535f45  4e414d53  7700313d    N A B L E _ S Y S M A N = 1 . w
[7ffff790]  69646e69  3a433d72  4e49575c  53574f44    i n d i r = C : \ W I N D O W S
[7ffff7a0]  45535500  4f525052  454c4946  5c3a433d    . U S E R P R O F I L E = C : \
[7ffff7b0]  72657355  69675c73  00696c75  52455355    U s e r s \ g i u l i . U S E R
[7ffff7c0]  454d414e  7569673d  5500696c  44524553    N A M E = g i u l i . U S E R D
[7ffff7d0]  49414d4f  4f525f4e  4e494d41  4f525047    O M A I N _ R O A M I N G P R O
[7ffff7e0]  454c4946  5549473d  5f41494c  4f4e454c    F I L E = G I U L I A _ L E N O
[7ffff7f0]  31584f56  45535500  4d4f4452  3d4e4941    V O X 1 _ U S E R D O M A I N =
```

In questa porzione di memoria verranno memorizzate i **dati dell'utente** (caricati in memoria centrale), **le variabili temporanee** che servono durante l'esecuzione di **procedure e altro (si faccia riferimento alle prossime lezioni)**.

QtSpim

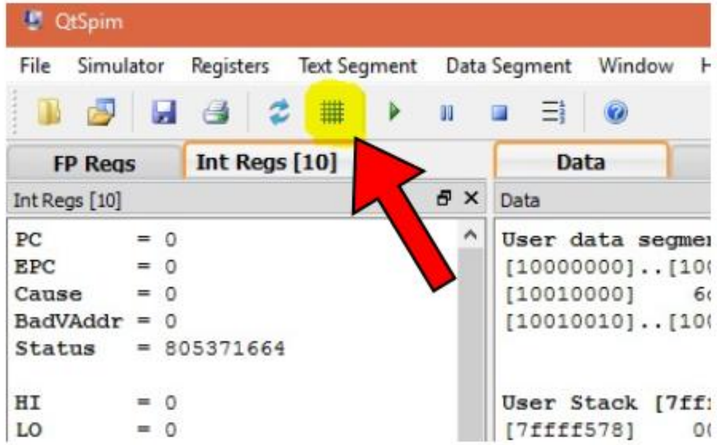
QtSpim

PULSANTI UTILI PER QUESTA ESERCITAZIONE



PULSANTE «CLEAR REGISTER»

CANCELLA IL CONTENUTO DEI REGISTRI, MA NON LA MEMORIA (AREA DATI)



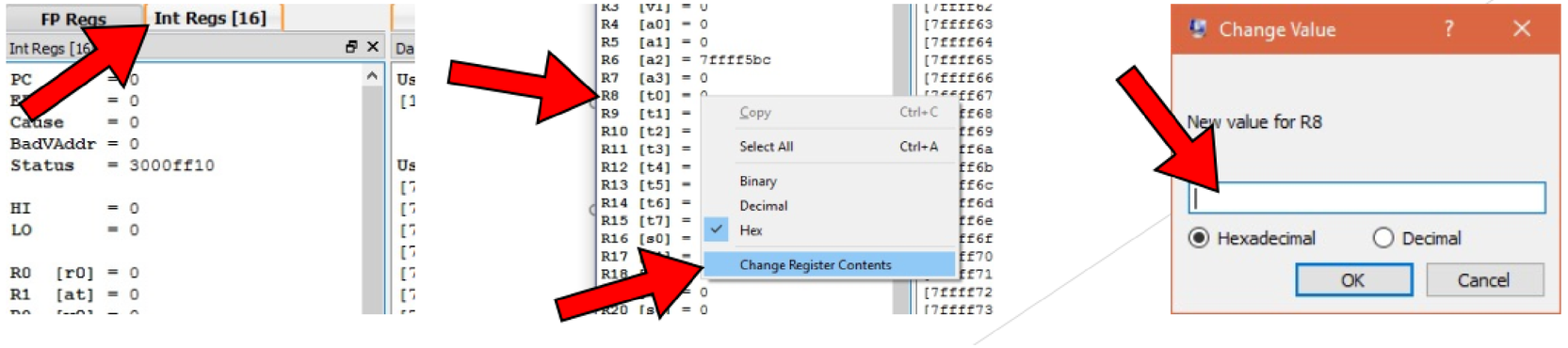
PULSANTE «REINITIALIZE SIMULATOR»

REIMPOSTA LA MEMORIA (AREA DATI), E I REGISTRI

QtSpim

COME CAMBIARE IL VALORE DI UN REGISTRO (e.g. R8)?

- SELEZIONARE LA SCHEDA DEI REGISTRI DEGLI INTERI «INT REGS [16]» ED INDIVIDUARE IL REGISTRO R8 (Register8)
- FARE CLICK CON IL PULSANTE DESTRO IN CORRISPONDENZA DEL REGISTRO R8
- SELEZIONARE «CHANGE REGISTER CONTENTS»
- INSERIRE IL VALORE (IN FORMATO HEX oppure DEC)



QtSpim

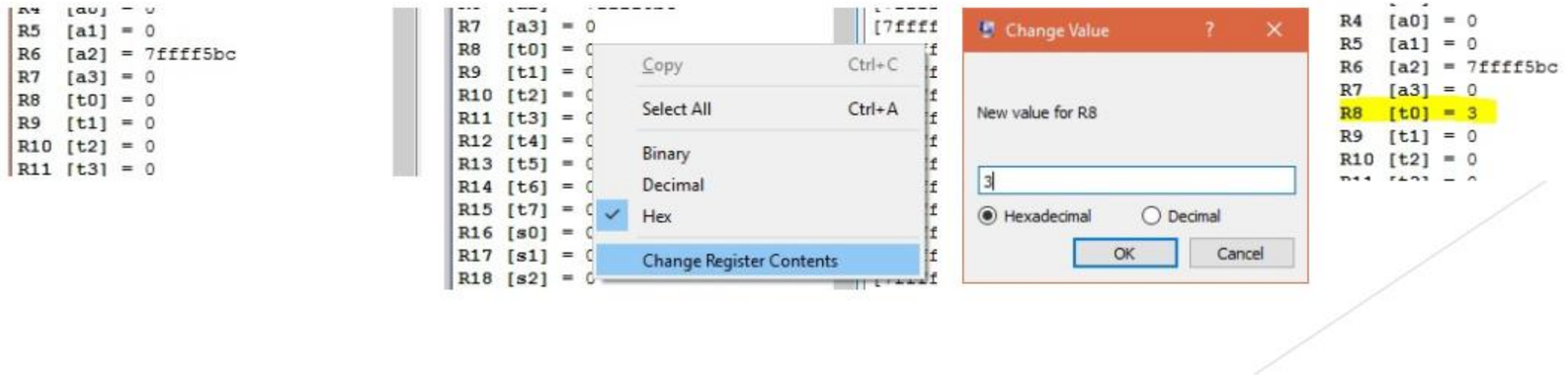
ESEMPIO: IMPOSTARE R8 CON IL VALORE 0x3

1
INDIVIDUO R8

2
CLICK DESTRO +
«CHANGE REGISTER...»

3
SCRIVO IL NUMERO

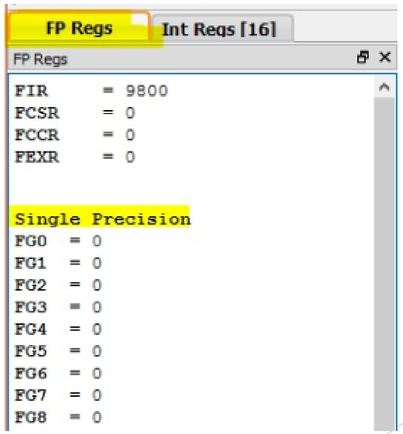
4
RISULTATO



QtSpim

REGISTRI FLOATING POINT SINGLE PRECISION (VIRGOLA MOBILE, SINGOLA PRECISIONE)

- SI VISUALIZZANO ABILITANDO LA SCHEDA «FP Regs»
- SI MODIFICANO ALLO STESSO MODO DEI REGISTRI INTERI
- LA RAPPRESENTAZIONE DEI NUMERI SEGUE LO **STANDARD IEEE754**



N.B. il contenuto del registro è visualizzato a partire dalla prima cifra diversa da 0

Materiale per la lezione

- Appendix A (da pagina 49)

Prossima lezione: 14 ***marzo, h.11:00, aula 5B***