UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Instruction Set Architecture (ISA)

Prof.ssa Giulia Cisotto

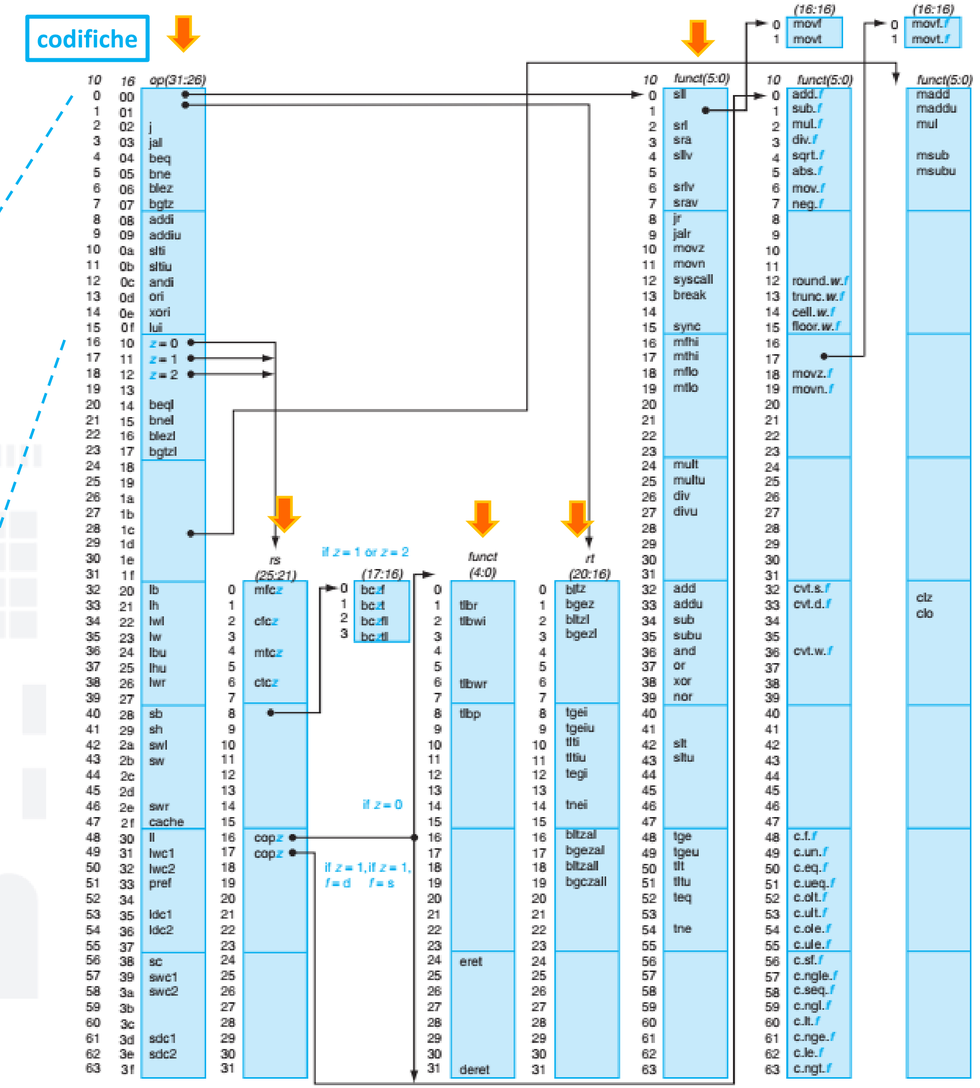
giulia.cisotto@units.it

Trieste, 10 marzo 2026

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)

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.

Esempio:

00000001000010010101000000100000

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

Addition (with overflow)

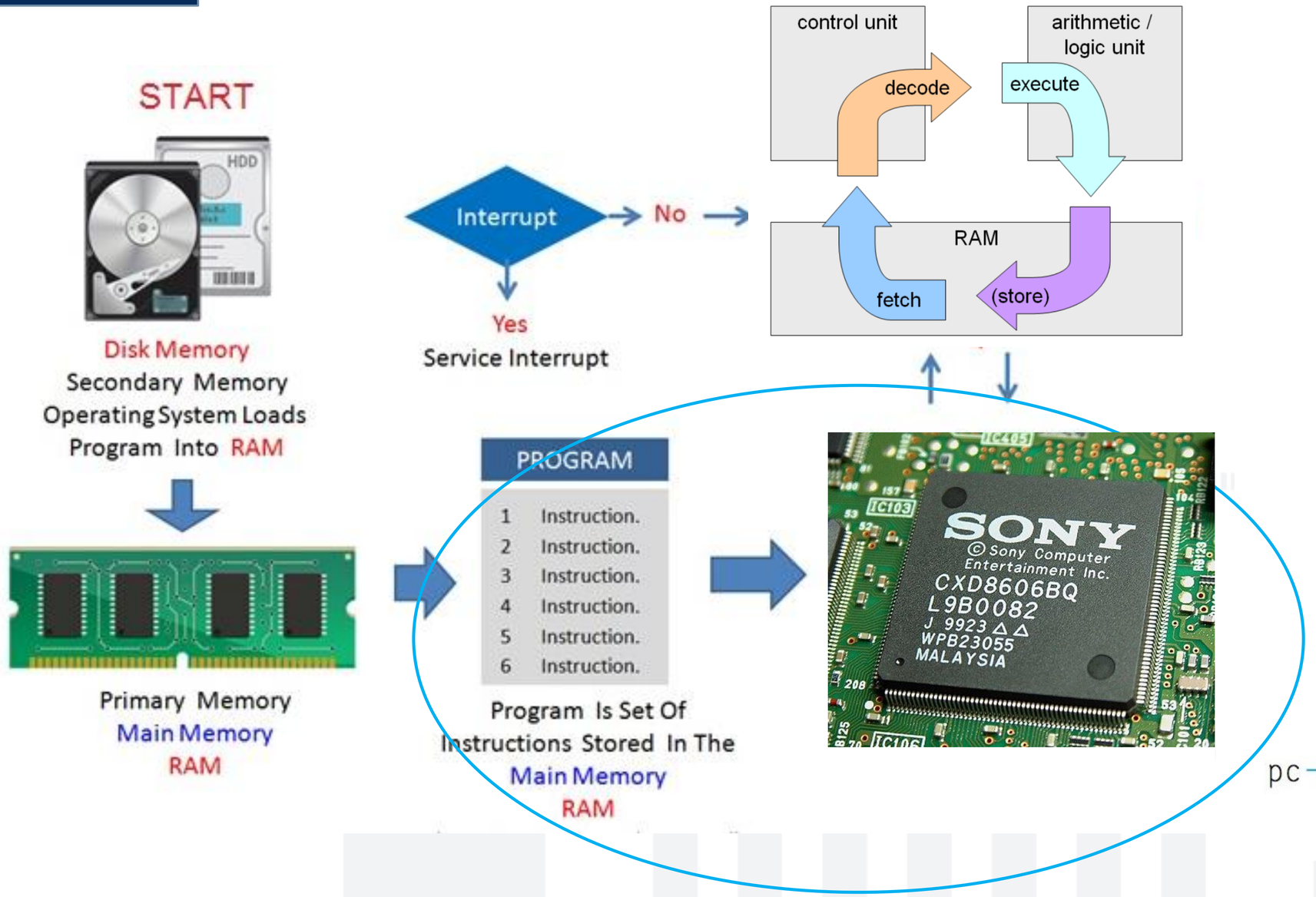
add rd, rs, rt

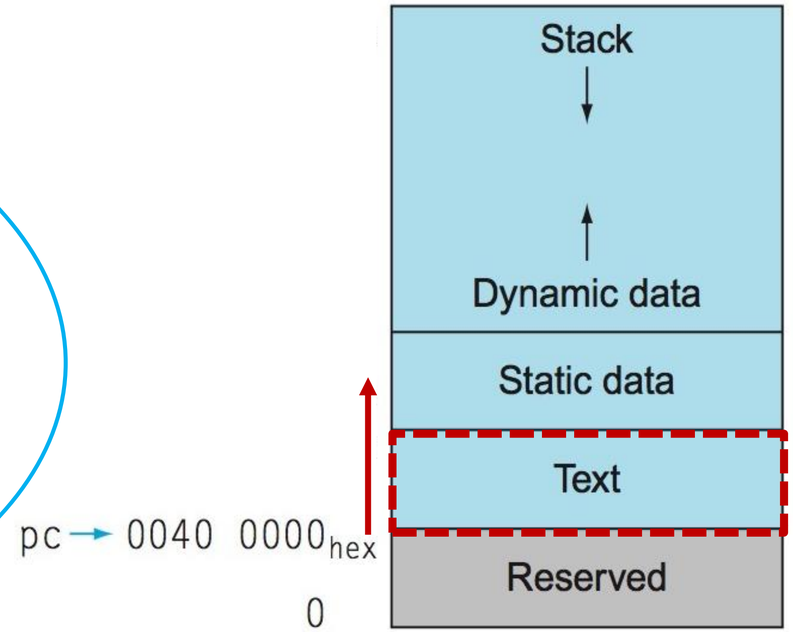
0	rs	rt	rd	0	0x20
6	5	5	5	5	6

add $10, $8, $9

Appendice A p.51

RECAP





Ruolo del puntatore
PROGRAM COUNTER (PC)

Dopo che un'istruzione viene letta (*fetch*), **il PC viene automaticamente incrementato di 4**, passando all'istruzione successiva.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

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

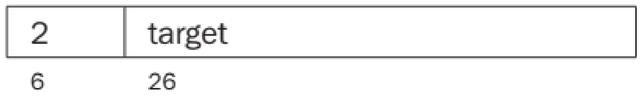
Esempio:

00001000000000000000010000111010

«salta all'indirizzo *target address*»

Jump

`j target`



Unconditionally jump to the instruction at target.

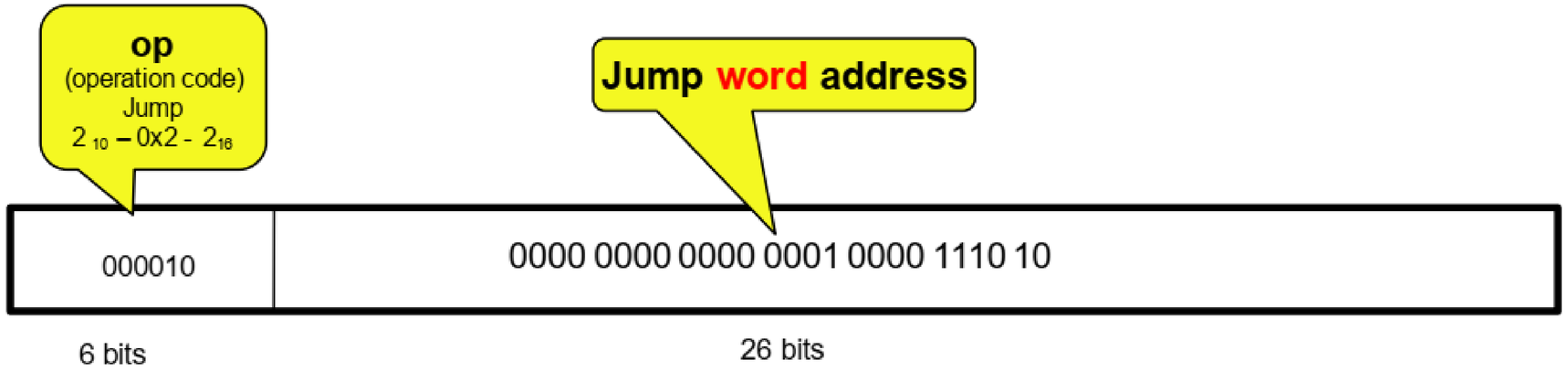
j addr

Appendice A p.51

AGENDA DI OGGI

- *Recap (ISA, formati R-type, J-type)*
- Formati istruzioni MIPS32
- Indirizzamento alla memoria
- *Catena programmativa (parte 1)*
- *Struttura di un programma in Assembly*

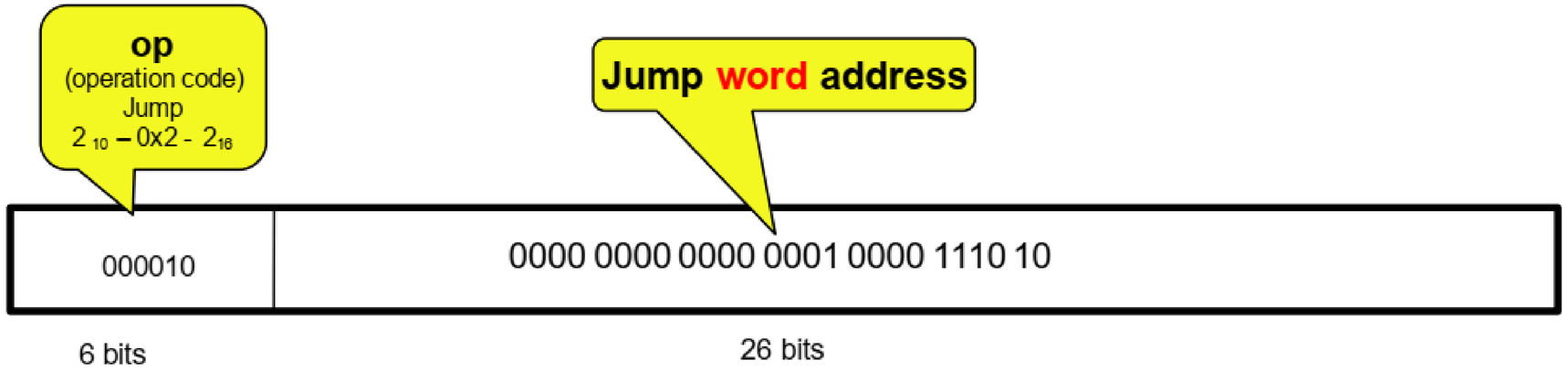
FORMATO J-TYPE: ESEMPIO



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

Per ottenere l'indirizzo effettivo bisogna **«calcolarlo»**.

FORMATO J-TYPE: ESEMPIO



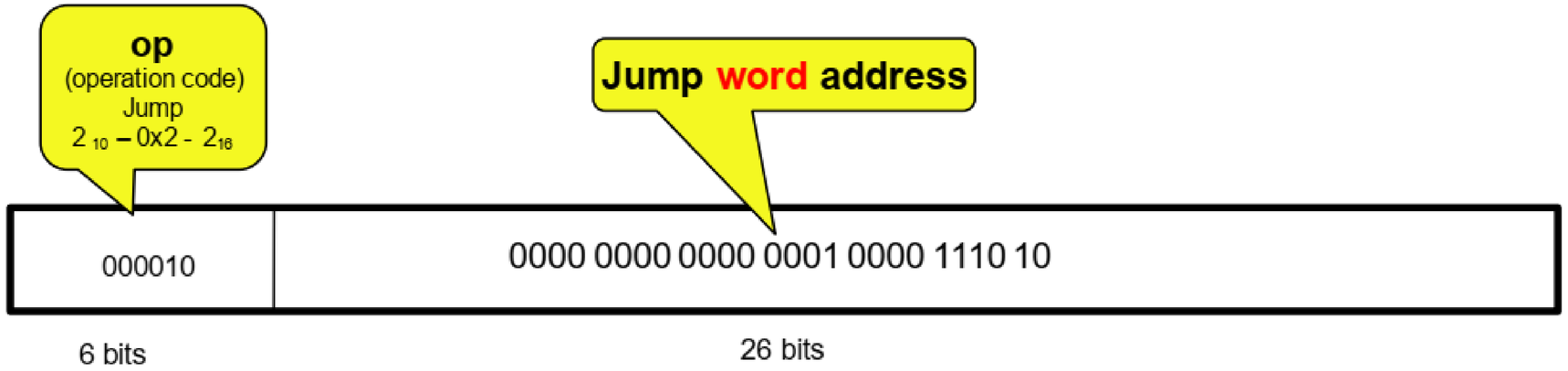
1.Shiftiamo il valore a sinistra di 2 bit (perché gli indirizzi MIPS sono allineati a 4 byte).

Codifica hex del target address: 0x0000043A

Indirizzo finale di salto: 0x0000043A << 2 = 0x000010E8

Nota. Ricordate che shiftare a sinistra di 2 bit equivale a **moltiplicare il numero per 4** (poiché 2^2=4).
In effetti: 0x**0000043A** = **1082**$_{10}$
1082$_{10}$ x 4 = 4328$_{10}$ = 0x**000010E8**

FORMATO J-TYPE: ESEMPIO



2. Aggiungiamo i 4 bit più significativi del PC corrente

Leggo il PC(*) 0000 0000 0100 0000 0000 0000 0011 0100

Compongo l'indirizzo finale 0000 0000 0000 0000 0001 0000 1110 1000

Carico in PC l'indirizzo: **0000** 0000 0000 0000 0001 0000 1110 1000 = $000010E8_{16}$

Invece che eseguire l'istruzione che era prevista (*), verrà eseguita l'istruzione contenuta all'indirizzo $000010E8_{16}$

FORMATO J-TYPE: ESEMPIO

Quindi **con 26 *bit*** si indirizzano **2^{28} *byte* di memoria** programma oppure 2^{26} *word*

Il range di indirizzi a cui posso arrivare in questo modo di indirizzamento (relativo al PC) sono tutti quelli del tipo XXXX 0000 0000 0000 0001 0000 1110 10**00**

26 bit

26 bit → 2^{26} configurazioni → Ciascuna di esse indirizza 1 byte ogni 4, ovvero l'inizio di word consecutive

→ 2^{28} configurazioni, se lascio variare anche gli ultimi due bit(LSB) → 2^{28} indirizzi (se ammetto anche gli indirizzi non multipli di 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.

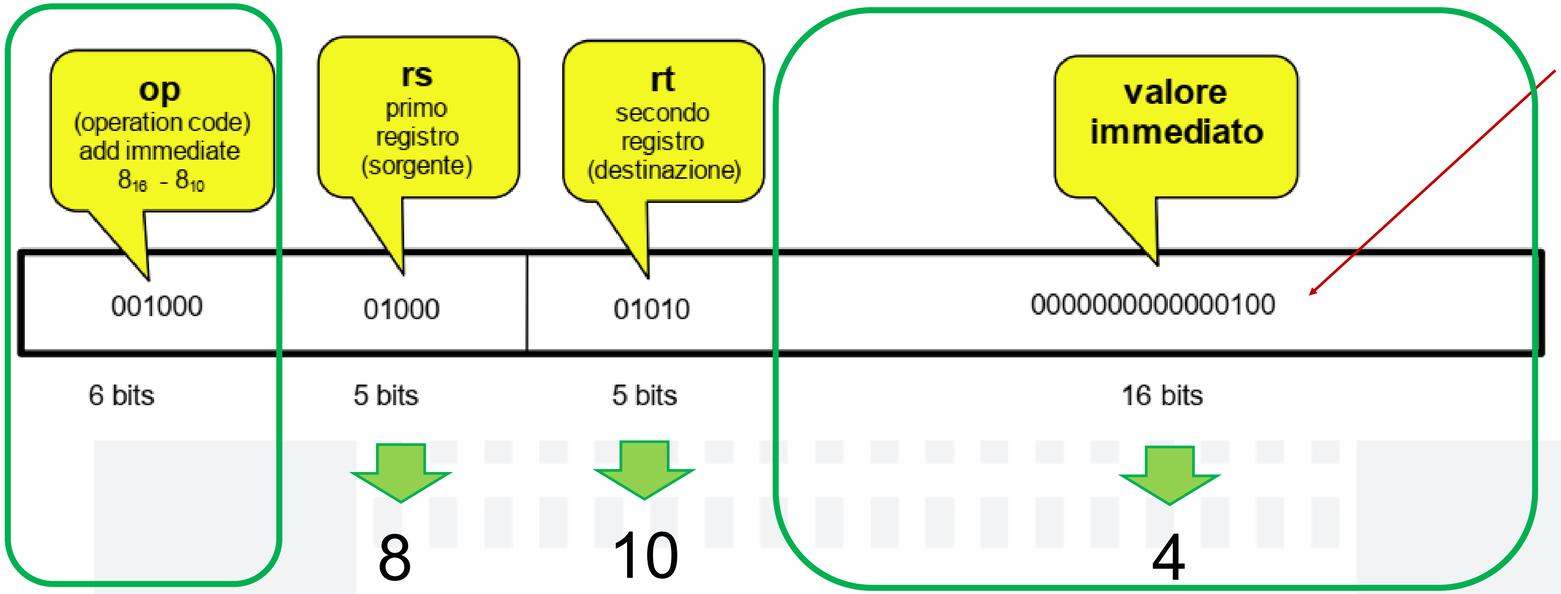
immediate (16 bit) → Valore immediato (costante o offset).

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

FORMATO I-TYPE: ADD IMMEDIATE

00100001000010100000000000000100

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



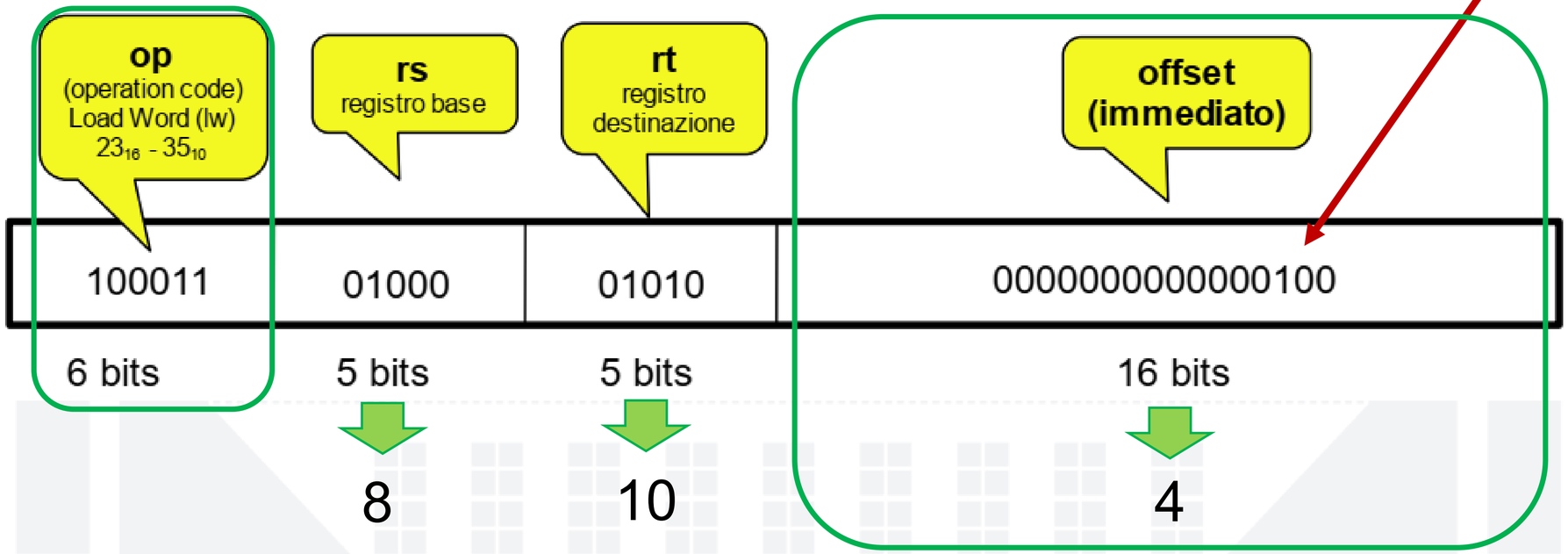
Nota. Qual è il **range di valori immediati** che si può esprimere con 16 bit in complemento a 2?
(-32768 <= valore <= 32767).

addi $10, $8, 4

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Immediate: il valore decimale che è codificato in questo campo (con i 16 bit a disposizione) va interpretato come **numero di byte.**

FORMATO I-TYPE: LOAD WORD



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

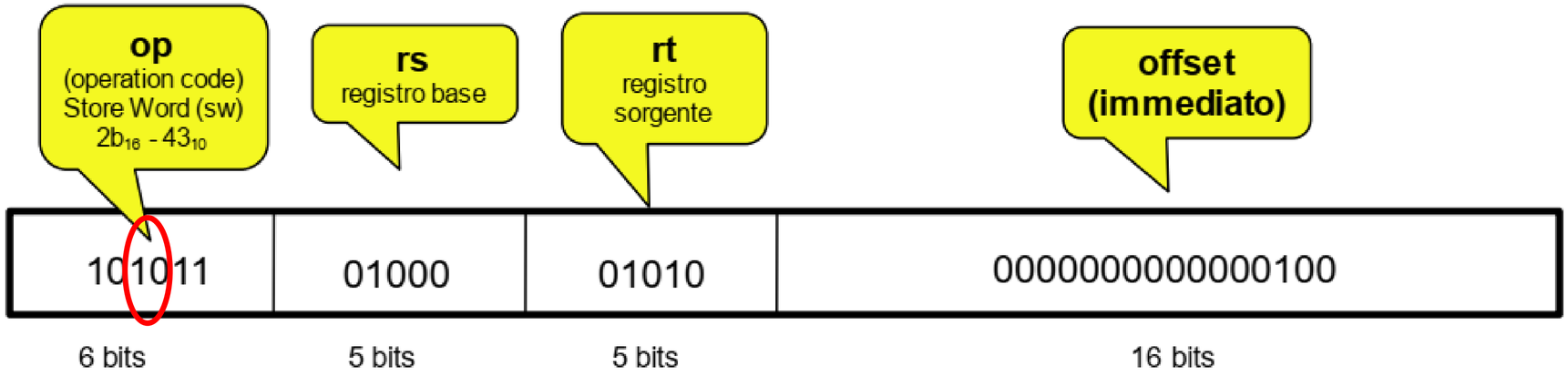
Usando un **numero di byte** e volendo noi accedere ad una word in memoria (stiamo facendo load *word*), **questo numero deve sempre essere un multiplo di 4.**

lw $10, 4($8)

lw $t0, **1**($s1) ❌ ERRORE DI ACCESSO NON ALLINEATO!

Indirizzo di memoria di PROVENIENZA!

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!

Usando un **numero di byte** e volendo noi accedere ad una word in memoria (stiamo eseguendo store *word*), **questo numero deve sempre essere un multiplo di 4.**

NOTA sull'allineamento per **store word** e **load word**

Usando un **numero di byte** e volendo noi accedere ad una word in memoria (stiamo facendo load *word*), **questo numero deve sempre essere un multiplo di 4.**

lw $10, 4($8)

lw $t0, **1**($s1) ❌ ERRORE DI ACCESSO NON ALLINEATO!

sw $10, 4($8)

Usando un **numero di byte** e volendo noi accedere ad una word in memoria (stiamo eseguendo store *word*), **questo numero deve sempre essere un multiplo di 4.**

L'*immediate* deve essere un multiplo di 4 (CORRETTO).

Se si vuole **accedere al byte**, si può fare, ma vanno usate le istruzioni di **load *byte*** e **store *byte***.

lb $10, 1($8)

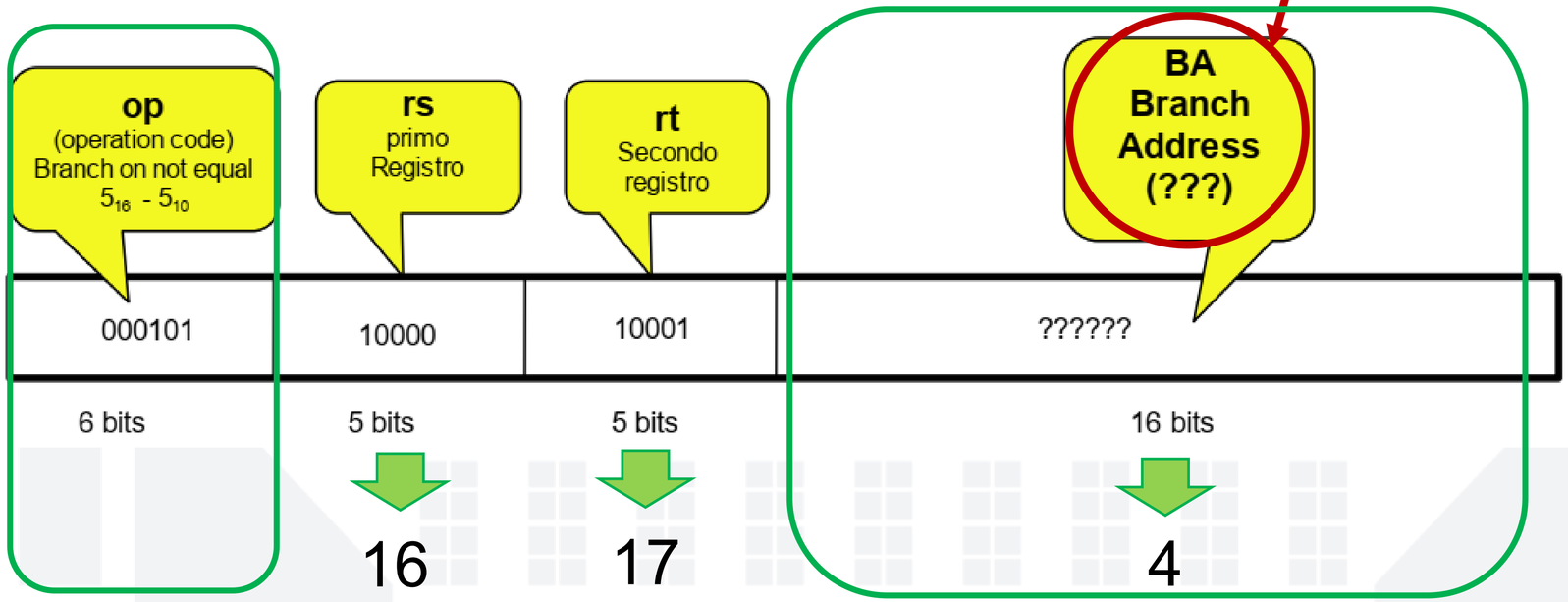
`sb` e `lb` non richiedono allineamento

FORMATO I-TYPE: BRANCH

Nota. Espressa in n. di **word (in CA2)**

Perché?

Ricorda: le istruzioni sono memorizzate su 4 Byte = 1 word



Salta a Branch Address se il contenuto del registro 16 (rs) è diverso dal contenuto del registro 17 (rt)

BranchAddr rappresenta il numero di word (4 byte) da saltare rispetto all'indirizzo memorizzato nel *Program Counter* ➔ nuovo PC = PC + BranchAddr

```
bne $16, $17, Exit
```

BRANCH VS JUMP

Branch con offset in numero di word (es. beq, bne)

- Il salto è **relativo al Program Counter**.
- L'indirizzo di destinazione è calcolato come:

$$PC(+4) + (offset \times 4)$$

- L'offset è espresso in **numero di word** (non byte).
- Il campo immediato è tipicamente **16 bit**, quindi il range è limitato: $\pm 2^{15}$ *word* $= \pm 32768$ *word* cioè circa ± 128 *KB*(*)

Serve quindi per **salti locali**, tipicamente cicli (loop), istruzioni if/else, piccoli blocchi di codice vicini.

Salto incondizionato (j, jal)

- L'istruzione contiene un **jump target address** (26 bit).

- L'indirizzo finale viene costruito combinando i **26 bit dell'istruzione con i bit alti del PC**.

Range molto più grande: circa **256 MB di spazio di indirizzi** nello stesso segmento (indirizzo diretto).

Serve per **salti lontani, chiamate di funzione**, **trasferimenti tra parti distanti del programma**

Tipo istruzione	Indirizzo	Range	Uso tipico
branch (beq, bne)	offset relativo al PC (in word)	Piccolo (± 128 *KB)*	loop, if
jump (j, jal)	indirizzo quasi assoluto	molto grande (256 *MB*)	funzioni, salti lontani

(*) 1 word = 4 byte. Max salto: 32768 x 4 = 131072 byte, 1 KB = 1024 byte ➔ 131072 byte sono 128 KB

INDIRIZZAMENTO IN MIPS32

L'**indirizzamento** in informatica e nelle architetture dei processori (come MIPS32) si riferisce al **modo in cui un'istruzione specifica l'ubicazione dei dati o dell'indirizzo di salto** all'interno della memoria o dei registri.

In altre parole, l'indirizzamento definisce **come il processore interpreta gli operandi** di un'istruzione per eseguire operazioni su dati memorizzati in registri, memoria o forniti direttamente nell'istruzione stessa.

CATEGORIE DI INDIRIZZAMENTO IN MIPS32 (1/2)

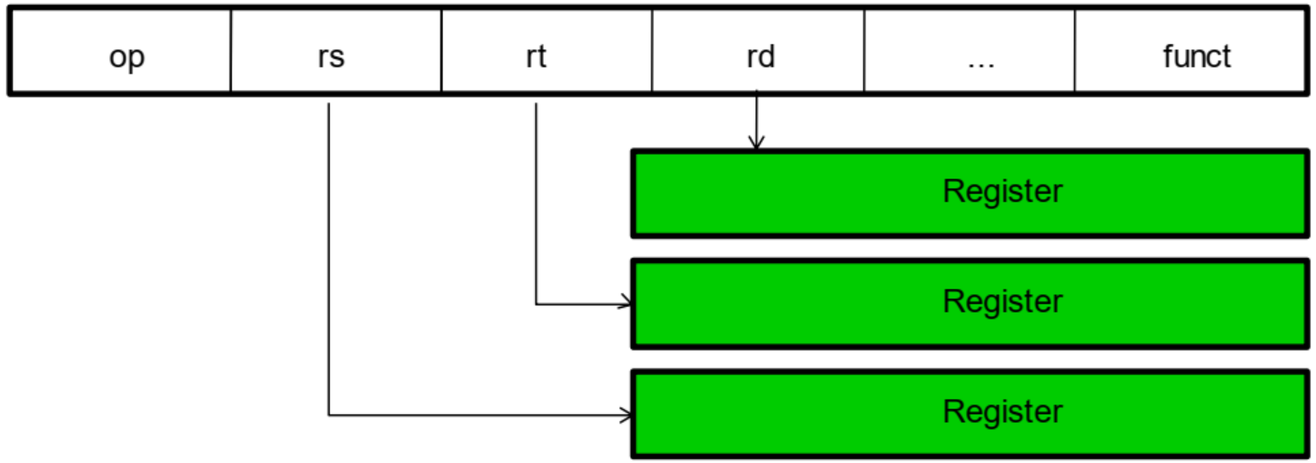
Immediate addressing

addi $t0, $zero, 10

L'operando è un valore costante incluso direttamente nell'istruzione.

Register addressing

add $10, $8, $9



L'operando è contenuto in un registro della CPU.

Jump addressing

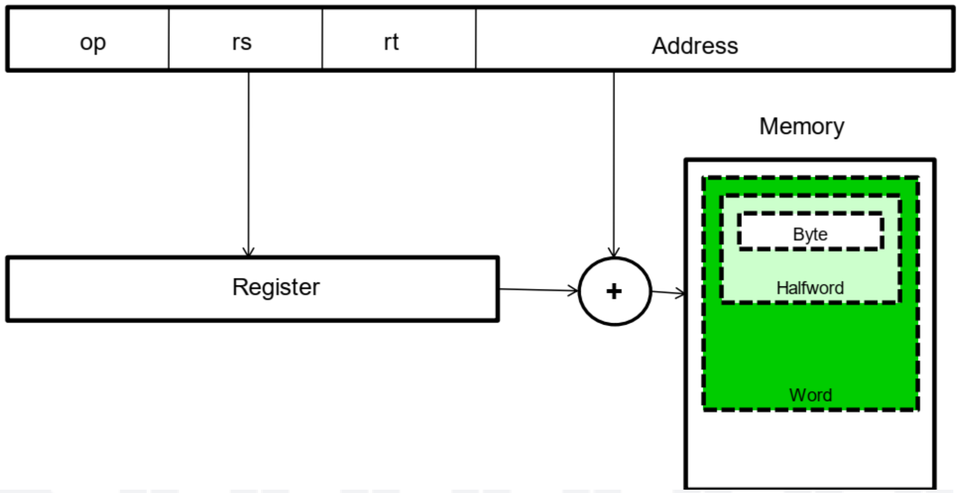
j 0x000010E8

L'operando è un **indirizzo assoluto** (parziale) memorizzato nei 26 bit dell'istruzione.

CATEGORIE DI INDIRIZZAMENTO IN MIPS32 (2/2)

Base addressing

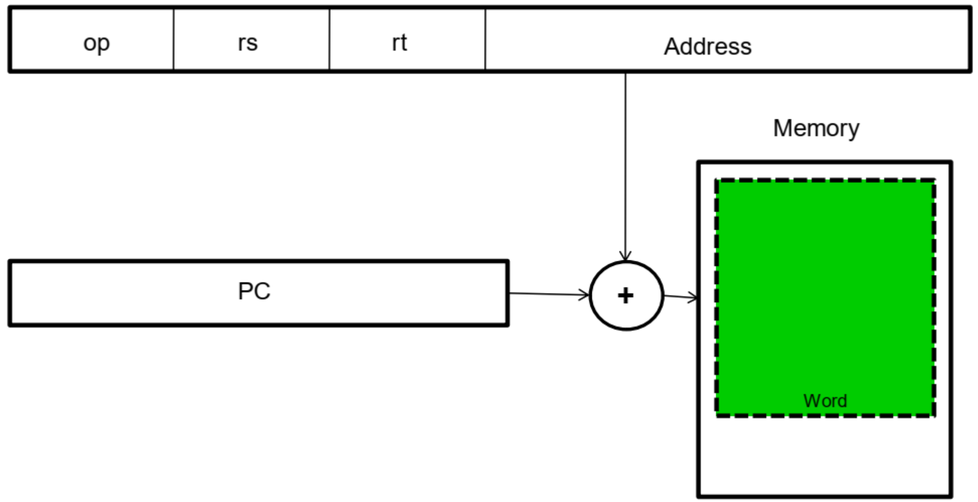
lw $4, 0($29)



L'operando è memorizzato in RAM, e si usa un registro base più un offset.

PC addressing

bne $16, $17, Exit



L'indirizzo di destinazione è calcolato sommando un offset al valore del Program Counter (PC).

ESERCIZIO 1

Determinare a quale istruzione macchina MIPS corrisponde la sequenza binaria

00000000011010010010000000100000

Soluzione nelle slide delle prossime lezioni

ESERCIZIO 2

Determinare a quale istruzione macchina MIPS corrisponde la sequenza binaria

00100010111010110000000001100000

Soluzione nelle slide della prossima lezione

ESERCIZIO 3

A quale istruzione macchina MIPS corrisponde il

codice esadecimale: 0x8fa40000

Soluzione nelle slide della prossima lezione

Materiale per la lezione

- Patterson & Hennessy, Cap.2
- Patterson & Hennessy, Cap.5
- Appendix A (da pagina 49)

Prossima lezione: 11 ***marzo, h.14:00, aula 3B***