Università degli Studi di Milano «La Statale»
Dipartimento di Informatica

Lezione 12:

CPU e linguaggio MIPS

Parte C: istruzioni di tipo «I»

Marco Tarini

55

Prossima istruzione di cui vogliamo dotarci

Istruzione assembly già vista: `ADD $30, $6, $11`

A parole: *« Somma il Registro 6 al Registro 11 e memorizza il risultato nel Registro 30 »*

Istruzione assembly nuova: `ADDI $30, $3, 240`

A parole: *« Somma il Registro 3 con il **valore 240** e memorizza il risultato nel Registro 30 »*

valore "immediato"

58

In linguaggio macchina MIPS:

ADD $30, $6, $11 — istruzione di tipo "R" (registro)

0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0

Operazione fra registri	6	11	30	---	SUM
OPCODE	*RS*	*RT*	*RD*	*---*	*FUNCTION*

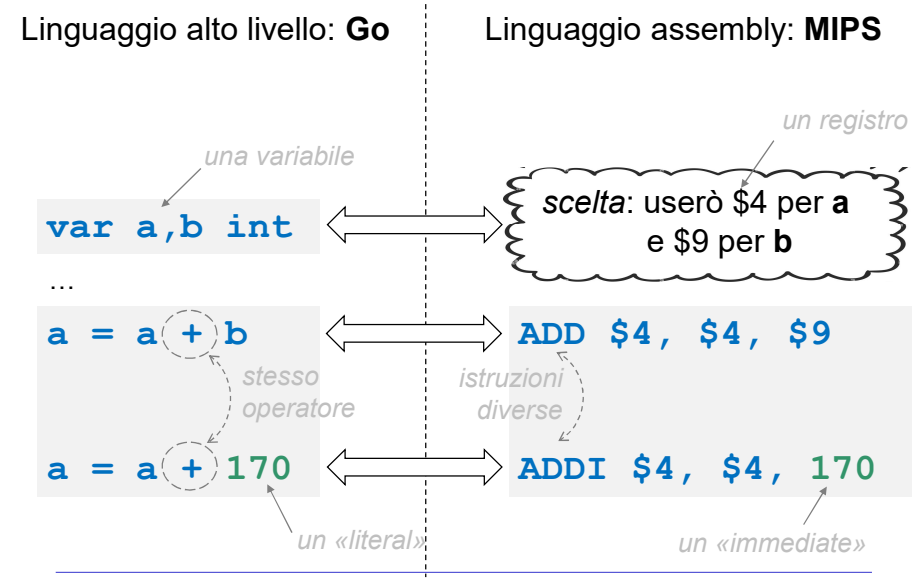
ADDI $30, $3, 240 — istruzione di tipo "I" (immediato)

0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0

Add register to immediate	3	30	240
OPCODE	*RS*	*RT*	*IMMEDIATE (16 bits)*

59

C'è somma e somma (linguaggi alto e basso livello a confronto)



60

Un'altra istruzione di tipo "I"

0	0	1	1	1	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0

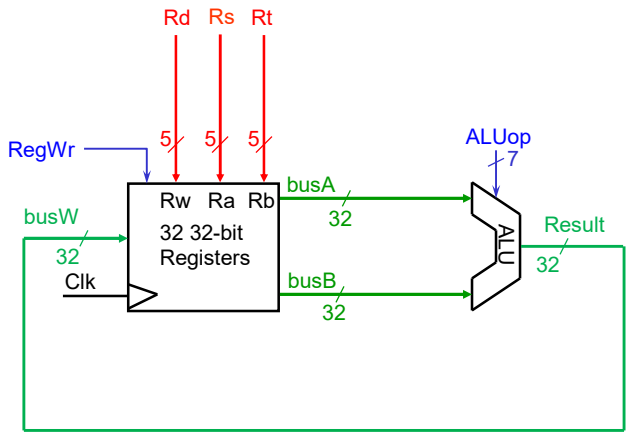
XOR between register to immediate	3	11	240
OPCODE	*RS*	*RT*	*IMMEDIATE (16 bits)*

Tradotta in assembly MIPS: `XORI $11, $3, 240`

A parole: *« Fai lo XOR bit-a-bit fra il Registro 3 e* **0..011110000** *e memorizza il risultato nel Registro 11 »*

61

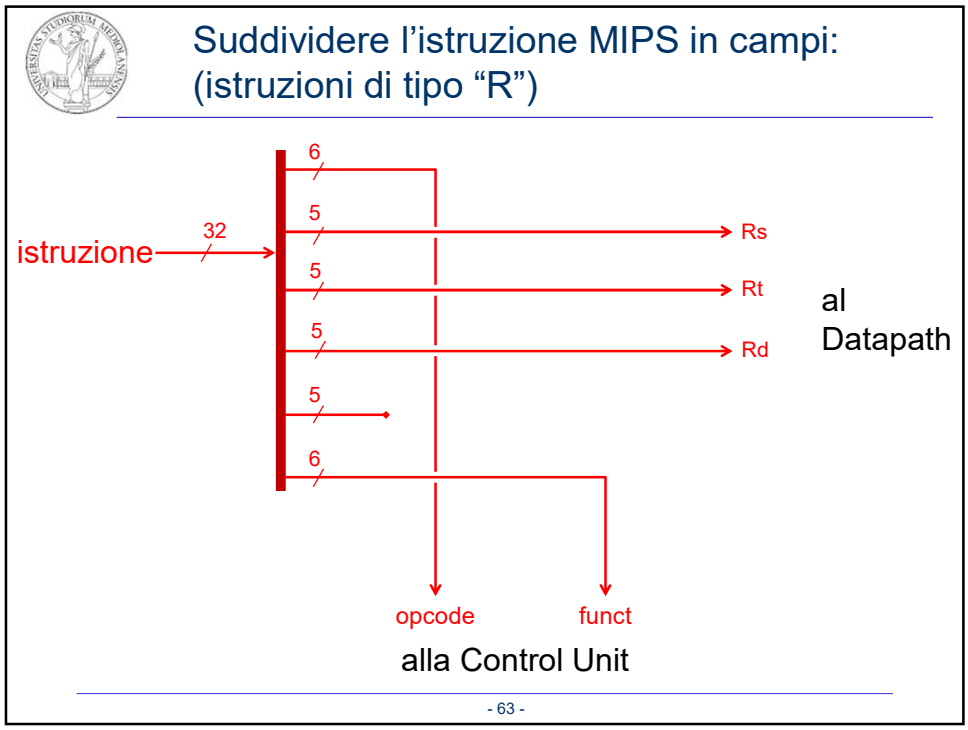
Datapath fino a questo momento



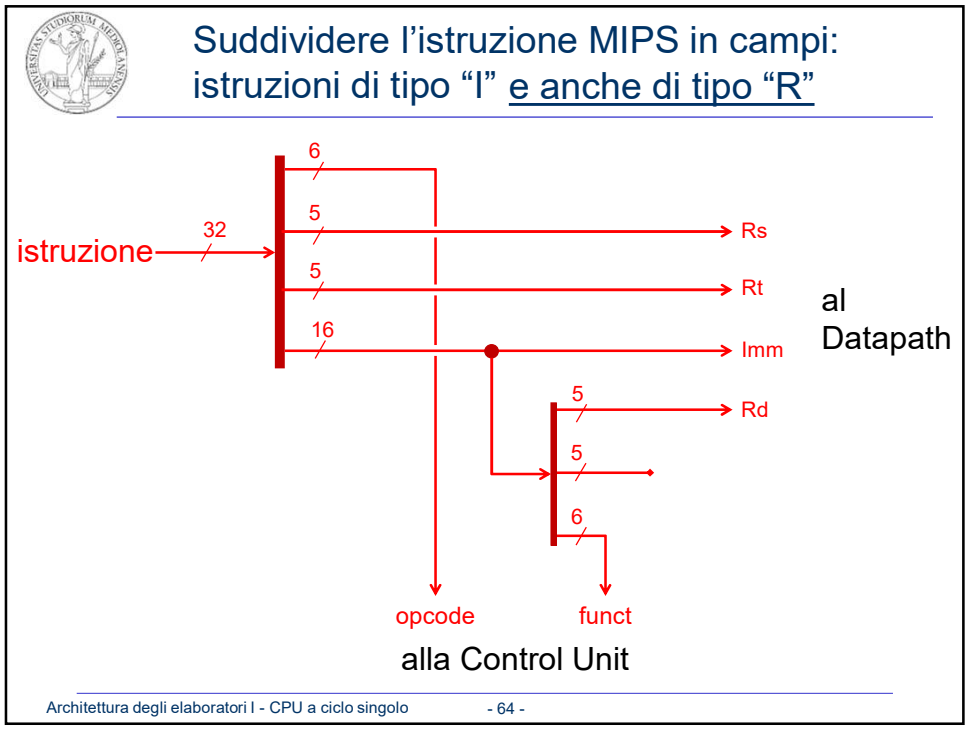
Rosso: da Istruz.
Blu: da U.C.

- 62 -

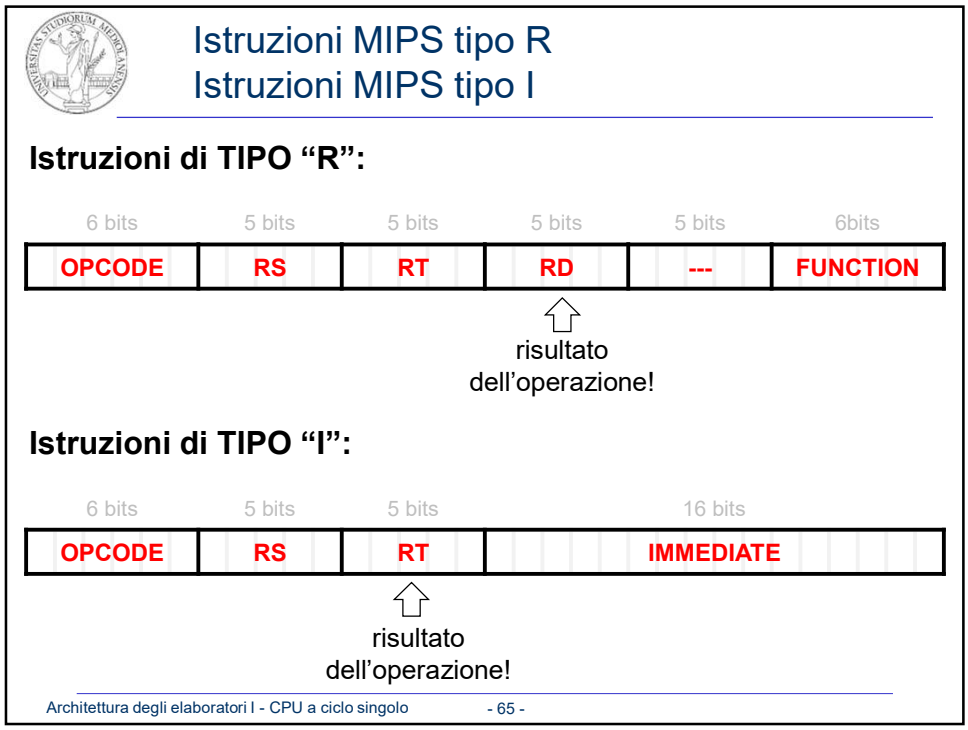
62


Suddividere l'istruzione MIPS in campi:
(istruzioni di tipo "R")
istruzione
32
6
5
5
5
5
6
Rs
Rt
Rd
al
Datapath
opcode
funct
alla Control Unit
- 63 -

63

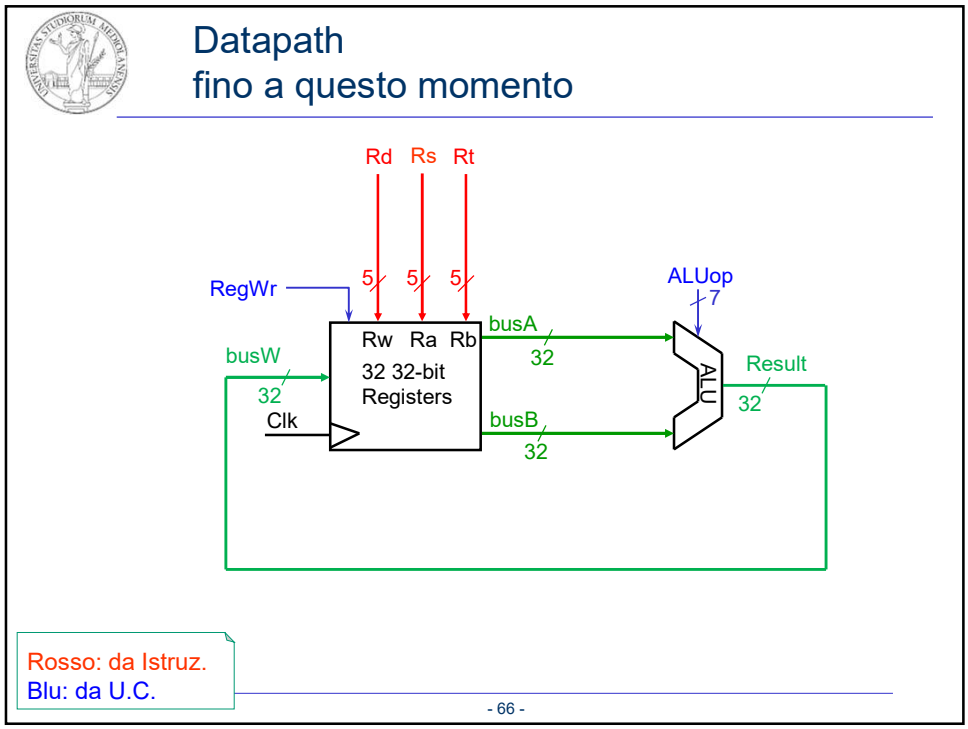

Suddividere l'istruzione MIPS in campi:
istruzioni di tipo "I" e anche di tipo "R"
istruzione
32
6
5
5
16
5
5
6
Rs
Rt
Imm
Rd
al
Datapath
opcode
funct
alla Control Unit
Architettura degli elaboratori I - CPU a ciclo singolo
- 64 -

64

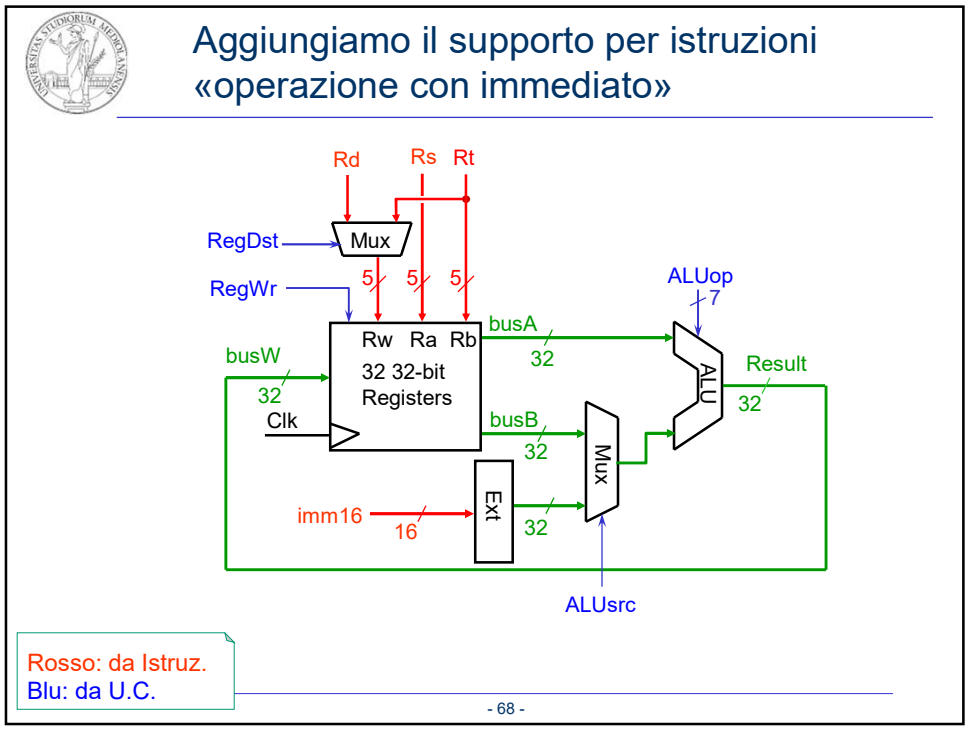

Istruzioni MIPS tipo R
Istruzioni MIPS tipo I
Istruzioni di TIPO "R":
6 bits
5 bits
5 bits
5 bits
5 bits
6bits
OPCODE
RS
RT
RD

FUNCTION
risultato
dell'operazione!
Istruzioni di TIPO "I":
6 bits
5 bits
5 bits
16 bits
OPCODE
RS
RT
IMMEDIATE
risultato
dell'operazione!
Architettura degli elaboratori I - CPU a ciclo singolo
- 65 -

65

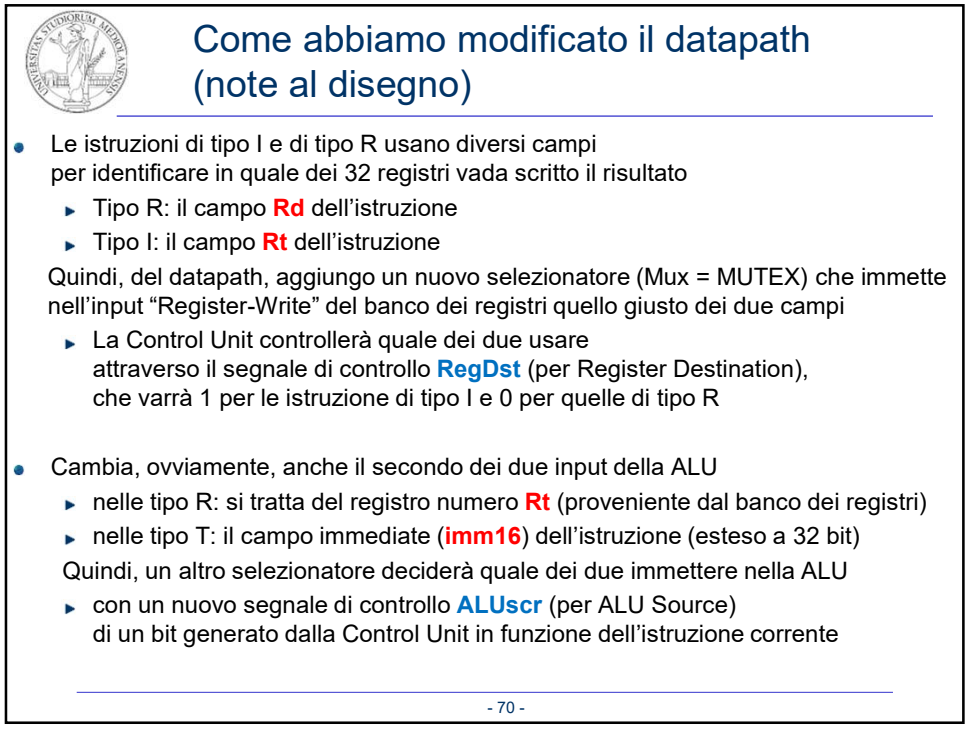

Datapath
fino a questo momento
Rd
Rs
Rt
5
5
5
RegWr
ALUop
7
busA
32
Rw Ra Rb
32 32-bit
Registers
busW
32
Clk
busB
32
ALU
Result
32
Rosso: da Istruz.
Blu: da U.C.
- 66 -

66



Aggiungiamo il supporto per istruzioni «operazione con immediato»

68



Come abbiamo modificato il datapath (note al disegno)

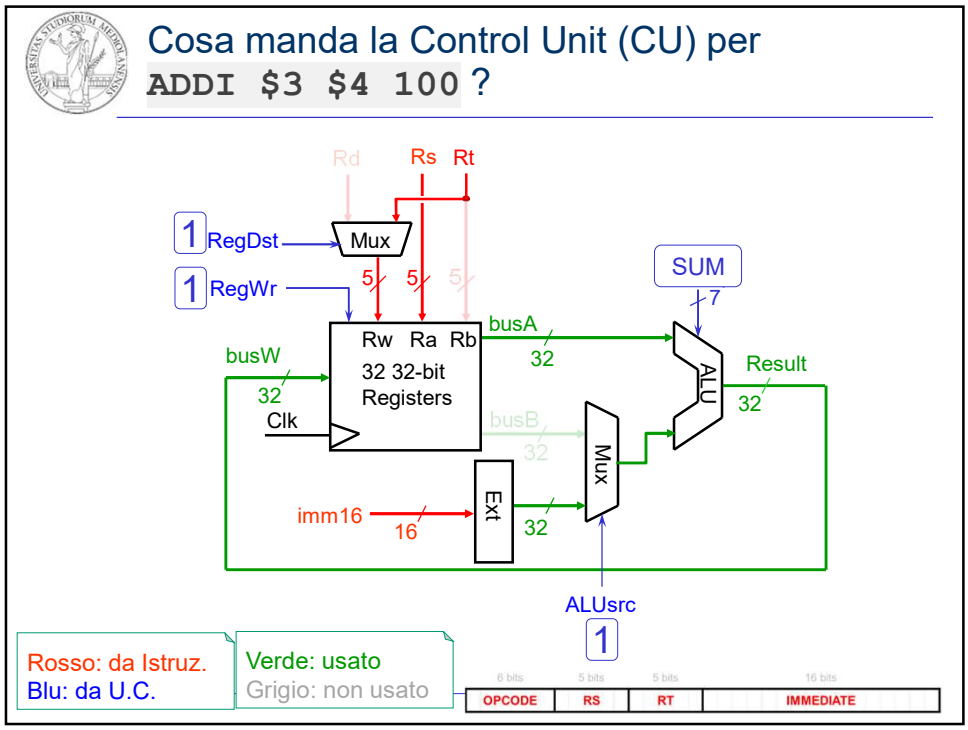
- Le istruzioni di tipo I e di tipo R usano diversi campi per identificare in quale dei 32 registri vada scritto il risultato
  - Tipo R: il campo **Rd** dell'istruzione
  - Tipo I: il campo **Rt** dell'istruzione

  Quindi, del datapath, aggiungo un nuovo selezionatore (Mux = MUTEX) che immette nell'input "Register-Write" del banco dei registri quello giusto dei due campi
  - La Control Unit controllerà quale dei due usare attraverso il segnale di controllo **RegDst** (per Register Destination), che varrà 1 per le istruzione di tipo I e 0 per quelle di tipo R

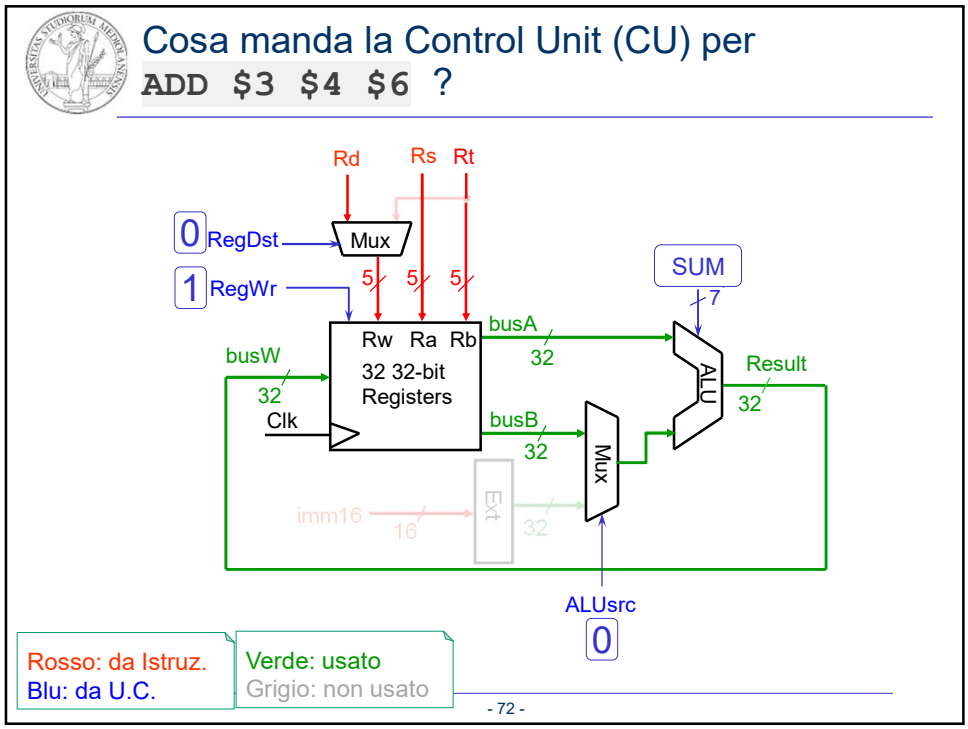
- Cambia, ovviamente, anche il secondo dei due input della ALU
  - nelle tipo R: si tratta del registro numero **Rt** (proveniente dal banco dei registri)
  - nelle tipo T: il campo immediate (**imm16**) dell'istruzione (esteso a 32 bit)

  Quindi, un altro selezionatore deciderà quale dei due immettere nella ALU
  - con un nuovo segnale di controllo **ALUscr** (per ALU Source) di un bit generato dalla Control Unit in funzione dell'istruzione corrente

70


Cosa manda la Control Unit (CU) per
ADDI $3 $4 100 ?
Rd
Rs
Rt
1
RegDst
Mux
1
RegWr
5
5
5
SUM
7
busA
32
Rw Ra Rb
32 32-bit
Registers
busW
32
Clk
ALU
Result
32
busB
32
Mux
imm16
16
Ext
32
ALUsrc
1
Rosso: da Istruz.
Blu: da U.C.
Verde: usato
Grigio: non usato
6 bits
5 bits
5 bits
16 bits
OPCODE
RS
RT
IMMEDIATE

71


Cosa manda la Control Unit (CU) per
ADD $3 $4 $6 ?
Rd
Rs
Rt
0
RegDst
Mux
1
RegWr
5
5
5
SUM
7
busA
32
Rw Ra Rb
32 32-bit
Registers
busW
32
Clk
ALU
Result
32
busB
32
Mux
imm16
16
Ext
32
ALUsrc
0
Rosso: da Istruz.
Blu: da U.C.
Verde: usato
Grigio: non usato
- 72 -

72

Alcune istruzioni aritmetiche-logiche di tipo I ("I = con immediate")

Istruzione	Sintassi assembly (esempio)	Semantica (dell'esempio)	Note
*add **immediate***	`addi $1 $2 150`	$1 = $2 + 150	Check di overflow
*add **immediate** unsigned*	`addiu $1 $2 150`	$1 = $2 + 150	Niente check
*or **immediate***	`ori $1 $2 35`	$1 = $2 ∨ 35	Bit a bit (in inglese: bit-wise)
*and **immediate***	`andi $1 $2 35`	$1 = $2 ∧ 35	
*xor **immediate***	`xori $1 $2 35`	$1 = $2 ⊕ 35	
shift logical left	`sll $1 $2 9`	$1 = $2 ≪ 9	
shift logical right	`slr $1 $2 9`	$1 = $2 ≫ 9	da sx spunta 0
shift aritmetical right	`sar $1 $2 9`	$1 = $2 ≫ 9	da sx spunta il MSB

73

E la differenza con immediate?

- Nuovamente, filosofia RISC in azione.
- Invece di

```
SUBI $3 $4 150
```

<= Subtract Immediate?
non esiste (no, neanche come pseudo istruzione)

basta usare

```
ADDI $3 $4 -150
```


74

Operazioni di shift con 2° operando costante: nome dell'istruzione in assembly

- Nome dell'istruzione assembly (e del codice mnemonico):
  - quando il 2° operando è un registro (operazione di tipo R)
    le operazioni di shift si chiamano «shift … variable» (**v** nel codice mnemonico)
  - quando il 2° operando è costante (un immediate),
    non c'è «variable» nel nome
- Questo è un po' inconsistente con quello che succede con le altre operazioni
  - Paragona:

con 2° parametro registro	*con 2° parametro immediate*
add	add **immediate**
xor	xor **immediate**
shift logical right **variable**	shift logical right
shift logical left **variable**	shift logical left

75

Operazioni di shift con 2° operando costante: in linguaggio macchina (mini-dettaglio)

- **Confessione:** (per la precisione)
  - In questo corso, semplifichiamo un po' la cose rispetto al reale linguaggio macchina MIPS-32 (come definito dallo standard)
  - Nel linguaggio reale, le istruzioni di *shift* non *variabile,* come ***shift logical left,*** sono anch'esse di «tipo R», come quelle *variable*
  - Il 2° operando non è il campo «immediate» di una istruzione di «tipo I»
    Invece è nascosto qui, il quel campo di 5 bit delle «tipo R» che abbiamo ignorato:

```
SLL $5, $12, 6
```

La reale traduzione in linguaggio macchina:

0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0

Operaz fra Registri	12	*(unused)*	5	6	Shift a sinistra
OPCODE	*RS*	*RT*	*RD*	*SHIFT VAL*	*FUNCTION*

- 76 -

76

Come scrivere gli **immediate** in assembly MIPS (cioè: sintassi dei *literal* di quel linguaggio)

- Come abbiamo visto, un valore immediate consiste in 16 bit
- In assembly, posso descrivere il valori immediate in vari modi:

In base 10:

```
ADDI $3 $4 150
```

Incluso, con segno (verrà tradotto in CP2)!

```
ADDI $3 $4 -150
```

Oppure in base 16:

```
ADDI $3 $4 0xABBA
```

Oppure usando il carattere del numero in codice ASCII (nota gli apicetti)

```
ADDI $3 $4 '!'
```

In ciascun caso, l'**assembler** traduce il «**literal**» in 16 bit,
quando volge l'istruzione in **linguaggio macchina** (grazie, assembler!)

77

Inizializzazione esplicita dei registri

- Come caricare dei valori nei registri?
- Esempio:
  - caricare in $6 il valore 27
- L'Instruction Set MIPS è così RISC che
  non ha un istruzione per inizializzare i registri !
- Questo perché possiamo utilizzare (fra altre possibilità)
  l'istruzione «somma fra registro e valore immediato»

```
addi $6 $0 27
```

Registro speciale,
il cui valore è sempre 0

78

Inizializzazione esplicita dei registri: la pseudo istruzione Load immediate

- Questa soluzione però è poco leggibile
- il linguaggio assembly MIPS ci mette quindi a disposizione una **pseudo-istruzione** più chiara per ottenere lo stesso effetto:

«Load immediate»

```
li $6 27
```

TRADUZIONE AUTOMATICA
effettuata dell'assembler

```
addiu $6 $0 27
```

79

Inizializzazione esplicita dei registri: la pseudo istruzione Load immediate

- Problema: il campo immediate è di soli 16 bit.
  - Viene esteso con 0 per riempire il registro di 32 bit
- A volte, vogliamo inizializzare *tutti i 32 bit* di un registro. Come fare?
- Ad esempio, come inizializziamo il registro $9 al valore 0xABCD1234 ?

```
li $9 0xABCD1234
```

TRADUZIONE AUTOMATICA
effettuata dall'assembler

```
addiu $9 $0 0xABCD
sll   $9 $9 16
ori   $9 $9 0x1234
```

$9 ← 0x | 0 | 0 | 0 | 0 | A | B | C | D |

$9 ← 0x | A | B | C | D | 0 | 0 | 0 | 0 |

$9 ← 0x | A | B | C | D | 1 | 2 | 3 | 4 |

80

Inizializzazione esplicita dei registri: la pseudo istruzione Load immediate

- Ottimizzazione: per risparmiare un'istruzione nel programma, il MIPS mette a disposizione un'apposta istruzione detta «**load upper immediate**» (`lui`)
  - la sua implementazione HW: è solo un'altra funz prevista dalla ALU

f(A,B) = B<<16

```
li $9 0xABCD1234
```

TRADUZIONE AUTOMATICA
effettuata dall'assembler

```
lui $9 0xABCD
ori  $9 $9 0x1234
```

$9 ← 0x A B C D 0 0 0 0

$9 ← 0x A B C D 1 2 3 4

81

Estensione con 0 VS estensione in segno (ripasso, con un esempio)

In binario senza segno

- Su 4 bit:
  1101
  denota tredici;
- allora su 8 bit
  00001101
  denota ancora tredici
  (banalmente)

- **Per estendere basta aggiungere tanti 0 a sx**
- detta **«estensione con 0»**

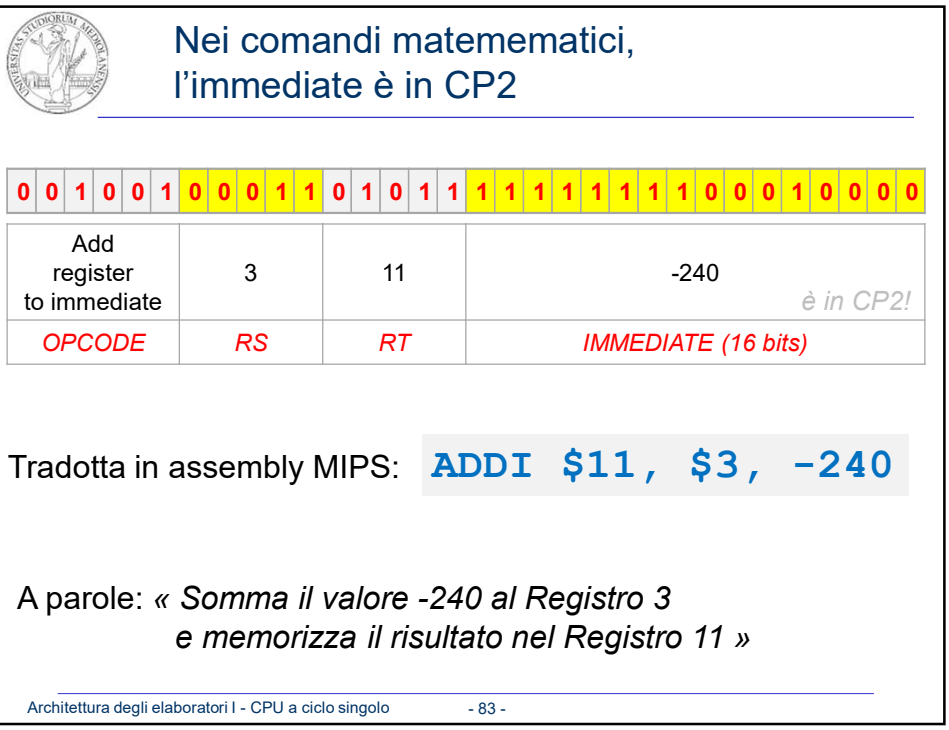
In complemento a 2

- Su 4 bit,
  per i positivi: 0101
  denota più cinque;
- allora su 8 bit: **0000**0101
  denota ancora più cinque
- per i negativi: 1101
  denota meno tre;
  allora su 8 bit: **1111**1101
  denota ancora meno tre

- **Per estendere, basta ripetere il MSB a sx**
- detta **«estensione in segno»**

Architettura degli elaboratori I - CPU a ciclo singolo - 82 -

82



Nei comandi matemematici, l'immediate è in CP2

0	0	1	0	0	1	0	0	0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0

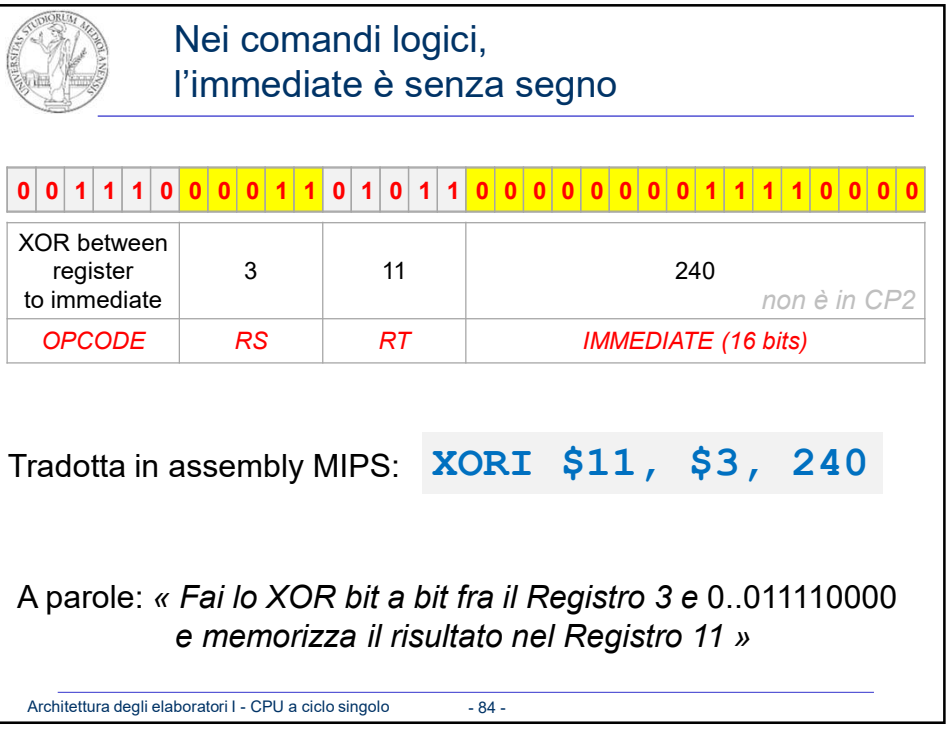
Add register to immediate	3	11	-240 *è in CP2!*
OPCODE	*RS*	*RT*	*IMMEDIATE (16 bits)*

Tradotta in assembly MIPS: `ADDI $11, $3, -240`

A parole: *« Somma il valore -240 al Registro 3 e memorizza il risultato nel Registro 11 »*

Architettura degli elaboratori I - CPU a ciclo singolo - 83 -

83



Nei comandi logici, l'immediate è senza segno

0	0	1	1	1	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0

XOR between register to immediate	3	11	240 *non è in CP2*
OPCODE	*RS*	*RT*	*IMMEDIATE (16 bits)*

Tradotta in assembly MIPS: `XORI $11, $3, 240`

A parole: *« Fai lo XOR bit a bit fra il Registro 3 e* 0..011110000 *e memorizza il risultato nel Registro 11 »*

Architettura degli elaboratori I - CPU a ciclo singolo - 84 -

84

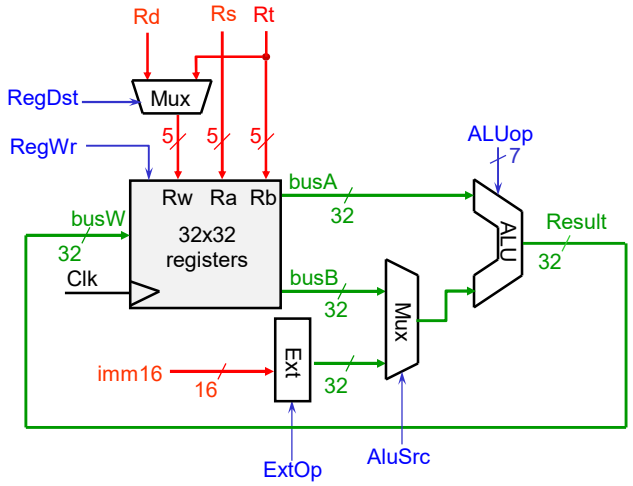
Operazioni con immediate in MIPS: matematiche VS logiche

- In MIPS, nei comandi con immediate **matematici** (come ADD) il parametro *immediate* va interpretato in CP2
  - E di conseguenza va esteso (a 32 bit) ***in segno***
  - La quantità Immediate può essere quindi sia positiva che negativa
  - Scelta strategica dell'IS!
    Riduce il numero di operazioni nell IS (filosofia RISC in azione).
  - Non ci sarà infatti bisogno di un'operazione "SUBI" (per sottrarre con Immediate): basterà ADD-izionare con valori imm. negativi
  - Nota: c'è comunque bisogno di un'operazione SUB fra registri, distinta da ADD, per eseguire sottrazioni *fra registri*
- Invece nelle comandi **logici** (come XOR o AND) il parametro immediate non è in CP2 e quindi va esteso ***con zero***
- Come gestire la diversa modalità di estensione? Risposta:
  - La Control Unit "sceglie" come va esteso, a seconda di OP-Code, mandando un apposito comando al datapath (chiamiamolo "extOp")
  - Nel datapath questo comando determina il comportamento di un **Estesore a scelta** (vedi lezione sui blocchi logici)

85

Datapath completo (per ora)

Rosso: da Istruz.
Blu: da U.C.



86