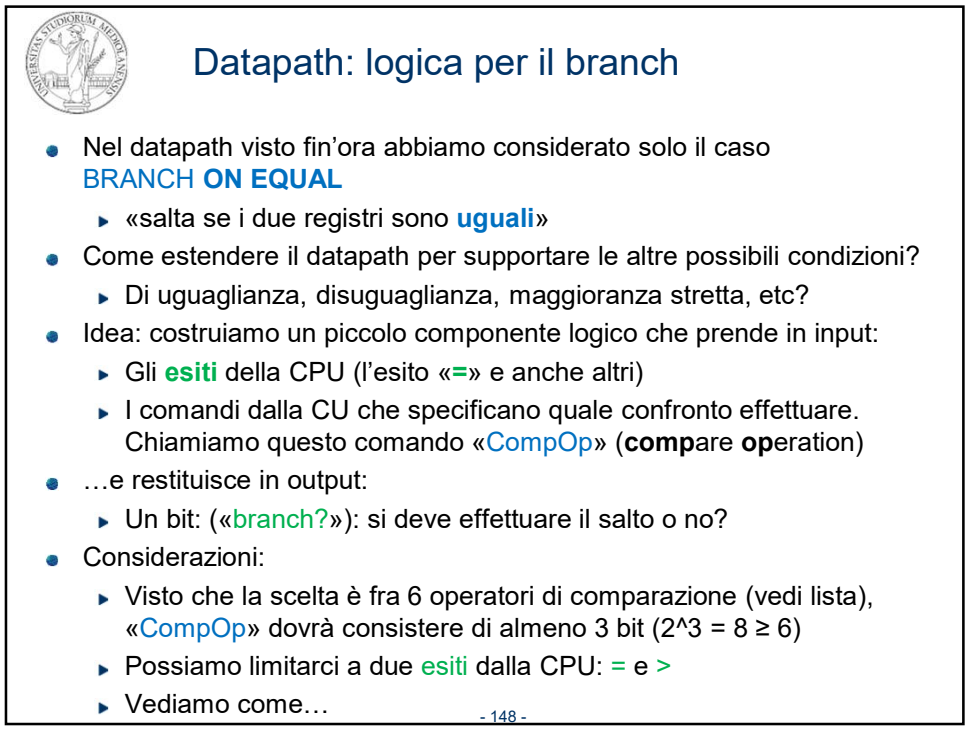
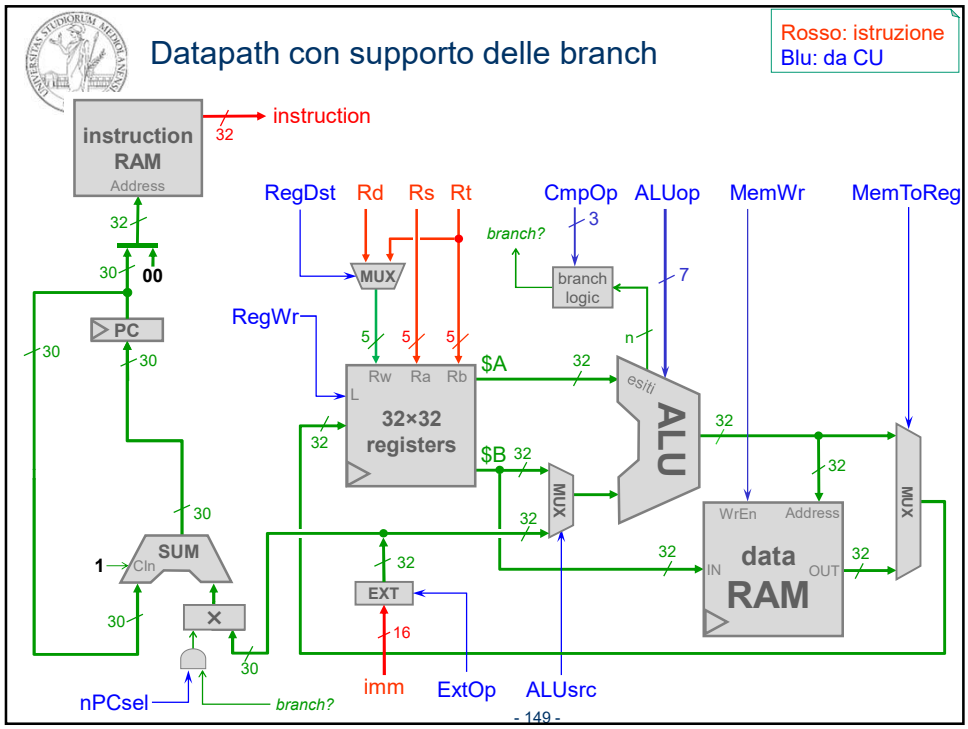


Datapath: logica per il branch

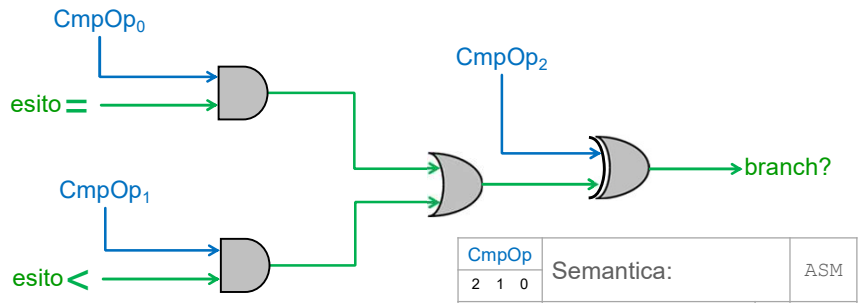
- Nel datapath visto fin'ora abbiamo considerato solo il caso BRANCH **ON EQUAL**
  - «salta se i due registri sono **uguali**»
- Come estendere il datapath per supportare le altre possibili condizioni?
  - Di uguaglianza, disuguaglianza, maggioranza stretta, etc?
- Idea: costruiamo un piccolo componente logico che prende in input:
  - Gli **esiti** della CPU (l'esito «**=**» e anche altri)
  - I comandi dalla CU che specificano quale confronto effettuare. Chiamiamo questo comando «CompOp» (**comp**are **op**eration)
- …e restituisce in output:
  - Un bit: («branch?»): si deve effettuare il salto o no?
- Considerazioni:
  - Visto che la scelta è fra 6 operatori di comparazione (vedi lista), «CompOp» dovrà consistere di almeno 3 bit (2^3 = 8 ≥ 6)
  - Possiamo limitarci a due esiti dalla CPU: = e >
  - Vediamo come…

148



149

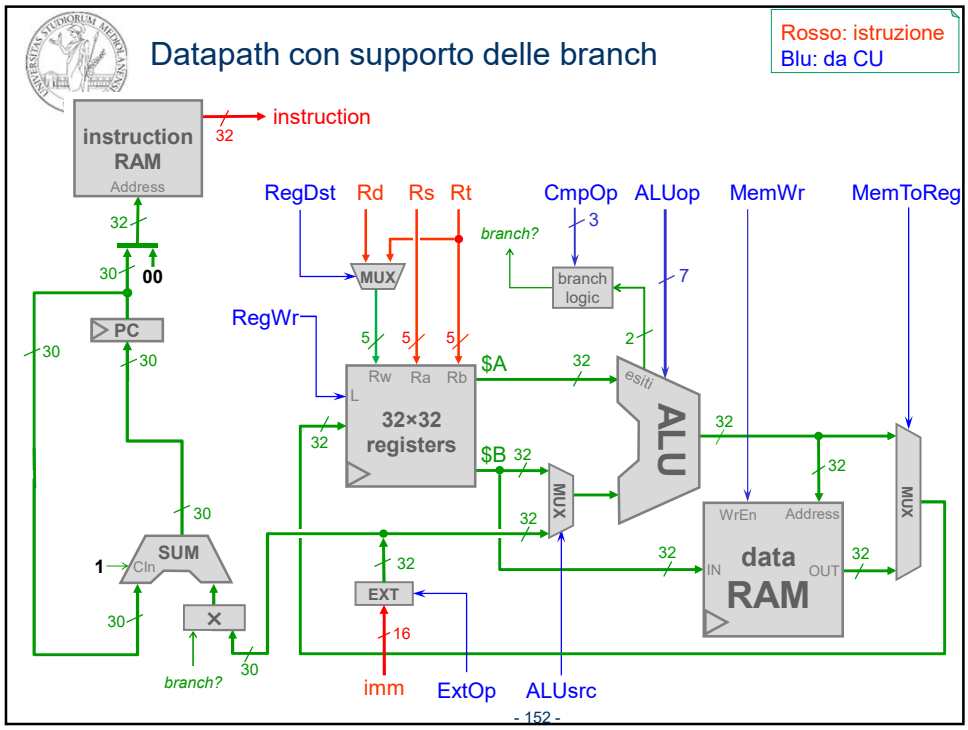
Branch Logic



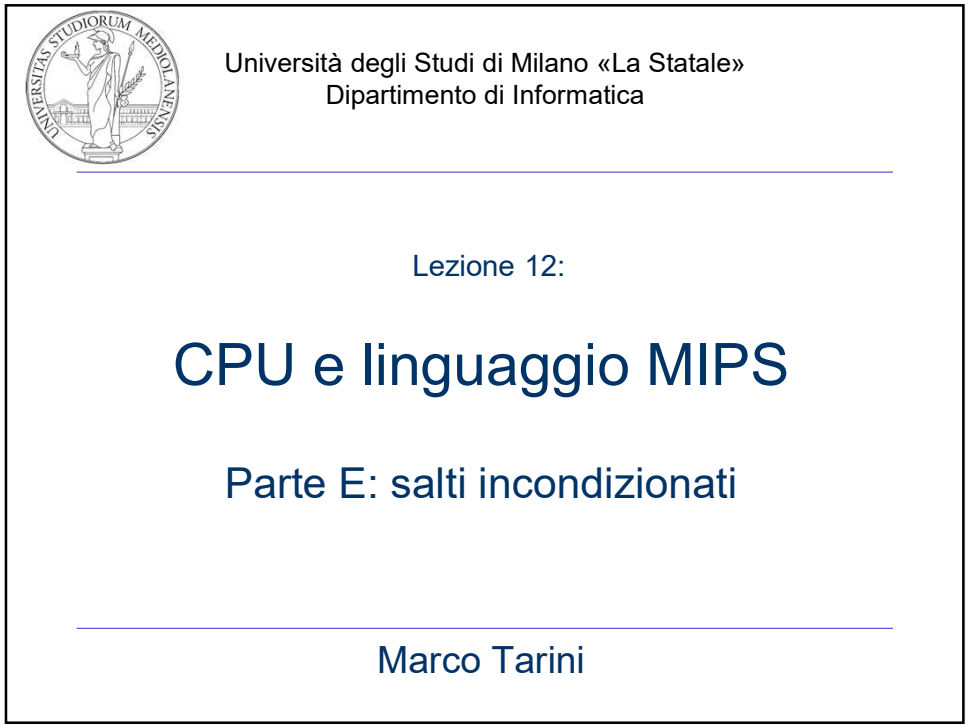
CmpOp 2 1 0	Semantica:		ASM
0 0 0	*never*		
0 0 1	equal	=	EQ
0 1 0	less than	<	LT
0 1 1	less or equal	≤	LE
1 0 0	*always*		
1 0 1	not equal	≠	NE
1 1 0	greater or equal	≥	GE
1 1 1	greater than	>	GT

Branch logic: osservazioni sulla soluzione vista

- Ciascuno degli otto possibili valori di CmpOp ha una semantica utile:
  - Tutte le sei possibili comparazioni (=, <, >, ≠, ≤, ≥)
  - I due valori speciali («Never» - 000, «Always» - 100), che producono sempre 0 o 1 rispettivamente per il bit «branch?» (a prescindere dagli esiti prodotti dalla ALU)
- Il valore «never» ci consente di rimuovere il controllo nPCsel usato nella nostra soluzione precedente
  - per le istruzioni che non sono di branch, la **CU** può mandare semplicemente «never» come CmpOp (000)
- Il valore «always» consente all'**istruction set MIPS** di includere facilmente un'istruzione di **branch**, cioè «salto condizionato», con condizione… «sempre» (cioè, in pratica, un salto non condizionato)
  - Quest'istruzione in assembly MIPS si scrive con la sitassi, ad es: `b 4` «salta sempre 4 istruzioni avanti a quella successiva»
  - Ma vedremo a breve istruzioni di salto propriamente «non condizionato»


Datapath con supporto delle branch
Rosso: istruzione
Blu: da CU
instruction RAM
Address
instruction
32
RegDst
Rd
Rs
Rt
CmpOp
ALUop
MemWr
MemToReg
branch?
MUX
branch logic
3
7
RegWr
PC
00
30
5
2
Rw
Ra
Rb
$A
$B
32×32 registers
ALU
esiti
SUM
Cin
1
EXT
16
imm
ExtOp
ALUsrc
WrEn
Address
data RAM
IN
OUT
- 152 -

152


Università degli Studi di Milano «La Statale»
Dipartimento di Informatica
Lezione 12:
CPU e linguaggio MIPS
Parte E: salti incondizionati
Marco Tarini

153

Prossima istruzione: **jump** (salti incondizionati)

- A differenza delle **branch** (salti condizionati), le istruzioni di **jump** (salti incondizionati) effettuano sempre il salto
  - modificano sempre il PC ad un nuovo valore, detto **target address** (diverso, in generale, da quello dell'istruzione successiva)
  - Vantaggio: essendo un'istruzione più semplice (non deve specificare quali registri paragonare, né quale tipo di di paragone) una jump può dedicare più spazio al target address
- Il MIPS prevede due tipi di istruzioni jump :
  - « **Jump** »
    il target address è codificato nell'istruzione stessa
    (come un immediate, ma con più bits)
  - « **Jump register** »
    il target address è il valore di un registro a scelta.
    (La vedremo la prossima lezione)

- 154 -

154

Istruzione MIPS "jump"

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

Jump (always)	5592405
OPCODE	*ADDRESS*

Tradotta in assembly MIPS: `J 5592405`

A parole: *« La prossima istruzione è quella specificata »*

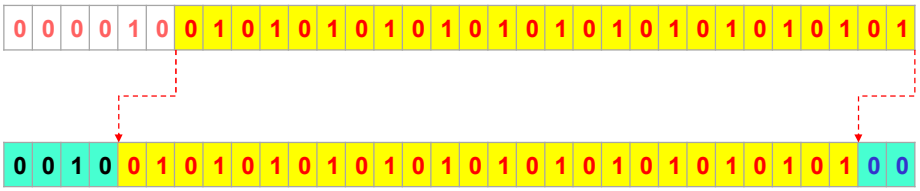
- 155 -

155

Effetti dell'istruzione "Jump"

PC corrente:

Istruzione (Jump):



PC dopo il jump:

$$PC = (PC \ \& \ 0xf0000000) \mid (target << 2);$$

156

Salti in linguaggio macchina MIPS

Conseguenza: l'istruzione Jump …

- non può modificare i primi 4 bits del PC
  - per es, una jump all'indirizzo `0xC-------` può saltare solo ad un'altra istruzione di indirizzo `0xC-------`
  - In altre parole non può saltare fuori dal suo «blocco» di istruzioni
- salta sempre ad istruzioni allineate al word
  - ma è ok, tanto sarebbe un errore fare altrimenti.

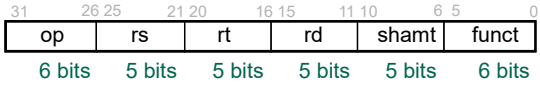
l'insieme delle word in RAM ad indirizzi che iniziano per una lettera, per es, `0xC`

157

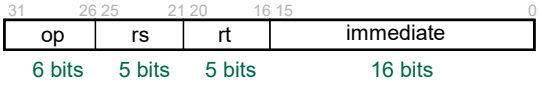
Sommario: i tre possibili formati delle istruzioni MIPS

- Il campo OPCODE (OP) determina come interpretare i campi successivi (quanti sono, e quanto grandi sono).

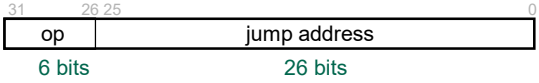
- **formato «R»** (registro)



- **formato «I»** (immediate)



- **formato «J»** (jump)

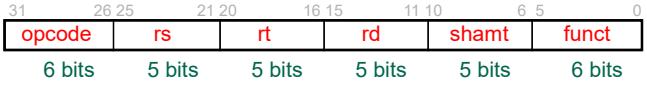


- 158 -

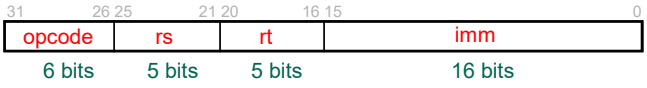
158

Sommario: i tre possibili formati delle istruzioni MIPS

R-type:



I-type:



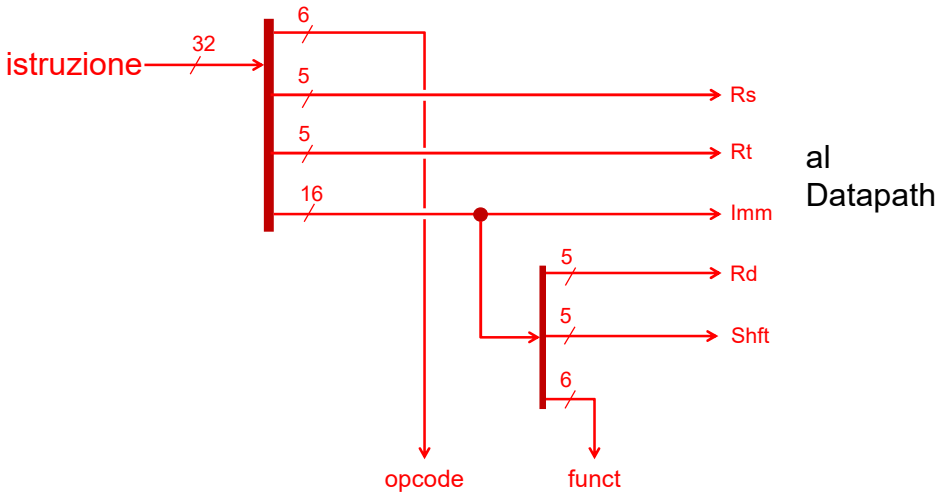
J-type:

- opcode: codice dell'operazione (determina anche il formato!)
- rs, rt, rd: indici dei registri degli operandi / del risultato
- shamt (shift amount): di quanto shiftare (a dx o sx) il risultato
- funct: quale varianti dell'operazione indicata da op
- imm: valore immediato (operando, oppure lunghezza del salto)
- jaddr: indirizzo (address) di destinazione di un salto

- 159 -

159

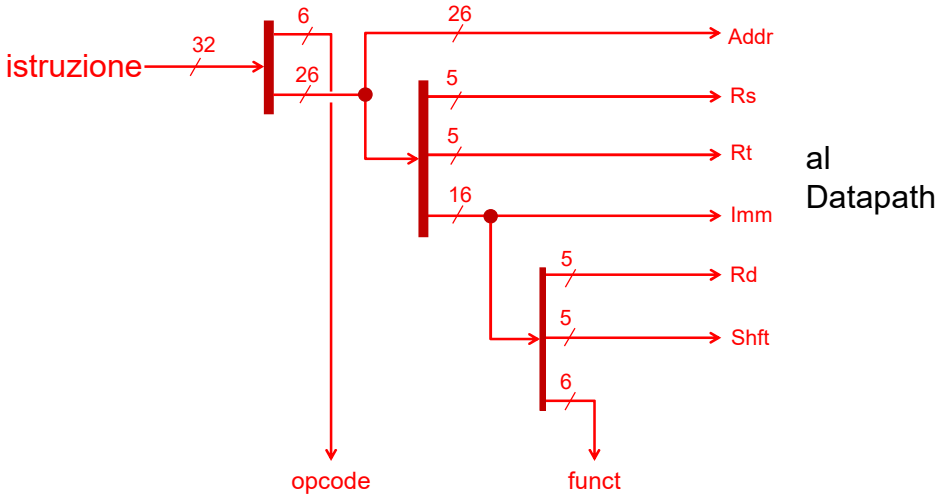
Suddividere l'istruzione MIPS in campi: finora



- 160 -

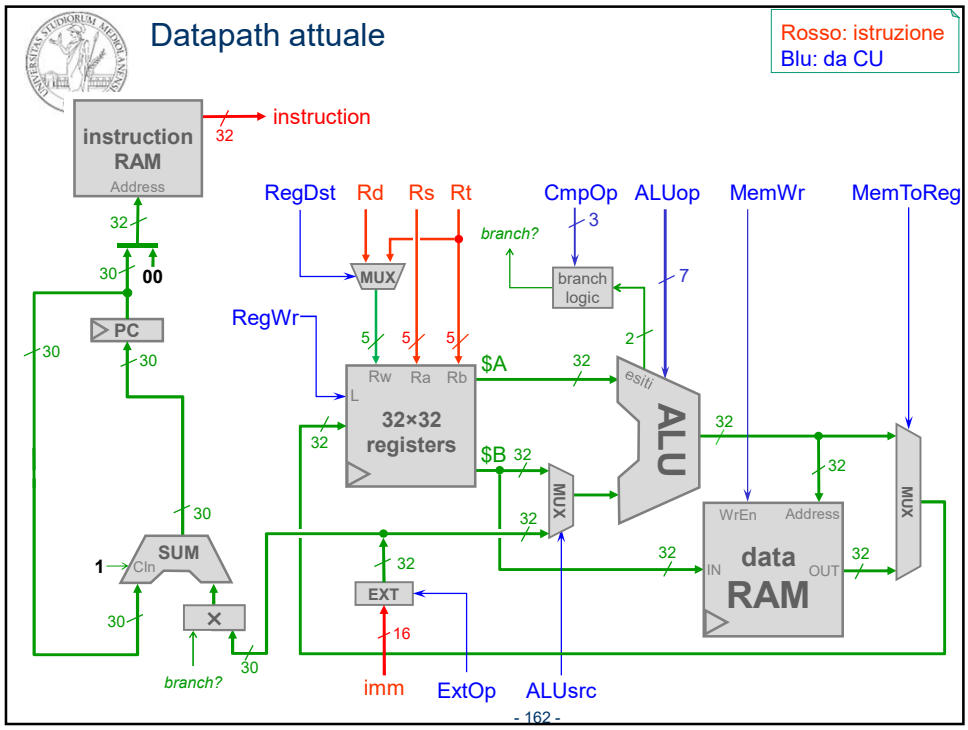
160

Suddividere l'istruzione MIPS in campi: con il supporto per istruzioni di tipo "J"

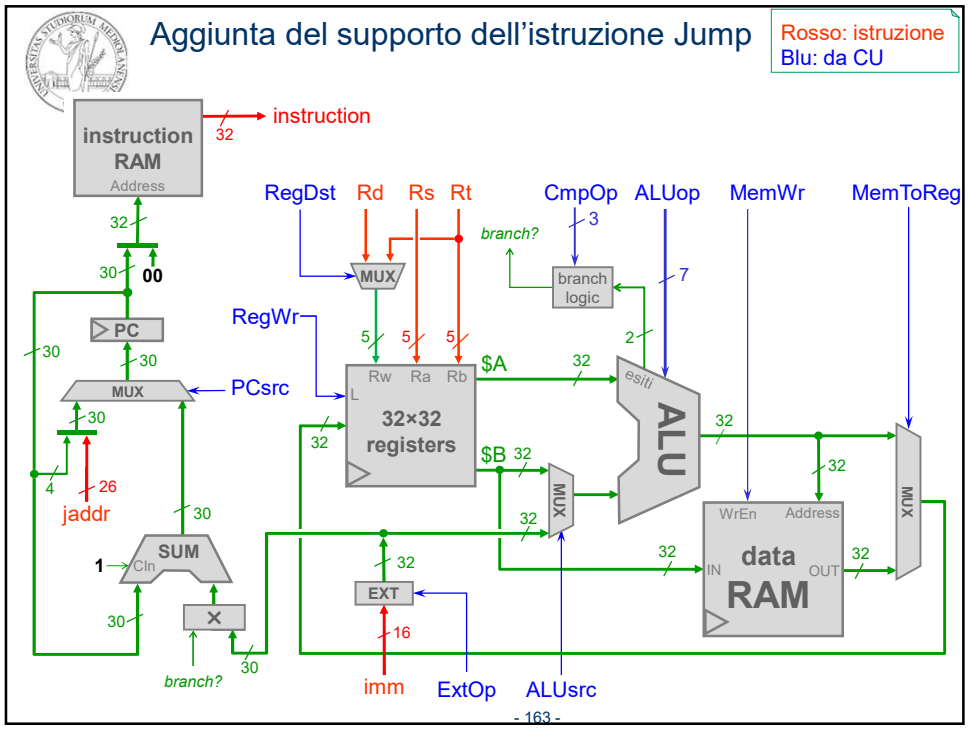


- 161 -

161


Datapath attuale
Rosso: istruzione
Blu: da CU
instruction RAM
Address
instruction
32
RegDst
Rd
Rs
Rt
CmpOp
ALUop
MemWr
MemToReg
branch?
3
MUX
branch logic
7
RegWr
5
5
5
2
PC
30
00
$A
32
Rw
Ra
Rb
L
32×32 registers
esiti
ALU
$B
32
MUX
WrEn
Address
data RAM
IN
OUT
SUM
1
Cin
EXT
16
imm
ExtOp
ALUsrc
branch?
- 162 -

162


Aggiunta del supporto dell'istruzione Jump
Rosso: istruzione
Blu: da CU
instruction RAM
Address
instruction
32
RegDst
Rd
Rs
Rt
CmpOp
ALUop
MemWr
MemToReg
branch?
3
MUX
branch logic
7
RegWr
PCsrc
5
5
5
2
PC
30
00
MUX
$A
32
Rw
Ra
Rb
L
32×32 registers
esiti
ALU
$B
32
MUX
WrEn
Address
data RAM
IN
OUT
jaddr
4
26
SUM
1
Cin
EXT
16
imm
ExtOp
ALUsrc
branch?
- 163 -

163

Nota allo schema precedente

- Nel datapath, il campo jaddr (un campo dell'istruzione di 26 bit) viene completato con i 4 bit più significative del PC Corrente
- Il risultato (di 30 bit) viene immesso in un Multiplexer, che "decide" quale valore memorizzare nel PC (di 30 bit):
  - Il nuovo valore ottenuto dal jaddr, *oppure,*
  - Il valore "standard" del PC corrente, incrementato di 1, più l'eventuale incremento per la branch?
- Un nuovo, apposito bit di comando dalla CU, PCsrc («PC source»), pilota questo multiplexer.
- In ogni caso, il registro PC viene completato con due bit «00» a destra, per ottenere l'indirizzo della prossima istruzione