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

Lezione 12:

CPU e linguaggio MIPS

Parte E: fetch e branch

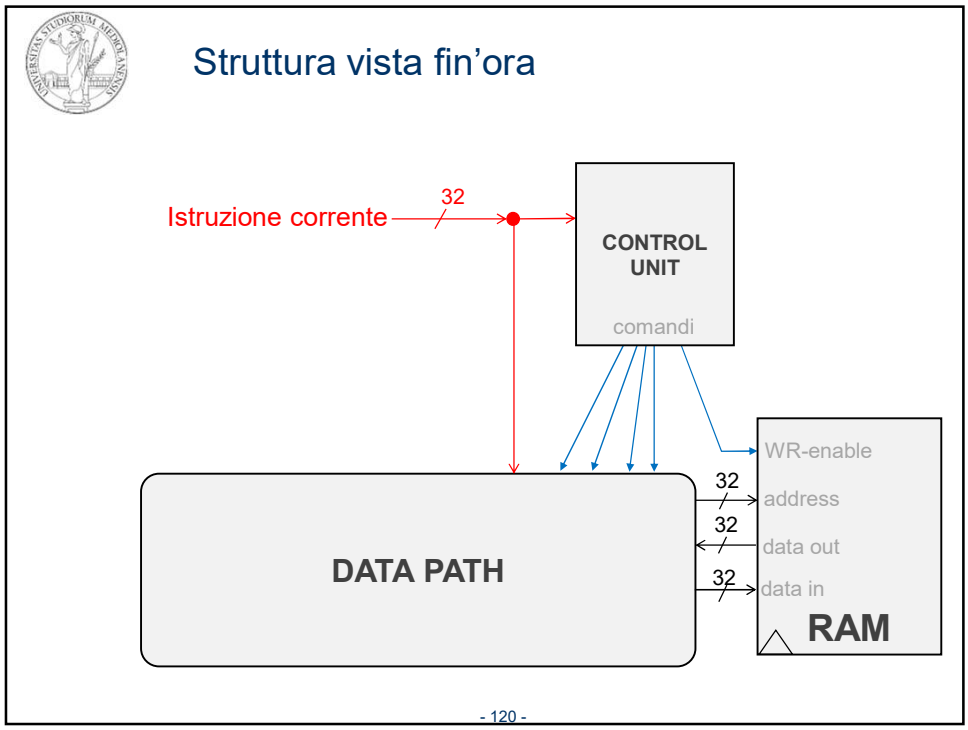
Marco Tarini

118

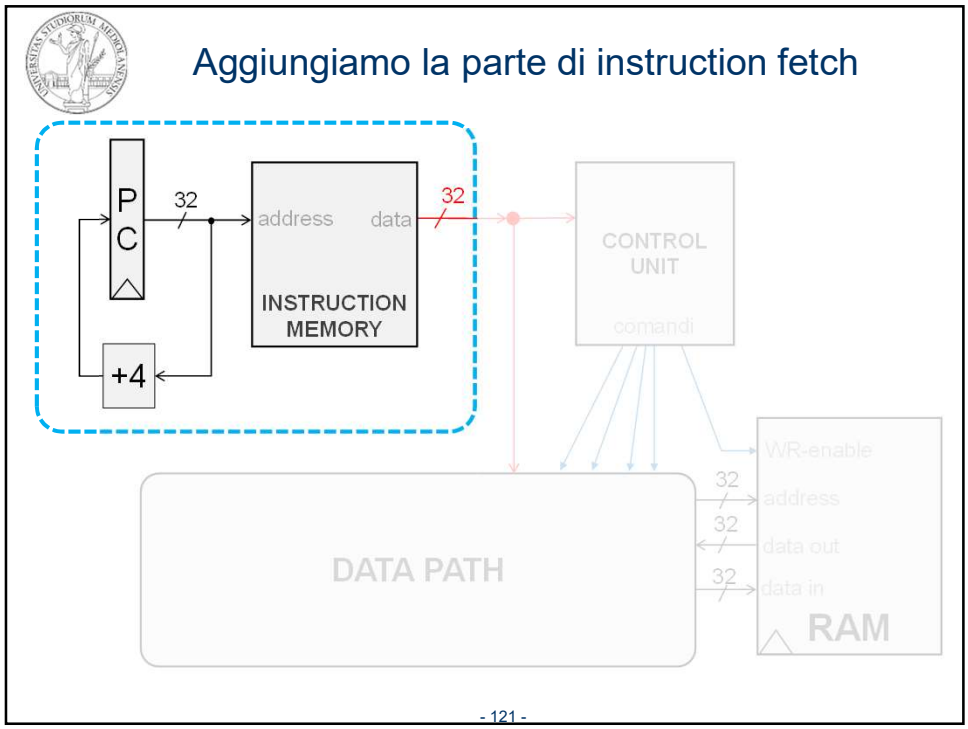
Next: la parte «fetch» del ciclo «fetch and execute»

- Normalmente, a prescindere da quale istruzione viene eseguita, la CPU deve passare alla istruzione successiva del programma
  - Salvo eccezioni, che vediamo più avanti
- Per far questo bisogna incrementare il Program Counter (PC), cioè il registro che memorizza l'indirizzo in RAM dell'istruzione da leggere
  - Come tutti i registri, il PC assumerà il nuovo valore (cioè quello precedente, ma incrementato), al termine del ciclo di clock (sincronia in scrittura)
  - Questo è parte della fase detta «write back» del ciclo di clock, quella in cui tutti i valori computati nel corrente ciclo sono finalmente memorizzati da una parte o l'altra (RAM, registri, etc)
- Come sappiamo, ogni ciclo di clock inizia con la lettura dell'istruzione corrente dalla memoria, alla locazione contenuta nel PC
  - fase fetch

119


Struttura vista fin'ora
Istruzione corrente
32
CONTROL UNIT
comandi
WR-enable
32
address
32
data out
32
data in
DATA PATH
RAM
- 120 -

120


Aggiungiamo la parte di instruction fetch
P C
32
address
data
32
INSTRUCTION MEMORY
+4
CONTROL UNIT
comandi
WR-enable
32
address
32
data out
32
data in
DATA PATH
RAM
- 121 -

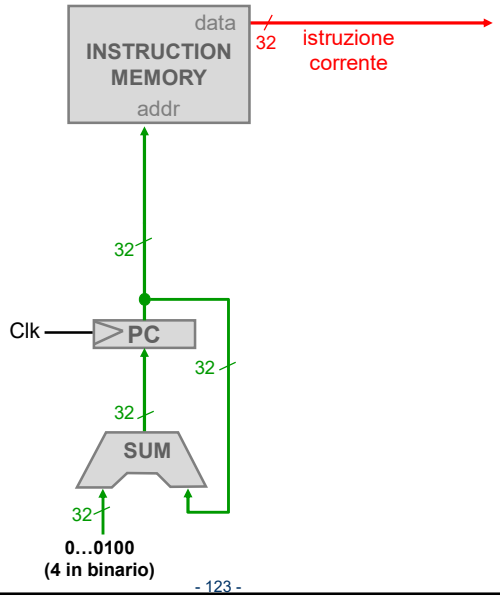
121

Il Program Counter (PC)

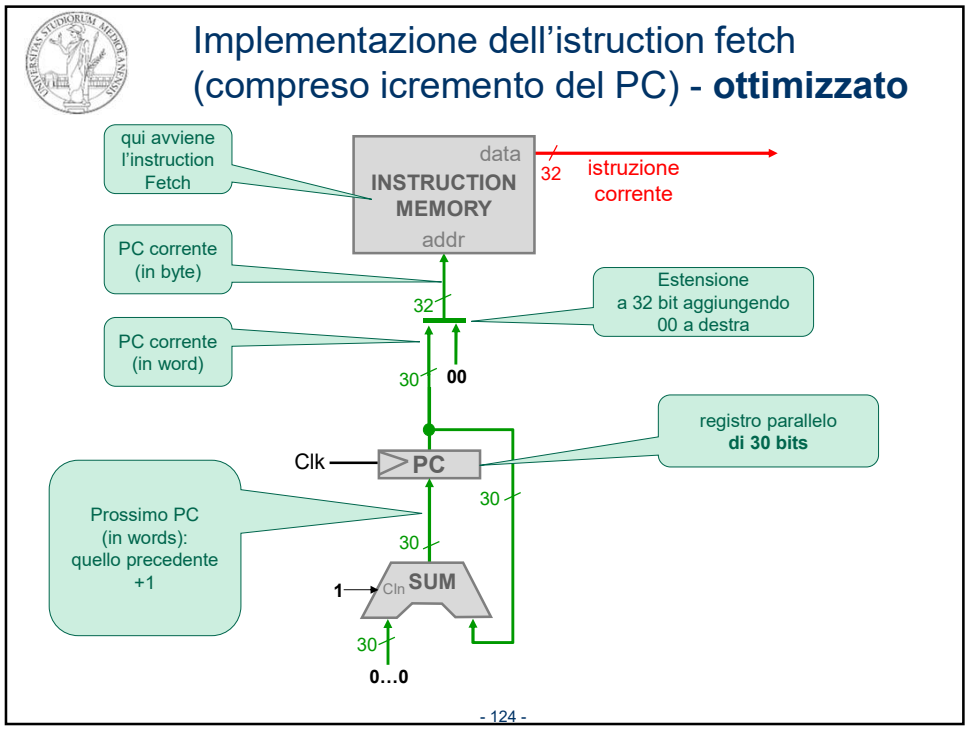
- **Program Counter** = l'indirizzo dell'istruzione (1word) di cui fare il *fetch*
  - Viene incrementato di 4 byte (1word) al termine di ogni istruzione
  - Il suo incremento determina l'inizio del ciclo successivo (lettura, decodifica ed esecuzione dell'istruzione successiva)
- Il PC è gestito internamente dalla CPU
  - non è esposto all'utente: l'utente (il programmatore MIPS) non può leggerlo o scriverlo direttamente; non è necessario che se ne occupi (direttamente)
- Ottimizzazione (scelta progettuale): usiamo un registro PC di soli 30 bit che memorizza i 30 bit più significativi del Program Counter (di 32 bits). Conseguenze:
  - Prima di usare il PC come indirizzo per accedere alla memoria, dovremo appendere due bit 00 a destra, per ottenere l'indirizzo reale (in byte)
  - Per passare all'istruzione successiva, devo incrementare il PC di 1 (e non più di 4) – vantaggioso!

122

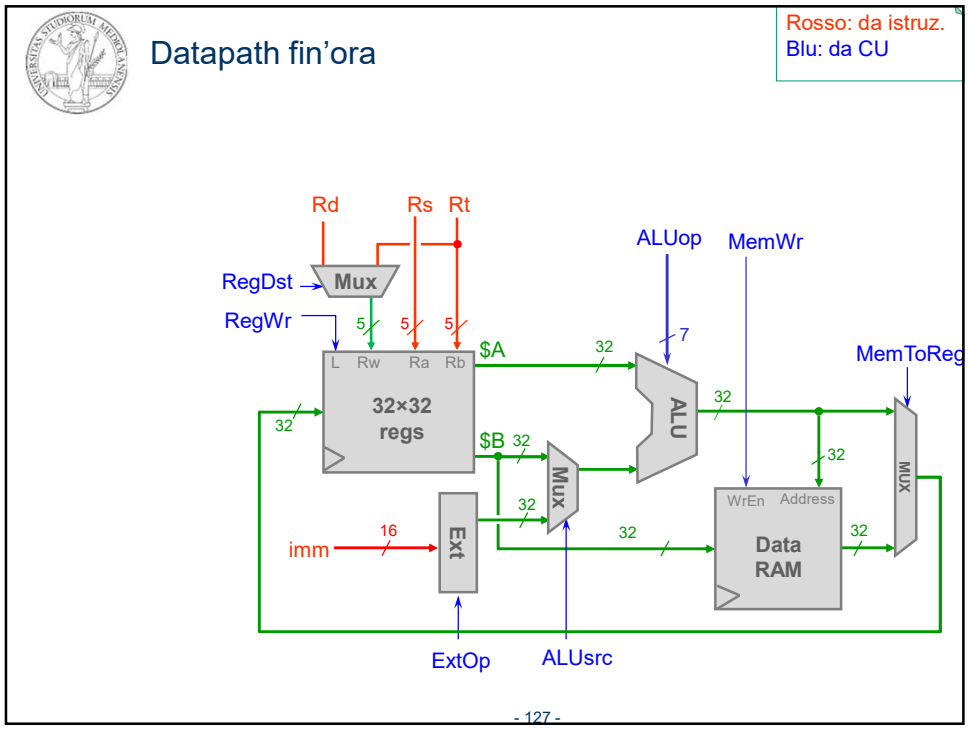
Implementazione dell'istruction fetch (compreso icremento del PC)



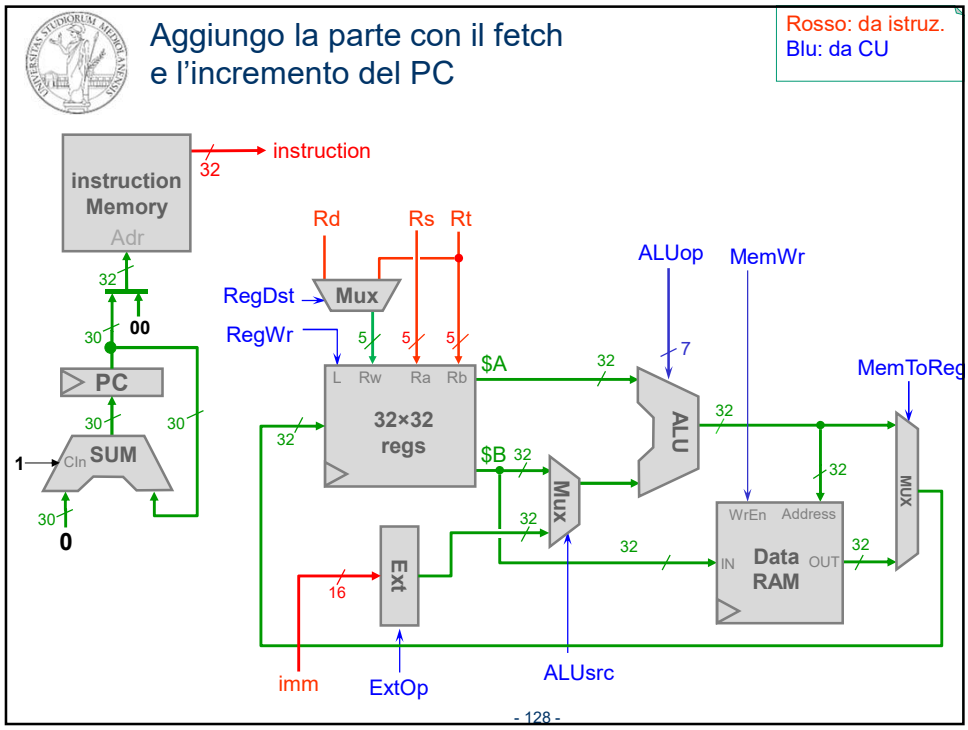
123


Implementazione dell'istruction fetch
(compreso icremento del PC) - ottimizzato
qui avviene l'instruction Fetch
data
INSTRUCTION MEMORY
addr
32
istruzione corrente
PC corrente (in byte)
32
Estensione a 32 bit aggiungendo 00 a destra
PC corrente (in word)
30
00
registro parallelo di 30 bits
Clk
PC
30
Prossimo PC (in words): quello precedente +1
30
1
Cin
SUM
30
0…0
- 124 -

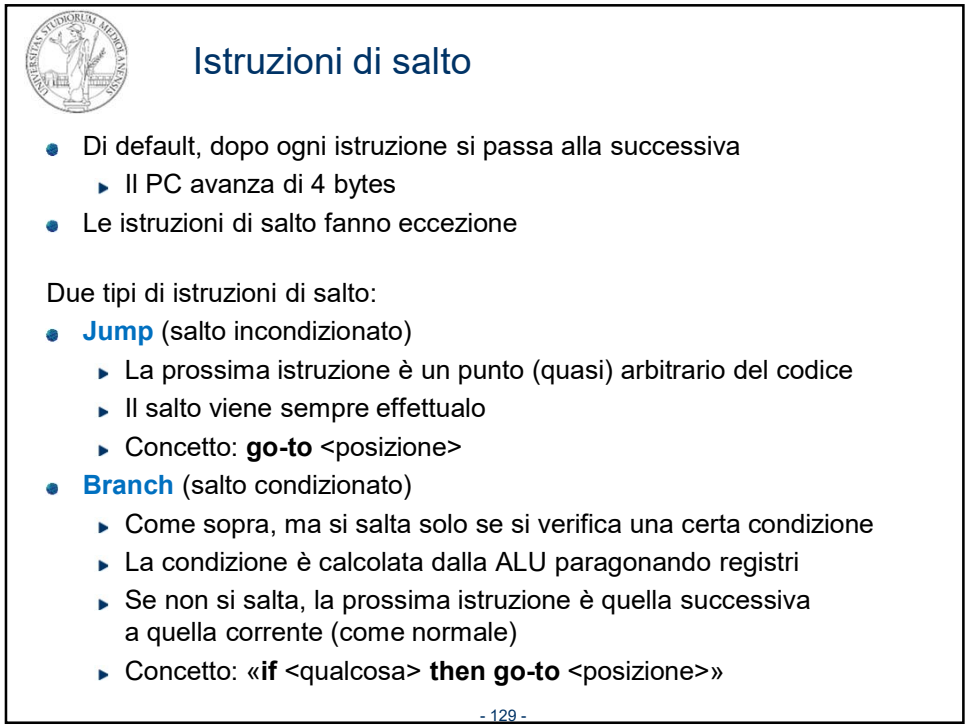
124


Datapath fin'ora
Rosso: da istruz.
Blu: da CU
Rd
Rs
Rt
RegDst
Mux
RegWr
5
5
5
L
Rw
Ra
Rb
32×32 regs
32
$A
32
$B
32
ALUop
7
ALU
MemWr
MemToReg
32
32
32
Mux
32
imm
16
Ext
32
WrEn
Address
Data RAM
32
MUX
ExtOp
ALUsrc
- 127 -

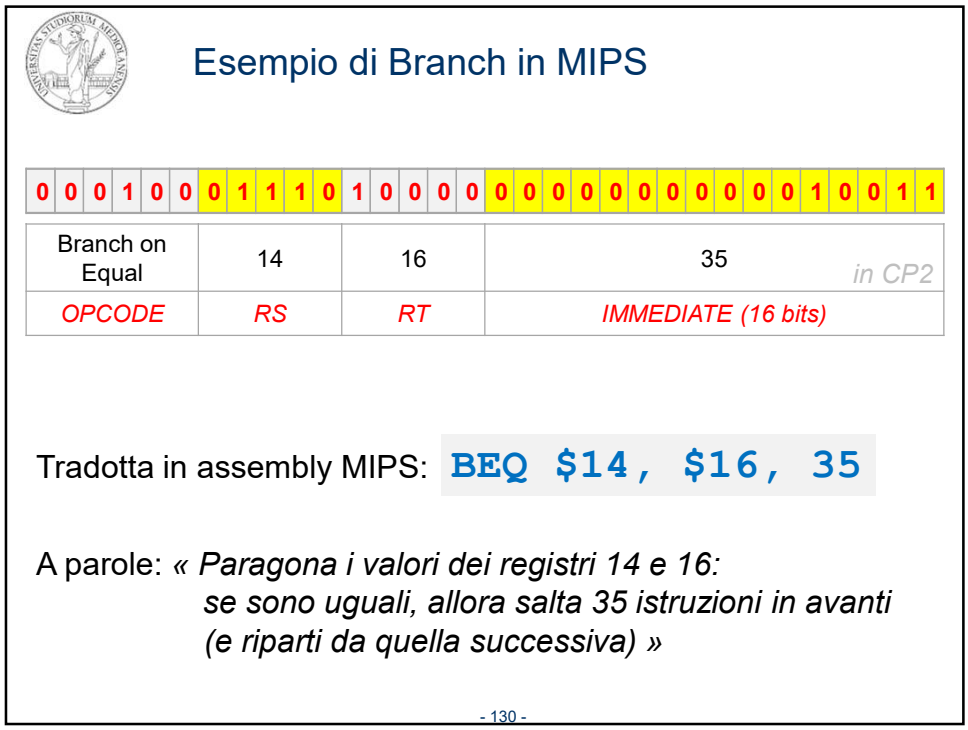
127


Aggiungo la parte con il fetch
e l'incremento del PC
Rosso: da istruz.
Blu: da CU
instruction Memory
Adr
instruction
32
Rd
Rs
Rt
RegDst
Mux
RegWr
ALUop
MemWr
MemToReg
PC
00
30
SUM
Cin
1
0
$A
$B
32×32 regs
L
Rw
Ra
Rb
ALU
Mux
WrEn
Address
Data RAM
IN
OUT
Ext
16
imm
ExtOp
ALUsrc
- 128 -

128


Istruzioni di salto
Di default, dopo ogni istruzione si passa alla successiva
Il PC avanza di 4 bytes
Le istruzioni di salto fanno eccezione
Due tipi di istruzioni di salto:
Jump (salto incondizionato)
La prossima istruzione è un punto (quasi) arbitrario del codice
Il salto viene sempre effettualo
Concetto: go-to <posizione>
Branch (salto condizionato)
Come sopra, ma si salta solo se si verifica una certa condizione
La condizione è calcolata dalla ALU paragonando registri
Se non si salta, la prossima istruzione è quella successiva a quella corrente (come normale)
Concetto: «if <qualcosa> then go-to <posizione>»
- 129 -

129



Esempio di Branch in MIPS

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

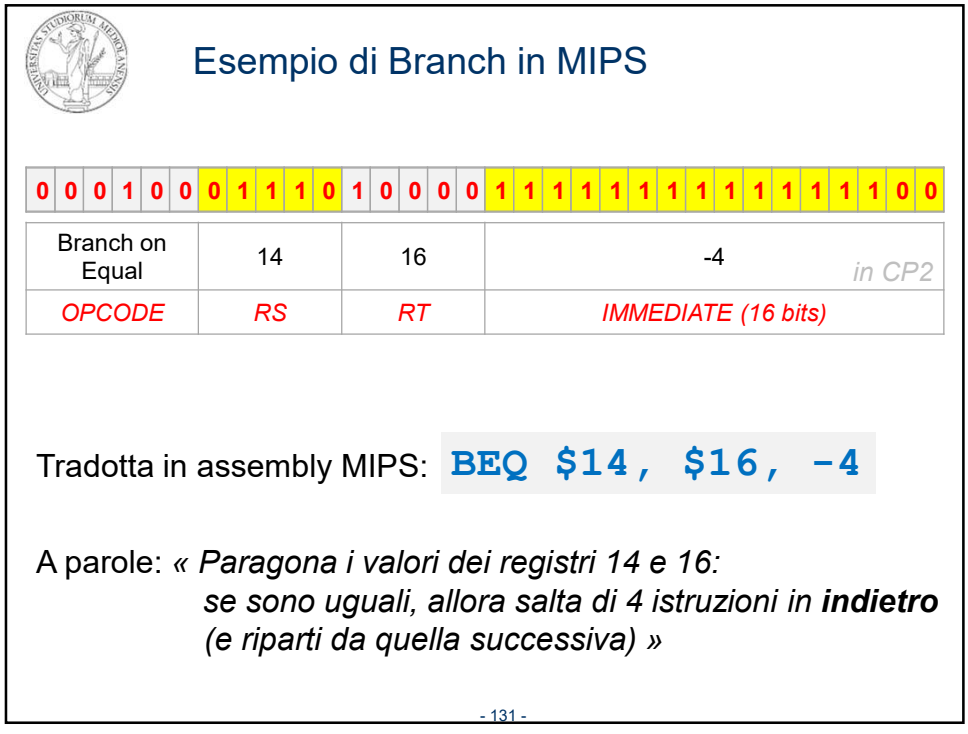
Branch on Equal	14	16	35 in CP2
OPCODE	*RS*	*RT*	*IMMEDIATE (16 bits)*

Tradotta in assembly MIPS: **`BEQ $14, $16, 35`**

A parole: *« Paragona i valori dei registri 14 e 16:
se sono uguali, allora salta 35 istruzioni in avanti
(e riparti da quella successiva) »*

- 130 -

130



Esempio di Branch in MIPS

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

Branch on Equal	14	16	-4 in CP2
OPCODE	*RS*	*RT*	*IMMEDIATE (16 bits)*

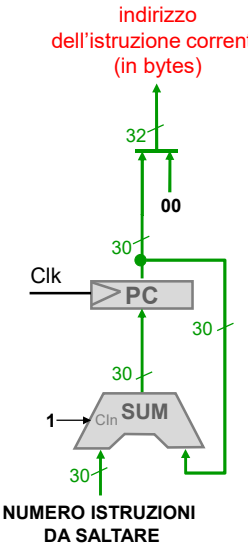
Tradotta in assembly MIPS: **`BEQ $14, $16, -4`**

A parole: *« Paragona i valori dei registri 14 e 16:
se sono uguali, allora salta di 4 istruzioni in **indietro***
(e riparti da quella successiva) »

- 131 -

131

Aggiornamento del PC (computo dell'address della prossima istr.)

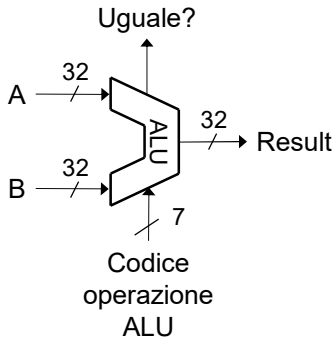


- 132 -

132

I bit di esito della ALU (ripasso)

- Oltre alla parola «risultato», una ALU presenta in output bit di **esito**
- Sono l'output della ALU per alcune determinate operazioni
- Ad esempio, quando si chiede alla ALU un operazione di confronto, il bit di esito «uguaglianza» vale
  - 1 se gli input A e B sono uguali
  - 0 altrimenti

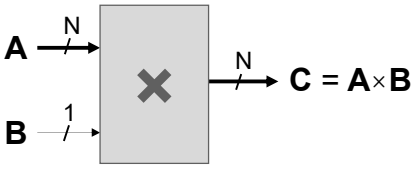


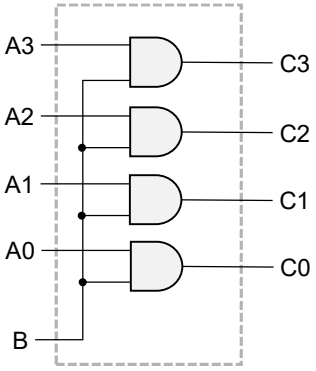
- 133 -

133

Un blocco funzionale per moltiplicazione…
…quando il secondo termine ha 1 bit ☺

Realizzazione (banale!):
(qui, per N = 4)





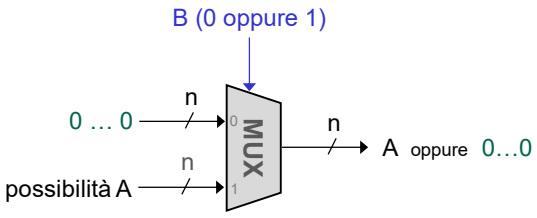
(è un circuito che annulla A quando il bit B vale 0, altrimenti lascia A inalterato in output)

- 134 -

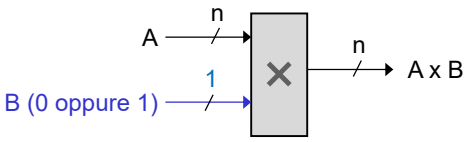
134

«Moltiplicatori» N per 1

- E' la versione ottimizzata di un MUX a 2 vie quando una delle due alternative è costituita da tutti zero

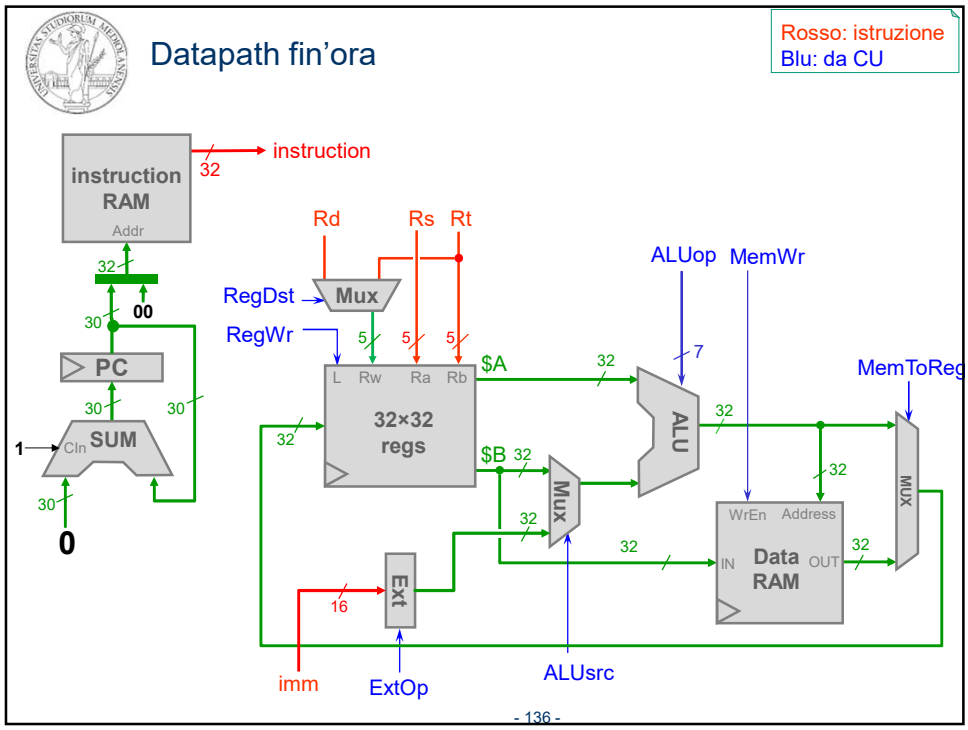


equivalente a:

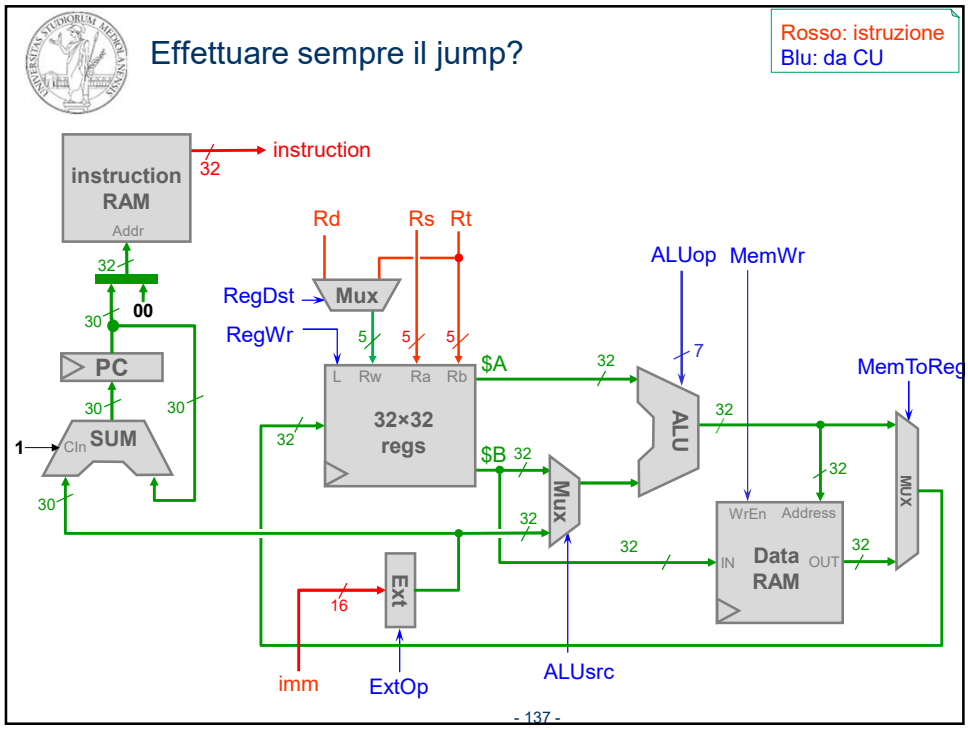


- 135 -

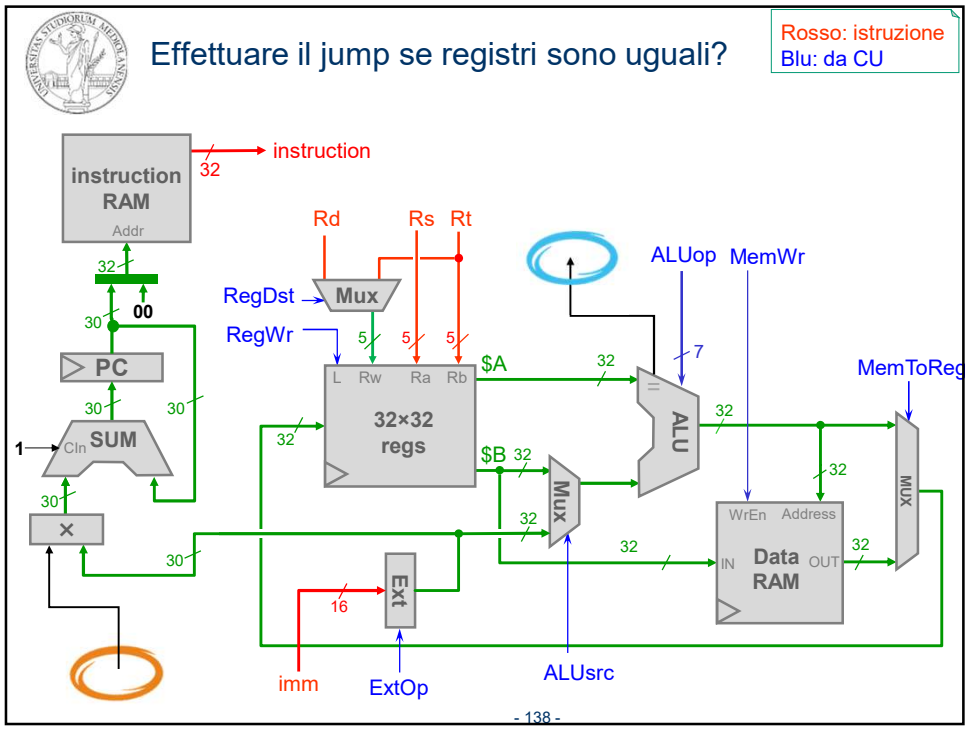
135


Datapath fin'ora
Rosso: istruzione
Blu: da CU
instruction RAM
Addr
instruction
32
Rd
Rs
Rt
RegDst
Mux
RegWr
5
5
5
ALUop
MemWr
7
MemToReg
L
Rw
Ra
Rb
32×32 regs
$A
$B
32
ALU
Mux
MUX
WrEn
Address
Data RAM
IN
OUT
PC
00
30
SUM
Cin
1
0
Ext
16
imm
ExtOp
ALUsrc
- 136 -

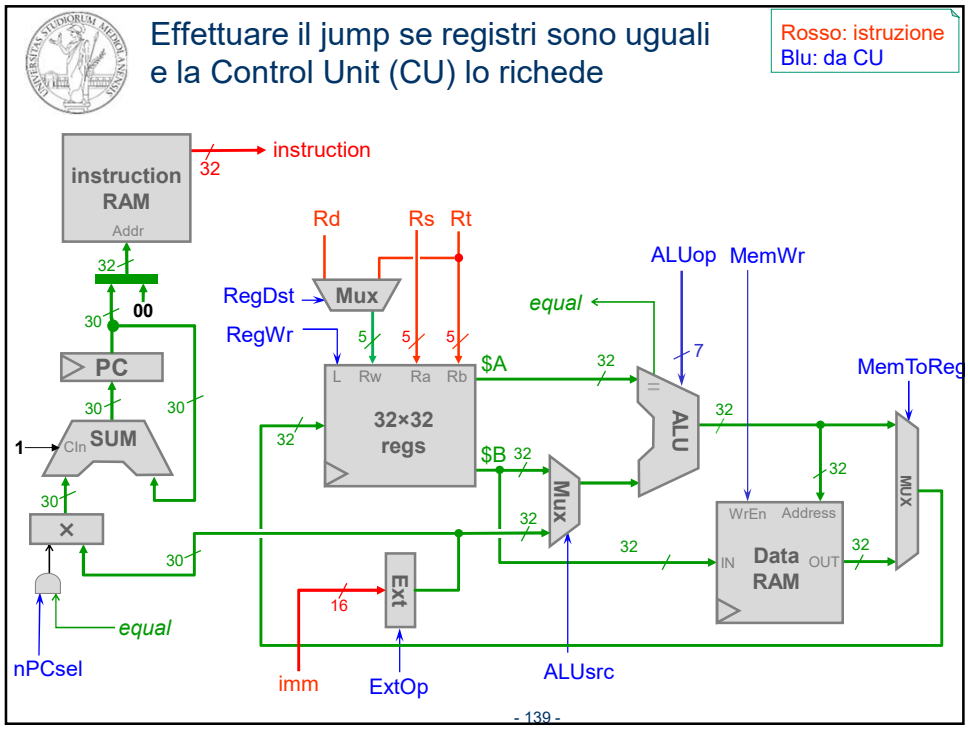
136


Effettuare sempre il jump?
Rosso: istruzione
Blu: da CU
instruction RAM
Addr
instruction
32
Rd
Rs
Rt
RegDst
Mux
RegWr
5
5
5
ALUop
MemWr
7
MemToReg
L
Rw
Ra
Rb
32×32 regs
$A
$B
32
ALU
Mux
MUX
WrEn
Address
Data RAM
IN
OUT
PC
00
30
SUM
Cin
1
Ext
16
imm
ExtOp
ALUsrc
- 137 -

137


Effettuare il jump se registri sono uguali?
Rosso: istruzione
Blu: da CU
instruction RAM
Addr
instruction
32
Rd
Rs
Rt
RegDst
Mux
RegWr
ALUop
MemWr
MemToReg
PC
00
SUM
Cin
1
32×32 regs
L
Rw
Ra
Rb
$A
$B
ALU
Mux
Ext
16
imm
ExtOp
ALUsrc
WrEn
Address
Data RAM
IN
OUT
MUX
- 138 -

138


Effettuare il jump se registri sono uguali
e la Control Unit (CU) lo richede
Rosso: istruzione
Blu: da CU
instruction RAM
Addr
instruction
32
Rd
Rs
Rt
RegDst
Mux
RegWr
equal
ALUop
MemWr
MemToReg
PC
00
SUM
Cin
1
32×32 regs
L
Rw
Ra
Rb
$A
$B
ALU
Mux
Ext
16
imm
ExtOp
ALUsrc
nPCsel
equal
WrEn
Address
Data RAM
IN
OUT
MUX
- 139 -

139

Implementazione del supporto HW per l'istruzione "branch on equal": note

- Le branch sono istruzioni di tipo «I»
  - hanno due campi «registri» che in questo caso indentificano gli indici dei due registri da paragonare, e un campo immediate di 16 bit, che in questo caso riporta il numero di istruzioni da saltare
- Il valore immediate, esteso a 32 bit, viene immesso (anche) come addendo nell'addizionatore che incrementa il PC
  - L'estensione, in questo caso, è IN SEGNO, perché il numero di istruzioni da saltare è espresso in CP2 (può essere neg o pos)
  - Dopo l'estensione vanno anche decurtati i due bit più significativi (lasciandone solo 30), perché il nostro PC è di 30 bit
- Questo valore di incremento del PC viene però azzerato se l'**esito** "uguaglianza" dalla ALU (che paragona i due registri $rs e $rt) vale 0
  - In questo caso, il PC viene incrementato solo di 1 (attraverso cin)
- L'esito viene messo in AND con un apposito comando di controllo della Control Unit, nPCsel (next PC selector) che se vale 0 annulla il salto
  - questo comando vale 1 solo nelle istruzioni di branch

- 140 -

140

Altra pseudo-istruzione MIPS di branch (già supportata dall'HW visto)

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

Branch on equal	14	0	35 *in CP2*
OPCODE	*RS*	*RT*	*IMMEDIATE (16 bits)*

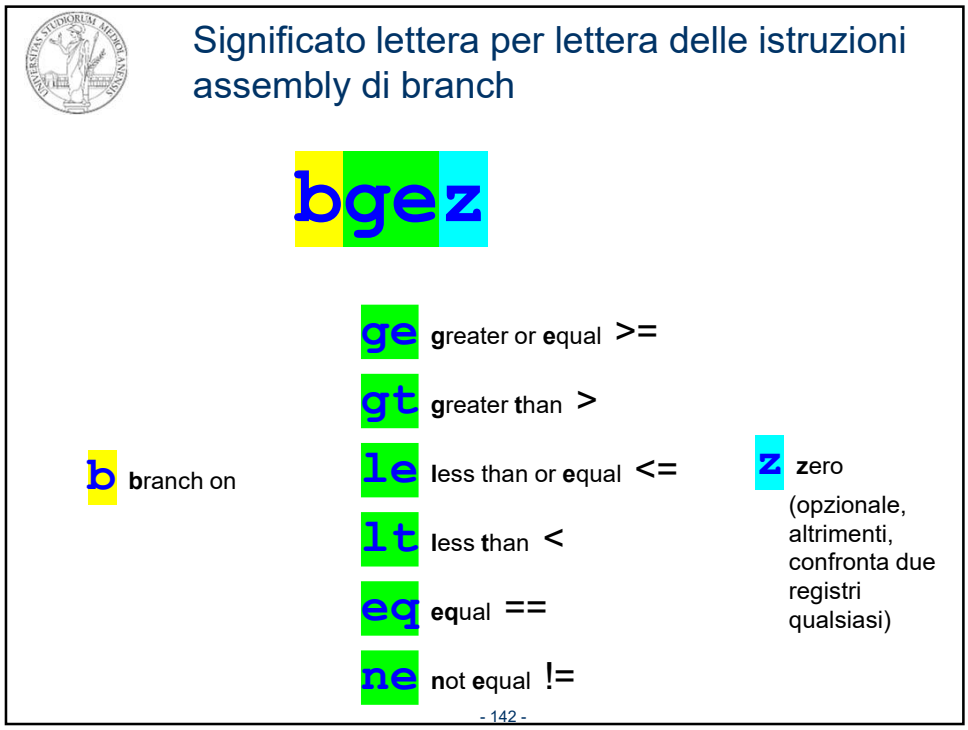
Tradotta in assembly MIPS: `BEQ $14, $0, 35`

oppure: `BEQZ $14, 35`

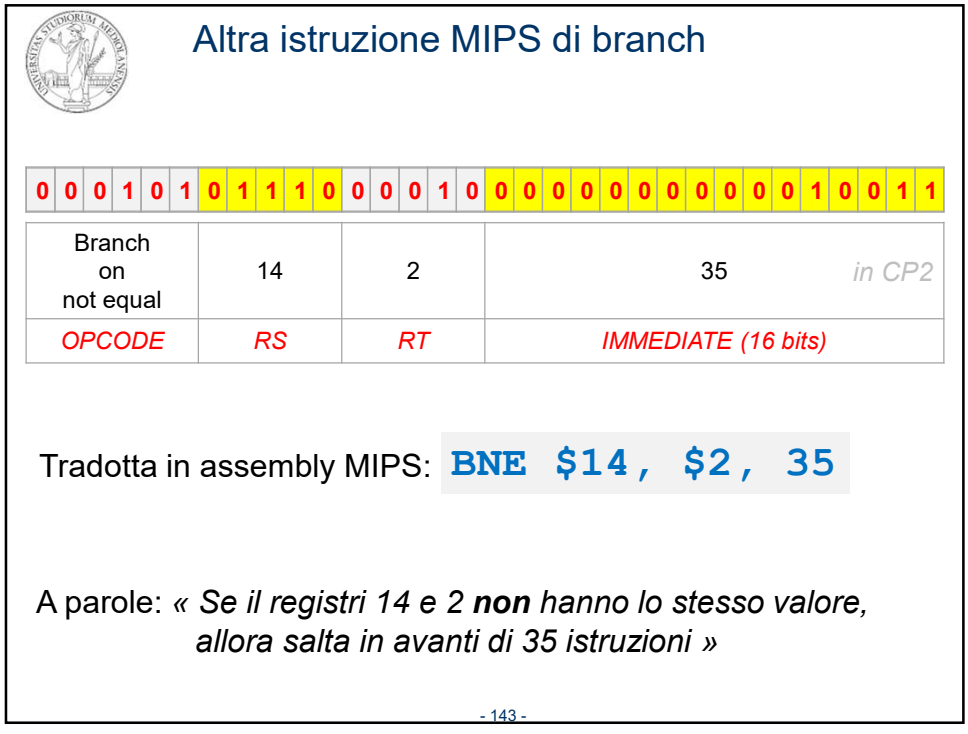
A parole: *« Se il valore del registro 14 è proprio 0, allora salta in avanti di 35 istruzioni »*

- 141 -

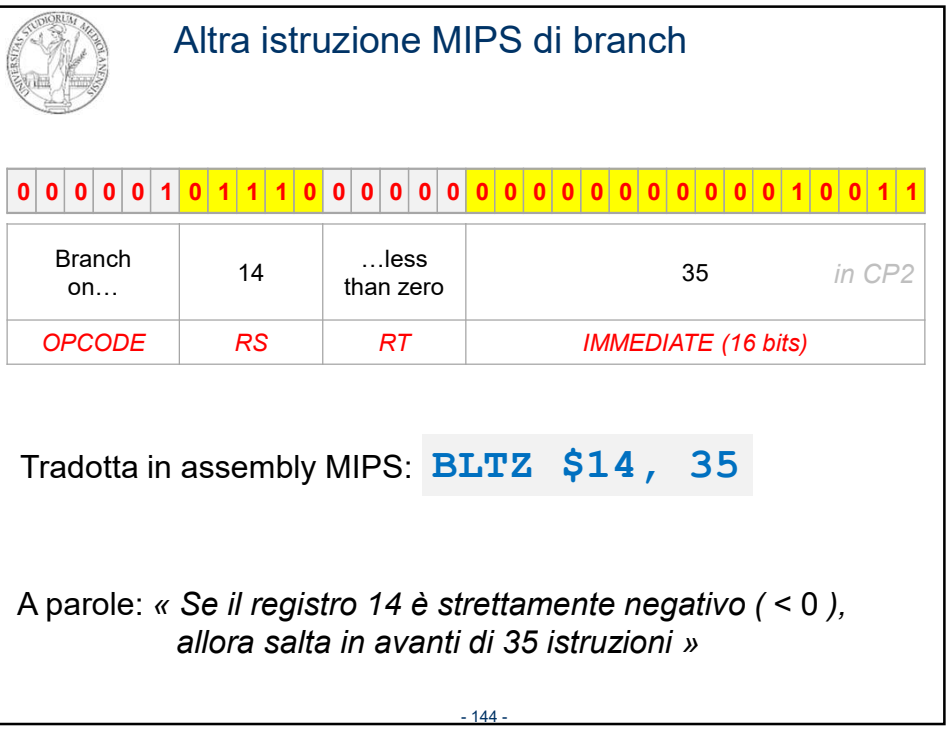
141


Significato lettera per lettera delle istruzioni assembly di branch
bgez
b branch on
ge greater or equal >=
gt greater than >
le less than or equal <=
lt less than <
eq equal ==
ne not equal !=
z zero
(opzionale, altrimenti, confronta due registri qualsiasi)
- 142 -

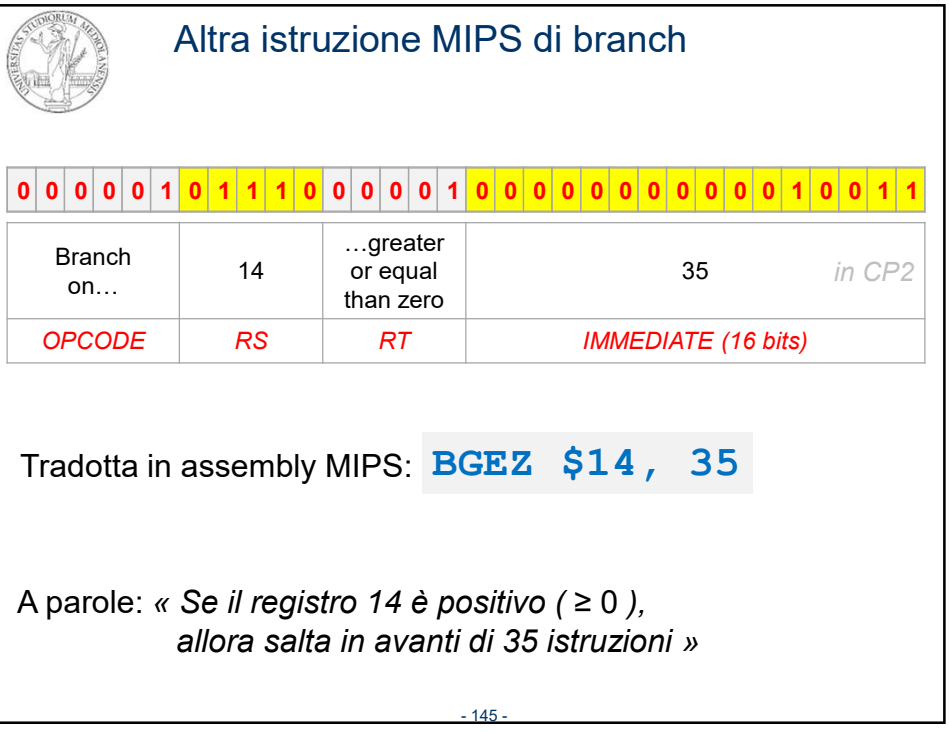
142


Altra istruzione MIPS di branch
0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
Branch on not equal
14
2
35
in CP2
OPCODE
RS
RT
IMMEDIATE (16 bits)
Tradotta in assembly MIPS: BNE $14, $2, 35
A parole: « Se il registri 14 e 2 non hanno lo stesso valore, allora salta in avanti di 35 istruzioni »
- 143 -

143


Altra istruzione MIPS di branch
0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
Branch on…
14
…less than zero
35
in CP2
OPCODE
RS
RT
IMMEDIATE (16 bits)
Tradotta in assembly MIPS: BLTZ $14, 35
A parole: « Se il registro 14 è strettamente negativo (< 0), allora salta in avanti di 35 istruzioni »
- 144 -

144


Altra istruzione MIPS di branch
0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
Branch on…
14
…greater or equal than zero
35
in CP2
OPCODE
RS
RT
IMMEDIATE (16 bits)
Tradotta in assembly MIPS: BGEZ $14, 35
A parole: « Se il registro 14 è positivo (≥ 0), allora salta in avanti di 35 istruzioni »
- 145 -

145

Alcune istruzioni di Branch in assembly MIPS

- Con confronto fra due registri

`beq $ra $rb +20`	branch on *equal*	$ra = $rb
`bne $ra $rb -3`	branch on *not equal*	$ra ≠ $rb
`blt $ra $rb +5`	branch on *less then*	$ra < $rb

- Con confronto fra registro e zero

`bgez $ra +20`	branch on *greater-or-equal zero*	$ra ≥ 0
`bgtz $ra +6`	branch on *greater-than zero*	$ra > 0
`blez $ra 7`	branch on *less-or-equal to zero*	$ra ≤ 0
`bltz $ra 30`	branch on *less-than zero*	$ra < 0

146

Domande ed esercizi

- Quale valore numerico “immediate” deve presentare una istruzione di branch per saltare a… se stessa?
  - Cosa succede all’esecuzione del programma, se il salto viene effettuato?
- Mettiti nei panni della CU e decidi ciascuno dei comandi inviare al datapath per effettuare una istruzione BEQ
- Quante istruzioni si possono saltare, al massimo, in avanti o indietro, con un comando branch?
- Come mai abbiamo bisogno di un addizionatore separato per effettuare l’incremento del PC?
  Non si potrebbe usare l’addizionatore presente della ALU?

- 147 -

147