mercoledì 29 novembre 2017 08:45

DATA FLOW

x[i] y[i]
(a+b) * (c-d)

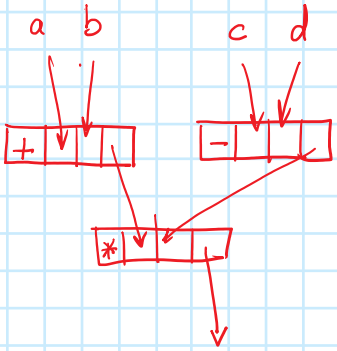
t1 * t2

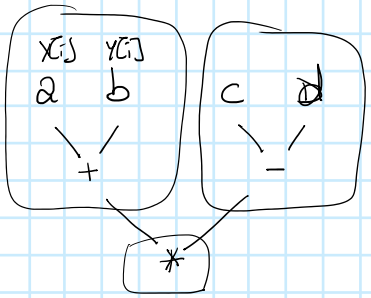
—

—

—

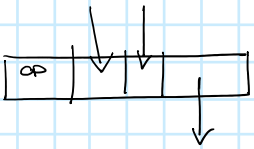
② LOAD x[i]
LOAD y[i]
ADD R1
① SUB c-d R2
③ MUL





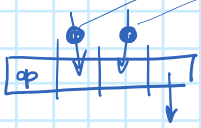
GRAFO DATA FLOW

Istruzioni DF



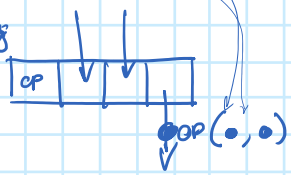
"Calcolo" di un grafo DF

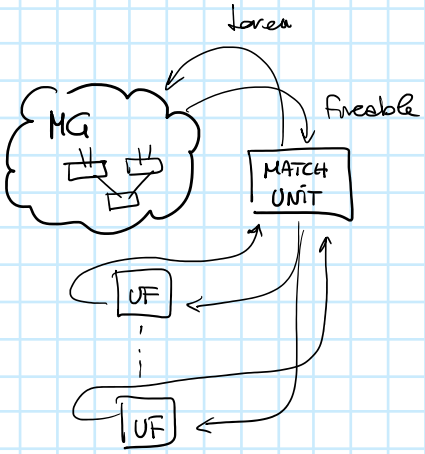
fireable



"tutti gli archi entranti hanno un token"

Regola di firing

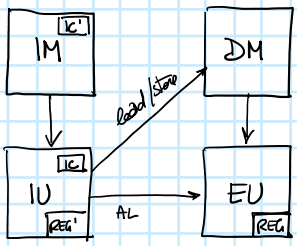




PROCESSORE D-RISC "PIPELINE" (HARVARD)

mercoledì 29 novembre 2017 09:35

Instruction memory
fetch



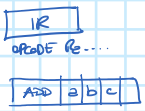
data memory
execute LD/ST

Instruction unit
decode + exec salti

execution unit
execute AL / LD

ADD

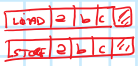
IM → IU



IU decodifica
→ EU
< +, a, b, c >

EU
$R_a + R_b \rightarrow R_c$

LOAD STORE
IM → IU

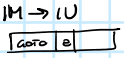


IU decodifica
$R_a' + R_b'$
< "LD", Ind, R_c' > → DM
< "ST", Ind, val >
→ EU < "ld", c >

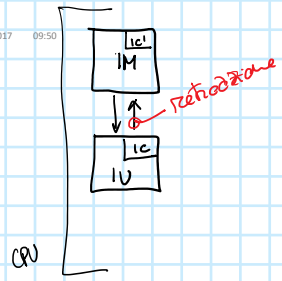
DM esegue op richiesta
rac LD invio a EU
< val >

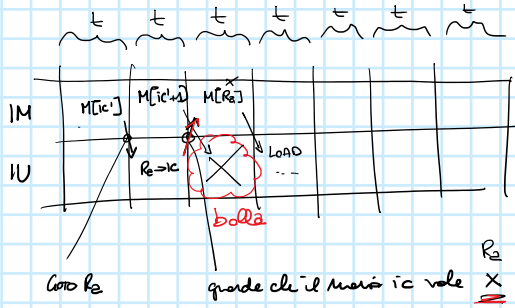
EU
IU → < "ld", c >
EU → < val >
} val → R_c

SALTI IM → IU



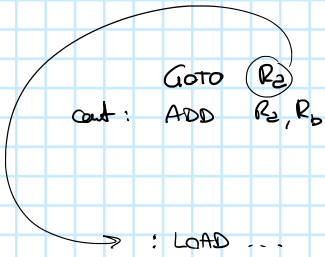
IU
$R_a \rightarrow$ IC
modo ad IM il valore di IC

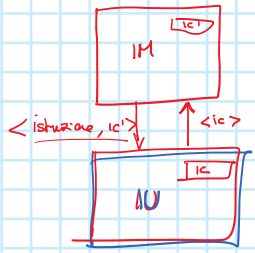




← tempo di servizio (o latenza, tanto IM ed IU sono sequenziali) di IM ed IU

GOTO R2 — grande che il mio ic vale X R2





while true {
se IU mi vuole new IC
then vuole dalla IU M[new IC]
new IC → ic'
else vuole dalla IU M[ic']
ic'++
}

```
while (true) {
...
if ( <istruzione, ic'>.IC' == IC)
then (decodifica <istruzione, ic'>.istruzione)
else nop
```

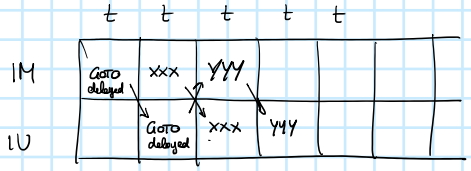
delayed branch

```
GOTO R2, delayed=1
XXX
```

come se

```
XXX
GOTO R2
```

R2 → YYY

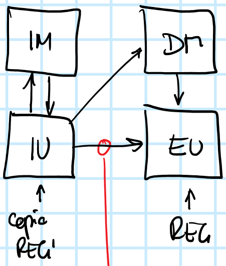


stessa bolla

esse "Bernstein"

XXX non altera registri letti da istruzione di salto

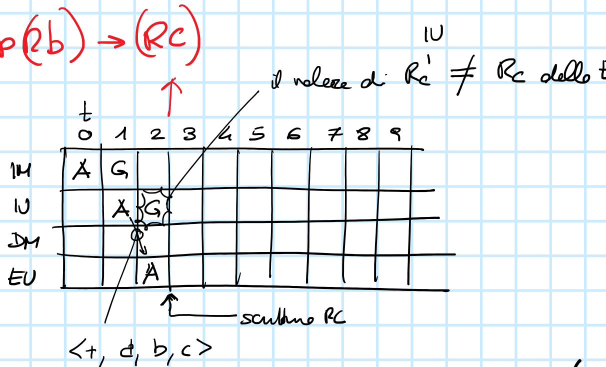
mercoledì 29 novembre 2017 10:21



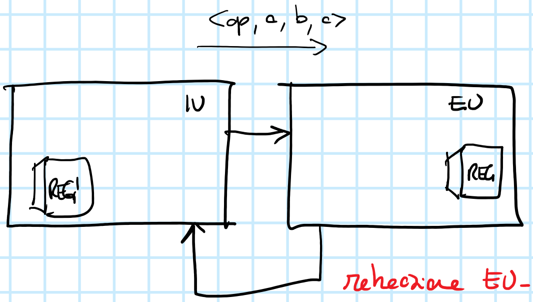
ADD Ra, Rb, Rc
GOTO Rc
XXX

YYY

<op, a, b, c> ⇒ EU: fai (Ra) op (Rb) → (Rc)



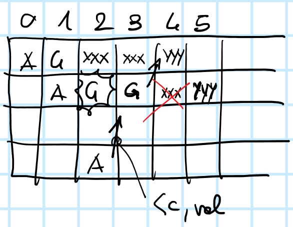
evidenzia il problema

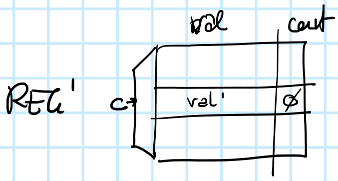


retroazione EU-IU

nuovi valori dei reg

<c, val>





sse IU manda <op, - - c> ad EU

cont (c) ++

quando IU legge Rc

se cont (Rc) ≠ 0 ⇒ si blocca

ogni volta che arriva < val, x > dal canale retroazione EU → IU

val → Reg'[x]. valore

Reg'[x]. cont --

ADD Ra, Rb, Rc
GOTO Rc
XXX

YYY

bolla dovuta a "dipendenza logica"

lettura di leggere in Unità X qualcosa scritto da Unità Y

IU 1) manda a EU <+, a, b, c>

2) cont(Rc) ++

EU → IU < val, c >

IU val → REG'[c]. valore

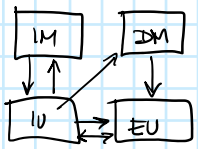
REG'[c]. cont --

bolla "da salto"

esecuzione del programma

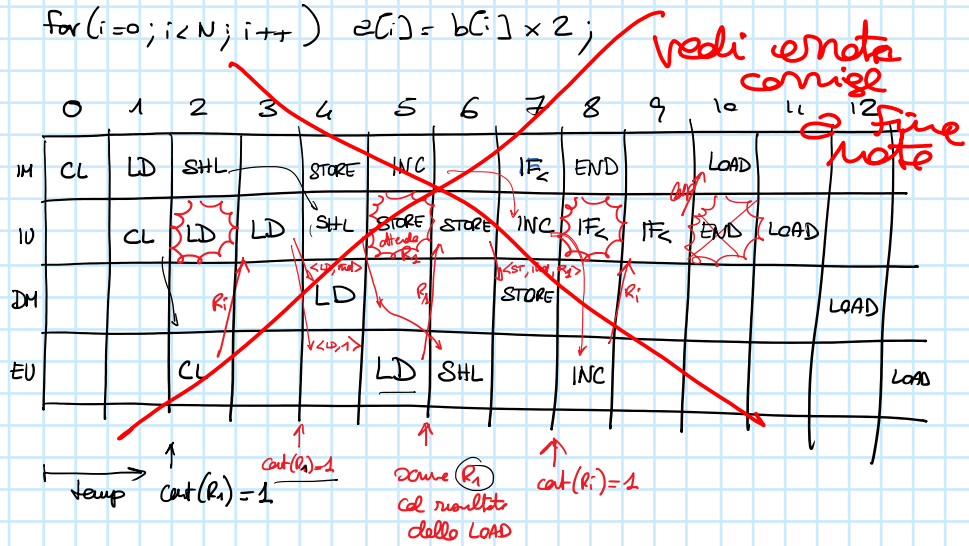
```
t  CLEAR  Ri
loop: t  LOAD   R_baseB, Ri, R1
      t  SHL    R1, #1, R1
      t  STORE  R_baseA, Ri, R1
      t  INC    Ri
      t  IF<    Ri, RN, loop
      END
```

Tid = 6t




```
for(i=0; i<N; i++) c[i] = b[i] × 2;
```

vedi nota correzione fine nota



tempo di completamento del codice: init + 1ª iterazione } 10t in questo caso

$$\varepsilon = \frac{6t}{10t} = 0.6 \qquad T_{id} \,/\, T(n)$$

2 problemi

↙ bolle "di salto"

→ bolle "di dipendenza logica"

SOLUZIONI

usare delayed branch

⇓

Riorganizzare codice assembler D-RISC

loop: SUB R_i, #1, R_i

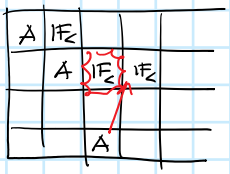
...

ADD R_2, R_D, R_C

$IF_<$ R_C, R_N, loop (loop+1)

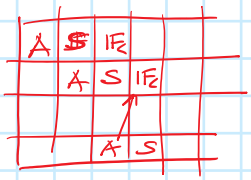
R_i --

K=1



3t x 2 ist

ε = 2/3



3t x 3 ist

ε = 1

ERRATA CORRIGE (esecuzione programma for(i=0; i<N; i++) a[i] = b[i] * 2;)

Td = 6t

loop: t CLEAR R_i ⟷ ADD R_0 R_0 R_1

t LOAD $R_{base\,B}$, R_i, R_1

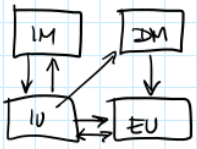
t SHL R_1, #1, R_1

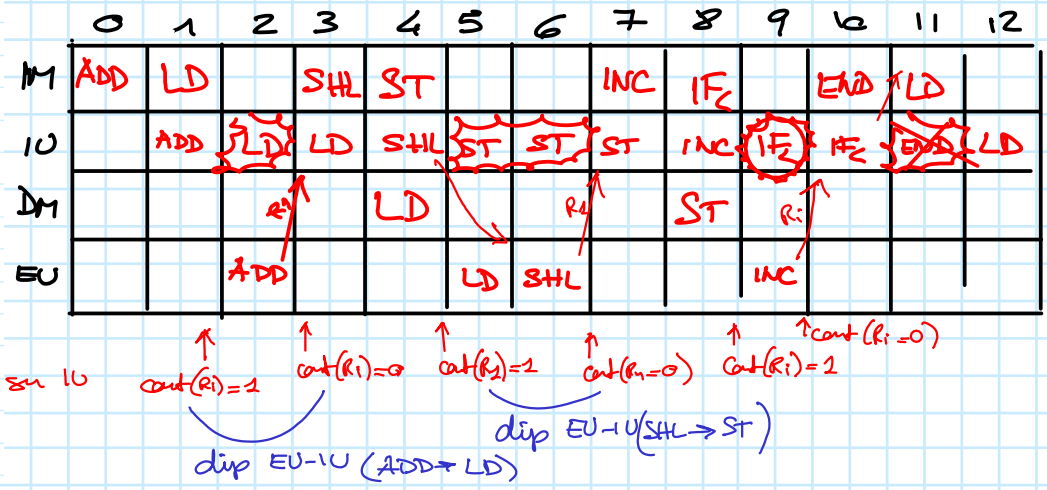
t STORE $R_{base\,A}$, R_i, R_1

t INC R_i

t $IF_<$ R_i, R_N, loop

END





11t

$T_{id} = 6t$ $T_e = 11t$ $\varepsilon = \frac{6t}{11t} = 0,54$

nello schema precedente (quello che ora è barrato X) mancava il conto corretto relativo all'azzeramento del contatore di R_1, incrementato prima di inviare SHL ad EU e testato sulla IU per decodificare la STORE ▽