Domande 1

Note Title 11/07/2007

2) le due microistruzioni relative a chiamata e decodifica delle istruzioni sono le stesse della dispensa

ch0. IC → IND, "read" → op, set RDYout, ch1

ch1. (RDYin, or(ESITO) = 0 –) nop, ch1

(= 10) DATAIN → IR
REG[DATAIN.R1] → A
REG[DATAIN.R2] → B
reset RDYin
DATAIN-COP → RC

(= 11) ESITO → N[29..31]
reset RDYin
trattamento-eccezioni

(si potevano considerare anche altre versioni, purché corrette)

di conseguenza, l'esecuzione della nuova istruzione può essere implementata con le due micro-istruzioni che seguono:

fmem=0. A + B → MD, "read" → OP, set RDYout

fmem=1. (RDYin, or(ESITO), (REG[IR.R3] - DATAIN), INT
= 0 ---) nop, fmem=1
(= 11 --) come in ch1 : halt ecc
(= 1010) IC + 1 → IC, ch0
(= 1011) IC+1 → IC, halt_int

(= 1000) IC + IR.TARGET → IC, ch0
(= 1001) IC + IR.TARGET → IC, halt_int

Nello if mem = Ø si ordina la lettura della cella di memoria da leggere

Nello if mem = 1 si testano tutte le condizioni necessarie a completare l'esecuzione delle istruzioni:

(Reg[IR.R3] – DATAIN)

è il test sul bit ZERO della ALU che ci dice se o no dobbiamo saltare a TARGET (IC ← IC+TARGET) oppure no (IC ← IC+1)

(RDYin)

è il test che permette di attendere la lettura della memoria

(ESITO & INT)

permettono di saltare adeguatamente interruzioni ed eccezioni

Valutazione del tempo medio di elaborazione:

fetch e decodifica: $2\tau + t_A$

esecuzione: $2\tau + t_A$

$4\tau + 2t_A$

b) assumendo che $N > 0$
(e quindi compilando il for come se fosse un repeat)

```
loop:   IF_MEM= R_basex , R_i , R_K , continue
        ADD   R_i , R_K , R_temp
        STORE R_basex , R_i , R_temp

continue: INC  R_i
          IF<  R_i , R_N , loop
```

c) la compilazione con l'ASM RISC "tradizionale" sarebbe stata invece la seguente:

```
loop:   LOAD    R_base_x, Ri, Rtemp
        IF=     Rtemp, Rx, continue
        ADD     Ri, Rx, Rtemp
        STORE   R_base_x, Ri, Rtemp
continue: INC   Ri
        IF<     Ri, RN, loop
```

c'è una istruzione in più in ogni iterazione

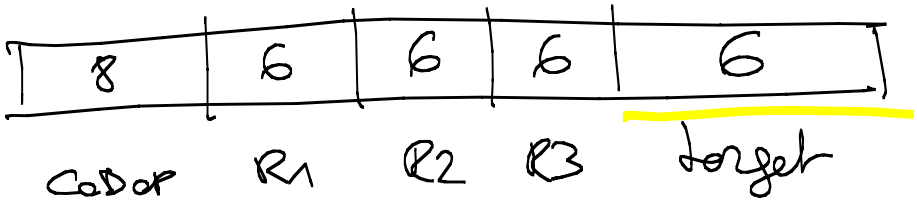
Il tempo di completamento, delle
assunzioni comuni nei due casi,
varia solo perché nel primo caso
abbiamo una IFMEM= che sostituisce
2 istruzioni (LOAD, IF=) nel secondo
caso.

Nel primo caso, si paga dunque
un accesso in memoria in meno
(quello di 1 fetch) e alcuni T
in meno perché si "esegue" solo
la IFMEM= (2T + t) invece
della LOAD + IF= ...

Il numero dei fault non cambia:
Si può assumere che 6 istruzioni
(parole) invece che 5 non
cambino il pattern di accesso
in cache: se i blocchi fossero da 2^3
parole (o più grandi), il loop
potrebbe occupare un solo blocco,
in caso contrario ($< 2^3$, ma
questo non è molto realistico)
occuperebbe 2 blocchi. In ogni
caso si può assumere che non
venga mai cacciata dalla cache.

Il ciclo di clock del processore è lo stesso nei due casi, per come è costruito il precodice che interpreta lo IFMEM =

d)



posso realizzare salti

a 2^5 locazioni indietro

o $2^5 - 1$ locazioni avanti

la somma (max)

Quindi possiamo saltare

32 istruzioni indietro

o 31 istruzioni in avanti

Domande 2

1) P va in esecuzione
   - P si blocca nella 1a send
   - va in esecuzione Q
   - esegue la receive,
   - sblocca P che va in ready
   - Q si blocca nella 2a receive
   - P esegue la 2a send
   - sblocca Q che va in ready
   - P si blocca nella send successiva

... etc etc

2) se $K \geq N$

- P esegue tutte le send (si può bloccare nell'ultima se $K = N$)
- Q va in esecuzione ed esegue tutte le receive.
- se $K = N$ l'ultima receive sblocca P

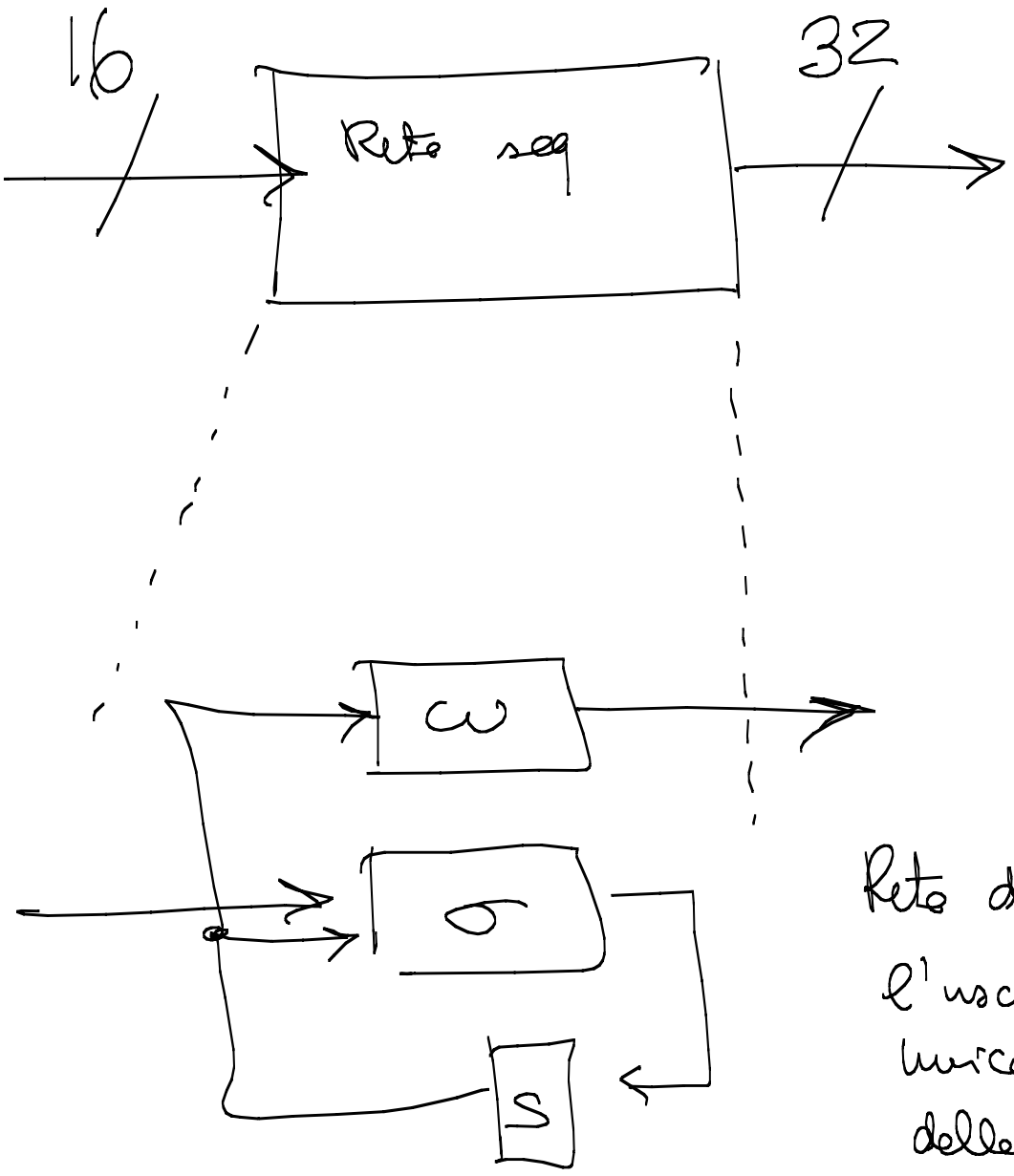
Se invece $K < N$

- P esegue K send poi si blocca

- Q esegue 1 receive e blocca P deve in pronta
- Q esegue le altre $K-1$ receive
- Q si blocca nella $K+1$ esima receive
- va in esecuzione P esegue 1 send e blocca Q

...

Domanda 3



Rete di Moore:
l'uscita dipende unicamente dallo stato interno (funzione CU)

$\omega = id$

$\sigma =$