Architetture degli elaboratori – A.A. 2006-2007

Appello del 5 giugno 2007

Domanda 1

Si progetti un'unità cache ad indirizzamento diretto. La capacità della cache è di 16K blocchi (linee). Ogni blocco ha 4 parole.

La memoria cache è interfacciata al sottosistema P/MMU ed alla memoria principale M mediante interfacce standard, come quelle viste durante il corso. Lo studente può fare le proprie (ragionevoli) assunzioni sul sottosistema di memoria principale ma non ne è richiesto un progetto dettagliato.

L'indirizzo di memoria principale è di 32 bit.

Per la realizzazione della memoria interna alla cache si hanno a disposizione componenti logici di memoria da 16K parole ciascuno (MEM).

Si assuma che le scritture avvengano mediante protocollo write-through e si trattino opportunamente i fault di cache, mantenendone la trasparenza rispetto al processore centrale. Per il trattamento dei fault, si assuma che il dato richiesto sia sempre presente nella memoria principale. Sono esplicitamente richiesti:

- il microprogramma dell'unità;
- la valutazione del ciclo di clock, con la discussione del suo impatto sul processore;
- la valutazione del tempo necessario ad effettuare una lettura o una scrittura nell'ipotesi che non via siano fault di cache.

Domanda 2

Di un insieme di vettori $X_1 \ldots X_m$ si vogliono calcolare le funzioni *f* e *g*. In particolare

$$f(X_j) = (x_{j1} + \ldots + x_{jn}) / n \qquad \text{e} \qquad g(X_j) = (x_{j1} * \ldots * x_{jn}) / n^2$$

dove i vari x_{ji} sono gli elementi di un generico X_j. Il calcolo delle *f* e delle *g* avviene utilizzando processi appositi (PROC_F e PROC_G), che ricevono i vettori e restituiscono la risposta utilizzando due canali asincroni ad una posizione, uno per richiedere la richiesta di calcolo della funzione, con i relativi parametri, ed uno per restituire il risultato di tale calcolo. Un processo MASTER provvede a richiedere il calcolo di *f* o di *g* a seconda del valore restituito da una certa funzione booleana *FoG* eseguita localmente. La funzione *FoG* restituisce true (va calcolata *f*) se il primo elemento del vettore è pari, diversamente restituisce false (e va calcolata *g*).

a) scrivere il codice dei processi MASTER e PROC_F nel linguaggio concorrente utilizzato nel corso;

b) fornire il compilato in assembler RISC dei processi MASTER e PROC_F;

c) dettagliare la memoria virtuale del processo PROC_F;

d) discutere le implicazioni della scelta di canali asincroni ad una posizione.

Domanda 1

Si possono adottare diverse soluzioni, a seconda delle ipotesi sui dettagli non specificati nel testo. In particolare, si può fare ogni tipo di assunzione (ragionevole) sulla modalità di accesso al livello superiore della gerarchia di memoria, così come sul tipo di implementazione della cache (accesso in un singolo ciclo di clock, ma lungo, oppure accesso in più di un singolo ciclo di clock, di durata più limitata).

Nel seguito mostriamo una possibile soluzione.

Assumiamo che l'interfaccia verso il processore MMU disponga dei registri:

- in uscita verso P/MMU
  - OUTp per i dati, ACKp, ESITOp
- in ingresso da P/MMU
  - INp per i dati, INDp, OPp

e di avere l'interfaccia verso il livello superiore della memoria con i registri:

- in uscita verso il sottosistema di memoria
  - OUTm, RDYm, OPm, INDm
- in ingresso dal sottosistema di memoria
  - INm, ACKm, ESITOm

Gli indirizzi inviati alla cache sono indirizzi fisici inviati dalla MMU su 32 bit, con 16 bit di tag (IND[31..16] o IND[TAG]), 2 bit di offset (IND[0..1] o IND[OFF]) e i rimanenti 14 bit di TAG (IND[TAG]). Eventuali fault di memoria principale sono gestiti da MMU e P. I fault di cache devono essere gestiti a microprogramma, in maniera trasparente, rispetto al processore.

Prima soluzione

Assumiamo che l'accesso alla memoria del livello superiore avvenga in parallelo su quattro parole. Ovvero che si possono leggere o scrivere quattro parole contemporaneamente, utilizzando registri INm e OUTm di ampiezza pari a 4 parole, cioè 128 bit. Assumiamo inoltre che l'interfaccia con processore e MMU sia di tipo classico, quindi alla parola.

Sotto questa condizione possiamo pensare di cercare di realizzare l'accesso in un unico ciclo di clock.

Il microprogramma può essere realizzato come descritto nell'allegato 1

In questo caso, la valutazione sul ciclo di clock e la valutazione del tempo per una lettura o scrittura con cache hit è quella dell'allegato 2.

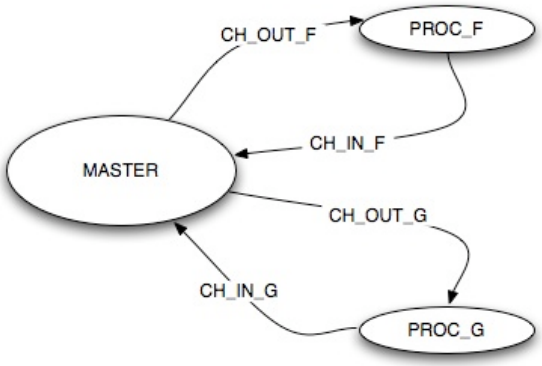
Soluzioni alternative

Possiamo considerare invece la soluzione in cui il test sulla presenza del dato in cache (funzione test) scriva il risultato in un registro/flag consultato nella microistruzione successiva. Questo comporta una riduzione della lunghezza del ciclo di clock (un solo ta) ma richiede un ulteriore ciclo di clock per effettuare sia l'accesso in lettura che quello in scrittura.

Possiamo anche considerare una memoria principale interallacciata, cosa che comporta la realizzazione di un ciclo di letture per ricevere dati da M in caso di fault. Visto che la cosa non ha impatto sugli accesso con hit (sia per le letture che per le scritture) questo non comporta una diversa valutazione dei tempi di accesso in assenza di fault.

Domanda 2

Processi e canali sono organizzati come segue:



Il processo MASTER, nell'ipotesi che i vettori siano in memoria, nel vettore X[i], può essere scritto come segue:

```
master ::

     for(i=0; i<N; i++) {
          if(FoG(X[i])) {
               send(CH_OUT_F, X[i]);
               receive(CH_IN_F, Y[i]);
          } else {
               send(CH_OUT_G, X[i]);
               receive(CH_IN_G, Y[i]);
          }
     }
```

mentre PROC_F può essere scritto come segue:

```
proc_f ::

     while(true) {
          res = 0;
          receive(CH_OUT_F, X);
          for(i=0; i<N; i++)
               res = res + X[i];
          res = res / N;
          send(CH_IN_F, Y);
     }
```

La compilazione del master in assembler RISC è una cosa tipo

```
for:      move R_baseX, R_in_FoG  ; prepara param per FoG
          call FoG, R_ret         ; chiama la funzione
          if=0 R_out_FoG, doF     ; scegli cosa eseguire
doG:      move R_baseX, R_in_send_msg   ; caso G
          move R_ch_out_g, R_in_send_ch ; param send
          call R_send, R_ret            ; chiamata send
```

```
            ; e chiamata impediata alla receive
            move R_ch_in_g, R_ch_receive_in_ch
            call R_receive, R_ret
            ; memorizza il risultato
            store R_base_Y, Ri, R_receive_out_msg
            ; continua il loop, in realta' potevo saltare
            ; sotto, alle stesse istruzioni ripetute
            inc R_i
            if< R_i, R_N, for
            goto fine
doF:        move R_baseX, R_in_send_msg
            move R_ch_out_g, R_in_send_ch
            call R_send, R_ret
            move R_ch_in_g, R_ch_receive_in_ch
            store R_base_Y, Ri, R_receive_out_msg
            inc R_i
            if< R_i, R_N, for
fine:       end
```

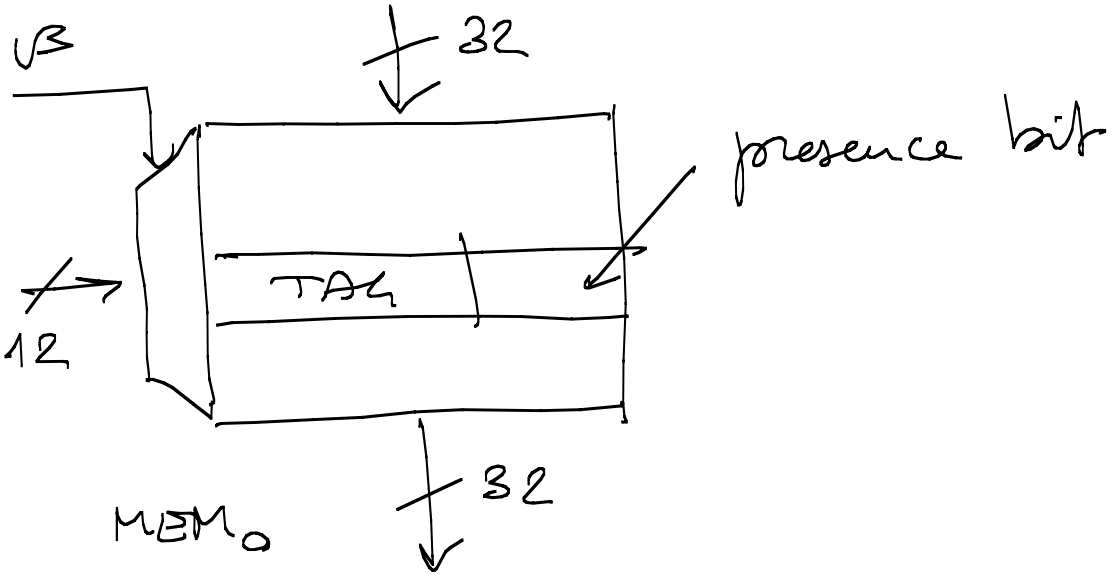
La compilazione di FUNC_F è lasciata per esercizio.

La mappa di memoria virtuale per il processo MASTER contiene il segmento programma, il segmento dati (vettore di vettori X e Y), PCB e descrittori di canali, handler delle interruzioni etc. come al solito.

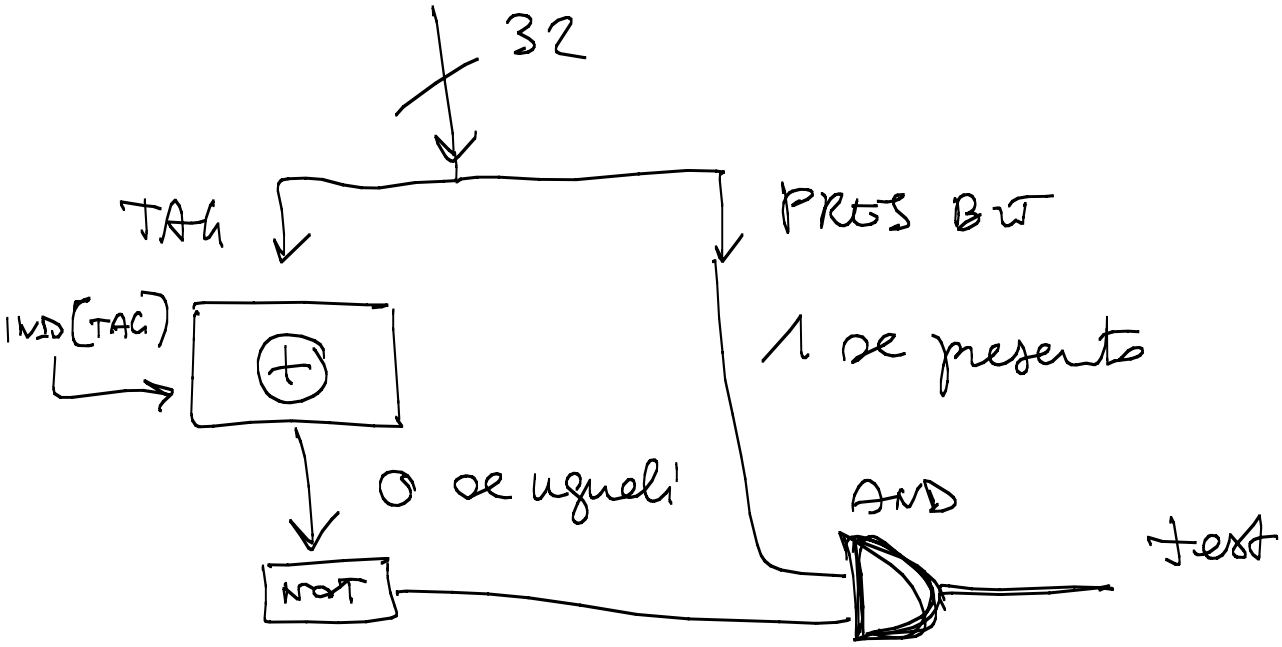
Non c'e' nessuna implicazioni relativa all'uso dei canali asincroni ad una posizione. La comunicazione fra MASTER e FUNC_X è una comunicazione a domanda risposta e l'asincronia del canale non viene di fatto sfruttata.

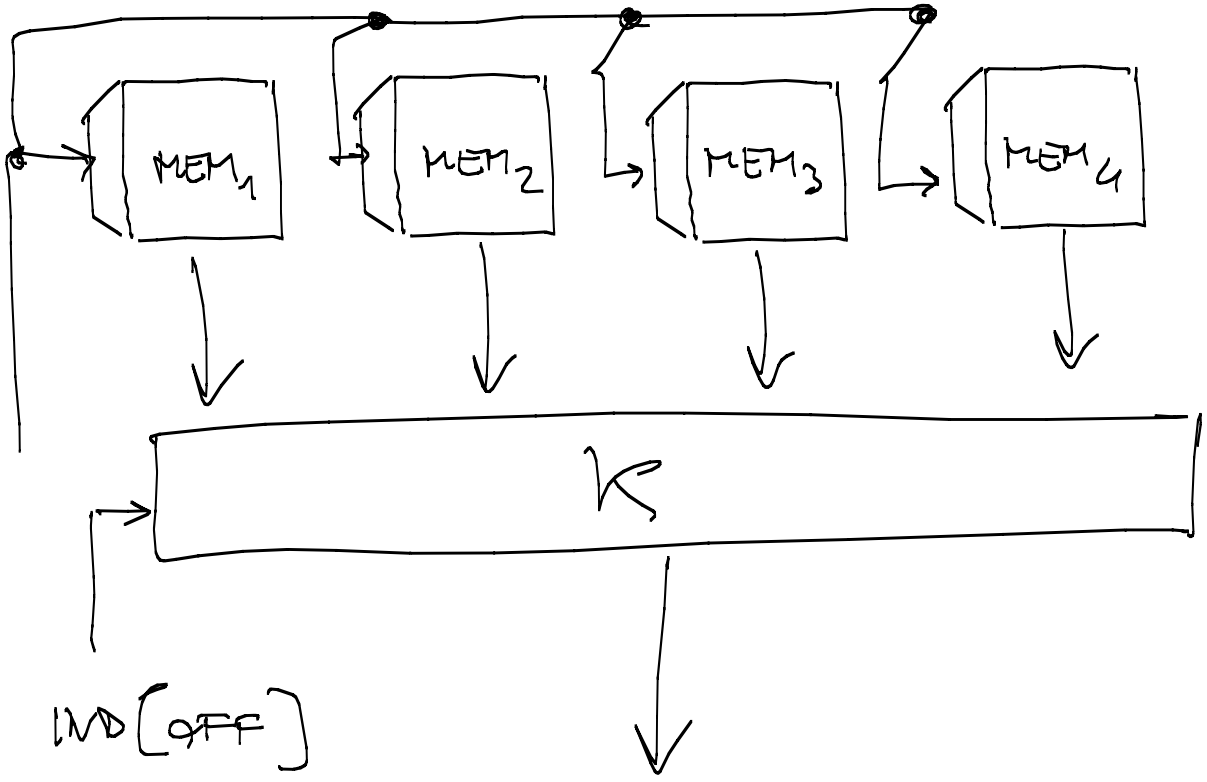
Soluzione con accesso parallelo a M

Note Title 05/06/2007



modulo da 16K parole che contiene TAG e bit di presenza





lettura di 1 parola della cache

μ-programma (schema con commenti)

Note Title 05/06/2007

$\emptyset$. (RDY_p, TEST(IND, MEM_0), $op = \emptyset$ --) $nop, \emptyset$

controllo che
IND[TAG] = MEM_0[IND[BLOCK]][TAG]
e che
MEM_0[IND[BLOCK]][PRESBIT] = 1
e restituisce
1 se vero
0 se falso

(=1 1 $\emptyset$) letture, cache hit

$MEM_{IND[OFF]}$[IND[BLOCK]] → OUT_p,
reset RDY_p, set ACK_p,
$\emptyset$ → $ESITO_p$, $\emptyset$.

→ modulo di memoria indirizzato dall'OFFSET: uso un K

(=111) scrittura, cache hit

scrivono tutto in M

x implementare il write through

$MEM_1[IND[BLOCK]] \rightarrow OUT_m[0]$,

$MEM_4[IND[BLOCK]] \rightarrow OUT_m[3]$,

$IND_p[31..2]\emptyset\emptyset \rightarrow IND_m$,

$OP_p \rightarrow OP_m$, set RDY_m,

copia il blocco verso la M

$IN_p \rightarrow MEM_{IND[OFF]}[IND[BLOCK]]$,

scrivi nel blocco di cache

1. e salta alla μ-istruzione 1

(=100) lettura, cache fault

$IND_p[31..2]00 \rightarrow IND_m$,

$OP_p \rightarrow OP_m$, set RDY_m, 2.

comanda la lettura del blocco in M

(=101) scrittura, cache fault;
questo è il caso complicato
xché dobbiamo istanziare
un nuovo blocco.
Per via del write through non
occorre salvare il vecchio
blocco.
leggiamo il blocco + le
altre parole ...

$IND_p[31..2]00 \rightarrow IND_m$,
"read" $\rightarrow OP_m$, set RDY_m, 3.

comando la lettura ↑

1. $(ACK_m = 0)$ nop, 1

attendo scrittura, questa può essere verso una coda per ragioni di efficienza, anziché verso la memoria vera e propria

$(=1)$ scrittura write through effettuata, mando risposta al processore

$ESITO_m \rightarrow ESITO_p$,

reset ACK_m, reset RDY_p,

set ACK_p, $\emptyset$.

e poi ritorno alla prima microistruzione

2. $(ACK_m, ESITO_m = 0 -)$ mop. 2

$(= 1\ 0)$ blocco nell'interfaccia

$IN_m[0] \rightarrow MEM_1[IND_p[BLOCCO]]$

$\vdots$

$IN_m[3] \rightarrow MEM_4[IND_p[BLOCCO]]$

scrivi blocco in cache ↑

$MEM_a[IND_p[BLOCCO]][TAG] = IND_p[TAG]$

$MEM_a[IND_p[BLOCCO]][PRESBIT] = 1$ ↑

aggiorna info di controllo

$ESITO_m \rightarrow ESITO_p$

reset ACK_m, reset RDY_p,

set ACK_p, $\emptyset$

$(= 1\ 1)$ ordine di lettura

$ESITO_m \rightarrow ESITO_p$, reset ACK_m

reset RDY_m, set ACK_m, $\emptyset$

3. $(ACK_m, IND_p[OFF] = 0 --)$ nop, 3

$(=100)$

$IN_p \rightarrow OUT_m[00] \rightarrow MEM_1[IND[BLOCCO]]$

$IN_m[1] \rightarrow OUT_m[1] \rightarrow MEM_2[IND[BLOCCO]]$

$IN_m[2] \rightarrow OUT_m[2] \rightarrow MEM_3[IND[BLOCCO]]$

$IN_m[3] \rightarrow OUT_m[3] \rightarrow MEM_4[IND[BLOCCO]]$

"write" $\rightarrow OP_m$, reset ACK_m, set RDY_m, 4.

$(=101)$ ...

$(=110)$ ...

$(=111)$ ...

manca trasferimento dato

stesso caso con IN_p nella posizione giusta e il resto da IN_m (4 parole)

qui ordina il write back e scrive la cache

4. $(ACK_m, ESITO_m = 0 -)$ $Mdp, 4$

$(=10)$

$MEM_d[IND_p[BLOCCO]][PRESBIT] = 1$

$MEM_d[IND_p[BLOCCO]][TAG] = IND_p[TAG]$

$0 \rightarrow ESITO_p$, reset ACK_m,

reset RDY_p, set ACK_p, $\emptyset$

$(=11)$

$ESITO_m \rightarrow ESITO_p$, reset ACK_m,

reset RDY_p, set ACK_p, $\emptyset$

Valutazione cicle di clock

Note Title 05/06/2007

T_{cwpa} tempo necessario x stabilizzare le variabili di condizionamento

→ t_a + $3t_p$

x accedere alla MEM

x calcolare test

$T_{cwpc} \equiv T_{\sigma_{PC}} \equiv$ assumiamo che sia $2t_p$

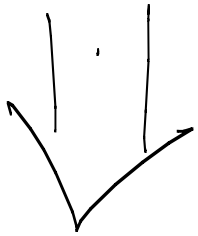
$T_{\sigma_{pa}}$ tempo necessario x calcolare nuovo stato interno

→ t_a (devo poter fare accesso a

$MEM_1 \dots MEM_4$)

Complessivamente:

$$\tau \approx 2\tau_a + k\tau_p$$



Quindi, siccome è
la stessa del processore, avrà
un processore centrale "lento"

Tempo x Lettura (cache hit)

Con questa soluzione
ho un singolo ciclo di clock

↳ esegue solo la
μ-istruzione ∅.

x scrittura

in questo caso esegue la
∅. e la 1. e ha un
tempo di accesso (in scrittura)
x il livello superiore della
gerarchia di memoria

quindi $2\tau + t_{a_M}$

questo potrebbe essere una scrittura in una CODA che disaccoppia M da cache, quindi un tempo relativamente basso.