Architettura degli Elaboratori, a.a. 2006-07

Appello del 18 gennaio 2007

Domanda 1

Una unità di elaborazione U calcola la funzione

$$D = f(A,B,C) = \sum_{i=0}^{N-1} (A_i + B_i + C_i)$$

dove A, B, C sono vettori di *N* interi e D è un intero.

Nel caso che, durante il calcolo di *f*, venga generata una condizione di overflow, il calcolo non viene completato.

U riceve da una unità U1 prima il valore di *N* (rappresentato su parola) e successivamente, *una alla volta*, le *N* triple di valori (A_i, B_i, C_i). Alla stessa U1 viene ritornato un valore booleano ESITO che indica se il calcolo di *f* è stato completato con successo o meno e, nel caso di successo, il valore del risultato D.

Fornendo le spiegazioni adeguate:

a) scrivere il microprogramma di U;

b) nel caso di calcolo completato con successo, valutare il tempo medio di elaborazione di U in funzione di *N* e del ritardo t_p di una porta logica con al massimo 8 ingressi, tendo conto che una ALU ha tempo di stabilizzazione di $5\ t_p$.

Domanda 2

Si consideri un programma applicativo che calcola la funzione *f* della Domanda 1, assumendo, stavolta, che i vettori A, B e C siano allocati in memoria, che il risultato (D, ESITO) sia ritornato in memoria, e che N = 4Mega.

a) Compilare il programma in assembler Risc. Il linguaggio assembler dispone dell'istruzione IF_OVERFLOW OFFSET riferita al risultato dell'istruzione immediatamente precedente.

Il programma va compilato come procedura, fornendo adeguate spiegazioni.

b) Del *processo* che esegue il programma si mostri e si spieghi la mappa di allocazione della memoria virtuale.

c) Valutare il numero di fault di cache (nel caso che non si verifichino overflow) per una architettura in cui sia presente solo la cache di primo livello, associativa su insiemi con 4 insiemi per blocco, blocchi di 16 parole e 256 insiemi.

Domanda 3

Si spieghi come implementare la lettura di un blocco di *n* parole, richiesta da un processo applicativo ad un dispositivo di ingresso DEV, assumendo che i servizi del sistema operativo siano realizzati secondo il modello a processi comunicanti. Il valore di *n* varia da richiesta a richiesta.

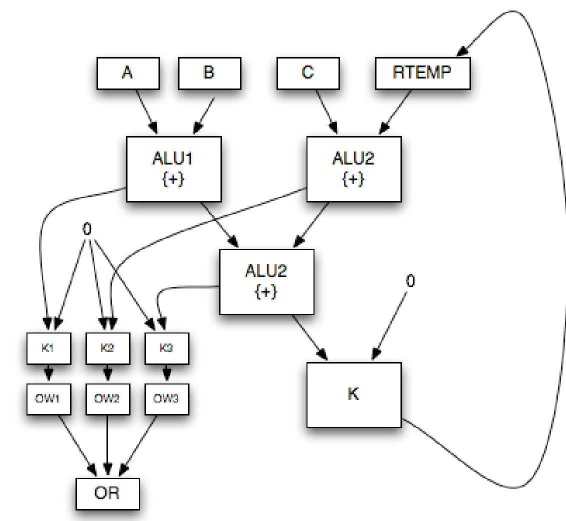
Dare una soluzione nel caso in cui l'unità di I/O, associata a DEV, sia in grado di eseguire primitive *send/receive*, ed una soluzione nel caso in cui non lo sia.

Domanda 1: soluzione

L'unità U riceve da U1 valori di N e delle triple che rappresentano (A_i, B_i, C_i) mediante i registri A, B e C e relativi RDYIN e ACKIN. Trasmette l'esito del calcolo di *f* e l'eventuale risultato mediante i registri ESITO e D ed i relativi RDYOUT e ACKOUT. Nell'ipotesi di non porre limitazioni sulle risorse da includere nella parte operativa, il microprogramma dell'unità potrebbe essere il seguente (si assume che ESITO=0 denoti una condizione di assenza di errori):

0. (RDYIN=0) nop, 0;
   (=1) A → NTEMP, 0 → RTEMP, reset RDYIN, set ACKIN,
   0 → OW_1, 0 → OW_2, 0 → OW_3, 1
1. (RDYIN,or(OW_1,OW_2,OW_3),$NTEMP_0$=0--) nop, 1;
   (=100) NTEMP-1 → NTEMP, (A+B)+(RTEMP+C) →RTEMP, reset RDYIN, set ACKIN, 1
   (=110) 1 → ESITO, 0 → D, set RDYOUT, reset ACKOUT, 0
   (=101) 0 → ESITO, RTEMP → D, set RDYOUT, reset ACKOUT, 0
   (=111) 1 → ESITO, 0 → D, set RDYOUT, reset ACKOUT,
   A → NTEMP, 0 → RTEMP, reset RDYIN, set ACKIN, 1

La parte operativa derivata secondo il procedimento formale prevede l'utilizzo di 4 ALU (ramo =100 della seconda microistruzione): una (op = {DEC}) per il decremento del numero di iterazioni ancora da eseguire e tre per il calcolo del nuovo valore di RTEMP (op = {ADD}). Ciascuna delle tre ALU che effettuano la somma avrà un'uscita di un bit diretta al registro OW_i (i numero della ALU). I valori dei registri, mediante una semplice porta OR a tre ingressi vengono utilizzati, al ciclo successivo, per testare se siamo in presenza di una condizione di overflow. In questo caso, la computazione viene terminata immediatamente.



In questo caso, per ottenere il tempo medio di elaborazione (nel caso di successo) dobbiamo derivare il tempo di ciclo in funzione di t_p. Ricordando che il tempo di ciclo può essere calcolato come:

$$T = T_{\omega PO} + \max \{T_{\omega PC} + T_{\sigma PO}, T_{\sigma PC}\} + \delta$$

possiamo assumere che:

- $T_{\omega PO} = t_p$ *or dei 3 valori dei registri OWi*
- $T_{\omega PC} = 2t_p$ *2 soli stati, 3 variabili di condizionamento, 2 livelli di logica*
- $T_{\sigma PC} = 2t_p$ *idem come sopra*
- $T_{\sigma PO} = 2(5t_p)+ 2t_p$ *2 ALU in cascata + K per scrivere registro*
- $\delta = t_p$ *solita assunzione sul segnale di clock*

e quindi possiamo concludere che

$$T = t_p + (2\ t_p + 12\ t_p) + t_p = 16\ t_p$$

Da cui otteniamo che il tempo medio, in caso di successo è

$$16\ (1 + N)\ t_p \cong 16\ N\ t_p$$

Il termine (1+N) deriva dal fatto che la prima microistruzione è eseguita una volta sola mentre la seconda viene eseguita N volte (in caso di computazione con successo, ovviamente).

Domanda 2: soluzione

Il codice ad alto livello della procedura da compilare è il seguente:

```
ris = 0;
for(int i=0; i<N; i++) {
      temp = A[i] + B[i];
      if(OVERFLOW) { ris = 0;  esito = 1; return; }
      temp = temp + C[i];
      if(OVERFLOW) { ris = 0;  esito = 1; return; }
      ris = ris + temp;
      if(OVERFLOW) { ris = 0;  esito = 1; return; }
}
esito = 0;
return;
```

La procedura può essere compilata come segue. Assumiamo che Ra, Rb, Rc, Resito e Rris contengano in ingresso alla procedura il valore della base dei vettori A, B e C e l'indirizzo in memoria delle locazioni che conterranno l'esito e il risultato, che Rn contenga in ingresso alla procedura il valore N e infine che Rret contenga l'indirizzo di ritorno della procedura.

```
CALC_F:     CLEAR Rr                ; azzeramento ris parziale
            CLEAR Ri                ; azzeramento var iterazione
LOOP:       LOAD Ra, Ri, R1
            LOAD Rb, Ri, R2
            ADD R1, R2, R3          ; Ai + Bi
            IF_OVERFLOW ERR_END     ; se overflow
            LOAD Rc, Ri, R4
            ADD R3, R4, R4          ; Ai + Bi + Ci
            IF_OVERFLOW ERR_END
            ADD R4, Rr, Rr          ; D = D + Ai + Bi + Ci
            IF_OVERFLOW ERR_END
            INC Ri                  ; variabile d'iterazione
            IF<= Ri, Rn, LOOP       ; itera se non ho finito
```

```
ERR_END:    STORE Resito, R0, #1    ; esito non buono
            STORE Rris, R0, R0      ; risultato non significativo
            GOTO Rret               ; ritorno controllo al chiamante
END:        STORE Resito, R0, R0    ; esito positivo
            STORE Rris, R0, Rr      ; risultato in memoria
            GOTO Rret               ; ritorno controllo al chiamante
```

Il codice della nostra applicazione sarebbe stato questo più il codice del main dell'applicazione:

```
MAIN:       MOV RbaseA, Ra
            MOV RbaseB, Rb
            MOV RbaseC, Rc
            MOV RdimN, Rn
            MOV Rindesito, Resito
            MOV Rindrisultato, Rris
            CALL CALC_F, Rret
            END
```

Notare che in linea di principio, se questa fosse l'applicazione, si potrebbe assumere che il compilatore allochi l'indirizzo base dei vettori e la la lunghezza dei vettori già nei registri opportuni, nel qual caso le MOV sarebbero assolutamente inutili e quindi il codice sarebbe semplicemente:

```
MAIN:       CALL CALC_F, Rret
            END
```

In entrambi i casi, la mappa di allocazione della memoria virtuale del processo che esegue l'applicazione potrebbe essere strutturata come segue:

- indirizzi 0—18, codice della procedura *inizializzato*
- indirizzi 19—20, codice del main *inizializzato*
- indirizzi 21 — (4M+20), vettore A *inizializzato*
- indirizzi (4M+21)—(8M+20), vettore B *inizializzato*
- indirizzi (8M+21)—(16M+20), vettore C *inizializzato*
- indirizzo (16M+21), N *inizializzato*
- indirizzo (16M+22), Esito *non inizializzato*
- indirizzo (16M+23), D *non inizializzato*
- indirizzi (16M+24) —(16M+1K+23), *PCB di tutti i processi, handler delle eccezioni e delle interruzioni, codice delle primitive di comunicazione e del kernel (routine per lo scheduling). In particolare, nella porzione del PCB relativo al processo destinata ad accogliere i valori dei registri generali, la parola relativa ad Ra conterrà la base di A in memoria (cioè 21), quella relativa a Rb conterrà la base di B in memoria (cioè 4M-21) etc. Inoltre, in questa parte della memoria virtuale troviamo anche le informazioni necessarie per interagire con il gestore della memoria (per esempio, i nomi dei canali di comunicazione col gestore della memoria).*

Per quanto riguarda il numero di fault, possiamo assumere che anche se gli indirizzi fisici dei tre vettori, una volta tradotti dagli indirizzi logici nella MMU, mappassero sullo stesso insieme, non avremmo problemi in quanto la cache è associativa a 4 vie e quindi in grado di contenere tre distinte parti del working set nello stesso insieme. Possiamo quindi assumere di essere in presenza dei soli fault fisiologici, cioè di

$$\#fault_codice + \#fault_vettori + \#fault_scalari =$$
$$(\#num_istruzioni / \sigma) + \#vettori * (N / \sigma) + (\#variabili_scalari / \sigma) =$$
$$2 + 3*(4M/16) + 1 = 3 + 256K \cong 256K$$

ovvero (oltre a quelli per il codice e per le variabili scalari) solo di quelli necessari a trasferire in cache i blocchi di memoria principale la prima volta che vengono riferiti per poi accedere direttamente alla cache, senza fault, i rimanenti (16-1) altri riferimenti a quel blocco.

Domanda 3: soluzione

Il processo utente utilizza un canale asincrono ad una posizione verso l'unità di I/O, mediante il quale trasmette la tupla

< *n* , *ch_risposta* >

ed un canale sincrono *ch_risposta* sul quale si mette in attesa della risposta (a pezzi di k parole ciascuno, in generale con $k<n$).

Nel caso l'unità di I/O associata al dispositivo sia in grado di eseguire primitive di comunicazione, sarà la stessa unità a comunicare al processo utente, mediante una *send* sul canale ricevuto da DEV, la conclusione dell'operazione ed il suo esito. Diversamente, l'unità di I/O genererà un'interruzione e la routine di trattamento dell'interruzione provvederà ad implementare la *send* sul canale su cui il processo utente è in attesa.

Domanda 1: varianti

Variante 1

Se avessimo voluto ottimizzare la lunghezza del ciclo di clock, avremmo potuto utilizzare solo due ALU nella parte operativa, una per il calcolo (incrementale) del risultato e l'altra per il decremento del numero di iterazioni da eseguire. Il microcodice corrispondente sarebbe dunque stato del tipo:

0. (RDYIN=0) nop, 0; set ACKIN, reset RDYIN, A → NTEMP,
   0 → RTEMP, 0 → OW_1, 0 → OW_2, 1
1. (RDYIN,NTEMP0,OW_1=0--) nop, 1;
   (=100) A+B → TEMP1, C+RTEMP → RTEMP, reset RDYIN,
   set ACKIN, 2
   (=110) 0 → ESITO, RTEMP → D, set RDYOUT, reset ACKOUT,
   set ACKIN, reset RDYIN, A → NTEMP, 0 → RTEMP, 1
2. (=1-1) 1 → ESITO, 0 → D, set RDYOUT, reset ACKOUT,
   set ACKIN, reset RDYIN, A → NTEMP, 0 → RTEMP, 1
3. (or(OW_1,OW_2)=1) 1 → ESITO, 0 → D, set RDYOUT, reset ACKOUT, 0
   (=0) TEMP1+RTEMP → RTEMP, NTEMP-1 → NTEMP, 1

In questo caso utilizziamo al massimo 2 ALU (una per decrementare il contatore e una per accumulare il risultato), quindi il calcolo del contributo di ogni singola tripla avviene in due cicli successivi. Si testa alla fine di ogni ciclo l'owerflow che si sia verificato o in una delle due ALU utilizzate per il calcolo dei parziali (in questo caso calcoliamo f come $D_{i+1} = (A_i + B_i) + (D_i + C_i)$ e quindi in un ciclo calcoliamo $(A_i + B_i)$ e $(D_i + C_i)$ e nel ciclo successivo, in assenza di owerflow, calcoliamo $(A_i+B_i) + (D_i + C_i)$) o nella singola ALU utilizzata per accumulare il risultato. Se il test rileva un overflow, viene restituito un esito pari a 1 con risultato 0. Diversamente si prosegue con il calcolo e ci si arresta solo quando si arriva ad aver calcolato il contributo di tutte le N tuple.

In questo caso, secondo gli stessi ragionamenti della proposta di soluzione della domanda 1 con 4 ALU, avremmo:

- $T_{\omega PO} = t_p$ — *or dei 3 valori dei registri OWi*
- $T_{\omega PC} = 2t_p$ — *2 soli stati, 3 variabili di condizionamento, 2 livelli di logica*
- $T_{\sigma PC} = 2t_p$ — *idem come sopra*
- $T_{\sigma PO} = 5t_p + 2t_p$ — *2 ALU in cascata + K per scrivere registro*
- $\delta = t_p$ — *solita assunzione sul segnale di clock*

e quindi possiamo concludere che

$$T = t_p + (2\ t_p + 5\ t_p) + t_p = 9\ t_p$$

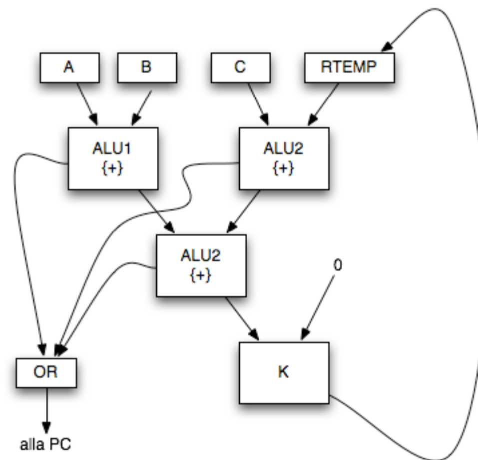
Questo porta ad un tempo medio pari a

$$9\ (1 + 2N)\ t_p \cong 18\ N\ t_p$$

Il tempo di ciclo è minore che nel caso precedente, ma le microistruzioni che si eseguono sono sostanzialmente il doppio. Dal momento che il tempo di ciclo non è la metà del tempo di ciclo precedente, il tempo medio di completamento è maggiore.

Variante 2

Si sarebbe potuto anche effettuare il test del valore *or*(OW_1, OW_2, ...) senza utilizzare registri, che peraltro andrebbero inizializzati, ma calcolando direttamente l'*or* sui flag prodotti in uscita dalle ALU. La cosa non ha impatto sul tempo del ciclo (si sposta il ritardo della valutazione da un termine all'altro ma il risultato non cambia).
In particolare, in questo caso, avremmo una parte operativa tipo:



In questo caso, il valore dell'OR non dipende da variabili di controllo in quanto le ALU fanno sempre la stessa operazione. Il costo computazionale della cascata delle ALU va contato nel $T_{\omega PO}$ dove dobbiamo contare un tempo necessario per stabilizzare le due ALU in cascata e l'OR a tre ingressi. Tuttavia, dal momento che il calcolo effettuato dalle ALU non dipende dai valori in ingresso alla PO (variabili di controllo) ma solo dallo stato interno, in questo caso il tempo di stabilizzazione delle ALU non deve essere considerato di nuovo nel $T_{\sigma PO}$, ma in questo tempo va solo considerato il ritardo introdotto dal commutatore K. Quindi complessivamente, il tempo di ciclo può essere stimato come:

$(5t_p + 5t_p + t_p)$	per la stabilizzazione del $T_{\omega PO}$ (si stabilizza anche l'ingresso di K, che però ancora deve ricevere i segnali di controllo dalla PC)
$(2t_p)$	per la stabilizzazione di $T_{\sigma PC}$
$(2t_p)$	stabilizzazione di K
t_p	scrittura nei registri (δ)

per un totale di 16 t_p, cioè lo stesso del caso con i registri per i flag di overflow. Notare come in questo caso la PO sia di fatto più semplice (tre registri da 1 bit in meno e tre commutatori in meno) ma il calcolo del tempo di ciclo, per essere eseguito correttamente, deve tener conto che una larga parte della porzione di PO mostrata in figura lavora indipendentemente dall'uscita prodotta dalla PC, e pertanto la formula delle dispense deve essere "interpretata" *cum grano salis*.