Architettura degli Elaboratori - Correzione

Appello del 6 giugno 2006, a.a. 2005/06

Domanda 1

Una unità di elaborazione U è connessa ad una gerarchia di memoria costituita da una memoria principale M di capacità 512M parole e da una memoria cache C di capacità 128K parole. U e l'unità che contiene C appartengono allo stesso chip. M è interallacciata con 8 moduli ed ha ciclo di clock uguale a 5 volte quello di U. C ha blocchi di 8 parole, opera su domanda ed è indirizzata con il metodo diretto. Tutti i collegamenti inter-chip hanno latenza di trasmissione uguale a 5 volte il ciclo di clock di U.

U riceve in ingresso messaggi (IND, DIM) dove IND è un indirizzo di M, e DIM è un intero positivo. U invia in uscita il numero degli elementi dell'array avente indirizzo base IND e dimensione DIM che hanno la proprietà di essere dispari.

Indicando con t_p è il ritardo di una porta logica con al massimo 8 ingressi, una ALU ha ritardo uguale a 5tp, e la durata dell'impulso di clock è uguale a t_p.

a. Descrivere le interfacce e scrivere il microprogramma di U.

b. Valutare il tempo medio di elaborazione di U in funzione di t_p, spiegando adeguatamente la risposta.

c. Discutere, sia nel caso generale che nel caso particolare, le implicazioni dell'adozione di una cache C indirizzata con il metodo associativo su insiemi a 2 vie.

Domanda 2

Si consideri il seguente pezzo di programma: da un file presente su un dispositivo FS viene letta una sequenza di N interi, memorizzate in un array di N posizioni; l'array viene successivamente elaborato mediante una funzione F e scritto come file nello stesso dispositivo con un altro nome:

```
int A[N], i, eof, pos=0;  …
do {
      read_file ("file_orig", pos, 4*N, A, eof);
      for (i = 0; i < N; i++)
            A[i] = F(A[i]);
      write_file ("file_dest", pos, 4*N, A);
      pos = pos + 4*N;
      } while not(eof); …
```

A livello del linguaggio delle applicazioni, il dispositivo FS è visto attraverso i comandi read_file(nome_file, posizione, n, x, esito) e write_file(nome_file, posizione, n, x), dove x è il nome di una variabile composta da n parole.

Il linguaggio concorrente Lc con cui è scritto il sistema operativo di OPS è quello a scambio di messaggi delle "Note sul livello dei processi".

a) Mostrare e spiegare la compilazione in Lc del programma in un processo APPL, supponendo che all'unità I/O_FS sia associato un processo Driver_FS. Per semplicità, non si considerino eccezioni generate dalle operazioni write_file, read_file.

b) Disegnare lo schema dei processi e dei canali coinvolti, e discutere le scelte effettuate riguardo il numero dei canali.

c) Spiegare in seguito a quali eventi il processo APPL può transire in stato di attesa, ed in seguito a quali eventi può essere risvegliato, considerando esplicitamente anche il caso in cui non esiste il processo Driver_FS.

d) Si compili in assembler il programma considerando che le primitive send e receive sono disponibili sotto forma di procedure.

Domanda 1 (traccia)

```
0. (RDYIN=0) nop, 0;
   (=1) set ACKIN, reset RDYIN, IND+1→BASE, IND→INDC, DIM-2→I, 0→COUNT,
        set RDYOUTC, 1

1. (I0,RDYINC,DATAIN31,ACKOUT=1111)  reset RDYINC, set RDYOUT,
                                     reset ACKOUT, COUNT+1→OUT, 0;
   (=1101) reset RDYINC, set RDYOUT, reset ACKOUT, COUNT→OUT, 0;
   (=10-1) nop, 1;
   (=1--0) nop, 1;
   (=00--) nop, 1;
   (=010-) reset RDYINC, I-1→I, BASE+1→BASE, BASE→INDC,
           set RDYOUTC,1;
   (=011-) reset RDYINC, COUNT+1→COUNT, I-1→I, BASE+1→BASE, BASE→INDC,
           set RDYOUTC,1
```

a) L'unità U comunica con U0 mediante le interface (RDYIN, IND, DIM, ACKIN), U1 mediante le interfacce (RDYOUT, COUNT, ACKOUT), la memoria cache C mediante le interface (RDYINC, OP, INDC, DATAIN, RDYOUTC). IND è dimensionato a 29 bit, I a 30 bit (segno e modulo).

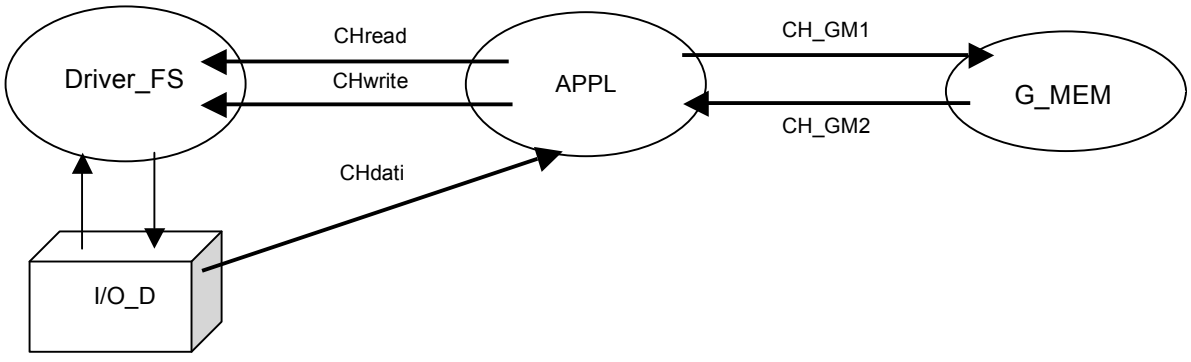
b) Il ciclo di clock si calcola in modo banale. Vista il numero ridotto di variabili di condizionamento e di micro-operazioni, entrambe le reti della PC si realizzano a due livelli di logica**. Nota bene:** Tutte le soluzioni algoritmiche ragionevoli utilizzano la funzione identità per ω_{PO} ($T\omega_{PO} = 0$). La soluzione canonica per la parte operativa impiega commutatori sia sulle ALU sia sui registri, quindi ha $T\sigma_{PO} = 9tp$. È possibile ottimizzare la parte operativa in modo che valga $T\sigma_{PO} = 7tp$. Questa soluzione è accettabile solo se adeguatamente spiegata e giustificata. Il tempo medio di elaborazione è banale da calcolare ($TME\_id = (DIM+1)*(\tau+Tc)$, $Tc = 1\tau$, $N_{fault} = N/8$, $T_{transf} = 23\tau$).

c) Nel caso generale un cache indirizzata con il metodo associativo su insiemi ha tempo di accesso superiore rispetto ad una equivalente indirizzata con il metodo diretto (2τ invece di 1τ, si noti che non è presente la MMU), costo di realizzazione leggermente superiore, e flessibilità decisamente superiore, paragonabile ad una cache completamente associativa (in particolare sono ridotti i conflitti nell'allocazione dei blocchi nella cache). Nel caso particolare, una cache indirizzata con il metodo associativo su insiemi (o anche associativo) non potrebbe nessun vantaggio perché non porterebbe alla diminuzione del numero di fault (l'algoritmo non esprime riuso), anzi porterebbe ad un peggioramento del tempo medio di elaborazione a causa del più alto tempo di accesso.

Domanda 2 (traccia)

Il processo Driver_FS dispone di due canali (asimmetrici) d'ingresso, di identificatori *CHread* e *CHwrite*, corrispondenti alle due operazioni da esso implementate per virtualizzare FS: attraverso *CHread* riceve la dimensione del blocco di dati e l'identificatore del canale *CHdati* su cui l'applicazione si aspetta di ricevere il blocco di dati (il cui valore verrà assegnato ad *A* oppure a *B*); attraverso *CHwrite* riceve la dimensione ed il valore del blocco di dati da inviare a D.

Lo schema a processi comunicanti della computazione L_c è mostrato nella figura seguente:



Per minimizzare il tempo di comunicazione dei valori degli array *A* e *B*, il canale *CHdati* ha come mittente direttamente il *processo esterno* I/O_FS. A questo scopo, il valore dell'identificatore *CHdati*, che Driver_D ha ricevuto in una variabile di tipo *channelname*, viene inoltrato da Driver_FS ad I/O_FS con il messaggio di richiesta di lettura del blocco. Poiché quando I/O_FS eseguità una *send* su *CHdati* troverà APPL in attesa, se (come di regola) l'implementazione delle *send-receive* fa uso di ottimizzazioni, la copia del valore di *A* o di *B* avverrà direttamente in tali variabili, minimizzando complessivamente il numero di copie.

Oltre ai canali di comunicazione con Driver_FS e I/O_FS, APPL utilizza i canali per il *trattamento dell'eccezione di fault di pagina* (*CH_GM1, CH_GM2*), mediante i quali comunica al processo gestore della memoria principale i parametri relativi ad una situazione di fault di pagina, ed attende la segnalazione che la pagina richiesta sia stata allocata e trasferita in M e che la Tabella di Rilocazione sia stata aggiornata.

Tutti i canali di ingresso di APPL sono asincroni di una posizione: ciò è sufficiente ad assicurare che i mittenti delle comunicazioni su tali canali non si sospenderanno mai nell'esecuzione delle *send* relative, trattandosi di colloqui a domanda e risposta. Pur non essendo visibile in APPL, anche i canali di ingresso degli altri processi conviene che siano asincroni.

La compilazione del processo applicativo nel linguaggio L_c è:

```
APPL:: channel in CHdati (1), CH_GM2 (1); channel out CHread, CHwrite, CH_GM1;
       <altre dichiarazioni>;

do {
      send (CHread, (CHdati, "file_orig", pos, 4*N);
      receive (CHdati, (A, eof));
      for (i = 0; i < N; i++)
            A[i] = F(A[i]);
      send (CHwrite, ("file_dest", pos, 4*N, A);
      pos = pos + 4*N;
      } while not(eof); …
```

Dove i parametri `"file_orig"` e `"file_dest"` possono avere come tipo una stringa di lunghezza costante (`char[M]`). Al fine di semplificare il punto d) si può assumere che tali parametri siano codificati con un valore numerico intero.

c) Il processo APPL non può mai sospendersi sulla primitiva *send(CHread, ...),* a condizione che *CHread* sia asincrono (è sufficiente il grado di asincronia uguale a uno), visto che il colloquio per implementare il comando *read_file* è a domanda e risposta. Lo stesso vale per la *send(CH_GM1, ...).*

APPL può sospendersi (si sospende) sulle primitive *receive(CHdati, ...)* e *receive(CH_GM2, ...)*, proprio in quanto il colloquio è a domanda e risposta.

APPL può inoltre sospendersi sulla primitiva *send(CHwrite, ...),* se l'esecuzione di tale *send* è ripetuta un numero di volte maggiore del grado di asincronia di *CHwrite* senza che il partner abbia ancora eseguito una *receive* corrispondente.

APPL verrà risvegliato dai partner delle comunicazioni, su cui si è sospeso, quando questi eseguiranno le primitive corrispondenti. Nel caso che il partner sia il processo esterno I/O_FS, la sveglia di APPL viene effettuata *in seguito all'interruzione* che I/O_FS invia quando, *e se*, rileva (attraverso informazioni nel descrittore di canale *CHdati*: campo *wait*) che APPL si trova in attesa. Il processo attualmente in esecuzione sul processore, una volta completata la fase firmware del trattamento interruzioni, esegue l'Handler che consiste proprio nella procedura di sveglia processo.

d) La compilazione *in assembler* produce, il codice del supporto a tempo di esecuzione di L_c: supporto delle primitive *send* e *receive*, incluso lo scheduling a basso livello, trattamento di interruzioni ed eccezioni. Il compilazione inizializza tutte le strutture dati del supporto di L_c. La compilazione del programma è banale: due loop annidati con test di fine ciclo in coda, e 4 chiamate di funzioni: send, receive, F, send, il cui indirizzo sarà messo a disposizione staticamente in alcuni registri dal compilatore. I parametri di queste funzioni saranno passati mediante registri generali. È necessario stabilire in quali registri le funzioni (specialmente send e receive) aspettano i parametri, e quali sono questi parametri in dipendenza della rappresentazione eseguibile delle primitive.