# Architettura degli Elaboratori, a.a. 2005-06

*Appello del 2 febbario 2006*

## Correzione

### Premessa

La domanda D1.a è trattata nelle Dispense, Cap. VII, sez. 2.2.3. Le domande D1.b e D1.c sono argomenti classici della progettazione di unità firmware. La domanda D2 è trattata in alcuni esercizi sulle gerarchie di memoria; una domanda analoga è presente nel compito del primo appello (vedere la correzione). La domanda D3 è descritta nelle Dispense per quanto riguarda l'impostazione generale, ed è stata oggetto di uno specifico esercizio (esattamente quello richiesto in D3) completamente svolto a lezione.

### Domanda 1

Una unità di elaborazione U ha una interfaccia d'ingresso ( RDYIN, ACKIN, OP (2 bit), IN1 (32), IN2 (32), IN3 (32) ) ed una interfaccia di uscita ( RDYOUT, ACKOUT, OUT (32 bit) ). L'unità è descritta dal seguente microprogramma:

0. (RDYIN = 0 ) nop, 0;

   ( = 1 ) reset RDYIN, set ACKIN, IN1 → A, IN2 → B, IN3 → C, IN3 – 1 → I, φ(OP) → RC

// esecuzione dell'operazione esterna con OP = 00 //

1. (segno(A – B) = 0) A + C → A, 2; ( = 1) B + C → B, 3
2. (ACKOUT = 0) nop, 2; ( =1 ) reset ACKOUT, set RDYOUT, $sh_d$(A) → OUT, 0
3. (ACKOUT = 0) nop, 3; ( =1 ) reset ACKOUT, set RDYOUT, $sh_d$(B) → OUT, 0

// esecuzione dell'operazione esterna con OP = 01 //

4. ($I_0$, ACKOUT = 0 - ) A – B → A, I – 1 → I, 4;

   ( = 1 0 ) nop, 4; ( = 1 1 ) reset ACKOUT, set RDYOUT, A → OUT, 0

// esecuzione dell'operazione esterna con OP = 1 – //

5. (ACKOUT = 0) nop, 5; ( =1 ) reset ACKOUT, set RDYOUT, A + B – C → OUT, 0

*a)* Relativamente all'operazione elementare φ(OP) → RC: spiegarne il significato ed, *in dettaglio*, i vantaggi e svantaggi della sua utilizzazione, darne una implementazione dettagliata, valutarne il ritardo di stabilizzazione, e ricavare l'espressione booleana della variabile di controllo associata a tale operazione.

*b)* Ricavare lo schema della Parte Operativa di U.

*c)* Valutare il ciclo di clock di U in funzione del ritardo $t_p$ di una porta logica con al massimo 8 ingressi. Una ALU ha ritardo uguale a $5t_p$ e la durata dell'impulso di clock è uguale a $t_p$.

*****

**a)** L'operazione elementare φ(OP) → RC adotta la tecnica del salto forzato per effettuare la decodifica del codice operativo OP dell'operazione esterna. La tecnica è descritta nelle Dispense, Cap. VII, sez. 2.2.3; nel seguito faremo riferimento allo schema di implementazione della fig. 4.

Il potenziale vantaggio della tecnica del salto forzato è la riduzione della complessità di progettazione della PC: riducendo di $n$ il numero degli ingressi della rete combinatoria ($\omega_{PC}$, $\sigma_{PC1}$) di fig. 4, la complessità di progettazione si abbassa dell'ordine di $2^n$. Nel nostro caso specifico, la riduzione in complessità è numericamente modesta, essendo OP di $n = 2$ bit: considerando solo le righe distinte, la tabella di verità delle funzioni di PC ha tre righe in meno rispetto alla versione in cui si testi esplicitamente OP. Il test esplicito di OP non avrebbe avuto nemmeno l'effetto di aumentare il ritardo delle funzioni di PC. **Nota***: il ritardo di stabilizzazione è un aspetto distinto dalla complessità di progettazione*.

Per contro, lo svantaggio è sensibile in termini di tempo di elaborazione: testando esplicitamente OP nella microistruzione 0 si risparmierebbe almeno un ciclo di clock su ogni operazione esterna: il guadagno è significativo per la prima (testando nella 0 anche segno (IN1 – IN2) si risparmierebbe un ulteriore ciclo di clock) e per la terza operazione esterna (che, testando anche ACKOUT, si concluderebbe direttamente nella 0).

La definizione della funzione φ può essere data solo in forma tabellare. Detti $Z_0$, $Z_1$, $Z_2$ i bit del secondo ingresso di $K_{RC}$ (fig. 4), la tabella di verità di φ è:

| $OP_0$ | $OP_1$ | $Z_0$ | $Z_1$ | $Z_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | – | 1 | 0 | 1 |

Le espressioni booleane delle variabili di uscita sono:

$$Z_0 = \overline{OP_0}\, OP_1 + OP_0$$
$$Z_1 = 0$$
$$Z_2 = \overline{OP_0}\, \overline{OP_1} + OP_0$$

Il ritardo di stabilizzazione di φ è $2t_p$, complessivamente $4t_p$ tenendo conto della presenza di $K_{RC}$.

L'espressione booleana della variabile di controllo $\alpha_{RC}$ (fig. 4) si ricava direttamente dal microprogramma:

$$\alpha_{RC} = \overline{y_0}\ \overline{y_1}\ \overline{y_2}\ RDYIN$$

con $y_0$, $y_1$, $y_2$ variabili dello stato interno presente.

**b)** In questa sede, per la PO dell'unità diamo soltanto lo schema delle reti di calcolo, tenendo conto che:

- la variabile di condizionamento *segno (A – B)* deve essere ottenuta come uscita secondaria di una ALU capace di eseguire solo A – B, affinché sia soddisfatta la condizione necessaria per la correttezza della PO;
- oltre alla suddetta ALU, ne servono altre due visto che il massimo parallelismo si ha tra A – B e I – 1;
- l'operazione A + B – C richiede due ALU in cascata, che possono essere fatte coincidere con le due suddette, scegliendo opportunamente ingressi e operazioni.

Una scelta è la seguente:

- $ALU_1$ esegue solo A – B;
- $ALU_2$ esegue le operazioni (+,–, $sh_d$), commutando sul primo ingresso A e B, e sul secondo ingresso C e B;
- $ALU_3$ esegue le operazioni (–, –1), commutando sul primo ingresso IN3, I e l'uscita di $ALU_2$, ed avendo come secondo ingresso C.

**c)** Il ritardo della funzione $\omega_{PO}$ è quello di ALU1, quindi $5t_p$. Il ritardo di $\omega_{PC}$ è uguale a $2t_p$, in quanto il massimo numero di variabili in ingresso a una porta AND è 5 ed il massimo numero di termini messi in OR è certamente minore di 8. Il ritardo complessivo di $\sigma_{PC}$ è $4t_p$, dove

- una parte (ritardo $2t_p$), relativa a $\sigma_{PC1}$, si stabilizza in parallelo alla $\omega_{PC}$, oppure, se relativa a φ, si stabilizza in parallelo alla $\omega_{PO}$,
- l'altra parte (ritardo $2t_p$), relativa a $K_{RC}$, si stabilizza in parallelo alla $\sigma_{PO}$.

Il ritardo di $\sigma_{PO}$ è quello dell'operazione elementare A + B – C → OUT, che comporta la stabilizzazione di due ALU e tre commutatori a due livelli di logica, quindi $16t_p$.

Il ciclo di clock vale dunque:

$$\tau = T_{\omega_{PO}} + T_{\omega_{PC}} + T_{\sigma_{PO}} + \delta = 24t_p$$

## Domanda 2

Il programma mostrato (prodotto di matrici righe per colonne) è eseguito da un elaboratore la cui CPU contiene una cache primaria operante su domanda, capacità 64K parole, blocchi di 16 parole, e scritture gestite con il metodo Write-Through.

Determinare l'insieme di lavoro del programma, spiegare in che modo è possibile che questo risieda interamente in cache e, in tale ipotesi, determinare il numero medio di fault di cache.

```
int A[N][N], int B[N][N], int C[N][N];
        { for (i = 0; i < N; i ++)
                for (j = 0; j < N; j ++)
                        { C[i]][j] = 0;
                          for (k = 0; k < N; k ++)
                                C[i][j] = C[i][j] + A[i][k] * B[k][j]
                        }
        }
```

*****

Il programma ha tempo di elaborazione $\Theta(N^3)$, ed in particolare effettua $\sim 2N^3$ accessi in memoria.

L'array A è di $N^2$ interi e quindi di $N^2/\sigma$ blocchi; gli accessi ad A provocano $N^2/\sigma$ fault, essendo la cache operante su domanda.

Sempre perché la cache opera su domanda, non è possibile evitare che, durante la prima iterazione del ciclo più esterno (controllato da *i*), gli accessi a B provochino $N^2/\sigma$ fault. Si verificano anche $N^2/\sigma$ fault per l'accesso a C, che comunque non hanno effetto sul tempo di accesso in memoria.

L'insieme di lavoro è costituito da un blocco di istruzioni, un blocco di A, tutto B ($N^2/\sigma$ blocchi), ed un blocco di C.

La condizione affinché l'insieme di lavoro risieda interamente in cache, una volta acquisito, è che i blocchi di B non vengano sostituiti durante l'esecuzione del programma: Questo comporta:

*1)* che tale insieme di lavoro sia di capacità minore di 64K parole; allo scopo, N deve essere $\leq 255$;

*2)* che l'architettura della macchina assembler e della macchina firmware permettano di specificare che alcuni blocchi non devono essere sostituiti; questo si può ottenere con istruzioni assembler speciali o con notazioni aggiunte alle istruzioni di LOAD/STORE; l'informazione contenuta in tali istruzioni viene trasferita all'unità cache, la quale la ricorda nella tabella usata per la traduzione dell'indirizzo e controllo fault, nelle entrate relative ai blocchi in questione.

In tali condizioni, il numero di fault è $\sim 2\ N^2/\sigma = N^2/8$.

## Domanda 3

Scrivere *e spiegare* il trattamento dell'eccezione di fault di pagina relativamente a tutte le funzionalità eseguite nello spazio di indirizzamento del processo che ha rilevato l'eccezione, mostrando in dettaglio tutte le strutture dati usate da tali funzionalità. Valgono le seguenti caratteristiche:

- la gestione della memoria principale è effettuata da un processo al quale, nel caso di eccezione di fault di pagina, è interamente demandato il compito di: scegliere la pagina da sostituire, provocare la sostituzione di tale pagina, aggiornare la tabella di rilocazione del processo che ha rilevato l'eccezione, e segnalare a tale processo il completamento con successo della suddetta sequenza di azioni;
- la cooperazione tra processi è descritta mediante il linguaggio concorrente Lc delle "Note sul livello dei processi";
- le primitive di comunicazione tra processi sono disponibili al compilatore sotto forma di procedure.

*****

Le funzionalità eseguite nello spazio di indirizzamento del processo che ha rilevato l'eccezione comprendono la fase firmware e tutta la fase assembler, incluso l'Handler specifico.

La fase firmware è (se ESITO è stato salvato nel registro firmware ESITO1):

*tratt_ecc. ESITO1 → RG[Resito], IND → RG[Rindirizzo], IC → RG[Rritorno], RG[Routine] → IC, ch0*

In questo modo, sono stati passati, alla Routine di interfaccia verso gli Handler del Trattamento Eccezioni, solo i parametri strettamente necessari a questo livello, cioè quelli visibili solo a livello firmware (codice dell'eccezione e indirizzo logico della richiesta di accesso in memoria che ha provocato l'eccezione). Altri parametri noti a livello assembler (ad esempio, indirizzo del PCB, della Tabella di Rilocazione, ecc) potranno essere ricavati successivamente, in quanto tutta la fase assembler è eseguita, per definizione, nello spazio logico di indirizzamento del processo che ha provocato l'eccezione.

La fase assembler, eseguita a interruzioni disabilitate, è annidata nella Routine di interfaccia:

```
LOAD   RTabEcc, Resito, Rhandler, DI
CALL   Rhandler, Rret
GOTO   Rritorno, EI
```

Struttura dati usata: Tabella degli Handler, il cui indirizzo è contenuto nel registro di indirizzo RTabEcc.

La procedura dello Handler per l'eccezione di fault di pagina esegue le seguenti azioni in linguaggio Lc:

```
channel in CH_GM2 (1), …; channel out CH_GM1, …;
send ( CH_GM1, ( CH_GM2, esito, indirizzo, indirizzo_TabRil) );
receive ( CH_GM2, ( ) )
return
```

con CH_GM1 e CH_GM2 canali verso e dal processo Gestore della Memoria Principale.

Nella memoria virtuale del processo il compilatore alloca, in particolare, le seguenti strutture:

- array di costanti *inizializzate* ai valori degli identificatori di tutti i canali usati dal processo; l'indirizzo base di questo array è contenuto nel registro di indirizzo Rcanali. Supponiamo che CH_GM1 e CH_GM2 siano i primi due elementi di questo array;
- messaggio dato dalla quadrupla ( CH_GM2, esito, indirizzo, indirizzo_TabRil), con indirizzo base contenuto nel registro di indirizzo RmsgGM;
- i descrittori dei canali CH_GM1 e CH_GM2 puntati dalla Tabella dei Canali di indirizzo noto;
- la variabile targa in cui copiare il messaggio al Gestore della Memoria, se trovato in attesa nell'esecuzione della *send* su CH_GM1;
- il PCB del processo, il cui indirizzo base è contenuto nel registro di indirizzo Rpcb, e lo spazio per i PCB di tutti gli altri processi, incluso quello del Gestore della Memoria che può essere riferito dal processo per l'eventuale sveglia nell'esecuzione della *send*. Supponiamo che la Tabella di Rilocazione sia puntata dalla prima posizione del PCB.

I due parametri delle procedure *send* e *receive* sono passati attraverso due registri generali: Rchid (identificatore del canale di comunicazione) e Rmsg (indirizzo della variabile messaggio o targa).

La compilazione dello Handler è la seguente:

```
// passaggio del primo parametro della send //
    LOAD   Rcanali, 0, Rchid
// formazione del messaggio in memoria //
    LOAD   Rcanali, 1, Rchgm2
    STORE  RmsgGM, 0, Rchgm2
    STORE  RmsgGM, 1, Resito
    STORE  RmsgGM, 2, Rindirizzo
    LOAD   Rpcb, 0, RTabRil
    STORE  RmsgGM, 3, RTabRil
// passaggio del secondo parametro della send //
    MOVE   RmsgGM, Rmsg
```

```
// chiamata della procedura send //
        CALL   Rsend, Rret1
// passaggio del primo parametro della receive //
        MOVE   Rchgm2, Rchid
// chiamata della procedura receive; il secondo parametro non corrisponde ad alcun oggetto in memoria //
        CALL   Receive, Rret1
// ritorno alla Routine di interfaccia //
        GOTO   Rret
```

Il processo Gestore della Memoria effettua le azioni specificate nel testo. La *send* su CH_GM2 avrà come effetto di risvegliare il processo che si era sospeso sulla *receive* da CH_GM2 (non si era, invece, sospeso sulla *send* su CH_GM1, se questo è asincrono con grado di asincronia uguale a uno).