Architettura degli Elaboratori

Prima prova di verifica intermedia - A.A. 2014-2015

Riportare Nome, Cognome, Numero di matricola e Corso di appartenenza su tutti i fogli consegnati.
I risultati saranno resi noti sulle pagine web dei docenti appena disponibili

Esercizio 1

Una unità di elaborazione U contiene una componente memoria A di 64 posizioni e comunica, mediante interfacce dotate di sincronizzatori a transizione di livello, con una unità U_1 e una unità U_2. U riceve da U_1 tre valori, COP (2 bit) e I (6 bit) e V (32 bit) ed (solo in certi casi) invia ad U_2 un valore Y (32 bit). L'unità U esegue le seguenti operazioni esterne:

- se COP= 0, assegna V a tutte le locazioni di A. In questo caso I non è significativo, e non vi è nessuna comunicazione verso U_2;
- se COP=1, assegna V ad A[I]. In questo caso non vi è nessuna comunicazione verso U_2;
- se COP=2, U trasmette ad U_2 A[I];
- se COP=3, U trasmette ad U_2 il massimo dei valori contenuti in A e decrementa tale valore di I unità.

Progettare la PO a partire dal microprogramma eseguibile e successivamente:

- indicare il numero di ingressi e di uscite della PC,
- calcolare il ciclo di clock τ dell'unità, e
- calcolare il tempo medio di elaborazione delle operazioni esterne,

sapendo che

- il ritardo della rete che calcola le uscite della PC ($T_{\omega PC}$) è pari a $2t_p$,
- il ritardo della rete che calcola il prossimo stato della PC ($T_{\sigma PC}$) è pari a $2t_p$,
- il tempo di accesso in lettura del componente memoria A è $2t_p$, e quello in scrittura è t_p,
- il ritardo di stabilizzazione della ALU è $5t_p$.

Esercizio 2

Si consideri il seguente frammento di codice assembler D-RISC:

```
1.              CLEAR R1
2.              CLEAR R2
3.              CLEAR R3
4.              ADDI R0, #128, R4
5.              ADDI R0, #64, R3
6.  ciclo:      IF= R1, R4 fine
7.              LOAD R3, R1, R5
8.              ADD R2, R5, R2
9.              INCR R1
10.             GOTO ciclo
11. fine:       STORE R3, R0, R2
12.             END
```

e se ne fornisca il numero di accessi in memoria effettuati durante la sua esecuzione.

Facoltativo: fornire una versione ottimizzata del codice.

Traccia di soluzione

Esercizio 1

L'unità U ha un'interfaccia verso U1 costituita dai registri I (6 bit), COP (2 bit) e V (32 bit) in ingresso, dal sincronizzatore a transizione di livello in ingresso RDYin, e dal sincronizzatore a transizione di livello in uscita ACKin. L'interfaccia verso U2 avrà il registro in uscita Y (32 bit), il sincronizzatore a transizione di livelli in uscita RDYout e quello in ingresso ACKout.

Nel caso delle prime due operazioni esterne non vi è comunicazione con U2. La prima e la quarta operazione esterna richiede un ciclo che scorra tutta la memoria A.

Il microprogramma, assumendo che C sia un registro da 7 bit (C_0 è il bit più significativo e C_m sono i 6 bit meno significativi, come al solito), potrebbe essere dunque scritto come segue:

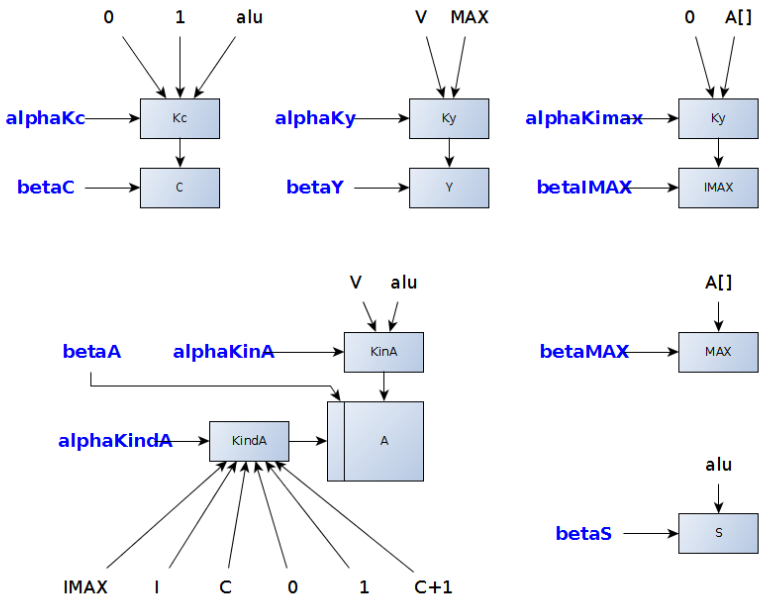
0. // se non ci sono ingressi attendine uno
   (RDY_{in}, COP, ACK_{out}=0---) nop, 0
   // prima operazione esterna: inizializza il contatore e vai al loop
   (=100-) 0→C, 1
   // seconda operazione esterna: esegui assegnamento e torna all'inizio
   (=101-) V→A[I], reset RDY_{in}, set ACK_{in},0
   // terza operazione esterna: si completa in un ciclo, se non c'è l'ACK a scrivere
   // attendi che venga comunicato
   (=1100) nop,0
   // quando arriva, concludi l'operazione scrivendo sul registro di uscita e torna alla 0.
   (=1101) A[I]→Y, reset RDY_{in}, set ACK_{in}, reset ACK_{out}, set RDY_{out}, 0
   // quarta operazione, inizializza massimo al primo elemento, quindi preara il test
   // sul secondo elemento e vai all'assegnazione del flag S
   (=111-) A[0]→MAX, 0→IMAX, 1→C, 3
1. // prima operazione: assegna il valore V alla posizione corrente di A e continua
   (C_0=0) V→A[C_m], C+1→C, 1;
   // se siamo alla fine, concludi l'operazione e torna alla 0
   (=1) set ACK_{in}, reset RDY_{in}, 0
2. // quarta operazione; iterazione i=> controlla se A[C] è il nuovo Massimo
   // se non lo è prosegui il ciclo e prepara il nuovo valore da testare
   (C_0,S,ACK_{out}=00-) C+1→C, segno(MAX-A[C+1])→S , 2
   // se lo è, prosegui ma aggiorna massimo e indice del massimo
   (=01-) C+1→C, segno(MAX-A[C+1])→S, C→IMAX, A[C]→MAX, 2
   // se sei alla fine, attendi la disponibilità a ricevere massimo da parte di U2
   (=1-0) nop, 2
   // mandala, chiudi la comunicazione e torna alla 0
   (=1-1) MAX →Y, A[IMAX] - I →A[IMAX], reset ACK_{out}, set RDY_{out}, 0
3. // assegna il falg che controlla se il massimo è nella posizione
   segno(MAX-A[1])→S, 2

Le uscite verso la PC sono le 5 variabili di condizionamento:

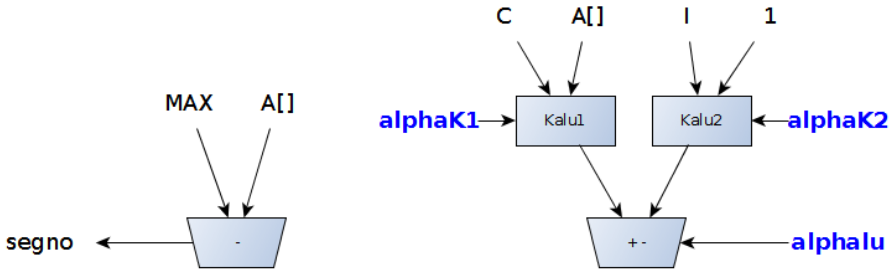
$$RDY_{in}, COP, ACK_{out}, C_0, S$$

Gli ingressi dalla PC sono gli α e β necessari a comandare la parte operativa, che progettiamo in

accordo con il procedimento formale visto a lezione. Le classi di operazioni per i registri sono:



Le classi di operazioni elementari (con le relative strutture) saranno invece:



Dunque possiamo concludere che gli ingressi dalla PC saranno costituiti da tutti gli α e β riportati in figura per un totale di 17 ingressi (α_{KindA} e α_{Kc} sono da 3 e 2 bit, rispettivamente).

Il ciclo di clock lo calcoliamo considerando la maggiorazione data dalla formula

$$\tau = T_{\omega PO} + \max\{T_{\sigma PC}, (T_{\omega PC}+T_{\sigma PO})\} + \delta$$

considerando i seguenti valori:

- $T_{\sigma PO} = t_k + t_a + t_{alu} = 2t_p + 2t_p + 5t_p = 9t_p$ (per le segno(MAX-A[…])))
  La A[IMAX] - I →A[IMAX] è ugualmente complessa. Infatti per il calcolo della parte destra richeide lo stesso tempo. Il selettore per la scrittura stabilizza in parallelo e dunque il $T_{\sigma PO}$ è

sempre lo stesso
- $T_{\omega PO} = 0$.

Considerando che $T_{\omega PC} = T_{\sigma PC} = 2t_p$, avremo che

$$\tau = T_{\omega PO} + \max\{T_{\sigma PC}, (T_{\omega PC}+T_{\sigma PO})\} + \delta = 0 + \max\{2t_p,2t_p+9t_p\}+t_p = 13\ t_p$$

Non avendo variabili di condizionamento complesse, il calcolo "per frasi" del ciclo di clock (ovvero il calcolo del ciclo di clock come massimo dei tempi relativi a tutte le frasi nel microcodice) avrebbe portato allo stesso risultato.

Le operazioni esterne richiedono (al netto delle attese sull'unità che invia e su quella che riceve) 1+N+1, 1, 1 e 1+1+N+1 τ, rispettivamente (la seconda e la terza operazione si concludono nella 0., la prima e la quarta operazione esterna eseguono la 0., poi la 1. o la 2. La quarta operazione esegue anche la 3. N volte (N=64) e infine la 1. o la 2. una volta per inviare l'uscita). In questo conto non consideriamo tempi di attesa sulle unità esterne, ovvero consideriamo che nella 0. non sia mai RDY_{in}=0 e nella 2. non sia mai ACK_{out}=0.

Volendo dare un tempo medio di esecuzione, a parità di probabilità per le quattro operazioni

$$T = ((2+N) + 1 + 1 + (3+N))\ \tau\ /4 = (7+2N)\ \tau\ /4$$

$$= 135\ \tau\ /4 = 33.75\tau$$

Si sarebbe potuto utilizzare nel microprogramma una variabile di condizionamento complessa per la quarta operazione esterna. In questo caso avremmo potuto utilizzare un codice tipo

0. (RDY_{in}, COP, ACK_{out}=0---) nop, 0; *// attesa ingressi*
   …
   (=111-) 0 →MAX, 0→C, I→TEMP, reset RDY_{in}, set ACK_{in}, 1; *// quarta operazione*
1. (C_0, segno(MAX-A[C_m])=00) C+1→C, 2; *// controllo se maggiore*
   (=01) A[C_m]→MAX, C_m → IMAX, C+1→C, 2; *// eventuale cambio max*
   (=1-) …
2. …

In questo caso però occorreva garantire la condizione di correttezza utilizzando una memoria a doppia porta, con un indirizzo utilizzato (senza commutatori) per realizzare la segno(MAX-A[C_m]).

Esercizio 2

Il codice implementa un ciclo eseguito per 128 volte (variabile che conta il numero delle iterazioni in R1, inizializzazione di R4 a 128 (si ricordi che R0 vale sempre 0, come da definizione dell'assembler D-RISC), controllo se R1=R4 per testare la fine del ciclo nella 6.). Le istruzioni dalla 1. alla 5. vengono dunque eseguite una volta sola, come la 11. e la 12. Le istruzioni dalla 6. alla 10. vengono invece eseguite 128 volte. Nel corpo del ciclo c'è una singola istruzione che accede dati in memoria

(la LOAD 7.). Fuori dal corpo c'è un'unica istruzione che accede dati in memoria (la STORE 11.).

Complessivamente abbiamo dunque:

- 5 + 128(5) + 2 accessi alla memoria per il fetch delle istruzioni, e
- 128 + 1 accessi per i dati

Per un totale di 8 + 6*128 = 776 accessi.

Parte facoltativa:

Il codice del ciclo poteva essere ottimizzato sapendo che il corpo viene eseguito un numero di volte strettamente maggiore di 1, utilizzando lo schema visto a lezione:

etichetta: <compilazione corpo>
<incremento variabile d'iterazione>
<se nel range, vai a etichetta>
<istruzione che segue il ciclo…>

Quindi trasformando il codice del testo nel codice:

```
0.              CLEAR R1
1.              CLEAR R2
2.              CLEAR R3
3.              ADDI R0, #128, R4
4.              ADDI R0, #64, R3
5.  ciclo:      LOAD R3, R1, R5
6.              ADD R2, R5, R2
7.              INCR R1
8.              IF< R1, R4, ciclo
9.  fine:       STORE R3, R0, R2
10.             END
```

otteniamo una istruzione in meno nel corpo del ciclo, che ci fa risparmiare 128 operazioni di fetch in memoria.

La 2. era superflua, visto che nella 4. si andava a scrivere immediatamente un altro valore in R3 senza che il valore precedente fosse stato letto. Dunque si poteva ottimizzare il codice ancora (senza un grande effetto sul numero totale delle istruzioni eseguite, peraltro) come segue:

```
0.              CLEAR R1
1.              CLEAR R2
2.              ADDI R0, #128, R4
3.              ADDI R0, #64, R3
4.  ciclo:      LOAD R3, R1, R5
5.              ADD R2, R5, R2
6.              INCR R1
7.              IF< R1, R4, ciclo
8.  fine:       STORE R3, R0, R2
9.              END
```