Architettura degli elaboratori – A.A. 2017–18
Quarto appello – 5 luglio 2018

Riportare su tutti i fogli consegnati in alto a destra: nome, cognome, matricola e corso di appartenenza.
I risultati saranno comunicati via web appena disponibili, contestualmente al calendario degli orali.

Domanda 1

Una unità firmware U_{buf} realizza un buffer che implementa una politica LIFO fra due unità U_1 e U_2. U_1 spedisce parole (interi da 32 bit) verso U_{buf} che li passa ad U_2, quando questa ne fa richiesta. U_{buf} utilizza una memoria interna M da 1K parole per mantenere i dati spediti da U_1 e non ancora richiesti da U_2.
Si richiede di progettare U_{buf} e di calcolarne la lunghezza del ciclo di clock τ in funzione di t_p, t_{alu} e t_a, tempi di stabilizzazione delle uscite di una porta logica con 8 ingressi, di una ALU e tempo di accesso della memoria M.

Domanda 2

Un sistema di elaborazione è costituito da processore, MMU, cache di primo livello (C1, unica per dati e istruzioni, associativa a 2 vie con linee da 16 parole) e memoria principale M (da 2G, interallacciata, 4 moduli).
Il sistema esegue un programma che computa *prefisso(+)* di un vettore in memoria. La funzione *prefisso(⊕)* applicata ad un vettore $x_0, x_1, \ldots, x_{n-1}$ calcola $x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$.
Si chiede di fornire la traccia degli indirizzi utilizzata per gli accessi nella cache e di determinare, motivando la risposta adeguatamente, il numero di fault in C1, con o senza ottimizzazioni.

Domanda 3

Si assuma di avere a disposizione porte logiche da 4 ingressi che stabilizzano in un n*sec*. Fornire il tempo di accesso alla memoria M della domanda 1 realizzata utilizzando questo tipo di porte.

Traccia di soluzione

Domanda 1

Il controllo dell'unità considera, nell'ordine, le seguenti situazioni:

1. U1 sta spedendo una parola e U2 ne sta richiedendo una → la parola spedita da U1 è immediatamente passata ad U2, secondo la politica LIFO
2. U1 non sta spedendo, U2 richiede una parola e la memoria locale non è vuota → si passa ad U2 l'ultima parola memorizzata in M
3. U2 non richiede una parola, U1 sta spedendo una parola e la memoria interna non è piena → la parola spedita da U1 è memorizzata sullo stack implementato nella memoria interna M
4. U1 tenta di spedire con memoria piena e U2 che non sta richiedendo una parola, U2 richiede una parola con memoria vuota e U1 non sta spedendo → attesa.

Assumiamo che entrambe le unità interagiscano con Ubuf utilizzando indicatori a transizione di livello. L'interfaccia vero U1 sarà costituita da URDY1, UACK1, IN, rispettivamente indicatori a transizione di livello in ingresso e uscita e registro in ingresso da 1 parola. Quella verso U2 da URDY2, UACK2 e OUT, rispettivamente indicatori a transizione di livello in ingresso e uscita e registro in uscita da 1 parola. U1 richiederà la spedizione di una parola verso Ubuf settando URDY1. U2 chiederà di ricevere una parola da Ubuf settando URDY2.

Per la memoria interna M, gestita come uno stack, utilizziamo due registri: TOP (10 bit), che punta sempre alla prima posizione vuota, se lo stack non è pieno, e N (11 bit), che contiene il numero di elementi presenti nello stack. Entrambi sono inizializzati a 0. La condizione di memoria piena corrisponderà a N0=1, quella di coda vuota corrisponderà a OR(N)=0.

Il microcodice sarà dunque del tipo:

0. (URDY1,URDY2,OR(TOP),TOP0=00--) nop, 0
   (=1011) nop, 0
   (=010-) nop, 0
   (=11--) IN→OUT, reset URDY1, set UACK1, reset URDY2, set UACK2, 0
   (=10-0) IN → M[TOP], TOP+1→TOP, reset URDY1, set UACK1, 0
   (=011-) M[TOP-1] → OUT, TOP-1→TOP, reset URDY2, set UACK2, 0

Visto che abbiamo una sola microistruzione, l'unità firmware può essere realizzata con una unica rete sequenziale (la PO) cui aggiungiamo una rete combinatoria per calcolare le variabili di controllo. Detta rete ha 4 ingressi (le variabili di condizionamento) e genera gli alpha e beta necessari per l'esecuzione delle varie frasi, che sono 6. Quindi abbiamo un livello AND con 4 ingressi (1tp) e un livello OR con 6 ingressi max (ancora 1tp) → totale 2tp.

Consideriamo il microcodice:

1. risorse di stato: OUT, TOP, M. OUT ha un commutatore in ingresso che sceglie fra IN e uscita di M, TOP ha un solo ingresso che è uscita di una ALU, dunque non necessita di

commutatore, M ha in ingresso solo IN e quindi non necessita del commutatore, ma come ingressi ha TOP o TOP-1 quindi serve un commutatore Kind ($\alpha_{out,}$ α_{Kind})

2. Risorse di calcolo: per la quinta e sesta frase serve una ALU, in grado di fare entrambe sia incrementi che decrementi (α_{alu})
3. Indicatori a transizione di livello: tutti richiedono un beta per set o reset (β_{rdy1},β_{rdy2},β_{ck1},β_{ack2})

Le variabili di condizionamento sono solo 4. OR(N) richiede 2tp per fare l'OR di 32 ingressi. Alpha e Beta, una volta note le var di condizionamento, sono calcolati in 1tp (max 4 ingressi). Il percorso più lungo nella sigma PO è il max fra la scrittura in un registro con un commutatore (2tp) e un tempo di accesso alla memoria che utilizza un risultato di una ALU come indirizzo (ta+talu-tk).

Dunque il ciclo di clock può essere calcolato come

$\tau = (T\omega PO + T\omega PC) + T\sigma PO + \delta = (2tp + 2tp) + \max\{2tp, ta+talu+tk\} + tp = 7tp + talu + ta$

tenendo conto del fatto che ta + talu +tk > 2tp (ovviamente).

Domanda 2

Il prefix si calcola con lo pseudocodice

```
for(int i=1; i<N; i++)
        a[i] = a[i]+a[i-1];
```

che compila nel codice D-RISC:

```
init:   ADD R0, #1, Ri
loop:   LOAD Rbase, Ri, R1
        SUB Ri, #1, Ri1
        LOAD Rbase, Ri1, R2
        ADD R1, R2, R1
        STORE Rbase, Ri, R1
        INC Ri
        IF< Ri, RN, loop
cont:   END
```

Senza ottimizzazioni, la traccia degli indirizzi generati verso la cache sarà:
μ(init), μ(loop), μ(Rbase+1), μ(loop+1), μ(loop+2), μ(Rbase), μ(loop+3), μ(loop+4), μ(Rbase), μ(loop+5),μ(loop+6), μ(loop), μ(Rbase+2), μ(loop+1), …

dove μ(x), è il risultato della traduzione dell'indirizzo logico x in un indirizzo fisico, effettuata utilizzando la tabella di rilocazione.

Per gli accessi in memoria valgono località e riuso sia per il codice che per l'array A. Senza ottimizzazioni, c'è un fault per il codice e N/sigma fault per i dati. Se modifichiamo che LOAD in aggiungendo il flag "prefetch" avremo il solo fault iniziale per l'array A, riducendo quindi il numero di fault a 2.

Domanda 3

Una memoria da 1K ha il selettore di scrittura che da 10 bit di controllo e un ingresso (il beta) passa il valore di beta su una delle K uscite. Dunque abbiamo 11 bit in ingresso al livello AND, senza livello OR. Con porte da 4 ingressi servono due livelli (parte intera superiore del log in base 4 di 11). Per il commutatore di lettura, abbiamo K ingressi, 10 bit di controllo e un bit di uscita (commutatore da 1 bit replicato 32 volte per ottenere la parola). Dei K ingressi, solo uno è significativo, dunque il livello AND ha 11 ingressi, ma il livello OR ha 1K ingressi. Dunque servono due livelli di porte AND e 5 livelli OR per un totale di 7 livelli.

Il tempo di accesso di M sarà dunque di 7 *nsec*.