Architetture degli elaboratori A.A. 2010-2011
6° appello – 7 febbraio 2012

Riportare su ogni foglio, nome, cognome, matricola, corso e programma (OLD-0/1, NEW). I risultati saranno pubblicati sulle pagine WEB dei docenti appena disponibili. Traccia e bozza di soluzione su didawiki.

Domanda 1 (tutti)

Una unità U è interfacciata con due unità P_{in} e P_{out} e con un sottosistema di memoria M di capacità 2G parole.

U riceve da P_{in} una parola X da 32 bit, la considera divisa in due campi (i 20 bit più significativi nel campo A e i 12 bit meno significativi nel campo B) ed invia all'unità P_{out} la parola Y ottenuta sostituendo in X A con il valore TAB[A].

La tabella TAB è contenuta interamente nella memoria M a partire da un indirizzo contenuto nel registro INDTAB interno ad U. Per velocizzare il calcolo, U contiene una memoria C da 64 posizioni organizzata come una cache ad accesso diretto con σ =1. Il calcolo di TAB[A] avviene utilizzando C: nel caso in cui C non contenga l'entry di TAB relativa ad A, U provvede ad aggiornare l'informazione in C accedendo alla tabella TAB in M prima di inviare Y a P_{out}.

Sia $4t_p$ il tempo di accesso di C e $20t_p$ il tempo di accesso a M, sullo stesso chip di U, e 5tp il tempo di stabilizzazione della ALU. Si fornisca il microprogramma dell'unità U avendo cura di ottimizzare il tempo necessario alla traduzione di X in Y e si determinino T e τ.

Domanda 2 (tutti)

Si consideri il problema della ricerca di un elemento in una lista realizzata mediante la tecnica dei doppi puntatori. Ogni elemento della lista è formato da una parola VAL che contiene il valore dell'elemento seguita dai puntatori NEXT e PREV rispettivamente all'elemento successivo e precedente nella lista.

Si compili in assembler D-RISC una procedura, che accetta due parametri in ingresso (*l'indirizzo della struttura dati contenente il puntatore* al primo elemento della lista, e il valore da ricercare) e restituisce un intero (0 non trovato, 1 trovato) dopo aver rimosso (se presente) il valore cercato dalla lista. Si assume che l'elemento ricercato, se presente, sia presente in un sola posizione della lista, non in quella iniziale, e che il passaggio dei parametri avvenga mediante registri.

Si fornisca il tempo di completamento per la procedura in funzione della lunghezza della lista (N), assumendo che nella metà dei casi la ricerca avvenga per un valore non presente nella lista.

NEW, OLD-0: la lista è privata del processo e l'architettura della CPU è pipeline scalare.

OLD-1: la lista è *condivisa* tra più processi, con puntatori condivisi realizzati secondo il metodo degli indirizzi logici distinti, e l'architettura della CPU è tradizionale.

In entrambi i casi, la cache dati primaria è completamente associativa, su domanda, di capacità 32K parole e blocchi di 8 parole, e la cache secondaria è on-chip e contiene l'intera lista.

Bozza di soluzione

Domanda 1

Consideriamo dapprima la memoria C. Ogni campo di C dovrà contenere:

1) Un bit di validità, V, che indica se la posizione è occupata o meno
2) Un campo TAG, di 14 bit (20-6)
3) Un campo W da 20 bit (σ=1, quindi questo è il campo che contiene il valore di TAB[A], se i TAG corrispondono e il bit di validità è a 1)

Per tradurre X in Y si procede come segue:

a) Si accede alla memoria C e si controlla se in posizione X.A%64 (indirizzamento diretto) sia presente nel campo TAG il valore X.A/64 (gestione dell'informazione come in una cache).
   - Se il campo C[X.A%64].TAG è diverso da A/64, l'entry non è relativa al valore X corrente. Pertanto si legge il valore M[INDTAB+A] e si aggiorna la entry con TAG=A/64, V=1 e W=M[INDTAB+A]
b) Si calcola Y = C[X.A%64].W .. X.B e lo si invia all'unità P_{out}.

Per ottimizzare il tempo di calcolo della traduzione cerchiamo di minimizzare il tempo relativo alla traduzione in caso di presenza delle informazioni necessarie nella cache C e ottimizziamo al massimo tutti i tempi necessari al calcolo delle variabili di condizionamento.

Il microprogramma dell'unità può essere scritto come segue:

0. (RDY_{in},ACK_{out},uguali(C[(X.A)%64].TAG, (X.A)/64),C[(X.A)%64].V=0---) nop, 0
   (=1011) nop, 0
   (=1111) set ACK_{in}, reset RDY_{in}, set RDY_{out}, reset ACK_{out}, (C[(X.A)%64].W)..(X.B)→Y, 0
   (=1-0-,1-10) set ACK_{in}, reset RDY_{in}, INDTAB+(X.A)→IND, "*read*"→OP,set RDY_{Mout}, 1
1. (RDY_{Min},ACK_{out}=0-,10) nop, 1
   (=11) reset RDY_{Min}, set RDY_{out}, reset ACK_{out}, DATAIN..(X.B)→Y,
   <X.A/64, 1, DATAIN> →C[(X.A)%64], 0

Ai fini della minimizzazione del tempo $T_{\omega PO}$, consideriamo il confronto fra i due valori a 14 bit. Tale confronto può essere effettuato con 14 confrontatori in parallelo ($2t_p$) i cui valori vengono posti in OR (2 livelli con porte da 8 ingressi max, quindi altri $2t_p$) per un totale di $4t_p$. Il costo delle operazioni %64 e /64 è ovviamente 0 (accesso a parti di un registro). Il tempo di accesso alla memoria C è di $4t_p$, quindi il costo complessivo della $T_{\omega PO}$ è da considerarsi $8t_p$.

La micro operazione più lunga è il calcolo di IND nella microistruzione 0 che richiede 5tp.

Utilizzando la formula che fornisce la maggiorazione del tempo di ciclo τ otteniamo dunque

$\tau = T_{\omega PO} + \max\{T_{\omega PC}+T_{\sigma PO}, T_{\sigma PC}\} + \delta = 8t_p + 2t_p + 5t_p + t_p = 16t_p$

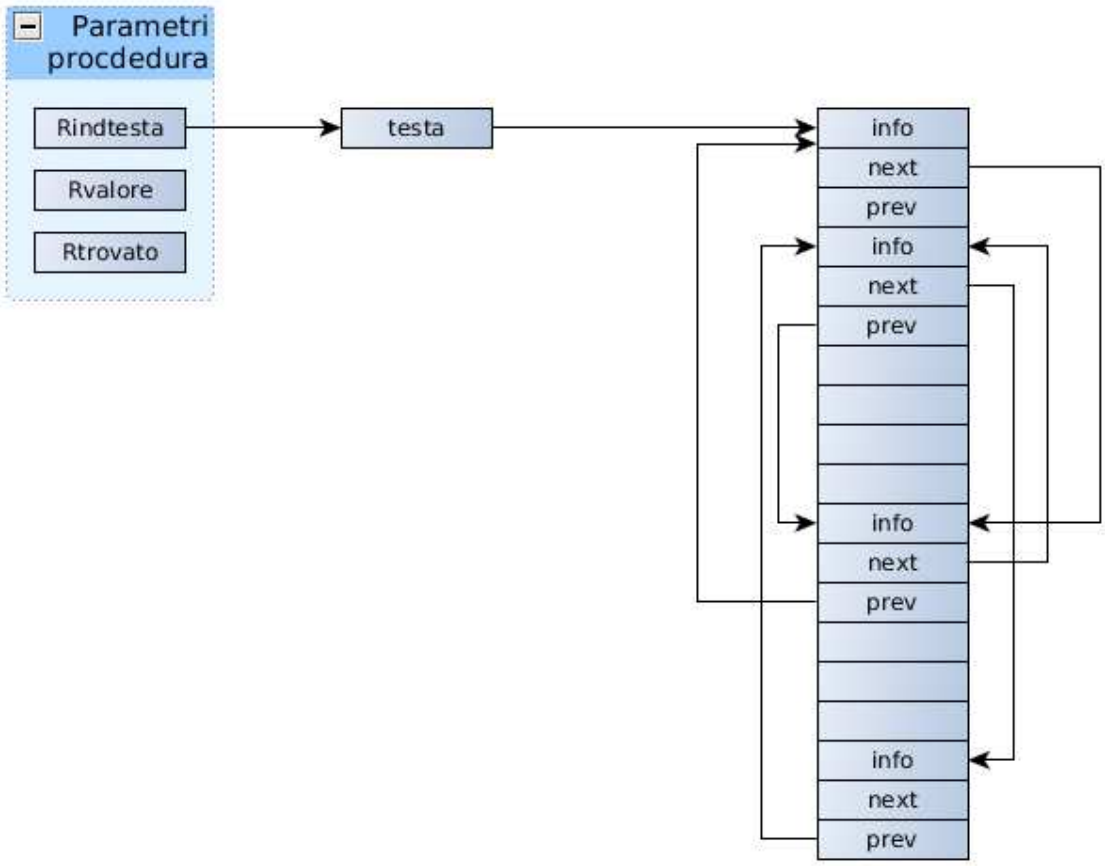
Dunque nel caso in cui la memoria C contenga i dati necessari alla traduzione, il tempo medio di servizio della operazione di traduzione sarà τ, mentre nel caso occorra accedere alla tabella TAB in memoria, occorreranno $2\tau+t_a$.

Se diversamente avessimo utilizzato una microistruzione per memorizzare uguali(C[(X.A)%64].TAG, (X.A)/64) in un registro da un bit e calcolare contemporaneamente INDTAB+(X.A)→IN, avremmo potuto rendere nullo il T_{wPO} con conseguente riduzione del ciclo di clock a τ = 11tp. Tuttavia in questo caso avremmo avuto necessità di due microistruzioni per completare la traduzione nel caso favorevole e di tre nel caso fosse necessario un accesso in memoria.

0. (RDY_{in}=0) nop,0
   (=1) uguali(C[(X.A)%64].TAG, (X.A)/64)→F, C[(X.A)%64].V→VTEMP, reset RDY_{in}, set ACK_{in}, 1
1. (ACK_{out},F,VTEMP=011) nop, 0
   (=111) set RDY_{out}, reset ACK_{out}, (C[(X.A)%64].W)..(X.B)→Y, 0
   (=-0-,-10) INDTAB+(X.A)→IND, "read"→OP,set RDY_{Mout}, 1
2. (RDY_{Min},ACK_{out}=0-,10) nop, 1
   (=11) reset RDY_{Min}, set RDY_{out}, reset ACK_{out}, DATAIN..(X.B)→Y,
   <X.A/64, 1, DATAIN> →C[(X.A)%64], 0

Domanda 2

Una possibile rappresentazione dello scenario è la seguente:



Lo pseudo codice della procedura sarà dunque:

```
p = testa;
while(p != 0 && p.info != valore)  p=p.next;
if(p==0) trovato= false
else {  p.prev.next = p.next; trovato = true; }
return trovato;
```

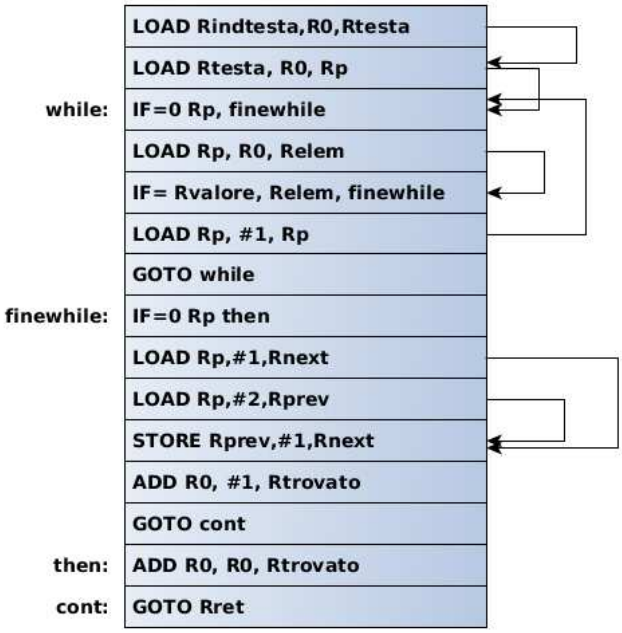
(NEW, OLD0)

La compilazione in codice D-RISC è la seguente:

	LOAD Rindtesta,R0,Rtesta	*recupero indirizzo puntatore testa*
	LOAD Rtesta, R0, Rp	*puntatore p al primo elemento*
while:	IF=0 Rp, finewhile	*se NULL termina il ciclo*
	LOAD Rp, R0, Relem	*recupera valore elemento corrente*
	IF= Rvalore, Relem, finewhile	*se trovato termina il ciclo*
	LOAD Rp, #1, Rp	*altrimenti p = p.next*
	GOTO while	*e ricomincia il ciclo*
finewhile:	IF=0 Rp then	*se p=NULL non l'ho trovato*
	LOAD Rp,#1,Rnext	*se l'ho trovato recupero @next*
	LOAD Rp,#2,Rprev	*recupero @prev*
	STORE Rprev,#1,Rnext	*p.prev.next = p.next*
	ADD R0, #1, Rtrovato	*trovato = true*
	GOTO cont	*fine else*
then:	ADD R0, R0, Rtrovato	*trovato = true*
cont:	GOTO Rret	*ritorno dalla procedura*

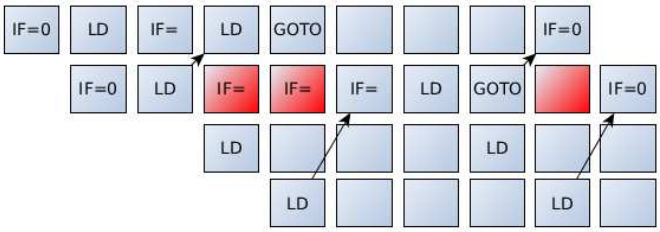
avendo assunto che il valore *true* e *false* siano rappresentati da 1 e 0 e che un puntatore NULL sia rappresentato mediante lo 0.

Le dipendenze logiche sono le seguenti:



L'efficienza della procedura è determinata dall'efficienza del ciclo while, percorso N volte (N lunghezza della lista) se non si trova l'elemento e mediamente N/2 volte se l'elemento risulta invece presente.

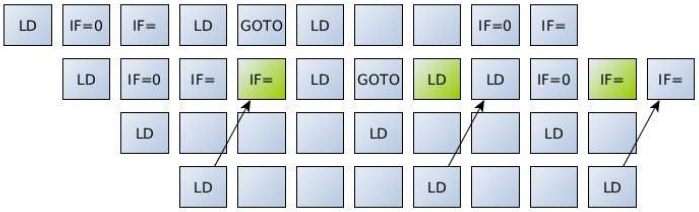
Analizzando il codice, osserviamo prima il comportamento a regime del primo *while*:



Notiamo che l'efficienza è pari a 5/8 per via della bolla introdotta dalla dipendenza sul secondo IF (k=1, d=2) e della bolla legata alla salto di fine ciclo.
Possiamo ottimizzare il ciclo riscrivendone il codice come segue:

while:	**LOAD Rp, R0, Relem**
	IF=0 Rp, finewhile
	IF= Rvalore, Relem, finewhile
	LOAD Rp, #1, Rp
	GOTO while, DELAYED=1
	LOAD Rp, R0, Relem

Carichiamo comunque il valore dell'elemento corrente prima di entrare nel ciclo. Successivamente facciamo i due controlli di fine ciclo e utilizziamo un salto ritardato per ciclare. Nello slot inseriamo nuovamente il caricamento del valore dell'elemento corrente. Con questo nuovo codice riusciamo a raggiungere un'efficienza pari a 5/7.



La parte rimanente del codice viene eseguita una volta sola. Con probabilità Ptrovato si segue il ramo ELSE dell'ultimo IF (7 istruzioni), con probabilità (1-Ptrovato) si segue il ramo THEN (3 istruzioni). In entrambi i casi abbiamo una bolla da 1 per il salto finale, nel ramo ELSE abbiamo una bolla da 2 per la STORE che dipende dalla LOAD.
Gli accessi alla lista comportano fault in cache primaria. L'organizzazione a lista non gode delle proprietà di località tipiche dei vettori e degli array. Possiamo dunque considerare un fault di cache primaria per ognuno degli accessi alla lista. σ=8 garantisce che il trattamento del fault carica tutte le parole relative all'elemento corrente della lista (tre parole per elemento).
Quindi il tempo di completamento della procedura può essere espresso come

Tinit + Twhile + Tfine

Tinit = 6t (due dipendenze k=1, d=2)
Twhile = 0.5 * N/2 * (7t+Tfault) + 0.5 * N * (7t + Tfault) = ¾ * N/2 * (7t+Tfault)
Tfine = ½ * 10t + ½ 4t

OLD-1

Nel caso di liste condivise occorre pensare che i puntatori siano identificatori unici che vanno risolti mediante accesso ad una opportuna tabella che per ogni identificatore contiene l'indirizzo logico relativo. Ogni parte del codice che prevede l'accesso ad un puntatore deve essere quindi realizzata

mediante l'accesso alla tabella. Per esempio, il codice del corpo del while, che contiene il codice relativo a **p=p.next** dovrebbe essere riscritto come segue: