Esercizio: memoria virtuale

Assumendo una memoria fisica di capienza 4 pagine, inizialmente vuota, una memoria virtuale di capienza 8 pagine (numerate da 0 a 7) e la seguente sequenza di riferimenti:

0(0),1(3),7(5),2(6),3(8),2(9),7(9),1(11),0(12),3(15)

dove la notazione X(Y) indica una richiesta dalla pagina X effettuata al tempo Y, mostrare l'evoluzione della lista mantenuta dall'algoritmo *second chance* (seconda possibilità) e confrontarne comportamento e costo realizzativo con quello dell'algoritmo LRU (least recently used).

Soluzione

In tabella (vedi lucido successivo) possiamo seguire il variare delle informazioni di controllo associate alle pagine presenti in memoria principale al succedersi delle richieste di pagina, dove T denota il tempo di caricamento della pagina, P l'indice di pagina ed R il bit di riferimento. Dalla tabella è facile rilevare come l'algoritmo dato degeneri in puro FIFO laddove tutte le pagine presenti siano state recentemente riferite. Gli istanti in cui ciò avviene sono marcati con un *.

In rapporto all'algoritmo LRU, l'algoritmo dato presenta un minor costo implementativo (in quanto quest'ultimo non riordina la lista delle pagine ad ogni riferimento, come invece previsto dall'LRU) ed una minore efficacia (in quanto esso può rimpiazzare la pagina più recentemente riferita, come marcato in tabella con un **•,** la quale ha, secondo il principio di località, elevata probabilità di essere riferita nell'immediato futuro, ciò che non accade con l'LRU).

Soluzione

sequenza richieste

	0(0)	1(3)	7(5)	2(6)*	3(8)	2(9)
T	0	0 ←3	0 ←3 ←5	0 ←3 ←5 ←6	8 ←8 ←8 ←8	8 ←8 ←8 ←8
P	0	0 ←1	0 ←1 ←7	0 ←1 ←7 ←2	1 ←7 ←2 ←3	1 ←7 ←2 ←3
R	1	1 ←1	1 ←1 ←1	1 ←1 ←1 ←1	0 ←0 ←0 ←1	0 ←0 ←1 ←1

	7(9)	1(11)*•	0(12)	3(15)
T	8 ←8 ←8 ←8	8 ←8 ←8 ←8	12 ←12 ←12 ←12	12 ←12 ←12 ←12
P	1 ←7 ←2 ←3	1 ←7 ←2 ←3	7 ←2 ←3 ←0	7 ←2 ←3 ←0
R	0 ←1 ←1 ←1	1 ←1 ←1 ←1	0 ←0 ←0 ←1	0 ←0 ←1 ←1

Esercizio: pipeline

Sia data una architettura di CPU con pipeline convenzionale a 5 fasi ed il corrispondente instruction set di tipo RISC:

load mem, reg ! trasferisce da memoria a registro
store reg, mem ! trasferisce da registro a memoria
mov reg1 reg2 ! trasferisce da reg2 a reg1
<**op**> reg1, reg2 ! esegue la funzione op(reg1,reg2) ponendo il risultato in reg1
jnz etichetta ! salta all'etichetta se l'operazione precedente non ha prodotto zero
jmp etichetta ! salta incondizionatamente all'etichetta

Si indichi per quali istruzioni di tale instruction set l'esecuzione comporti accesso in memoria, specificando in quale/i fase/i di pipeline esso abbia luogo.

Soluzione

L'instruction set mostrato nel testo del problema fa chiaramente riferimento ad una architettura Load/Store.

Un' architettura di tipo Load/Store accede alla memoria all'atto del prelievo dell'istruzione da eseguire (fase di 'fetch' della pipeline di CPU) ed ad ogni occorrenza di istruzione Load (fase di 'read') o Store (fase di 'write').

Esercizio: cache

Sia data la seguente sequenza di istruzioni assembler (ideali) e la corrispondente codifica esadecimale (ideale) con codice operativo ed operandi di ampiezza 1 byte:

```
      mov 4, R1          ! B0 04 ! inizializza R1 al valore 4 decimale
      mov 0, R2          ! B1 00 ! inizializza R2 al valore 0 decimale
L:    mov [R1], R3       ! BE 02  ! pone in R3 il contenuto di mem[R1]
      add R2, R3         ! A1 02  ! somma R3 ad R2 ponendo il risultato in R2
      dec R1                    ! 90        ! decrementa R1
      jnz L                     ! 75 F9     ! ritorna ad L se R1 non vale 0
      ret                        ! 16       ! ritorna al chiamante senza modificare il
contenuto
                                             ! dei registri R1,R2,R3
```

Si assuma la presenza di cache separate per istruzioni e per dati, ciascuna contenente 4 posizioni di ampiezza 1B, inizialmente vuote, entrambe con associazione diretta (N=1). Si assuma che la sequenza di programma di cui sopra sia memorizzata a partire dalla locazione **decimale** 12 di una memoria con il contenuto **esadecimale** mostrato in figura (vedi lucido successivo). Si mostri il contenuto di entrambe le cache al termine della suddetta esecuzione.

Esercizio: cache

Contenuto memoria

0	01	00	03	02
4	0A	07	04	71
8	37	24	11	00
12	B0	04	B1	00
16	BE	02	A1	02
20	90	75	F9	16

Soluzione

La CPU emette indirizzi relativi alla zona istruzioni durante la fase di fetch ed indirizzi relativi alla zona dati durante la fase di lettura (read) degli operandi non immediati, ossia quelli non presenti su registro o direttamente forniti dall'istruzione. Pertanto, la sequenza di indirizzi emessa dall'esecuzione del programma è la seguente:

istruzioni: 12, 13, 14, 15, $\{16, 17, 18, 19, 20, 21, 22\}_{\text{ripetuto 5 volte}}$, 23

indirizzi memoria dati: 4, 3, 2, 1

Ne segue che, al termine della sequenza, le rispettive cache avranno il seguente contenuto:

cache istruzioni

90	75	F9	16

cache dati

0A	00	03	02