Compito del Corso di Architettura degli Elaboratori 1

Anno Accademico 2004/2005

Appello del 22 Marzo 2005 - Soluzione esercizi pratici

Istruzioni

- Scrivere *Nome*, *Cognome* e *Matricola* su **ogni** foglio.
- Scrivere la risposta nello spazio bianco al di sotto della domanda; Non è possibile allegare fogli aggiuntivi, quindi cercate di essere chiari e non prolissi.
- In caso di errori indicate chiaramente quale parte della risposta deve essere considerata; annullate le parti non pertinenti.
- Assicurarsi che non manchi alcun foglio al momento della consegna.

Esercizi pratici

es9

Sia data la seguente sequenza di istruzioni assembler, dove i dati immediati sono espressi in esadecimale ed il registro `R0` contiene il valore 0:

```
LB   R3, 100(R0)   ! load byte da mem[100+[R0]]
ADD  R2, R0, R0    ! R2 = R0 + R0
LB   R1, 108(R2)   ! load byte da mem[108+[R2]]
ADDI R1, R1, 3     ! R1 = R1 + 3
ADDI R2, R2, 4     ! R2 = R2 + 4
SB   R1, 108(R2)   ! store byte in mem[108+[R2]]
SUB  R4, R3, R2    ! R4 = R3 - R2
BGTZ R4, -6        ! PC = PC - 6 se [R4] > 0
                   ! cioe' salta alla istruzione LB   R1, 108(R2)
```

Si assuma la presenza di due cache, una dati ed una istruzioni. La cache dati, in particolare, è di ampiezza 16B, con dimensione di blocco 4B, inizialmente vuota, ed associazione 2-way (con politica di rimpiazzo LRU e politica di scrittura write-through). Si assuma che la memeoria abbia il contenuto esadecimale mostrato di seguito (si esprimano gli indirizzi su 12 bit):

Indirizzo	byte	byte	byte	byte
100	0C	00	07	02
104	00	00	00	00
108	AE	13	A1	23
10C	A1	42	90	75
110	B9	16	00	00
114	0A	07	03	71

Si mostri come sia il contenuto della cache dati che il contenuto della memoria cambia a causa della esecuzione del codice assembler.

Soluzione

Poiché un blocco è costituito da 4B, e la cache è di 16B si avranno $16/4 = 4$ linee. Essendo l'associatività a due linee (2-way), ci saranno due insiemi (insieme 0 e insieme 1) ognuno di 2 linee. Quindi i 12 bit di indirizzo saranno suddivisi nel seguente modo: i 2 bit meno significativi individueranno il byte all'interno del blocco, il terzo bit da destra individuerà l'insieme (0 o 1), ed i restanti bit costituiranno il tag. Mostriamo di seguito l'evoluzione del contenuto dei registri, dei riferimenti a memoria, della cache dati (solo quando cambia) e della memoria (solo quando cambia). Per la cache dati, nel caso in cui tutte e due le linee di un insieme (set) siano libere, si sceglie la linea con indirizzo minore per la allocazione (scelta arbitraria: si poteva usare un criterio diverso).

codice eseguito	[R1] hex	[R2] hex	[R3] hex	[R4] hex	ind. rif. memoria hex / binario	cache dati: set 0 [linea 0] / [linea 1]	cache dati: set 1 [linea 2] / [linea 3]	modifica memoria mem[ind.] = cont.
						[linea 0] t: tag r: rif. [linea 1] t: tag r: rif.	[linea 2] t: tag r: rif. [linea 3] t: tag r: rif.	
LB R3, 100(R0)	?	?	C	?	100 000100000000	[0C|00|07|02] t:000100000 r:miss [ ] t: r:		
ADD R2, R0, R0	?	0	C	?				
LB R1, 108(R2)	AE	0	C	?	108 000100001000	[0C|00|07|02] t:000100000 r: [AE|13|A1|23] t:000100001 r:miss		
ADDI R1, R1, 3	B1	0	C	?				
ADDI R2, R2, 4	B1	4	C	?				
SB R1, 108(R2)	B1	4	C	?	10C 000100001100	[0C|00|07|02] t:000100000 r: [AE|13|A1|23] t:000100001 r:	[A1|42|90|75] t:000100001 r:miss [ ] t: r:	
						[0C|00|07|02] t:000100000 r: [AE|13|A1|23] t:000100001 r:	[B1|42|90|75] t:000100001 r:write-th. [ ] t: r:	mem[10C] = [R1] = B1
SUB R4, R3, R2	B1	4	C	8				
BGTZ R4, -6	B1	4	C	8				
LB R1, 108(R2)	B1	4	C	8	10C 000100001100	[0C|00|07|02] t:000100000 r: [AE|13|A1|23] t:000100001 r:	[B1|42|90|75] t:000100001 r:hit [ ] t: r:	
ADDI R1, R1, 3	B4	4	C	8				
ADDI R2, R2, 4	B4	8	C	8				
SB R1, 108(R2)	B4	8	C	8	110 000100010000	[B9|16|00|00] t:000100010 r:miss (LRU) [AE|13|A1|23] t:000100001	[B1|42|90|75] t:000100001 r: [ ] t:	

```
                                                  r:              r:

                                                  [B4|16|00|00]   [B1|42|90|75]     mem[110] = [R1]
                                                  t:000100010     t:000100001               = B4
                                                  r:write-th.     r:
                                                  [AE|13|A1|23]   [           ]
                                                  t:000100001     t:
                                                  r:              r:
SUB  R4, R3, R2       B4   8    C    4
BGTZ R4, -6           B4   8    C    4
LB   R1, 108(R2)      B4   8    C    4      110   [B4|16|00|00]   [B1|42|90|75]
                                          000100010000 t:000100010     t:000100001
                                                  r:hit           r:
                                                  [AE|13|A1|23]   [           ]
                                                  t:000100001     t:
                                                  r:              r:
ADDI R1, R1, 3        B7   8    C    4
ADDI R2, R2, 4        B7   C    C    4
SB   R1, 108(R2)      B7   C    C    4      114   [B4|16|00|00]   [B1|42|90|75]
                                          000100010100 t:000100010     t:000100001
                                                  r:              r:
                                                  [AE|13|A1|23]   [0A|07|03|71]
                                                  t:000100001     t:000100010
                                                  r:              r:miss

                                                  [B4|16|00|00]   [B1|42|90|75]
                                                  t:000100010     t:000100001
                                                  r:              r:
                                                  [AE|13|A1|23]   [B7|07|03|71]     mem[114] = [R1]
                                                  t:000100001     t:000100001               = B7
                                                  r:              r:write-th.
SUB  R4, R3, R2       B7   C    C    0
BGTZ R4, -6           B7   C    C    0
```

es10

Si consideri il codice assembler dell'esercizio precedente (es9) e la pipeline MIPS a 5 stadi vista a lezione, senza possibilità di data-forwarding, ma con possibilità di scrittura e successiva lettura dei registri in uno stesso ciclo di clock. Si assuma che ogni operazione di memoria impieghi un solo ciclo di clock e che i salti condizionali siano predetti come "taken" (presi). Si mostri il diagramma degli stadi della pipeline per l'esecuzione del codice fino alla prima occorrenza (inclusa) della istruzione `BGTZ R4, -6`.

Dire inoltre quanti cicli occorrono per completare l'esecuzione del codice.

Soluzione

Non essendoci possibilità di data-forwarding, tutte le dipendenze da dati che non si riducono a scrittura e successiva lettura di un registro provocano stallo, come descritto di seguito

	CICLI CLOCK													
istruzione	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LB R3, 100(R0)	IF	ID	EX	ME	WB									
ADD R2, R0, R0		IF	ID	EX	ME	WB								
LB R1, 108(R2)			IF	ID	ID	ID	EX	ME	WB					
ADDI R1, R1, 3				IF	IF	IF	ID	ID	ID	EX	ME	WB		
ADDI R2, R2, 4							IF	IF	IF	ID	EX	ME	WB	
SB R1, 108(R2)										IF	ID	ID	ID	EX
SUB R4, R3, R2											IF	IF	IF	ID
BGTZ R4, -6														IF

	CICLI CLOCK				
istruzione	15	16	17	18	19
LB R3, 100(R0)					
ADD R2, R0, R0					
LB R1, 108(R2)					
ADDI R1, R1, 3					
ADDI R2, R2, 4					
SB R1, 108(R2)	ME	WB			
SUB R4, R3, R2	EX	ME	WB		
BGTZ R4, -6	ID	ID	ID	EX	ME

Il fetch della istruzione dopo il primo salto (predetto correttamente come preso) avviene effettivamente al ciclo 17 (anche se inizia al ciclo 15), mentre per il secondo salto avviene al ciclo 29 (perché non c'è stallo alla seconda esecuzione di `LB R1, 108(R2)`) e poi al ciclo 43 si completa l'ultima istruzione di salto (non preso) con la fase `MEM`.