Lösningar till tentamen i kursen EDA330

Datorsystemteknik D

15/1 1999

Följande är skisser till lösningar av uppgifterna. Full poäng på en uppgift kräver i de flesta fall en något fylligare motivering. I en del fall är alternativa lösningar möjliga.

1.

a. $9{,}375 = 8+1+1/4+1/8 = 1001{,}011_2 = (-1)^0 * 1{,}001011_2 * 2^{130-127}$
Sign = 0
Significand field = $00101100000000000000000_2$
Exponent field = $130 = 10000010_2$
Den binära representationen är $01000001000101100000000000000000_2$

För att ladda en så stor konstant i MIPS måste först de sexton mest signifikanta bitarna laddas med en lui-instruktion, och därefter kan man lägga till de sexton minst signifikanta med addi eller ori. I detta fall är dock de sexton minst signifikanta bitarna noll, så de behöver inte laddas explicit eftersom lui ändå sätter dessa bitar till noll.

Koden som behövs består alltså av en enda instruktion:
`lui $4, 0100000100010110`$_2$

Denna instruktion har op-kod 15, med noll i rs-fältet, 4 i rt-fältet, och konstanten i återstående bitar. Alltså 001111 (op) 00000 (rs) 00100 (rt) 0100000100010110 (konstantens sexton mest signifikanta bitar), eller ihopskrivet:
00111100000001000100000100010110

b. **Se kursboken (figurerna 4.17-4.18)**.

2.

a. Tabellen nedan ger innehållet i varje cacheblock efter varje referens genom att ange tag för de block som ligger i cacheminnet. De fyra första kolumnerna ger byte-adressen, motsvarande block-adress (i detta fall byte-adress/4), motsvarande set-nummer (i detta fall block-adress modulo 4), och tag (i detta fall block-adress/4). Tag för senast refererade cache-block är markerad med fetstil.

Set/block					0				1				2				3			
Byte	Block	Tag	Set	Hit	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
12	3	0	3	n													**0**			
30	7	1	3	n													0	**1**		
13	3	0	3	y													**0**	1		
76	19	4	3	n													0	1	**4**	
5	1	0	1	n					**0**								0	1	4	
14	3	0	3	y					0								**0**	1	4	
44	11	2	3	n					0								0	1	4	**2**
116	29	7	1	n					0	**7**							0	1	4	2
15	3	0	3	y					0	7							**0**	1	4	2
61	15	3	3	n					0	7							0	**3**	4	2
101	25	6	1	n					0	7	**6**						0	3	4	2
76	19	4	3	y					0	7	6						0	3	**4**	2
6	1	0	1	y					**0**	7	6						0	3	4	2
40	10	2	2	n					0	7	6		**2**				0	3	4	2
31	7	1	3	n					0	7	6		2				0	3	4	**1**

Hit rate = 5/15 = 33%

b. Beteckna blockstorleken B räknad i byte, och datakapaciteten räknad i byte N. Antal bitar för blockoffset = b = $\log_2$ B. Antal block i cacheminnet = N/B. Antal set i cacheminnet = N/2B. Antal bitar för setindex = s = $\log_2$ N/2B = $\log_2$ N - b-1. Antal bitar för tag = t = 30 - s - b = 30 - $\log_2$ N + b + 1 - b = 31 - $\log_2$ N.

Antal bitar att lagra per block= 8B+ 31 - $\log_2$ N + 1 = 8B + 32 - $\log_2$ N.
Antal byte att lagra per block = B + (32 - $\log_2$ N)/8 = B + 4 - $\log_2$ N/8
Totalt antal byte i cachminnet = 80 K = N/B * (B + 4 - $\log_2$ N/8).

Tillåtna värden på B är givna i uppgiften, och vi ser att $\log_2$ N måste vara

jämnt delbart med 8, dvs N är 2^{16}, 2^8 eller 2^0 (större N är inte möjliga eftersom N inte kan överskrida totala cache-storleken). Det är nu lätt att se att enda möjliga lösningen är N=2^{16} och B=8. Alltså är **datakapaciteten 64 KB och blockstorleken 8 B**.

3.

a. **Se kursboken.** Register addressing, base addressing, immediate addressing, PC-relative addressing.

b.

```
slt  $1, $15, $0
bne  $1, $0, DEST

beq  $15, $0, DEST

slt  $1, $0, $15
bne  $1, $0, DEST

slt $1, $15, $0
beq $1, $0, DEST
```

c. **Se kursboken.** Hoppadressen beräknas genom att addera en konstant operand till det aktuella PC-värdet (=adress till hoppinstruktionen + 4). Operanden ryms i 16 bitar, men multipliceras med 4 innan den adderas till programräknarvärdet, och hoppavstånden sträcker sig alltså mellan -2^{18} och +2^{18}-4. Längsta avståndet från hoppinstruktionen blir alltså 2^{18} byte. Längre villkorliga hopp kan konstrueras enligt följande exempel:

```
L1:       beq  $1, $0, L2
```

kan om abs(L2-L1)>2^{18} ersättas med:

```
L1:       bne  $1, $0, L3
          j    L2
L3:
```

4. Vi börjar med att titta på de data- och hoppkonflikter som kan uppstå. Mellan lw och beq krävs en extra cykel eftersom datakonflikten däremellan inte kan lösas med forwarding. Dessutom krävs en extra cykel efter varje taget hopp (beq, bne, j, jr) på grund av assume-not-taken-strategin.

   Programmet består i huvudsak av en snurra som föregås av en instruktion (som initierar snurräknaren $4 till 64) och efterföljs av en instruktion som troligen är ett subrutinåterhopp. Snurran inleds med lw-instruktionen och avslutas med bne-instruktionen. Innehållet i snurran består utöver lw och snurrhoppet bne av två vägar som väljs av beq. Om beq inte tas följs väg 1: addi, sw, j som tar 4 cykler inklusive extra cykel efter j-instruktion (taget hopp). Annars följs väg 2: add, addi, addi som tar 4 cykler inklusive den extra cykeln efter beq som blir nödvändig i detta fall.

   Snurran genomlöps 16 gånger (64/4), och i varv 1, 5, 9, och 13 läser lw första ordet från ett block som vi inte vet om det redan finns i D-cache eller ej. Vi måste då räkna med att det kan blir cache-miss och lägga till maximal miss penalty (beräknas längre fram). Övriga varv vet vi att det refererade ordet ligger i ett block som hämtats i ett tidigare varv (eller ännu tidigare) och alltså säkert finns i D-cache. Det kan alltså aldrig bli miss i dessa fall eftersom det inte finns något som kan ha orsakat att blocket kastats ut sedan det hämtades in. Totalt tar alltså ett varv i snurran maximalt 1 (lw) + max miss penalty + 1 (datakonflikt efter lw) + 1 (beq) + 4 (längsta väg efter beq enligt ovan) + 1 (bne) + 1 (extra cykel efter tagen bne) = 9 + max miss penalty cykler utom för sista varvet som tar en cykel mindre eftersom bne då inte tas. Totalt tar alltså koden maximalt 1 (addi) + 16*9 - 1 + 4*max miss penalty + 1 (jr) + 1 (extra cykel efter jr) = 146 + 4*max miss penalty.

   Så långt är inte missar i I-cache medräknade. Totalt består koden av 11 instruktioner vilket tar upp maximalt 4 block (ett block = 4 ord = 4 instruktioner) beroende på var koden ligger i minnet (första instruktionen behöver ju inte ligga på en blockgräns). Vi måste också enligt uppgiften räkna med eventuell fördröjning i inhämtningen av instruktionen efter den sista i vår kod (jr), vilket innebär att vi rör oss med maximalt 5 block som ska in till I-cache. De fyra första av dessa ligger konsekutivt i minnet, och såvida inte I-cache är mycket litet så kan de alla ligga i cache samtidigt utan att kasta ut varandra. Vi kan alltså anta att maximalt 5 missar uppstår i I-cache under kodexekveringen. Det ger ett extra tillskott av 5*max miss penalty cykler till WCET-uppskattningen. Observera att några av dessa extra cykler eventuellt överlappar med andra extracykler på grund av konflikter som vi infört ovan, t.ex. mellan lw och beq. Vi kan dock inte räkna med detta eftersom vi behöver en säker gräns för WCET. Totala uppskattningen av WCET blir alltså 146 + 9*max miss penalty cykler.

   Det återstår då bara att beräkna max miss penalty. Räknat i busscykler tar en blockläsning till cache (=miss penalty) enligt uppgiften som mest 12 (maximal arbitrering och väntetid) +1 (skicka adress) +6 (vänta på första ordet) +4 (överför 4 ord överlappat med åtkomst av resterande tre ord) = 23 busscykler. Detta motsvarar 46 processorcykler eftersom bussen går på halva processorns frekvens.

   Vi får alltså slutligen att en säker övre gräns för WCET för den exekverade koden är 146 + 9*46 = **560 cykler** tills instruktionen efter jr hämtas in. En undre gräns för exekveringstiden är enligt beräkningen 146 cykler = 26% av övre gränsen.

5.

Deluppgift	1	X	2
a			2
b			2
c		X	
d	1		
e	1		
f		X	
g	1		
h		X	
i			2
j			2
k	1		
l		X	