Lösningar till tentamen i kursen EDA330

Datorsystemteknik D

21/8 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. $2{,}375 = 2+1/4+1/8 = 10{,}011_2 = (-1)^0 * 1{,}0011_2 * 2^{128-127}$
Sign = 0
Significand field = $00110000000000000000000_2$
Exponent field = 128 = 10000000_2
Den binära representationen är $01000000000110000000000000000000_2$

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 \$7, 0100000000011000_2

Denna instruktion har op-kod 15, med noll i rs-fältet, 7i rt-fältet, och konstanten i återstående bitar. Alltså 001111 (op) 00000 (rs) 00111 (rt) 0100000000011000 (konstantens sexton mest signifikanta bitar), eller ihopskrivet:
00111100000001110100000000011000

b. **Se kursboken.**
1. Dividera talet som har minst exponent med 2 (=högerskift, glöm ej inledande ettan) och inkrementera exponenten tills båda talen har samma exponent.
2. Addera signifikanderna.
3. Normalisera resultatet genom upprepade höger- eller vänsterskift av signifikanden kombinerat med motsvarande ökning/minskning av exponenten. Kontrollera om overflow/underflow uppstår.
4. Avrunda signifikanden till rätt antal bitar. Upprepa från steg 3 om resultatet efter avrundning inte är normaliserat.

2.

a. **Programmet tar ett index till en tabell (XTAB) där en adress slås upp. På den adressen läses och uppdateras ett värde. Om värdet är noll så sätts det till 7, annars minskas det med 1.**

b. **Indata tas från register $a0. I register $v0 finns efter funktionen det uppdaterade räknarvärdet, och register $v1 den uppdaterade adressen**.

c. **5 extra cykler om det inlästa värdet är skilt från noll, och 5 extra cykler om det inlästa värdet är noll**.

d. I den givna koden behöver endast en återhoppsinstruktion, jr $ra, läggas till allra sist (efter etiketten L2). Dessutom måste en etikett (label), t.ex. READ sättas på första instruktionen. Inga register behöver sparas undan eftersom endast $a0, $v0, och $v1 används, och inga ytterligare subrutinanrop görs från subrutinen. Den modifierade koden blir alltså:

```
READ:
        sll     $a0, $a0, 2
        lw      $v1, XTAB($a0)
        lw      $v0, 0($v1)
        beq     $v0, $zero, L1
        addi    $v0, $v0, -1
        beq     $zero, $zero, L2
L1:     addi    $v0, $zero, 7
L2:     sw      $v0, 0($v1)
        jr      $ra
```

Anrop av subrutinen kan då enkelt ske genom:

```
        addi    $a0, $zero, index
        jal     READ
```

Denna lösning förutsätter att adressen som motsvarar READ kan representeras i en jump-instruktion.

3.

a. **Risken att data kastas ut minskar eftersom mer data får plats**.

b. **Rumslokaliteten utnyttjas bättre**.

c. **Risken att data kastas ut på grund av konflikt om samma cacheplats minskar**.

d. **Det kan bero på att en lösning som minskar miss rate istället kan förlänga cacheåtkomsttiden, och därmed eventuellt även processorns klockcykeltid**.

e. Undersök konfigurationerna i ordning efter sjunkande miss rate. Tag reda på hur många bitar som måste lagras per set. Det bestämmer hur många RAM-komponenter som måste användas parallellt. T.ex:

4 KB cache med associtivitet 2 och 2 ord/block har
512 block fördelade på 256 set
blockoffset kräver 3 bitar (8B/block)
index kräver 8 bitar (för att adressera 256 set)
därför kräver tag 28-8-3 = 17 bitar
valid, dirty (write-back), och replacement ($\log_2(2)=1$) kräver 3 bitar
data för ett block kräver 2*4*8 = 64 bitar
ett block kräver alltså 64 + 17 + 3 = 84 bitar
varje set kräver därmed 2*84 = 168 bitar
vilket kräver 168/16 = 10,5, dvs 11 RAM-komponenter parallellt
det behövs en adress per set, så alla set ryms i 11 parallella RAM
denna konfiguration är alltså inte möjlig

Med detta resonemang fås att lägsta miss rate som kan åstadkommas är **19,2%**, vilket fås med konfigurationen 2 KB datalagringskapacitet,associativitet 1, 2 ord/block. Denna konfiguration kräver 6 RAM-komponenter.

4.

a. $T = T_{CPU} + T_{I/O}$
Väntetiden för sensorvärden är $T_{I/O}$ och blir med testsensorn 100*1 ms = 0,1 s och med de riktiga sensorn 100*10 ms = 1 s. CPU-tiden skiljer inte mellan de två fallen för standard-RAM, och är alltså T_{CPU} = 3,5 s - 0,1 s = 3,4 s. Då fås att den eftersökta körningstiden blir 3,4 s + 1 s = **4,4 s**.

b. $T_{CPU} = I * CPI * T_C$
$T_C = 1/(100\ MHz) = 10\ ns$
$CPI = CPI_0 + P_{mem} * P_{miss} * T_{penalty}$
$I = 2*10^8$
$P_{mem} = 4*10^7 / 2*10^8 = 0,2$
$T_{penalty1} = 200\ ns / 10\ ns = 20$ cykler
$T_{penalty2} = 100\ ns / 10\ ns = 10$ cykler
$3,5\ s = 2*10^8 * (CPI_0 + 0,2 * P_{miss} * 20) * 10\ ns + 0,1\ s$
$3,1\ s = 2*10^8 * (CPI_0 + 0,2 * P_{miss} * 10) * 10\ ns + 0,1\ s$
$CPI_0 = 1,3$
$P_{miss} = 0,1$
Vi halverar nu missannolikheten till 0,05 och får körningstiden:
$2*10^8 * (1,3 + 0,2 * 0,05 * 20) * 10\ ns + 1\ s$ = **4 s**

c. Körningstid med nya RAM-typen:
$2*10^8 * (1,3 + 0,2 * 0,05 * 10) * 10\ ns + 1\ s = 3,8\ s$
Körningstid med ändrad klockfrekvens (f):
$2*10^8 * (1,3 + 0,2 * 0,05 * 200*10^{-9}*f) * 1/f + 1\ s =$
$2,6*10^8/f + 1,4$
Om körningstiden med ändrad frekvens inte ska vara längre än med den nya RAM-typen så fås villkoret f 108,3 MHz.
Ja, det är möjligt att uppnå samma prestandaförbättring med en justering av klockfrekvensen.

5.

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