Lösningar till tentamen i kursen EDA330 för D och EDA370 för E

Datorsystemteknik

25/5 2002

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

1.

a. $-769*2^{-41} = -(512+256+1)*2^{-41} = -1100000001_2*2^{-41} = -1{,}100000001_2*2^{-32}$
sign = 1 (negativt)
exponent=-32 => exponent field = exponent + 127 = 95 = 01011111_2
significand field = $10000001000000000000000_2$
Den binära representationen är $10101111110000000100000000000000_2$

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 bitarna med addi eller ori.

Koden som behövs består alltså av två instruktioner:
`lui $4, 1010111111000000`$_2$
`addi $4, $4, 0100000000000000`$_2$

b. Tvåkomplementsform och talet är negativt. Ta fram det positiva talet genom att ta tvåkomplementet:
Invertera: $01010000001111111101111111111111_2$
Addera 1: $01010000001111111100000000000000_2$
Talet kan skrivas som $(2^{16} + 2^{14} + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0)*2^{14} =$
$(65536 + 16384 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1)*2^{14} = 82175*2^{14}$
Det sökta talet är alltså $\mathbf{-82175*2^{14}}$

c. 101011 11110 00000 0100000000000000 = op = 101011 = 43 = sw (format I), rs = 11110 = 30 ($fp), rt = 00000 ($zero), imm = 0100000000000000 = 16384.
Instruktionen är **sw $zero, 16384($fp)**

d.

```
          1 0 0 1
        X 0 1 1 1 0
        -----------
          0 0 0 0      00 = gör inget
        - 1 0 0 1      10 = subtrahera
        0 0 0 0        11 = gör inget
      0 0 0 0          11 = gör inget
+ 1 0 0 1              01 = addera
-----------------
  0 1 1 1 1 1 1 0
```

2.

a. 1 block = 64 ord = 2^6 ord = 128 byte = **1024 bitar**. Av adressen krävs alltså 6 bitar i block-offset. Cacheminnet kan lagra 4 K ord = 64 block. 4-vägs associativitet innebär 4 ord/set, dvs. totalt 16 set i cacheminnet. Av adressen krävs därför 4 indexbitar för att hitta i vilket set ett block kan ligga. Det finns maximalt 1 M ord primärminne så de fysiska adresserna som används av cachen måste vara 20 bitar breda (2^{20} = 1 M ord). För varje block i cacheminnet måste därför 20 - 6 - 4 = **10 bitar tag** lagras. Dessutom krävs **2 bitar för LRU** och **1 valid bit** per block. Det krävs alltså 1024 + 10 + 2 + 1 = 1037 bitar per block. Med 64 block blir totala antalet bitar cachen måste lagra 64*1037 = **66368 bitar**.

b. Från access av adress 0 till adress 4095 fylls cachen med 64 ord åt gången vilket innebär 64 missar. Från access av adress 4096 till 4351 sker ytterligare 4 missar. Eftersom LRU används är det blocken med innehållet i de lägsta adresserna som ersätts. När man sedan börjar accessa adress 0 och framåt igen sker därför först 4 missar och de block som ersätts är nu de med innehållet i de numera lägsta (tidigare näst lägsta) adresserna. Det betyder att det fås ytterligare 4 missar sammanlagt fem gånger per varv. Det hela upprepas ytterligare 8 gånger. Alltså fås totalt (64 + 4 + 5*4*9) = 248 missar. Resterande 4352*10-248 = 43272 accesser är träffar.

   Vid miss tar det 6 busscykler för de 2 första orden (1 cykel för adress och 5 cykler för de första 2 orden) + 3 busscykler för efterkommande 6 ord. Detta upprepas 8 gånger (= 64 ord) => (6+3)*8 busscykler = 72 busscykler => 720 ns vid 100 MHz klockfrekvens.

   Vid träff accessas cachen med klockfrekvensen 400 MHz => 2,5 ns per access.

   Den totala accesstiden vid användning av cache blir således 248*720 + 43272*2,5 ns = 286740 ns.

   Den totala accesstiden då cachen inte används blir 4352/2 accesser per varv * 10 varv * 6 busscykler per access *10 ns/busscykel = 1305600 ns

   Dvs. 1305600/286740 = **4,55 ggr snabbare med cachen**.

c. Direct mapped cache innebär att inga LRU bitar behövs. Fortfarande krävs 6 bitars block-offset (1 block = 64 ord = 2^6 ord = 128 byte = **1024 bitar**). Cacheminnet kan lagra 4 K ord = 64 block. = 2^6 block. Av adressen krävs därför 6 indexbitar. Med 20 bitars adresser måste därför 20 – 6 – 6 = **8 bitar tag** lagras. Dessutom krävs fortfarande **1 valid bit** per block. Det krävs alltså 1024 + 8 + 1 = 1033 bitar per block. Med 64 block blir totala antalet bitar cachen måste lagra 64*1033 = **66112 bitar**.

   Från access av adress 0 till adress 4095 fylls cachen med 64 ord åt gången vilket innebär 64 missar. Från access av adress 4096 till 4351 sker ytterligare 4 missar. Det blir blocken med innehållet i de lägsta adresserna som ersätts. När man sedan börjar accessa adress 0 och framåt igen sker därför först 4 missar fram till access av adress 4096 då ytterligare 4 missar fås. Det hela upprepas ytterligare 8 gånger. Alltså fås: (64 + 4 + (4+4)*9) = 140 missar och således 4352*10-140 = 43380 träffar.
   Samma accesstid vid träff och miss som i uppg. b gör att den totala accesstiden vid användning av cache blir 140*720 + 43380*2,5 ns = 209250 ns.
   Den totala accesstiden enligt är enligt uppg. b 1305600 ns. Dvs. 1305600/209250 = **6,24 ggr snabbare med cachen**.

3. 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 två instruktioner (som initierar snurräknaren $4 till 64, och en annan räknare till 0) 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: sll, sw, j som tar 4 cykler inklusive extra cykel efter j-instruktion (taget hopp). Annars följs väg 2: addi som tar 2 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 (addi) + 1 (bne) + 1 (extra cykel efter tagen bne) = 10 + max miss penalty cykler utom för sista varvet som tar en cykel mindre eftersom bne då inte tas. Totalt tar alltså koden maximalt 2 (addi, addi) + 16*10 - 1 + 4*max miss penalty + 1 (jr) + 1 (extra cykel efter jr) = 163 + 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 163 + 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 2 (maximal arbitrering och väntetid) +1 (skicka adress) +2 (vänta på första ordet) +4 (överför 4 ord överlappat med åtkomst av resterande tre ord) = 9 busscykler. Detta motsvarar 18 processorcykler eftersom bussen går på halva processorns frekvens.
Vi får alltså slutligen att en säker övre gräns för WCET för den exekverade koden är 163 + 9*18 = 325 cykler tills instruktionen efter jr hämtas in. En undre gräns för exekveringstiden är enligt beräkningen 163 cykler = 50% av övre gränsen.

4.

a. Ett skivminne rymmer 8 ytor * 30000 spår/yta * 500 sektorer/spår * 512 bytes/sektor = $6{,}144*10^{10}$ bytes.
4*64 bitar data överförs på 5 busscykler (= $5/(200*10^6)$ s = 25 ns) => $4*8/(25*10^{-9})$ bytes/s = $1{,}28*10^9$ bytes/s. Bussen är ledig för I/O 40% av tiden => $0{,}4*1{,}28*10^9$ bytes/s = $5{,}12*10^8$ bytes/s tillgänglig bandbredd för I/O.
Peak bandwidth för ett skivminne är (7200 varv/minut) / (60 s/minut) * 512 bytes/sektor * 500 sektorer/varv = $3{,}072*10^7$ bytes/s. Antalet skivminnen som kan accessas samtidigt = $5{,}12*10^8$ / $3{,}072*10^7$ st = 16 st.
Den totala skivminneskapaciteten = 16 * $6{,}144*10^{10}$ bytes = **$9{,}83*10^{11}$ bytes**.

b. Total accesstid = seek time + rotational delay + tid för läsning/skrivning = 10 ms + 0,5 varv / ((7200 varv/minut) / (60 s/minut)) + 8*1024 bytes / ($3{,}072*10^7$ bytes/s) = 10 ms + 4,17 ms + 0,27 ms = 14,44 ms varav söktid = seek time + rotational delay = 14,17 ms => 14,17/14,44 = **98 % ägnas åt sökning**.

c. Total accesstid = (10 ms + 0,5 varv / ((7200 varv/minut) / (60 s/minut)) + 8*1024 bytes / ($3{,}072*10^7$ bytes/s)) * 0,01 + (8*1024 bytes / ($3{,}072*10^7$ bytes/s)) * 0,99 = 14,44*0,01 + 0,267 * 0,99 ms = 0,4087 ms varav söktid = (10 ms + 0,5 varv / ((7200 varv/minut) / (60 s/minut)) * 0,01 = 14,17 * 0,01 ms = 0,1417 ms => 0,1417 / 0,4087 = **35 % ägnas åt sökning**.

d. **Se kursboken.**

5.

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