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

Datorsystemteknik

19/5 2001

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. Figuren visar ett exempel på datakonflikt hos register $t5 mellan instruktionerna `add $t5, $t1, $t2` och `lw $t4, 100($t5)`.

b. Konflikten kan lösas antingen med data forwarding (**se avsnitt 6.5 i kursboken**), pipeline stall eller med hjälp av kompilatorn (kasta om instruktionerna eller stoppa in nop instruktioner).

c. 4 instruktioner för att fylla pipeline + 2 instruktioner innan loop L1 + (5 + 2 + (10 för subrutin L3) + (1 + 2 + (10 för subrutin L3) + 2)*40 för inre loop L2 + 2)*10 för loop L1 + 3 = 6199 instruktioner (=I). Med formeln för CPU tid = I * CPI * T_C där antal klockcykler per instruktion CPI = 1 och en klockcykeltid T_C = 2 ns (500 MHz klockfrekvens) fås CPU-tid = **12398 ns**.

d. Ersätt nop i delay branch luckorna med andra instruktioner där så är möjligt. Ersätt additioner med multiplikation med 4 (skifta 2 steg). Sätt $a2 till 40 i början av loop L1 ($t1 alltid 10). Flytta initiering av $t0 utanför loop L1. T ex:

```
        addi   $a1,   $zero, 10
        addi   $t0,   $zero, 1
L1:     addi   $a2,   $zero,  40
        jal    L3
        sub    $a1,   $a1,   $t0
L2:     jal    L3
        sub    $a2,   $a2,   $t0
        bne    $a2,   $zero, L2
        sw     $v0,   0($a3)
        bne    $a1,   $zero, L1
        nop
        j      L4
        sw     $v0,   4($a3)
L3:     sw     $t1,   0($sp)
        lw     $t1,   0($a1)
        add    $v0,   $v0,   $t1
        lw     $t1,   0($a2)
        add    $v0,   $v0,   $t1
        sll    $v0,   2
        jr     $ra
        lw     $t1,   0($sp)
L4:
```

2.

a. 1 block = 32 byte = 256 bitar. Av adressen krävs alltså 5 bitar i block-offset. Cacheminnet kan lagra 64 KB/32B = 2048 block. Fyrvägs associativitet innebär 4 block/set, och alltså totalt 512 set i cacheminnet. Av adressen krävs därför 9 (2^9 = 512) indexbitar för att hitta i vilket set ett block kan ligga. Det finns maximalt 256 MB primärminne så de fysiska adresserna som används av cachen måste vara 28 bitar breda (2^{28} B=256 MB). För varje block i cacheminnet måste därför 28-9-5 = 14 bitar tag lagras. För varje block krävs

dessutom en valid bit och en dirty bit (pga write-back-strategin) samt 2-bitars tidsstämpel för LRU. Totalt krävs alltså 256+14+1+1+2 = 274 bitar/block. Med 2048 block blir totala cache-storleken 2048*274 = **561 152 bitar**.

b. Sidoffset kräver 28 bitar (2^{28} = 256 MB). Varje sida pekar ut sidtabell med 4 KB blockstorlek (2^{12} B = 4 KB). Virtuella sidnummer kräver därmed 28-12 = 16 bitar, vilket motsvarar 2^{16} = 64 K sidor. Fysiska sidnummer kräver 28-12 = 16 bitar. I sidtabellen ska för varje virtuell sida kunna lagras fysiskt sidnummer (16 bitar), valid bit, dirty bit, protection bit, och två bitar för utbytesalgoritmen, dvs totalt 21 bitar. Eftersom varje sidtabell-entry måste vara ett helt antal bytes krävs 24 bitar (=3 B). Varje sidtabell kräver därmed 64 K sidor * 3 B/sida = **192 KB**.

c. TLB kan lagra information om 128 sidöversättningar. Den information som behöver lagras är fysiskt sidnummer (16 bitar), protection bit, och dirty bit (talar om ifall sidan uppdaterats), valid bit, tag och en bits tidsstämpel för LRU. Tvåvägs associativitet innebär 2 sidöversättningar/set, och alltså 64 set. Det krävs därför 6 (2^6 = 64) indexbitar för att hitta vilket set en sidöversättning kan ligga. De virtuella adresserna är 52 bitar breda och med 4 KB blockstorlek och 6 bitars index krävs alltså 52-12-6 = 34 bitars tag. Alltså måste 16 + 1 + 1 + 1 + 34 + 1 = 54 bitar lagras för varje plats i TLB, och totala antalet bitar i TLB blir 128*54 = **6912 bitar**.

d. För att indexera cacheminnet krävs de 5+9 = 14 minst signifikanta bitarna av adressen. Den del av de virtuella adresserna som inte behöver översättas, sid-offset, är endast 12 bitar. Översättningen av sidnummer måste därför göras innan uppslagningen i cacheminnet kan ske. Översättningen med hjälp av TLB tar 3 ns och cache-uppslagningen tar därefter 5 ns. Vid träff i TLB och cache tar därför en minnesåtkomst 8 ns. Då denna tid enligt uppgiften definierar en processorklockcykel, kan processorn som bäst köra med **125 MHz** klockfrekvens.

3. a. Prestanda/pris = 1/(exekveringstid*pris). Exekveringstiden är endast beroende av CPI talet eftersom antal instruktioner och klockfrekvensen är konstant => prestanda/pris är proportionelig mot 1/(CPI*pris). Dvs. för att maximera prestanda/pris, så skall vi minimera CPI*pris.

CPI=CPIperfect+MemStall.
Först behöver vi ta reda på CPIperfect för den gamla processorn.
CPI=800MHz*20sek/10e9=1.6 clocks per instr.
För att räkna ut MemStall för den gamla processorn behöver vi veta hur stor andel av alla accesser som missar i de olika cachenivåerna.
Missar i L1 per instr.=0.01+0.05*0.25
Missar i L2 per instr.=(0.01+0.05*0.25)*(1-0.9)
MemStall=(0.01+0.05*0.25)*30e-9*800e6+(0.01+0.05*0.25)*(1-0.9)*
*125e-9*800e6=0.7650 clocks per instr
CPIperfect=1.6-0.7650=0.8350 clocks per instr

För att få CPI för de nya processorkonfigurationerna behöver vi räkna ut MemStall för varje ny cachekonfiguration.

MemStall_0kB=(0.01+0.05*0.25)*10e-9*1000e6+(0.01+0.05*0.25)*
*(1-0.0)*125e-9*1000e6= 3.0375 clocks per instr
MemStall_256kB=(0.01+0.05*0.25)*10e-9*1000e6+(0.01+0.05*0.25)*
*(1-0.55)*125e-9*1000e6= 1.4906 clocks per instr

MemStall_512kB=(0.01+0.05*0.25)*10e-9*1000e6+(0.01+0.05*0.25)*
*(1-0.75)*125e-9*1000e6= 0.9281 clocks per instr
MemStall_1024kB=(0.01+0.05*0.25)*10e-9*1000e6+(0.01+0.05*0.25)*
*(1-0.80)*125e-9*1000e6= 0.7875 clocks per instr
MemStall_2048kB=(0.01+0.05*0.25)*10e-9*1000e6+(0.01+0.05*0.25)*
*(1-0.85)*125e-9*1000e6= 0.6469 clocks per instr

CPI_0kB = 0.8350+3.0375 = 3.8725
CPI_256kB = 0.8350+1.4906 = 2.3256
CPI_512kB = 0.8350+0.9281 = 1.7631
CPI_1024kB = 0.8350+0.7875 = 1.6225
CPI_2048kB = 0.8350+0.6469 = 1.4819

CPI_0kB*pris_0kB = 3.8725*250 = 968.125
CPI_256kB*pris_256kB = 2.3256*350 = 813.96
CPI_512kB*pris_512kB = 1.7631*500 = 881.55
CPI_1024kB*pris_1024kB = 1.6225*1000 = 1622.5
CPI_2048kB*pris_2048kB = 1.4819*3000 = 4445.7

256kB cache ger bäst prestanda per pris.

b. Genom att använda det vi har beräknat i uppgift a så får vi.
T=I*CPI*Tc=10e9*2.3256*1/1000e6=**23.25 sekunder**.

c. Vi vill alltså ha reda på vilken cachestorlek som ger en MemStall som är mindre än MemStall för den gamla processorn.
Den med 2048kB cache.

4.

a. Dela upp händelserna i busscykler, och översätt sedan till processorcykler. Det går två processorcykler på en busscykel. Eftersom write-back tillämpas måste man också ta hänsyn till att en tillbakaskrivning av ett block kan krävas innan det sökta blocket läses in. Observera att var och en av dessa två blocköverföringar måste hanteras som en separat transaktion.

Busscykel 0: Cachemissen detekteras och begäran skickas ut om tillgång till bussen. I värsta fall inträffar cachemissen i den första av de två processorcyklerna som utgör denna busscykel, varför ett bidrag på en processorcykel fås till totala väntetiden. Påföljande busscykler bidrar alla med två processorcykler.

Busscykel 1: Tillstånd att använda bussen ges direkt eftersom det antas att bussen inte är belastad av annan trafik.

Busscykel 2: Adress till det cacheblock som ska skrivas ut sänds över bussen till minnet.

Busscykel 3-17: Åtkomst av blocket görs i minnet. Eftersom minnesåtkomsttiden är 75 ns och busscykeltiden är 5 ns (1/(200 MHz)), så tar detta 15 busscykler.

Busscykel 18: Första ordet skrivs till minnet.

Busscykel 19: Andra ordet skrivs till minnet.

Busscykel 20: Tredje ordet skrivs till minnet.

Busscykel 21: Fjärde ordet skrivs till minnet. I och med detta släpps bussen för denna transaktion. Eftersom det sökta cacheblocket också ska läsas in i ytterligare en transaktion så begärs dock omedelbart tillstånd för ytterligare en transaktion.

Busscykel 22: Tillstånd att använda bussen ges direkt eftersom det antas att bussen inte är belastad av annan trafik.

Busscykel 23: Adress till det cacheblock som ska skrivas ut sänds över bussen till minnet.

Busscykel 24-38: Åtkomst av blocket görs i minnet. Eftersom minnesåtkomsttiden är 75 ns och busscykeltiden är 5 ns (1/(200 MHz)), så tar detta 15 busscykler.

Busscykel 39: Första ordet skrivs till cacheminnet.

Busscykel 40: Andra ordet skrivs till cacheminnet.

Busscykel 41: Tredje ordet skrivs till cacheminnet.

Busscykel 42: Fjärde ordet skrivs till cacheminnet. I och med detta släpps bussen för denna transaktion. När denna cykel är slut så är också cachemissen hanterad.

Totalt tar det alltså som mest 42 busscykler = 84 processorcykler + 1 processorcykel (från busscykel 0) = **85 processorcykler** från det att en cachemiss upptäcks till det sökta blocket laddats i cacheminnet.

b. Av resonemanget i förra deluppgiften framgår att en busstransaktion tar 21 busscykler inklusive cykeln då tillstånd att använda bussen ges. Sådana transaktioner kan som mest följa direkt efter varandra. Då varje transaktion innebär att 4 ord = 16 byte data överförs på bussen, och bussen har busscykeltiden 5 ns, fås att den maximala effektiva bandbredden är 16B/(21*5 ns) = **152 MB/s**.

c. **Se avsnitt 8.4 i kursboken.**

5.

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