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

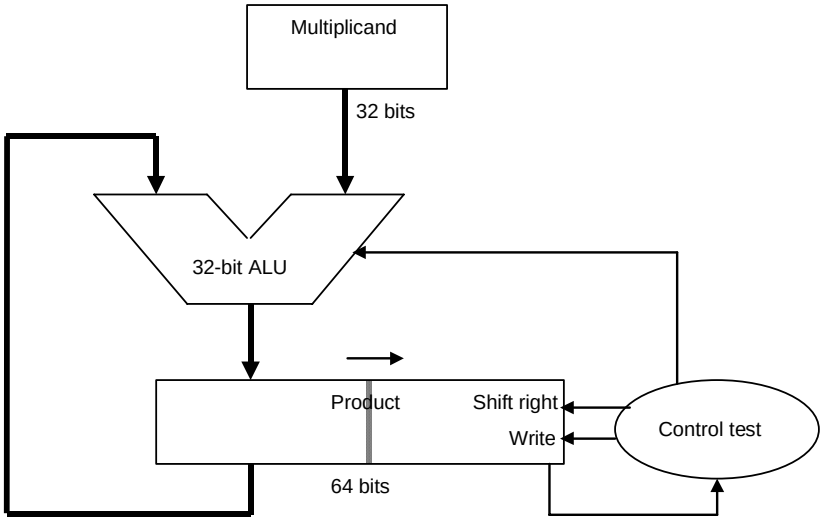
Datorsystemteknik

12/1 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. **Se kursboken.**
Placera multiplikatorn i de 32 minst signifikanta bitarna av produktregistret. Använd 32-bitars addition och placera resultatet av additionerna i de 32 mest signifikanta bitarna av produktregistret.



b.

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

c. sign = 1 (negativt)
exponent field = $01101000_2 = 104_{10}$ = exponent +127 =>
exponent = -23
significand field = $01000000000000000000000_2$ =>
significand = 1.01_2
Flyttalet är $-1.01 * 2^{-23} = -101 * 2^{-25} =$ **$-5 * 2^{-25}$**

d. Tvåkomplementsform, och talet är negativt. Ta fram det positiva talet genom att ta tvåkomplementet:
Invertera: 01001011110111111111111111111111
Addera 1: 01001011111000000000000000000000
Detta tal kan skrivas $2^{30} + 2^{27} + 2^{25} + 2^{24} + 2^{23} + 2^{22} + 2^{21}$ =
$(2^9 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) * 2^{21}$ =
$(512 + 64 + 16 + 8 + 4 + 2 + 1) * 2^{21} = 607 * 2^{21}$
Det sökta talet är alltså **$-607 * 2^{21}$** .

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**.

3.

a. Bussen utnyttjas effektivast vid överföring av 4 dataord åt gången. Fyra ord överförs då på fem bussyckler, och det går 100 * 10^6 busscykler/s (100 MHz). Maximal databandbredd för bussen är alltså 4 ord/5 cykler * 4 B/ord * 100 * 10^6 busscykler/s = **320 MB/s**.

b. Processorn utför 500 MHz/1,0 = 500 * 10^6 instruktioner/s. Dela upp bandbreddskravet i bidrag från läsningar i instruktionscache, läsningar i datacache, och skrivningar i datacache.
Läsning i instruktionscache: 500 * 10^6 läsningar/s * 0,01 missar/läsning = 5 * 10^6 missar/s, 1 block = 4 ord = 16 B per miss, 16 B/miss * 5 * 10^6 missar/s = 80 MB/s.
Läsning i datacache: 0,25 läsningar/instruktion * 500 * 10^6 instruktioner/s * 0,02 missar/läsning * 16 B/miss = 40 MB/s.

Skrivning i datacache: 0,05 skrivningar/instruktion * 500 * 10^6 instruktioner/s = 25 * 10^6 skrivningar/s, write-through innebär 1 ord=4 B på bussen för varje skrivning, 25 * 10^6 skrivningar/s * 4 B/skrivning = 100 MB/s.
Total databandbredd för cacheåtkomster 80 + 40 + 100 MB/s = **220 MB/s**.

c. Överföring av 128 KB tar 0,2 ms + 7 ms + 128 KB/(10 MB/s) = 20 ms. På 20 ms överförs 2*128 KB, vilket leder till en effektiv databandbredd på 256 KB/20 ms = **12,8 MB/s**.

d. Här måste beaktas hur stor andel av tiden som bussen är upptagen av trafik till och från cacheminnen. Dela som tidigare upp i tre bidrag. Läsningar i instruktionscache tar 5 busscykler/miss * 1/(100 * 10^6) s/busscykel = 0,05 µs/ miss, missar inträffar i genomsnitt varje 1/5 * 10^6 s, och tar därför upp 0,05 * 10^{-6} * 5 * 10^6 = 25% av tiden på bussen. Läsningar i datacache tar också 0,05 µs/ miss, missar inträffar i genomsnitt varje 1/(500 * 10^6 * 0,02) s = 1/10^7 s, och tar därför upp 0,05 * 10^{-6} * 10^7 * 0,25 = 12,5% av tiden. Skrivningar i datacache tar 2 busscykler * 1/(100 * 10^6) s/busscykel = 0,02 µs, och tar därför upp 0,02 * 10^{-6} * 500 * 10^6 * 0,05 = 50% av tiden.
Totalt är alltså bussen upptagen med cacheminnestrafik 87,5% av tiden, och är därmed ledig för I/O 12,5% av tiden. Vid I/O utnyttjas bussens maximala databandbredd 320 MB/s. I/O kan därför totalt tillåtas en databandbredd på 0,125 * 320 MB/s = 40 MB/s. Det innebär att bakplansbussen kan belastas med maximalt 40/12,8 = **3 I/O-bussar**.

4.

a. **Se avsnitt 6.7 i kursboken** och nedan.

b. Observation: snurran löper två varv. 1+ 2*4 + 1 = 10 instruktioner ska exekveras. Inga stalls pga datakonflikter. Om rätt instruktion alltid hämtas efter hoppinstruktionen så fås inte heller några styrkonflikter. Första instruktionen utförs då i WB-steget i cykel 5, och sista instruktionen i cykel 5+9 = 14. Utan styrkonflikter tar programmet alltså 14 cykler, och det återstår bara att för varje metod beräkna hur många extra cykler (t.ex pga stalls) som krävs.
Always stall: I detta fall görs en stall till hoppinstruktionen utförts i MEM-steget. Det blir alltså ett tillägg av tre cykler för varje gång hoppinstruktionen utförs, dvs 2*3 = 6 extra cykler. **Vid always stall tar programmet 20 cykler**.
Assume not taken:
I detta fall gissar man att hoppet inte ska tas och hämtar därför instruktioner direkt efter hoppinstruktionen. Om det är en felaktig gissning, så upptäcks detta efter tre cykler som alltså blir bortkastade. I detta fall tas hoppet en gång,och inte en gång. Det blir alltså en felaktig gissning som kostar tre extra cykler. **Vid assume not taken tar programmet 17 cykler**.
Assume taken:
Detta är raka motsatsen till föregående fall, och eftersom hoppet utförs en gång och passeras en gång, så blir effekten densamma. Det blir alltså en felaktig gissning som kostar tre extra cykler. **Vid assume taken tar programmet 17 cykler**.

(Kommentar. Vad skulle hända om man använde delayed branch:
I detta fall läggs utgår man från att instruktionerna direkt efter hoppinstruk tionen kommer att utföras, och man planerar därför koden efter detta. Utfallet här blir alltså beroende av om man bedömer att instruktionerna direkt efter hoppinstruktionen är “ofarliga” eller ej. För att vara på den säkra sidan kan

man alltid lägg in nop-instruktioner, i detta fall tre stycken, direkt efter hopp-instruktionen. Här skulle kostnaden bli densamma som vid always stall. Det är också möjligt att man bedömer att sw-instruktionen och de därpå närmast följande två instruktionerna inte gör någon skada om de utförs även om hoppet tas. I så fall skulle man inte få någon extra kostnad alls. Mellanting mellan dessa båda lösningar är naturligtvis också möjliga, så vid delayed branch tar programmet 14, 16, 18, eller 20 cykler beroende på vad som antas om instruktionerna efter hopp-instruktionen.)

5.

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