TSEA28 Datorteknik Y, lösningar till tentamen 20-06-01

1. a) Kopiera GRx till AR, subtrahera med minne, och om positivt avsluta instruktionen. Testa positivt/negativt mha N och O-flaggan (N=O ifall positivt). Om negativt svar ska värden byta plats vilket görs med AR som mellanresultatsregister. Väljer placering av koden på mikrokodsadress 20 (antar den och fram till 0x30 är ledig).

```
Adress Mikrokod                          Kommentar
  20   buss:=GRx, AR:=buss, R:=0, S:=0,  ; Läs GRx till AR
       uPC:=uPC+1                        ;
  21   buss:=PM, AR:=AR-buss, uPC:=uPC+1 ; Räkna upp AR, påverka flaggor
  22   buss:=PM, AR:=buss,               ; Kopiera minnesvärde till AR
       uPC:=25 om N=1 annars uPC:=uPC+1  ; Testa N=1 (avgör om negativt)
  23   uPC:=0 om O=0 annars uPC:=uPC+1   ; N=0, O=0 => positivt, nästa instr.
  24   buss:=GRx, PM:=buss, R:=0, S:=0,  ; negativt (N=0,O=1), minsta till PM
       uPC:=27                           ; avsluta med störst till GRx
  25   uPC:=0 om O=1 annars uPC:=uPC+1   ; N=1, O=1 => positivt, nästa instr.
  26   buss:=GRx, PM:=buss, R:=0, S:=0,  ; negativt (N=1,O=0), minsta till PM
  27   buss:=AR, GRx:=buss, R:=0, S:=0,  ; största till GRx, nästa instr.
       UPC:=0
```

b) K1:s innehåll. 4 bit opcode => 16 adresser, 128 olika adresser i mikrokodsminnet => 7 bitars data.

Adress	Värde	Adress	Värde
0000	-------	1000	1000010
0001	-------	1001	0111000
0010	0110101	1010	-------
0011	-------	1011	-------
0100	-------	1100	-------
0101	0010100	1101	-------
0110	0110000	1110	-------
0111	-------	1111	-------

c) Räkna klockcykler inklusive hämtfas. 0xa123-0x6543=0x3be0 => N=0, O=1. Mikrokodsrader som körs blir: 0,1,2,3,20,21,22,23,24,27. Det blir alltså i detta fall 10 klockcykler.

2. a) DRAM läser ut data i burst (många ord på en gång). Detta stämmer bättre med minnesaccess för en dator med cache eftersom varje cachemiss också hämtas i som burst. DRAM är också långsamt och blir därför lämpligt med en dator med cache. Refresh för DRAM kan också döljas genom att flera läsningar/skrivningar till/från cache kan göras medan DRAM utför refresh.

b) Indexbitarna lagras inte, de används som adress i en cache.

c) Fördel: slipper hantera stack när subrutinanrop görs (kan göra ytterligare subrutinanrop direkt), vilket minskar antal instruktioner. Nackdel: varje subrutinanrop och återhopp behöver göra minnesaccesser (blir långsammare).

d) Data forwarding minskar bara bubblor i samband att data från en ALU-beräkning ska användas i nästa instruktion. Detta påverkar inte styrkonflikter eftersom styrkonflikter beror på att instruktioner från fel adress hämtas då hopp-instruktionen inte påverkat PC tillräckligt fort.

TSEA28 Datorteknik Y, lösningar till tentamen 20-06-01

3. Subrutinen är enklast att beskriva som initiering av resultvärde, därefter en loop som adderar värde från tabell och beräknar nästa radadress, och loop avslutas när radadress blir första radadress. Använd r2 anger hur långt in i tabellen data finns.

```
Subrutin:   mov   r1,#0          ; nollställ summa
            mov   r2,#0          ; starta på 1:a raden i tabellen
loop:       ldrsh r3,[r0,r2]     ; hämta 16 bitar 2-komplement
            add   r1,r1,r3
            add   r2,r2,#2       ; peka på sista 16 bitarna på raden
            ldrh  r2,[r0,r2]     ; hämta nästa radnummer
            lsls  r2,r2,#2       ; multiplicera med 4, sätt flaggor
            bne   loop
            bx    lr
```

4. Avbrottsrutin så alla generella register måste återställas efter anrop. För ARM Cortex-M görs detta automatiskt för R0-R3 och R12.

```
GPIOADATA   .equ 0x400043fc
GPIOBDATA   .equ 0x400053fc

avbrott:  push  {lr,r4,r5,r6}; måste spara undan register eftersom det är avbrott
          mov   r0,#(GPIOADATA & 0xffff)
          movt  r0,#(GPIOADATA >> 16)
          ldrb  r1,[r0]
          mov   r4,#0x80    ; peka på bit 7
          mov   r5,#0       ; antal bitar som varit 1
loop1     ands  r6,r4,r1    ; testa bit som r4 pekar på
          beq   next1       ; biten var 0
          add   r5,#1       ; biten var 1
next1     lsrs  r4,#1       ; peka på nästa bit
          bne   loop1

          mov   r2,#(GPIOBDATA & 0xffff)
          movt  r2,#(GPIOBDATA >> 16)
          ldrb  r3,[r2]
          mov   r4,#0x80    ; initiera bit att undersöka (bit 7 först)
          mov   r12,#0      ; antal hittade ettor
loop2     ands  r6,r4,r3    ; testa aktuell bit
          beq   next2       ; hittade inte bit, ta nästa
          add   r12,#1      ; hittade en etta, räkna upp antal hittade ettor
next2     lsrs  r4,#1       ; ta nästa bit (nästa bit till höger)
          bne   loop2       ; finns det några kvar? Om så ta gör om test

cmp       r5,r12            ; se vilken som är störst
          ble   moreb
          bl    Sub1        ; korrekt subrutinanrop till annan subrutin
          strb  r3,[r0]     ; spara värdet från portB i den portA
          b     klar        ; avsluta
```

```
moreb   bl      Sub2        ; anropa andra subrutinen
        strb    r1,[r2]     ; skriv värdet från portA i portB

klar    pop     {lr,r4,r5,r6}
        bx      lr
```

5. a) $-25_{10} = -(25_{10}) = -(011001_{2C}) = (100110+1)_{2C}=100111_{2C}$

b) $-0x67 = -(01100111_{2C}) = (10011000+1)_{2C} = 10011001_{2C}$.

$0x34 = 00110100_{2C}$

```
 Carry:   11
         00110100
        +10011001
         --------
         11001101
```

Svar: 11001101 = 0xCD

c) Flaggor från subtraktionen i b): Z=0, C=0, N=1, O=0

Motivering: Eftersom svar inte är 0 => Z=0, Carry ut => C=0, teckenbit (MSB) = 1 => N=1, och om indata tolkats som 2-komplementstal skulle ett positivt och negativt tal adderas => belopp hos slutresultat måste vara mindre än indatatermernas belopp => inget spill => O=0. Alternativt för O-flaggan: carry in till MSB är 0 och carry ut från MSB är 0 => O = 0.

d) Bitvis AND

```
    00110100
AND 10010110
------------
    00010100
```

Flaggorna Z=0 eftersom svaret ej är 0, och N=0 eftersom MSB i svaret = 0.

6. a) Varje cacheline är 128 byte, och varje pixel är 16 bitar (2 byte). Det går därför plats 64 pixlar per cacheline. Om en rad ska läsas fås därför 640/64=10 cachemissar. Men om raden inte börjar på en adress jämnt delbar med 128 kommer istället 1 extra cachemiss fås, dvs 11 cachemissar. Svaret blir därför: 10 eller 11 (mest sannolikt 11).

b) 128 byte i en cacheline => 7 bitars byteposition. Totalt 32 bitars adress. Antal bitar i tag totalt-indexbitar-bytepositionsbitar = 32-8-7 = 17 bitar.

c) Antal cachelines i en väg kommer vara 2^8 = 256 cachelines. En väg i cachen kommer därför innehåll 256*128 = 32768 byte. Hela cachen är 128*1024 byte stor, så det blir (128*1024)/32768 = 4 vägar i cachen