TSEA28 Datorteknik Y, lösningar till tentamen 19-05-28

1. a) Kopiera argument från minne till AR (ASR redan satt korrekt baserat på M-bitarna). Subtrahera GRx från AR (glöm inte S och R bitarna). Instruktionen är klar om Z=0. Annars ökas PC med 1 och instruktionen avslutas.

```
Adress Mikrokod                          Kommentar
  10   buss:=PM, AR:=buss,               ; Läs argument från PM
       uPC:=uPC+1                        ;
  11   buss:=GRx, AR:=AR-buss, R:=0,     ; Subtrahera GRx, påverka flaggor
       S:=0,uPC:=uPC+1
  12   uPC:=0 om Z=0, annars uPC:=uPC+1  ; Kontrollera om lika, hoppa till 0
                                         ; om olika
  13   uPC:=0, PC:=PC+1                  ; Räkna upp PC (dvs hoppa över 1
                                         ; instruktion, start nästa instruktion
```

b) Adresseringsmod väljs från M-bitarna som mha K2 hoppar till rätt rutin. 128 ord i mikrominnet => 7 bitar brett data i K2.

Adress	Värde
00	0000011
01	0000100
10	0000101
11	0000111

c) Instruktionskodning: opcode 0xD, Grx=0x2, M=0x00, A:=0xE1 => 0xD8E1

2. a) En pipelined processor arbetar med flera instruktioner per klockcykler, men varje instruktion tar flera klockcykler att slutföra. Om bubblor uppstår i pipelinen kommer mindre än 1 instruktion per klockcykel slutföras.

b) Vid förhämtning gissar cache på adresser som troligen kommer begäras av processorn i framtiden och hämtar dessa. Målet är att hämta data som processorn senare kommer begära.

c) Resultatet av en addition (eller subtraktion) kan inte ge Z=1 och N=1 samtidigt, då N=1 betyder att MSB-biten är ett, och Z=1 fås bara om alla bitar (inklusive MSB) = 0. O-flaggan har ingen inverkan på detta.

d) Vid data-forwarding i en pipelinad processor skickas data direkt från utdata från ALU till indata till ALU utan att först lagras i registerfilen. På detta sätt undviks bubblor i pipeline pga datakonflikt.

e) SRAM är ett Statiskt RAM. Detta är ett volatilt minne (tappar värde vid spänningsbortfall). SRAM är snabba och kräver ingen refresh (i motsats till DRAM). Varje minnescell tar mer yta på kretsen än motsvarande bit i DRAM.

3. Registeranvändning: r1: antal element kvar att kontrollera, r0: adress till nästa element att kontrollera, r2: antal kontrollerade element (nollställs vid start), r3: största värdets position i listan, r4: största värdet så här lång. 2-komplement data => BLT, BGT, BLE, BGE kan användas som vilkorliga hopp.

TSEA28 Datorteknik Y, lösningar till tentamen 19-05-28

ARM Cortex-M4:

```
Subrutin:   mov   r2,#0         ; Nuvarande position i vektor (startar på 0)
            mov   r3,#0         ; Position för största värdet
            ldr   r4,[r0],#4    ; Hämta 1:a värdet i vektor
next:       subs  r1,#1         ; minska antal värdet att testa
            beq   slut          ; Hoppa ur loop om klar
            add   r2,r2,#1      ; öka elementräknare
            ldr   r5,[r0],#4    ; hämta nästa värde
            cmp   r5,r4         ; (r5-r4)
            blt   next          ; hoppa om r5 < r4 (gamla värdet störst)
            mov   r4,r5         ; Nya värdet större, uppdatera största värde
            mov   r3,r2         ; vilken position största värdet har
            b     next          ; testa om klar med listan
slut:       mov   r0,r4
            mov   r1,r3
            bx    lr
```

4. Avbrott kräver att alla register återställs. Eftersom anropet till subrutin1 förändrar r4, r5 och lr måste dessa sparas innan anropet. I processorn sparas annars r0-r3 automatiskt vid avbrott (och kan därför användas fritt).

ARM Cortex-M4:

```
Avbrott:    push  {r4,r5,lr}      ; spara register
            bl    Subrutin1
            and   r0,r0,#0xf      ; maska fram de 4 minst signfikanta bitarna
            mov   r1,#(0x40001238 & 0xffff)    ; r1 pekar på adress
            movt  r1,#(0x40001238 >> 16)
            ldrb  r2,[r1]         ; hämta gamla värdet i minnet (1 byte)
            and   r2,r2,#0xf0     ; behåll de gamla bitarna
            orr   r0,r0,r2        ; lägg ihop bitarna
            strb  r0,[r1]         ; spara nya värdet i minnet
            pop   {r4,r5,lr}      ; återställ register
            bx    lr
```

5. a) $-5_{10} = -(0101_2) = (1010 + 0001)_{2C} = 1011_{2C}$. Teckenförläng $1011_{2C} = 111011_{2C}$.

b) $0xC * 0xB = 1011_2 * 1100_2$

```
    1011
   *1100
--------
    0000
   0000
  1011
+1011
--------
10000100
```

$1000\ 0100_2 = 84_{16}$

c) Summering i 16-bitars dator genererar 16 bitars resultat. Utdata är oberoende om indata ses som positiva heltal eller som 2-komplement.

```
Carry: 111       1
        0110001000100011
       +1011110110101100
        ----------------
        0001111111001111
```

Flaggan Z = 0 då resultat inte är 0. Flaggan C = 1 då carry fås (utresultat större än vad som kan representeras om indata ses som positiva heltal). Flaggan N = 0 (MSB bit = 0). Flaggan O = 0 då resultatet får plats om indata ses som 2-komplement (antingen baserat på carry in i MSB = carry out, alternativt undersök värdet om 2-komplement: 0x6223-0x4254=0x3fcf vilket är mindre än 0x7fff).

d) 2-komplementstal av olika längd: Måste teckenförlänga det kortaste innan addition.

```
Carry:  11 1 1
         1110101
        +0110101
        --------
         0101010
```

0101010_{2C} = 32+8+2=42

6. a) Antal rader (= antal cachelines) totalt i cache är 128*1024/64=2048 rader. Dessa delas upp mellan 4 vägar => Antal rader i en väg = 2048/4 = 512 rader.

b) 512 rader i en väg innebär 9 bitars index. Bytepositionen i en cacheline behöver 6 bitar (2^6 = 64). Alla övriga bitar i adressen blir tag, dvs 64-9-6=49 bitar tag.

c) Adressen på 64 bitar delas upp i (från höger) bit 0 – bit 5 = byteposition, bit 6 – bit 14 index, bit 15 – bit 63 tag. Steglängden x1000 har en 1:a i bitposition 12. När adressen räknas uppåt kommer därför endast bit 12 – bit 14 att räkna upp. Efter 8 olika adresser blir då index samma som det var från början. Det finns 4 vägar så cachen klarar av 4 kopior av samma index innan data kastas ut => 4 * 8 = 32 läsningar kan göras innan data kastas ut.

```
       tag           index   byteposition
0100 0000 0000 0000 0|000 0000 00|00 0000    1:a adressen
0100 0000 0000 0000 0|001 0000 00|00 0000    2:a adressen
0100 0000 0000 0000 0|010 0000 00|00 0000    3:e adressen
                    :
0100 0000 0000 0000 0|111 0000 00|00 0000    8:e adressen
0100 0000 0000 0000 1|000 0000 00|00 0000    9:e adressen, samma index som 1:a
                    :
0100 0000 0000 0001 1|111 0000 00|00 0000    32:a adressen, samma index 4 ggr
0100 0000 0000 0010 0|000 0000 00|00 0000    33:e adressen, samma index 5 ggr! Kastar ut
```