TSEA28 Datorteknik Y, lösningar till tentamen 18-08-21, reviderad 181029

1. a) IR = antal (de 8 minst signifikanta bitarna), GRx = från, GRy (M-fältet) = destination. Flytta data, räkna upp adresserna, och minska värdet i IR med 1. Kontrollera om de 8 minst signifikanta bitarna = 0, i så fall är instruktionen klar.

```
Adress Mikrokod                          Kommentar
  n    buss:=GRx, ASR:=buss, R:=0, S:=0, ; Läs indata; Peka på adress från GRx
       uPC:=uPC+1                        ; med ASR-registret
 n+1   buss:=PM, AR:=buss, uPC:=uPC+1    ; Spara värdet i AR
 n+2   buss:=GRx, ASR:=buss, R:=0, S:=1  ; Peka på adress som GRy (M-fältet)
       uPC:=uPC+1                        ; innehåller, placeras i ASR
 n+3   buss:=AR, PM:=buss, uPC:=uPC+1    ; Spara värdet på destinationsadress
 n+4   buss:=GRx, AR:=buss, R:=0, S:=0   ; räkna upp GRx mha AR
       uPC:=uPC+1
 n+5   buss:=uM(värde 1), AR:=AR+buss,   ; öka adress med 1
       uPC:=uPC+1
 n+6   buss:=AR, GRx:=buss, R:=0, S:=0   ; spara nya adressen i GRx
       uPC:=uPC+1
 n+4   buss:=GRx, AR:=buss, R:=0, S:=1   ; räkna upp GRy mha AR
       uPC:=uPC+1
 n+5   buss:=uM(värde 1), AR:=AR+buss,   ; öka adress med 1
       uPC:=uPC+1
 n+6   buss:=AR, GRx:=buss, R:=0, S:=1   ; spara nya adressen i GRx
       uPC:=uPC+1
 n+7   buss:=IR, uPC:=uPC+1, AR:=buss    ; räkna ned antal flyttar att göra
       uPC:=uPC+1
 n+8   buss:=uM(värde 1), AR:=AR-buss    ; Minska värdet med 1
       uPC:=uPC+1
 n+9   buss:=AR,IR:=buss,uPC:=uPC+1      ; Spara det minskade värdet
 n+10  buss:=uM(värde 0xff), uPC:=uPC+1  ; ta bort effekt av OPCODE, GRx och M
       AR:=AR AND buss
 n+11  uPC:=n om Z=0 annars uPC:=uPC+1   ; nästa varv i loop om inte noll
 n+12  uPC:=0                            ; nästa instruktion
```

En liten kommentar: I datormodellen i tentan finns ingen möjlighet att kopiera A-fältet från IR till LC-registret. Trots detta har inga poängavdrag gjorts om LC använts för att lösa uppgiften.

b) Med 10 bitar kan 2^10=1024 olika maskinkoder avkodas mha K1.

2. a) En fullt associativ cache kan placera vilken adress som helst på varje enskild cacheline. En gruppassociativ cache har endast ett begränsat antal cacheline som en viss adress kan placeras på (antal vägar). Notera att fullt associativ är inte samma som direktmappad cache (den senare är en gruppassociativ cache med bara en väg).

b) Delayed branch innebär att även instruktionen efter en hoppinstruktion utförs.

c) En VLIW-processor utför långa instruktionsord uppbyggda av ett flertal små instruktioner i varje klockcykel. Dessa långa instruktionsord sätts samman vid kompileringen. En superskalär processor utför flera små instruktioner i varje klockcykel, men det är processorn själv som väljer vilka instruktionerna ska vara.

d) Nej, det finns olika många register i olika processorer. Exempelvis har ARM 16 generella register, medan mikrokodsmodellen bara har 4.

e) Nej, en processor kan vara implementerad utan mikrokod. Processorn i pipelineexemplen i boken saknar mikrokod.

3. Subrutinen är enklast att beskriva som en loop över 8 tecken, där man plockar fram ett tecken åt gången. behöver hämta värde, placera i utvektor, sedan hämta adress till nästa värde, testa om värdet är noll, och om inte hoppa till början. Flytta siffran (4-bitarna) längst till vänster genom rotation så bitarna istället hamnar längst till höger. Eftersom ARM saknar rotation till vänster så roteras istället åt höger så samma effekt fås.

ARM Cortex-M4:

```
Subrutin:   mov   r2,#8           ; nollställ räknare för antal varv
loop:       ror   r0,#(32-4)      ; rotera 4 steg åt vänster (vänstra siffran
                                  ;flyttas längst till höger
            and   r1,r0,#0x0f     ; maska fram högra siffran (4 bitar)
            add   r1,r1,#0x30     ; gör om till ASCII (stämmer för 0-9)
            cmp   r1,#0x3a        ; A-F?
            blt   numeric         ;   nej, 0-9, klara med ASCII
            add   r1,#(0x41-0x3a)       ; ja, justera ASCII till A-F
numeric:    bl    0x01234568      ; Skriv ut ASCII-tecknet
            subs  r2,r2,#1        ; klara med alla siffror?
            bne   loop            ; nej, ta nästa
            bx    lr
```

68000:

```
Subrutin:   move.b#8,D2           ; Antal tecken att skicka ut
loop:       rol.l   #4,D0         ; flytta vänsta tecknet längst till höger
            mov.l   D0,D1
            and.l   #$0f,D1       ; Maska fram högra siffran (4 bit)
            add.b   #$30,D1             ; gör om till ASCII (stämmer för 0-9)
            cmp     #$3a,D1             ; A-F?
            blt     numeric             ; nej, klar att skriva ut
            add.b   #($41-$3A),D1       ; ja, korrigera för A-F
numeric:    jsr     $01234568
            sub     #1,D2
            bne     loop
            rts
```

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

TSEA28 Datorteknik Y, lösningar till tentamen 18-08-21, reviderad 181029

Arm Cortex-M4:

```
inputport .field   0x401000C0,32
destaddr  .field   0x20001010,32

avbrott:  cpsid    i             ; stoppa även avbrott med högre prioritet
          ldr      r1,inputport  ; peka på indataportens adress
          ldr      r2,destaddr   ; peka på plats där destinationsadressen ligger
          ldr      r3,[r2]       ; hämta destinationsadressen
          mov      r12,#512      ; antal byte att flytta
loop:     ldrb     r0,[r1]       ; läs en byte
          strb     r0,[r3],#1    ; spara i minnet, öka adress med 1
          subs     r12,r12,#1    ; räkna ned antal byte kvar att flytta
          bne      loop          ; om fler, ta nästa
          str      r3,[r2]       ; spara adress
          cpsie    i             ; tillåt avbrott igen
          bx       lr
```

68000:

```
Avbrott:  or.w     #$0700,SR          ; stoppa ytterligare avbrott (återställs av RTE)
          move.l   D0,-(A7)           ; push D0 på stack
          move.l   A0,-(A7)           ; push D1 på stack
          move.l   #512,D0            ; antal data att flytta
          move.l   $20001010,A0       ; hämta startadress att placera data på
loop:     move.b   $401000C0,(A0)+    ; flytta data till minne, öka A0 med 1
          sub.l    #1,D0              ; räkna ned antal kvar att kopiera
          bne      loop:              ; klar? Om inte hoppa
          move.l   A0,$20001010       ; spara adress
          move.l   (A7)+,A0           ; återställ registren
          move.l   (A7)+,D0
          rte                         ; klar med avbrottet
```

5. a) 2-komplementstal => båda måste teckenförlängas till 8 bitar för att svaret ska bli korrekt.

```
 00100110
+11110111
 --------
 00011101
```

b) $0xE3 = 11100011_2$, $0x86 = 10000110_2$.

```
 11100011
+10000110
 --------
 01101001
```

Summering av MSB-bitarna ger carry => C=1. Resultatet är inte 0 => Z=0, MSB=0 => N=0. Summering av MSB-1 bitarna ger ingen carry. Olika carry från MSB och MSB-1 => O=1.

Alternativt om man ser värdena som 2-komplementstal så adderas 2 negativa tal, men N säger att resultatet är positivt => har fått spill => O = 1.

c) $-5_{10} = -(5_{10}) = -(000101_2) = (111010+1)_{2C} = 111011_{2C}$

d) Flaggorna N (negativ) och O (overflow/spill) visar om resultatet av subtraktionen av två tvåkomplementstal är negativt. N och O ska vara olika för att svaret ska vara negativt.

6. a) 9 bitars index => 2^9 cachelines per väg. 2-vägs => 2*2^9 cachelines totalt. 32 byte/cacheline => 32*2*2^9 = 32 KByte cache.

b) 32-byte cacheline => 5 minst signfikanta bitarna i adress används för att peka på byte i cacheline. Bit 5 upp till bit 13 av adressen används för index. Resten, bit 14 t o m 31 används för tag. Startadress 0x00004000 (det är en 32-bitars processor) har tagvärde 1, index 0, bytepos 0. Nästa adress 0x00004100 har tagvärde 1, index 8, bytepos 0. Index ökar alltså med 8 i varje varv, och efter 64 läsningar är adressen 0x00007F00 (tag 1, index 0x1F8). När 65:e läsningen görs på adress 0x00008000 kommer tag öka till 2, index återgår till 0. Då cacheline med index 0 är upptagen i väg 1 kommer den lagras i väg 2. Efter ytterligare 63 läsningar har även väg 2 fyllts, och den 129:e läsningen tvingar bort data från läsningen av 0x000040000. Svar: 128 läsningar innan data kastas bort ur cache.

c) Write-through: Varje skrivning skickas också till minnet samtidigt, Write-back: Skrivning uppdaterar cache men inte minne förrän cacheline töms.