TSEA28 Datorteknik Y, lösningar till tentamen 170602 reviderad 170808

1. a) Mikrokodsadress 3, 4 respektive 5 anropas beroende på adresseringsmod. Direktadressering behöver lägga till en kopiering av IR till HR. Omedelbar adressering behöver lägga till kopiering av PC innan uppräkning av PC. Indirekt adressering startar raden innan, och andra radens kopiering av ASR behöver även göras till HR. Notera att det inte går att kopiera flera värden direkt (både till ASR och HR samtidigt) och att det inte går att läsa ASRs värde. Det är inte heller lämpligt att använda någon ALU-operation som ändrar flaggornas värde.

```
Adress Mikrokod                                Kommentar
   3   buss:=IR, HR:=buss, uPC:=uPC+1          ; direktadressering (M=00)
                                               ; Kopiera IR (adressdelen) till HR
   4   buss:=IR, ASR:=buss, uPC:=K1(OP)        ; Kopiera även IR till ASR
                                               ; och hoppa till rätt instruktion
   5   buss:=PC, HR:=buss, uPC:=uPC+1          ; omedelbar adressering (M=01)
                                               ; Kopiera PC till HR
   6   buss:=PC, ASR:=buss, PC:=PC+1,          ; Kopiera även PC till ASR, öka PC
       uPC:=K1(OP)                             ; med 1 och hoppa till instruktionen
   7   buss:=IR, ASR:=buss, uPC:=uPC+1         ; indirekt adressering (M=10)
                                               ; peka på adress i minnet
   8   buss:=PM, HR:=buss, uPC:=uPC+1          ; Hämta adress till argument ur
                                               ; minnet och placera i HR
   9   buss:=PM, ASR:=buss, uPC=K1(OP)         ; Placera adress även i ASR, hoppa
                                               ; till instruktion
```

Notera: Detta tar mer plats, så de tre instruktionerna STORE, JNE och LOAD behöver flyttas till adress 10 t o m 13. Alternativt kan nya versioner av adresseringsmoderna läggas på adress 11 och framåt, med hopp tillbaks till orginalkoden.

b) Utökning av mikrokod flyttar adresserna. Fortfarande 4 olika adresser där 4:e adressen inte är känd (odefinierad).

```
Adress Mikrokodsadress
   00   0000011  ; direkt adressering
   01   0000101  ; omedelbar adressering
   10   0000111  ; indirekt adressering
   11   -------
```

c) Litet fel i 2:a exemplet: Ska vara Z=1 som gör att GR1, GR3 och flaggor inte påverkas.

Algoritm: Kontrollera först om Z=1, avsluta om så. Annars flytta GRx till AR, addera GRy till AR, och flytta AR till GRy.

```
Adress Mikrokod                                Kommentar
  14   buss:=GRx, R:=0, S:=0, AR:=buss,        ; flytta GRx till AR, och avsluta
       uPC := 0 om Z=1                         ; om Z=1 (hopp till uaddr 0)
  15   buss:=GRx, R:=0, S:=1, AR:=AR+buss      ; påverka flaggor, addera GRx med
       uPC:=uPC+1                              ; GRy
  16   buss:=AR, R:=0, S:=1, Grx:=Buss         ; Spara resultat i GRy, starta
       uPC:=0                                  ; nästa instruktion
```

2. a) En pipelinead RISC-processor når bara sin höga prestanda om den kan starta en ny instruktion i varje klockcykel. Om fel instruktion startas pga hopp eller subrutinanrop så måste bubblor placeras i pipelinen. Dvs då väntar processorn på att hitta nästa instruktion att starta. Ju

större pipelinedjup desto större problem. Branch prediction beräknar vilken som ska vara nästa instruktion baserat på tidigare sekvens av instruktioner, vilket ofta (men inte alltid) stämmer.

b) En VLIW-processor startar flera delinstruktioner per klockcykel, men det är programmeraren som måste ange vilka dessa ska vara. En superskalär processor startar flera enklare instruktioner per klockcykel och det är processorn som bestämmer vilka och hur många det är som ska startas samtidigt.

c) SRAM (statiskt RAM) behåller minnesinnehållet så länge som strömmen är på, medan DRAM (dynamiskt RAM) tappar minnesinnehållet om det inte återläses (refresh) inom en viss tid (vanligen ett antal millisekunder mellan varje läsning). SRAM bygger på vanlig logic, medan DRAM använder kapacitanser för att lagra datavärdet.

d) DMA (Direct Memory Access) låter en processor slippa sköta överföringar av stora datamängder mellan I/O-enheter och minnet. DMA-enheten styr själv överföringen till/från I/O-enheten medan processorn utför andra instruktioner.

e) De gamla processorerna var ungefär lika snabba som minnet. Dvs det var ingen fördel med att använda en cache, eftersom svarstiden från en cache och svarstiden från minnet direkt skulle varit ungefär lika. En modern processor är ett flertal gånger snabbare än minnet, så utan cache skulle den tillbringa mesta tiden med att vänta på minnet (jfr labb 5, uppgift 1). Dvs läs och skrivhastighet för minnet har inte ökat lika mycket som klockfrekvensen hos processorn.

3. Algoritm: Loop: Läs nästa värde, öka pekare med 1, och kontrollera om värde=0. Om inte, utför xor, öka nyckel med 1 och skriv tillbaks till minnet och starta nästa varv i loop.

```
Subrutin:   move.b    (A0),D1     ; hämta nästa byte i meddelandet
            cmp.b     #0,D1       ; Testa om byte är 0
            bne       kryptera    ; Om inte noll ska det krypteras
            rts                   ; Om noll är vi klara
kryptera:   eor.b     D0,D1       ; gör exor mellan nyckel och data
            move.b    D1,(A0)+    ; Spara krypterad data
            add.b     #1,D0       ; Öka nyckel med 1
            jmp       Subrutin    ; Nästa databyte
```

4. a) De två talen är negativa, och deras summa är också negativ.

```
Carry-bitar:1111111     1111
             1011001000000011 ($B203)
            +1111111110101101 ($FFAD)
           ------------------
             1011000110110000 ($B1B0)
```

En snabb kontroll (byt till positiv heltalsform mha invertering av alla bitar följt av addition med 1): $B203=-($4DFC+1)=-$4DFD. $FFAD=-($0052+1)=-$0053. -$4DFD+(-$0053)= -

($4DFD+$0053)= -($4E50). Konvertera till 2-komplement: ($B1AF+1) = $B1B0. Vilket stämmer med svaret ovan. Inga extra bitar behövs för att representera svaret.

b) Svaret är negativt, utan overflow och inte noll. Carry visar minnessiffra om talen istället skulle ses som positiva heltal. Därför är Z=0, N=1, O=0, C=1.

c) Teckenförlängning behöver göras för att additionerna av de två talen ska kunna fungera. Utöka till 6-bitar representation (kopiera teckenbit)

```
  111010
 +111011
--------
  110101
```

Snabb koll: 111010_{2C} = -$(000101+1)_2$=-6, 111011_{2C}= -$(000100+1)_2$ = -5. Summan (-6)+(-5)=-11 = -$(001011)_2$=$(110100+1)_{2C}$=110101_{2C}. Vilket stämmer.

5. Avbrottsrutin betyder att alla register som ska användas måste först sparas på stacken. Avslutningen måste ske med RTE.

```
Avbrott:  move.l   D0,-(A7)     ; spara D0 på stack
          move.b   $10080,D0    ; läs in temperatur
          and.b    #$3F,D0      ; nollställ bit 6 och 7
          cmp.b    D0,$5000     ; jämför med målvärde
          beq      klar         ; lika, avsluta
          blt      mindre       ; lägre värde, minska
          move.b   $10084,D0    ; större, öka fläkt
          and.b    #$1F,D0      ; ta fram fläktvarvtal
          cmp      #$1F,D0      ; är redan på max?
          beq      max          ; ja
          add.b    #1,$10084    ; nej, öka varvtal med 1
          bra      klar

max:      move.b   #$9F,$10084  ; max: bit 7 = 1, maxvärde varvtal
          bra      klar

mindre:   move.b   $10084,D0    ; hämta aktuellt styrvärde
          and.b    #$1F,$10084  ; nollställ bit 7 (även 5 och 6)
          beq      klar         ; om resultat = 0 är redan varvtal 0
          sub.b    #1,$10084    ; annars minska med 1
klar:     move.l   (A7)+,D0     ; återställe register
          rte
```

6. a) 65536 byte cache totalt, varje cacheline är 64 byte => 65536/64=1024 cachelines

b) 16-vägs gruppassociativ => 1024/16=64 olika index => 6 bitar för index.

c) Om sekvensen startar på en slumpmässig adress är det mellan 0 och 63 ytterligare värden som kan användas. Alla antal är lika sannolika, så medelvärdet på antal cacheträffar efter första cachemissen är 31.5.