TSEA28 Datorteknik Y, lösningar till tentamen 20-10-21 reviderad 230525

1. a) Beräkna GRx-GRy. Kontrollera N och O-flaggan. Om O = N ta nästa instruktion, annars beräkna fält A (IR) + PC. PC innehåller redan adress till nästa instruktion. Sätt PC till ny adress och starta nästa instruktion. Optimering inlagd genom att starta beräkning av hoppadress (IR+PC) även om vi inte vet om hoppadressen behövs.

```
Adress Mikrokod                          Kommentar
  20   buss:=GRx, AR:=buss, R:=0, S:=0,  ; Läs GRx till AR
       uPC:=uPC+1                        ;
  21   buss:=GRx, AR:=AR-buss, R:=0,     ; Beräkna GRx-GRy (AR-GRy)
       S:=1, uPC:=uPC+1                  ; påverka flaggor
  22   buss:=PC, AR:=buss,               ; Kopiera aktuellt PC-värde till AR
       uPC:=25 om N=0 annars uPC:=uPC+1  ; Testa N=0 (avgör om negativt)
  23   buss:=IR, AR:=AR+buss             ; Beräkna ny adress
       uPC:=0 om O=1 annars uPC:=uPC+1   ; N=1, O=1 => positivt, nästa instr.
  24   buss:=AR, PC:=buss, uPC:=0        ; Utför hopp, avsluta instruktionen
  25   buss:=IR, AR:=AR+buss             ; beräkna ny adress
       uPC:=0 om O=0 annars uPC:=uPC+1   ; N=0, O=0 => positivt, nästa instr.
  26   buss:=AR, PC:=buss, uPC:=0        ; Utför hopp, avsluta instruktionen
```

b) K2:s innehåll. 2 bit M-fält => 4 adresser, 128 olika adresser i mikrokodsminnet => 7 bitars data.

Adress	Värde
0000	0000011
0001	0000100
0010	0000101
0011	0000111

2. a) En styrkonflikt kan bara uppstå om två instruktioner utförs helt eller delvis parallellt, dvs det kräver att processorn börjar hämta nästa instruktion innan föregående instruktion är färdig. Detta är definitionen av en pipelinad process, att nästa instruktion startar innan föregående är avslutad. En processor som inte är pipelinad avslutar alltid föregående instruktion, och därför kan inte en styrkonflikt fås.

b) SRAM är snabbare, SRAM tar större chipyta, SRAM kostar mer per byte. DRAM bygger på lagring av laddning i kapacitanser, DRAM kräver refresh för att behålla minnesinnehåll.

c) Icke-volatilt minne (som behåller minnesinnehåll vid spänningsbortfall) behövs i en dator eftersom processorn måste ha ett program att köra när strömmen slås på.

d) Virtuellt minne kan minska minnesbehov i en dator genom att låta primärminne tillfälligt lagras på sekundärminne (hårddisk eller liknande). Virtuellt minne kan även minska minnesbehov genom att bara allokera det minne som verkligen läses/skrivs. Virtuellt minne kan även minska minnesbehov genom att låta olika program dela samma minnesyta för t ex gemensamma subrutiner.

e) Register renaming löser problemet med WAR konflikt (Write After Read). Detta är konflikter som uppstår när en tidigare instruktion läser ett register som en senare instruktion ska

skriva till. I en superskalär processor (speciellt om out of order execution används) kan dessa instruktion hamna i samma klockcykel, och då kan inte samma register användas eftersom datat ännu inte lästs ut när instruktionen ska skriva ett nytt värde i det. Registerfilen består då ofta av fler register än de som anges i programmeringsmodellen. Exempel på instruktionsföljd:

```
str r0,[r1]      ; spara r0 i minne
mov r0,#123      ; nytt värde i r0, kräver att föregående instruktion klar först
```

3. Subrutinen läser antal värden, ökar den med 1 och sparar tillbaks. Därefter börjar data flyttas framåt med start i slutet av tabellen tills rätt plats blir ledig. Slutligen sparas nya värdet i tabellen.

```
Subrutin:   ldr   r3,[r0,#-2]   ; Hämta antal värden i tabell
            add   r3,r3,#1      ; Nya antalet efter nytt värde adderats
            strh  r3,[r0,#-2]   ; Spara nya antalet

            add   r4,r3,r0      ; beräkna sista plats i tabell (efter tillägg)
            add   r4,r4,r3      ; 2*antal beräknas som antal+antal

            sub   r5,r3,r2      ; beräkna antal att flytta (antal-önskad pos)

loop:       subs  r5,r5,#1      ; Klar med alla flytt?
            beq   spara         ; Om noll ska data hamna först

            ldrh  r6,[r4,#-4]   ; flytta ett värde
            strh  r6,[r4,#-2]   ;

            sub   r4,r4,#2      ; peka på föregående värde
            b     loop

spara:      strh  r1,[r4,#-2]   ; spara nya värdet
            bx    lr
```

4. Avbrottsrutin så alla generella register måste återställas efter anrop. För ARM Cortex-M görs detta automatiskt för R0-R3 och R12. Denna lösning är exempel på enkel men mycket kod. Det går även att göra motsvarande i en loop.

```
IOPORT          .equ 0x40001000

avbrott:  cpsid   i
          push    {lr,r4,r5}    ; måste spara undan register eftersom det är avbrott
          mov     r0,#(IOPORT & 0xffff)    ; Peka på I/O port
          movt    r0,#(IOPORT >> 16)
          ldr     r0,[r0]       ; Läs alla 32 bitar på en gång (krav!)

          and     r1,r0,#0x0000000f ; 4 minst signfikanta bitar (nibble 0)
          lsl     r5,r1,#28         ; flytta till mest signifikanta bitar
```

```
        and     r1,r0,#0x000000f0 ; nästa nibble (nibble 1)
        lsl     r4,r1,#20
        orr     r5,r5,r4      ; lägg till bitarna i r5

        and     r1,r0,#0x00000f00 ; nästa nibble (nibble 2)
        lsl     r4,r1,#12
        orr     r5,r5,r4      ; lägg till bitarna i r5

        and     r1,r0,#0x0000f000 ; nästa nibble (nibble 3)
        lsl     r4,r1,#4
        orr     r5,r5,r4      ; lägg till bitarna i r5

        and     r1,r0,#0x000f0000 ; nästa nibble (nibble 4)
        lsr     r4,r1,#4
        orr     r5,r5,r4      ; lägg till bitarna i r5

        and     r1,r0,#0x00f00000 ; nästa nibble (nibble 5)
        lsr     r4,r1,#12
        orr     r5,r5,r4      ; lägg till bitarna i r5

        and     r1,r0,#0x0f000000 ; nästa nibble (nibble 6)
        lsr     r4,r1,#20
        orr     r5,r5,r4      ; lägg till bitarna i r5

        and     r1,r0,#0xf0000000 ; nästa nibble (nibble 7)
        lsr     r4,r1,#28
        orr     r5,r5,r4      ; lägg till bitarna i r5

        bl      Subrutin1
        pop     {lr,r4,r5}
        cpsie   i
        bx      lr
```

5. a) $-9_{10} = -(9_{10}) = -(001001_{2C}) = (110110+1)_{2C}=110111_{2C}$

b) 94=64+16+8+4+2, 65=64+1. 8-bitars dator => 8 bitars ordlängd.

$94+65=159 = 128+16+8+4+2+1=10011001_{2C}$.

```
Carry:   1

         01011110
        +01000001
         --------
         10011111
```

Z=0 (svar inte 0), C=0 (ingen carry ut från bitposition 7),

N=1 (bitposition 7 i svaret = 1), O=1 (olika carry in out ut ur bitposition 7, alternativ motivering att två positiva tal ger negativt svar).

c) Skift till vänster motsvarar multiplikation med 2 för varje bitposition. Skift med 3 steg motvarar multiplikation med 2*2*2 = 8. Svar multiplikation med 8.

6. a) 32 byte per cacheline, 2^7 cachelines per väg, 8 vägar => 32*2^7*8 = 32768 byte.

b) Extra minne för lagring av tag. 32-7-5=20 bitar tag per cacheline. 2^7*8 cachelines => 20*2^7*8 = 20480 bitar.

c) Index startar på bitposition 5, och slutar på bitposition 11. Steglängd i adress best av en 1:a på bitposition 8, så bitpositionerna 11-8 i index som ökar för varje ny adress. Det är 4 bitar och därför 16 läsningar innan samma index används igen. 8 vägar => 8*16=64 läsningar innan samma index dyker upp en 9:e gång vilket tvingar tidigare data att kastas.