TSEA28 Datorteknik Y, lösningar till tentamen 18xxxx, reviderad 180524

1. a) Eftersom M-fältet används för att ange GRy så kommer inte ASR innehålla korrekt adress. ADDLONG behöver först addera den minst signifikanta delen och samtidigt påverka flaggorna, sedan räkna fram nästa adress (använder här PC) och slutligen addera mest signfikant del. Addition Kopiera GRx till AR, minska AR med 1 (från uM), spara nya värdet i GRx och hoppa till address 0 om inte noll, annars flytta IR till PC för att genomföra hoppet. Placera mikrokoden på 1:a lediga adress, dvs adress 11.

```
Adress Mikrokod                              Kommentar
  11   buss:=IR, R:=0, S:=0, AR:=buss,      ; IR till AR
       uPC:=uPC+1
  12   AR:=AR+buss, buss:=uM(=1)            ; AR=AR+1, påverka ej flaggor
  13   buss:=AR, ASR:=buss, uPC:=uPC+1      ; AR till ASR
  14   buss:=GRx, R:=0, S:=1, AR:=buss,     ; GRy till AR
       uPC:=uPC+1
  15   buss:=PM, AR:=AR+buss, uPC:=uPC+1    ; addera PM(ASR), påverka flaggor
  16   buss:=AR, GRx:=buss,R:=0,S:=1        ; minst signifikant del i GRy
  17   buss:=IR, ASR:=buss, uPC:=uPC+1      ; IR till ASR
  18   buss:=GRx, R:=0, S:=0, AR:=buss,     ; GRx till AR, hoppa över 1 adress
       uPC:=20 om C=0 annars uPC+1          ; om C=0
  19   AR:=AR+buss, buss:=uM(=1)            ; AR=AR+1, påverka flaggor
  20   AR:=AR+buss, buss:=PM, uPC:=0        ; Addera PM(ASR), påverka flaggor,
  21   buss:=AR, GRx:=buss, R:=0, S:=0,     ; Spara mest signifikant ord i GRx,
       uPC:=0                               ; hopp till 0 (nästa instruktion)
```

Denna lösning har ett specialfall som inte fungerar: Om GRy+M(A+1) ger carry samt GRx = 0x7fff eller 0xffff och M(A) = 0. Då fås inte C=1 utan C=0 respektive O-flaggan blir fel. Detta fall bortses ifrån i lösningen ovan. Specialfallen kan hanteras genom att först testa om M(A)=0, och om så är fallet görs allt som ovan utom additionen på rad 20.

b) 6 olika tider beroende på hur M-fältet inställt samt om 1:a additionen gav C=1. Adress 0-2 utförs alltid. Beroende på M utförs 3, 4 eller 5+6. Adresseringsmod M=11 är inte definierad. Från adress 11 och framåt är det antingen 10 eller 11 adresser som utförs, beroende på C-flaggan efter addition på rad 17. Detta ger möjliga exekveringstider av 3+1+10, 3+2+10, 3+1+11 eller 3+2+11. Dvs det kan ta 14, 15 eller 16 klockcykler.

c) Opcode = 0001, GRx=10, M=01, A=0x51 => 0x1951

2. a) I en VLIW-processor innehåller varje instruktion en fast uppsättning av mindre instruktioner att utföra parallellt i varje klockcykel. Det går inte att ändra ordning eller tidpunkt på varje deloperation, utan den bestäms vid kompileringen. I en superskalär processor startas flera mindre instruktioner samtidigt, och processorn kan själv avgöra vilka som ska startas samtidigt.

b) En styrkonflikt innebär att processorn inte vet var nästa instruktion att utföra finns. T ex kan detta bero på att ett vilkorligt hopp utförs, och innan villkoret är beräknat kan inte nästa instruktion hämtas.

c) Ja, två olika arkitekturer kan implementera samma instruktionsuppsättning. Exempel på detta är intel x86 kod som implementeras i helt olika arkitekturer av Intel och AMD.

d) RAM (Random Access Memory) tillåter skrivning av data från processorn in i minnet, medan ROM (Read Only Memory) endast tillåter läsning. Skrivning i RAM ska vara

ungefär lika snabb som läsning. RAM är volatilt (tappar innehåll vid spänningsbortfall) medan ROM är icke-volatilt (behåller innehåll även vid spänningsbortfall).

e) Aritmetisk skift kopierar teckenbiten vid skiftning, medan logisk skift skiftar in 0 till vänster.

3. Antagandet att rutinen ska vänta till I/O-enheten är redo framgick inte så tydligt. Om man antar detta är följande en lämplig lösning:

ARM Cortex-M:

```
IOFLAG:     .field 0x10040,32
IODATA:     .field 0x10042,32

SkickatillIO: ldr   r2,IOFLAG   ; Peka på statusregister
              ldrb  r3,[r2]     ; Läs statusregister
              ands  r3,r3,#0x02 ; Testa bit 1
              bne   SkickatillIO ; Bit 1 = 1, upptagen, försök igen
              ldr   r2,IODATA   ; Peka på dataregister
              ldrb  r3,[r0],#1  ; Läs 1 byte, peka på nästa
              strb  r3,[r2]     ; skicka ut värdet på dataregister
              subs  r1,r1,#1    ; räkna ned antal att skicka
              bne   SkickatillIO ; Inte klar, hoppa
              bx    lr
```

68000:

```
SkickatillIO: move.b    $10040,D1    ; hämta info från I/O om vi kan skicka
              and.b     #$02,D1      ; Testa om bit 1=0
              bne       SkickatillIO ; Bit var 1, vänta på tillgänglig I/O
              move.b    (A0)+,$10042 ; Skicka ut värde
              sub.w     #$01,D1      ; Kontrollera om klar
              bne       SkickatillIO ; nej, fler tecken kvar, skicka dom
              rts                    ; ja, hoppa tillbaks
```

4. a) Avbrottsrutin => måste spara register och återställa vid avslutning.

ARM Cortex-M: Automatiskt sparat register r0-r3 på stack.

```
Display:    .field 0x10080, 32
Tabell:     .field 0x5000,32
indexval:   .field 0x10082,32
Utseende:   .field 0x5010,32

Avbrott:    ldr   r0,indexval ; Peka på indexvalsadress (väljer byte i Tabell)
            ldr   r1,[r0]     ; läs indexval
            and   r1,r1,#0x03 ; maska fram bit1 och bit0
            ldr   r0,Tabell   ; Start av Tabell
            add   r0,r0,r1    ; Lägg till hur långt in i Tabell
            ldrb  r1,[r0]     ; Läs värdet i Tabell
```

```
                ldr     r0,Utseende     ; Peka på början av Utseende
                add     r0,r0,r1        ; Beräkna var i Utseende
                ldrb    r1,[r0]         ;    rätt utseende
                ldr     r0,Display      ; Peka på display
                strb    r1,[r0]         ; uppdatera display
                bx      lr
```

68000:

```
        Avbrott:    move.l      A0,-(A7)        ; Spara A0 på stack
                    move.l      D0,-(A7)        ; Spara D0 på stack
                    move.b      $10082,D0       ; hämta bit 1 och 0 (index till $5000)
                    and.l       #$03,D0         ; Nollställ alla bitar utom 1 och 0
                    add.l       #$5000,D0       ; Beräkna adress i tabell på $5000
                    move.l      D0,A0           ; peka på rätt värde
                    move.b      (A0),D0         ; Hämta indexvärde till tabell på $5010
                    add.l       #$5010,D0       ; Beräkna adress i tabell på $5010
                    move.l      D0,A0           ; peka på rätt sifferutseende
                    move.b      (A0),$10080     ; skicka ut på display
                    move.l      (A7)+,D0        ; återställ register
                    move.l      (A7)+,A0
                    rte                         ; klar med avbrott
```

b)

ARM Cortex-M:

Sparar automatiskt vid avbrott r0-r3, r12,LR,PC,PSR, dvs 8 ord á 4 byte styck. => Totalt 32 byte.

68000:

Vid avbrottet kommer stacken fyllas med återhoppsadress (4 byte), flaggor (2 byte) samt två 32-bitars register (A0 och D0) dvs totalt 4+2+4+4 = 14 byte.

5. a) 110-01101. Utöka först indata till 5 bitar => teckenförläng

```
Lånebitar:    L (decimalt)
          11110   (-2)
         -01101   (13)
       ---------
          10001   (-15)
```

Svar: 10001

b) $42_{10}=(32+8+2)_{10}=101010_2=2A_{16}$. Kontroll: 2*16+10=42

c) $B3+$AD=$160, dvs carry ut.

```
Carry:   1  11111
          10110011
         +10101101
         ---------
          01100000
```

Flaggorna blir Z=0 (resultat inte 0), C=1 (addition av positiva heltal större än tillgänglig storlek), N=0 (bit 7 = 0), O=1 (spill om 2-komplementstal adderas, dvs flaggan N visar fel värde).

TSEA28 Datorteknik Y, lösningar till tentamen 18xxxx, reviderad 180524

6. a) 16384/16=1024 cachelines

b) De 1024 cacheline delas upp i 2 grupper om 512 cacheline. Det behövs 9 bitar för att peka på en rad i en sådan grupp. Dessa 9 bitar kallas index.

c) Av adressen används de 4 minst signifikanta bitarna till position i cacheline, de följande 9 bitarna används för att peka på cachline (index). Resterande (32-9-4=19) bitar används till tag. Steglängden \$100 = 0000 0001 0000 0000_2. Varje ökning av steg ökar index-värdet med 0 0001 0000_2. Indexvärdet blir då 0x010, 0x020 etc. När index nått värdet 0x1f0 börjar det om på 0 igen. Detta nås efter 32 steg. Eftersom det finns två vägar i cachen kan samma index användas två gånger innan tidigare värde behöver kastas ur cache. Det betyder att 2*32 värden kan läsas innan cachelines börjar ersättas.