TSEA28 Datorteknik Y, lösningar till tentamen 23-08-15

1. a) Beräkna GRx+GRy. Om N-flaggan = O-flaggan är summan positiv, och då ska PC ökas med värdet i A-fältet av opcoden (dvs IR-registret). Eftersom PC redan räknats upp pga adresseringsmode är vi redan på PC+1 respektive PC+2 när A ska läggas till.

```
Adress Mikrokod                              Kommentar
0x0b   buss:=GRx, AR:=buss, R:=0, S:=0,      ; Kopiera GRx till AR
       uPC:=uPC+1                            ;
0x0c   buss:=GRx, R:=0, S:=1, AR:=AR-buss,   ; Addera GRy till AR, påverka
       uPC:=uPC+1                            ; flaggor
0x0d   buss:=PC, AR:=buss,                   ; Kopiera PC till AR, testa
       uPC:=0x11 om N=1 annars uPC:=uPC+1    ; N-flaggan
0x0e   uPC:=0x0 om O=1 annars uPC:=uPC+1     ; N=0, nästa instruktion om O=0
0x0f   AR:=AR+buss, buss:=IR, uPC:=uPC+1     ; N=0,O=0, Öka PC med A
0x10   PC:=buss,buss:=AR, uPC:=0             ; ny adress till PC, nästa instr.
0x11   uPC:=0x0 om O=0 annars uPC:=uPC+1     ; N=1, nästa instruktion om O=0
0x12   AR:=AR+buss, buss:=IR, uPC:=0x10      ; positivt, öka adress i AR
```

b) Räkna alla klockcykler från instruktionens start, dvs från rad 0. Raderna som utförs blir 0, 1, 2, 4, 5, 11, 12, 13, 14, 15, 16. Totalt blir det 11 klockcykler.

2. a) Virtuellt minne översätter alla adresser hos programmet från logiska till fysiska adresser mha av en tabell. Om den tabellen för ett visst program inte innehåller referenser till de fysiska adresserna för ett annat program så kan inte det första programmet komma åt data i det andra programmets minneyta. Om inte referenser till andra programmets fysiska adresser finns i tabellen för första programmet så kan inte heller andra programmet komma åt första programmets data.

b) Båda processortyperna kan starta flera delinstruktioner (VLIW) eller flera instruktioner (superskalär), men en VLIW processor är begränsad till att bara starta dom instruktioner som angivits, medan en superskalär processor kan vänta med att starta och/eller starta några instruktioner från föregående klockcykel exakt dom delinstruktioner som finns i instruktionen som ska utföras. En superskalär processor kan istället vänta med vissa instruktioner och kombinera ihop andra delinstruktioner från föregående och aktuell tidpunkt.

c) DRAM är komplicerat att kommunicera med och tar lång tid att läsa/skriva. Därför används det inte som cacheminne.

d) En addition av två negativa tvåkomplementstal kan ge upphov till N=0. Exempel på addition av två 8-bitars tal i en 8-bitars processor: 0x80+0x80 = 0x00 (dvs N=0). Spillflaggan visar när inte stämmer med summans egentliga tecken.

e) Antal minnesvärden som hämtas från minnet kan öka om en processor utrustas med cache. Exempel: Ett program läser en gång var 16:e byte i en 1024 byte lång vektor. Utan cache blir det 16 byte som hämtas från minnet. Med en cache med cachelinelängd = 16 hämtas även värdena runt omkring de efterfrågade värdena så totalt 16*16=256 byte läses istället.

TSEA28 Datorteknik Y, lösningar till tentamen 23-08-15

3. Programidé: Hämta 1 byte, skriv först ut mest signifikant nibble, sedan minst signifikant nibble. Översättning består i att lägga till ASCII-värdet för 0, och om värdet är 0xA eller större måste skillnaden mellan 0x41 och 0x3a (dvs 7) läggas till så 0xA blir ASCII 0x41 etc. Räkna sist ned räknaren för antal byte, och om inte klar skriv ut nästa.

```
subrutin:   push  {lr}          ; måste återställa lr innan retur
            mov   r3,r0         ; kom ihåg startadress för värdena
            mov   r4,r1         ; kom ihåg antal byte att skriva ut
loop:       ldrb  r5,[r3],#1    ; hämta nästa byte att skriva ut, öka adress
            shr   r0,r5,#4      ; skifta 4 bitar till höger
            add   r0,#0x30      ; Skapa siffror 0-9
            cmp   r0,#0x3a      ; Se om siffror korrekt
            blt   highnibble    ; siffra 0-9, gå vidare till utskrift
            add   r0,#7         ; 0x3a -> 0x41 för A, etc.
highnibble: bl    Subrutin1     ; skriv ut mest signifikant nibble
            and   r0,r5,#0xf    ; minst signifikant nibble
            add   r0,#0x30      ; Skapa siffror 0-9
            cmp   r0,#0x3a      ; Se om siffror korrekt
            blt   lownibble     ; siffra 0-9, gå vidare till utskrift
            add   r0,#7         ; 0x3a -> 0x41 för A, etc.
lownibble:  bl    Subrutin1     ; skriv ut mest signifikant nibble
            subs  r4,#1         ; se om klar med alla byte
            bne   loop          ; nej, ta nästa byte
            pop   {lr}          ; återställ lr
            bx    lr
```

4. Avbrottsrutin så alla generella register måste återställas efter anrop. För ARM Cortex-M görs detta automatiskt för R0-R3 och R12. Använd ldrh respektive strb för att bara läsa/skriva endast det antal byte som angetts i uppgiften.

```
IOPORT      .equ 0x40001000

avbrott:    cpsid   i           ; se till att inga andra avbrott kan ske
            mov     r3,#(IOPORT & 0xffff)   ; Peka på I/O port
            movt    r3,#(IOPORT >> 16)
            ldrb    r0,[r3]     ; Läs en byte
            lsl     r0,#26      ; teckenförläng (flytta bit 5-0 till 31-26)
            asr     r0,#26      ; Skifta tillbaks, kopiera teckenbiten
            mov     r1,#(0x20001000 & 0xffff)       ; peka på minne
            movt    r1,#(0x20001000 >> 16)
            ldr     r2,[r1]     ; hämta
            add     r2,r0       ; lägg till
            str     r2,[r1]     ; spara summan i minnet
            mov     r12,r2      ; save the new sum in a register
            cmp     r2,#0       ; testa teckenbit
            bge     positive    ; om bitvärde 0 positivt
            mov     r1,#0
positive:   sub     r2,r1,r2    ; r2 = 0-r2 dvs byt tecken på r2
```

```
            mov     r1,#(0x00123456 & 0xffff)       ; lång konstant
            movt    r1,#(0x00123456 >> 16)
            cmp     r2,r1       ; testa om absolutbelopp större än konstant
            ble     smaller     ; no, smaller so quit
            ldrb    r0,[r3,#0x10]       ; read byte to change bits in
            orr     r0,#0x08    ; set bit 3 in the value to 1
            cmp     r12,#0      ; check sign of new sum in memory
            blt     negative    ; negative sum
            and     r0,#0xfb    ; positive sum, bit 2 set to 0
            b       done        ; write back to I/O port
negative    orr     r0,#04      ; set bit 2 to 1
done        strb    r0,[r3,#0x10]       ; write value back to I/O
smaller     cpsie   i           ; enable interrupts again
            bx      lr          ; return from interrupt
```

5. a) EORS är en instruktion som beräknar bitvis xor av två register (r0 och r1) och placerar värdet i r0. De två värdena är 0x12345678 och 0x06543210. Resultatet blir då 0x14606468. Eftersom mest signifikant bit MSB i resultatet är 0 => N-flaggan = 0. Eftersom resultat inte är 0 => Z=0.

b) $110101_{2C} = -1*2^5+1*2^4+0*2^3+1*2^2+0*2^1+1*2^0 = -32 + 16 + 4 + 1 = -11_{10}$.

c) $011101_{2C}+1101_{2C} = 0000011101_{2C}+1111111101_{2C} = 0000011010_{2C}$

6. a) 16384 byte i cacheminnet, 2^(32-21) unika cachepositioner => associativitet 16384/(2^(32-21)) = 16384/2^11 = $2^{14}/2^{11} = 2^3 = 8$. Så det finns 8 olika platser för varje adress att placeras i cachen.

b) Antal adressbitar för byteposition = $_2\log 32 = 5$ bitar. Antal bitar i index = Antal adressbitar – antal bitar för tag – antal bitar för byteposition = 32 – 21 – 5 = 6 bitar.

c) De adresser som ska läsas är sekvensen nedan, med tag (21 bit), index (6 bit) och byteposition (5 bit) samt cachemiss/träff indikerat (tom cache från början)

0x2456789E: tag=0x48ACF, index=0x04, byteposition=0x1E, cachemiss
0x245678A4: tag=0x48ACF, index=0x05, byteposition=0x04, cachemiss
0x245678AA: tag=0x48ACF, index=0x05, byteposition=0x0A, cacheträff
0x245678B0: tag=0x48ACF, index=0x05, byteposition=0x10, cacheträff
0x245678B6: tag=0x48ACF, index=0x05, byteposition=0x16, cacheträff
0x245678BC: tag=0x48ACF, index=0x05, byteposition=0x1C, cacheträff
0x245678C2: tag=0x48ACF, index=0x06, byteposition=0x02, cachemiss
0x245678C8: tag=0x48ACF, index=0x06, byteposition=0x08, cacheträff

Så slutsatsen blir att det är 3 cachemissar som sker under läsningen av sekvensen.