TSEA28 Datorteknik Y, lösningar till tentamen 22-05-31, modifierad 23-06-14

1. a) Beräkna GRx-PM(ASR). Kopiera GRx till AR, kontrollera N och O-flaggan. Om O = N var differensen positiv så kopiera GRx till GR3 (GRx redan kopierad till AR), annars kopiera PM till GR3.

```
Adress Mikrokod                              Kommentar
0x0c   buss:=GRx, AR:=buss, R:=0, S:=0,      ; Läs GRx till AR
       uPC:=uPC+1                            ;
0x0d   buss:=PM, AR:=AR-buss, uPC:=uPC+1     ; Beräkna Grx-PM, påverka flaggor
0x0e   buss:=GRx, AR:=buss,                  ; Kopiera aktuellt GRx-värde till AR
       uPC:=0x11 om N=1 annars uPC:=uPC+1    ; Testa N=0 (avgör om negativt)
0x0f   uPC:=0x12 om O=1 annars uPC:=uPC+1    ; N=0, O=1 => positivt (GRx störst)
0x10   buss:=AR, GRX:=buss, R:=1, uPC:=0     ; N=0,O=0, GRx störst, sätt GR3=GRx,
                                             ; (kopia GRx i AR) nästa instruktion
0x11   uPC:=0x10 om O=1 annars uPC:=uPC+1    ; N=1, O=1 => positivt, GRx störst
0x12   buss:=PM, GRx:=buss, R:=1, uPC:=0     ; Tilldela GR3=GRx, nästa instr.
```

b) K2:s innehåll. 2 bit M-fält => 4 adresser, 128 olika adresser i mikrokodsminnet => 7 bitars data (varje bit i ett register visas i figur 1 som en punkt i registret).

Adress	Data
00	0000011
01	0000101
10	0000111
11	0001001

2. a) En VLIW processor startar flera delinstruktioner per klockcykel i form av stora komplexa instruktioner. Dessa definieras vid kompilering, så det är alltid samma kombination av delinstruktioner som utförs parallellt. En superskalär processor utför enklare instruktioner, men kan starta flera sådana per klockcykel. Den superskalära processorn avgör själv under körningen vilka instruktioner som kan startas parallellt varje klockcykel. (parallellt, flera små instruktioner, fast/förvalt)

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

c) Datakonflikter (oftast RAW-konflikt)

d) Ja, ifall man har en pipelinad process. Då kommer nästa instruktion (första instruktionen i subrutinen) att utföra inte ligga på minnesplatsen efter subrutinanropsinstruktionen, utan på ett annat ställe. Den linjära adress-sekvensen som beskriver adresserna som instruktionerna hämtas från i minnet bryts och därför blir det en styrkonflikt. I en pipelinad processor förutsätts en linjär sekvens för att nästa instruktion ska kunna hämtas utan att nuvarande instruktion beräknats klart, i detta fall att instruktionen efter subrutinanropet startas innan PC ändras.

e) RISC=Reduced Instruction Set Computer, CISC=Complex Instruction Set Computer.

TSEA28 Datorteknik Y, lösningar till tentamen 22-05-31, modifierad 23-06-14

3. Subrutinen startar med att sätta resultatregister r3=minsta möjliga värde (0x80000000). I en loop gå igenom tabellen. Om ett värde med rätt tag hittas så teckenförlängs detta och jämförs med resultatregistret. När loop är slut anropa Subrutin1 och sätt sedan r0 till största värdet.

Antag att det alltid finns minst 1 värde i tabellen, och att det alltid finns minst 1 värde med rätt markering.

```
Subrutin:   mov   r3,#0x80000000    ; Sätt hittills största värde till minsta
loop        ldrsh r4,[r0],#2        ; hämta 16-bit tvåkomplementvärde
            and   r5,r4,#0xf        ; ta fram bit 3-0
            cmp   r5,r2             ; hittat rätt markering?
            bne   checkdone         ; fel marking, ta nästa varv i loop
            asr   r4,#4             ; skifta 4 bit höger aritmetiskt
            cmp   r4,r3             ; Är nya värdet mindre?
            ble   checkdone         ; nej, ta nästa varv i loop
            mov   r3,r4             ; ja, nytt största värde
checkdone   subs  r1,#1             ; gjort ytterligare ett varv, fler kvar?
            bne   loop              ; inte klar, ta nästa varv
doneloop    mov   r1,r3             ; Subrutin1 vill ha största i r1
            push  {lr}              ; Är i subrutin, måste spara LR
            bl    Subrutin1         ; Förstör LR, r0, r1, r2
            pop   {lr}              ; återställ lr
            mov   r0,r3             ; största värdet till r0
            bx    lr                ; klar, hoppa ur subrutin.
```

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

```
IOPORT          .equ 0x40001000

avbrott:  cpsid   i               ; se till att inga andra avbrott kan ske
          push    {lr,r5,r7}      ; 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 bara en gång (krav!)

          and     r1,r0,#0x0000003f ; 6 minst signfikanta bitar, sensor 1
          and     r2,r0,#0x003f0000 ; bit 21-16, sensor 2
          lsr     r2,#16            ; placera sensor 2 i bit 5-0
          cmp     r1,r2             ; jämför sensor1 med sensor2
          ble     done              ; sensor 2 större, klar
          bl      Subrutin2
done:     pop     {lr,r5,r7}        ; återställ register (r0-r2 fixas automatiskt)
          cpsie   i
          bx      lr
```

5. a) $-19_{10} = -(19_{10}) = -(00010011_{2C}) = (11101100+1)_{2C}=11101101_{2C} = 0xed$

b) Skriv om (111010_{2C}-0111_{2C}) till summation med 8 bitars ordlängd: (111010_{2C}-0111_{2C})=(11111010_{2C}+(-00000111_{2C})) = (11111010_{2C}+($11111000+1)_{2C}$)=($11111010+11111001)_{2C}$

```
Carry:  11111

         11111010
        +11111001
         --------
         11110011
```

Svar: 11110011_{2C}

Kontrollräkna gärna: 111010_{2C}= -32+16+8+2=-6, 0111_{2C}=7, svaret är -128 + 64 + 32 + 16 + 2 + 1 = -13 vilket stämmer med -6 -7 = -6 + (-7) = -13.

c) 0x75+0x53=0xC8. Summan blir inte noll => Z=0, Summan blir inte större än 0xFF => C=0, MSB av resultat=1 => N=1, om indata ses som 2-komplement fås ett värde som har MSB=1, trots att indata är positiva => O=1.

6. a) Med 8 byte per cacheline och index som visar att det finns 2^8 rader per väg ger detta 8*2^8=2048 byte per väg i cachen, och det blir då 32768/2048=16 vägar totalt.

b) För byteposition behövs 3 bitar (2^3 = 8). Tag består då av totalt antal – indexbitar - bytepositionsbitar = 32-8-3 = 21 bitar.

c) De adresser som ska läsas är sekvensen nedan, med tag, index och byteposition indikerat

0x1234567: tag=0x2468, index=0xac, byteposition=0x7 : Cachemiss
0x123456d: tag=0x2468, index=0xad, byteposition=0x5 : Cachemiss
0x1234573: tag=0x2468, index=0xae, byteposition=0x3 : Cachemiss
0x1234579: tag=0x2468, index=0xaf, byteposition=0x1 : Cachemiss
0x123457f: tag=0x2468, index=0xaf, byteposition=0x7 : Cacheträff
0x1234585: tag=0x2468, index=0xb0, byteposition=0x5 : Cachemiss

Totalt blir det 5 cachemissar (och 1 cacheträff) när sekvensen läses.