TSEA28 Datorteknik Y, lösningar till tentamen 18-05-29, reviderad 181026

1. a) Första delen av mikrokodsminnet hanterar adresseringsmode. ASR-registret innehåller därför rätt adress till minnet när instruktionen körs igång. Mikrokoden för instruktionen MAX hämtar därför värde i minnet och placerar i AR, och subtraherar sedan GRx. Om C=1 är GRx större och nästa instruktion startas. Annars ska värdet i minnet placeras i GRx. Notera att N och O inte kan användas eftersom indata är positiva heltal, inte 2-komplement.

```
Adress Mikrokod                                Kommentar
   7   buss:=PM, AR:=buss,                     ; PM till AR
       uPC:=uPC+1
   8   AR:=AR-buss, buss:=GRx, R:=0, S:=0,     ; beräkna PM-GRx, sätter C-flaggan
       uPC:=uPC+1
   9   uPC:=0 om C=1                           ; GRx var större, ta nästa instr.
  10   buss:=PM, R:=0, S:=0, GRx:=buss,        ; Kopiera PM till GRx, nästa instr.
       UPC:=0
```

b) 4 bitar opcode, 7 bitar mikrokodsadresser

Adress	Innehåll	Adress	Innehåll
0 0000	0011011	8 1000	-------
1 0001	0001111	9 1001	0010011
2 0010	0010111	10 1010	-------
3 0011	-------	11 1011	-------
4 0100	-------	12 1100	-------
5 0101	-------	13 1101	-------
6 0110	-------	14 1110	-------
7 0111	-------	15 1111	-------

2. a) En CISC-arkitektur är inte pipelinead, utan utför en instruktion i taget. Den får därför inte några styrkonflikter, vilket branch prediction används för att undvika. Det blir därför ingen ökad prestanda för CISC-processorn. En pipelinead RISC-processor kan få styrkonflikter, och därför ger branch prediction en ökad prestanda.

b) DMA = Direct memory access. Detta är en enhet som på egen hand kan göra minnesöverföringar på en buss utan att processorn är direkt inblandad.

c) FLASH = Ett icke-volatilt minne som behåller sitt innehåll även vid spänningsbortfall. Skrivhastighet är mycket lägre än för ett DRAM. DRAM = volatilt minne, dvs det tappar innehållet när spänningen försvinner. Kräver också refresh, dvs alla minnesceller måste kontinuerligt läsas för att uppdatera innehållet. DRAM är dyrare per minnescell jämfört med FLASH. Skrivning i FLASH kräver att större block av data nollställs innan skrivning.

d) En von-Neumann arkitektur har ett delat program och dataminne (sitter på samma buss), medan en Harvard-arkitektur har ett separat minne med databuss för programmet och ett annat minne och databuss för data.

e) En styrkonflikt i en pipelinead processor innebär att processorn inte vet vilken adress nästa instruktion ska finnas på utan behöver vänta på resultatet av t ex en villkorsjämförelse eller en beräkning av en hoppadress innan rätt instruktion hämtas.

TSEA28 Datorteknik Y, lösningar till tentamen 18-05-29, reviderad 181026

3. Subrutinen behöver hämta värde, placera i utvektor, sedan hämta adress till nästa värde, testa om värdet är noll, och om inte hoppa till början.

ARM Cortex-M4:

```
Subrutin:   mov   r3,#0        ; nollställ räknare för antal data
loop:       ldr   r2,[r0],#4   ; Läs data, öka pekare r0 med 4
            str   r2,[r1],#4   ; spara data i utvektor
            add   r3,r3,#1     ; öka räknare med 1
            ldr   r0,[r0]      ; hämta adress till nästa värde
            cmp   r0,#0        ; slut på listan?
            bne   loop         ; nej, ta nästa
            bx    lr
```

68000:

```
Subrutin:   move.l #0,D0       ; nollställ antal kopierade data
loop:       move.l (A0)+,D1    ; Hämta data, peka på adressvärdet
            move.l D1,(A1)+    ; Spara data i utdatavektor, öka pekare
            add.l  #1,D0       ; öka antal kopierade data
            mov.l  (A0),A0     ; hämta adress till nästa data
            cmp.l  #0,A0       ; Test om det är det sista
            bne    loop        ; inte noll, ta nästa
            rts
```

4. Litet fel i uppgiften: udda tal har LSB = 1 (ger inga minuspoäng oberoende om jämn eller udda används). Algoritm: Sätt aktuellt värde = 0. Läs värde, om värde udda så jämför med aktuellt värde och behåll största värdet. Skicka slutligen ut det på utporten. Rutinen är en avbrottsrutin, använda register måste sparas på stacken. För arm är r0-r3, r12,lr,pc samt psr-registren redan sparade på stacken. För 68000 är endast statusregister och PC sparat på stacken.

Arm Cortex-M4:

```
inputadr  .field    0x40001000,32

avbrott:  mov     r0,#0        ; antag alla jämna först
          ldr     r1,inputadr  ; peka på indataadressen
          mov     r2,#4        ; totalt 4 indata att läsa
loop:     ldrb    r3,[r1],#16  ; läs in, peka på nästa indata
          ands    r12,r3,#1    ; testa om udda
          beq     jamn         ; nej, jämn så testa om alla lästs
          cmp     r0,r3        ; jämför r0-r3
          blo     jamn         ; r3 < r0, hoppa test om alla lästs
          mov     r0,r3        ; r3 var större, uppdatera r0
jamn:     subs    r2,r2,#1     ; minska räknare med 1, se om vi är klara
          bne     loop:        ; inte klar, ta nästa varv
          strb    r0,[r1]      ; r1 pekar automatiskt på utdataport ($40001040)
          bx      lr
```

TSEA28 Datorteknik Y, lösningar till tentamen 18-05-29, reviderad 181026

68000:

```
Avbrott: move.l   D0,-(A7)          ; push D0 på stack
         move.l   D1,-(A7)          ; push D1 på stack
         move.b #0,D0               ; start med värde 0
         move.b   $40001000,D1
         and.b    #1,D1             ; testa bit 0
         beq      jamn1             ; LSB=0, jämnt data, ta nästa
         move.b   $40001000,D0      ; Alltid större än 0
jamn1:   move.b   $40001010,D1
         and.b    #1,D1
         beq      jamn2             ; LSB=0, jämnt data, ta nästa
         cmp.b    $40001010,D0      ; vilket är störst? (D0-indata beräknas)
         bcc      jamn2             ; BLT används för 2-komplement
         move.b   $40001010,D0      ; nytt största udda tal.
jamn2:   move.b   $40001020,D1
         and.b    #1,D1
         beq      jamn3             ; LSB=0, jämnt data, ta nästa
         cmp.b    $40001020,D0      ; vilket är störst? (D0-indata beräknas)
         bcc      jamn3             ; BLT används för 2-komplement
         move.b   $40001020,D0      ; nytt största udda tal.
jamn3:   move.b   $40001030,D1
         and.b    #1,D1
         beq      jamn4             ; LSB=0, jämnt data, ta nästa
         cmp.b    $40001030,D0      ; vilket är störst? (D0-indata beräknas)
         bcc      jamn4             ; BLT används för 2-komplement
         move.b   $40001030,D0      ; nytt största udda tal
jamn4:   move.b   D0,$40001040      ; skriv resultatet
         move.l   (A7)+,D1          ; återställ registren
         move.l   (A7)+,D0
         rte                        ; klar med avbrottet
```

5. a) 101_{2C} = 1111101_{2C}, dvs teckenbiten kopieras 4 gånger för att skapa ett 7-bitars 2-komplementstal av ett 3-bitars 2-komplementstal.

b) 7-bitarstal => 0x1b = 0011011, 0x2d = 0101101. Byt tecken på 2:a argumentet: -0101101 = 1010010+1=1010011. Skriv om som addition 0011011 – 0101101 = 0011011 + (-0101101) = 0011011 + 1010011. Beräkna summan:

```
Carry:     1  11
          0011011
         +1010011
         --------
          1101110
```

Kontroll: 0x1b=27, 0x2d=45 => differens=27-45=-18. -18 i 2-komplementsform: -18 =-(0010010)=$(1101101+1)_{2C}$=1101110_{2C}, vilket stämmer med svaret ovan.

c) 1101*0101

```
       1101
      *0101
     -------
       1101
      0000
     1101
   +0000
  ---------
   01000001
```

Kontroll: Positiva heltal => 1101=13, 0101=5, 13*5=65 = 64+1=1000001 vilket stämmer.

6. a) 16384/64= 256 cachelines

b) 4-vägs cache => 256/4= 64 cacheline per väg => 6 bitar index. 64 byte/cacheline => 6 bitar för byteposition i cacheline. Återstår då 32-6-6=20 bitar som blir tag.

c) 32 bitar = 4 byte per instruktion. 64 byte per cacheline => 64/4=16 instruktioner per cacheline.