Maskinorienterad programmering 2010-08-23

1 a) DAA 18 07 (1p)

b) 18 03 12 34 56 78
MOVW #$1234, $5678 (2p)

c) EXG D,X
XGDX (1p)

d) DEC -$100,Y 63 E9 00 (2p)

e) LBRN $1000 Nästa instruktion på adr $108C$_{16}$
18 21 FF 74

```
  15 15 16
  1000
- 108C
  FF74
```

(2p)

f) H-flaggan används av instruktionen DAA för att den visar när det har uppstått en minnessiffra (carry) från den högra nibblen till den vänstra vid en addition. Detta behövs vid addition av NBCD-tal för att vissa när den högra nibblen är en otillåten NBCD-siffra, trots att den ser korrekt ut 0-3. (2p)

g) Tal med tecken $20 ≙ 32_{10} $32-W \leq 0$; $32 \leq W$
[-128, +127] $32 \leq W \leq 127$ (2p)

h) Tal med tecken eftersom N-flaggan används
$78 ≙ 120_{10} ; $120-W < 0$; $120 < W$; $120 < W \leq 127$
Detta gäller om overflow ej inträffar (V=0)
Om overflow inträffar (V=1), dvs $120-W > 127$, utförs också hoppet eftersom N-flaggan då blir fel.
$120-W > 127$; $-7 > W$; $-128 \leq W < -7$ som tal utan tecken motsvaras detta $256-128 \leq W < 256-7$; $128 \leq W < 249$
Alltså $120 < W \leq 127$ } dvs $120 < W < 249$ (3p)
$128 \leq W < 249$ }

i) Asynkron: Korta avstånd, liten datamängd, låg datahastighet
Synkron: Stor datamängd, längre avstånd, hög datahastighet (2p)

j) Man använder "stuffbitar" dvs en inverterad bit efter ett antal lika bitar. (1p)

k) $-378{,}625 ≙ -101111010.101 = -1.01111010101 \cdot 2^8 ≙$
1 10000111 01111010101 0...0 = C3BD5000$_{16}$ (2p)
– 127+8 23 st

(1) forts L) $|N| = mantissa \cdot 2^{exp}$; $|N|_{max} = mantissa_{max} \cdot 2^{exp_{max}}$

$|N|_{min} = mantissa_{min} \cdot 2^{exp_{min}}$

$mantissa_{max} = 1.111\ldots \approx 2$

$mantissa_{min} = 1.000\ldots = 1$

$exp_{max} = 2^{n-1} - 1 = 2^{6-1} - 1 = 31$

$exp_{min} = -2^{n-1} = -2^{6-1} = -32$

$|N|_{max} \approx 2 \cdot 2^{31} = 2^{32} = 4 \cdot 2^{30} \approx 4(\underbrace{2^{10}}_{\approx 10^3})^3 \approx \underline{4 \cdot 10^9}$

$|N|_{min} = 1 \cdot 2^{-32} = 0{,}25 \cdot (2^{-10})^3 \approx \underline{0{,}25 \cdot 10^{-9}}$ (2p)

---

(2) a)

```
SWAP  PSHD
      PSHX
LOOP  LDAA  ,X     Hämta byte från sträng
      BEQ   SWEX   Nollbyte?
      TFR   A,B    Nej, swap byte
      LSRA         Hög nibble till låg
      LSRA
      LSRA
      LSRA
      ASLB         Låg nibble till hög
      ASLB
      ASLB
      ASLB
      ABA          Sätt ihop nibbles
      STAA  1,X+
      BRA   LOOP   Nästa byte
SWEX  PULX         Återställ reg
      PULD
      RTS
```

(5p)

---

b)

```
SUBRUT  LDY   ,SP    Hämta "återhoppsadr" från stack
        LEAY  5,Y    Justera adressen till efter inparametrar
        STY   ,SP    Uppdatera till korrekt återhoppsadress
        LDAA  -5,Y   Inpar1
        LDX   -4,Y   Inpar2
        LDY   -2,Y   Inpar3
        :
```

(4p)

③ Reserverad area: 0,5k ≙ 2^9

Start 0000H = 0000 0000 0 0000 0000 (15 … 9 8 … 0)

0000 000 | 1 1111 1111 | = 01FFH (slut)

I/O-port:

0200H = 0000 0010 0000 0000

RWM-modul: 4k ≙ 2^{12}

Start F000H = 1111 | 0000 0000 0000 | = FFFFH (slut)

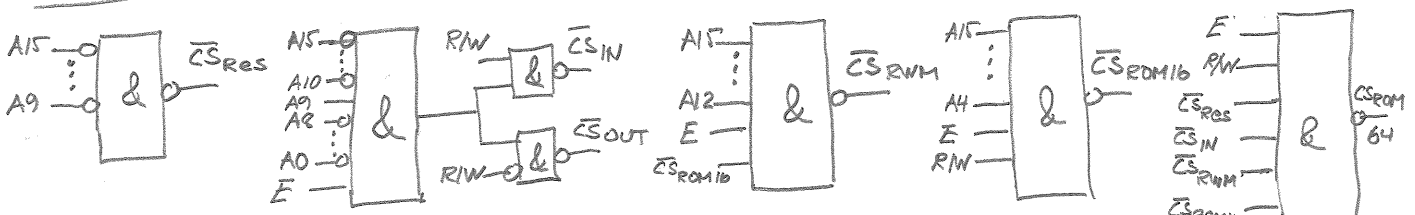
1111 | 1111 1111 1111 |

ROM_{16} 16 = 2^4

Start FFF0H = 1111 1111 1111 | 0000 | = FFFFH (slut)

1111 1111 1111 1111 | 1111 |

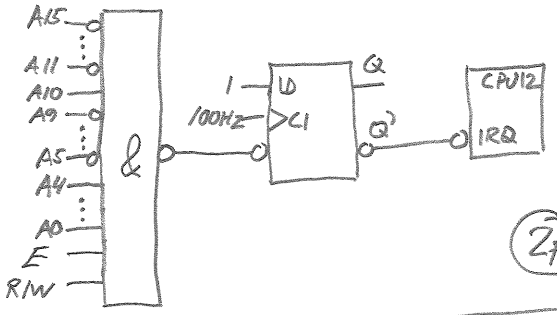
ROM_{64k} (Hela adressrummet)



8p

RWM: F000H–FFEFH
ROM16: FFF0H–FFFFH
ROM64k: 0201H–EFFFH

④ a) 41FH = 0000 0100 0001 1111 (A15 … A9 … A4 … 0)



2p

b)

```
CT5      EQU   $1880      Räknare 20Hz
CT6000   EQU   $1881      Räknare minut
CLEAR    EQU   $41F       Reset avbr vippa

IRQRUT   TST   CLEAR      Nolla IRQ-vippa
         DEC   CT5         Räkn 20Hz
         BNE   NEXT        Lösa PORTI?
                           Ja. Återställ räkn
         MOVB  #5,CT5      Lås port
         MOVB  PORTI,SEI
NEXT     LDX   CT6000      Minska minuträkn
         DEX
         STX   CT6000
         BNE   IRQEX       Minut?
         MOVW  #6000,CT6000  Ja. Återställ räkn
         MOVB  PORT2,SE2     Lås port
IRQEX    RTI
```

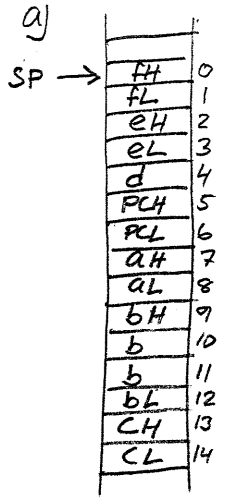
4p

c)

```
IRQVECT  EQU   $FFF2
   :
MOVW  #IRQRUT,IRQVECT   Init
TST   CLEAR             Reset avbr vippa
MOVB  #5,CNT5           Init räkn 20Hz
MOVW  #6000,CT6000      Init räkn minut
CLI                     Aktiverar avbrott (IRQ)
   :
```

3p

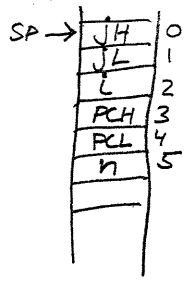
(5) a)



(2p)

b)

```
func6   LEAS   -3,SP
        LDD    #0
        STD    ,SP      j=0
        CLR    2,SP     i=0
LOOP    LDAB   2,SP     i
        CMPB   5,SP     i<n ?
        BGE    READY    i>=n
        LDAB   5,SP     n
        SEX    B,D      char->int
        ADDD   ,SP      n+j
        STD    ,SP      j=n+j
        INC    2,SP     i=i+1
        BRA    LOOP
READY   LDD    ,SP      return j
        LEAS   3,SP
        RTS
```



(5p)

(6) a) **Svar:** Minnesmodulerna har begränsad kapacitet och för lång accesstid. För program och data gäller "locality of reference in time and space", vilket innebär att bara en mycket liten del av adressrummet utnyttjas under ett godtyckligt valt kort tidsintervall och att sannolikheten är stor att processorn kommer att använda adresser i närheten av den adress den redan använder. Det innebär att man skulle kunna klara sig med ett betydligt mindre minne under förutsättning att det innehåller den data processorn behöver.

Man kan därför sänka accesstiden genom att sätta in ett litet men mycket snabbt minne (cache) mellan processorn och "main memory". Processorn läser och skriver bara i cacheminnet och när data saknas där hämtas saknade data och närliggande data till cache från primärminnet. Sannolikheten är sedan stor att de följande accesserna görs i cacheminnet.

Primärminnesstorleken kan man "öka" (virtuellt minne) genom att låta hela adressrummet finnas på en hårddisk, medan bara det som används för tillfället finns i det verkliga mindre primärminnet (det fysiska minnet). Enligt samma resonemang som för cacheminnet ovan är sannolikheten stor att data som processorn behöver verkligen finns i det fysiska minnet.

(3p)

b) 1. Flash-minnen har begränsad "livslängd" för skrivningar i minnet.

2. Radering av skrivna data är komplicerad. Den görs blockvis.

(2p)