Lösningsförslag tenta 2013-01-19 **(v1 med reservation för eventuella fel!)**

1. a) ANDCC #$FD ↔ CLV **(2p)**

b) 0C EA 32 10 0F → BSET $3210,Y,#$0F **(2p)**

c) SUBA -$B6,X → A0 E1 4A ($0B6_{2k}$ = F4A, dvs 14A med 9 bitar.) **(2p)**

d) Bägge utför heltalsdivision med 2, men ASR gör det för tal med tecken medan LSR gör det för tal utan tecken. **(2p)**

e) 2457_{16}: IBEQ A,$2400 → 04 90 A6 ($2400_{16} - 245A_{16} = FFA6_{16}$, dvs $1A6_{16}$ med 9 bitar.) **(2p)**

f) BGT avser tal med tecken. Det innebär att vi skall tolka data som tal i intervallet [–128,127]. Talet $A0_{16} = 10 \cdot 16 = 160_{10}$ och tolkas då som talet – (256 – 160) = – 96.

NEGA bildar talet – W. CMPA utför subtraktionen: –W – (– 96) och BGT testar hoppvillkoret: –W – (– 96) > 0 vilket ger 96 > W. Med hänsyn till talområdet för W får vi: $-128 \leq W < 96$.

Eftersom 2k-representation används delar vi upp talområdet i en positiv och en negativ del.
$0 \leq W < 96$ och $-128 \leq W \leq -1$ vilket ger $256 - 128 \leq W \leq 256 - 1$ dvs. $128 \leq W \leq 255$

Hoppet utförs om: $0 \leq W < 96$ eller $128 \leq W \leq 255$ **(3p)**

g) BPL testar N-flaggan och avser därför tal med tecken (utan gardering för overflow). Talområde: [-128, 127]. Talet $10_{16} = 16_{10}$.

CMPB utför subtraktionen: 16 – W och BPL testar hoppvillkoret: $16 - W \geq 0$ vilket ger $16 \geq W$. Med hänsyn till talområdet för W får vi: $-128 \leq W \leq 16$.

Om overflow inträffar får dock N-flaggan fel värde och hoppet kommer då inte att utföras.

Här inträffar overflow om $16 - W \geq 128$, dvs. $16 - 128 \geq W$, eller $-112 \geq W$.

Vi får då hoppvillkoret: $-111 \leq W \leq 16$

Eftersom vi använder 2k-representation delar vi upp talintervallet i en positiv och en negativ del:
$0 \leq W \leq 16$ och $-111 \leq W \leq -1$, vilket motsvarar $256 - 111 \leq W \leq 256 - 1$ eller $145 \leq W \leq 255$
Hoppet utförs om: $0 \leq W \leq 16$ eller $145 \leq W \leq 255$ **(4p)**

h) Sk. stuffbitar med inverterat värde skjuts in efter en sekvens med ett antal (5) lika bitar och medför att det finns flanker tillräckligt ofta i bitströmmen för att noderna skall behålla synkronismen. **(2p)**

i) Format: s/c/f Karakteristikans (c) högsta värde ($255 = 1111\ 1111_2$) används för att markera ∞. Minustecken ger s =1. f = 0. $N_\infty = 1/111\ 1111\ 1/000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = FF800000_{16}$ **(1p)**

2. a)

```
NIBADD   PSHA              Spara reg på stack
         PSHY
         CLR     OFLCNT    Nollställ räknare för overflow
         STAB    DATCNT    Kopia av dataordsräknare
         BEQ     NIBEX     Uthopp om dataordsräknare = 0
NIBLOOP  LDAA    ,Y        Hämta dataord
         TFR     A,B       Skapa kopia
         ANDA    #$0F      Maska fram låg nibble
         STAA    COPY      Kopia av låg nibble
         TFR     B,A       Återställ dataord
         LSRB               Högerjustera hög nibble
         LSRB
         LSRB
         LSRB
         ORAB    #$F0      Ettställ alla bitar i hög nibble
         ADDB    COPY      Addera nibblar
         BCC     NOFLW     Overflow? Nej
         INC     OFLCNT    Ja, öka overflowräknare
NOFLW    ANDB    #$0F      Maska fram låg nibble efter addition
         ANDA    #$F0      Maska fram hög nibble i ursprungliga dataordet
         ABA               Kombinera hög och låg nibble
         STAA    1,Y+      Placera nya dataordet i tabell och öka pekare
         DEC     DATCNT    Minska dataordsräknare
         BNE     NIBLOOP   Behandla nästa dataord om ej färdigt
NIBEX    LDAB    OFLCNT    Färdigt. Hämta overflowräknare för retur
         PULY               Återställ reg från stack
         PULA
         RTS

OFLCNT   RMB     1
DATCNT   RMB     1
COPY     RMB     1

         (ORG    $2800     Tabell för test
         FCB  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
         FCB  $10,$11,$12,$13,$14,$15,$16,$17,$18,$19,$1A,$1B,$1C,$1D,$1E,$1F
         FCB  $20,$21,$22,$23,$24,$25,$26,$27,$28,$29,$2A,$2B,$2C,$2D,$2E,$2F
         FCB  $30,$31,$32,$33,$34,$35,$36,$37,$38,$39,$3A,$3B,$3C,$3D,$3E,$3F
         FCB  $40,$41,$42,$43,$44,$45,$46,$47,$48,$49,$4A)
```

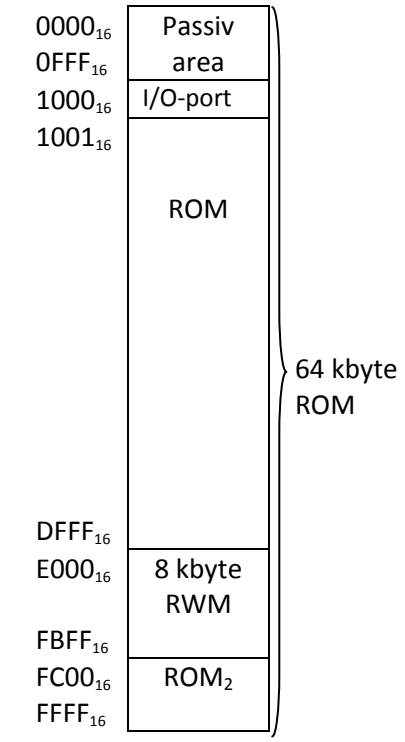
(8p)

b)

```
START    LDS     #$3D00    Initiering av stack
         LDAB    #75       Antal dataord i tabell
         LDY     #$2800    Pekare till tabellstart
         JSR     NIBADD
```

(2p)

3.



Passiv area, $4k = 2^{12}$

A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: 0000_{16} =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: $0FFF_{16}$ =	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	CS															

RWM, $8k = 2^{13}$

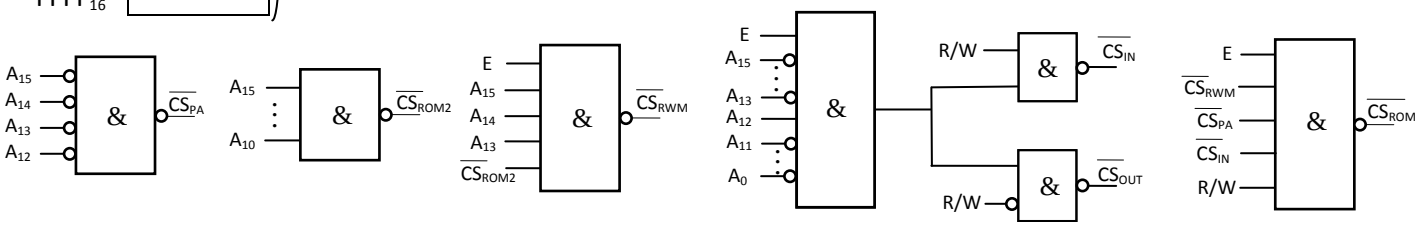
A	15	14	13	12	11	10	9	8	7	12 st			3	2	1	0
Start: $E000_{16}$ =	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: $FFFF_{16}$ =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	CS															

ROM_2, $1k = 2^{10}$

A	15	14	13	12	11	10	9	8	13 st			4	3	2	1	0
Start: $FC00_{16}$ =	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Slut: $FFFF_{16}$ =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	CS						10 st									

I/O-port:

A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr: 1000_{16} =	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
							CS									



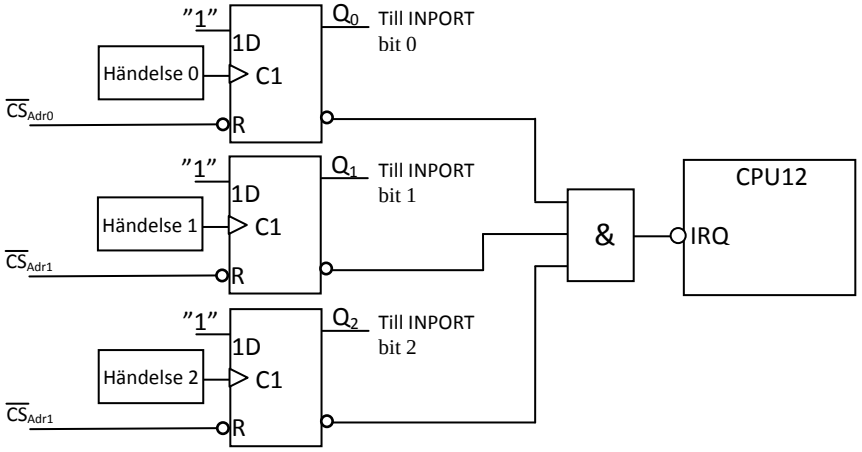
(8p)

4. Antag att varje händelse genererar en puls.

a) Huvudprogrammet måste initiera avbrottssystemet, dvs. se till att hopp görs till avbrottsrutinens adress vid avbrottsförfrågan (IRQ) genom att skriva in adressen till avbrottsrutinen IRQ-vektorn. Eventuella variabler som kan behövas av servicerutinerna för de tre olika avbrottskällorna måste också initieras. Avbrottsvipporna skall nollställas för att falska avbrott inte skall uppträda vid start. Som sista åtgärd skall avbrottsystemet aktiveras genom att I-flaggan i CC-registret nollställs. **(3p)**

b) Avbrottsrutinen måste först identifiera avbrottskällan genom att läsa av vilket Q_i-värde enligt figuren nedan som är aktivt och sedan hoppa till aktuell servicerutin. I servicerutinen skall de uppgifter utföras som är förknippade med det aktuella avbrottet. Före återhopp till avbrottsrutinen och senare återhopp till det avbrutna programmet med RTI måste också avbrottsvippan nollställas med en puls på vippans RESET-ingång genom att en läsning eller skrivning görs på aktuell adress enligt figuren nedan. **(3p)**

c)



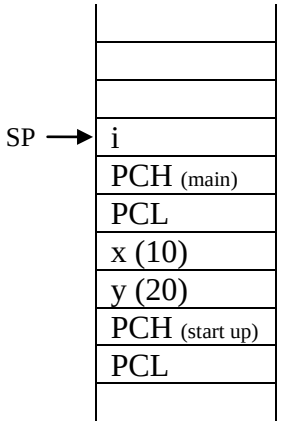
(3p)

5. a)

```
main:  LDAB   #$14     y-värde
       PSHB
       LDAB   #$0A     x-värde
       PSHB
       JSR    func
       LEAS   2,SP     Justera stack
       RTS

func:  LEAS   -1,SP    Plats för i-värde
for:   CLR    0,SP     i=0
for1:  LDAB   0,SP     Hämta i
       CMPB   #4       Övre gräns
       BGT    out      for-sats färdig
       LDAB   3,SP     for-kropp. Hämta x
       ADDB   4,SP     Addera med y
       STAB   4,SP     Uppdatera y
       ADDB   z        Addera z till y
       STAB   z        Uppdatera z
       INC    0,SP     i=i+1
       BRA    for1     Nästa varv
out:   LDAB   z        Returnera z
       LEAS   1,SP
       RTS

z:     FCB    $32      z är en global variabel
```



(7p)

b)

i	***x***	***y***	***z***
0	10	30	80
1	10	40	80+40 = 120
2	10	50	120+50 = 170 (-86) overflow
3	10	60	179+60 = 230 (-26)
4	10	70	230+70 = 300 mod 256 = 44

Returvärde: *z = 44* **(3p)**

6. a) ”Associative mapping” och ”Set-associative mapping”. **(2p)**

b) ”Direct mapping”:

De låga adressbitarna används av både cache och ”main memory”. Övriga (höga) bitar från ”main memory”-adressen kallas ”tag” och lagras tillsammans med dataorden i cache. Vid läsning av data (från cache) jämförs de höga bitarna i adressen med motsvarande bitar i ”tag”. Om de är lika är det det korrekta ordet som har lästs (”hit”), men om de är olika måste rätt ord hämtas från ”main memory” (”miss”).

Nackdel:
Denna princip innebär alltså att alla dataord som finns på adresser med samma låga adressbitar placeras på samma ställe i cache och sannolikheten för ”adresskrock” därför är relativt stor.

Fördel:
Metoden är enkel att implementera och ger snabb access. **(2p)**