Lösningsförslag tenta 2012-08-27 **(v1 med reservation för eventuella fel!)**

1. a) EXG SP,Y → B7 F6 eller B7 E7 **(1p)**

b) 18 07 → DAA **(1p)**

c) 0F F2 42 3A 07 60 → 1800_{16}: BRCLR $423A,SP,#$07,$1866 **(2p)**

d) LSLA = ASLA **(1p)**

e) COM -$90,X → 61 E1 70 **(2p)**

f) 1500_{16}: LBNE $8000 Nästa OP-kod finns på adr 1504_{16}.
Offset (qq rr) = Till – Från = $8000_{16} - 1504_{16} = 6AFC_{16}$. Maskinkod: 18 26 6A FC **(2p)**

g) $50 = 80. BHI (>) avser hopp för tal utan tecken [0, 255]. Villkor: 80 – W > 0; 80 > W
$0 \leq W < 80$ **(2p)**

h) $50 = 80. BPL (≥ 0) avser hopp för tal med tecken [–128, 127]. Hopp om teckenflaggan N = 0.

Villkor: $W + 80 \geq 0$; $W \geq -80$ eller $-80 \leq W \leq 127$ när hänsyn tas till talområdet.

Om overflow inträffar får dock N-flaggan fel värde.

Här inträffar overflow vid additionen om $W + 80 \geq 128$, dvs. $W \geq 128 - 80 = 48$.

Hoppvillkoret blir då: $-80 \leq W < 48$.

Eftersom vi använder 2k-representation behandlar vi den negativa delen separat:

$-80 \leq W \leq -1$. Det verkliga intervallet blir då: $256 - 80 \leq W \leq 256 - 1$ eller $176 \leq W \leq 255$

Hoppet kommer att utföras i intervallen: $0 \leq W < 48$ och $176 \leq W \leq 255$. **(3p)**

i) Principen kallas något slarvigt för ”bankat minne”. Man avdelar en del av adressutrymmet där man kan nå en av flera olika minnesbankar. Den extra adressdel som behövs för att välja minnesbank kan finnas i speciella instruktioner som kan utnyttja minnesbankarna. Man kan också komplettera adressbussens bitar med extrabitar via en utport. **(3p)**

j) Vid synkron överföring använder sändare och mottagare samma klocksignal medan asynkron överföring innebär att sändare och mottagare använder olika klocksignaler. **(1p)**

k) Frame = ram. För att mottagaren skall upptäcka att sändaren börjar sända ett tecken eller motsvarande så börjar sändningen med en startsignal (oftast en startbit) och avslutas med en stopsignal (oftast en eller flera stoppbitar med inverterat värde av startbiten). **(2p)**

l) Meddelandeordning: 6343_{16}, $635A_{16}$, $636B_{16}$, och $63FF_{16}$. Noderna tävlar om bussen genom att sända meddelandenas identifierare (arbitrering). Varje nod känner av om dess sända bit motsvarar bussvärdet och slutar sända vid avvikelse.

```
635A:   1 1 0 0 0 1 1 0 1 0 1 - - - -
636B:   1 1 0 0 0 1 1 0 1 1 - - - - -
6343:   1 1 0 0 0 1 1 0 1 0 0 0 0 1 1 Detta sänds först.
63FF:   1 1 0 0 0 1 1 1 - - - - - - -
```
(3p)

m) $N_{max} = 1.111...1 \cdot 2^{127} \approx 2 \cdot 2^{127} = 2^{128} = 0{,}25 \cdot 2^{130} = 0{,}25 \cdot (2^{10})^{13} \approx 0{,}25 \cdot (10^3)^{13} = 0{,}25 \cdot 10^{39} = \underline{2{,}5 \cdot 10^{38}}$

$N_{min} = 1.000...0 \cdot 2^{-126} = 16 \cdot 2^{-130} \approx 16 \cdot (10^{-3})^{13} = 16 \cdot 10^{-39} = \underline{1{,}6 \cdot 10^{-38}}$ **(3p)**

2. a)

```
*       Subrutin EPTST
EPTST   PSHA            Spara reg på stack
LOOP    LDAA  ,X        Hämta från  textsträng
        BEQ   EPEX1     0? Ja, hoppa ut med C = 0
        JSR   ACNT      Räkna antal ettor
        ANDA  #1        Maska fram bit 0
        SEC
        BNE   EPEX2     Udda paritet, hoppa ut med C = 1
        INX             Öka pekare
        BRA   LOOP      Nästa byte
EPEX1   CLC             Uppdatera D-reg
EPEX2   PULA            Återställ reg
        RTS
```

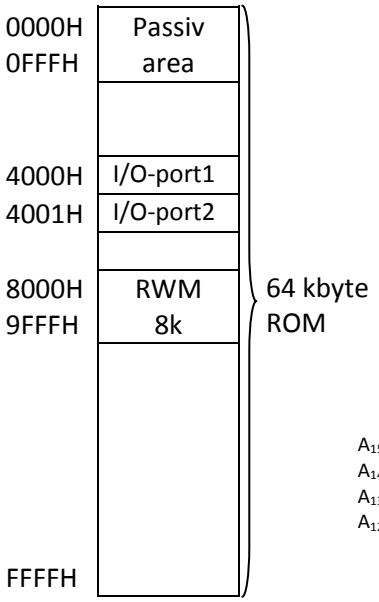
(7p)

b)

```
*       Huvudprogram
        ORG   $1000     Startadress
START   LDS   #$2000    BOS
        LDX   #TEXT     Pekare till textsträng
        JSR   EPTST     Undersök textsträng
        LBRA  $1400
*       Exempelsträng
TEXT    FCS   ”Denna text skall undersökas beträffande paritet!”
        FCB   0         Nollterminering
```

(3p)

3.



Passiv area, 4k = 2^{12}

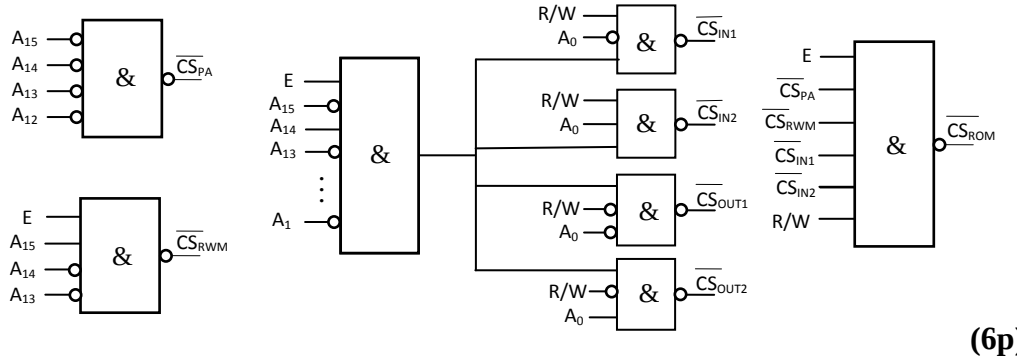
	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: 0000H	=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: 0FFFH	=	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
		CS				12 st											

RWM, 8k = 2^{13}

	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: 8000H	=	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: 9FFFH	=	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
		CS			13 st												

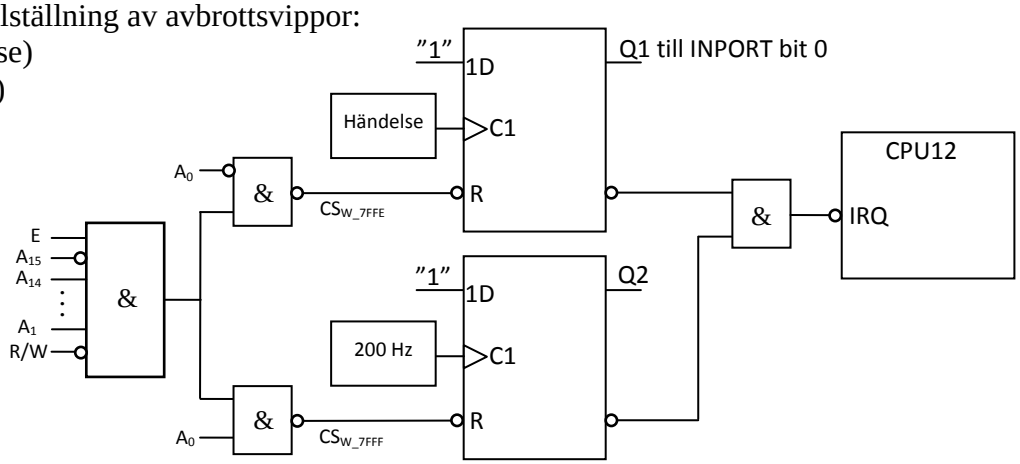
I/O-portar:

	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr: 4000H	=	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4001H	=	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
									CS								



(6p)

4. a) Adresser för nollställning av avbrottsvippor:
$7FFE_{16}$ (Händelse)
$7FFF_{16}$ (200 Hz)



(2p)

b) Program 0

```
BOS_0     EQU   $7000
BOS_1     EQU   $F000

IRQVECT   EQU   $FFF2
IRQCLR1   EQU   $7FFE
IRQCLR2   EQU   $7FFF

EVENT     EQU   $7E00
SPSAVE    EQU   $7E02

          ORG   $1000
Prog_0    MOVW  #Prog_1,BOS_1-2     Skapa stack för Prog_1
          MOVB  #$C0,BOS_1-9        I = 0, S = X = 1 övriga = 0

          MOVW  #BOS_1-9,SPSAVE     Sparad stackpekare för program 1

          MOVW  #IRQR,IRQVECT       Initiera avbrottsvektor
          CLR   IRQCLR1             Nollställ båda avbrottsvipporna
          CLR   IRQCLR2

          MOVW  #0,EVENT            Nollställ EVENT-räknare

          LDS   #BOS_0              Stackpekare för Prog_A
          CLI                       Aktivera IRQ-avbrott

LOOP_0    BRA   LOOP_0
```

(4p)

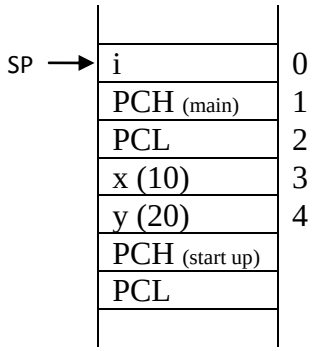
c) Avbrottsrutin för händelseräkning och processbyte

```
IRQR    LDAA  INPORT    Vilket avbrott?
        ANDA  #1        Bit0 = 0?
        BEQ   SWITCH    Ja,processbyte

        LDX   EVENT     Händelseavbrott. Öka räknare
        INX
        STX   EVENT     Uppdatera räknare

        CLR   IRQCLR1   Nollställ händelsevippan
        BRA   IRQEX

SWITCH  TFR   SP,X      Byt stackpekare
        LDS   SPSAVE    Nya stackpekaren hämtas till SP
        STX   SPSAVE    Gamla stackpekaren sparas i SP_SAVE

        CLR   IRQCLR2   Nollställ 200 Hz-vippan

IRQEX   RTI             Återvänd
```

(4p)

5. a)

```
char z = 0x60;                  z       FCB     $60     (96)

void main(){                    main
 func(10,20);                           LDAB    #20
                                        PSHB
                                        LDAB    #10
                                        PSHB
                                        JSR     func
                                        LEAS    2,SP
}                                       RTS


 char func(char x, char y){     func    LEAS    -1,SP
  char i;

  for ( i = 0; i <= 15; i = i + 3)      for     CLR     0,SP

                                floop   LDAB    0,SP
                                        CMPB    #15
                                        BLE     if
                                        BRA     return

      if(i < x)                 if      LDAB    0,SP
                                        CMPB    3,SP
                                        BGE     big
     y = z + y;                         LDAB    4,SP
                                        ADDB    _z
                                        STAB    4,SP

                                big     LDAB    0,SP
                                        ADDB    #3
                                        STAB    0,SP
                                        BRA     floop

 return y;                      return  LDAB    4,SP
                                        LEAS    1,SP
 }                                      RTS
```



(6p)

b) För *i = 0* till 9 adderas 96 4 gånger till y-värdet 20 i ”if-satsen”. 20 + 4*96 = 404. Eftersom y är en char (8 bitar) representeras värdet modulo 2^8 = 256. 404 – 256 = 148. Dessutom är y ett tal med tecken i intervallet [–128, 127]. Det innebär att 148 skall tolkas som det negativa talet –(256 – 148) = –108
Returvärde: *148 som skall tolkas som –108* **(2p)**