Lösningsförslag tenta 2011-08-22 (v1 med reservation för eventuella fel!)

1. a) TFR D,X → B7 45 **(1p)**

b) E4 35 → ANDB 6,X+ **(1p)**

c) 0D E2 12 80 55 → BCLR $1280,X,#$55 **(2p)**

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

e) ADDB %100,Y = ADDB 4,Y → EB 44 **(2p)**

f) 1200_{16}: DBNE A,$1136 → 04 30 rr (lb = 30 eftersom hoppet görs bakåt i minnet)
Nästa OP-kod finns på adr 1203_{16}. rr = Destination – Avstamp = $1136_{16} - 1203_{16} = \text{FF}33_{16}$
DBNE A,$1136 → 04 30 33 **(2p)**

g) $3F = %00111111 Om någon av bitarna 0 - 5 i W har värdet 1, så görs hoppet.
Hoppet görs alltså för alla W-värden [0, 255] utom för W = $00_2 \cdot 2^6$, $01_2 \cdot 2^6$, $10_2 \cdot 2^6$ eller $11_2 \cdot 2^6$, dvs. alla W-värden [0, 255] utom för W = 0, 64, 128 eller 192. **(2p)**

h) Teckenflaggan N avgör om hoppet skall göras. Det innebär att vi skall tolka data som tal med tecken, med talområdet [-128, 127]. Talet $3F_{16} = 3 \cdot 16 + 15 = 63_{10}$.

COMB bildar talet W_{1k}. Eftersom $W_{2k} = W_{1k} + 1$ tolkas som –W så tolkas W_{1k} som –W –1.
Additionen som utförs tolkas då som: –W – 1 + 63 = –W + 62

Hoppvillkoret blir: –W + 62 < 0 (dvs. negativt, N = 1) eller W > 62

När hänsyn tas till talområdet blir hoppvillkoret: 62 < W < 128.

Om overflow inträffar får dock N-flaggan fel värde.

Här inträffar overflow vid additionen om –W + 62 > 127, dvs. –127 + 62 > W, eller W < -65.
Undre gränsen för W ger: –128 ≤ W < –65, eftersom vi använder 2k-representation blir det verkliga intervallet 256 – 128 ≤ W < 256 – 65 eller 128 ≤ W < 191
Även i detta fall blir N = 1, eftersom N-värdet är felaktigt, och hoppet utförs.

De två intervallen kan slås samman och blir då: 62 < W < 191 **(4p)**

i) Om data- och klocksignal överförs på separata ledningar så kan ”skevningseffekten” ställa till problem vid för hög datahastighet i förhållande till avståndet. Vid längre avstånd och högre datahastighet måste man kombinera man klock- och datasignal på samma ledning. **(1p)**

j) Databitarna är okodade så det krävs att signalen ändrar värde tillräckligt ofta för att noderna inte skall tappa synkronismen, eftersom de synkroniserar sig på flankerna i datasignalen. Vid ett visst antal lika bitar efter varandra (5 st) så infogas en inverterad bit (stuffbit). **(2p)**

k) $N_{max} = 1.111...1 \cdot 2^{127} \approx 2 \cdot 2^{127} = 2^{128} = 0{,}25 \cdot 2^{130} = 0{,}25 \cdot (2^{10})^{13} \approx 0{,}25 \cdot (10^3)^{13} = 0{,}25 \cdot 10^{39} = 2{,}5 \cdot 10^{38}$
$N_{min} = 1.000...0 \cdot 2^{-126} = 16 \cdot 2^{-130} \approx 16 \cdot (10^{-3})^{13} = 16 \cdot 10^{-39} = 1{,}6 \cdot 10^{-38}$ **(3p)**

l) Det krävs att en liten del av data används mycket frekvent (”*locality of reference*”). **(2p)**

2. a)

```
*        Subrutin ADD10
ADD10    PSHX              Spara reg på stack
         PSHY

         CLRB              Overflowräknare LO
         CLR     HICNT     Overflowräknare HI
LOOP     LDAA    1,X+      Hämta från STRING1
         CMPA    #$FF      Slut?
         BEQ     ADDEX     Ja, hoppa ut

         ADDA    ,Y        Addera NBCD-tal från STRING2
         DAA               Decimaljustera
         STAA    1,Y+      Skriv tillbaka till STRING2
         BCC     LOOP      Ej overflow, nästa byte

         INCB              Räkna overflow LO
         BNE  LOOP         Nästa byte
         INC  HICNT        Räkna overflow HI

         BRA  LOOP         Nästa byte
ADDEX    LDAA    HICNT     Uppdatera D-reg

         PULY              Återställ reg
         PULX
         RTS
HICNT    RMB 1             Overflowräknare HI
```

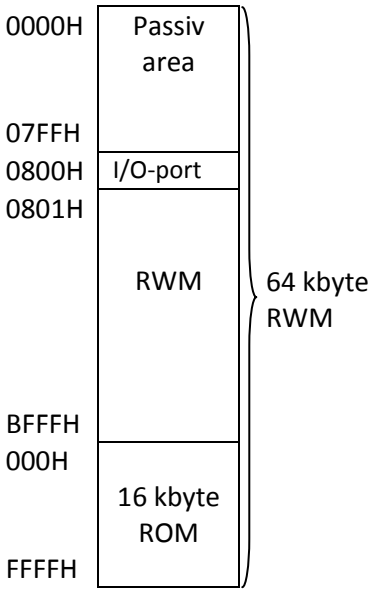
(7p)

b)

```
*        Huvudprogram

         ORG     $1000     Startadress
START    LDS     #$2800    BOS
         LDX     #STRING1  Pekare till sträng1
         LDY     #STRING2  Pekare till sträng2

         JSR     ADD10     Addera sträng1 till sträng2

         LBRA    $1900
*        Exempelsträngar
STRING1  FCB  $10,$11,$12,$13,$14,$15,$16,$17,$18,$19,$20,$21,$22,$FF
STRING2  FCB  $80,$81,$82,$83,$84,$85,$86,$87,$88,$89,$90,$91,$92,$FF
```

(3p)

3.



Passiv area, 2k = 2^{11}

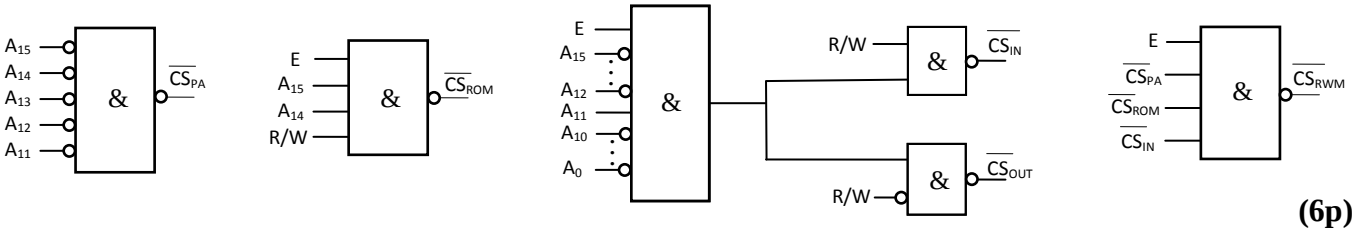
	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: 0000H =		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: 07FFH =		0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
		CS					11 st										

ROM, 16k = 2^{14}

	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: C000H =		1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: FFFFH =		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		CS		14 st													

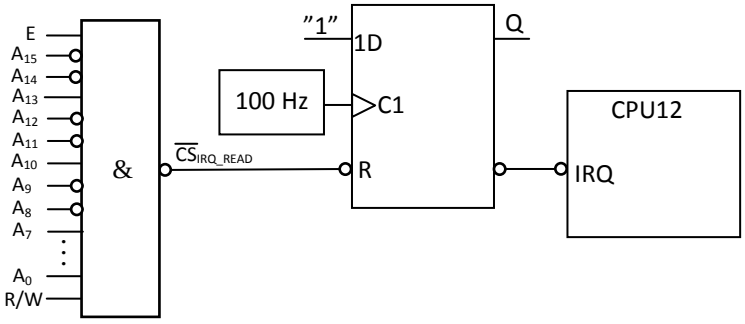
I/O-port:

	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr: 0800H =		0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
		CS															



(6p)

4. a) Adress för nollställning av avbrottsvippa:
$24FF_{16} = 0010\ 0100\ 1111\ 1111_2$



(2p)

b) Avbrottsrutin

```
IRQR    TST     IR_FF           Nollställ avbrottsvippa
        DEC     ACOUNT          25 Hz-räknare
        BNE     CHK_1HZ         Ej 25 Hz

        MOVB    PORTA,VALA
        MOVB    #4,ACOUNT       Ominitiera räknare för 25 Hz
CHK_1HZ DEC     BCOUNT          1 Hz-räknare
        BNE     EX_IR           Ej 1 Hz

        MOVB    PORTB,VALB
        MOVB    #100,BCOUNT     Ominitiera räknare för 1 Hz
EX_IR   RTI
```

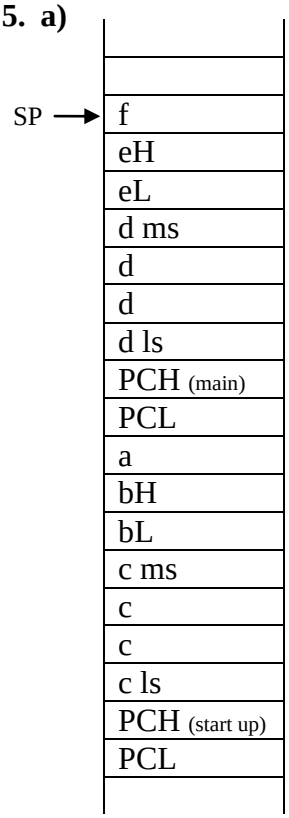
(4p)

c) Avsnitt av huvudprogram

```
ACOUNT  EQU     $1420
BCOUNT  EQU     $1421
IR_FF   EQU     $24FF

START   LDS     #$2800          Sätt SP
        MOVW    #IRQR,$FFF2     Sätt avbrottsvektorn
        TST     IR_FF           Nollställ avbrottsvippa
        MOVB    #4,ACOUNT       Räknare för 25 Hz
        MOVB    #100,BCOUNT     Räknare för 1 Hz

        CLI                     Aktivera avbrottssystem
        .
        .
        .
```

(3p)

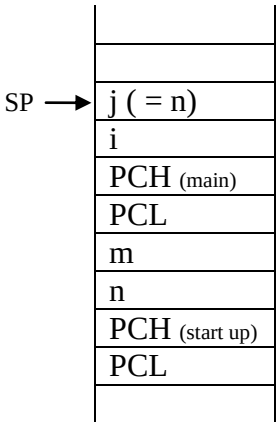
5. a)



(3p)

b)

```
char funcb(char m, char n) {      _funcb:   LEAS    -2,SP
    char i;
    char j = n;                             LDAB    5,SP
                                            STAB    0,SP

    for ( i = 0; i < 2; i++ )               CLR     1,SP
                                  _4:       LDAB    1,SP
                                            CMPB    #2
                                            BLE     _3
                                            BRA     _5

        do j = j + m;             _3:       LDAB    0,SP
                                            ADDB    4,SP
                                            STAB    0,SP

        while (j < 100);                    CMPB    #100
                                            BLT     _3

                                            INC     1,SP
                                            BRA     _4

return j;                         _5:       LDAB    0,SP
}                                           LEAS    2,SP
                                            RTS
```



(6p)

c) För *i = 0* adderas *20* till *50* tre gånger i ”*do-while*” tills *j = 110*, sedan ökas *i* till *1* i *for*-satsen och *j = 110 + 20 = 130* bildas, men detta tolkas som det negativa talet *–(256 – 130) = – 126* och additionen fortsätter *12* gånger i ”*do-while*” tills *j = 114*, som är större än *100*.
Returvärde: *j = 114* **(3p)**