Lösningsförslag tenta 2011-03-14 **(v1 med reservation för eventuella fel!)**

1. a) LDS #$3000 → CF 30 00 **(1p)**

b) 18 16 → SBA **(1p)**

c) 1800_{16}: 0E E9 00 0F 80 Operationskoden 0E gäller för instruktionen BRSET med indexerad adressering.

Andra byten xb = E9 ger att typen är –n,Y med 9-bitars offset med formatet: 0E E9 *ff mm rr*, där *ff* är de 8 låga bitarna av 9-bitarstalet –n = 100_{16}. n = $–(100000000_2)_{2k} = –100000000_2 = –100_{16}$

mm = $0F_{16}$ är en mask och *rr* = 80_{16} en 8-bitars offset för ett PC-relativt hopp, från adressen till nästa OP-kod, dvs 1805_{16}.

Offset = Tilladr – Frånadr ger att *Tilladr = Offset + Frånadr* = $FF80_{16} + 1805_{16} = 1785_{16}$.

Instruktionen är alltså: BRSET -$100,Y,#$0F,$1785 **(3p)**

d) SEV = ORCC #$02 **(1p)**

e) CPY $100,SP innebär indexerad adressering med det positiva talet 100_{16} som offset. Talet 100_{16} kräver 10-bitars ordlängd inklusive teckenbiten och därmed måste 16-bitars offset användas.
CPY $100,SP → AD F2 01 00 **(2p)**

f) 1200_{16}: LBRA $8000 → 18 20 qq rr, där qqrr är avståndet (offset) från nästa OP-kod till destinationen. Nästa OP-kod finns på adr 1204_{16}.

Offset = Tilladr – Frånadr = $8000_{16} – 1204_{16} = 6DFC_{16}$.

LBRA $8000 → 18 20 6D FC. Nästa instruktion utförs alltså på adressen 8000_{16}. **(2p)**

g) BHI (>) avser tal med tecken. Det innebär att vi skall tolka data som tal i intervallet [0, 255]. Talet $40_{16} = 4·16 = 64_{10}$.

CMPA utför subtraktionen: 64 – W och hoppvillkoret blir: 64 – W > 0 eller W < 64.
När hänsyn tas till talområdet blir hoppvillkoret: $0 \le W < 64$. **(2p)**

h) BPL (≥ 0) avser tal med tecken. Det innebär att vi skall tolka data som tal i intervallet [-128, 127]. Talet $A5_{16} = 10·16 + 5 = 165_{10}$ tolkas som det negativa talet –(256 – 165) = –91.

ADDB utför additionen: –91 + W och hoppvillkoret blir: –91 + W ≥ 0 eller W ≥ 91.
När hänsyn tas till talområdet blir hoppvillkoret: $91 \le W \le 127$.

Eftersom flaggvillkoret inte tar hänsyn till overflow behöver man testa det fallet.
Overflow inträffar om –91 + W < –128, dvs W < 91 – 128 = –37 och intervallet blir:
$–128 \le W < –37$. Även i detta fall utförs hoppet eftersom N-flaggan då får fel värde.

Eftersom vi använder 2k-representation blir det verkliga intervallet: $256 –128 \le W < 256 – 37$ eller $128 \le W < 219$

De båda intervallen kan kombineras och blir då: $91 \le W < 219$ **(4p)**

i) Om ledningarna är för långa eller om datahastigheten är för hög så anländer data och klocka ej samtidigt till mottagaren. **(1p)**

j) Om avståndet är för stort mellan noderna kommer löptiden för signalen att bli så stor att noderna inte kan synkroniseras inom bitintervallen. **(2p)**

k) $43935000_{16} = 0/100\ 0011\ 1/001\ 0011\ 0101\ 0000\ 0000\ 0000_2$ (s = 0, c = 135, f = $.00100110101_2$)
exp = 135 – 127 = 8 N = +1.00100110.10100…0 $*2^8$ = 100100110.101 = $294{,}625_{10}$ **(2p)**

l) 6 decimala siffror ger 10^6 olika talkombinationer. Eftersom $10^3 \approx 2^{10}$ är $10^6 = (10^3)^2 \approx$
$\approx (2^{10})^2 = 2^{20}$ vilket motsvarar 20 bitar. **(2p)**

m 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. **(2p)**

2. a)

```
*
*   Subrutin ADD2
*
ADD2    PSHX            Spara register
        PSHY
*
        CLRB            Nollställ C-räknare låg byte
        CLR   CCNTH       - ” -       hög byte

ADLOOP  LDAA  1,X+      Hämta databyte från STRING1
        ADDA  1,X+      Addera till nästa byte

        BCC   NOCY      Carry = 0? Ja, ej overflow
        INCB            Öka carryräknare (overflow)
        BNE   NOCY
        INC   CCNTH     Öka hög byte

NOCY    STAA  1,Y+      Placera summan i STRING2
        BNE   ADLOOP    Nästa varv
*                       (Automatisk nollterminering)

        LDAA  CCNTH     Hämta overflowräknare hög byte

        PULY            Återställ register
        PULX
        RTS             Retur: Antal ”overflow” i D-reg

CCNTH   RMB   1         C-räknare hög byte
```

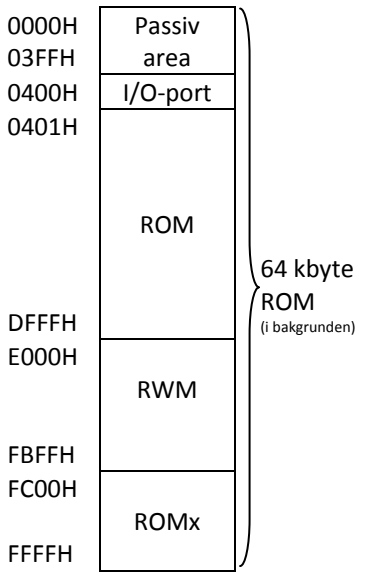
(7p)

b)

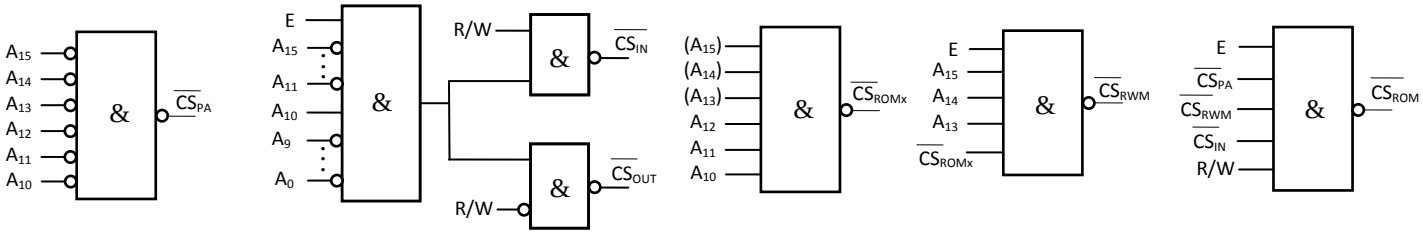
```
                        ~
        LDD   #$FFEC    2          (-20)
DLOOP   LDY   #$FFF6    2*20       (-10)
YLOOP   IBNE  Y,YLOOP   3*10*20
        LBRN  YLOOP     3*20
        IBNE  D,DLOOP   3*20
```

N = 2 + (2 + 3*10 + 3 +3)*20 = <u>762</u> st **(3p)**

3.

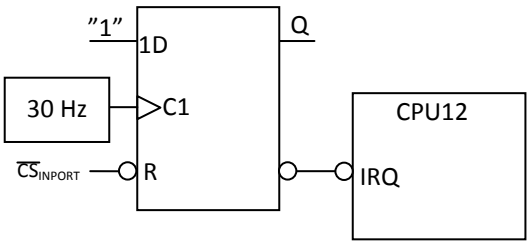


		A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Passiv area $1k = 2^{10}$	Start: 0000H =		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Slut: 03FFH =		0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
					CS													
I/O-port:		A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Adr: 0400H =		0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
										CS								
RWM: $8k = 2^{13}$		A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Start: E000H =		1	1	1	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														
ROMx: $1k = 2^{10}$		A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Start: FC00H =		1	1	1	1	1	1	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													



(8p)

4. a)



(2p)

b)

```
IRQRUT  MOVB   INPORT,INVAR   Läs inport och uppdatera INVAR och nolla vippa.
        DEC    COUNT         10 Hz-räknaren COUNT = 0?
        BNE    IRQEX         Nej, tillbaka till huvudprogram

        MOVB   #3,COUNT      Ja, ominitiera 10 Hz-räknaren
        JSR    CONTROL       Sköt utmaning via CONTROL
IRQEX   RTI                  Tillbaka till huvudprogram
```

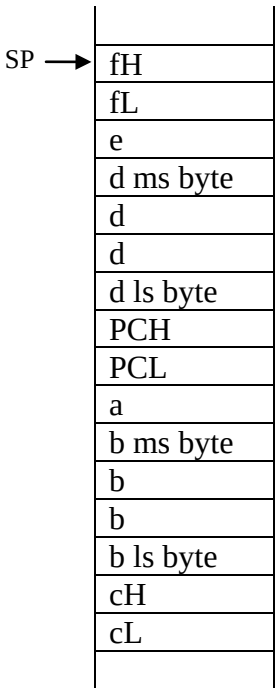
(4p)

c)

```
        (LDS    #BOS          Bottom of stack)
        …                     .
        MOVW   #IRQRUT,$FFF2 Initiera avbrottsvektorn
        MOVB   #3,COUNT      Initiera räknare för 10 Hz
        TST    INPORT        Nollställ avbrottsvippan
        CLI                  Aktivera avbrottssystemet
```

(2p)

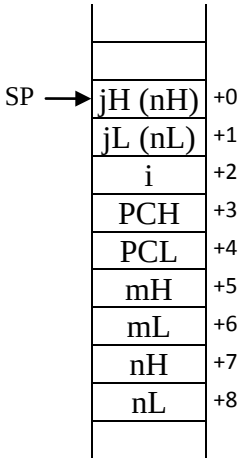
5. a)



(3p)

b)

```
int funcb(int m, int n) {        funcb     LEAS   -3,SP
    unsigned char i;
    int j = n;                             LDD    7,SP
                                           STD    0,SP
    do{                          DOLOOP
     for ( i = 0; i < 200; i++ )           CLR    2,SP

                                 FORLOOP   LDAB   2,SP
                                           CMPB   #200
                                           BHS    WHILE

      j = j + m;                           LDD    0,SP
                                           ADDD   5,SP
                                           STD    0,SP
                                           INC    2,SP
                                           BRA    FORLOOP

    } while (j <= 10000);        WHILE     LDD    0,SP
                                           CPD    #10000
                                           BLE    DOLOOP

  return j;                                LDD    0,SP

  }                                        LEAS   3,SP
                                           RTS
```



(6p)