Lösningsförslag tenta 2012-01-14 (v1 med reservation för eventuella fel!)

1. a) ORCC #1 ↔ SEC **(2p)**

b) 64 EB 24 38 → LSR [$2438,Y] **(2p)**

c) MOVB 2,X,4,Y → 18 0A 02 44 **(2p)**

d) 4876_{16}: LBGT $3518 → 18 2E qq rr
Nästa OP-kod finns på adr $487A_{16}$. qq rr = Till – Från = $3518_{16} – 487A_{16} = EC9E_{16}$
LBGT $3518 → 18 2E EC 9E **(2p)**

e) BHI avser tal utan tecken. Det innebär att vi skall tolka data som tal i intervallet [0, 255].
Talet $50_{16} = 5 \cdot 16 = 80_{10}$.

NEGA bildar talet $W_{2k} = 256 – W$ för $W > 0$. (Fallet W = 0 behandlas separat nedan.)
CMPA utför subtraktionen: $256 – W – 80 = 176 – W$.

Hoppvillkoret blir: $176 – W > 0$ eller $W < 176$.
När hänsyn tas till talområdet blir hoppvillkoret: $1 \leq W < 176$.

Om W = 0 gäller att NEGA bildar talet 0. CMPA utför då subtraktionen: $0 – 80 < 0$. (Ej hopp.)

Hoppvillkoret är alltså: $1 \leq W < 176$ **(3p)**

f) Hoppet utförs om teckenflaggan N = 1. Det innebär att vi skall tolka data som tal med tecken, med talområdet [-128, 127].

Talet $A3_{16} = 10 \cdot 16 + 3 = 163_{10}$ tolkas som det negativa talet $–(256 – 163) = –93$.

CMPB utför subtraktionen: $–93 – W$.

Hoppvillkoret blir: $–93 – W < 0$ (dvs. negativt, N = 1) eller $W > –93$
När hänsyn tas till talområdet blir hoppvillkoret: $–93 < W \leq 127$.

Om overflow inträffar får dock N-flaggan fel värde.

Här inträffar overflow vid subtraktionen om $–93 – W < –128$, dvs. $128 – 93 < W$, eller $W > 35$.

Vi får då hoppvillkoret: $–93 < W \leq 35$

Eftersom vi använder 2k-representation delar vi upp talintervallet i en positiv och en negativ del:
$0 \leq W \leq 35$ och $–93 < W \leq –1$, vilket motsvarar $256 –93 < W \leq 256 –1$ eller $163 < W \leq 255$
Hoppet utförs om: $0 \leq W \leq 35$ eller $163 < W \leq 255$ **(4p)**

g) Principen kallas arbitrering och innebär att alla meddelanden som kan sändas på bussen inleds med en unik identifierare som får tävla med andra samtidiga meddelanden genom att en av logiknivåerna på bussen (0) är dominant. Alla noder som sänder ett meddelande läser också av logiknivån på bussen och jämför med logiknivån den själv sänder. När värdena skiljer sig åt, dvs. den sänder en etta och läser en nolla, så avbryter den sändningen. Till sist kommer en ensam nod att sända sitt meddelande och medan övriga väntar tills bussen är ledig. **(2p)**

h) Format: s/c/f N = $120{,}875 = 1111000.111_2 = 1.111000111_2 \cdot 2^6$
s = 0 (+); c = $127 + 6 = 128 + 5 = 1000\ 0101_2$; f = 11100011100…0
$N_{flyt} = 0/100\ 0010\ 1/111\ 0001\ 1100\ 0000\ 0000\ 0000_2 = 42F1C000_{16}$ **(2p)**

i) Idealiskt får hela cacheminnet plats på processorchipet, men i verkligheten är utrymmet begränsat. I moderna processorer använder man tillgänglig chiparea till att lägga in flera processorer istället för att öka cachestorleken eftersom snabbhetsvinsten inte står i proportion till storleksökningen av cacheminnet på processorchipet. Man lägger istället till extra nivåer av cache utanför processor-chipet. **(1p)**

2. a)

```
BINASC  PSHX            Spara på stack
        LDX     #ASCTAB Pekare till ASCII-tabell
        ANDB    #$0F    Maska fram höger nibble
        LDAB    B,X     Hämta ASCII-tecken från tabell
        ANDB    #$7F    Maska bort eventuell bit 7
        PULX            Återställ stack
        RTS
ASCTAB  FCS  "0123456789ABCDEF"  Tabell med ASCII-tecken
```

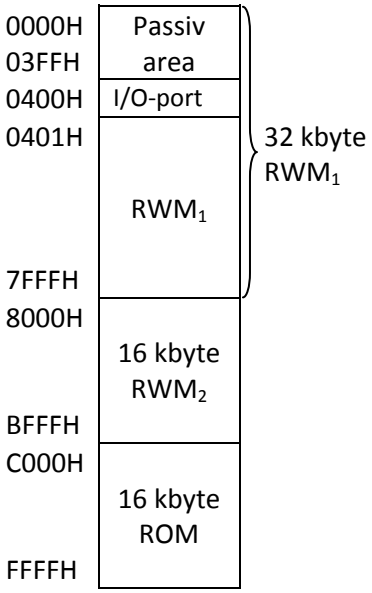
(4p)

b)

```
SWAP    PSHX            Spara på stack
        PSHD
        PSHB            Ordräknare till stack
        TST     ,SP     Färdigt?
        BEQ     SWEXIT  Ja
LOOP3   LDAA    ,X      Hämta dataord från tabell
        TFR     A,B     Gör kopia
        LSLD            Flytta låg nibble i reg A till vänster
        LSLD            och hög nibble i reg B till höger i reg A
        LSLD
        LSLD
        STAA    1,X+    Uppdatera tabell
        DEC     ,SP     Minska ordräknare
        BNE     LOOP3   Nästa byte om ej färdigt
SWEXIT  LEAS    1,SP    Justera stacken för ordräknaren
        PULD            Återställ stack
        PULX
        RTS
```

(7p)

3.



Passiv area, 1k = 2^{10}

	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: 03FFH =		0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
		CS						10 st									

RWM_1, 32k = 2^{15}

	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: 7FFFH =		0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		CS	15 st														

RWM_2, 16k = 2^{14}

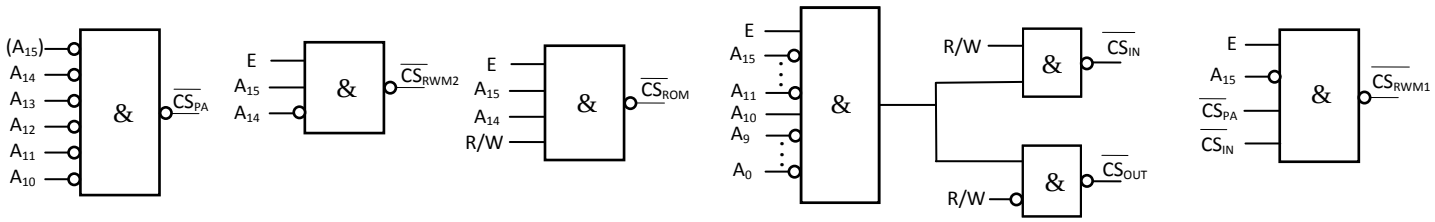
	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: 8000H =		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: BFFFH =		1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		CS		14 st													

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													

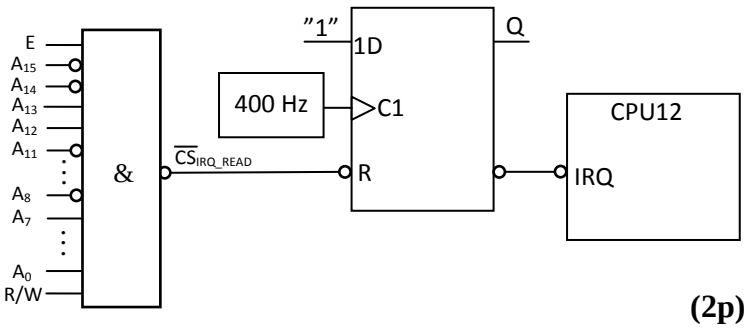
I/O-port:

	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr: 0400H =		0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
							CS										



(8p)

4. a) Adress för nollställning av avbrottsvippa:
$30FF_{16} = 0011\ 0000\ 1111\ 1111_2$



(2p)

b) Avbrottsrutin

```
IRQR    TST   IR_FF      Nollställ avbrottsvippa
        DEC   ACOUNT     10 Hz-räknare
        BNE   CHK_MIN    Ej 10 Hz

        MOVB  PORTA,ADRA
        MOVB  #40,ACOUNT Ominitiera räknare för 10 Hz
CHK_MIN LDX   BCOUNT     Minuträknare
        DEX
        STX   BCOUNT
        BNE   EX_IR      Ej hel minut

        MOVB  PORTB,ADRB
        MOVW  #24000,BCOUNT Ominitiera räknare för minut
EX_IR   RTI
```

(4p)

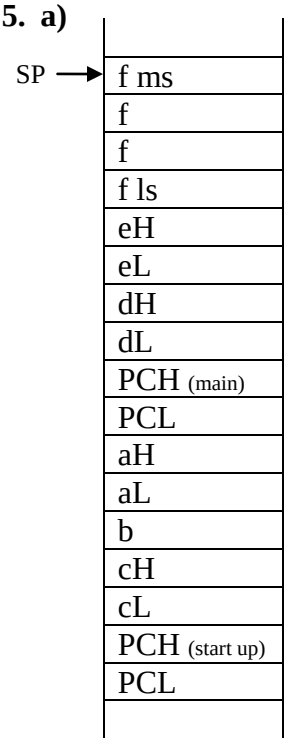
c) Avsnitt av huvudprogram

```
ACOUNT  EQU   $1FF0
BCOUNT  EQU   $1FF1
IR_FF   EQU   $30FF

START   LDS   #BOS          Initiera stackpekaren
        MOVW  #IRQR,$FFF2   Sätt avbrottsvektorn
        TST   IR_FF         Nollställ avbrottsvippa
        MOVB  #40,ACOUNT    Räknare för 10 Hz
        MOVW  #24000,BCOUNT Räknare för 1 minut
        CLI                 Aktivera avbrottssystem
        .
        .
        .
```

(3p)

5. a)

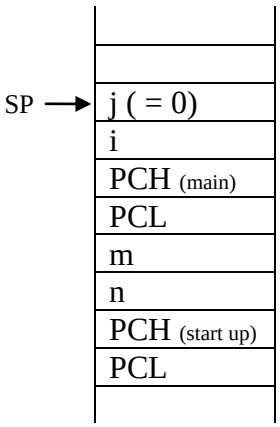


(3p)

b)

```
char funcb(char m, char n) {    _funcb:  LEAS   -2,SP
    char i;
    char j = 0;                          CLR    0,SP
    for ( i = 0; i < m; i++ )            CLR    1,SP
                                 _4:     LDAB   1,SP
                                         CMPB   4,SP
                                         BGE    _5
        j = j + 2*n;                     LDAB   5,SP
                                         ASLB
                                         ADDB   0,SP
                                         STAB   0,SP
                                         INC    1,SP
                                         BRA    _4

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



(6p)

c) Värdet *50* adderas till variabeln *j* sex gånger. *6·50 = 300*. Eftersom *j* har typen *char* har den ordlängden 8 bitar och rymmer tal i intervallet [0, 255]. Egentligen är talintervallet [-128, 127] eftersom *char* avser tal med tecken, men detta är ju bara en fråga om hur värdet tolkas. I en 8-bitars variabel räknas värdet modulo 256. *(300) mod 256 = 300 – 256 = 44*.
Returvärde: *j = 44*

(3p)