Lösningsförslag tenta 2010-03-08 (v1 med reservation för eventuella fel!)

1. a) CLC → ANDCC "$FE **(1p)**

b) 6B E2 16 24 → STAB $1624,X **(2p)**

c) TFR B,X ↔ SEX B,X **(1p)**

d) CPY -$FF,SP → AD F1 01 **(2p)**

e) 1057_{16}: BRN $1000 → 21 rr. Nästa OP-kod finns på adr 1059_{16}.
Offset(rr) = Till – Från = $1000_{16} - 1059_{16} = \text{FF}A7_{16}$ BRN $1000 → 21 A7
Nästa instruktion som utförs finns på adressen 1059_{16}. **(2p)**

f) BGE (≥) avser tal med tecken. Det innebär att vi skall tolka data som tal i intervallet [-128, 127].
Talet $C0_{16} = 12 \cdot 16 = 192_{10}$ tolkas som det negativa talet –(256 – 192) = –64.

Eftersom flaggvillkoret tar hänsyn till overflow behöver man inte testa det fallet.
CMPB utför subtraktionen: –64 – W och hoppvillkoret blir: –64 – W ≥ 0 eller W ≤ –64.
När hänsyn tas till talområdet blir hoppvillkoret: –128 ≤ W ≤ –64.

Eftersom vi använder 2k-representation blir det verkliga intervallet: 256 –128 ≤ W ≤ 256 – 64

Hoppvillkoret blir då: 128 ≤ W ≤ 192 **(2p)**

g) BPL (≥ 0) avser tal med tecken. Det innebär att vi skall tolka data som tal i intervallet [-128, 127].
Talet $65_{16} = 6 \cdot 16 + 5 = 101_{10}$.

CMPB utför subtraktionen: 101 – W och hoppvillkoret blir: 101 – W ≥ 0 eller W ≤ 101.
När hänsyn tas till talområdet blir hoppvillkoret: –128 ≤ W ≤ 101.

Eftersom flaggvillkoret (N = 0) inte tar hänsyn till overflow måste man testa om det kan inträffa.
Overflow inträffar vid subtraktionen, om 101 – W > 127, dvs 101 – 127 = –26 > W.

Det innebär att ett hopp kommer att utföras om –26 ≤ W ≤ 101. Eftersom vi använder 2k-representation för negativa tal delar vi upp talintervallet i en positiv och en negativ del:
0 ≤ W ≤ 101 och –26 ≤ W ≤ –1. Det negativa intervallet ersätts av: 256 – 26 ≤ W ≤ 256 – 1,
dvs 230 ≤ W ≤ 255.

Hopp utförs alltså om: 0 ≤ W ≤ 101 eller 230 ≤ W ≤ 255 **(3p)**

h) Vid asynkron seriekommunikation har sändare och mottagare olika klocksignal. Vid synkron seriekommunikation har de i princip samma klocka (vilket oftast åstadkoms genom att klocksignalen inkluderas i datasignalen). **(1p)**

i) 0, 35_{16}, 85_{16}, 539_{16} och $7FF_{16}$ (Lägre identiferarvärde ger högre prioritet.) **(2p)**

j) N = 653,375 = $(1010001101.011)_2 = 1.010001101011\ 2^9$; s = 0 (+), c = 9 + 127 = 136 = 128 + 8 =
= $(10001000)_2$; Nflyt = s/c/f = 0/10001000/01000110101100000000000 = 44235800_{16} **(2p)**

k) 12 decimala siffror ger 10^{12} olika talkombinationer. Eftersom $10^3 \approx 2^{10}$ är $10^{12} = (10^3)^4 \approx$
$\approx (2^{10})^4 = 2^{40}$ vilket motsvarar 40 bitar. **(2p)**

2. a)

```
*
*       Subrutin ADDNIB
*
        ADDNIB  PSHD            Spara på stack
                PSHX
                PSHY
        *
        ADLOOP  LDAA    1,X+    Hämta databyte från insträng
                TFR     A,B     Kopiera databyte
                ANDA    #$0F    Maska fram bit 3-0
                LSRB            Flytta bit 7-4 i kopian till bit 3-0
                LSRB
                LSRB
                LSRB
                ABA             Utför nibbleaddition
                STAA    1,Y+    Placera summan i utsträng
                BNE     ADLOOP  Nästa byte om summan ej 0.
*                               (Automatisk nollterminering)
        ADDEX   PULY            Återställ
                PULX
                PULD
                RTS
```

(5p)

b)

```
                ORG     $1000       ~
                LDAA    #$F6        1
        ALOOP   LDX     #$FF9C      2       *10
        XLOOP   IBNE    X,XLOOP     3*100   *10
                LBRN    ALOOP       3       *10
                IBNE    A,ALOOP     3       *10
```

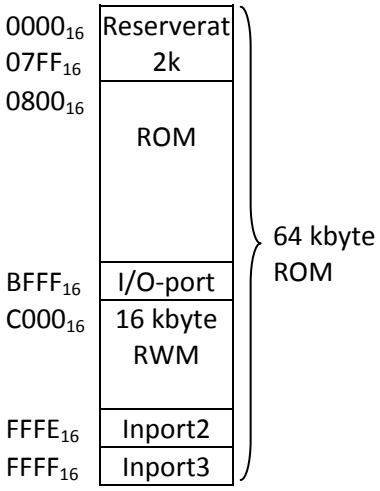
N = 1+(2+3*100+3+3)*10 = 3081 **(3p)**

c)

```
        SUBRUT  LDY     ,SP     Stackinnehåll till Y-reg
                LDAA    ,Y      Inparameter 1till A-reg
                LDX     1,Y     Inparameter 2 till X-reg
                LEAY    3,Y     Justera Y till återhoppsadress
                STY     ,SP     Uppdatera SP med återhoppsadress
                .
                .
                RTS
```

(4p)

3.



Passiv area, 2k = 2^{11}

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: $07FF_{16}$ =	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
	CS					11 st										

Inp1/Utport:

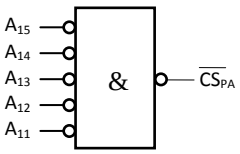
A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr: $BFFF_{16}$ =	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						CS										

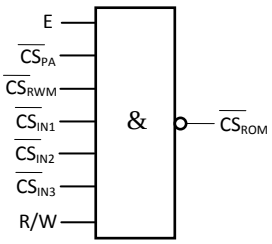
RWM, 16k = 2^{14}

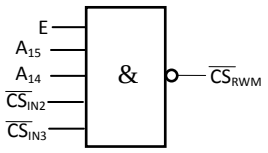
A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: $C000_{16}$ =	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: $FFFF_{16}$ =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	CS		14 st													

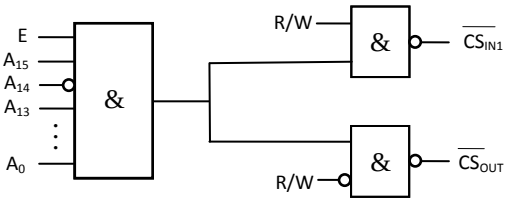
Inport2/3:

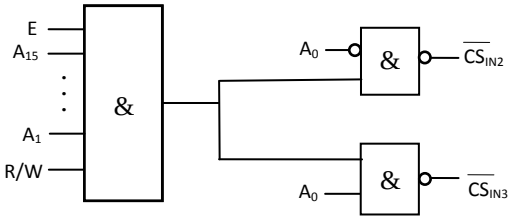
A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr: $FFFX_{16}$ =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	x
								CS								





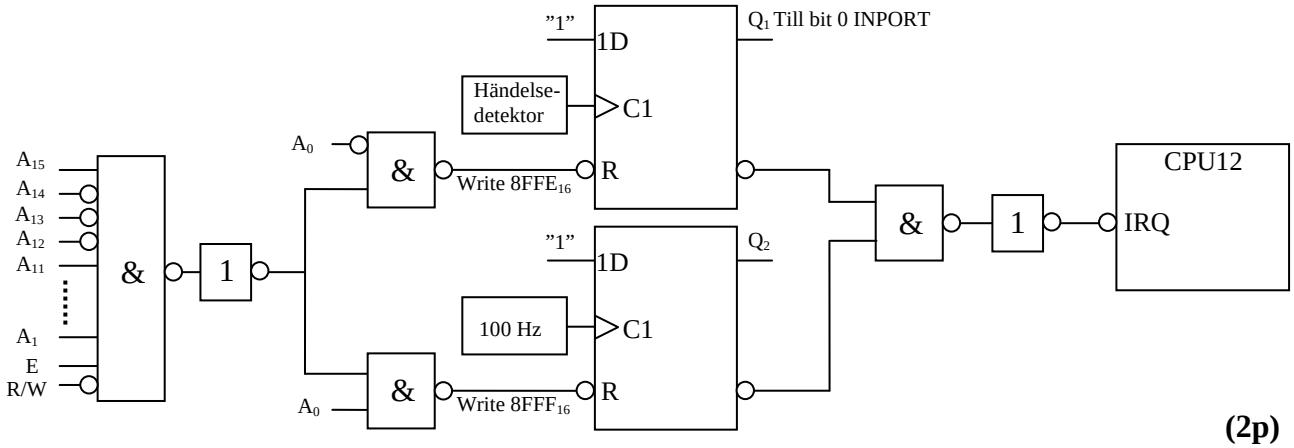






(8p)

4. a)



(2p)

b)

```
BOSA    EQU   $2000
BOSB    EQU   $3000
ProgA   EQU   $1000
ProgB   EQU   $2000
IRQVEC  EQU   $FFF2
IRQFF   EQU   $8FFE
*
        ORG   $1E00     Globala variabler
EVENT   RMB   1         Händelseräknare
SPSAVE  RMB   2         Passiv stackpekare
```

```
        ORG   ProgA
        CLR   IRQFF     Nollställ avbrottsvippa Event
        CLR   IRQFF+1   Nollställ avbrottsvippa 100Hz
        CLR   EVENT     Nollställ händelseräknaren
        MOVW  #IRQR,IRQVEC
        LDS   #BOSB-9   Stackpekare B efter avbrott
        MOVB  #$C0,0,SP Init CCR i B-stacken
        MOVW  #ProgB,7,SP Init returadress till prog B
        STS   SPSAVE    Andra programmets stack
        LDS   #BOSA     Init A-stack
        CLI             Aktivera avbrott
P_A     BRA   P_A       Evighetsslinga A
```

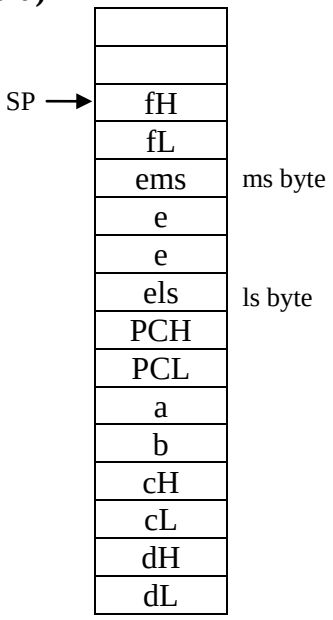
(4p)

c)

```
IRQR    LDAA  INPORT    Vilken avbrottskälla?
        ANDA  #1        Bit 0?
        BEQ   HZ100     Nej, processbyte
        CLR   IRQFF     Ja, nollställ avbrottsvippa Event
        INC   EVENT     Öka händelseräknaren
        BRA   IRQEX     Hoppa ut från IRQR
HZ100   CLR   IRQFF+1   Nollställ avbrottsvippa 100Hz
        LDD   SPSAVE    Hämta andra stackpekaren
        STS   SPSAVE    Spara aktuell stackpekare
        TFR   D,SP      Byt stackpekaren till den andra
IRQEX   RTI
```

(3p)

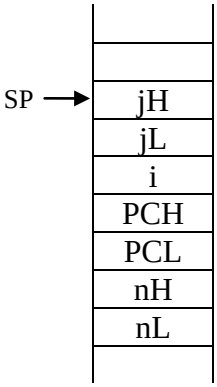
5. a)



(3p)

b)

```
int funcb(int n){   _funcb:     LEAS  -3,SP
  char i;
  int j;
  i = 1;                        LDAB    #1
                                STAB    2,SP
  j = n;                        LDD     5,SP
                                STD     0,SP
  while (i < 50 ){      while   LDAB    2,SP
                                CMPB    #$32
                                BGE     return
    if (j > 5)                  LDD     0,SP
                                CPD     #5
                                BLE     else
      i = i + 10;               LDAB    2,SP
                                ADDB    #10
                                STAB    2,SP
                                BRA     while
    else                 else
      j = i + j;                LDAB    2,SP
                                SEX     B,D
                                ADDD    0,SP
                                STD     0,SP
                                BRA     while
  }
  return j;             return  LDD     0,SP
}                               LEAS    3,SP
                                RTS
```



(5p)

6.

Minnesmodulerna har begränsad kapacitet och för lång accesstid. 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. Det innebär att man skulle kunna klara sig med ett betydligt mindre minne under förutsättning att det innehåller den data processorn behöver.

Man kan därför sänka accesstiden genom att sätta in ett litet men mycket snabbt minne (cache) mellan processorn och ”main memory”. Processorn läser och skriver bara i cacheminnet och när data saknas där hämtas saknade data och närliggande data till cache från primärminnet. Sannolikheten är sedan stor att de följande accesserna görs i cacheminnet.

Primärminnesstorleken kan man ”öka” (virtuellt minne) genom att låta hela adressrummet finnas på en hårddisk, medan bara det som används för tillfället finns i det verkliga mindre primärminnet (det fysiska minnet). Enligt samma resonemang som för cacheminnet ovan är sannolikheten stor att data som processorn behöver verkligen finns i det fysiska minnet. **(3p)**