



Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D

EDA486 Programmering av inbyggda system Z

DAT016 Programmering av inbyggda system IT

DIT152 Programmering av inbyggda system GU

Fredag 19 augusti 2016, kl. 8.30 - 12.30

Examinator

Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:

Roger Johansson

Tillåtna hjälpmedel

Häftet

Instruktionslista för CPU12

Quick Guide, Laborationsdator MD407 med tillbehör

Inget annat än rättelser och understrykningar får vara införda i häftena.

Du får också använda bladet:

C Reference Card

samt *en* av böckerna:

*Vägen till C,
Bilting, Skansholm*

*C från början
Skansholm*

*The C Programming Language,
Kernighan, Ritchie (även utskrift av PDF-
versionen, hel eller delvis)*

Endast "överstrykningar" och understrykningar får vara införda i boken.

Tabellverk eller miniräknare får ej användas.

Lösningar

anslås senast dagen efter tentamen via kursens hemsida.

Granskning

Tid och plats anges på kursens hemsida.

Allmänt

Tentamen är anpassad för både det äldre laborationssystemet (*MC12*) och det nyare (*MD407*). Varje uppgift föregås av någon av bokstäverna A, B eller C.

A: uppgift avser laborationssystem *MC12*.

B: uppgift avser laborationssystem *MD407*.

C: uppgift kan besvaras av alla.

Du väljer själv om du vill besvara A eller B-märkta uppgifter. Du ska inte besvara både A och B-märkta uppgifter. Om din lösning innehåller både A och B-märkta uppgifter kommer de A-märkta uppgifterna att bedömas.

Siffror inom parentes anger full poäng på uppgiften.

För full poäng krävs att:

- redovisningen av svar och lösningar är läslig och tydlig. Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- lösningen ej är onödigt komplicerad.
- du har motiverat dina val och ställningstaganden
- assemblerprogram är utformade enligt de råd och anvisningar som givits under kursen och tillräckligt dokumenterade.
- C-program är utformade enligt de råd och anvisningar som getts under kursen. I programtexterna skall raderna dras in så att man tydligt ser programmets struktur.

Betygsättning

För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända.

Tentamenspoäng ger slutbetyg enligt:

(EDA/DAT/LEU):

$20p \leq \text{betyg } 3 < 30p \leq \text{betyg } 4 < 40p \leq \text{betyg } 5$

respektive (DIT):

$20p \leq \text{betyg } G < 35p \leq VG$

Uppgift A-1 (10p) Kodningskonventioner (C/asmblerspråk)

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (se bilaga 1).

a) I ett C-program har vi följande deklaringer:

```
void func(long adam, unsigned char bertil, struct one * ceasar )
{
    long          a = adam;
    unsigned int  b = bertil;
    struct one    *c = ceasar;
    /* Övrig kod i funktionen är bortklippt eftersom vi bara
       betraktar anropskonventionerna. */
}
```

Översätt *hela* funktionen `func`, som den är beskriven, till HCS12 asmblerspråk, såväl *prolog* som *tilldelningar* och *epilog* ska alltså finnas med. Speciellt ska du börja med att beskriva aktiveringsposten, dvs. stackens utseende i funktionen och där riktningen för minskande adresser hos aktiveringsposten framgår. (6p)

b) I samma C-program har vi dessutom följande deklaringer givna på ”toppnivå” (global synlighet):

```
struct one * c;
long      a;
char      b;
```

Visa hur variabeldeklaringerna översätts till assemblerdirektiv. Beskriv dessutom hur följande funktionsanrop översätts till assemblerkod. (4p)

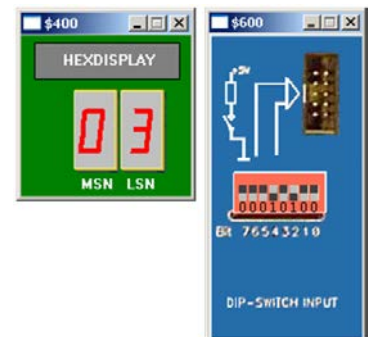
```
func( a , b , c );
```

Uppgift A-2 (6p) Assemblerprogrammering

En 8-bitars strömbrytare, ”DIP_SWITCH” är ansluten till adress \$600 och en displayenhet ”HEXDISPLAY” som visar en byte i form av två hexadecimala siffror är ansluten till adress \$400 i ett MC12 mikrodatorsystem.

Konstruera en subrutin `DipHex` som läser av strömbrytaren och indikerar den minst signifikanta påslagna biten genom att skriva dess position, räknat från höger, till displayenheten. Om exempelvis bitarna 2 och 4 utgör ettställda strömbrytare ska positionen för bit 2, (dvs. 3) skrivas till displayenheten.

Om ingen strömbrytare är ettställd ska siffran 0 skrivas till displayen. Speciellt gäller att endast symboler ska användas för absoluta adresser.



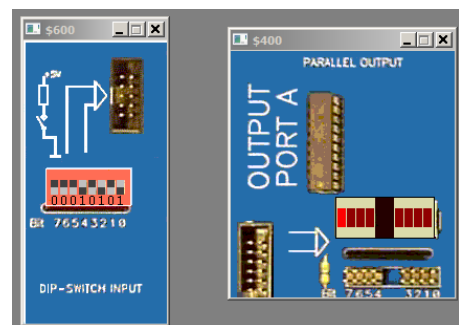
Uppgift A-3 (6p) In och utmatning beskriven i C

I denna uppgift ska du bland annat demonstrera hur absolutadressering utförs i C. För full poäng ska du visa hur preprocessordirektiv och ev. typdeklaringer används för att skapa begriplig programkod.

En 8-bitars strömbrytare är ansluten till adress 0x600 och en 8-bitars diodramp är ansluten till adress 0x400 i ett MC12 mikrodatorsystem.

Skriv en funktion `void OddIndicator(void)` som

- läser de 8 bitarna från strömbrytaren
- om antalet ett-ställda bitar är udda ska nu b_7 hos ljusdiodrampen tändas, annars ska b_0 hos ljusdiodrampen tändas.



Uppgift A-4 (6p) Avbrott, HCS12

Redogör för hur avbrott går till i ett HCS12-system, med en yttre enhet ansluten till IRQ-ingången på processorn. Ditt svar skall bland annat innehålla

- en övergripande beskrivning av vad avbrott är och hur det fungerar
- en beskrivning av de olika programrutiner som är förknippade med avbrott (eventuellt pseudokod / assembler)

Uppgift B-1 (10p)

En ljusdiod har anslutits till en pinne hos port D enligt figuren till höger.

Skriv ett program, i C, som kontinuerligt tänds dioden under en halv sekund och därefter släcker den under en halv sekund. Din lösning ska fördelas på följande deluppgifter:



- Visa hur SysTick kan användas för att skapa en (blockerande) fördröjning om 250 mikrosekunder (en fjärdedels millisekund), med funktionen `void delay_250mys(void);`. Systemets klockfrekvens är 168 MHz. (6p)
- Skriv en funktion `void init_app(void);` som sätter upp port D, bit 0 som utsignal, "push-pull". Övriga inställningar i porten ska behållas, dvs. får inte ändras av initieringen. (2p)
- Skriv huvudprogrammet som får dioden att blinka. Du kan använda `delay_250mys()` och `init_app()` även om du inte besvarat a) och/eller b). (2p)

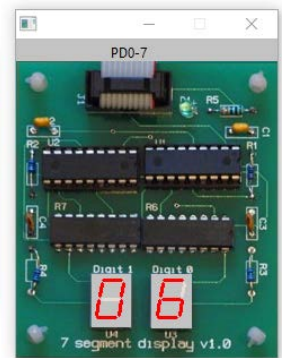
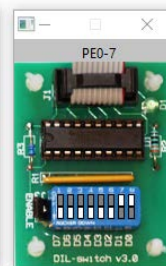
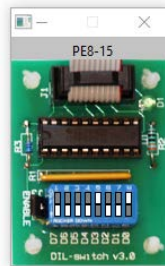
Uppgift B-2 (6p)

Två DIL-strömbrytare och en Double hexadecimal display kopplas enligt:

- DIL1 till Port E (0-7),
- DIL2 till Port E (8-15)
- Display till port D (0-7):

Skriv ett program i ARM/Thumb assemblerspråk som:

- Initierar portar D och E.
- Kontinuerlig läser de inställda värdena från DIL-strömbrytarna, subtraherar (DIL1) från (DIL2) och skriver resultatet till displayen.



Uppgift B-3 (8p)

Förbered en enkel applikation som använder PD1 hos MD407 som avbrottsingång. Beskriv initieringsfunktioner och avbrotts hantering, dvs. färdigställ följande:

```
void irq_handler ( void )
{
    Om avbrott från EXT11:
        kvittera avbrott från EXT11
}
void enable_interrupt( void )
{
    Nollställ processorns avbrottsmask
}
void app_init ( void )
{
    Koppla PD1 till avbrottslina EXT11;

    Konfigurera EXT11 för att generera avbrott;
    Konfigurera för avbrott på negativ flank

    Sätt upp avbrottsvektor

    Konfigurera de bitar i NVIC som kontrollerar den avbrottslina som EXT11 kopplats till
}
```

Uppgift B-4 (4p)

Visa hur en USART-modul kan beskrivas med en sammansatt datatyp `struct`, i C. Förutsatt att registeråtkomster alltid är 32 bitars ord.

Uppgift C-5 (8p) Programmering med pekare

Funktionen `notInString` kan beskrivas på följande sätt:

```
int notInString ( const char * str1, const char * str2 );
```

Returns the length of the initial portion of *str1* which consists only of characters that are *not* part of *str2*.

Parameters

str1 C string to be scanned.

str2 C string containing the characters to match.

Return value

The length of the initial portion of *str1* containing only characters that does not appear in *str2*.

Therefore, if all of the characters in *str1* are in *str2*, the function returns zero if only the first character in *str1* is not in *str2*.

If none of the characters in *str1* are in *str2* the function returns the length of the entire *str1* string.

Exempel på användning:

```
int main()
{
    char string[]="Elm7a";
    printf("The number length is %d.\n", notInString (string,"1234567890"));
    return 0;
}
```

Utskriften ska då bli: "The number length is 3." eftersom tre tecken (Elm) som inte finns i referenssträngen föregår det första tecknet (7) som ju finns i referenssträngen.

Din uppgift är att, i C, skriva funktionen `notInString`. Du får inte använda dig av indexering, utan måste utnyttja pekare. Du får inte anropa någon annan standardfunktion. Du måste alltså skriva all kod själv.

Uppgift C-6 (14p) Maskinnära programmering i C

a) Skapa en struct `tObject` som innehåller följande medlemmar:

- en pekare `pName`,
- ålder
- samt att man efteråt skall kunna instansiera variabler av typen utan att behöva ange nyckelordet `struct`. `T ex Object a;` (3p)

b) Initiera din instans med direktinitiering, dvs tillsammans med deklarationen. Använd ditt eget namn och ålder. (2p)

c) Punktnotation: Inkrementera åldern med medlem-notation. (1p)

d) Programmering av inbrottslarm. (8p)

En utrustning för automatisk övervakning av inbrottslarm har åtta ingångar. Till sju av dessa kopplas man sensorer. Till den åttonde ingången kopplas en knapp. Utrustningen har också en utgång till vilken en högtalare (siren) har kopplats. Utrustningen styrs av en dator via två åttabits register: ett styrregister och ett dataregister på de hexadecimala adresserna 500 resp. 501. Styrregistret är endast skrivbart.

Dataregistret är både läs- och skrivbart. Följande bitar används:

Bit 0: (ONOFF), huvudströmbrytare för utrustningen (1 == ON, 0 == OFF)

Bit 1: (Sound_ONOFF), anger om utgången till högtalaren ska vara aktiv (1) eller passiv (0)

Bit 2: (IE) anger om utrustningen ska generera avbrott när någon av insignalerna blir hög (1)

När en insignal till utrustningen blir hög sätts automatiskt motsvarande bit i dataregistret till 1. Bitarna 0-6 används för sensorerna och bit nr 7 till knappen. Om biten IE är påslagen genereras ett avbrott när någon av insignalerna blir hög. Ett avbrott kvitteras genom att man nollställer motsvarande bit i dataregistret.

Förutom övervakningsutrustningen finns också en display med lampor. Displayen används för att indikera vilka sensorer som givit utslag och styrs via ett skrivbart åttabits register på den hexadecimala adressen 600. Genom att skriva en etta eller nolla i en bit i detta register tänds resp. släcks motsvarande lampa på displayen.

Din uppgift är att skriva en modul som styr övervakningsutrustningen. Din modul ska skrivas helt i C. Den ska innehålla de två funktionerna `alarm_on` och `alarm_off` vilka ska kunna anropas från en

annan del av programmet. Dessutom ska det finnas kod som tar hand om avbrott från övervakningsutrustningen. Funktionen `alarm_on` ska göra alla initieringar som behövs för att aktivera övervakningsutrustningen. I detta ingår att se till att avbrott aktiveras. Funktionen `alarm_off` ska helt enkelt stänga av utrustningen. Du ska också skriva en funktion `sensor_inter` som anropas vid varje avbrott. Du ska inte skriva processorspecifika delar av avbrottshanteringen, dvs. du behöver inte bry dig om *hur* funktionen anropas vid avbrottet.

När utrustningen ger avbrott ska det fungera på följande sätt: Om avbrottet kommer från någon av de sju ingångar som är kopplade till sensorer ska motsvarande lampa på displayen tändas. De eventuella lampor som tidigare tänts ska förbli tända. Om användaren inte tryckt på knappen ska högtalaren också kopplas på. När användaren trycker på knappen första gången ska högtalaren stängas av, men de lampor som är tända på displayen ska förbli tända. Om användaren trycker på knappen en andra gång ska alla lampor på displayen släckas. Man ska då återkomma till utgångsläget. Tanken är alltså att när ett larm inträffas så uppmärksammas användaren på detta genom att högtalaren tjuiter. Han trycker då en gång på knappen för att stänga av ljudet och kan sedan i lugn och ro på displayen se vilken eller vilka sensorer som givit utslag. När orsaken till larmet klargjorts kan han trycka en gång till på knappen för att återställa allt.

Observera att du måste skriva all C-kod, inklusive de deklARATIONER och DEFINITIONER som behövs. Ange också i vilka filer koden lämpligen placeras.

Bilaga 1: Kompilatorkonvention XCC12:

- Parametrar överförs till en funktion via stacken och den anropande funktionen återställer stacken efter funktionsanropet.
- Då parametrarna placeras på stacken bearbetas parameterlistan från höger till vänster.
- Lokala variabler översätts i den ordning de påträffas i källtexten.
- *Prolog* kallas den kod som reserverar utrymme för lokala variabler.
- *Epilog* kallas den kod som återställer (återlämnar) utrymme för lokala variabler.
- Den del av stacken som används för parametrar och lokala variabler kallas *aktiveringspost*.
- Beroende på datatyp används för returparameter HC12:s register enligt följande tabell:

Storlek	Benämning	C-typ	Register
8 bitar	byte	char	B
16 bitar	word	short int och pekartyp	D
32 bitar	long float	long int float	Y/D

Bilaga 2: Assemblerdirektiv för MC68HC12.

Assemblerspråket använder sig av mnemoniska beteckningar som tillverkaren Freescale specificerat för maskininstruktioner och instruktioner till assemblern, s.k. pseudoinstruktioner eller assemblerdirektiv. Pseudoinstruktionerna framgår av följande tabell:

Direktiv	Förklaring
ORG N	Placerar den efterföljande koden med början på adress N (ORG för ORiGin = ursprung)
L: RMB N	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adress L. (RMB för Reserve Memory Bytes)
L: EQU N	Ger label L konstantvärdet N (EQU för EQUates = beräknas till)
L: FCB N1 , N2 . .	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FCB för Form Constant Byte)
L: FDB N1 , N2 . .	Avsätter i följd i minnet ett bytepar (två bytes) för varje argument. Respektive bytepar ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FDB för Form Double Byte)
L: FCS "ABC"	Avsätter en följd av bytes i minnet, en för varje tecken i teckensträngen "ABC". Respektive byte ges ASCII-värdet för A, B, C etc. Följden placeras med början på adress L. (FCS för Form Caracater String)


Bilaga 3: Assemblerdirektiv för ST32F407.

Direktiv	Förklaring
L: .SPACE N	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adress L.
L: .BYTE N1 , N2 . .	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
L: .HWORD N1 , N2 . .	Avsätter i följd i minnet ett 16 bitars ord för varje argument. Respektive ord ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
L: .WORD N1 , N2 . .	Avsätter i följd i minnet ett 32 bitars ord för varje argument. Respektive ord ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
.ALIGN	Garanterar att påföljande adress är jämnt delbar med 4 ("word aligned")

Lösningförslag

Uppgift A-1a:

Beskrivning av aktiveringspost

<i>minnesanvändning</i>	<i>adress</i>	
ceasar	15, SP	<i>minskande adress</i> 
bertil	14, SP	
adam	10, SP	
PC (vid JSR)		
a	4, SP	
b	2, SP	
c	0, SP	

```

; void func(long adam, unsigned char bertil, struct one * ceasar )
_func:
; {
; LEAS -8,SP
; long a = adam;
; LDY 10,SP
; LDD 12,SP
; STY 4,SP
; STD 6,SP
; unsigned int b = bertil;
; LDAB 14,SP
; CLRA
; STD 2,SP
; struct one *c = ceasar;
; LDD 15,SP
; STD 0,SP
; /* Övrig kod i funktionen är bortklippt eftersom vi bara
; betraktar anropskonventionerna. */
; }
; LEAS 8,SP
; RTS
    
```

Uppgift A-1b: (4p):

```

_c:   RMB   2
_a:   RMB   4
_b:   RMB   1
      LDD   _c
      PSHD
      LDAB  _b
      PSHB
      LDD   2+_a
      PSHD
      LDD   _a
      PSHD
      JSR   _func
      LEAS  7,SP
    
```

Uppgift A-2:

```

; Symboliska adresser
DipSwitch EQU $600
HexDisp EQU $400
    
```

```

; Subrutin DipHex
DipHex:  LDAA  DipSwitch
        CLRB

DipHex10: TSTA
        BEQ   DipHex20

        INCB
        LSRA
        BCC   DipHex10

DipHex20: STAB  HexDisp
        RTS
    
```

Uppgift A-3:

```

typedef  unsigned char * port8ptr;
#define  OUT *((port8ptr) 0x400)
#define  IN  *((port8ptr) 0x600)

void  OddIndicator( void )
{
    unsigned char count, portvalue;

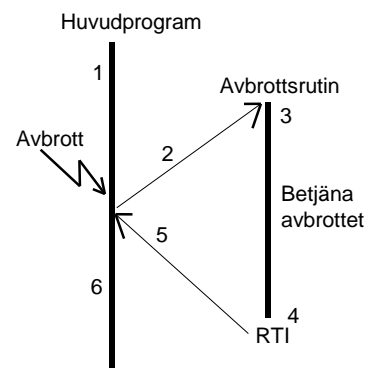
    portvalue = IN;
    count = 0;
    while( portvalue )
    {
        if( portvalue & 1 )
            count++;
        portvalue = portvalue >> 1;
    }
    if( count & 1 )
        OUT = 0x80;
    else
        OUT = 1;
}
    
```

Uppgift A-4:

Ett sätt att avbryta processorns normala exekvering.

Händelseförloppet vid ett avbrott.

1. Processorn vars I-flagga är nollställd känner att IRQ är aktiverad.
2. Processorn sparar alla register på stacken. I-flaggan ettställs. Därefter läser processorn startadressen för avbrottsrutinen från den aktuella avbrottsvektorn som placeras i PC
3. Avbrottsrutinen startas
4. Avbrottsrutinen avslutas med instruktionen RTI dvs...
5. register innehållen återställs från stacken (Restore Status) Återhopp till huvudprogram med en nollställd I-flagga.
6. Därmed återstartas huvudprogrammet där det blev avbrutet.



Programrutiner: (Initieringsrutin och Avbrottsrutin)

```

* Initieringsrutin (En generell)
IrqInit
    ---  Init eventuella variabler som ingår i avbrottsrutinen
    ---  Nollställ avbrottsvippan
    LDX  #IrqRutin          Initiera avbrottsvektor
    STX  $FFF2
    CLI                               Aktivera avbrottsmekanismen

* Avbrottsrutin IrqRutin: En generell avbrottsrutin
IrqRutin
    ---  Kvitteta avbrott
    ---  Betjäna (Serva) Avbrottet
    rti
    
```

Uppgift B-1:


```

a)
#define SysTickCtrl ((volatile unsigned int *) (0xE000E010))
#define SysTickLoad ((volatile unsigned int *) (0xE000E014))
#define SysTickVal ((volatile unsigned int *) (0xE000E018))

void delay_250mys( void )
{
    /* SystemCoreClock = 168000000 */
    *SysTickCtrl = 0;
    *SysTickLoad = ( (168000/4) -1 );
    *SysTickVal = 0;
    *SysTickCtrl = 5;
    while( (*SysTickCtrl & 0x10000 )== 0 );
    *SysTickCtrl = 0;
}

b)
#define portBargraphModer ((volatile unsigned int *) (0x40020C00))
#define portBargraphOtyper ((volatile unsigned short int *) (0x40020C04))
void init_app( void )
{
    /* PORT D */
    *portBargraphModer    &= ~3; /* återställ bit 0 mode */
    *portBargraphModer    |= 1; /* bit 0 sätts som utgång */
    *portBargraphOtyper   &= ~1; /* återställ bit 0 typ, är nu push/pull */
}

c)
void main(void)
{
    int i;
    init_app();
    while(1)
    {
        *portBargraphOdrLow = 0;
        for(i=0;i<2000;i++) delay_250mys();
        *portBargraphOdrLow = 0xFF;
        for(i=0;i<2000;i++) delay_250mys();
    }
}

```

Uppgift B-2:

```

start:
@ initiera port D0-D7 som utport
LDR    R0,=0x00005555
LDR    R1,=0x40020C00
STR    R0,[R1]
@ initiera port E0-E15 som inport
LDR    R0,=0
LDR    R1,=0x40021000
STR    R0,[R1]
@ adressen till port D:s ut-dataregister till R5
LDR    R5,=0x40020C14
@ adressen till port E:s in-dataregister till R6
LDR    R6,=0x40021010

main:
LDRB   R1,[R6]
LDRB   R0,[R6,#1]
SUB    R0,R0,R1
STRB   R0,[R5]
B      main

```

Uppgift B-3:

```

#define SYSCFG_EXTICR1    0x40013800
#define EXTI_IMR          0x40013C00
#define EXTI_FTSCR       0x40013C0C
#define EXTI_PR          0x40013C14
#define NVIC_ISER0       0xE000E100
#define EXTI1_IRQ_BPOS   1<<1
#define EXTI1_IRQVEC     0x2001C05C
#define NVIC_EXTI1_IRQ_BPOS (1<<7)

```

```

void irq_handler ( void )
{
    /* Om avbrott från EXTI1:
       kvittera avbrott från EXTI1 */
    if( *(unsigned int *) EXTI_PR) & EXTI1_IRQ_BPOS )
    {
        *(unsigned int *) EXTI_PR) |= EXTI1_IRQ_BPOS;
    }
}

void enable_interrupt( void )
{
    /* Nollställ processorns avbrottsmask */
    __asm (
        "      cpsie i\n"      /* set I=0 */
    );
}
eller:
void enable_interrupt ( void ) __attribute__( ( naked ) );
void enable_interrupt( void )
{
    /* Nollställ processorns avbrottsmask */
    __asm (
        "      cpsie i\n"      /* set I=0 */
        "      bx      lr\n"
    );
}

void app_init ( void )
{
    /* Koppla PD1 till avbrottslina EXTI1 */
    *((unsigned int *) SYSCFG_EXTICR1) |= 0x0030;
    /* Konfigurera EXTI1 för att generera avbrott */
    *((unsigned int *) EXTI_IMR) |= EXTI1_IRQ_BPOS;
    /* Konfigurera för avbrott på negativ flank */
    *((unsigned int *) EXTI_FTSR) |= EXTI1_IRQ_BPOS;
    /* Sätt upp avbrottsvektor */
    *((void (**)(void) ) EXTI1_IRQVEC ) = irq_handler;
    /* Konfigurera den bit i NVIC som kontrollerar den avbrottslina som EXTI1 kopplats till */
    *((unsigned int *) NVIC_ISER0) |= NVIC_EXTI1_IRQ_BPOS;
}

```

Uppgift B-4:

```

typedef struct _usart
{
    volatile unsigned int sr;
    volatile unsigned int dr;      // +0x4
    volatile unsigned int brr;     // +0x8
    volatile unsigned int cr1;     // +0xc
    volatile unsigned int cr2;     // +0x10
    volatile unsigned int cr3;     // +0x14
    volatile unsigned int gtr;     // +0x18
} USART;

```

Uppgift C-5:

```

int notInString(const char* cs, const char* ct)
{
    int n;
    const char* p;
    for(n=0; *cs; cs++, n++)
    {
        for(p=ct; *p && (*p != *cs); p++)
            ;
        if (!*p)
            break;
    }
    return n;
}

```

Uppgift C-6

```

a)
typedef struct tObject { // tObject kan skippas
    char *pName;
    int age;
}

```

```
} Object ;
```

```
b) Object a = {"Sven", 23};
```

```
c) a.age++;
```

d)

```
/* Filen alarm.h */
extern void alarm_on(void);
extern void alarm_off(void);
extern void sensorinter(void);

/* Filen definitioner.h */
typedef unsigned char port;
typedef port *portptr;
typedef void (*vec) (void); // avbrottsvektor
typedef vec *vecptr; // pekare till // avbrottsvektor

#define SENSOR_ADR 0x500
#define SENSOR_CTRL (*(portptr) SENSOR_ADR)
#define SENSOR_ON 0x1
#define SOUND_ON 0x2
#define SENSOR_IE 0x4
#define SENSOR_DATA (*(portptr) (SENSOR_ADR+1))
#define BUTTON_PUSHED 0x80
#define SENSOR_VEC_ADR 0xFFE0
#define SENSOR_VEC (*(vecptr) SENSOR_VEC_ADR)
#define DISPLAY_ADR 0x600
#define DISPLAY (*(portptr) DISPLAY_ADR)

/* Filen assembler.h */
extern void sensortrap(void);

/* Filen alarm.c */
#include "alarm.h"
#include "definitioner.h"
#include "assembler.h"

void alarm_on(void) {
    SENSOR_CTRL = SENSOR_ON | SENSOR_IE;
}

extern void alarm_off() {
    SENSOR_CTRL = 0;
}

static port data = 0; // skuggregister
static int pushed = 0; // har knappen tryckts in en gång tidigare?

void sensorinter(void) {
    if (!(SENSOR_DATA & BUTTON_PUSHED)) { // avbrott från en sensor
        data = data | SENSOR_DATA;
        if (!pushed)
            SENSOR_CTRL = SENSOR_CTRL | SOUND_ON; // slå på ljudet
    }
    else { // avbrott från knappen
        if (!pushed) // första tryckning
            SENSOR_CTRL = SENSOR_CTRL & ~SOUND_ON; // stäng av ljudet
        else // andra tryckning
            data = 0; // återställ
        pushed = (pushed+1) % 2; // ändra tillstånd
    }
    SENSOR_DATA = 0; // kvittera avbrottet
    DISPLAY = data; // visa indikatorer
}
```