Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA482 Programmering av inbyggda system D
EDA487 Programmering av inbyggda system Z
DAT017 Maskinorienterad programmering IT
DIT151 Maskinorienterad programmering GU

Lördag 3 juni 2017, kl. 8.30 - 12.30

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:
Roger Johansson

Tillåtna hjälpmedel

Utgåvor som distribuerats inom ramen för kursen, häftena:

- *Instruktionslista för CPU12*
- *Quick Guide, Laborationsdator MD407 med tillbehör*

Inget annat än understrykningar (”överstrykningar”) får vara införda i dessa häften.

Du får också använda:

- *Rättelser till Quick Guide, Laborationsdator MD407 med tillbehör*
- *C Reference Card*

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 laborations-systemet (*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.Du kan alltså inte växla mellan A och B uppgifter. Antingen löser du enbart A uppgifter eller enbart B 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.

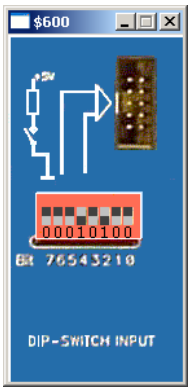
För full poäng krävs att:

- 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 getts under kursen.
- C-program är utformade enligt de råd och anvisningar som getts under kursen. I programtexterna skall raderna dras in så att man tydligt ser programmens struktur.

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

Tentamenspoäng ger slutbetyg enligt:
(EDA/DAT/LEU):
$20p \leq$ betyg 3 $< 30p \leq$ betyg 4 $< 40p \leq$ betyg 5
respektive (DIT):
$20p \leq$ betyg G $< 35p \leq$ VG

Uppgift A-1 (8p) *In/utmatning i C*



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

En 8-bitars strömbrytare är ansluten till adress 0x600 och en displayenhet som visar en *byte* i form av två hexadecimala siffror är ansluten till adress 0x400 i ett MC12 mikrodatorsystem. Konstruera en funktion

```
void ff1( void )
```

som oavbrutet 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.

Uppgift A-2 (8p) *Assemblerprogrammering*

En C-funktion definieras enligt följande:

```
int bitcount(unsigned char value)
{
   unsigned char count;
   for ( count=0; value != 0; value >>= 1);
   {
      if (value & 1)
         count++;
   }
   return count;
}
```

Skriv, i assemblerspråk för HCS12, en subrutin `_bitcount` som *motsvarar* C-funktionen. Du *behöver* alltså inte göra någon regelrätt översättning av C-funktionen utan bara betrakta den som en specifikation av vad assemblerrutinen ska utföra. För denna uppgift gäller alltså *inga* C-anropskonventioner utan följande specifikation ska gälla för din subrutin:

```
;  subrutin bitcount
;  bestämmer antalet ettor i 'value' med storleken 'byte'.
;  Indata:
;           register B innehåller 'value'
;  Returvärde:
;           register B innehåller antalet ettor i 'value'
```

Uppgift A-3 (8p) *Kodningskonventioner (C/assemblerspråk)*

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (bilaga 1).
Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

```
void  func( char *b, char a )
{
   char  c;
   char  *d; ....
```

dessutom har följande C-deklarationer gjorts på ”toppnivå” (global synlighet):

```
char     *aa,bb;
```

a) Visa hur variabeldeklarationerna på toppnivå översätts till assemblerdirektiv för HCS12.

b) Med variabeldeklarationerna i a), visa hur följande funktionsanrop översätts till assemblerkod för HCS12:

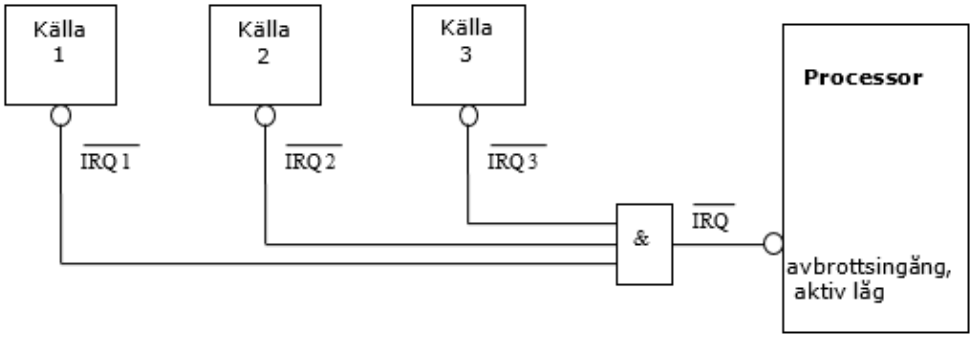
```
func( aa , bb );
```

c) Beskriv *aktiveringsposten*, dvs. stackens utseende i funktionen.
Visa tydligt riktningen för *minskande adresser* hos aktiveringsposten.

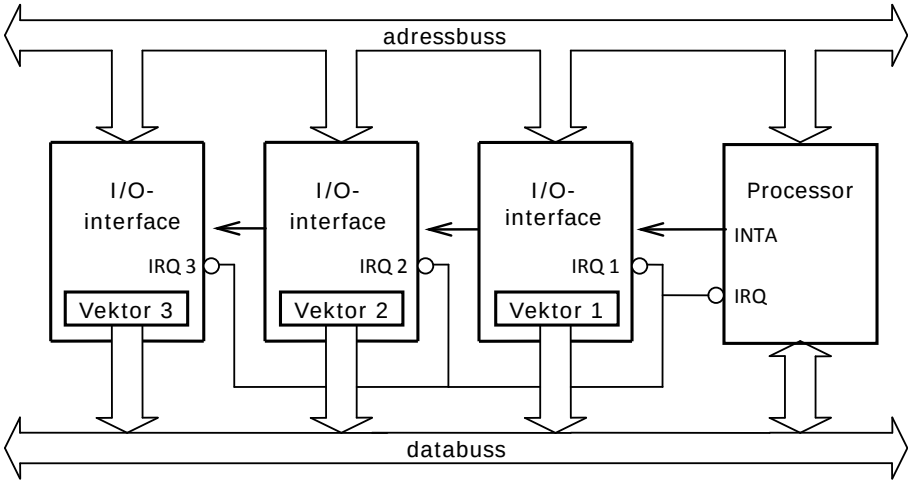
Uppgift A-4 (8p) *Avbrott*

I kursen har vi beskrivit två metoder för att identifiera och betjäna den enhet som begärt avbrott. Den ena metoden kallas "polling" och den andra vektorisering genom "daisy-chaining".

a) Förklara, utifrån följande figur, hur processorn dirigeras till rätt servicerutin vid "polling"! (4p)

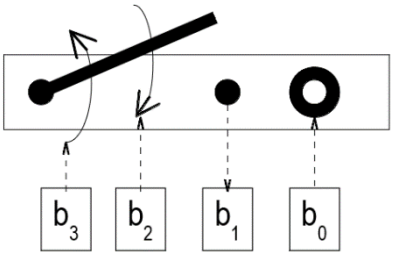


b) Förklara, utifrån följande figur hur processorn dirigeras till rätt servicerutin vid vektorisering (4p)



Uppgift B-1 (16p)

En ”sluss” ska konstrueras med hjälp av elektroniskt styrda dörrar. En dörr har ett 4-bitars digitalt gränssnitt (se figur till höger) som beskrivs av följande:



Funktioner för att öppna och stänga dörren är flank-triggade. En funktion för att ge larm genom att tända en varningslampa är nivå-triggad, dessutom finns en statussignal som anger om dörren är öppen eller stängd.

- En positiv flank på bit 3 öppnar dörren.
- En negativ flank på bit 2 stänger dörren.
- Bit 1 är 1 om dörren är stängd, annars är denna bit 0.
- Då bit 0 är 1 tänds en varningslampa som monterats vid dörren. Då bit 0 är 0 är denna varningslampa släckt.

I mekaniken finns en viss tröghet så det kan ta upp till 1 sekund att öppna eller stänga en dörr. Av denna anledning måste funktioner för realtidsfördröjning användas. Dessa ska implementeras med hjälp av SYSTICK.

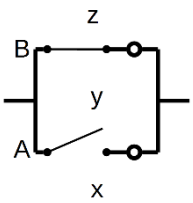
a) Använd SYSTICK för att konstruera en blockerande fördröjningsfunktion `void delay10ms(void);` som blockerar det anropande programmet i 10 ms. (3p)

b) Två dörrar, ska nu anslutas till en *MD407* via port D. Dörr 'A' ansluts till PD7-PD4, dörr 'B' ansluts till PD3-PD0.

Konstruera tre funktioner, `initdoor`, `opendoor` och `closedoor` enligt följande specifikationer:

`void initdoor( void );` initiera GPIOD för användning med dörrarna. PD8-PD15 ska ställas som inport.
`int opendoor(char c);` öppna dörr ('A' eller 'B', som parameter c), vänta maximalt 1 sekund i 10 ms intervall. Om dörren öppnas inom 1 sekund ska funktionen returnera 1, annars ska funktionen returnera 0.
`int closedoor(char c);` stäng dörr ('A' eller 'B', som parameter c), vänta maximalt 1 sekund i 10 ms intervall. Om dörren är stängd inom 1 sekund ska funktionen returnera 1, annars ska funktionen returnera 0.
Det är tillåtet att använda fördröjningsfunktionen `delay10ms()` även om du inte löst den uppgiften. (7p)

c) Utöver de båda dörrarna A och B placeras nu tre rörelsegivare i anslutning till dörrarna. Dessa betecknas x,y och z (se figur till höger) och fungerar enligt:



- x, kopplad till PD8: Bit 1 är 1 om någon person finns i utrymmet, annars 0.
- y, kopplad till PD9: Bit 1 är 1 om någon person finns i utrymmet, annars 0.
- z, kopplad till PD10: Bit 1 är 1 om någon person finns i utrymmet, annars 0.

Konstruera ett huvudprogram som tillåter passage från x till z. Applikationen ska:

1. Initiera systemet, öppna dörr A och stäng dörr B – förutsätt att inget fel uppstår här.
2. Så länge inget fel uppstår:

- kontrollera om det finns någon person i 'y' – i så fall, stäng dörr A, öppna dörr B, vänta tills utrymme ’y’ är tomt, stäng därefter dörr B och öppna dörr A.

Om något fel uppstår:

- en dörr öppnas eller stängs inte som den ska – ge larm vid båda dörrar och avbryt exekveringen av huvudprogrammet.

Det finns inga speciella prestandakrav, det räcker med att implementera funktionen enligt ovan. (6p)

Uppgift B-2 (7p)

Följande deklarationer är givna:

```
char *cp1, *cp2;
```

a) Koda följande funktionsanrop i ARM-v6 assemblerspråk.

```
swap( &cp1, &cp2 );
```

Kompilatorkonventioner för GCC ska användas (2p).

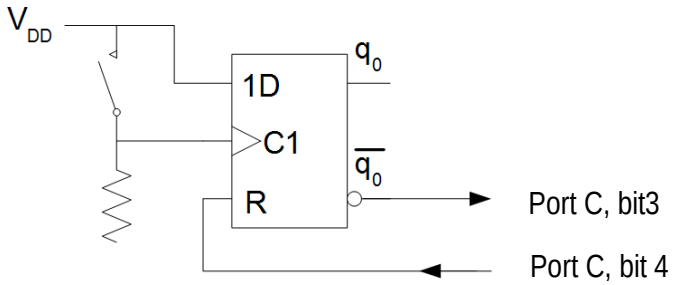
Funktionen `swap` skiftar två pekare i minnet, den definieras:

```
void swap( char **a, char **b )
{
   char *c;
   c = *b;
   *b = *a;
   *a = c;
}
```

b) Implementera motsvarande subrutin i ARM-v6 assemblerspråk (5p).

Uppgift B-3 (7p)

En avbrottsvippa kopplas till *MD407* enligt följande figur:



Den externa avbrottsmekanismen (EXTI) ska användas för att detektera knapptryckningar. Ingen hänsyn behöver tas till eventuella kontaktstudsar.

a) Visa med en funktion `app_init` hur avbrottsmekanismerna initieras, dvs. IO-pinnar konfigureras, EXTI och NVIC initieras. Du får förutsätta att inga andra IO-pinnar i port C ska användas. (4p)
b) Visa en komplett avbrottsrutin `irq_handler` som kvitterar (återställer) avbrott efter en knappnedtryckning så att systemet kan detektera nästa nedtryckning. Du får förutsätta att inga andra avbrott förekommer.(2p)
c) Var, i *MD407*, (vilken adress) ska adressen till avbrottsrutinen placeras? (1p)

För full poäng krävs att din lösning är tydlig, fullständig och att du använt lämpliga makrodefinitioner för registeradresser.

Uppgift B-4 (2p)

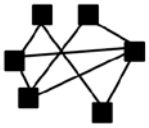
Visa hur en GPIO-modul kan beskrivas med en sammansatt datatyp `struct`, i C.

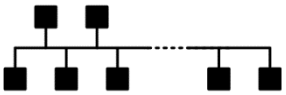
Uppgift C-5 (8p)

a) Visa med ett exempel hur du kan skapa en integer-variabel, `count`, som bara är åtkomlig inuti en funktion och som behåller sitt värde mellan funktionsanrop? (2p)

b) Visa en funktion `clear(…)` som nollställer en integer-variabel (dvs sätter den = 0) via en in-parameter som utgör ett referensanrop. (2p)

c) Vad kallas de nätverkstopologier som illustreras på följande sätt? (2p)





d) Ge två olika exempel på kommunikationsmedia vid seriekommunikation. (2p)

Uppgift C-6 (10p) *Maskinnära programmering i C*

Deluppgift a (5p)

Konstruera en C-funktion `timeout` som skall vänta tills antingen en viss tid gått eller tills en viss händelse inträffat. Funktionen skall ha två parametrar. Den första skall vara en pekare till en funktion som är parameterlös och som returnerar ett heltal. Detta heltal anger om händelsen man väntar på inträffat (1) eller inte (0). Den andra parametern anger time-out intervallet, uttryckt i millisekunder. Denna parameter skall vara av typen `long int`. Den andra parametern kan vara negativ och detta skall då tolkas som att time-out intervallet är oändligt långt. Som resultat skall funktionen `timeout` ge värdet 0 om den händelse den väntat på inträffat och värdet 1 om time-out intervallet löpt ut.

Du får förutsätta att det finns en *färdigskriven* modul `realtime` bestående av filerna `realtime.h` och `realtime.c`. I denna modul finns en privat (`static`) variabel `intnum` av typen `long int` som håller reda på antalet avbrott som kommit från en klockkrets ansluten till processorn. Klockkretsen ger ett avbrott var 5:e millisekund. Modulen har två funktioner som deklareras i filen `realtime.h`, funktionen `start_clock` som initierar avbrottsmekanismen och nollställer variablen `intnum` samt funktionen `get_intnum` som avläser och returnerar värdet i variabeln `intnum`. Funktionen `timeout` får förutsätta att funktionen `start_clock` har anropats tidigare.

Placera din funktion i en ny modul med namnet `timeout`. Skriv de filer som behövs för detta.

Deluppgift b (5p)

I ett hus skall ett inbrottsalarm installeras. En infravärmesensor som ger ett avbrott till datorn då en värmekälla detekteras skall användas. Sensorn har ett kombinerat 8-bitars status- och kontrollregister på adressen $A024_{16}$. Sensorn startas och stängs av genom att bit nr 7 i registret sätts till 1 resp. 0. När en värmekälla upptäcks sätter sensorn bit nr 6 i registret till 1. Om man har satt bit nr 1 i registret till 1 genererar sensorn även en avbrottssignal. Sensorn kan återställas efter ett alarm genom att man sätter bit 6 till 0. Avbrottsvektorn för sensorn ligger på adressen $FE20_{16}$.

Den andra deluppgiften är att i C skriva ett program som hanterar sensorn och som ger ett larm om sensorn indikerar en värmekälla. För att undvika falskt alarm skall programmet utformas så att det krävs två separata indikationer inom 15 sekunder från sensorn innan larm ges. Du får anta att det finns en färdigskriven funktion `raise_alarm` som man kan anropa för att ge larm.

Du får använda dig av funktionerna i modulerna `realtime` och `timeout` från a-uppgiften. (Det får du göra även om du inte löst den deluppgiften.) Din uppgift är alltså att skriva main-funktionen samt de funktioner som initierar sensorn och hanterar avbrott från den. För enkelhets skull får du anta att det finns en färdig avbrottsrutin med namnet `alarm_trap` som anropar en funktion med namnet `handle_alarm`. Du ska alltså själv skriva funktionen `handle_alarm`. Du kan vidare förutsätta att adressen till avbrottsrutinen (`alarm_trap`) är inlagd i avbrottsvektorn. Det behöver du därför inte göra själv.

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 assemblatorn, 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 Caracter 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”)

Lösningsförslag

Uppgift A-1:

```
typedef unsigned char *port8ptr;
#define ML4OUT_ADR 0x400
#define ML4IN_ADR 0x600
#define ML4OUT *((port8ptr) ML4OUT_ADR)
#define ML4IN *((port8ptr) ML4IN_ADR)

void    ff1( void )
{
        unsigned char pattern, bitpos;
        while( 1 )
        {
            pattern = ML4IN;

            if( ! pattern )
                bitpos = 0;
            else{
                for( bitpos = 1; bitpos < 8; bitpos++ )
                {
                    if( pattern & 1 )
                        break;
                    pattern >>= 1;
                }
            }
            ML4OUT = bitpos;
        }
}
```

Uppgift A-2:

```
_bitcount:
;  {
;  Registerallokering:
;      reg B: value
;      reg A: count
;       unsigned char count;
;
;       for ( count=0; value != 0; value >>= 1)
    CLRA          ; count=0
bitcount_2:
    TSTB          ; value != 0
    BNE    bitcount_4
    BRA    bitcount_5

bitcount_3:
    LSRB          ; value >>= 1
    BRA    bitcount_2

bitcount_4:
;     8 |    {
;     9 |       if (value & 01)
    BITB   #1
    BEQ    bitcount_3
;    10 |          count++;
    INCA
;    11 |    }
    BRA    bitcount_3

bitcount_5:
;    12 |
;    13 |    return count;
;    14 | }
    TFR    A,B
    RTS
```

Uppgift A-3:

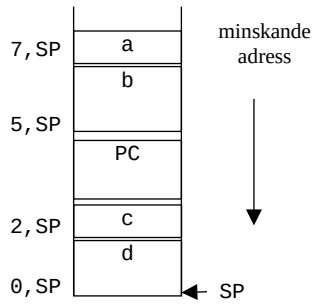
a)

```
_aa    RMB    2
_bb    RMB    1
```

b)

```
LDAB   _bb
PSHB
LDD    _aa
PSHD
JSR    _func
LEAS   3,SP
```

c)



Uppgift A-4:

a) När processorn upptäckt att den har en avbrottsbegäran så exekverar den tillståndssekvensen för avbrottshantering. I denna så läses avbrottsvektorn som i det normala CPU12-fallet ligger på adressen FFF2:FFF3H. Avbrottsvektorn leder till avbrottsrutinen. Vid tre avbrottskällor och "polling" så identifierar man den avbrottsbegärande källan i själva avbrottsrutinen. Genom statustest, dvs. undersökning av avbrottsindikatorer i de anslutna avbrottskällornas statusregister, dirigeras processorn till rätt servicerutin.

b) Vid vektorisering genom "daisy-chaining" så har varje källa sin egen avbrottsvektor. I tillståndssekvensen för avbrottshantering kommer processorn i verkligheten inte att läsa på adressen FFF2:FFF3H, utan istället från den källa som begärt avbrott. Det anordnas med en speciell CS-logik som förhindrar att CS=INTA (INTerrupt Acknowledge) når adressen FFF2:FFF3H och istället når den avbrottsbegärande källan enligt uppgifttextens figur.

Uppgift B-1:

a)
```
#define    STK_CTRL   ((volatile unsigned int *)(0xE000E010))
#define    STK_LOAD   ((volatile unsigned int *)(0xE000E014))
#define    STK_VAL    ((volatile unsigned int *)(0xE000E018))

void delay_10ms( void )
{
    /* SystemCoreClock = 168000000 */
    *STK_CTRL = 0;
    *STK_LOAD = ( (1680000) -1 );
    *STK_VAL = 0;
    *STK_CTRL = 5;
    while( (*SysTickCtrl & 0x10000 )== 0 );
    *STK_CTRL = 0;
}
```

b)
```
#define    GPIO_D_BASE    0x40020C00     /* MD407 port D */
#define    GPIO_D_MODER   ((volatile unsigned int *)    (GPIO_D_BASE))
#define    GPIO_D_OTYPER  ((volatile unsigned short *)  (GPIO_D_BASE+0x4))
#define    GPIO_D_PUPDR   ((volatile unsigned int *)    (GPIO_D_BASE+0xC))
#define    GPIO_D_IDR     ((volatile unsigned short *)  (GPIO_D_BASE+0x10))
#define    GPIO_D_ODR     ((volatile unsigned short *)  (GPIO_D_BASE+0x14))

void initdoor( void )
{
    *GPIO_D_MODER  = 0x00005252;
    *GPIO_D_OTYPER = 0;
    *GPIO_D_PUPDR  = 0;
}


int opendoor( char c )
{
    int i;
    if(c == 'A')
    {
        /* Skapa positiv flank på dörr A b3 */
        *GPIO_D_ODR = 0;
        *GPIO_D_ODR = (1<<7);
        for( i = 0; i < 100; i++ )
        {
            if( (*GPIO_D_IDR & (1<<5) )==0)
                return 1;  /* Dörr öppnad */
            delay_10ms();
        }
        return 0; /* Kunde inte öppna dörren */
    }else{
        /* Skapa positiv flank på dörr B b3 */
        *GPIO_D_ODR = 0;
        *GPIO_D_ODR = (1<<3);
        for( i = 0; i < 100; i++ )
        {
            if( (*GPIO_D_IDR & (1<<1) )==0)
                return 1;  /* Dörr öppnad */
            delay_10ms();
        }
        return 0; /* Kunde inte öppna dörren */
    }
}

int closedoor( char c )
{
    if(c == 'A')
    {
        /* Skapa negativ flank på dörr A b2 */
        *GPIO_D_ODR = (1<<6);
        *GPIO_D_ODR = 0;
        for( i = 0; i < 100; i++ )
        {
            if(*GPIO_D_IDR & (1<<5))
                return 1;  /* Dörr stängd */
            delay_10ms();
        }
        return 0; /* Kunde inte stänga dörren */

    }else{
        /* Skapa negativ flank på dörr B b2 */
        *GPIO_D_ODR = (1<<2);
        *GPIO_D_ODR = 0;
        for( i = 0; i < 100; i++ )
```

```
        {
            if(*GPIO_D_IDR & (1<<1))
                return 1;  /* Dörr stängd */
            delay_10ms();
        }
        return 0; /* Kunde inte stänga dörren */
    }
}
```

c)

```
void    main( void )
{
    initdoor();
    opendoor('A');
    closedoor('B');
    while( 1 )
    {
        if( *GPIO_D_IDR & (1<<9) )
        {   /* någon i utrymme 'y' */
            if ( ! closedoor( 'A' ) )
            {   /* Ge larm och avbryt */
                *GPIO_D_ODR = (1<<0)|(1<<4) ;
                return;
            }
            if ( ! opendoor( 'B' ) )
            {   /* Ge larm och avbryt */
                *GPIO_D_ODR = (1<<0)|(1<<4) ;
                return;
            }
            while ( *GPIO_D_IDR & (1<<9) ); /* Vänta tills utrymmet tomt */
            if ( ! closedoor( 'B' ) )
            {   /* Ge larm och avbryt */
                *GPIO_D_ODR = (1<<0)|(1<<4) ;
                return;
            }
            if ( ! opendoor( 'A' ) )
            {   /* Ge larm och avbryt */
                *GPIO_D_ODR = (1<<0)|(1<<4) ;
                return;
            }
        }
    }

}
```

Uppgift B-2:

a)

```
LDR    R0,=cp1
LDR    R1,=cp2
BL     swap
```

b)

```
@  void   swap( char **a, char **b )
@  char *c;
@  c = *b;
    LDR    R3,[R1]
@  *b = *a;
    LDR    R2,[R0]
    STR    R2,[R1]
@  *a = c;
    STR    R3,[R0]
    BX     LR
```

Uppgift B-3:

a)

```
#define NVIC_EXTI3_IRQ_BPOS   (1<<9)
#define EXTI3_IRQ_BPOS        (1<<3)

#define GPIOC_MODER    ((volatile unsigned int *)    0x40020800)
#define GPIOC_OTYPER   ((volatile unsigned short *)  0x40020804)
#define GPIOC_PUPDR    ((volatile unsigned int *)    0x4002080C)
#define GPIOC_ODRLOW   ((volatile unsigned char *)   0x40020814)
```

```
#define SYSCFG_EXTICR1 ((volatile unsigned short *)  0x40013808)
#define EXTI_IMR       ((volatile unsigned int *)    0x40013C00)
#define EXTI_RTSR      ((volatile unsigned int *)    0x40013C08)
#define EXTI_FTSR      ((volatile unsigned int *)    0x40013C0C)
#define EXTI_PR        ((volatile unsigned int *)    0x40013C14)

#define NVIC_ISER0     ((volatile unsigned int *)    0xE000E100)

void app_init( void )
{
    *GPIOC_MODER = 0x100;      /* Data direction digital io */
    *GPIOC_PUPDR = 0;          /* Input floating */
    *GPIOC_OTYPER = 0;         /* Output push/pull */

    *GPIOC_ODRLOW = 0x10;      /* RESET Flipflop */
    *GPIOC_ODRLOW = ~0x10;

    *SYSCFG_EXTICR) |= 0x2000;     /* PC3->EXTI3  */
    *EXTI_IMR |= EXTI3_IRQ_BPOS;
    *EXTI_FTSR |= EXTI3_IRQ_BPOS;
    *EXTI_RTSR &= ~EXTI3_IRQ_BPOS;

    *NVIC_ISER0 |= NVIC_EXTI3_IRQ_BPOS;
}
```

b)

```
void irq_handler ( void )
{
    *GPIOC_ODRLOW = 0x10;      /* RESET Flipflop */
    *GPIOC_ODRLOW = ~0x10;
    *EXTI_PR) |= EXTI3_IRQ_BPOS;
}
```

c)

```
Svar:    0x2001C064
```

Uppgift B-4:

```
typedef struct _gpio
{
volatile unsigned int moder;
volatile unsigned int otyper;     // +0x4
volatile unsigned int ospeedr;    // +0x8
volatile unsigned int pupdr;      // +0xc
volatile unsigned int idr;        // +0x10
volatile unsigned int odr;        // +0x14
volatile unsigned int bsrr;       // +0x18
volatile unsigned int lckr;       // +0x1C
volatile unsigned int afrl;       // +0x20
volatile unsigned int afrh;       // +0x24
} GPIO;
```

Uppgift C-5:

a)

```
void f( void )
{
    static int count;
    …
}
```

b)

```
void clear(int* p)
{
    *p = 0;
}
```

c)

“Mask” respektive “buss”

d)

Koppartråd, koaxialkabel, optofiber.

Uppgift 6a:

```
// I filen timeout.h
int timeout( int (*f)(void), long int max_time);

// I filen timeout.c
#include "realtime.h"
#include "timeout.h"
int timeout(int (*f)(void), long int max_time) {
  long int stop;
  stop = get_intnum() + max_time / 5;
  if (max_time > 0)
    while(!f() && get_intnum() < stop)
      ;
  else
    while(!f())
      ;
  return f();
}
```

Uppgift 6b:

```
// I filen alarm_ports.h
#define SENSORREG_ADR     0xA024
#define SENSORREG         *((portptr) SENSORREG_ADR)
#define sensor_on_bit        0x80
#define sensor_signal_bit    0x40
#define sensor_intr_bit      0x02
#define set(x, mask) (x) = (x) | (mask)
#define clear(x, mask) (x) = (x) & ~(mask)

// I filen alarm.c
#include "realtime.h"
#include "timeout.h"
#include "alarm_ports.h"
#include "alarm_inter.h"

void raise_alarm();
static int signal;

void init_alarm() {
  port shadow = 0;
  signal = 0;
  set(shadow, sensor_on_bit);
  set(shadow, sensor_intr_bit);
  SENSORREG = shadow;
}

void handle_alarm(void) {     // Anropas vid avbrott
  clear(SENSORREG, sensor_signal_bit);
  signal = 1;
}

int signal_on(void) {
  return signal;
}

int main() {
  start_clock();
  init_alarm();
  while (1) {
    if (signal) {   // första signal
      signal = 0;
      if (!timeout(signal_on, 15000)) { // vänta på andra signal
        signal = 0;
        raise_alarm();
      }
    }
  }
}
```