Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA482 (EDA481) Maskinorienterad programmering D
EDA487 (EDA486) Maskinorienterad programmering Z
DAT017 (DAT016) Maskinorienterad programmering IT
DIT151 Maskinorienterad programmering GU

Fredag 6 oktober 2017, kl. 8.30 - 12.30

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:
Roger Johansson, tel. 772 57 29
Ulf Assarsson, tel. 772 17 75

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.

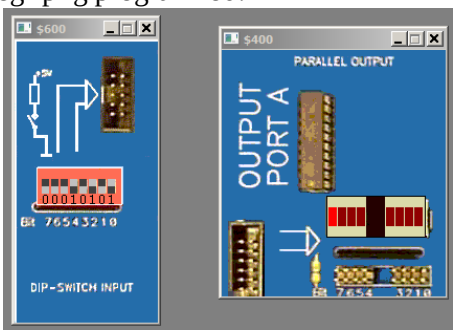
Betygsättning
För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända.

Maximal poäng är 50 oxh 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 typdeklarationer 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 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-2 (6p) *Assemblerprogrammering*

Vissa instruktionssekvenser kan inte åstadkommas med hjälp av giltiga standard-C satser. Exempel på detta är att påverka enskilda bitar i processorns statusregister (CCR). Se även bilaga 1.

a) Implementera en assembler subrutin som kan anropas från ett C-program.

```
unsigned char getCCR( void );
```

- returvärdet är innehållet i CCR.

b) Implementera en assembler subrutin som kan anropas från ett C-program.

```
void setCCR( unsigned char value );
```

- parameter `value` anger nya värden för bitarna i CCR.

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

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (se 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     *e,f;
```

a) Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12. (2p)

b) Visa också hur följande funktionsanrop översätts till assemblerkod för HCS12: (3p)

```
   func( e , f );
```

c) Visa hur utrymme för lokala variabler skapas och aktiveringspostens utseende (dvs. stacken) i funktionen '`func`'.

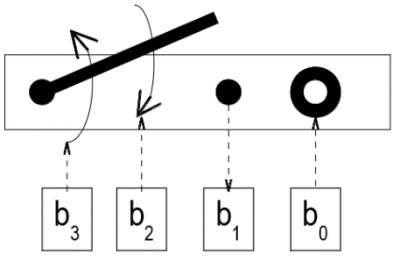
Uppgift A-4 (6p) *Avbrott*

Redogör detaljerat för hur avbrott går till i ett HC12-system, där du har en (endast en) yttre enhet ansluten till IRQ-ingången på HC12:an!

Ditt svar skall innehålla

- en steg för steg-beskrivning av hur ett avbrott går till, tips: rita gärna figur.
- en övergripande beskrivning av de olika programrutiner som är förknippade med avbrott (eventuellt pseudokod / assembler)

Uppgift B-1 (8p)



En elektroniskt styrd dörr ska konstrueras. Dörren 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) En dörr, ska nu anslutas till en *MD407* via port D. Portpinnarna PD3-PD0 ska användas.

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

`void initdoor( void );` initiera GPIOD för användning med dörren. Oanvända pinnar i porten ska ställas som ingångar.

`int opendoor(void);` öppna dörr, 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(void);` stäng dörr, 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. (5p)

Uppgift B-2 (6p)

Följande funktion `f` har kodats i ARM assemblerspråk med tillämpade kompilatorkonventioner:

```
f:      MOV     R1,#0
.L0:
        LDRB    R2,[R0]
        CMP     R2,#0
        BEQ     .L1
        ADD     R0,R0,#1
        ADD     R1,R1,#1
        B       .L0
.L1:
        MOV     R0,R1           @ return integer;
        BX      LR
```

Analysera assemblerkoden, koda en motsvarande funktion i C.

Uppgift B-3 (8p)

a) De globala variablerna `j` och `vecc` är definierade enligt: `int j; char vecc[80];` Visa en kodsekvens, i ARM assemblerspråk, som evaluerar uttrycket `vecc[j+1]` till register R0. (4p)

b) Utgå från att följande deklarationer är gjorda på global nivå.

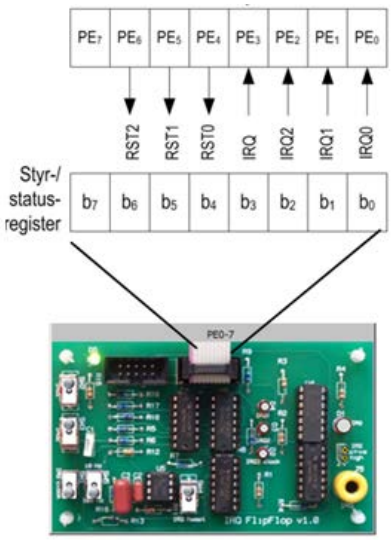
```
int     a,b,c;
```

Visa en kodsekvens, i ARM assemblerspråk, som evaluerar följande uttrycks värde till register R0. (4p)

```
(a + b * c ) <<1
```

Uppgift B-4 (6p)

Under laborationerna har du arbetat med ett laborationskort som genererar olika typer av avbrott, och kopplats till *MD407* enligt figuren till höger.



Den externa avbrottsmekanismen (EXTI) ska användas för att detektera ett godtyckligt avbrott (negativ flank) från laborationskortet. 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. (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)

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

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? (4p)

b) På vilket sätt är följande programkonstruktion problematisk? (2p)

```
void f( char *pcp )
{
   char *cp;
   *pcp = *cp;
}
```

c) Ange de, av följande rader som behandlas av *preprocessorn* i C? (2p)

```
   main()
   # include
   typedef
   # ifdef
```

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

Standardfunktionen **`strcat`** kan beskrivas på följande sätt:

```
char * strcat ( char * destination, const char * source );
```

Concatenate strings

Appends a copy of the *source* string to the *destination* string. The terminating null character in *destination* is overwritten by the first character of *source*, and a new null-character is appended at the end of the new string formed by the concatenation of both in *destination*.

Parameters
destination
Pointer to the destination array, which should contain a C string, and be large enough to contain the concatenated resulting string.
source
C string to be appended. This should not overlap *destination*.

Return Value

destination is returned.

Din uppgift är att, i C, skriva en egen definition av funktionen strcat. 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-7 (8p) *Maskinnära programmering i C*

Man har längs vissa vägsträckor utrustat hastighetsskyltarna med radiosändare som sänder ut information om den högsta tillåtna hastigheten. De fordon som passerar kan vara utrustade med radiomottagare som tar emot denna information. Den mottagna informationen lagras i ett register i mottagarna. Detta register kopplas till fordonets dator så att det kan avläsas från denna. Registret innehåller 16 bitar och har kopplats till den hexadecimala adressen 900 i datorn. Om en radiomottagare inte mottagit någon information den senaste minuten nollställs automatiskt detta register.

I ett fordon finns också en "intelligent" digital hastighetsmätare som har en display som på vanligt sätt visar den aktuella hastigheten. Till hastighetsmätaren hör två register vilka kan anslutas till datorn, ett styrregister på den hexadecimala adressen B04 och ett dataregister på den hexadecimala adressen B08. Båda dessa register består av 16 bitar. För att slå på ”intelligensen” i hastighetsmätaren skall man sätta bit nr 0 i styrregistret till 1 och för att slå på avbrottsmekanismen skall bit nr 6 sättas till 1. Bitarna 12-15 i styrregistret innehåller felinformation (felkoder). Om inget fel inträffat är dessa bitar alla 0. I hastighetsmätarens dataregister kan man lägga in den högsta tillåtna hastigheten. Om fordonets aktuella hastighet överstiger hastigheten i dataregistret genererar hastighetsmätaren automatiskt en avbrottssignal till datorn. En förutsättning för detta är dock att ”intelligensen” och avbrottsmekanismen slagits på. Avbrottet upprepas var tredje sekund tills hastigheten inte längre överstiger hastigheten i dataregistret. Hastighetsmätaren kan även generera avbrott om något fel uppstått.

Till datorn finns slutligen kopplat en krets som genererar en ljudsignal. Till denna krets hör ett 16-bitars styrregister på den hexadecimala adressen B60. Om man lägger en etta i bit nr 0 i detta register kommer en signal att ljuda under ett kort ögonblick (kortare än tre sekunder). Därefter tystnar signalen och bit nummer 0 återställs automatiskt till 0

Uppgiften är att i C skriva det program som behövs i fordonets dator, en funktion `void speed(void)`, för att samordna de kretsar som beskrivits ovan. Programmet skall var 10:e sekund avläsa den högsta tillåtna hastigheten från radiomottagaren. Om radiomottagaren mottagit information skall den högsta tillåtna hastigheten läggas i hastighetsmätarens dataregister och hastighetsmätaren skall initieras så att den kan generera avbrott. Om ingen information mottagits skall hastighetsmätaren inte kunna generera avbrott.

En avbrottsfunktion `void speed_trap()`ska också konstrueras. Om ett avbrott genererats av hastighetsmätaren och det inte är något fel skall ljudsignalen slås på. Om något fel uppstått man kan ignorera detta och helt enkelt nollställa de bitar som innehåller felkoden. Du får anta att avbrottsvektorn är korrekt initierad med adressen till en C-funktion `void speed_trap()`.

I denna uppgift får du förutsätta att en funktion `void hold(unsigned long int msDelay)`är färdig-skriven och kan användas. Den ger en fördröjning med så många millisekunder som dess parameter anger.

För full poäng på uppgiften ska du göra lämpliga symboliska definitioner, separera definitioner och kod i filerna ”`speed.h`” respektive ”`speed.c`”.

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 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-2:

a)
```
_getCCR:
          TFR    CCR,B
          RTS
```

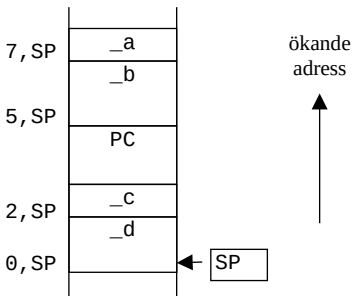
b)
```
_setCCR:

          LDAB   2,SP
          TFR    B,CCR
          RTS
```

Uppgift A-3:

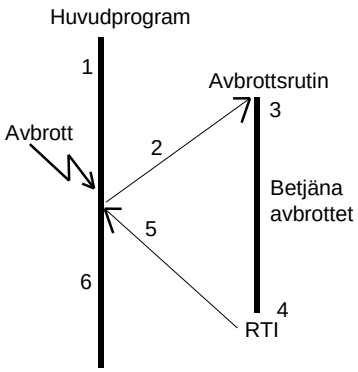
a)
```
_e  RMB 2
_f  RMB 1
```

b)
```
   LDAB   _f
   PSHB
   LDD    _e
   PSHD
   JSR    _func
   LEAS   3,SP
```



c)
```
   LEAS   -3,SP
```

Uppgift A-4:



Steg för steg beskrivning av händelseförloppet vid ett avbrott.

1. Processorn vars I-flagga är nollställd känner att IRQ är aktiverad.
2. Processorn sparar alla register på stacken (Save Status). I-flaggan ettställs. Därefter läser processorn startadressen för avbrottsrutinen från den aktuella avbrottsvektorn, denna startadress 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
       ldx    #IrqRutin          Initiera avbrottsvektor
       stx    $fff2
       cli                          Aktivera avbrottsingången
```

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

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  = 0x00000051;
    *GPIO_D_OTYPER = 0;
    *GPIO_D_PUPDR  = 0;
}


int opendoor( char c )
{
    int i;
    /* Skapa positiv flank på 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 )
{
    /* Skapa negativ flank 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 */
}
```

Uppgift B-2:

```
int f( char * str )
{  int rval = 0;
   while( *str++ ) rval++;
   return rval;
}
```

Uppgift B-3:

a)

```
    LDR    R0,j
    LDR    R1,=vecc
    LDRB   R0,[R0,R1]
```

b)

```
    LDR    R0,a
    LDR    R1,b
    LDR    R2,c
    MUL    R1,R2
    ADD    R0,R0,R1
    LSL    R0,R0,#1
```

Uppgift B-4:

a)

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

#define GPIOD_MODER    ((volatile unsigned int *)    0x40020C00)
#define GPIOD_OTYPER   ((volatile unsigned short *)  0x40020C04)
#define GPIOD_PUPDR    ((volatile unsigned int *)    0x40020C0C)
#define GPIOD_ODRLOW   ((volatile unsigned char *)   0x40020C14)

#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 )
{
    *GPIOD_MODER = 0x00001500;    /* Bit 4,5,6, digital ut, övriga digital in */
    *GPIOD_PUPDR = 0;             /* Nollställ bit 1: "Input floating" */
    *GPIOD_OTYPER = 0;            /* Nollställ bit 1: "Output push/pull" */

    *GPIOC_ODRLOW = 0x70;         /* Återställ alla vippor */
    *GPIOC_ODRLOW = ~0x70;

    *SYSCFG_EXTICR1 &= 0xF000;
    *SYSCFG_EXTICR1 |= 0x4000;          /* PE3->EXTI3  */
    *EXTI_IMR |= EXTI3_IRQ_BPOS;
    *EXTI_RTSR &= ~EXTI3_IRQ_BPOS;      /* EJ Avbrott på POSITIV flank */
    *EXTI_FTSR |= EXTI3_IRQ_BPOS;       /* Avbrott på negativ flank */
    *NVIC_ISER0 = NVIC_EXTI3_IRQ_BPOS;  /* Aktivera avbrott i NVIC */
}
```

b)

```
void irq_handler ( void )
{
    *GPIOD_ODRLOW = 0x70;         /* Återställ D-vippan */
    *GPIOD_ODRLOW = ~0x70;
    *EXTI_PR |= EXTI3_IRQ_BPOS;   /* Återställ EXTI1 "Pending Register" */
}
```

Uppgift C-5:

a)

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

b)

Eftersom pekaren `cp` är oinitierad kommer detta att resultera i en läsning från någon slumpmässig adress i minnet.

c)

```
    # include
    # ifdef
```

Uppgift C-6:

```
char *strcat(char *s1, const char *s2)
{
   char *save = s1;

   while (*s1 != 0)
      s1++;
   while (*s2 != 0)
      *s1++ = *s2++;
   *s1 = 0;
   return(save);
}
```

Uppgift C-7:

```
// Filen speed.h
    // hastighetsmätaren
    #define SPEED_REG   *((unsigned short int *) 0xB04)
    #define SPEED_DAT   *((unsigned short int *) 0xB08)
    #define ENABLE_BIT 0x1
    #define INTERRUPT_BIT 0x40
    #define ERROR_BITS 0xF000

    // mottagaren
    #define RECEIVER_DAT   *((unsigned short int *) 0x900)

    // alarm
    #define ALARM_DAT   *((unsigned short int *) 0xB60)
    #define ALARM_ON_BIT 0x1

// Filen speed.c
    #include "speed.h"

    void speed( void ) {
     short int avlast;
     while(1) {
       if ((avlast= RECEIVER_DAT) > 0) {
         SPEED_DAT = avlast;
         SPEED_REG = ENABLE_BIT | INTERRUPT_BIT;
       }
       else
         SPEED_REG = 0;
       hold(10000);
     }
     }

     void speed_trap() {
       if (SPEED_REG & ERROR_BITS)
         SPEED_REG = SPEED_REG & ~ERROR_BITS;
       else
         ALARM_DAT = ALARM_ON_BIT;
     }
```