Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

DAT017 Maskinorienterad programmering IT

DIT151 Maskinorienterad programmering GU

EDA481 Programmering av inbyggda system D

EDA486 Programmering av inbyggda system Z

DAT016 Programmering av inbyggda system IT

DIT152 Programmering av inbyggda system GU

Tisdag 10 januari 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 MOP*

Understrykningar (”överstrykningar”) får vara införda i dessa häften.
Instruktionslista för CPU12 får innehålla rättelser.
Quick Guide MOP får inte innehålla några egna anteckningar. Rättelser till *Quick Guide* finns som bilaga 3 i denna tes.

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. 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 (14p)

a) Redogör för vad som händer vid RESET och varför detta sker. (2p)

b) Förklara kortfattat vad som händer vid ett IRQ avbrott om I-flaggan i CC är nollställd. (2p)

c) Vid IRQ-avbrott sätts I-flaggan automatiskt till 1. Varför sker detta? (1p)

d) Visa med en instruktionssekvens hur man i en IRQ-avbrottsrutin kan förhindra att processorn utför nya avbrott efter återhopp till det avbrutna programmet. (3p)

För resterande deluppgifter gäller följande beskrivning av CRG-kretsen hos MC12:

Adress	Mnemonic	Namn
$34	SYNR	Synthesizer Register
$35	REFDV	Reference Divide Register
$37	CRGFLG	Flags Register
$38	CRGINT	Interrupt Enable Register
$39	CLKSEL	Clock Select Register

e) Visa hur du, i C, kan representera CRG-kretsens första register så att exempelvis satsen: `*SYNR = 0x32;` är syntaktisk korrekt. (2p)

f) Visa hur kretsen kan representeras med en sammansatt datatyp `struct.` Visa också hur du sedan definierar en pekare till en CRG-krets som i stället har basadress 0x200. (4p)

Uppgift A-2 (6p) *Assemblerprogrammering*

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (bilaga 1). Följande deklarationer (data och funktioner) är givna i "C". Implementera motsvarande deklarationer i assemblerspråk för HC12.

```
int max( short a, short b)
{
   if( a >= b )
      return a;
   else
      return b;
}
```

```
short c,d,e;

void  callmax(void )
{
   e = max( c,d);

}
```

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

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (bilaga 1).
Funktionen `f` (nedan) hanterar några lokala variabler men utför dessutom en rad beräkningar där globala variabler ingår (visas ej här), man kan därför inte göra några antaganden om innehållet i de lokala variablerna vid retur-satsen.

```
int   f( void )
{
char  a,b;
char  *c;

   c = (char *) 0xFF32;
   a = 'a';
   b = 0x32;
/*
   .. diverse kod som inte är relevant för uppgiften
*/
   return ( a + b + *c );
}
```

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

b) Översätt funktionen till assemblerkod för HC12.

Uppgift B-1 (14p)

En periferienhet med ett 8 bitars gränssnitt ska anslutas till ett MD407-system.

Periferienhet								
bit	7	6	5	4	3	2	1	0
access	write	read	read	read	read	read	read	read
namn	S	D	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0

Från RESET är bit D 0 och VAL-bitarna odefinierade.

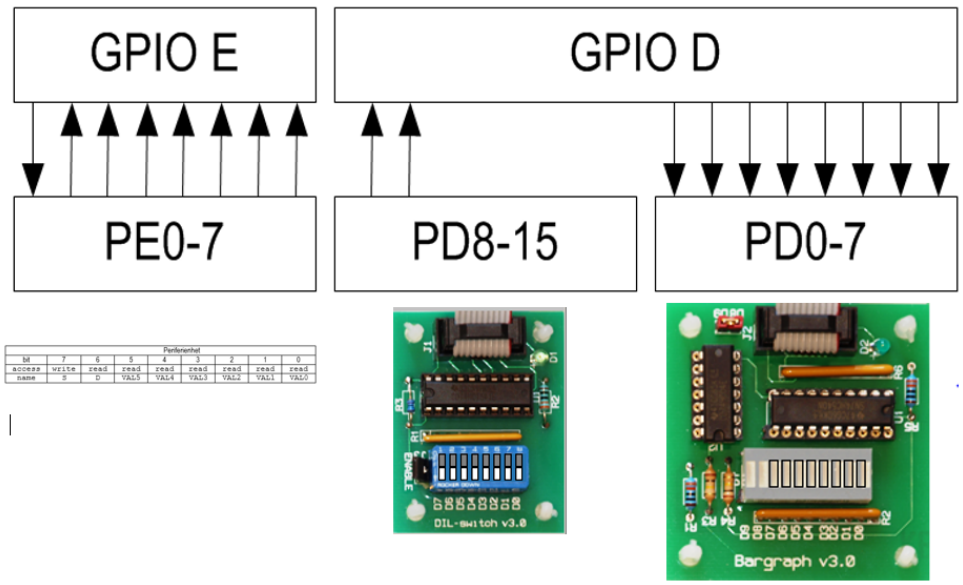
Periferienheten styrs och fungerar på följande sätt:

- Programvaran sätter bit S till 1 för att starta operationen. Då resultatet är färdigt sätts bit D till 1 av periferienheten. Nu är värdet VAL giltigt och kan läsas av från dessa bitar i gränssnittet. Ett giltigt värde är alltid skilt från 0, observera dock att då D är 0 kan VAL innehålla ett värde skilt från 0 trots att detta är ogiltigt.
- Programvaran ska nu läsa av värdet val.
- Slutligen återställs periferienheten genom att bit S sätts till 0 av programvaran. Periferikretsen nollställer då bit D vilket indikerar att värdet VAL inte längre är giltigt.
- Periferikretsen är nu klar för nästa operation.

Normalt produceras resultatet av periferikretsen mycket fort, på kortare tid än 1ms. Vid enstaka tillfällen kan det ta betydligt längre tid, upp till 500 ms. Som specialfall gäller att någon gång misslyckas periferikretsen, biten D kommer då aldrig att bli 1, ev. tidigare indata från periferikretsen ska då också betraktas som ogiltigt. Konstruera en applikation som kontinuerligt läser ett giltigt värde från periferikretsen och skriver detta till en diodramp. Dessutom ska 2 bitar läsas från en strömställare och skrivas till diodrampen. Port E (0-7) ska användas för periferikretsen, medan port D (bit 15 och bit 14) ska användas för strömställare och port D (0-7) till diodrampen. Eftersom de två bitarna från dipswitchen ska uppdateras kontinuerligt utan onödig fördröjning måste någon form av *timeout*-funktion konstrueras.

Dela upp applikationen i följande dellösningar:

a) En initieringsfunktion `init_app`, där du visar hur portarna D och E ska initieras, alla utgångar ska vara *push-pull*, alla ingångar ska vara *pull-down*. (2p)



b) Använd SYSTICK för att implementera en icke blockerande fungerande fördröjning, utgå från att systemets klockfrekvens är 168 MHz, dvs. 1 mikrosekund är 168 räknade klockpulser. Använd avbrottsperioden 1 ms. *Ledning*: Under laboration 4.1 konstruerade du en lösning som kan vara lämplig att använda här. (6p)

(uppgiften fortsätter på nästa sida)

c) Skriv huvudprogrammet (6p), följande algoritm kan användas:

```
algoritm:  main
    variabler: c, peripheral_input;
    init_app;
    peripheral_input = 0;
    repetera:
        Om bit D==0
            S=1; // initiera periferikretsen för omvandling
            starta timeout;
        c = dipswitch, bitar 7 och 6;
        Om bit D==1     // Periferikrets klar...
            peripheral_input = VALbitar;
            avbryt timeout;
        annars
            om timeout       // invalidera indata */
                peripheral_input = 0;
                S=0; // avbryt omvandling
        skriv ut:  c | peripheral_input ;
```

Uppgift B-2 (8p)

I denna uppgift ska du förutsätta samma konventioner som i GCC för ARM. Följande deklarationer (data och funktioner) är givna i "C". Implementera motsvarande deklarationer i assemblerspråk för ARM-v6.

```
int max( short a, short b)
{
   if( a >= b )
      return a;
   else
      return b;
}

short c,d,e;

void  callmax(void )
{
   e = max( c,d);

}
```

Uppgift B-3 (6p)

Förbered en enkel applikation som använder PD15 hos MD407 som avbrottsingång. Dvs, skriv, i C, en sekvens som:

- Kopplar PD15 till EXTI15
- Konfigurerar EXTI15 generera avbrott på negativ flank
- Konfigurerar NVIC.

Ange också den vektor som ska initieras för avbrottet.

Du kan förutsätta att alla moduler startats och behöver inte ta hänsyn till klockor (RCC). Observera dock att andra eventuella EXTI-konfigurationer inte får ändras av din programsekvens.

Uppgift C-4 (8p) *C-programmering*

Konstruera en C-funktion som undersöker en parameter med avseende på antalet 1-ställda bitar.
Funktionen deklareras:

```
int  bitcheck( unsigned int *pp, int * num );
```

- `pp` är en pekare till det värde som ska undersökas
- `num` är en pekare till en plats för returvärde, dvs. antalet 1-ställda bitar hos parametern

Funktionen ska returnera 1 om antalet ettor hos parametern är udda, annars ska returvärdet vara 0.

Uppgift C-5 (14p) *Maskinnära programmering i C*

Antag att en dator används för enkel tidtagning vid en idrottstävling. Till datorn finns kopplat två sensorer samt en klockkrets. (Dessutom finns en sifferindikator, men den behöver inte programmeras i denna uppgift.) Den första sensorn känner när startskottet går och den andra när den tävlande passerar mållinjen. De två sensorerna är kopplade till *samma* 16-bitars styrregister. Detta ligger på adressen 1234 (hex). Styrregistret aktiveras och inaktiveras genom att bit nr 0 i det sätts till 1 resp. 0. Om registret är inaktiverat påverkas det inte av inkommande signaler, men om det är aktiverat gäller följande: När en signal kommer från någon av de två sensorerna sätts bit nr 7 i registret till 1. Om man har satt bit nr 6 i registret till 1 genereras då även en avbrottssignal. Styrregistret skall återställas efter ett avbrott genom att man sätter bit 7 till 0.

Klockkretsen är kopplad till ett annat 16-bitars styrregister, vilket ligger på adressen 1230 (hex). Detta styrregister har samma konfiguration och fungerar på samma sätt som styrregistret för sensorerna. Den enda skillnaden är de inkommande signalerna kommer från klockkretsen istället för sensorerna. Klockkretsen genererar 1000 signaler per sekund.

Uppgiften är att skriva ett C-program som gör en tidsmätning. När programmet startar skall det visa tiden 0 på en display och vänta tills startskottet går. När detta sker skall klockan aktiveras och tiden skall visas fortlöpande på displayen. Displayen skall visa tiden uttryckt i hundradels sekunder och den visade tiden skall uppdateras hundra gånger per sekund. När den tävlande passerar mållinjen skall klockan stoppas och sluttiden visas konstant på displayen. Programmet behöver bara klara en tidsmätning. (Vill man göra en ny får man starta om programmet genom att trycka på reset-knappen.) Du får anta att tiden för instruktionsexekvering är försumbar.

Du får förutsätta att det finns en färdigskriven C-funktion `void display( unsigned long )`. När den anropas visar den parameterns värde på en sifferindikator.

Du får också förutsätta att avbrottsmekanismer är initierade och att en funktion `void clocktrap(void)` anropas vid avbrott från klockan samt en funktion `void sensortrap(void)` anropas vid avbrott från någon sensor, du måste dock skriva dessa funktioner.

Skriv programmet i C. För full poäng ska du strukturera din lösning, visa hur olika delar lämpligen placeras i olika filer och använda lämpliga definitioner av typer och makron.

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

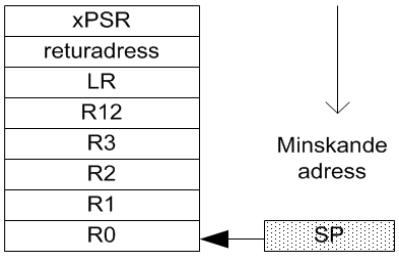
Bilaga 2: Assemblerdirektiv för MC68HC12.

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: Rättelser, tillägg till ”Quick Guide för MOP”

Quick guiden ska kompletteras med följande figur:



Stackens utseende vid undantagshantering

Quick guiden ska kompletteras med följande tabell:
Assemblerdirektiv:

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”)

Sidan 10: växlat BCC och BCS, ska vara:

C-operator	Betydelse	Datatyp	Instruktion
==	Lika	signed/unsigned	BEQ
!=	Skild från	signed/unsigned	BNE
<	Mindre än	signed	BLT
		unsigned	BCC
<=	Mindre än eller lika	signed	BLE
		unsigned	BLS
>	Större än	signed	BGT
		unsigned	BHI
>=	Större än eller lika	signed	BGE
		unsigned	BCS

Sidan 12, fel basadresser för USART, ska vara:

USART
Universal synchronous asynchronous receiver transmitter
USART1: 0x40011000
USART2: 0x40004400

Sidan 17, beskrivning av EXTI_PR, ska vara:

EXTI_PR *Pending Register*

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x14										rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_PR

Bit PR[22..0]:
Motsvarande bit sätts i detta register då ett triggvillkor är uppfyllt. Biten återställs genom att skrivas med '1'.
0: Ingen Trigg.
1: Trigg har uppträtt

Lösningsförslag

Uppgift A-1:

a) RESET är en asynkron signal till processorn med syftet att återställa processorns i ett väldefinierat bygynnelsetillstånd.

b) Processorn sparar alla registerinnehåll på stacken, läser adressen till avbrottshanteringsrutinen från avbrottsvektorn och placerar denna i PC.

c) För att undvika en rekursiv behandling av samma avbrott.

d) Modifiera det CC-innehåll som återställs från stacken vid RTI, dvs:

```
LDAB   0,SP
ORAB   #$10
STAB   0,SP
```

e)

```
#define   SYNR   ((volatile unsigned char *) 0x34)
```

f)

```
struct crg{
   volatile unsigned char synr;
   volatile unsigned char refdv;
   volatile unsigned char UNUSED;
   volatile unsigned char crgflg;
   volatile unsigned char crgint;
};
#define CRG_BASE   ((struct crg *) 0x200)
```

Uppgift A-2:

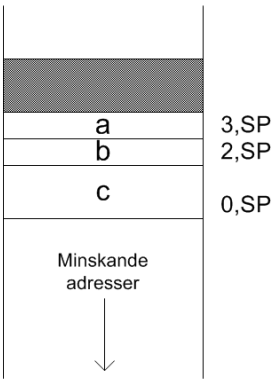
```
_c  RMB    2
_d  RMB    2
_e  RMB    2

max:
    LDD    2,SP
    CMPD   4,SP
    BGE    max_exit:
    LDD    4,SP
max_exit:
    RTS


callmax:
    LDD    _d
    PSHD
    LDD    _c
    PSHD
    JSR    callmax
    LEAS   4,SP
    STD    _e
```

Uppgift A-3:

a)



b)

```
f:  LEAS    -4,SP
    LDD     #$FF32
    STD     0,SP
    LDAB    #'a'
    STAB    3,SP
    LDAB    #$32
    STAB    2,SP
    ,,,
    LDX     0,SP
    LDAB    ,X
    ADDB    2,SP
    ADDB    3,SP
    SEX     B,D
    LEAS    4,SP
    RTS
```

Uppgift B-1:

```
#define    STK_CTRL          ((volatile unsigned int *)    0xE000E010 )
#define    STK_LOAD          ((volatile unsigned int *)    0xE000E014 )
#define    STK_VAL           ((volatile unsigned int *)    0xE000E018 )
#define    PORT_D_BASE    0x40020C00
#define    GPIO_D_MODER      ((volatile unsigned int *)    (PORT_D_BASE))
#define    GPIO_D_OTYPER     ((volatile unsigned short *)  (PORT_D_BASE+0x4))
#define    GPIO_D_PUPDR      ((volatile unsigned int *)    (PORT_D_BASE+0xC))
#define    GPIO_D_HIGH_IDR   ((volatile unsigned char *)   (PORT_D_BASE+0x11))
#define    GPIO_D_LOW_ODR    ((volatile unsigned char *)   (PORT_D_BASE+0x14))
#define    PORT_E_BASE    0x40021000
#define    GPIO_E_MODER      ((volatile unsigned int *)    (PORT_E_BASE))
#define    GPIO_E_OTYPER     ((volatile unsigned short *)  (PORT_E_BASE+0x4))
#define    GPIO_E_PUPDR      ((volatile unsigned int *)    (PORT_E_BASE+0xC))
#define    GPIO_E_LOW_IDR    ((volatile unsigned char *)   (PORT_E_BASE+0x10))
```

a)

```
void init_app( void )
{
    /*  PORT D, b15-8  ingångar, b7-b0 utgångar  */
    *GPIO_D_MODER  = 0x00005555;
    *GPIO_D_PUPDR  = 0xAAAA0000;      /* pull down */
    *GPIO_D_OTYPER  = 0x00000000;     /* push/pull */
    /*  PORT E, b6-0 ingångar, b7 utgång  */
    *GPIO_E_MODER  = 0x4000;
    *GPIO_E_PUPDR  = 0x2AAA;      /* b6-0 pull down */
    *GPIO_E_OTYPER  = 0x0000;     /* b7 push/pull */
}
```

b)

```
static int irq_count;
void start_1ms_delay( void )
{
    *STK_CTRL = 0;
    *STK_LOAD = ( 168000 - 1 );   /* 1 ms */
    *STK_VAL = 0;
    *STK_CTRL = 7;
}
void   systick_irq_handler ( void )
{
    *STK_CTRL = 0;
    irq_count -- ;
    if( irq_count > 0 )
        start_1ms_delay();
}
void start_timeout( void )
{
    *((void (**)(void) ) 0x2001C03C ) = systick_irq_handler;
    irq_count = 500;
    start_1ms_delay();
}
void stop_timeout( void )
{
    irq_count = 500;
    *STK_CTRL = 0;
}
```

c)

```
void main(void)
{
    unsigned char c, peripheral_input;
    init_app();
    peripheral_input = 0;
    while(1)
    {
        if( (*GPIO_E_LOW_IDR & 0x40 ) ==0)
        {   /* initiera periferikrets... */
            *GPIO_E_LOW_ODR = 0x80;   /* S = 1 */
            start_timeout();
        }
        c = *GPIO_D_HIGH_IDR & 0xC0;  /* bit 7 och 6 */
        if( *GPIO_E_LOW_IDR & 0x40 )
        {   /* Periferikrets klar... */
            peripheral_input = *GPIO_E_LOW_IDR & 0x3F;
            stop_timeout();
        }else{
            if( irq_count == 0 )
            {   /* invalidera indata */
                peripheral_input = 0;
                *GPIO_E_LOW_ODR = 0;  /* S = 0 */
                *GPIO_E_LOW_IDR = 0;  /* VAL = 0 */
            }
        }
        GPIO_D_LOW_ODR = c | peripheral_input ;
    }
```

```
}
```

Uppgift B-2:

```
    .thumb_func
max:
**** int max( short a, short b)
**** {
****    if( a >= b )
****        return a;
****    else
****        return b;
    mov     r2,r1
****    if( a >= b )
    cmp     r0,r1
    blt     max_exit
****    else
    movs    r2,r0
max_exit:
**** }
    movs    r0,r2
    bx      lr


callmax:
**** short c,d,e;
****
void    callmax(void )
**** {
    push    { lr}
****    e = bmax( c,d);
    ldr     r0,=c
    ldrh    r0,[r0]
    sxth    r0,r0
    ldr     r1,=d
    ldrh    r1,[r1]
    sxth    r1,r1
    bl      max
    ldr     r3,=e
    strh    r0,[r3]
    pop     {pc}

    .align
c:  .hword  1
d:  .hword  2
e:  .hword  3
```

Uppgift B-3:

```
    *((unsigned int *) SYSCFG_EXTICR4) &= 0x0FFF;   /* nollställ bitfält EXTI15 */
    *((unsigned int *) SYSCFG_EXTICR4) |= 0x3000;   /* PD15->EXTI15  */
    *((unsigned int *) EXTI_IMR) |= (1<<15);        /* aktivera avbrott EXTI15 */
    *((unsigned int *) EXTI_FTSR) |= (1<<15);       /* aktivera trigger på negativ flank */
    *((unsigned int *) EXTI_RTSR) &= ~(1<<15);      /* deaktivera trigger på positiv flank */
    *((unsigned int *) NVIC_ISER1) |= (1<<8);       /* aktivera avbrott i NVIC */
```

Vektor nummer 40 (offset 0xE0)

Uppgift C-4:

```
int bitcheck( unsigned int *pp, int *num)
{
    *num = 0;
    while(*pp)
    {
        if( *pp & 1 ) *num++;
        *pp >>= 1;
    }
    return *num & 1;
}
```

Uppgift C-5

```
/* I fil xxx.h */
typedef unsigned short port;
typedef port *portptr;
#define set(r, mask)    (r) = (r) | (mask);
#define clear(r, mask)  (r) = (r) & ~(mask);
/* Klockregistret */
#define CLOCKREG_ADR     0x1230
#define CLOCKREG         *((portptr) CLOCKREG_ADR)
/* Sensorerregistret */
#define SENSORREG_ADR     0x1234
#define SENSORREG         *((portptr) SENSORREG_ADR)

#define enable_bit        0x01
#define intr_bit          0x40
#define done_bit          0x80

void display(long);
void sensortrap(void);
void clocktrap(void);

/* I filen xxx.c */

#include "xxx.h"
static unsigned long int tick = 0;
static volatile int started;
static volatile int stopped;

void init_clock(void) {
  tick = 0;
  started = 0;
  stopped = 0;
  set( CLOCKREG, intr_bit);
}

void clockinter(void) {
  clear( CLOCKREG, done_bit);
  tick++;
}

void init_sensor(void) {
  set( SENSORREG, enable_bit | intr_bit);
}

void sensorinter(void) {
  clear(SENSORREG, done_bit);
  if (!started) {
    set(CLOCKREG, enable_bit);
    started = 1;
  }
  else {
    clear(CLOCKREG, enable_bit);
    stopped = 1;
  }
}

int main() {
  unsigned long int next;
  init_clock();
  init_sensor();
  display(0);
  while (!started)
    ;
  while(!stopped) {
    display(tick / 10);
    /* vänta 0.01 sek */
    next = tick + 10;
    while(tick < next && !stopped)
      ;
  }
  display(tick / 10);
}
```