Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

LEU500 Maskinorienterad programmering

Måndag 13 mars 2017, kl. 14.00 - 18.00

Examinator
Lars Bengtsson tel. 772 8441

Kontaktperson under tentamen:
Ulf Assarsson tel. 772 17 75

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 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 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:

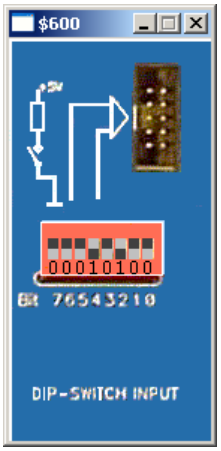
Adress	Mnemonic	Namn
$34	SYNR	Synthesizer Register
$35	REFDV	Reference Divide Register
$37	CRGFLG	Flags Register
$38	CRGINT	Interrupt Enable Register
$39	CLKSEL	Clock Select Register

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 (8p) *Programmering i C*

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



Konstruera en funktion **`void`** `FindFirst1(` **`void`** `)` 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 är ettställda på strömbrytaren ska positionen för bit 2, (dvs. 3) skrivas till displayenheten (se figuren). Om ingen strömbrytare är ettställd ska siffran 0 skrivas till displayen.

Uppgift A-3 (8p) *Assemblerprogrammering*

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

a) (5p) Följande funktion finns given i “C”. Skriv motsvarande funktion i assemblerspråk för HC12.

```
void  f1( unsigned char c )
{
   *( unsigned char *) 0x600 = c ;
   delay();
   c = c >> 1;
   *( unsigned char *) 0x600 = c ;
}
```

b) (3p) Följande C-deklarationer har gjorts på ”toppnivå” (global synlighet).

```
char    *cp;
char    *identify( char **cp);
```

visa hur följande funktionsanrop kan översättas till assemblerspråk för HC12

```
cp = identify( &cp );
```

Uppgift B-1 (14p)

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

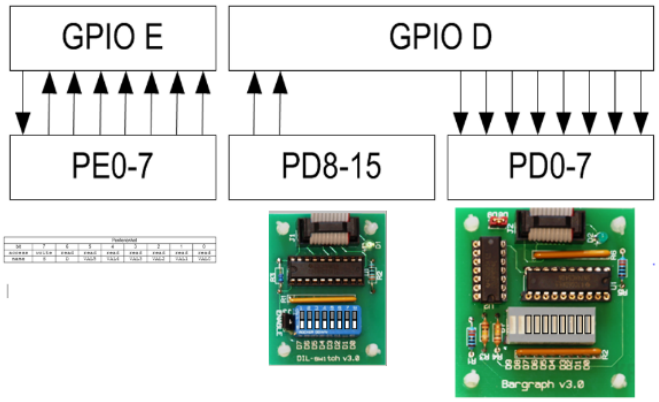
Periferienhet								
bit	7	6	5	4	3	2	1	0
access	read/ write	read	read	read	read	read	read	read
namn	S	D	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0

Från RESET är bit S=0, medan bit D och VAL-bitarna är odefinierade.

Periferienheten styrs och fungerar på följande sätt:

- Programvaran sätter bit S till 1 för att starta operationen. Kretsen behöver nu 1ms innan statusbiten D är giltig. 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.

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 strömställaren ska uppdateras kontinuerligt utan onödig fördröjning kan inte fördröjningsfunktionen utformas som en blockerande funktion.



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*. IO-pinnar som inte används kan anses vara odefinierade. Dessutom ska samtliga register (portar) som avänds i denna uppgift här deklareras med typkonvertering så som vi rekommenderat i kursen, dvs. ”`#define GPIO_D_MODER …`” etc. (4p)

b) Använd SYSTICK för att implementera en icke blockerande fördröjning, dvs. en funktion som returnerar *true* om 1 ms passerat annars *false*. Utgå från att systemets klockfrekvens är 168 MHz, dvs. 1 mikrosekund är 168 räknade klockpulser. Använd avbrottsperioden 1 ms. *Ledning*: Under laboration 4.1 (”meddelandeskickning”) konstruerade du en lösning som kan användas här. (5p)

c) Skriv huvudprogrammet (5p), följande algoritm kan användas:

```
algoritm:  main
      variabler: c, peripheral_input;
      init_app;
      peripheral_input = 0;

repetera:
            Om bit S==0
                  S=1;
                  starta 1ms fördröjning;
            c = dipswitch, bitar 7 och 6;
            Om bit D är giltig, dvs. 1 ms passerat, och bit D==1
                  peripheral_input = VALbitar;
                  S=0;
            skriv ut:  c | peripheral_input ;
```

Uppgift B-2 (10p)

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 min( signed char a, signed char b)
{
   if( a < b )
      return a;
   else
      return b;
}

signed char c,d,e;

void  callmin(void )
{
   e = min( c, d);

}
```

Ledning: Under "Kompilatorkonventioner" i "Quick Guide" framgår hur parametrar och returvärden (resultat) ska placeras i register. Tänk också på att typkonvertering ska ske före funktionsanropet.

Uppgift B-3 (6p)

Förbered en enkel applikation som använder PD10 hos MD407 som avbrottsingång. Dvs, skriv, i C, en sekvens som:

- Kopplar PD10 till EXTI10
- Konfigurerar EXTI10 för att generera avbrott på *negativ* flank
- Konfigurerar NVIC.

Ange också offseten i vektortabellen för 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- eller NVIC- konfigurationer *inte* får ändras av din programsekvens. För full poäng ska du visa hur preprocessordirektiv och ev. typdeklarationer används för att skapa begriplig programkod. Typkonverteringar ska göras på sådant sätt som rekommenderats i kursen.

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

Konstruera en C-funktion som undersöker en parameter med avseende på antalet 0-ställda bitar.
Funktionen deklareras:

```
int  bitcheck( unsigned int *p, long * num );
```

- `p` är en pekare till det värde som ska undersökas
- `num` är en pekare till en plats dit antalet 0-ställda bitar hos parametern `p`, ska skrivas

Returvärdet för funktionen ska vara ett värde skilt från 1 om antalet nollor hos parametern är jämt delbart med 4, annars ska returvärdet vara 0.

Uppgift C-5 (12p) *Maskinnära programmering i C*

En robotarm styrs av en dator via fem 8-bitars register: ett styr-/status- register på adressen 0x0800, samt två dataregister på adresserna 0x0801 respektive 0x0802. Styrregistret används för att kontrollera robotarmens rörelser och dataregistren används för att ange x- respektive y-koordinater som mål vid robotarmens förflyttning. Dessutom finns ytterligare två positionsregister på adresserna 0x0803 och 0x0804 som anger de aktuella x- respektive y-koordinaterna för robotarmen. Dessa register är endast läsbara, medan övriga register är både läs- och skrivbara. Följande tabell beskriver registren i robotens gränssnitt:

ROBOT											
Adress		7	6	5	4	3	2	1	0	Mnemonic	Namn
0x0800	*)	IE	ACT		IACK			ERR	IRQ	ctrl	Styr-/status- register
0x0801	R/W	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	datax	Data X
0x0802	R/W	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	datay	Data Y
0x0803	R	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	posx	Position X
0x0804	R	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	posy	Position Y

*) IE= Interrupt Enable, ACT= Activate, IACK= Interrupt Acknowledge, ERR=Error, IRQ=Interrupt request.

Då IE sätts till 1 aktiveras avbrottsmekanismen.
Då ACT sätts till 1 kommer robotarmen att aktiveras och röra sig mot koordinater DataX och DataY.
Vid avbrott sätts IRQ till 1 av hårdvaran.
Då en etta skrivs till IACK återställs IRQ.
Bitar IE, ACT och IACK är endast skrivbara, ERR och IRQ är endast läsbara.

a) (2p) Visa en C-deklaration i form av en sammansatt datatyp (**`struct`**) som beskriver gränssnittet till roboten. Visa också lämpliga symboliska definitioner av bitarna i styrregistret.

b) (4p) Följande funktioner ska nu implementeras:

`void` `init (`**`void`**`)`	initierar roboten, placerar robotarmen i läge 0,0.
`void` `move (`**`int`** `x, `**`int`** `y)`	startar och fullföljer förflyttning, dvs. initierar register och väntar tills robotarmen är i vald position (x=DataX=PositionX och y=DataY=PositionY)

Visa en lösning som *inte* använder avbrott (”busy wait”).

c) (2p) Om IE sätts till 1 aktiveras avbrottsmekanismen. Detta innebär att ett avbrott genereras då DataX=PositionX och DataY=PositionY. För att kunna använda avbrottsmekanismerna behöver vi en avbrottsfunktion som deklareras enligt följande:

`void` `robotirq(`**`void`**`);`	anropas vid avbrott

Visa nu en korrekt typdefinition (`simplefunc`) av en pekare till en sådan funktion, dvs. ett typalias så att en pekare `fp` till en sådan funktion kan deklareras som `simplefunc fp;`

d) (4p) Visa en lösning som utnyttjar avbrott, och som implementerar följande funktioner:

`void` `init(void);`	initierar roboten
`void` `move(`**`int`** `x, `**`int`** `y);`	startar förflyttning av robotarmen till (x, y)
`int` `status(`**`void`**`)`	ger ett värde: 0 om robotarmen är i vila, 1 om robotarmen är i rörelse och -1 om fel uppstått

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

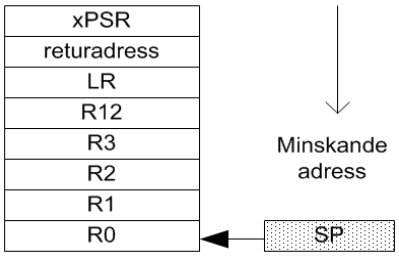
Bilaga 2: Assemblerdirektiv för MC68HC12.

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.(2p)

b) Processorn sparar alla registerinnehåll på stacken, läser adressen till avbrottshanteringsrutinen från avbrottsvektorn och placerar denna i PC. (2p)

c) För att undvika en rekursiv behandling av samma avbrott.(1p)

d) Modifiera det CC-innehåll som återställs från stacken vid RTI, dvs: (3p)

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

e) (2p)

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

f) (4p)

```
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: (6p)

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

void   FindFirst1( 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-3:

a) (5p)

```
;     4 | void   FindFirst1( unsigned char c )
_FindFirst1:
;     5 | {
;     6 |     *( unsigned char *) 0x600 = c ;
    LDAB   2,SP
    STAB   $600
;     7 |     delay();
    JSR_delay
;     8 |     c = c >> 1;
    LDAB   2,SP
    LSRB
    STAB   2,SP
;     9 |     *( unsigned char *) 0x600 = c ;
    STAB   $600
;    10 | }
    RTS
```

b) (3p)

```
    LDX    #_cp
    PSHX
    JSR    _identify
    LEAS      2,SP
    STD    _cp
```

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, b7 utgång , b6-0 ingångar    */
    *GPIO_E_MODER  = 0x4000;
    *GPIO_E_PUPDR  = 0x2AAA;      /* b6-0 pull down */
    *GPIO_E_OTYPER  = 0x0000;     /* b7 push/pull */
}
```

b)

```
static int d_valid;

void   systick_irq_handler ( void )
{
    *STK_CTRL = 0;
    d_valid = 1;
}
void start_timeout( void )
{
    *((void (**)(void) ) 0x2001C03C ) = systick_irq_handler;
    *STK_CTRL = 0;
    *STK_LOAD = ( 168000 - 1 );   /* 1 ms */
    *STK_VAL = 0;
    *STK_CTRL = 7;
    d_valid = 0;
}
```

c)

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

Uppgift B-2: (4p)

```
    .thumb_func
min:
**** int min( char a, char b)
**** {
****    if( a < b )
    CMP     R0,R1
    BLT     max_exit    @ ‘a’ redan i R0
****    else
    MOV     R0,R1       @ ‘b’ till R0
max_exit:
**** }
    BX      LR
```

(6p)

```
@ Konvention säger att hela register R0,R1 används för parameter.
@ Teckenutvidgning ska därför ske FÖRE anrop
callmin:
**** short c,d,e;
****
void    callmin(void )
**** {
    PUSH    {LR}
****    e = min( c, d);
    LDR     R0,=c
    LDRB    R0,[R0]
    SXTB    R0,R0
    LDR     R1,=d
    LDRB    R1,[R1]
    SXTB    R1,R1
    BL      min
    LDR     R3,=e
    STRB    R0,[R3]
    POP     {PC}

    .ALIGN
c:  .SPACE  1
d:  .SPACE  1
e:  .SPACE  1
```

Uppgift B-3:

Endast de bitar som konfigurerar EXTI10 får/ska initieras.

```
#define SYSCFG_EXTICR3 0x40013810
#define EXTI_IMR       0x40013C00
#define EXTI_FTSR      0x40013C0C
#define EXTI_RTSR      0x40013C08
#define NVIC_ISER1     0xE000E104

*((volatile unsigned short *) SYSCFG_EXTICR3) &= 0xF0FF;    /* nollställ bitfält EXTI10 */
*((volatile unsigned short *) SYSCFG_EXTICR3) |= 0x0300;    /* PD10->EXTI10  */
*((volatile unsigned int *)   EXTI_IMR) |= (1<<10);         /* aktivera avbrott EXTI10 */
*((volatile unsigned int *)   EXTI_FTSR) |= (1<<10);        /* aktivera trigger på negativ flank */
*((volatile unsigned int *)   EXTI_RTSR) &= ~(1<<10);       /* deaktivera trigger på positiv flank */
*((volatile unsigned int *)   NVIC_ISER1) |= (1<<8);        /* aktivera avbrott i NVIC */
```

Vektor nummer 40 (offset 0xE0)

Uppgift C-4:

```
int bitcheck( unsigned int *p, int *num)
{   /* Vi räknar ettorna, det är enklast…*/
    while(*p)
    {
        if( *pp & 1 ) *num++;
        *p >>= 1;
    }
    *num = 32 - *num;  /* Antal nollor */
    return !(*num & 3);
}
```

Uppgift C-5

Uppgift 5a:

```
typedef struct sROBOT{
    volatile unsigned char ctrl;
    volatile unsigned char datax;
    volatile unsigned char datay;
    volatile unsigned char posx;
    volatile unsigned char posy;
}ROBOT, *PROBOT;

#define IE      0x80
#define ACT     0x40
#define IACK    0x10
#define ERR     2
#define IRQ     1
```

Uppgift 5b:

```
void move( int x, int y )
{
  ((PROBOT) (0x800))->datax = x;
  ((PROBOT) (0x800))->datay = y;
  ((PROBOT) (0x800))->ctrl = ACT;

  while( (((PROBOT) (0x800))->posx != x )
   || (((PROBOT) (0x800))->posy != y ) ) ;
}

void init( void )
{
  move( 0,0 );
}
```

Uppgift 5c:

```
typedef    void (* simplefunc ) (void);
```

Uppgift 5d:

```
static int robot_status;
void init(void)
{
    ((PROBOT) (0x800))->ctrl = 0; /* passiva styrsignaler */
    robot_status = 0;
}

void move(int x, int y)
{
    ((PROBOT) (0x800))->datax = x;
    ((PROBOT) (0x800))->datay = y;
    robot_status = 1;
    ((PROBOT) (0x800))->ctrl = ACT|IE;
}

int status( void )
{
    return robot_status;
}
void robotirq( void )
{
    if( ((PROBOT) (0x800))->ctrl & ERR )
        robot_status = -1;
    else
        robot_status = 0;
    ((PROBOT) (0x800))->ctrl = 0;
}
```