Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D
EDA486 Programmering av inbyggda system Z
DAT016 Programmering av inbyggda system IT
DIT152 Programmering av inbyggda system GU

Fredag 7 oktober 2016, kl. 14.00 - 18.00

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:
Roger Johansson

Tillåtna hjälpmedel

Häftet

Instruktionslista för CPU12
Quick Guide, Laborationsdator MD407 med tillbehör

Inget annat än rättelser och understrykningar får vara införda i häftena.
Du får också använda bladet:

C Reference Card

samt *en* av böckerna:

Vägen till C,
Bilting, Skansholm

C från början
Skansholm

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

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

Tabellverk eller miniräknare får ej användas.

Lösningar
anslås senast dagen efter tentamen via kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

Allmänt
Tentamen är anpassad för både det äldre 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 givits under kursen och tillräckligt dokumenterade.
- C-program är utformade enligt de råd och anvisningar som getts under kursen. I programtexterna skall raderna dras in så att man tydligt ser programmens struktur.

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

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

Uppgift A-1 (12p) *Maskinnära programmering i C/avbrott*

I ett fordon finns en farthållare "Cruise control" som direkt kan påverka motorns styrsystem och få fordonet att hålla en konstant hastighet. Farthållaren är uppbyggd kring en HCS12 microcontroller. Gränssnittet till farthållaren har följande utseende:

Parallel port Cruise Control (PortCC)										
Address	7	6	5	4	3	2	1	0	Mnemonic	Namn
0xA00					INC	DEC	LOCK	AUTO	CTRL	Control register
0xA01	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	SM	State machine register
0xA02	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	VEL	Velocity register
0xA03					IREQ	DREQ	CROFF	CRON	IRQ	Interrupt request register
0xA04	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	IADD	Interrupt address high
0xA05	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0		Interrupt address low

CTRL	Styrregister för modulen, kontrollerar farthållarens funktion. I styrregistret är biten AUTO nivåstyrd, medan övriga bitar är flankstyrda. registret är endast skrivbart, en läsning från styrregistret returnerar alltid värdet 0xFF.	
	AUTO	Biten används tillsammans med registret SM för att aktivera farthållaren. Genom att skriva 0 till denna bit deaktiveras farthållaren.
	LOCK	då farthållaren är aktiv kommer en positiv transition hos denna bit (0→1) att låsa hastigheten, dvs. fordonets aktuella hastighet, fås automatiskt från motorstyrenheten, kopieras till farthållarens hastighetstregister VEL.
	DEC	då farthållaren är aktiv kommer en positiv transition hos denna bit (0→1) att minska innehållet i hastighetsregistret med 1, dvs. minska den inställda hastigheten med 1,6 km/tim
	INC	då farthållaren är aktiv kommer en positiv transition hos denna bit (0→1) att öka innehållet i hastighetsregistret med 1, dvs. öka den inställda hastigheten med 1,6 km/tim
SM	Registret påverkar den interna tillståndsmaskinen som används för att aktivera farthållaren	
VEL	Hastigheten anges i steg om 1,6 km/tim. Farthållarens minimala hastighet är 1,6 km/tim då VEL är 0x00	
IRQ	Statusbitar som aktiveras av farthållarens omkopplarfunktioner. Dessa bitar är samtidigt kvittensbitar för de olika funktionerna	
	CRON	Biten sätts till 1 då Cruise On aktiveras, biten nollställs då programmet skriver 1 till denna bit
	CROFF	Biten sätts till 1 då Cruise OFF aktiveras, biten nollställs då programmet skriver 1 till denna bit
	DREQ	Biten sätts till 1 då Decrease aktiveras, biten nollställs då programmet skriver 1 till denna bit
	IREQ	Biten sätts till 1 då Increase aktiveras, biten nollställs då programmet skriver 1 till denna bit
IADD	Avbrottsvektor, adressen till avbrottsrutinen (2×8 bitar) ska placeras i dessa register	

Via styrregistret CTRL kontrolleras farthållarens funktion. För att inte farthållaren enkelt ska kunna aktiveras ofrivilligt, har den försetts med en tillståndsmaskin, som används vid aktiveringen av farthållaren. För att aktivera farthållaren krävs att följande algoritm utförs:

0→AUTO
0x55→SM
0xAA→SM
0x55→SM
1→AUTO

Föraren kan påverka farthållningen med hjälp av följande omkopplarfunktioner

Cruise ON:	Aktivera farthållare och lås farthållarhastigheten till fordonets aktuella hastighet
Cruise OFF:	Deaktivera farthållare
Increase:	Inställd hastighet ökas med 1,6 km/tim
Decrease:	Inställd hastighet minskas med 1,6 km/tim

Då föraren pressar ned broms- eller gaspedal utförs automatiskt funktionen Cruise OFF.

Farthållarens programvara ska utformas som ett enkelt huvudprogram med tillhörande avbrottsrutin. Observera att farthållarens avbrottssystem alltid är aktiverat varför systemet måste förberedas för att ta emot avbrott omedelbart efter uppstart. Följande subrutiner finns givna:

```
       export    _cli
_cli:  cli
       rts
       export    _isr
       import    _cruiseISR
_isr:  jsr       _cruiseISR
       rti
```

Implementera farthållarens programvara, programkoden ska organiseras i två filer "cruiseControl.h" och "cruiseControl.c". Du ska visa och använda en lämplig deklaration av PortCC med användning av en struct. Tänk på att grundläggande felkontroller måste utföras av programmet. INC, DEC eller LOCK får inte aktiveras om farthållaren inte är aktiv. Farthållarens hastighet måste också kontrolleras innan INC eller DEC utförs.

Uppgift A-2 (8p) *Assemblerprogrammering*

Följande funktion finns given i "C".
Implementera en motsvarande subrutin i assemblerspråk för HC12.

Du ska inte förutsätta några speciella kompilatorkonventioner i denna uppgift. Parametern 'c' skickas i register D med anropet, returvärdet från subrutinen ska också skickas i register D.

```
int isxdigit (int c)
{
   if( ( c >= '0' && c <= '9')  ||
       ( c >= 'a' && c <= 'f')  ||
       ( c >= 'A' && c <= 'F') )
          return 1;
   return 0;
}
```

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

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (bilaga 2).
Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

```
void  func( char *bertram, char adam )
{
   char  cicero;
   char  *dorian; ....
```

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

```
char     *ada,beda;
```

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

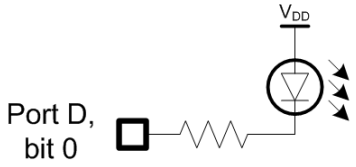
b) Med variabeldeklarationerna i a), visa hur följande funktionsanrop översätts till assemblerkod för HCS12:

```
func( ada , beda );
```

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

Uppgift B-1 (14p)

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



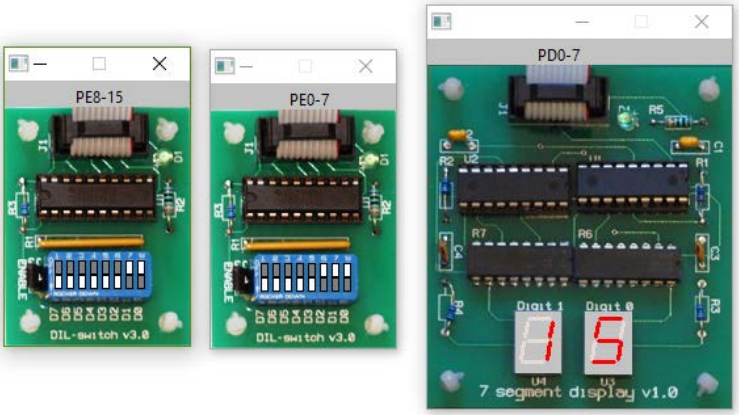
Skriv ett program, i C, som kontinuerligt tänder dioden under en sekund och därefter släcker den under en sekund. Din lösning ska fördelas på följande deluppgifter:

a) Visa hur SysTick kan användas för att skapa en (blockerande) fördröjning om 500 mikrosekunder (en halv millisekund), med funktionen `void delay_500mys( void);`. Systemets klockfrekvens är 168 MHz. (6p)
b) Skriv en funktion `void init_app( void );` som sätter upp port D, bit 0 som utsignal, ”push-pull”. Övriga inställningar i porten ska behållas, dvs. får inte ändras av initieringen. (2p).
c) Skriv huvudprogrammet som får dioden att blinka. Du kan använda `delay_500mys()` och `init_app()` även om du inte besvarat a) och/eller b). (2p)
d) Visa hur en SysTick-modul kan beskrivas med en sammansatt datatyp `struct`, i C. Förutsätt att registeråtkomster alltid är 32 bitars ord. (4p)

Uppgift B-2 (6p)

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

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



Skriv ett program i ARM/Thumb assemblerspråk som:

- Initierar portar D och E.
- Kontinuerlig läser de inställda värdena från DIL-strömbrytarna, multiplicerar dessa och skriver de 8 minst signifikanta bitarna av resultatet till displayen. De avlästa värdena ska betraktas som tal utan tecken.

Uppgift B-3 (8p)

Förbered en enkel applikation som använder PE3 hos MD407 som avbrottsingång. Beskriv initieringsfunktioner och avbrottshantering, dvs. färdigställ följande:

```
void irq_handler ( void )
{
    Om avbrott från EXTI3:
        kvittera avbrott från EXTI3
}

void enable_interrupt( void )
{
    Nollställ processorns avbrottsmask
}

void app_init ( void )
{
    Koppla PE3 till avbrottslina EXTI3;

    Konfigurera EXTI3 för att generera avbrott;
    Konfigurera för avbrott på negativ flank

    Sätt upp avbrottsvektor

    Konfigurera de bitar i NVIC som kontrollerar den avbrottslina som EXTI3 kopplats till
}
```

Uppgift C-5 (8p) *Programmering med pekare*

Uppgiften är att skriva en C-funktion med namnet `split` som delar upp en text i delar, s.k. *tokens*. Varje token avgränsas i texten av ett eller flera blanka tecken, med undantag för den första token som inte behöver ha något blankt tecken före och den sista som inte behöver ha något blankt tecken efter. Texten `"Nu tentar vi MOP!"` skall t.ex. delas upp i fyra tokens: `"Nu"`, `"tentar"`, `"vi"` och `"MOP!"`.

Funktionen skall ha följande deklaration:

```
void split(char *s, char *tab[], int max);
```

Parametern `s` är en pekare till den text som skall delas upp. Du får förutsätta att denna text avslutas med ett noll-tecken. Parametern `tab` är en pekare till ett oinitierat fält. Du får förutsätta att minnesutrymme för själva fältet har allokerats i den anropande funktionen. Antalet element i fältet anges av parametern `max`.

Funktionen `split` skall fylla i fältet `tab` så att dess komponenter pekar på de tokens som finns i texten `s`. Om fältet `tab` innehåller färre element än antalet tokens skall bara så många tokens pekas ut som antalet element i `tab` medger. Om det å andra sidan finns färre tokens än antalet element i `tab` så skall de överflödiga elementen i `tab` fyllas i med tomma pekare. Funktionen `split` skall dessutom i texten `s` lägga in noll-tecken efter varje token.

I din lösning får du inte använda dig av indexering, eller några färdiga standardfunktioner, utan du måste skriva allt själv.

Här visas ett litet testprogram som anropar funktionen `split`.

```
#define N 10
main() {
  int i;
  char txt[] = "Nu tentar  vi  MOP! ";
  char *tokens[N];
  split(txt, tokens, N);
  for (i=0; i< N && tokens[i]; i++)
    printf("%s\n", tokens[i]);
}
```

Utskriften från testprogrammet skall då bli:

```
Nu
tentar
vi
MOP!
```

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

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) En robotarm styrs av en dator, via fem 8-bitars register: ett styr-/status- register på adressen 0x0800 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		ACK			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

*) Bitar IE, ACT och IACK är läs- och skrivbara, ERR och IRQ är endast läsbara.

ACT: Activate, sätts till 1 för att aktivera robotarmen, efter aktivering kommer denna att röra sig mot målkoordinaterna angivna i dataregistren. Positionsregistren uppdateras av roboten allt eftersom armen rör sig.

IE: Interrupt Enable, sätts till 1 för att aktivera avbrottsmekanismen i gränssnittet, efter aktivering genereras ett avbrott då robotarmen nått målkoordinaterna, dvs. data och positionsregistren överensstämmer.

IRQ: Om avbrott är aktiverat sätts denna bit till 1 då innehållen i data och positionsregistren överensstämmer.

ERR: Om fel inträffat, som gör att robotarmen inte kan röra sig mot dataregistrens koordinater, stoppas robotarmen, denna bit sätts till 1, om avbrott är aktiverat sätts då också IRQ-biten.

ACK: Acknowledge, då denna bit sätts till 1 återställs bitarna ERR och IRQ till 0 av robotens gränssnitt.

För att starta robotarmen krävs att:

1. Dataregistren initieras med målkoordinaterna.
2. Robotarmen aktiveras.
3. Om avbrott ska användas måste också avbrottsmekanismen aktiveras.
4. Då robotarmen nått målkordinaterna ska den deaktiveras.

Följande funktioner ska nu implementeras, visa en lösning som *inte* använder avbrott (4p):

`void init (void)`	initierar roboten, placerar robotarmen i läge 0,0.
`void move (int x, int y)`	utför förflyttning av robotarmen till (x, y)

För att kunna använda avbrottsmekanismerna behöver vi två funktioner som inte enkelt kan implementeras med C-kod, dom kan dock beskrivas enligt följande:

`void robotirq(void);`	utförs vid avbrott, ska enbart anropa en C-rutin (`robottrap`).
`void cleari(void);`	denna rutin aktiverar avbrottsmekanismen hos processorn.

Förutsätt att ovanstående båda funktioner är givna och visa en lösning som utnyttjar avbrott, och som implementerar följande funktioner (6p):

`void init( void );`	initierar roboten och förbereder systemet för avbrottshantering.
`void move(int x, int y);`	startar förflyttning av robotarmen till (x, y), avbrott ska genereras då armen nått målkoordinaterna.
`int status(void);`	ger ett värde: 0 om robotarmen är i vila, 1 om robotarmen är i rörelse och -1 om fel uppstått
`void robottrap(void);`	anropas, från assemblerrutin `robotirq`, vid avbrott

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:

```
// i fil "cruiseControl.h"
typedef struct sCruiseControl{
    volatile unsigned char ctrl;
    volatile unsigned char sm;
    volatile unsigned char vel;
    volatile unsigned char ir;
    volatile unsigned short ivec;
}CRUISE_CONTROL;
#define CRUISE_CONTROL_BASE   0xA00

#define INC     8
#define DEC     4
#define LOCK    2
#define AUTO    1
#define IREQ    8
#define DREQ    4
#define CROFF   2
#define CRON    1

#define CruiseC ((CRUISE_CONTROL *) CRUISE_CONTROL_BASE)

extern void cli( void );
extern void isr( void );

// i fil "cruiseControl.c"
static unsigned char auto_copy;      /* "skuggbit" av icke läsbar bit */
void    cruiseISR( void )
{
    unsigned char stat = CruiseC->ir;
    /* prioritetsbaserad kontroll */
    if( stat & CROFF )
    {
        CruiseC->ctrl = 0;           /* återställ alla bitar */
        auto_copy = 0;
        CruiseC->ir |= CROFF;        /* kvittera avbrott */
    }else if( stat & CRON )
    {
        CruiseC->ctrl = 0;           /* återställ alla bitar */
        CruiseC->sm = 0x55;          /* aktiveringsalgoritm */
        CruiseC->sm = 0xAA;
        CruiseC->sm = 0x55;
        CruiseC->ctrl = AUTO;        /* aktivera farthållaren */
        CruiseC->ctrl = LOCK|AUTO;   /* lås hastigheten */
        auto_copy = 1;
        CruiseC->ir |= CRON;         /* kvittera avbrott */
    }else if( stat & DREQ )
    {
        if( auto_copy && CruiseC->vel )
            CruiseC->ctrl = DEC|AUTO; /* minska hastigheten */
        CruiseC->ir |= DREQ;         /* kvittera avbrott */
    }else if( stat & IREQ )
    {
        if( auto_copy && CruiseC->vel != 0xFF )
            CruiseC->ctrl = INC|AUTO; /* öka hastigheten */
        CruiseC->ir |= IREQ;         /* kvittera avbrott */
    }
}

int     main()
{
    CruiseC->ivec = isr; ;           /* avbrottsvektor */
    CruiseC->ir = 0xF;               /* återställ alla IRQ */
    CruiseC->ctrl = 0;               /* deaktivera allt */
    auto_copy = 0;

    cli();                           /* starta applikationen */
    while( 1 );                      /* oändlig slinga */
}
```

Uppgift A-2:

```
isxdigit:
;{
;   if( ( c >= '0' && c <= '9')  ||
;       ( c >= 'a' && c <= 'f')  ||
;       ( c >= 'A' && c <= 'F'))
    CPD     #'0'
    BLT     _1
    CPD     #'9'
    BLE     isxdigit_0
    CPD     #'A'
    BLT     _1
    CPD     #'F'
    BLE     isxdigit_0
    CPD     #'a'
    BLT     isxdigit_1
    CPD     #'f'
    BGT     isxdigit_1
isxdigit _0:
;       return 1;
    LDD #1
    RTS
isxdigit_1:
;   return 0;
    CLRA
    CLRB
; }
    RTS
```

Uppgift A-3:

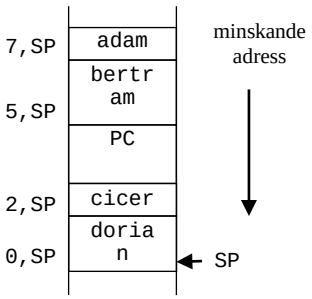
a)

```
_ada    MB      2
_beda   RMB     1
```

b)

```
        LDAB    _beda
        PSHB
        LDD     _ada
        PSHD
        JSR     _func
        LEAS    3,SP
```

c)



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_500mys( void )
{
    /* SystemCoreClock = 168000000 */
    * STK_CTRL = 0;
    * STK_LOAD = ( (168000/2)-1 );
    * STK_VAL = 0;
    * STK_CTRL = 5;
    while( (*STK_CTRL & 0x10000 )== 0 );
    * STK_CTRL = 0;
}
```

b)

```
#define    GPIO_D_MODER   ((volatile unsigned int *)(0x40020C00))
#define    GPIO_D_OTYPER  ((volatile unsigned int *)(0x40020C04))

void init_app( void )
{
    /* PORT D */
    * GPIO_D_MODER    &= ~3; /* återställ bit 0 mode */
    * GPIO_D_MODER    |= 5;  /* bit 0 sätts som utgång */
    * GPIO_D_OTYPER   &= ~1; /* återställ bit 0 typ, är nu push/pull */
}
```

c)

```
#define    GPIO_D_MODER   ((volatile unsigned int *)(0x40020C00))
#define    GPIO_D_OTYPER  ((volatile unsigned int *)(0x40020C04))
#define    GPIO_D_ODR_LOW ((volatile unsigned int *)(0x40020C14))
void main(void)
{
    int i;
    init_app();
    while(1)
    {
        * GPIO_D_ODR_LOW = 0;
        for(i=0;i<2000;i++) delay_500mys();
        * GPIO_D_ODR_LOW = 0xFF;
        for(i=0;i<2000;i++) delay_500mys();
    }
}
```

Uppgift B-2:

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

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

Uppgift B-3:

```
#define EXTI_PR               0x40013C14
#define EXTI3_IRQ_BPOS        (1<<3)
#define EXTI3_IRQVEC          0x2001C064
#define NVIC_EXTI3_IRQ_BPOS  (1<<9)
```

```
void irq_handler ( void )
{
    /* Om avbrott från EXTI3:
         kvittera avbrott från EXTI3 */
    if(   *((unsigned int *) EXTI_PR) & EXTI3_IRQ_BPOS )
    {
        *((unsigned int *) EXTI_PR) |= EXTI3_IRQ_BPOS;
    }
}

void enable_interrupt( void )
{
    /* Nollställ processorns avbrottsmask */
    __asm (
    "      cpsie  i\n"      /* set I=0 */
    );
}
```

eller:

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

void app_init ( void )
{
    /* Koppla PE3 till avbrottslina EXTI3 */
     *((unsigned int *) SYSCFG_EXTICR1) |= 0x4000;
    /* Konfigurera EXTI3 för att generera avbrott */
     *((unsigned int *) EXTI_IMR) |= EXTI3_IRQ_BPOS;
    /* Konfigurera för avbrott på negativ flank */
     *((unsigned int *) EXTI_FTSR) |= EXTI3_IRQ_BPOS;
    /* Sätt upp avbrottsvektor */
     *((void (**)(void) ) EXTI3_IRQVEC ) = irq_handler;
    /* Konfigurera den bit i NVIC som kontrollerar den avbrottslina som EXTI3 kopplats till */
       *((unsigned int *) NVIC_ISER0) |= NVIC_EXTI3_IRQ_BPOS;
}
```

Uppgift B-4:

```
typedef struct _systick
{
   volatile unsigned int stk_ctrl;
   volatile unsigned int stk_load;  // +0x4
   volatile unsigned int stk_val;   // +0x8
   volatile unsigned int stk_calib; // +0xc
} SYSTICK;
```

Uppgift C-5:

```
   void split(char *s, char *tab[], int max) {
      int i;
      for (i=0; i<max; i++)
        tab[i] = 0;
      for (i=0; i<max; i++) {
        while (*s && *s == ' ')
          s++;
        if (!*s)
          return;
        tab[i] = s;
        while (*s && *s != ' ') {
          s++;
        }
        if (!*s)
          return;
        *s++ = '\0';
      }
    }
```

Uppgift C-6

a)

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

b)

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

#define ROBOT_CTRL     ((volatile unsigned char *) 0x800)
#define ROBOT_DATA_X   ((volatile unsigned char *) 0x801)
#define ROBOT_DATA_Y   ((volatile unsigned char *) 0x802)
#define ROBOT_POS_X    ((volatile unsigned char *) 0x803)
#define ROBOT_POS_Y    ((volatile unsigned char *) 0x804)

void move( int x, int y )
{
  *ROBOT_DATA_X = x;
  *ROBOT_DATA_Y = y;
  *ROBOT_CTRL |= ACT;
  while( ( *ROBOT_POS_X != x ) || ( *ROBOT_POS_Y != y )) ;
  *ROBOT_CTRL = ~ACT;
}

void init( void )
{
  *ROBOT_CTRL = 0;
  move( 0,0 );
}

extern void robotirq( void );
static int robot_status;
void init( void )
{
    *ROBOT_CTRL = 0;                /* passiva styrsignaler */
    robot_status = 0;
    cleari();
}

void move(int x, int y)
{
    *ROBOT_DATA_X = x;
    *ROBOT_DATA_Y = y;
    robot_status = 1;
    *ROBOT_CTRL |= (ACT|IE);
}

int status( void )
{
    return robot_status;
}
void robotirq( void )
{
    if(*ROBOT_CTRL & ERR )
        robot_status = -1;
    else
        robot_status = 0;
    *ROBOT_CTRL |= ACK;       /* kvittera avbrott/felflagga */
    *ROBOT_CTRL &= ~(ACT|IE); /* deaktivera... */
}
```