Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA482 Maskinorienterad programmering D
EDA487 Maskinorienterad programmering Z
DAT017 Maskinorienterad programmering IT
DIT151 Maskinorienterad programmering GU

Fredag 18 augusti 2017, kl. 8.30 - 12.30

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:
Andreas Wieden, tel 772 10 23
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:

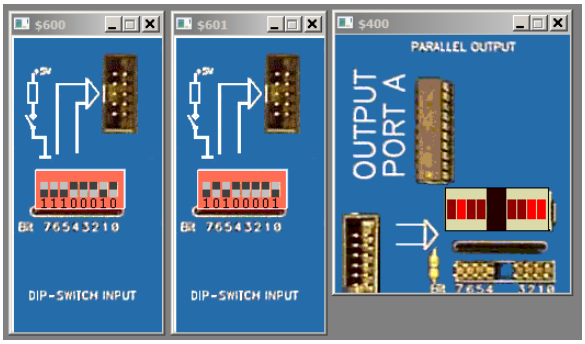
- redovisningen av svar och lösningar är läslig och tydlig. Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- lösningen ej är onödigt komplicerad.
- du har motiverat dina val och ställningstaganden
- assemblerprogram är utformade enligt de råd och anvisningar som 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 (6p) *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.



Två strömbrytare och en ljusdiodramp, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DipSwitchEor( void )
```

som kontinuerligt bildar logiskt EXKLUSIVT ELLER (XOR) av värdena som läses från strömbrytarna och därefter skriver detta värde till ljusdiodrampen.

Uppgift A-2 (8p) *Assemblerprogrammering*

Följande funktion `modyBit` finns given i "C". (Funktionen kombinerar subrutinerna `OUTONE` respektive `OUTZERO` från laborationskursen).

```
void  modyBit( unsigned char bit_number, unsigned char bit_value )
{
   static   char  shadow;
   static   char  set[]=   {0x80,0x40,0x20,0x10,8,4,2,1};
   static   char  clear[]= {0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE};

   if( bit_number > 7 )    return;
   if( bit_value > 1 )     return;
   if( bit_value == 1 )
      shadow = shadow | set[bit_number];
   else
      shadow = shadow & clear[bit_number];
   *((unsigned char *) 0x400) = shadow;
}
```

Skriv motsvarande funktion `MODYBIT` i assemblerspråk för HC12. I denna uppgift ska du INTE ta hänsyn till kompilatorkonventioner utan i stället skickas parametrarna till `OUTBIT` enligt:

`bit_number` i register B

`bit_value` i register A

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

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

a) I ett C-program har vi följande deklarationer:

```
void  func(int *pi, char *pc, long *pl )
{
   char  lc;
   int   li;
   long  *lpl;

   lc = *pc;
   li = *pi;
   lpl = pl;
   /* Övrig kod i funktionen är bortklippt eftersom vi bara
      betraktar anropskonventionerna. */
}
```

Översätt *hela* funktionen `func`, som den är beskriven, till HCS12 assemblerspråk, såväl *prolog* som *tilldelningar* och *epilog* ska alltså finnas med. Speciellt ska du börja med att beskriva aktiveringsposten, dvs. stackens utseende i funktionen och där riktningen för minskande adresser hos aktiveringsposten framgår. (6p)

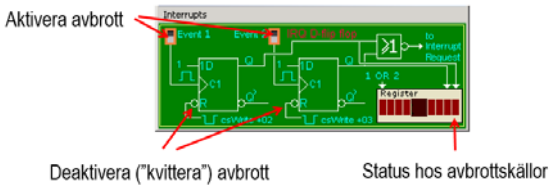
b) I samma C-program har vi dessuton följande deklarationer givna på ”toppnivå” (global synlighet):

```
char  * b;
int   * a;
long  * c;
```

Visa hur variabeldeklarationerna översätts till assemblerdirektiv. (2p)

Uppgift A-4 (6p) *Avbrott*

Under laborationerna har du kommit i kontakt med laborationskortet ML19, försett med två avbrottsvippor.



Gränssnittet består av 4 st. 8-bitars register, 3 av dessa används.

Address	7	6	5	4	3	2	1	0	Mnemonic	Namn
$DC0	IRQ						IRQ1	IRQ2	STATUS	Status Register
$DC1										
$DC2									IRQ1ACK	Interrupt 1 acknowledge
$DC3									IRQ2ACK	Interrupt 2 acknowledge

Statusregistret innehåller IRQ-status, dvs. en ettställd bit representerar avbrott. Avbrottskällorna separeras med bitarna b_1 och b_0. b_7 ettställs om någon avbrottskälla är aktiv. Avbrotten kvitteras genom en skrivning till motsvarande *acknowledge*-register.
Visa hur du utformar avbottshanteringen i programdelar (C och assembler) så att du minimerar mängden assemblerkod som krävs för implementeringen. Det betyder att du utformar:
C-funktioner för att:

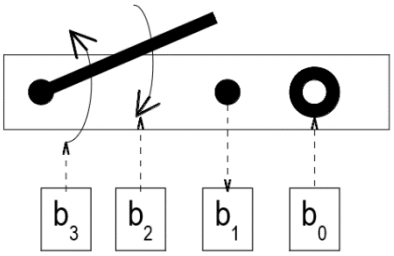
- initiera systemet för avbrott, avbrottsvektor 0x3FF2 ska användas.
- avbrottshanteringsfunktion som kvitterar det begärda avbrottet.

dessutom i assembler:

- nödvändiga stödfunktioner

Uppgift B-1 (14p)

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



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

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

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

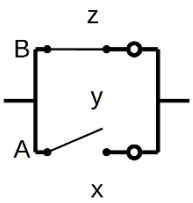
a) Använd SYSTICK för att konstruera en blockerande fördröjningsfunktion `void delay10ms(void);` som blockerar det anropande programmet i 10 ms. (3p)

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

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

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

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



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

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

1. Initiera systemet, öppna dörr A och stäng dörr B – förutsätt att inget fel uppstår här.
2. Så länge inget fel uppstår:
- kontrollera om det finns någon person i 'y' – i så fall, stäng dörr A, öppna dörr B, vänta tills utrymme ’y’ är tomt, stäng därefter dörr B och öppna dörr A.

Om något fel uppstår:

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

Det finns inga speciella prestandakrav, det räcker med att implementera funktionen enligt ovan. (4p)

Uppgift B-2 (4p)

Vi har de globala deklarationerna: `int i,j; char vm[10][10];` Visa en kodsekvens som evaluerar uttrycket `vm[i][j+1]` till register R0.

Uppgift B-3 (4p)

Utgå från att följande deklarationer är gjorda på global nivå.

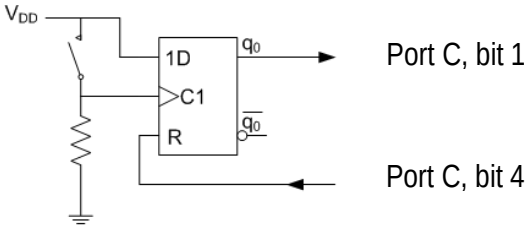
```
short   f(void);
int     a,b;
```

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

```
f() + a * ~( b & 0xFF );
```

Uppgift B-4 (6p)

En avbrottsvippa kopplas till *MD407* enligt följande figur:



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

a) Visa med en funktion `app_init` hur avbrottsmekanismerna initieras, dvs. IO-pinnar konfigureras, EXTI och NVIC initieras. Observera, övriga IO-pinnar i port C får *inte* ändras av din initiering. (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.

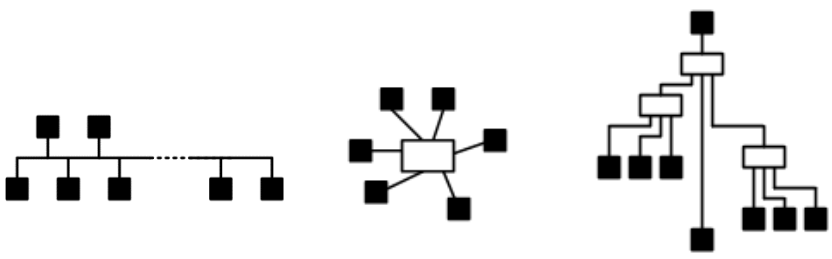
Uppgift C-5 (8p)

a) Visa typdeklarationer för funktioner som tillåter anrop direkt från ett C-program via en pekarvariabel av denna typ (3p).
   - typen `vfunc` har inga parametrar och inget returvärde.
   - typen `pfunc` har en parameter av typen unsigned char, men inget returvärde.
   - typen `rfunc` har inga parametrar men returvärde av typen unsigned char.

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

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

c) Vad kallas de nätverkstopologier som illustreras på följande sätt? (3p)



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

Inom matematiken säger man att två vektorer ($u_1, u_2, \ldots u_n$) och ($v_1, v_2, \ldots v_n$) är ortogonala om summan $\sum_{k=1}^{n} u_k v_k$ är lika med noll. Skriv en C-funktion

```
int ortogonala(int *v1, int *v2, int n);
```

som avgör om två heltalsvektorer är ortogonala. Funktionen skall ge ett sanningsvärde som resultat. Vektorerna skall representeras med hjälp av två heltalsfält. I funktionen får du inte använda dig av indexering, utan du måste använda pekare för att stega igenom fälten.

Uppgift C-7 (8p) *Maskinnära programmering i C*

Denna uppgift handlar om att konstruera drivrutiner för en tänkt kommunikationsenhet. Enheten kan ta emot och sända data från resp. till en annan dator. Enheten har därför två separata kanaler: en mottagningskanal och en sändningskanal. Uppgiften behandlar dock endast mottagningskanalen,

Drivrutinerna för enheten använder sig av två buffertar: en inbuffert i vilken inlästa data läggs och en utbuffert som utgående data hämtas ifrån. Det är meningen att ett program som vill använda sig av kommunikationsenheten ska göra det via dessa buffertar. Vill programmet ta emot data via kommunikationsenheten ska det alltså hämta data från inbufferten och vill det sända data skall det lägga dessa data i utbufferten. Det finns en färdigskriven modul `buffer`, för buffertar som du ska använda dig av. Den har följande `.h` fil:

```
/* Filen buffer.h */
typedef struct bufferstruct buffer;  /* Full definition finns i filen buffer.c */
buffer *create_buffer(int max);   /* Ger en buffert med plats för max element */
int put(buffer *b, char c);       /* Försöker lägga in c sist i *b.
                                     Returnerar true om det gick, false annars */
int get(buffer *b, char *pc);     /* Försöker hämta första tecknet i *b till *pc.
                                     Returnerar true om det gick, false annars */
```

Följande specifikationer gäller för kommunikationenheten: Det finns fyra åttabitars register: `SEND_DATA`, `SEND_CTRL`, `RECEIVE_DATA` och `RECEIVE _CTRL`. Dessa ligger på de hexadecimala adresserna 400, 401, 402 resp. 403. I dataregistren finns de data som ska tas emot eller sändas. I respektive kontrollregister används tre bitar:

Bit 0, `RDY` sätts automatiskt till 1 när en överföring är klar. Den ska nollställas vid en ny överföring.
Bit 1, `GO` sätter man till 1 för att starta en överföring. Den nollställs automatiskt.
Bit 2, `IE` anger om enheten skall ge avbrott när en överföring är klar.

Konstruera den del av drivrutinerna som har att göra med mottagning. Det ska finnas en fil `driver.h` med en tillhörande fil `driver.c`. Lägg dessutom alla specifikationer för hårdvaran i en egen fil `channel.h`. Ange speciellt hur du placerat kod/deklarationer i dessa olika filer.

Filen `driver.h` ska utgöra gränssnittet mot övriga program i datorn. I denna fil ska två funktioner deklareras: `init_input` som initierar mottagningsdelen av enheten och `get_input_buffer` som ger en pekare till den inbuffert drivrutinen skapat och använder sig av. Dessa två funktioner ska förstås definieras i filen `driver.c`. I initieringsfunktionen avbrott aktiveras och en buffert skapas. Låt den ha 100 platser. Initieringsfunktionen måste vara utformad så att det inte kan skapas mer än en inbuffert, även om funktionen skulle bli anropad mer än en gång.

I filen `driver.c` skall det dessutom finnas en funktion `notify_input`. Denna anropas varje gång ett avbrott sker. Du behöver inte hantera själva avbrottet me i funktionen `notify_input` ska det inlästa tecknet tas om hand och enheten sedan ställas i ett tillstånd som tillåter en ny överföring. Du behöver inte ta hänsyn till ev. spill, dvs. kontrollera att ett inläst tecken ryms i bufferten.

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 IN1 *((port8ptr) 0x600)
#define IN2 *((port8ptr) 0x601)

void  DipSwitchEor( void )
{
   while( 1 )
   {
      OUT = IN1 ^ IN2;
   }
}
```

Uppgift A-2:

```
;  Data och variabler
shadow:   RMB    1
set:      FCB    $80,$40,$20,$10,8,4,2,1
clear:    FCB    $7F,$BF,$DF,$EF,$F7,$FB,$FD,$FE

MODYBIT:
   CMPB   #7        ; if( bit_number > 7 )
   BHI    MODYBIT_exit
   CMPA   #1        ; if( bit_value > 1 )
   BHI    MODYBIT_exit
   BNE    MODYBIT_1 ; if( bit_value == 0 )
;
   LDX    #set      ; if( bit_value == 1 ) shadow = shadow | set[bit_number];
   LDAB   B,X
   ORAB   shadow
   BRA    MODYBIT_2
MODYBIT_1:
   LDX    #clear    ; else shadow = shadow & clear[bit_number];
   LDAB   B,X
   ANDB   shadow
MODYBIT_2:
   STAB   shadow
   STAB   $400      ; *((unsigned char *) 0x400) = shadow;
MODYBIT_exit:
   RTS
```

Uppgift A-3:

a)

minnesanvändning	*adress*
pl	11,SP
pc	9,SP
pi	7,SP
PC (vid JSR)	
lc	4,SP
li	2,SP
lpl	0,SP

minskande adress ↓

```
_func:
   LEAS   -5,SP
```

```
    LDX   9,SP
    LDAB  ,X        ; lc = *pc;
    STAB  4,SP

    LDX   7,SP
    LDD   ,X        ; li = *pi;
    STD   2,SP

    LDD   11,SP        ; lpl = pl;
    STD   0,SP

    LEAS  5,SP
    RTS
```

b)

```
_b RMB   2
_a RMB   2
_c RMB   2
```

Uppgift A-4:

I C:

```
#define   Status       *( ( volatile unsigned char *) 0xDC0)
#define   Irq1Ack      *( ( volatile unsigned char *) 0xDC2)
#define   Irq2Ack      *( ( volatile unsigned char *) 0xDC3)
void init( void )
{
   extern void CLEARI(void);  /* assemblerrutin, nollställ I-flagga */
   extern void IRQ( void );   /* assemblerrutin, avbrott */

   Irq1Ack = 0;        /* återställ avbrottsvippor */
   Irq2Ack = 0;
   *((void(**)) 0x3FF2) = IRQ;   /* sätt avbrottsvektor */
   CLEARI();       /* nollställ I */
}
void   AtIRQ( void )
{
   if (Status & 1 )
      Irq1Ack = 0;
   else if (Status & 2 )
      Irq2Ack = 0;
}
```

i assembler:

```
_CLEARI:  CLI
          RTS

_IRQ:     JSR    _AtIRQ
          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  = 0x00005151;
    *GPIO_D_OTYPER = 0;
    *GPIO_D_PUPDR  = 0;
}


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

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

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

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

c)

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

}
```

Uppgift B-2:

```
@ adress = vm + i*10 + j + 1
   LDR   R3,i
   LDR   R1,j
   LSL   R2,R3,#2   @ R2 = (i*4)
   ADD   R3,R2,R3   @ R3 = (i*4+i)
   LSL   R2,R3,#1   @ R2 = (i*4+i)*2 = i*10
   LDR   R3,=vm
   ADD   R3,R3,R2   @ R3 = vm + i*10
   ADD   R3,R3,R1   @ R3 = vm + i*10 + j
   LDRB  R0,[R3,#1] @ R0 = M(char)( vm + i*10 + j+1
```

Uppgift B-3:

```
   BL    f
   SXTH  R0,R0
   LDR   R3,b
   MOV   R2,#0xFF
   AND   R3,R3,R2
   MVN   R3,R3
   LDR   R2,a
   MUL   R2,R2,R3
   ADD   R0,R0,R2
```

Uppgift B-4:

a)

```
#define NVIC_EXTI1_IRQ_BPOS   (1<<7)
#define EXTI1_IRQ_BPOS        (1<<1)
```

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

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

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

void app_init( void )
{
   *GPIOC_MODER &= ~0x30A;    /* Återställ bit 1 och bit 4 */
   *GPIOC_MODER |= 0x100;     /* Bit 4, digital ut, bit 1 digital in */
   *GPIOC_PUPDR &= ~1;        /* Nollställ bit 1: "Input floating" */
   *GPIOC_OTYPER &= ~1;       /* Nollställ bit 1: "Output push/pull" */

   *GPIOC_ODRLOW = 0x10;      /* Återställ D-vippan */
   *GPIOC_ODRLOW = ~0x10;

   *SYSCFG_EXTICR &= 0xFF0F;
   *SYSCFG_EXTICR |= 0x0020;           /* PC1->EXTI1  */
   *EXTI_IMR |= EXTI1_IRQ_BPOS;
   *EXTI_RTSR |= EXTI1_IRQ_BPOS;       /* Avbrott på POSITIV flank */
   *EXTI_FTSR &= ~EXTI1_IRQ_BPOS;      /* EJ Avbrott på negativ flank */
   *NVIC_ISER0 |= NVIC_EXTI1_IRQ_BPOS; /* Aktivera avbrott i NVIC */
}
```

b)

```
void irq_handler ( void )
{
   *GPIOC_ODRLOW = 0x10;         /* Återställ D-vippan */
   *GPIOC_ODRLOW = ~0x10;
   *EXTI_PR |= EXTI1_IRQ_BPOS;   /* Återställ EXTI1 "Pending Register" */
}
```

Uppgift C-5:

a)

```
typedef void (* vfunc )(void);
typedef void (* pfunc )(char);
typedef char (* rfunc )(void);
```

b)

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

c) Buss, stjärna och träd

Uppgift C-6:

```
int ortogonala(int *v1, int *v2, int n) {
   int sum=0;
   while( n>0 )
   {
      sum += *v1++ * *v2++;
      n--;
   }
   return sum==0;
}
```

Uppgift C-7:

```
/* filen driver.h */
#include "buffer.h"
void init_input(void);
buffer *get_input_buffer();

/* filen channel.h */
#define RECEIVE_DATA (*(unsigned char *) 0x400 )
```

```
#define RECEIVE_CTRL (*(unsigned char *) 0x401 )
#define RDY 0x01
#define GO  0x02
#define IE  0x04

/* filen driver.c */
#include "driver.h"
#include "channel.h"
#include "buffer.h"
#define BUF_SIZE 100

static buffer *input_buf = 0;
extern void input_intr(void);   /* avbrottsrutin */

void init_input(void) {
     if (!input_buf) {
        input_buf = create_buffer(BUF_SIZE);
        RECEIVE_CTRL = IE;
     }
}

buffer *get_input_buffer() {
  return input_buf;
}

void notify_input(void) {   /* anropas av input_intr  */
   if (RECEIVE_CTRL & RDY) {
     (void) put(input_buf, RECEIVE_DATA);
     RECEIVE_CTRL = GO & IE;
  }
}
```