Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D
EDA486 Programmering av inbyggda system Z
DAT015 Maskinorienterad programmering IT
DIT152 Programmering av inbyggda system GU

Fredag 23 Augusti 2013, kl. 8.30 - 12.30

Examinatorer
Roger Johansson, tel. 772 57 29
Jan Skansholm, tel. 772 10 12

Kontaktpersoner under tentamen
Roger Johansson/Jan Skansholm

Tillåtna hjälpmedel

Häftet

Instruktionslista för CPU12

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

C Reference Card

samt *en* av böckerna:

Vägen till C,
Bilting, Skansholm

The C Programming Language,
Kerninghan, Ritchie

Endast rättelser 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.

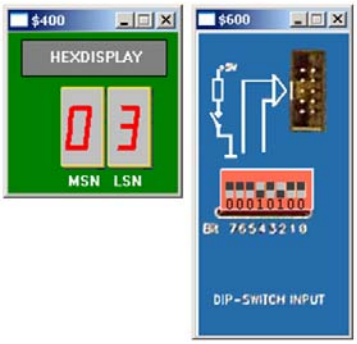
Allmänt
Siffror inom parentes anger full poäng på uppgiften. **För full poäng krävs att**:

- 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 (EDA/DAT):
20p≤betyg 3 < 30p ≤ betyg 4 < 40p ≤ betyg 5
respektive (DIT):
20p≤betyg G < 35p ≤ VG

Uppgift 1 (6p) *Assemblerprogrammering*



En 8-bitars strömbrytare, ”DIP_SWITCH” är ansluten till adress $600 och en displayenhet ”HEXDISPLAY” som visar en byte i form av två hexadecimala siffror är ansluten till adress $400 i ett MC12 mikrodatorsystem.

Konstruera en subrutin `DipHex` 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 utgör ettställda strömbrytare ska positionen för bit 2, (dvs. 3) skrivas till displayenheten.
Om ingen strömbrytare är ettställd ska siffran 0 skrivas till displayen.
Speciellt gäller att endast symboler ska användas för absoluta adresser.

Uppgift 2 (10p) *Användning av sammansatta datatyper/avbrottshantering*

Parallellporten `Port P`, i ett HCS12-system kan programmeras så att varje bit kan utgöra antingen en insignal, eller en utsignal. Portarna som används för insignaler kan dessutom konfigureras så att ett avbrott genereras då en yttre enhet ändrat värdet hos insignalen. Porten har tre olika register, som specificeras enligt följande:

Parallel port P (PORTP)											
Address		7	6	5	4	3	2	1	0	Mnemonic	Namn
$700	R	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	DDR	Data Direction Register
	W	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN		
$701	R	IF	IF	IF	IF	IF	IF	IF	IF	ICIE	Input Change Interrupt
	W	IEA	IEA	IEA	IEA	IEA	IEA	IEA	IEA		
$702	R	0	0	0	0	0	0	0	0	DATA	Data Register
	W	1	1	1	1	1	1	1	1		

- DDR: 1 anger att positionen är en utsignal, 0 anger att positionen är en insignal. Bitarna kan programmeras oberoende av varandra, dvs. godtycklig kombination av insignaler och utsignaler kan åstadkommas. Registret är både skrivbart och läsbart i sin helhet.
- ICIE: Består av olika delar (R=IF/W=IEA).
  - IEA (Interrupt Enable/Acknowledge). Biten är 0 efter RESET. Då 1 (Interrupt Enable) skrivs till biten aktiveras avbrottsgenerering vid ändring av motsvarande bit i DATA-registret om denna programmerats som insignal. Om motsvarande bit i DDR i stället programmerats som utsignal, genereras inga avbrott. IEA-biten har då ingen funktion. Då 1 skrivs till en bit som tidigare satts till 1, fungerar detta i stället som en Interrupt Acknowledge-funktion, dvs. IF (Interrupt Flag) nollställs. För att helt återställa avbrottsmekanismen för denna bit i DATA-registret skrivs 0 till IEA.
  - IF (Interrupt Flag) Biten är 0 efter RESET. Då motsvarande bit i DDR är programmerad som en insignal och motsvarande IEA är 1, sätts IF till 1 och ett avbrott (IRQ) genereras, avbrottsvektor FFF2.
- DATA: Består i själva verket av två olika register (R,W):
  - R: innehåller insignaler för de bitar som programmerats som insignaler. Endast 0 får skrivas, till en bit som är programmerad som insignal.
  - W: används då biten är programmerad som en utsignal. Då en bit som är programmerad som utsignal läses kommer detta alltid att resultera i värdet 1, oavsett vilket värde som tidigare skrivits till databiten.

a) Visa en lämplig deklaration av porten med användning av en `struct`. Visa också en funktion, `void portPinit(void)` som initierar port P, så att bitarna b_7-b_4 används som en 4-bitars inport och bitarna b_3-b_0 används som en 4-bitars utport. Då någon av inportens bitar ändras ska avbrott genereras. (3p)

b) Visa en funktion, `void outPortP(unsigned char c )` som matar ut bitarna b_3-b_0, av `c`, till port P. (1p)

c) Visa hur du implementerar en avbrottsfunktion, `void irqPortP( void )` som kvitterar ett avbrott från någon av portens ingångar. (3p)

d) Visa nödvändiga programdelar i assemblerspråk, dvs. hur avbrottsrutinen definieras, avbrottsvektorn initieras (antag att FFF2 är läs- och skrivbart minne) och hur processorn förbereds för att acceptera avbrotten i ett huvudprogram. Använd endast standard-C konstruktioner och/eller assemblerspråk för HCS12. (3p)

Uppgift 3 (10p) *Kodningskonventioner (C/assemblerspråk)*

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

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

```
void  func(unsigned int adam, char bertil, unsigned short * ceasar )
{
   char  a = bertil;
   int   b = adam;
   long  *c = ceasar;
   /* Övrig kod i funktionen är bortklippt eftersom vi bara
      betraktar anropskonventionerna. */
}
```

Översätt hela funktionen func, som den är beskriven till HCS12 assemblerspråk. 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):

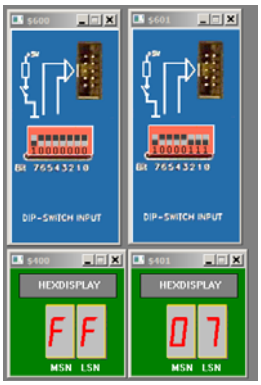
```
short *     c;
int         a;
char        b;
```

Visa hur variabeldeklarationerna översätts till assemblerdirektiv. Beskriv dessutom hur följande funktionsanrop översätts till assemblerkod. (4p)

```
func( a , b , c );
```

Uppgift 4 (6p) *In och utmatning beskriven 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 två displayenheter, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 och 0x401 i ett MC12 mikrodatorsystem.
Konstruera en funktion

```
void AddSigned8bitTo16( void )
```

som adderar de två värdena som läses från strömbrytarna (tolka som tal *med* tecken) och därefter presenterar resultatet som ett 16 bitars tal på displayindikatorerna.

Uppgift 5 (8p) *Programmering med pekare*

I standarden för C beskrivs funktionen `qsort` på följande sätt:

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

The `qsort` *function sorts an array of* `nmemb` *objects, the initial element of which is pointed to by* `base`*. The size of each object is specified by* `size`*.*

The contents of the array are sorted into ascending order according to a comparison function pointed to by `compar`*, which is called with two arguments that point to the objects being compared. The function shall return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two elements compare as equal, their order in the resulting sorted array is unspecified.*

The `qsort` *function returns no value.*

Din uppgift är nu att skriva en egen sorteringsfunktion, `mysort`, som fungerar på samma sätt som `qsort` och alltså kan anropas på samma sätt. Du behöver inte använda sorteringsalgoritmen *quicksort* utan kan utnyttja vilken algoritm som helst, t.ex. den enklare *bubble sort*. Du får inte anropa några standardfunktioner utan måste skriva all kod själv. Du får inte heller använda indexering utan måste utnyttja pekare. Tips: Arbeta internt i funktionen med typen **`char`**. (Du får förutsätta att en variabel av denna typ är en byte lång.)

Uppgift 6 (10p) *Maskinnära programmering i C*

I en vägtunnel bildas koloxid och en fläkt har till uppgift att vädra ut denna. Dessutom kan det samlas vatten längst ner i tunneln. En pump har därför till uppgift att då och då pumpa bort vatten så att vägbanan inte blir vattenfylld.

Uppgiften är att skriva ett program som övervakar och kontrollerar koloxidhalten och vattennivån med hjälp av fläkten och pumpen. Programmet skall dessutom reglera trafikljus så att trafiken kan stängas av tillfälligt om koloxidhalten eller vattennivån blir för hög. Om trafiken har stoppats skall den åter släppas fri om när båda de avlästa värdena ligger på en ofarlig nivå.

Fläkten och pumpen startas och stoppas via var sitt kontrollregister. Det finns dessutom två A/D-omvandlare: en som avläser koloxidhalten och en som mäter vattennivån. Till varje A/D-omvandlare finns dels ett kontrollregister som används för att initiera en avläsning och dels ett dataregister i vilket det avlästa värdet placeras. Trafikljuset regleras också med ett kontrollregister. Alla register består av 16 bitar.

Registren har följande adresser:

`A/D-omvandlare för koloxid, kontrollregister:`	$AA10_{16}$
`A/D-omvandlare för koloxid, dataregister:`	$AA12_{16}$
`Kontrollregister för fläkt:`	$AA14_{16}$
`A/D-omvandlare för vattennivå, kontrollregister:`	$AA16_{16}$
`A/D-omvandlare för vattennivå, dataregister:`	$AA18_{16}$
`Kontrollregister för pump:`	$AA1A_{16}$
`Kontrollregister för trafikljus:`	$AA1C_{16}$

Observera att kontrollregistren *inte* är läsbara. Man kan bara skriva till dem. Du måste därför använda dig av skuggregister.

Fläkten och pumpen startas och stoppas genom att en etta resp. nolla skrivs i bit nummer 10 i dess kontrollregister. När man skall göra en avläsning med hjälp av en A/D-omvandlare skriver man en etta i bit nummer 10 i tillhörande kontrollregister. Man måste sedan vänta en tiondels sekund innan värdet i dataregistret har stabiliserats. (Ingen avbrottsmekanism används.)

A/D-omvandlarna ger normalt värden i intervallet 0 till 1023. Ett negativt värde i dataregistret indikerar att en avläsning misslyckats. För koloxidhalten gäller att värden i intervallet 0-200 betraktas som låga, värden i intervallet 201-400 som höga och ännu högre värden som farliga. För vattennivån gäller på motsvarande sätt att värden i intervallet 0-300 betraktas som låga, värden 300-799 som höga och högre värden som farliga.

Om man skriver en etta i bit nummer 15 i kontrollregistret för trafikljuset sätts ljuset till rött. Skriver man en nolla blir ljuset grönt.

Din uppgift är att skriva övervakningsprogrammet. Till din hjälp skall du använda realtidskärnan som presenterats på en av föreläsningarna. Filen `process.h` visas i bilagan och du får anropa alla funktioner som deklareras i denna.

Programmet skall innehålla tre processer: en som avläser koloxidhalten, en som avläser vattennivån och en som styr trafikljuset. Alla skall exekvera utan att avslutas. Koloxidhalten skall avläsas var 20:e sekund och vattennivån var 30:e. Om någon avläsning misslyckas skall programmet ge en varningsutskrift, men sedan hoppa över den aktuella avläsningen och fortsätta normalt. Utskrifter från programmet kan för enkelhets skull göras med standardfunktionen `printf`.

Om programmet när det avläser koloxidhalten eller vattennivån finner att denna är på den nivå som betraktas som hög eller farlig skall fläkten resp. pumpen startas (om den inte redan är igång). Om det avlästa värdet är på den nivå som betraktas som låg skall fläkten resp. pumpen, (om den är igång) stoppas. Om ett avläst värde för koloxiden eller vattennivån är farligt högt skall detta rapporteras så att trafiken kan stoppas. Rapporteringen med hjälp av två globala variabler (tillgängliga för alla funktioner) av typen _Bool som anger om koloxiden resp. vattennivån är farligt hög.

Den process som styr trafikljuset skall regelbundet, var 10:e sekund, undersöka de globala variablerna och om så behövs slå på rött eller grönt ljus.

Det är viktigt att bara en process i taget kan avläsa och ändra någon av de två globala variablerna. Därför skall man betrakta dem som en delad resurs och skydda dem med en semafor.

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 – ”process.h”

```
// Filen process.h
#ifndef PROCESS_H
  #define PROCESS_H

  #define DEFAULT_STACK_SIZE 128
  #define MINIMUM_PRIORITY 1
  #define DEFAULT_PRIORITY MINIMUM_PRIORITY+9

  typedef struct process_struct process;      // definieras i filen process.c
  typedef struct semaphore_struct semaphore;  // definieras i filen process.c
  typedef void (*function)(void);

  extern void init_processes(void);
  extern process *create_process(function f, int prio, int stack_size);
  extern void start_process(process *);
  extern process *running_process(void);
  extern int get_process_id(process *);
  extern unsigned long int get_time(void);    // resultat ges i ms
  extern void delay_process(process *p, long int t);    // t ges i ms
  extern int  get_process_priority(process *);
  extern void set_process_priority(process *p, int prio);
  extern void terminate_process(process *p);
  extern int terminated(process *p);
  extern semaphore *create_semaphore(int init_value);
  extern void signal(semaphore *);
  extern void wait(semaphore *);

  // macron för förenklade funktionsanrop (i brist på överlagrade funktioner)
  #define new_process(f)       create_process((f), DEFAULT_PRIORITY, DEFAULT_STACK_SIZE)
  #define get_id()             get_process_id(running_process())
  #define delay(t)             delay_process(running_process(), (t))
  #define get_priority()       get_process_priority(running_process())
  #define set_priority(i)      set_process_priority(running_process(), (i));
  #define terminate()          terminate_process(running_process())
#endif
```

Lösningsförslag

Uppgift 1:

```
; Symboliska adresser
DipSwitch EQU   $600
HexDisp   EQU   $400

; Subrutin DipHex
DipHex:   LDAA  DipSwitch
          CLRB

DipHex10: TSTA
          BEQ   DipHex20

          INCB
          LSRA
          BCC   DipHex10

DipHex20: STAB  HexDisp
          RTS
```

Uppgift 2a (3p):

```
typedef    struct sPortP{
       volatile unsigned char ddr;
       volatile unsigned char icie;
       volatile unsigned char data;
}PORTP;
#define      PORTP_BASE   0x700
#define      portP ((PORTP *)(PORTP_BASE))

void portPinit( void )
{
       portP->ddr = 0x0F;     /* b7-b4 inport, b3-b0 utport */
       portP->icie = 0xF0; /* b7-b4 inportar, avbrott aktiveras */
}
```

Uppgift 2b (1p):

```
void outPortP( unsigned char c )
{
       portP->data = c & 0x0F ;   /* b7-b4 ska vara 0 */
}
```

Uppgift 2c (3p):

```
void   irqPortP( void )
{
       switch( portP->data & 0cF0 )     /* bestäm avbrottskälla */
   {   /* kvittera avbrott */
    case   0x80:  portP-> icie = 0x80; break;
    case   0x40:  portP-> icie = 0x40; break;
    case   0x20:  portP-> icie = 0x20; break;
    case   0x10:  portP-> icie = 0x10; break;
   }
}
```

Uppgift 2d (3p):

```
Assembler:
   ; initieringar i huvudprogram...
       IMPORT _irqPortP

       MOVW      #PortPirq,$FFF2
       CLI

; avbrottsrutin
PortPirq:
       JSR    _irqPortP
       RTI
```

Uppgift 3a:

Beskrivning av aktiveringspost

minnesanvändning	*stackoffset*
ceasar	10,SP
bertil	9,SP
adam	7,SP
PC (vid JSR)	
a	4,SP
b	2,SP
c	0,SP

```
_func:
    LEAS   -5,SP
;  char   a = bertil;
    LDAB   9,SP
    STAB   4,SP
;  int b = adam;
    LDD    7,SP
    STD    2,SP
;  long   *c = ceasar;
    LDD    10,SP
    STD    0,SP

    LEAS   5,SP
    RTS
```

Uppgift 3b:

```
_c:    RMB 2
_a: RMB 2
_b:    RMB 1

    LDD _c
    PSHD
    LDAB   _b
    PSHB
    LDD _a
    PSHD
    JSR _func
    LEAS   5,SP
```

Uppgift 4:

```
typedef  char    *port8ptr;
typedef  short   *port16ptr;

#define ML4OUT_ADR 0x400
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT16 *((port16ptr) ML4OUT_ADR)

#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void   AddSigned8bitTo16( void )
{
       short s;
       while( 1 )
       {
          s = ( short ) ML4IN1;
          s = s + ( short ) ML4IN2;
          ML4OUT16 = s;
       }
}
```

Uppgift 5:

```
void mycpy(char *to, const char *from, size_t size) {
  while (size-- > 0)
    *to++ = *from++;
}

void mysort(void *base, size_t n, size_t size,
       int (*compare) (const void *, const void *)) {

   char *b = base;
   char temp[size];
   _Bool swapped;
   do {
      swapped = 0;
      for (char *p = b;  p < b+(n-1)*size; p += size) {
         if (compare(p, p+size) > 0) {
            mycpy(temp, p, size);
            mycpy(p, p+size, size);
            mycpy(p+size, temp, size);
            swapped = 1;
         }
      }
   } while (swapped);
}
```

Uppgift 6:

```
// Filen tunnel.h
typedef short int port;      // antar 16-bitars short int
typedef port *portptr;

// Användbara makron
#define set(r, mask)    (r) = (r) | mask
#define clear(r, mask)  (r) = (r) & ~mask

#define CO_CTRL_ADR        0xaa10
#define CO_DATA_ADR        0xaa12
#define FAN_CTRL_ADR       0xaa14
#define WATER_CTRL_ADR     0xaa16
#define WATER_DATA_ADR     0xaa18
#define PUMP_CTRL_ADR      0xaa1A
#define SIGNAL_CTRL_ADR    0xaa1C

#define CO_CTRL            *((portptr) CO_CTRL_ADR)
#define CO_DATA            *((portptr) CO_DATA_ADR)
#define FAN_CTRL           *((portptr) FAN_CTRL_ADR)
#define WATER_CTRL         *((portptr) WATER_CTRL_ADR)
#define WATER_DATA         *((portptr) WATER_DATA_ADR)
#define PUMP_CTRL          *((portptr) PUMP_CTRL_ADR)
#define SIGNAL_CTRL        *((portptr) SIGNAL_CTRL_ADR)

#define device_operation_bit      0x0400
#define signal_operation_bit      0x8000

// Filen tunnel.c
#include "tunnel.h"
#include "process.h"
#include <stdio.h>
#include <limits.h>

#define DANGER_CO 401
#define HIGH_CO   201
#define DANGER_WATER 801
#define HIGH_WATER   301

static  _Bool danger_co  = 0;
static  _Bool danger_water = 0;
static  semaphore *data_sem;
```

```
void co_monitor(void) {
   port co_level, fan_ctrl=0;
   _Bool fan_running = 0;
   while (1) {
      set(fan_ctrl, device_operation_bit);
      CO_CTRL = fan_ctrl;                  // starta avläsning av koloxid
      delay(100);
      co_level = CO_DATA;
      if (co_level < 0)
         printf("Fel på läsare av koloxidnivån");
      if (co_level < HIGH_CO && fan_running) {
         clear(fan_ctrl, device_operation_bit);
         FAN_CTRL = fan_ctrl;
         fan_running = 0;
      }else if (co_level >= HIGH_CO && !fan_running) {
         set(fan_ctrl, device_operation_bit);
         FAN_CTRL = fan_ctrl;
         fan_running = 1;
      }
      wait(data_sem);
      danger_co = co_level >= DANGER_CO;
      signal(data_sem);
      delay(20000);
   }
}

void water_monitor(void) {
   port water_level, pump_ctrl=0;
   _Bool pump_running = 0;
   while (1) {
      set(pump_ctrl, device_operation_bit);
      WATER_CTRL = pump_ctrl; // starta avläsning av vattennivån
      delay(100);
      water_level = WATER_DATA;
      if (water_level < 0)
         printf("Fel på läsare av vattennivån");
      if (water_level < HIGH_WATER && pump_running) {
         clear(pump_ctrl, device_operation_bit);
         PUMP_CTRL = pump_ctrl;
         pump_running = 0;
      }else if (water_level >= HIGH_WATER && !pump_running){
         set(pump_ctrl, device_operation_bit);
         PUMP_CTRL = pump_ctrl;
         pump_running = 1;
      }
      wait(data_sem);
      danger_water = water_level >= DANGER_WATER;
      signal(data_sem);
      delay(30000);
   }
}

void traffic_monitor(void) {
   port signal_ctrl=0;
   _Bool red_light = 0;
   clear(signal_ctrl, signal_operation_bit);
   while(1) {
      wait(data_sem);
      if ((danger_co || danger_water) && !red_light) {
         set(signal_ctrl, signal_operation_bit);
         SIGNAL_CTRL = signal_ctrl;
         red_light = 1;
      }else if ((!danger_co && !danger_water) && red_light) {
         clear(signal_ctrl, signal_operation_bit);
         SIGNAL_CTRL = signal_ctrl;
         red_light = 0;
      }
   signal(data_sem);
   delay(10000);
   }
}
```

```
int main() {
    process *co_proc, *water_proc, *traffic_proc;
    init_processes();
    data_sem = create_semaphore(1);
    co_proc = new_process(co_monitor);
    water_proc = new_process(water_monitor);
    traffic_proc = new_process(traffic_monitor);
    start_process(co_proc);
    start_process(water_proc);
    start_process(traffic_proc);
    delay(UINT_MAX);
}
```