Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

LEU500 – Maskinorienterad programmering

Exempel 3

Examinator
Roger Johansson, tel. 772 57 29

Kontaktpersoner under tentamen
Roger Johansson

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.

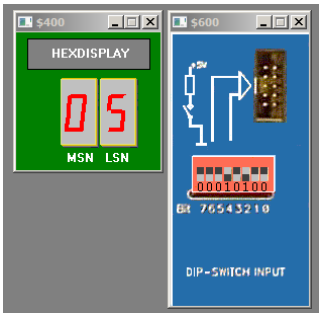
Allmänt
Siffror inom parentes anger full poäng på uppgiften. **För full poäng krävs att**:

- 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:
20p≤betyg 3 < 30p ≤ betyg 4 < 40p ≤ betyg 5

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 `DipHexReversed` som läser av strömbrytaren och indikerar den mest 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 4, (dvs. 5) 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 (8p) *Användning av sammansatta datatyper/avbrottshantering*

Följande figur beskriver en förenklad variant av CRG-modulen med de register som används för den enkla räknaren hos HCS12:

Clock Reset Generator (CRG)											
Offset		7	6	5	4	3	2	1	0	Mnemonic	Namn
$30	R	RTIF	PORF	LVRF	LOCKIF	LOCK	SCMIE	SCMIF	SCM	CRGFLG	Flags Register
	W										
$31	R	RTIE	0	0	LOCKIE	0	0	SCMIE	0	CRGINT	Interrupt Enable Register
	W										
$32	R	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0	RTICTL	RTI Control Register
	W										

Detta system har en 10 MHz oscillator. Räknaren ska programmeras för att generera periodiska avbrott med c:a 10ms intervall. *Ledning*: 3×2^{15} pulser/period ger tillräcklig noggrannhet. Räknaren använder avbrottsmekanismen hos HCS12, kopplad till vektor 0x3FF2.

RTR [3:0]	RTR[6:4]							
	000 (OFF)	001	010	011	100	101	110	111
0000	OFF	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
0001	OFF	$2x2^{10}$	$2x2^{11}$	$2x2^{12}$	$2x2^{13}$	$2x2^{14}$	$2x2^{15}$	$2x2^{16}$
0010	OFF	$3x2^{10}$	$3x2^{11}$	$3x2^{12}$	$3x2^{13}$	$3x2^{14}$	$3x2^{15}$	$3x2^{16}$
0011	OFF	$4x2^{10}$	$4x2^{11}$	$4x2^{12}$	$4x2^{13}$	$4x2^{14}$	$4x2^{15}$	$4x2^{16}$
0100	OFF	$5x2^{10}$	$5x2^{11}$	$5x2^{12}$	$5x2^{13}$	$5x2^{14}$	$5x2^{15}$	$5x2^{16}$
0101	OFF	$6x2^{10}$	$6x2^{11}$	$6x2^{12}$	$6x2^{13}$	$6x2^{14}$	$6x2^{15}$	$6x2^{16}$
0110	OFF	$7x2^{10}$	$7x2^{11}$	$7x2^{12}$	$7x2^{13}$	$7x2^{14}$	$7x2^{15}$	$7x2^{16}$
0111	OFF	$8x2^{10}$	$8x2^{11}$	$8x2^{12}$	$8x2^{13}$	$8x2^{14}$	$8x2^{15}$	$8x2^{16}$
1000	OFF	$9x2^{10}$	$9x2^{11}$	$9x2^{12}$	$9x2^{13}$	$9x2^{14}$	$9x2^{15}$	$9x2^{16}$
1001	OFF	$10x2^{10}$	$10x2^{11}$	$10x2^{12}$	$10x2^{13}$	$10x2^{14}$	$10x2^{15}$	$10x2^{16}$
1010	OFF	$11x2^{10}$	$11x2^{11}$	$11x2^{12}$	$11x2^{13}$	$11x2^{14}$	$11x2^{15}$	$11x2^{16}$
1011	OFF	$12x2^{10}$	$12x2^{11}$	$12x2^{12}$	$12x2^{13}$	$12x2^{14}$	$12x2^{15}$	$12x2^{16}$
1100	OFF	$13x2^{10}$	$13x2^{11}$	$13x2^{12}$	$13x2^{13}$	$13x2^{14}$	$13x2^{15}$	$13x2^{16}$
1101	OFF	$14x2^{10}$	$14x2^{11}$	$14x2^{12}$	$14x2^{13}$	$14x2^{14}$	$14x2^{15}$	$14x2^{16}$
1110	OFF	$15x2^{10}$	$15x2^{11}$	$15x2^{12}$	$15x2^{13}$	$15x2^{14}$	$15x2^{15}$	$15x2^{16}$
1111	OFF	$16x2^{10}$	$16x2^{11}$	$16x2^{12}$	$16x2^{13}$	$16x2^{14}$	$16x2^{15}$	$16x2^{16}$

Programpaketet ska bestå av delar implementerade såväl i assemblerspråk som i ’C’. Fyra olika funktioner ska implementeras:

En initieringsrutin, i form av en C-funktion: `void RTInit(void)`
En servicerutin för avbrottet, i form av en C-funktion: `void AtRTIrq(void)`
En avbrottsrutin `RTIRQ`, i assemblerspråk som anropar servicerutinen
En assemblerrutin `CLEARI` som nollställer avbrottsmasken hos HCS12

a) Visa en lämplig typdeklaration i form av en ’C’-`struct` för CRG-modulen, enligt figuren ovan.(2p)

b) Visa assemblerrutinen `CLEARI` och avbrottsrutinen `RTIRQ`. (2p)

c) Konstruera `RTInit` som:
initierar räknaren för att generera avbrott med den angivna periodtiden
förbereder HCS12 för att hantera avbrott från räknaren.

Typdeklarationen från uppgift a) ska användas.(3p)

d) Konstruera `AtRTIrq` som kvitterar ett avbrott från räknaren. Typdeklarationen från uppgift a) ska användas.(1p)

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(long adam, unsigned char bertil, struct one * ceasar )
{
   long          a = adam;
   unsigned int  b = bertil;
   struct one    *c = ceasar;
   /* 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):

```
struct one * c;
long        a;
char        b;
```

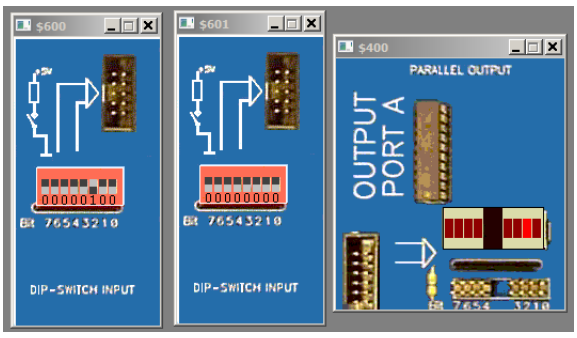
Visa hur variabeldeklarationerna översätts till assemblerdirektiv. Beskriv dessutom hur följande funktionsanrop översätts till assemblerkod. (4p)

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

Uppgift 4 (8p) *In och utmatning beskriven i C*

I denna uppgift ska du bland annat demonstrera hur absolutadressering utförs i C. För full poäng måste du visa hur preprocessordirektiv och typdeklarationer då 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 CondRunDiode ( void )
```

- som oupphörligt jämför strömbrytarnas värden
- dessutom skriver ut ett *rinnande ljus* på diodrampen.

Det rinnande ljuset består i att en diod i taget tänds upp.

- Om värdet hos strömbrytaren på adress 0x600 är störst ska ljuset rinna från höger till vänster
- Om värdet hos strömbrytaren på adress 0x601 är störst ska ljuset rinna från vänster till höger
- Om värdena är lika ska det rinnande ljuset stannas.

Från början ska dioden längst till vänster vara tänd.
Du behöver här *inte* ta hänsyn till att fördröjningar krävs för det rinnande ljuset.

Uppgift 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 några färdiga standardfunktioner, utan du måste skriva allt själv.

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

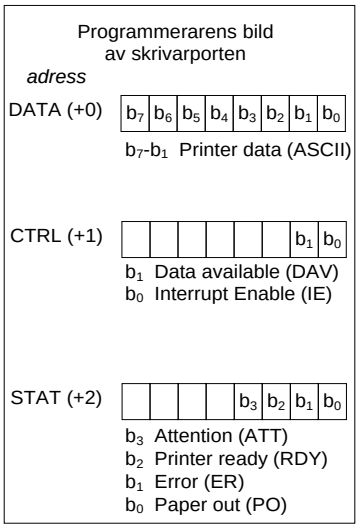
```
#define N 10
main() {
  int i;
  char txt[] = "Nu tentar  vi  MOP! ";
  char *tokens[N];
  split(txt, tokens, N);
  for (i=0; i< N && tokens[i]; i++)
    printf("%s\n", tokens[i]);
}
```

Utskriften från programmet skall bli:

```
Nu
tentar
vi
MOP!
```

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

Till en enkel skrivare hör de tre åttabitars register som visas i vidstående figur. Basadressen är 600 hexadecimalt. För att starta utskrift av ett tecken placeras tecknet i dataregistret och därefter skrivs en etta i bit 0 i kontrollregistret. Om man vill att skrivaren skall generera ett avbrott när utskriften är klar skall även bit 1 i kontrollregistret sättas. Om man angivit att avbrott skall användas skall avbrott kvitteras genom att bitarna 0 och 1 i kontrollregistret nollställs. När ett tecken skrivits klart sätter skrivaren en etta i bit 2 i statusregistret. Bitarna 0 och 1 sätter skrivaren om pappret är slut eller något fel inträffat. Bit 3 i statusregistret sätts om någon av bitarna 0 till 2 är satt. (Bitarna 0 till 2 i statusregistret visas även med lysdioder på själva skrivaren så att användaren kan se vilken status skrivaren befinner sig i. Det finns också en reset-knapp på skrivaren som användaren kan trycka på för att generera ett avbrott.)



Avbrottsvektorn som hör till skrivaren har adressen 3FF4 hexadecimalt.

Assemblerrutinen som visas här är given:

```
* Filen asm.s12
    segment init
    export  _printtrap
    export  _sei
    export  _cli
    import _print_inter
_printtrap: jsr _print_inter
            rti
_sei:       sei
            rts
_cli:       cli
            rts
```

Din uppgift är att, helt i C, skriva en modul (både .h fil och .c fil) som styr skrivaren. Du måste också ev. skriva någon ytterligare fil för att det skall gå att kompilera dina filer.
För att undvika busy-wait skall modulen internt använda en kö. Det finns en färdigskriven kö-modul som kan användas. Denna har include-filen:

```
// Filen queue.h
#ifndef QUEUE_H
#define QUEUE_H
struct qstruct;                 // qstruct definieras i .c filen
typedef struct qstruct *Queue;  // typen Queue
typedef unsigned char Data;     // typen Data
Queue new_queue();              // skapar en ny kö
void delete_queue(Queue q);     // tar bort kön helt och hållet
void clear(Queue q);            // tar bort köelementen men behåller kön
int  size(Queue q);             // ger köns aktuella längd
int  add(Queue q, Data c);  //lägger in c sist i kön, ger 1 om OK, 0 annars
int copy_first(Queue q, Data *pc); //kopierar första elementet dit pc pekar
                                   //ändrar inte kön, ger 1 om OK, 0 annars
void remove_first(Queue q);        //tar bort det första elementet
#endif
```

Din modul skall innehålla följande två interna hjälpfunktioner, vilka *inte* skall kunna anropas utifrån:

- `init_printer` , skall om den inte anropats tidigare initiera avbrottsvektorn för skrivaren, skapa en ny kö och nollställa processorns interrupt-flagga.
- `print_next`, skall kontrollera att skrivaren är redo att ta emot ett nytt tecken och i så fall hämta det första tecknet från kön och initiera utskrift av tecknet. Om kön är tom, papperet är slut eller om det är något fel på skrivaren skall inget göras.

Modulen skall ha två funktioner som kan anropas från andra delar av programmet:

- `print` , har en parameter av typen `unsigned char`, ger som resultat värdet 1 eller 0 beroende på om utskriften lyckades eller inte. Skall initiera skrivaren (om så behövs) och lägga tecknet som gavs som parameter sist i kön. Skall initiera utskrift av nästa tecken om så behövs.

`print_inter`, saknar parametrar och resultat, anropas från avbrottsrutinen när avbrott skett. Skall kvittera avbrottet och initiera utskrift av nästa tecken.

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)

Lösningsförslag

Uppgift 1 (6p):

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

; Subrutin DipHexReversed

DipHexReversed:
             CLRB
             LDAA      DipSwitch
             BEQ       DipHexReversed20
             LDAB      #9

DipHexReversed10:
             DECB
             BEQ       DipHexReversed20
             LSLA
             BCC       DipHexReversed10

DipHexReversed20:
             STAB   HexDisp
             RTS
```

Uppgift 2a (2p):

```
typedef struct sCRG{
       volatile unsigned char crgflg;
       volatile unsigned char crgint;
       volatile unsigned char rtictl;
}CRG, *PCRG ;
```

Uppgift 2b (2p):

```
CLEARI:   CLI
          RTS

RTIRQ: JSR AtRTIrq
          RTI
```

Uppgift 2c (3p):

```
void RTInit(void)
{
   extern void AtRTIrq(void);
   extern void CLEARI(void);
   ((PCRG) (0x30))->rtictl = 0x62; /* periodtid 10 ms: 0110 0010 */
   ((PCRG) (0x30))->crgint = 0x80; /* RTIE <-1: aktivera avbrott */
   *(unsigned short *) 0x3FF2 = AtRTIrq;
   CLEARI();

}
```

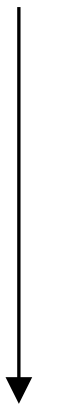
Uppgift 2d (1p):

```
void AtRTIrq(void)
{
   ((PCRG) (0x30))->rtictl = 0x80; /* RTIE <- 1: kvittera avbrott  */
}
```

Uppgift 3a (6p):

Beskrivning av aktiveringspost

minnesanvändning	*adress*
ceasar	15,SP
bertil	14,SP
adam	10,SP
PC (vid JSR)	
a	4,SP
b	2,SP
c	0,SP




```
;  void   func(long adam, unsigned char bertil, struct one * ceasar )
_func:
;  {
   LEAS   -8,SP
;  long             a = adam;
   LDY    10,SP
   LDD    12,SP
   STY    4,SP
   STD    6,SP
;  unsigned int b = bertil;
   LDAB   14,SP
   CLRA
   STD    2,SP
;  struct one   *c = ceasar;
   LDD    15,SP
   STD    0,SP
;  /* Övrig kod i funktionen är bortklippt eftersom vi bara
;     betraktar anropskonventionerna. */
;  }
   LEAS   8,SP
   RTS
```

Uppgift 3b (4p):

```
_c:    RMB    2
_a:    RMB    4
_b:    RMB    1
       LDD    _c
       PSHD
       LDAB   _b
       PSHB
       LDD    2+_a
       PSHD
       LDD    _a
       PSHD
       JSR    _func
       LEAS   7,SP
```

Uppgift 4 (8p):

```
typedef   unsigned char * port8ptr;
#define DISPLAY *((port8ptr) 0x400)
#define DIPSWITCH1 *((port8ptr) 0x600)
#define DIPSWITCH2 *((port8ptr) 0x601)

void   CondRunDiode( void )
{
   unsigned char value;
   value = 0x80;             /* initialvärde */
   while( 1 )
   {
      if( DIPSWITCH1 > DIPSWITCH2 )
      {  /* ljus rinner åt vänster */
         DISPLAY = value;
         value = value << 1;
         if( value == 0 ) /* över kanten? ... */
            value = 1;          /* böja om från höger */
      }else if ( DIPSWITCH1 < DIPSWITCH2 )
      {  /* ljus rinner åt höger */
         DISPLAY = value;
         value = value >> 1;
         if( value == 0 ) /* över kanten? ... */
            value = 0x80;   /* böja om från vänster */

      }else     /* ljus står still */
         DISPLAY = value;
   }
}
```

Uppgift 5 (8p):

```
   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 6 (10p):

```
/* filen asm.h, ges av 'asm.s12' och behövs för C-rutinerna */
void printtrap(void);   // avbrottrutin, anropar print_inter
void sei(void);         // sätter avbrottsflaggan i processorn
void cli(void);         // nollställer avbrottsflaggan i processorn

/* .c */
#include "queue.h"
#include "asm.h"
#include <stddef.h>

typedef unsigned char port;
typedef port *portptr;
#define DATA_REG (*(portptr) 0x600 )
#define CTRL_REG (*(portptr) 0x601 )
#define DAV         2
#define IE          1
#define STAT_REG (*(portptr) 0x602 )
#define RDY         4
#define ER          2
#define PO          1

typedef void (*vec) (void);
typedef vec *vecptr;
#define PRINT_VEC_ADR 0x3FF4
#define PRINT_VEC *((vecptr) PRINT_VEC_ADR)

static Queue q = NULL;

static void print_next() {
    unsigned char c;
    if (! (STAT_REG & RDY))
      return;      /* Skrivaren upptagen med utskrift, avbrott kommer när tecknet är klart
*/
    else if ((STAT_REG & (ER | PO)))
      return;      /* Fel, kan inget göra här */
    else if (copy_first(q, &c)) { /* hämta nästa tecken ur kön */
        remove_first(q);
        DATA_REG = c;
        CTRL_REG = DAV | IE;
    }
}

static void init_printer(void) {
    if (q == NULL) {
        q = new_queue();
        PRINT_VEC = printtrap;
        cli();
    }
}

void print_inter() {
     CTRL_REG = 0;
     print_next();
}

int print(unsigned char c) {
    init_printer();
    if (!add(q, c))
      return 0;
    if (size(q) == 1)
      print_next();
    return 1;
}

/* .h-fil med prototypdeklarationer */
extern void print_inter(void);    // anropas vid avbrott
extern int print(unsigned char);  // ger 1 om OK, 0 annars
```