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 22 augusti 2014, kl. 8.30 - 12.30, Maskinsalar

Examinator
Roger Johansson, tel. 772 57 29

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

Uppgift 1 (12p) *Användning av sammansatta datatyper/maskinnära programmering*

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 2 (6p) *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 *b, char a )
{
   char  c;
   char  *d; ....
```

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

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

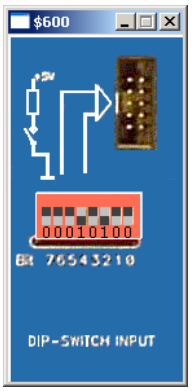
a) Visa hur variabeldeklarationerna på toppnivå översätts till assemblerdirektiv för HCS12.

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 3 (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 eventuellt typdeklarationer används för att skapa begriplig programkod.

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

```
void ff1( void )
```

som oavbrutet 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.

Uppgift 4 (8p) *Programmering med pekare*

Standardfunktionen `strspn` kan beskrivas på följande sätt:

`int strspn ( const char * str1, const char * str2 );`

Returns the length of the initial portion of *str1* which consists only of characters that are part of *str2*.

Parameters
str1 C string to be scanned.
str2 C string containing the characters to match.
Return value
The length of the initial portion of *str1* containing only characters that appear in *str2*. Therefore, if all of the characters in *str1* are in *str2*, the function returns the length of the entire *str1* string, and if the first character in *str1* is not in *str2*, the function returns zero.

Exempel på användning:

```
int main()
{
  char string[]="7803 Elm St.7";
  printf("The number length is %d.\n",strspn(string,"1234567890"));
  return 0;
}
```

Utskriften ska då bli: ”The number length is 4”.

Din uppgift är att, i C, skriva en egen definition av funktionen `strspn`. Du får inte använda dig av indexering, utan måste utnyttja pekare. Du får inte anropa någon annan standardfunktion. Du måste alltså skriva all kod själv.

Uppgift 5 (8p) *Assemblerprogrammering*

Följande funktion `print` finns given i 'C'. Skriv motsvarande funktion `PRINT` i assemblerspråk för HCS12. I denna uppgift ska du *inte* ta hänsyn till kompilatorkonventioner utan i stället skickas parametern `s` till `PRINT` i register `X`.

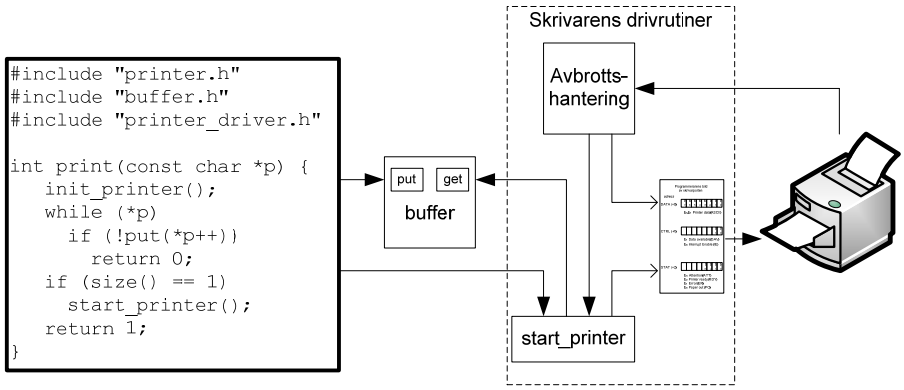
```
void print ( char *s )
{
   while( *s )
   {
      while( (*( char *) 0x701) & 4 )
      {}
      *( char *) 0x700 = *s;
      s++;
   }
}
```

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

Implementering av rutiner för "buffrad" utskrift till en enkel skrivare.

I bilaga 3 beskrivs gränssnittet till en mycket enkel skrivare. I denna uppgift ska du skriva modulen (både .h och .c filer) "drivrutiner" för denna skrivare. I följande figur illustreras sambanden mellan programdelarna. Funktionen "print" visar hur en textsträng skrivs ut till skrivaren. En modul för bufferthanteringen finns given (du skall alltså inte skriva denna). Buffertmodulen specificeras enligt följande:

```
/* buffer.h */
extern int put(unsigned char);        /* lägg tecken i buffert, returnera 0 om fel, 1 annars */
extern int get(unsigned char *);      /* ta tecken från buffert, returnera 0 om fel, 1 annars */
extern int size();          /* returnera antalet tecken som för tillfället finns i bufferten */
```



Modulen med skrivarens drivrutiner implementeras i tre filer: en header-fil (`print_driver.h`), en fil med källtext för C-rutinerna (`print_driver.c`) samt en fil med assemblerrutinen för avbrottshanteringen (`print_low.asm`).

En parameterlös funktion `init_printer` ska initiera avbrottsvektorn som har adressen 0x3FF2.

En parameterlös funktion `start_printer`, anropas från "print" och från avbrottshantering,

1. ska kontrollera att inget fel uppstått. Om så är fallet (*Error*, *Paper Out*) ska inga fler tecken skrivas ut.
2. ska kontrollera att skrivaren är klar att ta emot tecken, om inte ska inget tecken skrivas ut.
3. om inget fel inträffat och skrivaren är beredd, ska funktionen hämta ett tecken från bufferten och skriva ut det till skrivaren.

En avbrottsrutin `atIRQ`, ska

1. kvittera avbrott
2. anropa `start_printer`
3. återgå från avbrottshantering

Printerportens basadress är 0x800.

- Skriv rutinerna `init_printer` och `start_printer` i ANSI-C.
- Skriv avbrottshanteringsrutinen `atIRQ`.

För full poäng ska du visa hur preprocessordirektiv och ev. typdeklarationer används för att skapa begriplig programkod samt tydligt ange i vilken typ av fil (`.h`, `.c` eller `.asm`) varje lösningsdel ska placeras.

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

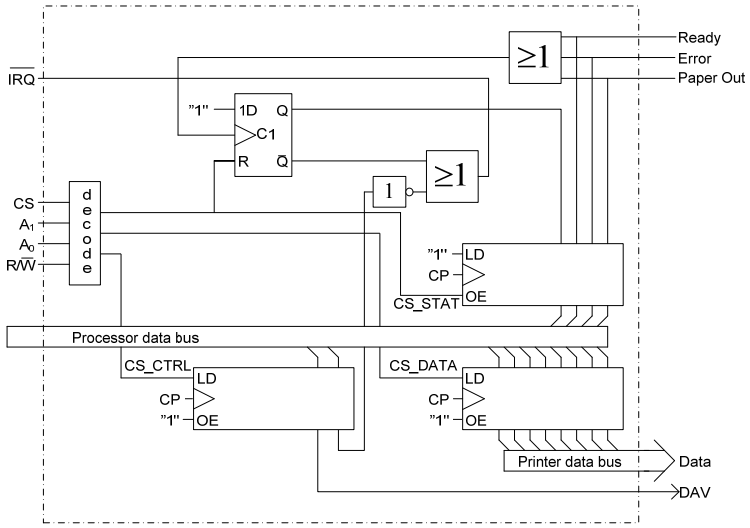
Bilaga 2 - Assemblerdirektiv för MC68HC12.

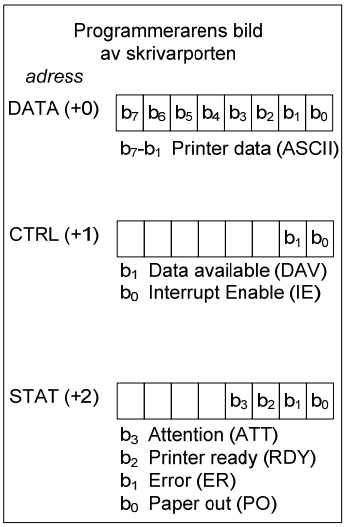
Assemblerspråket använder sig av mnemoniska beteckningar som tillverkaren Freescale specificerat för maskininstruktioner och instruktioner till assemblatorn, s.k. pseudoinstruktioner eller assemblerdirektiv. Pseudoinstruktionerna framgår av följande tabell:

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: Gränssnitt till enkel skrivare

Fökjande figur illustrerar ett gränssnitt mot en mycket enkel skrivare.





Skrivaren har följande statussignaler, samtliga ”aktivt höga”:

- Ready: Skrivarens buffert är tom och skrivaren är klar att ta emot ett nytt tecken.
- Error: Något internt fel har uppstått i skrivaren, felet måste åtgärdas innan utskrift kan fortsätta.
- Paper Out: Pappersmagasinet är tomt, måste åtgärdas innan utskrift kan fortsätta.
- Attention: Någon av signalerna ”Ready”, ”Error”, ”Paper Out” är aktiverad.

Skrivaren genererar IRQ (Interrupt ReQest) om signalen ”Attention” aktiveras samtidigt som biten ”Interrupt Enable (IE) i styrregistret (CTRL), är ett. Programvara måste användas för att klarlägga orsaken till avbrottsbegäran.

Skrivaren kan fås att skriva ut ett ACII-tecken med hjälp av följande handskakningsprotokoll:

HÄNDELSER		
	mikrodator	*skrivare*
t_1	kontrollera om skrivaren är beredd aktivera skrivare för utskrift	
t_2		skrivaren känner av signalen ”data finns” och blir ”upptagen”
t_3	skriv tecken till skrivarens dataregister, aktivera styrsignal ”data finns”	
t_4		tecken skrivs ut på papper
t_5	vänta tills tecken skrivits ut	
t_6		efter att ha skrivit ut tecken signalerar skrivaren nytt status
t_7	deaktivera skrivare om fler tecken, upprepa från t_1	

Lösningsförslag

Uppgift 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 */
}
```

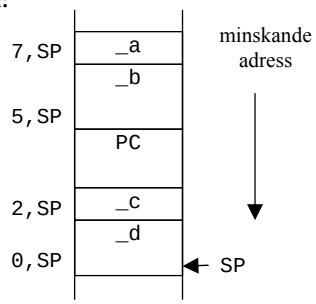
Uppgift 2a:

```
_a      RMB     2
_b      RMB     1
```

Uppgift 2b:

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

Uppgift 2c:



Uppgift 3:

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

void    ff1( void )
{
        unsigned char pattern, bitpos;
        while( 1 )
        {
            pattern = ML4IN;

            if( ! pattern )
                bitpos = 0;
            else{
                for( bitpos = 1; bitpos < 8; bitpos++ )
                {
                    if( pattern & 1 )
                        break;
                    pattern >>= 1;
                }
            }
            ML4OUT = bitpos;
        }
}
```

Uppgift 4:

```
int strspn(const char* cs, const char* ct)
{
          int n;
          const char* p;
          for(n=0; *cs; cs++, n++)
          {
                    for(p=ct; *p && *p != *cs; p++)
                              ;
                    if (!*p)
                              break;
          }
          return n;
}
```

Uppgift 5:

```
void    print( char *s )
PRINT:
; {
;       while( *s )
PRINT_1:
        TST     ,X
```

```
        BEQ     PRINT_2
;       {
;           while( (*( char *) 0x701) & 4 )
;           {}
PRINT_3:
        LDAB    $701
        ANDB    #4
        BNE     PRINT_3

;       DATA = *s;
;       s++;
        LDAB    1,X+
        STAB    $700
        BRA     PRINT_1
PRINT_2:
;       }
; }
    RTS
```

Uppgift 6:

a)

```
/* print_driver.h */
extern void init_printer(void);
extern void start_printer(void);

#define DATA_REG   (*(unsigned char *) 0x800 )
#define CTRL_REG   (*(unsigned char *) 0x801 )
#define STAT_REG   (*(unsigned char *) 0x802 )
```

b)

```
;   print_low.asm
_atIRQ:     CLR     0x801              ; DAV=0, IE=0, och kvittera avbrott
            JSR     start_printer
            RTI
```

c)

```
/* print_driver.c */

extern void atIRQ(void);   /* Avbrottsrutin i assembler */
static int init = 0;

typedef void (*vec) (void);

void init_printer( void )
{
    if (!init) {
      *( (vec *) 0x3FF2 )) = atIRQ;
      init = 1;
    }
}


void start_printer( void )
{
    unsigned char t;
    if( STAT_REG & 3 )
    {   /* Feltillstånd, kan inget göra */
        return;
    }
    if( ! (STAT_REG & 4 ) )
    {   /* Skrivaren upptagen med utskrift
        returnera till bufferthanterare eftersom
        avbrott kommer då skrivaren är klar med pågående tecken */
        return;
    }
    if( get (&t) )
    {   /* Om det fortfarande finns tecken i bufferten ... */
        DATA_REG = t;
        CTRL_REG = 3;  /* DAV=1 och IE=1 */
    }
}
```