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

Måndag 13 januari 2014, kl. 14.00 - 18.00

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 (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
$500	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		
$501	R	IF	IF	IF	IF	IF	IF	IF	IF	ICIE	Input Change Interrupt
	W	IEA	IEA	IEA	IEA	IEA	IEA	IEA	IEA		
$502	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_2-b_0 används som en 3-bitars inport och bitarna b_7-b_3 används som en 5-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_4-b_0, av `c`, till bitarna b_7-b_3 hos 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 2 (6p) *Assemblerprogrammering*

Följande funktion `bitMask` finns given som en specification i "C". Funktionen bestämmer en så kallad "bitmask" baserad på inparametrarnas värden.

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

   if( ( bit_number > 7 ) || ( bit_value > 1 ) )
      return 0;
   if( bit_value )
      return ( set[bit_number] );
   else
      return ( clear[bit_number] );
}
```

Skriv motsvarande funktion BITMASK i assemblerspråk för HC12. Observera att i denna uppgift ska du **inte** ta hänsyn till kompilatorkonventioner för XCC12. I stället skickas parametrarna enligt:

`bit_number` i register B
`bit_value` i register A
returvärde från funktionen i register B

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

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

Inledningen (parameterlistan, lokala variabler och några tilldelningar) för en funktion ser ut på följande sätt:

```
void  func( char *b, char a )
{
   char  c;
   char  *d;
   c = a;
   d = b;

   /* kod utelämnad */
}
```

a) Översätt *hela* funktionen `func`, som den är beskriven till HCS12 assemblerspråk.
Speciellt *ska* du börja med att beskriva *aktiveringsposten*, dvs. stackens utseende i funktionen.
Visa tydligt riktningen för *minskande adresser* hos aktiveringsposten. (6p)

b) Implementera en assembler subrutin som kan anropas från ett C-program.

```
unsigned char getCCR( void );
```

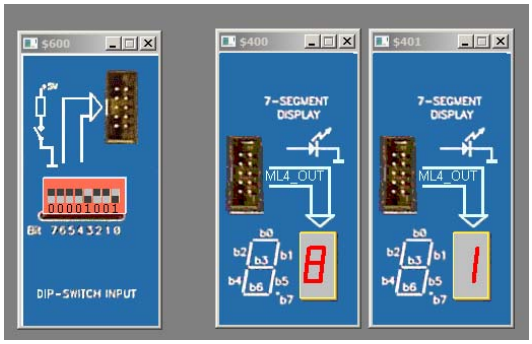
- returvärdet är innehållet i CCR (2p).

c) Implementera en assembler subrutin som kan anropas från ett C-program.

```
void setCCR( unsigned char value );
```

- parameter `value` anger nya värden för bitarna i CCR (2p).

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.

En strömbrytare är ansluten till address 0x600 och två stycken sju-sifferindikatorer är anslutna till adresser 0x400 och 0x401 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DisplaySquare( void )
```

som läser ett värde (0..9) från strömbrytarna, därefter beräknar kvadraten för detta värde och slutligen skriver resultatet på decimal form till sju-sifferindikatorerna. Se figuren som illustrerar inlästa värdet 9 med kvadraten 81.

Om det avlästa värdet är större än 9 kan dess kvadrat inte visas med två indikatorer och då ska i stället felkoden ’E’ (Error) visas på båda indikatorerna. Du har tillgång till en tabell i minnet med segmentkoder för de hexadecimala siffrorna [0..F] (mönster för sifferindikatorn) enligt

```
unsigned char SegCodes[]={ 0x77,0x22,0x5B,0x6B,0x2E,0x6D,0x7D,0x23,
                           0x7F,0x6F,0x3F,0x7C,0x55,0x7A,0x5D,0x18 };
```

Segmentkoden för bokstaven ’E’ ges av:

```
#define     ERROR_CODE     0x5D
```

Uppgift 5 (8p) *Programmering med pekare*

I denna uppgift får du *inte* använda dig av någon *standardfunktion* i C, utan du måste skriva allt själv.

I C finns en standardfunktion som heter `strtol` som används för att omvandla en text till ett numeriskt värde av typen **`long int`**.
Skriv en egen, något förenklad, version av funktionen `strtol`. Din funktion skall ha deklarationen:

```
long int my_strtol(const char *s, const char **endptr);
```

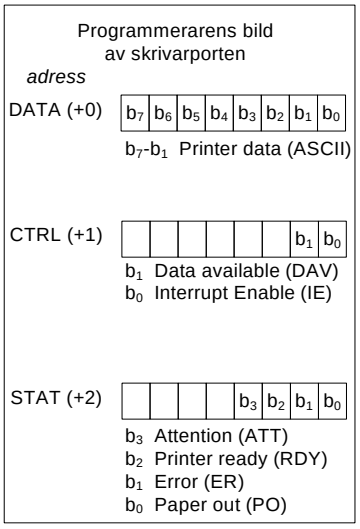
Funktionen `my_strtol` får en text som första parameter. Texten förväntas innehålla ett heltal representerat som en följd av enstaka tecken. Funktionens uppgift är att omvandla heltalet till ett värde av typen `long int` och returnera detta värde. Det får finnas ett godtyckligt antal blanka tecken (mellanslag) först i texten. Därefter får det finnas ett plustecken eller ett minustecken. Finns det inget plus- eller minustecken skall talet uppfattas som positivt. Själva talet skall bestå av ett godtyckligt antal tecken i intervallet ’0’ till ’9’. När funktionen stöter på ett tecken som *inte* kan ingå i ett heltal skall den avsluta avkodningen av talet och talet skall returneras. (Om inga siffror påträffades skall värdet 0 returneras.) Om den andra parametern till funktionen inte har värdet NULL skall den pekare den pekar på sättas att peka på det första tecknet som inte kan ingå i heltalet. Om den andra parametern har värdet NULL skall den inte ändras.

Ett par exempel:

- Om `s` pekar på texten ” 234ABC” skall funktionen returnera värdet 234 och den pekare den andra parametern pekar på skall sättas att peka på tecknet ’A’.
- Om `s` pekar på texten ” -960” skall funktionen returnera värdet -960 och den pekare den andra parametern pekar på skall sättas att peka på tecknet ’\0’.

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 1a:

```
typedef    struct sPortP{
       volatile unsigned char ddr;
       volatile unsigned char icie;
       volatile unsigned char data;
}PORTP;
#define      PORTP_BASE   0x500
#define      portP ((PORTP *)(PORTP_BASE))

void portPinit( void )
{
       portP->ddr = 0xF8;     /* b7-b3 utport, b2-b0 inport */
       portP->icie = 7;       /* b2-b0 inportar, avbrott aktiveras */
}
```

Uppgift 1b:

```
void outPortP( unsigned char c )
{
       portP->data = c << 3;
}
```

Uppgift 1c:

```
void   irqPortP( void )
{
    switch( portP->data & 7 )     /* bestäm avbrottskälla */
   {   /* kvittera rätt avbrott */
       case   4: portP-> icie = 4; break;
       case   2: portP-> icie = 2; break;
       case   1: portP-> icie = 1; break;
   }
}
```

Uppgift 1d:

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

       MOVW      #PortPirq,$FFF2
       CLI

; avbrottsrutin
PortPirq:
       JSR    _irqPortP
       RTI
```

Uppgift 2:

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

MASKBIT:
          CMPB   #7              ; if( bit_number > 7 )
          BHI    maskbit_exit_0
          CMPA   #1              ; if( bit_value > 1 )
          BHI    maskbit_exit_0
          BNE    maskbit_1       ; if( bit_value)
          LDX    #set            ; return ( set[bit_number] );
          BRA    maskbit_2
maskbit_1:
          LDX    #clear          ; return ( clear[bit_number] );
maskbit_2:
          LDAB   B,X
          RTS
maskbit_exit_0:
          CLRB                   ; return 0
maskbit_exit:
          RTS
```

Uppgift 3a:

a) Beskrivning av aktiveringspost

minnesanvändning	*stackoffset*	*minskande adress*
_a	7,SP	
_b	5,SP	
PC (vid JSR)		
_c	2,SP	
_d	0,SP	

Kodning av funktionen:

```
func:
       LEAS   -3,SP
;  c = a;
       LDAB   7,SP
       STAB   2,SP
;  d = b;
       LDD    5,SP
       STD    0,SP
       LEAS   3,SP
       RTS
```

Uppgift 3b:

```
_getCCR:
          TFR    CCR,B
          RTS
```

Uppgift 3c:

```
_setCCR:
          LDAB   2,SP
          TFR    B,CCR
          RTS
```

Uppgift 4:

```
typedef  unsigned char *port8ptr;
#define ML4OUTH     *(( port8ptr ) 0x400)
#define ML4OUTL     *(( port8ptr ) 0x401)
#define ML4IN       *(( port8ptr ) 0x600)
void   DisplaySquare( void ){
   char c;
   c = ML4IN;
   if( c < 10 ){
      c = c*c;  /* kvadrera */
      ML4OUTH = SegCodes[c / 10 ];  /* mest signifikant */
      ML4OUTL = SegCodes[c % 10 ];  /* minst signifikant */
   }else{
      ML4OUTH = ERROR_CODE;  /* felkod till båda indikatorer */
      ML4OUTL = ERROR_CODE;
   }
}
```

Uppgift 5:

```
long int my_strtol(const char *s, const char **endptr){
   long int i = 0;
   char c;
   while ( *s == ' ')
      s++;
   c = *s;
   if (c == '+' || c == '-')
      s++;
   while (*s >= '0' && *s <= '9') {
      i = i * 10 + *s - '0';
      s++;
   }
   if (endptr != NULL)
      *endptr = s;
   if (c == '-')
      i = -i;
   return i;
}
```

Uppgift 6:

```
/* filen asm.h, rutiner är implementerade i ’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
```