Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D
EDA486 Programmering av inbyggda system Z
DIT152 Programmering av inbyggda system GU

Fredag 31 Maj 2013, 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) *Assemblerprogrammering*

En C-funktion definieras enligt följande:

```
int bitcount(unsigned char value)
{
   unsigned char count;
   for ( count=0; value != 0; value >>= 1);
   {
      if (value & 1)
         count++;
   }
   return count;
}
```

Skriv, i assemblerspråk för HCS12, en subrutin `_bitcount` som *motsvarar* C-funktionen. Du *behöver* alltså inte göra någon regelrätt översättning av C-funktionen utan bara betrakta den som en specifikation av vad assemblerrutinen ska utföra. För denna uppgift gäller alltså *inga* C-anropskonventioner utan följande specifikation ska gälla för din subrutin:

```
;  subrutin bitcount
;  bestämmer antalet ettor i 'value' med storleken 'byte'.
;  Indata:
;           register B innehåller 'value'
;  Returvärde:
;           register B innehåller antalet ettor i 'value'
```

Uppgift 2 (6p) *Användning av sammansatta datatyper*

Parallellporten Port P, i ett HCS12-system kan programmeras så att varje bit kan utgöra antingen en insignal, eller en utsignal. Porten har två olika register, som specificeras enligt följande:

Parallel port P (PORTP)											
Address		7	6	5	4	3	2	1	0	Mnemonic	Namn
$700	R	0	0	0	0	0	0	0	0	DDR	Data Direction Register
	W										
$701	R	0	0	0	0	0	0	0	0	DATA	Data Register
	W	1	1	1	1	1	1	1	1		

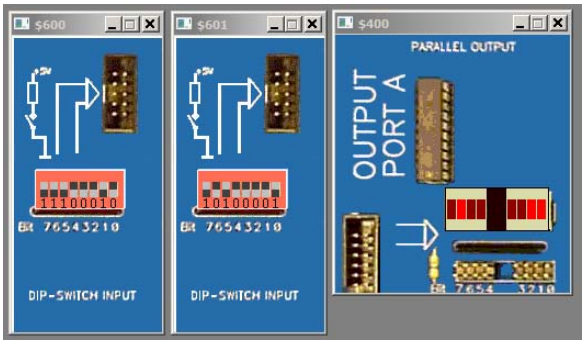
- DDR: 0 anger att positionen är en utsignal, 1 anger att positionen är en insignal. Bitarna kan programmeras oberoende av varandra, dvs. godtycklig kombination av insignaler och utsignaler kan åstadkommas. Registret är både skrivbart och läsbart i sin helhet.
- DATA: Består i själva verket av två olika 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_5 används som en 3-bitars inport och bitarna b_4-b_0 används som en 5-bitars utport.

b) Visa en funktion, **`void`** `outPortP(`**`unsigned char`** `c )` som matar ut bitarna b_4-b_0, av `c`, till port P.

c) Visa en funktion, **`unsigned char`** `inPortP(` **`void`** `)` som returnerar bitarna b_7-b_5 hos port P som en **`unsigned char`**, dvs. värden i intervallet 0 t.o.m. 7.

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 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 4 (8p) *Programmering med pekare*

Skriv en C-funktion med följande deklaration:

```
char *xcpy(const char *q1, char *q2);
```

Funktionen skall kopiera den text `q1` pekar på till det utrymme som pekas ut av `q2`, men vid kopieringen skall alla *inledande och avslutande* s.k. *vita tecken* utelämnas. Med vita tecken menas mellanslag, tabulator och nyradstecken. Tänk på att det kan finnas vita tecken inne i den text som skall kopieras. Dessa tecken skall förstås tas med i kopieringen.

Funktionen `xcpy` skall som resultat ge en pekare till kopian av texten. Du får inte använda dig av någon av C:s standardfunktioner, utan du måste skriva all kod själv.

Tips. Leta först framifrån efter första icke vita tecken. Sök sedan upp slutet på texten och leta därifrån bakåt efter sista icke vita tecken. Kopiera sedan den mellanliggande texten till det utrymme som pekas ut av `q2`.

Uppgift 5 (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:

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

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

```
struct namn *  cilla;
int            bertina;
char           adamo;
```

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

```
func( adamo , bertina , cilla );
```

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

Uppgiften är att konstruera en modul som gör det möjligt att schemalägga anrop av C-funktioner i ett mycket enkelt realtidsprogram. Modulens include-fil heter `scheduler.h` och har följande utseende:

```
typedef void (*function)(void);
extern void init_scheduler(void);
extern _Bool schedule(function f, unsigned long int interval);
```

Meningen är att man, efter att ha initierat modulen genom att anropa funktionen `init_scheduler`, skall anropa funktionen `schedule` för var och en av de funktioner man vill skall exekveras. I varje anrop ger man en pekare till funktionen och anger hur ofta man vill att funktionen skall anropas. Intervallet mellan anropen ges i enheten millisekunder. När detta skett skall modulen se till att funktionerna automatiskt anropas. Funtionen `schedule` returnerar ett värde som anger om inläggningen gick bra eller inte.

Din uppgift är att implementera filen `scheduler.c`. Eftersom man i denna fil skall ta hand om avbrott från realtidsklockan innebär din uppgift att du *också skall skriva den assemblerrutin* som behövs för att hantera avbrott och att du skall initiera avbrottsvektorn på lämpligt sätt.

Här kommer först lite tekniska data om realtidsklockan. (Alla adresser ges i hexadecimalt format.) Avbrottsvektorn har en adressen FFF0. Man behöver känna till tre 8-bitars register: CRGFLG på adressen 37, CRGINT på adressen 38 och RTICTL på adressen 3B. För att aktivera avbrott från realtidsklockan skall man skriva en etta i bit nummer 7 i CRGINT. I registret RTICTL skall man lägga in en kod som bestämmer med vilken frekvens realtidsklockan genererar avbrott. Koden 49 (hexadecimalt) ger avbrott ungefär var 10:e millisekund. Varje gång ett avbrott har skett skall man begära ett nytt genom att skriva en etta i bit nummer 7 i registret CRGFLG.

Filen `scheduler.c` skall förstås innehålla definitioner av de funktioner som deklareras i filen `scheduler.h`.

Initiering av realtidsklockan och avbrottsvektorn skall göras i funktionen `init_scheduler`.

Funktionen `schedule` skall spara informationen om funktioner och tidsintervall. För att åstadkomma detta kan man internt i modulen använda en tabell med information om funktionerna. Denna tabell kan vara en array (ett fält) där varje element är en struct som innehåller tre komponenter: funktionspekaren, tidsintervallet och nästa tidpunkt när funktionen skall anropas. Denna tidpunkt kan sättas till aktuell tidpunkt plus tidsintervallet när man lägger in en ny funktion i tabellen. Det är lämpligt att ange tidpunkten som antal avbrott från realtidsklockan (antalet tick) istället för millisekunder.

Filen `scheduler.c` skall, förutom definitioner av de funktioner som deklareras i filen `scheduler.h`, även innehålla definitionen av en funktion `clock_inter` som skall anropas varje gång ett avbrott har skett. (Det är din uppgift att se till att denna funktion anropas.) Funktion `clock_inter` skall varje gång den anropas räkna upp en variabel som håller reda på hur många avbrott som kommit (antalet tick). Sedan skall `clock_inter` undersöka om det är dags att anropa någon av de funktioner som finns i tabellen. Fast detta skall bara göras om inte någon av dessa redan är anropad (och alltså avbrottet kom när denna exekverades). För att se om någon funktion skall anropas väljs i tabellen den funktion `f` ut som har lägst värde för nästa tidpunkt. Om denna tidpunkt är mindre än eller lika med aktuell tidpunkt anropas `f`. När man kommer tillbaka från `f` ökas i tabellen nästa tidpunkt för `f` med tidsintervallet.

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:

```
_bitcount:
;     4 | {
;  Registerallokering:
;      reg B: value
;      reg A: count
;     5 |    unsigned char count;
;     6 |
;     7 |    for ( count=0; value != 0; value >>= 1)
   CLRA          ; count=0
bitcount_2:
   TSTB          ; value != 0
   BNE    bitcount_4
   BRA    bitcount_5

bitcount_3:
   LSRB          ; value >>= 1
   BRA    bitcount_2

bitcount_4:
;     8 |    {
;     9 |       if (value & 01)
   BITB   #1
   BEQ    bitcount_3
;    10 |          count++;
   INCA
;    11 |    }
   BRA    bitcount_3

bitcount_5:
;    12 |
;    13 |    return count;
;    14 | }
   TFR    A,B
   RTS
```

Uppgift 2a (2p):

```
typedef  struct sPortP{
   volatile unsigned char ddr;
   volatile unsigned char data;
}PORTP, *PPORTP;

#define  PORTP_BASE  0x700


#define  portP ((PORTP *)(PORTP_BASE))

void portPinit( void )
{
   portP->ddr = 0xE0;
}
```

Uppgift 2b (2p):

```
unsigned char inPortP( void )
{
   return (( portP->data & 0xE0 )>> 5)  ;
}
```

Uppgift 2c (2p):

```
void outPortP( unsigned char c )
{
   portP->data = c & 0x1F ;
}
```

Uppgift 3:

```
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 4:

```
char *xcpy(const char *s1, char *s2) {
      const char *p1;
      char *p2;
      /* Hoppa över inledande vita tecken */
      while (*s1 && *s1 == ' ' || *s1 == '\t' || *s1 == '\n')
        s1++;
      /* Leta reda slutet av texten */
      for (p1=s1; *p1; p1++)
        ;
      /* Sök sista icke-vita tecken */
      for (p1--; p1>s1 && *p1 == ' ' || *p1 == '\t' || *p1 == '\n'; p1--)
        ;
      /* Kopiera till s2 */
      for (p2=s2; s1<=p1;)
        *p2++ = *s1++;
      *p2 = '\0';
      return s2;
    }
```

Uppgift 5a:

Beskrivning av aktiveringspost

minnesanvändning	*stackoffset*
ceasar	10,SP
bertil	8,SP
adam	7,SP
PC (vid JSR)	
a	4,SP
b	2,SP
c	0,SP

```
_func:
   LEAS     -5,SP
;  char  a = adam;
   LDAB     7,SP
   STAB     4,SP
;  int   b = bertil;
   LDD      8,SP
   STD      2,SP
;  long     *c = ceasar;
   LDD      10,SP
   STD      0,SP
```

```
   LEAS     5,SP
   RTS
```

Uppgift 5b:

```
_cilla:     RMB   2
_bertina:   RMB   2
_adamo:     RMB   1

   LDD   _cilla
   PSHD
   LDD   _bertina
   PSHD
   LDAB  _adamo
   PSHB
   JSR   _func
   LEAS  5,SP
```

Uppgift 6:

Filen med avbrottsrutinen (assembler):

```
      segment  text
      export   _clocktrap
      import   _clock_inter

_clocktrap:
      JSR    _clock_inter
      RTI
```

Filen `clockports.h`:

```
typedef unsigned char port;        // port
typedef port *portptr;             // pekare till en port
typedef void (*vec) (void);        // avbrottsvektor
typedef vec *vecptr;               // pekare till en avbrottsvektor
#define setbits(r, mask)    (r) = (r) | (mask)
#define CRG_VEC_ADR      0xFFF0    // Adress till avbrottsvektor
#define CRG_VEC          *((vecptr) CRG_VEC_ADR)
#define CRGFLG_ADR      0x37       // För att kvittera avbrott
#define CRGFLG          *((portptr) CRGFLG_ADR)
#define rtif_bit        0x80
#define CRGINT_ADR      0x38       // För att aktivera avbrott
#define CRGINT          *((portptr) CRGINT_ADR)
#define rtie_bit        0x80
#define RTICTL_ADR      0x3B       // För att ange avbrottsintervall
#define RTICTL          *((portptr) RTICTL_ADR)
#define TIME_INTERVAL   10         // Antal ms mellan avbrott
#define TIME_CODE       0x49       // Ger ungefär ovanstående
```

Filen `scheduler.c`:

```
#include "scheduler.h"
#include "clockports.h"
#define N 100

struct tab_struct {
  function f;
  unsigned long int interval;
  unsigned long int next_time;
} tab[N];

static int num = 0;                          // antalet registrerade funktioner
static _Bool busy = 0;                       // anger om någon funktion exekveras
static unsigned long int current_time = 0;  // tick, räknas upp vid varje avbrott

void init_scheduler(void) {
  current_time = 0;            // Initiera räknaren
  extern void clocktrap();     // Avbrottsrutinen (assembler)
  CRG_VEC = clocktrap;         // Initiera avbrottsvektor
  setbits(CRGINT, rtie_bit);   // Aktivera timer-avbrott
  RTICTL = TIME_CODE;          // Sätt verklig avbrottsfrekvens
}

void clock_inter(void) {       // anropas vid avbrott
  current_time++;
  setbits(CRGFLG, rtif_bit);
  if (!busy && num > 0) {
    busy = 1;
    // sök den funktion som ska anropas först
    int first = 0;
    for (int i=1; i<num; i++)
      if (tab[i].next_time < tab[first].next_time)
        first = i;
    if (tab[first].next_time <= current_time) {  // är det dags?
      tab[first].f();
      tab[first].next_time += tab[first].interval;
    }
    busy = 0;
  }
}

_Bool schedule(function f, unsigned long int interval) {
  if (num < N) {
    tab[num].f = f;
    tab[num].interval = interval / TIME_INTERVAL;  // skala om från ms -> tick
    tab[num].next_time = current_time + tab[num].interval;
    num++;
    return 1;
  }
  else return 0;
}
```