Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

Programmering av inbyggda system

Exempel 1

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 (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.

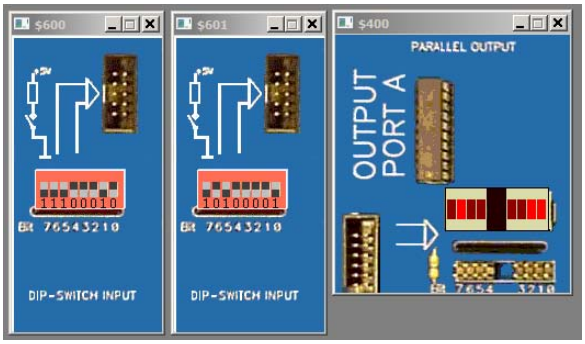
a) Visa en lämplig deklaration av porten med användning av en **`struct`**. Visa också en funktion, **`void`** `portPinit(`**`void`**`)` som initierar port P så att bitarna b_7-b_5 används som en 3-bitars inport och bitarna b_4-b_0 används som en 5-bitars utport.

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*

En utrustning för automatisk övervakning av inbrottslarm har åtta ingångar. Till sju av dessa kopplar man sensorer. Till den åttonde ingången kopplas en knapp. Utrustningen har också en utgång till vilken en högtalare (siren) har kopplats. Utrustningen styrs av en dator via två åttabitars register: ett styrregister och ett dataregister på de hexadecimala adresserna 500 resp. 501. Styrregistret är endast skrivbart. Följande bitar används:

Bit 0: (ONOFF), huvudströmbrytare för utrustningen (1 == ON, 0 == OFF)
Bit 1: (Sound_ONOFF), anger om utgången till högtalaren ska vara aktiv (1) eller passiv (0)
Bit 2. (IE) anger om utrustningen ska generera avbrott när någon av insignalerna blir hög (1)

Dataregistret är både läs- och skrivbart. När en insignal till utrustningen blir hög sätts automatiskt motsvarande bit i dataregistret till 1. Bitarna 0-6 används för sensorerna och bit nr 7 till knappen. Om biten IE är påslagen genereras ett avbrott när någon av insignalerna blir hög. Avbrottsvektorn har den hexadecimala adressen FFE0. Ett avbrott kvitteras genom att man nollställer dataregistret.

Förutom övervakningsutrustningen finns också en display med lampor. Displayen används för att indikera vilka sensorer som givit utslag och styrs via ett skrivbart åttabitars register på den hexadecimala adressen 600. Genom att skriva en etta eller nolla i en bit i detta register tänds resp. släcks motsvarande lampa på displayen.

Din uppgift är att skriva en modul som styr övervakningsutrustningen. För att förenkla det hela något är följande assemblerfil given:

```
                segment text
                export   _sensortrap
                export   _cli
                import   _sensorinter
_sensortrap:    JSR      _sensorinter
                RTI
_cli:           CLI
                RTS
```

Din modul ska skrivas helt i C. Den ska innehålla de två funktionerna `alarm_on` och `alarm_off` vilka ska kunna anropas från en annan del av programmet. Dessutom ska det finnas kod som tar hand om avbrott från övervakningsutrustningen. Funktionen `alarm_on` ska göra alla initieringar som behövs för att aktivera övervakningsutrustningen. I detta ingår att se till att avbrott aktiveras och styrs till lämplig avbrottsrutin. Funktionen `alarm_off` ska helt enkelt stänga av utrustningen.

När utrustningen ger avbrott ska det fungera på följande sätt: Om avbrottet kommer från någon av de sju ingångar som är kopplade till sensorer ska motsvarande lampa på displayen tändas. De eventuella lampor som tidigare tänts ska förbli tända. Om användaren inte tryckt på knappen ska högtalaren också kopplas på. När användaren trycker på knappen första gången ska högtalaren stängas av, men de lampor som är ända på displayen ska förbli tända. Om användaren trycker på knappen en andra gång ska alla lampor på displayen släckas. Man ska då återkomma till utgångsläget. Tanken är alltså att när ett larm inträffas så uppmärsammas användaren på detta genom att högalaren tjuter. Han trycker då en gång på knappen för att stänga av ljudet och kan sedan i lugn och ro på displayen se vilken eller vilka sensorer som givit utslag. När orsaken till larmet klargjorts kan han trycka en gång till på knappen för att återställa allt.

Observera att du måste skriva all C-kod, inklusive de deklarationer och definitioner som behövs. Ange också i vilka filer koden lämpligen 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)

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 alarm.h */
extern void alarm_on(void);
extern void alarm_off(void);
extern void sensorinter(void);

/* Filen definitioner.h */
typedef unsigned char port;
typedef port *portptr;
typedef void (*vec) (void); // avbrottsvektor
typedef vec *vecptr; // pekare till // avbrottsvektor

#define SENSOR_ADR 0x500
#define SENSOR_CTRL (*((portptr) SENSOR_ADR))
#define SENSOR_ON  0x1
#define SOUND_ON   0x2
#define SENSOR_IE  0x4
#define SENSOR_DATA (*((portptr) (SENSOR_ADR+1)))
#define BUTTON_PUSHED 0x80
#define SENSOR_VEC_ADR 0xFFE0
#define SENSOR_VEC (*((vecptr) SENSOR_VEC_ADR))
#define DISPLAY_ADR 0x600
#define DISPLAY (*((portptr) DISPLAY_ADR))

/* Filen assembler.h */
extern void sensortrap(void);
extern void cli(void);

/* Filen alarm.c */
#include "alarm.h"
#include "definitioner.h"
#include "assembler.h"

void alarm_on(void) {
  SENSOR_VEC = sensortrap;
  SENSOR_CTRL = SENSOR_ON | SENSOR_IE;
  cli();
}

extern void alarm_off() {
  SENSOR_CTRL = 0;
}

static port data  = 0;    // skuggregister
static int pushed = 0;    // har knappen tryckts in en gång tidigare?

void sensorinter(void) {
  if (!(SENSOR_DATA & BUTTON_PUSHED)) {     // avbrott från en sensor
      data = data | SENSOR_DATA;
    if (!pushed)
      SENSOR_CTRL = SENSOR_CTRL | SOUND_ON;  // slå på ljudet
  }
  else {                                     // avbrott från knappen
    if (!pushed)       // första tryckning
      SENSOR_CTRL = SENSOR_CTRL & ~SOUND_ON; // stäng av ljudet
    else             // andra tryckning
      data = 0;      // återställ
```

```
    pushed = (pushed+1) % 2;   // ändra tillstånd
  }
  SENSOR_DATA = 0;  // kvittera avbrottet
  DISPLAY = data;   // visa indikatorer
}
```