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

Tisdag 3 juni 2014, kl. 14.00 - 18.00, Maskinsalar

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

Uppgift 1 (8p) *Användning av sammansatta datatyper*

En parallellport, `PortP`, 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
$A00	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		
$A01	R									DATA	Data Register
	W										

- DDR:
  - 1 anger att bitpositionen är en utsignal,
  - 0 anger att bitpositionen 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, *i annat fall är portens beteende odefinierat*.
  - 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 `PortP` med användning av en `struct`. Visa också en funktion, `void portPinit(void)` som initierar `PortP` så att bitarna b_7-b_4 används som en 4-bitars inport och bitarna b_3-b_0 används som en 4-bitars utport.

b) Visa en funktion, `void outPortP(unsigned char c )` som matar ut bitarna b_3-b_0, av `c`, till `PortP`.

c) Visa en funktion, `signed char inPortP( void )` som returnerar bitarna b_7-b_4 hos `PortP` som en (teckenutvidgad) `signed char`, dvs. tolkningen av dessa fyra bitar är tal med tecken och talintervallet är därför -8 .. 7.

Uppgift 2 (8p) *Assemblerprogrammering*

Följande funktion `modyBit` finns given i "C". (Funktionen kombinerar subrutinerna `OUTONE` respektive `OUTZERO` från laborationskursen).

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

   if( bit_number > 7 )    return;
   if( bit_value > 1 )     return;
   if( bit_value == 1 )
      shadow = shadow | set[bit_number];
   else
      shadow = shadow & clear[bit_number];
   *((unsigned char *) 0x400) = shadow;
}
```

Skriv motsvarande funktion `MODYBIT` i assemblerspråk för HC12. I denna uppgift ska du INTE ta hänsyn till kompilatorkonventioner utan i stället skickas parametrarna till `OUTBIT` enligt:

`bit_number` i register B

`bit_value` i register A

Uppgift 3 (12p) *Kodningskonventioner (C/assemblerspråk)*

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

Följande C-deklaration har gjorts på ”toppnivå” (global synlighet):

```
unsigned char     *pvec[4];
```

a) Visa hur deklarationen översätts till assemblerdirektiv.

b) Implementera en subrutin (assemblerspråk), som kan anropas från ett C-program, med prototypen:

```
unsigned char * getSP( void );
```

Funktionen ska returnera det värde som stackpekaren SP innehåller vid den punkt i programflödet där anropet utförs.

c) Med deklarationer enligt a) och b), visa hur följande tilldelningssats kan kodas i HCS12 assemblerspråk.

```
pvec[3] = getSP();
```

d) Följande C-funktion är given:

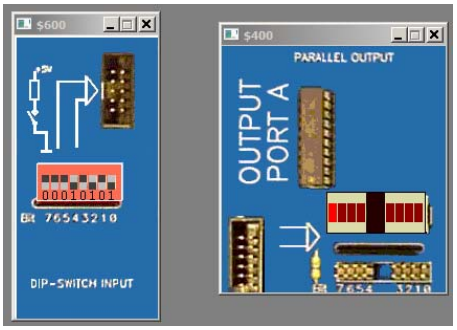
```
unsigned char ismin( unsigned char a, unsigned char b)
{
   if( a < b ) return 1;
   return 0;
}
```

Visa motsvarande funktion i assemblerspråk, speciellt ska du beskriva aktiveringsposten i funktionen.

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 ev. typdeklarationer används för att skapa begriplig programkod.

En 8-bitars strömbrytare är ansluten till adress 0x600 och en 8-bitars diodramp är ansluten till adress 0x400 i ett MC12 mikrodatorsystem.



Skriv en funktion `void OddIndicator(void)` som

- läser de 8 bitarna från från strömbrytaren
- om antalet ett-ställda bitar är udda ska nu b_7 hos ljusdiodrampen tändas, annars ska b_0 hos ljusdiodrampen tändas.

Uppgift 5 (6p) *Programmering med pekare*

Standardfunktionen `strcmp` kan beskrivas på följande sätt:

```
int strcmp (const char *String1 , const char *String2 );
```

The **strcmp** subroutine compares strings in memory.
The *String1* and *String2* parameters point to strings. A string is an array of characters terminated by a null character.
The **strcmp** subroutine performs a case-sensitive comparison of the string pointed to by the *String1* parameter and the string pointed to by the *String2* parameter, and analyzes the extended ASCII character set values of the characters in each string. The **strcmp** subroutine compares **unsigned char** data types. The **strcmp** subroutine then returns a value that is:

- Less than 0 if the value of string *String1* is lexicographically less than string *String2*.
- Equal to 0 if the value of string *String1* is lexicographically equal to string *String2*.
- Greater than 0 if the value of string *String1* is lexicographically greater than string *String2*.

Din uppgift är att, i C, skriva en egen definition av funktionen `strcmp`. 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 6 (10p) *Maskinnära programmering i C*

Man har längs vissa vägsträckor utrustat hastighetsskyltarna med radiosändare som sänder ut information om den högsta tillåtna hastigheten. De fordon som passerar kan vara utrustade med radiomottagare som tar emot denna information. Den mottagna informationen lagras i ett register i mottagarna. Detta register kopplas till fordonets dator så att det kan avläsas från denna. Registret innehåller 16 bitar och har kopplats till den hexadecimala adressen 900 i datorn. Om en radiomottagare inte mottagit någon information den senaste minuten nollställs automatiskt detta register.

I ett fordon finns också en "intelligent" digital hastighetsmätare som har en display som på vanligt sätt visar den aktuella hastigheten. Till hastighetsmätaren hör två register vilka kan anslutas till datorn, ett styrregister på den hexadecimala adressen B04 och ett dataregister på den hexadecimala adressen B08. Båda dessa register består av 16 bitar. För att slå på ”intelligensen” i hastighetsmätaren skall man sätta bit nr 0 i styrregistret till 1 och för att slå på avbrottsmekanismen skall bit nr 6 sättas till 1. Bitarna 12-15 i styrregistret innehåller felinformation (felkoder). Om inget fel inträffat är dessa bitar alla 0. I hastighetsmätarens dataregister kan man lägga in den högsta tillåtna hastigheten. Om fordonets aktuella hastighet överstiger hastigheten i dataregistret genererar hastighetsmätaren automatiskt en avbrottssignal till datorn. En förutsättning för detta är dock att ”intelligensen” och avbrottsmekanismen slagits på. Avbrottsvektorn ligger på adressen 3EF4 hexadecimal. Avbrottet upprepas var tredje sekund tills hastigheten inte längre överstiger hastigheten i dataregistret. Hastighetsmätaren kan även generera avbrott om något fel uppstått.

Till datorn finns slutligen kopplat en krets som genererar en ljudsignal. Till denna krets hör ett 16-bitars styrregister på den hexadecimala adressen B60. Om man lägger en etta i bit nr 0 i detta register kommer en signal att ljuda under ett kort ögonblick (kortare än tre sekunder). Därefter tystnar signalen och bit nummer 0 återställs automatiskt till 0

Uppgiften är att i C skriva det program som behövs i fordonets dator för att samordna de kretsar som beskrivits ovan. Programmet skall var 10:e sekund avläsa den högsta tillåtna hastigheten från radiomottagaren. Om radiomottagaren mottagit information skall den högsta tillåtna hastigheten läggas i hastighetsmätarens dataregister och hastighetsmätaren skall initieras så att den kan generera avbrott. Om ingen information mottagits skall hastighetsmätaren inte kunna generera avbrott.

Om ett avbrott genererats av hastighetsmätaren och det inte är något fel skall ljudsignalen slås på. Om något fel uppstått man kan ignorera detta och helt enkelt nollställa de bitar som innehåller felkoden.

Du måste också initiera avbrottsvektorn och skriva den assemblerrutin man kommer till då avbrott inträffar.

I denna uppgift får du förutsätta att funktionen `hold` som du konstruerade på laborationerna är färdigskriven och kan användas. Den ger som du minns en fördröjning med så många millisekunder som dess parameter anger: `void hold(time_type msDelay);` Du får också förutsätta att typen `short int` implementeras med 16 bitar.

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 data;
}PORTP, *PPORTP;
#define      PORTP_BASE   0xA00
#define      portP ((PORTP *)(PORTP_BASE))

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

Uppgift 1b:

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

Uppgift 1c:

```
signed char inPortP( void )
{
   signed char c;
   c = (portP->data & 0xF0)>> 4;
   if( c & 8 )
      c = c | 0xF0;    /* teckenutvidga */
   return c  ;
}
```

Uppgift 2:

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

MODYBIT:
   CMPB   #7        ; if( bit_number > 7 )
   BHI    outbit_exit
   CMPA   #1        ; if( bit_value > 1 )
   BHI    outbit_exit
   BNE    outbit_1  ; if( bit_value == 0 )
;
   LDX    #set      ; if( bit_value == 1 ) shadow = shadow | set[bit_number];
   LDAB   B,X
   ORAB   shadow
   BRA    outbit_2
outbit_1:
   LDX    #clear    ; else shadow = shadow & clear[bit_number];
   LDAB   B,X
   ANDB   shadow
outbit_2:
   STAB   shadow
   STAB   $400      ; *((unsigned char *) 0x400) = shadow;
outbit_exit:
   RTS
```

Uppgift 3a:

```
_pvec:    RMB    8
```

Uppgift 3b:

```
getSP:
          TFR    SP,D
          ADDD   #2
          RTS
```

Uppgift 3c:

```
          JSR    _getSP
          STD    _pvec+6
```

Uppgift 3d:

```
;  unsigned char ismin( unsigned char a, unsigned char b)
_ismin:
;  {
;      if( a < b ) return 1;
       LDAB   2,SP
       CMPB   3,SP
       BCC    _ismin_2
       LDAB   #1
       BRA    _ismin_3
_ismin_2:
;          return 0;
       CLRB
;      }
_ismin_3:
       RTS
```

Uppgift 4:

```
typedef   unsigned char * port8ptr;
#define OUT *((port8ptr) 0x400)
#define IN  *((port8ptr) 0x600)

void   OddIndicator( void )
{
   unsigned char count, portvalue;

   portvalue = IN;
   count = 0;
   while( portvalue )
   {
      if( portvalue & 1 )
         count++;
      portvalue = portvalue >> 1;
   }
   if( count & 1 )
      OUT = 0x80;
   else
      OUT = 1;
}
```

Uppgift 5:

```
int strcmp(const char *s1, const char *s2)
{
   while (1) {
      if (*s1 != *s2)
         return (int)(*s1 - *s2);
      if (*s1 == 0)
         return(0);
      s1++;
      s2++;
   }
}
```

Uppgift 6:

```
// Filen hastighet.h
   // grundläggande definitioner
   typedef short int port;
   typedef port *portptr;
   typedef void (*vec) (void);
   typedef vec *vecptr;

   // hastighetsmätaren
   #define HASTIGHET_REG_ADR 0xB04
   #define HASTIGHET_DAT_ADR 0xB08
   #define HASTIGHET_VEC_ADR 0x3EF4
   #define HASTIGHET_REG   *((portptr) HASTIGHET_REG_ADR)
   #define HASTIGHET_DAT   *((portptr) HASTIGHET_DAT_ADR)
   #define HASTIGHET_VEC   *((vecptr)  HASTIGHET_VEC_ADR)
   #define ENABLE_BIT 0x1
   #define INTERRUPT_BIT 0x40
   #define ERROR_BITS 0xF000

   // mottagaren
   #define MOTTAGARE_DAT_ADR 0x900
   #define MOTTAGARE_DAT   *((portptr) MOTTAGARE_DAT_ADR)

   // alarm
   #define ALARM_DAT_ADR 0xB60
   #define ALARM_DAT   *((portptr) ALARM_DAT_ADR)
   #define ALARM_ON_BIT 0x1

   // Filen time.h
   void hold(long int);

   // Filen inter.h
   void hast_trap();

   // Filen inter.s12
      segment text
      define _hast_trap
      extern _hast_inter
   _hast_trap:  jsr _hast_inter
         rti

   // Filen main.c
   #include "hastighet.h"
   #include "time.h"
   #include "inter.h"

   main() {
    short int avlast;
    HASTIGHET_VEC = hast_trap;
    while(1) {
      if ((avlast=MOTTAGARE_DAT) > 0) {
        HASTIGHET_DAT = avlast;
        HASTIGHET_REG = ENABLE_BIT | INTERRUPT_BIT;
      }
      else
        HASTIGHET_REG = 0;
      hold(10000);
    }
    }

    void hast_inter() {
      if (HASTIGHET_REG & ERROR_BITS)
        HASTIGHET_REG = HASTIGHET_REG & ~ERROR_BITS;
      else
        ALARM_DAT = ALARM_ON_BIT;
    }
```