Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

LEU500 Maskinorienterad programmering, DAI,EI,MEI

Måndag 14 mars 2016, kl. 14.00 - 18.00

Examinator
Lars Bengtsson tel. 772 8441

Kontaktperson under tentamen:
Roger Johansson, tel. 772 57 29

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

Uppgift 1 (10p) *Kodningskonventioner (C/assemblerspråk)*

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

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     *a,b;
```

a) Visa hur variabeldeklarationerna på toppnivå översätts till assemblerdirektiv för HCS12. (2p)

b) Med variabeldeklarationerna i a), visa hur följande funktionsanrop översätts till assemblerkod för HCS12: (2p)

```
func( a , b );
```

c) Visa hur utrymme för lokala variabler skapas och aktiveringspostens utseende (dvs. stacken) i funktionen '`func`'. (2p)

d) Visa hur följande tilldelningar kan utföras i '`func`' (6p)

```
c = a;
d = b;
c = *b;
d = &a;
```

Uppgift 2 (8p) *Avbrott*

Man vill skapa en generell funktion som installerar en avbrottshanterare på någon avbrottsvektor och samtidigt förbereder processorn genom att nollställa I-flaggan i CCR. Följande exempel visar hur en avbrottsrutin `myC_irq_handler` installeras på avbrottsvektor 0x3FF2.
Returvärdet från funktionen sparas i `oldvec`, så att vi senare kan återinstallera den befintliga avbrottsvektorn om så skulle behövas:

```
typedef void (*IrqHandler) (void);
extern void myC_irq_handler(void);
....
IrqHandler * oldvec;
....
oldvec = install_irq_handler( myC_irq_handler, (IrqHandler **) 0x3FF2 );
```

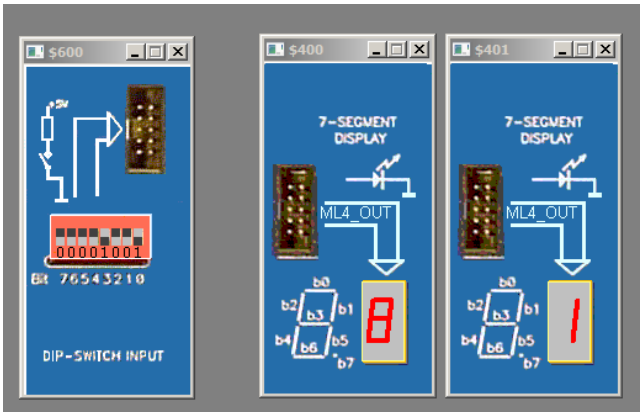
Funktionen implementeras först i ’C’, enligt följande.

```
IrqHandler * install_irq_handler( IrqHandler irq, IrqHandler ** vector)
 {
   IrqHandler * p;

   p = *vector;
   *( (IrqHandler *) vector) = irq;

   /* Assemblerinstruktion "CLI" */
   return p;
 }
```

Eftersom funktionen inte uppfyller specifikationen fullständigt (I-flaggan nollställs inte) bestämmer man sig nu för att koda om *hela* funktionen i assemblerspråk.
Skriv subrutinen `_install_irq_handler` i HCS12 assembler. Subrutinen ska kunna anropas från ett C-program enligt exemplet ovan och du måste därför tillämpa XCC12’s konventioner.

Uppgift 3 (10p) *In och utmatning beskriven i C*

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



Konstruera en funktion

```
void DisplayNBCD( 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 };
```

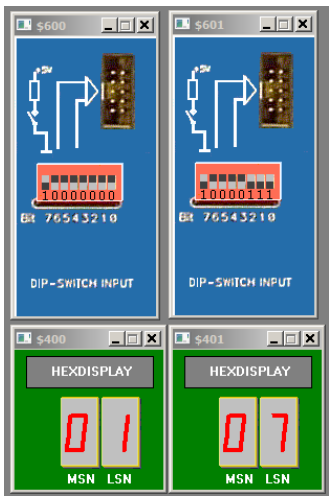
Segmentkoden för bokstaven ’E’ ges av:

```
#define      ERROR_CODE      0x5D
```

Uppgift 4 (6p) *Assemblerprogrammering*

Två strömbrytare och två indikatorer, enligt figuren till höger, är anslutna till adresser $600 och $601, respektive adress $400 och $401 i ett MC12-system.

Konstruera en subrutin `AddUnsigned8bitTo16,` i HCS12 assemblerspråk, som adderar de två värdena som ställs in på strömbrytarna (dessa ska tolkas som tal utan tecken) och därefter presenterar resultatet som ett 16 bitars tal på displayindikatorerna.
Subrutinen ska utformas så att avläsning och indikering görs en gång.



Speciellt gäller att endast symboler ska definieras och användas för absoluta adresser.

Uppgift 5 (6p) *Programmering med pekare*

Standardfunktionen **`strcat`** kan beskrivas på följande sätt:

```
char * strcat ( char * destination, const char * source );
```

Concatenate strings

Appends a copy of the *source* string to the *destination* string. The terminating null character in *destination* is overwritten by the first character of *source*, and a new null-character is appended at the end of the new string formed by the concatenation of both in *destination*.

Parameters

destination
Pointer to the destination array, which should contain a C string, and be large enough to contain the concatenated resulting string.
source
C string to be appended. This should not overlap *destination*.

Return Value

destination is returned.

Exempel på användning:

```
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[80];
  strcpy (str,"these ");
  strcat (str,"strings ");
  strcat (str,"are ");
  strcat (str,"concatenated.");
  puts (str);
  return 0;

}
```

Utskriften blir: `these strings are concatenated.`

Din uppgift är att, i C, skriva en egen definition av funktionen strcat. 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*

På laborationskortet *ML4*, finns bl.a. en *stegmotor*. Från ett program kan man styra denna motor via porten `ML4OUT` på adressen 0x400. Från denna port kopplas fyra enpoliga kablar till stegmotorn, på vilken de ansluts till fyra faser som kallas RÖD, BLÅ, GUL och VIT. Kablarna är anslutna till de höga bitarna, b7-b4, på utporten `ML4OUT` enligt följande:

	b7	b6	b5	b4	b3	b2	b1	b0
ML4OUT	RÖD	BLÅ	GUL	VIT	x	x	x	x

Stegmotorns axel kan fås att rotera genom att man lägger ut höga eller låga signaler till de olika faserna. För att få stegmotorn att rotera ett steg skall 0V kopplas till två av faserna medan de två andra faserna skall kopplas till +5V.

Här motsvarar +5V hög signal, dvs. en etta på `ML4OUT` och 0V låg signal, en nolla på `ML4OUT`. Riktningen som stegmotorn roterar framgår av följande tabell.

Stegmotorns rotationsriktning				
Fas	MEDURS → ← MOTURS			
	1	**2**	**3**	**4**
RÖD	+5V	+5V	GND	GND
BLÅ	GND	GND	+5V	+5V
GUL	GND	+5V	+5V	GND
VIT	+5V	GND	GND	+5V

Som tabellen visar finns det fyra tillstånd. Man ansluter först låg nivå (0V) till blå och gul fas samtidigt som faserna röd och vit får en hög nivå (+5V). För rotation medurs ändras i nästa tillstånd två av faserna, nämligen gul och vit som inverteras, osv. Varje tillståndsövergång innebär att stegmotorn roterar ett steg.

Eftersom stegmotorn behöver en viss tid för att rotera ett steg kan man inte byta tillstånd hur fort som helst. Du kan här anta att 10 ms är en tillräckligt lång tid mellan tillståndsbytena.

Uppgiften är att i C skriva en programmodul bestående av filerna `stegmotor.h` och `stegmotor.c`. I modulen skall det finnas två funktioner:

`void motor_init( void )`, som initierar stegmotorn så att den sätts i starttillståndet, och:

`void motor_vrid( int steg, int rotation)` som roterar stegmotorn ett visst antal steg.

Funktionen `motor_vrid` har alltså två parametrar, en som anger antalet steg motorn skall rotera och en som anger om rotationen skall ske medurs eller moturs. Om rotationen skall ske medurs ges denna parameter värdet 1 annars ges den värdet 0. Varje gång funktionen `motor_vrid` anropas skall motorn börja vrida sig utgående från den position den befinner sig i som resultat av tidigare anrop av funktionen. Du skall också skriva de definitioner som är nödvändiga för att programmet skall kunna komma åt utporten på kortet *ML4*.

I denna uppgift får du förutsätta att funktionen `hold( time_type ms)` 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.

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:

a)

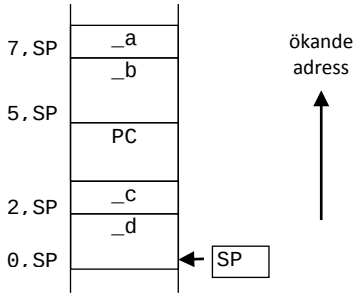
```
_a      RMB    2
_b      RMB    1
```

b)

```
        LDAB   _b
        PSHB
        LDD    _a
        PSHD
        JSR    _func
        LEAS   3,SP
```

c)

```
        LEAS   -3,SP
```



d)

```
;  c = a;
       LDAB   7,SP
       STAB   2,SP
;  d = b;
       LDD    5,SP
       STD    0,SP
;      c = *b;
       LDAB   [5,SP]
       STAB   2,SP
;  d = &a;
       LEAX   7,SP
       STX    0,SP
```

Uppgift 2:

```
   EXPORT _install_irq_handler   [r,2]
;    IrqHandler * install_irq_handler( IrqHandler irq, IrqHandler ** vector)
_install_irq_handler:
;    {
   LEAS   -2,SP
;      IrqHandler * p;
;      p = *vector;
   LDD    [6,SP]
   STD    0,SP
;      *( (IrqHandler *) vector) = irq;
   LDD    4,SP
   STD    [6,SP]
;
;      /* Assemblerinstruktion "CLI" */
;      return p;
   LDD    0,SP
;     }
   LEAS   2,SP
   RTS
```

Uppgift 3:

```
#define ML4OUTH     *(( unsigned char *) 0x400)
#define ML4OUTL     *(( unsigned char *) 0x401)
#define ML4IN       *(( unsigned char *) 0x600)
```

```
void   DisplayNBCD( 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 4:

```
; Adressdefinitioner
SWITCH1   EQU    $600
SWITCH2   EQU    $601
DISPLAY   EQU    $400

; Subrutin
AddUnsigned8bitTo16:
          LDAB   SWITCH1
          CLRA
          PSHD
          LDAB   SWITCH2
          ADDD   2,SP+
          STD    DISPLAY
          RTS
```

Uppgift 5:

```
char *strcat(char *s1, const char *s2)
{
   char *save = s1;

   while (*s1 != 0)
      s1++;
   while (*s2 != 0)
      *s1++ = *s2++;
   *s1 = 0;
   return(save);
}
```

Uppgift 6:

```
// Filen ports.h
#define ML4OUT_ADR 0x400
#define ML4OUT     *((portptr) ML4OUT_ADR)

// Filen stegmotor.h
void motor_init();
void motor_vrid(int antal_steg, int medurs);

// Filen stegmotor.c
#include "stegmotor.h"
#include "ports.h"
#include "clock.h"     // innehåller deklaration av funktionen hold

static port tillstand[] = {0x90, 0xA0, 0x60, 0x50};
static int aktuellt_tillstand= 0;

void motor_init()
{
       aktuellt_tillstand = 0;
}

void motor_vrid(int antal_steg, int medurs)
```

```
{
        int steg = (medurs) ? 1 : -1;
        int i;
        for (i=0; i<=antal_steg; i++) {
                aktuellt_tillstand =(aktuellt_tillstand+4+steg) % 4;
                ML4OUT = tillstand[aktuellt_tillstand];
                hold(10);
        }
}
```