Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

LEU500 Maskinorienterad programmering, DAI,EI,MEI

Fredag 19 augusti 2016, kl. 8.30 - 12.30

Examinator
Lars Bengtsson

Kontaktperson under tentamen:
Lars Bengtsson tel. 772 8441

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 (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:

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

Översätt *hela* funktionen `func`, som den är beskriven, till HCS12 assemblerspråk, såväl *prolog* som *tilldelningar* och *epilog* ska alltså finnas med. Speciellt ska du börja med att beskriva aktiveringsposten, dvs. stackens utseende i funktionen och där riktningen för minskande adresser hos aktiveringsposten framgår. (6p)

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

```
struct one * c;
long         a;
char         b;
```

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

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

Uppgift 2 (6p) *Assemblerprogrammering*

En 8-bitars strömbrytare, ”DIP_SWITCH” är ansluten till adress $600 och en displayenhet ”HEXDISPLAY” som visar en byte i form av två hexadecimala siffror är ansluten till adress $400 i ett MC12 mikrodatorsystem.

Konstruera en subrutin `DipHex` som läser av strömbrytaren och indikerar den minst signifikanta påslagna biten genom att skriva dess position, räknat från höger, till displayenheten. Om exempelvis bitarna 2 och 4 utgör ettställda strömbrytare ska positionen för bit 2, (dvs. 3) skrivas till displayenheten.

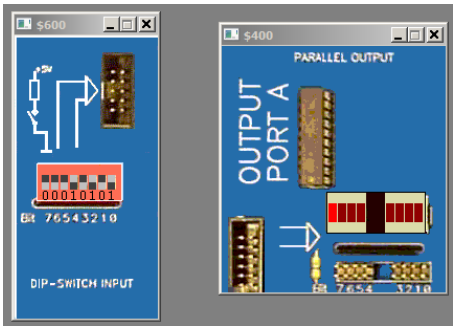
Om ingen strömbrytare är ettställd ska siffran 0 skrivas till displayen.

Speciellt gäller att endast symboler ska användas för absoluta adresser.

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 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 4 (6p) *Avbrott, HCS12*

Redogör för hur avbrott går till i ett HCS12-system, med en yttre enhet ansluten till IRQ-ingången på processorn. Ditt svar skall bland annat innehålla

- en övergripande beskrivning av vad avbrott är och hur det fungerar
- en beskrivning av de olika programrutiner som är förknippade med avbrott (eventuellt pseudokod / assembler)

Uppgift 5 (8p) *Programmering med pekare*

Funktionen `notInString` kan beskrivas på följande sätt:

```
int notInString ( const char * str1, const char * str2 );
```

Returns the length of the initial portion of *str1* which consists only of characters that are ***not*** part of *str2*.

Parameters
str1 C string to be scanned.
str2 C string containing the characters to match.

Return value

The length of the initial portion of *str1* containing only characters that does not appear in *str2*.
Therefore, if all of the characters in *str1* are in *str2*, the function returns zero if only the first character in *str1* is not in *str2*.
If none of the characters in *str1* are in *str2* the function returns the length of the entire *str1* string,

Exempel på användning:

```
int main()
{
  char string[]="Elm7a";
  printf("The number length is %d.\n", notInString(string,"1234567890"));
  return 0;
}
```

Utskriften ska då bli: ”The number length is 3.” eftersom tre tecken (Elm) som inte finns i referenssträngen föregår det första tecknet (7) som ju finns i referenssträngen.
Din uppgift är att, i C, skriva funktionen `notInString`. 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 (14p) *Maskinnära programmering i C*

a) Skapa en `struct tObject` som innehåller följande medlemmar:
- en pekare pName,
- ålder
- samt att man efteråt skall kunna instansiera variabler av typen utan att behöva ange nyckelordet `struct`. T ex `Object a`; (3p)

b) Initiera din instans med direktinitiering, dvs tillsammans med deklarationen. Använd ditt eget namn och ålder. (2p)

c) Punktnotation: Inkrementera åldern med medlem-notation. (1p)

d) Programmering av inbrottslarm. (8p)

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. Dataregistret är både läs- och 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)

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. Ett avbrott kvitteras genom att man nollställer motsvarande bit i 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. 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. Funktionen `alarm_off` ska helt enkelt stänga av utrustningen. Du ska också skriva en funktion `sensorinter` som anropas vid varje avbrott. Du ska inte skriva processorspecifika delar av avbrottshanteringen, dvs. du behöver inte bry dig om *hur* funktionen anropas vid avbrottet.

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 tända 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 1a:

Beskrivning av aktiveringspost

minnesanvändning	*adress*
ceasar	15,SP
bertil	14,SP
adam	10,SP
PC (vid JSR)	
a	4,SP
b	2,SP
c	0,SP

minskande adress ↓

```
;  void   func(long adam, unsigned char bertil, struct one * ceasar )
_func:
;  {
   LEAS   -8,SP
;  long            a = adam;
   LDY    10,SP
   LDD    12,SP
   STY    4,SP
   STD    6,SP
;  unsigned int b = bertil;
   LDAB   14,SP
   CLRA
   STD    2,SP
;  struct one   *c = ceasar;
   LDD    15,SP
   STD    0,SP
;  /* Övrig kod i funktionen är bortklippt eftersom vi bara
;     betraktar anropskonventionerna. */
;  }
   LEAS   8,SP
   RTS
```

Uppgift 1b: (4p):

```
_c:    RMB    2
_a:    RMB    4
_b:    RMB    1
       LDD    _c
       PSHD
       LDAB   _b
       PSHB
       LDD    2+_a
       PSHD
       LDD    _a
       PSHD
       JSR    _func

       LEAS   7,SP
```

Uppgift 2:

```
; Symboliska adresser
DipSwitch EQU    $600
HexDisp   EQU    $400
```

```
; Subrutin DipHex
DipHex:   LDAA   DipSwitch
          CLRB

DipHex10: TSTA
          BEQ    DipHex20

          INCB
          LSRA
          BCC    DipHex10

DipHex20: STAB   HexDisp
          RTS
```

Uppgift 3:

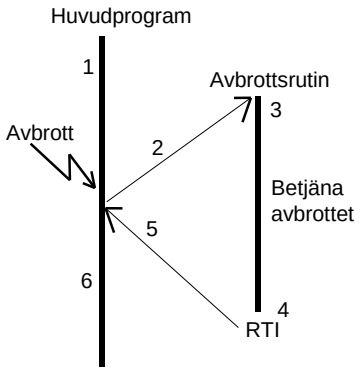
```
typedef   unsigned char * port8ptr;
#define OUT *((port8ptr) 0x400)
#define IN  *((port8ptr) 0x600)

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 4:

Ett sätt att avbryta processorns normala exekvering.

Händelseförloppet vid ett avbrott.

1. Processorn vars I-flagga är nollställd känner att IRQ är aktiverad.
2. Processorn sparar alla register på stacken. I-flaggan ettställs.
   Därefter läser processorn startadressen för avbrottsrutinen från den aktuella avbrottsvektorn som placeras i PC
3. Avbrottsrutinen startas
4. Avbrottsrutinen avslutas med instruktionen RTI dvs…
5. register innehållen återställs från stacken (Restore Status)
   Återhopp till huvudprogram med en nollställd I-flagga.
6. Därmed återstartas huvudprogrammet där det blev avbrutet.



Programrutiner: (Initieringsrutin och Avbrottsrutin)

```
* Initieringsrutin (En generell)
IrqInit
       ---    Init eventuella variabler som ingår i avbrottsrutinen
       ---       Nollställ avbrottsvippan
       LDX    #IrqRutin          Initiera avbrottsvektor
       STX    $FFF2
       CLI                       Aktivera avbrottsmekanismen
* Avbrottsrutin IrqRutin: En generell avbrottsrutin
IrqRutin
          ---       Kvitteta avbrott
          ---       Betjäna (Serva) Avbrottet
          rti
```

Uppgift 5:

```
int notInString(const char* cs, const char* ct)
{
   int n;
   const char* p;
   for(n=0; *cs; cs++, n++)
   {
      for(p=ct; *p && (*p == *cs); p++)
         ;
      if (!*p)
         break;
   }
   return n;
}
```

Uppgift 6

a)

```
typedef struct tObject {   // tObject kan skippas
   char   *pName;
   int    age;
} Object ;
```

b) `Object a = {“Sven”, 23};`

c) `a.age++;`

d)

```
/* 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);

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

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

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
}
```