Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D
EDA486 Programmering av inbyggda system Z
DAT016 Programmering av inbyggda system IT
DIT152 Programmering av inbyggda system GU
LEU500 Maskinorienterad programmering, DAI,EI,MEI

Tisdag 13 januari 2015, kl. 14.00 - 18.00, VV-salar

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson 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 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 2).

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

```
char    *e,f;
```

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

b) (2p) Visa också hur följande funktionsanrop översätts till assemblerkod för HCS12:

```
func( e , f );
```

c) (6p) Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

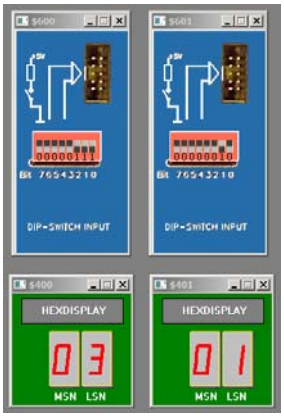
```
void  function( long b, char * a )
{
   int   c;
   char  d;

   d = 0;
   c = b;
   a = (char *) 0x1000;
   /* Övrig kod i funktionen är bortklippt eftersom vi bara
      betraktar anropskonventionerna. */
}
```

Översätt funktionen `function`, som den är beskriven till HCS12 assemblerspråk, dvs. visa hur utrymme för lokala variabler skapas och hur tilldelningarna kan utföras.

Speciellt ska du börja med att beskriva aktiveringsposten, dvs. stackens utseende i funktionen. Visa tydligt riktningen för minskande adresser hos aktiveringsposten.

Uppgift 2 (6p) *In och utmatning beskriven i C*

I denna uppgift ska du bland annat demonstrera hur absolutadressering utförs i C. Visa speciellt hur preprocessordirektiv och typdeklarationer används för att skapa begriplig programkod.



Två 8-bitars strömbrytare, är anslutna till adresserna 0x600,0x601 och två displayenheter som var och en visar en byte i form av två hexadecimala siffror är anslutna till adresserna 0x400 och 0x401 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DivModHex( void )
```

som läser de båda strömbrytarnas inställda värden.

Om värdet på adress 0x601 är noll ska 0xFF visas på båda displayenheter. Om värdet på adress 0x601 är skilt från noll ska resultatet av heltalsdivisionen mellan värden på adress 0x600 och 0x601 visas på displayindikator med adress 0x400 och resultatet av restdivisionen av samma tal visas på indikator med adress 0x401.

Uppgift 3 (8p) *Programmering med pekare*

För ett fält med heltal kan man definiera en operation *rotation vänster* på fältet som en operation som placerar fältets första heltal på den sista platsen i fältet, det andra heltalet på den första platsen, det tredje heltalet på den andra platsen o.s.v.

Skriv en funktion `rotateleft` som roterar ett fält med heltal ett godtyckligt antal steg åt vänster. Funktionen skall ha tre parametrar, en pekare till fältet som skall roteras samt två heltal vilka anger fältets längd respektive antalet rotationssteg. Parametrarna ska ges i denna ordning, funktionen har inget returvärde och får inte ha sidoeffekter, med heltal förstås datatypen **`int`**.

Lösningen får inte använda anrop till standardfunktioner från bibliotek, koden får heller inte använda indexering utan måste implementeras med hjälp av pekartyper.

Uppgift 4 (10p) *Assemblerprogrammering*

Följande funktion finns given i "C". Implementera en motsvarande subrutin i assemblerspråk för HC12. Du ska inte förutsätta några speciella kompilatorkonventioner i denna uppgift. Parametern 'cp' skickas i register X med anropet, returvärdet från subrutinen ska finnas i register Y efter att subrutinen utförts.

```
int   convert(char *cp )
{
   int converted = 0;
   while( *cp ){
      if( *cp < 0 ){
         *cp = -(*cp);
         converted++;
      }
      cp++;
   }
   return converted;
}
```

Uppgift 5 (16p) *Maskinnära programmering i C*

En robotarm styrs av en dator, via fem 8-bitars register: ett styr-/status- register på adressen 0x0800 två dataregister på adresserna 0x0801 respektive 0x0802. Styrregistret används för att kontrollera robotarmens rörelser och dataregistren används för att ange x- respektive y-koordinater som mål vid robotarmens förflyttning. Dessutom finns ytterligare två positionsregister på adresserna 0x0803 och 0x0804 som anger de aktuella x- respektive y-koordinaterna för robotarmen, dessa register är endast läsbara medan övriga register är både läs- och skrivbara. Följande tabell beskriver registren i robotens gränssnitt:

ROBOT											
Adress		7	6	5	4	3	2	1	0	Mnemonic	Namn
0x0800	*)	IE	ACT		IACK			ERR	IRQ	ctrl	Styr-/status- register
0x0801	R/W	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	datax	Data X
0x0802	R/W	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	datay	Data Y
0x0803	R	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	posx	Position X
0x0804	R	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	posy	Position Y

*) Bitar IE, ACT och IACK är endast skrivbara, ERR och IRQ är endast läsbara.

a) (2p) Visa en C-deklaration i form av en sammansatt datatyp (**`struct`**) som beskriver gränssnittet till roboten. Visa också lämpliga symboliska definitioner av bitarna i styrregistret.

b) (5p) Följande funktioner ska nu implementeras:

`void init (void)`	initierar roboten, placerar robotarmen i läge 0,0.
`void move (int x, int y)`	startar förflyttning av robotarmen till (x, y)

Visa en lösning som inte använder avbrott ("busy wait").

För att kunna använda avbrottsmekanismerna behöver vi två funktioner som inte kan implementeras med C-kod, dom kan dock beskrivas enligt följande:

`void robottrap(void);`	rutin som utförs vid avbrott, ska enbart anropa en C-rutin (`robotirq`).
`void cleari(void);`	denna rutin nollställer I-flaggan i statusregistret.

c) (2p) Visa nu en korrekt typdefinition (`asmfunc`) av en pekare till sådana funktioner.

d) (2p) Visa hur funktionerna `robottrap` och `cleari` implementeras i HCS12 assemblerspråk.

e) (5p) Visa en lösning som utnyttjar avbrott, och som implementerar följande funktioner:

`void init(asmfunc f);`	initierar roboten och förbereder systemet för avbrottshantering
`void move(int x, int y);`	startar förflyttning av robotarmen till (x, y)
`int status(void)`	ger ett värde: 0 om robotarmen är i vila, 1 om robotarmen är i rörelse och -1 om fel uppstått
`void robotirq(void)`	anropas vid avbrott

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

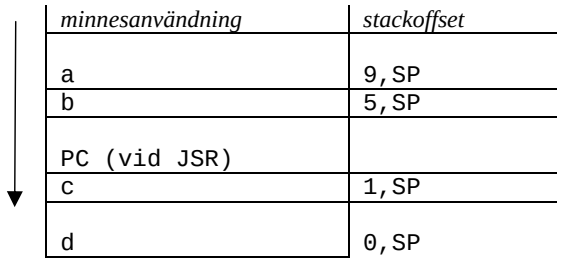
Uppgift 1a:

```
_e      RMB    2
_f      RMB    1
```

Uppgift 1b:

```
        LDAB   _f
        PSHB
        LDD    _e
        PSHD
        JSR    _func
        LEAS   3,SP
```

Uppgift 1c:



minnesanvändning	*stackoffset*
a	9,SP
b	5,SP
PC (vid JSR)	
c	1,SP
d	0,SP

```
_function:
    LEAS   -3,SP
; d = 0;
    CLR    0,SP
; c = b;
    LDD    7,SP1
    STD    1,SP
; a = (char *) 0x1000;
    LDD    #$1000
    STD    9,SP
    LEAS   3,SP
    RTS
```

Uppgift 2:

```
typedef unsigned char  *port8ptr;

#define ML4OUT_ADR1 0x400
#define ML4OUT_ADR2 0x401
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT1 *((port8ptr) ML4OUT_ADR1)
#define ML4OUT2 *((port8ptr) ML4OUT_ADR2)

#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void   DivModHex( void )
{
    unsigned char q,r,pa;
    pa = ML4IN2;
    if( pa != 0 )
    {
        q = ML4IN1/pa;
        r = ML4IN1%pa;
    }else{
        q = 0xFF;
        r = 0xFF;
    }
    ML4OUT1 = q;
    ML4OUT2 = r;
}
```

Uppgift 3:

```
void rotateleft(int *f, int l, int steps) {
    int i, *current, *next, save;

    for (;steps > 0; steps--) {
        current  = next = f;
        next++;
        i = l;
        save = *current; /* kommer att bli det sista elementet */
        while( i>1 )     /* för alla element utom det sista... */
        {
            *current++ = *next++;  /* skifta element vänster */
            i--;
        }
        *current = save; /* sista element på plats */
  }
}
```

Uppgift 4:

```
;  int convert(char *cp )
;   Parametrar:    'cp' i register X vid anrop
;   resultatet i register Y vid utträde
convert:
; {
;     int converted = 0;
    LDY    #0
;     while( *cp ){
convert_2:
    TST    ,X
    BEQ    convert_3
;       if( *cp < 0 ){
    BGE    convert_4
;          *cp = -(*cp);
    NEG    ,X
;          converted++;
    INY
convert_4:
    INX
;       }
;       cp++;
;       }
    BRA    convert_2
convert_3:
;       return converted;
;    }

    RTS
```

Uppgift 5a:

```
typedef struct sROBOT{
    volatile unsigned char ctrl;
    volatile unsigned char datax;
    volatile unsigned char datay;
    volatile unsigned char posx;
    volatile unsigned char posy;
}ROBOT, *PROBOT;

#define IE  0x80
#define ACT 0x40
#define IACK   0x10
#define ERR 2
#define IRQ 1
```

Uppgift 5b:

```
void move( int x, int y )
{
  ((PROBOT) (0x800))->datax = x;
  ((PROBOT) (0x800))->datay = y;
  ((PROBOT) (0x800))->ctrl = ACT;

  while( (((PROBOT) (0x800))->posx != x )
   || (((PROBOT) (0x800))->posy != y ) ) ;
}

void init( void )
{
  move( 0,0 );
}
```

Uppgift 5c:

```
typedef    void (* asmfunc ) (void);
```

Uppgift 5d:

```
import _robotirq
    export _robottrap
_robottrap:
    JSR _robotirq
    RTI

    export _cleari
_cleari:
    CLI
    RTS
```

Uppgift 5e:

```
static int robot_status;
void init(asmfunc f)
{
    *( (asmfunc *) 0x3F80)  = f;  /* sätt avbrottsvektor */
    ((PROBOT) (0x800))->ctrl = 0; /* passiva styrsignaler */
    robot_status = 0;
    cleari();
}

void move(int x, int y)
{
    ((PROBOT) (0x800))->datax = x;
    ((PROBOT) (0x800))->datay = y;
    robot_status = 1;
    ((PROBOT) (0x800))->ctrl = ACT|IE;
}

int status( void )
{
    return robot_status;
}
void robotirq( void )
{
    if( ((PROBOT) (0x800))->ctrl & ERR )
        robot_status = -1;
    else
        robot_status = 0;
    ((PROBOT) (0x800))->ctrl = 0;
}
```