Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

LEU500 Maskinorienterad programmering

Måndag 16 mars 2015, 14.00-18.00

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:
20p≤betyg 3 < 30p ≤ betyg 4 < 40p ≤ betyg 5

Uppgift 1 (6p)

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	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		
$701	R	0	0	0	0	0	0	0	0	DATA	Data Register
	W	1	1	1	1	1	1	1	1		

I figuren anges registrens innehåll efter "RESET".

- DDR: 1 anger att positionen är en utsignal, 0 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 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 2 (10p) *Kodningskonventioner (C/assemblerspråk)*

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

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

```
unsigned char     p;
unsigned int      a;
unsigned int      g_array[5];
int               *k;
```

a) Visa hur dessa deklarationer översätts till assemblerdirektiv för HCS12.

b) Med variabeldeklarationerna i a), visa hur följande tilldelningssats kan kodas i HCS12 assemblerspråk.

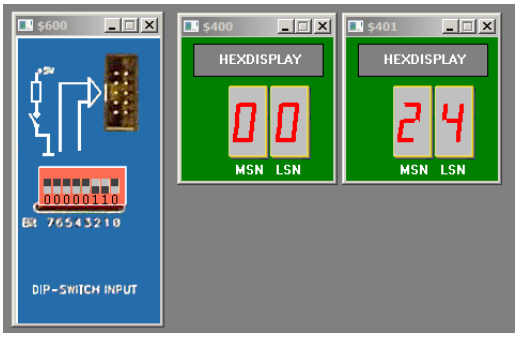
```
a = g_array[3];
```

c) Implementera en assembler subrutin som kan anropas från ett C-program.

```
unsigned short int getY( void );
```

Funktionen ska returnera det värde som register Y innehåller vid den punkt i programflödet där anropet görs.

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.
En strömbrytare och två visarindikatorer, enligt figuren till höger, är anslutna till adresser 0x600, respektive adress 0x400 och 0x401 i ett MC12 mikrodatorsystem.
Konstruera en funktion

```
void Square( void )
```

som bestämmer kvadraten på det värde som läses från strömbrytaren (tolka som tal utan tecken) och därefter presenterar resultatet som ett 16 bitars hexadecimalt tal på indikatorerna.

Uppgift 4 (8p) *Programmering med pekare*

Uppgiften är att skriva en C-funktion med namnet `leta` som har följande deklaration:

```
char *leta (const char *tecken, char *text);
```

Funktionen skall undersöka textsträngen `text` och returnera en pekare till det första tecknet i denna som finns bland tecknen i textsträngen `tecken`. Om inget av tecknen i `tecken` återfinns i `text` skall funktionen returnera en tom pekare. Båda textsträngarna avslutas med nolltecken, men dessa ingår inte i sökningen.

Exempel: Programraderna:

```
char t[] = "Kapitel 1.3, sidan 27";
const char siff[] = "0123456789";
printf("Siffror i texten \"%s\":\n", t);
for (char *p=t; (p=leta(siff,p)) != NULL; p++)
   printf ("%c ", *p);
printf ("\n");
```

skall ge utskriften:

```
Siffror i texten "Kapitel 1.3, sidan 27":
1 3 2 7
```

I denna uppgift får du inte använda någon standardfunktion, utan du måste skriva allt själv. Du får inte heller använda indexering, utan ska använda pekare.

Uppgift 5 (8p) *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 'c' skickas i register D med anropet, returvärdet från subrutinen ska också skickas i register D.

```
int isxdigit (int c)
{
   if( ( c >= '0' && c <= '9')  ||
       ( c >= 'a' && c <= 'f')  ||
       ( c >= 'A' && c <= 'F') )
          return 1;
   return 0;
}
```

Uppgift 6 (12p) *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		ACK			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.

ACT: Activate, sätts till 1 för att aktivera robotarmen, efter aktivering kommer denna att röra sig mot målkoordinaterna angivna i dataregistren. Positionsregistren uppdateras av roboten allt eftersom armen rör sig.

IE: Interrupt Enable, sätts till 1 för att aktivera avbrottsmekanismen i gränssnittet, efter aktivering genereras ett avbrott då robotarmen nått målkoordinaterna, dvs. data och positionsregistren överensstämmer. Avbrottsvektorns adress är 0xFF84.

IRQ: Om avbrott är aktiverat sätts denna bit till 1 då innehållen i data och positionsregistren överensstämmer.

ERR: Om fel inträffat, som gör att robotarmen inte kan röra sig mot dataregistrens koordinater, stoppas robotarmen, denna bit sätts till 1, om avbrott är aktiverat sätts då också IRQ-biten.

ACK: Acknowledge, då denna bit sätts till 1 återställs bitarna ERR och IRQ till 0 av robotens gränssnitt.

För att starta robotarmen krävs att:

1. Dataregistren initieras med målkoordinaterna.
2. Robotarmen aktiveras.
3. Om avbrott ska användas måste också avbrottsmekanismen aktiveras.
4. Då robotarmen nått målkordinaterna ska den deaktiveras.

Observera, för full poäng ska du i denna uppgift tydligt ange i vilken typ av fil (`.h`, `.c` eller `.asm`) varje lösningsdel ska placeras.

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

b) (3p) Följande funktioner ska nu implementeras:

`void init (void)`	initierar roboten, placerar robotarmen i läge 0,0.
`void move (int x, int y)`	utför 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 robotirq(void);`	rutin som utförs vid avbrott, ska enbart anropa en C-rutin (`robottrap`).
`void cleari(void);`	denna rutin nollställer I-flaggan i statusregistret.

c) (2p) Visa hur funktionerna `robotirq` och `cleari` implementeras i HCS12 assemblerspråk.

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

`void init( void );`	initierar roboten och förbereder systemet för avbrottshantering.
`void move(int x, int y);`	startar förflyttning av robotarmen till (x, y), avbrott ska genereras då armen nått målkoordinaterna.
`int status(void);`	ger ett värde: 0 om robotarmen är i vila, 1 om robotarmen är i rörelse och -1 om fel uppstått
`void robottrap(void);`	anropas, från assemblerrutin, 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 1:

a)

```
#define      PORTP_DDR    ((volatile unsigned char *) 0x700 )
#define      PORTP_DATA   ((volatile unsigned char *) 0x701 )

void portPinit( void )
{
       *PORTP_DDR = ~0xE0;
}
```

b)

```
unsigned char inPortP( void )
{
   return (( *PORTP_DATA  & 0xE0 )>> 5)  ;
}
```

c)

```
void outPortP( unsigned char c )
{
       *PORTP_DATA  = c & 0x1F ;
}
```

Uppgift 2a:

```
_p        RMB    1
_a        RMB    2
_g_array  RMB    10
_k        RMB    2
```

Uppgift 2b:

```
          LDD    _g_array+6
          STD_a
```

Uppgift 2c:

```
getSP:
          TFR    Y,D
          RTS
```

Uppgift 3:

```
typedef  unsigned char *port8ptr;
typedef  unsigned int  *port16ptr;
#define ML4OUT_ADR 0x400
#define ML4IN_ADR  0x600
#define ML4OUT16 *((port16ptr) ML4OUT_ADR)
#define ML4IN *((port8ptr) ML4IN_ADR)

void   Square( void )
{
   unsigned short int s;
   s = ( unsigned short ) ML4IN;
   s = s * s;
   ML4OUT16 = s;
}
```

Uppgift 4:

```
char *leta( char *tecken, char *text)
{
  char *p2;
  char *p1;
  for (p2=text; *p2; p2++)
  {
    for (p1=tecken; *p1; p1++)
      if (*p2 == *p1)
        return p2;
  }
  return (char *) 0;
}
```

Uppgift 5:

```
isxdigit:
;{
;  if( ( c >= '0' && c <= '9')  ||
;      ( c >= 'a' && c <= 'f')  ||
;      ( c >= 'A' && c <= 'F'))
    CPD    #'0'
    BLT    _1
    CPD    #'9'
    BLE    isxdigit_0
    CPD    #'A'
    BLT    _1
    CPD    #'F'
    BLE    isxdigit_0
    CPD    #'a'
    BLT    isxdigit_1
    CPD    #'f'
    BGT    isxdigit_1
isxdigit _0:
;     return 1;
    LDD #1
    RTS
isxdigit_1:
;  return 0;
    CLRA
    CLRB
; }
    RTS
```

Uppgift 6a:

```
/* i .h-fil */
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 6b:

```
/* i .c-fil */
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 ) ) ;

  ((PROBOT) (0x800))->ctrl &= ~ACT;
}

void init( void )
{
  ((PROBOT) (0x800))->ctrl = 0;
  move( 0,0 );
}
```

Uppgift 6c:

```
/* i .asm-fil */
```

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

    export  _cleari
_cleari:
    CLI
    RTS
```

Uppgift 6d:

```
/* i .c-fil */
extern void robotirq( void );
static int robot_status;
void init( void )
{
    *( (unsigned short *) 0xFF84)  = robotirq;   /* 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 |= ACK;         /* kvittera avbrott/felflagga */
    ((PROBOT) (0x800))->ctrl &= ~(ACT|IE);   /* nollställ bitar... */

}
```