Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA480 Maskinorienterad programmering D
EDA485 Maskinorienterad programmering Z
DAT015 Maskinorienterad programmering IT
DIT151 Maskinorienterad programmering GU

Måndag 22 augusti 2011, kl. 14.00 - 18.00

Examinatorer
Roger Johansson, tel. 772 57 29
Jan Skansholm, tel. 772 10 12

Kontaktpersoner under tentamen
Roger Johansson/Jan Skansholm

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

Inget annat än rättelser och understrykningar får vara införda i boken.

Tabellverk och 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 (10p) *Kodningskonventioner*

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (bilaga 2).

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

```
char    *a,*b,*c;
char    *min( char *a, char *b );
```

a) Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.

b) Visa hur följande sats översätts till assemblerkod för HCS12:

```
c = min( a , b );
```

Vissa instruktionssekvenser kan inte åstadkommas med hjälp av giltiga standard-C satser. Exempel på detta är att påverka enskilda bitar i processorns statusregister (CCR).

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

```
unsigned char getCCR( void );
```

- returvärdet är innehållet i CCR.

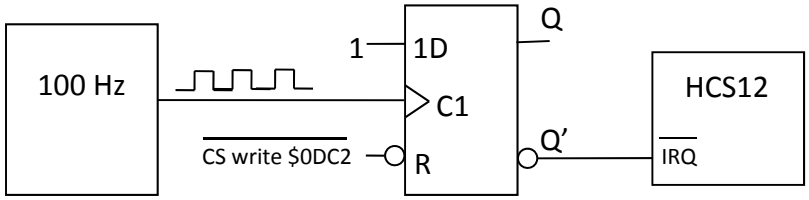
d) Implementera en assembler subrutin som kan anropas från ett C-program.

```
void setCCR( unsigned char value );
```

- parameter `value` anger nya värden för bitarna i CCR.

Uppgift 2 (8p) *Avbrott*

En pulsgenerator är ansluten via en avbrottsvippa till IRQ-ingången på ett MC12-system. Pulsgeneratorn har en frekvens på 100 Hz. För att nollställa avbrottsvippan krävs en skrivning på adressen $0DC2. Pulsgeneratorn är den enda anslutna avbrottskällan till IRQ-ingången på processorn.



a) Skriv en avbrottsrutin (IRQCNT) som läser en 8-bitars inport (IRQIN, adress $0600) och adderar det inlästa värdet till en 32-bitars variabel (IRQVAR). Både IRQIN och IRQVAR är variabler på tvåkomplementsform.

b) Skriv en initieringsrutin IRQINIT som initierar avbrottssystemet och som gör att IRQCNT anropas vid avbrott och att IRQVAR nollställs från början.

Uppgift 3 (6p) *Assemblerprogrammering*

Följande funktion finns given i “C”. Skriv motsvarande subrutin i assemblerspråk för HC12. För parameteröverföringen ska du anta att pekaren `s` placerats i register X före subrutinanropet.

```
#define DATA   *( char *) 0x700
#define STATUS *( char *) 0x701
void printerprint( char *s )
{
  while( *s )
  {
     while( STATUS & 1 )
     {}
     DATA = *s;
     s++;
  }
}
```

Uppgift 4 (6p) *Programmering med pekare*

Skriv en C-funktion med följande deklaration:

```
char *xcpy(const char *q1, char *q2);
```

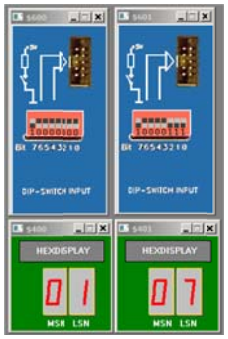
Funktionen skall kopiera den text q1 pekar på till det utrymme som pekas ut av q2, men vid kopieringen skall alla inledande och avslutande s.k. vita tecken *inte* tas med. Med vita tecken menas mellanslag, tabulator och nyradstecken. Tänk på att det kan finnas vita tecken *inne* i den text som skall kopieras. Dessa tecken skall förstås tas med. Funktionen xcpy skall som resultat ge en pekare till kopian av texten. Du får inte använda dig av några av C:s standardfunktioner, utan du måste skriva allt själv. *Tips*. Leta först framifrån efter första icke vita tecken. Sök sedan upp slutet på texten och leta därifrån bakåt efter sista icke vita tecken. Kopiera sedan den mellanliggande texten till det utrymme som pekas ut av q2.

Uppgift 5 (6p) *Maskinnära programmering i C*

Två strömbrytare och två displayenheter, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 och 0x401 i ett MC12 mikrodatorsystem. Konstruera en funktion

```
void AddUnsigned8bitTo16( void )
```

som adderar de två värdena som läses från strömbrytarna (tolka som tal utan tecken) och därefter presenterar resultatet som ett 16 bitars tal på displayindikatorerna.



Uppgift 6 *Programmering i C*

Denna uppgift handlar om att konstruera drivrutiner för en tänkt kommunikationsenhet. Enheten kan ta emot och sända data från resp. till en annan dator. Enheten har därför två separata kanaler: en mottagningskanal och en sändningskanal. Deluppgift a behandlar endast mottagningskanalen, medan deluppgift b behandlar sändingskanalen samt koppling mellan kanalerna.

Drivrutinerna för enheten använder sig av två buffertar: en inbuffert i vilken inlästa data läggs och en utbuffert som utgående data hämtas ifrån. Det är meningen att ett program som vill använda sig av kommunikationsenheten ska göra det via dessa buffertar. Vill programmet ta emot data via kommunikationsenheten ska det alltså hämta data från inbufferten och vill det sända data skall det lägga dessa data i utbufferten. Det finns en färdigskriven modul för buffertar som du ska använda dig av. (Men du behöver inte skriva dess `.c` fil.) Den har följande `.h` fil:

```
/* Filen buffer.h */
typedef struct bufferstruct buffer;  /* Full definition finns i filen buffer.c */
buffer *create_buffer(int max);   /* Ger en buffert med plats för max element */
int put(buffer *b, char c);       /* Försöker lägga in c sist i *b.
                                     Ger true om det gick, false annars */
int get(buffer *b, char *pc);     /* Försöker hämta första tecknet i *b till *pc.
                                     Ger true om det gick, false annars */
```

Följande specifikationer gäller för kommunikationenheten:
Det finns fyra åttabitars register: `SEND_DATA`, `SEND_CTRL`, `RECEIVE_DATA` och `RECEIVE _CTRL`. Dessa ligger på de hexadecimala adresserna 400, 401, 402 resp. 403. I dataregistren finns de data som ska tas emot eller sändas. I kontrollregistren används tre bitar:
Bit 0, `RDY` sätts automatiskt till 1 när en överföring är klar. Den ska nollställas vid en ny överföring.
Bit 1, `GO` sätter man till 1 för att starta en överföring. Den nollställs automatiskt.
Bit 2, `IE` anger om enheten skall ge avbrott när en överföring är klar.

Avbrottsvektorn för mottagningskanalen ligger på den hexadecimala adressen FFE8 och för sändningskanalen på FFEC.

Deluppgift a (8p)
Konstruera den del av drivrutinerna som har att göra med mottagning. Det ska bl.a finnas en fil `driver.h` med en tillhörande fil `driver.c`. Lägg dessutom gärna alla specifikationer för hårdvaran i en egen fil `channel.h`. Filen `driver.h` ska utgöra gränssnittet mot övriga program i datorn. I denna fil ska två funktioner deklareras: `init_input` som initierar mottagningsdelen av enheten och `get_input_buffer` som ger en pekare till den inbuffert drivrutinen skapat och använder sig av.

Dessa två funktioner ska förstås definieras i filen `driver.c`. I initieringsfunktionen ska bl.a. bufferten skapas. Låt den ha 100 platser. Initieringsfunktionen måste vara utformad så att det inte kan skapas mer än en inbuffert, även om funktionen skulle bli anropad mer än en gång.

I filen `driver.c` skall det dessutom finnas en funktion `notify_input`. Denna skall anropas varje gång ett avbrott sker. Detta görs från en assemblerrutin med namnet `input_intr`. Du behöver inte skriva `input_intr`, men du måste se till att avbrottsvektorn initieras så att den pekar på den. I funktionen `notify_input` ska det inlästa tecknet tas om hand och enheten sedan ställas i ett tillstånd som tillåter en ny överföring. I denna deluppgift behöver du inte kontrollera att ett inläst tecken ryms i bufferten.

Deluppgift b (6p)
Konstruera den del av drivrutinerna som har att göra med sändning. Utöka filen `driver.h` med deklarationer av följande tre funktioner: `init_output` som initierar sändningsdelen av enheten, `get_output_buffer` som ger en pekare till den utbuffert drivrutinen använder och `notify_output` som egentligen anropas varje gång ett avbrott har skett men som även måste kunna anropas från andra delar av programmet för att ”väcka” sändaren när den varit overksam.

Lägg till definitioner av dessa tre funktioner i filen `driver.c`. Initieringsfunktionen ska fungera på liknande sätt som för mottagninsdelen. Funktionen `notify_output` anropas från en avbrottsrutin i assembler med namnet `output_intr`. Du behöver inte skriva denna assemblerrutin men måste även här se till att avbrottsvektorn pekar rätt.

I funktionen `notify_output` ska normalt nästa tecken hämtas från utbufferten och sändas ut, men det ska finnas en koppling mellan mottagardelen och sändardelen av enheten. I deluppgift a behövde du inte ta hänsyn till om ett mottaget tecken fick plats i inbufferten, men det ska du göra nu. Det ska fungera på följande sätt.

Varje gång man i funktionen `notify_input` fått in ett tecken ska man kvittera det. Om det mottagna tecknet kunde läggas i inbufferten ska man skicka tecknet ACK (ASCII-kod 6, hexadecimalt) som svar och om tecknet inte kunde tas emot ska man skicka tecknet NACK (ASCII-kod 15, hexadecimalt). Själva sändningen ska förstås göras med hjälp av sändningsdelen. Sändning av dessa kvitton har prioritet framför sändning av normala data, vilket innebär att funktionen `notify_output` alltid måste kontrollera om ACK eller NACK väntar på att sändas innan den hämtar ett tecken från utbufferten. För att förenkla de något får du förutsätta att högst ett ACK- eller NACK-tecken väntar på att sändas i varje ögonblick.

Du måste här förstås också visa vilka tillägg som ska göras i funktionen `notify_input` från deluppgift a för att det hela ska fungera.

Bilaga 1 - 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)

Bilaga 2: Kompilatorkonvention XCC12:

Parametrar överförs till en funktion via stacken.
Då parametrarna placeras på stacken bearbetas parameterlistan från höger till vänster.
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

Lösningsförslag uppgift 1

a)

```
_a        RMB    2
_b        RMB    2
_c        RMB    2
```

b)

```
          LDD    _b
          PSHD
          LDD    _a
          PSHD
          JSR    _min
          LEAS   4,SP
          STD    _c
```

c)

```
_getCCR:
          TFR    CCR,B
          RTS
```

d)

```
_setCCR:
          LDAB   2,SP
          TFR    B,CCR
          RTS
```

Lösningsförslag uppgift 2

```
*****************************************************************
* AVBROTTSRUTIN-IRQCNT
* Beskrivning:      Läs 8-bitars tal (tvåkomplement) från port ($0600).
*                   Typkonvertera och addera till 32-bitars tal (IRQVAR)
*                   Kvittera avbrott (skrivning till adress $0DC2)
* Anrop:            via IRQ
*****************************************************************
IRQIN         EQU       $0600
IRQCLR        EQU       $0DC2

IRQCNT:       LEAS      -1,SP         ; plats för tecken-byte
              LDAB      IRQIN         ; läs 8 bitar
              SEX       B,D           ; teckenutvidga till 16 bitar i D
              STAA      0,SP          ; spara tecken-byte
              ADDD      IRQVAR+2      ; addera bit0-15
              STD       IRQVAR+2      ; uppdatera bit 0-15
              LDD       IRQVAR        ; bit 16-31
              ADCB      0,SP          ; addera bit 16-23
              ADCA      0,SP          ; addera bit 24-31
              STD       IRQVAR
              CLR       IRQCLR        ; nollställ avbrottsvippan
              LEAS      1,SP          ; återställ stacken
              RTI

IRQVAR        RMB       4
```

b)

```
*****************************************************************
* SUBRUTIN-IRQINIT
* Beskrivning:      Rutinen nollställer D-vippan, lägger in adressen
*                   till avbrottsrutinen på adressen $3FF2 och
*                   förbereder systemet för avbrott genom att I-flaggan
*                   nollställs.
* Anrop:            JSR    IRQINIT
*****************************************************************
IRQINIT:      MOVW      #0,IRQVAR       ; Init variabel
              MOVW      #0,IRQVAR+2
              CLR       IRQCLR          ; nollställ avbrottsvippan
              MOVW      #IRQCNT,$3FF2   ; avbrottsvektor
              CLI
              RTS
```

Lösningsförslag uppgift 3

```
void  printerprint( char *s )
_printerprint:
; {
;     while( *s )
printerprint1:
          TST       ,X
          BEQ       printerprint2
;     {
;         while( !( STATUS & 1) )
;         {}
printerprint3:
          LDAB      $0701
          ANDB      #$01
          BEQ       printerprint3

;     DATA = *s;
          LDAB      ,X
          STAB      $0700
;     s++;
          LEAX  1,X
          BRA       printerprint1
printerprint2:
;     }
; }
          RTS
```

Lösningsförslag uppgift 4

```
char *xcpy(const char *s1, char *s2) {
      const char *p1;
      char *p2;
      /* Hoppa över inledande vita tecken */
      while (*s1 && *s1 == ' ' || *s1 == '\t' || *s1 == '\n')
        s1++;
      /* Leta reda slutet av texten */
      for (p1=s1; *p1; p1++)
        ;
      /* Sök sista icke-vita tecken */
      for (p1--; p1>s1 && *p1 == ' ' || *p1 == '\t' || *p1 == '\n'; p1--)
        ;
      /* Kopiera till s2 */
      for (p2=s2; s1<=p1;)
        *p2++ = *s1++;
      *p2 = '\0';
      return s2;
    }
```

Lösningsförslag uppgift 5

```
typedef unsigned char  *port8ptr;
typedef unsigned short *port16ptr;

#define ML4OUT_ADR 0x400
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT16 *((port16ptr) ML4OUT_ADR)

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

void  AddUnsigned8bitTo16( void )
{
      unsigned short int s;
```

```
      while( 1 )
      {
         s = ( unsigned short ) ML4IN1;
         s = s + ( unsigned short ) ML4IN2;
         ML4OUT16 = s;
      }
}
```

Lösningsförslag uppgift 6

```
/* filen driver.h */
/* deluppgift a */
#include "buffer.h"
void init_input(void);
buffer *get_input_buffer();

/* deluppgift b */
void init_output(void);
buffer *get_output_buffer();
void notify_output(void);

/* filen channel.h */
typedef void (*vec) (void);  /* avbrottsvektor */
typedef vec *vecptr;         /* pekare till avbrottsvektor */

/* till deluppgift a */
#define RECEIVE_DATA (*(unsigned char *) 0x400 )
#define RECEIVE_CTRL (*(unsigned char *) 0x401 )
#define RECEIVE_VEC_ADR  0xFFE8
#define RECEIVE_VEC      *((vecptr) RECEIVE_VEC_ADR)
#define RDY 0x01
#define GO  0x02
#define IE  0x04

/* till deluppgift b */
#define SEND_DATA (*(unsigned char *) 0x402 )
#define SEND_CTRL (*(unsigned char *) 0x403 )
#define SEND_VEC_ADR  0xFFEC
#define SEND_VEC      *((vecptr) SEND_VEC_ADR)
#define ACK  0x06
#define NACK 0x15

/* filen driver.c */
/* deluppgift a */
#include "driver.h"
#include "channel.h"
#include "buffer.h"
#define BUF_SIZE 100

static buffer *input_buf = 0;
extern void input_intr(void);   /* avbrottsrutin i assembler */

void init_input(void) {
     if (!input_buf) {
        input_buf = create_buffer(BUF_SIZE);
        RECEIVE_VEC = input_intr;
        RECEIVE_CTRL = IE;
     }
}

buffer *get_input_buffer() {
  return input_buf;
}

void notify_input(void) {   /* anropas av input_intr  */
   if (RECEIVE_CTRL & RDY) {
     int ok = put(input_buf, RECEIVE_DATA);
     RECEIVE_CTRL = GO & IE;
     /* tillägg i deluppgift b */
     if (ok)
       answer = ACK;
     else
       answer = NACK;
```

```
     notify_output();
  }
}

/* deluppgift b */
static char answer = 0;
static buffer *output_buf = 0;
extern void output_intr(void);  /* avbrottsrutin i assembler */

void init_output(void) {
     if (!output_buf) {
        output_buf = create_buffer(BUF_SIZE);
        SEND_VEC = output_intr;
     }
     SEND_CTRL = RDY & IE;
}

buffer *get_output_buffer() {
  return output_buf;
}

void notify_output(void) {  /* anropas av output_intr eller av användare */
   char c;
   if (SEND_CTRL & RDY) {
     if (answer) {
       SEND_DATA = answer;
       answer = 0;
     }
     else if (get(input_buf, &c)) {
       SEND_DATA = c;
     }
     SEND_CTRL = GO & IE;
  }
}
```