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

Fredag 21 augusti 2015, kl. 8.30 - 12.30

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 (16p) *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 short    a;
unsigned short    g_array[50];
short             *k;
```

a) (4p) Visa hur dessa deklarationer översätts till assemblerdirektiv för HCS12.

b) (2p) Med variabeldeklarationerna i a), visa hur följande tilldelningssats kan kodas i HCS12 assemblerspråk. `a = g_array[32];`

c) (4p) Implementera en assembler subrutin som kan anropas från ett C-program.

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

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

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

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

   d = 0;
   c = b;
   e = a + (char *) 0x2000;
   /* 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 (4p) *Avbrott*

Ett gränssnitt består av 2 st. 8-bitars register.

Address	7	6	5	4	3	2	1	0	Mnemonic	Namn
$EC0	IRQ						IRQ1	IRQ2	STATUS	Status Register
$EC1							IACK1	IACK2	IACK	Interrupt acknowledge

Statusregistret innehåller IRQ-status, dvs. en ettställd bit representerar avbrott. Avbrottskällorna separeras med bitarna b_1 och b_0. b_7 ettställs om någon avbrottskälla är aktiv. Avbrotten kvitteras genom att 1 skrivs till motsvarande bit i *acknowledge*-registret.
Visa hur du utformar avbottshanteringen i programdelar (C och assembler) så att du minimerar mängden assemblerkod som krävs för implementeringen. Det betyder att du utformar:
C-funktioner för att:

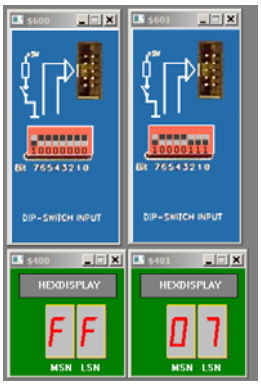
- initiera systemet för avbrott, avbrottsvektor 0x3FF2 ska användas.
- avbrottshanteringsfunktion som kvitterar det begärda avbrottet.

dessutom i assembler:

- nödvändiga stödfunktioner

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.



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 AddSigned8bitTo16( void )
```

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

Uppgift 4 (6p) *Programmering med pekare*

Standardfunktionen `strncmp` kan beskrivas på följande sätt:

```
int strncmp ( const char * str1, const char * str2, size_t num );
```

Jämför upp till `num` tecknen i C-strängen `str1` med C-strängen `str2`. Funktionen startar med att jämföra det första tecknet i varje sträng. Om de är lika med varandra, fortsätter jämförelsen med följande par tills tecknen skiljer sig, tills ett avslutande noll-tecken (’\0’) påträffas, eller tills `num` tecken överensstämmer i strängarna, beroende på vilket som inträffar först.

Parameterar

`str1`
: C-sträng för jämförelse.

`str2`
: C-sträng för jämförelse.

`num`
: Maximalt antal tecken att jämföra.
typen `size_t` är ett heltal utan tecken.

Returvärdet är ett heltal som indikerar relationen mellan strängarna:

värde	*indikerar*
`<0`	första tecknet som skiljer har ett mindre ASCII-värde i *str1* än i *str2*
`0`	strängarna är identiska
`>0`	första tecknet som skiljer har ett större ASCII-värde i *str1* än i *str2*

Din uppgift är att, i C, skriva en egen definition av funktionen `strncmp`. 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 5 (6p) *Assemblerprogrammering*

I C-biblioteket, kontrollerar funktionen `isalnum()` om ett tecken är *alfanumeriskt*, dvs. antingen alfabetiskt (a-z eller A-Z) eller numeriskt (0-9). Om ett tecken som skickas till `isalnum()` är alfanumeriskt (antingen alfabetet eller nummer), returneras ett heltal skilt från 0, om tecknet inte är alfanumeriskt returneras 0.

```
int isalnum(int argument);
```

Funktion `isalnum()` tar ett enda argument i form av ett heltal och returnerar ett heltal. Även om, `isalnum()` tar heltal som argument, antas den parameter som skickas till `isalnum()`vara ett tecken och det motsvarande ASCII-värdet används.

Skriv motsvarande subrutin `ISALNUM` i assemblerspråk för HC12. Observera att i denna uppgift ska du *inte* ta hänsyn till kompilatorkonventioner för XCC12. I stället skickas `argument` i register X och returvärdet från subrutinen i register D.

Se även bilaga ”ASCII-tabellen”.

Uppgift 6 (12p) *Maskinnära programmering i C*

Deluppgift a (5p)

Konstruera en C-funktion `timeout` som skall vänta tills antingen en viss tid gått eller tills en viss händelse inträffat. Funktionen skall ha två parametrar. Den första skall vara en pekare till en funktion som är parameterlös och som returnerar ett heltal. Detta heltal anger om händelsen man väntar på inträffat (1) eller inte (0). Den andra parametern anger time-out intervallet, uttryckt i millisekunder. Denna parameter skall vara av typen `long int`. Den andra parametern kan vara negativ och detta skall då tolkas som att time-out intervallet är oändligt långt. Som resultat skall funktionen `timeout` ge värdet 0 om den händelse den väntat på inträffat och värdet 1 om time-out intervallet löpt ut.

Du får förutsätta att det finns en *färdigskriven* modul `realtime` bestående av filerna `realtime.h` och `realtime.c`. I denna modul finns en privat (`static`) variabel `intnum` av typen `long int` som håller reda på antalet avbrott som kommit från en klockkrets ansluten till processorn. Klockkretsen ger ett avbrott var 5:e millisekund. Modulen har två funktioner som deklareras i filen `realtime.h`, funktionen `start_clock` som initierar avbrottsmekanismen och nollställer variablen `intnum` samt funktionen `get_intnum` som avläser och returnerar värdet i variabeln `intnum`. Funktionen `timeout` får förutsätta att funktionen `start_clock` har anropats tidigare.

Placera din funktion i en ny modul med namnet `my_realtime`. Skriv de filer som behövs för detta.

Deluppgift b (7p)

I ett hus skall ett inbrottsalarm installeras. En infravärmesensor som ger ett avbrott till datorn då en värmekälla detekteras skall användas. Sensorn har ett kombinerat 8-bitars status- och kontrollregister på adressen $A024_{16}$. Sensorn startas och stängs av genom att bit nr 7 i registret sätts till 1 resp. 0. När en värmekälla upptäcks sätter sensorn bit nr 6 i registret till 1. Om man har satt bit nr 1 i registret till 1 genererar sensorn även en avbrottssignal. Sensorn kan återställas efter ett alarm genom att man sätter bit 6 till 0. Avbrottsvektorn för sensorn ligger på adressen $FE20_{16}$.

Den andra deluppgiften är att i C skriva ett program som hanterar sensorn och som ger ett larm om sensorn indikerar en värmekälla. För att undvika falskt alarm skall programmet utformas så att det krävs två separata indikationer inom 15 sekunder från sensorn innan larm ges. Du får anta att det finns en färdigskriven funktion `raise_alarm` som man kan anropa för att ge larm.

Du får använda dig av funktionerna i modulerna `realtime` och `my_realtime` från a-uppgiften. (Det får du göra även om du inte löst den deluppgiften.) Din uppgift är alltså att skriva main-funktionen samt de funktioner som initierar sensorn och hanterar avbrott från den. För enkelhets skull får du anta att det i en modul med namnet `alarm_inter` finns en färdig avbrottsrutin med namnet `alarm_trap` skriven i assembler. Det enda denna funktion gör är att anropa en funktion med namnet `handle_alarm`. Du måste alltså själv skriva funktionen `handle_alarm`. Du kan däremot inte förutsätta att adressen till avbrottsrutinen (`alarm_trap`) är inlagd i avbrottsvektorn. Det måste du göra själv.

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)

Bilaga 3: ASCII tabell

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
0	NUL	20		10	DLE	30	0	40	@	50	P	60	`	70	p
1	SOH	21	!	11	DC1	31	1	41	A	51	Q	61	a	71	q
2	STX	22	"	12	DC2	32	2	42	B	52	R	62	b	72	r
3	ETX	23	#	13	DC3	33	3	43	C	53	S	63	c	73	s
4	EOT	24	$	14	DC4	34	4	44	D	54	T	64	d	74	t
5	ENQ	25	%	15	NAK	35	5	45	E	55	U	65	e	75	u
6	ACK	26	&	16	SYN	36	6	46	F	56	V	66	f	76	v
7	BEL	27	'	17	ETB	37	7	47	G	57	W	67	g	77	w
8	BS	28	(	18	CAN	38	8	48	H	58	X	68	h	78	x
9	HT	29	)	19	EM	39	9	49	I	59	Y	69	i	79	y
A	LF	2A	*	1A	SUB	3A	:	4A	J	5A	Z	6A	j	7A	z
B	VT	2B	+	1B	ESC	3B	;	4B	K	5B	[	6B	k	7B	{
C	FF	2C	,	1C	FS	3C	<	4C	L	5C	\	6C	l	7C	|
D	CR	2D	-	1D	GS	3D	=	4D	M	5D	]	6D	m	7D	}
E	SO	2E	.	1E	RS	3E	>	4E	N	5E	^	6E	n	7E	~
F	S1	2F	/	1F	US	3F	?	4F	O	5F	_	6F	o	7F	DEL

Lösningsförslag

Uppgift 1a:

```
_p        RMB   1
_a        RMB   2
_g_array  RMB   100
_k        RMB   2
```

Uppgift 1b:

```
          LDD   _g_array+64
          STD   _a
```

Uppgift 1c:

```
getX:     TFR   X,D
          RTS
```

Uppgift 1d:

minnesanvändning	*adress*
a	11,SP
b	9,SP
PC returadress	7,SP
c	3,SP
d	2,SP
e	0,SP

minskande adress ↓

```
_function:
   LEAS   -7,SP
;    d = 0;
   CLR 2,SP
;    c = b;
   LDD    9,SP
   LDY    #0
   STY    3,SP
   STD    5,SP
;    21 |    e = a + 0x2000;
   LDX    11,SP
   LEAX   $2000,X
   STX    0,SP
;    /* Övrig kod i funktionen är bortklippt eftersom vi bara
;    betraktar anropskonventionerna. */
;   LEAS  7,SP
   RTS
```

Uppgift 2:

I C:

```
#define   STATUS    *( ( volatile unsigned char *) 0xEC0)
#define   IACK      *( ( volatile unsigned char *) 0xEC1)

void init( void )
{
   extern void CLEARI(void);        /* assemblerrutin, nollställ I-flagga */
   extern void IRQ( void );         /* assemblerrutin, avbrott */
   IACK = 3;                        /* återställ avbrottsvippor */
   *((void(**)) 0x3FF2) = IRQ;      /* sätt avbrottsvektor */
   CLEARI();                        /* nollställ I */
}
void   AtIRQ( void )
{
   if (STATUS & 1 )
      IACK = 1;
   else if (STATUS & 2 )
      IACK = 2;
}
```

i assembler:

```
_CLEARI:  CLI
          RTS

_IRQ:     JSR    _AtIRQ
          RTI
```

Uppgift 3:

```
typedef  char    *port8ptr;
typedef  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   AddSigned8bitTo16( void )
{
       short s;
       while( 1 )
       {
          s = ( short ) ML4IN1;
          s = s + ( short ) ML4IN2;
          ML4OUT16 = s;
       }
}
```

Uppgift 4:

```
int strncmp(const char *s1, const char *s2, size_t n)
{
   while (1) {
      if (*s1 != *s2)
         return(*s1 - *s2);
      if (*s1 == 0 || --n == 0)
         return(0);
      s1++;
      s2++;
   }
}
```

Uppgift 5:

```
ISALNUM:
   CPX    #'A'      ($41)
   BLT    ISALNUM_1
   CPX    #'Z'      ($5A)
   BLE    ISALNUM_3
ISALNUM_1:
   CPX    #'a'      ($61)
   BLT    ISALNUM_2
   CPX    #'z'      ($7A)
   BLE    ISALNUM_3
ISALNUM_2:
   CPX    #'0'      ($30)
   BLT    ISALNUM_4
   CPX    #'9'      ($39)
   BGT    ISALNUM_4
ISALNUM_3:
   LDD    #1
   RTS
ISALNUM_4:
   CLRA
   CLRB
   RTS
```

Uppgift 6a:

```
// I filen my_realtime.h
int timeout( int (*f)(void), long int max_time);

// I filen my_realtime.c
#include "realtime.h"
#include "my_realtime.h"
int timeout(int (*f)(void), long int max_time) {
  long int stop;
  stop = get_intnum() + max_time / 5;
  if (max_time > 0)
    while(!f() && get_intnum() < stop)
      ;
  else
    while(!f())
      ;
  return f();
}
```

Uppgift 6b:

```
// I filen alarm_ports.h
#define SENSORREG_ADR     0xA024
#define SENSORREG         *((portptr) SENSORREG_ADR)
#define SENSOR_VEC_ADR      0xFE20   // Adress till avbrottsvektor
#define SENSOR_VEC          *((vecptr) SENSOR_VEC_ADR)
#define sensor_on_bit        0x80
#define sensor_signal_bit    0x40
#define sensor_intr_bit      0x02
#define set(x, mask) (x) = (x) | (mask)
#define clear(x, mask) (x) = (x) & ~(mask)

// I filen alarm.c
#include "realtime.h"
#include "my_realtime.h"
#include "alarm_ports.h"
#include "alarm_inter.h"

void raise_alarm();
static int signal;

void init_alarm() {
  port shadow = 0;
  signal = 0;
  SENSOR_VEC = alarm_trap;
  set(shadow, sensor_on_bit);
  set(shadow, sensor_intr_bit);
  SENSORREG = shadow;
}

void handle_alarm(void) {     // Anropas vid avbrott
  clear(SENSORREG, sensor_signal_bit);
  signal = 1;
}

int signal_on(void) {
  return signal;
}

int main() {
  start_clock();
  init_alarm();
  while (1) {
    if (signal) {   // första signal
      signal = 0;
      if (!timeout(signal_on, 15000)) { // vänta på andra signal
        signal = 0;
        raise_alarm();
      }
    }
  }
}
```