Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D
EDA486 Programmering av inbyggda system Z
DIT152 Programmering av inbyggda system GU

Torsdag 4 juni 2015, kl. 14.00 - 18.00, Hörsalsvägen

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
(även i svensk översättning)

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**:

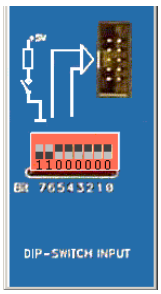
- redovisningen av svar och lösningar är läslig och tydlig. Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- lösningen ej är onödigt komplicerad.
- du har motiverat dina val och ställningstaganden
- assemblerprogram är utformade enligt de råd och anvisningar som 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.

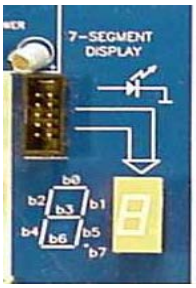
Betygsättning
För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända.

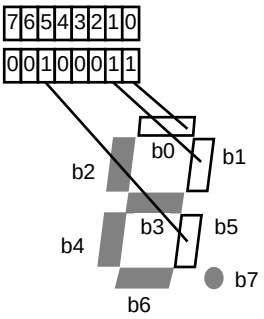
Tentamenspoäng ger slutbetyg enligt:
(EDA/DAT/LEU):
20p≤betyg 3 < 30p ≤ betyg 4 < 40p ≤ betyg 5
respektive (DIT):
20p≤betyg G < 35p ≤ VG

Uppgift 1 (8p) *Assemblerprogrammering*

En 8-bitars strömbrytare "DIP-SWITCH INPUT" och en sju-sifferindikator "7-SEGMENT DISPLAY" är anslutna till adresserna $0600 respektive $0400 i ett MC12 mikrodatorsystem.



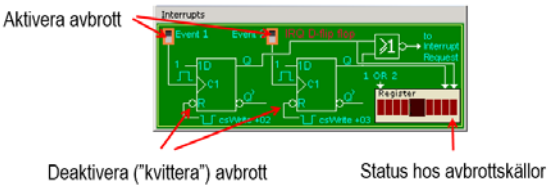




Skriv en subrutin `DisplayHEX` som kontinuerligt läser inporten (strömbrytarna) och skriver ett motsvarande värde som hexadecimal siffra till utporten (7-sifferindikatorn).
Översättningen av binär siffra till 7-segmetsrepresentation framgår av illustrationen längst till höger i figuren.
Om någon av bitarna 6, 5 eller 4 hos inporten är ettställd skall endast decimalpunkten tändas på indikatorn annars gäller att:
När bit 7 på inporten är ettställd skall sifferindikatorn släckas helt. När bit 7 på inporten är nollställd skall sifferindikatorn tändas och visa den hexadecimala siffra som är inställd på strömbrytarens bitar 3,2,1 och 0.

Uppgift 2 (6p) *Avbrott*

Under laborationerna har du kommit i kontakt med laborationskortet ML19, försett med två avbrottsvippor.



Gränssnittet består av 4 st. 8-bitars register, 3 av dessa används.

Address	7	6	5	4	3	2	1	0	Mnemonic	Namn
$DC0	IRQ						IRQ1	IRQ2	STATUS	Status Register
$DC1										
$DC2									IRQ1ACK	Interrupt 1 acknowledge
$DC3									IRQ2ACK	Interrupt 2 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 en skrivning till motsvarande *acknowledge*-register.
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) *Programmering med pekare*

Standardfunktionen `strncat` kan beskrivas på följande sätt:

```
char *strncat(char *s1, const char * s2, unsigned int n);
```
The `strncat` function appends not more than *n* characters from the array pointed to by `s2` to the end of the string pointed to by `s1`. The initial character of `s2` overwrites the null character at the end of `s1`. A terminating null character is always appended to the result. (Thus, the maximum number of characters that can end up in the array pointed to by `s1` is `strlen(s1)+n+1`.) If copying takes place between objects that overlap, the behavior is undefined.
The `strncat` function returns the value of `s1`.

Skriv en egen version av funktionen `strncat`. Du får *inte* använda indexering, utan måste använda dig av pekare. Du får *inte* heller använda dig av någon annan standardfunktion, utan måste skriva *allt* själv.

Uppgift 4 (8p) *Kodningskonventioner (C/assemblerspråk)*

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

a) I ett C-program har vi följande deklarationer:

```
void  func(int *pi, char *pc, long *pl )
{
   char  lc;
   int   li;
   long  *lpl;

   lc = *pc;
   li = *pi;
   lpl = pl;
   /* 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):

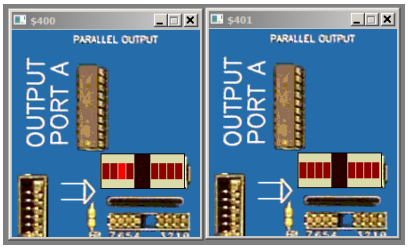
```
char  * b;
int   * a;
long  * c;
```

Visa hur variabeldeklarationerna översätts till assemblerdirektiv. (2p)

Uppgift 5 (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 ramper med ljusdioder, enligt figuren till höger, är anslutna till adress 0x400 och 0x401 i ett MC12 mikrodatorsystem.



Du ska konstruera en funktion

```
void rljush16( void )
```

som får dioderna att bete sig som ett kontinuerligt "rinnande ljus" där dioderna tänds upp en och en från vänster till höger. Efter det att bit 0 hos diodrampen på adress 0x400 släckts ska bit 7 hos diodrampen på adress 0x401 tändas. Då dioden för bit 0 på adress 0x401 släckts, ska det rinnande ljuset börja om från bit 7 på adress 0x400, osv. Använd (den givna) funktionen `void blinkdelay( void )`, för att åstadkomma fördröjning vid visningen av varje bit.

Uppgift 6 (16p) *Maskinnära programmering i C*

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:

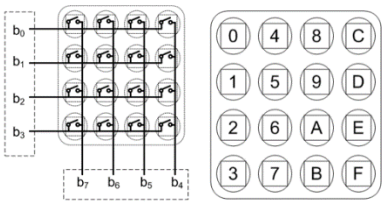
Parallel port P (PORTP)											
Address		7	6	5	4	3	2	1	0	Mnemonic	Namn
$700	R	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	DDR	Data Direction Register
	W	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN		
$701	R	0	0	0	0	0	0	0	0	DATA	Data Register
	W	1	1	1	1	1	1	1	1		

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 lämplig deklaration av porten med användning av en `struct` (2p).

Till porten ansluts nu ett tangentbord för koincidensavkodning, tangentbordet beskrivs av följande:



Till vänster visas hur tangentbordets rader ska anslutas till bitar b_3-b_0, medan kolumnerna ansluts till bitar b_7-b_4. En *sluten* brytare representerar en *nedtryckt* tangent. Avkodningen ska göras så att en rad åt gången aktiveras (1) och kolumnerna läses därefter av. Om samliga kolumner då är 0 betyder alltså detta att ingen tangent är nedtryckt.

b) Konstruera följande funktioner för tangentbordets gränssnitt (6p):

Visa en funktion, **`void`** `portKbdinit(`**`void`**`)` som initierar port P så att tangentbordet kan anslutas till porten.

En funktion `kbdActivate(`**`int`** `row)` , `row` kan vara 1,2,3 eller 4, räknat uppifrån, som aktiverar motsvarande rad, om `row` är 0 ska *alla* rader aktiveras.

En funktion **`int`** `kbdGetColumn(`**`void`**`)` som returnerar den kolumn (1,2,3,4 räknat från vänster) som är aktiv eller 0 om ingen kolumn är aktiv. Även om flera kolumner aktiveras samtidigt ska bara den första returneras.

c) Konstruera en funktion (8p):

```
unsigned char keyb( void )
```

Denna skall som resultat ge numret på den tangent som trycktes ner. Numreringen framgår av figuren till höger ovan.

- Funktionen skall först vänta tills alla tangenter är uppsläppta. Därefter skall den aktivera en rad i taget och avläsa kolumnernas utsignaler ända tills någon tangent tryckts ner.
- Då en nedtryck tangent konstaterats ska funktionen vänta 200 ms och därefter göra en ny avläsning. Om fortfarande samma tangent är nedtryckt är detta en korrekt tangentnedtryckning.
- Slutligen skall funktionen returnera tangentens nummer.

Du får förutsätta att det finns en färdig C-funktion **`void`** `hold( time_type ms );` som ger en fördröjning, parametern anger hur lång fördröjningen skall vara, enheten är millisekunder.

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 1:

```
ML4_INPUT:      EQU   $0600
ML4_OUTPUT:     EQU   $0400
ERROR_CODE:     EQU   $5D

DisplayHEX:     LDX   #SegCodes       ; Pekare till tabell
DisplayHEX1:    LDAA  ML4_INPUT       ; Läs strömbrytare
                BITA  #%01110000      ; Testa bit 6,5 och 4
                BEQ   DisplayHEX2     ; Om 0,0,0 fortsätt

                LDAB  #$80            ; Kod för decimalpunkt
                STAB  ML4_OUTPUT      ; Visa
                BRA   DisplayHEX1
DisplayHEX2:
                BITA  #%10000000      ; Testa bit 7
                BEQ   DisplayHEX3
                CLR   ML4_OUTPUT      ; Bit 7=1, släck
                BRA   DisplayHEX1

DisplayHEX3:    LDAB  A,X             ; Hämta segmentkod för [0,F]
                STAB  ML4_OUTPUT      ; Visa siffra
                BRA   DisplayHEX1

SegCodes:       FCB   $77,$22,$5B,$6B,$2E,$6D,$7D,$23
                FCB   $7F,$6F,$3F,$7C,$55,$7A,$5D,$1D
```

Uppgift 2:

I C:

```
#define   Status       *( ( volatile unsigned char *) 0xDC0)
#define   Irq1Ack      *( ( volatile unsigned char *) 0xDC2)
#define   Irq2Ack      *( ( volatile unsigned char *) 0xDC3)
void init( void )
{
   extern void CLEARI(void);  /* assemblerrutin, nollställ I-flagga */
   extern void IRQ( void );   /* assemblerrutin, avbrott */

   Irq1Ack = 0;        /* återställ avbrottsvippor */
   Irq2Ack = 0;
   *((void(**)) 0x3FF2) = IRQ;   /* sätt avbrottsvektor */
   CLEARI();        /* nollställ I */
}
void   AtIRQ( void )
{
   if (Status & 1 )
      Irq1Ack = 0;
   else if (Status & 2 )
      Irq2Ack = 0;
}
```

i assembler:

```
_CLEARI:  CLI
          RTS

_IRQ:     JSR    _AtIRQ
          RTI
```

Uppgift 3:

```
char *strncat(char *s1, const char *s2, unsigned int n)
{
   char *save = s1;

   while (*s1 != 0)
      s1++;
   while ((*s2 != 0) && (n > 0 ) )
   {
      *s1++ = *s2++;
      n--;
   }
   *s1 = 0;
   return(save);
}
```

Uppgift 4a:

minnesanvändning	*adress*
pl	11,SP
pc	9,SP
pi	7,SP
PC (vid JSR)	
lc	4,SP
li	2,SP
lpl	0,SP

minskande adress

```
_func:
    LEAS   -5,SP

    LDX    9,SP
    LDAB   ,X        ; lc = *pc;
    STAB   4,SP

    LDX    7,SP
    LDD    ,X        ; li = *pi;
    STD    2,SP

    LDD    11,SP       ; lpl = pl;
    STD    0,SP

    LEAS   5,SP
    RTS
```

Uppgift 4b:

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

Uppgift 5:

```
typedef unsigned int *port16ptr;
#define ML4OUT_ADR 0x400
#define ML4OUT16 *((port16ptr) ML4OUT_ADR)

void   rljush16( void )
{
       unsigned int c = 0;
       while( 1 )
       {
          if( c == 0 )
             c = 0x8000;
          ML4OUT16 = c;
          blinkdelay();
          c = c >> 1 ;
       }
}
```

Uppgift 6a:

```
typedef   struct sPortP{
       volatile unsigned char ddr;
       volatile unsigned char data;
}PORTP, *PPORTP;
#define      PORTP_BASE   0x700
#define      portP ((PORTP *)(PORTP_BASE))
```

b)

```
void portKbdinit ( void )
{
       portP->ddr = 0xF;
}

void kbdActivate( unsigned int row )
{
   switch( row )
   {
      case  1: portP->data = 1 ;   break;
      case  2: portP->data = 2 ;   break;
      case  3: portP->data = 4 ;   break;
      case  4: portP->data = 8 ;   break;
      case  0: portP->data = 0xF;  break;
   }
}

int kbdGetColumn ( void )
{
   unsigned char c;
   c = portP->data >> 4;
   if( c == 0 ) return 0;
   if ( c & 8 ) return 1;
   if ( c & 4 ) return 2;
   if ( c & 2 ) return 3;
   return 4;
}
```

c)

```
int keyb(void)
{
       int row, col;

       kbdActivate( 0 );          /* aktivera alla rader */
       while (kbdGetColumn() )    /* är någon tangent nedtryckt? */
          ;                       /* i så fall, vänta */

       while (1) {  /* upprepa tills någon tangent trycks ner */
          /* löp igenom alla rader och låt */
          for (row=1; row <=4 ; row++ ) {
             kbdActivate( row );                 /* aktivera raden   */
             if( col = kbdGetColumn () )         /* någon tangent nedtryckt ? */
             {
                hold( 200 );
                if( col == kbdGetColumn () )        /* fortfarande nedtryckt ? */
                {  /* ja, returnera tangentkod */
                   return 4*(row-1)+(col-1);
                }
             }
          }
       }
}
```