Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

DAT017 (DAT016) Maskinorienterad programmering IT
EDA482 (EDA481) Maskinorienterad programmering D
EDA487 (EDA486) Maskinorienterad programmering Z
DIT151 Maskinorienterad programmering GU
LEU500 Maskinorienterad programmering D,E,ME (hing)

Tisdag 9 januari 2018, kl. 8.30 - 12.30

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:
Roger Johansson, tel. 772 57 29

Tillåtna hjälpmedel

Utgåvor som distribuerats inom ramen för kursen, häftet:

- *Quick Guide, Laborationsdator MD407 med tillbehör*

Inget annat än understrykningar (”överstrykningar”) får vara införda i dessa häften.

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 getts under kursen.
- 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.

Maximal poäng är 50 och tentamenspoäng ger slutbetyg enligt: (EDA/DAT/LEU):
$20p \leq$ betyg $3 < 30p \leq$ betyg $4 < 40p \leq$ betyg 5
respektive (DIT):
$20p \leq$ betyg $G < 35p \leq VG$

Uppgift 1 (10p) *Assemblerprogrammering*

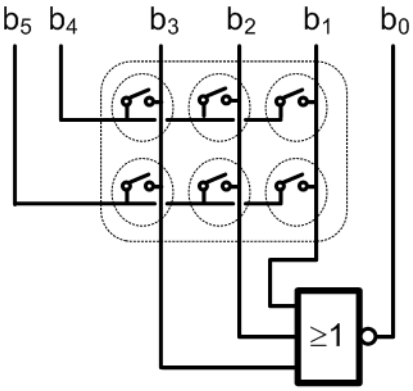
a) Deklarationerna `int a,b,c;` är givna på "toppnivå". Visa hur tilldelningen `a+=(b-c);` kodas i ARM v6 assemblerspråk.(4p)

b) Funktionen `int o( int )` är definierad. Visa hur följande funktion kan kodas i ARM v6 assemblerspråk. (6p)

```
int f ( int a )
{
  if ( a > o(a+5) )
    return 1;
  return 0;
}
```

Uppgift 2 (16p) *C-programmering*

Ett tangentbord för inmatning av sex olika tecken ska konstrueras. Sex stycken återfjädrande omkopplaare ansluts därför till port E hos *MD407*, på följande sätt:



Bit 0-3 kopplas till ingångar, bit 4 och bit 5 kopplas till utgångar hos port E.

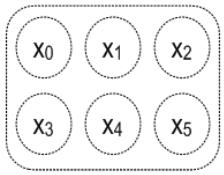
Nedtryckta tangenter kan detekteras genom att '1' skrivs till någon av bit 4 eller bit 5, därefter avläses bit 3-bit 1. För att detta ska vara tillförlitligt måste dessa bitars ingångar förses med "*pull-down*" samtidigt som utgångarna ska vara "*push-pull*". Metoden kallas *koincidensavsökning*.

Kretsen kan också användas för att känna av en tangentnedtryckning med hjälp av avbrott. För detta kopplas kolumnerna till en NOR-grind, vars utgång är kopplad till bit 0 hos port E, denna måste ha "*pull-up*".

Anm: Vi påminner om NOR-grindens funktion: utsignalen är '1' så länge alla ingångar är '0'. Då någon av ingångarna blir '1' blir utsignalen '0'.

a) Visa hur port E ska initieras för användning med tangentbordet. Bitarna b6 och b7 används inte, bitarna b8-b15 ska konfigureras som utport ("*push-pull*"). (4p)

b) Visa en programsekvens som konfigurerar så att bit 0 kan kopplas till avbrottssystemet hos MD407 via EXTI-modulen. Avbrott ska ske på negativ flank. Tänk på att endast de bitar som ska konfigureras får ändras, övriga ska ha kvar sina initialvärden. (4p)



7	6	5	4	3	2	1	0
		X0	X1	X2	X3	X4	X5

`char`

Tangenternas tillstånd kan representeras med en `char`, där en nedtryckt tangent motsvaras av att motsvarande bit är 1, annars är biten 0. Här används inte bitarna 6 och 7 och ska därför alltid vara 0.

c) Visa en tangentbordsfunktion `unsigned char get(void)`, som returnerar tangenternas tillstånd. (4p)

d) En enkel utmatningsenhet för visning av två hexadecimal siffror, i form av en `char` ska användas som indikator och ansluts därför till port E, bit 8-15. Skriv en funktion `void disp_kbd( void )` som kontinuerligt läser av tangentbordet (du får använda lösningen till uppgift c. även om du inte gjort den). Om exakt en tangent är nedtryckt, ska dess tillstånd, representerade av värdena 0-5, skrivas till visningsenhetens minst signifikanta hexadecimala siffra. Om ingen tangent är nedtryckt ska 0F skrivas till visningsenheten. Om två eller flera tangenter är nedtryckta ska FF skrivas till visningsenheten.(4p)

För full poäng krävs att dina lösningar är tydliga, fullständiga och att du använt lämpliga makrodefinitioner för registeradresser.

Uppgift 3 (6p)

Besvara kortfattat följande frågor.

a) Vad är den principiella skillnaden mellan statisk- och dynamisk testning? (2p)

b) För överföring av data mellan datorer används ofta seriekommunikation. Vad menar man med begreppet *protokoll*, i detta sammanhang? (3p)

c) Visa, med en figur, en nätverksstruktur av typen *stjärna*. (1p)

Uppgift 4 (6p)

Konstruera en C-funktion som undersöker en parameter med avseende på antalet 1-ställda bitar.

Funktionen deklareras:

```
int  bitcheck( unsigned int *pp, int * num );
```

`pp` är en pekare till det värde som ska undersökas
`num` är en pekare till en plats för returvärde, dvs. antalet 1-ställda bitar hos parametern
Funktionen ska returnera 1 om antalet ettor hos parametern är udda, annars ska funktionsvärdet vara 0.

Uppgift 5 (12p) *C-programmering*

Konstruera ett minimalt programpaket med *drivrutiner* för en USART-krets. Datastrukturer och funktioner ska utformas enligt följande:

a) Visa en typdefinition av en `struct` som avbildar USART-kretsen. Denna ska sedan användas vid konstruktion av funktionerna.

b) En funktion `void usart_put( unsigned char c )`, som matar ut ett tecken (`c`) via USART1.

c) En funktion `unsigned char usart_get( void )`, som returnerar ett tecken från USART1:s mottagarregister. Om inget nytt tecken kommit, ska funktionen returnera 0. Mottagarfunktionen ska utformas för användning av avbrott.

d) En funktion `void usart_init( unsigned short baud )` som initierar USART1 inför användningen (dubbelriktad kommunikation med protokollparametrar: 8 databitar 1 startbit, 1 stoppbit, ingen paritet). Parametern `baud` innehåller registerordet (USART_BRR) för den baudrate som ska användas.

Funktionerna behöver inte ta hänsyn till eventuella överföringsfel. Du kan givetsvis konstruera hjälpfunktioner, utöver de specificerade.

För full poäng på uppgiften ska koden vara enkelt läsbar, exempelvis ska tydliga macron användas för att specificera bitar i de olika registren.

Lösningsförslag

Uppgift 1a:

```
    LDR R3,b     @ R3← b
    LDR R2,c     @ R2← c
    SUB R3,R3,R2 @ r3← (b-c)
    LDR R2,=a    @ R2← &a
    LDR R1,[R2]  @ R1← a
    ADD R1,R1,R3 @ R1← a+(b-c)
    STR R1,[R2]  @ a← a+(b-c)
```

Uppgift 1b:

```
f:  PUSH  {R4,LR}
    MOV   R4,R0@ spill parameter till R4
    ADD   R0,R0,#5
    BL    o
    CMP   R4,R0
    BGT   .L1
    MOV   R0,#0
    B     .L2
.L1:
    MOV   R0,#1
.L2:
    POP   {R4,PC}
```

Uppgift 2a:

```
#define GPIO_E_MODER       ((volatile unsigned int *)0x40021000)
#define GPIO_E_OTYPER      ((volatile unsigned int *)0x40021004)
#define GPIO_E_PUPDR       ((volatile unsigned int *)0x4002100C)

  *GPIO_E_MODER = 0x55550500;
  /* bit 3,2,1 PULL DOWN, Bit 0 Pull up, ty IRQ aktiv låg */
  *GPIO_E_PUPDR = 0x000000A9;
  *GPIO_E_OTYPER = 0;
```

Uppgift 2b:

```
#define NVIC_EXTI0_IRQ_BPOS   (1<<6)
#define EXTI0_IRQ_BPOS        (1<<0)

#define SYSCFG_EXTICR1((volatile unsigned short *)  0x40013808)
#define EXTI_IMR          ((volatile unsigned int *)    0x40013C00)
#define EXTI_RTSR     ((volatile unsigned int *)    0x40013C08)
#define EXTI_FTSR     ((volatile unsigned int *)    0x40013C0C)
#define EXTI_PR       ((volatile unsigned int *)    0x40013C14)

#define NVIC_ISER0        ((volatile unsigned int *)    0xE000E100)

    *SYSCFG_EXTICR &= 0xFFF0;
    *SYSCFG_EXTICR |= 0x0004;            /* PE0->EXTI0  */
    *EXTI_IMR |= EXTI0_IRQ_BPOS;
    *EXTI_RTSR &= ~EXTI0_IRQ_BPOS;           /* EJ Avbrott på POSITIV flank */
    *EXTI_FTSR |= EXTI0_IRQ_BPOS;        /* Avbrott på NEGATIV flank */
    *NVIC_ISER0 |= NVIC_EXTI0_IRQ_BPOS; /* Aktivera avbrott i NVIC */
```

Uppgift 2c:

```
#define GPIO_E_ODR_LOW     ((volatile unsigned char *)0x40021014)
#define GPIO_E_IDR_LOW     ((volatile unsigned char *)0x40021010)
unsigned char get(void)
{
  unsigned char row1, row2;
  *GPIO_E_ODR_LOW = 0x10;
  row1 = (*GPIO_E_IDR_LOW << 2);
  *GPIO_E_ODR_LOW = 0x20;
  row2 = (*GPIO_E_IDR_LOW >> 1);
  return ( row1 | row2 ) &  0x3F;
}
```

Uppgift 2d:

```
#define GPIO_E_ODR_HIGH    ((volatile unsigned char *)0x40021015)
void disp_kbd(void)
{
  char c;
  while(1)
  {
    c = get();
    switch(c)
    {
      case 0: *GPIO_E_ODR_HIGH = 0x0F; break;  /* ingen knapp nedtryckt */
      case 1: *GPIO_E_ODR_HIGH = 0x01; break;
      case 2: *GPIO_E_ODR_HIGH = 0x02; break;
      case 8: *GPIO_E_ODR_HIGH = 0x03; break;
      case 16: *GPIO_E_ODR_HIGH = 0x04; break;
      case 32: *GPIO_E_ODR_HIGH = 0x05; break;
      case 64: *GPIO_E_ODR_HIGH = 0x00; break;
          default:
      *GPIO_E_ODR_HIGH = 0xFF; break; /* fler än en knapp nedtryckt */
    }
  }
}
```

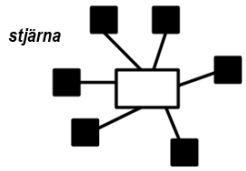
Uppgift 3a:

Statisk test utförs som dokumentgranskning medan dynamiska testmetoder förutsätter att programmet exekveras och utsätts för olika stimuli.

Uppgift 3b:

Med *protokoll* menas *regler* för hur en dataöverföring skall gå till, hur många databitar som skall skickas, hur snabbt databitarna skickas, vilka spänningsnivåer som används, hur handskakningsförloppet skall gå till, osv.

Uppgift 3c:



Uppgift 4:

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;

  while(*pp)
  {
    if( *pp & 1 )
      retval++;
    *pp >>= 1;
  }
  *num = retval;
  return retval & 1;
}
```

Uppgift 5a:

```
typedef struct tag_usart
{
    volatile unsigned short sr;
    volatile unsigned short Unused0;
    volatile unsigned short dr;
    volatile unsigned short Unused1;
    volatile unsigned short brr;
    volatile unsigned short Unused2;
    volatile unsigned short cr1;
    volatile unsigned short Unused3;
    volatile unsigned short cr2;
    volatile unsigned short Unused4;
    volatile unsigned short cr3;
    volatile unsigned short Unused5;
    volatile unsigned short gtpr;
} USART;
#define USART1  ((USART *) 0x40011000)
```

Uppgift 5b:

```
#define TXE         (1<<7) /* transmitter empty */
void    usart_put( unsigned char c )
{
    while (( USART1->sr & TXE )==0);
    USART1->dr = (unsigned short) c;

}
```

Uppgift 5c:

```
#define RXNE           (1<<5) /* receiver not empty */
void    usart_irq_routine( void )
{
    if( USART1->sr & RXNE )
        inbuf = (char) USART1->dr;
}

static unsigned char inbuf;
char    usart_get ( void )
{
    unsigned char  c = inbuf;
    inbuf = 0;
    return c;
}
```

Uppgift 5d:

```
#define NVIC_USART1_ISER          0xE000E104
#define NVIC_USART1_IRQ_BPOS      (1<<5)
#define USART1_IRQVEC             0x2001C0D4
#define EN          (1<<13) /* USART enable */
#define RXNEIE  (1<<5)  /* Receive interrupt enable */
#define RE          (1<<2)  /* Receiver enable */
#define TE          (1<<3)  /* transmitter enable */

void usart_init( unsigned short baud )
{
    inbuf= 0;
    *((void (**)(void) ) USART1_IRQVEC )  = usart_irq_routine;
   *((unsigned int *) NVIC_USART1_ISER) |= NVIC_USART1_IRQ_BPOS;
    USART1->brr = baud;
    USART1->cr2 = 0;
    USART1->cr3 = 0;
    USART1->cr1 = EN | RXNEIE | RE | TE ;
}
```