Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA482 Maskinorienterad programmering D
EDA483 Maskinorienterad programmering E
DAT017 Maskinorienterad programmering IT
DIT153 Maskinorienterad programmering GU

Måndag 13 mars 2023, kl. 14.00 - 18.00

Examinator

Roger Johansson, tel. 772 57 29
Erik Sintorn, tel. 772 52 29

Kontaktpersoner under tentamen:
Erik Sintorn, tel. 0729744821

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 detta häfte.

Tabellverk eller miniräknare får ej användas.

Lösningar
Anslås senast dagen efter tentamen via kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

Allmänt
Svar kan avges på svenska eller engelska.

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 tentamenspoäng är 40 och ger slutbetyg enligt:

$16p \leq$ betyg $3 < 24p \leq$ betyg $4 < 32p \leq$ betyg 5

Uppgift 1 (14p)

(a) Vi har deklarationerna:

```
signed   char   a;
signed   short *b;
unsigned int    c;
signed   char   foo( unsigned int );
```

Visa hur såväl variabeldeklarationerna som följande funktionsanrop kodas i ARMv6 assemblerspråk. (4p):

```
*b = (signed short) foo(a+c);
```

(b) Vi har deklarationerna:

```
int x, y[32];
```

Visa en kodsekvens i ARMv6 assemblerspråk där resultatet av uttrycket

```
y[x+1] + y[x-1]
```

evalueras till register R0 (4p).

(c) Följande funktion är given i C. Visa hur den kodas i ARMv6 assemblerspråk. För full poäng måste de kompilatorkonventioner vi använt i kursen följas, dessutom ska kommentarer i assemblerprogrammet utformas så att assemblerkoden kan kopplas till motsvarande konstruktion i C-programmet (6p).

```
unsigned int sumPositive( int *data, int length )
{
   int i;
   unsigned int sum = 0;
   for( i = 0; i < length; i++ )
   {
      if( data[i] > 0 )
         sum = sum + data[i];
   }
   return sum;
}
```

Uppgift 2 (6p)

C-funktionen `find_pos_of_setmsb` som bestämmer den mest signifikanta biten som är 1, i en parameter, ska konstrueras. Funktionen deklareras:

```
int find_pos_of_setmsb(unsigned int value, unsigned char * position);
```

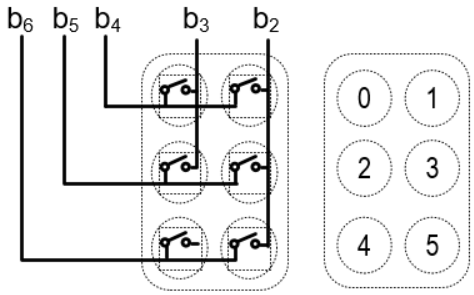
`value` är det värde som ska undersökas `position` är en pekare till en plats för resultatvärde, dvs. positionen för den mest signifikanta biten. Observera att den första positionen är 0.

Funktionen skall returnera 1 om någon 1:a hittas och annars skall funktionen returnera 0.

Exempel: Om `value` är 0x30 så skall värdet `position` pekar på sättas till 5 och funktionen skall returnera 1.

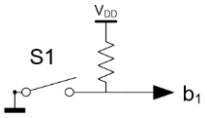
Uppgift 3 (10p)

Ett tangentbord för inmatning av olika tecken ska konstrueras. Sex stycken återfjädrande omkopplare ansluts därför till port E hos *MD407*, enligt figuren till höger. En nedtryckt tangent ska alltså representera ett värde 0-5.



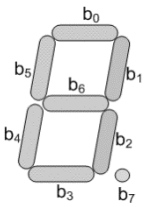
Nedtryckta tangenter kan detekteras genom att '1' skrivs till någon av bit 6, bit 5 eller bit 4, därefter avläses bit 3 respektive bit 2. För att detta ska vara tillförlitligt måste dessa bitars ingångar förses med "*pull-down*" samtidigt som utgångarna ska vara "*push-pull*".

Utöver tangentbordet ansluts en strömställare, S1 med "pull-up"-resistor. Via b_1 av port E detekteras om S1 är öppen eller sluten.



Visningsenhet

Utöver detta ansluts en utmatningsenhet i form av en Sju-sifferindikator där de enskilda segmenten tänds av '1' i en motsvarande sifferposition, se figur till höger. Indikatorn används för visning av en hexadecimal siffra och ansluts till port D b0-b7.



a) Visa med en funktion `void gpio_init(void)` hur port E och D ska initieras för användning med tangentbord och visningsenhet. (3p)

b) Visa en tangentbordsfunktion `unsigned char get(void)`, som returnerar tangentbordets tillstånd. Detta kan vara 0-5 om en tangent tryckts ned, 0xE om ingen tangent tryckts ned eller 0xF om tangentbordet är låst. Om flera tangenter trycks ned samtidigt ska en av tangentkoderna (godtyckligt vilken) returneras. Strömbrytare S1 används för att låsa och låsa upp tangentbordet. Tangentbordet ska vara låst då S1 är sluten. (4p)

c) Skriv ett huvudprogram som kontinuerligt skriver ut information från tangentbordet i form av en hexadecimal siffra. Värdet från funktionen `get` ska först översättas till segmentskod för sjusifferindikatorn, därefter skrivs segmentskoden tillutmatningsenheten. (3p)

I din lösning ska du använda *lämpliga makrodefinitioner för registeradresser*.

Uppgift 4 (8p)

I denna uppgift ska ett avbrott från SysTick konfigureras och en tillhörande avbrottshanterare konstrueras. Avbrottet ska genereras kontinuerligt med frekvensen 1000 Hz. Avbrottshanteraren ska uppdatera ett register (`DAC_OUTPUT`) med data från en array (`samples[]`). Vid varje avbrott ska avbrottshanteraren läsa in ett (1) element från arrayen och skriva till `DAC_OUTPUT`, vid påföljande avbrott ska nästa element i arrayen läsas in, osv. När sista elementet har lästs in ska avbrotthanteraren börja om från början av arrayen.

- Konstruera en funktion `systick_init()` som konfigurerar ett avbrott specfikation ovan.
- Konstruera en funktion `systick_irq_handler()` som hanterar avbrottet enligt specifikationen ovan.

Följande definitioner och variabler är givna:

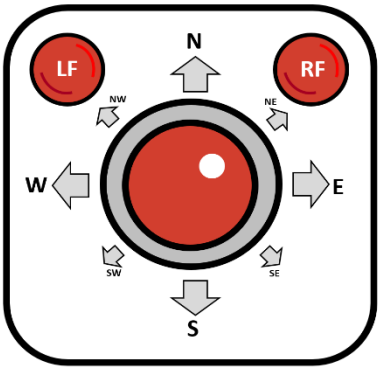
```
#define DAC_OUTPUT ((int*)0x4000741C)
#define SAMPLE_COUNT 42
int samples[SAMPLE_COUNT];
```

Uppgift 5 (12p)

En ”joystick” kopplas till *MD407* för att kunna styra en ”gubbe” i ett spel. Joystickens är kopplad till de övre 8 bitarna i GPIOE porten på följande vis:

15	14	13	12	11	10	9	8
N	**W**	**S**	**E**			**LF**	**RF**
r	r	r	r			r	r

Riktningssignaler:
N – Stickan är tryckt framåt
W – Stickan är tryckt åt vänster
S – Stickan är tryckt bakåt
E – Stickan är tryckt åt höger
Knappsignaler:
LF – Vänstra knappen nedtryckt
RF – Högra knappen nedtryckt



Notera att två riktningar kan vara nedtryckta på samma gång (joysticken kan till exempel vara tryckt ”framåt och åt höger”, men inte ”framåt och bakåt” samtidigt).

Alla signaler från joysticken är '1' för en aktiv signal och flytande annars.

Något annat kan tänkas kopplas in på de lägre 8 bitarna i GPIOE, så deras funktion skall inte ändras.

a) Skriv en funktion `void InitGPIO()` som initierar GPIO modulen för att kunna läsa av joysticken (2p).

b) Skriv en funktion `void UpdateJoystickDirection()` som updaterar en global variabeldirection, definerad som:

```
typedef enum { N=0x8, NW=0xC, W=0x4, SW=0x6, S=0x2,
               SE=0x3, E=0x1, NE=0x9, NONE=0x0} Dir;
Dir direction; (2p)
```

Den globala variabeln skall uppdateras omedelbart, via avbrott, när joystickens tillstånd ändras,.

c) Skriv en funktion `void InitSYSCFG()` som konfigurerar SYSCFG så att de relevanta IO pinnarna på Port E dirigeras till motsvarande avbrottslinor. (2p)
d) Skriv en funktion void `InitEXTI()` som konfigurerar EXTI modulen så att avbrott genereras när riktningtillståndet ändras. (2p)
e) Skriv en funktion void `InitNVIC()` som möjliggör dessa avbrott och initierar avbrottsvektorn så att `UpdateJoystickDirection()` körs när riktningstillståndet ändras (Vektortabellen är redan relokerad till 0x2001C000). (2p)
f) Vad saknas i `UpdateJoystickDirection()` för att systemet skall fungera efter första avbrottet? (2p)

Lösningsförslag

Uppgift 1 a

```
   .align
a: .space 1@ 1 byte to a
b: .space 4@ 4 bytes to b
c: .space 4@ 4 bytes to c

LDR  R0,=a @ &a -> R0
LDRB R0,[R0]   @ a -> R0
SXTB R0,R0 @ sign extend
LDR  R1,c  @ c -> R1
ADD  R0,R0,R1  @ a+c -> R0
BL   foo   @ call foo, argument in R0
LDR  R2,b  @ &b -> R2
STRH R0,[R2]   @ store short into b
```

Uppgift 1 b

```
LDR R0,=y  @ &y -> R0
LDR R1,x   @ x -> R1
ADD R2,R1,#1   @ x+1 -> R2
SUB R3,R1,#1   @ x-1 -> R3
LSL R2,#2  @ 4*(x+1) -> R2 (integer offset)
LSL R3,#2  @ 4*(x-1) -> R3 (integer offset)
LDR R4,[R0,R2] @ y[x+1] -> R4
LDR R5,[R0,R3] @ y[x-1] -> R5
ADD R0,R4,R5   @ y[x+1]+y[x-1] -> R0
```

Uppgift 1 c

```
sumPositive:
@ R0: int *data
@ R1: int length
@ R4: i
@ R5: sum
@ R6: loading data[i]
    PUSH    {R4,R5,R6} @ save contents of R4, R5 and R6 in stack
    MOV     R5,#0      @ 0 -> sum
    MOV     R4,#0      @ 0 -> i
for_loop:
    CMP     R4,R1      @ compare i with length
    BGE     for_exit   @ if i >= length then exit the loop
    LSL     R6,R4,#2   @ i*4 (integer offset) -> R6
    LDR     R6,[R0,R6] @ data[i] -> R6
    CMP     R6, #0     @ compare data[i] with 0
    BLE     skip       @ jump to skip if data[i] <= 0
    ADD     R5,R5,R6   @ sum = sum + data[i]
skip:
    ADD     R4,R4,#1   @ i++
    B       for_loop
for_exit:
    MOV     R0,R5      @ sum -> R0 (for the return)
    POP     {R4,R5,R6} @ restore contents of R4, R5, R6
    BX      LR
```

Uppgift 2

```
int find_pos_of_setmsb(unsigned int value, unsigned char *position){
    *position=0;
     while(value){
         if(value==1)
             return 1;
         (*position)++;
         value = value >> 1;
    }
    return 0;
}
```

Uppgift 3 a

```
/* Port D */
#define    PORT_D_BASE    0x40020C00
#define    GPIO_D_MODER   ((volatile unsigned int *)(PORT_D_BASE))
#define    GPIO_D_ODRLOW  ((volatile unsigned char *)   (PORT_D_BASE+0x14))
#define    GPIO_D_OTYPER  ((volatile unsigned short *)  (PORT_D_BASE+0x4))

/* Port E */
#define    PORT_E_BASE    0x40021000
#define    GPIO_E_MODER   ((volatile unsigned int *)(PORT_D_BASE))
#define    GPIO_E_OTYPER  ((volatile unsigned short *)  (PORT_D_BASE+0x4))
#define    GPIO_E_OSPEEDR ((volatile unsigned int *)    (PORT_D_BASE+0x8))
#define    GPIO_E_PUPDR   ((volatile unsigned int *)(PORT_D_BASE+0xC))
#define    GPIO_E_IDRLOW  ((volatile unsigned char *)   (PORT_D_BASE+0x10))
#define    GPIO_E_ODRLOW  ((volatile unsigned char *)   (PORT_D_BASE+0x14))

void init_gpio( void )
{
    *GPIO_E_MODER = 0x00001500;
    *GPIO_E_PUPDR = 0x000000A0;
    *GPIO_E_OTYPER= 0x00000000;
    *GPIO_D_MODER = 0x00005555;
    *GPIO_D_OTYPER= 0x00000000;
}
```

Uppgift 3 b

```
unsigned char get( void )
{
    if( *GPIO_E_IDRLOW & 2 )
        return 0xF;

    *GPIO_E_ODRLOW = 0x10;
    if( *GPIO_E_IDRLOW & 8 )
        return 0;
    if( *GPIO_E_IDRLOW & 4 )
        return 1;

    *GPIO_E_ODRLOW = 0x20;
    if( *GPIO_E_IDRLOW & 8 )
        return 2;
    if( *GPIO_E_IDRLOW & 4 )
        return 3;

    *GPIO_E_ODRLOW = 0x40;
    if( *GPIO_E_IDRLOW & 8 )
        return 4;
    if( *GPIO_E_IDRLOW & 4 )
        return 5;

    return 0xE;
}
```

Uppgift 3 c

```
void main(void)
{
static unsigned char segCode[]={
    0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
    0x7F,0x67,0x77,0x7C,0x39,0x5E,0x79,0x71 };

    init_gpio();
    while( 1 )
    {
        *GPIO_D_ODRLOW = seg_tab[ get() ];
    }
}
```

Uppgift 4:

```
#define DAC_OUTPUT ((int*)0x4000741C)
#define SAMPLE_COUNT 42
```

```
int samples[SAMPLE_COUNT];

#define STK_CTRL ((volatile unsigned int *)(0xE000E010))
#define STK_LOAD ((volatile unsigned int *)(0xE000E014))
#define STK_VAL ((volatile unsigned int *)(0xE000E018))

void systick_irq_handler();

void systick_init(void)
{
    *((void (**)(void) ) 0x2001C03C ) = systick_irq_handler;
    *STK_CTRL = 0;
    *STK_LOAD = 168 * 1000 - 1;
    *STK_VAL = 0;
    *STK_CTRL = 7;
}

void systick_irq_handler(void){
    static int i = 0;
    *DAC_OUTPUT = samples[i];
    i = (i + 1) % SAMPLE_COUNT;
    systick_init();
}
```

Uppgift 5 a

```
// (Port Mode Register):
#define GPIOE_MODER ((volatile unsigned int *)(0x40021000 + 0x0))
#define GPIOE_PUPDR ((volatile unsigned int *)(0x40021000 + 0xC))
void InitGPIO() {
        *GPIOE_MODER &= 0x0000FFFF;         (1p)
        // OTYPER and OSPEEDR do not need changing
        *GPIOE_PUPDR &= 0x0000FFFF;
        *GPIOE_PUPDR |= 0xAAAA0000;         (1p)
}
```

Uppgift 5 b

```
#define GPIOE_IDR_HIGH ((volatile unsigned char *)(0x40021000 + 0x11))
void UpdateJoystickDirection() {
    direction = *GPIOE_IDR_HIGH >> 4;
}
```

Uppgift 5 c

```
#define SYSCFG_EXTICR4 ((volatile unsigned short *)(0x40013800 + 0x14))
#define SYSCFG_EXTICR3 ((volatile unsigned short *)(0x40013800 + 0x10))
void InitSYSCFG() {
    *SYSCFG_EXTICR3 = 0x4444;
    *SYSCFG_EXTICR4 = 0x4444;
}
(helt okay om dom struntat I exticr3 för att dom tyckte fire knapparna var orelevanta)
```

Uppgift 5 d

```
#define EXTI_IMR ((volatile unsigned int*)(0x40013C00 + 0x0))
#define EXTI_RTSR ((volatile unsigned int*)(0x40013C00 + 0x8))
#define EXTI_FTSR ((volatile unsigned int*)(0x40013C00 + 0xC))

void InitEXTI() {
    *EXTI_IMR |= 0xF000;
    *EXTI_RTSR |= 0xF000;
    *EXTI_FTSR |= 0xF000;
}
```

Uppgift 5 e

```
#define NVIC_ISER1 ((volatile unsigned int*)(0xE000E100 + 0x4))
void InitNVIC() {
        *NVIC_ISER1 = (1<<8);
        *((void(**)(void)) 0x2001C0E0) = UpdateJoystickDirection;
}
```

Uppgift 5 f

```
#define EXTI_PR ((volatile unsigned int*)(0x40013C00 + 0x14))
void UpdateJoystickDirection() {
    if(*EXTI_PR & 0xF000) {
        ...;
        *EXTI_PR |= 0xF000;
    }
}
(”kvittera avbrottet” och/eller ”undvik EXTI 10 och 11” räcker som svar)
```