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)**

Måndag 12 mars 2018, kl. 14.00 - 18.00

---

**Examinator**
Roger Johansson, tel. 772 57 29

**Kontaktperson under tentamen**:
Roger Johansson, tel. 772 57 29
Ulf Assarsson,tel. 772 17 75

**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 (12p)

a) Deklarationerna `short a,b,c; int vi[18];` är givna på "toppnivå". Visa hur tilldelningen:

```
vi[c] = (a+b) & 0x0F;
```

kodas i ARM v6 assemblerspråk.(6p)

b) Funktionen `int testme ( int a, int b )` är definierad. Visa hur följande funktion kan kodas i assemblerspråk. Tillämpa samma kompilatorkonventioner som *gcc*. (6p)

```
int f ( int x, int y )
{
  if ( testme(x,y) != 10 )
    return 1;
  return 0;
}
```

## Uppgift 2 (10p)

MD407 ansluts till en skrivare och ett programpaket med funktioner för initiering och utskrift ska konstrueras. `init_app` för att konfigurera portar och initiera skrivaren. `printchar` för att skriva ut ett tecken och returnera status från operationen. `printstring` för att skriva ut en sträng med tecken och returnera status från operationen.

För full poäng ska du visa hur preprocessordirektiv och ev. typdeklarationer används för att skapa begriplig programkod samt tydligt ange i vilken typ av fil (.h, eller .c) varje lösningsdel ska placeras.

Skrivaren är kopplad till portar D och E hos *MD407*. Gränssnittet beskrivs av följande:

| GPIO | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | | | | | | | | | data | | | | | | | |
| | | | | | | | | | w | w | w | w | w | w | w | w |
| E | ctrl | | | | | | | | status | | | | | | | |
| | IE | IA | | | EN | | | RS | AT | IRQ | | PO | ER | RY | BS | |
| | rw | w | | | rw | | | rw | r | r | | r | r | r | r | |

Ett tecken skrivs ut genom att programmet:
1. kontrollerar om skrivaren är beredd att ta emot tecken.
2. placerar tecknet i dataregistret
3. aktiverar (”enable”) utskrift av ett tecken.

Anm: r: biten är läsbar, w:biten är skrivbar

AT: *Attention*: Biten sätts till 1 då någon av signalerna ”Ready”, ”Error”, ”Paper Out” är aktiverad. Om avbrott är aktiverat i styrregistret sätts även IRQ-biten.

PO: *Paper Out*: Pappersmagasinet är tomt, måste åtgärdas innan utskrift kan fortsätta, denna bit sätts till 1.

ER: *Error*. Då ett fel som gör att skrivaren inte kan fortsätta skriva ut tecken uppstår, stannar utskriften, och denna bit sätts till 1.

RY: Ready: Biten är 1 då skrivaren är beredd att ta emot ett nytt tecken för utskrift.

BS: Busy: Biten är 1 om skrivaren är upptagen med utskrift. Då skrivaren är upptagen accepteras endast RS-kommandot.

IRQ: *Interrupt Request*. Om avbrott är aktiverat sätts denna bit till 1 då AT-biten sätts, avbrottet är kopplat till processorn och har nummer 4 i vektortabellen. Systemets avbrottstabell har relokerats till adress 0x2001C000.

IE: *Interrupt Enable*, sätts till 1 för att aktivera avbrottsmekanismen i gränssnittet.

IA: *Interrupt Acknowledge*, då denna bit sätts till 1 återställs biten IRQ till 0 av robotens gränssnitt.

EN: *Enable*, vid en ”positiv flank” hos denna bit tar skrivaren ett tecken från dataregistret och skriver ut detta på pappret.

RS: *Reset*, då denna bit sätts till 1 återställs skrivaren och bitarna AT,PO, ER, sätts till 0 av skrivarens gränssnitt. Skrivaren kvarstannar i återställningstillståndet tills RS-biten återställs till 0.

a) (4p) Visa en funktion **`void`** `init_app(`**`void`**`)`, där portarna E och D initieras för att användas tillsammans med skrivaren. Det får förutsättas att de bitar som inte ingår i gränssnittet heller inte används. Alla skrivarens statussignaler är ”flytande” och aktivt höga. Funktionen ska även initiera (dvs. ”återställa”) skrivaren.

b) (6p) Utskriftsfunktionerna beskrivs av följande, implementera funktionerna.

| | |
|---|---|
| `int printchar ( char c );` | Om skrivaren är beredd att ta emot ett tecken för utskrift skickas detta och funktionen returnerar värdet **1**. Om skrivaren är upptagen med att skriva ut ett tecken ska värdet **0** returneras. Om skrivaren på grund av fel, eller slut på papper, väntar på åtgärd ska värdet **-1** returneras. |
| `int printstring ( char *s, int length);` | Funktionen skriver ut en textsträng med angiven längd. På grund av fördröjning hos skrivaren kan programmet tvingas göra flera försök, dock maximalt 100, för varje tecken. Om textsträngen skrivits ut korrekt, returneras värdet **1**, annars returneras värdet **0**. |

## Uppgift 3 (6p)

a) (3p) Under lektioner har vi diskuterat olika så kallade testprinciper. Ge tre exempel på principer som är giltiga för test.
b) (3p) En USART ska kommunicera med 115200 baud, 8 bitar ingen paritet. Visa hur kretsen ska initieras.

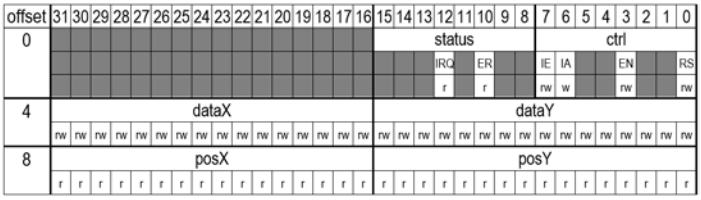
## Uppgift 4 (6p)

Konstruera en C-funktion som undersöker antalet 1-ställda bitar hos en parameter.

Funktionen deklareras:
`int oddbit(int a, unsigned int * num );`
`a` är det värde som ska undersökas, `num` är en pekare till en plats för ett resultat, dvs. antalet 1-ställda bitar hos parametern. Funktionen ska returnera 1 om antalet ettor hos `a` är udda, annars ska funktionsvärdet vara 0. Funktionen ska vara portabel, dvs. du kan inte göra antaganden om storleken av en `int`.

## Uppgift 5 (16p)

En robotarm styrs via ett gränssnitt med sex olika register: styrregister, statusregister, två dataregister och två positionsregister. Styrregistret används för att kontrollera robotarmens rörelser och dataregistren används för att ange x- respektive y-koordinater som mål vid robotarmens förflyttning. De båda positionsregisten anger de aktuella x- respektive y-koordinaterna för robotarmen. Följande beskriver robotens gränssnitt:




För att starta robotarmen krävs att:
1. Dataregistren initieras med målkoordinaterna.
2. Robotarmen aktiveras.
3. Om avbrott ska användas måste också avbrottsmekanismen aktiveras.
4. Då robotarmen nått målkordinaterna ska den deaktiveras.

Observera, för full poäng ska du i denna uppgift tydligt ange i vilken typ av fil (.h, eller .c) varje lösningsdel ska placeras.

Anm: r: biten är läsbar, w:biten är skrivbar

IRQ: *Interrupt Request*. Om avbrott är aktiverat sätts denna bit till 1 då innehållen i data och positionsregistren överensstämmer, avbrottet är kopplat till processorn och har nummer 4 i vektortabellen. Systemets avbrottstabell har relokerats till adress 0x2001C000.

ER: *Error*. Då ett fel som gör att robotarmen inte kan röra sig mot dataregistrens koordinater inträffat, stoppas robotarmen, och denna bit sätts till 1. Om avbrott är aktiverat sätts även IRQ-biten.

IE: *Interrupt Enable*, sätts till 1 för att aktivera avbrottsmekanismen i gränssnittet. Då avbrottsmekanismen är aktiverad genereras ett avbrott då data och positionsregistren överensstämmer, dvs. robotarmen nått målkoordinaterna.

IA: *Interrupt Acknowledge*, då denna bit sätts till 1 återställs biten IRQ till 0 av robotens gränssnitt.

EN: *Enable*, sätts till 1 för att aktivera robotarmen, efter aktivering kommer denna att röra sig mot målkoordinaterna angivna i dataregistren. Positionsregistren uppdateras av roboten allt eftersom armen rör sig.

RS: *Reset*, då denna bit sätts till 1 återställs bitarna ERR, IRQ,IE och EN till 0 av robotens gränssnitt. RS-biten måste därefter återställas till 0 för att robotarmen ska kunna aktiveras.

a) (3p) Visa en typdefinition i C (”`typedef`”) `ROBOT`, av en sammansatt datatyp (**`struct`**) som beskriver gränssnittet till roboten. Typdefinitionen ska utformas så att varje register kan adresseras individuellt via en

medlem i datastrukturen. Visa också lämpliga symboliska definitioner, i form av preprocessordirektiv, av bitarna i styrregistret.

b) (5p) Till höger visas en datastruktur kan användas för att beskriva en förflyttning mot en punkt.

```
typedef struct point
{
   unsigned short x;
   unsigned short y;
   int   timeout;
   int   exitval;
} POINT;
```

`x`,`y` anger slutmålet för förflyttningen, `timeout` anger maximal tid angivet i *ms*, som förflyttningen kan tillåtas ta. Värdet 0 innebär att det inte finns någon övre tidsgräns. `exitval` anger ett slutvärde som ska sättas av avbrottsrutinen ( se `robottrap` nedan) då förflyttningen avslutats.

För att implementera en timeout-bevakning ska du använda SYSTICK och skapa en *icke* blockerande fördröjning, det ska finnas tre funktioner:

| | |
|---|---|
| `void start_timeout(int timeout);` | Starta en timeout-period (`timeout` *ms*). Systemets klockfrekvens är 168 MHz, dvs. 1 mikrosekund motsvarar 168 räknade klockpulser. Eftersom upplösningen ska vara 1 *ms* ska också avbrottsintervallen vara 1 *ms*. |
| `void systick_irq_handler(void);` | Avbrottsfunktion: deaktivera SYSTICK, om ännu ej timeout, ska SYSTICK aktiveras för en ny period. |
| `int is_timeout(void);` | Returnera *true* om `timeout` *ms* passerat annars *false*. |

c) (8p) Implementera nu följande funktioner med hjälp av avbrottsmekanismerna:

| | |
|---|---|
| `int init(ROBOT *p);` | återställer och initierar roboten, förbereder systemet för avbrottshantering. Flyttar armen till koordinat 0,0 där ”oändlig” tid för förflyttningen är tillåten. Funktionen ska därefter vänta tills rörelsen avslutats och returnera värdet från `status`. (0 eller -1). |
| `void move(ROBOT *p,POINT *pt);` | startar förflyttning av robotarmen till punkt som anges av parameter `pt`. |
| `int status(ROBOT *p);` | ger ett returvärde: 0 om robotarmen är i vila, 1 om robotarmen är i rörelse och -1 om fel uppstått |
| `void robottrap(void);` | Avbrottsrutin, vid avbrott sätts variabeln `exitval` till något av slutvärdena:<br>0: Armen är vid målet inom utsatt tid.<br>1: Timeout, armen har inte nått målet inom utsatt tid.<br>-1: Fel, något problem uppstod med robotarmen.<br>I alla tre fall ska robotarmen deaktiveras och återställas. |

# Lösningsförslag

**Uppgift 1a:**

```
    LDR   R0,=a     @ R0← &a
    LDRH  R0,[R0]   @ R0← a
    SXTH  R0,R0     @ R0← (int) a
    LDR   R1,=b     @ R1← &b
    LDRH  R1,[R1]   @ R1← b
    SXTH  R1,R1     @ R1← (int) b
    ADD   R0,R0,R1  @ R0← (int)a+(int)b
    MOV   R1,#0x0F  @ R1← 0x0F
    AND   R0,R0,R1  @ R0← ((int)a+(int)b) & 0xF

    LDR   R1,=vi    @ R1← &vi
    LDR   R2,=c     @ R2← &c
    LDRH  R2,[R2]   @ R2← c
    LSL   R2,R2,#2  @ R2← i*sizeof(int)
    ADD   R1,R1,R2  @ R1← &vi+c
    STR   R0,[R1]   @ vi[c] ← ((int)a+(int)b) & 0xF
```

**Uppgift 1b:**

```
f:  PUSH  {LR}
    BL    testme
    CMP   R0,#10
    BNE   .L1
    MOV   R0,#0
    B     .L2
.L1:
    MOV   R0,#1
.L2:
    POP   {PC}
```

**Uppgift 2a:**

```
#define    GPIO_D_BASE    0x40020C00     /* MD407 port D */
#define    GPIO_D_MODER   ((volatile unsigned int *)    (GPIO_D_BASE))
#define    GPIO_D_OTYPER  ((volatile unsigned short *)  (GPIO_D_BASE+0x4))
#define    GPIO_D_PUPDR   ((volatile unsigned int *)    (GPIO_D_BASE+0xC))
#define    GPIO_D_ODR     ((volatile unsigned char *)   (GPIO_D_BASE+0x14))

#define    GPIO_E_BASE    0x40021000     /* MD407 port E */
#define    GPIO_E_MODER   ((volatile unsigned int *)    (GPIO_D_BASE))
#define    GPIO_E_OTYPER  ((volatile unsigned short *)  (GPIO_D_BASE+0x4))
#define    GPIO_E_PUPDR   ((volatile unsigned int *)    (GPIO_D_BASE+0xC))
#define    GPIO_E_ODR     ((volatile unsigned char *)   (GPIO_D_BASE+0x15))
#define    GPIO_E_IDR     ((volatile unsigned char *)   (GPIO_D_BASE+0x10))

#define    data_r     GPIO_D_ODR
#define    ctrl_r     GPIO_E_ODR
#define    status_r   GPIO_E_IDR

#define AT     (1<<7)
#define IRQ    (1<<6)
#define PO     (1<<4)
#define ER     (1<<3)
#define RY     (1<<2)
#define BS     (1<<1)
#define IE     (1<<7)
#define IA     (1<<6)
#define EN     (1<<3)
#define RS     (1<<0)


void init_app( void )
{
    *GPIO_D_MODER  = 0x00005555;
    *GPIO_D_OTYPER = 0;           /* punkt till punkt, använd push/pull */

    *GPIO_E_MODER  = 0x55550000;
    *GPIO_E_OTYPER = 0;           /* punkt till punkt, använd push/pull */
    *GPIO_E_PUPDR  = 0x0000AAAA;  /* Flytande, aktivt höga -> ”pull down” */

    *ctrl_r = RS;                 /* Initiera skrivare */
    *ctrl_r = 0;
}
```

**Uppgift 2b:**
Anm: Biten BS är komplementet av RY

```
int printchar (char c )
{
  if (*status_r & RY )   eller
  if ((*status_r & BS )==0)

  {
    *data_r = c;
    *ctrl_r &= ~EN;  /* generera en positiv flank */
    *ctrl_r |= EN;
    return 1;
  }else if(*status_r & (PO|ER) )
  {
    return -1;
  }else
    return 0;
}
int  printstring(unsigned char *s,  int length)
{
  int status;
  for( int j = 0; j < length; j++)
  {
    for( int i = 0; i < 100; i++ )
    {
       status = printchar ( s[j] );
       if( status < 0 ) return 0;
       if( status ) break;
    }
    if( ! status ) return 0;
  }
  return 1;
}
```

**Uppgift 3a:**

Tre av följande principer ska finnas i svaret:

- Test påvisar närvaro av fel.
- Det är i allmänhet inte möjligt att utföra fullständigt uttömmande test.
- Testning ska inledas tidigt i utvecklingen.
- Fel har en benägenhet att ”ansamlas” – test upptäcker ofta sådana ansamlingar
- Immunitetsparadoxen
- Test utformas beroende på sitt sammanhang
- ”Frånvaro av fel” – Att test inte indikerar fel betyder inte att de inte finns.

**Uppgift 3b:**

```
void   _init(void)
{
       USART1->brr = 0x2D9;
       USART1->cr3 = 0;
       USART1->cr2 = 0;
       USART1->cr1 = UE | TE | RE;
}
```

**Uppgift 4:**

```
int oddbit (int a, int *num)
{
  int  numbits = 0;

  while(a)
  {
    if( a & 1 )
      numbits ++;
   (unsigned int) a >>= 1;
  }
  *num = numbits;
  return *num & 1;
}
```

**Uppgift 5a:**

```
/* i .h-fil */
typedef struct sROBOT{
    volatile unsigned char ctrl;
    volatile unsigned char status;
    volatile unsigned short reserved;
    volatile unsigned short dataX;
    volatile unsigned short dataY;
    volatile unsigned short posY;
    volatile unsigned short posX;
}ROBOT;

#define IRQ     (1<<4)
```

```
#define ER      (1<<2)
#define IE      (1<<7)
#define IA      (1<<6)
#define EN      (1<<3)
#define RS      (1<<0)
```

**Uppgift 5b:**

```
/* i .c-fil */
static int timeout_counter;
void    systick_irq_handler ( void )
{
    timeout_counter --;
    *STK_CTRL = 0;
    if( irq_count == 0 )
    {   /* 1 ms har passerat… */
        return;
    }
    *STK_CTRL = 7; /* Ytterligare en period */
}
static int timeout_counter;
void start_timeout( int timeout )
{
    if( timeout <= 0 )
        return;
    *((void (**)(void) ) 0x2001C03C ) = systick_irq_handler;
    *STK_CTRL = 0;
    *STK_LOAD = ( 168000 - 1 );   /* 1 ms */
    *STK_VAL = 0;
    *STK_CTRL = 7;
    timeout_counter = timeout;
}
int is_timeout(void)
{
    return ( timeout_counter==0 );
}
```

**Uppgift 5c:**

```
/* i .c-fil */
static ROBOT *active_robot;
static POINT *active_point;
int init (ROBOT *p)
{
  int retval;
  POINT origin;
  p->ctrl = RS;  /* återställ robotarm */
  p->ctrl = 0;   /* Deaktivera alla styrsignaler */
  origin.x = 0;
  origin.y = 0;
  origin.timeout = 0;
  move( p, &origin );   /* Flytta robotarm till startposition */
  while( (retval = status( p ) ) == 1); /* vänta tills rörelse avslutad */
  return retval;
}
void move(ROBOT *p, POINT *pt )
{
  *( (void (**) (void)) 0x2001C050)  = robotirq;/* sätt avbrottsvektor */
  active_robot = p;     /* Behövs senare i avbrottsrutin */
  active_point = pt;   /* Behövs senare i avbrottsrutin */
  p->dataX = pt->x;
  p->dataY = pt->y;
  p->ctrl |= (EN | IE);
  start_timeout( pt-> timeout );
}
int status(ROBOT *p)
{
  if( p->status & ER )
    return -1;  /* Felindikator */
  if( p->status & EN )
    return 1;   /* Aktiverad, därför i rörelse */
  return 0;  / I vila */
}
void robottrap(void)
{
  active_robot->ctrl = IA;/* kvittera avbrott */
  if( active_robot->status & ER )
    active_point->exitval = -1;
  else if( is_timeout() )
    active_point->exitval = 1;
  else
    active_point->exitval = 0;
}
```