Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Hemtentamen med delvisa lösningsförslag

EDA482 Maskinorienterad programmering D
EDA487 Maskinorienterad programmering Z
DIT151 Maskinorienterad programmering GU
DAT017 Maskinorienterad programmering IT
DAT390 Maskinorienterad programmering Hing
LEU500 Maskinorienterad programmering Hing

Fredag 5 juni 2020, kl. 8.30 - 12.30 (14.30)

Examinator
Roger Johansson
Pedro Trancoso

Kontaktpersoner under tentamen:
Roger Johansson, epost (Canvas)
Pedro Trancoso, epost (Canvas)

Tillåtna hjälpmedel

Alla hjälpmedel är tillåtna så länge uppgifterna löses på individuell basis utan att konsultera någon annan än examinator under skrivningstiden.

Lösningsförslag
Anslås senast dagen efter tentamen via kursens hemsida.
Lösningsförslagen är endast vägledande för hur korrekt kod ska vara utformad och anvisar inte hur poängbedömning kommer att utföras.

Bedömning/Granskning
Tillfällen för granskning av bedömningar kommer att publiceras på respektive kurshemsida.

Allmänt
Svar kan avges på svenska eller engelska.

Tänk på att disponera din tid väl.

Börja med att läsa igenom alla uppgiftstexter.

Försök avge lösningsförslag på samtliga uppgifter.

Inlämning sker enligt anvisningar på Canvas. Observera att inlämningar efter tentamenstid + 10 min. (12.40) inte kommer att bedömas. (Betraktas som blank inlämning). Tentander som beviljats förlängd skrivtid lämnar in senast kl. 14.40.

Betygsättning för hemtentamen
Maximal poäng är 60 och tentamenspoäng ger betyg enligt: (EDA/DAT):
$30p \leq$ betyg 3 $< 40p \leq$ betyg 4 $< 50p \leq$ betyg 5.
respektive (DIT):
$30p \leq$ betyg G $< 45p \leq$ VG

För EDA482/EDA487/DIT151, lp4 vt2020

Tentamensbetyg ges av hemtentamen.

Som slutbetyg på kursen ges det högsta av betyg från hemtentamen respektive betygsbedömd laboration förutsatt att båda examinationsmoment är godkända (minst betyg 3).

För omtentamina:

Tentamensbetyg ges av hemtentamen.

Som slutbetyg på kursen ges betyg från hemtentamen under förutsättning att laborationskursen är godkänd.

Uppgift 1

Ett program, givet i C, ska översättas till ARMv6 assemblerspråk.

```
#define MAX 4

unsigned short student_year[MAX];
unsigned char  student_grade[MAX];
int c;

void init_data() {
  student_year[0]  = 2000;
  student_grade[0] = 4;
  student_year[1]  = 2000;
  student_grade[1] = 3;
  student_year[2]  = 1998;
  student_grade[2] = 5;
  student_year[3]  = 2002;
  student_grade[3] = 4;
}

int sum_grade( int year ) {
  int x = 0;
  for( int i=0; i < MAX; i++)
  {
    if( student_year[i] == year )
      x += student_grade[i];
  }
  return x;
}

int main( void ) {
  int a, b;
  init_data();
  a = sum_grade(2020);
  b = sum_grade(2000);
  c = a+b;
  // Det följer kod (inte visat här) som senare använder ‘c’
  // vi kan alltså inte “optimera bort” tilldelningen.
}
```

- Assemblerkoden ska kommenteras på sådant sätt att det tydligt framgår vilka delar av C-koden som översatts. Speciellt gäller att kodningen av uttrycksevaluering enkelt ska kunna följas med hjälp av kommentarerna. Varje enskild instruktion behöver dock inte kommenteras så länge kodningen sker enligt de principer vi använt i kursen.
- Användningen av register (registerallokering) ska beskrivas.

Bedömningskriterier:

- Lösningen kan ge maximalt 15p.
- En lösning som saknar, eller bara innehåller mycket bristfälliga, kommentarer kan ge maximalt 5 p.
- Kodningen ska följa de råd och anvisningar som getts under kursen.

Uppgift 2

Ett system för "reaktionstest" ska konstrueras.

Systemet består av en inmatningsenhet med 16 st. tangenter och en visningsenhet av ”sju-siffertyp”.

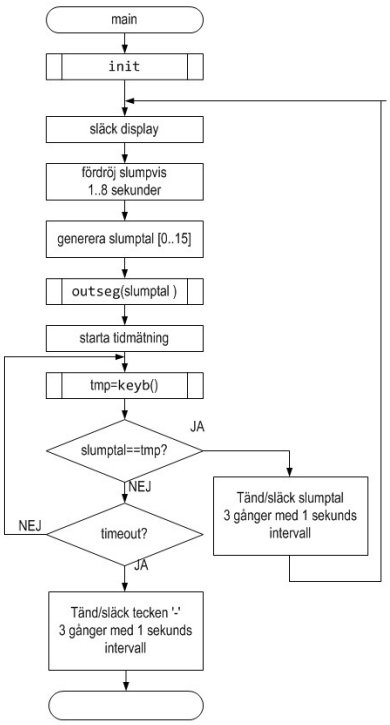
Efter en slumpvis fördröjning, 1 t.o.m 8 sekunder (i steg om sekunder) genereras ett slumptal 0..15. Detta slumptal ska nu visas på visningsenheten, därefter ska operatören så fort som möjligt trycka ned motsvarande tangent på tangentbordet.

Om reaktionstesten är 1 sekund eller mindre ska slumtalet tändas/släckas 3 gånger med 1 sekunds mellanrum, på visningsenheten.

Om reaktionstiden är större än 1 sekund ska reaktionstestet avslutas med att tecknet ’-’ tänds/släcks 3 gånger med 1 sekunds mellanrum.

Du får förutsätta och använda en funktion för slumptalsgenerering: `char random(void);` som genererar ett 7 bitars slumptal (0 till 127).

`init`, `outseg` och `keyb` ska implementeras som C-funktioner. I övrigt avgör du själv hur du vill dela upp applikationen.



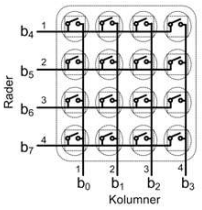
- Analysera systemets flödesdiagram. Föreslå och specificera C-funktioner och data och beskriv dessa funktioners gränssnitt (parametrar/returvärden). Beskrivningen ska vara kortfattad men tillräckligt detaljerad för att utgöra grund för en implementering.
- Implementera systemet "Reaktionstest" enligt din beskrivning, i programspråket C.

Bedömningskriterier:

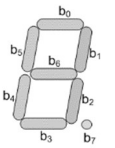
- SysTick ska användas för att skapa de realtidsfördröjningar som reaktionstestet kräver.
- Lösningen kan ge maximalt 15p.
- För full poäng krävs att dina lösningar är tydliga, fullständiga och att du använt lämpliga makrodefinitioner för GPIO-portarnas adresser.
- En lösning som saknar beskrivning/kommentarer, eller bara innehåller mycket bristfälliga, kommentarer kan ge maximalt 5 p.
- Kodningen ska följa de råd och exempel som getts under kursen.

Systembeskrivning
Laborationsdator MD407 och laborationsmoduler ”keypad” och ”sju.sifferindikator”

Laborationsmodul: ”keypad” ansluts till port D 8-15 enligt figuren till höger. På tangentbordet betecknar symbolen ’*’ slumptalet 14 (0xE) och symbolen ’#’ betecknar slumptalet 15 (0xF).



Laborationsmodul: ”sju-sifferindikator” ansluts till PD7-PD0.



Uppgift 3

Ett litet datainsamlingssystem ska konstrueras kring MD407. Systemet tar kontinuerligt emot och visar ett 16-bitars dataord (D0-D15) men det finns bara en 8-bitars port ledig så mottagningen sker med "multiplex"-teknik enlig följande:

Dataordet överförs via ett 8-bitars register med hjälp av två multiplexsignaler S1 och S2 (se följande tabell). Ett giltigt dataord skrivs, på hexadecimal form, till två visningsenheter för visning av D0-D7 och uppdateringen av visningsenheterna ska ske enligt följande funktionstabell:

S1	S2	Funktion PE7-PE15
0	0	Ingen giltig data, släck visningsenhet D0-D7
0	1	Ingen giltig data, släck visningsenhet D8-D15
1	0	Giltig data, läs från inport, visa som D0-D7
1	1	Giltig data, läs från inport, visa som D8-D15

- Beskriv, med ord, ett program med de delar som realiserar (dvs. bygger upp) datainsamlingssystemet. Dvs. föreslå och specificera C-funktioner och data och beskriv dessa funktioners gränssnitt (parametrar/returvärden). Beskrivningen ska vara kortfattad men tillräckligt detaljerad för att utgöra grund för en implementering (förverkligande).
- Implementera datainsamlingssystemet enligt din beskrivning, i programspråket C.

Bedömningskriterier:

- Lösningen kan ge maximalt 15p. För maximal poäng ska en av multiplexsignalerna användas som *extern avbrottsignal* med EXTI/NVIC.
- För full poäng krävs att dina lösningar är tydliga, fullständiga och att du använt lämpliga makrodefinitioner för GPIO-portarnas adresser.
- En lösning som saknar beskrivning/kommentarer, eller bara innehåller mycket bristfälliga, kommentarer kan ge maximalt 5 p.
- Kodningen ska följa de råd och exempel som getts under kursen.

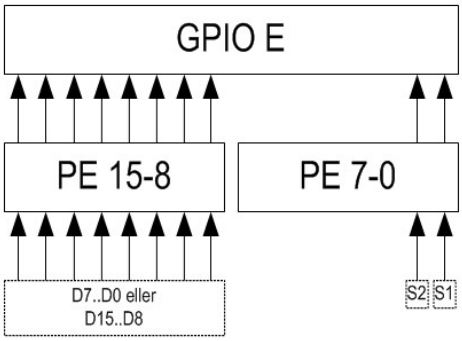
Systembeskrivning

Data samlas in från GPIO port E.

PE8-PE15: 8 bitars dataord (D0-D7 *eller* D8-D15)

Signalen S1 ansluts till PE0.

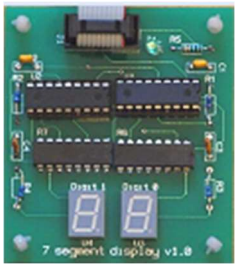
Signalen S2 ansluts till PE1.



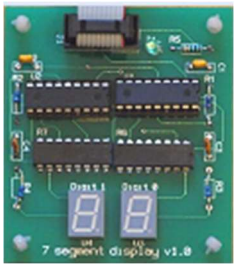
Två visningsenheter är anslutna till GPIO port D.

PD7-PD0 används för databitarna D0-D7.

PD8-PD15 används för databitarna D8-D15.



Visningsenhet PD15-PD8



Visningsenhet PD7-PD0

Uppgift 4

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. Registren i robotens gränssnitt beskrivs av följande:

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																	status								ctrl							
																				IP		ER			IE	IA			EN			RS
																				r		r			rw	w			rw			rw
4	dataX																dataY															
	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
8	posX																posY															
	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Anm: **r**: biten är enbart läsbar, **w**: biten är enbart skrivbar, **rw**: biten är både läsbar och skrivbar. Försök att skriva en bit som är enbart läsbar har ingen effekt. Försök att läsa en bit som är enbart skrivbar, returnerar alltid värdet 0.

IP: *In Position*. Biten sätts till 1 då innehållen i data och positionsregistren överensstämmer. Om avbrott är aktiverat (IE) genereras detta med nummer 4 i vektortabellen.

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 genereras även detta.

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 eller då ett fel uppstår.

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.

För att utföra en rörelse hos 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 armen deaktiveras.

Din uppgift är att konstruera de grundläggande funktioner som krävs för att styra robotarmen. Utöver detta ska du också visa ett huvudprogram som demonstrerar funktionerna.

- Huvudprogrammet ska börja med att initiera roboten genom att placera armen i position 0,0. Följande koordinater (x,y) ska ska därefter besökas, i tur och ordning, under demonstrationen: (1,2), (12,14), (25,27), (36,39). Din lösning ska dock tillåta en lista med godtyckliga koordinater.
- Styrregistret är placerat på adress 0x100000 i datorns minne.
- Efter en avslutad förflyttning måste robotarmen vila i innan nästa förflyttning kan ske. Vilan ska vara lika många millisekunder som skillnaden i x-led under den föregående förflyttningens. Du får förutsätta att en funktion `void delayMS(unsigned short ms)`, som utför en blockerande fördröjning (`ms` millisekunder) finns tillgänglig och kan användas.
- Beskriv, med ord, ett program de delar (funktioner och data) som realiserar robotens styrsystem. Dvs. specificera C-funktioner, beskriv dessa funktioners gränssnitt (parametrar/returvärden) och eventuella globala data. Beskrivningen ska vara kortfattad men tillräckligt detaljerad för att utgöra grund för en implementering.
- Implementera robotens styrsystem enligt beskrivningen, i programspråket C.

Bedömningskriterier:

- Lösningen kan ge maximalt 15p.
- En lösning som saknar beskrivning/kommentarer, eller bara innehåller mycket bristfälliga, kommentarer kan ge maximalt 5 p.
- Kodningen ska följa de råd och exempel som getts under kursen.

Uppgift 1:

En lösning som denna är tillräcklig för full poäng. Dvs. kommentarerna är tillräckliga för att följa i C-koden. Mindre fel i assemblersyntax kan resultera i mindre poängavdrag. Dock dras inte för "samma fel" flera gånger. Det är viktigt att ***operandstorlekar*** *hanteras rätt (med rätt load/store instruktion)*

Kompilatorkoventioner måste följas, dvs. funktioner ska vara korrekt utformade med prolog/epilog där så krävs.

Val av korrekt assemblerinstruktion är viktig, speciellt de olika BRANCH-instruktionerna. Poängavdrag om fel instruktion används.

```
; define global variables
student_year:
        .SPACE 8
        .ALIGN
student_grade:
        .SPACE 4
        .ALIGN
c:
        .SPACE 4
        .ALIGN

; implementation of the init_data function
init_data:
        ; student_year[0]  = 2000;
        LDR  R0, =student_year
        LDR  R2, =2000
        STRH R2, [R0]
        ; student_grade[0] = 4;
        LDR  R1, =student_grade
        LDR  R2, =4
        STRB R2, [R1]

        ; student_year[1]  = 2000;
        ADD  R0, R0, #2
        LDR  R2, =2000
        STRH R2, [R0]
        ; student_grade[1] = 3;
        ADD  R1, R1, #1
        LDR  R2, =3
        STRB R2, [R1]

        ; student_year[2]  = 1998;
        ADD  R0, R0, #2
        LDR  R2, =1998
        STRH R2, [R0]
        ; student_grade[2] = 5;
        ADD  R1, R1, #1
        LDR  R2, =5
        STRB R2, [R1]

        ; student_year[3]  = 2002;
        ADD  R0, R0, #2
        LDR  R2, =2002
        STRH R2, [R0]
        ; student_grade[3] = 4;
        ADD  R1, R1, #1
        LDR  R2, =4
        STRB R2, [R1]
```

```
; implementation of the sum_grade function
sum_grade:
        ; year is in R0
        PUSH {R4, R5, LR}
        LDR  R1, =0 ; use R1 for x
        LDR  R2, =0 ; use R2 for i
; use R3 for student_year access
        LDR  R3, =student_year
; use R4 for student_grade access
        LDR  R4, =student_grade
loop:
        CMP  R2, #4
        BGE  loop_exit
        LSL  R5, R2, #2
        LDRH R5, [R3,R5]
        CMP  R5, R0
        BNE  if_exit
        LDRB R5, [R4,R2]
        ADD  R1, R1, R5
if_exit:
        ADD  R2, R2, #1
        B    loop
loop_exit:
        MOV  R0, R1
        POP  {R4, R5, PC}

; implementation of the main function
main:
        ; R4 used for a and R5 for b
        ; init_data();
        BL   init_data
        ; a = sum_grade(2020);
        LDR  R0, =2020
        BL   sum_grade
        MOV  R4, R0
        ; b = sum_grade(2000);
        LDR  R0, =2000
        BL   sum_grade
        MOV  R5, R0
        ; c = a+b;
        ADD  R4, R4, R5
        LDR  R0, =c
        STR  R4, [R0]
```

Uppgift 2:

Huvuddelen av koden i denna uppgift kan kopieras från laboration 2, har man inte insett detta kan det bli svårt att konstruera lösning under den tillgängliga tiden. Har man däremot förstått detta tyder det på tillräckliga grundkunskaper, kanske har man till och med kommenterat detta vilket i så fall är bra.
Viktigt att kolla:

- *outseg ska kunna skriva ut '-' vilket ger ett litet tillägg till laborationen, annars identisk*
- *Hantering av SysTick, man ska visa att man förstått skillnaden mellan blockerande/icke-blockerande fördröjning.*
- *Implementeringen ska följa flödesplan.*

Beträffande kommentarer och beskrivningar:
Välskriven kod kan vara "självdokumenterande" beroende på val av programkonstruktion, variabelnamn etc. Vi kräver dock alltid att funktionsgränssnitten och den icke-blockerande funktionen för "timeout" ägnas uppmärksamhet.

I lösningsförslaget har kommentarer utelämnats för att inte ge missledande information om rättningskriterier.

```
#define PORT_D_BASE      0x40020C00
#define GPIO_D_MODER     ((volatile unsigned int *)  (PORT_D_BASE))
#define GPIO_D_OTYPER    ((volatile unsigned short *)(PORT_D_BASE+0x4))
#define GPIO_D_OSPEEDR   ((volatile unsigned int *)  (PORT_D_BASE+0x8))
#define GPIO_D_PUPDR     ((volatile unsigned int *)  (PORT_D_BASE+0xC))
#define GPIO_D_IDRLOW    ((volatile unsigned char *)(PORT_D_BASE+0x10))
#define GPIO_D_IDRHIGH   ((volatile unsigned char *)(PORT_D_BASE+0x11))
#define GPIO_D_ODRLOW    ((volatile unsigned char *) (PORT_D_BASE+0x14))
#define GPIO_D_ODRHIGH   ((volatile unsigned char *) (PORT_D_BASE+0x15))

#define STK_CTRL         ((volatile unsigned int *)(0xE000E010))
#define STK_LOAD         ((volatile unsigned int *)(0xE000E014))
#define STK_VAL          ((volatile unsigned int *)(0xE000E018))
#define SYSTICK_IRQVEC   ((volatile unsigned int *)0x2001C03C)

void init( void )
{
  *GPIO_D_MODER = 0x55005555;
  *GPIO_D_PUPDR = 0x00AA0000;
  *GPIO_D_OTYPER= 0x00000000;
}
static void kbdActivate( unsigned int row )
{
  switch( row )
  {
    case 1: *GPIO_D_ODRHIGH = 0x10;  break;
    case 2: *GPIO_D_ODRHIGH = 0x20;  break;
    case 3: *GPIO_D_ODRHIGH = 0x40;  break;
    case 4: *GPIO_D_ODRHIGH = 0x80;  break;
    case 0: *GPIO_D_ODRHIGH = 0x00;  break;
    }
}
static int kbdGetCol ( void )
{
    unsigned short c;
    c = *GPIO_D_IDRHIGH;
    if ( c & 0x8 )      return 4;
    if ( c & 0x4 )      return 3;
    if ( c & 0x2 )      return 2;
    if ( c & 0x1 )      return 1;
    return 0;
}
unsigned char keyb(void)
{
    unsigned char
key[]={1,2,3,0xA,4,5,6,0xB,7,8,9,0xC,0xE,0,0xF,0xD}
;
    int row, col;
    for (row=1; row <=4 ; row++ ) {
        kbdActivate( row );
        if( (col = kbdGetCol () ) )
        {
            return key [4*(row-1)+(col-1) ];
        }
    }
    *GPIO_D_ODRHIGH = 0;
    return 0xFF;
}
void outseg( char c )
{
static char segCode[]={
    0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
    0x7F,0x67,0x77,0x7C,0x39,0x5E,0x79,0x71, 0x40
};


    if( c > 16){
        *GPIO_D_ODRLOW = 0;
        return;
    }
    *GPIO_D_ODRLOW = segCode[c];
}


void blocked_delay_10ms( void )
{
    /* SystemCoreClock = 168000000 */
    *STK_CTRL = 0;
//  *STK_LOAD = ( 1680000-1 ); /* 10 ms */
    *STK_LOAD = ( 168-1 );  /* Simulator value */
    *STK_VAL = 0;
    *STK_CTRL = 5;
    while( (*STK_CTRL & 0x10000)==0);
}

void blocked_delay( unsigned int ms )
{
    while( ms/10 ){
        blocked_delay_10ms();
        ms -=10;
    }
}
```

```
static int timeout;
voidsystick_irq_handler( void );

void start_timeout_period( void )
{
    /* SystemCoreClock = 168000000 */
    *SYSTICK_IRQVEC = systick_irq_handler;
    *STK_CTRL = 0;
//  *STK_LOAD = ( 1680000-1 );
    *STK_LOAD = ( 1680-1 ); /* Simulator value */
    *STK_VAL = 0;
    *STK_CTRL = 7;
}




voidsystick_irq_handler( void )
{
    if( timeout ){
        timeout--;
        start_timeout_period( );
    }
}




void start_timeout( void )
{
    timeout = 500; /* 100 * 10 ms = 1 sec */
    start_timeout_period( );
}
```

```
void display3sec( char c )
{
    for( int i = 0; i<3;i++)
    {
        outseg(0xFF);
        blocked_delay( 500 );
        outseg(c);
        blocked_delay( 500 );
    }

}

void main(void)
{
    char tmp;
    init();
    while(1)
    {
        unsigned int t = random() >> 4;
        t++;
        blocked_delay( t * 1000 );
        t = random() >> 3;
        outseg( (char) t );
        start_timeout();
        do{
            tmp = keyb();
            if( t == tmp )
                break;
        }while (timeout );
        if( timeout )
        {
            display3sec( tmp );
        }else{
            display3sec( 16 );
            break;
        }
    }
}
```

Uppgift 3:

Här ska du ha identifierat likheterna med laboration 4. Såväl initieringar som avbrottshantering vara korrekt utformad. Den här typen av kod brukar också vara svårare att ”följa” då man läser den. Därför måste kommentarerna vara tydliga. I lösningsförslaget har dock kommentarer utelämnats för att inte ge missledande information om rättningskriterier.

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

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

#define PORT_E_BASE         ( ( volatile unsigned long *) 0x40021000)
#define PORT_E_INLOW    ( ( volatile char *) 0x40021010)
#define PORT_E_INHIGH       ( ( volatile char *) 0x40021011)
#define PORT_D_BASE         ( ( volatile unsigned long *) 0x40020C00)
#define PORT_D_OUTLOW       ( ( volatile char *) 0x40020C14)
#define PORT_D_OUTHIGH      ( ( volatile char *) 0x40020C15)

#define NVIC_EXTI1_IRQ_BPOS  (1<<7)
#define EXTI1_IRQ_BPOS  (1<<1)
#define EXTI1_IRQVEC    0x2001C05C

void irq_handler ( void );

void app_init(void)
{

    * PORT_D_BASE = 0x55555555;
    * PORT_E_BASE = 0;

    *SYSCFG_EXTICR1 |= 0x0040;
    *EXTI_IMR |= EXTI1_IRQ_BPOS;
    *EXTI_RTSR |= EXTI1_IRQ_BPOS;
    *EXTI_FTSR |= EXTI1_IRQ_BPOS;
    *((void (**)(void) ) EXTI1_IRQVEC ) = irq_handler;
    *NVIC_ISER0 |= NVIC_EXTI1_IRQ_BPOS;
}



void irq_handler ( void )
{
    if( *EXTI_PR & EXTI1_IRQ_BPOS )
    {
        if( *PORT_E_INLOW & 1 )
        {
            if( *PORT_E_INLOW & 2 )
                *PORT_D_OUTHIGH = *PORT_E_INHIGH;
            else
                *PORT_D_OUTLOW = *PORT_E_INHIGH;
        }else{
            if( *PORT_E_INLOW & 2 )
                *PORT_D_OUTHIGH = 0xFF;
            else
                *PORT_D_OUTLOW = 0xFF;
        }
        *EXTI_PR |= EXTI1_IRQ_BPOS;
    }
}


void main(void)
{
    app_init();
    while(1);
}
```

Uppgift 4:

Detta är i allt väsentligt en kopia av uppgifter som getts i tidigare salstentamina (2018).
Vi kräver därför här (utöver korrekt kod) beskrivande text som beskriver de ingående funktionerna.

Bland klurigheterna kan vara att sätta upp testfallet i ”mainfunktionen” dvs. datastruktur i form av en länkad lista som medger en godtycklig koordinatföljd. Eftersom skillnaden mellan koordinaterna kan vara mindre än 0 och delayMS tar en unsigned short, måste testprogrammet respektera detta genom att forma absolutbeloppet av koordinaternas differenser.

Här visas ett lösningsförslag där porten beskrivs med bitfält. Det är inte nödvändigt, alternativ form med explicit bithantering kan också ge full poäng.

I lösningsförslaget har kommentarer utelämnats för att inte ge missledande information om rättningskriterier.

```
typedef volatile struct sROBOT{
    union{
        char ctrl;
        char RS:1,:2, EN:1,:2,IA:1,IE:1;
     };
     union{
        char status;
        char :2, ER:1, :1, IP:1;
    };
    unsigned short reserved;
    unsigned short dataY;
    unsigned short dataX;
    unsigned short posY;
    unsigned short posX;
}ROBOT;

typedef struct point
{
    unsigned short x;
    unsigned short y;
    struct point * next;
} POINT;

/* Alternativ till bitfält */
#define ENBIT   (1<<3)
#define RSBIT   (1<<0)

void move(ROBOT *p, int x, int y )
{
  p->dataX = x;
  p->dataY = y;
  p->EN = 1;
  p->ctrl |= ENBIT; /* alternativ */

  while( ( p->posX != x )||(p->posY != y ) ) ;
  p->EN = 0;
  p->ctrl &= ~ENBIT; /* alternativ */
}
```

```
void init(ROBOT *p)
{
  p->RS = 1;
  p->ctrl |= RSBIT; /* alternativ */
  p->ctrl = 0;
  move( p,0,0 );
}



POINT p4 = {36,39,0};
POINT p3 = {25,27,&p4};
POINT p2 = {12,14,&p3};
POINT p1 = {1,2,&p2};

extern void delayMS( unsigned short ms);



void main(void)
{
    ROBOT *p = (ROBOT *) 0x100000;
    POINT *pt = &p1;
    unsigned short x,y;
    init( p );
    x = y = 0;
    while( pt != 0 )
    {
        move( p, pt->x, pt->y );
        if( pt->x > x) delayMS( pt->x - x );
        else if ( pt->x < x) delayMS( x - pt->x );
        x = pt->x;
        y = pt->y;
    }
}
```