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 21 augusti 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. 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 ≤ betyg 3 < 40p ≤ betyg 4 < 50p ≤ betyg 5.
respektive (DIT):
30p≤ betyg G < 45p ≤ VG

För EDA482/EDA487/DIT151, lp4 vt2020

Tentamensbetyg ges av hemtentamen.

Som slutbetyg på kursen ges en sammanfattande bedömning baserad på betyg från hemtentamen respektive betygsbedömd laboration förutsatt att båda dessa moment 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.

Bedömningskriterier vid hemtentamen:

En lösning bedöms utifrån tre olika aspekter där varje aspekt kan ge 0-5 poäng. En uppgift kan därför ge högst 15 poäng och lägst 0 poäng. De olika bedömningsaspekterna är:

- Kodkvalité och kodens fullständighet
- Dokumentation och kommentarers kvalité och fullständighet
- Relevans

Följande uppställningar ger kortfattade förtydliganden om de generella grunderna för respektive bedömning. Eftersom uppgifter kan ha skilda karaktärer och lösningar anta mycket olika former kan dock ytterligare (speciella) bedömningsgrunder komma att tillämpas.

Kodkvalité och kodens fullständighet

- Är koden syntaktiskt korrekt och följs de anvisningar och rekommendationer som kursen lärt ut?
- Följs eventuella kodkonventioner?
- Finns alla nödvändiga komponenter med, dvs. variabeldeklarationer och funktioner(subrutiner).

Bedömningsskala:

5. Högsta kvalitet och fullständig
4. Enstaka kvalitetsbrister men fullständig
3. Enstaka kvalitetsbrister, inte helt fullständig
2. Flera kvalitetsbrister och/eller ofullständighet
1. Stora kvalitetsbrister och/eller menlig ofullständighet
0. Kod kan inte bedömas eller saknas helt

Dokumentation och kommentarers kvalité och fullständighet

Dokumentation och kommentarer ska vara utformade på ett sätt som ger en uttömmande förklaring till hur koden är avsedd att fungera. Speciellt kontrolleras följande:

- Finns aktuella gränssnitt dokumenterade med förklaringar av parametrar och returvärden? För assemblerkod innebär detta också beskrivning av hur parametrar och returvärden överförs.
- Om uppgiften är att skriva C-kod, hur väl kopplas denna till de algoritmiska stegen i lösningen med hjälp av kommentarer? Dvs. om algoritmer är givna i uppgiften så ska dessa följas. Om konstruktion av algoritm ingår i lösningen ska denna först beskrivas.
- Om uppgiften är att skriva assemblerkod ska det finnas en tydlig koppling mellan de sekvenser av assemblerkoden och den C-kod (funktioner/variabeldeklarationer) som är assemblerkodens upphov.
- Det är tillåtet (inget krav) att använda kursens verktyg för att generera assemblerkod men tänk på att denna kod måste bearbetas och kommenenteras för att uppfylla kraven i bedömningskriterierna.

Tänk också på att en väl utförd modularisering av programmet tillsammans med väl valda funktionsnamn och variabelnamn i sig bidrar till "självdokumenterande kod" och kan minska behovet av explicit kommentering.

Bedömningsskala:

5. Högsta kvalitet och fullständig
4. Enstaka kvalitetsbrister men fullständig
3. Enstaka kvalitetsbrister, inte helt fullständig
2. Flera kvalitetsbrister och/eller ofullständighet
1. Stora kvalitetsbrister och/eller menlig ofullständighet
0. Dokumentation kan inte bedömas eller saknas helt

Relevans

Relevans baseras på kod *och* dokumentation och ska ge en sammanfattande bedömning av:

- Hur väl har uppgiften uppfattats och hur väl visar lösningen på en riktig förståelse?
- Finns speciella anvisningar för hur uppgiften ska lösas och hur har dessa i så fall efterlevts?

Relevans kan sällan bedömas fullt ut om kommentarer/dokumentation har stora brister eller saknas helt. I sådana fall når sällan denna bedömning över 1.

Bedömningsskala:

5. Högsta relevans
4. God relevans
3. Nöjaktig (godkänd) relevans
2. Bristfällig relevans
1. Mycket bristfällig relevans
0. Ej relevant/ej avgivet svar

Uppgift 1

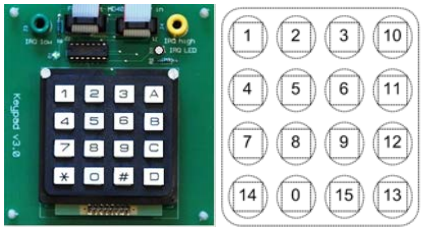
Ett program, givet i C, ska översättas till ARMv6 assemblerspråk.

```
#define port8ptrA ( ( volatile char *) 0x100000 )
#define port8ptrB ( ( volatile char *) 0x100001 )

void saveRelation( int );
int comparePorts( void );

/* Read and compare 100 values from ports A and B
 * Determine the percentage of values A less than or equal to B
 * Place the result in the global variable 'percentage'
 */
char result[100];
int   percentage;
void main(void)
{
   percentage = 0;
   for( int i=0; i<100;i++)
      saveRelation( comparePorts() );
   while( i < 100 ){
      if( result[i])
         percentage++;
   }
}

void saveRelation( int relation )
{
   static int items = 0;
   result[items++] = relation;
}

int comparePorts( void )
{
   if (*port8ptrA <= *port8ptrB)
      return 1;
   return 0;
}
```

- 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.
- Lokala variabler ska allokeras till register och användningen av register ska beskrivas.

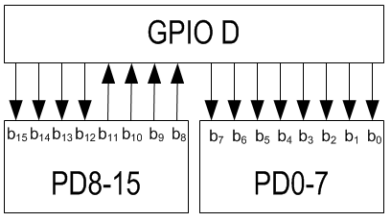
Uppgift 2

En applikation för inmatning från en keypad (figur 1), konvertering av inmatade värden och slutligen utskrift till en hexadecimal display (figur 2) ska konstrueras. Enheterna ansluts till *MD407* port D enligt figur 3.



Figur 1: Keypad och tangentkoder

Figur 2



Figur 3

Applikationens inläsningsrutin ska detektera en nedtryckt tangent och bestämma dess tangentkod och returnera denna till huvudprogrammet. För att inte samma nedtryckning ska resultera i flera likadana returvärden ska inläsningsfunktionen utformas enligt:

```
char keycode(void):
```

1) vänta tills en tangent trycks ned
2) bestäm tangentkoden
3) vänta tills tangenten släpps upp
4) returnera tangentkoden

Två parametrar ska läsas in: `value`, som läses in som två st. NBCD-siffror, endast tangentkoderna 0..9 är alltså giltiga här.

```
char get_valid_value( void ):
```

1) läs tangentkoder för två NBCD-siffror (andra tangentkoder ignoreras) från keypad:
2) tolka den första siffran som mest signifikanta fyra bitarna
3) tolka den andra siffran som minst signifikanta fyra bitarna
4) returnera detta värde

och `method`, endast tangentkoderna 10,11,12 och 13 är giltiga här

```
char get_valid_method( void ):
```

1) läs en tangentkod A,B,C eller D (andra tangentkoder ignoreras), från keypad
2) returnera giltig tangentkod

Efter inläsning av `value` och `method` används dessa för att bestämma det värde som ska skrivas till hexadecimal display med följande konverteringsfunktion:

```
char  valconvert(char method, char value):
```

Returvärdet bestäms av parametern `value` och parametern `method` enligt:

`method =0xA:` returnera `value`.
`method =0xB:` returnera *bitvis komplementet* av `value`.
`method =0xC:` returnera `value` skiftat ett steg till vänster.
`method =0xD:` returnera `value` skiftat ett steg till höger.

Utöver de beskrivna funktionerna består applikationen dessutom av:

```
void init( void ):
```

Konfigurera portpinnar, förbered keypad och hexadecimal display för användning av huvudprogramslingan.

```
int main(void):
```

1) initiera systemet

repetera:

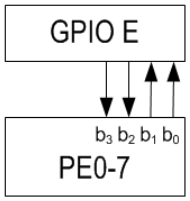
2) läs in `value` från keypad
3) läs in `method` från keypad
4) skriv konverterat värde till hexadecimal display

Implementera applikationen i programspråket C.

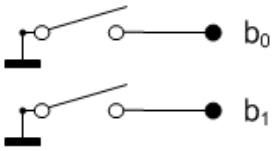
Uppgift 3

Ett system för för att generera två sammanhängande fyrkantsvågor ska konstrueras. I figur 3 beskrivs de önskade utsignalerna. Observera att dessa inte är helt symmetriska, en liten "dödtid" (*d*) förhindrar att omslagen mellan low och high sker exakt samtidigt. Två insignaler används för att specificera pulslängder och periodtid.

Strömställare och utgångarna för fyrkantsvågorna ansluts till laborationsdator *MD407* enligt figur 1. Två insignaler via PE0 och PE1 (figur 2) används för att ange pulslängderna som bestäms av *w* och *d* hos fyrkantsvågorna, se även figur 3 och tabellen nedan.



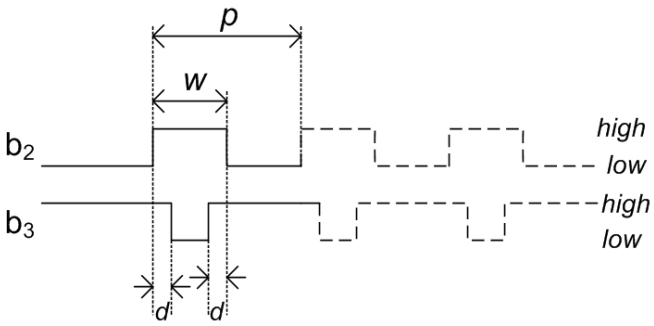
Figur 1



Figur 2

Systemets programvara skapar de båda pulserna som utsignaler på PE2 och PE3 med hjälp av realtidsfördröjningar.

Den längre pulsens varaktighet betecknas *w* och periodtiden för samma puls betecknas *p*. Fördröjningarna mellan pulsernas omslag betecknas d.

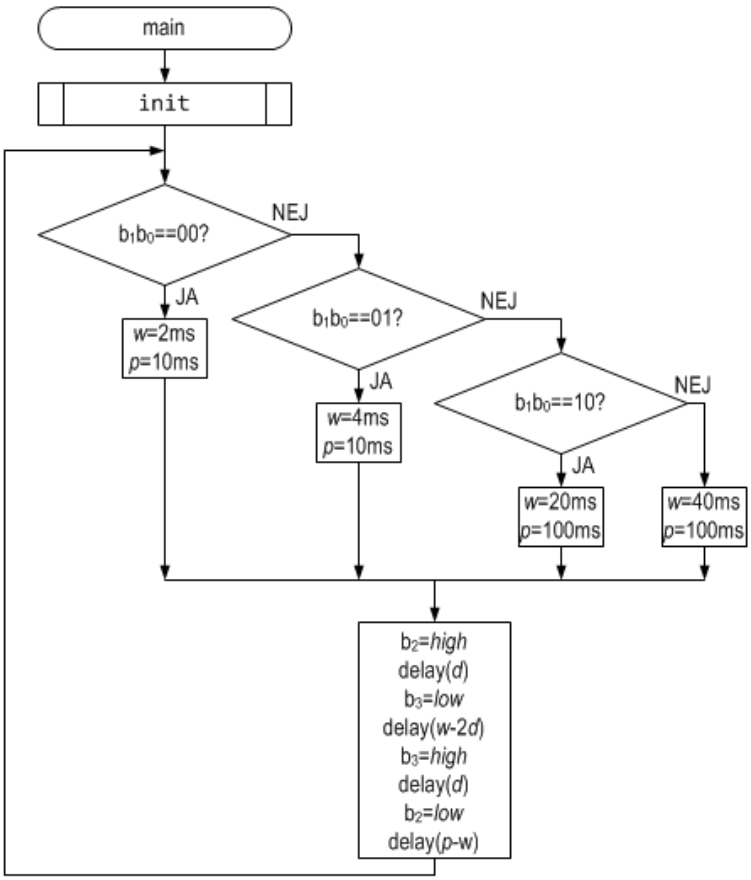


Figur 3

Sambanden mellan insignalerna och fyrkantsvågorna anges i följande tabell:

b_1	b_0	*w*	*d*	*p*
0	0	2ms	100µs	10ms
0	1	4 ms	100µs	10ms
1	0	20ms	100µs	100ms
1	1	40ms	100µs	100ms

Flödesdiagrammet till höger kan användas som utgångspunkt för implementeringen, dessutom gäller följande:



- Systemets programvara ska implementeras i programspråket C. Du får anta att alla exekveringstider är så korta att de kan anses försumbara i förhållande till realtidsfördröjningarna.
- Endast portpinarna PE0-PE3 få påverkas av programmet eftersom de övriga antas vara upptagna för annan användning.
- Insignalerna från b_0 och b_1 är inte definierade då respektive brytare är öppen. Dessa måste därför programmeras med en lämplig intern förspänning (*pull-up* eller *pull-down*).
- Utsignalerna b_2 och b_3 används för att driva vardera en last och ska därför vara *push*/*pull*.
- Realtidsfördröjningar åstadkoms via systemets räknarkrets SysTick.
- Funktionen `init` ska försätta systemet i lämpligt tillstånd innan huvudprogrammets slinga startas, dvs. initiera portar och sätta initiala utsignaler.

Uppgift 4

I ett fordon finns en farthållare "Cruise control" som direkt kan påverka motorns styrsystem och få fordonet att hålla en konstant hastighet. Föraren kan påverka farthållningen med hjälp av följande omkopplarfunktioner

Cruise ON:	Aktivera farthållare och lås farthållarhastigheten till fordonets aktuella hastighet
Cruise OFF:	Deaktivera farthållare
Increase:	Inställd hastighet ökas med 1,6 km/tim
Decrease:	Inställd hastighet minskas med 1,6 km/tim

Farthållaren är uppbyggd kring en ST407 enchipsdator. Gränssnittet till farthållaren har följande utseende där samtliga bitar är både läs- och skrivbara:

Parallel port Cruise Control (PortCC)										
Address	7	6	5	4	3	2	1	0	Mnemonic	Namn
0xA00					INC	DEC	LOCK	AUTO	CTRL	Control register
0xA01	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	SM	State machine register
0xA02	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	VEL	Velocity register
0xA03					IREQ	DREQ	CRON	CROFF	IRQ	Interrupt request register

CTRL	Styrregister för modulen, kontrollerar farthållarens funktion. I styrregistret är biten AUTO nivåstyrd, medan övriga bitar är flankstyrda.	
	AUTO	Biten används tillsammans med de övriga bitarna i styrregistret samt med registret SM för att aktivera farthållaren. Genom att skriva 0 till denna bit deaktiveras farthållaren.
	LOCK	då farthållaren är aktiv kommer en positiv transition hos denna bit (0→1) att låsa hastigheten, dvs. fordonets aktuella hastighet, fås automatiskt från motorstyrenheten, kopieras till farthållarens hastighetstregister VEL.
	DEC	då farthållaren är aktiv kommer en positiv transition hos denna bit (0→1) att minska innehållet i hastighetsregistret med 1, dvs. minska den inställda hastigheten med 1,6 km/tim
	INC	då farthållaren är aktiv kommer en positiv transition hos denna bit (0→1) att öka innehållet i hastighetsregistret med 1, dvs. öka den inställda hastigheten med 1,6 km/tim
SM	Registret påverkar den interna tillståndsmaskinen som används för att aktivera farthållaren	
VEL	Hastigheten anges i steg om 1,6 km/tim. Farthållarens minimala hastighet är 1,6 km/tim då VEL är 0x00	
IRQ	Statusbitar som aktiveras av farthållarens omkopplarfunktioner. Dessa bitar är samtidigt kvittensbitar för de olika funktionerna	
	CROFF	Biten är kopplad till port E bit 5, EXTI5 och sätts till 1 då Cruise OFF aktiveras, biten nollställs då programmet skriver 1 till denna bit
	CRON	Biten är kopplad till port E bit 6, EXTI6 och sätts till 1 då Cruise On aktiveras, biten nollställs då programmet skriver 1 till denna bit
	DREQ	Biten är kopplad till port E bit 7, EXTI7 och sätts till 1 då Decrease aktiveras, biten nollställs då programmet skriver 1 till denna bit
	IREQ	Biten är kopplad till port E bit 8, EXTI8 och sätts till 1 då Increase aktiveras, biten nollställs då programmet skriver 1 till denna bit

Via styrregistret `CTRL` kontrolleras farthållarens funktion. För att inte farthållaren enkelt ska kunna aktiveras ofrivilligt, har den försetts med en tillståndsmaskin, som används vid aktiveringen av farthållaren. För att aktivera farthållaren krävs att följande algoritm utförs:

```
0→AUTO
0x55→SM
0xAA→SM
0x55→SM
1→AUTO
```

- Farthållarens port ska representeras med en `struct`.
- Farthållarens program ska utformas i två delar, en funktion `init` som initierar systemet och en funktion `irq_handler` som hanterar alla avbrott.
- Grundläggande felkontroller måste utföras av programmet. `INC`, `DEC` eller `LOCK` får inte aktiveras om farthållaren inte är aktiv. Farthållarens hastighet måste också kontrolleras innan `INC` eller `DEC` utförs så att inte min- eller max-hastigheter överskrids.

Lösningsförslag:

En lösning tas fram exempelvis genom att kompilera C-koden till assembler med GCC/ARM-kompilatorn. Den resulterande koden är dock väldigt svårläst och måste modifieras för hand. Kravet att lokala variabler ska registerallokeras leder till att åtskilliga load/store försvinner.
En lösning som bara innehåller "dump" av kompilerad kod bedöms 2/0/1.

Uppgift 1:

```
@ char result[100];
result: .SPACE 100
.GLOBL result
@ int percentage
percentage: .SPACE 4
.GLOBL percentage

@ void main(void)
@ {
main: PUSH {R4,LR}
@ registers: R4 assigned to i
@   int i;
@   percentage = 0;
 LDR   R3,=percentage
 MOV   R2, #0
 STR   R2,[R3,#0]
@   for( i=0; i<100;i++)
 MOV   R4,#0
 B     .L1
@       saveRelation( comparePorts() );
.L0:
 BL    comparePorts
 BL    saveRelation
 ADD   R4,#1
.L1:
 CMP   R4,#99
 BLE   .L0

@   i = 0;
 MOV R4,#0
@   while( i < 100 ){
 B  .L3
.L2:
@       if( result[i])
 LDR   R2,=result
 ADD   R3,R2,R4
 LDRB  R3,[R3]
 CMP   R3,#0
 BEQ   .L3
@       percentage++;
 LDR   R3,=percentage
 LDR   R2,[R3]
 ADD   R2,#1
 STR   R2,[R3]
.L3:
 CMP   44,#99
 BLE   .L2
 POP   {R4,LR}
```

```
@   static int items = 0;
items: .WORD 0
@ void saveRelation( int relation )
@{
saveRelation:
@   result[items++] = relation;
 LDR   R3,items
 ADD   R1,R3,#1
 LDR   R2,=items
 STR   R1,[R2]
 LDR   R2,=result
 STRB  R0,[R2,R3]
 BX    LR


@int comparePorts( void )
@{
comparePorts:
@   if (*port8ptrA <= *port8ptrB)
 LDR   R3,=0x100000
 LDRB  R2,[R3]
 LDR   R3,=0x100001
 LDRB  R3,[R3]
 CMP   R2,R3
 BHI   .L1
@       return 1;
 MOV   R0,#1
 B     .L2
@   return 0;
 MOV   R0,#0
@}
.L2:
 BX    LR
```

Uppgift 2:

Definitioner och funktioner i vänstra kolumnen är kopierade direkt från laboration 2

```
/* Port D */
#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))
```

```
void init( void )
{
    *GPIO_D_MODER = 0x55005555;
    *GPIO_D_PUPDR = 0x00AA0000;
    *GPIO_D_OTYPER= 0x00000000;
    *GPIO_D_ODRLOW = 0;
    *GPIO_D_ODRHIGH = 0;
}

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;
    }
}

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;
}


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;
}
```

```
char keycode( void )
{
    char c;
    while( (c = keyb() )==0xFF);
    while( keyb() != 0xFF);
    return c;
}

char bitconvert( char method, char value )
{
    char rv;
    switch( method )
    {
        case 0xA: rv = value; break;
        case 0xB: rv = ~value ; break;
        case 0xC: rv = ( value << 1 ); break;
        case 0xD: rv = ( value >> 1 ); break;
        default: rv = 0;
    }
    return rv;
}


char get_valid_value( void )
{
    char c1, c2;
    do{
        c1 = keycode();
    }while( c1 >= 10);
    do{
        c2 = keycode();
    }while( c2 >= 10);

    return (c1 << 4) | c2;
}


char get_valid_method( void )
{
    char c;
    do{
        c = keycode();
    }while( (c < 10) || (c==14) || (c==15) );
    return c;
}

void main( void )
{
    char value, method;
    init();
    while( 1 )
    {
        value = get_valid_value();
        method = get_valid_method();
        *GPIO_D_ODRLOW = bitconvert(method, value);
    }
}
```

Uppgift 3:

```
#define PORT_E_BASE     0x40021000
#define GPIO_E_MODER    ((volatile unsigned int *)  (PORT_E_BASE))
#define GPIO_E_OTYPER   ((volatile unsigned short *)(PORT_E_BASE+0x4))
#define GPIO_E_PUPDR    ((volatile unsigned int *)  (PORT_E_BASE+0xC))
#define GPIO_E_IDRLOW   ((volatile unsigned char *)(PORT_E_BASE+0x10))
#define GPIO_E_ODRLOW   ((volatile unsigned char *) (PORT_E_BASE+0x14))

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

#define setbit(x)   ( *GPIO_E_ODRLOW |= (1<<x) )
#define clearbit(x) ( *GPIO_E_ODRLOW &= ~(1<<x))


void delay_100_micro( void )
{
    /* SystemCoreClock = 168000000
     * blocking delay
     */
    *STK_CTRL = 0;
    *STK_LOAD = ( 16800-1 );/* 100 us */
    *STK_VAL = 0;
    *STK_CTRL = 5;
    while( (*STK_CTRL & 0x10000)==0);
}
void delay_ms( unsigned int ms )
{
    /* Number of 100 usecs delays
     * are obtained from:
     * (ms * 10), assuming ms > 0
     */
    unsigned int delay = ms*10;
    while( delay ){
        delay_100_micro();
        delay--;
    }
}

void delay_shortpulse( unsigned int w )
{
    /* Number of 100 usecs delays
     * are obtained from:
     * (w *10) - 2, assuming w > 0
     */
    unsigned int delay = (w*10)-2;
    while( delay ){
        delay_100_micro();
        delay--;
    }
}
```

```
void init( void )
{
  /* Only b3-b0 are used here,
   * b4-b7 used bya other application,
   * hence Read/Modify/Write operations
   */
  *GPIO_E_MODER  &= 0xFFFFFF00;
  *GPIO_E_MODER  |= 0x00000050;
  *GPIO_E_PUPDR  &= 0xFFFFFF00;
  *GPIO_E_PUPDR  |= 0x00000055;
  *GPIO_E_OTYPER &= 0xFFFFFFF3;

  setbit(3);
  clearbit(2);
}


void main(void)
{
    unsigned int w,p;
    init();
    while( 1 )
    {
        switch( *GPIO_E_IDRLOW & 3 )
        {
            case 0: w = 2; p = 10; break;
            case 1: w = 4; p = 10; break;
            case 2: w = 20; p = 100; break;
            default: w = 40; p = 100; break;
        }
        setbit( 2 );
        delay_100_micro();
        clearbit(3);
        delay_shortpulse( w );
        setbit(3);
        delay_100_micro();
        clearbit(2);
        delay_ms( p-w );
    }
}
```

Uppgift 4:

Här ska du ha identifierat likheterna med laboration 4. Såväl initieringar som avbrottshantering ska vara korrekt utformad.
Speciellt gäller att avbrottsbegäran ska återställas såväl i porten som EXTI-modulen.
EXTI5, EXTI6 EXTI7 och EXTI8 delar avbrottsvektor. Endast en avbrottsrutin ska därför konstrueras.

```
#define SYSCFG_EXTICR2  ( ( volatile unsigned int *) 0x4001380C)
#define SYSCFG_EXTICR3  ( ( volatile unsigned int *) 0x40013810)
#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 NVIC_EXTI5_9_IRQ_BPOS    (1<<23)
#define EXTI5_IRQ_BPOS           (1<<5)
#define EXTI6_IRQ_BPOS           (1<<6)
#define EXTI7_IRQ_BPOS           (1<<7)
#define EXTI8_IRQ_BPOS           (1<<8)
#define EXTI5_9_IRQVEC           0x2001C09C

typedef struct sCruiseControl{
    volatile unsigned char ctrl;
    volatile unsigned char sm;
    volatile unsigned char vel;
    volatile unsigned char ir;
}CRUISE_CONTROL, *PCRUISE_CONTROL;
#define CruiseC (PCRUISE_CONTROL 0xA00)

#define INC     8
#define DEC     4
#define LOCK    2
#define AUTO    1
#define IREQ    8
#define DREQ    4
#define CRON    2
#define CROFF   1

void app_init(void)
{
    CruiseC->ctrl = 0;
    *SYSCFG_EXTICR2 |= 0x4440;
    *SYSCFG_EXTICR3 |= 0x0004;
    *EXTI_IMR |= ( EXTI5_IRQ_BPOS|EXTI6_IRQ_BPOS|EXTI7_IRQ_BPOS|EXTI8_IRQ_BPOS );
    *EXTI_RTSR |= ( EXTI5_IRQ_BPOS|EXTI6_IRQ_BPOS|EXTI7_IRQ_BPOS|EXTI8_IRQ_BPOS );
    *EXTI_FTSR &= ~( EXTI5_IRQ_BPOS|EXTI6_IRQ_BPOS|EXTI7_IRQ_BPOS|EXTI8_IRQ_BPOS );
    *((void (**)(void) ) EXTI5_9_IRQVEC ) = irq_handler;
    *NVIC_ISER0 |= NVIC_EXTI5_9_IRQ_BPOS;
}

void irq_handler( void )
{
    if( CruiseC->ir & AUTO ){
        if( CruiseC->ir & CROFF) {
            CruiseC->ctrl = 0;
            CruiseC->ir |= CROFF;
            *EXTI_PR |= EXTI5_IRQ_BPOS;
        }else if( CruiseC->ir & DREQ ) {
            if( CruiseC->vel != 0 )
                CruiseC->ctrl = DEC|AUTO;
            CruiseC->ir |= DREQ;
            *EXTI_PR |= EXTI7_IRQ_BPOS;
        }else if( CruiseC->ir & IREQ ) {
            if( CruiseC->vel != 0xFF )
                CruiseC->ctrl = INC|AUTO;
            CruiseC->ir |= IREQ;
            *EXTI_PR |= EXTI8_IRQ_BPOS;
        }
    }else if( CruiseC->ir & CRON ) {
            CruiseC->ctrl = 0;
            CruiseC->sm = 0x55;
            CruiseC->sm = 0xAA;
            CruiseC->sm = 0x55;
            CruiseC->ctrl = AUTO;
            CruiseC->ctrl = LOCK|AUTO;
            CruiseC->ir |= CRON;
            *EXTI_PR |= EXTI6_IRQ_BPOS;
    }
}
```