Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

# Tentamen med lösningsförslag

## EDA488 Maskinorienterad programmering Z

Måndag 17 mars 2025, kl. 14.00 - 18.00

---

**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 (”över-strykningar”) 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**

***Uppgifter 1 t.o.m 3 utgörs av flervalsfrågor***.
Försök först lösa uppgifterna, så som du lärt dig i kursen, jämför sedan med flervalsalternativen och ange det alternativ du anser är korrekt. Högst ett svarsalternativ får anges. Siffror inom parentes anger möjliga poäng på uppgiften.

Observera att, bland svarsalternativen finns alternativ som *kan* ge poängavdrag. Om du är mycket osäker på ditt svar bör du därför i stället välja att avstå.

Avge dina flervalssvar på blanketten, längst bak i tentamenstesen. Skicka med denna blankett med dina övriga lösningar.

***För övriga uppgifter gäller:***

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 poäng är 40 och tentamenspoäng ger slutbetyg enligt:
16p ≤ betyg 3 < 24p ≤ betyg 4 < 32p ≤ betyg 5

**Uppgift 1 (-1..4p)**

En C-funktion `f` är definierad enligt listning till höger.

```
int f(short a, short b) {
  for(int i=0; i<=a; i++) { b = b + b; }
  return b;
}
```

Ange det alternativ (A-H) som visar en korrekt implementering (dvs. hur funktionen ska kodas) i assemblerspråk.

**A**

```
f:  MOV R3, #0
l:  CMP R3, R0
    BHI r
    ADD R3, #1
    ADD R1, R1
    B l
r:
    BX LR
```

**B**

```
f:  LDR R0, =a
    LDR R0, [R0]
    MOV R3, #0
l:  CMP R3, R0
    BGT r
    ADD R3, #1
    ADD R1, R1
    B l
r:  MOV R0, R1
    BX LR
```

**C**

```
f:  PUSH {LR}
    MOV R3, #0
l:  CMP R3, R0
    BHI r
    ADD R3, #1
    ADD R1, R1
    B l
r:  MOV R0, R1
    POP {LR}
    BX LR
```

**D**

```
f:  MOV R3, #0
l:  CMP R3, R0
    BGT r
    ADD R3, #1
    ADD R1, R1
    B l
r:
    BX LR
```

**E**

```
f:  PUSH {LR}
    MOV R3, #0
l:  CMP R3, R0
    BHI r
    ADD R3, #1
    ADD R1, R1
    B l
r:
    POP {LR}
    BX LR
```

**F**

```
f:  LDR R0, =a
    LDR R0, [R0]
    MOV R3, #0
l:  CMP R3, R0
    BGT r
    ADD R3, #1
    ADD R1, R1
    B l
r:
    BX LR
```

**G**

```
f:  MOV R3, #0
l:  CMP R3, R0
    BGT r
    ADD R3, #1
    ADD R1, R1
    B l
r:  MOV R0, R1
    BX LR
```

**H**

```
f:  MOV R3, #0
l:  CMP R3, R0
    BHI r
    ADD R3, #1
    ADD R1, R1
    B l
r:  MOV R0, R1
    BX LR
```

**Uppgift 2 (-1..4p)**

Funktionen `apply_bit_mask(v, m)` uför bitvis AND mellan varje byte i en integer (4 bytes) som skickas via parametern `v`, och den byte som skickas i parametern `m`. Den anropande funktionens värde skall uppdateras via `v` och funktionen skall returnera antalet ändrade bitar.

Funktion `int count_bit_diff (int a, int b);` returnerar hur många bitar som skiljer sig mellan två integers:

Exempel: Om masken `m = 0b00001111` appliceras på värdet `v = 0xFFFFFFFF` skall värdet som `v` refererar till, efter funktionsanropet, vara `0x0F0F0F0F`, och funktionen skall returnera att 16 bitar ändrats.

Välj den kod som korrekt implementerar det beskrivna beteendet.

**A**

```
int apply_bit_mask(int *v, char m){
    int orig_v = *v;
    int *cptr = v;
    for (int i = 0; i < 4; i++){
        *cptr = *cptr & m;
        cptr++;
    }
    return count_bit_diff(orig_v, *v);
}
```

**B**

```
int apply_bit_mask(int v, char m){
    int orig_v = v;
    int *cptr = &v;
    for (int i = 0; i < 4; i++){
       cptr = cptr & m;
       cptr++;
    }
    return count_bit_diff(orig_v, v);
}
```

**C**

```
int apply_bit_mask(int *v, char m){
    int orig_v = *v;
    char *cptr = (char*)v;
    for (int i = 0; i < 4; i++){
        *(cptr + i) &= m;
    }
    return count_bit_diff(orig_v, *v);
}
```

**D**

```
int apply_bit_mask(int v, char *m){
    int orig_v = v;
    char *cptr = &v;
    for (int i = 0; i < 4; i++){
        *cptr = *cptr & *m;
        cptr += 8;
    }
    return count_bit_diff(orig_v, v);
}
```

**E**

```
int apply_bit_mask(int *v, char m){
    int orig_v = v;
    int *cptr = (char*)v;
    for (int i = 0; i < 4; i++){
       *cptr = *cptr & m;
       cptr += 8;
    }
    return count_bit_diff(orig_v, *v);
}
```

**F**

```
int apply_bit_mask(int v, char m){
    int orig_v = v;
    int *cptr = &v;
    for (int i = 0; i < 4; i++){
       *cptr = *cptr & m;
       cptr++;
    }
    return count_bit_diff(orig_v, v);
}
```

**Uppgift 3 (-2..6p)**

Följande deklarationer är givna som globala variabler i C:

```
unsigned short a = 2;
unsigned char  b = 4;
short c;
int d[20];
```

a) Vilket av följande exempel är en korrekt deklaration av variablerna i assembler? (-1..2p)
(om två eller fler är korrekta, välj den som tar minst plats i minnet)

**A**
```
    .ALIGN  1
a:  .HWORD  2
b:  .BYTE   4
c:  .SPACE  2
    .ALIGN  2
d:  .SPACE  80
```

**B**
```
    .ALIGN  1
a:  .SPACE  2
b:  .SPACE  1
    .ALIGN  1
c:  .SPACE  2
    .ALIGN  2
d:  .WORD   20
```

**C**
```
    .ALIGN  1
a:  .HWORD  2
b:  .BYTE   4
    .ALIGN  1
c:  .SPACE  2
    .ALIGN  2
d:  .SPACE  20
```

**D**
```
    .ALIGN  1
a:  .HWORD  2
b:  .BYTE   4
    .ALIGN  1
c:  .HWORD  2
    .ALIGN  2
d:  .SPACE  80
```

**E**
```
    .ALIGN  1
a:  .HWORD  2
b:  .BYTE   4
c:  .HWORD  2
    .ALIGN  2
d:  .SPACE  80
```

**F**
```
    .ALIGN  1
a:  .SPACE  2
b:  .SPACE  1
c:  .SPACE  2
    .ALIGN  2
d:  .SPACE  80
```

**G**
```
    .ALIGN  1
a:  .HWORD  2
b:  .BYTE   4
    .ALIGN  1
c:  .SPACE  2
    .ALIGN  2
d:  .SPACE  80
```

**H**
```
    .ALIGN  1
a:  .SPACE  2
b:  .SPACE  1
c:  .HWORD  2
    .ALIGN  2
d:  .SPACE  80
```

b) Följande tilldelning skall göras:

```
d[a + b] = c;
```

Vilken av följande lösningar är en korrekt implementation av tilldelningen i assembler? (-1..4)

**A**
```
LDR  R0, =a
LDR  R0, [R0]
LDR  R1, =b
LDR  R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #2
LDR  R2, =c
LDR  R2, [R2]
SXTH R2, R2
STRH R2, [R0, R1]
```

**B**
```
LDR  R0, =a
LDRH R0, [R0]
LDR  R1, =b
LDR  R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #4
LDR  R2, =c
LDRH R2, [R2]
STR  R2, [R0, R1]
```

**C**
```
LDR  R0, =a
LDRH R0, [R0]
LDR  R1, =b
LDRB R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #2
LDR  R2, =c
LDRH R2, [R2]
STRH R2, [R0, R1]
```

**D**
```
LDR  R0, =a
LDRH R0, [R0]
LDR  R1, =b
LDRB R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #2
LDR  R2, =c
LDR  R2, [R2]
SXTH R2, R2
STR  R2, [R0, R1]
```

**E**
```
LDR  R0, =a
LDRH R0, [R0]
LDR  R1, =b
LDRB R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #2
LDR  R2, =c
LDRH R2, [R2]
SXTH R2, R2
STR  R2, [R0, R1]
```

**F**
```
LDR  R0, =a
LDR  R0, [R0]
LDR  R1, =b
LDR  R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #4
LDR  R2, =c
LDR  R2, [R2]
SXTH R2, R2
STR  R2, [R0, R1]
```

**G**
```
LDR  R0, =a
LDRH R0, [R0]
LDR  R1, =b
LDR  R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #4
LDR  R2, =c
LDRH R2, [R2]
STRH R2, [R0, R1]
```

**H**
```
LDR  R0, =a
LDRH R0, [R0]
LDR  R1, =b
LDRB R1,[R1]
ADD  R0, R1
LDR  R1, =d
LSL  R0, #4
LDR  R2, =c
LDRH R2, [R2]
SXTH R2, R2
STRH R2, [R0, R1]
```

## Uppgift 4 (8p)

En enhet för direkt överföring till minnet (DMA, *direct memory access*), med följande gränssnitt, är ansluten till *MD407* på adress `0xF0008000`.

Registeruppsättning

| offset | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Register |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | | CTRL |
| 2 | | | | | | | | | | | | | | | | | STAT |
| 4 | | | | | | | | | | | | | | | | | ADDR_LOW |
| 6 | | | | | | | | | | | | | | | | | ADDR_HIGH |
| 8 | | | | | | | | | | | | | | | | | SIZE |

Åtkomst av register CTRL och STAT måste vara *byte*-format

Åtkomst av register ADDR_LOW kan vara word-format och ger då 32 bitars adress (little endian format)

CTRL (*Control*) Styrregister, samtliga bitar är läs- och skrivbara.

| offset | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0x00 | IRQPRI | | | | TRIG | | | EX |

EX: (*Execute*), starta utförande av DMA-överföring.
TRIG: (*Trigger*), villkor för att starta överföringen.
IRQPRI: (*Interrupt priority*), 0 är högsta prioritet, 0xF är ”inga avbrott”.

STAT (*Status*) Statusregister, samtliga bitar är läsbara.

| offset | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0x02 | FAILURECODE | | | | | FAIL | IRQ | BUSY |

BUSY: (*Busy*), upptagen med att utföra kommando. Sätts till 1 av hårdvaran då EX sätts till 1. Återställs automatiskt då överföringen är klar.
IRQ: (*Interrupt Request*), sätts till 1 av hårdvara då BUSY blir 0. Återställs då BUSY sätts till 1 av hårdbvara.
FAIL: (Failure): Fel har uppstått, beskrivs av FAILURECODE. Återställs automatiskt då EX sätts till 1.
FAULURECODE: då FAIL=1 innehåller fältet en kod som beskriver felet.

ADDR_LOW (Address Low)

| offset | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x04 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |

Registret håller de 16 minst signifikanta bitarna (A15..A0) av den 32-bitars adressen för direkt överföring av helt block (1kByte) till eller från minnet.

ADDR_HIGH (Address High)

| offset | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x06 | A31 | A30 | A29 | A28 | A27 | A26 | A25 | A24 | A23 | A22 | A21 | A20 | A19 | A18 | A17 | A16 |

Registret håller de 16 mest signifikanta bitarna (A31-A16) av den 32-bitars adressen för direkt överföring av helt block (1kByte) till eller från minnet.

SIZE, samtliga bitar är läs- och skrivbara.

| offset | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x08 | NUMBLOCKS | | | | | | | | | | | | | | | |

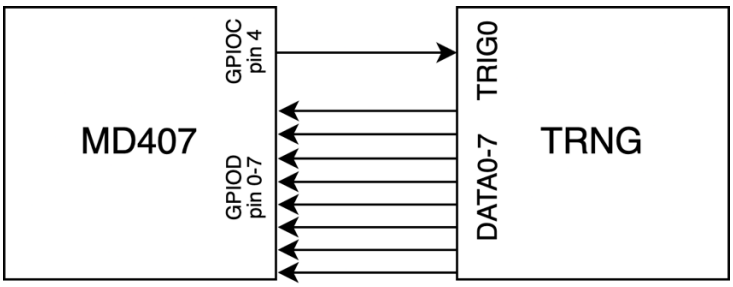
NUMBLOCKS, heltal utan tecken, antalet minnesblock (1024kByte) vid överföringen

a) Visa en lämplig typdeklaration (*struct*-format) med bitfältsdeklarationer för fälten i portens register (5p)
b) Använd deklarationen i a) och visa hur följande initiering kodas i ’C’. (3p)

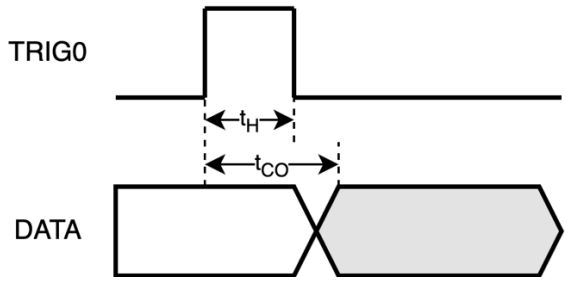
```
ADDR<- 0x20010000
SIZE<- 0x24
TRIG<- 4
IRQPRI<- 0xF
EX<- 1
```

**Uppgift 5 (8p)**

En 8-bitars slumpgeneratormodul (TRNG) är kopplad till GPIO-port C och D på en *MD407* enligt följande skiss.




Vid positiv flank på ingången TRIG0 genererar TRNG-modulen ett 8-bitars slumpmässigt tal som läggs ut på dess utgångar DATA0-7. Modulen opererar enligt följande tidskrav.




$t_H$ = 23 mikrosekunder (den kortaste tiden som TRIG0 måste vara '1' för att aktivera TRNG)
$t_{CO}$= 31 mikrosekunder (den tid det tar från positiv flank på TRIG0 tills det slumpmässiga talet har genererats).

a) Skriv en funktion `delay_1us()` som med hjälp av SYSTICK åstadkommer en blockerande fördröjning på 1 mikrosekund. För full poäng ska du också definiera och använda macrodefinitioner för registren i SYSTICK. Använd de namnkonventioner som finns givna i *QuickGuide*. (4p)

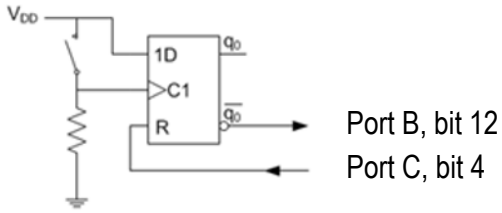
b) Skriv en funktion `get_random_char()` som triggar slumpgeneratorn och därefter läser och returnerar det slumpmässiga talet. Funktionen måste ta hänsyn till tidskraven ovan. Visa därför också en funktion `void delay_us( unsigned int us );` som fördröjer exekveringen us mikrosekunder och använder delay_1us() från uppgift a).
Följande makrodefinitioner är givna

```
#define GPIOC_ODR_LOW ((char *) 0x40020814)
#define GPIOD_IDR_LOW ((char *) 0x40020C10)
```

Du får anta att port C och D är korrekt inställda som utgång respektive ingång. (4p)

**Uppgift 6 (5p)**

En avbrottsvippa kopplas till *MD407* enligt följande figur:




*Vi påminner om vippans funktion*:

Då nivån på C1 ändras till 1 (positiv flank) ändras vippans interna tillstånd $q_0$ till 1D (1), om R samtidigt är 0.

Då R har värdet 1, tvingas $q_0$ till 0.

Den externa avbrottsmekanismen (EXTI) ska användas för att detektera knapptryckningar. Ingen hänsyn behöver tas till eventuella kontaktstudsar. Avbrott ska ske på negativ flank.

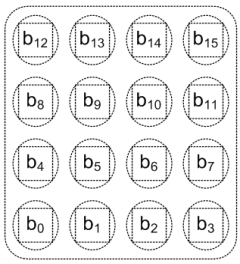
Förutsätt att makrodefinitioner för pekare till använda register finns givna med namnkonventioner enligt *QuickGuide*.

Visa med en funktion `app_init` hur avbrottsmekanismerna initieras, dvs. SYSCFG, EXTI, Port C, Port B och NVIC initieras.

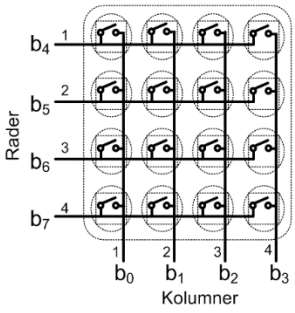
Tänk på att bara de portpinnar som berörs får ändras vid konfigureringen.

**Uppgift 7 (5p)**

Under laborationerna har du arbetat med ett enkelt tangentbord där tangenterna organiserats i rader och kolumner. Man vill kunna detektera flera samtidigt nedtryckta tangenter och därför konstruera en tangentbordsrutin som returnerar ett statusord (16 bitar) där varje tangent har en bitposition och värdet 1 indikerar en nedtryckt tangent medan värdet 0 indikerar en uppsläppt tangent. På grund av tangentbordets konstruktion är det lämpligt att välja avbildning enligt figuren till höger mellan bitposition och tangent.




Antag att port D, bit 0..7, kopplats till tangentbordet enligt figuren till höger. Eftersom flera tangenter kan tryckas ned samtidigt vill vi göra en ”kortslutningssäker” lösning för koincidensavsökning av tangentbordet. Utgångarna ska därför vara OPEN-DRAIN. De avlästa kolumnerna ska vara PULL UP.
Skapa en initieringsfunktion `void init_app( void )` som konfigurerar port D för användning med tangentbordet och en tangentbordsrutin `unsigned short keyb (void)` som söker av tangentbordet och placerar kolumnvärde för varje rad i en `unsigned short` och returnerar denna.




Förutsätt att makrodefinitioner för pekare till använda register finns givna med namnkonventioner enligt *QuickGuide*.

# Svarsblankett

## Anonym kod

**Maskinorienterad programmering, tentamen flervalsfrågor.**

**Kryssa i det alternativ du anser vara korrekt, eller avstå.**
**Lämna in denna blankett tillsammans med dina övriga lösningar.**

| Fråga | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | -1 | 0 | 2 | 0 | -1 | 4 | 3 |
| 2 | 1 | -1 | 4 | -1 | 0 | -1 | | |
| 3a | 1 | -1 | -1 | 1 | 0 | 0 | 2 | 0 |
| 3b | -1 | 0 | 1 | 2 | 4 | -1 | 0 | 0 |

# Lösningsförslag

**Uppgift 1 (4p):**

Svarsalternativ G

```
@ R0: parameter 'a'
@ R1: parameter 'b'
@ R3: lokal variabel 'i'
f:  MOV   R3,#0    @ i = 0;
_1: CMP   R3,R0    @ (i<=a)?
    BGT   _2        @
    ADD   R3,#1    @ i++;
    ADD   R1,R1    @ b = b+b;
    B     _1
_2: MOV   R0,R1    @ return b;
    BX    LR
```

**Uppgift 2 (4p):**

Svarsalternativ C

```
int apply_bit_mask(int *v, char m){
    int orig_v = *v;       /* Kopia av ursprungligt heltal */
    char *cptr = (char*)v; /* behöver pekare till byte */
    for (int i = 0; i < 4; i++){ /* för varje byte i heltalet... */
        *(cptr + i) &= m;  /* applicera mask på denna byte */
    }
    /* jämför heltal och returnera skillnad */
    return count_bit_diff(orig_v, *v);
}
```

**Uppgift 3a (-1..2p):**

Svarsalternativ G

```
    .ALIGN  1
a:  .HWORD  2
b:  .BYTE   4
    .ALIGN  1
c:  .SPACE  2
    .ALIGN  2
d:  .SPACE  80
```

**Uppgift 3b (-1..4p):**

Svarsalternativ E

```
    LDR     R0,=a       @ R0<- &a
    LDRH    R0,[R0]     @ R0<- a
    LDR     R1,=b       @ R1<- &b
    LDRB    R1,[R1]     @ R1<- b
    ADD     R0,R1       @ R0<- a+b
    LDR     R1,=d       @ R1<- &d
    LSL     R0,#2       @ R0<- (a+b)*4
    LDR     R2,=c       @ R2<- &c
    LDRH    R2,[R2]     @ R2<- c
    SXTH    R2,R2       @ R2<- (int) c
    STR     R2,[R0,R1] @ d[a+b] <- c
```

**Uppgift 4 (8p)**

**Uppgift 4 a (5p)**

```
struct dma {
  volatile unsigned char ex:1, trig:3, irqpri:4;
  volatile unsigned char :0,
busy:1,irq:1,fail:1,failurecode:5;
  union{
    volatile unsigned long addr;
  struct{
    volatile unsigned short addr_low;
    volatile unsigned short addr_high;
    }
  };
  volatile unsigned short size;
};
```

**Uppgift 4 b (3p)**

```
    ((struct dma *) 0xF0008000)->addr = 0x21000000;
    ((struct dma *) 0xF0008000)->size = 24;
    ((struct dma *) 0xF0008000)->trig = 4;
    ((struct dma *) 0xF0008000)->irqpri = 0xF;
    ((struct dma *) 0xF0008000)->ex = 1;
```

**Uppgift 5 a (4p)**

```
#define STK_BASE 0xE000E010
#define STK_CTRL ((volatile unsigned int *) (STK_BASE))
#define STK_LOAD ((volatile unsigned int *) (STK_BASE + 0x4))
#define STK_VAL ((volatile unsigned int *) (STK_BASE + 0x8))

void delay_1us(void){
  // stoppa systick
  *STK_CTRL = 0;

  // 168MHz -> 1us motsvarar 168 cykler
  *STK_LOAD = 168 - 1;
  *STK_VAL = 0;
  // starta nedräkning, utan avbrott
  *STK_CTRL = 5;
  // stoppa efter att COUNTFLAG ettställs
  while ((*STK_CTRL & (1 << 16) == 0);
  *STK_CTRL = 0;
}
```

**Uppgift 5 b (4p)**

```
void delay_us(int us){
  for(int i = 0; i < us; i++){
    delay_1us();
  }
}

char get_random_char(void){
  // positiv flank på TRIG0
  *GPIOC_ODR_LOW &= ~(1 << 4);
  *GPIOC_ODR_LOW |= (1 << 4);
  // vänta tH = 23us
  delay_us(23);
  // nollställ TRIG0
  *GPIOC_ODR_LOW &= ~(1 << 4);
  // vänta tCO - tH = 8 us
  delay_us(8);
  // läs in talet från DATA0-7
  char random_char = *GPIOD_IDR_LOW;
  return random_char;
}
```

**Uppgift 6 (5p)**

```
#define NVIC_EXTI12_IRQ_BPOS (1<<(40-32))
#define EXTI12_IRQ_BPOS          (1<<12)
void app_init( void )
{
  /* Port C, pin 4 utgång */
  *GPIOC_MODER  &= ~(0x300);
  *GPIOC_MODER  |= (0x100);
  /* Utgång ”push/pull” */
  *GPIOC_OTYPER &= ~(1<<4);

  /* Port B, pin 12 ingång */
  *GPIOB_MODER  &= ~(0x3000);
  /* Ingång ”floating” */
  *GPIOB_PUPDR  &= ~(0x3000);
  /* PB12->EXTI12  */
  *SYSCFG_EXTI4 &= ~(0x000F);
  *SYSCFG_EXTI4 |= 0x0004;
  *EXTI_IMR |= EXTI12_IRQ_BPOS;
  *EXTI_FTSR |= EXTI12_IRQ_BPOS;
  *EXTI_RTSR &= ~EXTI12_IRQ_BPOS;
  *NVIC_ISER1 |= NVIC_EXTI12_IRQ_BPOS;
}
```

**Uppgift 7 (5p)**

```
void init_app( void )
{
    /*  PORT D
    b8-b4 used for output to rows
    b3-b0  used for input from columns
    */
    *GPIO_D_MODER   = 0x00005500;
    /* Input, pull up */
    *GPIO_D_PUPDR   = 0x00000055;
    /* outputs are open drain */
    *GPIO_D_OTYPER  = 0x00F0;
}

unsigned short keyb (void)
{
    int row;
    unsigned short pad;
    *GPIO_D_ODR_LOW = ~0x80;
    pad = (unsigned short)   (*GPIO_D_IDR_LOW & 0xF);
    *GPIO_D_ODR_LOW = ~0x40;
    pad |= (unsigned short)  ((*GPIO_D_IDR_LOW <<4) & 0x00F0);
    *GPIO_D_ODR_LOW = ~0x20;
    pad |= (unsigned short)  ((*GPIO_D_IDR_LOW <<8) & 0x0F00);
    *GPIO_D_ODR_LOW = ~0x10;
    pad |= (unsigned short)  ((*GPIO_D_IDR_LOW <<12) & 0xF000);
    *GPIO_D_ODR_LOW =  0xFF;
    return pad^0xFFFF;  /* Pattern is inverse */
}
```