Riccardo Beniamino 24 40 54

- **Es 2 Challange 1**
- **Es 2 Punti 4,3,2,1**

Inizio Esercitazione 2, Challenge 1

```
// Esercitazione 2, Challenge 1
/* 1. Controlla la frequenza di ciascun LED RGB separatamente

Scrivi un programma che utilizzi le tue subroutine per aumentare
automaticamente i livelli di luminosità di ciascuno dei LED RGB.

È possibile utilizzare una funzione di rampa lineare per modificare
la luminosità del LED da zero al massimo per un certo periodo di tempo
(3-4 secondi), quindi reimpostare il livello di luminosità su zero dopo
aver raggiunto il valore massimo.

Usa diversi tempi di rampa in modo che
la luminosità di ciascun LED cambi a una velocità diversa.
Dovresti vedere il colore del LED RGB cambiare man mano che i segnali di
rampa progrediscono. */

.text
.global main

.set HW_PWM,  0x43C00000

main:
  //Imposto frequenze (periodi) diverse per i canali 0,1,2 ma incremento il duty allo stesso modo per tutti i
canali, in questo modo
  //le finestre PWM avanzano con velocita' (e granularita') diverse, cosi' ottengo un GRADIENTE di colore,
sul primo led
  mov r1,#0 //seleziono il canale 0, per impostarne la freq
  ldr r5, =0xEA60 //60'000
  bl hw_pwm_set_frequency

  mov r1,#1
  ldr r5, =0x13880 //80'000
  bl hw_pwm_set_frequency

  mov r1,#2
  ldr r5, =0x186A0 //100'000 freq=1kHz
  bl hw_pwm_set_frequency

  mov r1,#0 //rimetto a posto r1
  mov r5,#0

  mov r0,#0 //canale 0 da abilitare
  ////mov r1,#0 //setto solo la freq del canale 0
  mov r2,#0 //canale sul quale impostare il duty
  mov r3,#0 //r3 invece serve per tenere la quantita' del duty nel loop
  ldr r4, =0x186A0 //r4 serve per impostare il duty massimo

  ////bl hw_pwm_set_frequency

loop:
  bl constant_delay

 //devo fare un ciclo for per impostare lo stesso duty ai canali 0,1,2 ossia al primo led
 for0:
 bl hw_pwm_set_dutycicle
 adds r2,r2,#1
 cmp r2, #3
 blt for0 //se r2 < 3 ripeti il ciclo, fine for
 mov r2, #0 //rimetto r2 a posto

  adds r3,r3,#1

  cmp r3,#1 //se r3=1 sono alla prima iterazione e abilito i canali
  bne a
  //devo fare un cilco for per abilitare tutti i canali
  for1:
  bl hw_pwm_enable
  adds r0,r0,#1
  cmp r0, #3
```

```
  blt for1 //se r0 < 3 ripeti il ciclo, fine for
  mov r0,#0 //rimetto r0 a posto
  a:
  cmp r3,r4 //se r3 = r4 sono all'ultima iterazione e spengo i canali
  blt b
  //devo fare un cilco for per disabilitare tutti i canali
  for2:
  bl hw_pwm_disable
  adds r0,r0,#1
  cmp r0, #3
  blt for2 //se r0 , 3 ripeti il ciclo, fine for
  mov r0,#0 //rimetto r0 a posto

  mov r3,#0 //rimetto il duty a zero
  b:
b loop

hw_pwm_set_frequency:  //ricevo su r1 il canale sul quale impostare la frequenza
    push {r1,r2,r3,r4,r5} //ricevo su r5 il valore della frequenza
  cmp r1, #0 //controllo del valore compreso tra 0 e 5
  blt exit1
  cmp r1, #5
  bgt exit1

    ldr r2, =HW_PWM

  lsl r1,r1,#4 //inidce per offset 1->10...5->50
  add r1,r1,#4 //sommo 4, per ottenere l'offset giusto
  add r3,r1,r2

  //ldr r4, =0x5F5E100 //Numero della frequenza 100.000.000 per avere 1S, il blocco HW lavora a 100 MHz
  //ldr r4, =0x186A0 //= 100'000 = 1mS -> 1kHz

  str r5,[r3] //store all'inidirizzo puntato da r3

  exit1:
  pop {r1,r2,r3,r4,r5}
bx lr

hw_pwm_set_dutycicle:  //ricevo su r2 il canale sul quale impostare il dutycicle
  push {r2,r3,r4,r5} //ricevo su r3 la quantita' del dutycicle
  cmp r2, #0 //controllo del valore compreso tra 0 e 5
  blt exit2
  cmp r2, #5
  bgt exit2


    ldr r5, =HW_PWM
  lsl r2,r2,#4 //inidce per offset 1->10...5->50
  add r2,r2,#8 //sommo 8, per ottenere l'offset giusto
  add r4,r2,r5

  str r3,[r4] //store all'inidirizzo puntato da r3

  exit2:
  pop {r2,r3,r4,r5}
bx lr

hw_pwm_enable: //ricevo su r0 il canale da abilitare
  push {r0,r1,r2}
  cmp r0, #0 //controllo del valore compreso tra 0 e 5
  blt exit
  cmp r0, #5
  bgt exit

    ldr r1, =HW_PWM
  lsl r0,r0,#4 //inidce per offset 1->10...5->50
  add r1,r0,r1
  mov r2, #1
  str r2,[r1] //ricevo su r1 l'indirizzo da abilitare

  exit:
  pop {r0,r1,r2}
bx lr

hw_pwm_disable: //ricevo su r0 il canale da disabilitare
```

```
  push {r0,r1,r2}
  cmp r0, #0 //controllo del valore compreso tra 0 e 5
  blt exit3
  cmp r0, #5
  bgt exit3

    ldr r1, =HW_PWM
  lsl r0,r0,#4 //inidce per offset 1->10...5->50
  add r1,r0,r1
  mov r2, #0
  str r2,[r1] //ricevo su r1 l'indirizzo da disbilitare

  exit3:
  pop {r0,r1,r2}
bx lr

constant_delay:
  push {r5}
    ldr r5, = 0x61A8 //= 25'000
  delay_loop:
      subs r5, r5, #1   // Decrementa il conteggio
      bne delay_loop    // Ripeti finché il conteggio non è zero
  pop {r5}
bx lr             // Ritorna dalla subroutine
```

Fine Esercitazione 2, Challenge 1

Inizio Esercitazione 2, Punto 4

```
/* 4. Controlla la luminosità/colore dei LED RGB utilizzando PWM hardware */
.text
.global main

.equ LED_CTL, 0x41210000
.set SW_DATA, 0x41220000
.set HW_PWM,  0x43C00000

main:
        mov r1,#5
        set_freq:
        bl hw_pwm_set_frequency //setto la stessa frequenza per i canali 0...5
        subs r1,r1,#1                   //su r1 in uscita dico su quali canali impostare la freq
        bge set_freq
        mov r1,#0 //rimetto a zero r1 altrimenti mi va in underflow
loop:
        bl read_switch_value_3sx
        bl read_switch_value_8dx
        bl hw_pwm_set_dutycicle
        bl channel_select
b loop

read_switch_value_3sx:
//leggo i 3 switch a sinstra per vedere quale canale e' abilitato
//in uscita su r0 dico quale canale abilitare
//in uscita su r2 dico su quale canale impsotare il duty
        push {r1}
        ldr r1,=SW_DATA
        ldr r1, [r1]

        and r1, r1, #0xE00 // Maschera per mantenere solo i primi 3 bit (0xE00 in esadecimale)
        LSR r1, r1, #9 //Sposta i 3 bit più significativi in posizione meno significativa, shift a destra di
9 posizioni

        mov r0,r1 // copio r1 in r0 cosi dico quale canale abilitare
        mov r2,r1 // copio r1 in r2 cosi dico su quale canale impostare il duty

        pop {r1}
bx lr

channel_select:
//questa subrutine serve a selezionare quale canale e' acceso
//e quale canale e' spento, usa solo r0 e lr, su r0 ricevo il canale da abilitare
        push {lr}

        switch:
```

```
        cmp r0, #0
        beq case0
    cmp r0, #1
    beq case1
    cmp r0, #2
    beq case2
    cmp r0, #3
    beq case3
    cmp r0, #4
    beq case4
    cmp r0, #5
    beq case5

    b default_case

case0:
        bl hw_pwm_enable // Accendo solo il Canale 0
                                        // Spengo i canali 1...5
        mov r0,#1
        bl hw_pwm_disable

        mov r0,#2
        bl hw_pwm_disable

        mov r0,#3
        bl hw_pwm_disable

        mov r0,#4
        bl hw_pwm_disable

        mov r0,#5
        bl hw_pwm_disable

        mov r0,#0 //rimetto a 0 r0
        b end_switch
case1:
        bl hw_pwm_enable // Accendo solo il Canale 1
                                        // Spengo i canali 0,2,3,4,5
        mov r0,#0
        bl hw_pwm_disable

        mov r0,#2
        bl hw_pwm_disable

        mov r0,#3
        bl hw_pwm_disable

        mov r0,#4
        bl hw_pwm_disable

        mov r0,#5
        bl hw_pwm_disable

        mov r0,#1 //rimetto a 1 r0
    b end_switch

case2:
    bl hw_pwm_enable // Accendo solo il Canale 2
                                        // Spengo i canali 0,1,3,4,5
        mov r0,#0
        bl hw_pwm_disable

        mov r0,#1
        bl hw_pwm_disable

        mov r0,#3
        bl hw_pwm_disable

        mov r0,#4
        bl hw_pwm_disable

        mov r0,#5
        bl hw_pwm_disable

        mov r0,#2 //rimetto a 2 r0
```

```
    b end_switch

case3:
    bl hw_pwm_enable // Accendo solo il Canale 3
                                        // Spengo i canali 0,1,2,4,5
        mov r0,#0
        bl hw_pwm_disable

        mov r0,#1
        bl hw_pwm_disable

        mov r0,#2
        bl hw_pwm_disable

        mov r0,#4
        bl hw_pwm_disable

        mov r0,#5
        bl hw_pwm_disable

        mov r0,#3 //rimetto a 3 r0
    b end_switch

case4:
    bl hw_pwm_enable // Accendo solo il Canale 4
                                        // Spengo i canali 0,1,2,3,5
        mov r0,#0
        bl hw_pwm_disable

        mov r0,#1
        bl hw_pwm_disable

        mov r0,#2
        bl hw_pwm_disable

        mov r0,#3
        bl hw_pwm_disable

        mov r0,#5
        bl hw_pwm_disable

        mov r0,#4 //rimetto a 4 r0
    b end_switch

case5:
    bl hw_pwm_enable // Accendo solo il Canale 5
                                        // Spengo i canali 0,1,2,4
        mov r0,#0
        bl hw_pwm_disable

        mov r0,#1
        bl hw_pwm_disable

        mov r0,#2
        bl hw_pwm_disable

        mov r0,#4
        bl hw_pwm_disable

        mov r0,#5 //rimetto a 5 r0
    b end_switch

default_case:
    // Codice per il caso di default
        // Potrei far spegnere tutto qui, per esempio
end_switch:
    // Continua l'esecuzione dopo lo switch
    pop {lr}
bx lr

read_switch_value_8dx:
//leggo gli 8 switch a destra per impostare la luminosita' ossia il duty
//per ogni switch acceso ho un incremento del 12,5 % di luminosita'
//uso r1, r9 che ripristino e r3 per passare il parametro dutycicle in uscita
        push {r1,r9}
        ldr r1,=SW_DATA
```

```
        ldr r1, [r1]

    mov r9, #0xFF
        and r1,r1,r9 //Maschera per mantenere solo gli 8 bit meno significativi
        lsl r1,r1,#9 //faccio uno shift a sinistra di 9 bit cosi' al massimo ho 255*(2^9)=130.560

        mov r3,r1 //ricorda r3 e' quello che uso per passare il duty
        pop {r1,r9}
bx lr

hw_pwm_enable: //ricevo su r0 il canale da abilitare
        push {r0,r1,r2}
        cmp r0, #0 //controllo del valore compreso tra 0 e 5
        blt exit
        cmp r0, #5
        bgt exit

    ldr r1, =HW_PWM
        lsl r0,r0,#4 //inidce per offset 1->10...5->50
        add r1,r0,r1
        mov r2, #1
        str r2,[r1] //ricevo su r1 l'indirizzo da abilitare

        exit:
        pop {r0,r1,r2}
bx lr

hw_pwm_disable: //ricevo su r0 il canale da disabilitare
        push {r0,r1,r2}
        cmp r0, #0 //controllo del valore compreso tra 0 e 5
        blt exit3
        cmp r0, #5
        bgt exit3

    ldr r1, =HW_PWM
        lsl r0,r0,#4 //inidce per offset 1->10...5->50
        add r1,r0,r1
        mov r2, #0
        str r2,[r1] //ricevo su r1 l'indirizzo da disbilitare

        //mov r0,#0 //riporto r0 a zero
        exit3:
        pop {r0,r1,r2}
bx lr

hw_pwm_set_frequency:  //ricevo su r1 il canale sul quale impostare la frequenza
    push {r1,r2,r3,r4} //La frequenza e' costante per tutti i canali, vedi qui sotto
        cmp r1, #0 //controllo del valore compreso tra 0 e 5
        blt exit1
        cmp r1, #5
        bgt exit1

    ldr r2, =HW_PWM

        lsl r1,r1,#4 //inidce per offset 1->10...5->50
        add r1,r1,#4 //sommo 4, per ottenere l'offset giusto
        add r3,r1,r2
                                        //100.000.000=1sec , 100.000.000/1000=100.000=1ms -> in freq 1 kHz
        ldr r4, =0x1FE00
                                        //Freq=130.560=(2^9)*255 cosi' sfrutto gli 8 bit degli interruttori
per il dutycicle
                                        //cosi ho un incremento di luminosita' del 12,5% per ogni switch
acceso
        str r4,[r3] //store all'inidirizzo puntato da r3

        exit1:
        pop {r1,r2,r3,r4}
bx lr

hw_pwm_set_dutycicle:  //ricevo su r2 il canale sul quale impostare il dutycicle
        push {r2,r3,r4,r5} //ricevo su r3 la quantita' del dutycicle
        cmp r2, #0 //controllo del valore compreso tra 0 e 5
        blt exit2
        cmp r2, #5
        bgt exit2
```

```
    ldr r5, =HW_PWM
        lsl r2,r2,#4 //inidce per offset 1->10...5->50
        add r2,r2,#8 //sommo 8, per ottenere l'offset giusto
        add r4,r2,r5

        //ldr r3,=0x2FAF080 //prova
        str r3,[r4] //store all'inidirizzo puntato da r3

        exit2:
        pop {r2,r3,r4,r5}
bx lr
```

Fine Esercitazione 2, Punto 4

Inizio Esercitazione 2, Punto 3

```
/* 3. Controlla la luminosità del LED utilizzando una software PWM */
// Variante 2, migliorata ulteriormente, fedele al testo
.text
.global main

.equ LED_CTL, 0x41210000
.set SW_DATA, 0x41220000

.set Freq_Number, 0xF4240 //1 milioni 4.194.304 da scrivere in hex
.set Duty_Number, 0x186A0 //100 k

main:

loop:
        ldr r0, =Freq_Number
        bl read_switch_value
        //ldr r1, =Duty_Number
        bl pwm_software
b loop

read_switch_value:
        ldr r1,=SW_DATA
        ldr r1, [r1]    //SOLUZIONE, al posto della moltiplicazione, uso uno shift a sinistra
        lsl r1, r1, #10 //spostamento a sinistra di 20 (ora 10) bit, cosi sposto da 12 bit interruttori a 32
bit, per un tempo piu' lungo, ragiona
bx lr

pwm_software:
//r0 frequenza pwm
//r1 duty-cicle pwm
push {r4,lr} //pure r4
        //cmp r0, #0 //protezione, non accetto freq zero o duty zero
        //beq exit1
        //cmp r1, #0
        //beq exit1

    mov r3, r0 //r3 da preservare
        mov r4, r1 //r4 da preservare
                //exit1:

    mov r2,r4

        bl led0_on

        bl delay

        bl led0_off
        sub r2,r3,r4 //r2=r0-r1 //r2=r3-r4
        bl delay

pop {r4,lr} //pure r4
bx lr

led0_on:
        ldr r1, =LED_CTL
        ldr r0, [r1] //get current value
        orr r0,r0,#1 //set the first bit (don't affect other bits)
        str r0, [r1] //write back to LED_DATA
```

```
        mov r0, #0 //rimetto a zero r0
        mov r1, #0 //rimetto a zero r1, servono nel compare dentro pwm_software
bx lr //return from subroutine

led0_off:
        ldr r1, =LED_CTL
        ldr r0, [r1] //get current value
        bic r0,r0,#1 //Spegni il primo bit (LED), lasciando gli altri invariati
        str r0, [r1] //write back to LED_DATA

        mov r0, #0 //rimetto a zero r0
        mov r1, #0 //rimetto a zero r1, serrvono nel compare dentro pwm_software
bx lr //return from subroutine

delay:
    cmp r2, #0       // Controlla se r2 è minore o uguale a zero, PROTEZIONE UNDERFLOW
    ble exit         // Se r2 è minore o uguale a zero, esci dalla funzione

        delay_loop:
        subs r2, r2, #1  // Decrementa il contatore
        bne delay_loop   // Ripeti finché il contatore non è zero

        exit:
bx lr            // Ritorna dalla subroutine
```

Fine Esercitazione 2, Punto 3

Inizio Esercitazione 2, Punto 2

```
/*Esercitazione 2, punto 2*/
/* Modificare il programma blinky LED per controllare la velocità di lampeggio con i valori degli
interruttori. Si noti che il valore di ritardo utilizzato sopra era un numero a 32 bit con i bit 28
e 29 impostati su "1" e tutti gli altri bit impostati su "0". Poiché hai meno di 32 switch con
cui lavorare, dovrai ridimensionare il numero letto dagli switch. È possibile utilizzare
un'operazione di moltiplicazione per aumentare il valore letto dagli switch, ma esistono altri
modi per ottenere un numero maggiore dagli switch. Riesci a trovare un modo per
selezionare una gamma di tempi di ritardo ragionevoli dagli interruttori senza utilizzare una
istruzione di moltiplicazione? */
.text
.global main

.equ LED_CTL, 0x41210000
.set SW_DATA, 0x41220000

@blinks the led!
main:
        blinky_loop:
        bl led0_toggle
        bl read_switch_value
        //movw r0,#0 @load lower bits of r0 with 0
        //movt r0,#0x3000 @load top 16 bits with 0x3000
        bl delay
        b blinky_loop

read_switch_value:
        ldr r1,=SW_DATA
        ldr r0, [r1]    //SOLUZIONE, al posto della moltiplicazione, uso uno shift a sinistra
        lsl r0, r0, #20 //spostamento a sinistra di 20 bit, cosi sposto da 12 bit interruttori a 32 bit, per
un tempo piu' lungo, ragiona
bx lr

led0_toggle:
        ldr r1, =LED_CTL // Carica l'indirizzo del registro dei LED in r1
        ldr r0, [r1]     // Carica il valore corrente del registro dei LED in r0
        eor r0, r0, #1   // Commuta il primo bit (LED), lasciando gli altri invariati
        str r0, [r1]     // Scrivi il nuovo valore nel registro dei LED
bx lr            // Ritorna dal sottoprogramma

delay:
        subs r0,r0,#1
        bne delay
bx lr
```

Fine Esercitazione 2, Punto 2

Inizio Esercitazione 2, Punto 1

```
//Esercitazione 2, punto 1
//1. Controlla un LED usando le subroutine Assembly
//Da usare con il debugger
.text
.global main

.equ LED_CTL, 0x41210000

main:
bl led0_on
bl led0_off
bl led0_toggle

loop:
//bl led0_on
//bl long_delay

//bl led0_off
//bl long_delay
//bl led0_toggle
b loop

led0_on:
ldr r1, =LED_CTL
ldr r0, [r1] //get current value
orr r0,r0,#1 //set the first bit (don't affect other bits)
str r0, [r1] //write back to LED_DATA
bx lr //return from subroutine

led0_off: //mia
ldr r1, =LED_CTL
ldr r0, [r1] //get current value
//orr r0,r0,#0 //set the first bit (don't affect other bits)
bic r0,r0,#1 //Spegni il primo bit (LED), lasciando gli altri invariati
str r0, [r1] //write back to LED_DATA
bx lr //return from subroutine

led0_toggle:
    ldr r1, =LED_CTL    // Carica l'indirizzo del registro dei LED in r1
    ldr r0, [r1]        // Carica il valore corrente del registro dei LED in r0
    eor r0, r0, #1      // Commuta il primo bit (LED), lasciando gli altri invariati
    str r0, [r1]        // Scrivi il nuovo valore nel registro dei LED
    bx lr               // Ritorna dal sottoprogramma

long_delay: //mia
        mov r0, lr

        mov r1, #30000
        appoggio1:
        bl delay
        subs r1,r1, #1
        bne appoggio1

        mov lr, r0
        bx lr


delay: //mia
        mov r2, #3000  // Numero di iterazioni per il ritardo di 1 ms
    appoggio:
    subs r2, r2, #1    // Decrementa il contatore
    bne appoggio // Ripeti finché il contatore non è zero

    bx lr
```

Fine Esercitazione 2, Punto 1