Riccardo Beniamino 24 40 54

- **Es 1 Challange 3,2,1**
- **Es 1 Punti 5,4,3,2**

Inizio Esercitazione 1, Challenge 3

```
/* Challenges
3. Toggle del LED con la pressione del pulsante.
Far accendere il primo LED verde quando si preme il primo pulsante
e spegnerlo quando si preme nuovamente il pulsante. Il LED non
dovrebbe cambiare stato quando il pulsante viene rilasciato o
se il pulsante viene tenuto premuto.*/

.text
.global main

.equ LED_CTL, 0x41210000  //Definisco il simbolo LED_CTL e gi assegno il valore esadicmale 0x41210000,
indirizzo a 32 bit dove e' mappato in memoria
.set SW_DATA, 0x41220000  //... il simbolo LET_CTL e gli assegno 0x41220000, indirizzo a 32 bit, dove e'
mappato in memoria
.set BTN_DATA, 0x41200000 //... il simbolo BTN_DATA e gli assegno 0x41200000, inidirzzo a 32 bit, dove e'
mappato in memoria, ciascuna cifra esadeciamle descrive 4 byte

main:
  ldr r1,=SW_DATA
  ldr r2,=LED_CTL
  ldr r3,=BTN_DATA

loop:
  ldr r4,[r3] @r4 stato button
  cmp r4,#1 //Comparo il valore assunto dai button con 1
  bne loop //se r4 diverso da 1 torno al loop

  bl delay //delay anti rimbalzo, vado al delay e torno qui
  mov r0,#0 //prova

  ldr r4,[r3] @r4 stato button
  cmp r4,#1 //se il tasto e' ancora premuto dopo delay allora eseguo
  beq ctrl_long_press_button

  b loop

ctrl_long_press_button:
        mov r12, #20000

        ldr r4,[r3] @r4 stato button
    cmp r4,#1
    beq conta_tempo

    cmp r0, r12
    bls bottone1_premuto
    bhi loop

  b loop

conta_tempo:
    add r0,r0,#1
    bl delay
    b ctrl_long_press_button

bottone1_premuto:
        mov r0,#0 //prova
        cmp r8, #1
    beq bottone1_premuto_succ
    mov r8, #1
    str r8,[r2]

        bl blocco_gen

    b loop
bottone1_premuto_succ:

        mov r8, #0
    str r8,[r2]
```

```
        bl blocco_gen

    b loop

delay:
        mov r9, #2000  // Numero di iterazioni per il ritardo di 1 ms
    delay_annidato:
    subs r9, r9, #1    // Decrementa il contatore
    bne delay_annidato // Ripeti finché il contatore non è zero

    bx lr

blocco_gen:
        blocco:
    ldr r4,[r3] @r4 stato button
    cmp r4,#1 //Comparo il valore assunto dai button con 1
              //se continuo a tenere il pulsante premuto mi blocco qui dentro
    beq blocco
        bx lr

.end
```

Fine Esercitazione 1, Challenge 3

Inizio Esercitazione 1, Challenge 2

```
/* Challenges
2. Utilizzare i pulsanti per bloccare un valore binario.
Tratta i primi 8 switch come un valore binario a 8 bit e "blocca" il valore
degli switch in una posizione di memoria interna quando viene
premuto uno dei pulsanti.

Mostra il valore bloccato sui primi otto LED e dimostra che la modifica
degli interruttori non cambia il valore del LED fino a quando non
viene premuto nuovamente il pulsante di blocco.
Cancellare il valore (impostare il valore su zero) quando viene
premuto un pulsante diverso.*/

.text
.global main

.equ LED_CTL, 0x41210000  //Definisco il simbolo LED_CTL e gi assegno il valore esadicmale 0x41210000,
indirizzo a 32 bit dove e' mappato in memoria
.set SW_DATA, 0x41220000  //... il simbolo LET_CTL e gli assegno 0x41220000, indirizzo a 32 bit, dove e'
mappato in memoria
.set BTN_DATA, 0x41200000 //... il simbolo BTN_DATA e gli assegno 0x41200000, inidirzzo a 32 bit, dove e'
mappato in memoria, ciascuna cifra esadeciamle descrive 4 byte

main:
  ldr r1,=SW_DATA
  ldr r2,=LED_CTL
  ldr r3,=BTN_DATA

loop:
  ldr r0,[r1] @r0 stato switch
  ldr r4,[r3] @r4 stato button

  and r0,r0,#0xff //maschera da 0 a 255, in decimale corrisponde a 8 bit, i primi 8 interruttori

  cmp r4,#1 //Comparo il valore assunto dai button con 1
  beq bottone1_premuto //Se r4==1 Salto all'etichetta bottone1_premuto
  bhi altri_bottoni_premuti //se r4>1 Salto all' etichetta altri bottoni premuti
  b loop

bottone1_premuto:
        str r0,[r2] //Scrivo il valore del registro r0 nel registro dei led r2
b loop

altri_bottoni_premuti:
        //Lista di errori fatti
        //str #0,[r2] //Non funziona con l'immediato
        //mov r2, #0 //Carica il valore zero nel registro r2 //si blocca
        //ldr r8,[r2] //Inutile! Carico nel registro di appoggio r8, il valore del registro dei led r2

        mov r8, #0  //Creo un registro di appoggio r8 vuto
```

```
        str r8,[r2] //Carico il registro vuoto r8, sul registro dei led r2
b loop

.end
```

Fine Esercitazione 1, Challenge 2

Inizio Esercitazione 1, Challenge 1

```
/* Challenges
1. Controlla i LED utilizzando sia gli interruttori che i pulsanti.
Considera i primi quattro LED come una funzione dei primi 4 interruttori e dei
quattro pulsanti e accendi un LED quando l'interruttore o il pulsante corrispondente
è attivato, ma non quando entrambi sono attivati. */

.text
.global main

.equ LED_CTL, 0x41210000
.set SW_DATA, 0x41220000
.set BTN_DATA, 0x41200000

main:
  ldr r1,=SW_DATA
  ldr r2,=LED_CTL
  ldr r3,=BTN_DATA

loop:
  ldr r0,[r1] @r0 stato switch
  ldr r4,[r3] @r4 stato button
  eor r6,r0,r4
  and r6,r6,#0x0000000f
  str r6,[r2]
  b loop

.end
```

Fine Esercitazione 1, Challenge 1

Inizio Esercitazione 1, Punto 5

```
/* 5. Usa i confronti per controllare più LED

Trattare i valori dei primi 4 interruttori come un numero binario e accendere i primi
4 LED individualmente se il numero rientra in un certo intervallo: Accendere LED0 se
il valore degli interruttori è compreso tra 0 e 3; LED 1 se il valore è compreso tra 4 e
7; LED 2 se il valore è compreso tra 8 e 11; e LED 3 se il valore è compreso tra 12 e
15.

Deve essere acceso un solo LED alla volta.

Hint: la lettura dello switch register restituisce il valore di tutti e 12 gli switch, ma
siamo interessati solo ai primi quattro switch. Abbiamo bisogno di un modo per
controllare il valore binario solo dei primi quattro interruttori. Potresti
semplicemente richiedere che gli otto interruttori superiori siano tutti impostati su
0, ma la richiesta di determinati stati di input come questo è generalmente
considerata una cattiva pratica, in particolare quando puoi rimuovere tale requisito
con una semplice elaborazione.

In questo caso, puoi semplicemente "mascherare" i
valori di switch più alti mettendo in AND il valore nel registro switch con
0x0000000F. È chiaro perché funziona? */

@punto 1.5 ultimo
.text
.global main

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

main:
  ldr r1,=SW_DATA
  ldr r2,=LED_CTL
```

```
loop:
  ldr r0,[r1]
  and r0,r0,#0x0000000f
  cmp r0,#3
  bls tre_zero
  cmp r0,#7
  bls quattro_sette
  cmp r0,#11
  bls sette_undici
  cmp r0,#16
  bls dodici_sedici
  b loop

tre_zero:
  mov r0,#1
  str r0,[r2]
  b loop
quattro_sette:
  mov r0,#2
  str r0,[r2]
  b loop
sette_undici:
  mov r0,#4
  str r0,[r2]
  b loop
dodici_sedici:
  mov r0,#8
  str r0,[r2]
  b loop

.end
```

Fine Esercitazione 1, Punto 5

Inizio Esercitazione 1, Punto 4

```
@punto 1.4 penultimo
.text
.global main

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

main:
  ldr r1,=SW_DATA
  ldr r2,=LED_CTL

loop:
  ldr r0,[r1]
  cmp r0,#28
  beq age
  mov r3,#0xfff
  cmp r0,r3
  beq task
  mov r0,#0
  str r0,[r2]
  b loop

age:
  mov r0,#1
  str r0,[r2]
  b loop

task:
  mov r0,#2
  str r0,[r2]
  b loop
```

Fine Esercitazione 1, Punto 4

Inizio Esercitazione 1, Punto 2

```
.text
.global main @il linker deve sapere quale è il main e lo facciamo globale

@define constaants, these can be used ay symobls in your code
.equ LED_CTL, 0x41210000 @posizione in memoria del registro dei LED
.set SW_DATA, 0x41200000 @posione in memoria del registro degli switch

@the set and equ directives are equivalent and can be used interchangeably

main: @etichetta main che dice quale è la prima istruzione
        ldr r1,=SW_DATA @load switch adress from constant
        ldr r2,=LED_CTL @load LED address from costant

        loop: @etichetta che dice dove cè una etichetta di testo
                ldr r0,[r1] @load switch value *r1 -> r0
                mvn r0, r0 @negate all content of r0
                str r0,[r2] @store value to led register *r2 <- r0
                b loop @go back to "loop"
.end @fine del codice
```

Fine Esercitazione 1, Punto 2