

ARM

*Practical project for STM32H7 embedded system
(informative, additional content)*

Sum of two numbers and LED diode blinking

VSCoDe, CubeIDE

STM32H750B-DK Discovery development system

- Arm® Cortex® core-based microcontroller with 128 Kbytes (STM32H750XBH6) of Flash memory and 1 Mbyte of RAM, in TFBGA240+25 package

- 4.3" RGB interface LCD with touch panel connector

- Ethernet compliant with IEEE-802.3-2002, and POE

- USB OTG FS with Micro-AB connector

- SAI audio codec

- One ST-MEMS digital microphone

- 2 x 512-Mbit Quad-SPI NOR Flash memory

- 128-Mbit SDRAM

- 4-Gbyte on-board eMMC

- 1 user and reset push-button

- Fanout daughterboard

- 2 x FDCANs

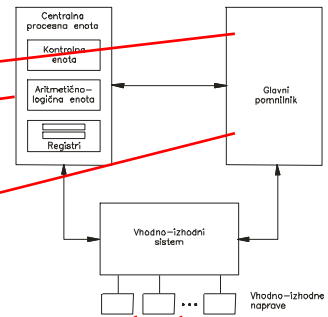
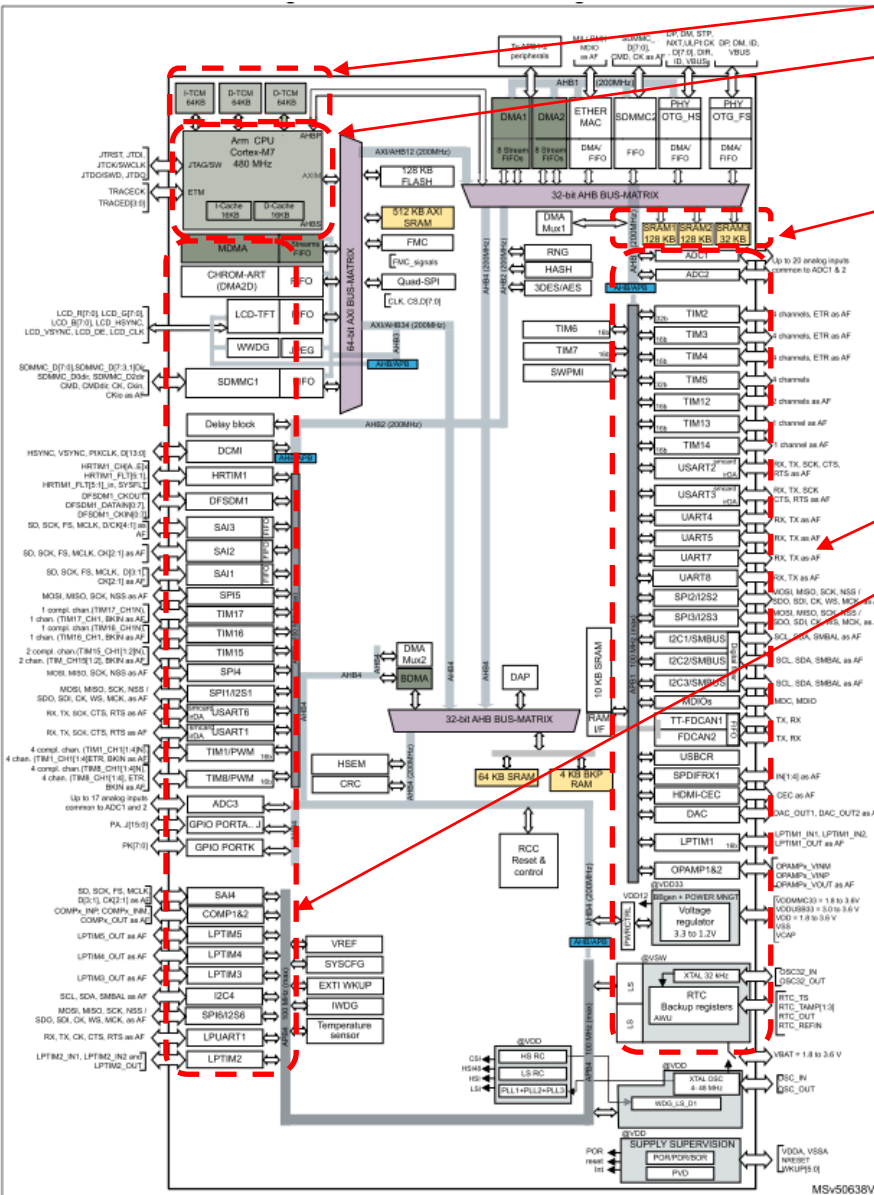
- Board connectors:

- USB FS Micro-AB connectors
- ST-LINK Micro-B USB connector
- USB power Micro-B connector
- Ethernet RJ45
- Stereo headset jack including analog microphone input
- Audio header for external speakers
- Arduino™ Uno V3 expansion connectors
- STMod+

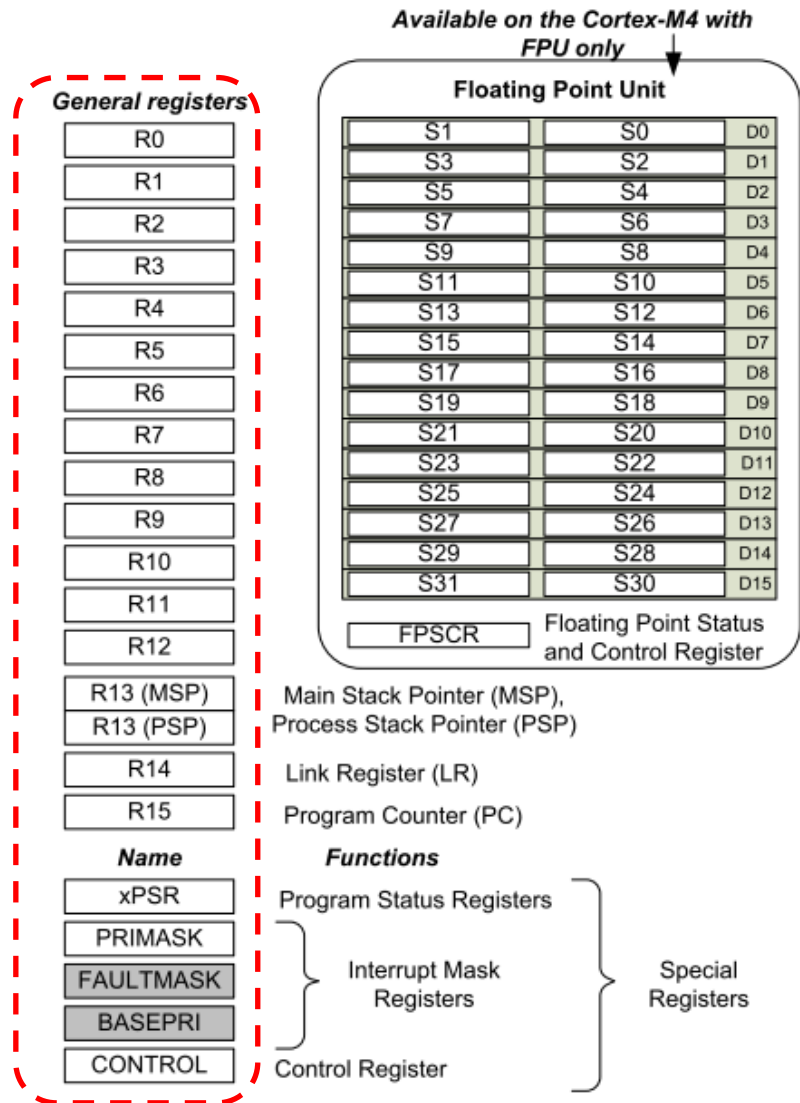


<https://www.st.com/en/evaluation-tools/stm32h750b-dk.html>

STM32H750XB MCU

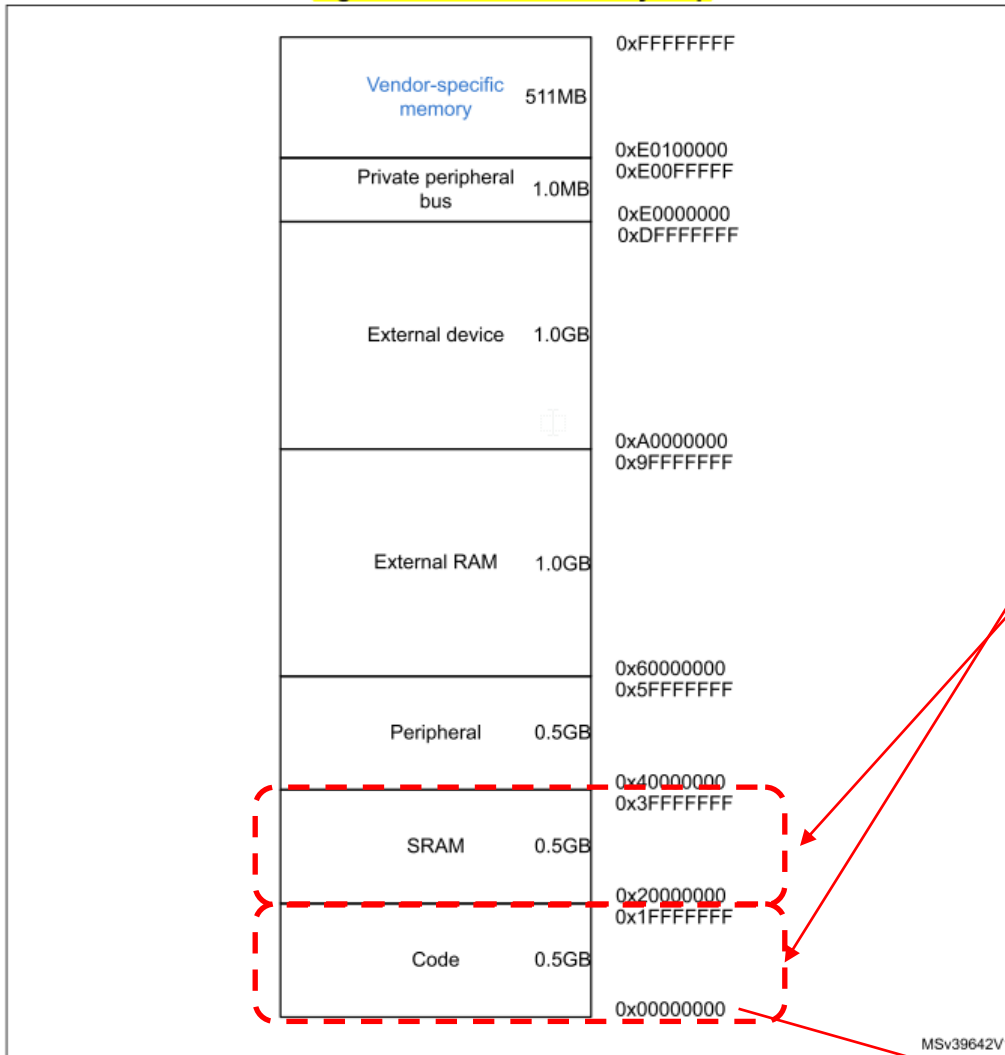


ARM Cortex M – Program model



ARM Cortex M – Memory (address) space

Figure 8. Processor memory map



Memory mapping description – linker

MEMORY

```

{
FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128K
DTCMRAM (xrw) : ORIGIN = 0x20000000, LENGTH = 128K
RAM_D1 (xrw) : ORIGIN = 0x24000000, LENGTH = 512K
RAM_D2 (xrw) : ORIGIN = 0x30000000, LENGTH = 288K
RAM_D3 (xrw) : ORIGIN = 0x38000000, LENGTH = 64K
ITCMRAM (xrw) : ORIGIN = 0x00000000, LENGTH = 64K
}
    
```

SysTick vector	1	0x0000003C
PendSV vector	1	0x00000038
Not used		0x00000034
Debug Monitor vector	1	0x00000030
SVC vector	1	0x0000002C
Not used		0x00000028
Not used		0x00000024
Not used		0x00000020
SecureFault (ARMv8-M Mainline)	1	0x0000001C
Usage Fault vector	1	0x00000018
Bus Fault vector	1	0x00000014
MemManage vector	1	0x00000010
HardFault vector	1	0x0000000C
NMI vector	1	0x00000008
Reset vector	1	0x00000004
MSP initial value		0x00000000

ARM Cortex M – Vector table – Initial start

Vector Table	Vector address (initial)
Interrupt#239 vector 1	0x000003FC
Interrupt#31 vector 1	0x000000BC
Interrupt#1 vector 1	0x00000044
Interrupt#0 vector 1	0x00000040
SysTick vector 1	0x0000003C
PendSV vector 1	0x00000038
Not used	0x00000034
Debug Monitor vector 1	0x00000030
SVC vector 1	0x0000002C
Not used	0x00000028
Not used	0x00000024
Not used	0x00000020
SecureFault (ARMv8-M Mainline) 1	0x0000001C
Usage Fault vector 1	0x00000018
Bus Fault vector 1	0x00000014
MemManage vector 1	0x00000010
HardFault vector 1	0x0000000C
NMI vector 1	0x00000008
Reset vector 1	0x00000004
MSP initial value	0x00000000

Address 4 („Reset vector“) contains the address of the 1st instruction of the „initialization“ part, i.e. the initial code is located in the `Reset_Handler` subroutine.

```

.section .text.Reset_Handler
    .weak Reset_Handler
    .type Reset_Handler, %function

Reset_Handler:
    ...

    ldr    sp, =_estack    /* set stack pointer */

    /* Call the application's entry point.*/

    bl    main

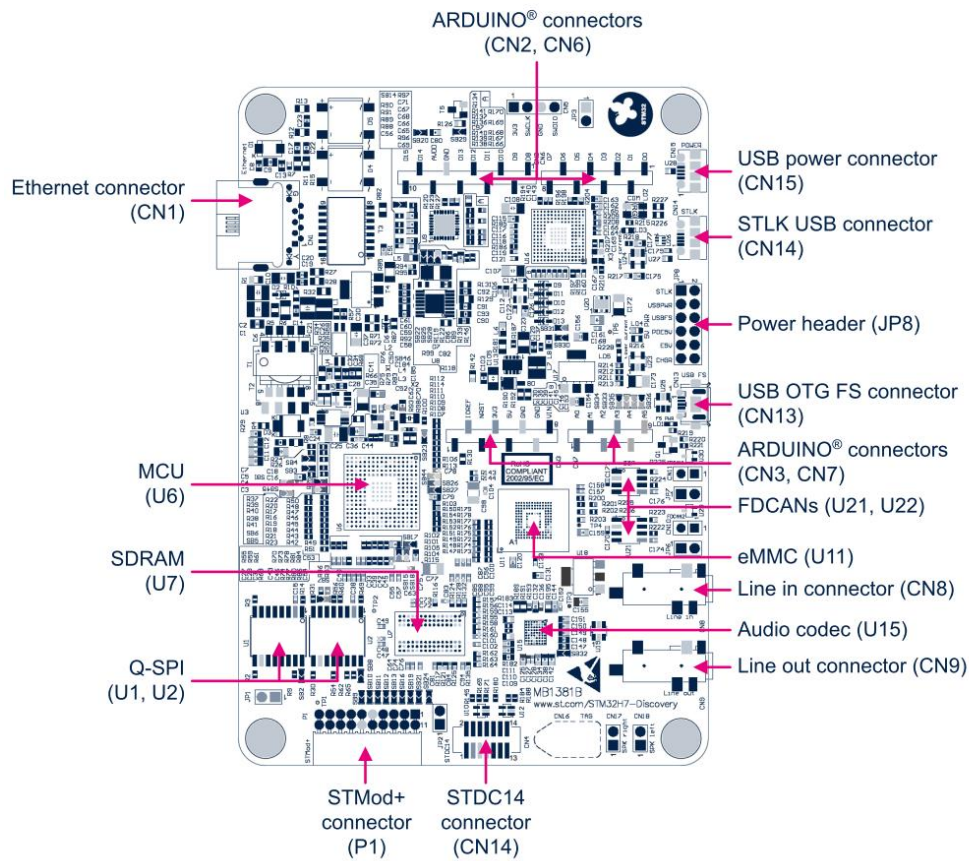
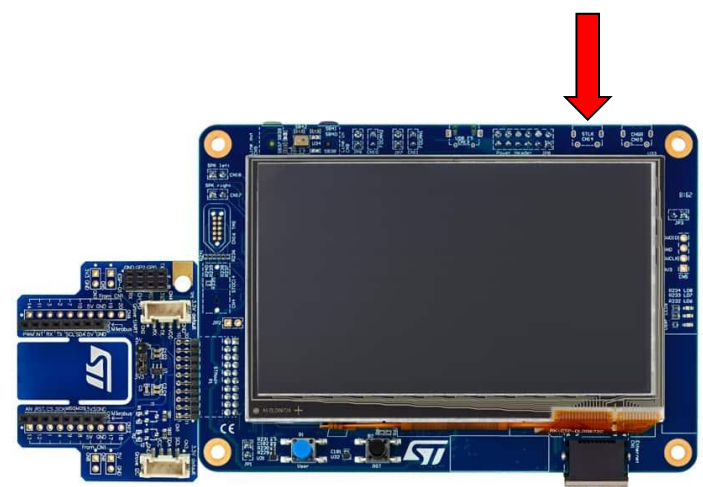
    bx    lr
  
```


Work on STM32H750B-DK board

Connection :

- **Micro USB** connector above the screen (arrow!), screen lights up.

Figure 5. STM32H745I-DISCO and STM32H750B-DK Discovery board bottom layout



Work on STM32H750B-DK board

Special project for STM32H750B-DK (e-učilnica) :

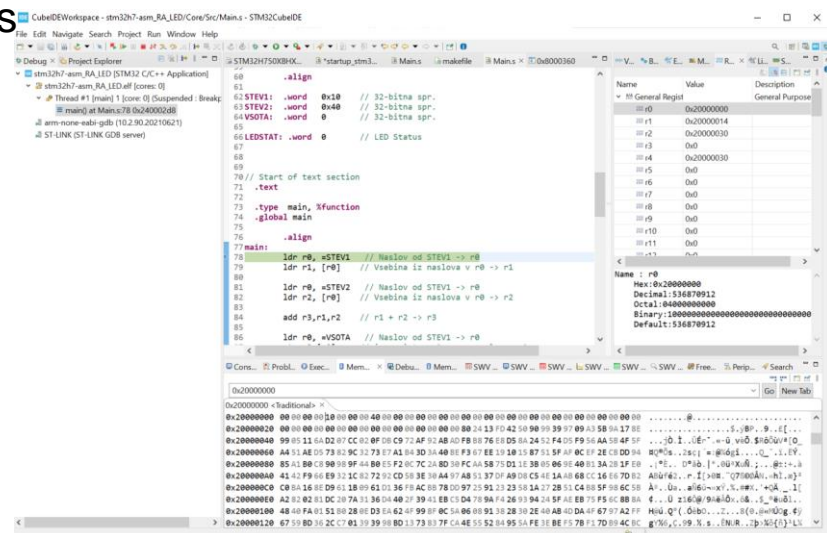
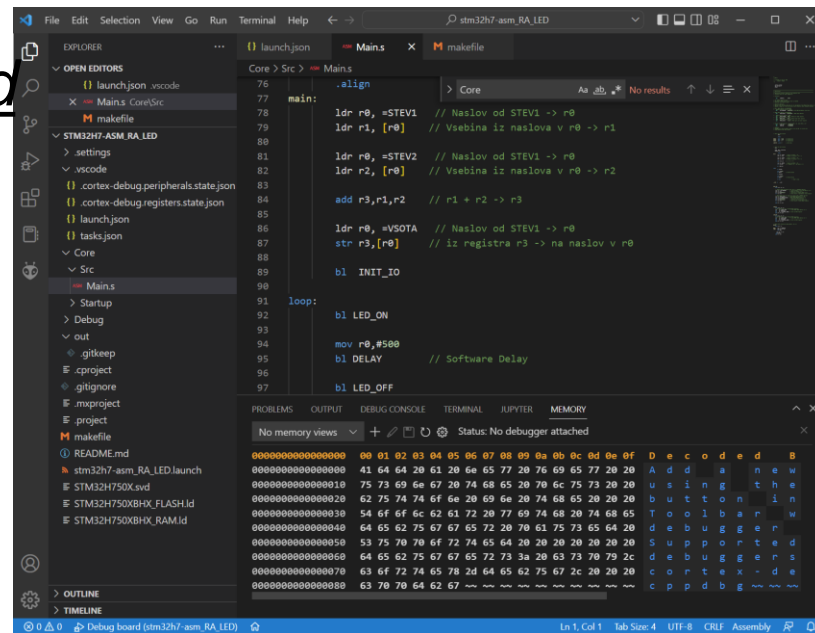
- 2 IDE environments (both „cross-platform“)

- VSCoDe (MS):

- Also usefull for other languages, projects
- Relies on external compilers
- All configurable (configurations)

- CubeIDE (ST):

- Customized za ST boards
- Tools for code generation, libraries setup - CubeMX
- Based on Eclipse IDE
- Easier installation



Specification of variables

```
Main.s :      // Start of data section
                .data                // začetek sekcije s spremenljivkami   (v RAM pomnilnik)

                .align

                STEV1: .word 0x10    // 32-bitna spr.
                STEV2: .word 0x40    // 32-bitna spr.
                VSOTA: .word 0       // 32-bitna spr.
```

Differences real board vs. CPULator simulator :

- Simulator (CPULator):
 - *Variables together with program*
 - *No special .data section*
 - *Simple instruction can be used to set the address in the register, e.g.:*
 - ***adr r0, STEV1***
- Board (STM32H750):
 - *Variables more distant from program*
 - *Variables in RAM memory, program in FLASH memory*
 - *Different instruction needed to set the address or any 32 bit value in the register, e.g.:*
 - ***ldr r0, =STEV1***

Program code

Main.s :

```
// Start of text section
.text                // začetek sekcije s
programom (v FLASH pomnilnik)
.type main, %function
.global main

.align

main:
ldr r0, =STEVI      // Naslov od STEVI -> r0
ldr r1, [r0]        // Vsebina iz naslova v r0 -> r1

ldr r0, =STEVI      // Naslov od STEVI -> r0
ldr r2, [r0]        // Vsebina iz naslova v r0 -> r2

add r3,r1,r2        // r1 + r2 -> r3

ldr r0, =VSOTA      // Naslov od STEVI -> r0
str r3,[r0]         // iz registra r3 -> na naslov v r0
```

stm32h7-asm_RA_LED.elf - /stm32h7-asm_RA_LED/Debug - 2. nov. 2022 08:23:53

Region	Start address	End addr...	Size	Free	Used	Usage (%)
RAM_EXEC	0x24000000	0x24020...	128 KB	127,03 KB	992 B	0,76%
DTCMRAM	0x20000000	0x20020...	128 KB	126,45 KB	1,55 KB	1,21%

Differences real board vs. CPULator simulator :

- Simulator (CPULator):
 - *We write pure program only, no „initialization“ part*
- Board (STM32H750):
 - *Main program is actually a subroutine:*
 - *Called from initialization part (startup_stm32h750xbhx.s)*
 - *„initialization“ part of the project is always started first when the system is powered up or reset key is pressed*
 - *During intensive development, we often store the program also in RAM memory (to save erase cycles in FLASH)*
 - *At the end, program is stored in FLASH memory to stay permanent.*

Initializaton part

startup_stm32h750xbhx.s:

g_pfnVectors:

```
.word _estack
.word Reset_Handler
.word NMI_Handler
.word HardFault_Handler
.word MemManage_Handler
.word BusFault_Handler
.word UsageFault_Handler
.word 0
.word 0
.word 0
.word SVC_Handler
.word DebugMon_Handler
.word 0
.word PendSV_Handler
.word SysTick_Handler
```

```
.section .text.Reset_Handler
.weak Reset_Handler
.type Reset_Handler, %function
```

Reset_Handler:

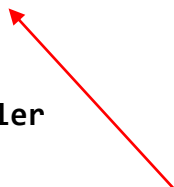
```
ldr sp, =_estack /* set stack pointer */

/* Call the application's entry point.*/

bl main

bx lr
```

SysTick vector	1	0x0000003C
PendSV vector	1	0x00000038
Not used		0x00000034
Debug Monitor vector	1	0x00000030
SVC vector	1	0x0000002C
Not used		0x00000028
Not used		0x00000024
Not used		0x00000020
SecureFault (ARMv8-M Mainline)	1	0x0000001C
Usage Fault vector	1	0x00000018
Bus Fault vector	1	0x00000014
MemManage vector	1	0x00000010
HardFault vector	1	0x0000000C
NMI vector	1	0x00000008
Reset vector	1	0x00000004
MSP initial value		0x00000000



Main program – sum and LED blinking

Main.s :

```
main:
    ldr r0, =STEVI // Naslov od STEVI -> r0 (simulator: adr r0,STEVI)
    ldr r1, [r0]    // Vsebina iz naslova v r0 -> r1

    ldr r0, =STEVI // Naslov od STEVI -> r0 (simulator: adr r0,STEVI)
    ldr r2, [r0]    // Vsebina iz naslova v r0 -> r2

    add r3,r1,r2    // r1 + r2 -> r3

    ldr r0, =VSOTA // Naslov od STEVI -> r0 (simulator: adr r0,VSOTA)
    str r3,[r0]    // iz registra r3 -> na naslov v r0
    ...
    bl INIT_IO     // Priprava V/I naprav (za kontrolo LED diode)

loop:
    bl LED_ON      // Podprogram za vklop LED diode

    mov r0,#500    // Podprogram za zakasnitev r0 msec
    bl DELAY

    bl LED_OFF     // Podprogram za izklop LED diode

    mov r0,#500    // Podprogram za zakasnitev r0 msec
    bl DELAY

    b loop        // Branch to loop (skok na vrstico loop:)

__end: b __end
```

Branch instruction (further explanation)

Branch is an instruction of the GOTO label type. Address of the instruction at the line with label is written to PC register.

b (Branch)

```
zanka:      ...
            sub r1, r1, #1
            b zanka @ GOTO zanka
```

Subroutines (further explanation)

When calling subroutines, we need to remember the return address !

Solution on ARM CPUs:

- **On subroutine call :**
 - return address (in pc) is stored in register r14 (link register = lr)
 - address of the 1st instruction of subroutine is put into pc
- **On return from subroutine**
 - return address restored from r14 (lr) to r15 (pc)
 - Made by push&pop instructions at the start and end of subroutine ->

Instruction to call subroutine:

```
BL SUBPROGR
```

- Branch with Link (L = 1) – store return address in r14.

Subroutines

Examples of subroutine calls

Case:

```

        bl  PODPROG
        ..
__end:   b  __end

PODPROG: push {r1, ..., lr}
        ..

        pop {r1, ..., pc}

```

LED_ON:

```

    push {r5, r6, lr}
// Set GPIOx Pins to 1 (through BSSR register)
ldr  r6, =GPIOI_BASE // Load GPIOI BASE address to r6
mov  r5, #LEDS_ON
str  r5, [r6,#GPIOx_BSSR] // Write to BSRR register
pop  {r5, r6, pc}

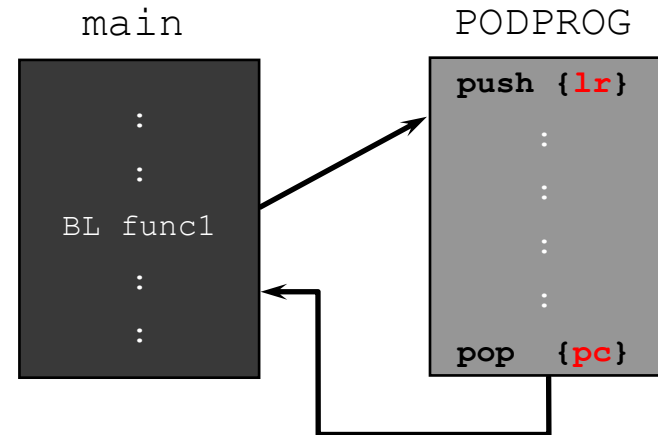
```

LED_OFF:

```

    push {r5, r6, lr}
// Set GPIOx Pins to 0 (through BSSR register)
ldr  r6, =GPIOI_BASE // Load GPIOI BASE address to r6
mov  r5, #LEDS_OFF
str  r5, [r6,#GPIOx_BSSR] // Write to BSRR register
pop  {r5, r6, pc}

```



Stack can be temporary storage also for used registers in subroutines, that change the content. After the return, their content can be restored from the stack.

```
// Delay with internal SW loop approx. r0 x ms
```

DELAY:

```
    push {r1, lr}
```

```
MSEC: ldr r1,=LEDELAY
```

```
LOOP: subs r1, r1, #1
      bne LOOP
```

```
    subs r0, r0, #1
```

```
    bne MSEC
```

```
    pop {r1, pc}
```

Challenges

- **Shorten time of on/off LED state**
- **Change proportion of on/off LED state**