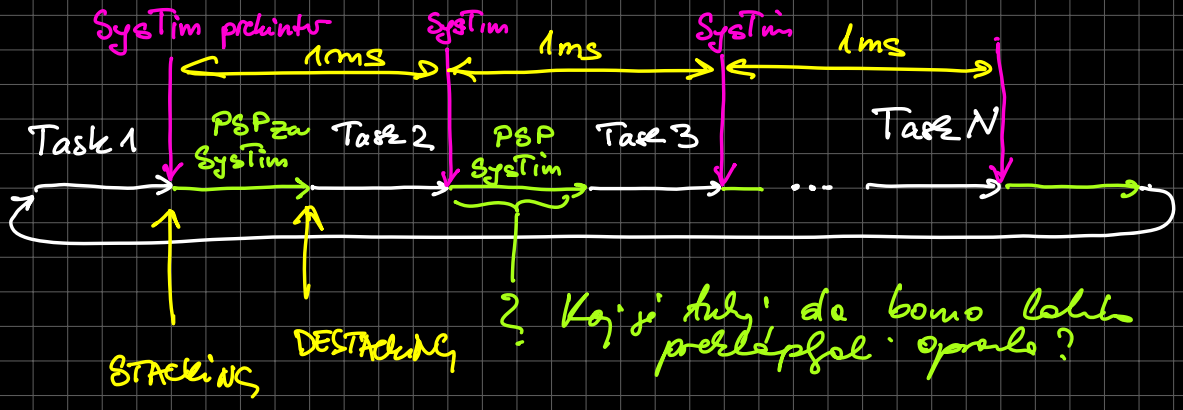
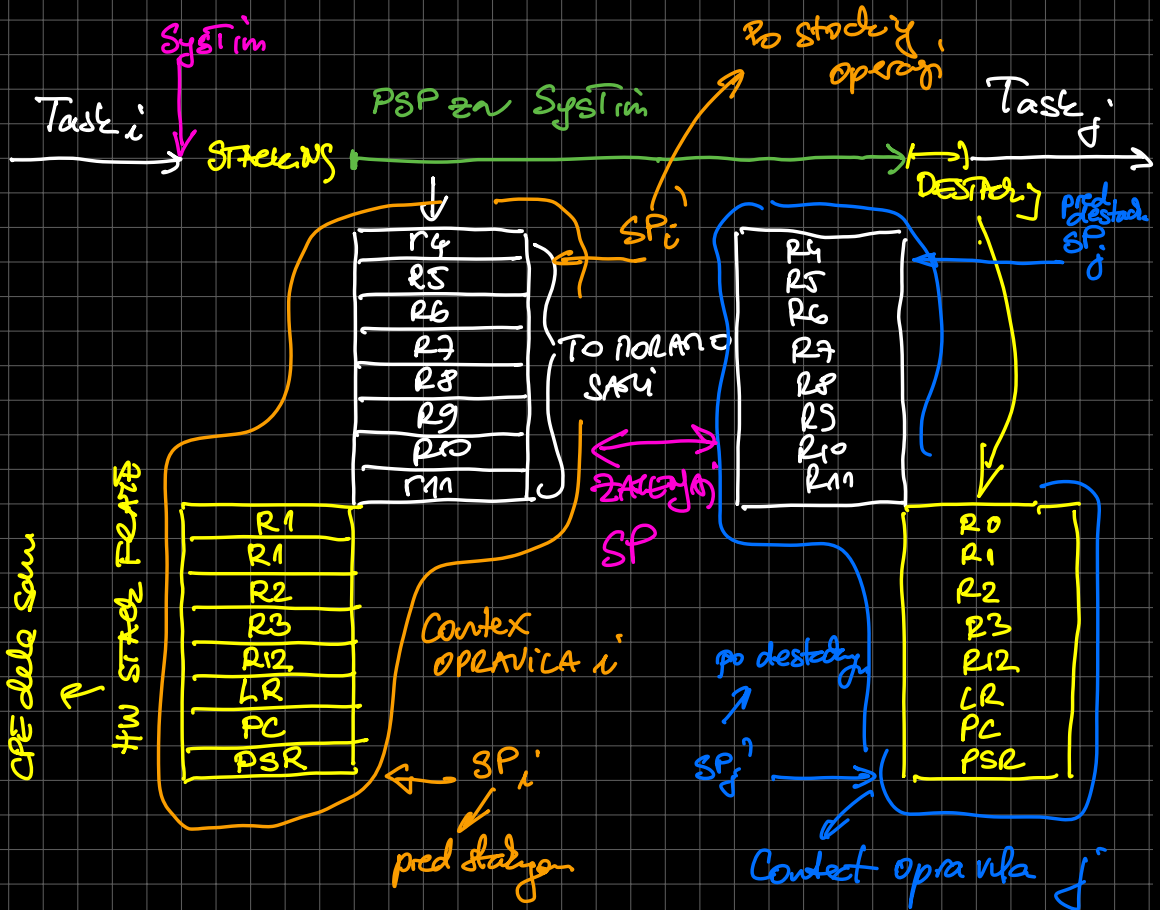
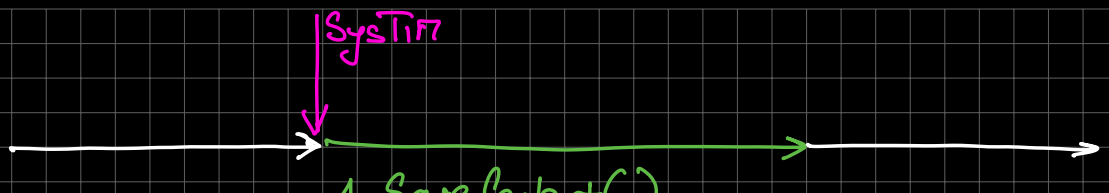
Preklapljanje opravil
(task / context switching)





→ vsako opravilo bo imelo svoj sklad v pomnilnik
↳ 1 kB



SaveContext: na sklad porine registre R4-R11

SwitchContext: $SP_i \leftarrow SP$

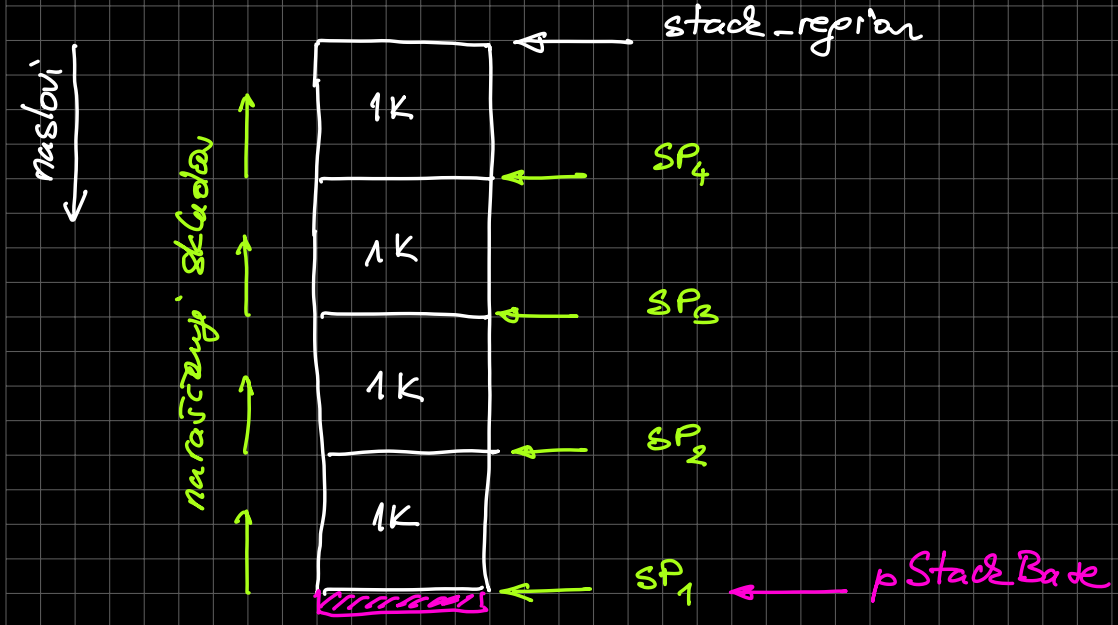
$SP \leftarrow SP_j$

RestoreContext: s sklada obnovi R4-R11

① Implementacija sklada za vsako opravilo

↳ za vsako opravilo 1K zaporednih pomnilniških besed

↳ za vsa opravila (N): N × 1K



uint32_t stack_region [N x 256];

1K

N ← število opravil

Inline assembler

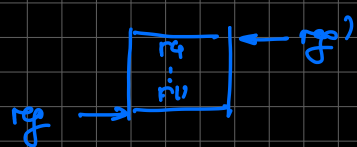
asm volatile (asm_template :
input operands :
output operands
);

```c
static inline void save_context(void) {
    uint32_t reg;

    asm volatile (  "MRS %0, psp\n\t"
            "STMFD %0!, {r4-r11}\n\t"
            "MSR psp, %0\n\t" : "=r" (reg) );
}
```

reg ← psp

psp ← reg



template : tukaj v našem primeru imamo 3 asemblerske ukaze

```c
static inline void restore_context(void) {
    uint32_t reg;

    asm volatile (  "MRS %0, psp\n\t"
            "LDMFD %0!, {r4-r11}\n\t"
            "MSR psp, %0\n\t" : "=r" (reg) );
}
```

Za vsako opravilo bomo hranili:

```c
/*
    Task state structure.
    In our simple scheduler, the task is determined only by its stack pointer and
    the start address (i.e. pointer to a function that implements the task)
    but in general, the structure should contain a task's state variable (flags)
    (e.g. RUNNING, STOPED, ACTIVE, ....)

    Pa3cio Bulic, 9.11.2020
*/
typedef struct {
    void* sp;               // task's stack pointer
    void (*pTask)(void);    // start address of the task
    // int flags;
} task_table_t;
```

skladovni kazalec (trenutna vrednost)

začetek kode opravila

Koda opravila

```c
void Task (void) {
    while(1) {
        ≡
    }
}
```

2.11.2021

Implementacija switch_context():

"zamenja trenutni SP s shranjenim kazalcem novega opravila"

Potrebujem TABELO OPRAVIL

↳ za vsako opravilo hranim:
SP in kazalec na kodo opravila

array struktur task_table_t

1. V tabelo opravil shranim trenutni PSP v SP prekinjenega opravila
2. Iz URNIKA OPRAVIL dobim index novega opravila
3. Iz tabele opravil preberem SP in pišem v register PSP

Branje trenutnega PSP-ja:

```
static inline void* read_process_stack_pointer(void) {
    void* result;
    /* read special register PSP into a general-purpose register (picked by compiler)
      and write it to result: */
    asm volatile ( "MRS %0, PSP\n\t" : "=r" (result) );

    return (result);
}
```

Pisanje v PSP:

```
static inline void write_process_stack_pointer(void* ptr) {

    asm volatile ( "MSR PSP, %0\n\t" : : "r" (ptr) );
}
```

v tabeli opravil za tekoče prekinjeno opravilo shranim PSP

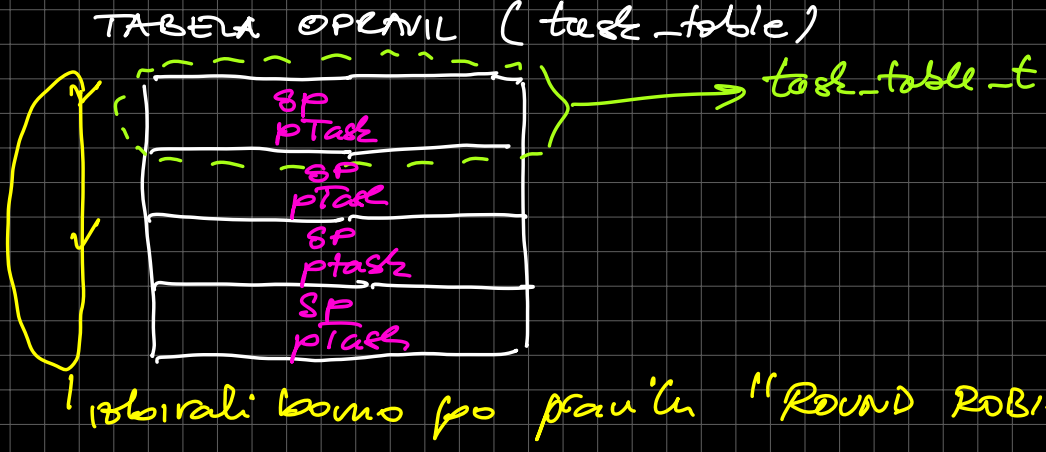
```
void switch_context (void) {
    // save current stack pointer to current task in task_table:
1   task_table[current_task].sp = read_process_stack_pointer();

    // select a new task in a round-robin fashion:
2   current_task ++;
    if (current_task == MAX_TASKS) {
        current_task = 0;
    }

    // select a new psp:
3   write_process_stack_pointer( task_table[current_task].sp );
}
```

index v tabeli opravil

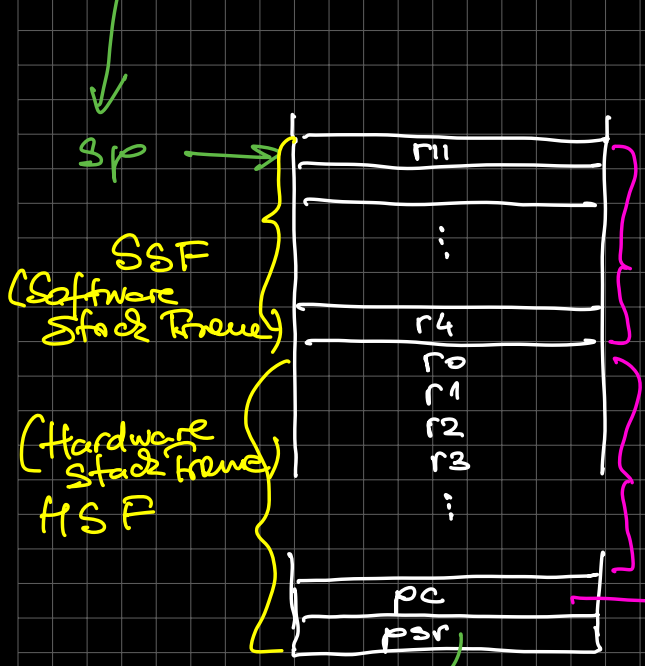
izberem novo opravilo



Vsako opravilo moramo USTVARITI !

1. za vsako opravilo moram imeti nastavljen skladovni kazalec SP v ustrezni strukturi v tabeli.

2. za vsako opravilo moram imeti inicializiran sklad.



Ampak s kakšnimi vrednostmi ???

→ prvi ukaz opravila

⇓

PISE V TABELO

⇓

naslov funkcije, ki implementira opravilo

default stanje tega registra po resetu

```c
void create_task (int task_id, void* task_stack_base_address, void (*pTask)(void)){

    hw_stack_frame_t* ptask_hw_frame;

    /* set the start address of the HW frame structure */
    ptask_hw_frame = (hw_stack_frame_t *) ((uint8_t *)task_stack_base_address - sizeof(hw_stack_frame_t));

    /* populate the HW stack frame with initial values : */
        /* NOTE: HW stack for Task0 (main()) in ignored as main() uses MAIN STACK */
    ptask_hw_frame -> r0 = 0;
    ptask_hw_frame -> r1 = 0;
    ptask_hw_frame -> r2 = 0;
    ptask_hw_frame -> r3 = 0;
    ptask_hw_frame -> r12 = 0;
    ptask_hw_frame -> lr = 0; // in our simple RTOS the tasks never finish so there is no need to delete the task
    ptask_hw_frame -> pc = (uint32_t)pTask;
    ptask_hw_frame -> psr = 0x01000000;

    // make room for SW stack frame and set the task's stack pointer:
    task_table[task_id].sp = (void *)( (uint8_t *)task_stack_base_address
                          - sizeof(hw_stack_frame_t)
                          - sizeof(sw_stack_frame_t) );
}
```

- → Implementacija opravila (`void (*pTask)(void)`)
- ↳ začetek skladu za to opravilo (`void* task_stack_base_address`)
- KAZALEC NA HSF (`hw_stack_frame_t* ptask_hw_frame`)
- (1) ↳ ZAČETEK PODROČJA V MEM, kjer bo sklad tega opravila
- (2) tole bi celo lahko odstranil (r0 … lr = 0)
- → 3. argument funkcije (`pTask`)
- ↳ default vrednost za PSR (`0x01000000`)
- (-32B) (`sizeof(hw_stack_frame_t)`)
- -32B (`sizeof(sw_stack_frame_t)`)

Na SSF in HSF lahko gledamo kot na dve podatkovni strukturi, ki hranita vsaka po 8 32-bitne vrednosti

⇓

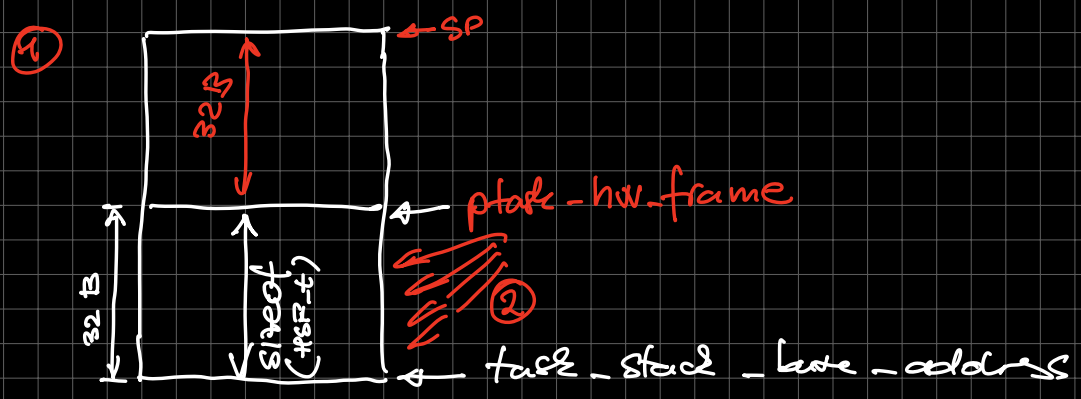
zaradi lažjega dela naredimo ti dve strukturi, ki sta pravzaprav abstrakciji tega dela okvirjev na skladu

```c
/*
    Software Stack Frame structure.

    Pa3cio Bulic, 9.11.2020
*/
typedef struct {
    uint32_t r4;
    uint32_t r5;
    uint32_t r6;
    uint32_t r7;
    uint32_t r8;
    uint32_t r9;
    uint32_t r10;
    uint32_t r11;
} sw_stack_frame_t;
```

```c
/*
    Hardware Stack Frame structure.

    Pa3cio Bulic, 9.11.2020
*/
typedef struct {
    uint32_t r0;
    uint32_t r1;
    uint32_t r2;
    uint32_t r3;
    uint32_t r12;
    uint32_t lr;
    uint32_t pc;
    uint32_t psr;
} hw_stack_frame_t;
```



Zgornjo funkcijo create_task() moramo klicati še vsako opravilo posebej:

```
void init_tasks (void) {

    int i;

        /* Task schedule
       sets tasks in the schedule: */
    task_table[0].pTask = &idle; /* this value will be overwritten after first SysTick exception....*/
        task_table[1].pTask = &task1;
    task_table[2].pTask = &idle;
    task_table[3].pTask = &task1;
    task_table[4].pTask = &idle;
        task_table[5].pTask = &task3;
        task_table[6].pTask = &task1;
    task_table[7].pTask = &idle;
    task_table[8].pTask = &task1;
    task_table[9].pTask = &task2;

    /* Create tasks....*/
    for (i = 0; i < MAX_TASKS; i++) {
      create_task (i, (uint8_t *)pTasksStackBase - i*sizeof(uint32_t)*TASKS_STACK_SIZE, task_table[i].pTask);
    }

        /* PSP has not been set until now, so set PSP to point to
       the top of stack of the first interrupted task (Task 0): */
    write_process_stack_pointer(task_table[0].sp);
}
```

nastavimo PC v tabeli opravil

USTVARIMO VSA OPRAVILA

1 KB