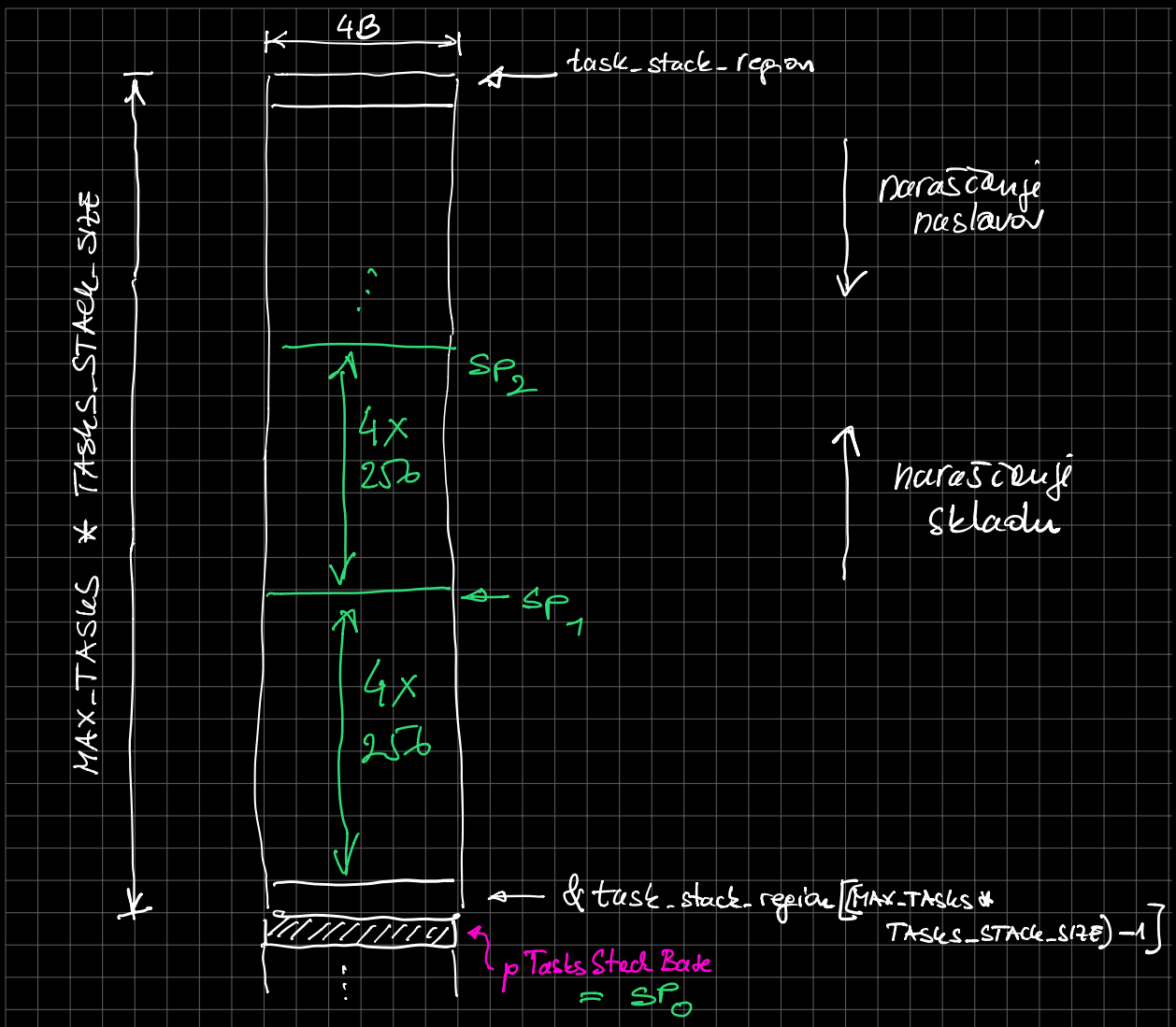
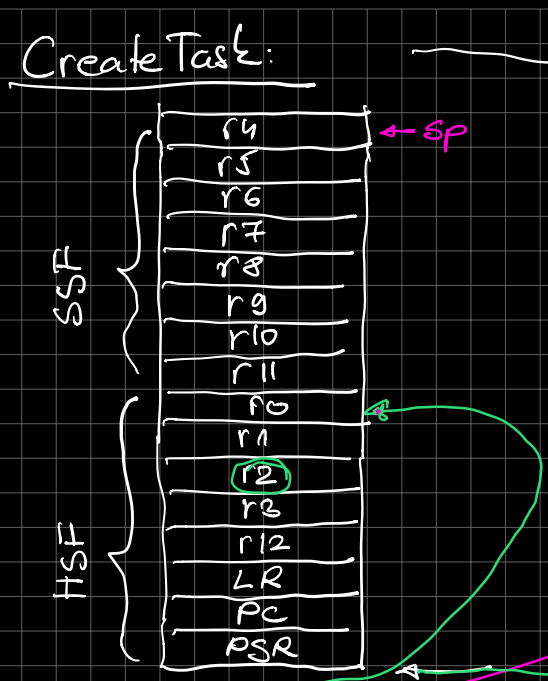



```c
/*
    Software Stack Frame structure.

    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;
```




```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) );
}
```

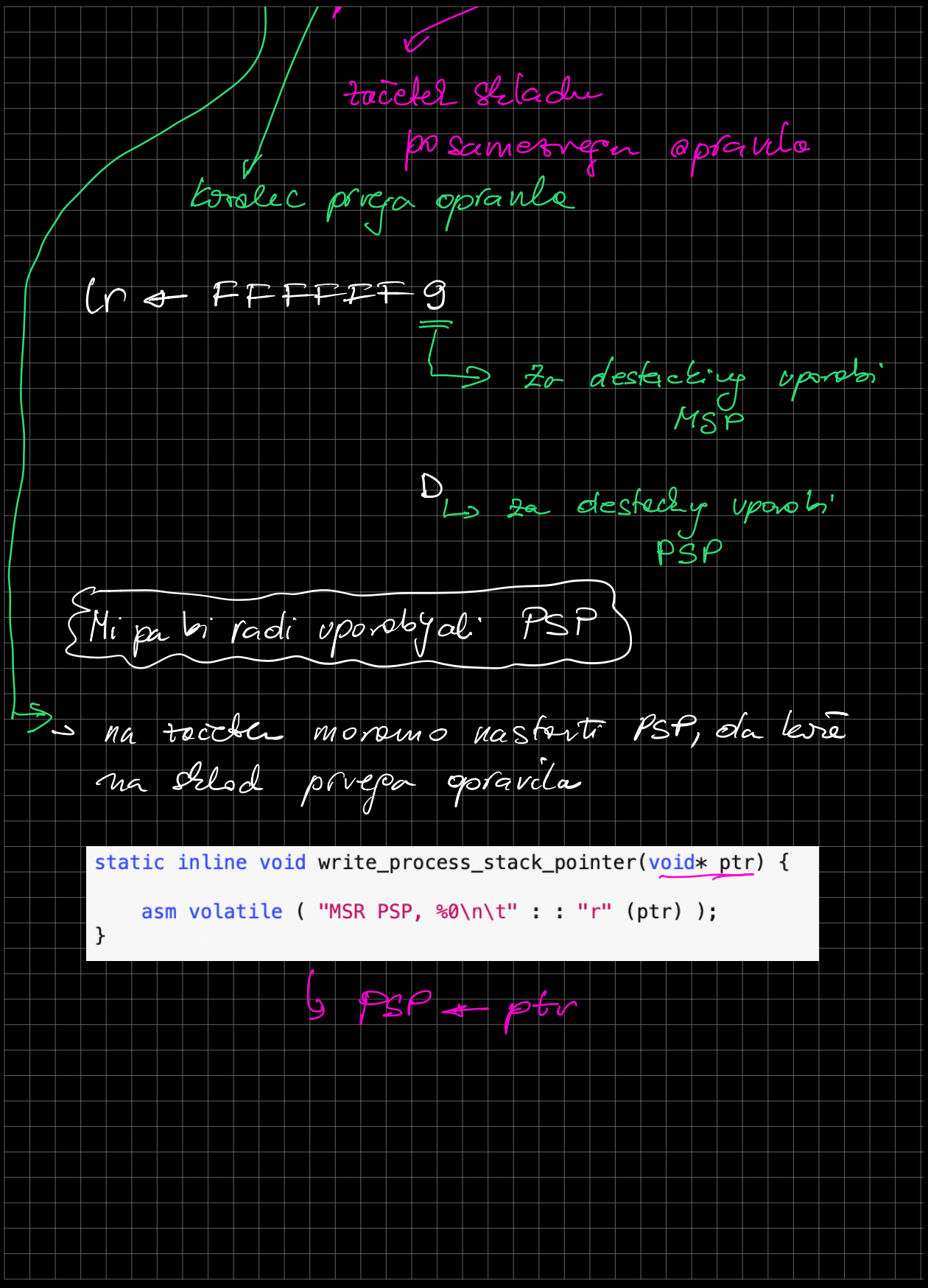
```c
void init_tasks (void) {

    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);
}
```


začetek sklada
za shranjevanje opravila
konec prejga opravila
lr ← FFFFFFF9
↳ za detekcijo uporabi MSP
↳ za detekcijo uporabi PSP
Mi pa bi radi uporabljali PSP
↳ na začetku moramo nastaviti PSP, da kaže na sklad prejga opravila
static inline void write_process_stack_pointer(void* ptr) {
asm volatile ("MSR PSP, %0\n\t" : : "r" (ptr));
}
↳ PSP ← ptr

V prekinitveno-servisni podprogram moram dodati 3 funkcije:

1. save_context

→ ta shrani registre r4-r11 na sklad

```
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) );
}
```

%0 ← PSP

r4-r11 shrani na sklad pri čemer za naslavljanje sklada uporabljam reg %0

PSP ← %0

2. switch_context: Zamenja skladovne kazalce

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

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

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

// IZBEREM NASLEDNJE OPRAVILO

to piše v PSP

shrani PSP v tabelo opravil na mesto prekinjenega opravila

Skladovni kazalec naslednjega opravila piši v PSP

```c
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);
}
```

%0 ← PSP
vrne %0

3. restore_context :

→ s sklada prebarem (obnovim) software stack frame (r4-r11)

```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) );
}
```

%0 ← PSP

s sklada bare registre r4-r11 in za naslavljanje sklada uporabljam %0

na koncu moram PSP ← %0

Izvajanje kode

- Ob vstopu v main():

  MSP = 0x2007FFF8

  PSP = 0x0 → neinicializirano

  nastavimo pa v init_tasks()

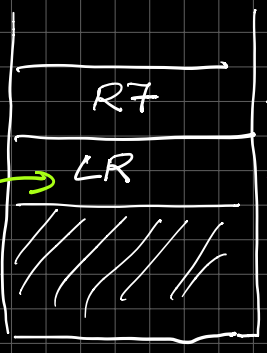
Ko izvedemo init_tasks():

PSP = 2000 22c4

- Ko izvedem save_context:

  PSP = 2000 22a4

Ob vstopu v PSP mi PSP na sklad porine najprej LR potem pa R7



FFFFFFF9 → desetiški z vrednostjo MSP

→ nočem je!! Zato ker jaz hočem desetiški z vrednostjo PSP-ja

Namesto tega vrnem FFFFFFFD

Kaj?

1. preberi kam MSP v neko spremenljivko

```
static inline void* read_main_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, MSP\n\t" : "=r" (result) );

    return (result);
}
```

(uint32_t) msp

2. msp = msp + 1

3. M[msp] ← FFFFFFFD