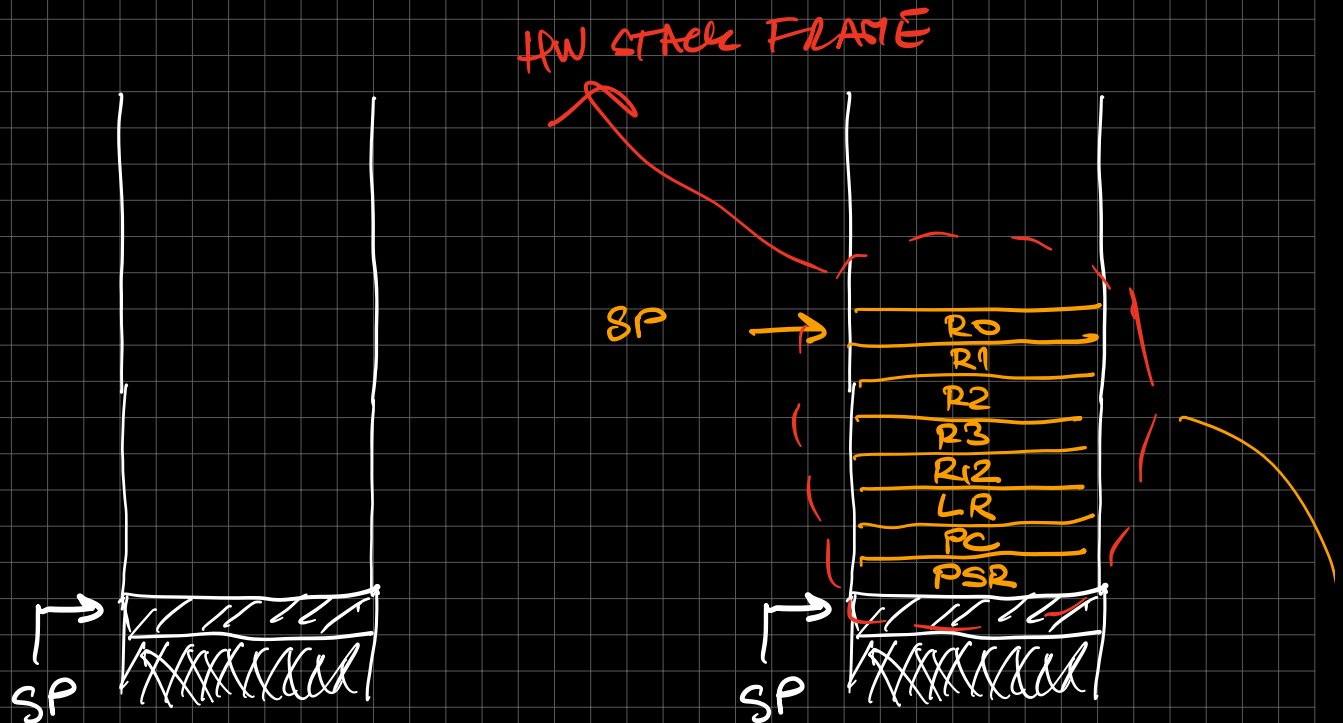
Task switching

Pred prekinitvijo:

Stacky operacij:



LR ← 0xFFFFFFF9

Sledi iz naslova PSP

Ob izstopu iz PSP (bx LR => pc ← LR)

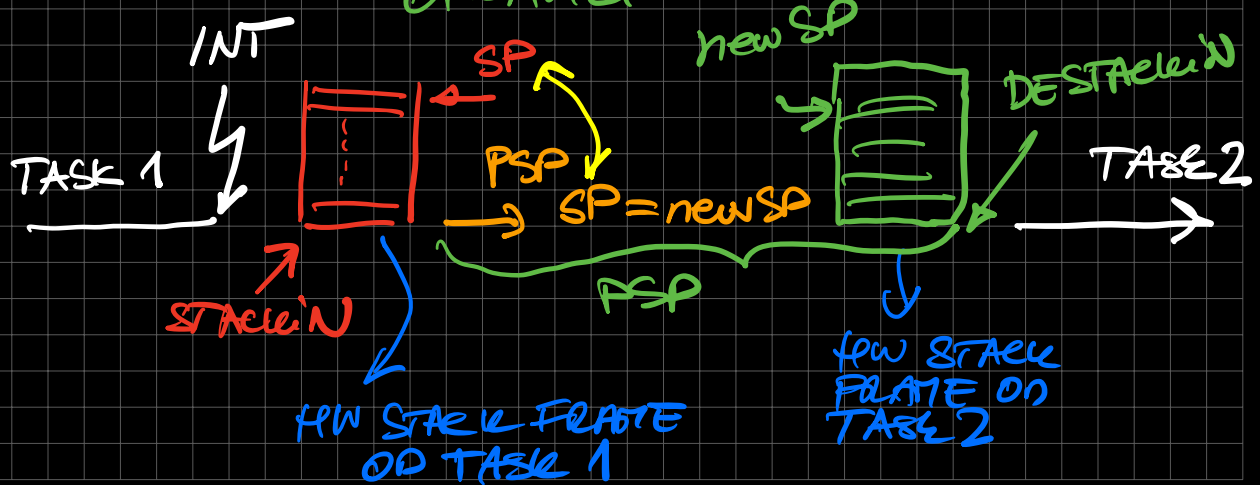
se zgodi DESTACKIRAJ OPERACIJA:
pobere iz PSP registre s sklada

Če je ob destacky operaciji ima SP enako vrednost kot ob stacky operaciji, se vrnemo v program, ki smo ga prekinili!

Ha! Če pa SP ob DE stacky operaciji nima enake vrednosti, kot pa stacky operaciji, potem se vrnemo nekam drugam

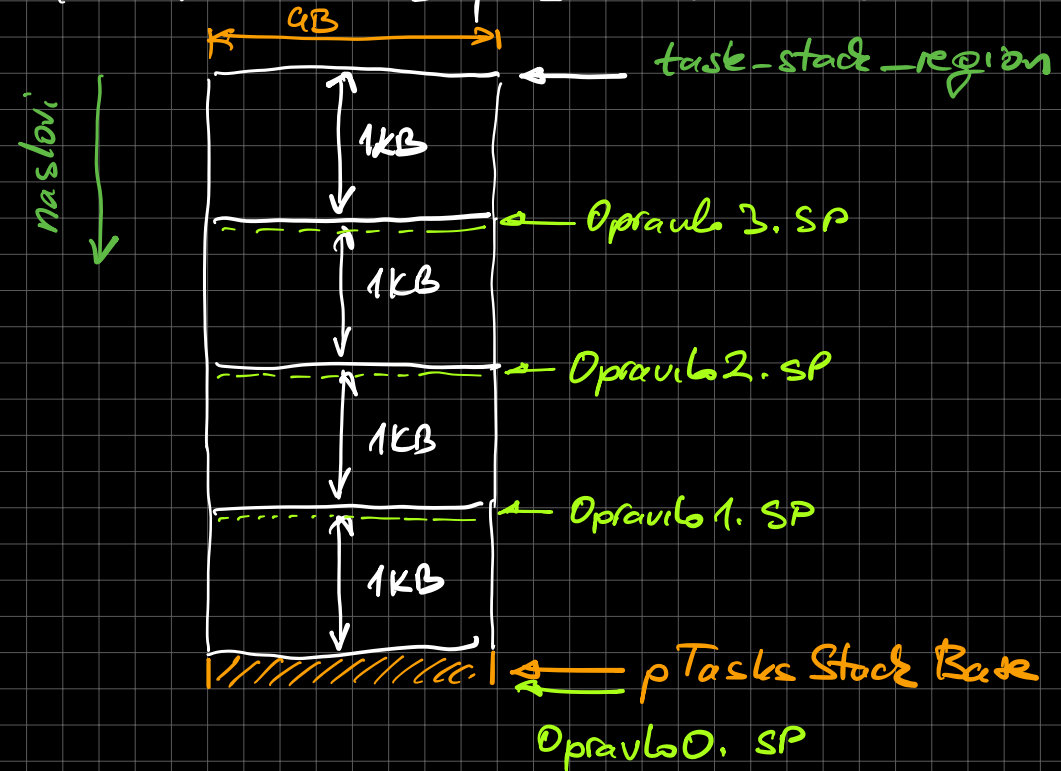
IDEJA: predem izvedemo destacky operacijo, zamenjamo SP tako, da bo kazal na nek drug HW STACK FRAME

TAKO SE BOMO VRNILI DRUGAM IN DE-FACTO ZAMENJALI OPRAVILI

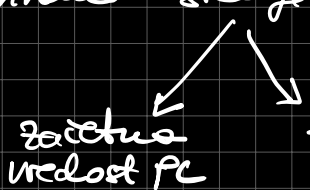


1. Vsak TASK mora imeti lasten Sklad

Predpostavimo 4 opravila. Vsako opravilo ima 1KB od. 256 x 4B prostora za sklad



Za vsako opravilo bomo imeli eno strukturo, ki bo hranila "shranjeno" opravilo → ne tisto, kar se ne hrani na HW St. Frame



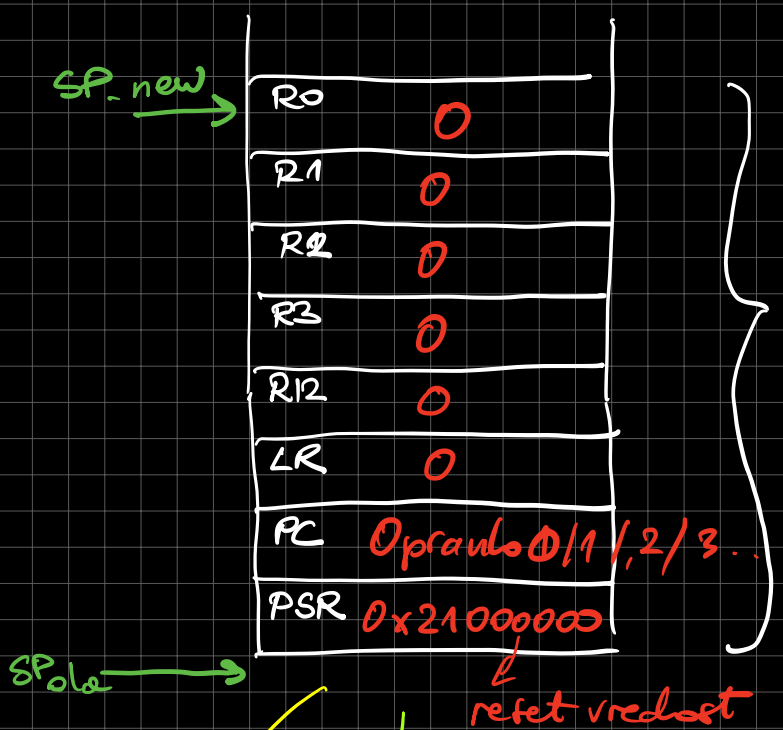

```
/*
    Task state structure.
    In our simple scheduler, the task is determined only by its stack pointer and
    the start address (i.e. pointer to a function that implements the task)
    but in general, the structure should contain a task's state variable (flags)
    (e.g. RUNNING, STOPED, ACTIVE,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

    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;
```

2. Za vsako opravilo moramo na skladu sami ustvariti "začetni" zapis HW stack Frame-a

↓

Ustvarimo zapolnitev prvih 8 32-bitnih lokacij na skladu:



↳ s tem ustvarimo opravilo

Da mi bo lažje tole zapisat na skladu, bom narisal abstrakcijo tega zapisa s ena podatkovna struktura:

```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;
```

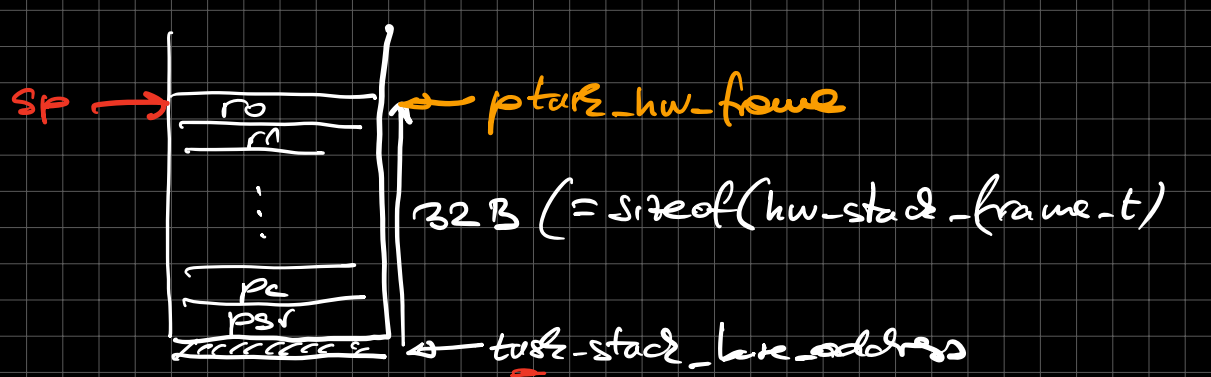
Za ustvarjanje opravila (= ustvarjanje začetnega zapisa na skladu):

```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)
                        - [illegible] );
}
```



Sedaj zoomamo funkcijo pokličemo kolikokrat bolle po opravil:

```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);
}
```

Opravilo 0

Opravilo 1

Opravilo 2

Opravilo 3

→ vstavimo 4 opravila

1 kB

→ naslednji

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

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