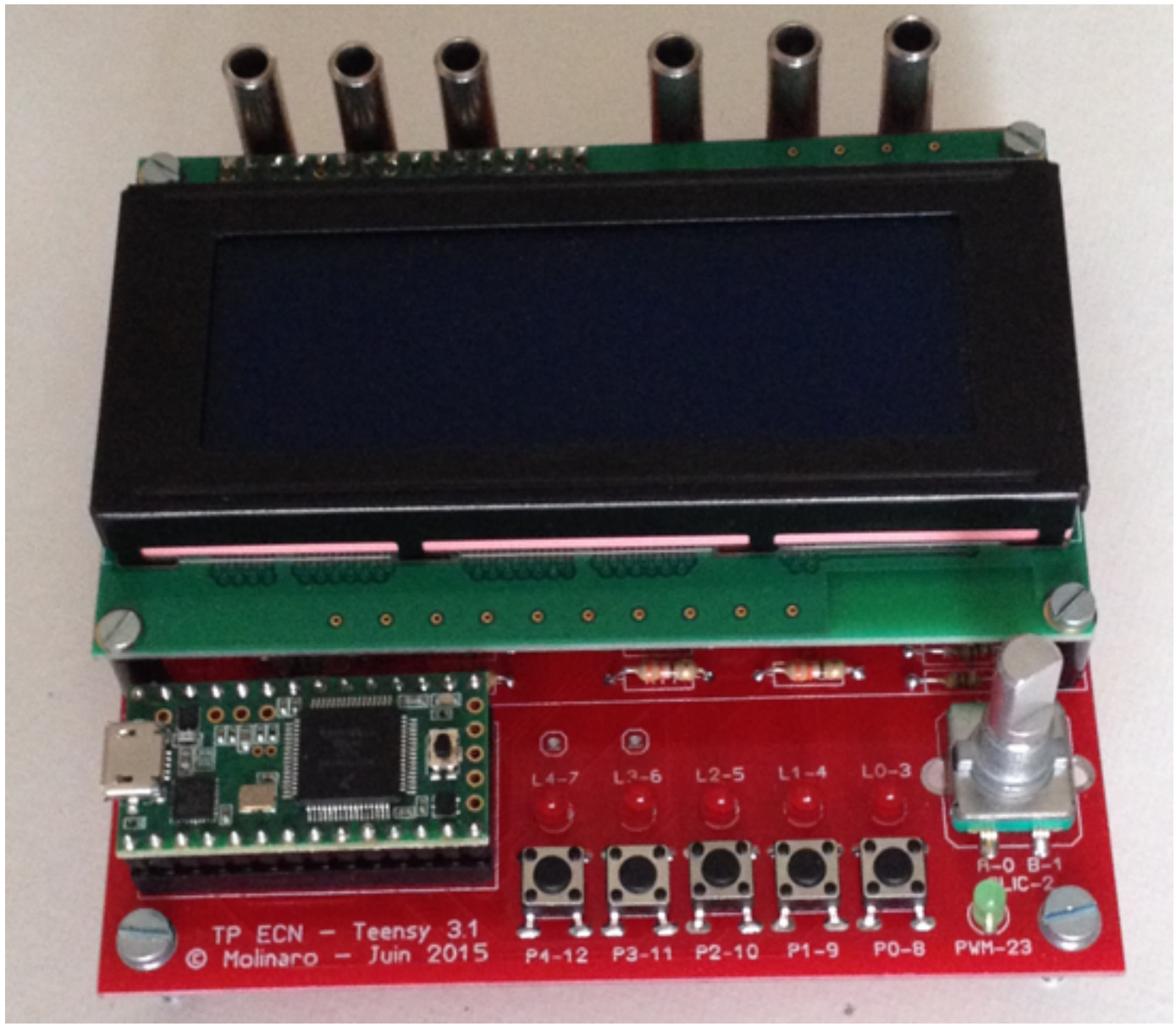
Temps Réel


L4-7
L3-6
L2-5
L1-4
L0-3
R-0 B-1
LIC-2
TP ECN - Teensy 3.1
© Molinaro - Juin 2015
P4-12
P3-11
P2-10
P1-9
P0-8
PWM-23

But de cette partie

Objectif :

- *modifier le programme précédent pour utiliser les deux pointeurs de pile du processeur.*

Travail à faire :

Réaliser un programme qui incrémente quatre variables globales :

- dans une routine d'interruption périodique ;
- dans la routine `loop`, par l'intermédiaire de quatre appels de service ;

Au bout de 5 secondes, la routine d'interruption périodique est désinstallée et les quatre variables sont affichées.

Les pointeurs de pile du processeur Cortex-M4

Par défaut, le processeur Cortex-M4 utilise un seul pointeur de pile, qui est utilisé aussi bien en le *Thread Mode* (mode d'exécution des *threads* dans un exécutif, des routines `setup` et `loop`) que dans les routines d'interruption.

Le but de ce cours est d'écrire un exécutif : une pile distincte pour chaque *thread*, une pile partagée entre tous les services de l'exécutif.

Le but du programme `09` est d'effectuer un premier pas dans cette direction en allouant une pile pour le *Thread Mode*, et une seconde pile pour les routines d'interruption.

Une caractéristique du processeur Cortex-M4 est qu'il facilite l'utilisation de ces deux piles par des registres de contrôle particuliers.

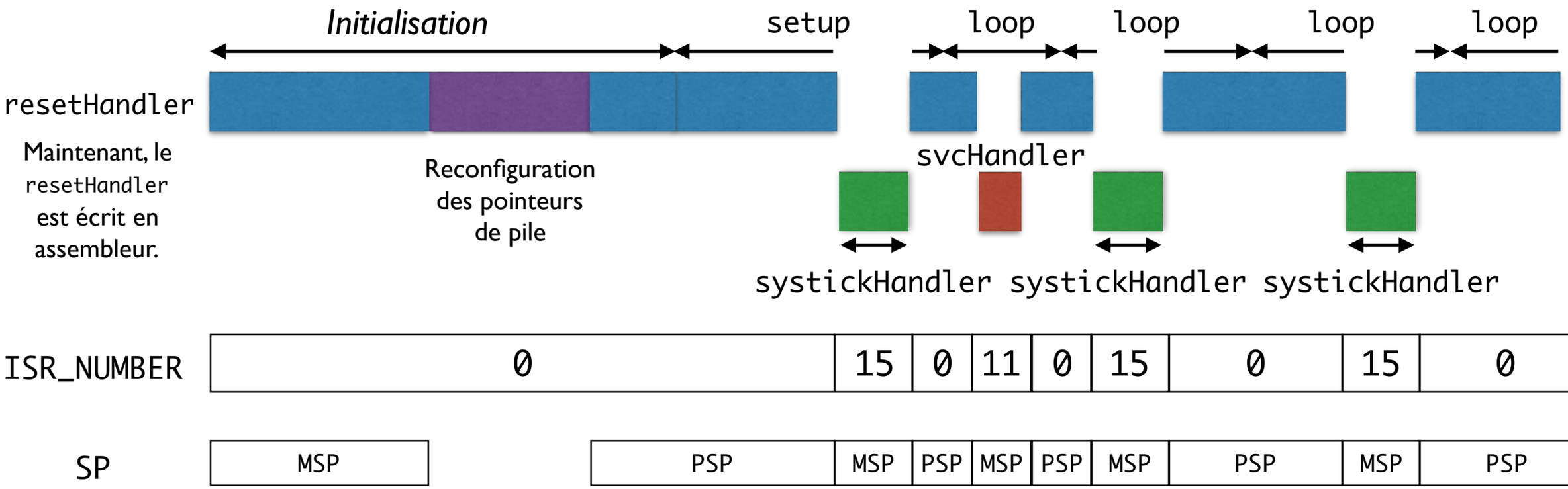
SP, MSP **et** PSP

Le registre R13 (SP) est en fait un faux registre, il désigne en fait soit MSP, soit PSP.

Par défaut, SP désigne MSP. C'est donc MSP qui a été utilisé dans les programmes 01 à 08.

À partir de ce programme (09), les deux pointeurs de pile seront utilisés.

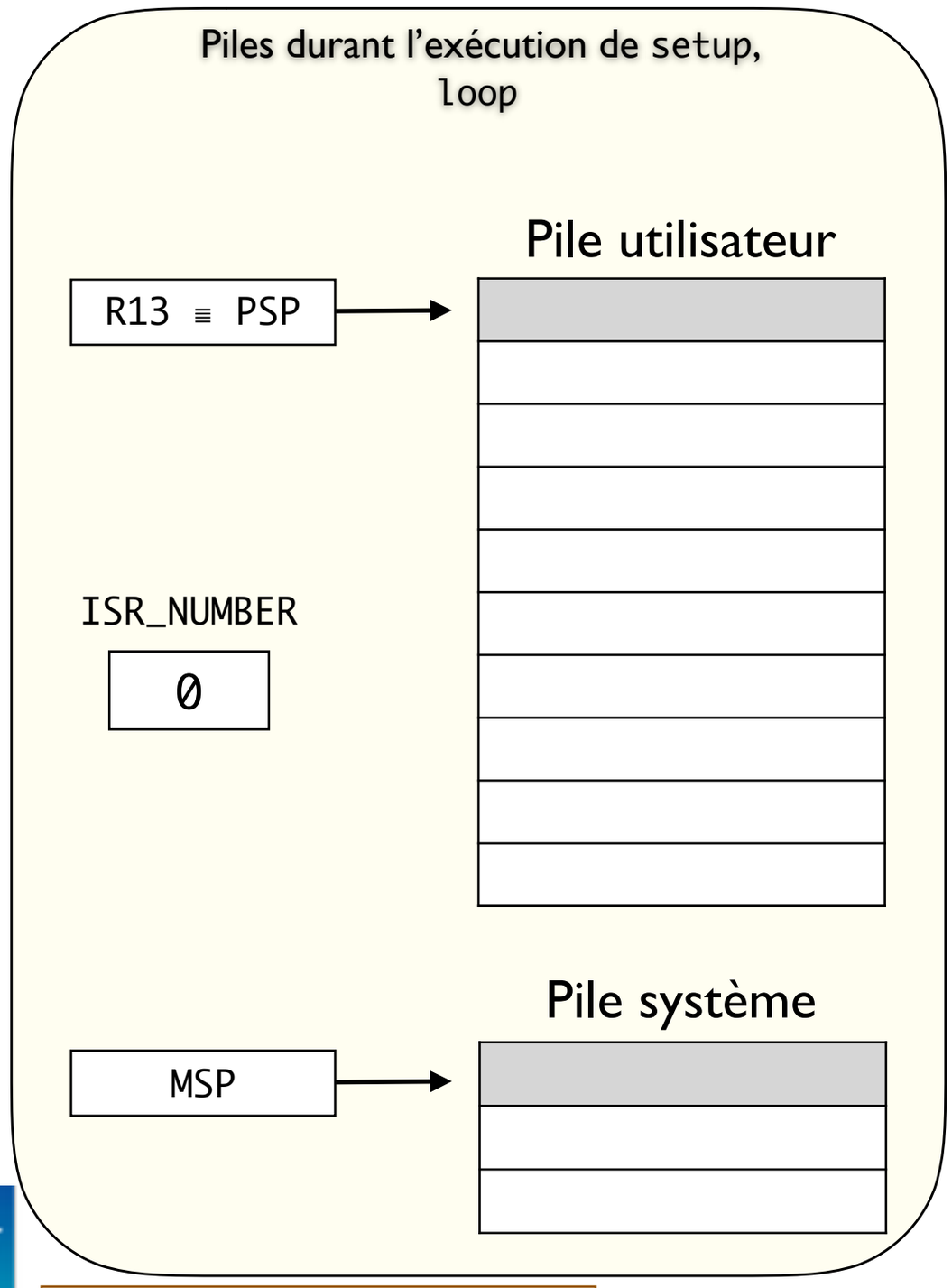
Exécution des programmes (valable pour les programmes `09` à `11`)

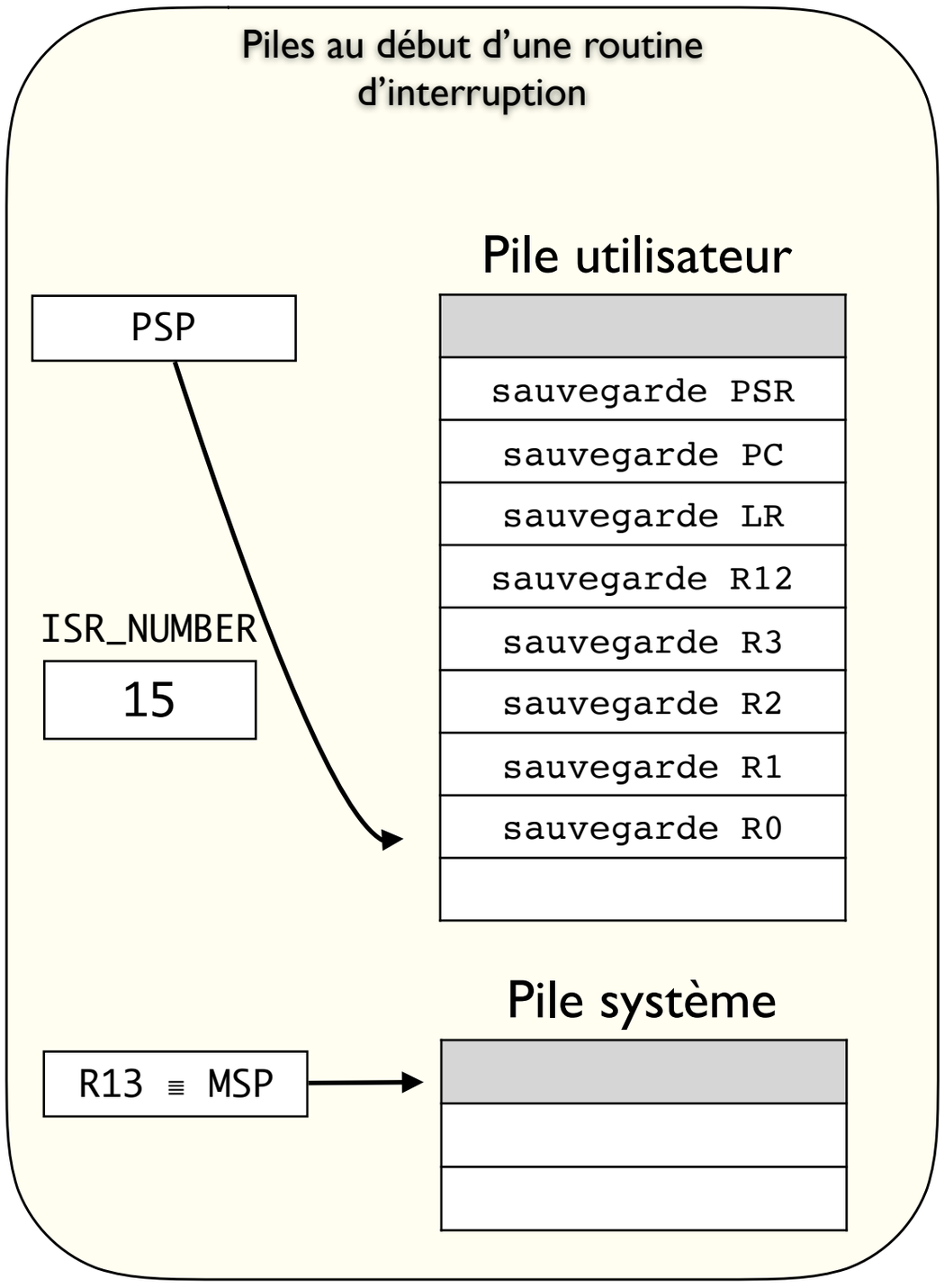


Dans la suite :

- MSP (*Main Stack Pointer*) sera appelé *pointeur de pile système* ;
- PSP (*Program Stack Pointer*) sera appelé *pointeur de pile utilisateur*.

Interruption et pointeurs de pile (valable pour les programmes `09` à `11`)





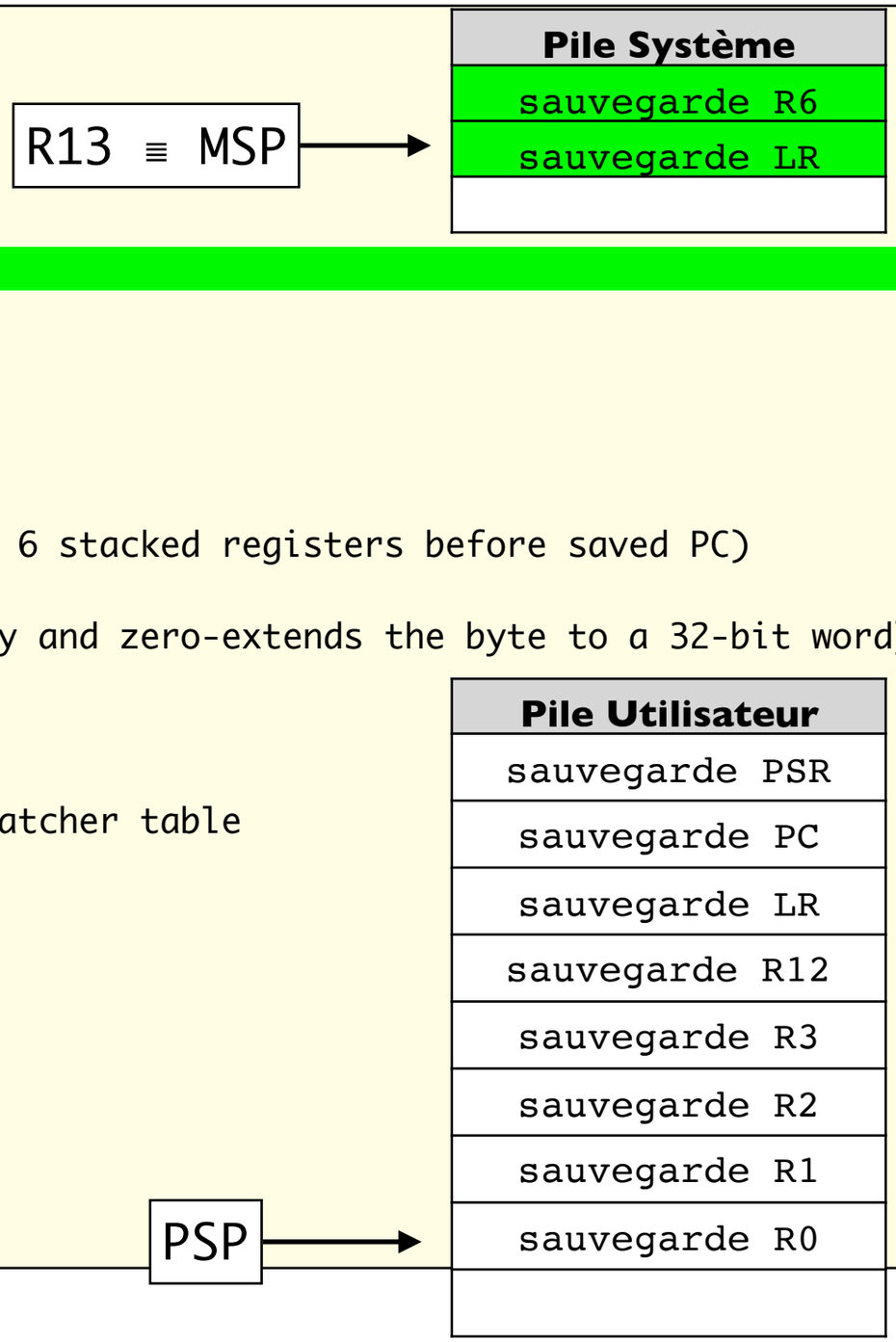
Écriture du `ResetHandler` en assembleur

```
.lcomm handlerStack, 1024 @ Pile utilisateur : 1Kio

@-------------------------------------------------------------------------------*

  .global resetHandler
  .type resetHandler, %function

resetHandler:
@--------- Init micro controller
  bl configure_microcontroller @ Appel routine écrite en C
@--------- Reconfiguration des pointeurs de pile
@---1- set PSP
  ldr r0, =handlerStack + 1024
  msr psp, r0
  isb
@---2- Set CONTROL register
  mov r0, #3
  msr control, r0
@--------- Execute user program
  bl setup
infiniteLoop:
  bl loop
  b  infiniteLoop
```

Comment est écrit le svcHandler (09 à 11)

Vous n'avez pas à l'écrire, il est fourni à la fin du fichier svc-handler.s (sur le serveur pédagogique).

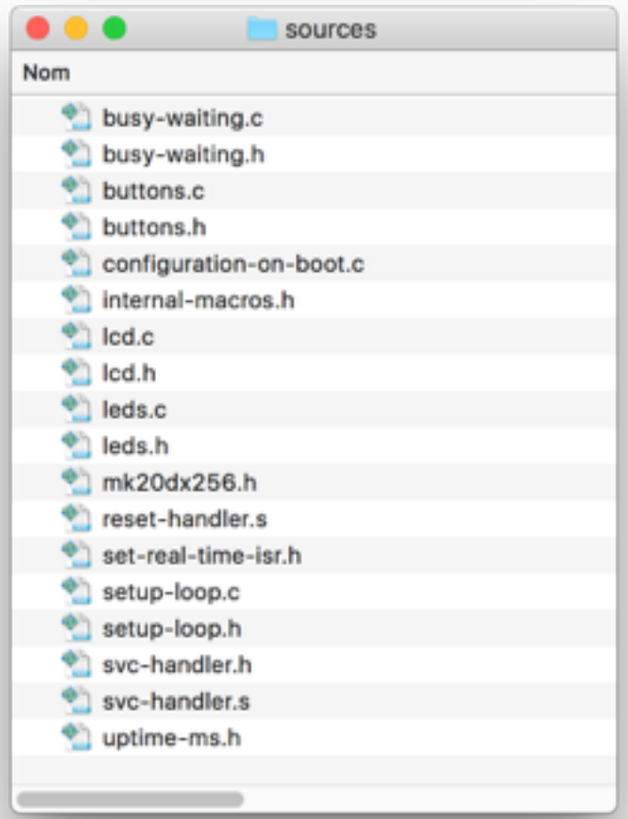
```
  .global svcHandler
  .type svcHandler, %function

svcHandler:
@--- Save preserved registers
  push  {r6, lr}
@--- r12 <- address of dispatcher
  ldr   r6, =__svc_dispatcher_table
  mov   r12, r6
@--- r6 <- Address of SVC instruction
  mrs   r6, psp       @ R6 <- thread SP
  ldr   r6, [r6, #24] @ R6 <- Address of instruction after SVC (24 : 6 stacked registers before saved PC)
  sub   r6, #2        @ R6 <- Address of SVC instruction
@--- r6 <- bits 0-7 of SVC instruction (LDRB loads a byte from memory and zero-extends the byte to a 32-bit word)
  ldrb  r6, [r6]
@--- r6 <- address of routine to call
  lsl   r6, #2     @ Multiply by 4, entries are 4 bytes words
  add   r6, r12    @ R6 <- Address of selected service entry in dispatcher table
@--- r6 <- address of routine to call
  ldr   r6, [r6]   @ R6 <- Address of selected service
@--- Call svc_ routine
  blx   r6
@--- Write return value in stack
  mrs   r6, psp       @ R6 <- thread SP
  stm   r6!, {r0, r1} @ Write r0, r1 to saved registers
@--- Return from interrupt
  pop   {r6, pc}
```



Travail à faire

Première étape :

- dupliquer le programme `08` précédent et le renommer ;
- récupérer sur le serveur pédagogique l'archive `09-sources.tbz` qui contient :
  - `svc-handler.s` ;
  - `reset-handler.s` ;
  - `configuration-on-boot.c` ;
- compléter le fichier `svc-handler.s` avec les appels des services.



Résultat attendu

Les quatre variables globales sont incrémentées de manière atomique, on obtient à chaque fois quatre valeurs identiques. Ces valeurs peuvent changer d'une exécution à l'autre. L'affichage est le même que pour le programme précédent `08`.