La mémoire et la gestion de données

Prof. Francesco Mondada
Dr. Frank Bonnet
IEM - STI - EPFL

Exemple https://www.youtube.com/watch?v=gp_D8r-2hwk

Ariane 5

Exemple (Wikipedia)

“En effet, l'accélération horizontale maximum produite par Ariane 4 donnait une valeur décimale d'environ 64. La valeur d'accélération horizontale de la fusée étant traitée dans un registre mémoire à 8 bits, …. Mais Ariane 5 était bien plus puissante et brutale : son accélération pouvait atteindre la valeur 300, qui … nécessite un registre à 9 bits. Ainsi, la variable codée sur 8 bits a connu un dépassement de capacité, … De ce dépassement de capacité a résulté une valeur absurde dans la variable, ne correspondant pas à la réalité. Par effet domino, le logiciel décida de l'autodestruction de la fusée …”

Résultat -> **500 millions de dollars de perdu à cause d’un dépassement de capacité d’un bit**

Introduction

Les ressources utilisées pour le traitement de données (calcul ou autre) dépendent fortement du type de variables utilisées, des opérations utilisées et du code généré par le compilateur.

Exemple de gestion de "variable": Dimensionnement d'un timer (gestion du TMR7):

```
#define TIMER_CLOCK 84000000                                    // APB1 clock

#define PRESCALER_TIM7 (TIMER_CLOCK/100000) // timer freq.: 100kHz

#define COUNTER_MAX_TIM7 100000        // timer max counter 1 sec
```

Même sur le STM**32**F4...les compteurs des timers sont codés sur un registre de taille **16 bits** -> overflow (16 bits = 0 -> 65535):

TIM6&TIM7 introduction

The basic timers TIM6 and TIM7 consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used as generic timers for time-base generation but they are also specifically used to drive the digital-to-analog converter (DAC). In fact, the timers are internally connected to the DAC and are able to drive it through their trigger outputs.

The timers are completely independent, and do not share any resources.

From the Reference Manual

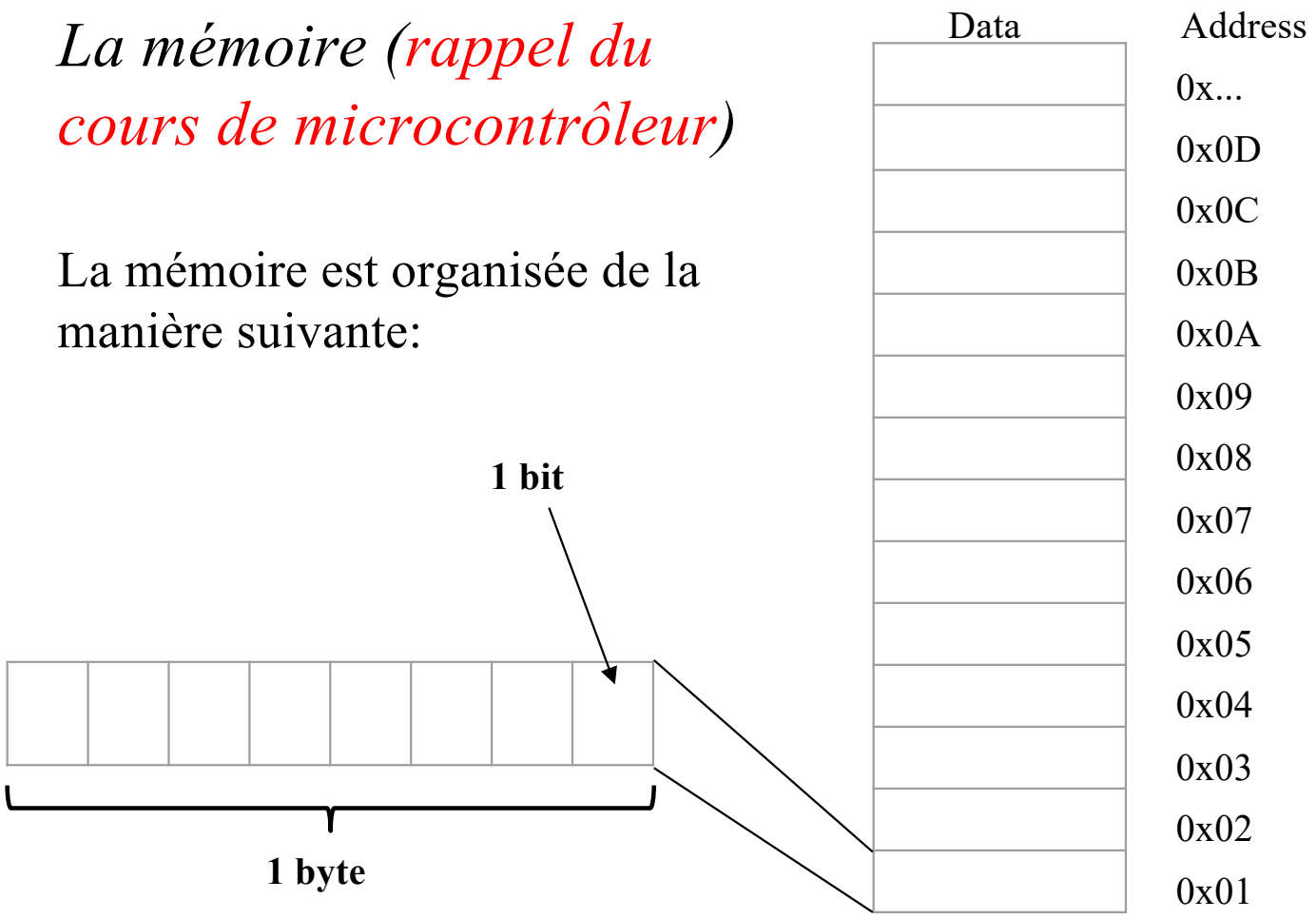
La mémoire

Sur vos laptop/PC, nous avons en général une très grande quantité de mémoire stockée sur des disques/SSD, jusqu'à plusieurs TBytes.

Sur les systèmes embarqués, comme le STM32F4, la quantité de mémoire est très limitée! On est plutôt dans l'ordre des 100 KBytes au MBytes. Il faut donc toujours penser à optimiser la taille que va prendre votre code, aussi bien en terme de mémoire flash que de mémoire RAM, ce qui implique par exemple de bien gérer les types des variables.

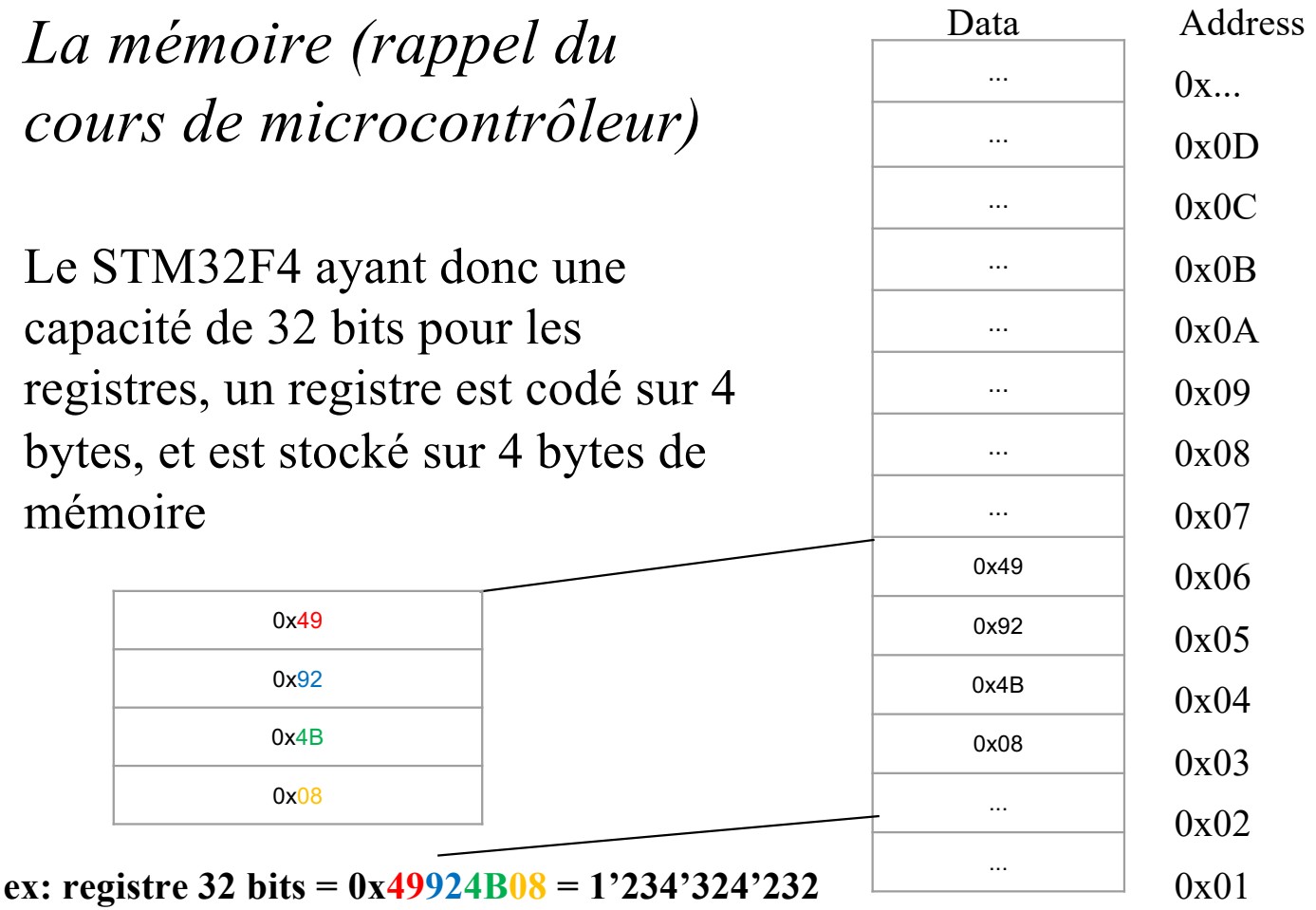
La mémoire (rappel du cours de microcontrôleur)

La mémoire est organisée de la manière suivante:



La mémoire (rappel du cours de microcontrôleur)

Le STM32F4 ayant donc une capacité de 32 bits pour les registres, un registre est codé sur 4 bytes, et est stocké sur 4 bytes de mémoire

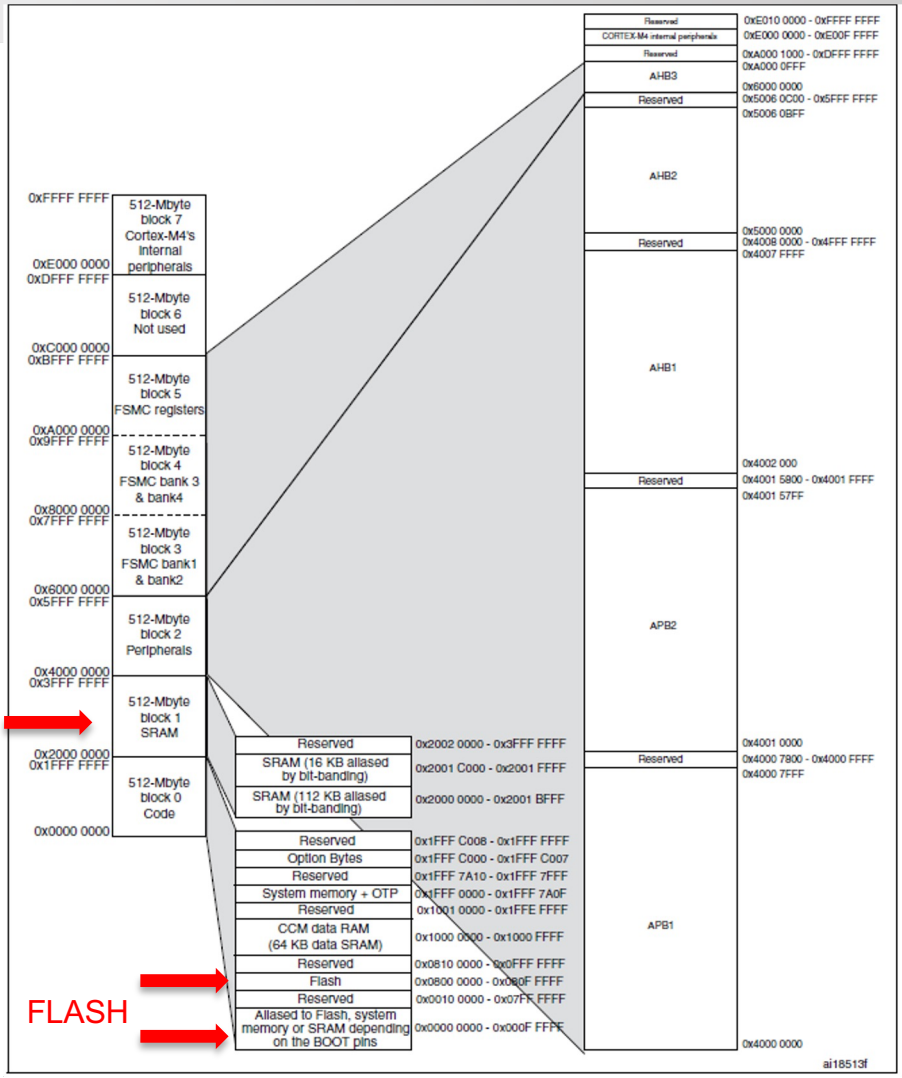


ex: registre 32 bits = 0x49924B08 = 1'234'324'232

La cartographie de la mémoire (Memory map)

Les zones de mémoire sont organisées d'une certaine manière qui est propre à chaque plateforme, et le compilateur doit connaître cette organisation pour savoir quelle adresse correspond à quel endroit/type de mémoire.

Pour le programmeur, cette information est contenue dans ce qui s'appelle le Memory Map du microcontrôleur. (cf datasheet du STM32F4 Ch. 4).



RAM

FLASH

STM32F407 datasheet

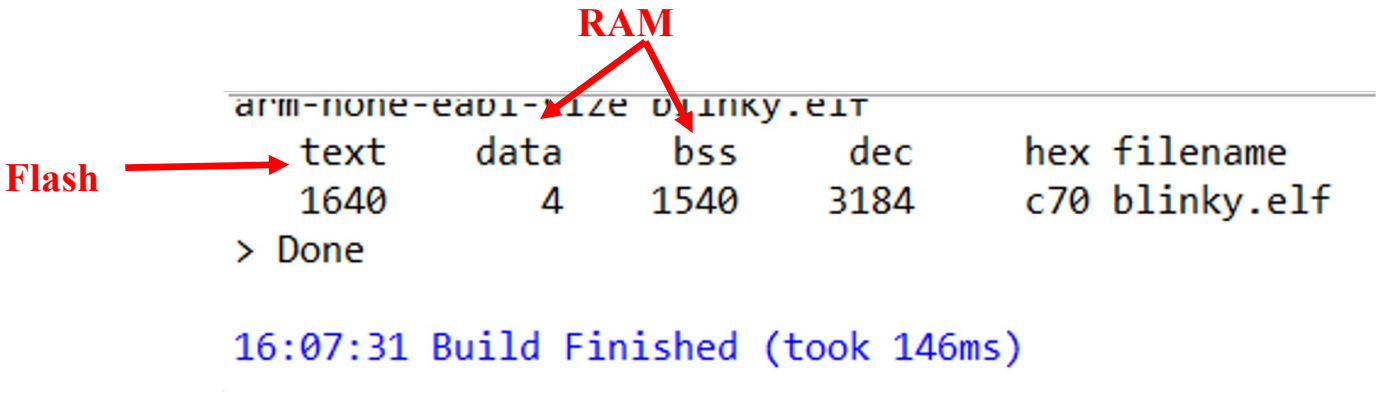
La cartographie de la mémoire (Memory map)

Il y a trois grandes catégories de mémoire:

- La mémoire programme (non-volatile, FLASH)
- La mémoire des données (volatile, RAM)
- Les registres (volatile, RAM)

La cartographie de la mémoire (Memory map)

C'est lors de la compilation que, en connaissant la plateforme sur laquelle on va faire fonctionner le code, le compilateur (avec l'aide du linker) va déterminer la taille et l'emplacement de la mémoire qui seront utilisés lors de l'exécution de notre code sur cette plateforme (voir TP1).



La cartographie de la mémoire (Memory map)

Rappel TP1:

- **text** : Segment pour le programme en mémoire Flash [Flash].
- **data** : Segment pour les variables globales et variables statiques initialisées non-nulles [RAM].
- **bss** : Segment pour les variables globales et variables statiques non-initialisées ou initialisées nulles [RAM].

La cartographie de la mémoire (Memory map)

Il faut compter encore comme segments de mémoire:

- **stack (pile)**: La pile est un emplacement de la mémoire utilisée lors de l'exécution de certaines tâches, notamment pour stocker les variables locales [RAM].
- **heap (tas)**: Le tas est un emplacement de la mémoire utilisé pour les allocation dynamiques [RAM].
- **rodata**: Les données constantes qui seront inchangées [Flash].
- **isr_vector**: La table des vecteurs d'interruption [Flash]

La cartographie de la mémoire (Memory map)

La Memory Map concernant le stockage des données est indiquée au compilateur dans un des fichiers appelé par le linker lors de la compilation (STM32F407VGTx_FLASH.ld):

```
MEMORY
{
        RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 128K
        CCMRAM (rw)      : ORIGIN = 0x10000000, LENGTH = 64K
        FLASH (rx)      : ORIGIN = 0x8000000, LENGTH = 1024K
}
```

Il est donc possible de réserver certaines parties de Flash ou de RAM pour des buts précis, mais, attention, il faut que cela reste cohérent avec la Memory Map du microcontrôleur, qui elle est donnée!

La cartographie de la mémoire (Memory map)

Dans le fichier STM32F407VGTx_FLASH.ld, il est aussi spécifié où vont être stockées les différents types de données (data, bss), sur la RAM ou la CCRAM.

En effet, sur le STM32F4, il y a deux types de RAM (Random Access Memory):

- **SRAM** : une RAM standard assez grosse (128 KB) mais peu rapide d'accès.
- **CCM RAM**: Une RAM deux fois plus petite en taille que la SRAM (64 KB) mais qui est rapide d'accès.

La cartographie de la mémoire (Memory map)

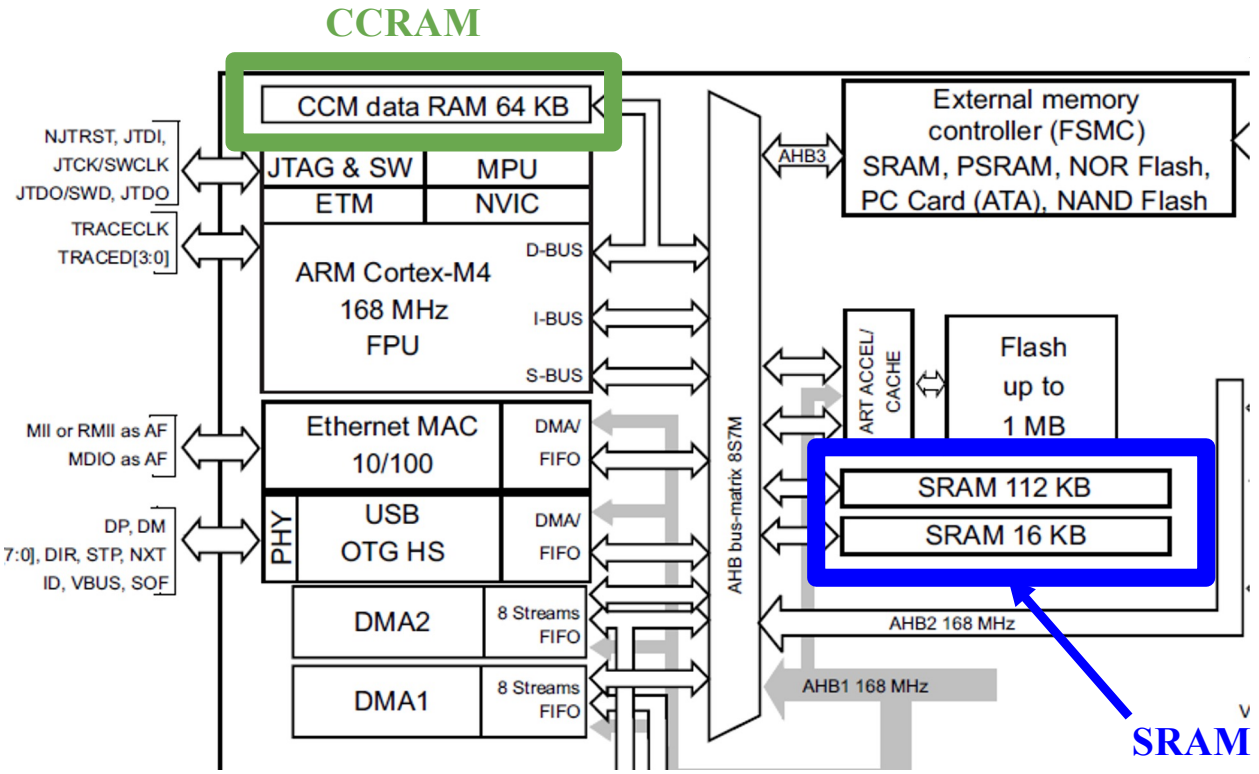


Schéma du microcontrôleur STM32F407 (Datasheet)

La cartographie de la mémoire (Memory map)

Dans le fichier STM32F407VGTx_FLASH.ld, dans notre cas, le segment bss et data sont stockés en SRAM, mais cela est configurable.

```
.bss :
{
  /* This is used by the startup in order to initialize
the .bss section */
  _sbss = .;        /* define a global symbol at bss
start */
  __bss_start__ = _sbss;
  *(.bss)
  *(.bss*)
  *(COMMON)

  . = ALIGN(4);
  _ebss = .;        /* define a global symbol at bss
end */
  __bss_end__ = _ebss;
} >RAM
```

```
.data :
{
  /* Initialized data sections goes into RAM
  . = ALIGN(4);
  _sdata = .;       /* create a global symbol at
data start */
  *(.data)          /* .data sections */
  *(.data*)         /* .data* sections */

  . = ALIGN(4);
  _edata = .;       /* define a global symbol at
data end */
} >RAM AT> FLASH
```

D'où vient le contenu

La cartographie de la mémoire (Memory map)

Dans le fichier STM32F407VGTx_FLASH.ld, les définitions de ce qui va dans le segment de mémoire Flash.

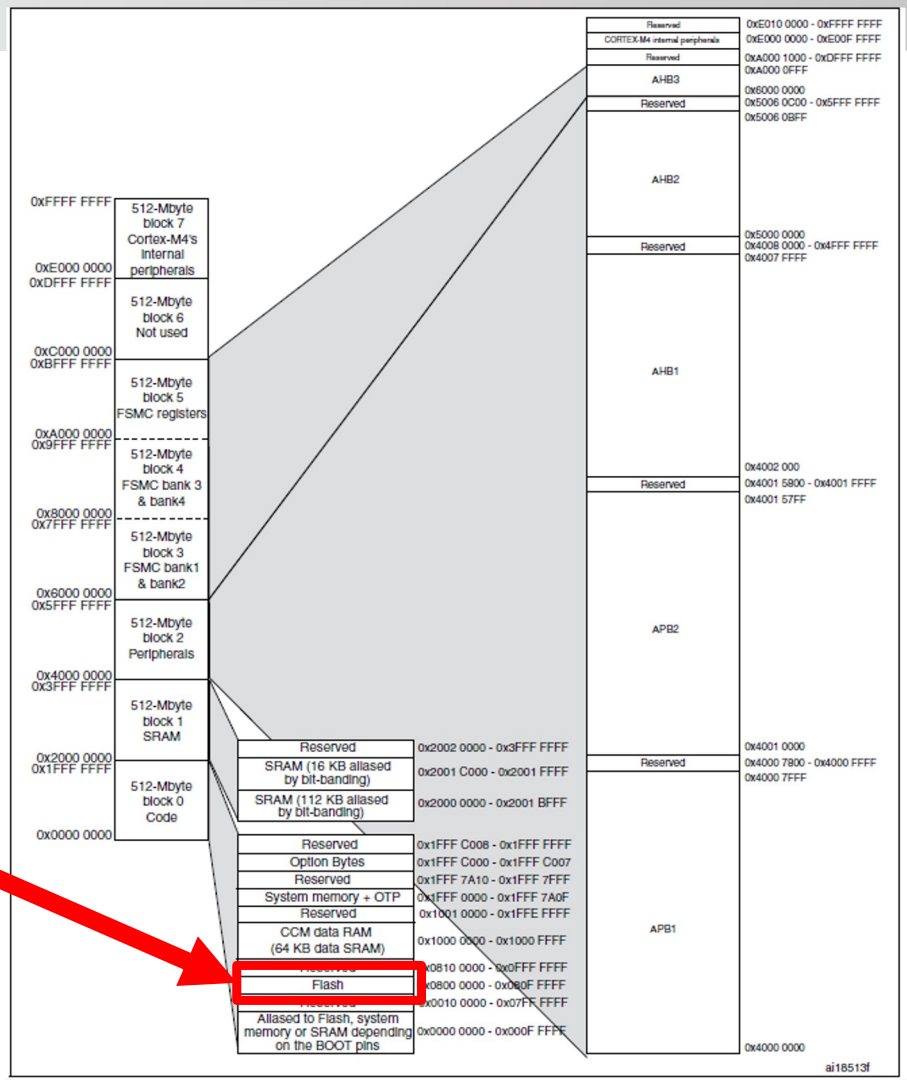
```
  /* Constant data goes into FLASH */
  .rodata :
  {
    . = ALIGN(4);
    *(.rodata)         /* .rodata sections (constants,
strings, etc.) */
    *(.rodata*)        /* .rodata* sections (constants,
strings, etc.) */
    . = ALIGN(4);
  } >FLASH
```

```
  .text :
  {
    . = ALIGN(4);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
    *(.eh_frame)

    KEEP (*(.init))
    KEEP (*(.fini))

    . = ALIGN(4);
    _etext = .;        /* define a global symbols at end of
code */
  } >FLASH
```

La mémoire programme

La mémoire programme

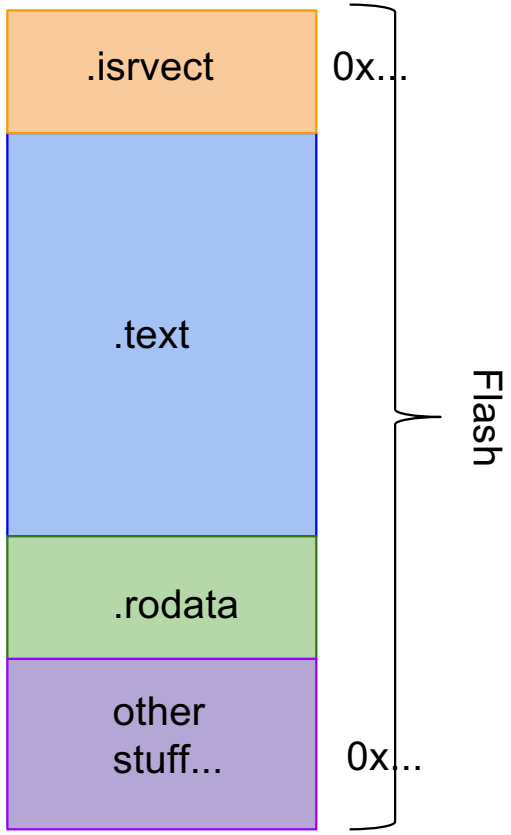
Le code que l'on écrit, une fois assemblé, est transformé en binaire, puis lorsque on programme le robot, il est stocké sur le segment de mémoire Flash.



La mémoire programme

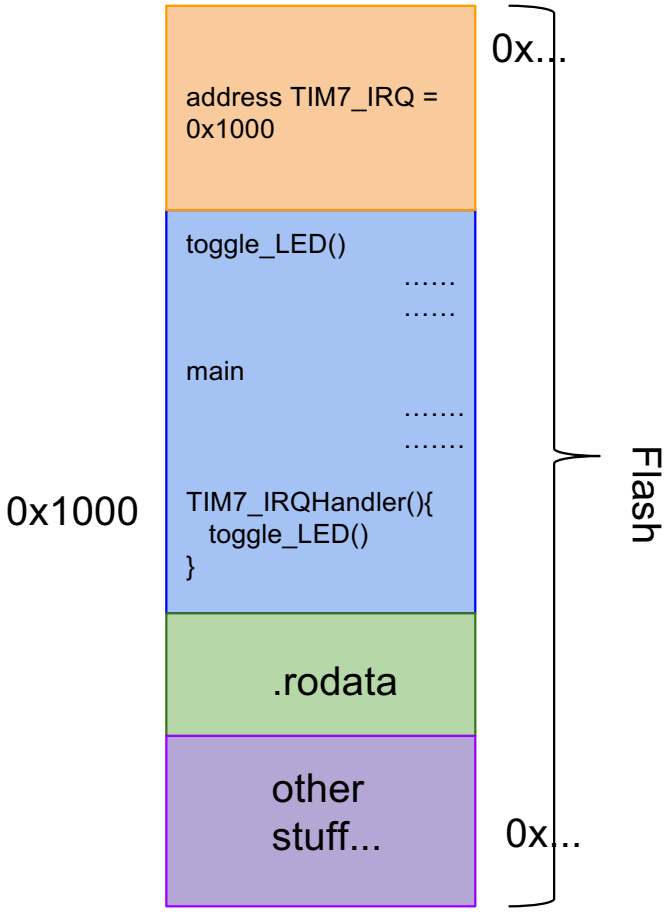
Le code flashé a une structure qui peut être simplifiée comme ceci:

- .isrvect: table des vecteurs d'interruption
- .text: contient le code que vous avez écrit
- .rodata: contient les constantes

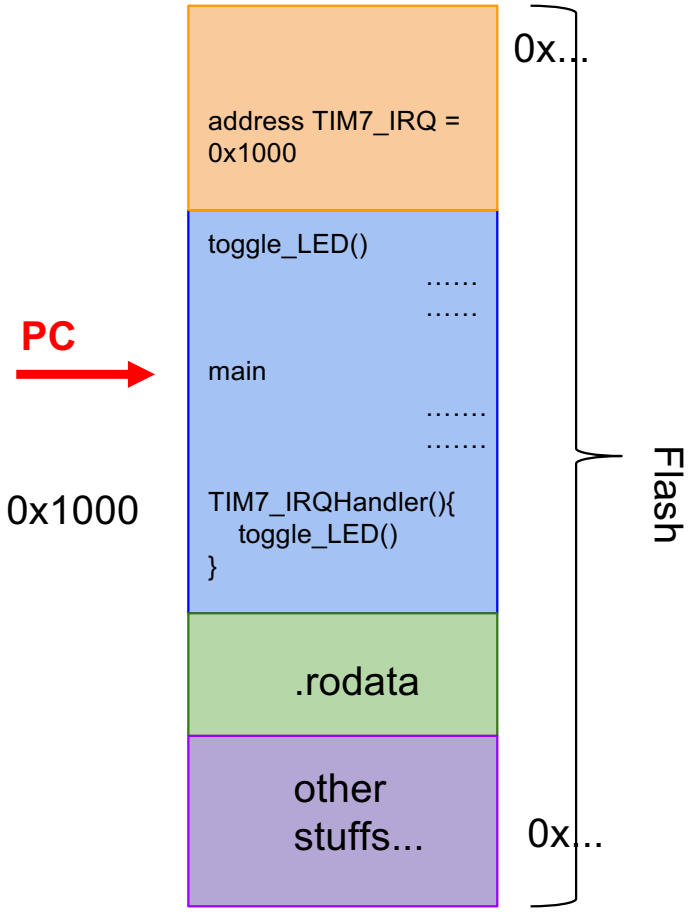


La mémoire programme, la table des vecteurs d'interruption

La table des vecteurs d'interruption va indiquer le liens au Program Counter (PC) entre les interruption générées lors de l'exécution du code (externe ou interne) et les routines d'interruptions écrites dans le code.

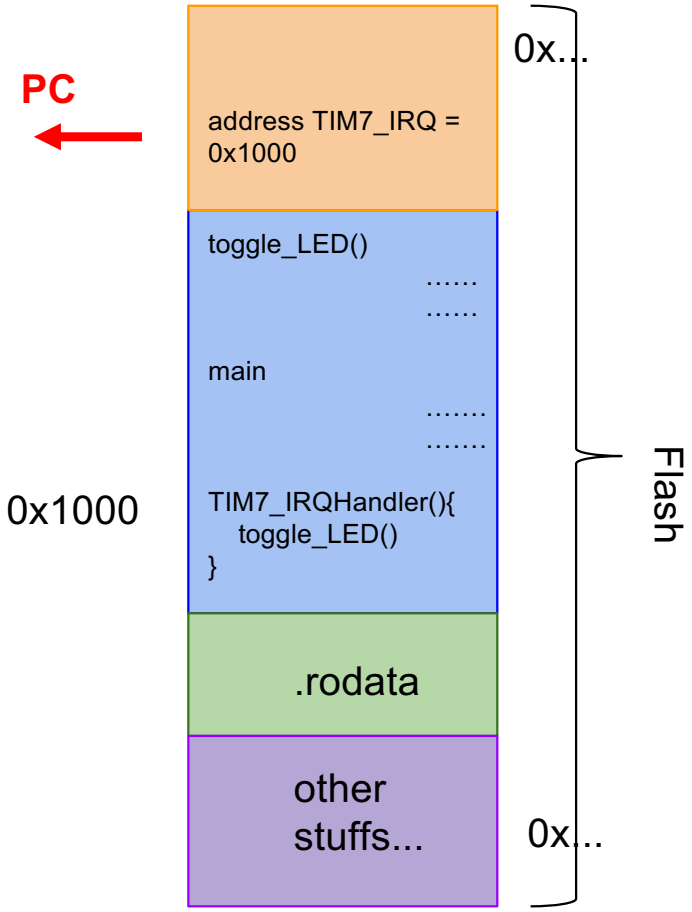


La mémoire programme, la table des vecteurs d'interruption



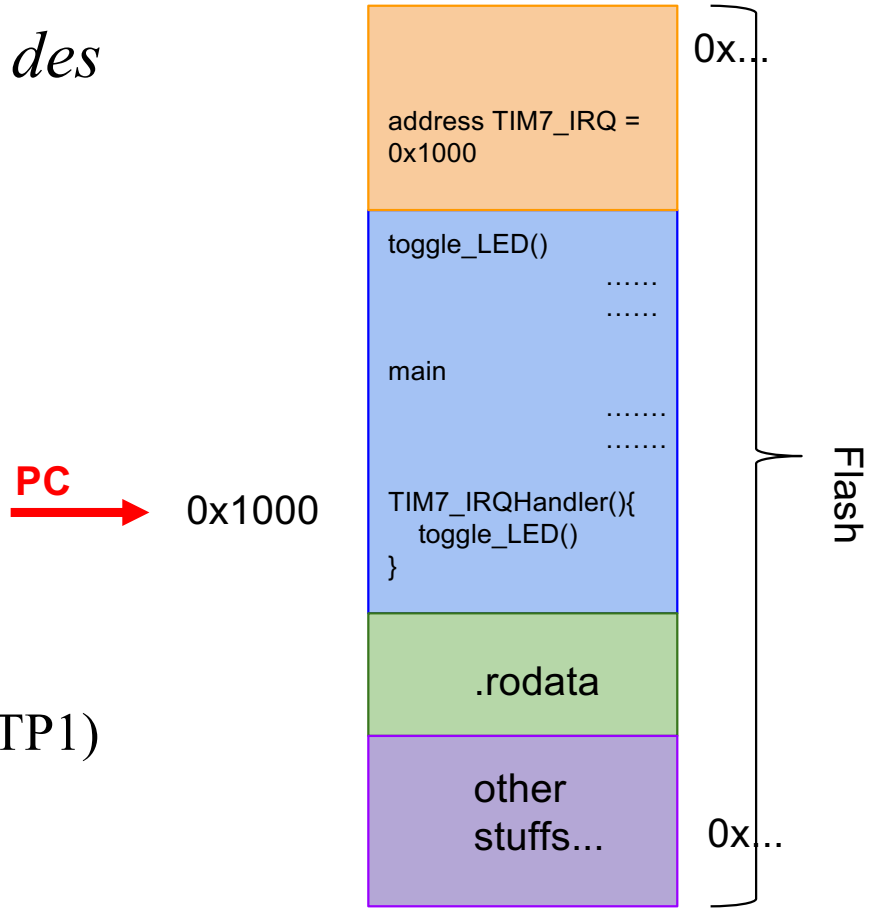
Le program counter exécute le code tranquillement

La mémoire programme, la table des vecteurs d'interruption



Une interruption générée par le Timer 7!

La mémoire programme, la table des vecteurs d'interruption



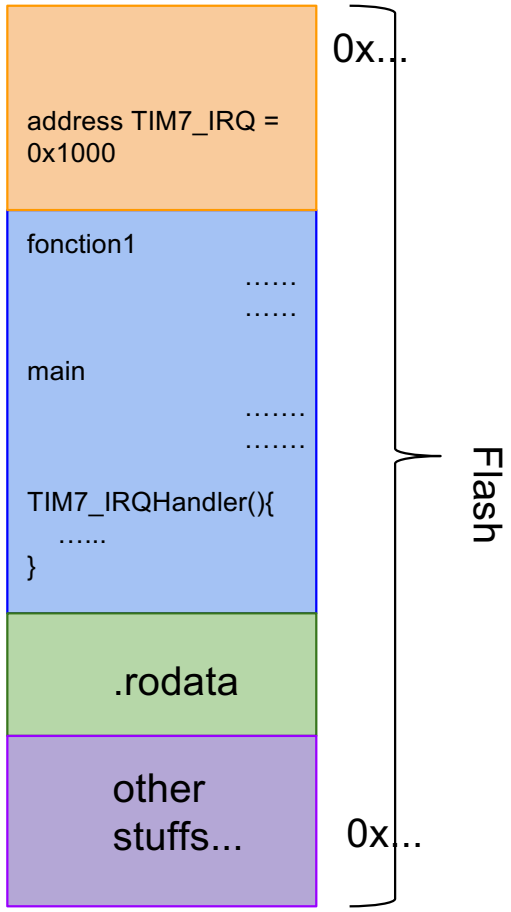
On toggle la LED de l'epuck2 (TP1)

La mémoire programme, .text et .const

Le segment .text contient donc tout le code écrit par le programmeur, et le .rodata notamment les variables constantes déclarées comme:

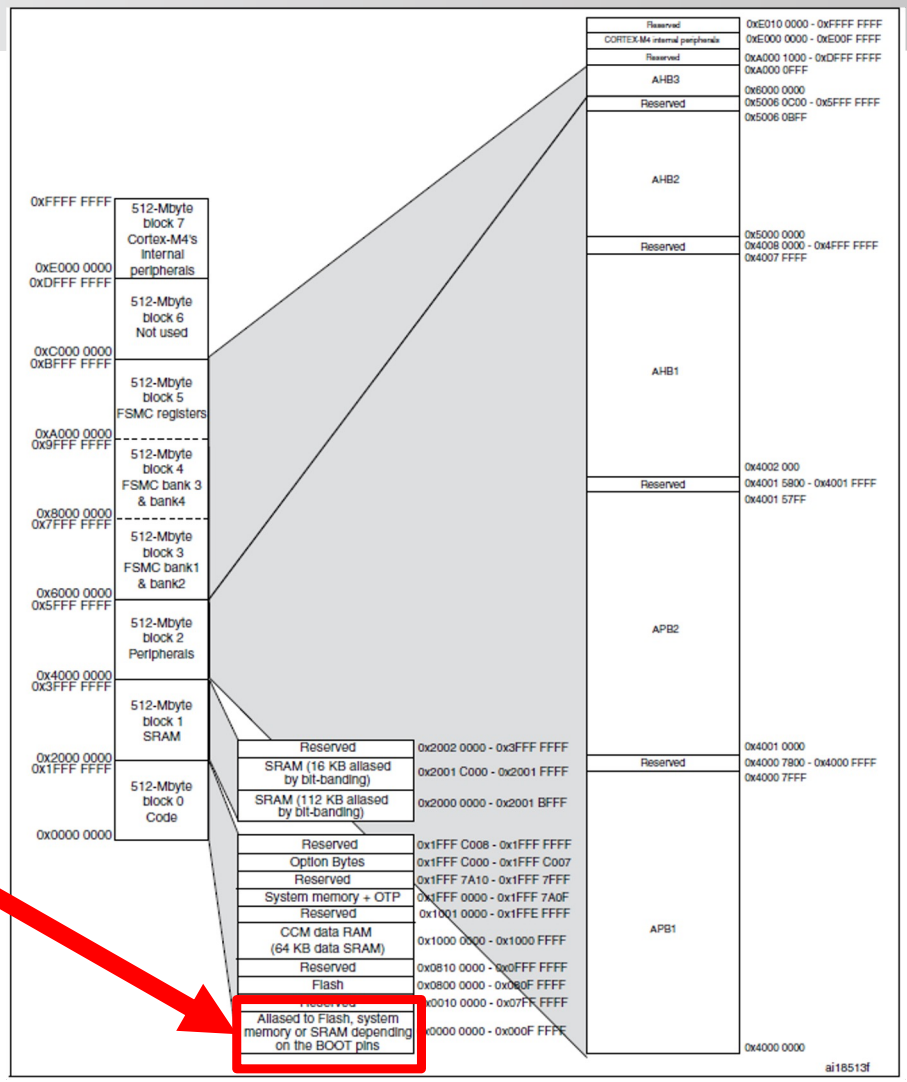
const int a = 2;

qui ne pourront pas être modifiée lors de l'exécution du code.



Le démarrage

Pour savoir où se trouve le code à exécuter lorsqu'on démarre, un vecteur d'interruption special est tout au début: le RESET



Le démarrage: RESET

Reset	Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.

Le RESET est une interruption spéciale à très haute priorité qui est appelée lors de la mise sous tension ou lors d'un RESET en fonctionnement.

Le démarrage: RESET

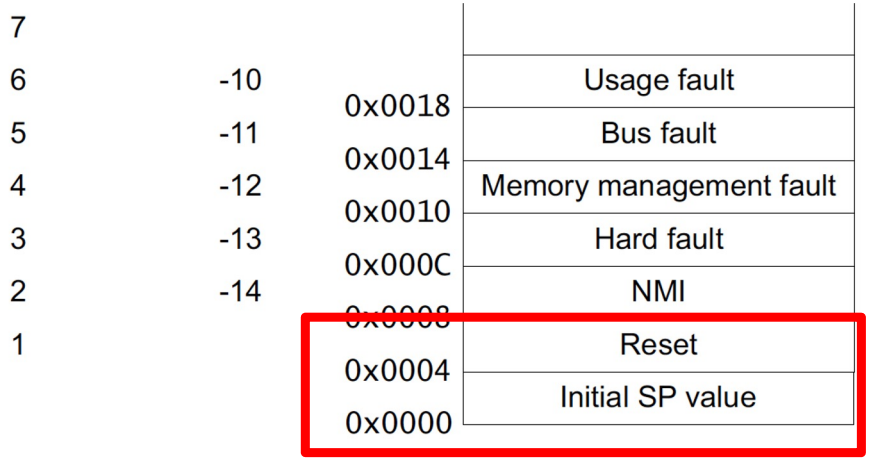


Figure 2-2 Vector table

On system reset, the vector table is fixed at address `0x00000000`. Privileged software can write to the VTOR to relocate the vector table start address to a different memory location, in the range `0x00000080` to `0x3FFFFF80`, see *Vector Table Offset Register* on page 4-16.

Il y a deux informations essentielles au démarrage: l'emplacement du stack et le début du code. Ces informations sont à l'adresse 0.

La cartographie de la mémoire (Registres des périphériques)

Les registres sont aussi stockés à des adresses bien précises. Exemple, le registre MODER des GPIOs:

8.4 GPIO registers

This section gives a detailed description of the GPIO registers.
For a summary of register bits, register address offsets and reset values, refer to *Table 39*.

The GPIO registers can be accessed by byte (8 bits), half-words (16 bits) or words (32 bits).

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

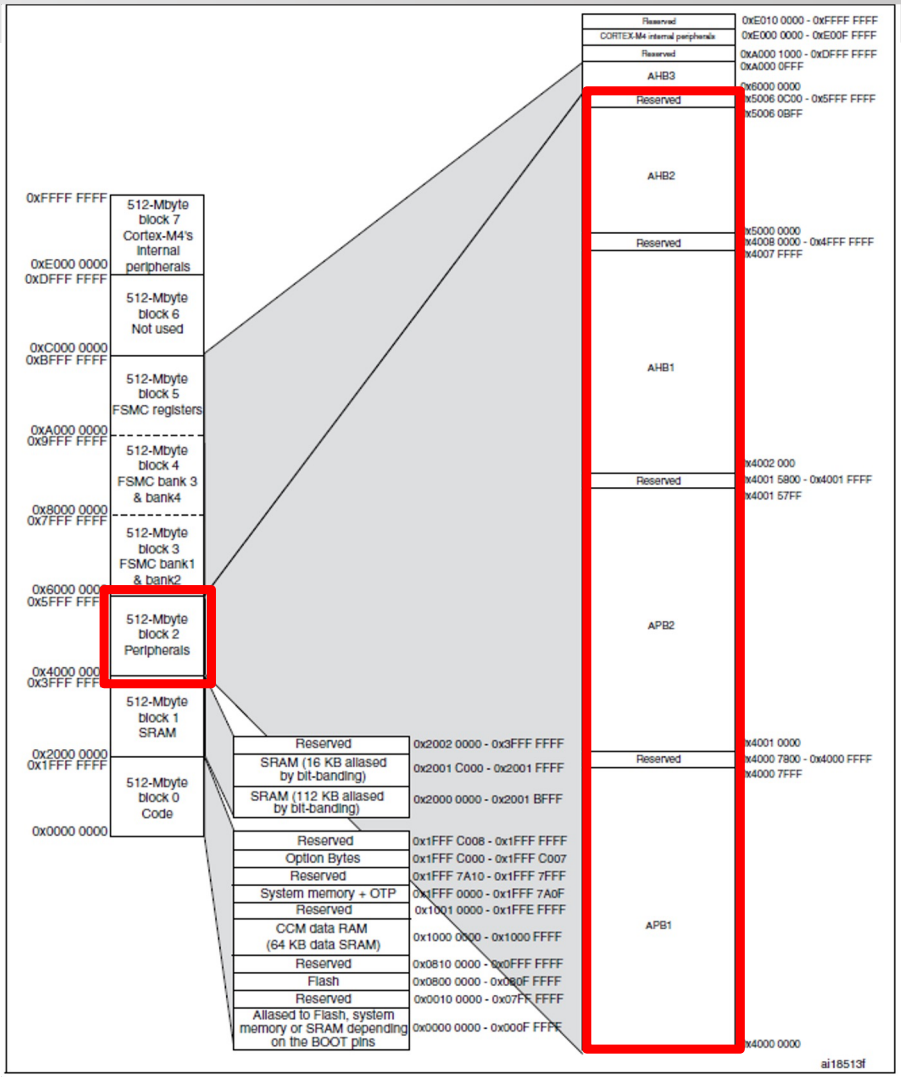
STM32F407 Reference Manual

Les registres

Les registres

Les registres pour configurer les GPIOs sont stockés et peuvent être modifiés en accédant à une zone où sont stockés tous les registres liés aux périphériques du microcontrôleur

STM32F407 datasheet



Registres (périphériques)

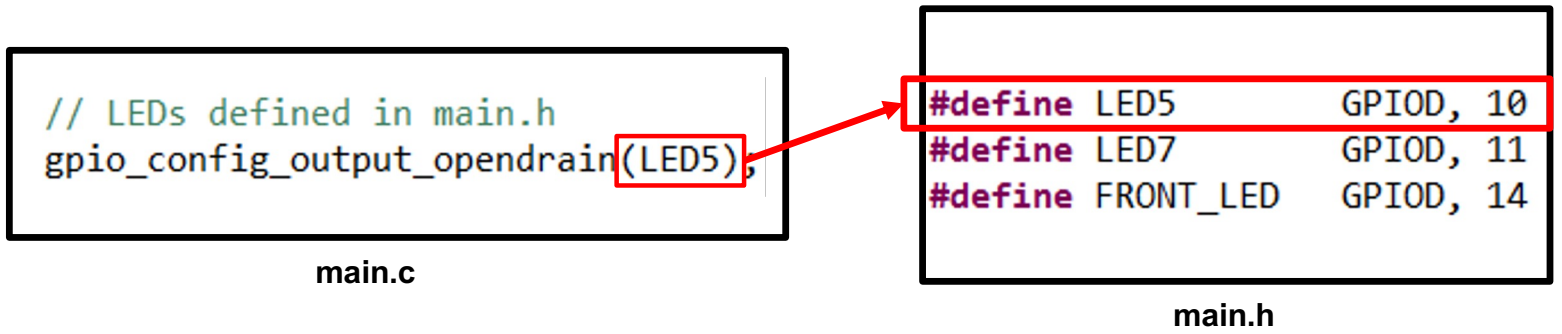
Exemple de code:

```
int main(void)
{

    // Enable GPIOB and GPIOD peripheral clock for the LEDs
    RCC->AHB1ENR    |= RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIODEN;

    // LEDs defined in main.h
    gpio_config_output_opendrain(LED5);

    // Set the LEDs
    gpio_set(LED5);

    while (1) {

    }
}
```

Code du TP1

Registres (périphériques)

Une manière de trouver cette information pour le programmeur sur STM32F en utilisant le fichier stm32f407xx.h:




```
void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin)
```

Registres (périphériques)

Dans le fichier stm32f407xx.h, l'adresse de base des periphériques est indiquée, ensuite, il s'agit seulement de faire des shifts pour retrouver l'adresse exacte du registre dans la mémoire.

```
#define LED5          GPIOD, 10
#define LED7          GPIOD, 11
#define FRONT_LED     GPIOD, 14
```

main.h

```
#define PERIPH_BASE           0x40000000U

#define AHB1PERIPH_BASE       (PERIPH_BASE + 0x00020000U)

#define GPIOC_BASE            (AHB1PERIPH_BASE + 0x0800U)
#define GPIOD_BASE            (AHB1PERIPH_BASE + 0x0C00U)
#define GPIOE_BASE            (AHB1PERIPH_BASE + 0x1000U)

#define GPIOC   ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD   ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE   ((GPIO_TypeDef *) GPIOE_BASE)
```

stm32F407xx.h

Registres (périphériques)

Tous les registres propres au GPIOD sont donc localisé en mémoire à l'adresse:

```
#define PERIPH_BASE    0x40000000U

#define AHB1PERIPH_BASE  (PERIPH_BASE + 0x00020000U)

#define GPIOC_BASE   (AHB1PERIPH_BASE + 0x0800U)
#define GPIOD_BASE   (AHB1PERIPH_BASE + 0x0C00U)
#define GPIOE_BASE   (AHB1PERIPH_BASE + 0x1000U)

#define GPIOC   ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD   ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE   ((GPIO_TypeDef *) GPIOE_BASE)
```

stm32F407xx.h

```
   0x40000000U
+  0x00020000U
+  0x00000C00U
----------------------
=.  0x40020C00U
```

Registres (périphériques)

Sinon, il y a le Reference Manual qui donne cette information:

0x4002 2000 - 0x4002 23FF	GPIOI	Section 8.4.11: GPIO register map on page 286
0x4002 1C00 - 0x4002 1FFF	GPIOH	
0x4002 1800 - 0x4002 1BFF	GPIOG	
0x4002 1400 - 0x4002 17FF	GPIOF	
0x4002 1000 - 0x4002 13FF	GPIOE	
0x4002 0C00 - 0x4002 0FFF	GPIOD	
0x4002 0800 - 0x4002 0BFF	GPIOC	
0x4002 0400 - 0x4002 07FF	GPIOB	
0x4002 0000 - 0x4002 03FF	GPIOA	

Registres (périphériques)

Même principe pour trouver des registres propres à la structure du GPIOD:

```c
void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin)
{
    // Output type open-drain : OTy = 1
    port->OTYPER |= (1 << pin);

    // Output data low : ODRy = 0
    port->ODR &= ~(1 << pin);

    // Floating, no pull-up/down : PUPDRy = 00
    port->PUPDR &= ~(3 << (pin * 2));

    // Output speed highest : OSPEEDRy = 11
    port->OSPEEDR |= (3 << (pin * 2));

    // Output mode : MODERy = 01
    port->MODER = (port->MODER & ~(3 << (pin * 2))) | (1 << (pin * 2));
}
```

gpio.c

```c
typedef struct
{
  __IO uint32_t MODER;
  __IO uint32_t OTYPER;
  __IO uint32_t OSPEEDR;
  __IO uint32_t PUPDR;
  __IO uint32_t IDR;
  __IO uint32_t ODR;
  __IO uint32_t BSRR;
  __IO uint32_t LCKR;
  __IO uint32_t AFR[2];
} GPIO_TypeDef;
```

stm32F407xx.h

Registres (périphériques)

Même principe pour trouver des registres propres à la structure du GPIOD

```c
void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin)
{
    // Output type open-drain : OTy = 1
    port->OTYPER |= (1 << pin);

    // Output data low : ODRy = 0
    port->ODR &= ~(1 << pin);

    // Floating, no pull-up/down : PUPDRy = 00
    port->PUPDR &= ~(3 << (pin * 2));

    // Output speed highest : OSPEEDRy = 11
    port->OSPEEDR |= (3 << (pin * 2));

    // Output mode : MODERy = 01
    port->MODER  = (port->MODER & ~(3 << (pin * 2))) | (1 << (pin * 2));
}
```

gpio.c

```c
typedef struct
{
  __IO uint32_t MODER;
  __IO uint32_t OTYPER;
  __IO uint32_t OSPEEDR;
  __IO uint32_t PUPDR;
  __IO uint32_t IDR;
  __IO uint32_t ODR;
  __IO uint32_t BSRR;
  __IO uint32_t LCKR;
  __IO uint32_t AFR[2];
} GPIO_TypeDef;
```

stm32F407xx.h

Registres (périphériques)

Les registres sont simplement définis comme un offset sur l'adresse attribuées au GPIOD -> 0x40020C00U + offet = adresse du registre

```
typedef struct
{
  __IO uint32_t MODER;    /*!< GPIO port mode register,               Address offset: 0x00      */
  __IO uint32_t OTYPER;   /*!< GPIO port output type register,        Address offset: 0x04      */
  __IO uint32_t OSPEEDR;  /*!< GPIO port output speed register,       Address offset: 0x08      */
  __IO uint32_t PUPDR;    /*!< GPIO port pull-up/pull-down register,  Address offset: 0x0C      */
  __IO uint32_t IDR;      /*!< GPIO port input data register,         Address offset: 0x10      */
  __IO uint32_t ODR;      /*!< GPIO port output data register,        Address offset: 0x14      */
  __IO uint32_t BSRR;     /*!< GPIO port bit set/reset register,      Address offset: 0x18      */
  __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address offset: 0x1C      */
  __IO uint32_t AFR[2];   /*!< GPIO alternate function registers,     Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

stm32F407xx.h

Registres (périphériques)

Le registre MODER se trouve à l'adresse mémoire 0x40020C00, le registre OTYPER 0x40020C04, etc.

8.4 GPIO registers

This section gives a detailed description of the GPIO registers.
For a summary of register bits, register address offsets and reset values, refer to *Table 39*.

The GPIO registers can be accessed by byte (8 bits), half-words (16 bits) or words (32 bits).

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Registres (périphériques)

Le registre MODER se trouve à l'adresse mémoire 0x40020C00, le registre OTYPER 0x40020C04, etc.

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

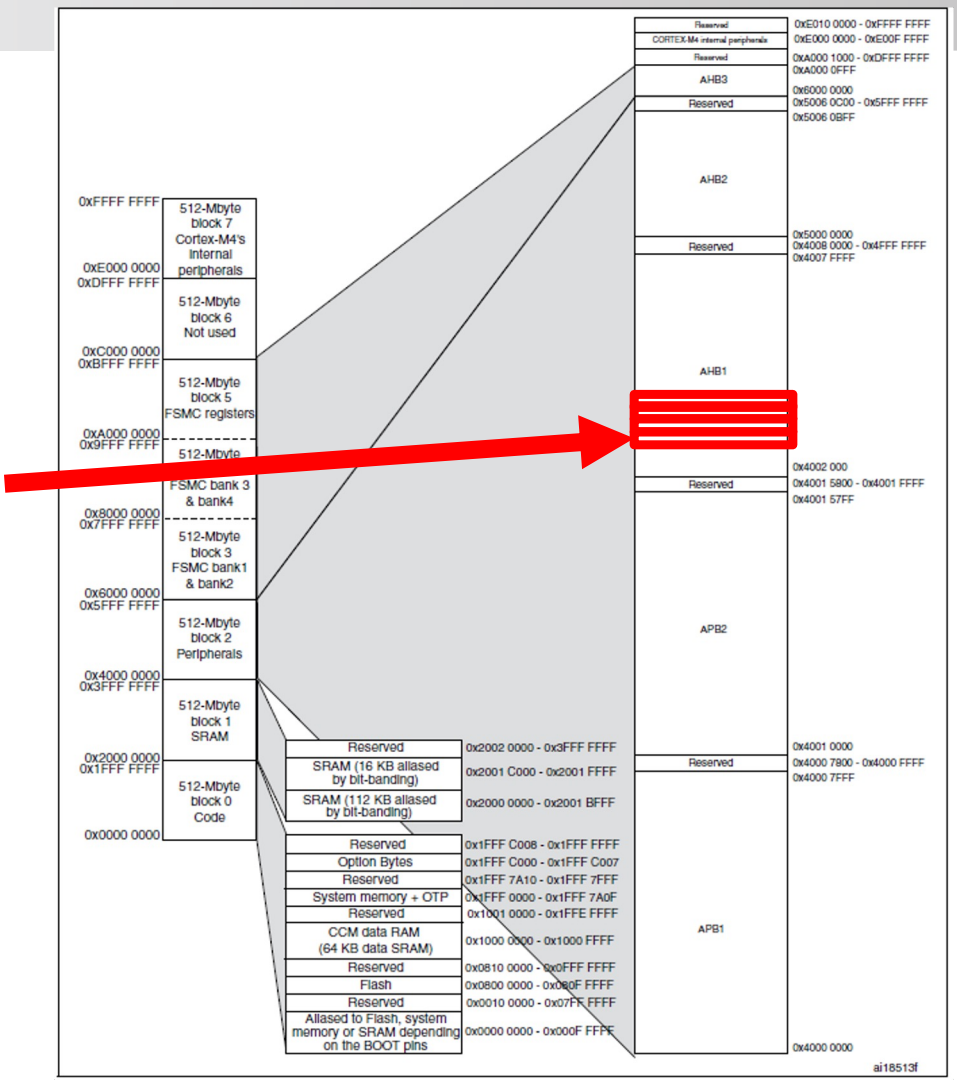
Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Registres (périphériques)

Le registre MODER (longueur 4 bytes) du port GPIOD se trouve à l'adresse mémoire 0x40020C00



La mémoire pour les données

La gestion des variables

Dans le cadre de la compilation, il est important de comprendre:

- Comment sont gérées les divers types de variables
- Quel code est généré par le compilateur
- Quelles possibilités offre le compilateur sur le processeur utilisé

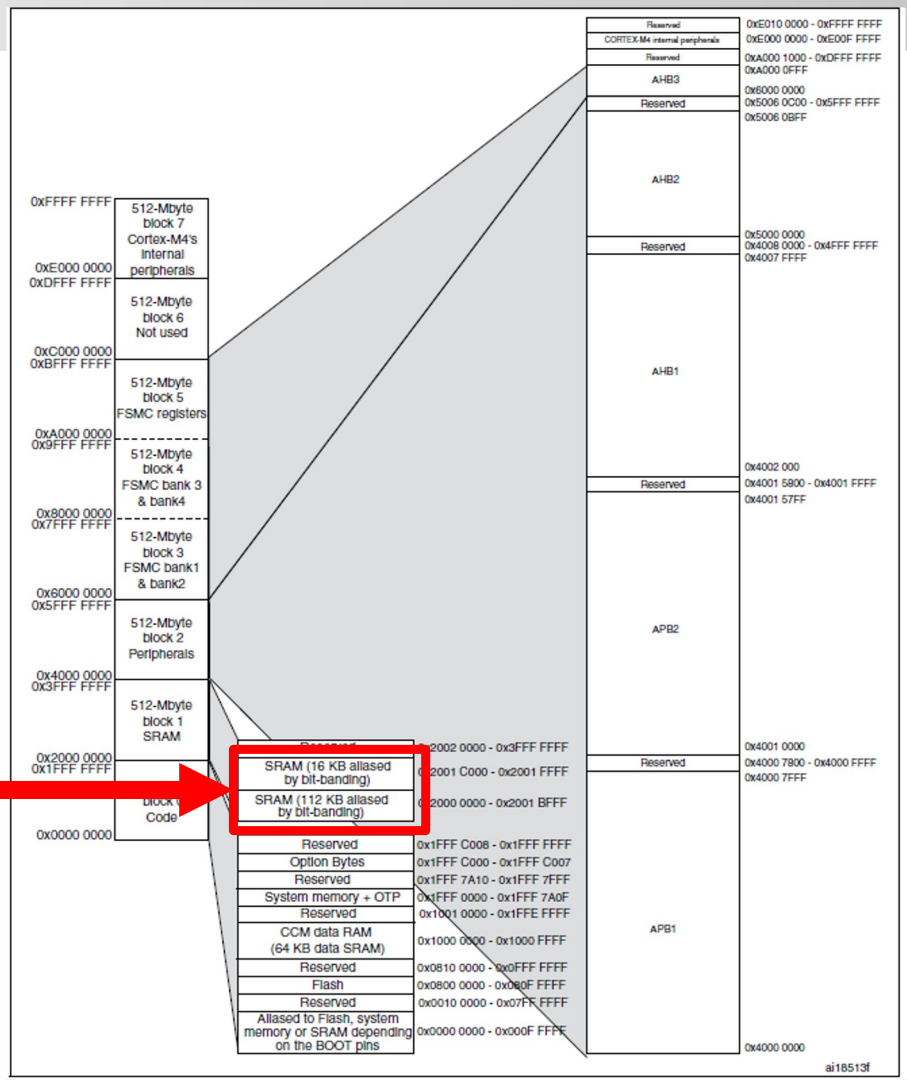
Pour cela nous allons observer les problèmes à partir de:

- Types de variables
- Gestion de types (cast)
- Gestion des cast implicite

La gestion des variables

C’est en SRAM que vont être stockées les variables globales et locales durant l’exécution du code.

Les variables locales seront sockées sur la stack (pile) qui est aussi en SRAM.



Types des variables:

Quelles valeurs peut prendre la variable? Quel type choisir?

Types	Taille (Bytes)	Range
char	1	-128 -> 127
unsigned char	1	0 -> 255
short	2	-32768 -> 32767
unsigned short	2	0 -> 65535
int	4	-2147483648 -> 2147483647
unsigned int	4	0 -> 4294967295
long	4	-2147483648 -> 2147483647
unsigned long	4	0 -> 4294967295
long long	8	-9223372036854775808 -> 9223372036854775807
unsigned long long	8	0 -> 18446744073709551615
float	4	1.17594351e-38 -> 3.40282347e+38
double	8	2.22507385850720138e-308 -> 1.79769313486231571e+308

Types des variables:

Plutôt que d'utiliser des char, short et des long, la convention est souvent d'utiliser la nomenclature intN_t ou uintN_t pour les entiers qui sont définies comme:

Type	Signe	Byte
int8_t	signé	1 (-> équivalent à un char)
uint8_t	non-signé	1
int16_t	signé	2 (-> équivalent à un short)
uint16_t	non-signé	2
int32_t	signé	4 (-> équivalent à un int)
uint32_t	non-signé	4
int64_t	signé	8 (-> équivalent à un long long)
uint64_t	non-signé	8

Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

Test.c

Compilation

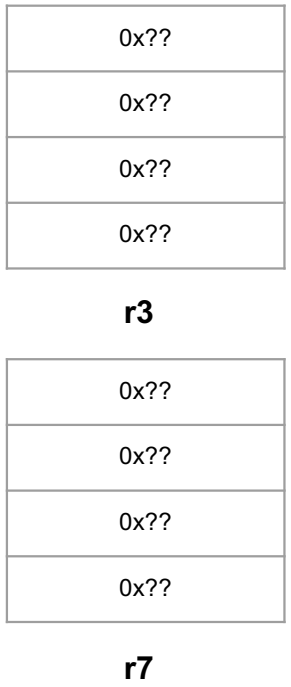
```
main:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        movs    r3, #5
        str     r3, [r7, #4]
        movs    r3, #17
        str     r3, [r7]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
```

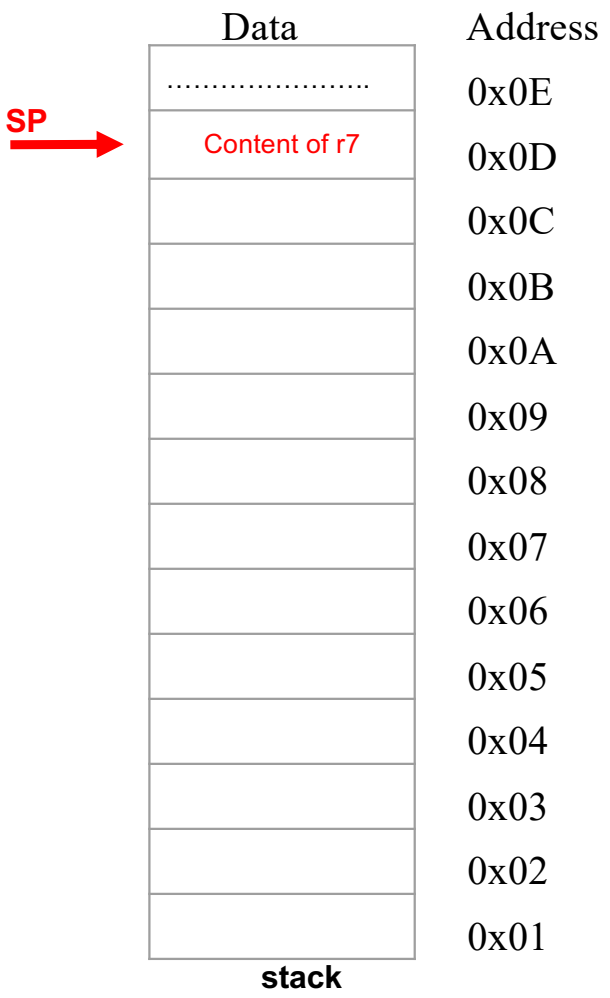
Test.s

Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        movs    r3, #5
        str     r3, [r7, #4]
        movs    r3, #17
        str     r3, [r7]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
```

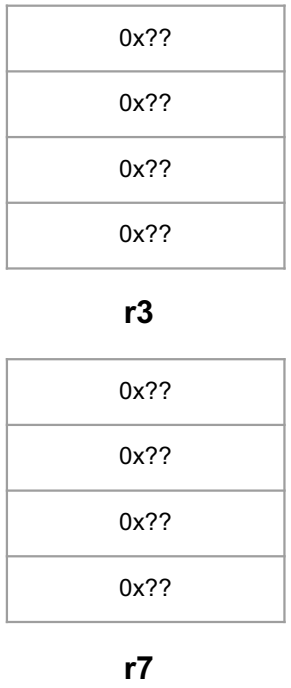


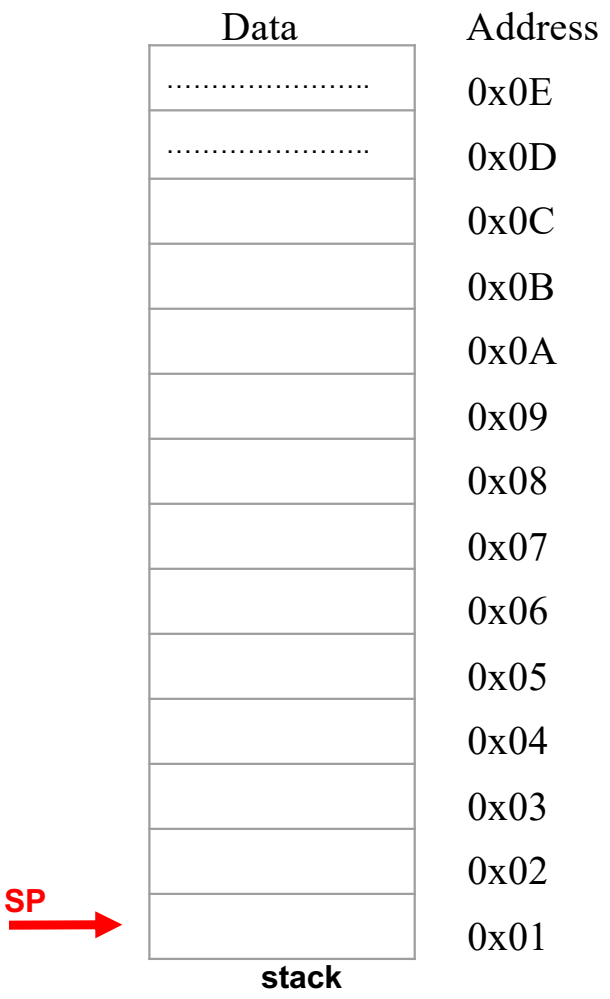


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```

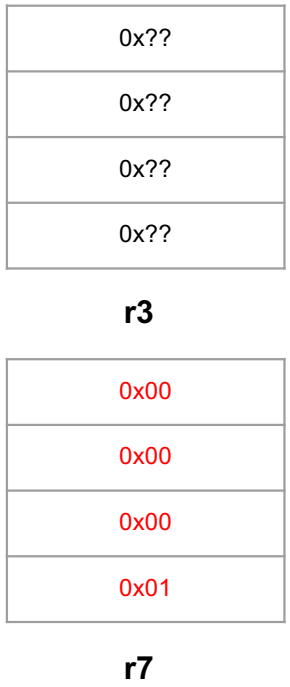


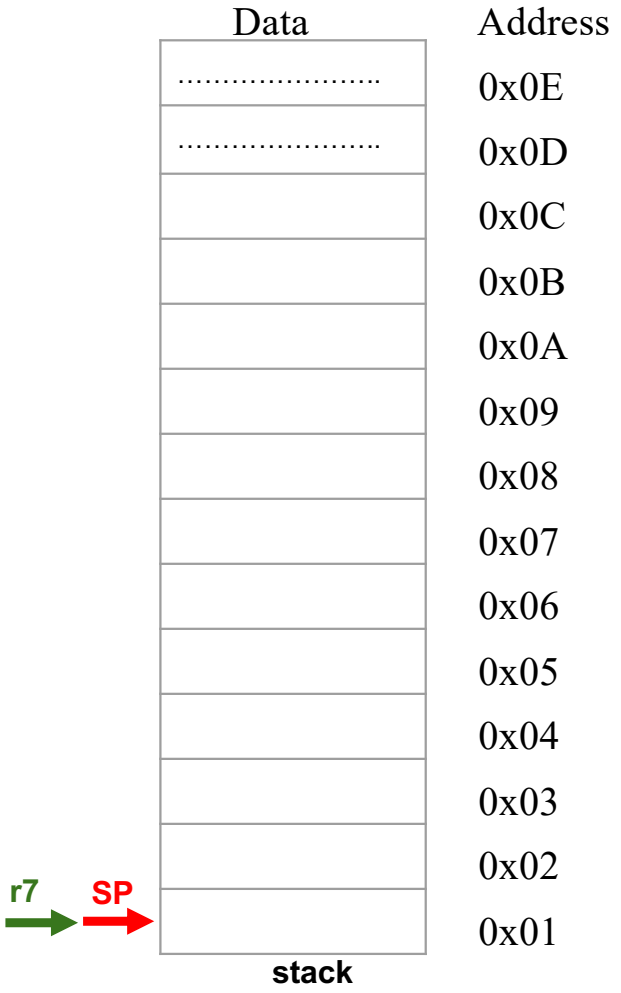


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```

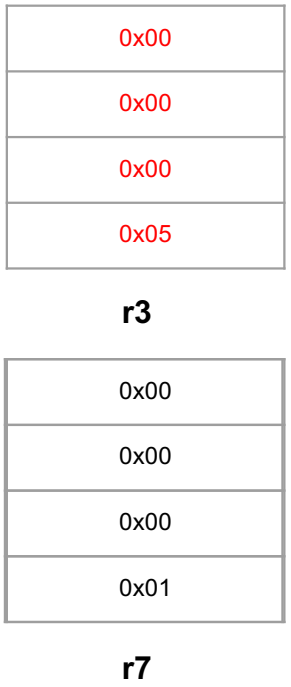


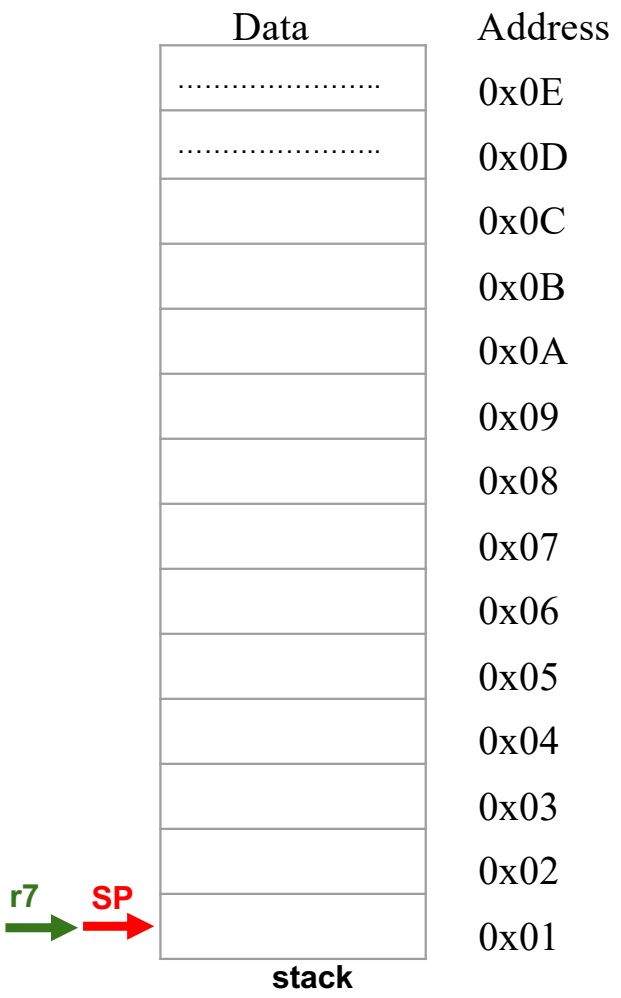


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```

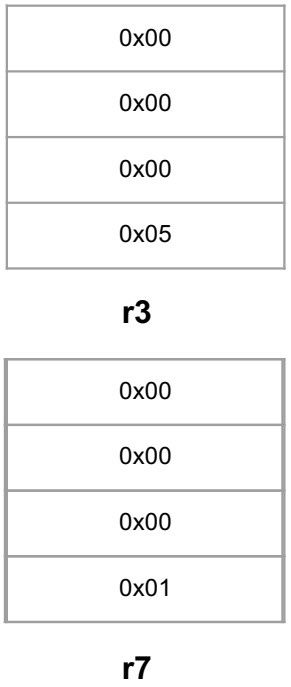


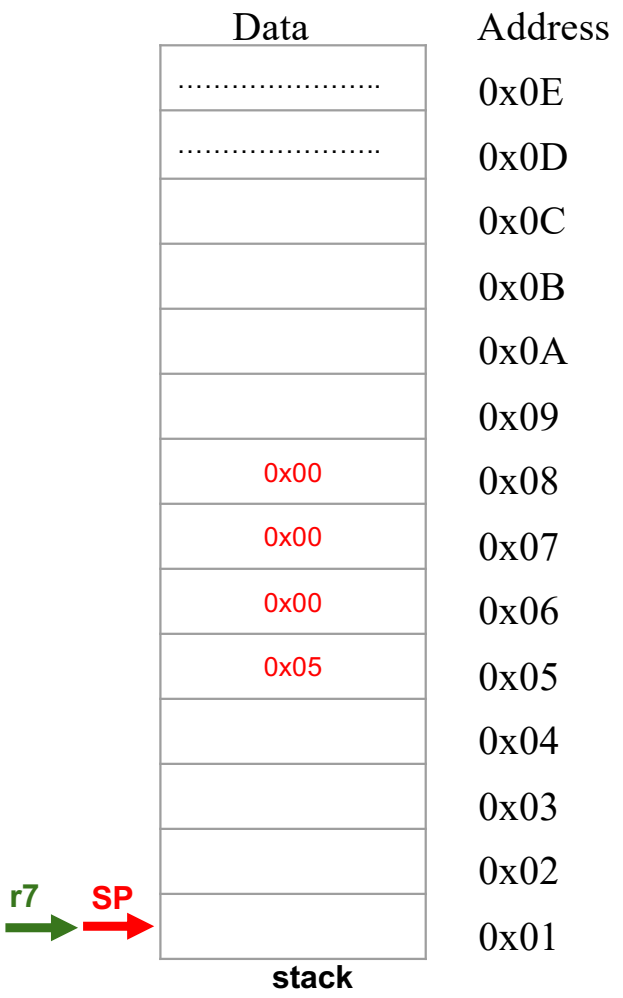


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```

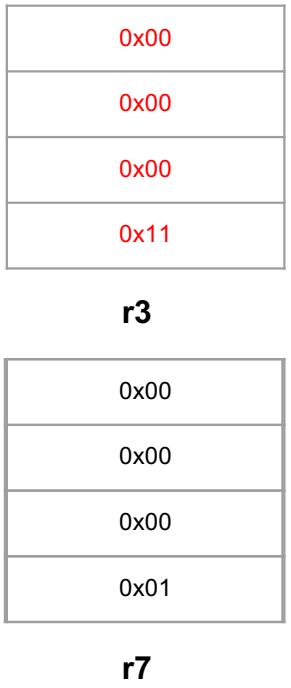


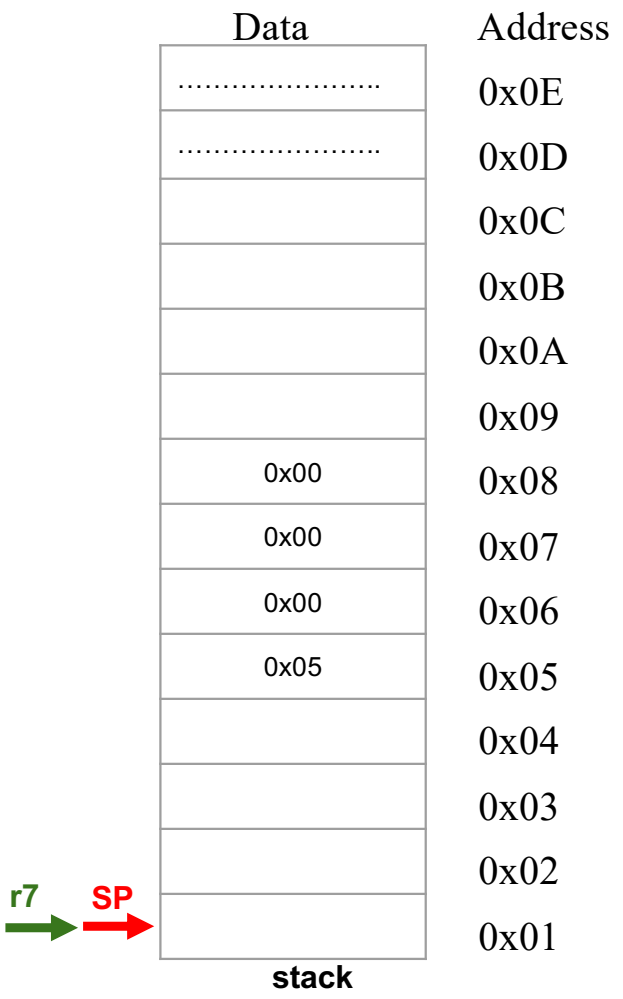


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```

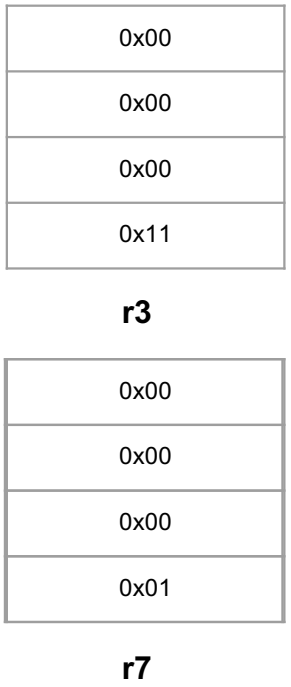


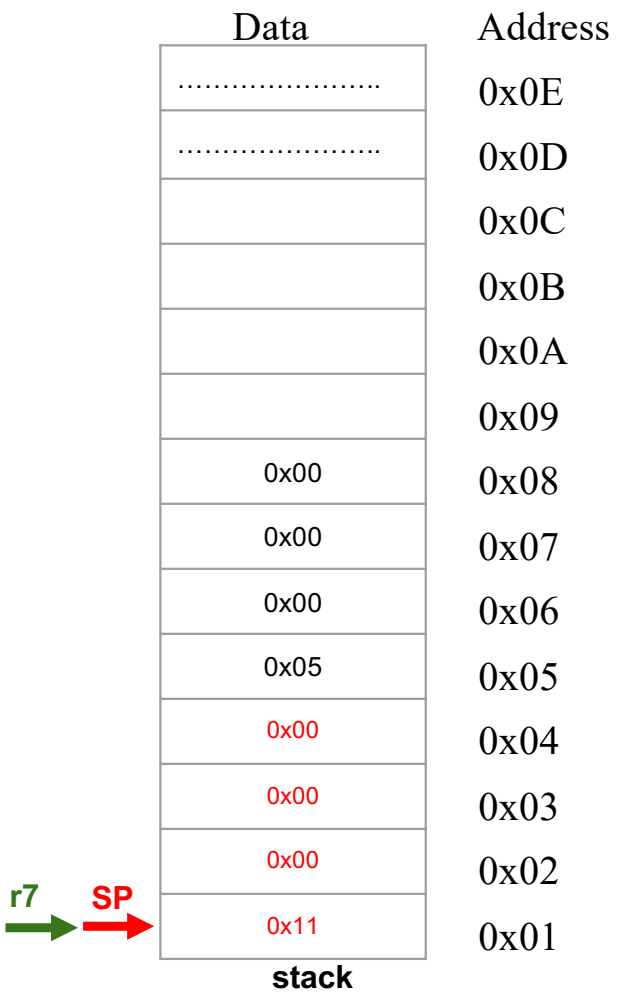


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```

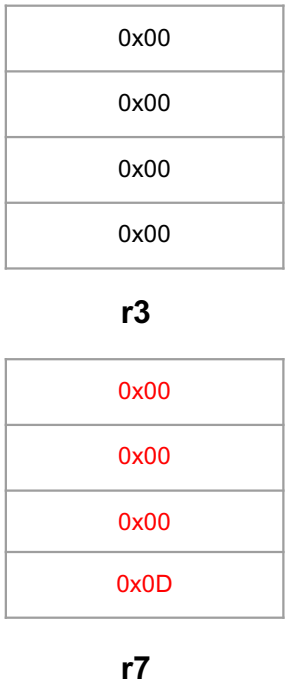


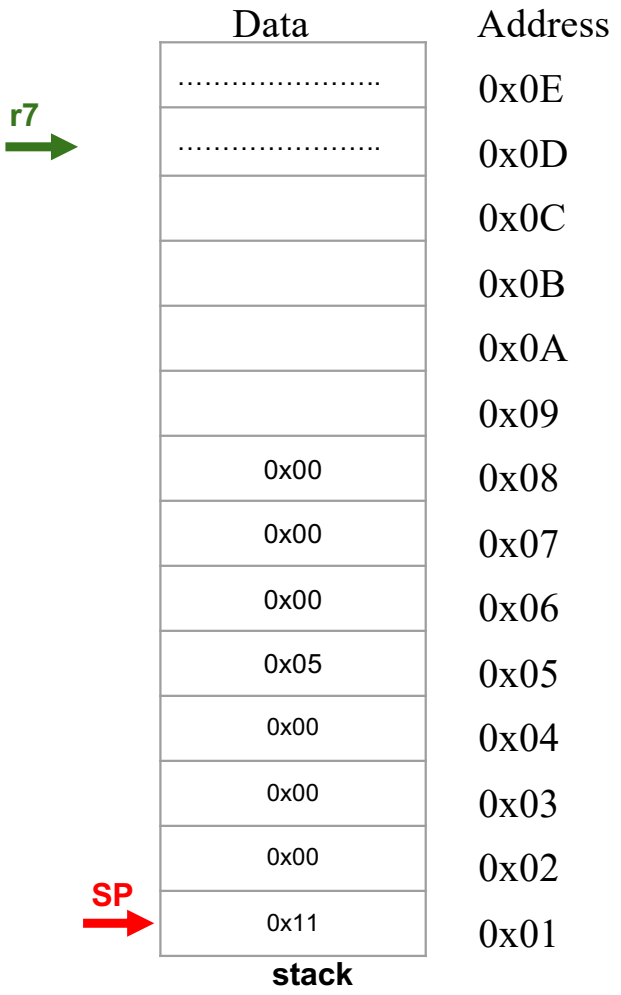


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```

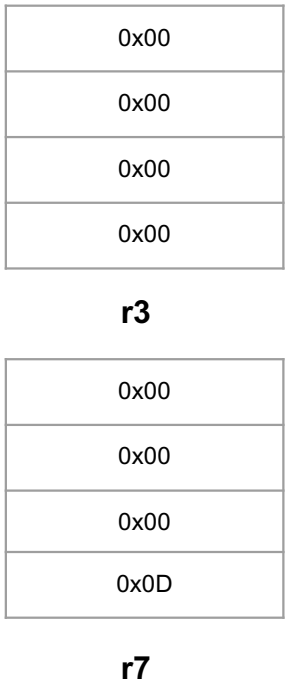


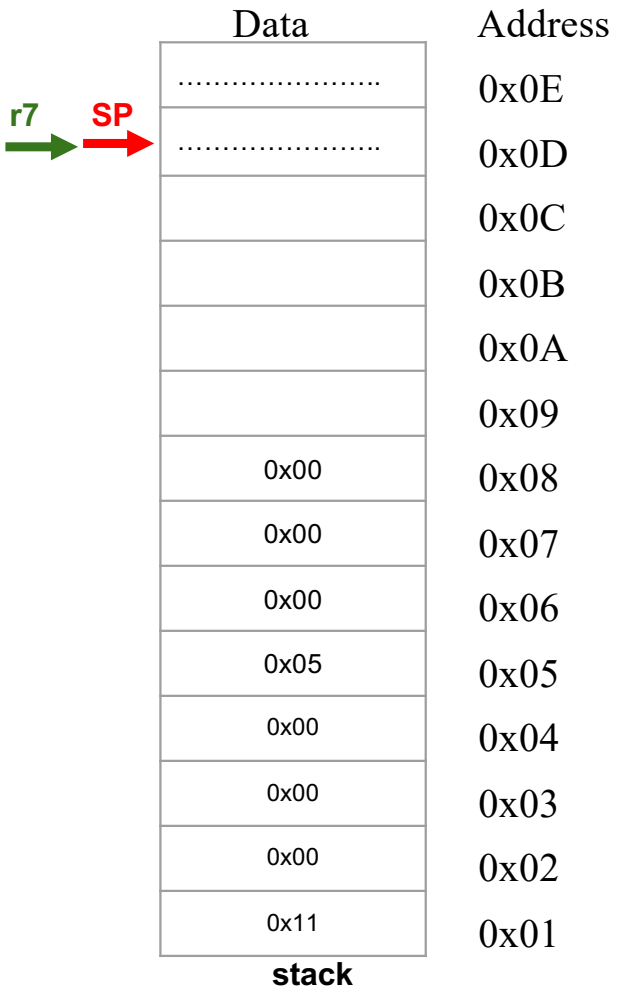


Gestion des variables locales:

```
int main()
{
        int a = 5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
    push    {r7}
    sub     sp, sp, #12
    add     r7, sp, #0
    movs    r3, #5
    str     r3, [r7, #4]
    movs    r3, #17
    str     r3, [r7]
    movs    r3, #0
    mov     r0, r3
    adds    r7, r7, #12
    mov     sp, r7
    @ sp needed
    pop     {r7}
```





Gestion des variables locales:

Si on résume:

- Les variables locales d'une fonction sont stockées sur la pile (qui se trouve dans la mémoire RAM).

- Au début de la fonction, un espace suffisant sur la pile est attribué pour stocker les variables locales

- Les variables locales sont créées lors de l'appel de la fonction et ensuite supprimées à la fin de la fonction.

Gestion des variables lors de passages de paramètres à des fonctions:

passage de paramètres

branchements

```
int foo(int param) {
        return param + 1;
}

int main() {

        int var1 = 1;

        int var2 = foo(var1);

        return 0;
}
```

```
main:
        push    {r7, lr}
        sub     sp, sp, #8
        add     r7, sp, #0
        movs    r3, #1
        str     r3, [r7, #4]
        ldr     r0, [r7, #4]
        bl      foo
        str     r0, [r7]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #8
        mov     sp, r7
        @ sp needed
        pop     {r7, pc}
```

```
foo:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        str     r0, [r7, #4]
        ldr     r3, [r7, #4]
        adds    r3, r3, #1
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
        bx      lr
```

Gestion des variables lors de passages de paramètres à des fonctions:

- Le registre r0 est utilisé pour passer le paramètre d'une fonction s'il n'y a qu'un seul paramètre 32bits, et pour retourner une valeur.

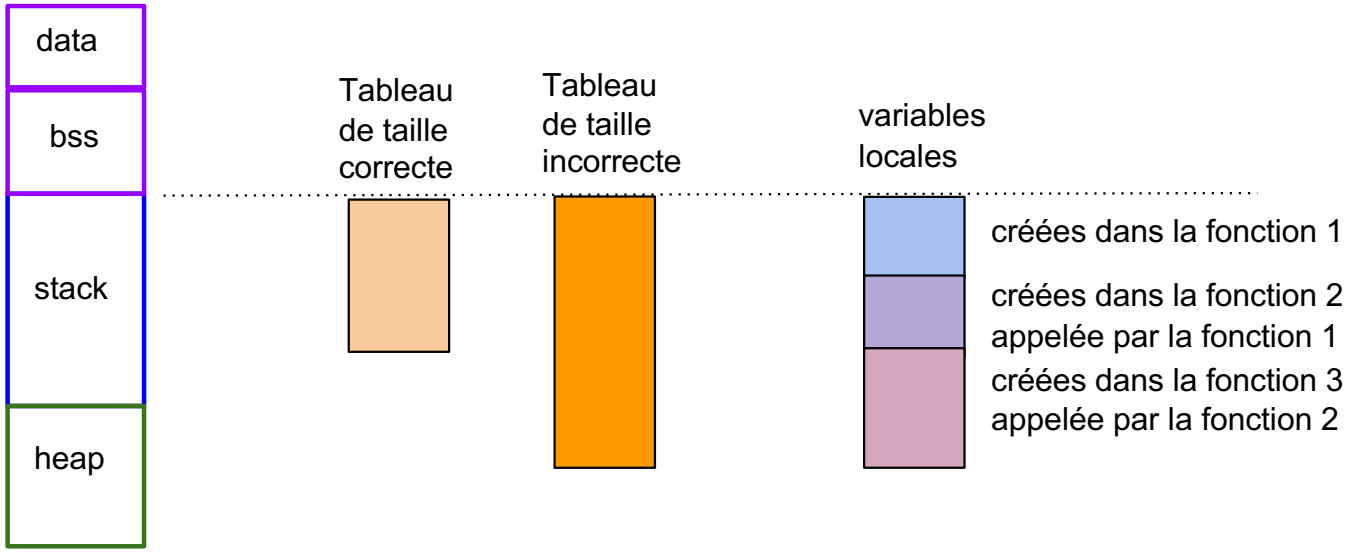
- S'il y a plus de paramètres passés à la fonction, quelques registres peuvent être utilisés pour peu de paramètres, sinon on passe par le stack

- Le retour d'une fonction appelée avec BL est fait avec le registre lr (link register). bx lr fait le branchement pour revenir à la fonction main initiale.

- Si on fait un appel BL dans une fonction, il faut sauver lr

Gestion des variables locales:

Comme la quantité de mémoire n'est pas infinie, il y a un toujours un risque d'overflow. Typiquement, si on génère de gros tableaux déclarés en variables locales, ou si on a une très grande quantités de fonctions "imbriquées" avec des variables déclarées en locales, une partie de la mémoire peut être corrompue!



Gestion des variables locales:

Une solution est de déclarer les gros tableaux en tant que variables **globales** ou **statiques**.

Mais attention, les variables globales par exemple peuvent poser des risques pour le programmeur car elles peuvent être accessible à plusieurs endroit, donc il ne faut pas en abuser!

Gestion des variables locales: nombres entiers négatifs:

```
int main()
{
        int a = -5;
        unsigned int b = 17;

        return 0;
}
```

```
main:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        mvn     r3, #4
        str     r3, [r7, #4]
        movs    r3, #17
        str     r3, [r7]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
```

MVN{S}{*cond*} *Rd*, *Operand2*

The MVN instruction takes the value of *Operand2*, performs a bitwise logical NOT operation on the value, and places the result into *Rd*.

From the Cortex-M4 Generic User Guide

Gestion des variables locales: cast

```
int main()
{
        int a = 12;
        char b = 17;

        b = (char) a;

        return 0;
}
```

```
main:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        movs    r3, #12
        str     r3, [r7, #4]
        movs    r3, #17
        strb    r3, [r7, #3]
        ldr     r3, [r7, #4]
        strb    r3, [r7, #3]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
```

3.4.2 LDR and STR, immediate offset

Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

Syntax

op{type}{cond} Rt, [Rn {, #offset}] ; immediate offset

type Is one of:

B unsigned byte, zero extend to 32 bits on loads.

From the Cortex-M4 Generic User Guide

Gestion des variables locales: cast

```
int main()
{
        int a = 1600;
        char b = 17;

        b = (char) a;

        return 0;
}
```


```
main:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        mov     r3, #1600
        str     r3, [r7, #4]
        movs    r3, #17
        strb    r3, [r7, #3]
        ldr     r3, [r7, #4]
        strb    r3, [r7, #3]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
```

Gestion des variables statiques:

```
int main()
{
        static int a = 27;
        int b = 48;

        a = b;

        return 0;
}
```

Les variables statiques ne sont pas stockées sur la pile, mais dans un autre endroit sur la RAM (bss ou data en fonction de leur initialisation). Elles sont gérées de manières différentes que les variables locales et peuvent être vues seulement depuis la fonction où elles sont déclarées.

```
main:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        movs    r3, #48
        str     r3, [r7, #4]
        ldr     r2, .L3
        ldr     r3, [r7, #4]
        str     r3, [r2]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
        bx      lr
.L4:
        .align  2
.L3:
        .word   a.4100
        .size   main, .-main
        .data
        .align  2
        .type   a.4100, %object
        .size   a.4100, 4
a.4100:
        .word   27
```

Gestion des variables globales:

```
int a = 27;

int main()
{
        int b = 48;

        a = b;

        return 0;
}
```

Les variables globales ne sont pas non-plus stockées sur la stack, mais dans un autre endroit sur la RAM (bss ou data en fonction de leur initialisation). Elles sont gérées d'une manière similaire aux variables statiques mais peuvent être utilisées de manière globales par plusieurs fonctions.

```
a:          .word       27
            .text
            .align      1
            .global     main
            .syntax unified
            .thumb
            .thumb_func
            .fpu softvfp
            .type       main, %function
main:
            push        {r7}
            sub         sp, sp, #12
            add         r7, sp, #0
            movs        r3, #48
            str         r3, [r7, #4]
            ldr         r2, .L3
            ldr         r3, [r7, #4]
            str         r3, [r2]
            movs        r3, #0
            mov         r0, r3
            adds        r7, r7, #12
            mov         sp, r7
            @ sp needed
            pop         {r7}
            bx          lr

            .align      2
.L4:
.L3:
            .word       a
```

Gestion des variables globales:

```
int a = 27;

int main()
{
        int b = 48;

        a = b;

        return 0;
}
```

ATTENTION à ne pas abuser des variables globales, belle source de bugs!!!!!!! Alternatives correctes:

- Variable locale statique: aussi hors stack, si ceci est le but
- Variable globale statique (que dans fichier, avec éventuellement des fonction get et put)

```
a:      .word   27
        .text
        .align  1
        .global main
        .syntax unified
        .thumb
        .thumb_func
        .fpu softvfp
        .type   main, %function

main:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        movs    r3, #48
        str     r3, [r7, #4]
        ldr     r2, .L3
        ldr     r3, [r7, #4]
        str     r3, [r2]
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r7}
        bx      lr

        .align  2

.L4:
.L3:
        .word   a
```

Allocation dynamique de mémoire

Il existe encore une autre manière d'allouer dynamiquement de la mémoire en utilisant le segment mémoire tas (heap) à l'aide par exemple de l'instruction `malloc`.

Ce sujet sera abordé plus tard dans le cours.

Résumé: choix d'un type de variable

1. Evaluer la gamme de variation de la variable
   Minimum et maximum
2. Evaluer la résolution nécessaire
3. Estimer le type de calcul nécessaire sur la variable
4. Vérifier le temps à disposition pour faire ces calculs
5. Choisir une unité pour la variable
6. Choisir le type de la variable

TP de cette semaine

Etude de la structure du compilateur C

Programmation d'un PWM pour régler l'intensité d'une LED

Programmation du contrôle de moteurs pas à pas en C

Création d'une librairie de contrôle de moteur pour ce robot

ATTENTION:
Venez à l'introduction à 10h15

Problème de audio dans l'enregistrement (et la transmission zoom)

1 / 6 | Next > | Last Respondent >>
<<< List of responses | Print this Response

Respondent: **- Anonymous -**

Evaluation du cours et du TP

Ceci est un questionnaire anonyme, donnez votre avis franchement, merci.

1 * Donnez votre évaluation du cours de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	○	○	◉
Forme	○	○	○	○	○	◉

2 * Donnez votre évaluation du TP de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	○	○	○	○	◉	○

3 Vos commentaires en détail (si nécessaire):

<< First Respondent | < Previous | 2 / 6 | Next > | Last Respondent >>
<<< List of responses | Print this Response

Respondent: **- Anonymous -**

Evaluation du cours et du TP

Ceci est un questionnaire anonyme, donnez votre avis franchement, merci.

1 * Donnez votre évaluation du cours de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	○	○	○	◉	○	○

2 * Donnez votre évaluation du TP de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	○	○	◉
Forme	○	○	◉	○	○	○

3 Vos commentaires en détail (si nécessaire):

Le git est un vrai labyrinthe. Avoir une section dédiée par TP (étapes d'installation, instructions, etc.) serait d'une grande aide. Peut-être même ajouter le readme de chaque branche dans une de ces sections. Le système de demande d'assistant est une superbe idée ! Ça marche super bien.

<< First Respondent | < Previous | 3 / 6 | Next > | Last Respondent >>
<<< List of responses | Print this Response

Respondent: **- Anonymous -**

Evaluation du cours et du TP

Ceci est un questionnaire anonyme, donnez votre avis franchement, merci.

1 * Donnez votre évaluation du cours de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	○	○	○	○	○	◉

2 * Donnez votre évaluation du TP de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	○	○	○	○	○	◉

3 Vos commentaires en détail (si nécessaire):

Respondent: **- Anonymous -**

Evaluation du cours et du TP

Ceci est un questionnaire anonyme, donnez votre avis franchement, merci.

1 * Donnez votre évaluation du cours de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	○	○	○	◉	○	○

2 * Donnez votre évaluation du TP de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	○	○	○	◉	○	○

3 Vos commentaires en détail (si nécessaire):

Le chatbot, on peut lui parler et lui envoyer 7-8 messages chaque 3 heures, si on ne paie pas 20 CHF par mois dommage car il est super pédagogique. Au lieu de répondre directement à nos questions, ils nous aident comme un assistant en nous posant des questions pour qu'on essaie de comprendre par nous-même. Bref, faites le gratuit svp

Respondent: **- Anonymous -**

Evaluation du cours et du TP

Ceci est un questionnaire anonyme, donnez votre avis franchement, merci.

1 * Donnez votre évaluation du cours de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	○	○	◉
Forme	○	○	○	○	○	◉

2 * Donnez votre évaluation du TP de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	◉	○	○	○	○	○

3 Vos commentaires en détail (si nécessaire):

Si les tps sont une partie aussi importante du programme je pense que c est necessaire d en faire une video (au moin par des assistants). Le cours lui est parfait :)

<< First Respondent | < Previous | 6 / 6
<<< List of responses | Print this Response

Respondent: **- Anonymous -**

Evaluation du cours et du TP

Ceci est un questionnaire anonyme, donnez votre avis franchement, merci.

1 * Donnez votre évaluation du cours de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	○	○	◉
Forme	○	○	○	○	○	◉

2 * Donnez votre évaluation du TP de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	○	○	◉
Forme	○	○	○	○	○	◉

3 Vos commentaires en détail (si nécessaire):

Le chatbot est très utile, surtout le fait qu'il nous pose de questions pour nous faire réfléchir et trouver les réponses par nous mêmes.