Gestion des ressources (suite cours 3)

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

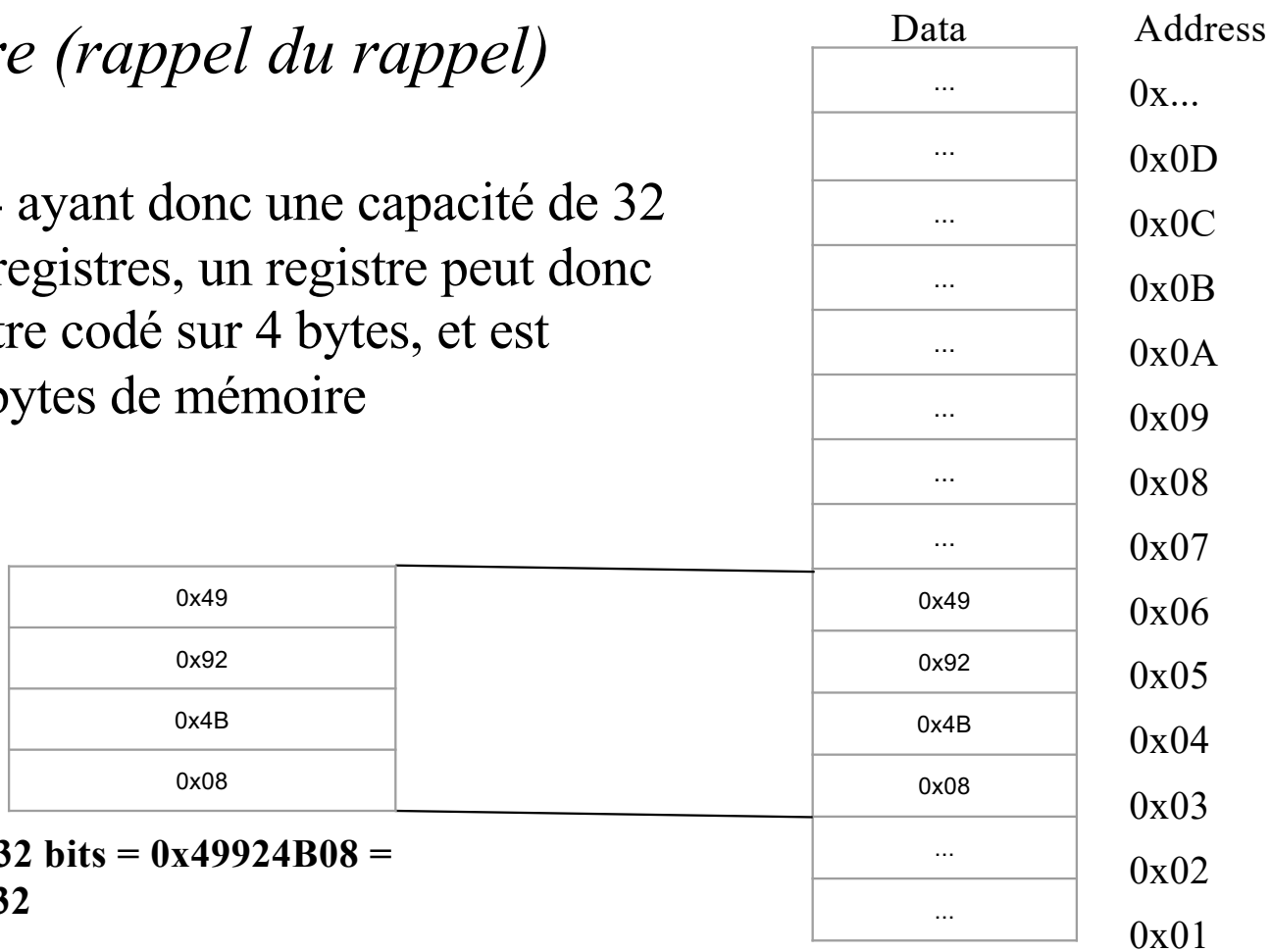
Les variables de types floats et les doubles

La semaine dernière, nous avons vu que la stack est utilisée pour stocker les variables locales entières.

Dans la première partie de cette leçon, nous allons voir comment sont gérés les calculs utilisant les types float (32 bits) / double (64 bits). Nous allons voir ici que la stack est aussi utilisée pour stocker les variables de types float et double locales, en utilisant le même principe de stockage, seulement avec une interprétation différente de la variable en fonction du type.

La mémoire (rappel du rappel)

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



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

Rappel de la semaine dernière

Nous avons vu comme étaient gérées les variables entières locales et globales:

```
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 nombres à virgules (float)

arm-none-eabi-gcc -save-temps=obj -mcpu=cortex-m4 -c essai.c -o essai.o

ABI = Application Binary Interface
AEABI = (Embedded) ABI for the ARM architecture

```
int main() {

    float PI = 3.1415;
    float mul = 2.0;

    float result = mul * PI;

    return 0;
}
```

Test.c

Compilation

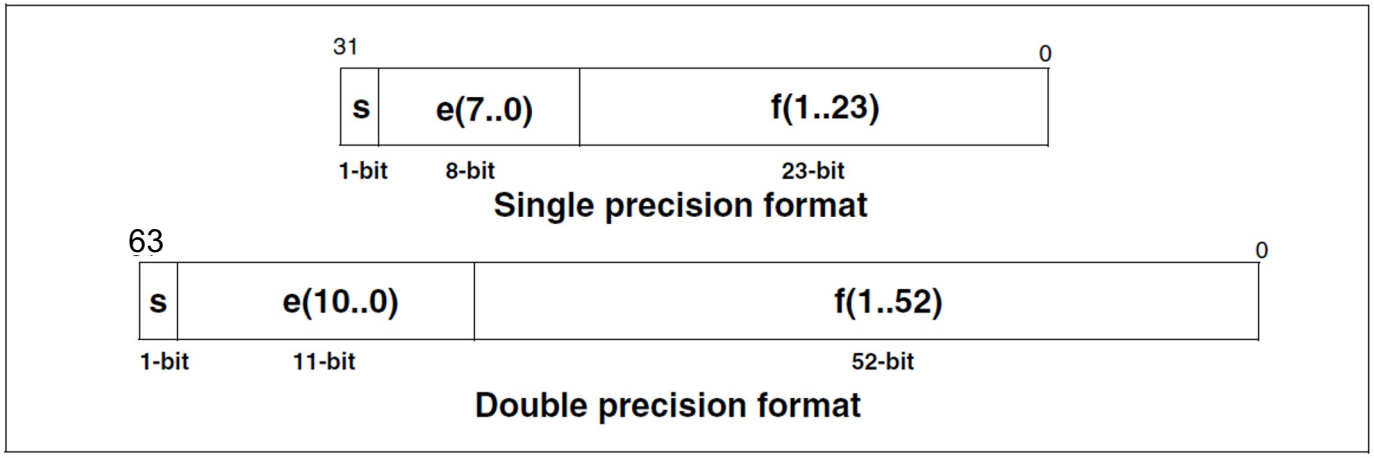
commentaire

```
main:
        push    {r7, lr}
        sub     sp, sp, #16
        add     r7, sp, #0
        ldr     r3, .L3
        str     r3, [r7, #12]           @ float
        mov     r3, #1073741824
        str     r3, [r7, #8] @ float
        ldr     r1, [r7, #12]           @ float
        ldr     r0, [r7, #8] @ float
        bl      __aeabi_fmul
        mov     r3, r0
        str     r3, [r7, #4] @ float
        …
        pop     {r7, pc}

.L4:
        .align  2
.L3:
        .word   1078529622
```

Test.s

Les variables de types floats et les doubles

Pour le processeur ARM Cortex M4 du microcontrôleur STM32F4, les nombres à virgules sont représentés en suivant la norme IEEE 754. Dans ce format on exprime le signe (s,) l'exposant (e) et partie fractionnaire (f)



Représentation des nombres: virgule flottante

1er pas: normalisation

$567 \rightarrow 5.67\ 10^2$ décimal

Comment représenter le nombre 13 en float?

$13 \rightarrow 0xD \rightarrow 1101 \rightarrow +1.101\ 2^3$ binaire

2ème pas: encodage

0	100 0001 0	101 0000 0000 0000 0000 0000
Sign (signe)	Exponent (exposant)	Fraction (mantisse)

BIAS = 127 = 0x7F
= 0111 1111
(100 0000 0 -> 1 , 100 0001 0 -> 3)

Exception: la valeur zéro est codée par des 0 partout

Représentation des nombres: virgule flottante

```
ldr r3, .L3
```

Pointe vers →

```
.L3:    .word   1078529622
```

Attention: décimal

Décodage:

→ 0x40490E56 En hexadécimal, plus facile à traiter

→

0	100 0000 0	100 1001 0000 1111 0101 0110
Sign (signe)	Exponent (exposant) BIAS = 127 = 0x7F	Fraction (mantisse)

→ $+1.10010010000111001010110 \ 2^{1}$

→ 1.57074999809×2

Gestion des nombres à virgules

```
arm-none-eabi-gcc -save-temps=obj -mcpu=cortex-m4 -c essai.c -o essai.o
```

```
int main() {

    const float PI = 3.1415;
    float mul = 2.0;

    float result = mul * PI;

    return 0;
}
```

Test.c

Compilation

```
main:
    push    {r7, lr}
    sub     sp, sp, #16
    add     r7, sp, #0
    ldr     r3, .L3
    str     r3, [r7, #12]           @ float
    mov     r3, #1073741824
    str     r3, [r7, #8] @ float
    ldr     r1, [r7, #12]           @ float
    ldr     r0, [r7, #8] @ float
    bl      __aeabi_fmul ??????
    mov     r3, r0
    str     r3, [r7, #4] @ float
    …
    pop     {r7, pc}
.L4:
    .align  2
.L3:
    .word   1078529622
    .size   main, .-main
```

Test.s

Gestion des nombres à virgules

Le processeur ARM Cortex-M4 est équipé d'une FPU (Floating Point Unit) permettant d'effectuer des opérations sur des nombres flottants en utilisant un mécanisme hardware, donc plus efficace que de le faire en software.

Cette unité doit être activée lors de la compilation du code. Si ce n'est pas le cas, les calculs en float/double vont être fait de manière software (`__aeabi_fmul`), et donc prendre beaucoup de cycles / temps!

Gestion des nombres à virgules

```
arm-none-eabi-gcc -save-temps=obj -mcpu=cortex-m4 -c essai.c -o essai.o
```

```
int main() {

    const float PI = 3.1415;
    float mul = 2.0;

    float result = mul * PI;

    return 0;
}
```

Test.c

Compilation

```
main:
    push    {r7, lr}
    sub     sp, sp, #16
    add     r7, sp, #0
    ldr     r3, .L3
    str     r3, [r7, #12]           @ float
    mov     r3, #1073741824
    str     r3, [r7, #8] @ float
    ldr     r1, [r7, #12]           @ float
    ldr     r0, [r7, #8] @ float
    bl      __aeabi_fmul
    mov     r3, r0
    str     r3, [r7, #4] @ float
    …
    pop     {r7, pc}
.L4:
    .align  2
.L3:
    .word   1078529622
    .size   main, .-main
```

Test.s

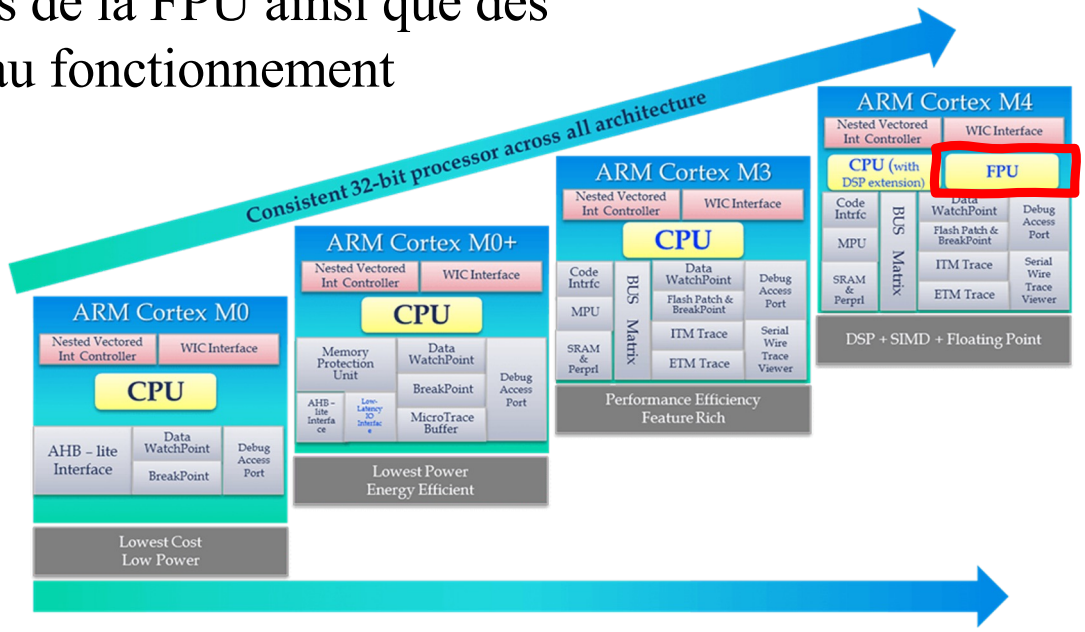
Gestion des nombres à virgules

```
17  BEGIN_ARM_FUNCTION __aeabi_fmul
18          .arch armv5t
19          MOV r12, 0x000000FF
20          ANDS r2, r12, r0, LSR #23 /* r2[8:31] = 0, r2[0:7] = a0.biased_exponent, APSR.Z = (a0.biased_exponent == 0 [zero or subnormal a0]) */
21          CMPNE r2, 0xFF /* APSR.Z ||= (a0.biased_exponent == 0xFF [infinity or NaN a0]) */
22          ANDSNE r3, r12, r1, LSR #23 /* If APSR.Z != 0 then r3[8:31] = 0, r3[0:7] = a1.biased_exponent, APSR.Z ||= (a1.biased_exponent == 0 [zero or subnormal a1]) */
23          CMPNE r3, 0xFF /* APSR.Z ||= (a1.biased_exponent == 0xFF [infinity or NaN a1]) */
24          BEQ .special_input /* Handle situations where any operand is zero, subnormal, infinite, or NaN */
25
26          /* Both operands are finite normalized normals */
27          EOR r12, r0, r1 /* r12[31] = a0.sign ^ a1.sign, r12[0:30] = garbage */
28          ADD r2, r3 /* r2 = a0.biased_exponent + a1.biased_exponent */
29          MOV r3, 0x80000000
30          ORR r0, r3, r0, LSL #8 /* r0[8:31] = a0.mantissa_with_implied_bit (r0[31] == 1), r0[0:7] = 0 */
31          ORR r1, r3, r1, LSL #8 /* r1[8:31] = a1.mantissa_with_implied_bit (r1[31] == 1), r1[0:7] = 0 */
32          UMULL r0, r3, r1, r0 /* r3:r0 = (a0.mantissa_with_implied_bit * a1.mantissa_with_implied_bit) << 16 */
33          AND r12, r12, 0x80000000 /* r12[31] = a0.sign ^ a1.sign, r12[0:30] = 0 */
34
35          CMP r3, 0x80000000 /* APSR.C = (r3 >= 0x80000000) */
36          ADDLO r3, r3, r3 /* If the high part of multiplication is only 31-bit wide, double it */
37          ADC r2, r2, -128 /* and add 1 to exponent. Here it is combined with subtracting 0x7F from biased exponent */
38
39          CMP r0, 0x0000001 /* APSR.C = (r0 != 0) */
40          LSRSCC r1, r3, #9 /* APSR.C ||= (r3[9] == 1)) */
41          ADC r3, r3, 0x0000007F /* If ((r0 != 0) || (r3[9] == 1)) add 1 to guard bit. If guard bit is also one, addition will propogate to the least significant bit. */
42
43          CMP r2, 0xFE
44          BHS .normal_path_special_output
45
46          ORR r0, r12, r2, LSL #23
47          ADD r0, r0, r3, LSR #8
48
49          BX lr
```

Gestion des nombres à virgules

On peut donc spécifier à la compilation que la FPU sera utilisée pour les opération avec des nombres à virgules.

Dès lors, des registres spécifiques de la FPU ainsi que des instructions assembleur dédiées au fonctionnement de la FPU (utilisant un hardware spécifique) vont être utilisés.
Ceci augmente la consommation du microcontrôleur...



Gestion des nombres à virgules (float)

```
arm-none-eabi-gcc -save-temps=obj -mcpu=cortex-m4 -c essai.c -o essai.o -mfloat-abi=hard
-mfpu=vfpv2
```

```
int main() {

        const float PI = 3.1415;
        float mul = 2.0;

        float result = mul * PI;

        return 0;
}
```

Test.c

Compilation

```
main:
    push    {r7}
    sub     sp, sp, #20
    add     r7, sp, #0
    ldr     r3, .L3
    str     r3, [r7, #12]           @ float
    mov     r3, #1073741824
    str     r3, [r7, #8] @ float
    vldr.32 s14, [r7, #8]
    vldr.32 s15, [r7, #12]
    vmul.f32 s15, s14, s15
    vstr.32 s15, [r7, #4]
      ...
    ldr     r7, [sp], #4
    bx      lr

.L3:
    .word   1078529622
```

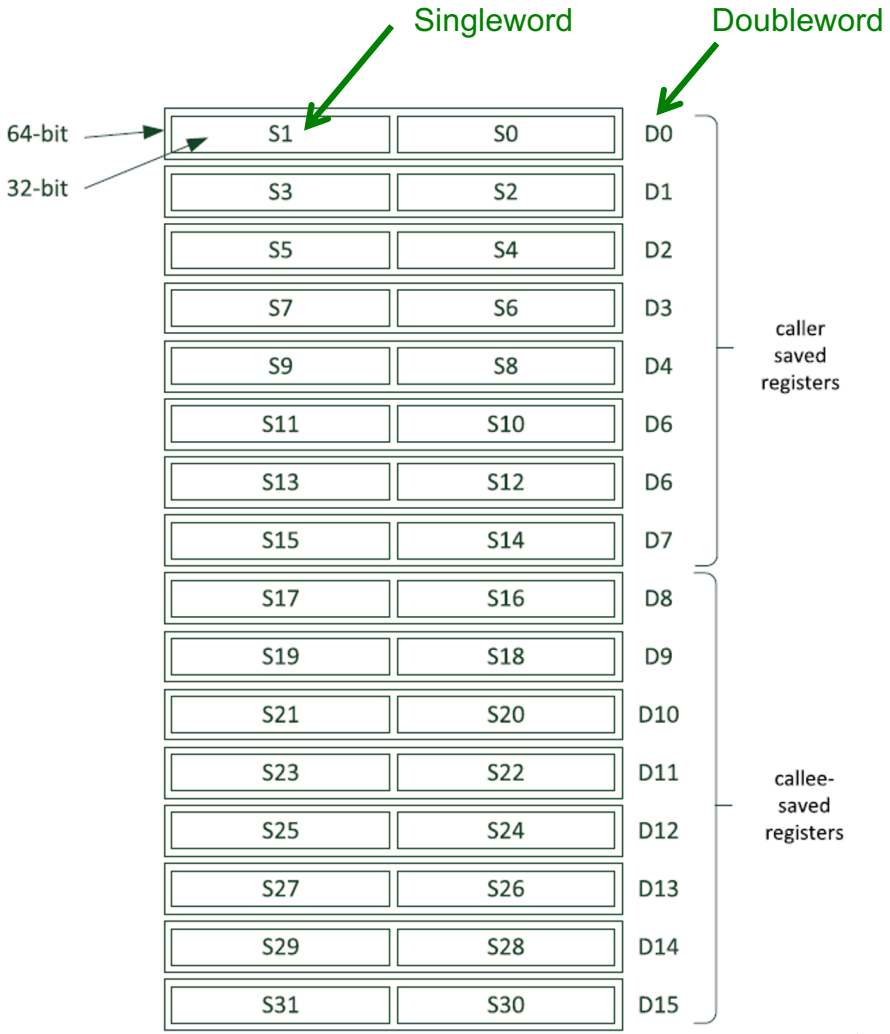
Test.s

Gestion des nombres à virgules

La FPU contient ses propres registres où seront stockés les variables de types float (32 bits) et double (64 bits).

A partir de ces registres, la FPU effectuera les opérations de bases, comme les multiplications, divisions, en utilisant des mécanismes hardware.

Attention aux float en interrupt, ces registres doivent être sauvegardés!



Gestion des nombres à virgules

Operation	Description	Assembler	Cycles
Move	top/bottom half of double to/from core register	VMOV	1
	immediate/float to float-register	VMOV	1
	two floats/one double to/from two core registers or one float to/from one core register	VMOV	2
	floating-point control/status to core register	VMRS	1
	core register to floating-point control/status	VMSR	1
Multiply	float	VMUL.F32	1
	then accumulate float	VMLA.F32	3
	then subtract float	VMLS.F32	3
	then accumulate then negate float	VNMLA.F32	3
	then subtract then negate float	VNMLS.F32	3

Cortex-M4
Technical Reference Manual

Gestion des nombres à virgules (double)

arm-none-eabi-gcc -save-temps=obj -mcpu=cortex-m4 -c essai.c -o essai.o -mfloat-abi=hard -mfpu=vfpv2

```
int main() {

        const double PI = 3.1415;
        double mul = 2.0;

        double result = mul * PI;

        return 0;
}
```

Test.c

Compilation

```
main:
        push    {r4, r7}
        sub     sp, sp, #24
        add     r7, sp, #0
        adr     r4, .L3
        ldrd    r3, [r4]
        strd    r3, [r7, #16]
        mov     r3, #0
        mov     r4, #1073741824
        strd    r3, [r7, #8]
        vldr.64 d17, [r7, #8]
        vldr.64 d16, [r7, #16]
        vmul.f64 d16, d17, d16
        vstr.64 d16, [r7]
        ...
        pop     {r4, r7}
        bx      lr
.L4:
        .align  3
.L3:
        .word   -1065151889
        .word   1074340298
```

Test.s

Conversion float to double

Le choix d'utiliser une variable de type double ou float dépendra de la précisions qu'on désire obtenir, ou la grandeur des nombres obtenus. Les conversions float <-> double sont aussi traîtées de manière hardware par défaut sur les processeurs:

```
int main() {

            const float PI = 3.1415;
            double mul = 2;

            double result =  PI;

            return 0;
}
```

float

```
ldr         r3, .L3
str         r3, [r7, #20]           @

mov         r3, #0
mov         r4, #1073741824
strd        r3, [r7, #8]
vldr.32     s15, [r7, #20]
vcvt.f64.f32 d16, s15
vstr.64     d16, [r7]
movs        r3, #0
```

Calculs complexes

Même avec une FPU, un calcul de type sin() ne pourra pas s'effectuer entièrement hardware, une partie de la génération du sinus se fera de manière software.

```
#include <math.h>

int main() {

            const double PI = 3.1415;
            double mul = 2.0;

            double result = sin(mul * PI);

            return 0;
}
```

```
mov         r3, #0
mov         r4, #1073741824
strd        r3, [r7, #8]
vldr.64     d17, [r7, #8]
vldr.64     d16, [r7, #16]
vmul.f64    d16, d17, d16
vmov.f64    d0, d16
bl          sin
vstr.64     d0, [r7]
```

Représentation des nombres: Tables stockées en mémoire

Un moyen d'éviter cela est de générer une Lookup table.

```
int i, sinus[360];
…
for(i=0;i<360;i++)
{
    sinus[i] = 10*sin(i/180*3.1415);
    printf("sin %d =
%d\n",i,sinus[i]);
}
```

Représentation des nombres: changement d'échelle

Même si, en activant la FPU, il est possible d'effectuer des opérations à virgule flottante de manière très rapide sur le microcontrôleur STM32F4, pas toujours c'est necessaire (consummation, type de µC): il existe certaines techniques pour se passer des nombres à virgules, comme le changement d'échelle.

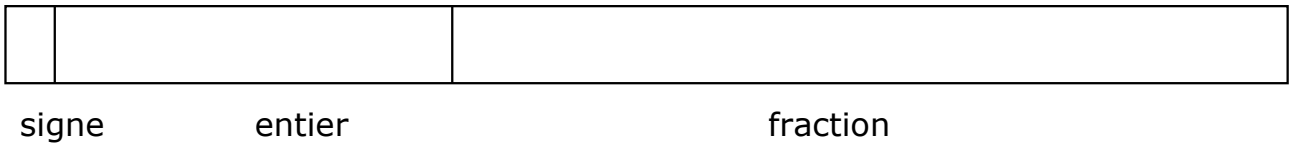
$$0.01 = 1\%$$

$$1.34m = 1340mm$$

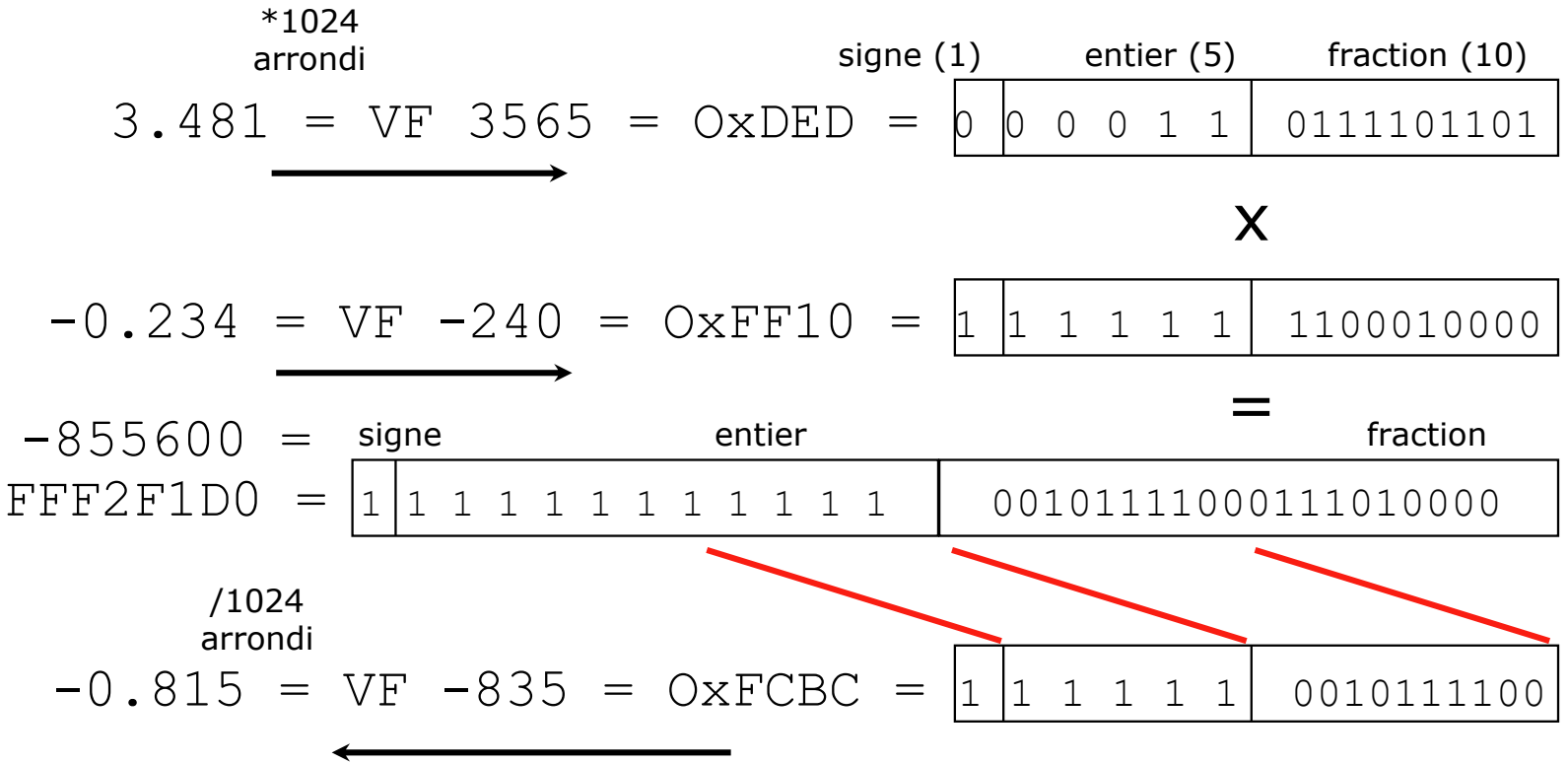
Représentation des nombres: Virgule fixe

Un autre moyen d'éviter l'utilisation de nombre float est de travailler avec les nombres à virgule fixe.

Représentation des nombres: virgule fixe



Calcul avec nombres en virgule fixe: multiplication



Gestion des nombres à virgules et des cast

Enfin, il est très important de se souvenir que certains casts sont effectués de manière implicite en fonction des types de variables.

```
int main() {

        const float PI = 3.1415;
        int mul = 2;

        int result = mul * PI;

        return 0;
}
```

Test.c

Compilation

```
ldr         r3, .L3
str         r3, [r7, #12]
@ float
movs        r3, #2
str         r3, [r7, #8]
ldr         r3, [r7, #8]
vmov        s15, r3     @ int
vcvt.f32.s32 s14, s15
vldr.32     s15, [r7, #12]
vmul.f32    s15, s14, s15
vcvt.s32.f32 s15, s15
vmov        r3, s15     @ int
str         r3, [r7, #4]


.L3:
.word       1078529622
```

Test.s

Gestion des nombres à virgules et des cast

La manière dont on “cast” une variable ou une opération va impacter le type de l’opération effectuée utilisant cette variable.

```
int main() {

  const float PI = 3.1415;
  int mul = 2;

  int result = (int) mul * PI;

  return 0;
}
```

Test.c

Compilation

Ceci ne change rien

```
ldr          r3, .L3
str          r3, [r7, #12]
@ float
movs         r3, #2
str          r3, [r7, #8]
ldr          r3, [r7, #8]
vmov         s15, r3     @ int
vcvt.f32.s32 s14, s15
vldr.32      s15, [r7, #12]
vmul.f32     s15, s14, s15
vcvt.s32.f32 s15, s15
vmov         r3, s15     @ int
str          r3, [r7, #4]


.L3:
.word        1078529622
```

Test.s

Gestion des nombres à virgules et des cast

La manière dont on “cast” une variable ou une opération va impacter le type de l’opération effectuée utilisant cette variable.

```
int main() {

  const float PI = 3.1415;
  int mul = 2;

  int result =  mul * (int)PI;

  return 0;
}
```

Test.c

Compilation

```
ldr          r3, .L3
str          r3, [r7, #12]            @ float

movs         r3, #2
str          r3, [r7, #8]
vldr.32      s15, [r7, #12]
vcvt.s32.f32 s15, s15
vmov         r2, s15      @ int
ldr          r3, [r7, #8]
mul          r3, r2, r3
str          r3, [r7, #4]


.word        1078529622
```

Ceci ramène le calcul en int, potentiellement perte de précision

Test.s

Gestion des nombres à virgules et des cast

La manière dont on “cast” une variable ou une opération va impacter le type de l’opération effectuée utilisant cette variable.

```
int main() {

    const float PI = 3.1415;
    int mul = 2;

    int result =  (int)(mul *PI);

    return 0;
}
```

Test.c

Compilation

Ceci ne change rien

```
ldr         r3, .L3
str         r3, [r7, #12]           @
                                    float
movs        r3, #2
str         r3, [r7, #8]
ldr         r3, [r7, #8]
vmov        s15, r3     @ int
vcvt.f32.s32 s14, s15
vldr.32     s15, [r7, #12]
vmul.f32    s15, s14, s15
vcvt.s32.f32 s15, s15
vmov        r3, s15     @ int
str         r3, [r7, #4]


.L3:
.word       1078529622
```

Test.s

Conseils sur la gestion des variables

Les microcontrôleurs (microprocesseurs) offrent des capacités toujours plus grandes pour ce qui est de la vitesse des calculs, grâce à la vitesse du processeur et à des unités comme la FPU ou le DSP, mais aussi de mémoire avec des capacités grandissante en RAM et en Flash. Ceci est accompagné d'une consummation plus grande.
Il est donc toujours utile d'optimiser son code pour obtenir de meilleurs résultats à l'execution, une reduction de la consummation et une meilleure intégration sur d'autres systèmes.

C'est pourquoi, pour les TPs et les miniprojets, pensez à utiliser les bons types de variables et de la manière la plus optimale possible! (voir checklist de la semaine passée)

RTOS
Real-Time Operating System

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

Introduction

Dans cette partie du cours, nous allons voir à quoi sert et comment fonctionne un système RTOS pour les systèmes embarqués.

Cela va impliquer l'introduction de concepts comme:

- Multi-tâches / multi-threading
- Scheduling

Introduction: programmation concurrente et temps réel

Programmation concurrente:
par opposition à la programmation séquentielle, elle est composée de processus en parallèle qui interagissent, utilisent des ressources communes etc.

Un robot qui a un contrôle de moteur, un processus de lecture de capteurs et un algorithme de contrôle globale est par nature “concurrent” avec des processus essentiellement asynchrones.

Mais: la mise à jour du PWM dans un profil de vitesse ne peut pas se faire “quand le processeur a du temps”. Il faut respecter des timing, faire du “temps réel”.

Introduction: le OS (operating system)

Le système opératif est un ensemble de couches logicielles qui fournissent des services aux programmeurs / applications:

- gestion de la mémoire
- gestion des entrées-sorties
- gestion des fichiers
- gestion des tâches
- gestion du temps (réel)

Systèmes d'exploitation en temps réel

Un RTOS est un système opérationnel dont les processus internes sont garantis d'être en accord avec les contraintes temps-réels hardwares et/ou logicielles. Un RTOS met plus de priorité sur la **fiabilité du timing** des tâches que sur la quantité de tâches réalisées. Les deux qualités fondamentales d'un système RTOS sont:

- Déterministe: La qualité de produire de manière constantes les mêmes types de résultats en fonctions des mêmes types de conditions.
- Prédictibilité: La qualité d'être prédictible avec un comportement programmé

Attention: Les systèmes RTOS sont souvent confondus avec les systèmes opérationnels "rapides". Même si l'efficacité est un des atout principal des RTOS, l'efficacité seule ne qualifie pas un OS pour être un RTOS

From http://www.chibios.org

Ceux-ci ne sont pas des RTOS...

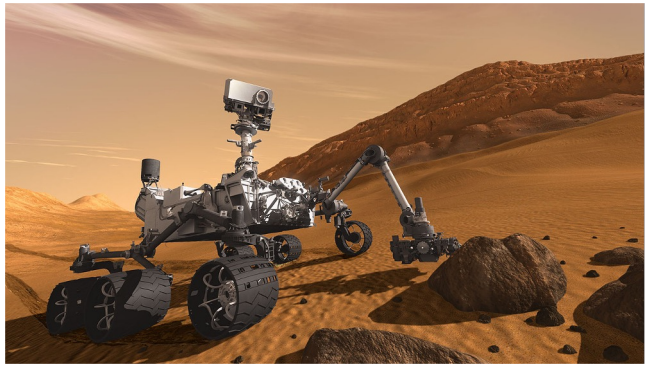
Exemples d'application des RTOS

Systèmes d'exploitation en temps réel et multi-tâches

Dans le domaine des systèmes embarqués, les RTOS sont très couramment utilisées. De nos jour, un robot doit en effet rarement effectuer une seule tâche, et ces tâches doivent être effectuées à des timing assez précis.

Exemple: Robot sur Mars (Curiosity)

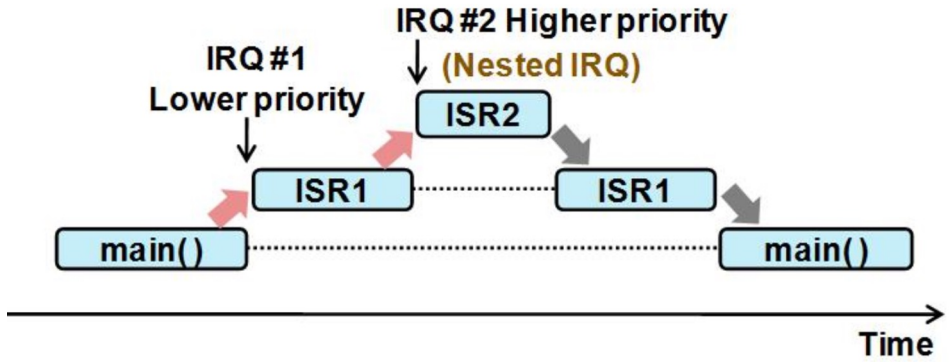
- Contrôler les moteurs pour avancer
- Capturer une vidéo de 5 minutes
- Communiquer avec la Terre à 10:00
- ...



Systèmes d'exploitation en temps réel et multi-tâches

Au niveau des systèmes embarqués, les RTOS se basent sur le système Timer/Interrupt qui permet de réaliser des tâches avec des timing précis (TP1: Led qui clignote à 1 seconde).

Les interrupts ont aussi des niveaux de priorités configurables, ce qui permet de pouvoir déterminer si une tâche à une priorité sur une autre, et peut parfois l'interrompre.



https://community.arm.com

Systèmes d'exploitation en temps réel et multi-tâches

La gestion de ces divers tâches peut vite devenir très compliquée suivant leurs nombres, complexités et le timing à respecter.

L'interaction entre tâches est un problème classique:
imaginez un processus (A) qui fait la lecture de l'image de la caméra et la stocke en mémoire, et un autre (B) qui de temps en temps vient prendre l'image pour en faire l'analyse.

- Si B vient lire l'image pendant que A la met à jour en mémoire, A aura une image corrompue (moitié la nouvelle et moitié la vieille, par exemple).
- Si B est plus rapide que A, il va faire des analyses pour rien, car il va lire plusieurs fois la même image.

Systèmes d'exploitation en temps réel et multi-tâches

Dans la suite du cours et des travaux pratiques, nous allons donc travailler avec un système d'exploitation en temps réels implémenté sur STM32F4 qui va permettre de gérer à un niveau supérieur la programmation multi-tâche.

Ce système s'appelle **ChibiOS**. C'est un système Open Source avec une implémentation sur divers plateformes, dont le STM32F4.
(voir http://www.chibios.org pour la documentation complète)

ChibiOS

Les travaux pratiques 3, 4 et 5 vont se baser sur ChibiOS.

La gestion des différents périphériques du robot e-puck2 (IMU, caméra, microphone, moteurs, etc.) se fera sur la base de cette librairie, afin de faciliter la gestion de plusieurs tâches à la fois tout en respectant le bon timing.

Les miniprojets se feront aussi sur la base de cette même librairie, afin d'exploiter les possiblités multi-tâches et temps réel.

Limitation sans Threads

```
void move(){
        //robot move the motors
}

void take_picture(){
        //robot take a picture
}

void communicate(){
        //robot communicate to Earth
}

void main(){
        while(1){
                move();
                take_picture();
                communicate();
                wait(5); //wait 5sec
        }
}
```

Base des systèmes multi-tâches

En informatique, la programmation multi-tâche est très commune.

La construction d'un code multi-thread est en principe la suivante:

```
void thread1(){
        //robot move the motors
}

void thread2(){
        //robot take a picture
}

void thread3(){
        //robot communicate to Earth
}

void main(){
        //initializations
}
```

Chacunes des tâches est gérée par sa propre Thread, ce qui permet de mieux les synchroniser Et gérer les niveaux de priorités

Thread en C : initialisation par crt0_v7m.s

```
/*
    ChibiOS - Copyright (C) 2006..2015 Giovanni Di Sirio.

    This file is part of ChibiOS.

    ChibiOS is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 3 of the License, or
    (at your option) any later version.

    ChibiOS is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see http://www.gnu.org/licenses/.
*/


/**
 * @file    crt0_v7m.s
 * @brief   Generic ARMv7-M (Cortex-M3/M4/M7) startup file for ChibiOS.
 *
 * @addtogroup ARMCMx_GCC_STARTUP_V7M
 * @{
 */

/*===========================================================================*/
/* Module constants.                                                         */
/*===========================================================================*/

...
```

Code de démarrage

Thread en C : crt0_v7m.s

```
/*===========================================================================*/
/* Module pre-compile time settings.                                         */
/*===========================================================================*/

...

/*===========================================================================*/
/* Code section.                                                             */
/*===========================================================================*/

...

/*
 * Reset handler.
 */
                .align  2
                .thumb_func
                .global Reset_Handler

Reset_Handler:

                /* Interrupts are globally masked initially.*/
                cpsid   i

                /* PSP stack pointers initialization.*/
                ldr     r0, =__process_stack_end__
                msr     PSP, r0
```

Process Stack Pointer(PSP) Initialisation en plus du main stack

Thread en C : crt0_v7m.s

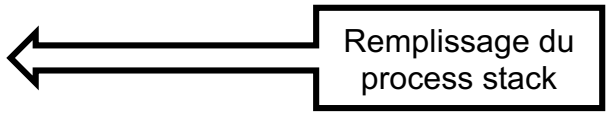
```
#if CRT0_INIT_FPU == TRUE
                /* FPU FPCCR initialization.*/
                movw    r0, #CRT0_FPCCR_INIT & 0xFFFF
                movt    r0, #CRT0_FPCCR_INIT >> 16
                movw    r1, #SCB_FPCCR & 0xFFFF
                movt    r1, #SCB_FPCCR >> 16
                str     r0, [r1]
...

#endif

                /* CONTROL register initialization as configured.*/
                msr     CONTROL, r0
                isb

                /* Early initialization..*/
                bl      __early_init

#if CRT0_INIT_STACKS == TRUE

                ldr     r0, =CRT0_STACKS_FILL_PATTERN
                /* Main Stack initialization. Note, it assumes that the
                   stack size is a multiple of 4 so the linker file must
                   ensure this.*/

                ldr     r1, =__main_stack_base__
                ldr     r2, =__main_stack_end__
Msloop:
                cmp     r1, r2
                itt     lo
                strlo   r0, [r1], #4
                blo     msloop
```

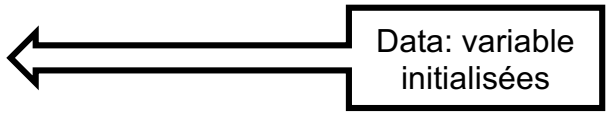
Se trouve dans crt1.s

Remplissage du main stack

Thread en C : crt0_v7m.s

```
                    /* Process Stack initialization. Note, it assumes that the
                       stack size is a multiple of 4 so the linker file must
                       ensure this.*/

                    ldr     r1, =__process_stack_base__
                    ldr     r2, =__process_stack_end__
Psloop:
                    cmp     r1, r2
                    itt     lo
                    strlo   r0, [r1], #4
                    blo     psloop
#endif

#if CRT0_INIT_DATA == TRUE
                    /* Data initialization. Note, it assumes that the DATA size
                      is a multiple of 4 so the linker file must ensure this.*/

                    ldr     r1, =_textdata
                    ldr     r2, =_data
                    ldr     r3, =_edata
Dloop:
                    cmp     r2, r3
                    ittt    lo
                    ldrlo   r0, [r1], #4
                    strlo   r0, [r2], #4
                    blo     dloop

#endif
```





Thread en C : crt0_v7m.s

```
#if CRT0_INIT_BSS == TRUE
                /* BSS initialization. Note, it assumes that the DATA size
                   is a multiple of 4 so the linker file must ensure this.*/

                movs    r0, #0
                ldr     r1, =_bss_start
                ldr     r2, =_bss_end
Bloop:
                cmp     r1, r2
                itt     lo
                strlo   r0, [r1], #4
                blo     bloop
#endif
                /* Late initialization..*/
                bl      __late_init

#if CRT0_CALL_CONSTRUCTORS == TRUE

                /* Constructors invocation.*/
                ldr     r4, =__init_array_start
                ldr     r5, =__init_array_end
Initloop:
                cmp     r4, r5
                bge     endinitloop
                ldr     r1, [r4], #4
                blx     r1
                b       initloop
endinitloop:

#endif
                /* Main program invocation, r0 contains the returned value.*/
                bl      main
```

Initialisation à zéro

Se trouve dans crt1.s

Le main est un process, lancé par crt0

Thread en C : crt0_v7m.s

```
#if CRT0_CALL_CONSTRUCTORS == TRUE

                /* Destructors invocation.*/
                ldr     r4, =__fini_array_start
                ldr     r5, =__fini_array_end
Finiloop:
                cmp     r4, r5
                bge     endfiniloop
                ldr     r1, [r4], #4
                blx     r1

                b       finiloop

endfiniloop:

#endif
                /* Branching to the defined exit handler.*/

                b       __default_exit

#endif /* !defined(__DOXYGEN__) */

/** @} */
```

Il y a une vie après le main!

Se trouve dans crt1.s, boucle infinie

Thread en C

```
void thread1(){
    int a;
    while(1){
        a = read_sensor();
        sleep(1000);
}
```

Type de la Thread

Espace contenant par exemple des déclarations de variables.

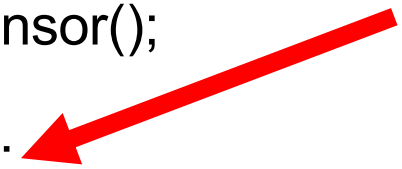
Le bloc de “base” composé en principe d’une boucle infinie

La tâche à effectuer

Une indication de la fréquence d’appel de cette tâche

Thread en C

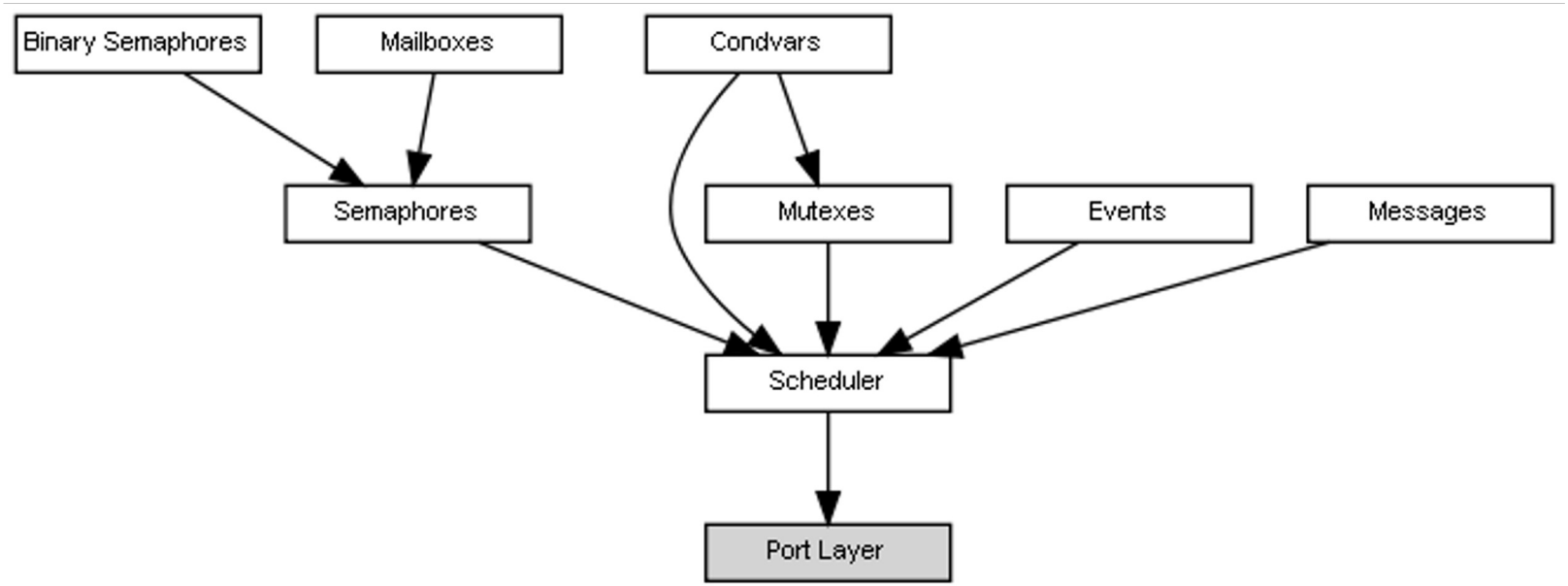
```
void thread1(){
    int a;
    while(1){
        a = read_sensor();
        sleep(1000);
}
```



Le sleep() ici n'est pas une boucle d'attente pendant laquelle on ne fait rien d'autre. Cela indique que la thread thread1 va entrer dans l'état sleep pendant un certains temps, et sera ensuite à nouveau exécutée après ce laps de temps. Pendant ce temps, d'autres tâches (threads) peuvent s'effectuer.

Architecture de ChibiOS

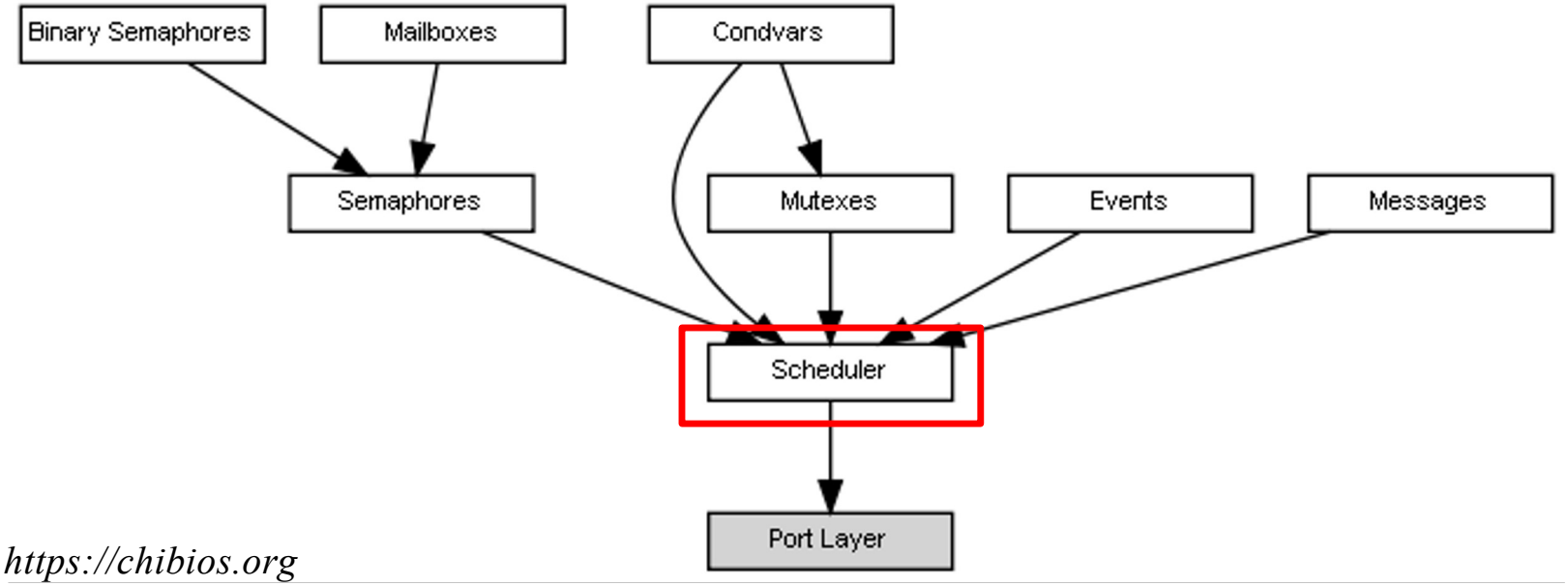
ChibiOS, comme tout OS, comporte un **noyau** (kernel) permettant la gestion du système de manière abstraite pour l'utilisateur.



https://chibios.org

Architecture de ChibiOS

On ne va pas voir tous les composants aujourd'hui en détail, mais notez que tous les composants de ce noyau semblent converger vers un élément en particulier qui est le **scheduler.**



https://chibios.org

Scheduling

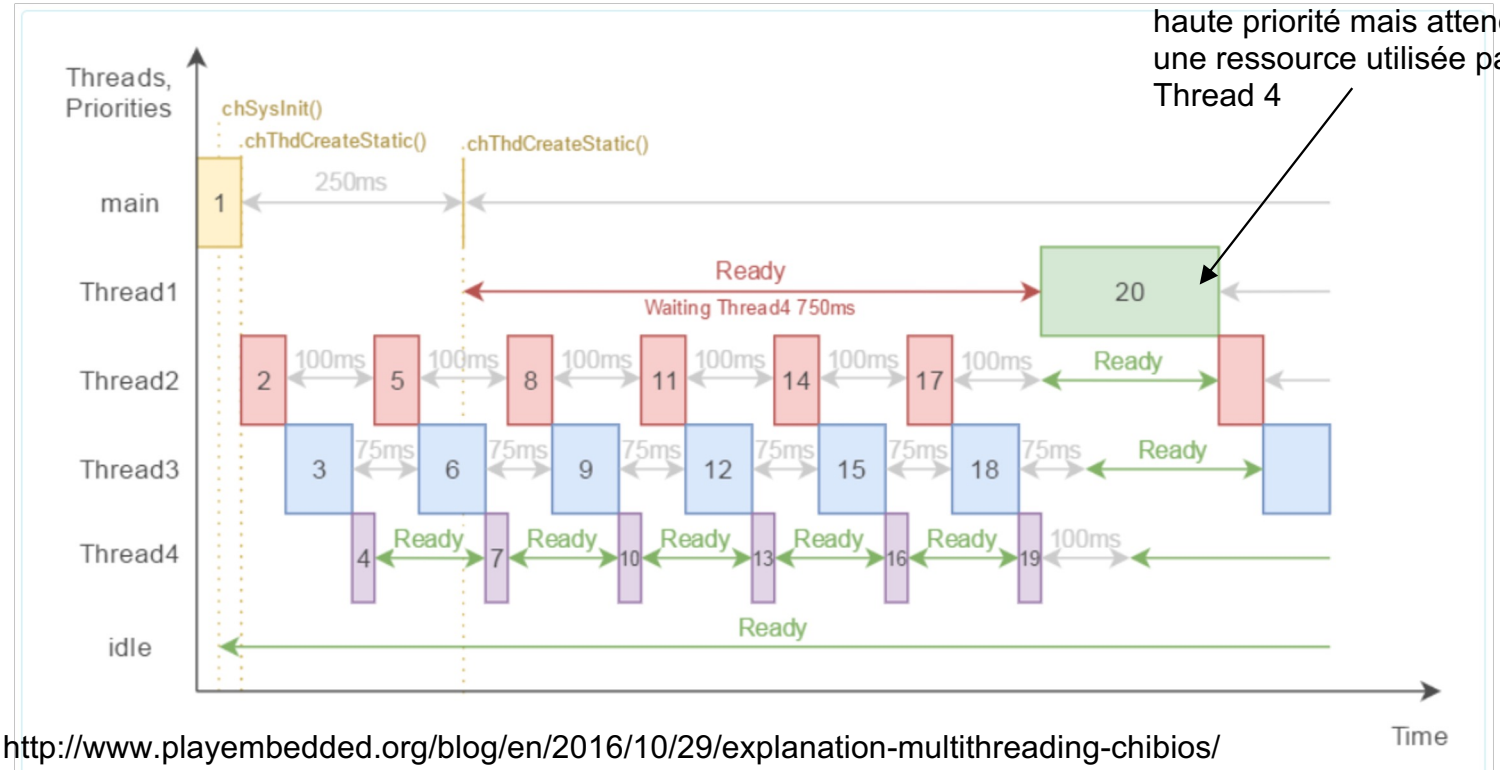
Sur un processseur avec un seul coeur, on peut seulement exécuter une tâche à la fois. Comment donc faire du multi-tâche? En alternant les tâches tout en tenant compte de leur priorité.

ChibiOS va justement permettre de gérer différentes tâches qui tournent “en parallèle”, en respectant les timings imposés et les priorités. C’est cela qui s’appelle en bon français du **Scheduling.**

Une fois que les tâches sont définies, avec des niveaux de priorités et des timing bien précis, le RTOS va s’occuper de faire la programmation de l’exécution de ces tâches.

Scheduling

Exemple d'un (mauvais) planning pour un système avec 4 tâches avec différents rythmes et utilisations de ressources.



http://www.playembedded.org/blog/en/2016/10/29/explanation-multithreading-chibios/

Exemple de déclaration de Thread avec ChibiOS

```
#include <ch.h>

/*
 * Working area for the LED flashing thread.
 */
static THD_WORKING_AREA(myThreadWorkingArea, 128);

/*
 * LED flashing thread.
 */
static THD_FUNCTION(myThread, arg) {

  while (true) {
    LED_ON();
    chThdSleepMilliseconds(500);
    LED_OFF();
    chThdSleepMilliseconds(500);
  }
}

int main(int argc, char *argv[]) {

  /* Starting the flashing LEDs thread.*/
  (void)chThdCreateStatic(myThreadWorkingArea, sizeof(myThreadWorkingArea),
                          NORMALPRIO, myThread, NULL);
  .
  .
  .
}
```

macro

Code faisant clignoter une LED, avec 500ms d'intervalles entre chaque changement d'état de la LED

Exemple avec ChibiOS

```
#include <ch.h>

/*
 * Working area for the LED flashing thread.
 */
static THD_WORKING_AREA(myThreadWorkingArea, 128);

/*
 * LED flashing thread.
 */
static THD_FUNCTION(myThread, arg) {

  while (true) {
    LED_ON();
    chThdSleepMilliseconds(500);
    LED_OFF();
    chThdSleepMilliseconds(500);
  }
}

int main(int argc, char *argv[]) {

  /* Starting the flashing LEDs thread.*/
  (void)chThdCreateStatic(myThreadWorkingArea, sizeof(myThreadWorkingArea),
                          NORMALPRIO, myThread, NULL);
  .
  .
  .
}
```

Allocation de la mémoire pour ce Thread

Fonctionnalité du Thread

Function main

Exemple avec ChibiOS

```
#include <ch.h>

/*
 * Working area for the LED flashing thread.
 */
static THD_WORKING_AREA(myThreadWorkingArea, 128);

/*
 * LED flashing thread.
 */
static THD_FUNCTION(myThread, arg) {

  while (true) {
    LED_ON();
    chThdSleepMilliseconds(500);
    LED_OFF();
    chThdSleepMilliseconds(500);
  }
}

int main(int argc, char *argv[]) {

  /* Starting the flashing LEDs thread.*/
  (void)chThdCreateStatic(myThreadWorkingArea, sizeof(myThreadWorkingArea),
                          NORMALPRIO, myThread, NULL);
  .
  .
  .
}
```

On alloue de la mémoire pour **la stack de ce thread**, sur laquelle on peut stocker les variables propres à cette thread

Qu'est-ce qui finit sur la stack?

Allocation de la mémoire pour ce Thread

Fonctionnalité du Thread

La définition de l'attente avant un nouvel appel à cette thread. Durant ce temps, d'autres Threads peuvent être exécutées!!!

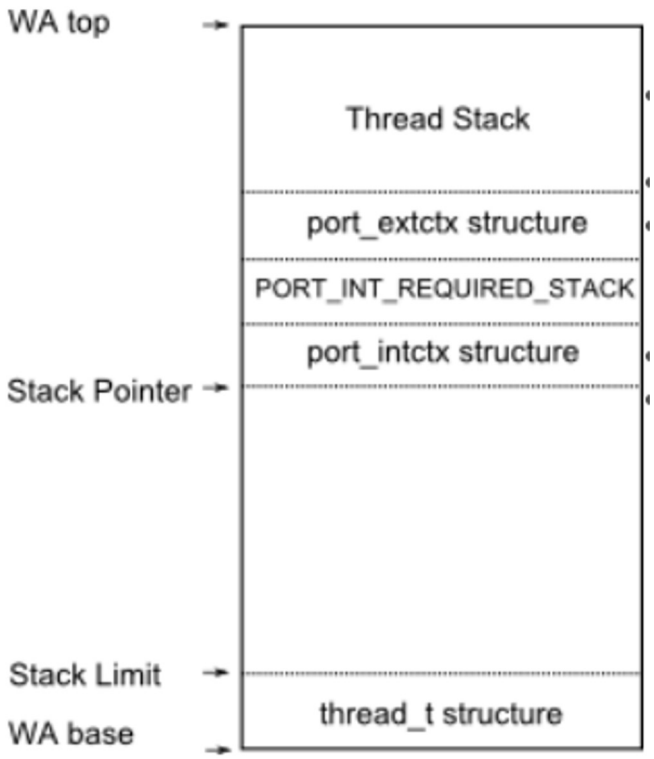
Et si on ne met pas de sleep???

Function main

Création/initialisation du Thread dans la fonction main

Thread Working Area

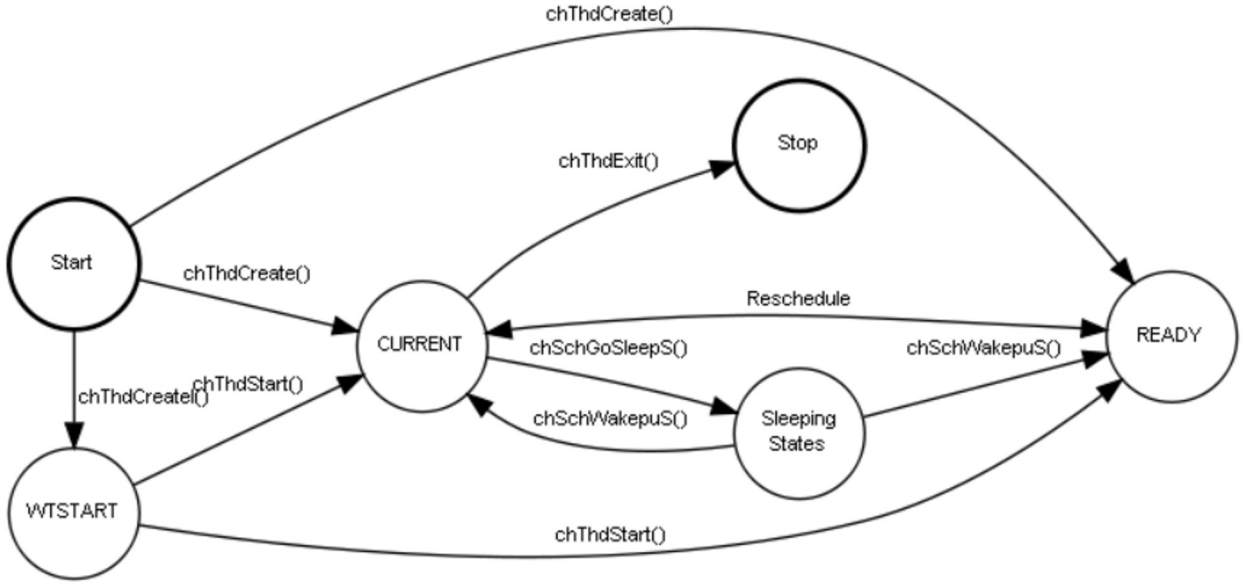
Lors de la création d'une Thread, on alloue de la mémoire pour le stack qui permet de stocker toutes les informations liées à cette Thread. Notamment les variables locales de cette Thread, les variables de gestions des interrupts (timing, priorités), les variables des sous-fonctions, etc.



https://chibios.org

Les Threads avec ChibiOS

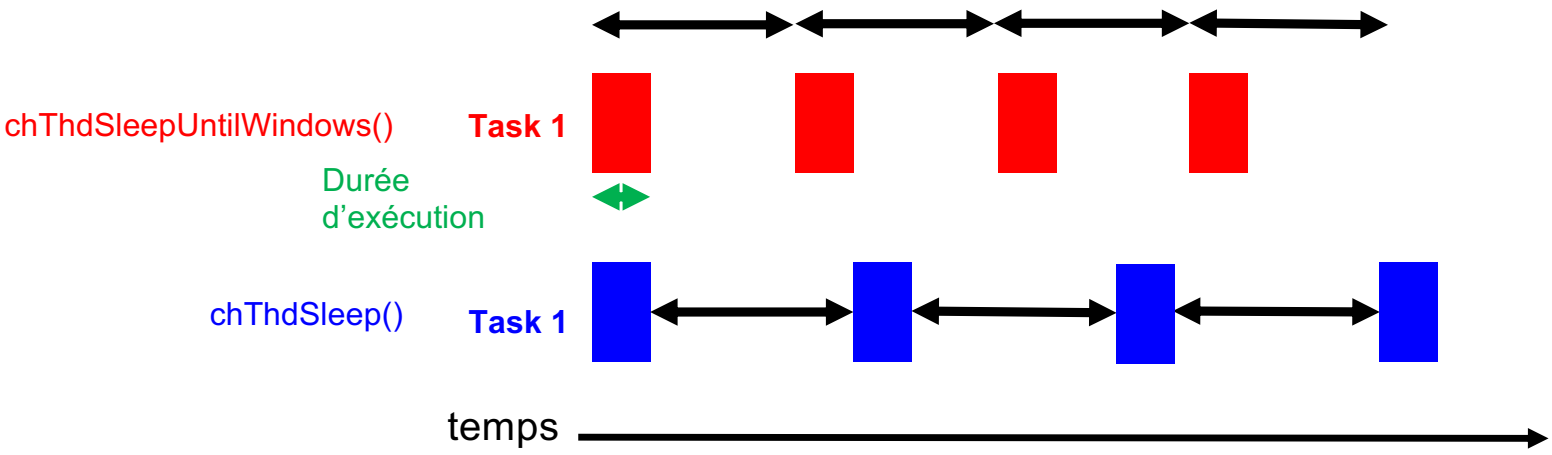
La durée de vie d'une Thread sur ChibiOS est dictée par une machine d'état. On verra qu'il est possible pendant l'exécution d'un programme de créer des Thread, de les supprimer, de les mettre en attente, etc.



https://chibios.org

Delay vs deadline

Il y deux principaux moyens de mettre un thread en attente. Une relative à la durée de Thread, et une ne la prenant pas en compte (TP3).



Application typique en robotique

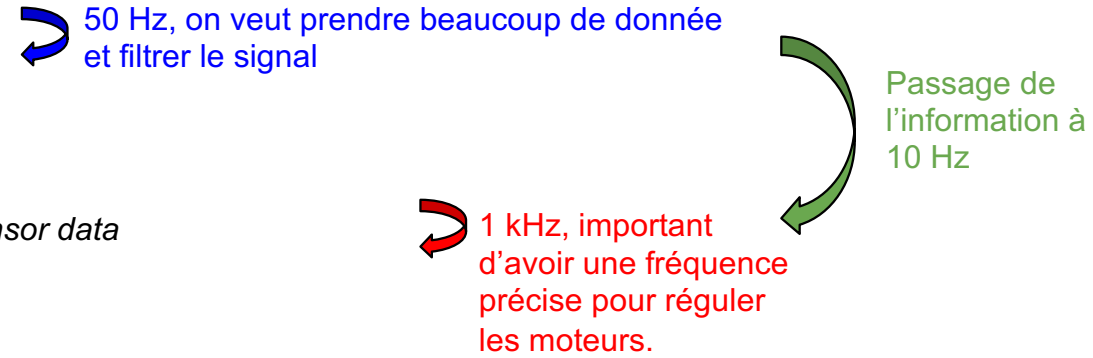
Un robot doit capter des données d'un capteur (ex: accéléromètre), et les utiliser pour contrôler un moteur.

```
void getSensor(){
        //robot get sensor data
}

void controlActuator(){
        //robot control the motors in function of the sensor data
}


void main(){
        //initializations
}
```

Application typique en robotique

Un robot doit capter des données d'un capteur (ex: accéléromètre), et les utiliser pour contrôler un moteur.

```
void getSensor(){
   //robot get sensor data
}

void controlActuator(){
   //robot control the motors in function of the sensor data
}


void main(){
                  //initializations
}
```



Lors du TP3, vous verrez un moyen efficace de transmettre des messages entre des Threads (librairie MessageBus)

TP de cette semaine: TP3

Prise en main de ChibiOS/RT.

Implémentation et gestion de Threads

Gestion de l'IMU (accéléromètre + gyroscope)

1 / 5 | 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):

Je trouve que le TP de cette semaine était assez dur à comprendre et que il nous aurait peut être fallu plus d'étapes d'intermédiaires afin de se trouver moins dépourvu face au tâches encore jamais réalisée. Sinon super le système des assistants marche bien

<< First Respondent | < Previous | 2 / 5 | 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 système d'appel des assistants est très efficace. Il faudrait si possible mettre à disposition sur moodle les slides d'introduction des TPs avant/au début de ceux-ci. Certaines instructions sont parfois vagues concernant les TPs, mais d'un côté cela nous pousse à aller chercher par nous même.

<< First Respondent | < Previous | 3 / 5 | 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):

J'apprécie l'effort visible fourni pour organiser les TP, beaucoup d'éléments sont prévus pour nous aider, ce qui est bien, merci ! Cependant, je me suis trouvé complètement perdu tout le long de ce TP. Je pense que les README.md des TP ne sont pas assez précis en général (surtout concernant les registres (leur rôle par exemple), et ce même avec les hints). Le lien entre le README et le code ne m'est pas toujours très clair (on me demande de créer une fonction qui existe déjà, ou un assistant me dit que dans le corrigé une fonction est créée alors que ce n'est pas demandé)

<< First Respondent | < Previous | 4 / 5 | 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 | 5 / 5
<<< 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):

Les TPs ne sont pas assez guidés : on ne comprend pas exactement l'utilité de chaque registre, leur fonctionnement ni comment les configurer, malgré la datasheet. Certaines informations essentielles à une bonne compréhension semblent manquer. Peut-être que des explications supplémentaires lors de l'introduction ou la mise à disposition de la correction pendant le TP pourraient aider à mieux assimiler les concepts.