Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 & 2

a.13 – Interfaçage assembleur - C

Classes T-2/I-2 // 2018-2019

Daniel Gachet | HEIA-FR/TIC
a.13 | 16.11.2018

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Contenu

- Introduction
- Appel de sous-routines
- Passage des paramètres
- Altération des registres
- Conventions

Le langage de programmation C permet d'accéder très facilement aux registres des périphériques. Cependant, certaines ressources internes du microprocesseur (registres, registres spéciaux, interruptions, …) ne sont accessibles que par l'intermédiaire de l'assembleur. Il est donc courant que des routines développées en assembleur doivent interfacer des méthodes écrites en C et vice-versa.

- **Points à résoudre**
  - Appel du sous-programme
  - Retour au programme principal (sauvegarde de l'adresse de retour)
  - Passage des paramètres d'entrée et de sortie et du résultat
  - Altération des registres
  - Déploiement en bibliothèques par des systèmes multitâches

Appel d'un sous-programme

- **Le processeur ARM fournit deux instructions pour l'appel de sous-routines**
  - **BL <target_address>** → adressage direct ±32MB
  - **BLX <Rx>** → adressage indirect

- **Ces instructions sauvent automatiquement la valeur du PC pointant sur la prochaine instruction à exécuter dans le registre R14. R14 est également appelé « Link Register » ou « LR ».**
- **Après que le programme ait transféré le contrôle au sous-programme, l'exécution suit l'ordre normal des instructions. Le sous-programme contient l'instruction finale BX LR (ou MOV PC, LR, ou LDMFD SP!, {PC}) qui redonne le contrôle au programme, lequel poursuit son exécution.**

```
                mov     r0, #10     @initialisation du 1er paramètre
                bl      subroutine  @appel de la sous-routine
                ldr     r1,=val     @instruction suivante
                str     r0,[r1]
                ...

subroutine:     mov     r1,r0       @début de la sous-routine
                ...
                bx      lr          @fin de la sous-routine(retour)
```

Technique pour le passage des paramètres (rappel)

- **Deux technique sont utilisées lorsqu'un programme passe les paramètres à un sous-programme :**
  - Passage par valeur
  - Passage par référence (adresse)

- **La technique de passage par valeur (call by value) donne une copie de la donnée actuelle:**
  - Le sous-programme reçoit une copie de la donnée.
  - Il peut la lire et la modifier sans que la donnée originale ne soit altérée
  - Il existe deux positions mémoires distinctes
  - La modification de l'une des positions n'a aucune conséquence sur l'autre position

- **La technique de passage par référence (call by reference) donne l'adresse de la donnée :**
  - Le sous-programme reçoit l'adresse de la donnée
  - Cette donnée est ainsi partagée entre le programme appelant et le sous-programme
  - La donnée ne se situe que dans une seule position mémoire
  - La modification du paramètre par le sous-programme est visible par le programme appelant

Mécanismes pour le passage des paramètres

Une part importante de l'écriture d'un sous-programme est de déterminer le mécanisme à utiliser pour le passage des paramètres

- **Deux mécanismes principaux**
  - Paramètres passés par les registres
    - Les registres offrent une méthode rapide pour le passage des paramètres. Cependant, cette méthode est limitée par le nombre de registres du microprocesseur.
  - Paramètres passés par la pile
    - Le passage de paramètres par la pile offre une approche généralisée. Cet avantage est contrebalancé par la lenteur de la technique (échange de données avec la mémoire)
- **Le choix d'une approche plutôt qu'une autre dépend fortement des possibilités du microprocesseur, du langage de programmation utilisé et de l'application à développer.**
- **Les compilateurs modernes utilisent couramment les deux mécanismes**

Passage des paramètres par les registres

- **Les paramètres d'un sous-programme peuvent être passés à travers les registres internes du microprocesseur (R0 à R3).**
- **Le sous-programme utilise la même méthode pour le retour du résultat**
- **Avec le passage de paramètres par registres, le programmeur doit s'assurer que chaque paramètre soit bien placé dans le registre défini par la convention de passage**
- **Les registres alloués pour chaque paramètre sont généralement définis dans les commentaires débutant le sous-programme, p. ex.**

```
int foo (const void* msg, void* dest);
```

  - R0 : adresse de msg
  - R1 : adresse de dest
  - R0 : valeur de retour

Passage des paramètres par la pile

- L'une des méthodes les plus utilisées pour le passage de paramètres à un sous-programme consiste à placer ces paramètres sur la pile
- Le passage de paramètres par la pile suppose une convention précisant l'ordre dans lequel sont placés ces paramètres. Le programme appelant place sur la pile les paramètres dans l'ordre préalablement défini. Le programme appelé accède à ces valeurs en utilisant un adressage indirect par registre avec déplacement.
- Afin de rendre la pile propre au retour, les paramètres doivent ensuite être détruits

- Programme appelant :

```
main:   sub     sp, #4          @réserve l'espace sur la pile
        ldr     r0,=val
        str     r0,[sp, #0]     @place le paramètre sur la pile
        bl      subroutine
        add     sp, #4          @restaure la pile
```

- Programme appelé :

```
subroutine:     ...
        ldr     r0, [sp, #0]    @prend la valeur du paramètre
        ...
        bx      lr
```

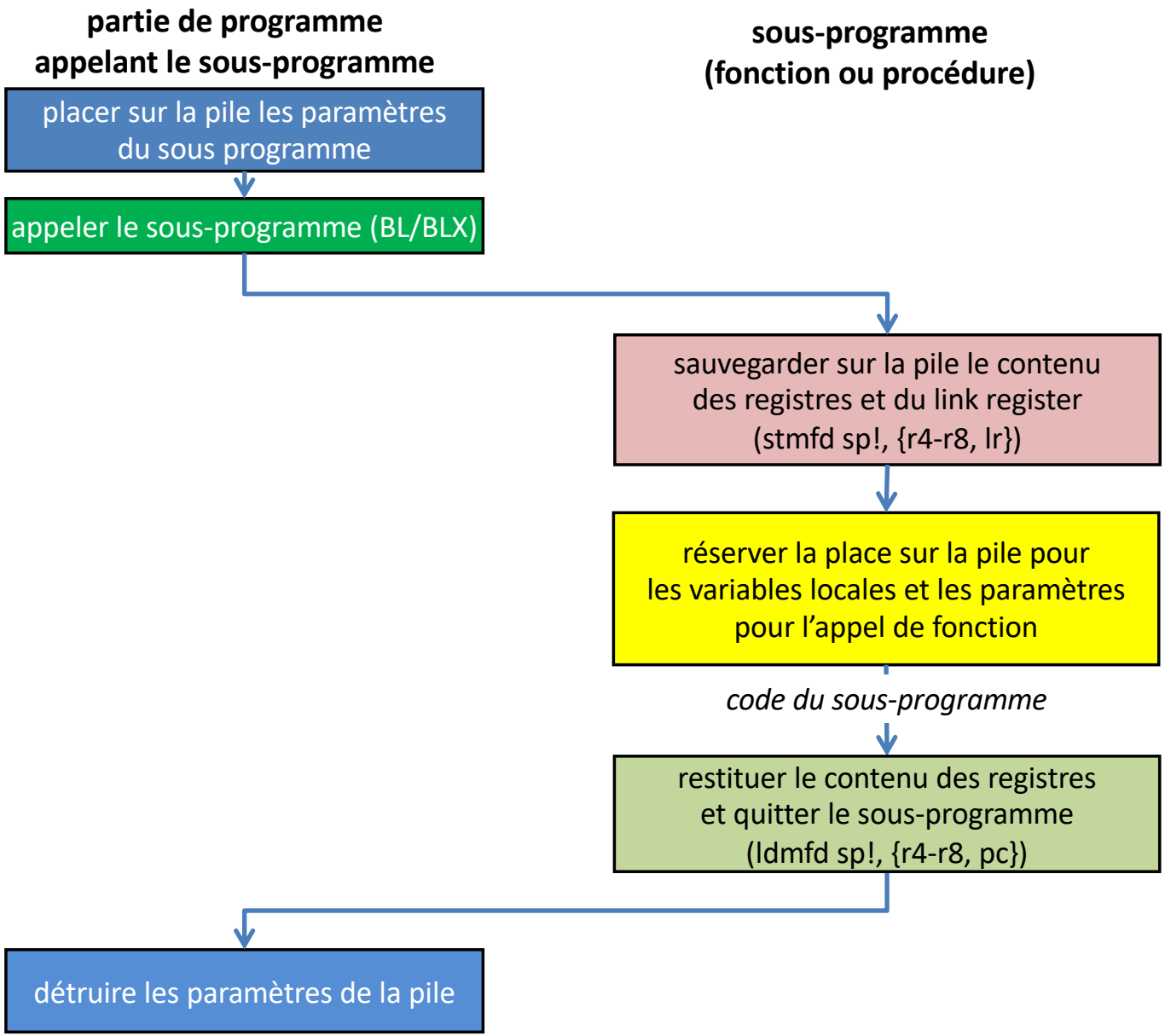
Altération des registres

Un sous-programme est généralement intégré dans un grand bloc d'instructions. Il cohabite avec le programme principal et les autres sous-programmes.

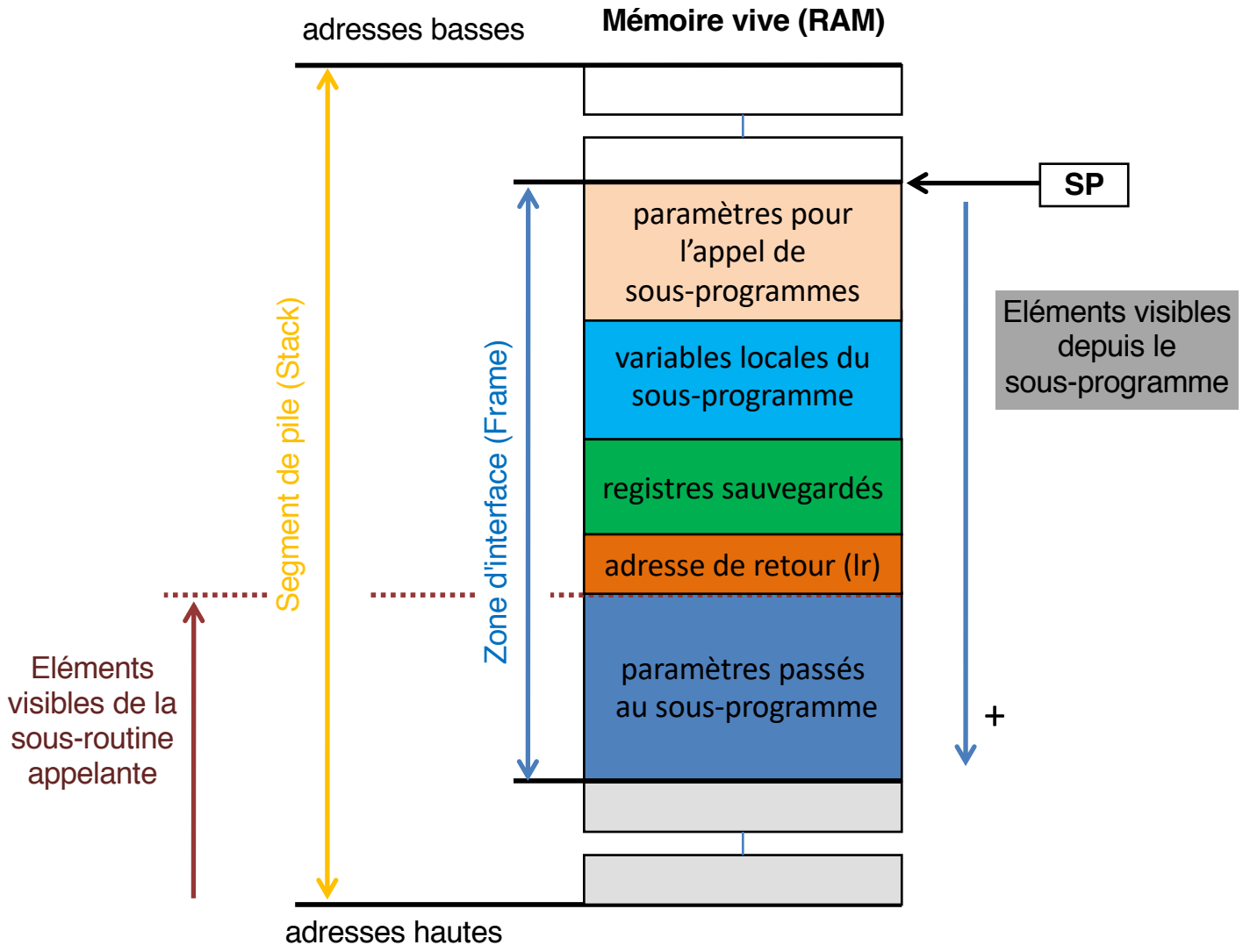
- **Le nombre de registres d'un processeur étant limité, chaque sous-programme ne peut pas se réserver quelques registres qu'il serait seul à utiliser.**
- **Par conséquent, un sous-programme peut modifier les registres utilisés par le programme appelant, s'il ne détruit pas la valeur originale de ces registres.**
- **Les registres utilisés par le sous-programme seront souvent sauvegardés avant modification, puis restitués avant le retour au programme principal.**
- **La pile sert naturellement de lieu de sauvegarde.**
- **Pour sauvegarder les registres, on peut utiliser les instructions `stmfd/ldmfd`, par exemple pour sauver et restituer le contenu des registres R4 à R8 et le LR**

```
stmfd   sp!,{r4-r8,lr}   // sauvegarde    → push {r4-r8,lr}
ldmfd   sp!,{r4-r8,pc}   // restauration  → pop  {r4-r8,pc}
```

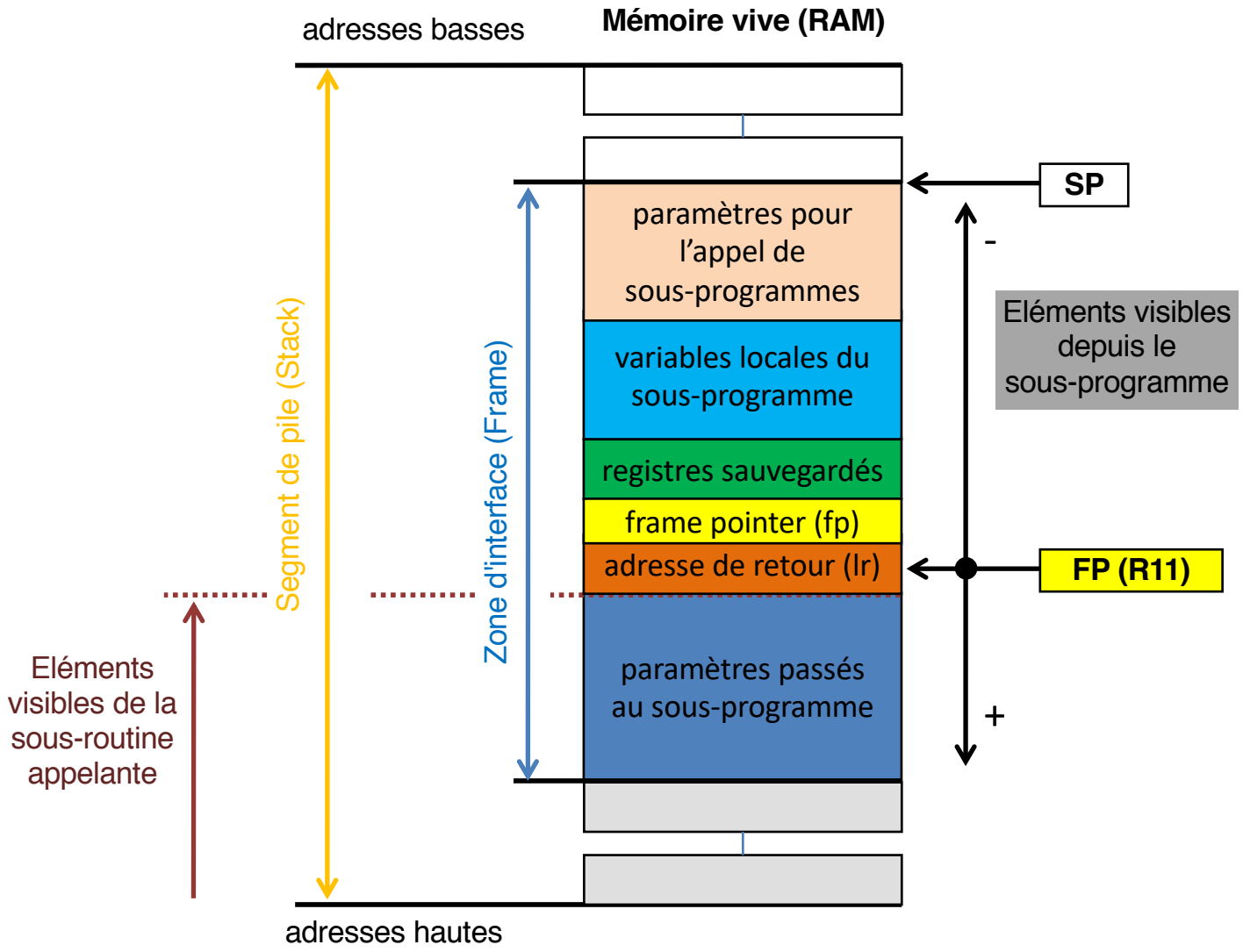
Séquence des opérations



Stack frame


adresses basses
Mémoire vive (RAM)
SP
paramètres pour l'appel de sous-programmes
variables locales du sous-programme
registres sauvegardés
adresse de retour (lr)
paramètres passés au sous-programme
Segment de pile (Stack)
Zone d'interface (Frame)
Eléments visibles depuis le sous-programme
Eléments visibles de la sous-routine appelante
+
adresses hautes

Frame pointer



Conventions C pour les processeurs ARM

- Les 4 premiers paramètres sont placés dans les registres R0 à R3
- Les paramètres suivants sont mis sur la pile du dernier au 5ème (de droite à gauche).
- Sur un processeur 32 bits, les caractères, les entiers et les types énumérés sont étendus à 32 bits avant d'être placés sur la pile.
- Le nombre de byte mis/réservés sur la pile doit être un multiple de 8
- Les registres ont la fonction suivante *(réf. 04_ARM_Architecture_Procedure_Call_Standard.pdf)*

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Table 2, Core registers and AAPCS usage

- **En informatique, la réentrance est la propriété pour une fonction d'être utilisable par plusieurs tâches simultanément . La réentrance permet d'éviter la duplication en mémoire vive d'un programme utilisé simultanément par plusieurs utilisateurs.**

- **Les conditions pour qu'un sous-programme soit réentrant sont:**
  - Ne doit pas utiliser de données statiques (globales) non-constantes
  - Ne doit retourner l'adresse de données statiques (globales) non-constantes
  - Ne doit traiter que des données fournies par le programme appelant
  - Ne doit pas reposer sur des verrous de ressources singleton
  - Ne doit pas modifier son propre code
  - Ne doit pas appeler des sous-programmes non-réentrants