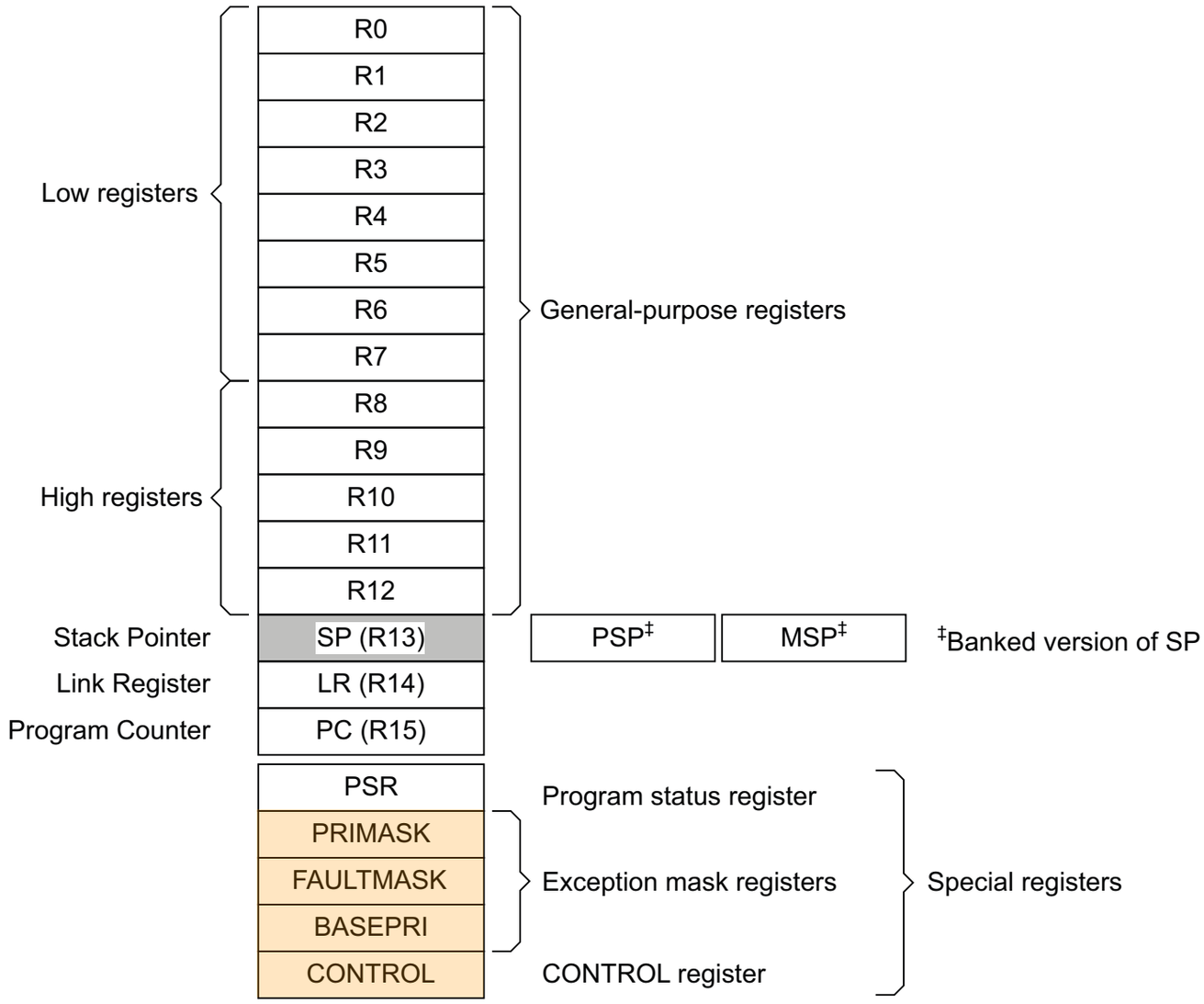
Comment porter Trampoline

Jean-Luc Béchennec — IRCCyN CNRS

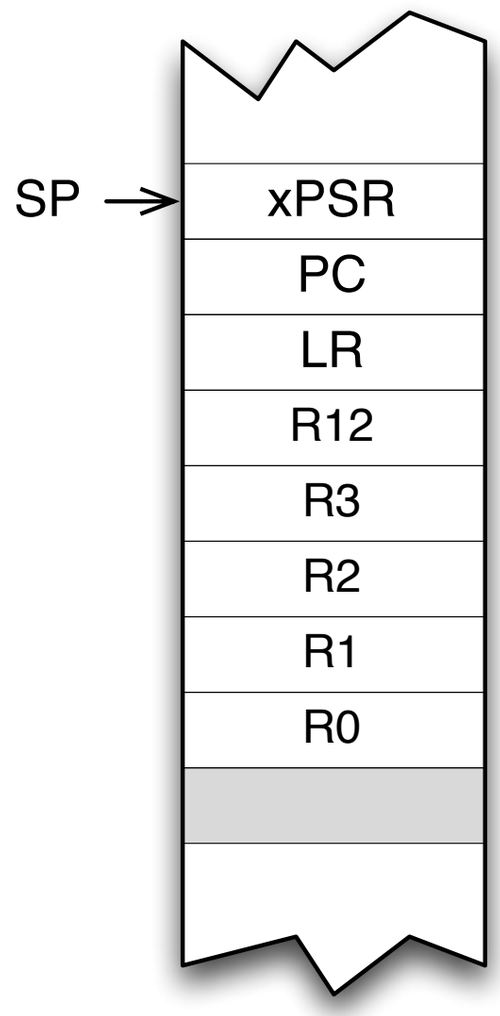
Contexte d'exécution

- Ensemble des registres utilisateur.
  - exemple sur ARM Cortex :



Contexte d'exécution (2)

- Pile après interruption



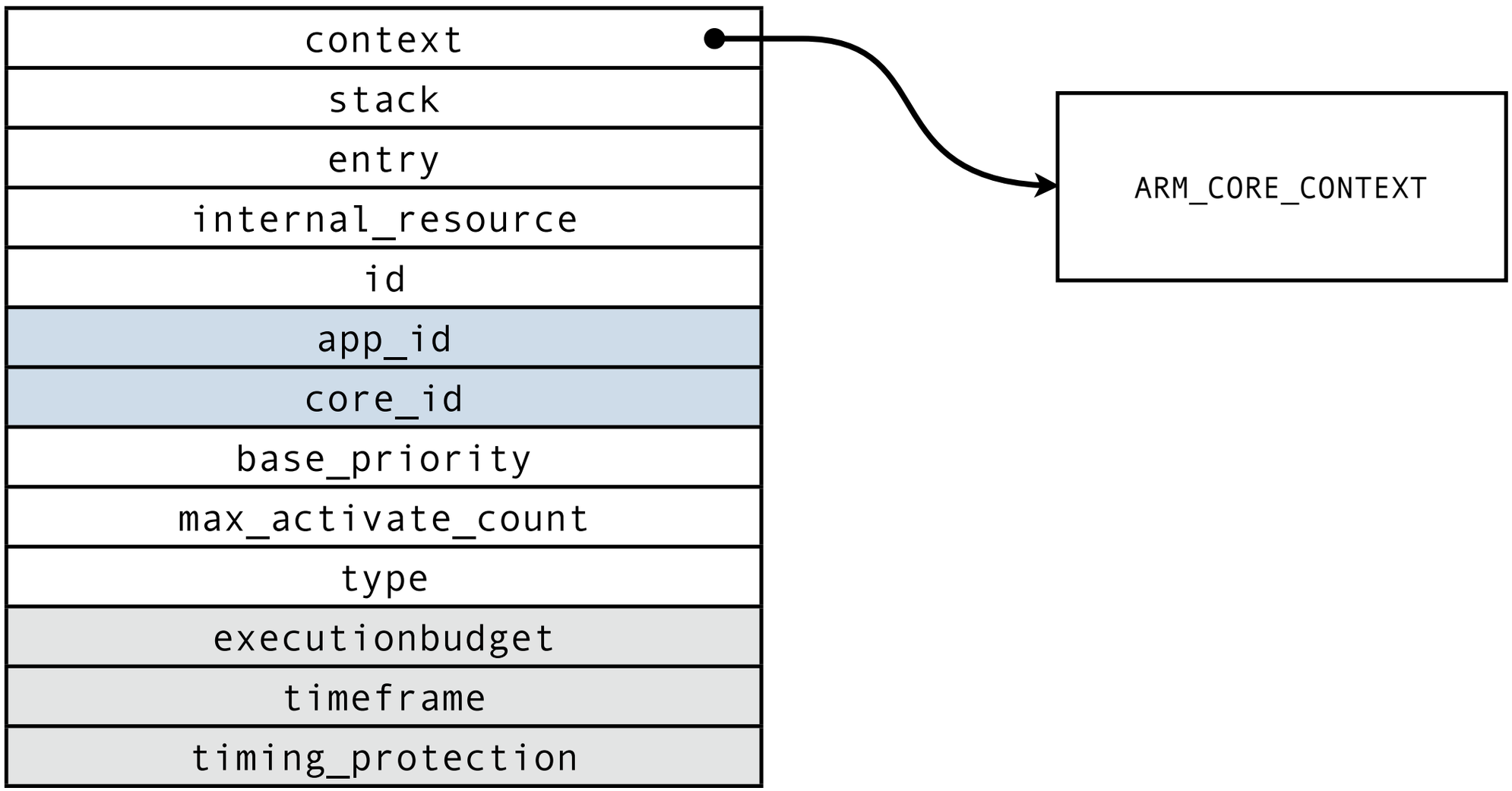
Zone de sauvegarde de contexte

- Registres utilisateurs moins ceux empilés lors de l'interruption.

```
struct ARM_CORE_CONTEXT {
   /* General Purpose Register r4-r11 */
   uint32 gpr[8];
   /* Stack Pointer - r13 */
   uint32 sp;
};
```

Zone de sauvegarde de contexte

- Descripteur de tâche statique



Instruction d'appel de service

- Exemple sur ARM : instruction `SVC`
  - La handler doit être logé dans le vecteur d'exception

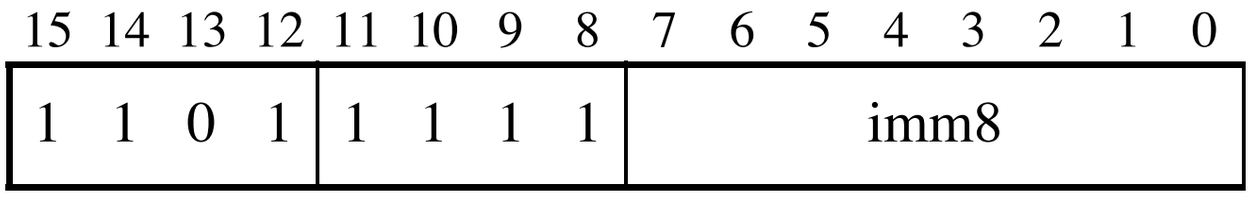
SVC (formerly SWI)

Generates a supervisor call. See *Exceptions* in the *ARM Architecture Reference Manual*.

Use it as a call to an operating system to provide a service.

Encoding T1 All versions of the Thumb ISA.

SVC<c> #<imm8>




```
imm32 = ZeroExtend(imm8, 32);
// imm32 is for assembly/disassembly, and is ignored by hardware. SVC handlers in some
// systems interpret imm8 in software, for example to determine the required service.
```

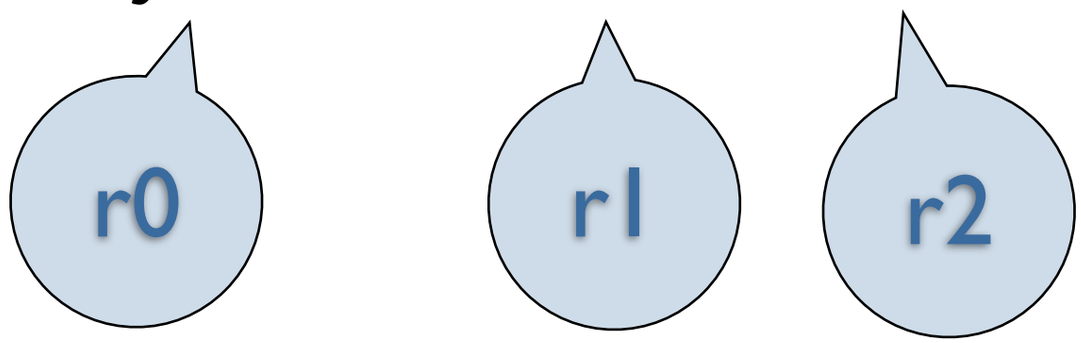
Connaître l'ABI

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.

Service côté utilisateur

- Exemple : SetRelAlarm

```
SetRelAlarm(myAlarm, 100, 200);
```




```
  .global SetRelAlarm
SetRelAlarm:
  li   r0,#OSSetRelAlarmServiceID
  sc
  blr
```

Le System Call Handler

1. Sauve des registres si nécessaire
2. Désactive la protection mémoire
3. Passe sur la pile système si nécessaire
4. Appelle le service
5. Effectue un changement de contexte si nécessaire et programme le MPU
6. appelle la fonction `tpl_run_elected` si nécessaire.
7. Passe sur la pile du processus si nécessaire
8. Active la protection mémoire
9. Restaure des registres si nécessaire
10. Retourne au processus

La structure `tpl_kern`

- Donne l'état de la tâche `running` et de la tâche `elected`

```
typedef struct
{
  P2CONST(tpl_proc_static, TYPEDEF, OS_CONST) s_running;
  P2CONST(tpl_proc_static, TYPEDEF, OS_CONST) s_elected;
  P2VAR(tpl_proc, TYPEDEF, OS_VAR)            running;
  P2VAR(tpl_proc, TYPEDEF, OS_VAR)            elected;
  VAR(uint32, TYPEDEF)                        running_id;
  VAR(uint32, TYPEDEF)                        elected_id;
  VAR(uint8, TYPEDEF)                         need_switch;
  VAR(tpl_bool, TYPEDEF)                      need_schedule;
#if WITH_MEMORY_PROTECTION == YES
  VAR(uint8, TYPEDEF)                         running_trusted;
#endif /* WITH_MEMORY_PROTECTION */
} tpl_kern_state;

#if NUMBER_OF_CORES == 1
extern VAR(tpl_kern_state, OS_VAR) tpl_kern;
#else
extern CONSTP2VAR(tpl_kern_state, OS_CONST, OS_VAR) tpl_kern[];
#endif
```

Implantation du portage (1)

- fichiers C et assembleur localisés dans `machines/<jeu d'instructions>/<micro>/<carte>`
- templates de production de code localisés dans `goilv2/templates/code/<jeu d'instructions>/<micro>/<carte>`
- OIL de configuration localisés dans `goilv2/templates/config/<jeu d'instructions>/<micro>/<carte>`
- templates de compilation localisés dans `goilv2/templates/compiler/<compilateur>/<jeu d'instructions>/ <micro>/<carte>`
- templates d'édition de liens localisés dans `goilv2/templates/linker/<linker>/<jeu d'instructions>/<micro>/<carte>`

Implantation du portage (2)

- Fichiers C et assembleur :
  - Handler d'appel système
  - Handler d'interruption
  - Initialisation de contexte `tpl_init_context`
  - Fonctions d'activation et de désactivation des IT `tpl_enable_interrupts` et `tpl_disable_interrupts`
  - Fonction de la tâche idle `idle_function`
  - Définitions de la pile système
  - Définition du contexte

Implantation du portage (3)

- Templates de production de code
  - Génération des services : `tpl_invoque_s.goilTemplate`, `service_call.goilTemplate`, `instruction_specific.goilTemplate` et `invoque_specific.goilTemplate`
  - Génération des instances de la structure de stockage de contexte et de la pile : `process_specific.goilTemplate`
  - Génération de la table des routines d'interruption : `interrupt_table.goilTemplate`

Implantation du portage (4)

- OIL de configuration.
  - Les fichiers config.oil le long du chemin sont agrégés.

```
CPU thumb2_ressource {
  COUNTER SystemCounter {
    SOURCE = SysTick;
  };
};
```

Implantation du portage (5)

- Templates de compilation
  - Génération du fichier `Compiler_Cfg.h` à partir de `Compiler_Cfg_h.goilTemplate`. Attributs pour la segmentation mémoire.
  - Génération du fichier `Compiler.h` à partir de `Compiler_h.goilTemplate`. Directives de placement des objets en mémoire.
  - Génération du fichier `MemMap.h` à partir de `MemMap_h.goilTemplate`. Définition des sections mémoire

Implantation du portage (6)

- Templates d'édition de liens
  - Production d'un link script à partir du template script.goilTemplate.

```
/*
 * code and consts of the processes of the applications
 */
. = ALIGN(4);
  apptext : {
    __PROGCONST_SECTION_START = .;
    __SEG_START_APP_CODE_CONST_RGN = .;
  . = ALIGN(4);
    *(.osApiConst)  /* API constants    */
    *(.rodata)      /* litteral strings (constants, strings, etc.) */
    *(.rodata*)     /* litteral strings (constants, strings, etc.) */
    . = ALIGN(4);
    *(.osApiCode)   /* API functions    */
    /* Sections for code of tasks and ISR */%
foreach proc in PROCESSES do
%
    *(.% !proc::KIND %_% !proc::NAME %Code)
%
end foreach
%
  } >FLASH
```