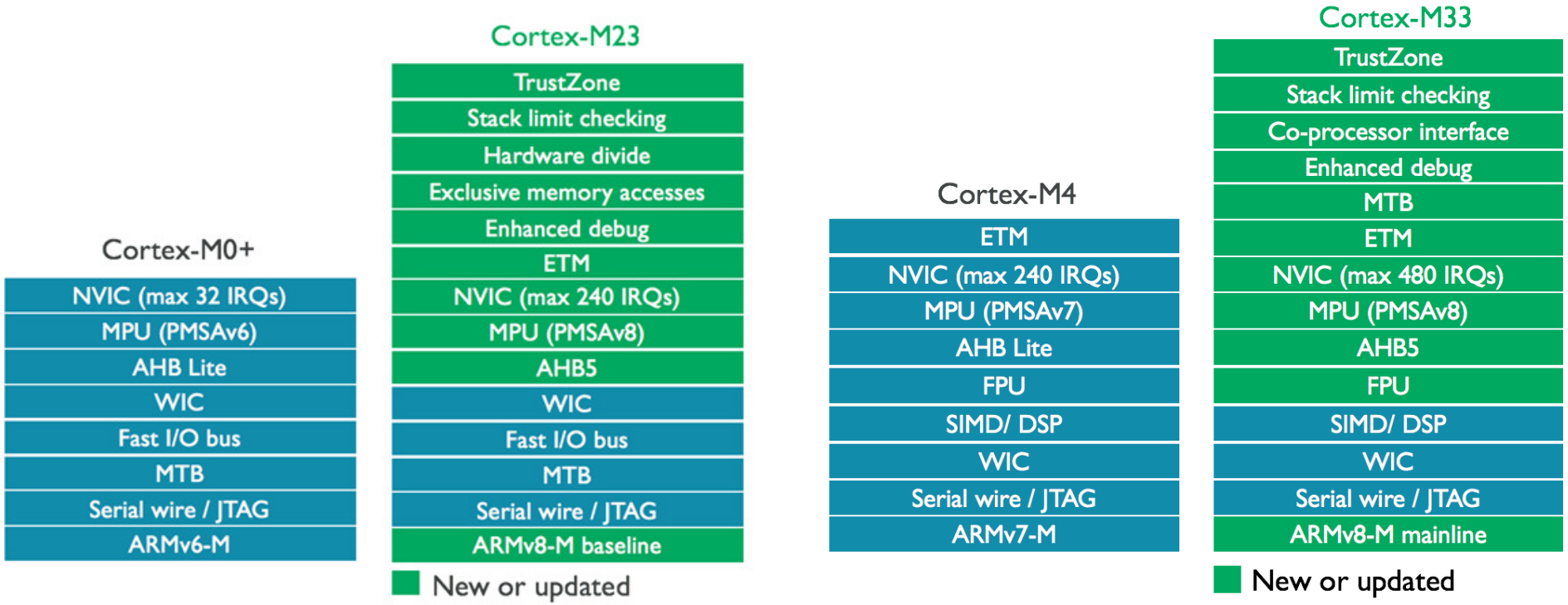
## 1. Modifications apportées

- Déclinaison en deux variantes : **Baseline** (Cortex-M23) et **Mainline** (Cortex-M33)

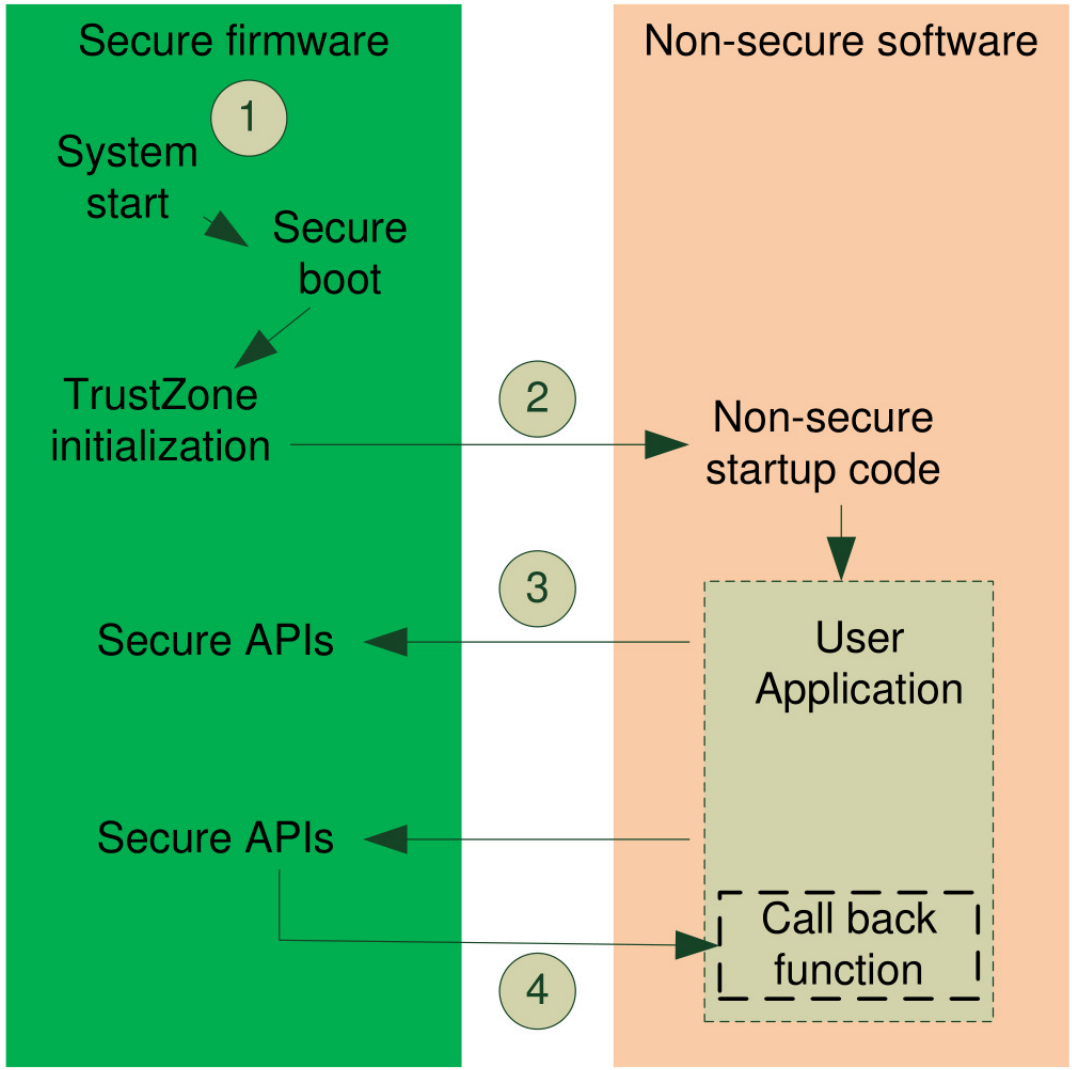



– Meilleure cohérence du jeu d'instruction entre les deux déclinaisons : 16bit immediate data loading instruction, hardware division, exclusive memory access (multiCPU), C11 and C++11 atomic type support, optional trustzone.

– MPU plus souple d'utilisation.

– Cortex-M33 vs Cortex-M23 : Harvard memory arch, FPU, SIMD/DSP instructions, stack limit register, coprocessor interface.

- Trust Zone : deux états du processeur : secure / non secure




(1) : démarrage du système en mode «secure» avec vérification que le firmware n'a pas été modifié.

(2) : démarrage du code «non secure» qui a son propre environnement d'exécution (startup, stack, heap) séparé de l'environnement «secure».

(3) : durant son exécution, le code «non secure» peut appeler du code «secure» par une API qui définit des points d'entrée contrôlés.

(4) : l'API «secure» peut permettre à du code «non secure» de s'exécuter en réaction à un évènement.

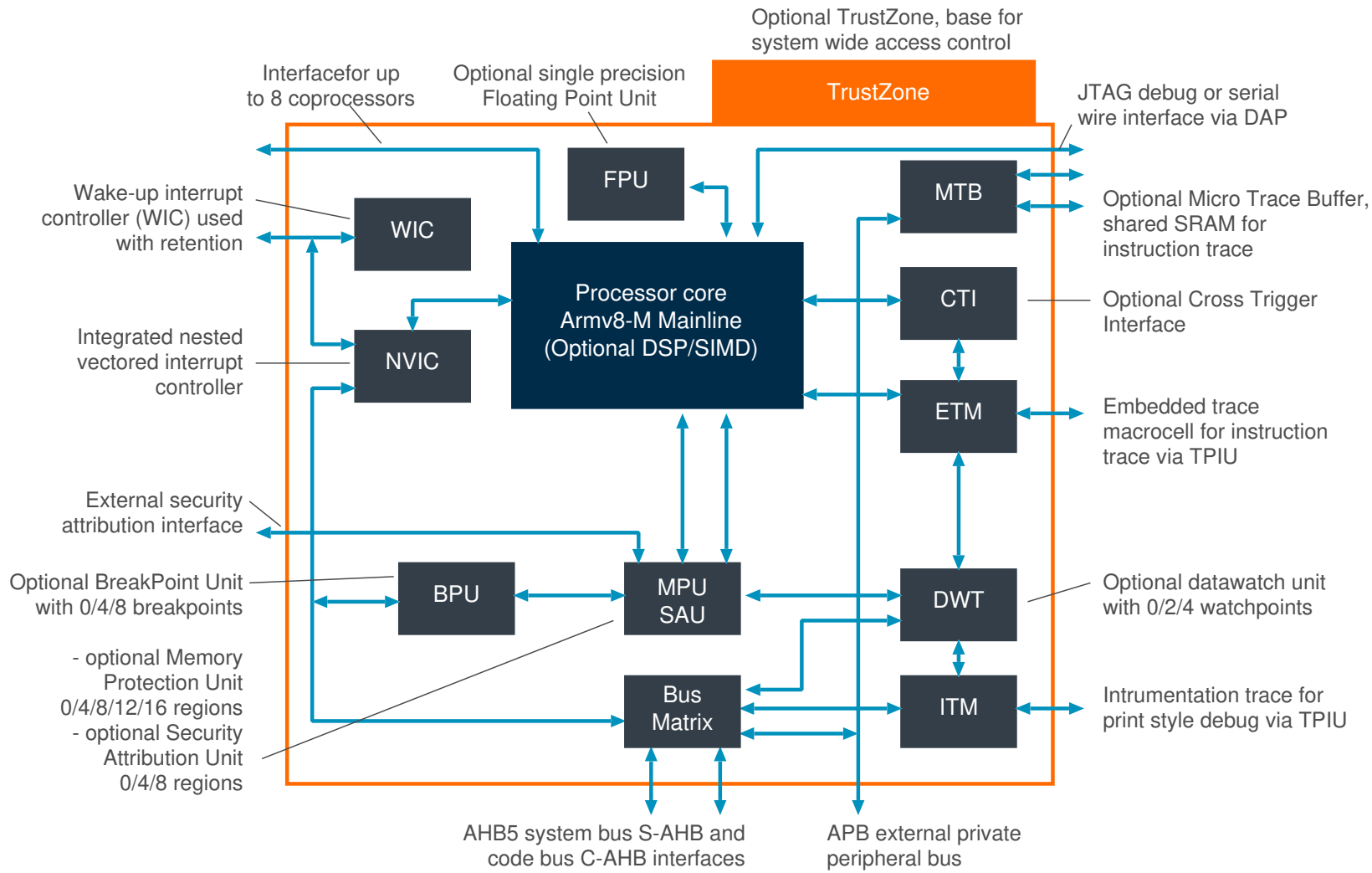
Lorsque le partitionnement secure/non sercure est activé, pas de possibilité de :

* faire un dump de la mémoire ou des registres
* faire du debug pas à pas, ni de trace

de l'exécution du mode «secure» depuis le mode «non secure». Pas de reverse engineering possible.

Exception «Secure Fault» sur le Cortex-M33, si tentative de violation de l'espace adressable (y compris par débordement de pile ou retour frauduleux d'exception).

Exemple du Cortex-M33

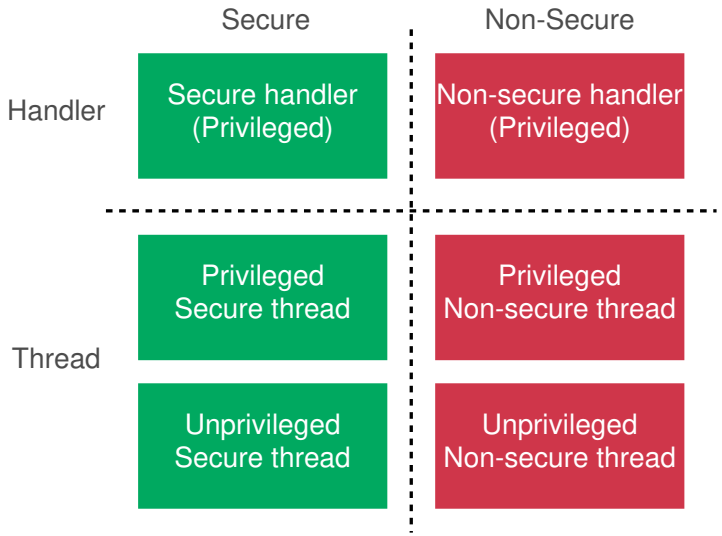


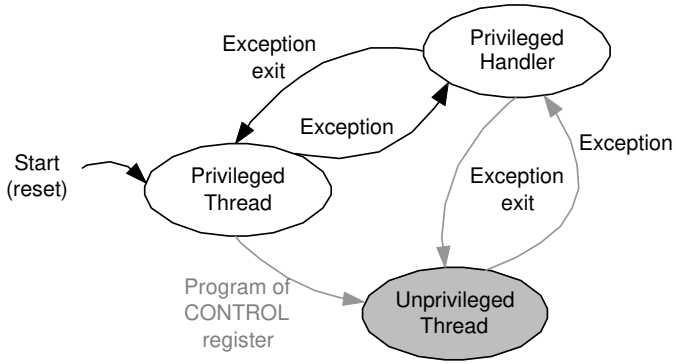

- Interface de coprocesseur : possibilité d'interfacer des coprocesseurs externes offrant une accélération matérielle pour des opérations spécifiques possédant leurs registres propres et des instructions qui viennent se rajouter au jeu d'instruction du processeur. Possibilité d'intégrer les coprocesseurs dans la protection offerte par la TrustZone.

# 2. Modèle d'exceptions du processeur Cortex-M33

## 2.1. Modèle de programmation du processeur Cortex-M33

- **modes de fonctionnement**



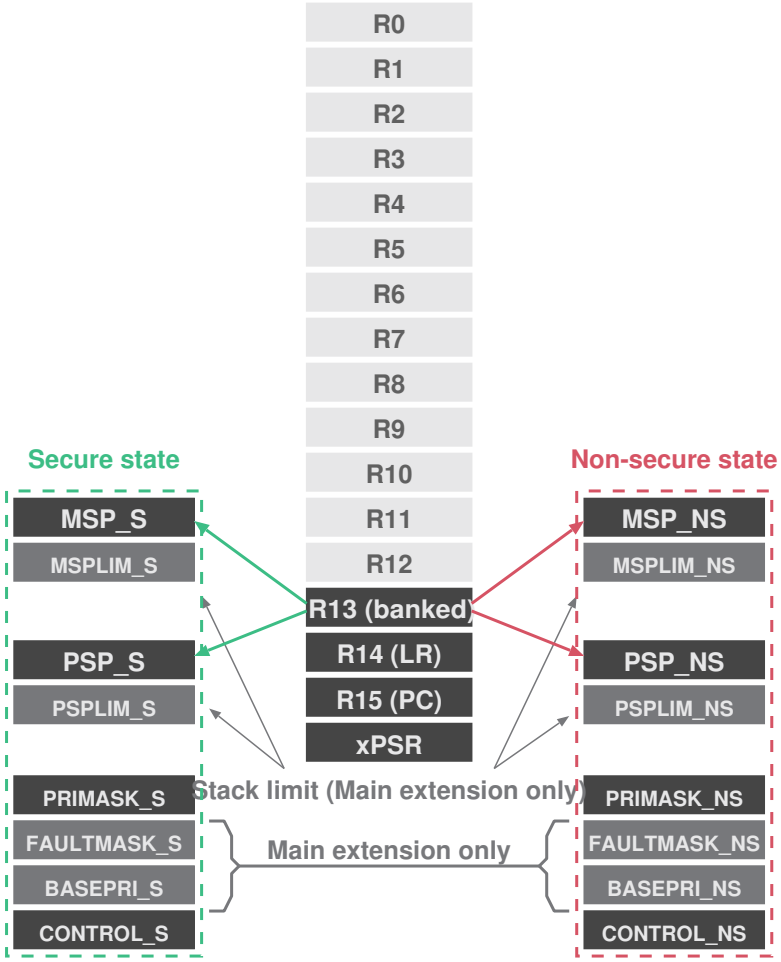

* Niveau de privilège pour l'exécution d'un programme
  - *privilégié* : il a accès à l'ensemble des instructions et des registres. Le pointeur de pile sp est typiquement msp.
  - *non privilégié* : il a un accès restreint à un certain nombre de ressources (registres système, configuration du NVIC) et aux instructions qui permettent de les modifier. Le pointeur de pile sp est msp ou psp (voir registre CONTROL).
* Modes de fonctionnement du processeur
  - *mode Thread* : mode d'exécution standard des programmes (fonctionnement privilégié ou non).

    *Modification du niveau de privilège*

    privilègié → non privilégié : registre CONTROL

    non privilégié → privilégié : traitement d'une exception/interruption
  - *mode Handler* : mode de traitement des exceptions/interruptions. Exécution privilégiée uniquement.
* Ressources dédoublées pour satisfaire aux exigences des modes Secure/Non-secure.

- **registres**




| Name | Type | Required privilege | Reset value | Description |
|---|---|---|---|---|
| `R0-R12` | RW | either | unknown | 32bit registers |
| `MSP` | RW | either | *0x00000000 | Main Stack Pointer |
| `PSP` | RW | either | unknown | Process Stack Pointer |
| `R14 / LR` | RW | either | 0xFFFFFFFF | Link Register |
| `R15 / PC` | RW | either | *0x00000004 | Program Counter |
| `MSPLIM` | RW | privileged | 0x00000000 | MSP Limit |
| `PSPLIM` | RW | privileged | 0x00000000 | PSP Limit |
| `PSR` | RW | privileged | 0x01000000 | Program Status Register |

Le registre `R13` est soit `MSP`, soit `PSP`, en fonction du pointeur de pile actif (voir le registre CONTROL). Lorsqu'on est en mode Handler, il s'agit de `MSP`.

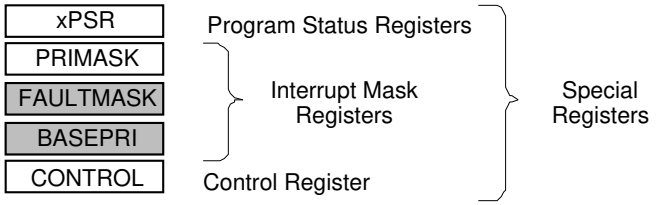
**Unité de calcul en virgule flottante** (FPU) : `S0-S31` : registres 32 bits (simple précision) ou `D0-D15` (double précision) si associés deux à deux. `FPSCR` : registre d'état.

**Règles d'utilisation des registres par les fonctions (AAPCS)** :

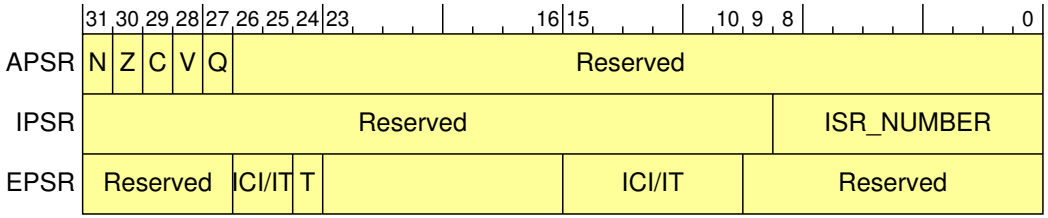
– Les registres `R4-R11` et `S16-S31` (si FPU) doivent être préservés lors des appels de fonctions.

– Les registres `R0-R3`, `R12` et `S0-15` servent pour passer des paramètres et renvoyer des résultats. Leurs valeurs n'ont pas besoin d'être préservées par les fonctions

- **registres spéciaux**




* Le **registre d'état** xPSR est la superposition de trois registres




| Name | Type | Required privilege | Reset value | Description |
|---|---|---|---|---|
| xPSR | RW | privileged | 0x01000000 | Program Status Register |
| ASPR | RW | either | unknown | Application Program Status Register |
| IPSR | RO | privileged | 0x00000000 | Interrupt Program Status Register |
| EPSR | RO | privileged | 0x01000000 | Execution Program Status Register |

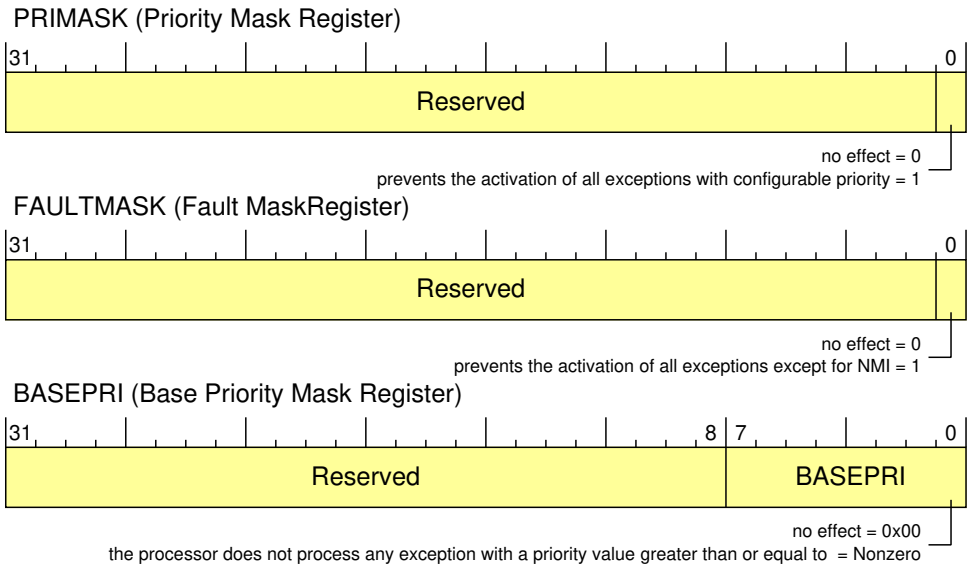
– Le registre APSR contient les indicateurs d'état NZCV de l'unité arithmétique et logique (nécessaires pour faire des tests). Le bit Q est lié à l'exécution d'instruction en arithmétique saturée.

– ISR_NUMBER intervient dans le traitement des exceptions (traitement des erreurs de fonctionnement) et interruptions (traitement de demandes matérielles externes au cœur du processeur).

– Le champ ICI/T (Interruptible Continuable Instruction / If Then instruction) permet de conserver l'état de l'exécution de certaines instructions qui peuvent être partiellement exécutées, puis interrompues pour exécuter des tâches jugées prioritaires, puis reprises et terminées plus tard.

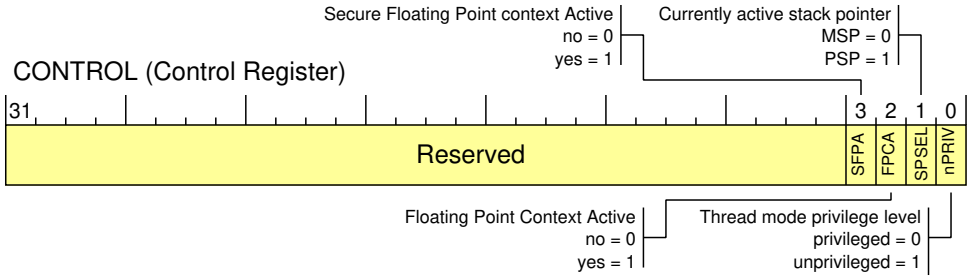
– Le bit T vaut toujours 1. Il est lié à l'usage par ces processeurs du jeu d'instruction dit "Thumb-2" qui est incompatible au niveau binaire avec le jeu d'instruction historique "ARM" exécuté par les Cortex-A (haut de gamme).

* **Registres de masquage des exceptions/interruptions** (Exception mask registers)




| Name | Type | Required privilege | Reset value |
|---|---|---|---|
| PRIMASK | RW | privileged | 0x00000000 |
| FAULTMASK | RW | privileged | 0x00000000 |
| BASEPRI | RW | privileged | 0x00000000 |

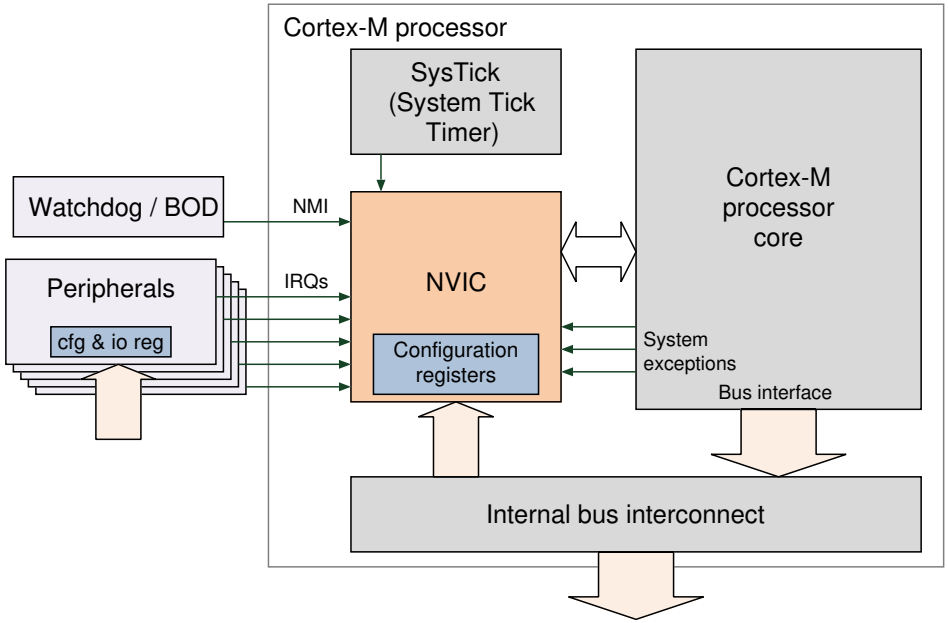
* **Registre de contrôle**




| Name | Type | Required privilege | Reset value |
|---|---|---|---|
| CONTROL | RW | privileged | 0x00000000 |

- **Au reset**, le processeur est dans l'état Secure, en mode Thread privilégié et le pointeur de pile sp est le MSP. Le MSP est initialisé à partir de la valeur à l'adresse 0x00000000. Le processeur exécute une fonction de Reset dont l'adresse est stockée à l'adresse 0x00000004.

## 2.2. Sources d'exceptions sur les microcontrôleurs à base de Cortex-Mx

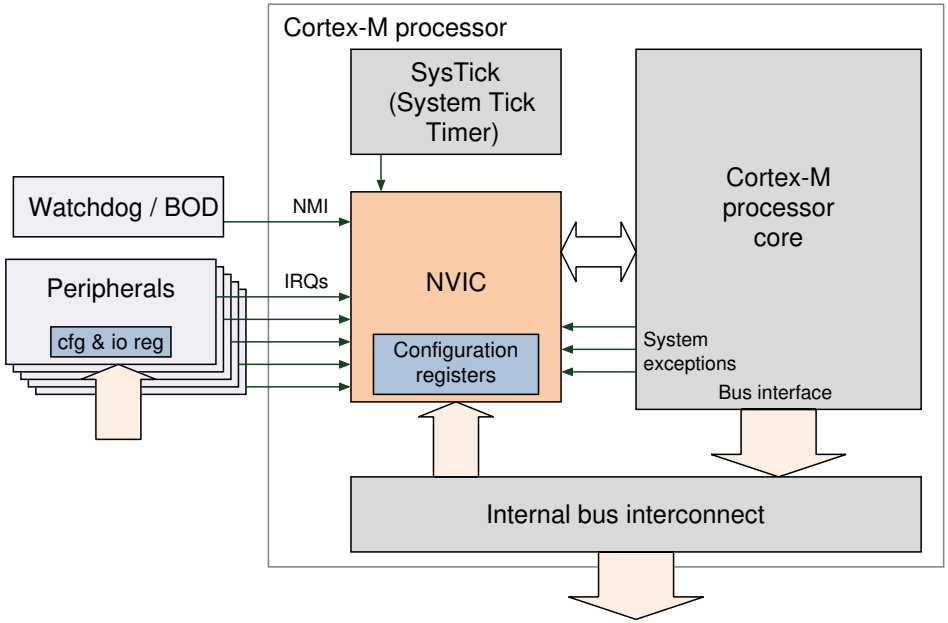



Tous les microcontrôleurs à base de processeur Cortex-M intègrent un **NVIC** (**Nested Vectored Interrupt Controller**). Il centralise les demandes d'exception/interruption (IRQ) et décide de l'ISR à exécuter en fonction de la source d'exception qui a fait la demande et de son niveau de priorité. Les sources d'exceptions reconnues sont :

- Exceptions systèmes
  - Erreurs d'exécution (System Faults) : Usage Fault, Bus Fault, Memory Manager Fault, Secure Fault[1]
  - Debug Monitor : associé à l'unité de debug intégré au processeur
  - SVC, PendSV : gestion des appels systèmes dans les systèmes d'exploitation

| Type d'erreur | Problème ayant pu causer l'erreur |
|---|---|
| Usage Fault | undefined instruction, invalid instruction (LDM/STM if unaligned address, divide by 0), invalid interrupt return address |
| Bus Fault | data + prefetch aborts, invalid size of transfert (byte on a word access only register) |
| Memory Manager Fault | MPU protection faults, write to a readonly region, execute from a nonexecutable memory region |
| Secure Fault | Violation de l'espace Secure : accès mémoire interdit, débordement de pile ou tentative de manipulation du code de retour d'une interruption |

[1]Si ces exceptions ont été désactivées ou si une erreur se produit pendant l'exécution d'un gestionnaire d'erreur (Fault Handler), l'erreur est promue en **Hard Fault**.




- Demandes d'interruptions matérielles (IRQ) réalisées par les coupleurs de périphérique du microcontrôleur.
- Timer SysTick : IRQ du Timer système intégré au microprocesseur Cortex-M
- NMI (Non maskable Interrupt) : IRQ associée généralement au Watchdog ou au détecteur de défaut d'alimentation (brown-out detector).

- Lockup : le processeur se verrouille (lockup) si une erreur de fonctionnement intervient pendant l'execution d'un gestionnaire de Hard Fault, ou de NMI. *Le processeur n'exécute plus aucune instruction*.

  Un déverrouillage est possible en exécutant le gestionnaire de NMI (activé par le Watchdog si lockup à partir du Hard Fault Handler), un reset matériel, ou par l'action d'un debugger.

Chaque source d'exception/interruption est associée à un gestionnaire (Exception/Fault/Interrupt Handler) dont l'adresse est stockée dans la **table des vecteurs**.

## 2.3. Table des vecteurs d'exception

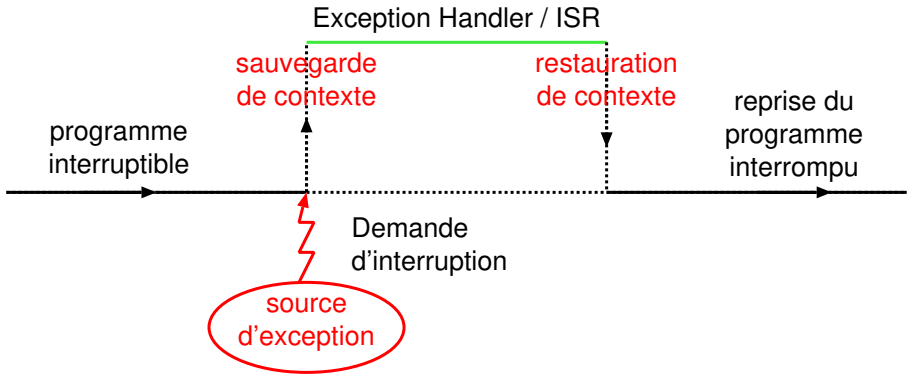
| Exception number | IRQ number | Offset | Vector table |
|---|---|---|---|
| 16+n | n | 0x0040+4n | Interrupt#n vector |
| | | | . . . |
| | | 0x004C | |
| 18 | 2 | 0x0048 | Interrupt#2 vector |
| 17 | 1 | 0x0044 | Interrupt#1 vector |
| 16 | 0 | 0x0040 | Interrupt#0 vector |
| 15 | -1 | 0x003C | Systick vector |
| 14 | -2 | 0x0038 | PendSV vector |
| 13 | | | Reserved |
| 12 | | 0x0030 | Debug Monitor vector |
| 11 | -5 | 0x002C | SVCall vector |
| 10 | | | Reserved |
| 9 | | | |
| 8 | | | |
| 7 | -9 | | **Secure fault vector** |
| 6 | -10 | 0x001A | Usage fault vector |
| 5 | -11 | 0x0018 | Bus fault vector |
| 4 | -12 | 0x0014 | Memory manage fault vector |
| 3 | -13 | 0x0010 | Hard fault vector |
| 2 | -14 | 0x000C | NMI vector |
| 1 | | 0x0008 | Reset vector |
| | | 0x0004 | Initial SP value |
| | | 0x0000 | |

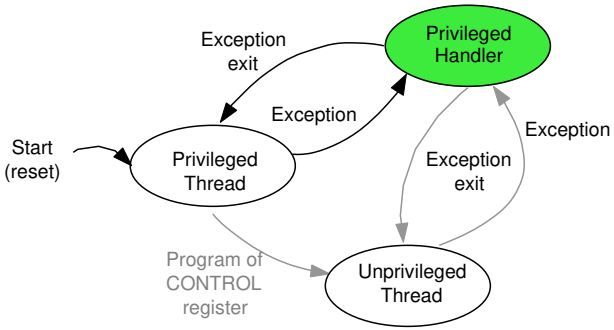
- La table des vecteurs est définie dans le fichier `startup/startup_lpc55s69_cm33_core0.c`
- Elle est composée, à l'adresse `0x00000000`[1] de la valeur d'initialisation du pointeur de pile `msp` (main stack pointer), et des vecteurs d'exception et d'interruption.
- Chaque vecteur d'exception/interruption stocke **l'adresse** du gestionnaire correspondant (System/Interrupt Handler).
- La table des vecteurs du Cortex-M33 permet de gérer jusqu'à 496 sources d'interruption hormis les 16 emplacements réservés de manière fixe pour les exceptions système.
- Chaque coupleur source d'IRQ est associé à un **IRQ number défini par le fabricant du microcontrôleur**. Voir, dans le projet, la liste dans les fichiers `device/LPC55S69_cm33_core0.h` et `startup/startup_lpc55s69_cm33_core0.c`
- Le **Reset vector** contient l'adresse de la fonction exécutée après un "reset" du système.
- Les états Secure et Non-secure disposent chacun d'une table des vecteurs.

[1]La position de la table des vecteurs peut être modifiée via le registre **Vectored Table Offset Register** `SCB->VTOR = offset` offset est tel que les 9 bits de poids faible doivent être nul (alignement minimum sur une frontière multiple de 128 mots de 32 bits). La table peut être en mémoire FLASH ou en RAM. Le registre est dupliqué : `SCB->VTOR_S` et `SCB->VTOR_NS`.

## 2.4. Séquencement des exceptions

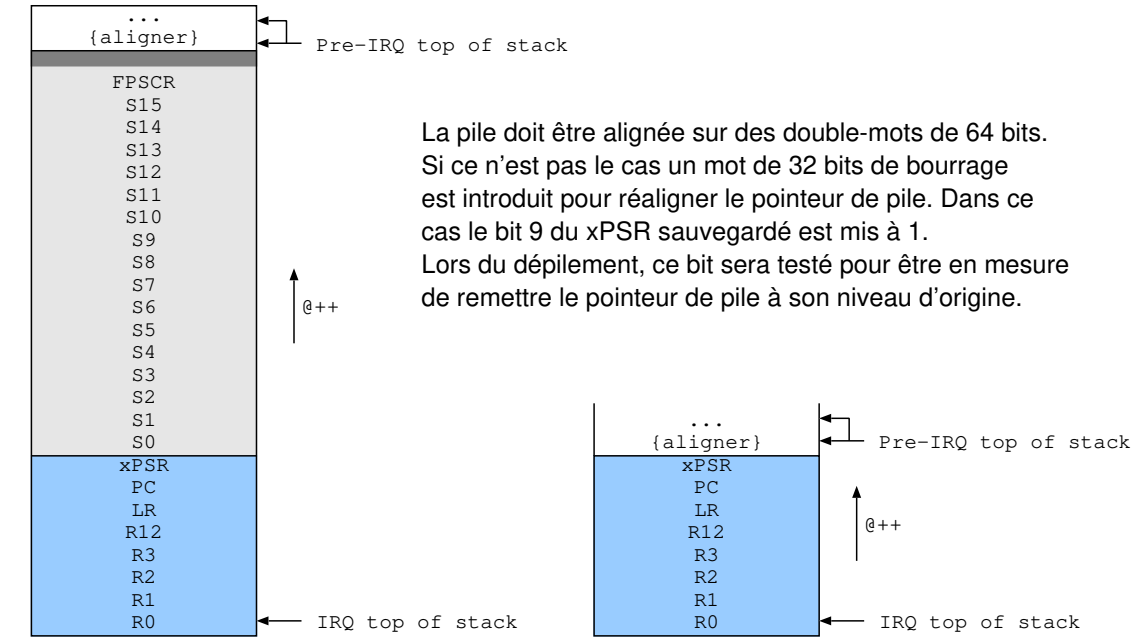
- Du point de vue de l'exécution du code, une **exception** ou **interruption**, est un événement qui arrête l'exécution d'un programme en cours pour exécuter un traitement considéré comme *plus prioritaire* à ce moment-là.




- Etat d'une demande d'exception :
  - **inactive** : aucune demande faite
  - **pending** : une exception/IRQ a été générée. Elle n'est pas encore traitée.
  - **active** : l'exception/IRQ est en cours de traitement.
  - **active & pending** : l'exception/IRQ est en cours de traitement. Une autre demande du même type a été générée.
- Contexte d'exécution de la routine de traitement de l'exception/IRQ :
  - mode : Handler
  - pointeur de pile : MSP

- Séquencement (compatible armv7-m et armv8-m)
  - ∗ Entrée dans la routine d'exception




Dans le cas où les registres FPU sont empilés (25 registres), un mot de 32 bits de bourrage est inséré en premier pour conserver l'alignement sur une adresse multiple de 64 bits

- **Sauvegarde automatique** des registres `R0-R3`, `R12`, `LR`, `PC`, `xPSR` (8 registres) et éventuellement des registres de la FPU[1] `S0-S15` et `FPSCR` **sur la pile courante** du programme interrompu.
- Le registre `LR` se voit ensuite affecté un **code qui indique comment devra se faire le retour**.
- **Saut à l'adresse du handler de l'exception/interruption obtenue via la table des vecteurs** et passage du processeur au **mode de fonctionnement Handler**.
- Modification des registres d'état et de contrôle :

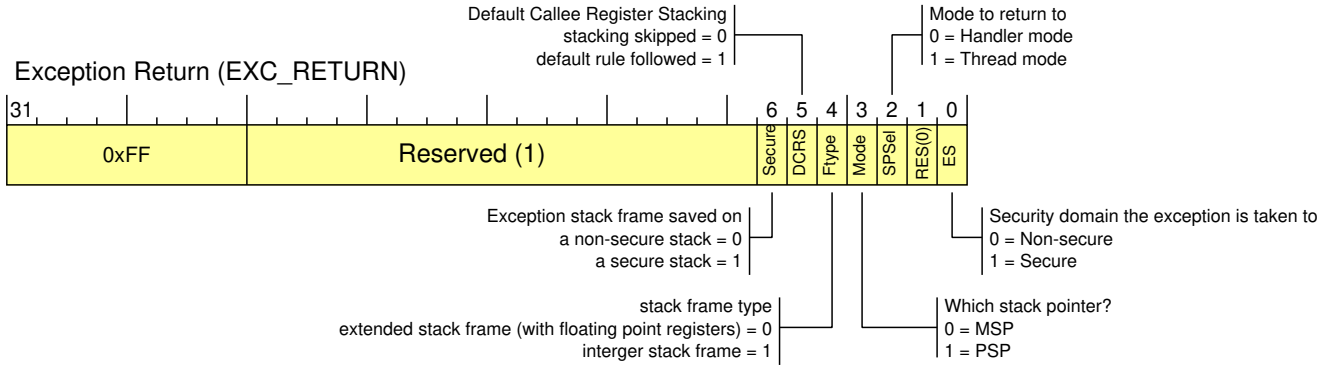
  `IPSR.ISR_NUMBER = 'Exception Number' ; EPSR.ICI/IT=0`

  `CONTROL.FPCA=0` (Lazy Stacking)

  `CONTROL.SPSEL=0` (utilisation du **pointeur de pile MSP**)

[1] Lazy stacking : l'espace est réservé sur la pile, mais les registres de la FPU ne sont empilés que si une instruction FPU est utilisée, auquel cas, on aura aussi CONTROL.FPCA=1.

* Retour de la routine d'exception : il s'effectue lorsqu'on copie le code placé dans le registre `LR` à l'entrée de la routine d'exception dans le `PC` (Lors du retour de la routine de traitement de l'exception/interruption). Le code de retour d'une exception est défini de la manière suivante :




Pour l'état **Secure**, on retrouve les valeurs standard de l'architecture armv7-m

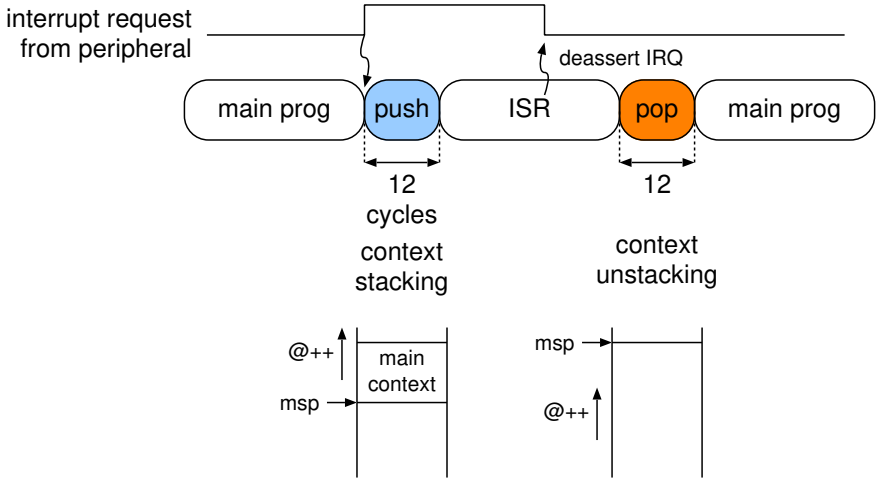
| FPU utilisée avant l'exception | FPU non utilisée avant l'exception | Actions à réaliser en sortie d'exception |
|---|---|---|
| `0xFFFFFFE1` | `0xFFFFFFF1` | retour au mode Handler, SP = MSP |
| `0xFFFFFFE9` | `0xFFFFFFF9` | retour au mode Thread, SP = MSP |
| `0xFFFFFFED` | `0xFFFFFFFD` | retour au mode Thread, SP = PSP |

Pour une implémentation sans TrustZone :

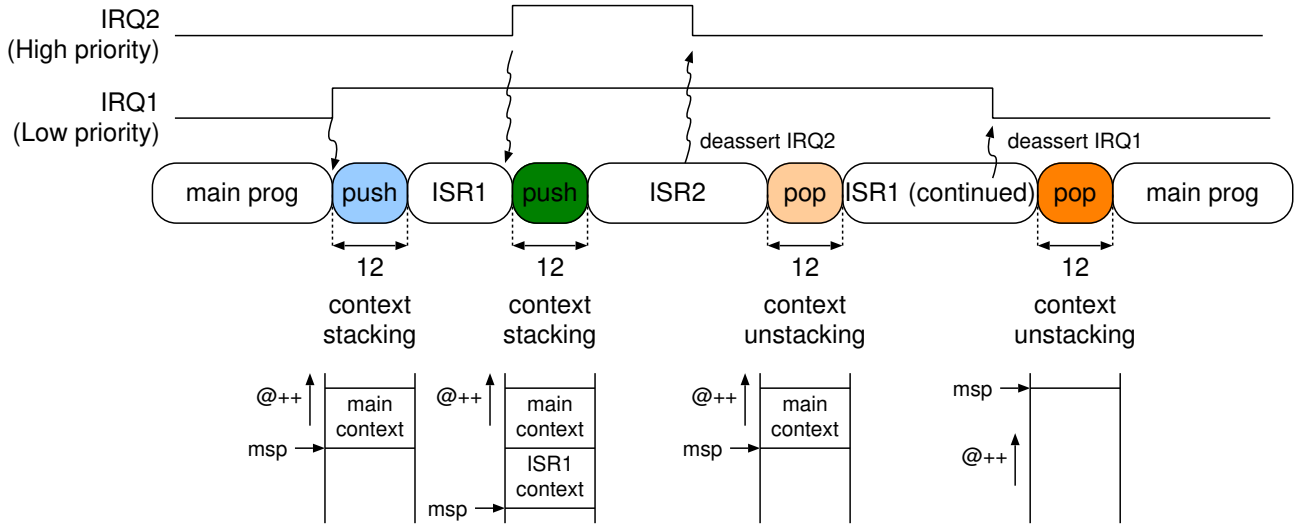
| | |
|---|---|
| `0xFFFFFFB0` | retour au mode Handler, SP = MSP |
| `0xFFFFFFB8` | retour au mode Thread, SP = MSP |
| `0xFFFFFFBC` | retour au mode Thread, SP = PSP |

Les actions nécessaires sont effectuées pour dépiler les valeurs et rétablir le registre `CONTROL` à sa valeur initiale.

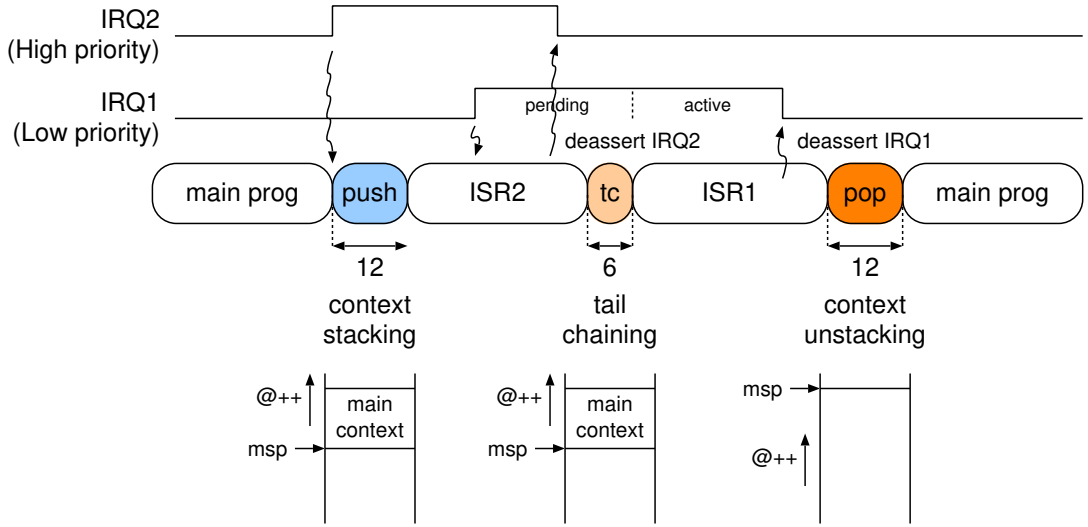
* Latence (avec une mémoire 0 wait-states) identique Cortex-M4 et Cortex-M33
  - Standard



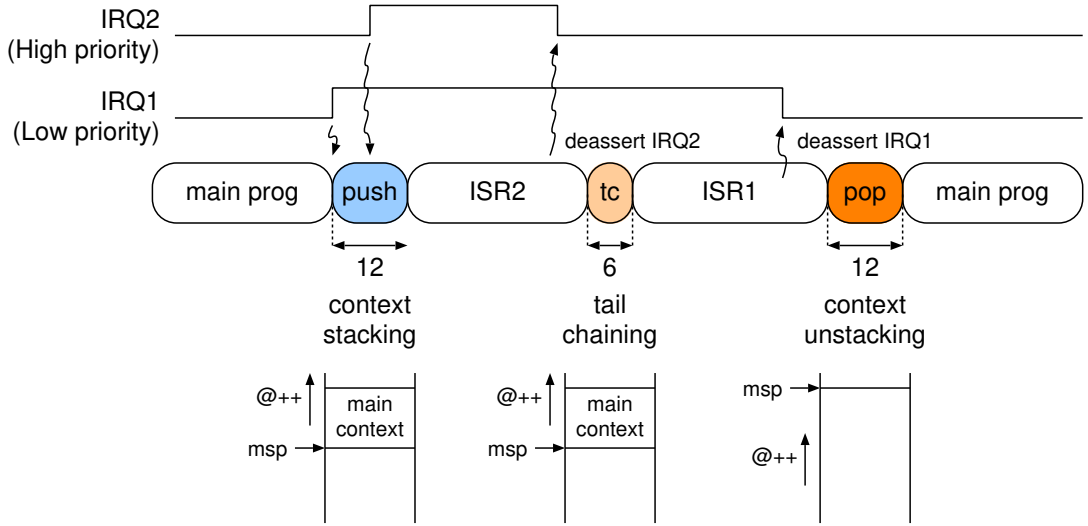

  - Préemption (ex : main stack = msp)



- Tail chaining (ex : main stack = msp)




- Late Arrival : si une demande d'interruption IRQ2 intervient pendant l'empilement du contexte destiné à servir une demande IRQ1 moins prioritaire, c'est l'ISR2 qui prend la main, profitant de l'empilement démarré pour servir l'IRQ1.



- Priorité/préemption/caractère synchrone ou asynchrone

| Exception number [a] | IRQ number [a] | Exception type | Priority | Vector address or offset [b] | Activation |
|---|---|---|---|---|---|
| 1 | - | Reset | -4, the highest | 0x00000004 | Asynchronous |
| 2 | -14 | NMI | -2 | 0x00000008 | Asynchronous |
| 3 | -13 | HardFault | -1 / -3 | 0x0000000C | - |
| 4 | -12 | MemManage | Configurable[c] | 0x00000010 | Synchronous |
| 5 | -11 | BusFault | Configurable[c] | 0x00000014 | Synchronous when precise, asynchronous when imprecise |
| 6 | -10 | UsageFault | Configurable[c] | 0x00000018 | Synchronous |
| 7 | -9 | SecureFault | Configurable[c] | 0x0000001A | Synchronous |
| 8-10 | - | Reserved | - | - | - |
| 11 | -5 | SVCall | Configurable[c] | 0x0000002C | Synchronous |
| 12-13 | - | Reserved | - | - | - |
| 14 | -2 | PendSV | Configurable[c] | 0x00000038 | Asynchronous |
| 15 | -1 | SysTick | Configurable[c] | 0x0000003C | Asynchronous |
| 16 | 0 | Interrupt (IRQ) | Configurable[d] | 0x00000040[e] | Asynchronous |

a. To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see *Interrupt Program Status Register*
b. See *Vector table* for more information.
c. See *System Handler Priority Registers*
d. See *Interrupt Priority Registers*
e. Increasing in steps of 4.

- Le niveau de priorité est configurable pour tous les types d'exception sauf pour les exceptions HardFault, NMI et Reset pour lesquelles il est fixe.

**Plus la valeur de priorité est faible, plus l'exception est prioritaire.**

- Les erreurs d'exécution sont reportées de manière **synchrone** à la tentative d'exécution de l'instruction précise. Il est possible de remonter à l'instruction et l'adresse qui ont causé l'erreur. On parle d'exception **precise**.

  L'exception peut devenir **imprécise** si le gestionnaire d'erreur n'est pas en mesure de s'exécuter immédiatement car préempté par un handler de plus grande priorité. On perd alors les informations relatives à l'instruction et l'adresse qui ont causé l'erreur. L'exécution du gestionnaire d'erreur devient alors **asynchrone** par rapport à l'exécution de l'instruction qui a causé l'erreur.

- L'activation d'une ISR (Interruption) est toujours **asynchrone** vis à vis du code qui s'exécutait précédemment. Un programme ne sait pas quand il va être interrompu par un évènement matériel.

# 3. Les exceptions SVC et Pend SV (armv7-m et armv8-m)

## 3.1. Les appels système : l'exception SVC

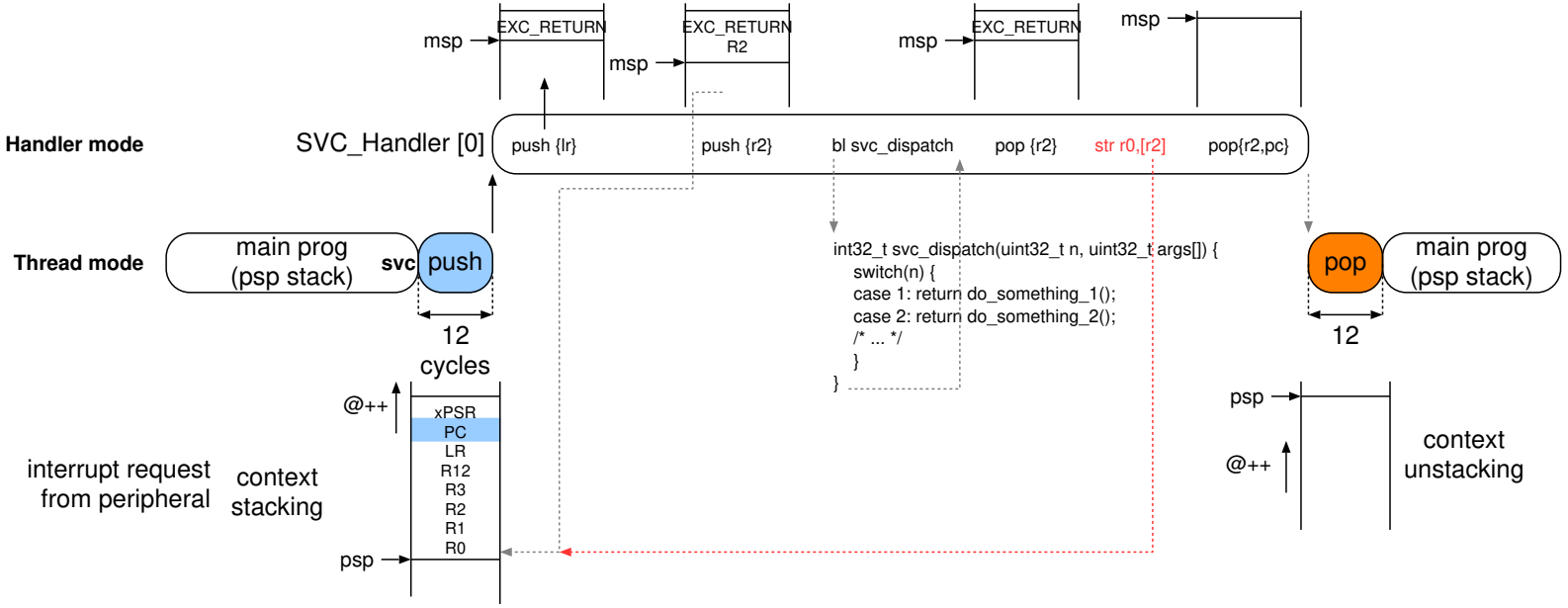
- L'exception SVC (SuperVisor Call) est déclenchée par l'exécution de l'instruction assembleur **svc**. L'instruction peut être accompagnée d'un numéro compris entre 0 et 255.
- Encapsulée dans une fonction C et associée à une méthode de protection des accès mémoire (MPU), l'instruction **svc** permet de :
  - mettre en place des points d'entrée pour accéder à des fonctionnalités relevant du «code système»,
  - mettre en place un modèle de fonctionnement sécurisé permettant de mettre en place une frontière entre du code «User» et du code «Système» (méthode historique avant TrustZone disponible sur un grand nombre de processeurs).
- Côté «User», l'instruction **svc** permet de définir un **appel système**.

  *Exemple :* l'appel système 12 de prototype

```
int read(int fd, void *buf, size_t len)
{
   int size_or_err;
   __asm volatile ("svc 12\n\tmov %0, r0" : "=r" (size_or_err));
   return size_or_err;
}
```

  - Les paramètres sont passés par les registres (4 max) : ici `r0` (le premier), `r1` et `r2`.
  - La vakeur de retour est renvoyée dans `r0`. L'instruction `mov` permet de lier la valeur de retour au paramètre de retour (ici `size_or_err`) défini dans la fonction.

- Mécanique d'utilisation des SVC




```
SVC_Handler:    push        {lr}            // save EXC_RETURN code
                tst         lr, #4          // if lr[2]==0
                mrseq       r2, msp         //   svc was called from handler mode, so the context is on msp stack
                addeq       r2, #4          // update msp because of the push
                mrsne       r2, psp         //   svc was called from thread mode,
                ldr         r0, [r2, #24]   // get stacked PC from the stack frame
                ldrb        r0, [r0, #-2]   // get SVC_NUMBER by reading the LSB of the svc instruction code
                mov         r1, r2          // make r1 point to the parameter list
                push        {r2}            // remember the stack used (r2)
                bl          svc_dispatch    // process svc according to SVC_NUMBER
                pop         {r2}            // write the result (r0) to the stack
                str         r0, [r2]        //   used when svc was called
                pop         {pc}            // back the svc caller
```

### 3.2. Changement de contexte dans un OS multitâche : l'exception PendSV

- Dans un OS mutitâche, chaque tâche dispose d'un contexte d'exécution propre (valeurs dans les registres, pile).
- Les tâches s'exécutent en mode Thread unprivileged avec psp comme pointeur de pile. Les IRQ sont autorisées.
- La commutation des tâches est de la responsabilité de l'OS, et peut intervenir suite à
  - une interruption du timer SysTick (round-robin),
  - un appel système exécuté par la tâche et qui conduit à son blocage (abandon volontaire du CPU ou demande de jeton sur un sémaphore).

  La commutation de tâche induit une commutation de contexte :
  - sauvegarde du contexte de la tâche qui est arrêtée (pour pouvoir reprendre plus tard à partir de ce point-là),
  - restauration du contexte de la tâche qui est démarrée

  Les contextes sont sauvegardés et restaurés à partir des piles utilisées par les tâches.

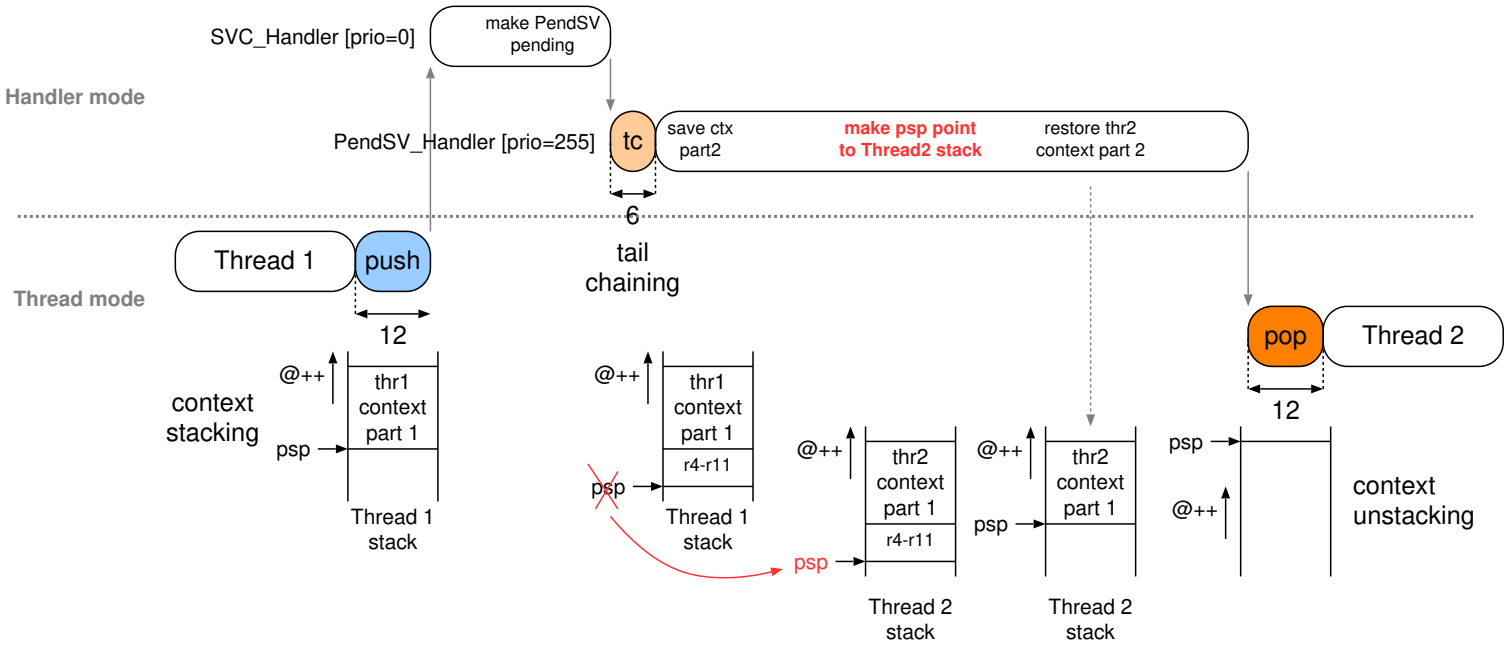
- La commutation de contexte proprement dit est confiée à l'exception **PendSV** qui doit être mise à l'état «pending» de manière explicite.

---

*SCB→ICSR* |= 1<<28; // set PendSV to pending

---

L'exception PendSV doit avoir la priorité minimale (255), de manière à ce que les interruptions matérielles soient traitées de manière plus prioritaires que l'exception PendSV (qui sera donc servie en dernier). Le but est d'obtenir la meilleure réactivité possible.




Les IRQ doivent utiliser des priorités NVIC : 0 < prio < 255.