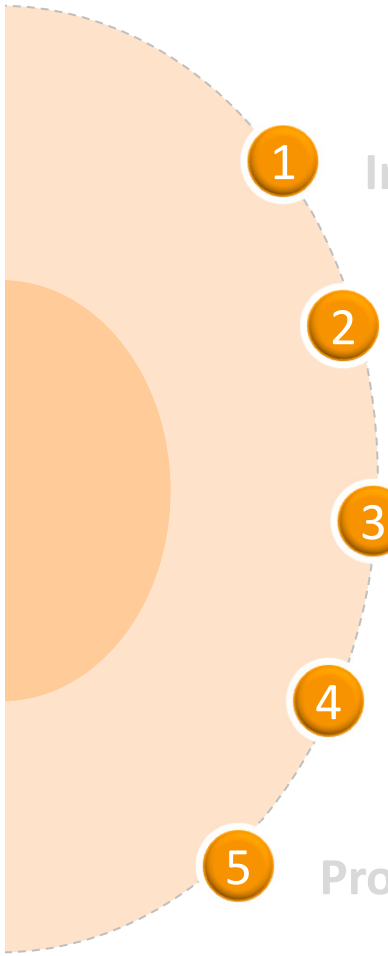
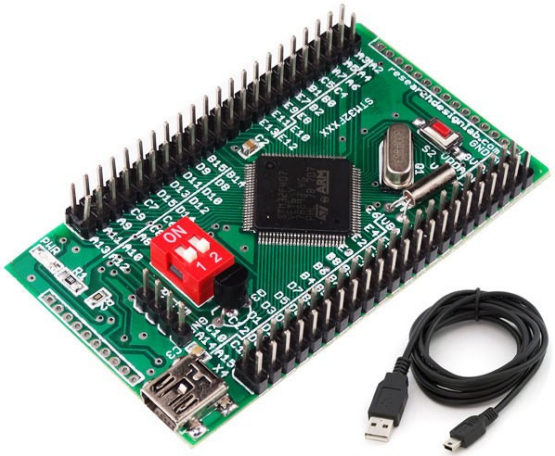
Plan du module



1. Introduction aux systèmes à microprocesseurs
2. Les mémoires
3. Les microprocesseurs
4. **Architecture d'un processeur RISC (Cortex M4)**
5. Programmation Assembleur Cortex M4

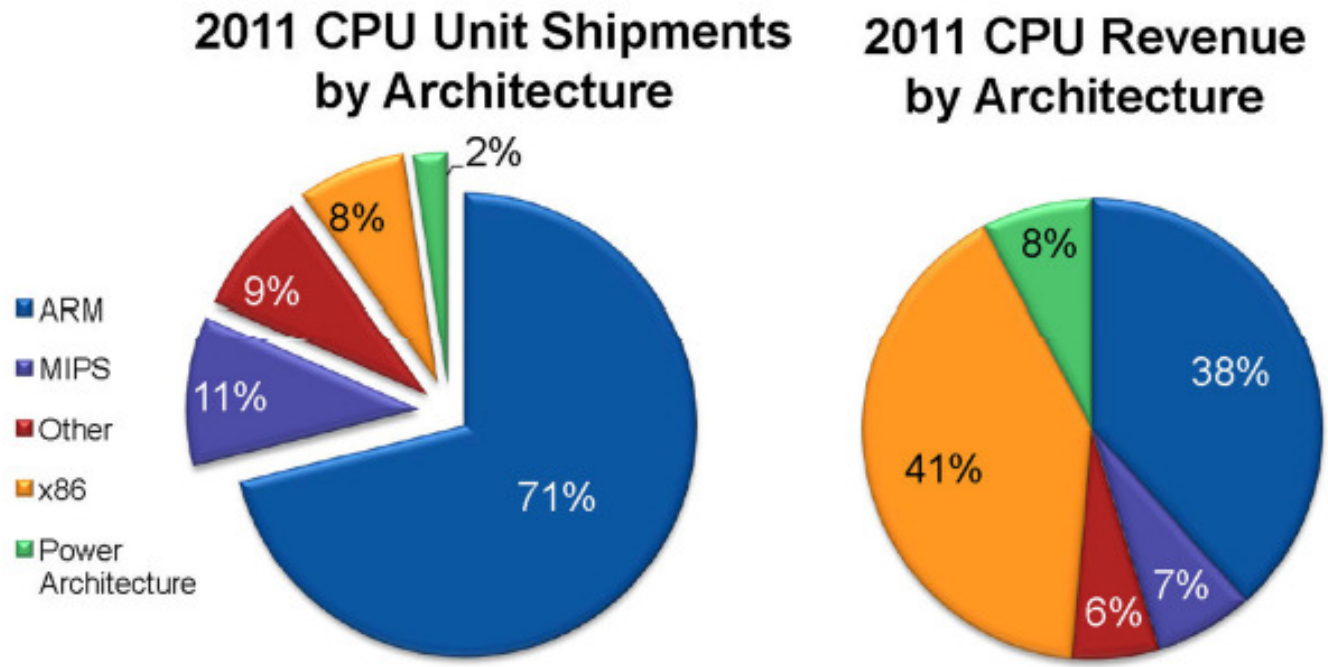
Pourquoi le Cortex M4?

- Un exemple typique de processeur embarqué
- Appartient à la famille des processeurs ARM leader sur le marché des processeurs embarqués (~10 milliards de processeurs vendus / an)
- Performances et consommation d'énergie intéressantes
- Support industriel large (outils de développement / documentation)
- Programmable en langages de haut niveau (C/C++) et bas niveau (assembleur) pour un contrôle total des ressources matérielles et une exploitation optimale des performances du matériel
- Le cœur du microcontrôleur STM32F4 qui équipe la carte « STM32F4 discovery » utilisée dans des TP didactiques



Société ARM

- **ARM** initialement Acorn Risc Machine, puis Advanced Risc Machine est une famille d'architectures RISC pour les processeurs, configurées dans différents environnements.
- La Société anglaise ARM Holdings est basée sur un modèle économique particulier de la microélectronique : **la conception de propriétés intellectuelles**. Ainsi il n'est pas possible d'acheter un processeur ARM comme c'est le cas pour Intel.
- Les cœurs ARM sont intégrés au sein de systèmes sur puces (SoC) complets.
- Les cœurs de processeurs ARM sont très présents dans les systèmes embarqués d'autres sociétés (ST, Sumsung...)

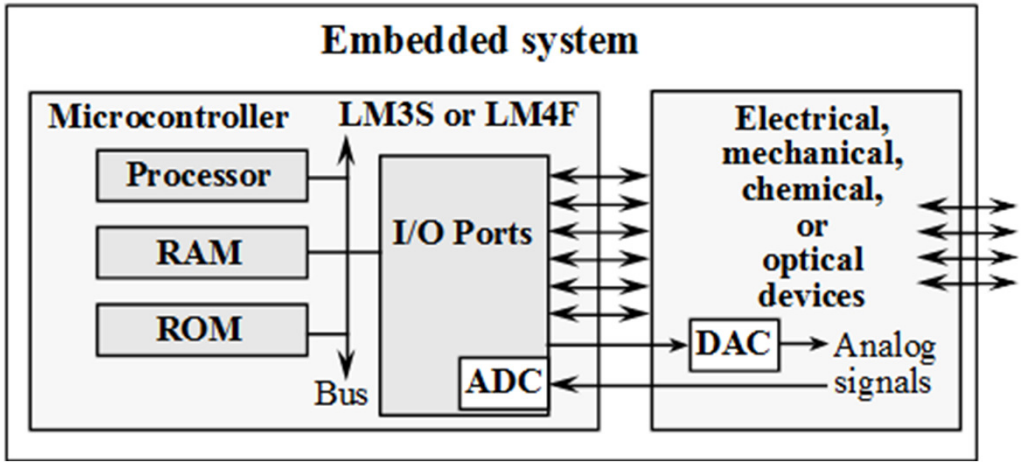


➢ **Architectures conçues par ARM**: ARMv1 ... ARM11, SecureCore, Cortex-M, Cortex-R, Cortex-A (32 bit/64 bit)

➢ **Architectures conçues par d'autres sociétés sous licence ARM** (jeu d'instructions ARM):
Strong ARM (Digital), Faraday (Faraday Technology), Xscale (Intel/ Marvell), Sheeva (Marvell), SnapDragon (QualComm), Ax (Apple), X-Gene (Applied Micro), Denver (Nvidia), ThunderX (Cavium), K12 (AMD), Exynos (Samsung)

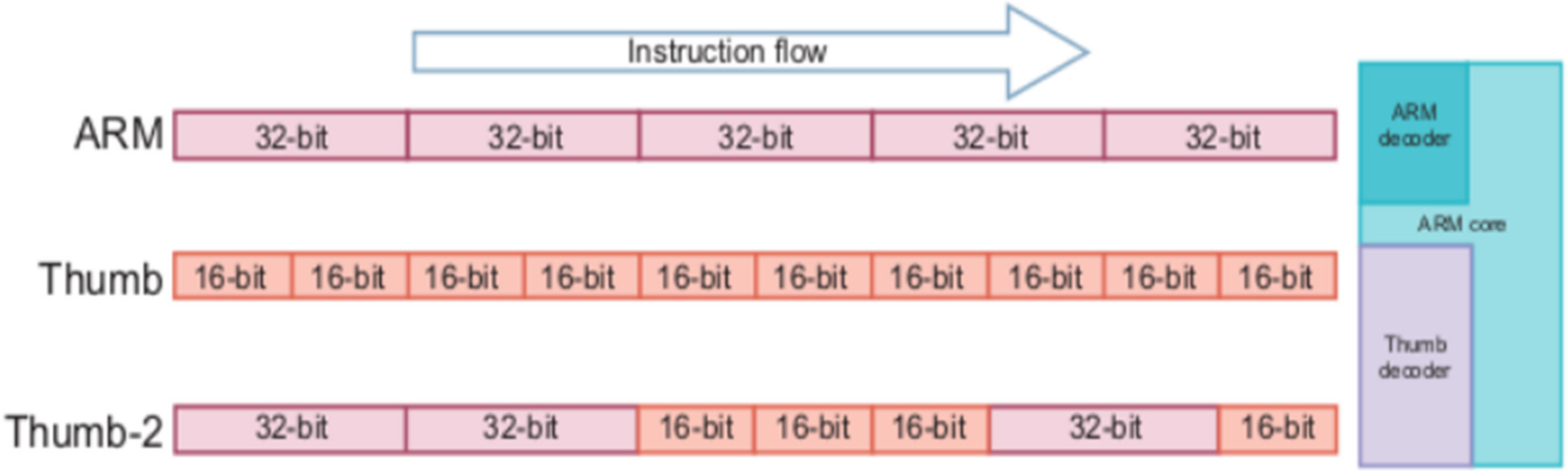
Famille ARM Cortex pour les microcontrôleurs

Year	Microcontroller
2004	Cortex-M3
2005	
2007	Cortex-M1
2009	Cortex-M0
2010	Cortex-M4(F)
2011	
2012	Cortex-M0+
2013	
2014	Cortex-M7(F)
2015	
2016	Cortex-M23 Cortex-M33 (F)
2017	



Le profil Cortex M

- Dédié pour les puces à microcontrôleurs, ou une partie dans des SoC pour contrôle de puissance ou contrôle des I/O ou contrôle des écrans tactiles ou contrôle des batteries intelligentes ou contrôle des capteurs...
- Optimisé pour les applications faible consommation d'énergie
- On peut avoir avec ce profil trois types d'instructions selon le mode (ARM, Thumb, Thumb-2)



Cœurs Cortex M3 et Cortex M4

- Les cœurs Cortex M3 et Cortex M4 sont semblables seulement que Cortex M4 ajoute des instructions DSP (Digital Signal Processing)

- Optionnellement on peut trouver dans Cortex M4 une unité du traitement des flottants FPU (Flotting-Point Unit) afin d'avoir Cortex-M4F

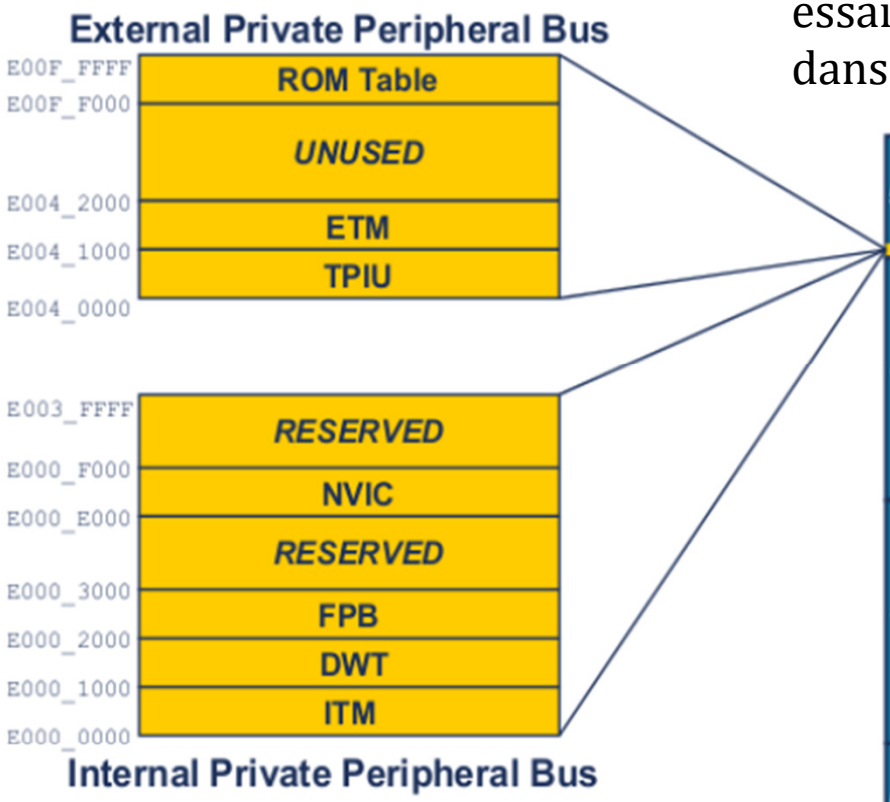
- Les deux processeurs sont utilisés dans des applications embarquées qui nécessitent une réponse rapide aux interruptions.

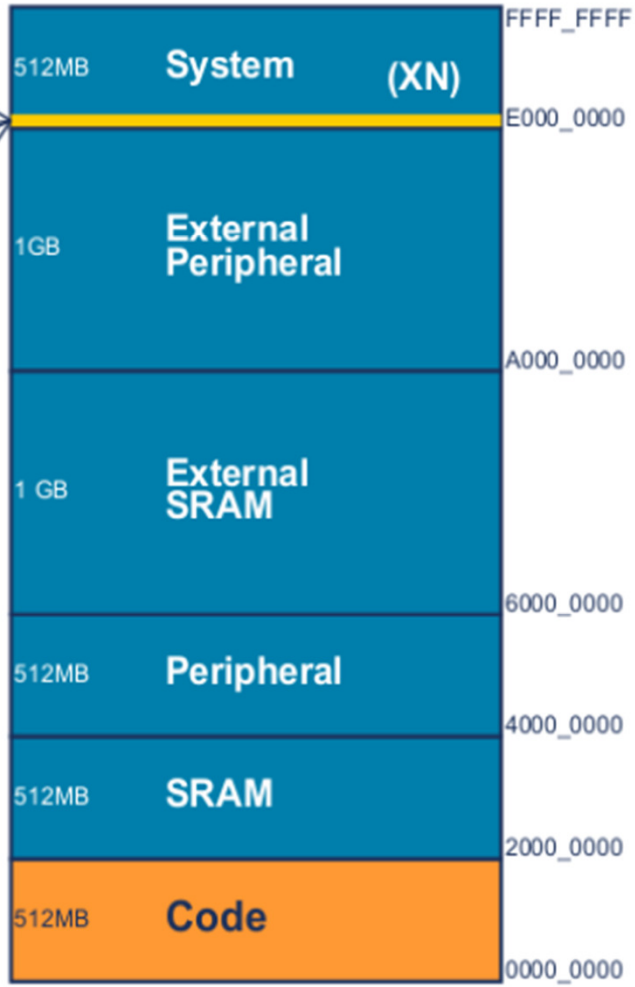
- Ils peuvent adresser 32 bits de mémoire totale (4 Go) selon une **Architecture de Harvard**

- Par défaut, un mot de mémoire est sur **8 bits**.

Memory Mapping

XN (Execute Never): interdiction de tout essai de chercher une instruction à décoder dans ces zones.





Note: dans votre programme nous nous limitons à la mémoire CODE (FLASH) et DATA (SRAM), NVIC

- Le 4 Go adressables par un cœur ARM Cortex M4 sont réparties entre la mémoire non-volatile (FLASH), la mémoire volatile interne et externe (RAM), les périphériques, des adresses réservées aux périphériques du vendeur et plus

- Ce cœur est un cœur Memory Mapped Input Output: les adresses des périphériques sont dans le même espace d'adresses que les mémoires.

- Un fabricant voulant intégrer ce cœur dans un microcontrôleur qui lui est propre pourra mettre jusqu'à 512 MB de FLASH, 512 de mémoire RAM interne, 1GB de mémoire RAM externe, plusieurs périphériques.

Architecture 32 bits

➢ Ces cœurs ont une architecture 32 bits: Bus de données sur 32 bits, Registres internes sur 32 bits

➢ Architecture Harvard:

✓ un bus système pour les données et les périphériques. Ce bus est connecté à une matrice d'interconnexion reliée à la mémoire SRAM et aux périphériques

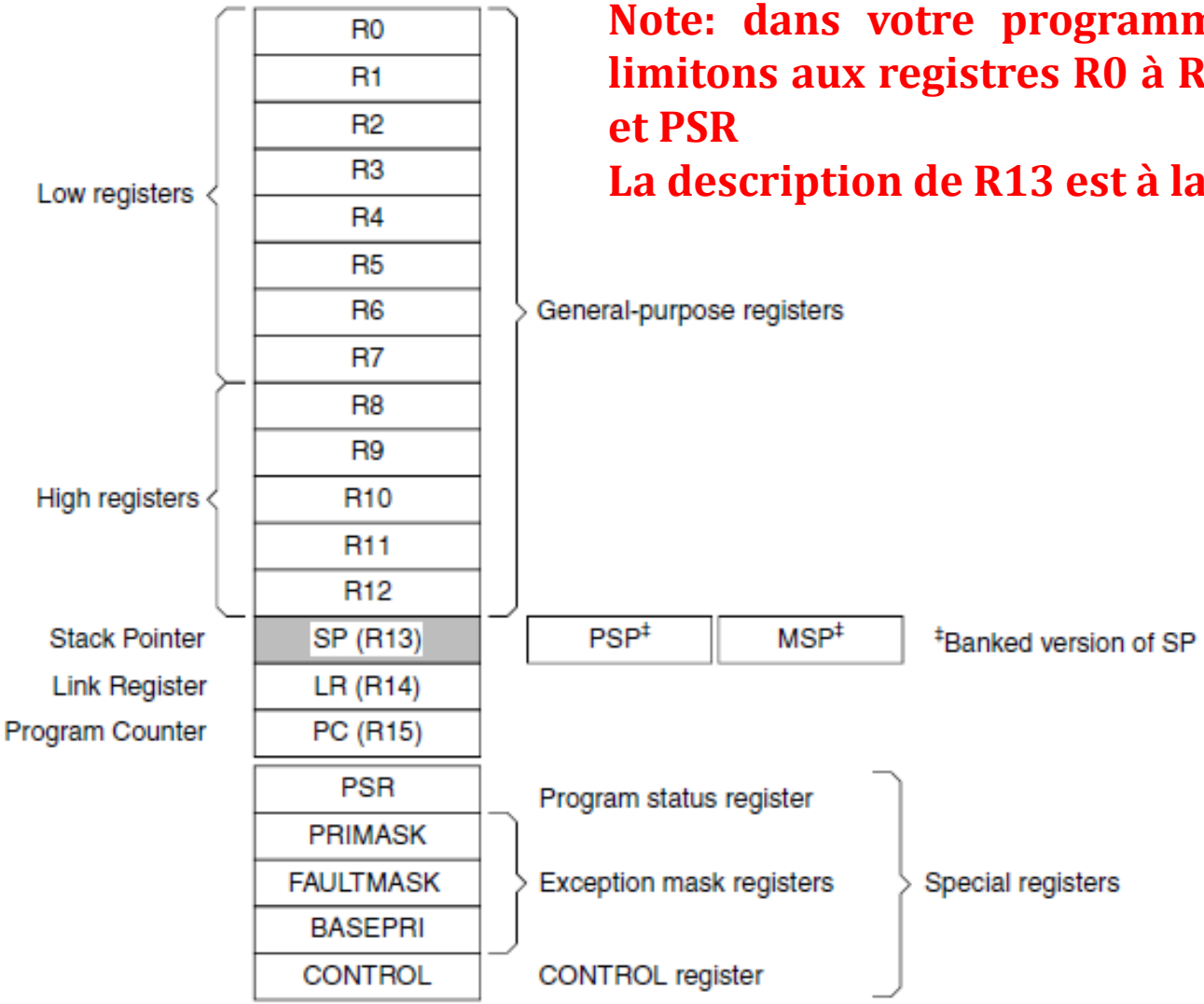
✓ deux bus pour les instructions (LCode et DCode) qui permettent de les charger (LCode) ou de lire la mémoire flash (DCode)

➢ Modèle Load-Store pour accéder aux données de la mémoire

➢ Instructions 16 ou 32 bits: mode Thumb2 (compromis entre performance et densité de code)

➢ Les instructions peuvent être exécutées sur des variables 8 bits (octet), 16 bits (demi-mot) ou 32 bits (mot).

Banc de registres



Note: dans votre programme nous nous limitons aux registres R0 à R12, R14 , R15 et PSR

La description de R13 est à la fin

➢ **R0-R12 : registres généraux**

✓ Les registres bas sont des registres disponibles pour toutes les instructions, encodées sur 16 ou 32 bits.

✓ Les registres hauts disponibles uniquement pour les instructions 32 bits.

✓ Il faut 3 bits pour désigner un registre bas et 4 bits pour désigner tous les registres de R0 à R15.

➢ **R15: registre PC ou compteur de programme**

Il indique l'adresse des instructions à être exécutées.

Le processeur faisait d'une façon continuelle :

1. charger l'instruction adressée par le PC
2. la décoder, et incrémente PC (+2 et/ou +4 selon le mode)
3. l'exécuter, ce qui modifie les registres et/ou la mémoire

Changer R15 équivaut à faire un saut à travers le code du programme

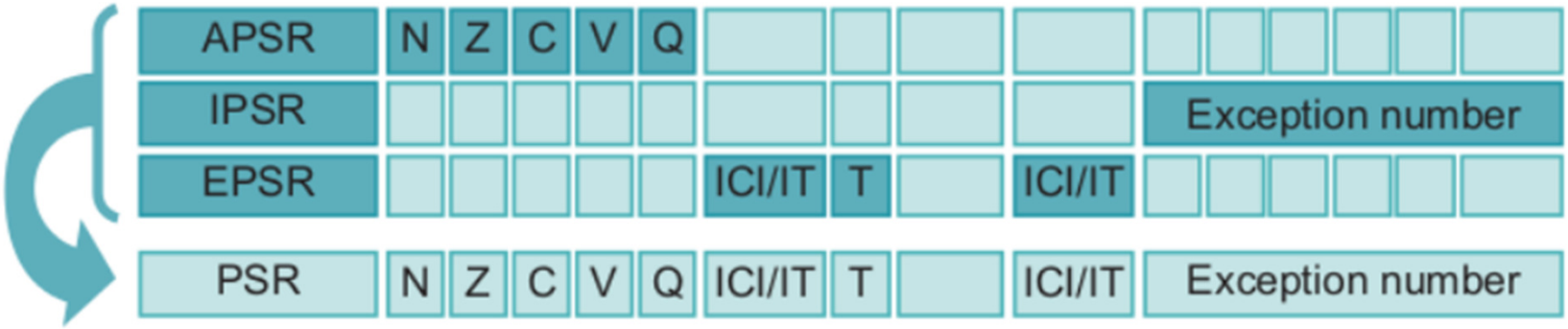
- **R14 : Link register (LR)**

il est automatiquement mis à jour lors d'appel de fonction. Il contient l'adresse de retour de la fonction (l'adresse de la fonction à être exécutée lorsque l'exécution de la fonction sera terminée)

- **Registre d'état PSR (Program Status Register)**

✓ Il indique l'état d'exécution d'un programme: la valeur des drapeaux de l'ALU (ASPR), l'état d'exécution (EPSR) ou le numéro de l'interruption en cours (IPSR)

✓ Peut être vu comme un tout (PSR) ou comme trois registres séparés (virtuellement), chaque registre met en relief une partie des bits



- PSR ne fait pas partie des 16 registres pouvant être adressés par les instructions arithmétiques/logiques du microprocesseur. Il faut utiliser des instructions spéciales pour lire et écrire ce registre.

- **APSR** contient 5 drapeaux (flags):
  - ✓ N indique si le résultat de l'ALU est négatif,
  - ✓ Z indique si le résultat est nul,
  - ✓ C (Carry) indique si l'ALU a produit une retenue lors d'une addition,
  - ✓ V (Overflow) indique que dans l'opération précédente il y'a un dépassement
  - ✓ Q (Sticky Saturation) indique que le résultat a saturé (exemple: minimums et maximums atteints)

- **EPSR** indique si le microprocesseur exécute actuellement une instruction If-Then ou une instruction Load/Store Multiple Registre.
- **ISPR** indique le numéro de l'interruption en cours.

Les interruptions

❖ **Exemples**

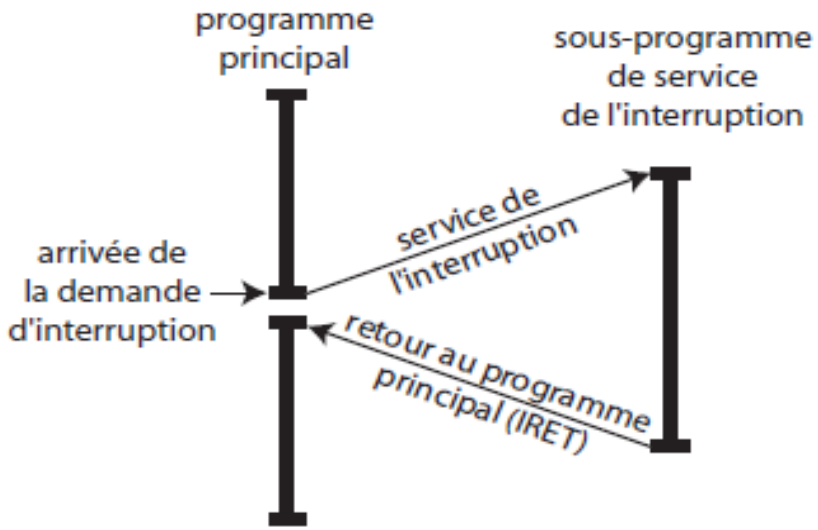
✓ appui sur un bouton

✓ déclenchement d'un capteur

✓ arrivée d'un message sur un canal de communication

✓ horloge temps réel (timer) informant qu'un temps est écoulé; par exemple "1 ms s'est écoulé"

→ **interruption:** Mécanisme qui permet d'interrompre l'exécution d'un programme en cours à l'arrivée d'un événement, d'exécuter du code correspondant puis de reprendre le programme interrompu



Les interruptions sur les cœurs ARM

Pour les cœurs ARM:

✓ **une *interruption*** *est spécifiquement liée à un événement* matériel

ex : arrivée d'un message du bloc UART, front montant sur le port GPIO, signal du Timer...

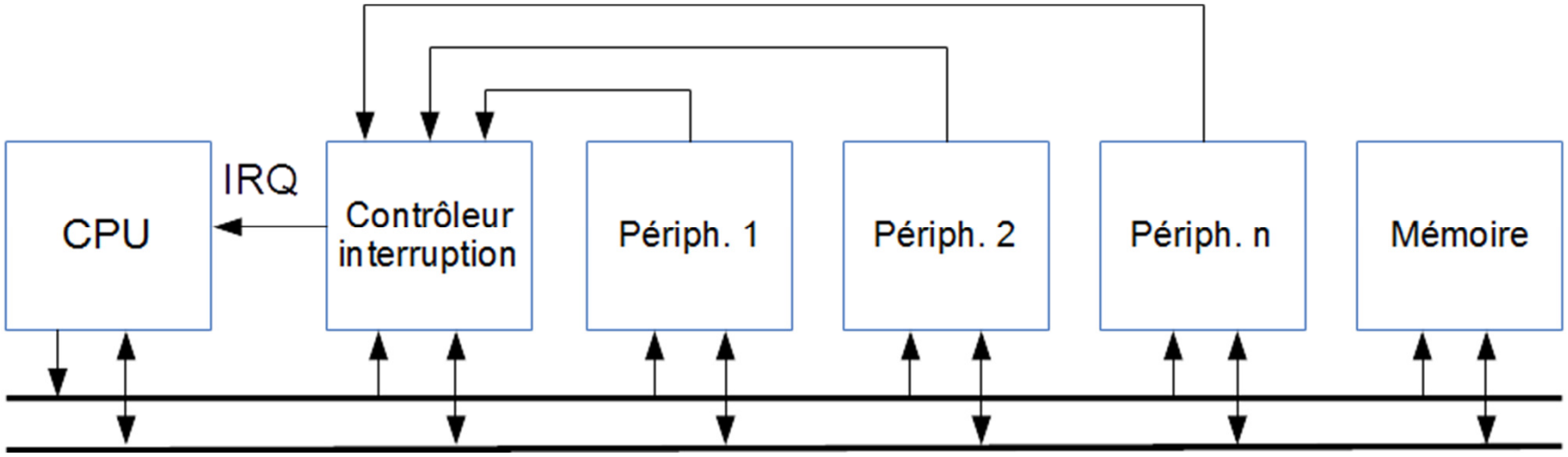
✓ **une *exception*** *peut être liée à un événement logiciel*

ex : division par zéro, erreur d'accès mémoire

Pour certains concepteurs ils considèrent les exceptions comme interruptions logicielles

Les interruptions sur Cortex M4

- de 32 à 256 sources possibles d'interruption (dépend des périphériques installés)
- Il existe un module contrôleur d'interruption appelé le Nested Vectored Interrupt Controller (NVIC) qui est un périphérique adressé entre les adresses 0xE000E000 et 0xE000EFFF (voir Memory Mapping)



❖ **Le rôle du Contrôleur d'interruption NVIC** est de:

✓ Recevoir les requêtes d'interruptions **matérielles** venant des périphériques

✓ Gérer les séquences d'interruptions, la priorité de chaque interruption...

✓ Émettre une requête d'interruption au processeur IRQ

❖ Le processeur utilise cette IRQ qui contient la nature de l'interruption, sa priorité (...) pour rechercher dans une table en mémoire l'adresse du sous-programme d'interruption à exécuter appelé routine d'interruption (Interrupt service routine ISR)

❖ La table est appelée **vecteur d'interruption**

Vecteur d'interruption

- Table en mémoire à une adresse bien définie

- Chaque entrée dans la table correspond à une interruption. Elle contient l'adresse de la routine d'interruption (Interrupt service routine ISR)

- Dans les **Cortex-M4**, cette table est stocké à l'adresse 0x4 de la mémoire FLASH
  - l'interruption 0 s'appelle Reset (appui sur bouton, reset software, allumage)
  - table[0] contient l'adresse du code où sauter en cas de reset

Address		Vector #
0x40 + 4*N	External N	16 + N
…	…	…
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

RQ: la case 0x0 contient l'adresse de début de la pile, copié dans R13 (ou SP) au démarrage.

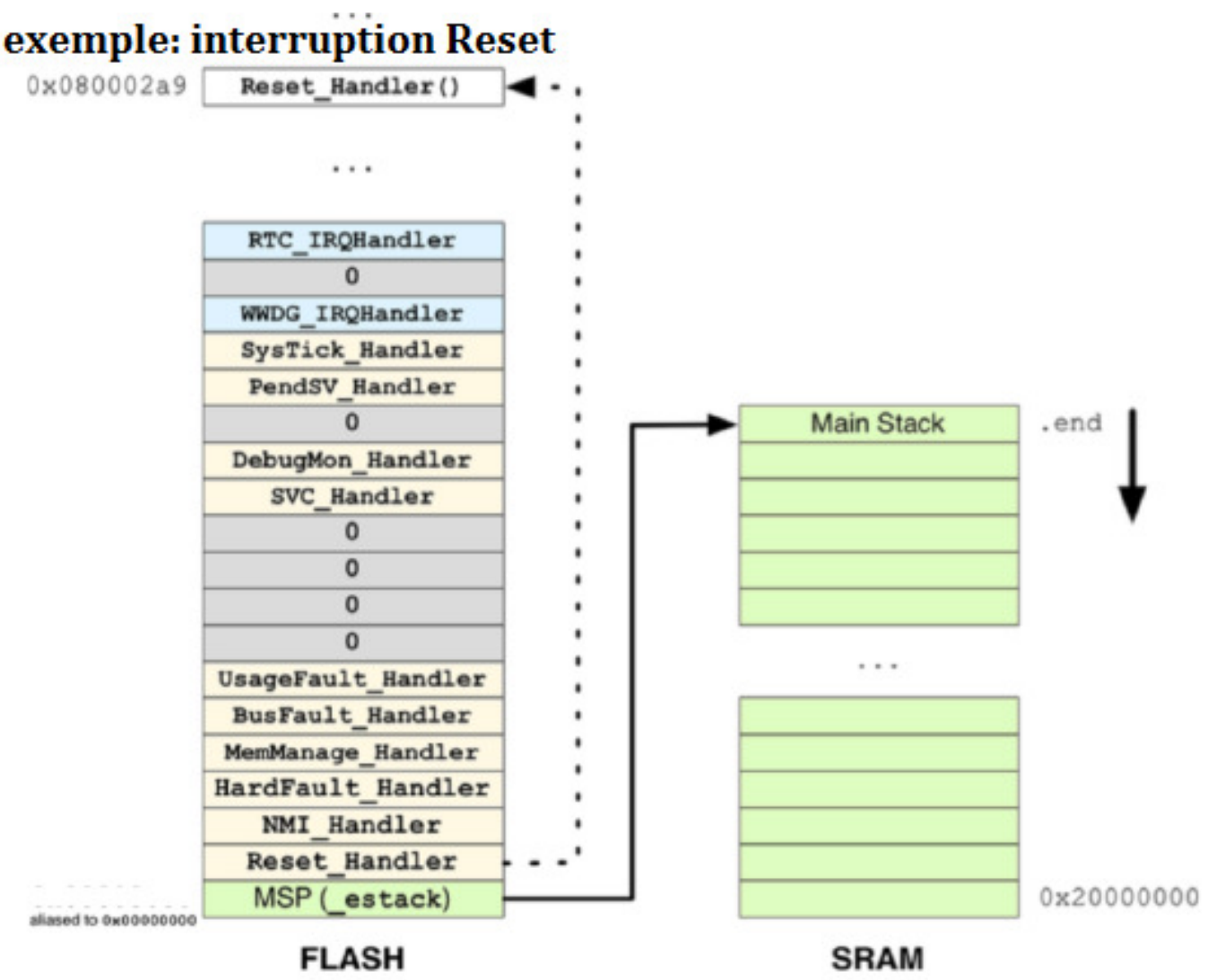
Modèle d'exécution en cas d'interruption

❖ si une interruption *i est levée par le NVIC :*

1) Enregistrer certains registres internes ainsi que le PC dans une zone mémoire SRAM appelée pile et dont l'adresse se trouve dans SP
2) lire dans le *vecteur d'interruption l'adresse stockée à la case*
3) écrire dans PC cette adresse
4) exécuter le code correspondant par avancement du PC

❖ À la fin de l'exécution du code d'une interruption:

1) Reprendre les valeurs des registres et du PC à partir de la pile
2) Exécution de la continuité du programme initial par avancement du PC


exemple: interruption Reset
0x080002a9
Reset_Handler()
RTC_IRQHandler
0
WWDG_IRQHandler
SysTick_Handler
PendSV_Handler
0
DebugMon_Handler
SVC_Handler
0
0
0
0
UsageFault_Handler
BusFault_Handler
MemManage_Handler
HardFault_Handler
NMI_Handler
Reset_Handler
MSP (_estack)
aliased to 0x00000000
FLASH
Main Stack
.end
0x20000000
SRAM

Interruption versus polling

Soit un processeur qui doit échanger des informations avec un périphérique. Il y a deux méthodes possibles pour recevoir les données provenant des périphériques :

- ***Scrutation périodique (ou polling) :*** le programme principal contient des instructions qui lisent cycliquement l'état des ports d'E/S.

Avantage : facilité de programmation.

Inconvénients :

- ✓ perte de temps s'il y a de nombreux périphériques à interroger ;
- ✓ de nouvelles données ne sont pas toujours présentes ;
- ✓ des données peuvent être perdues si elles changent rapidement.

- ***Interruption :*** lorsqu'une donnée apparaît sur un périphérique, le circuit d'E/S le signale au processeur pour que celui-ci effectue la lecture de la donnée : c'est une *demande d'interruption (IRQ : Interrupt Request)*