REDS
Reconfigurable & embedded Digital Systems

heig-vd
HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD
www.heig-vd.ch

Portage de systèmes d'exploitation (POS)
Accès mémoires

Prof. Daniel Rossier

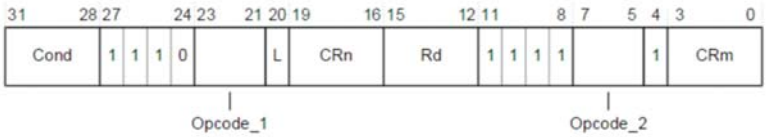
Version 1.3 (2017-2018)

Plan

- Coprocesseur ARM ***cp15***
- Gestion de la MMU
- Tables des pages

Coprocesseur ARM (1/5)

- Coprocesseur ARM ***cp15***
- Instructions privilégiées ***mcr*, mrc**



- *Lecture*

```
MRC{cond} P15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

- *Ecriture*

```
MCR{cond} P15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

3 Cours POS - Institut REDS/HEIG-VD - Introduction

Dans un processeur *ARM*, le coprocesseur permet d'accéder à des informations de type *système* comme l'adresse physique de la table de page de premier niveau.

Les registres de ce coprocesseur sont manipulés exclusivement au travers des instructions privilégiées comme *mcr* ou *mrc.*

Le registre de contrôle (aussi appelé *CR* pour *Control Register*) est présenté ci-dessus et contient des informations relatives à la configuration de base comme l'activation de la MMU, le test d'alignement sur les données, l'emplacement de la table des vecteurs, etc.

Coprocesseur ARM (2/5)

Instruction	Operation
MRC p15, 0, <Rd>, c0, c0, 0	Read ID Code Register
MRC p15, 0, <Rd>, c0, c0, 1	Read Cache Type Register
MRC p15, 0, <Rd>, c0, c0, 3	Read TLB Type Register
MRC p15, 0, <Rd>, c0, c0, 5	Read CPU ID Register
MRC p15, 0, <Rd>, c0, c0, 0	Read Proc Feature Register 0
MRC p15, 0, <Rd>, c0, c1, 1	Read Proc Feature Register 1
MRC p15, 0, <Rd>, c0, c1, 2	Read Debug Feature Register 0
MRC p15, 0, <Rd>, c0, c1, 4	Read Memory Feature Register 0
MRC p15, 0, <Rd>, c0, c1, 5	Read Memory Feature Register 1
MRC p15, 0, <Rd>, c0, c1, 6	Read Memory Feature Register 2
MRC p15, 0, <Rd>, c0, c1, 7	Read Memory Feature Register 3
MRC p15, 0, <Rd>, c0, c2, 0	Read ISA Feature Register 0
MRC p15, 0, <Rd>, c0, c2, 1	Read ISA Feature Register 1
MRC p15, 0, <Rd>, c0, c2, 2	Read ISA Feature Register 2
MRC p15, 0, <Rd>, c0, c2, 3	Read ISA Feature Register 3
MRC p15, 0, <Rd>, c0, c2, 4	Read ISA Feature Register 4
MRC p15, 0, <Rd>, c1, c0, 0	Read Control Register
MCR p15, 0, <Rd>, c1, c0, 0	Write Control Register
MRC p15, 0, <Rd>, c1, c0, 1	Read Auxiliary Control Register
MCR p15, 0, <Rd>, c1, c0, 1	Write Auxiliary Control Register
MRC p15, 0, <Rd>, c1, c0, 2	Read Coprocessor Access Control Register
MCR p15, 0, <Rd>, c1, c0, 2	Write Coprocessor Access Control Register
MRC p15, 0, <Rd>, c2, c0, 0	Read Translation Table Base Register 0
MCR p15, 0, <Rd>, c2, c0, 0	Write Translation Table Base Register 0
MRC p15, 0, <Rd>, c2, c0, 1	Read Translation Table Base Register 1
MCR p15, 0, <Rd>, c2, c0, 1	Write Translation Table Base Register 1
MRC p15, 0, <Rd>, c2, c0, 2	Read Translation Table Base Control Register
MCR p15, 0, <Rd>, c2, c0, 2	Write Translation Table Base Control Register
MRC p15, 0, <Rd>, c3, c0, 0	Read Domain Access Control Register
MCR p15, 0, <Rd>, c3, c0, 0	Write Domain Access Control Register

Instruction	Operation
MRC p15, 0, <Rd>, c5, c0, 0	Read Data Fault Status Register
MCR p15, 0, <Rd>, c5, c0, 0	Write Data Fault Status Register
MRC p15, 0, <Rd>, c5, c0, 1	Read Instruction Fault Status Register
MCR p15, 0, <Rd>, c5, c0, 1	Write Instruction Fault Status Register
MRC p15, 0, <Rd>, c6, c0, 0	Read Fault Address Register
MCR p15, 0, <Rd>, c6, c0, 0	Write Fault Address Register
MRC p15, 0, <Rd>, c6, c0, 1	Read Watchpoint Fault Address Register
MCR p15, 0, <Rd>, c6, c0, 1	Write Watchpoint Fault Address Register
MCR p15, 0, <Rd>, c7, c0, 4	Wait For Interrupt
MRC p15, 0, <Rd>, c7, c4, 0	PA Register
MCR p15, 0, <Rd>, c7, c5, 0	Invalidate Entire Instruction Cache Register
MCR p15, 0, <Rd>, c7, c5, 1	Invalidate Instruction Cache Line (using MVA) Register
MCR p15, 0, <Rd>, c7, c5, 2	Invalidate Instruction Cache Line (using Index) Register
MCR p15, 0, <Rd>, c7, c5, 4	Flush Prefetch Buffer Register
MCR p15, 0, <Rd>, c7, c5, 6	Flush Entire Branch Target Cache Register
MCR p15, 0, <Rd>, c7, c5, 7	Flush Branch Target Cache Entry Register
MCR p15, 0, <Rd>, c7, c6, 0	Invalidate Entire Data Cache Register
MCR p15, 0, <Rd>, c7, c6, 1	Invalidate Data Cache Line (using MVA) Register
MCR p15, 0, <Rd>, c7, c6, 2	Invalidate Data Cache Line (using Index) Register
MCR p15, 0, <Rd>, c7, c7, 0	Invalidate Both Caches Register
MCR p15, 0, <Rn>, c7, c8, 0	VA to PA with privileged read permission check Register
MCR p15, 0, <Rn>, c7, c8, 1	VA to PA with privileged write permission check Register
MCR p15, 0, <Rn>, c7, c8, 2	VA to PA with user read permission check Register
MCR p15, 0, <Rn>, c7, c8, 3	VA to PA with user write permission check Register
MCR p15, 0, <Rd>, c7, c10, 0	Clean Entire Data Cache Register
MCR p15, 0, <Rd>, c7, c10, 1	Clean Data Cache Line (using MVA) Register
MCR p15, 0, <Rd>, c7, c10, 2	Clean Data Cache Line (using Index) Register
MCR p15, 0, <Rd>, c7, c10, 4	Drain Synchronization Barrier Register
MCR p15, 0, <Rd>, c7, c10, 5	Data Memory Barrier Register
MCR p15, 0, <Rd>, c7, c14, 0	Clean and Invalidate Entire Data Cache Register
MCR p15, 0, <Rd>, c7, c14, 1	Clean and Invalidate Data Cache Line (using MVA) Register
MCR p15, 0, <Rd>, c7, c14, 2	Clean and Invalidate Data Cache Line (using Index) Register

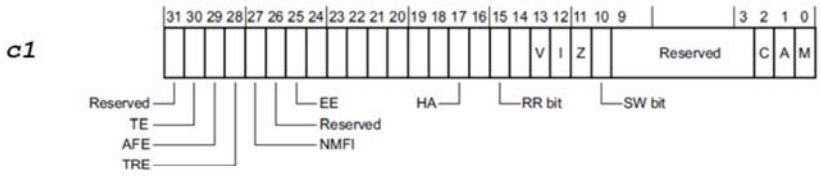
Coprocesseur ARM (3/5)

Instruction	Operation
MCR p15,0, <Rd>, c8, C5, 0	Invalidate Instruction TLB Register
MCR p15,0, <Rd>, c8, C5, 1	Invalidate Instruction TLB Single Entry Register
MCR p15,0, <Rd>, c8, C5, 2	Invalidate Instruction TLB Entry on ASID match Register
MCR p15,0, <Rd>, c8, C5, 3	Invalidate Instruction TLB Single Entry on MVA only Register
MCR p15,0, <Rd>, c8, C6, 0	Invalidate Data TLB Register
MCR p15,0, <Rd>, c8, C6, 1	Invalidate Data TLB Single Entry Register
MCR p15,0, <Rd>, c8, C6, 2	Invalidate Data TLB Entry on ASID match Register
MCR p15,0, <Rd>, c8, C6, 3	Invalidate Data TLB Single Entry on MVA only Register
MCR p15,0, <Rd>, c8, C7, 0	Invalidate Unified TLB Register
MCR p15,0, <Rd>, c8, C7, 1	Invalidate Unified TLB Single Entry Register
MCR p15,0, <Rd>, c8, C7, 2	Invalidate Unified TLB Entry on ASID match Register
MCR p15,0, <Rd>, c8, C7, 3	Invalidate Unified TLB Single Entry on MVA only Register
MRC p15, 0, <Rd>, c9, c0, 0	Read Data Cache Lockdown Register
MCR p15, 0, <Rd>, c9, c0, 0	Write Data Cache Lockdown Register
MRC p15, 0, <Rd>, c10, c0, 0	Read TLB Lockdown Register
MCR p15, 0, <Rd>, c10, c0, 0	Write TLB Lockdown Register
MRC p15, 0, <Rd>, c10, c2, 0	Read Primary Remap Register
MCR p15, 0, <Rd>, c10, c2, 0	Write Primary Remap Register
MRC p15, 0, <Rd>, c10, c2, 1	Read Normal Remap Register
MCR p15, 0, <Rd>, c10, c2, 1	Write Normal Remap Register
MRC p15, 0, <Rd>, c13, c0, 0	Read Process ID Register
MCR p15, 0, <Rd>, c13, c0, 0	Write Process ID Register
MRC p15, 0, <Rd>, c13, c0, 1	Read Context ID Register
MCR p15, 0, <Rd>, c13, c0, 1	Write Context ID Register
MRC p15, 0, <Rd>, c13, c0, 2	Read Thread ID User and Privileged Read Write Register
MCR p15, 0, <Rd>, c13, c0, 2	Write Thread ID User and Privileged Read Write Register
MRC p15, 0, <Rd>, c13, c0, 3	Read Thread ID User Read only Register
MCR p15, 0, <Rd>, c13, c0, 3	Write Thread ID User Read only Register
MRC p15, 0, <Rd>, c13, c0, 4	Read Thread ID Privileged Read Write only Register
MCR p15, 0, <Rd>, c13, c0, 4	Write Thread ID Privileged Read Write only Register

Instruction	Operation
MRC p15, 0, <Rd>, c15, c12, 0	Read Performance Monitor Control Register
MCR p15, 0, <Rd>, c15, c12, 0	Write Performance Monitor Control Register
MRC p15, 0, <Rd>, c15, c12, 1	Read Cycle Counter Register
MCR p15, 0, <Rd>, c15, c12, 1	Write Cycle Counter Register
MRC p15, 0, <Rd>, c15, c12, 2	Read Count Register 0
MCR p15, 0, <Rd>, c15, c12, 2	Write Count Register 0
MRC p15, 0, <Rd>, c15, c12, 3	Read Count Register 1
MCR p15, 0, <Rd>, c15, c12, 3	Write Count Register 1
MCR p15, 5, <Rd>, c15, c4, 2	Read Main TLB Entry Register
MCR p15, 5, <Rd>, c15, c4, 4	Write Main TLB Entry Register
MRC p15, 5, <Rd>, c15, c5, 2	Read Main TLB VA Register
MCR p15, 5, <Rd>, c15, c5, 2	Write Main TLB VA Register
MRC p15, 5, <Rd>, c15, c6, 2	Read Main TLB PA Register
MCR p15, 5, <Rd>, c15, c6, 2	Write Main TLB PA Register
MRC p15, 5, <Rd>, c15, c7, 2	Read Main TLB Attribute Register
MCR p15, 5, <Rd>, c15, c7, 2	Write Main TLB Attribute Register
MRC p15, 7, <Rd>, c15, c1, 0	Read TLB Debug Control Register
MCR p15, 7, <Rd>, c15, c1, 0	Write TLB Debug Control Register

Coprocesseur ARM (4/5)

- Activation/désactivation de la MMU



[0]	M bit	Banked	Enables the MMU: 0 = MMU disabled. This is the reset value. 1 = MMU enabled.

Mise à jour du registre de contrôle

```
mcr     p15, 0, r0, c1, c0, 0
mcr     p15, 0, r0, c1, c0, 0
```

Coprocesseur ARM (5/5)

- Configuration *MMU*

c2

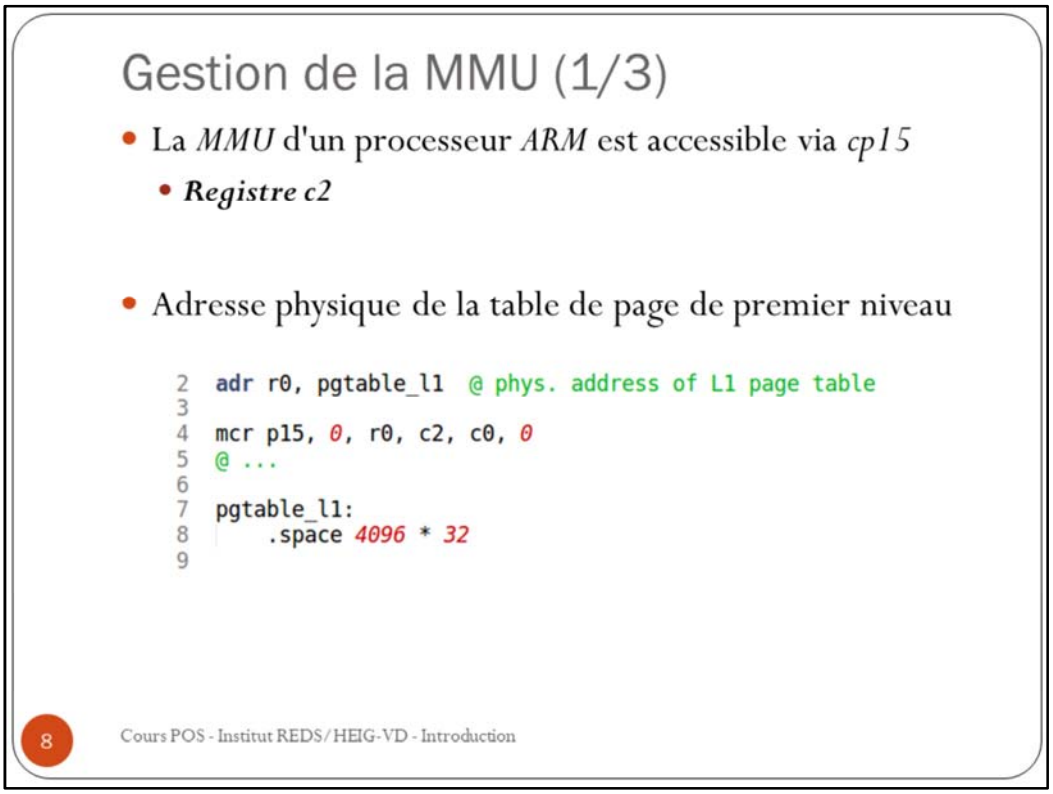
Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	TTBR0	RW	-	Translation Table Base Register 0
		1	TTBR1	RW	-	Translation Table Base Register 1
		2	TTBCR	RW	0x00000000[a]	Translation Table Base Control Register

a. In Secure state only. You must program the Non-secure version with the required value.

- Configuration du contrôle d'accès

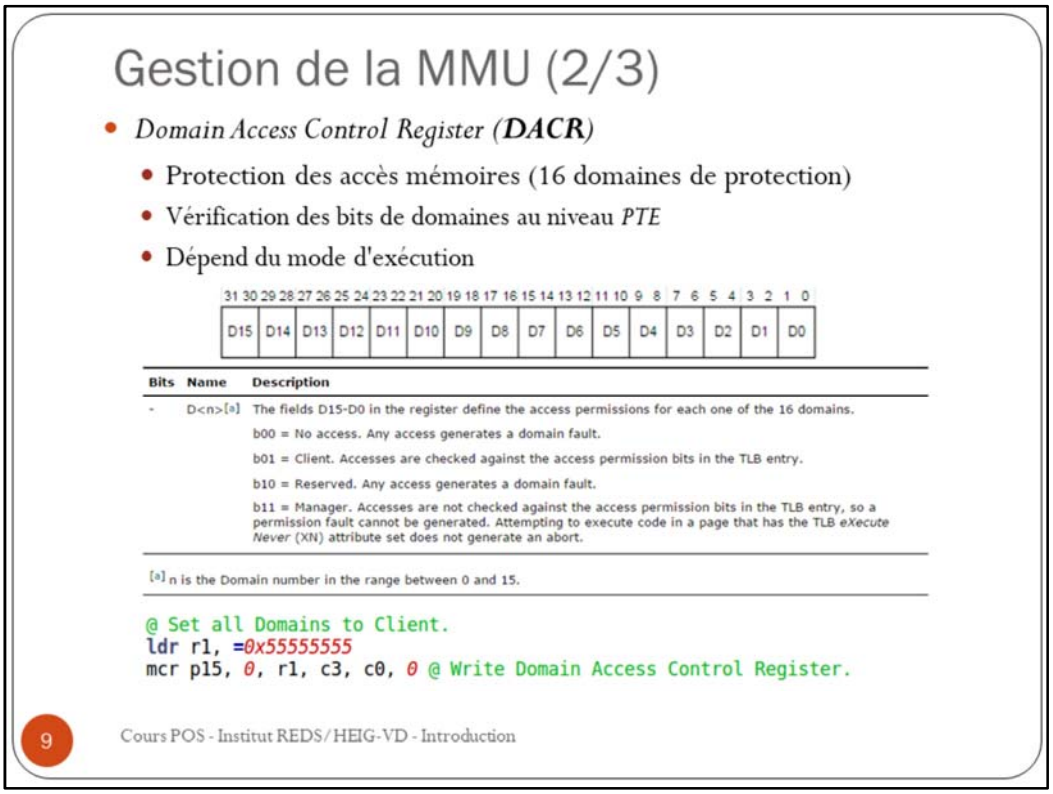
c3

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DACR	RW	-	Domain Access Control Register



Les registre *c2* et *c3* du coprocesseur *cp15* permettent respectivement de configurer l'adresse physique de la table de page de premier niveau ainsi que les droits d'accès aux différents domaines.

La notion de domaine permet de sécuriser les accès mémoires à un premier niveau, lorsque l'accès mémoire se fait au travers de la *MMU*. Le mécanisme est décrit dans les pages suivantes.

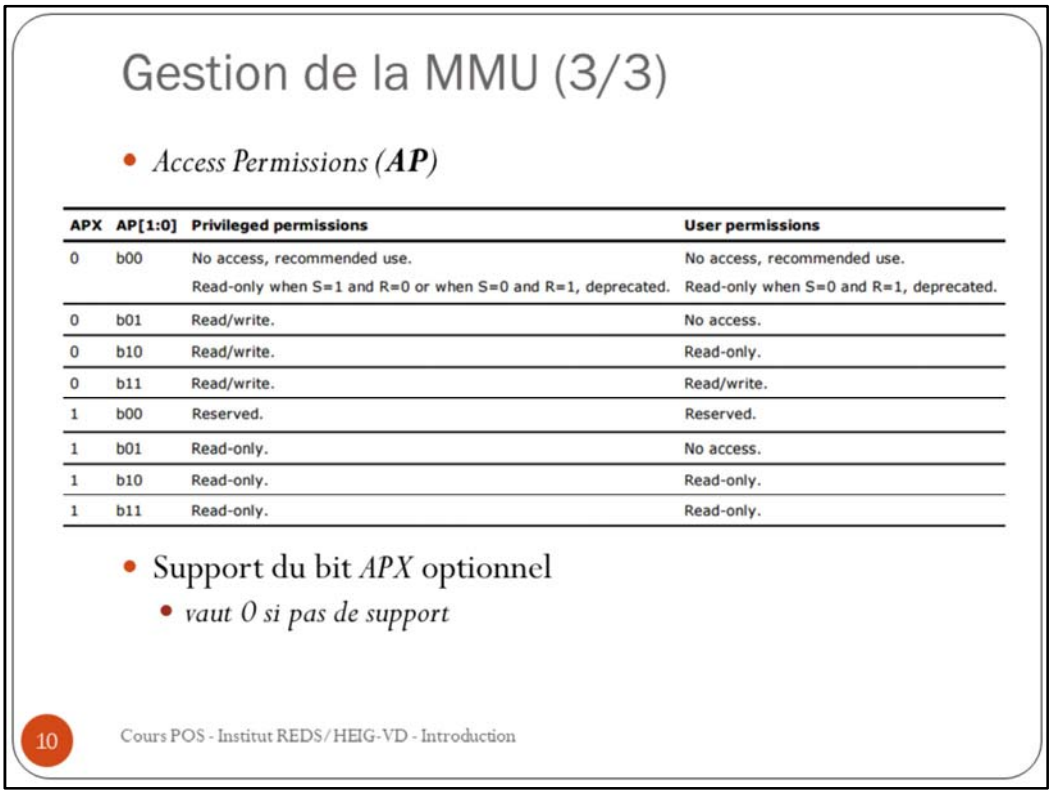


Les accès mémoires peuvent être sécurisés grâce à la notion de ***domaine***. On peut disposer de 16 domaines distincts, chacun pouvant disposer de sa propre politique.

Il existe ainsi 3 politiques principales encodées sur 2 bits : aucun accès autorisé, un mode ***client*** activant le contrôle d'accès, et le mode ***manager*** autorisant tout accès.

Un registre principale – appelé ***DACR*** *(Domain Access Control Register)* – indique le mode courant (*aucun accès, mode client, mode manager*) permettant de prendre en compte ou non les politiques attribuées aux domaines.

Lorsque le mode client est activé, la *MMU* vérifie les bits de permission (***Access Permission***) encodés dans la PTE pour valider l'accès mémoire. La définition des bits ***AP*** sont décrits sur la page suivante.



Gestion de la MMU (3/3)

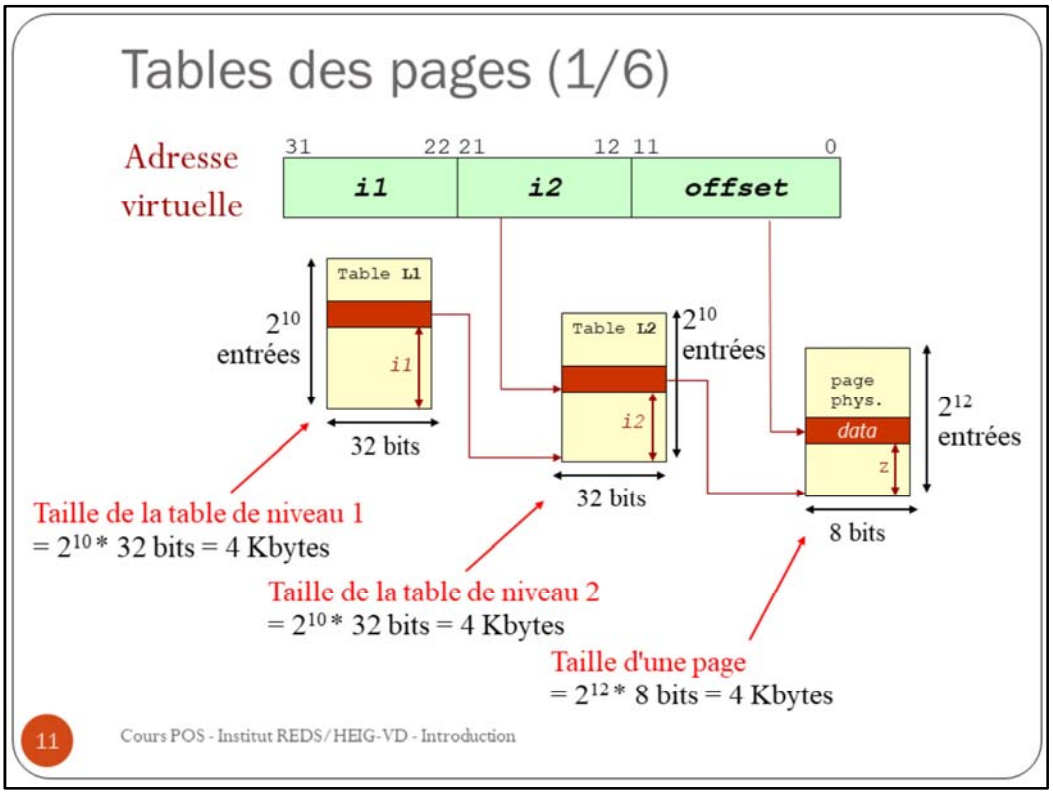
- *Access Permissions* (***AP***)

APX	AP[1:0]	Privileged permissions	User permissions
0	b00	No access, recommended use. Read-only when S=1 and R=0 or when S=0 and R=1, deprecated.	No access, recommended use. Read-only when S=0 and R=1, deprecated.
0	b01	Read/write.	No access.
0	b10	Read/write.	Read-only.
0	b11	Read/write.	Read/write.
1	b00	Reserved.	Reserved.
1	b01	Read-only.	No access.
1	b10	Read-only.	Read-only.
1	b11	Read-only.	Read-only.

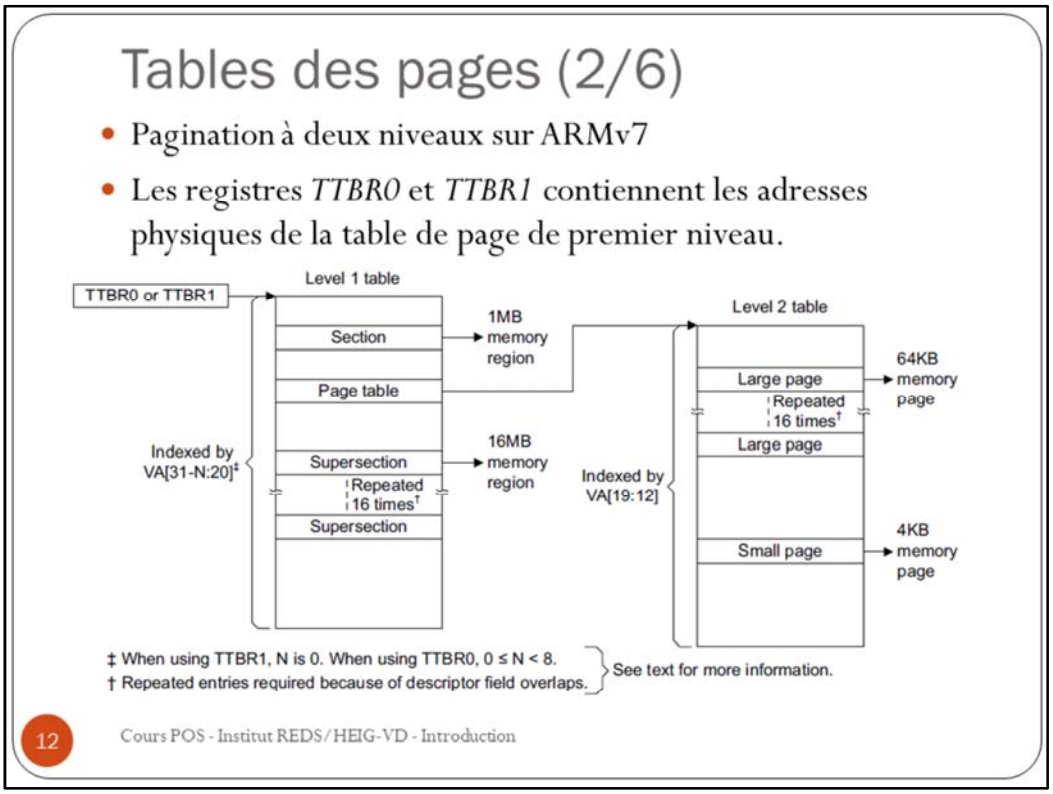
- Support du bit *APX* optionnel
  - *vaut 0 si pas de support*

Les bits de permission sont codés sur 2 bits (voire 3), et sont utilisés pour définir le type de permission : accès en lecture et/ou écriture, ou aucun accès autorisé, selon le mode d'exécution du processeur.

La *PTE* peut contenir plusieurs paires de *bits AP* permettant une granularité inférieure à la taille de page mappée (on parle alors de *sous-page*); la même *PTE* représente la même page subdivisée en sous-pages pour lesquelles des permissions différentes peuvent être attribuées. Par exemple, une page de 4 Ko est constituée de 4 sous-pages de 1 Ko.



L'exemple ci-dessus montre un découpage possible de l'adresse virtuelle en trois parties : **l'index de niveau 1** codé sur 10 bits, **l'index de niveau 2** codé sur 10 bits également, et **l'offset** codé sur 12 bits. Il s'agit d'un découpage habituel sur un système 32 bits de type *x86* par exemple.



Un cœur ARM supporte différentes granularités de mappage. A partir de la génération des CPUs ARMv7, il est possible de réaliser un mappage de page de 1 Mo (appelé *section*), de 16 Mo (supersecton), ou des pages de 4 Ko *(small)* ou de 16 Ko (*large*).

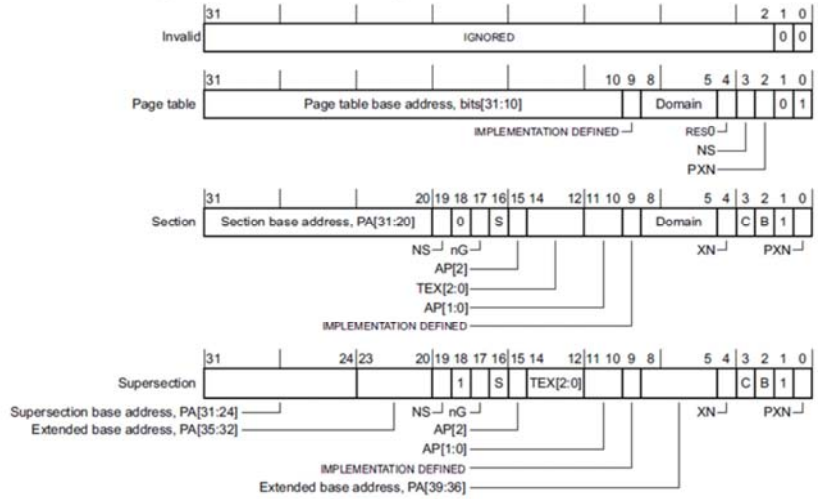
Dans un noyau Linux, on utilise fréquemment les mappages de section (zones contrôlées par le noyau, zones I/O) ainsi que les pages de 4 Ko (zones utilisateur et certaines zones noyau).

La structure de la table des pages de premier niveau est commune aux différents types de mappage, à savoir une table de 16 Ko contenant 4096 entrées (PTEs) d'une taille de 4 octets (32 bits) chacune. Les bits d'une PTE de premier niveau permet de déterminer le type de mappage.

La structure des tables des pages de second niveau varie en fonction du type de descripteur, selon que le mappage s'effectue sur des pages de 4 Ko ou de 64 Ko. Les deux types de descripteur sont présentés sur les pages suivantes.

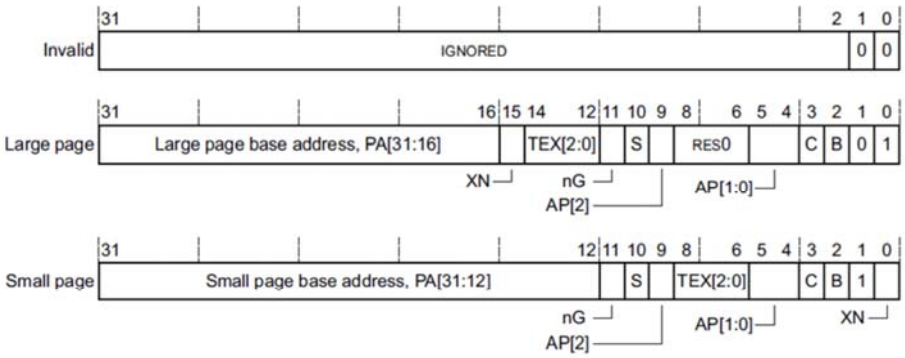
Tables des pages (3/6)

- Descripteur de PTE de premier niveau



Tables des pages (4/6)

- Descripteur de second niveau



Tables des pages (5/6)

- Description succincte des attributs de PTE

TEX[2:0], C, B

Memory region attribute bits, see *Memory region attributes* on page G4-4077.

These bits are not present in a descriptor for a Page table.

XN bit The Execute-never bit. Determines whether the PE can execute software from the addressed region, see *Execute-never restrictions on instruction fetching* on page G4-4071.

This bit is not present in a descriptor for a Page table.

PXN bit The Privileged execute-never bit. Determines whether the PE can execute software from the region when executing at PL1, see *Execute-never restrictions on instruction fetching* on page G4-4071.

Note

Memory accesses by software executing at EL2 always use the Long-descriptor translation table format.

When this bit is set to 1 in the descriptor for a Page table, it indicates that all memory pages described in the corresponding Page table are Privileged execute-never.

NS bit Non-secure bit. Specifies whether the translated PA is in the Secure or Non-secure address map, see *Control of Secure or Non-secure memory access, VMSAv8-32 Short-descriptor format* on page G4-4045.

This bit is not present in level 2 descriptors. The value of the NS bit in a level 1 descriptor for a Page table applies to all entries in the corresponding level 2 translation table.

Domain Domain field, see *Domains, Short-descriptor format only* on page G4-4073.

This field is not present in a Supersection entry. Memory described by Supersections is in domain 0.

This bit is not present in level 2 descriptors. The value of the Domain field in the level 1 descriptor for a Page table applies to all entries in the corresponding level 2 translation table.

An IMPLEMENTATION DEFINED bit

This bit is not present in level 2 descriptors.

Tables des pages (6/6)

- (suite)

AP[2], AP[1:0]

Access Permissions bits, see *Memory access control* on page G4-4068.

AP[0] can be configured as the *Access flag*, see *The Access flag* on page G4-4074.

These bits are not present in a descriptor for a Page table.

S bit Shareable bit. Used in determining the Shareability of the addressed region, see *Memory region attributes* on page G4-4077.

Note

The naming of this bit as the *Shareable* bit is carried forward from early versions of the ARM architecture. This name is no longer an adequate description of the interpretation of the bit.

This bit is not present in a descriptor for a Page table.

nG bit The not global bit. If a lookup using this descriptor is cached in a TLB, determines whether the TLB entry applies to all ASID values, or only to the current ASID value. See *Global and process-specific translation table entries* on page G4-4089.

This bit is not present in a descriptor for a Page table.

Bit[18], when bits[1:0] indicate a *Section or Supersection* descriptor

0 Descriptor is for a Section.

1 Descriptor is for a Supersection.

Références

- Pierre Ficheux, *Linux embarqué*, 3e édition, Éditions Eyrolles (2010)
- RTEMS Real Time Operating System (RTOS)
  http://rtems.org