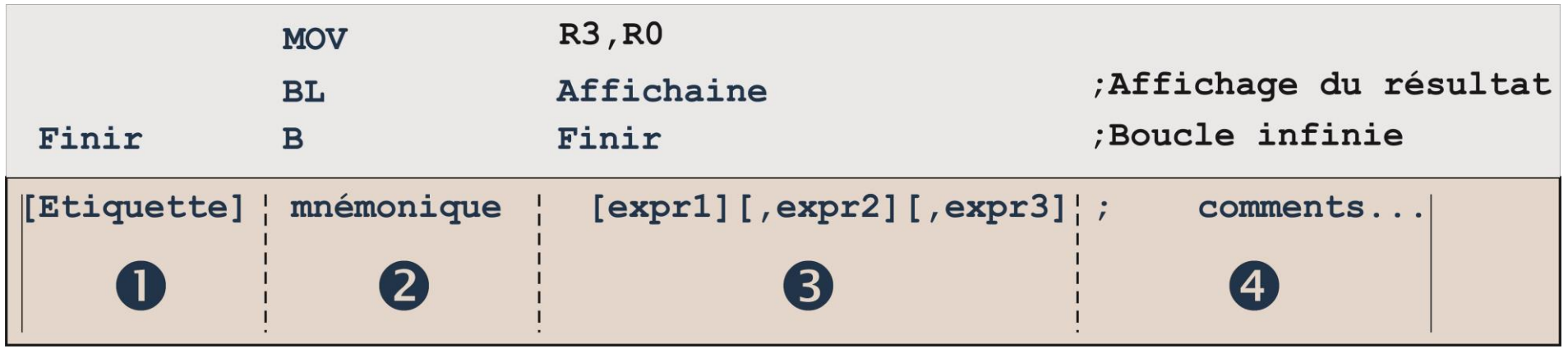
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Le jeu d'instruction

Connaitre ce que sait faire un **Cortex**

Revenons à une ligne type

- Le seul champ obligatoire est le champ ❷



- Le mnémonique = l'instruction fait à chaque cycle d'horloge.
- L'ensemble des mnémoniques = jeu d'instructions

Le jeu d'instructions

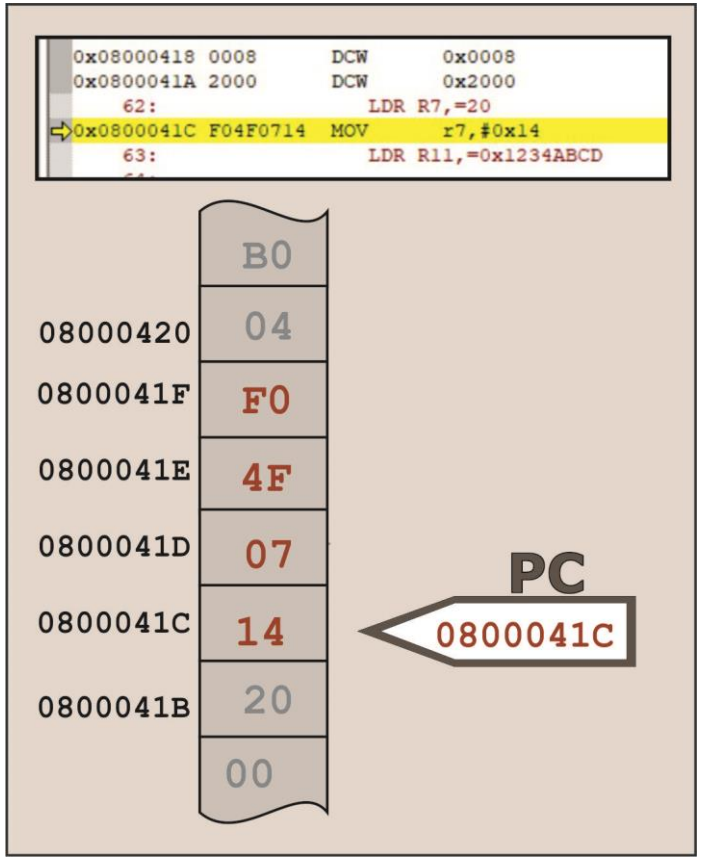
- Un programme = une suite d'instructions…
  - + que peut – on faire ?
    - peu de chose à chaque cycle
    - 1 instruction = 1 opération rudimentaire
    - ne connait directement pas les structures if, while, for…
  - + *avec quoi* ou *sur quoi* travaille l'instruction
    - uniquement des octets (8), ½ mots (16) ou mots (32)
    - représente **des entiers** (signée ou non) ou des caractères.
    - le Cortex M3 ne connait pas les nombre flottants
  - + si on veut faire + compliqué
    - combinaison d'instructions (structure algo par ex.)
    - utilisation de bibliothèques (travail en flottant par ex.)

Exécution d'une instruction

- En « nominal » 1 instruction par cycle d'horloge (72Mhz)
  - + mais nécessite 3 cycles pour réaliser l'instruction
- Pipeline en 3 étapes
  1. Récupération :
     - + lecture en mémoire et incrémentation du pointeur d'instructions
  2. Décodage de l'instruction :
     - + « préparation » de l'ALU
  3. Exécution
     - + réalise l'opération demandé
     - + peut mettre à jour les fanions dans le registre d'état **xPSR**

Codage et pointage des instructions

- Une instruction =
  - + codée sur 16 bits (jeu `Thumb`)
  - + ou codée 32 bits (jeu `Thumb-2`)
  - + si besoin d'une donnée immédiate 32 bits
    ⇒ extension à une *literal pool*
- Machine Harvard - Zone adresse :
  0x00000000 à ***0x1FFFFFFF***
- Le pointeur d'instruction (**PC**) est un registre qui contient l'adresse (flash) où se trouve l'instruction courante.
- Une fois l'instruction lue, **PC** est incrémenté de 2 (`Thumb`) ou 4 (`Thumb-2`)
- **PC** est aussi appelé **R15**



Codage et désassemblage

- Le codage d'une instruction est unique
- Pas d'interprétation possible
- Pour tout système informatique si on peut lire la mémoire, on peut récupérer le programme en Langage d'assemblage
- C'est le désassemblage
  - + ce n'est pas lisible pour autant
  - + manque les informations symboliques

```
0x08000346 0B01      LSRS     r1,r0,#12
0x0800034A F3AF8000  NOP.W
0x0800034E F89D0000  LDRB     r0,[sp,#0x00]
0x08000352 B908      CBNZ     r0,0x08000358
0x08000356 E000      B        0x0800035A
0x0800035A F04F0A00  MOV      r10,#0x00
0x0800035E F10B0B01  ADD      r11,r11,#0x01
0x08000362 2C00      CMP      r4,#0x00
0x08000366 EA4F018B  LSL      r1,r11,#2
0x0800036A A803      ADD      r0,sp,#0x0C
0x0800036E FF820004  DCD      0xFF820004
0x08000372 D101      BNE      0x08000378
0x08000376 F8212000  STRH     r2,[r1,r0]
0x0800037A 4642      MOV      r2,r8
0x0800037E 8000      STRH     r0,[r0,#0x00]
0x08000380 E005      B        0x0800038E
0x08000384 1B01      SUBS     r1,r0,r4
0x08000388 1C40      ADDS     r0,r0,#1
0x0800038C 2020      MOVS     r0,#0x20
0x08000390 D3F7      BCC      0x08000382
```

Le registre xPSR

- Quelle(s) différence(s) entre :

```
MOV R7,#0xBF
```

et

```
MOVS R7,#0xBF
```

- Les deux affectent 191 à R7
- Le **MOVS** affecte en plus les fanions
- Fanions = compte rendu sur ce qui s'est passé
- Nécessite le suffixe **S** au mnémonique
  - + vrai pour la grosse majorité des instructions
  - + attention ce **S** n'a aucun rapport avec la notion de signe !
  - + *coté facultatif : particularité de l'assembleur ARM*

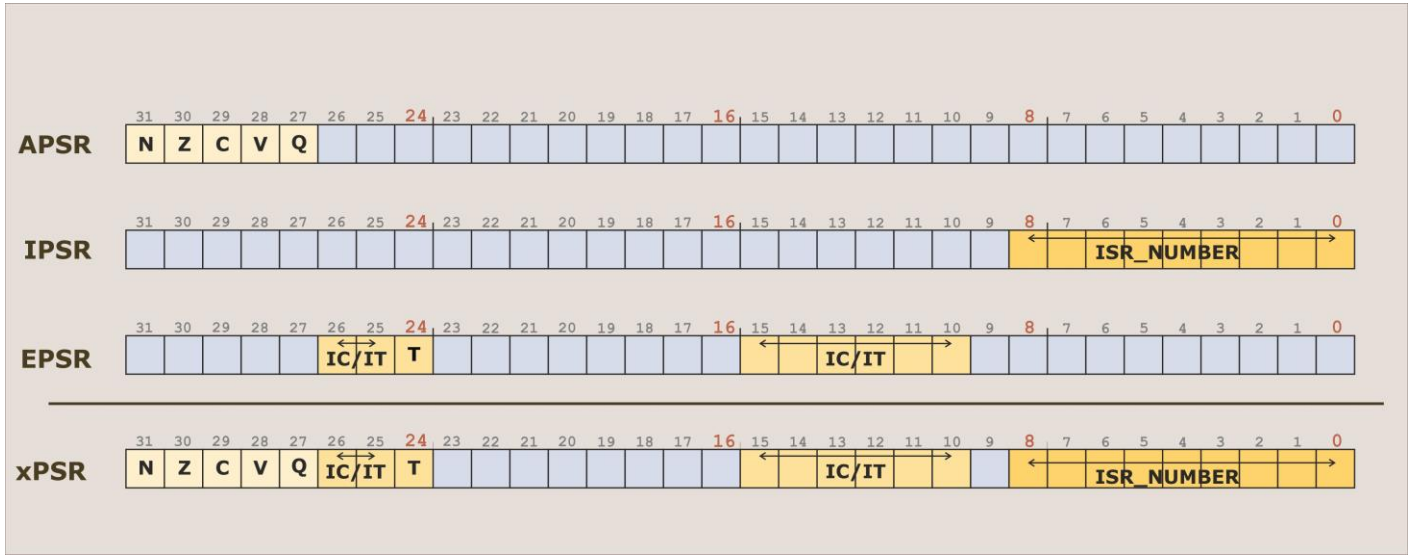
Dommage !

Les instructions « LDR » n'affecte pas les fanions

LDRSH existe mais le S a un sens différent

LDRSHS n'existe pas

Le registre 3 en 1 : xPSR



- ***APSR :*** Applicatif – Mode courant – Celui qui intéresse le programmeur

Contient les fanions N,Z,CV,Q

- ***IPSR :*** Interruption – Contient le numéro courant de l'exception ou de l'interruption
- ***EPSR :*** *Exécution - Contient des informations lors du déroulement d'instructions particulières*

Les fanions

- Fanion **C** (APSR.29): *Carry*
  - représente la retenue lors du calcul sur des grandeurs entières (non signées).
  - C = 1 ⇒ débordement lors de l'instruction précédente (résultat faux).
  - la connaissance de ce bit permet de travailler en précision multiple.
- Fanion **Z** (APSR.30) : *Zéro*
  - attention : vaut 1 si le résultat est nul.
- Indicateur **N** (APSR.31): *Negative*
  - recopie le bit de poids fort du résultat.
  - N à 1 ⇒ résultat négatif
- Indicateur **V** (APSR.28): *oVerflow*
  - V = 1 ⇒ débordement de la représentation signée (résultat signé faux).

Exemple : pour un octet = -128 -2 = -130 non codable sur un octet

$(-128) - 2 = (10000000)_2 - (00000010)$

$= (011111111)_2$

- Indicateur **Q** (APSR.27) : *Sticky saturation flag*
  - sens que pour 2 instructions de saturation USAT et SSAT :
  - Q = 1 ⇒ saturation de la variable traitée.

Composition du jeu

- Approximativement 114 mnémoniques

  + 17 pour les instructions arithmétiques basiques
  + 9 pour les décalages logiques
  + 9 pour les multiplications et divisions
  + 2 pour les instructions de saturation
  + 4 pour les instructions de changement de format
  + 10 pour des instructions arithmétiques particulières
  + 2 pour la récupération du registre xPSR
  + 8 pour les instructions de branchement 24 pour les opérations lecture/écriture unitaires
  + 12 pour les opérations lecture/écriture multiples
  + 17 pour des instructions diverses (¨WAIT, NOP,…)

Arithmétique (addition/soustraction/saturation)

Addition	Soustraction
ADD	SUB
C = A+B	C = A-B
ADC	SBC
C = A+B+ fanion c	C = A-B+ fanion c
	RSB
	C = -A+B

Saturation	
USAT	SSAT
C = unsigned satu(A)	C = signed satu(A)

- Lecture de la documentation :

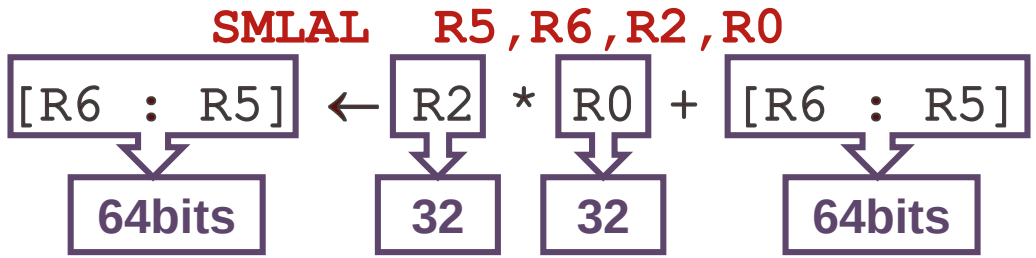
ADD{S}<c> {<Rd>,} <Rn>, #<const>

- {} : facultatif
- Suffixe S :
  - on demande la modification des fanions
- Suffixe <c> :
  - possibilité de conditionner l'instruction
  - **à n'utiliser que pour les branchements**
- 3 opérandes dont le premier est facultatif
  - <Rd> et <Rn> : registres généraux
  - #<const> : valeur immédiate

Arithmétique (multiplication/division)

instructions	
MUL	$C_{32} = A_{32} * B_{32}$
MLA	$C_{32} = A_{32} * B_{32} + D_{32}$
MLS	$C_{32} = D_{32} - A_{32} * B_{32}$
UMULL	unsigned : $C_{64} = A_{32} * B_{32}$
UMLAL	unsigned : $C_{64} = A_{32} * B_{32} + D_{64}$
SMULL	signed : $C_{64} = A_{32} * B_{32}$
SMLAL	signed : $C_{64} = A_{32} * B_{32} + D_{64}$
UDIV	unsigned : $C_{32} = A_{32} / B_{32}$
SIDV	signed : $C_{32} = A_{32} / B_{32}$

- Pas d'opérandes immédiats possibles
- **MUL**, **MLA**, **MLS**, **DIV**, **SDIV**
  - + attention aux débordements
- **UMULL**, **UMLAL**, **SMULL**, **SMLAL**
  - + instructions qui prennent 4 opérandes
  - + pseudo registre 64 bits
- Exemple :



Logique

Opérateur classique		Manipulation de bits		Décalage – Rotation	
ET	AND	Effacement de bits	BEC	Décalage ari. à droite	ASR
OU	ORR	Recopie de bits	BFI	Décalage à gauche	LSL
OU complémenté	ORN	Effacement de bits par masque ET	BIC	Décalage à droite	LSR
OU exclusif	EOR	Nombre de bits à 0	CLZ	Rotation à droite	ROR
Complément à 2	NEG	Transposition bits	RBIT	Rotation à droite	RRX
Complément à 1	MVN	Inversion pf-PF (octet)	REV		
		Inversion pf- PF (½ mots)	REV16		
		Inversion signée (½ mots)	REVSH		

Calcule évidement en complément à 2 mais l'inversion de bit existe

Microcontrôleur ⇒ beaucoup de manipulations au niveau du bit

Promotion

Promotion	
Promotion signée sur 32 bits d'un champs de bits	SBFX
Promotion signée sur 32 bits d'un octet	SXTB
Promotion signée sur 32 bits d'un ½ mot	SXTH
Promotion non signée sur 32 bits d'un champs de bits	UBFX
Promotion non signée sur 32 bits d'un octet	UXTB
Promotion non signée sur 32 bits d'un ½ mot	UXTH

- L'octet 1000 1100 représente :
  - 140 (non signé)
  - ou
  - -116 (signé)
  -
- Codée sur 16 bits cette valeur est :
  - 0000 0000 1000 1100
  - ou
  - 1111 1111 1000 1100

Transfert interne

Transfert	
Charger une adresse située en zone CODE	ADR
Affecte un registre général avec une valeur	MOV
Affecte le pf d'un registre général avec un e valeur	MOVT
Affectation d'un registre spécial	MRS
Ecriture d'un registre spécial	MRS

- Registre spéciaux :
  - le Cortex possède deux modes de fonctionnent (Thread et Handler)
  - permet de gérer certains privilèges
  - l'accès au régime spéciaux et donc l'utilisation de ces fonctions est lié à ces privilèges
- Pose la question des systèmes d'exploitation (O.S.)
- On travaillera en *bare metal* donc sans O.S.

Test

Test	
Addition **sans affectation**	CMN
Soustraction **sans affectation**	CMP
Ou exclusif **sans affectation**	TEQ
ET logique **sans affectation**	TST

- Les fanions sont les pivots de toutes les structures algorithmiques de base
  - peuvent être affectés avec le suffixe S
  - peuvent être affectés avec ces 4 instructions

Exemple :

Les bits de rang 7 et 3 du registre R3 sont ils à 0 ?

```
TST R3,#0x0088
```

```
R3    : 0000 … 0000 0000 1100
  and
0x88  : 0000 … 0000 1000 1000
--------------------------------
        0000 … 0000 0000 1000
```

- Le fanion Z donne l'information souhaitée
- Le résultat est non nul ⇒ le fanion Z vaut 0

⇒Conclusion du test : tous les bits testés ne sont pas à 0

Les sauts : la base

Saut	
Simple	B
Avec lien	BL
Avec lien et par registre	BLX
Simple par registre	BX
Conditionnée sur la nullité d'un registre	CBZ, CBNZ
Si…alors	IT
Table de saut	TBB, TBH

- Saut : affectation du registre PC avec l'adresse donnée en opérande
- L'adresse est :
  - étiquette du programme (**`B,BL,CBZ`**)
  - contenu d'un registre (**`BX,BLX`**)
- Les instructions **`IT`**, **`TBB`** et **`TBH`** sont très particulières à ARM
  - permettent de mettre en place (de façon limitée) certaines structures algo.
  - nous ne les utiliserons jamais pour rester « générique »

Les sauts : le saut sur place !

- A quoi sert ?

```
Finir        B Finir
```

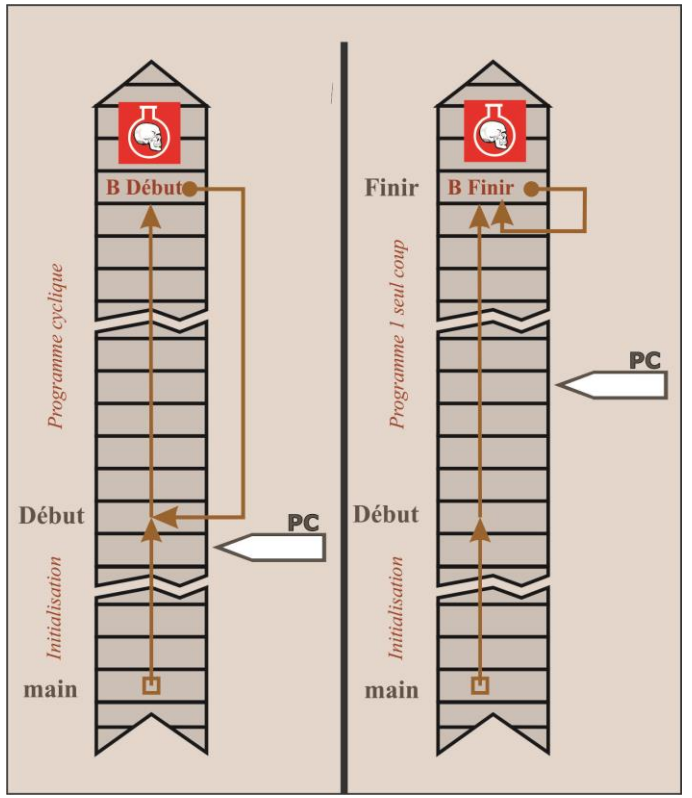
qui pourrait s'écrire

```
B .
```

  - + saute à l'adresse *Finir* (Ox08888436) qui correspond à la place où se trouve l'instruction elle-même
  - + ⇒ saut sur place

- Raison : programme en « bare metal »
  - + pas d'O.S.
  - + à la fin : soit on recommence, soit on saute sur place !

```
0x08000432 EA5F0804  MOVS     r8,r4
    69: Finir   B Finir          ; Boucle infinie
0x08000436 E7FE      B        0x08000436
0x08000438 ABCD      DCW      0xABCD
```

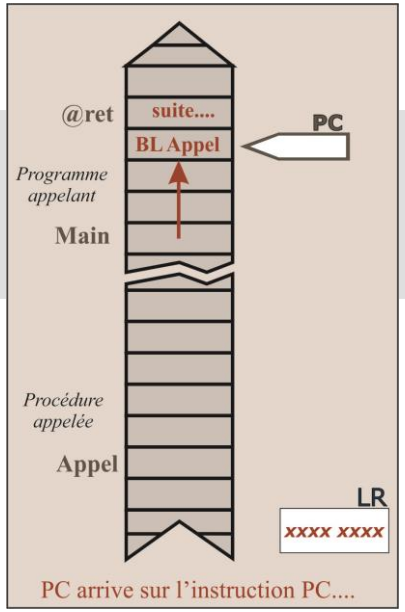


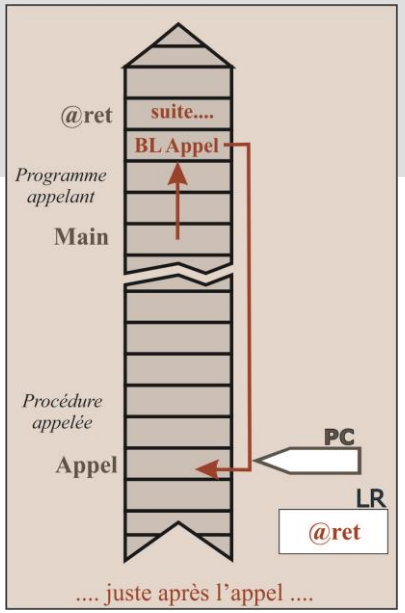
Les sauts : le saut avec lien

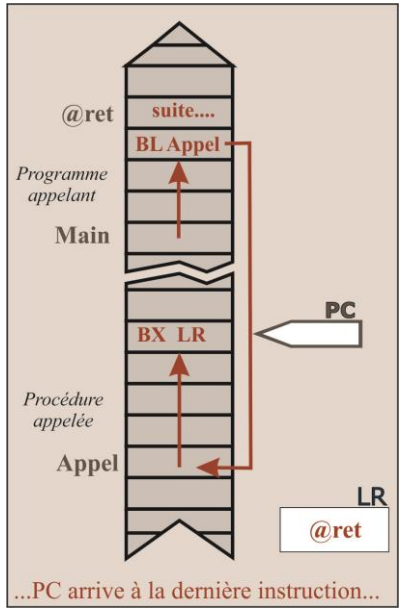
- **BL** et **BLX** = saut avec lien
- Permettent l'appel à une procédure
  - une procédure est repérée par une simple étiquette
  - **BL** affecte **PC** avec l'adresse donnée
  - **avant l'affectation** de **PC** le processeur stocke dans le registre **LR** la valeur de l'adresse suivante
  - dans le procédure l'instruction **BX LR** permet alors de revenir à l'endroit d'où l'appel a été fait

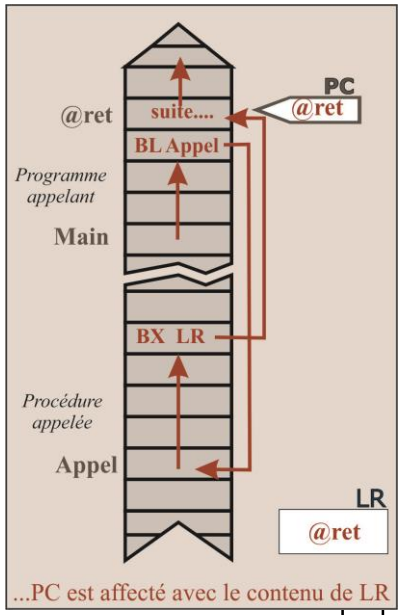
Stockage de l'@ de retour dans un registre

- **Spécifique ARM**
- **Si appel imbriqué ⇒ perte de l'information (LR écrasé)…**









Les sauts : le saut conditionné

- Permet de mettre en place toutes les structures algorithmiques de base
- De nombreuses instructions sont « conditionnables » mais nous ne l'appliquerons qu'à l'instruction **B**
- Condition = ajout d'un suffixe à l'instruction
  - + correspond à une combinaison logique des fanions
  - + si la condition est vérifiée l'instruction est effectuée
  - + si la condition n'est pas vérifiée PC passe à l'instruction suivante

Exemple :

```
Vide    SUBS R1,#1
        BNE Vide
```

Condition		Fanions
Égalité	EQ	Z = 1
Non égalité	NE	Z = 0
Dépassement de capacité	CS	C=1
Pas de dépassement de capacité	CC	C=0
Négatif	MI	N=1
Positif	PL	N=0
Dépassement	VS	V=1
Pas de dépassement	VC	V=0
Plus grand (signé)	HI	C=1 et Z=0
Plus petit ou égal (non signé)	LS	C = 0 OU Z = 1
Plus grand ou égal(signé)	GE	N = V
Plus petit (signé)	LT	N ≠V
Plus grand (signé)	GT	Z = 0 ET N = V
Plus petit ou égal(signé)	LE	Z = 1 OU N ≠V

Load/Store

- Déjà vu :
  - seules instructions pouvant accéder à la mémoire
  - la totalité du registre est affecté
  - ⇒ **promotion**
- Accès également multiples
  - pour la gestion des *floats*
  - pour l'accès à des structures de pile
  - instructions complexes nécessitant plusieurs cycles d'horloge.
  - Attention : les {} remplace les [] pour donner la liste des registres

```
Exemple :
        STM R1, {R2,R3,R5}
        LDM R1,{R7-R9}
```

	Lecture	Ecriture
Mot (32 bits)	LDR	STR
1/2 mot signé (16 bits)	LDRSH	STRH
1/2 mot n on signé (16 bits)	LDRH	STRH
Octet signé (8 bits)	LDRSB	STRB
Octet non signé (8 bits)	LRRB	STRB

Double	LDRD	STRD
Multiple @ ascendantes	LDM	STM
Multiple @ descendantes	LDMDB	STMDB

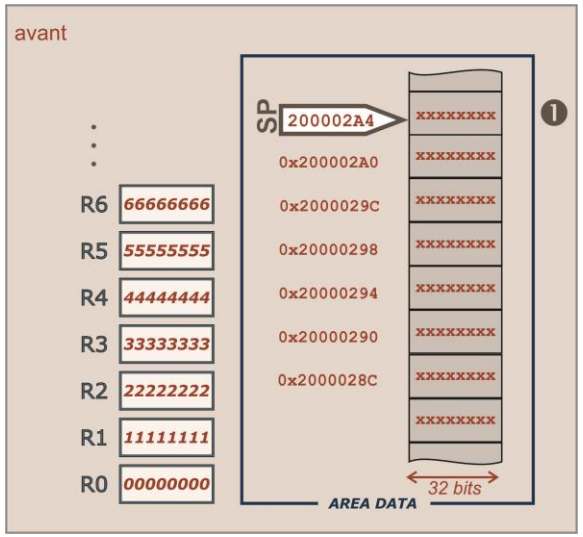
Load/Store spéciales : PUSH et POP

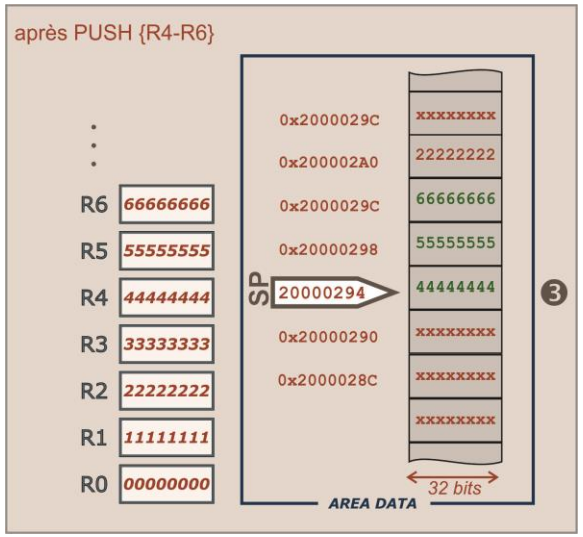
- Instructions de R/W sur la pile système
- Pile :
  - structure LIFO (Last In First Out)
  - besoin : sauvegarder, procédure, interruption
  - une application doit toujours avoir une pile « valide »
  - pointée par le registre dédié SP
- R/W uniquement le contenu de registres

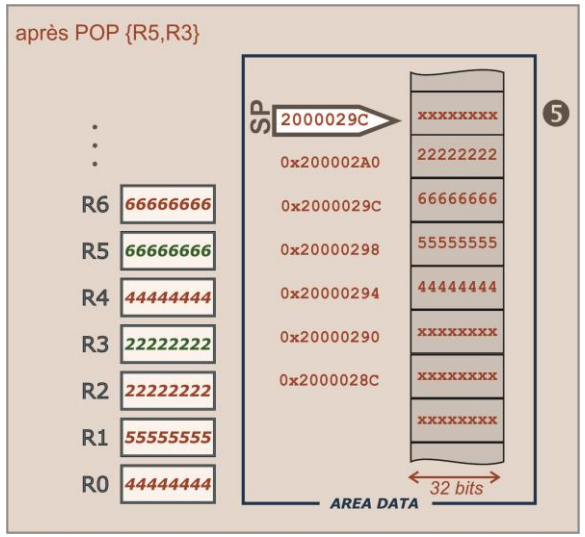
- Lecture simple:
  - PUSH {Rx} *({} obligatoire!*)
  - Décrémente SP puis écrit sur la pile
  - Équivaut donc à STR Rx,[PC,#-4] !
- Ecriture simple
  - POP {Rx}
  - lit sur la pile sur la pile puis incrémente SP
  - Équivaut donc à LDR Rx,[PC,] #4
- R/W multiples
  - {…} liste les registres :
    - push {R1,R4,R6} écrit depuis R1, R4 et R6
    - pop [R5,R7-R11} lit vers R5, R7,R8,R9 et R10

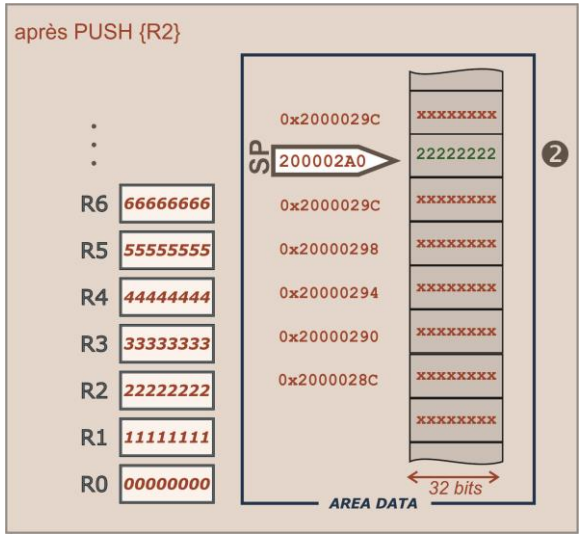
Exemple (discutable…)

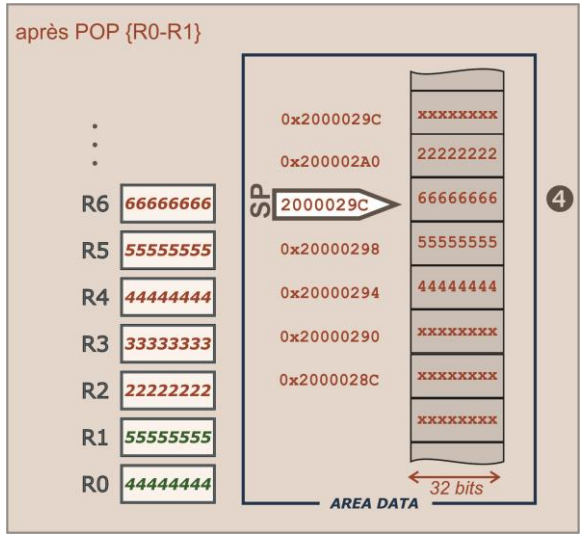
❶
PUSH {R0}
❷
PUSH {R4-R6}
❸
POP {R0-R1}
❹
POP {R5,R3}
❺











A la fin :

- + **OK car pas de dérive de pile**
- + **Discutable : « salade » de registres**

Les instructions « système »

- Instructions spécifiques et « dangereuses »
- Modifie le comportement du processeur
  - **ISB** : vide le pipeline
  - **CPS** : modifie l'état du Cortex
  - **…**
- Appel à des ressources extérieures
  - Unité de débug, coprocesseur,..)
  - **MCR** : écriture dans un registre du coproc.
  - **…**
- Echange d'informations
  - **SEV** : envoi de signal (mutilprocesseur)
  - **…**

Utilisation avancé du Cortex…

Inutile pour la suite du travail en TP

Quizz : composer la valeur de l'octet « solution »

- 2^7 : Une structure Load/Store veut dire que les instructions lisent et écrivent directement en mémoire
- 2^6 : Une «étiquette » est une adresse
- 2^5 : Le processeur peut multiplier des flottants
- 2^4 : Les sauts sont des appels à des procédures
- 2^3 : La promotion consiste à transformer un nombre signé en nombre non signé
- 2^2 : AND R3,#0x0000 lève le fanion z
- 2^1 : L'addition des nombres signés n'utilise pas d'instruction particulière parce que le Cortex utilise le complément à deux
- 2^0 : Le registre LR est automatiquement modifié par les instructions de saut