

Micropprocesseurs

Mouhcine CHAMI
D111
chami@inpt.ac.ma

Objectif et plan

Comprendre le fonctionnement de base d'un microprocesseur

- Introduction
- Principes de fonctionnement CPU
- Les architectures plus performantes
- Les mémoires
- Structure d'interconnexion : les bus
- Architecture de l'ARM v7
- Jeu d'instructions ARM

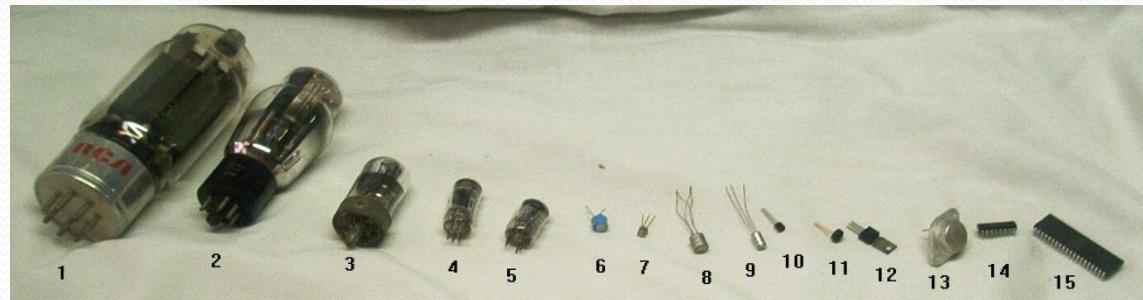
Définitions

- Processeur : Puce capable d'exécuter des instructions.
- Instructions machines : Un algorithme est décomposé en instruction élémentaires comme l'opération de l'addition
- Jeu d'instructions : Un ensemble d'instructions que peut exécuter le processeur
- Un Programme : Une séquence d'instructions qui forme un programme
- Les instructions sont stockées en mémoire sous forme de 0 et 1 (le binaire) : Langage machine
- Langage assembleur est le langage bas niveau mais il demande toujours un compilateur pour le transformer en langage machine
- Langage haut niveau : sous forme de texte avec une syntaxe spécifique, des compilateurs permettent la traduction d'un niveau vers un niveau inférieur, type C,

Évolutions des microprocesseurs

- **Technologique (niveau électronique) :**

- Tube à vide (1904)
- Transistor en semi-conducteur (du micro au nano)
- Nanotechnologie
- Loi de Moore



- **Au niveau de la mémoire :**

- 1 Ko jusqu'à quelques Go

- **Au niveau de l'architecture :**

- Du modèle Von Neumann
- Architecture plus parallèles (pipeline, superscalaire, multi cœur)

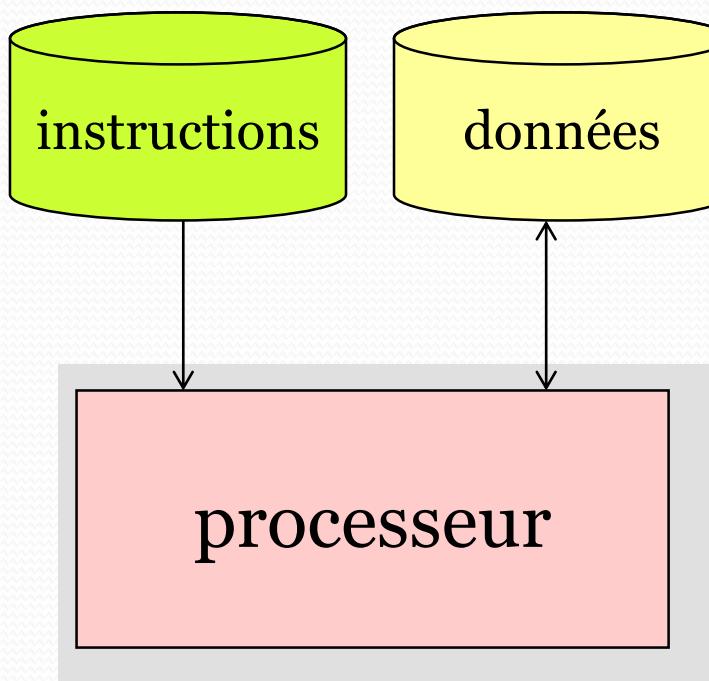
Performances des microprocesseurs

- **La course aux performances (MIPS, MOPS, FLOPS)**
 - Plus les machines sont performantes et plus les applications sont gourmandes
- **Les applications :**
 - Traitement d'images, visioconférences, jeux 3D, réalité virtuelle, réalité augmentée
 - Modélisation et simulation en climatologie, conception des circuits intégrés
- Pour accroître les performances, on se base sur l'évolution des technologies et aussi les architectures : parallélismes interne (au sein du processeur) ou externe (multi processeur)
- Loi de Moore (Nombre de transistors double tous les 24 mois)

Histoire

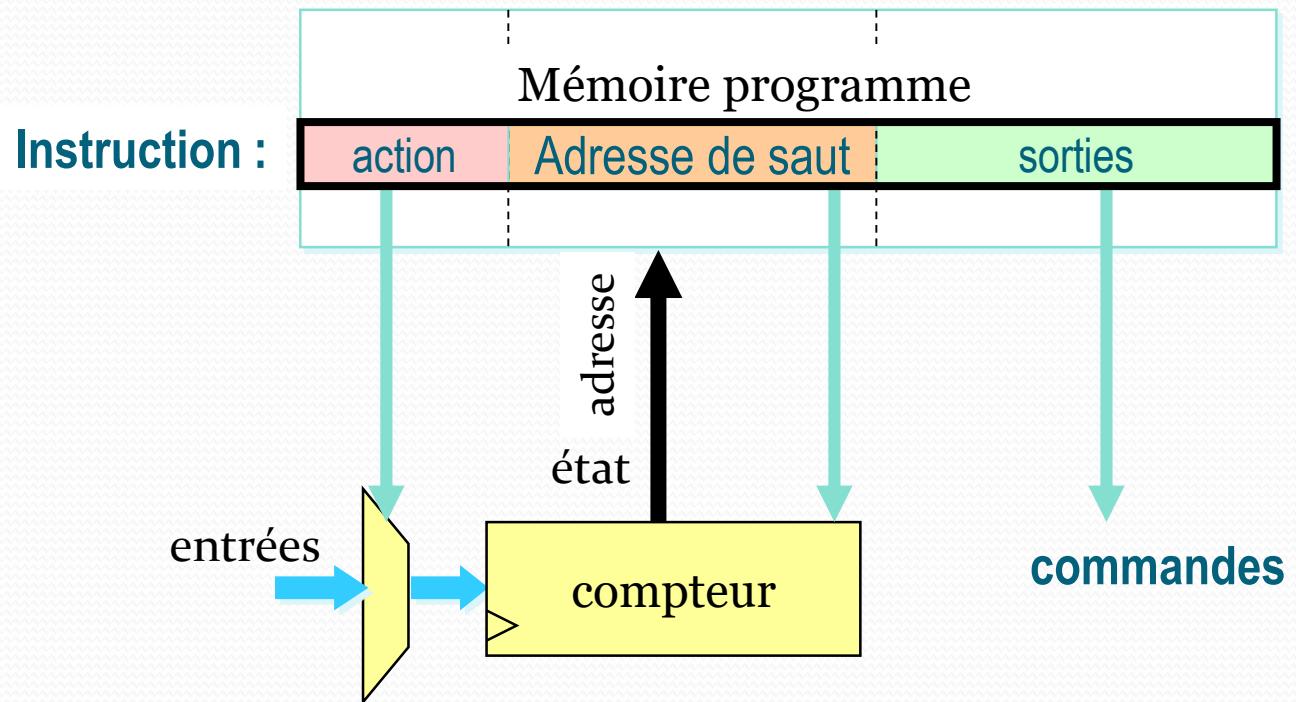
- 1943 : ENIAC (*Electronic Numerical Integrator And Computer*) reprogrammable, décimal et utilise des tubes à vides, conçus à des buts militaires, un ensemble de 20 calculateurs
 - **30 tonnes, 18000 tubes à vide et 70000 résistances**
 - **Consommation : 140 kW/h**
 - **Réalise 5000 additions/seconde/calculateur**
- **IAS et Architecture de Von Neumann (1946 -1952)**, Institut of Advanced Studies :
 - Mémoire principale : données et instructions
 - Unité arithmétique et logique
 - Unité de contrôle (interprétation d'instruction)
 - Dispositif d'entrée/sortie
 - Encore d'actualité...

Traitement logiciel ...



Intérêts :
Flexibilité
Temps de développement

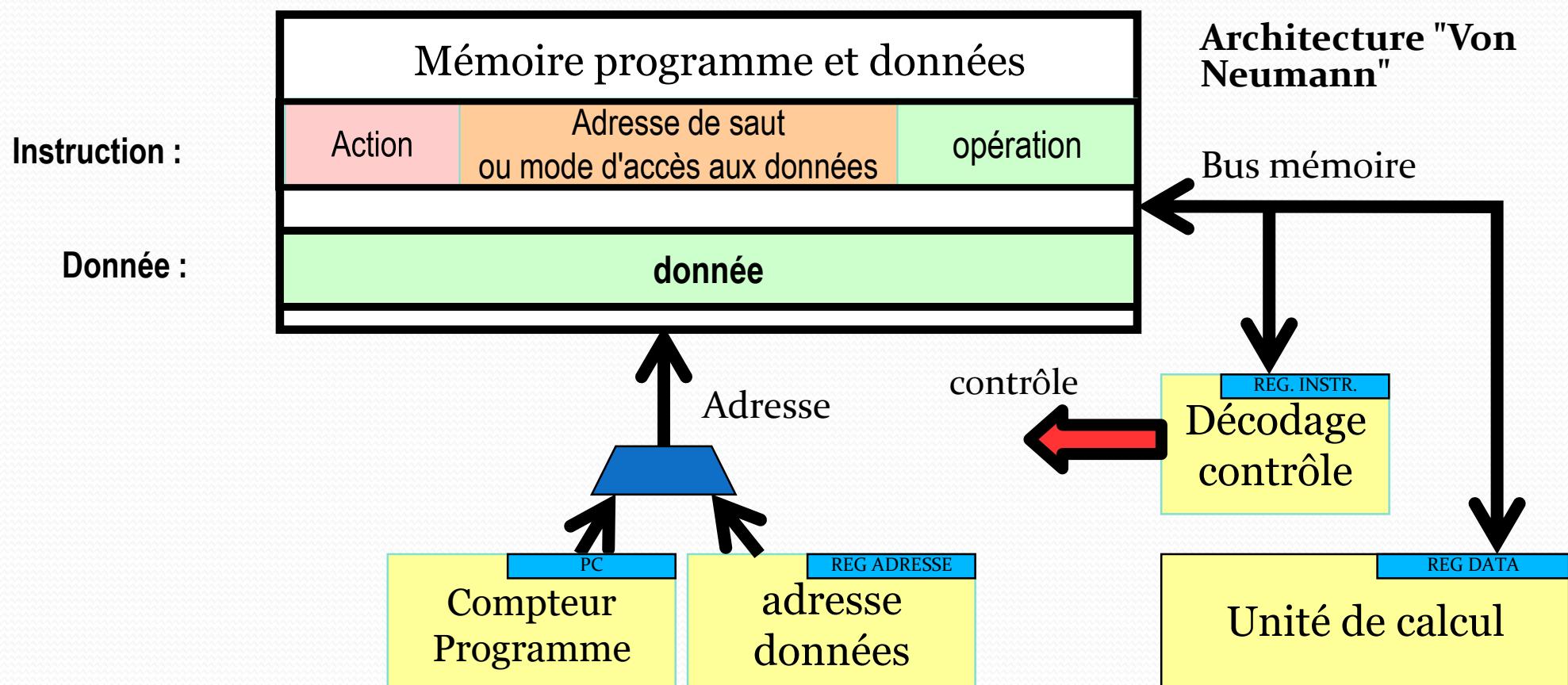
Automate programmable



Processeur

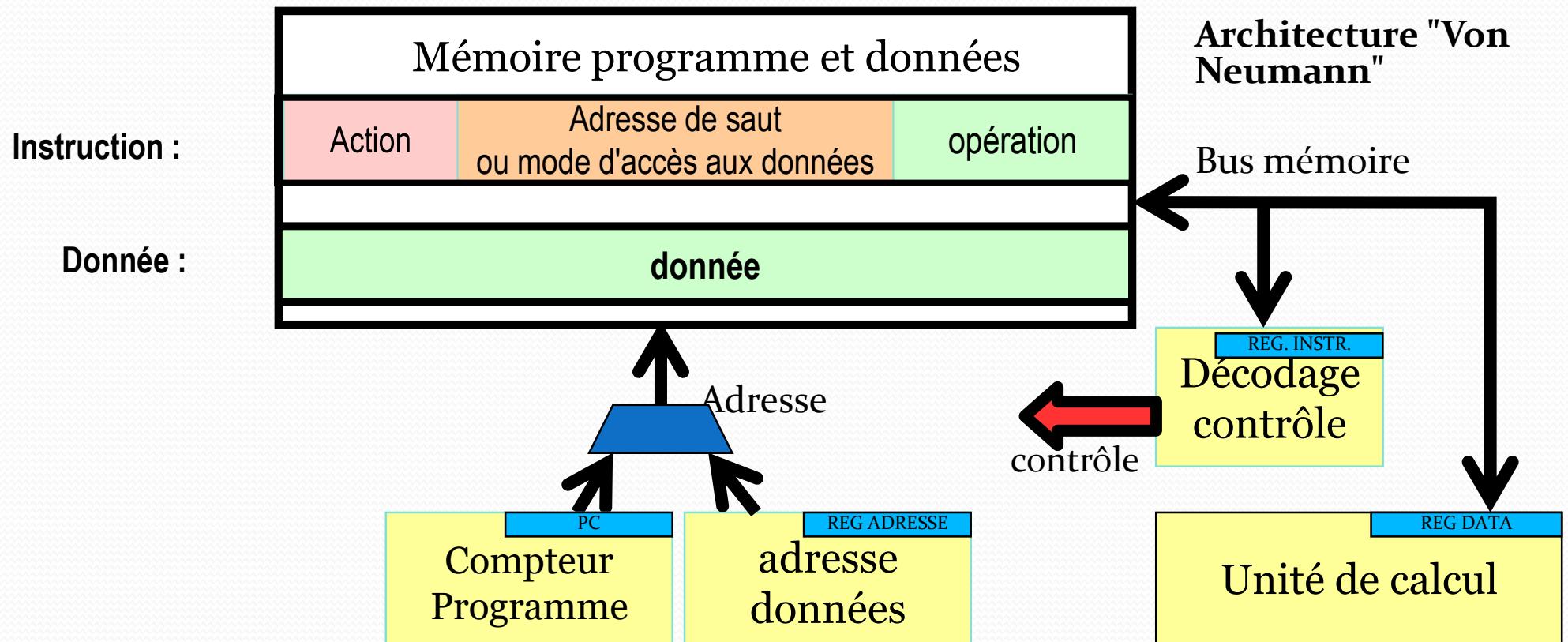
Architecture "Von Neumann"

Dédié au traitement de données



Processeur

Architecture "Von Neumann"



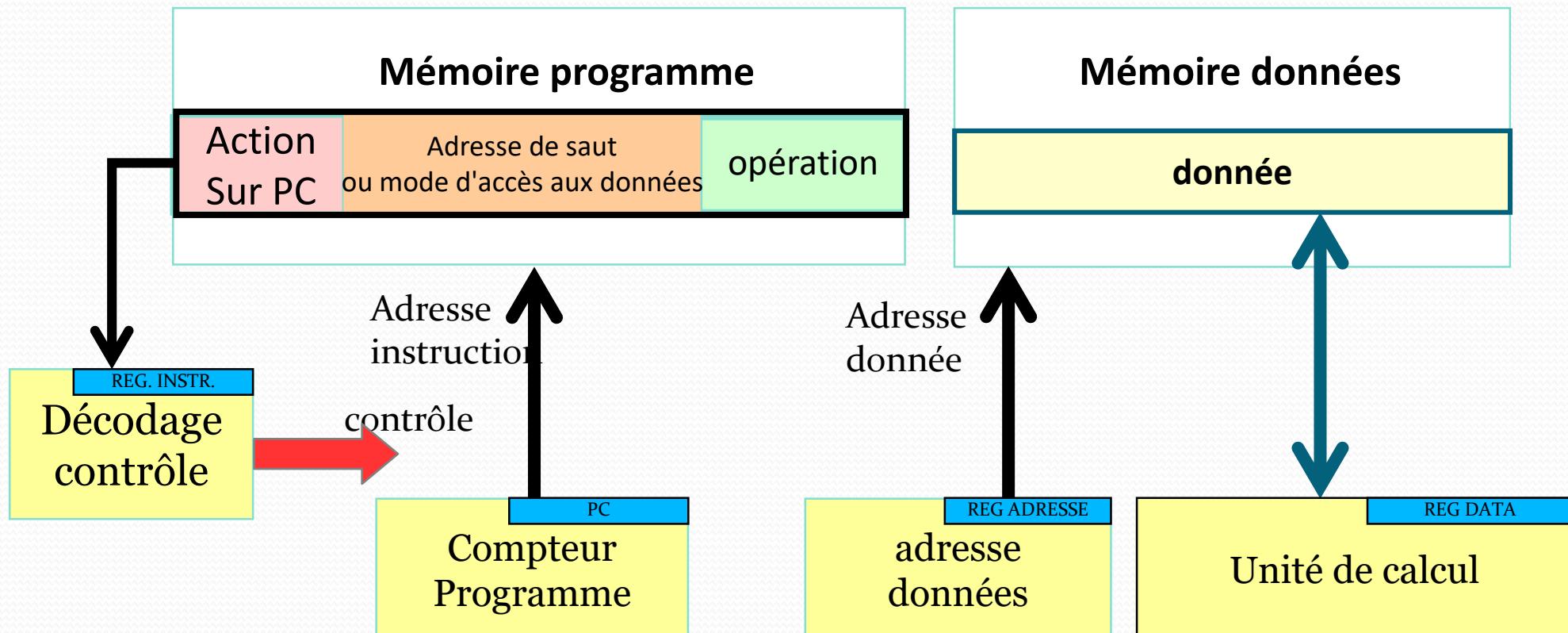
Garder l'adresse de l'instruction courante dans le « Program Counter » (PC)

1. Lire le contenu de la cellule adressée par le PC
2. $PC = PC + 1$
3. Interpréter contenu comme une instruction, et l'exécuter
4. Recommencer

Processeur

Architecture "Harvard"

Les mémoires programme et données sont séparées => accès et débits x 2



Types d'instructions

Traitement : Opérations arithmétiques et logiques

Transferts des données avec la mémoire: Load, store

OP	OP ₁	OP ₂
----	-----------------	-----------------

Contrôle :

- Branchements
- Branchements aux sous programmes
- Sauvegarde automatiquement de l'adresse de retour en **PILE**

LD	ADRESSE
----	---------

Système: Interruptions logicielles : Appel de fonctions du système d'exploitation

BR	ADRESSE
----	---------

Coprocesseur : Instructions spécifiques à un opérateur extérieur coprocesseur (vidéo et autre)

SWI	
-----	--

COP	N° COP
-----	--------

Exemple d'instruction

Instr 4

1110 0001 1010 0000 0011 0000 0000 0111

Binaire

e1a03007

Hexadécimal 0xE1 10 30 07

Mov r3, r7

Code mnémonique =>
Assembleur

Exemple : X86

```
1 int main(void){  
2     int a,b,c;  
3     a=3;  
4     b=5;  
5     c=a+b;  
6 }
```

```
1 main:  
2     pushl %ebp  
3     movl %esp,%ebp  
4     subl $12,%esp  
5     movl $3,-4(%ebp)  
6     movl $5,-8(%ebp)  
7     movl -4(%ebp),%eax  
8     addl -8(%ebp),%eax  
9     movl %eax,-12(%ebp)  
10    leave  
11    ret
```

Exemple : ARM

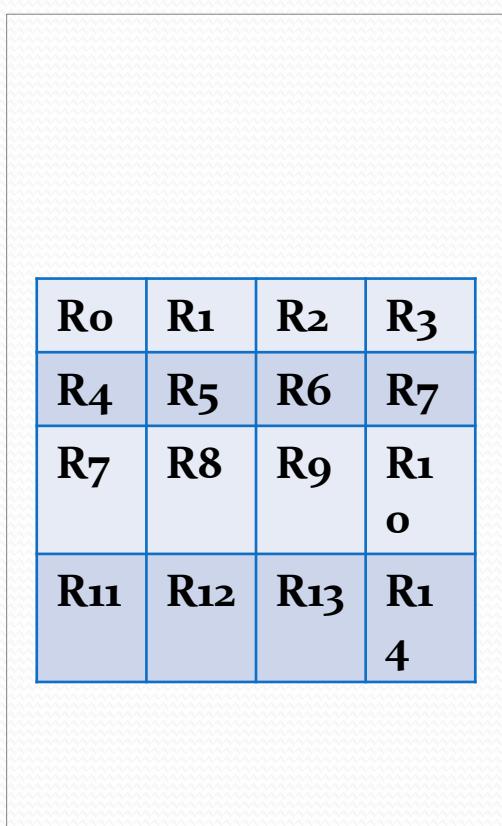
```
1 int main(void){  
2     int a,b,c;  
3     a=3;  
4     b=5;  
5     c=a+b;  
6 }
```

```
1 main:  
2     str    fp, [sp, #-4]!  
3     add    fp, sp, #0  
4     sub    sp, sp, #20  
5     mov    r3, #3  
6     str    r3, [fp, #-8]  
7     mov    r3, #5  
8     str    r3, [fp, #-12]  
9     ldr    r2, [fp, #-8]  
10    ldr    r3, [fp, #-12]  
11    add    r3, r2, r3  
12    str    r3, [fp, #-16]  
13    mov    r3, #0  
14    mov    r0, r3  
15    add    sp, fp, #0  
16    ldr    fp, [sp], #4  
17    bx    lr
```

Usage de la mémoire

- **Jeux d'instructions mémoire-mémoire**
 - Add adresseX, adresseY
 - Add adresseX, adresseY, adresseZ
- **Beaucoup de trafic mémoire qui est lent**
- **L'adressage des cases mémoire prend beaucoup de place dans l'instruction**
- **Mémoire, registreSolution : Jeux d'instruction registre-registre**
 - Mov Rx, adresseX
 - Mov adresseX, Rx
 - Add RX adresse1
 - Add RX, RY, RZ
 - ...

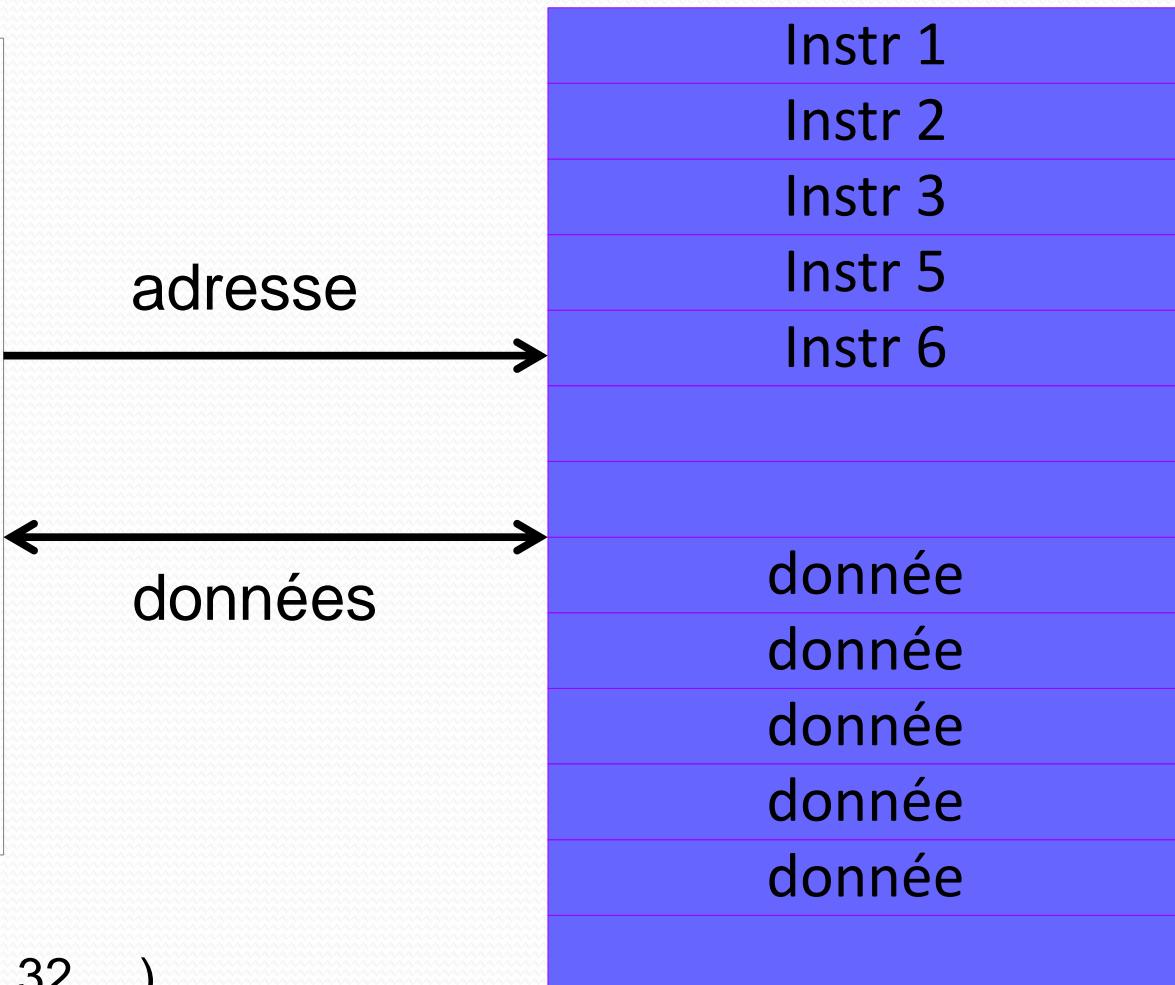
Les registres



Non adressables

Peu nombreux (4, 16, 32 ...)

Non typés



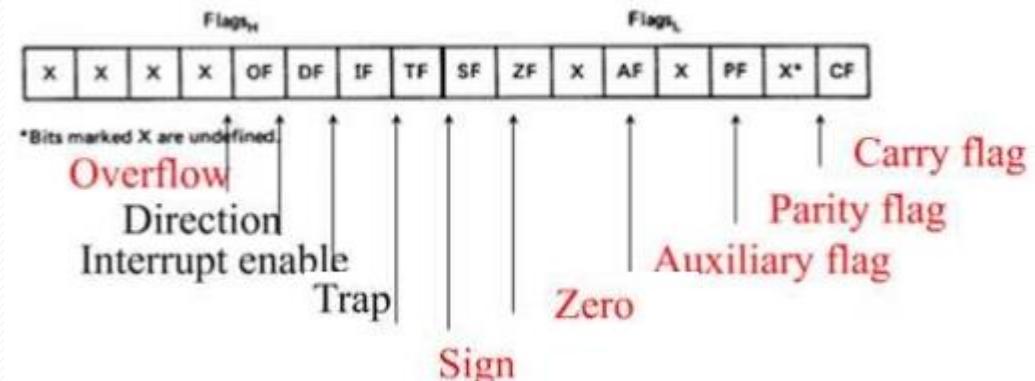
Les registres

Certains registres sont visibles au programmeur/compilateur.

Certains registres sont gérés automatiquement par le processeur. Lesquels?

Solution :

- PC (compteur du programme *program counter*)
- SP (pointeur de la pile *stack pointer*)
- Registres internes / non accessibles (e.g. IR)
- Flags



Les registres : exemple ARM v7

R ₀	Utilisable
R ₁	Utilisable
R ₂	Utilisable
R ₃	Utilisable
R ₄	Utilisable
R ₅	Utilisable
R ₆	Utilisable
R ₇	Utilisable
R ₈	Utilisable
R ₉	Utilisable
R ₁₀	Utilisable
R ₁₁	Utilisable
R ₁₂	Utilisable
R ₁₃	SP : stack pointer
R ₁₄	Link registre (adresse de retour)
R ₁₅	PC : Compteur du programme

Modes d'adressage

- Permet d'accéder aux données
- Objectif : souplesse, minimiser le nombre d'instructions

Mode	Accès aux données dans l'instruction
Immédiat	donnée elle même
Absolu	adresse de la donnée
Registre	numéro de registre
Registre Indirect	numéro de registre contenant l'adresse donnée (registre d'adresse)
Indirect	adresse d'une adresse donnée

Usage de la mémoire

- Mode indexé-indirect (pratique pour un tableau)
 - **ldr r2, [r3, r4]** ;**r2 <= MEM[r3+r4]**
- Mode indirect avec post-increment
 - **ldr r1, [r3], #4** ; **r1 <= MEM[r3] après r3 =r3+4**
- Mode relatif au PC (la destination du saut vers « .Label » est calculé relatif au PC, donc à l'adresse de l'instruction même. Ce calcul est fait par l'assembleur)
 - **b .Label**
 - ...
 - **.Label :**
 -

Exemples pour ARM

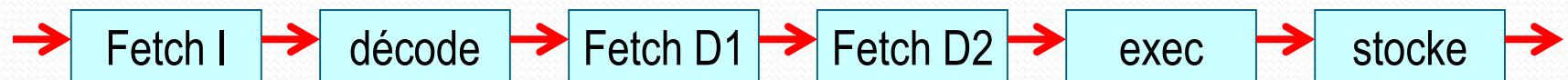
- Mode immédiat (Utilisation d'une valeur numérique)
 - **mov r2, #25**
- Mode registre (Utilisation d'une valeur venant d'un autre registre)
 - **mov r2, r7**
- Mode direct (Utilisation d'une valeur venant de la mémoire en spécifiant son adresse directement)
 - **ldr r2, #8001**
 - **str r2, #8001**
- Mode registre indirect (L'adresse est stockée dans un registre)
 - **ldr r2, [r3]**

Exemples pour ARM (suite)

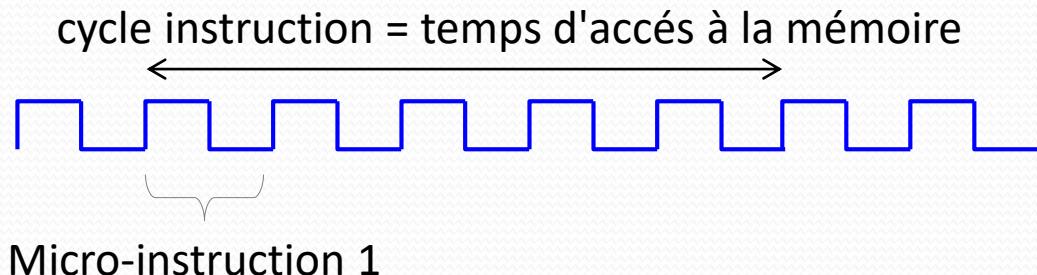
- Mode indexé-indirect (pratique pour un tableau)
 - **ldr r2, [r3, r4]** // $r2 \leq \text{MEM}[r3+r4]$
- Mode indirect avec post-increment
 - **ldr r1, [r3], #4** // $r1 \leq \text{MEM}[r3]$ après $r3 = r3 + 4$
- Mode relatif au PC (la destination du saut vers « .Label » est calculé relatif au PC, donc à l'adresse de l'instruction même. Ce calcul est fait par l'assembleur)
 - **b .Label**
 - ...
 - **.Label :**
 -

Processeur CISC

- L'accès à la mémoire extérieure est relativement lent
- L'exécution d'une instruction nécessite plusieurs étapes



- Objectif : Finir un programme avec le moins de lignes d'assembleur
- Idée : Utiliser le temps d'attente mémoire pour exécuter des instructions compliquées avec des modes d'adressage variés
- Comment : Réaliser un processeur capable de comprendre et d'exécuter une série d'opérations dans une même instruction

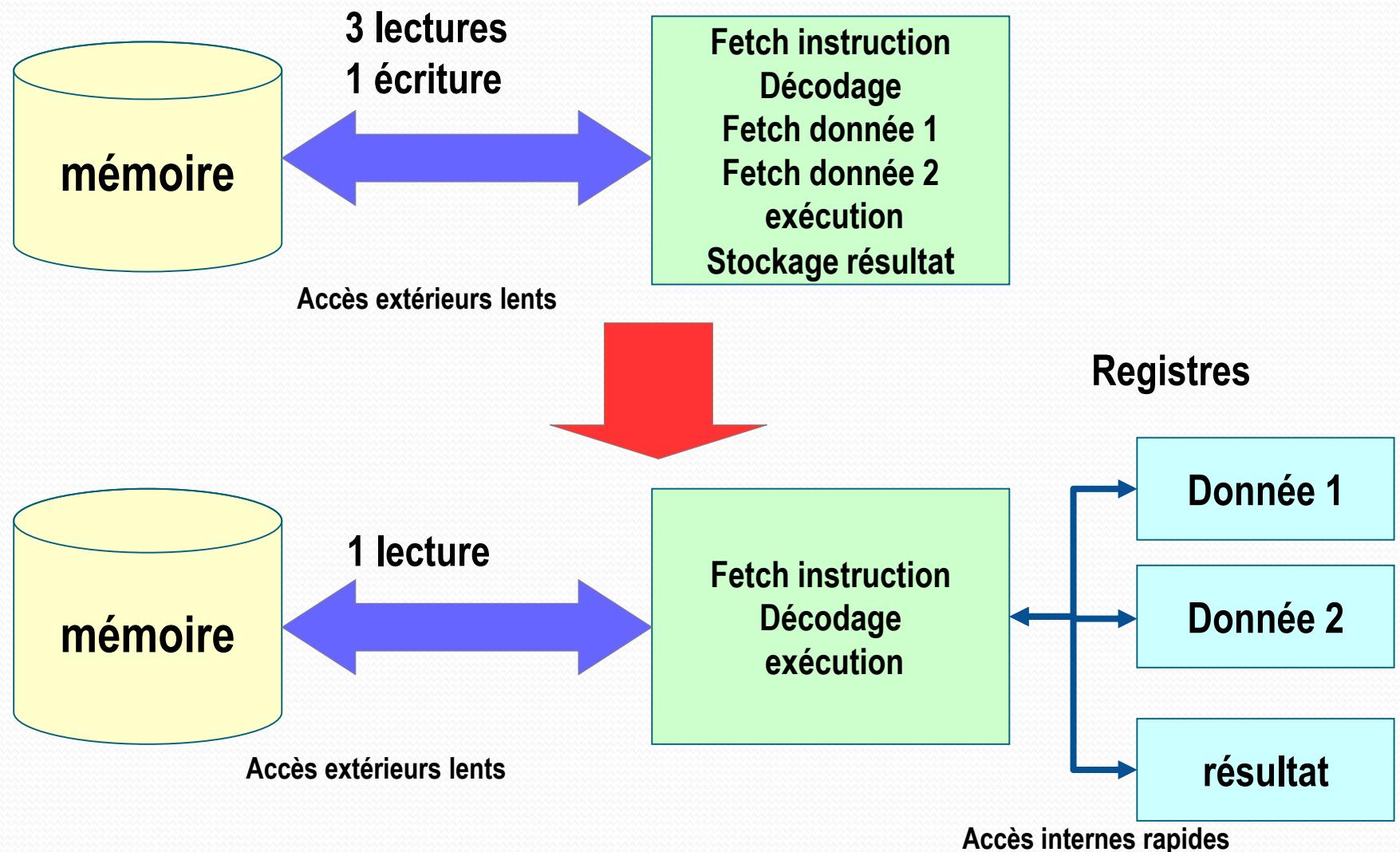


Processeur RISC

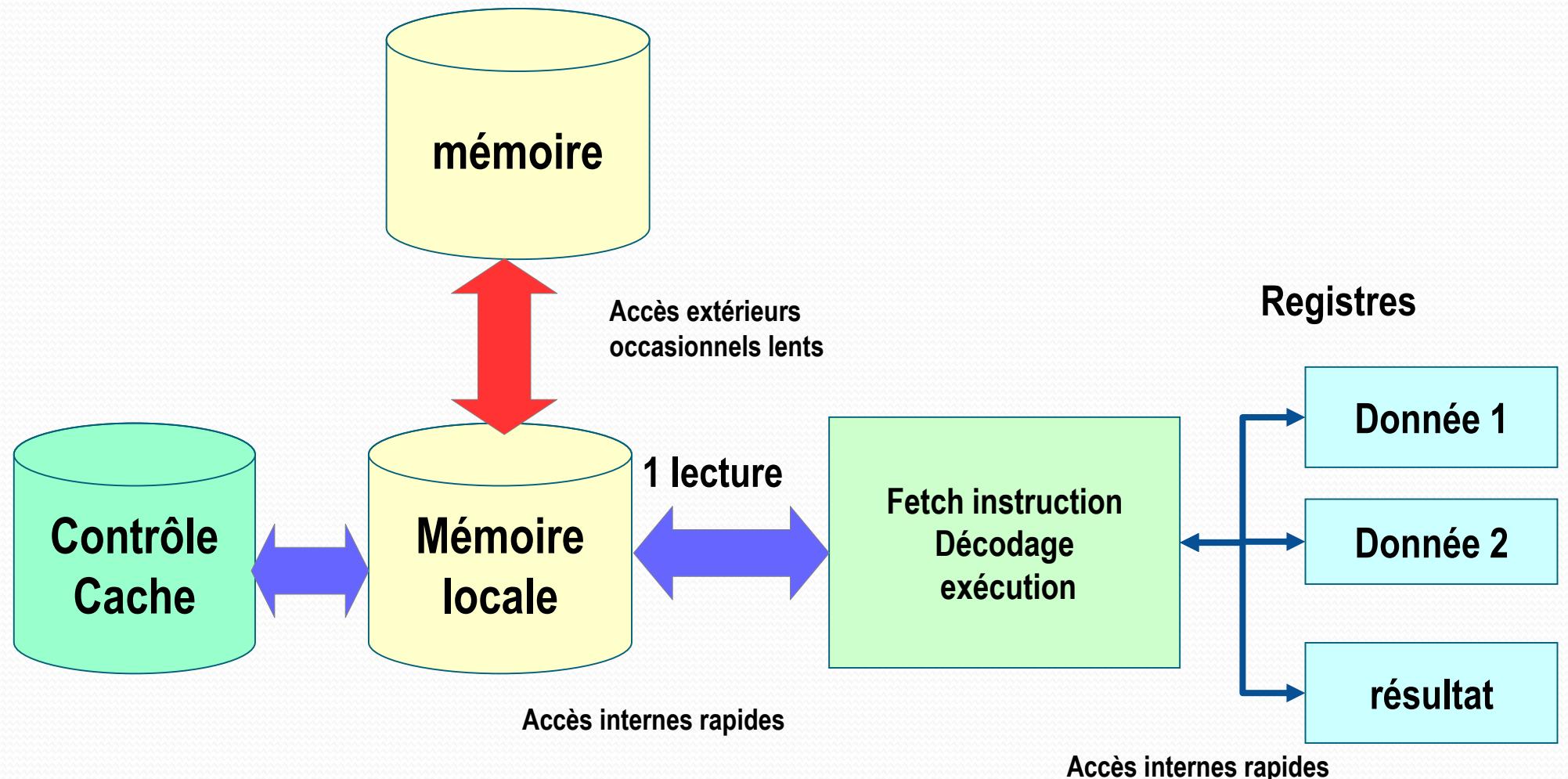
- Utiliser des registres, de la mémoire locale rapide ou « cache » pour éviter d'attendre
- Paralléliser les instructions : techniques pipeline
- Instructions simples au format fixe
- 2 instructions pour l'accès à la mémoire : LOAD STORE

Dans les années 80, cette approche devint possible grâce aux progrès technologiques . Le gain en performance doit tendre vers « une instruction en 1 cycle d'horloge ».

Utilisation de registres locaux



Utilisation de mémoire locale : cache

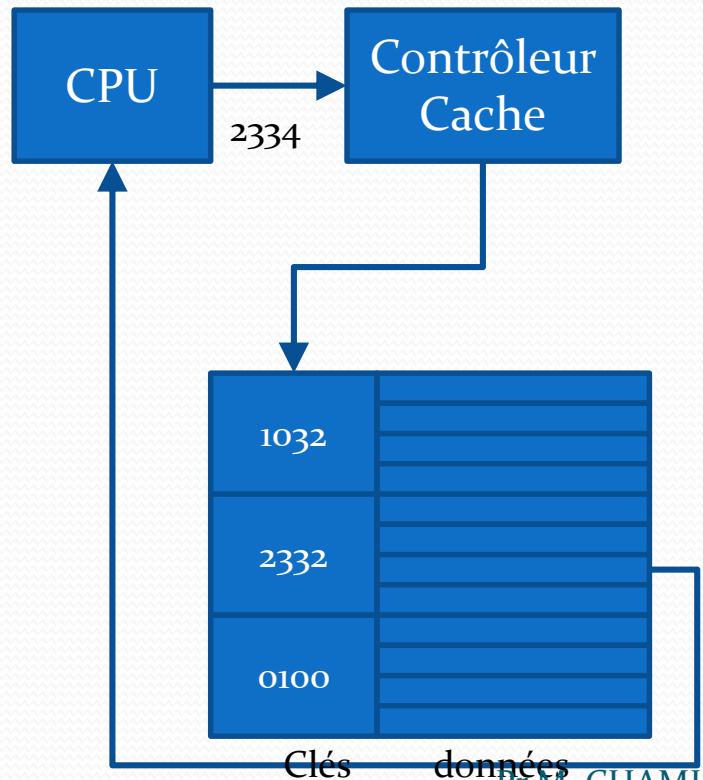


Les mémoire caches

- Typiquement, 90 % du temps d'exécution d'un programme est dépensé dans juste 10 % du code => principe de localité
 - Localité Temporelle
 - Une cellule mémoire référencé a plus de chance d'être référencée encore une autre fois
 - Localité Spatiale
 - Une cellule mémoire voisine a plus de chance d'être référencée (données stockées continûment)
 - Principe de fonctionnement : Lecture d'un mot
 - Si l'information est présente dans le cache on parle de succès (cache hit)
 - L'information n'est pas dans le cache – un échec (cache miss)
 - L'efficacité du cache dépend de son taux de succès

Les mémoire caches

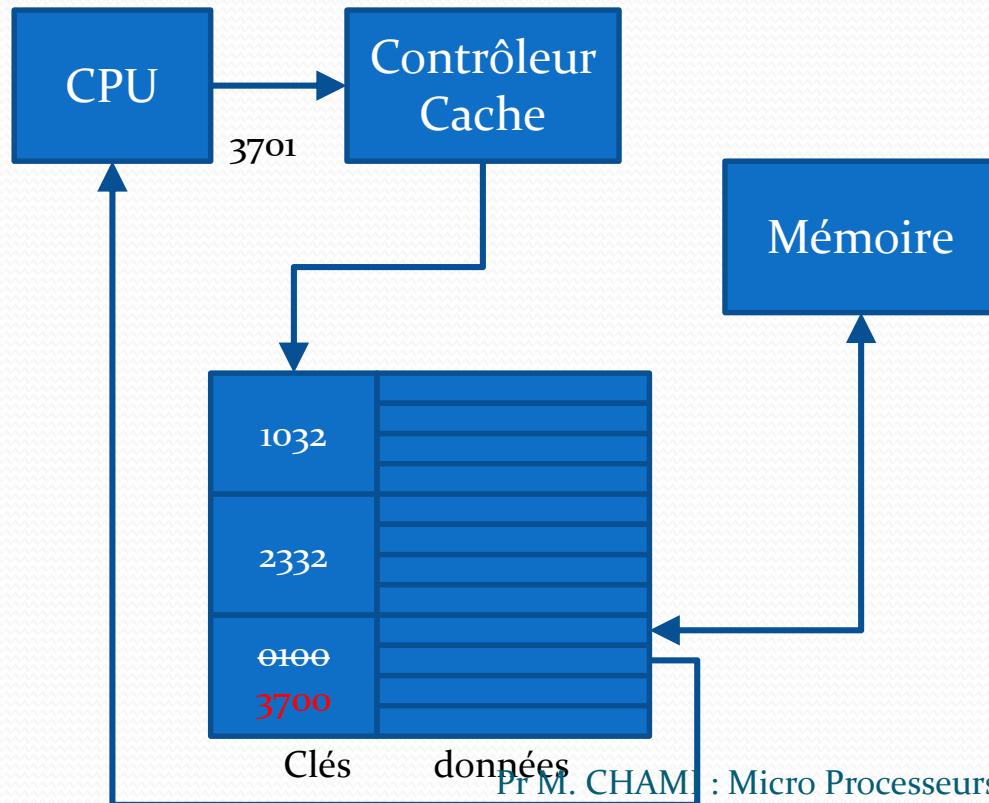
- Organisation et fonctionnement
- Le principe de localité → considérer la mémoire centrale comme une suite de blocs mémoires : lignes de mémoires



1. Toutes les requêtes de l'accès à la mémoire vont au contrôleur cache
2. Le contrôleur vérifie une clé si la ligne demandée est dans le cache
3. Cache hit → un mot du cache est utilisé

Les mémoire caches

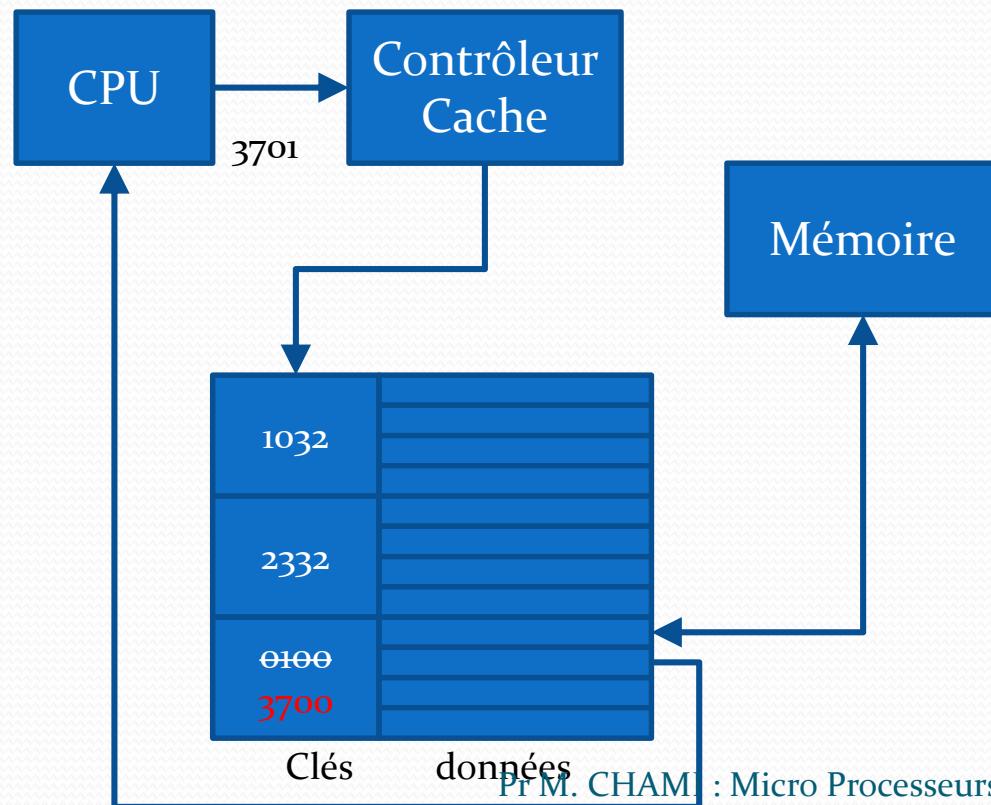
- Organisation et fonctionnement
- Le principe de localité → considérer la mémoire centrale comme une suite de blocs mémoires : lignes de mémoires



1. Toutes les requêtes de l'accès à la mémoire vont au contrôleur cache
2. Le contrôleur vérifie une clé si la ligne demandée est dans le cache
3. Cache miss
4. Le contrôleur choisit la ligne à remplacer
5. Remplace la ligne en cherchant les données de la mémoire
6. Traitement de la ligne

Les mémoire caches

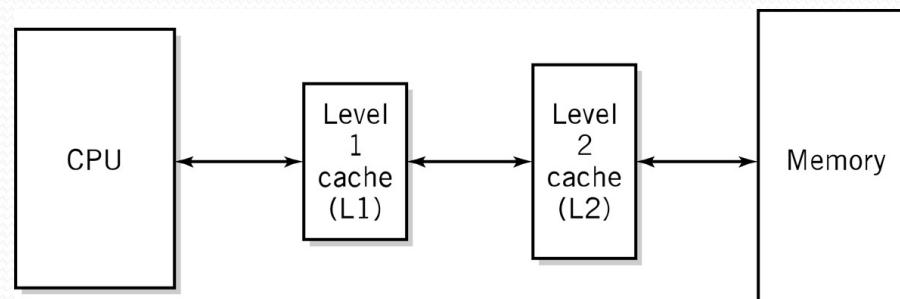
- Organisation et fonctionnement
- Le principe de localité → considérer la mémoire centrale comme une suite de blocs mémoires : lignes de mémoires



1. Toutes les requêtes de l'accès à la mémoire vont au contrôleur cache
2. Le contrôleur vérifie une clé si la ligne demandée est dans le cache
3. Cache miss
4. Le contrôleur choisit la ligne à remplacer
5. Remplace la ligne en cherchant les données de la mémoire
6. Traitement de la ligne

Nombre et localisation des caches

- Hiérarchie de mémoires cache
 - Le premier niveau de mémoire cache, petite et très rapide, est placé dans le processeur (cache de niveau 1)
 - Le deuxième niveau, de capacité plus importante et d'accès également rapide, est mis à l'extérieur du processeur (cache de niveau 2)
 - Le troisième niveau est constitué par la mémoire centrale
 - Pour être utile, le second niveau de cache doit avoir plus de mémoire que le premier
 - Des processeurs performant ont 3 niveaux de cache ex: Intel i7 ($L_1 = 4 \times 64$ Ko $L_2 = 4 \times 256$ Ko $L_3 = 8$ Mo)

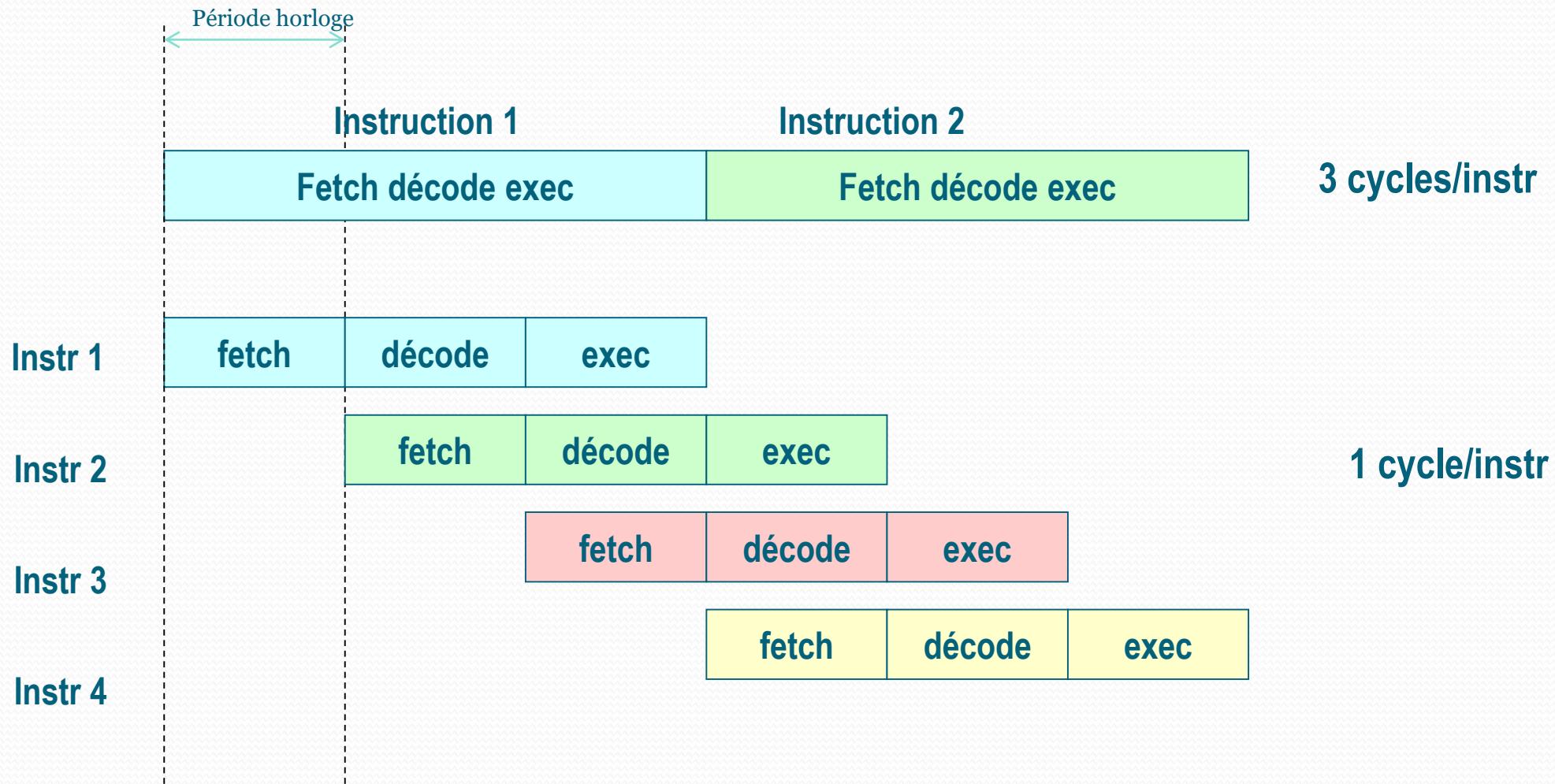


Chercher la performance

Nouvelles architectures :

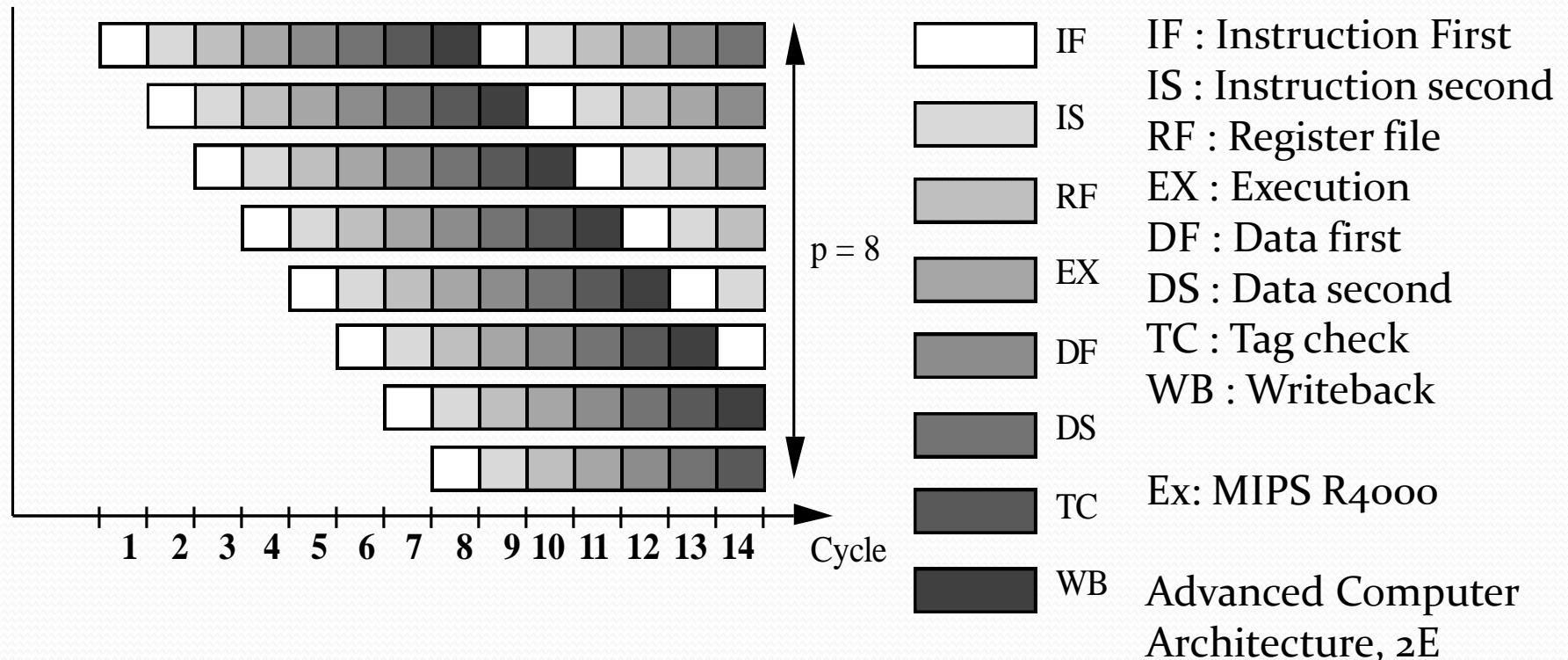
- Super Pipeline
- Super Scalaire
- VLIW
- Multi cœur

Utilisation du pipeline



Architectures Super-pipeline

- Très grandes fréquences d'horloge

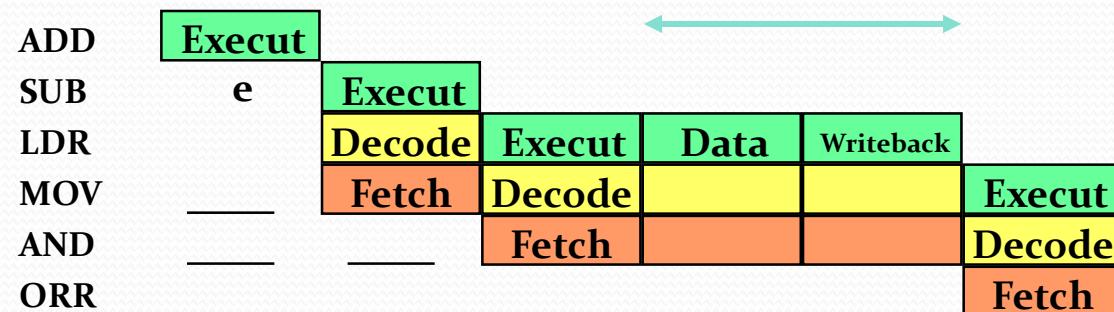


Problème : rupture du pipeline

2 causes principales :

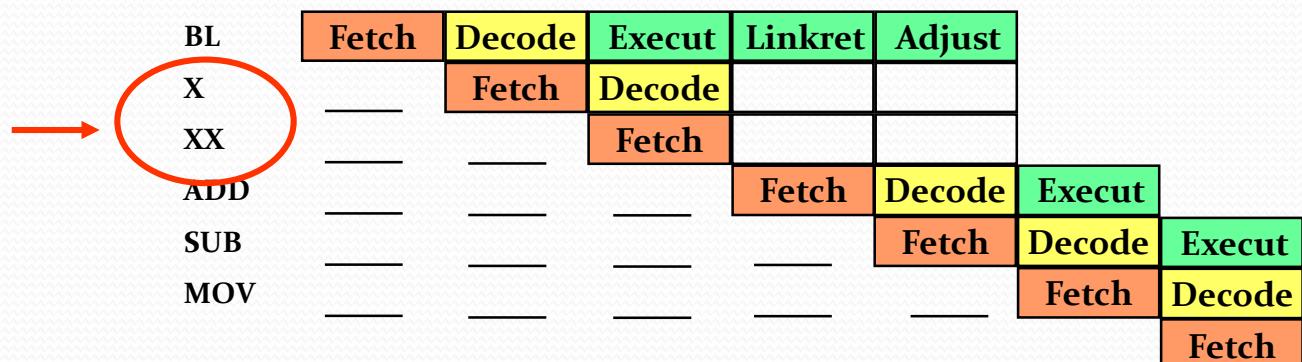
2 cycles perdus

Accès mémoire extérieure



branchements

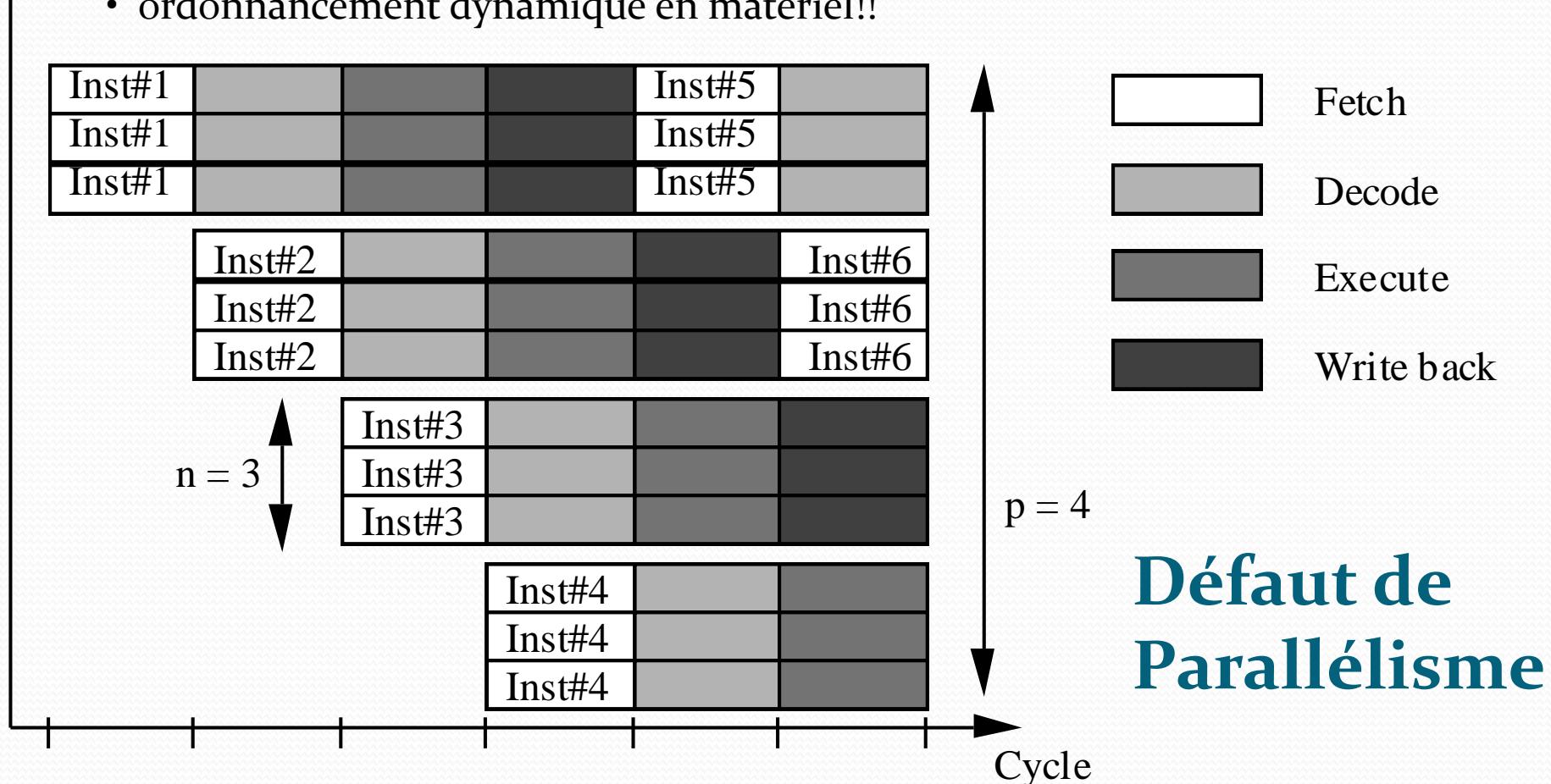
2 instructions perdues



Solutions : branchement retardé
 prédition de branchement

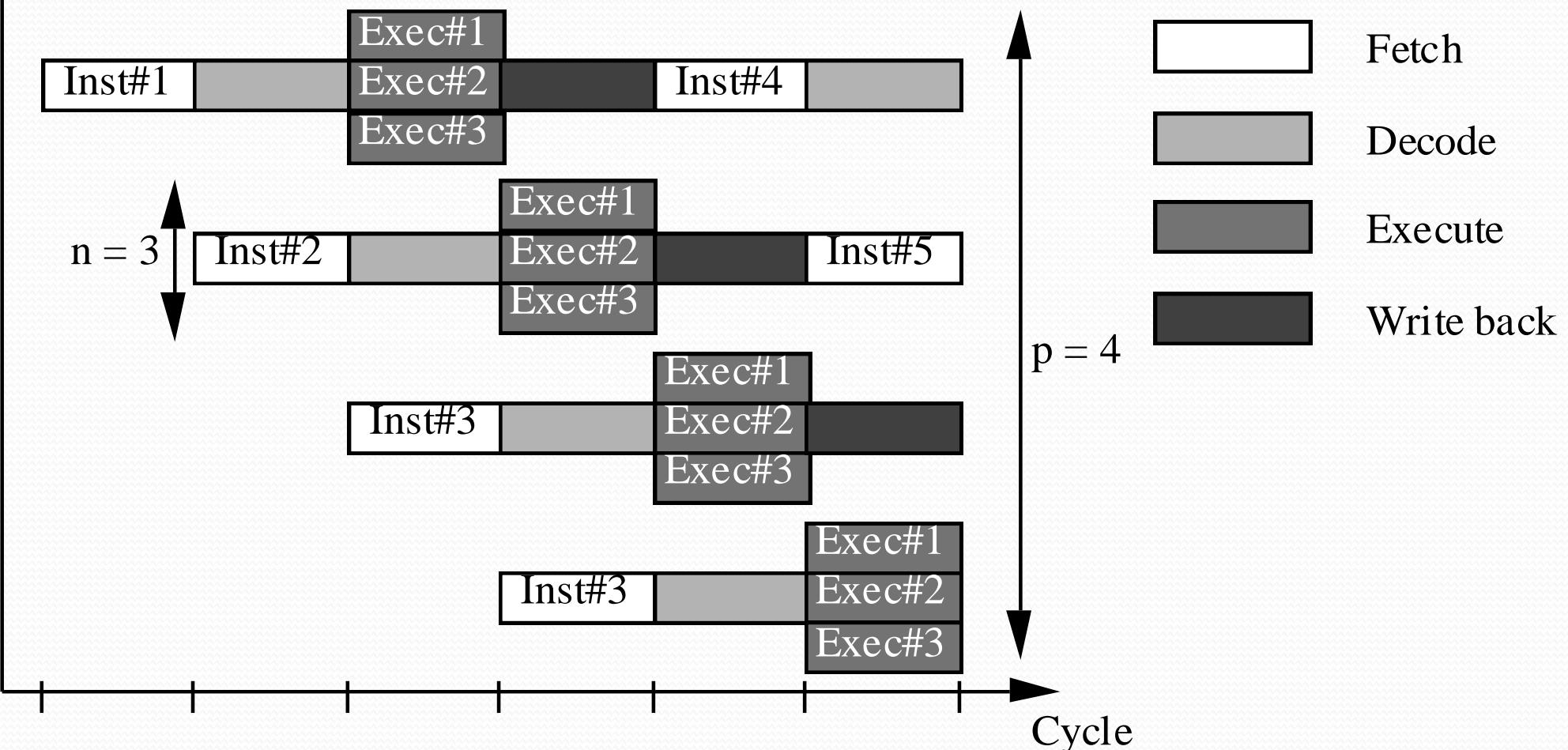
Architectures Super-scalaires

- n unités d'exécution
- n instructions délivrées par cycle
- ordonnancement dynamique en matériel!!

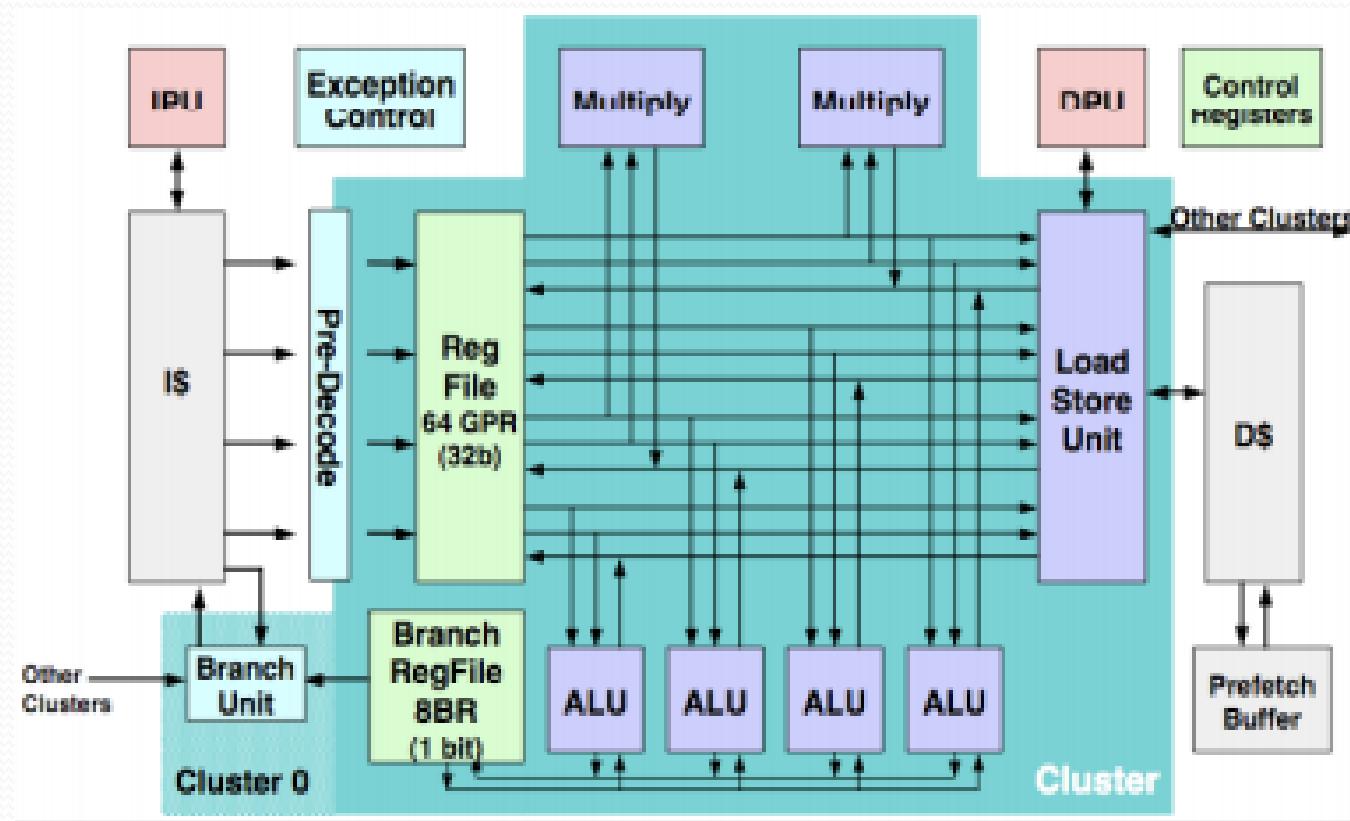


VLIW

- Une seule unité de contrôle, une instruction très longue par cycle.



Exemple ST200

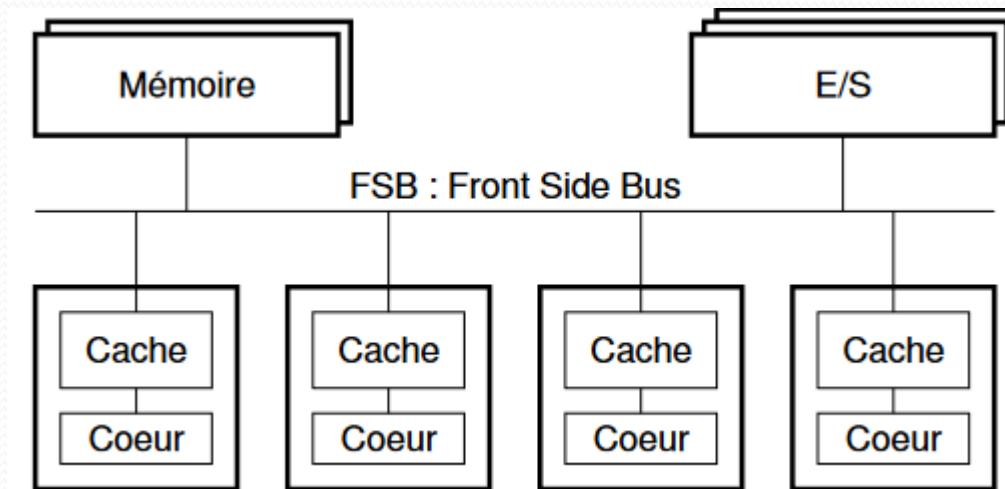


Le gros travail dans le compilateur

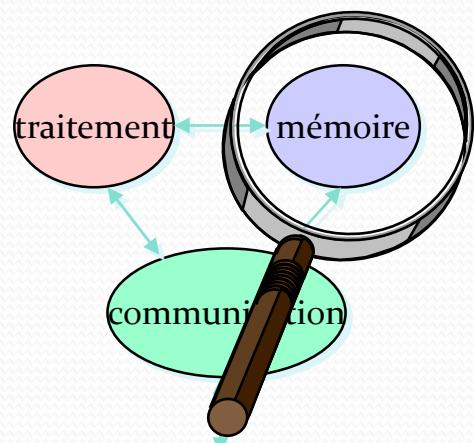
4 ALU
2 multiplicateurs

Architectures Multi-coeurs

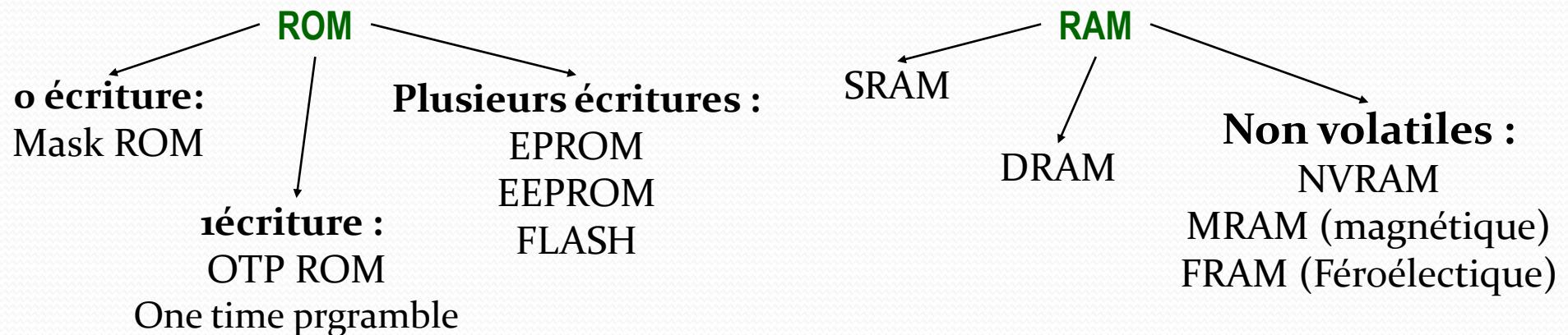
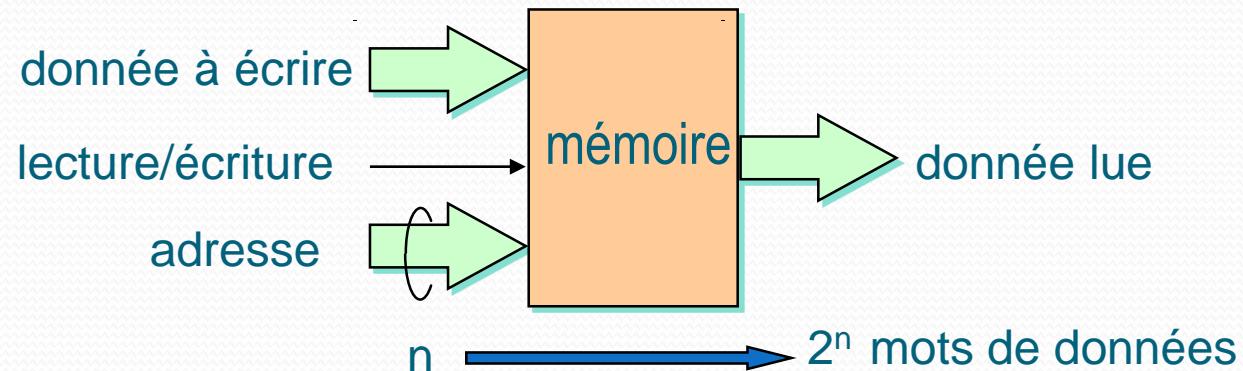
Chaque cœur a sa propre unité de calcul, propre cache, ...



Mémoire

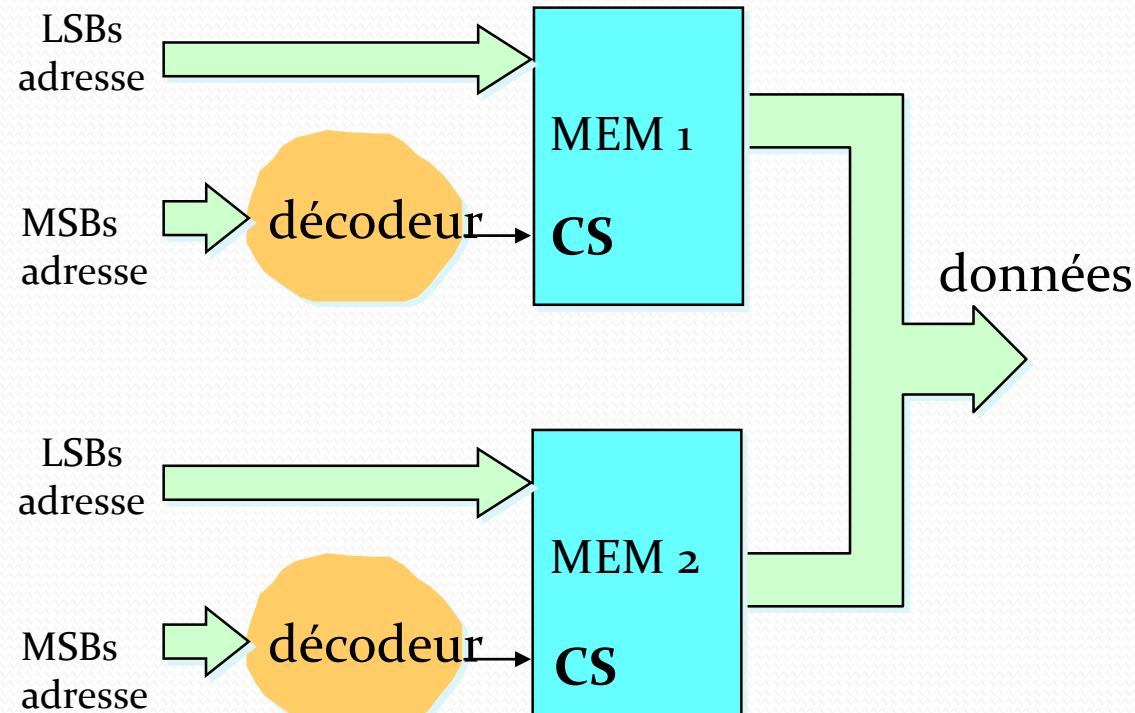


mémoire à semi-conducteur

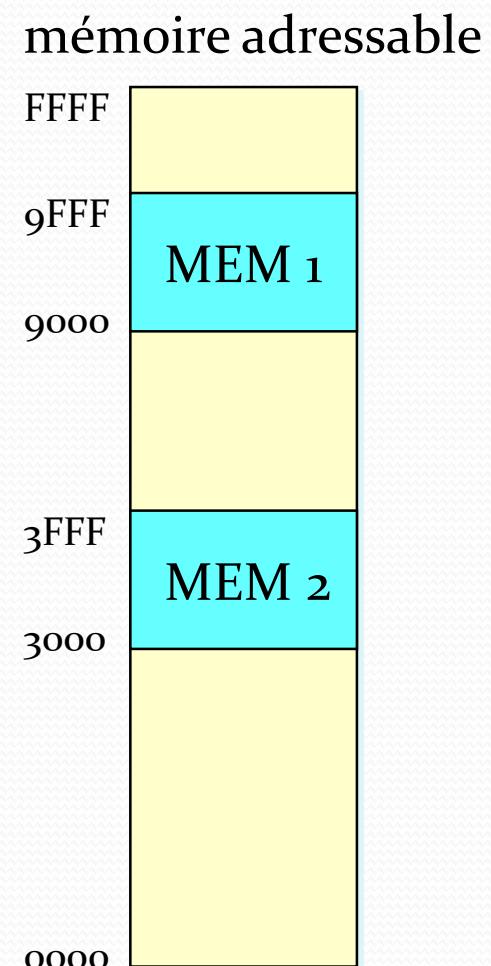


Mappe mémoire

La mappe mémoire indique les champs d'adresse mémoire dans un système

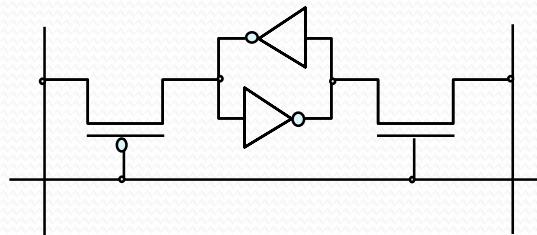


Chaque boîtier a son «*Chip Select*» (CS)



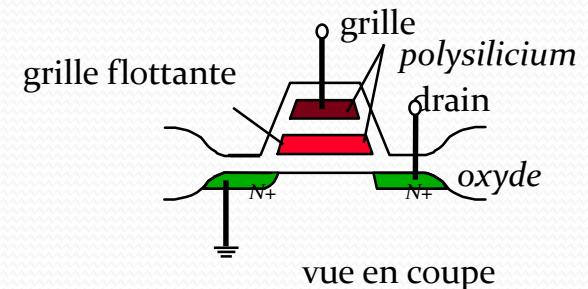
Principales technologies silicium

EEPROM, FLASH : basée sur les transistors à grille flottante

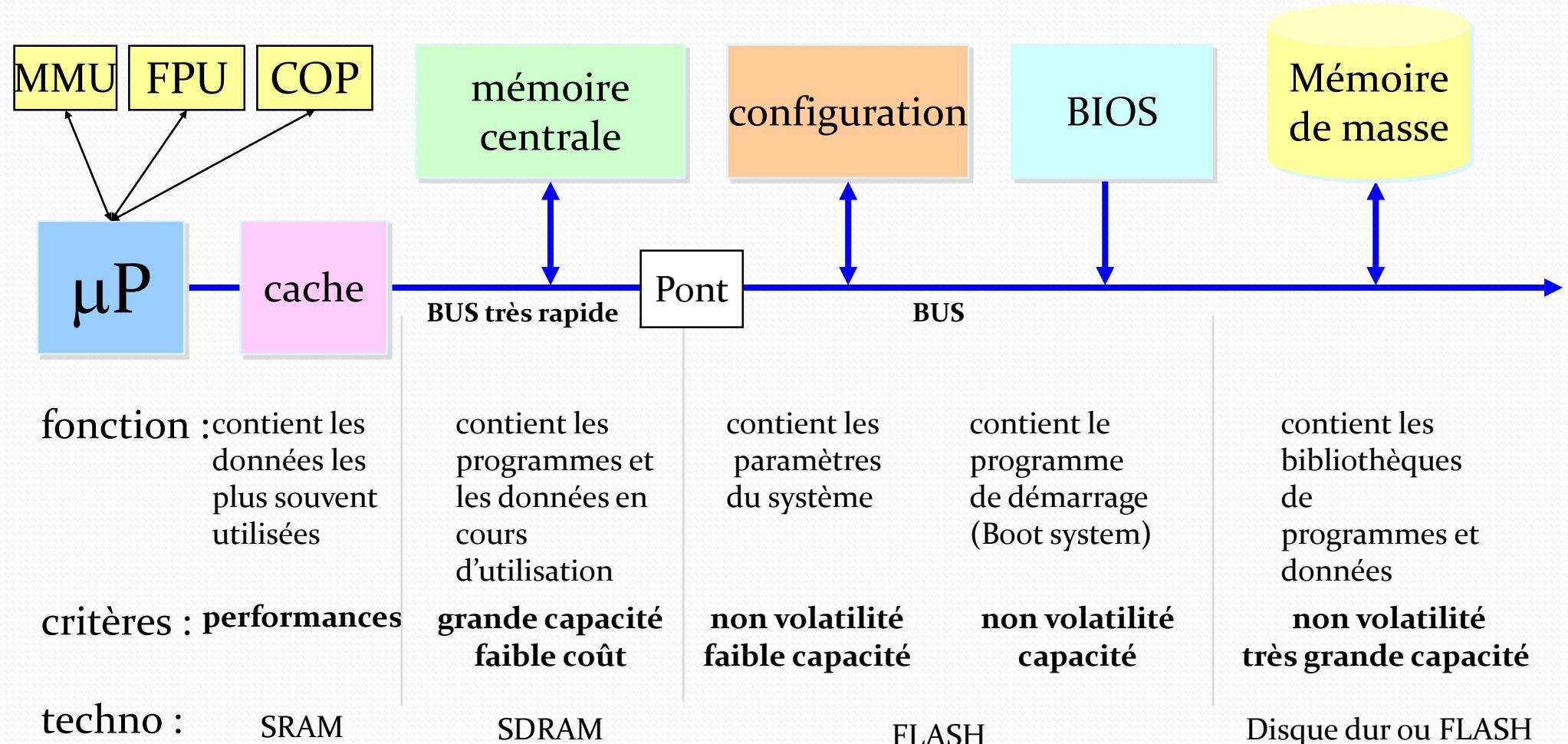


SRAM : CMOS

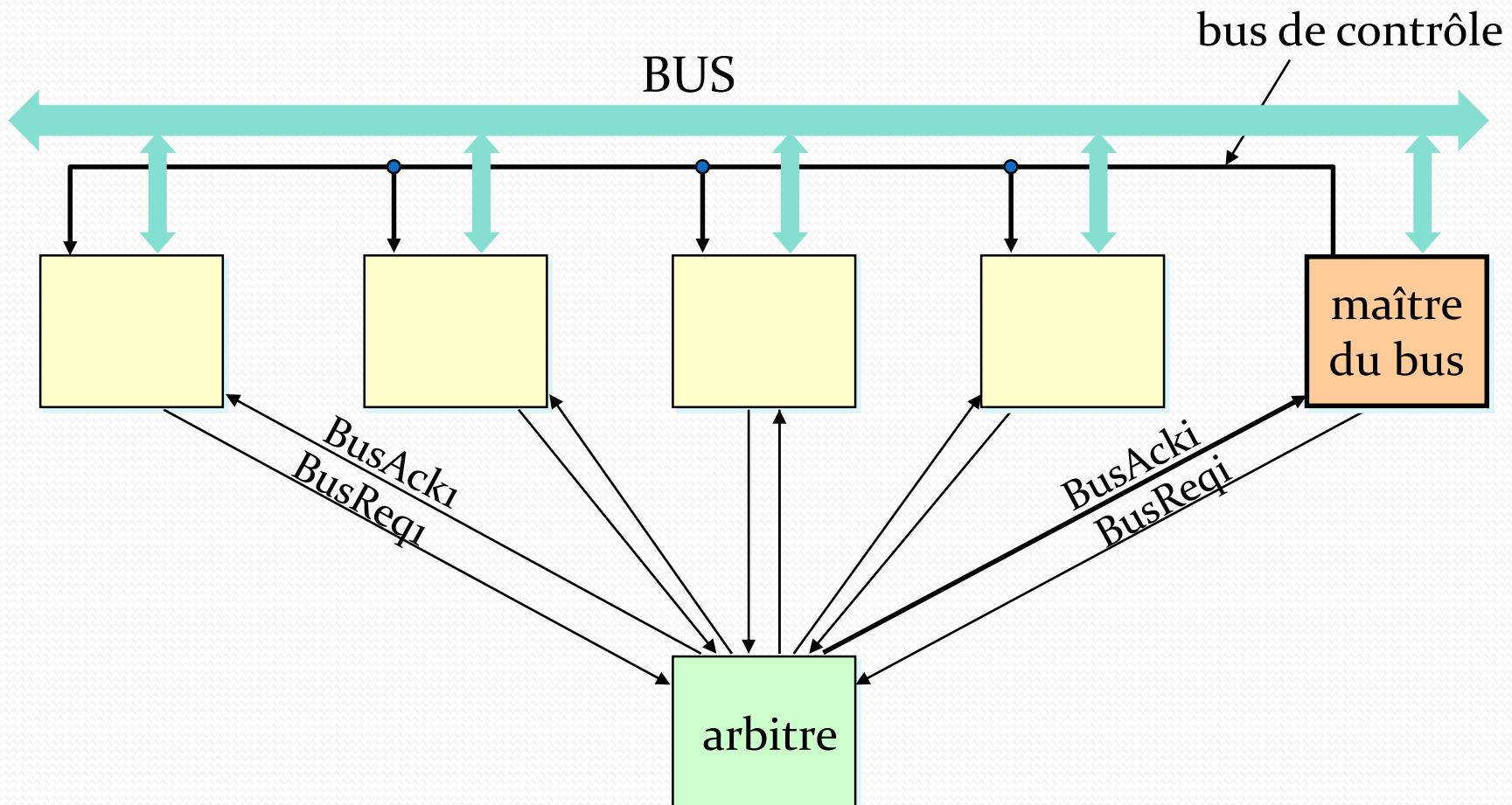
DRAM : capacité grille d'un transistor MOS
=> nécessité de rafraîchissement mais capacité x4



Architecture d'un système à microprocesseur

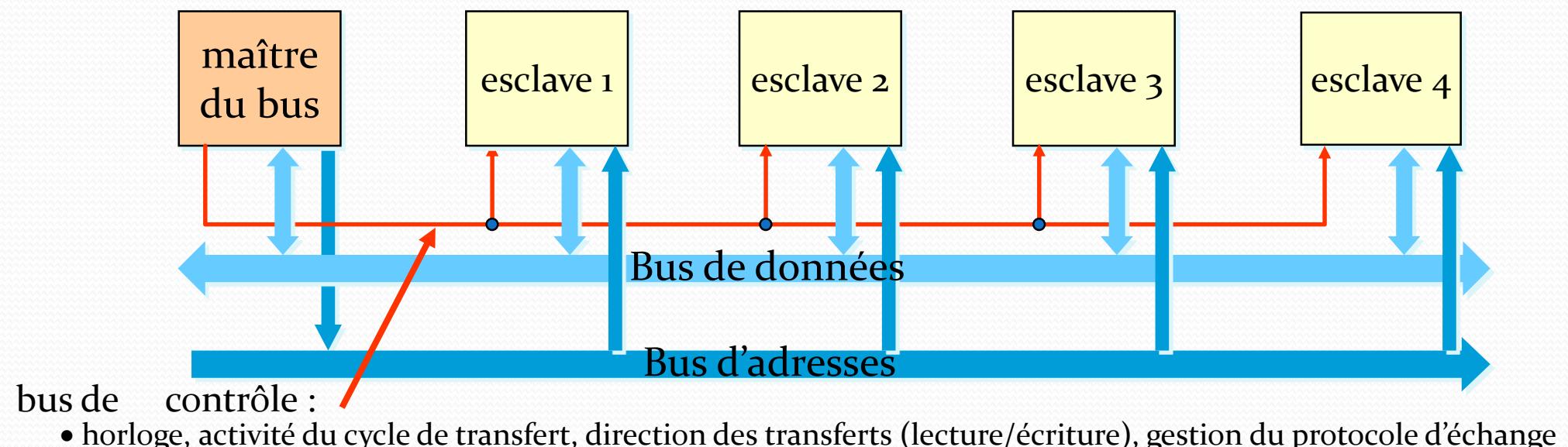


Arbitrage de bus



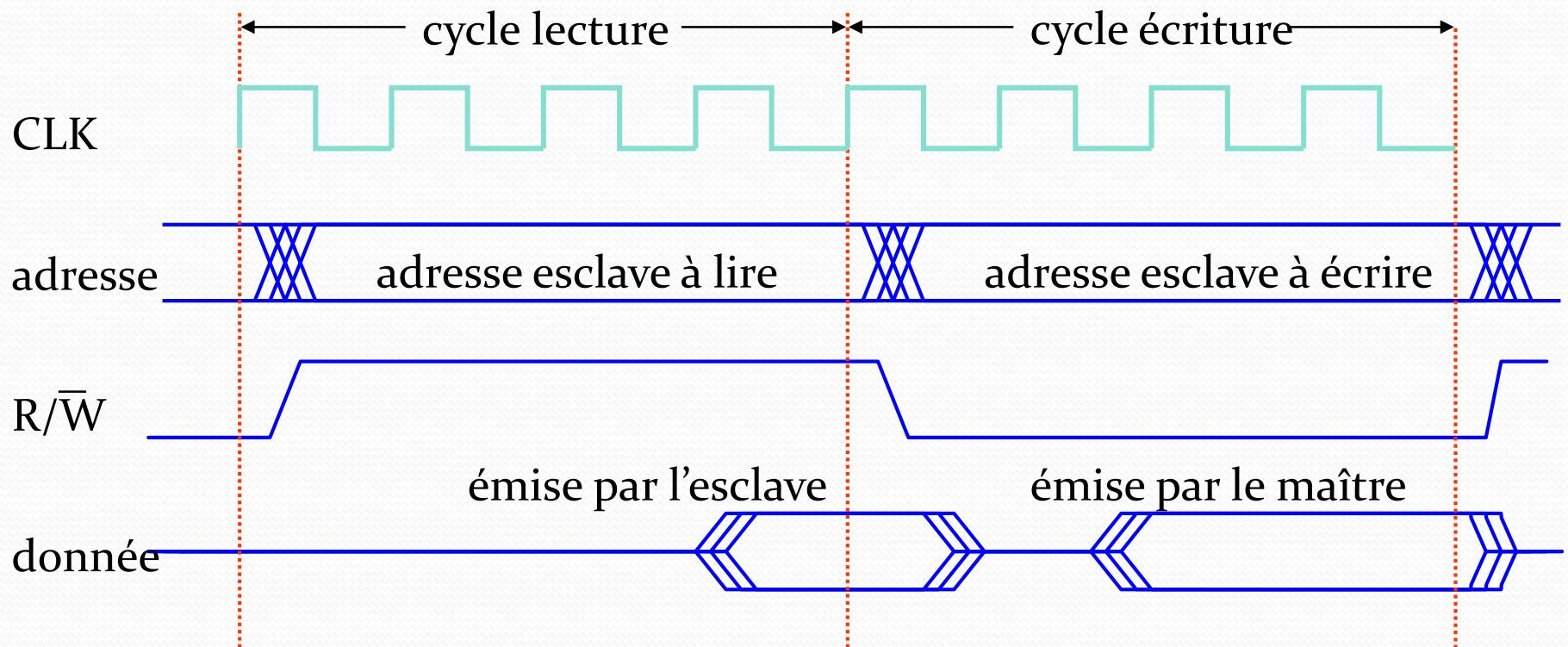
Transfert sur un bus

- Nécessité d'associer un champ d'adresse à un élément.
 - chaque élément a son propre décodeur d'adresse générant un "CS"
- Espace adressable
 - soit dans la mappe mémoire
 - soit dans la mappe E/S (n'existe pas toujours)



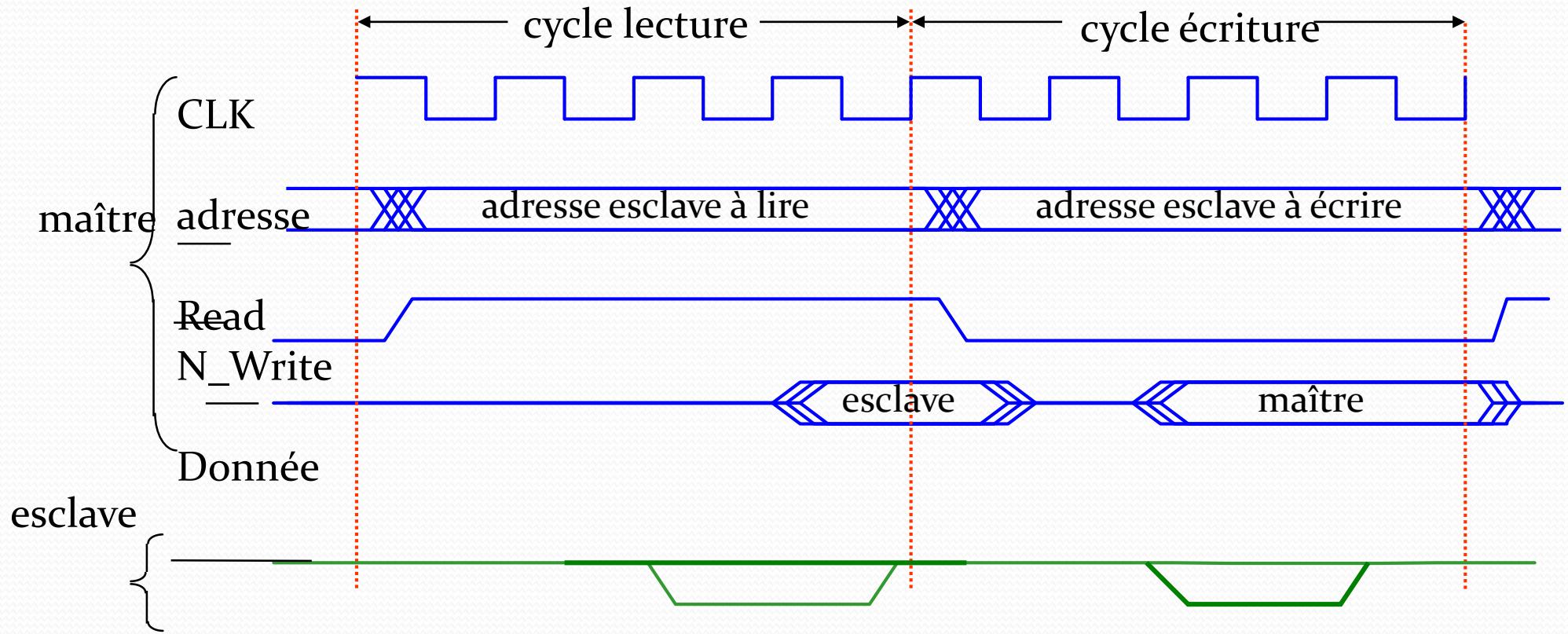
Protocole de communication synchrone

L'horloge rythme les échanges entre maître et esclave.
Le nombre de cycle est constant



Protocole de communication asynchrone

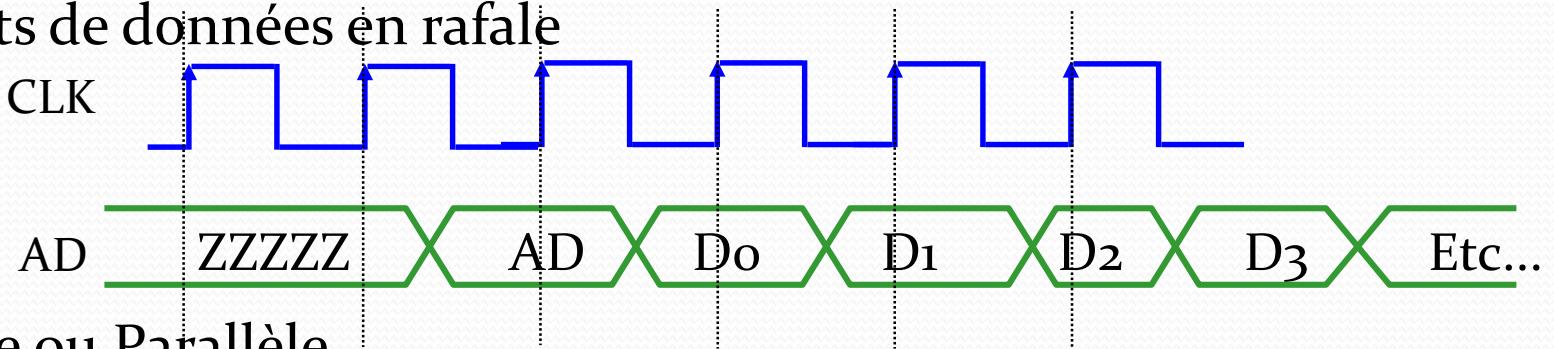
L'échange est assuré par un processus de “Handshake”, l'esclave renvoie au maître soit un signal pour acquiescer ou ralentir l'échange.



Caractéristiques de bus

- Protocoles
- Débits max
 - Transferts de données en rafale

Exemple : PCI

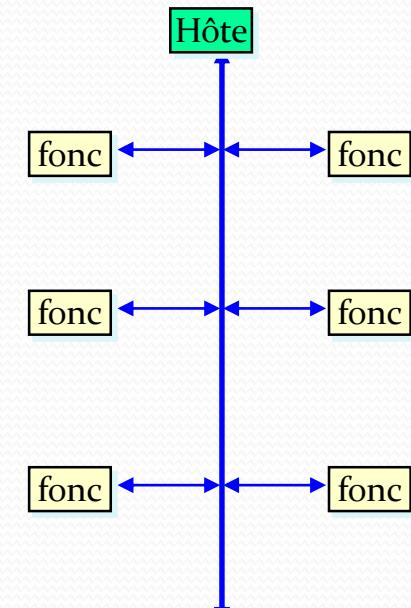
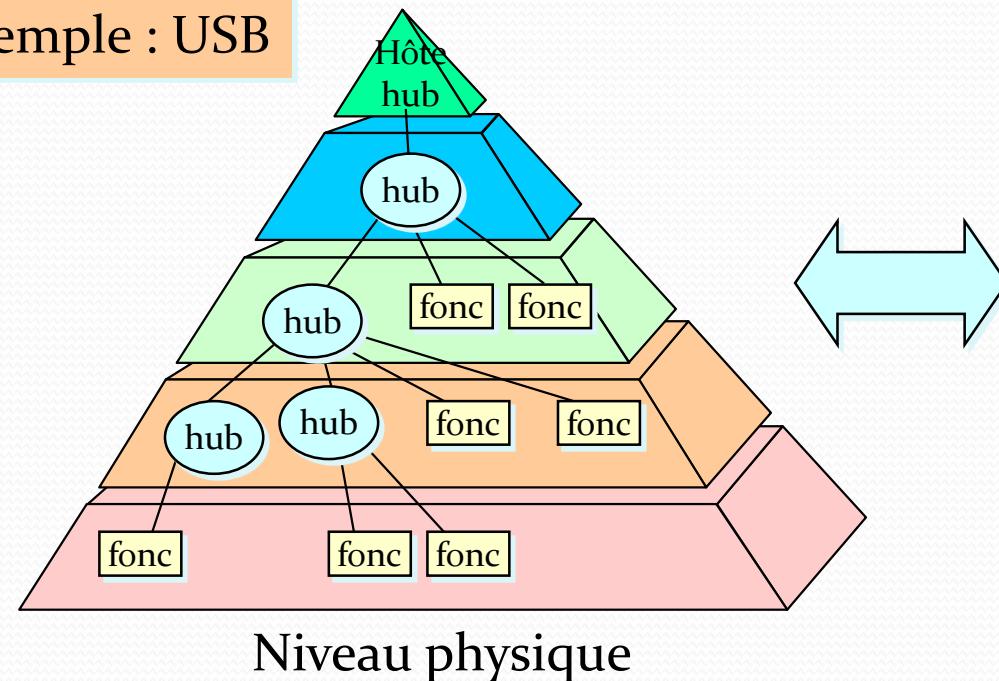


- Liaison Série ou Parallèle
- Caractéristiques électriques (tension, courant, impédance)
- Configuration automatique au démarrage ou "à chaud"
- Isochronisme
- Interruptions

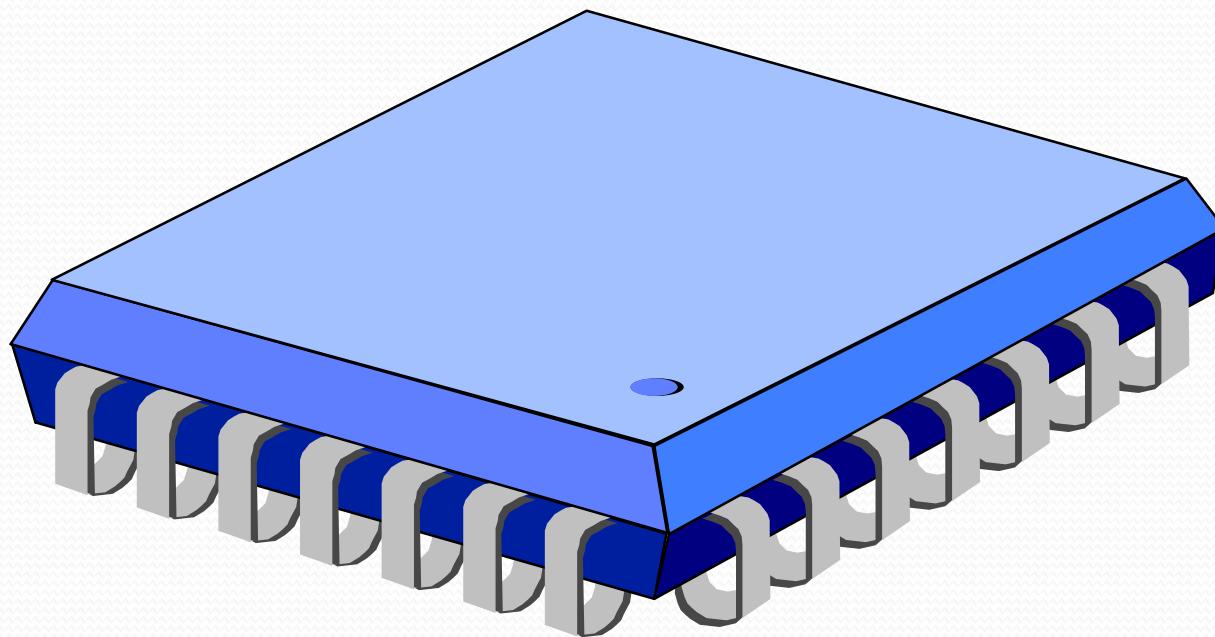
Topologie de bus

- Certains bus peuvent avoir une topologie point à point au niveau physique et une topologie de type Bus au niveau "liaison"

Exemple : USB



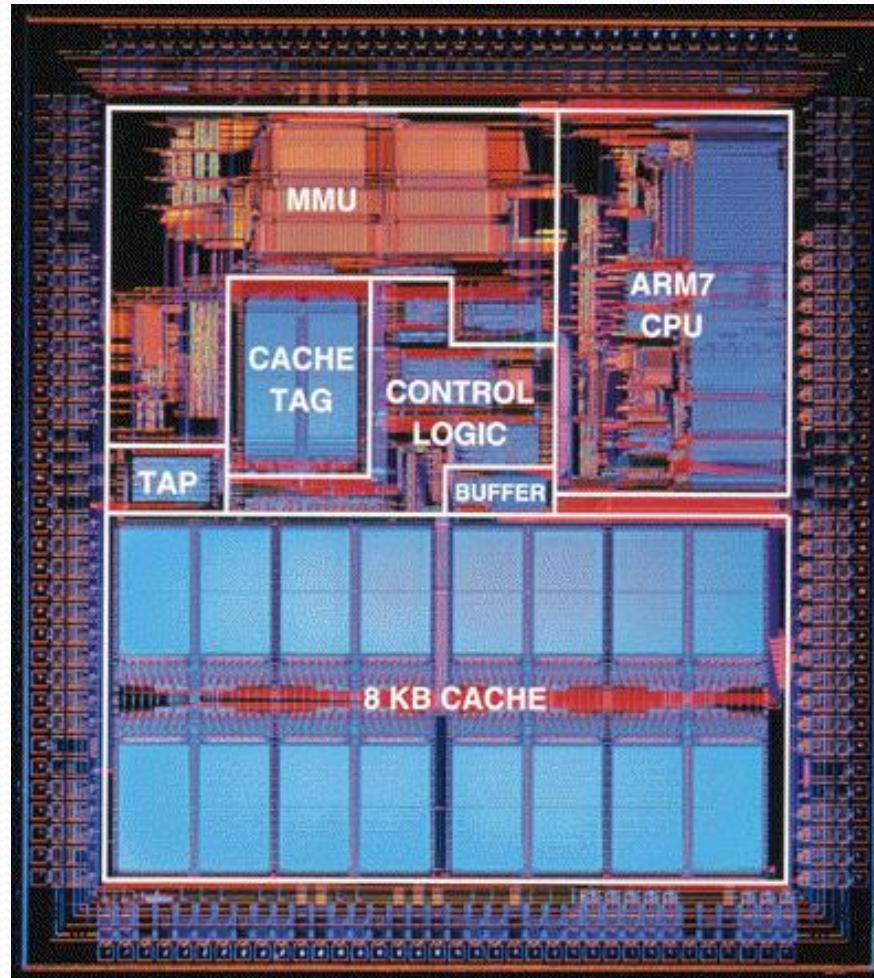
ARCHITECTURE de l'ARM7TDMI



Qu'est ce qu'un ARM7TDMI?

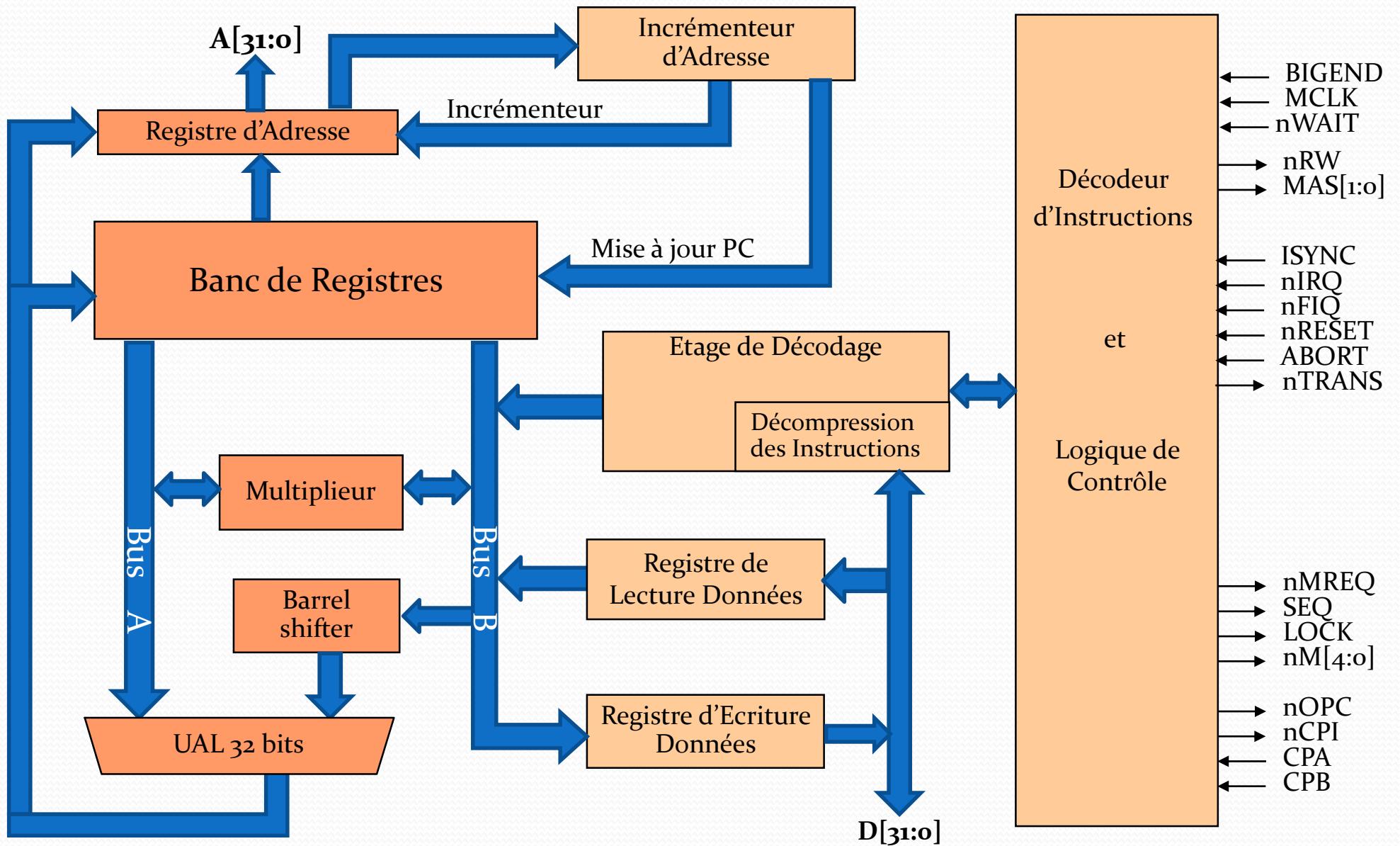
- Processeur à Architecture « Von Neumann »
- 3 étages de pipeline : Fetch, Decode, Execute
- Instructions sur 32 Bits
- 2 instructions d'accès à la mémoire LOAD et STORE
- T : support du mode "Thumb" (instructions sur 16 bits)
- D : extensions pour le debugging
- M : Multiplieur 32x8 et instructions pour résultats sur 64 bits.
- I : émulateur embarqué ("Embedded ICE")

Vue d'une puce utilisant un ARM7



ARM710 (25mm^2 en $0.5\mu\text{m}$ (1995), 2.9 mm^2 en $0.18\mu\text{m}$ (2000))

Vue simplifiée du Cœur ARM7TDMI

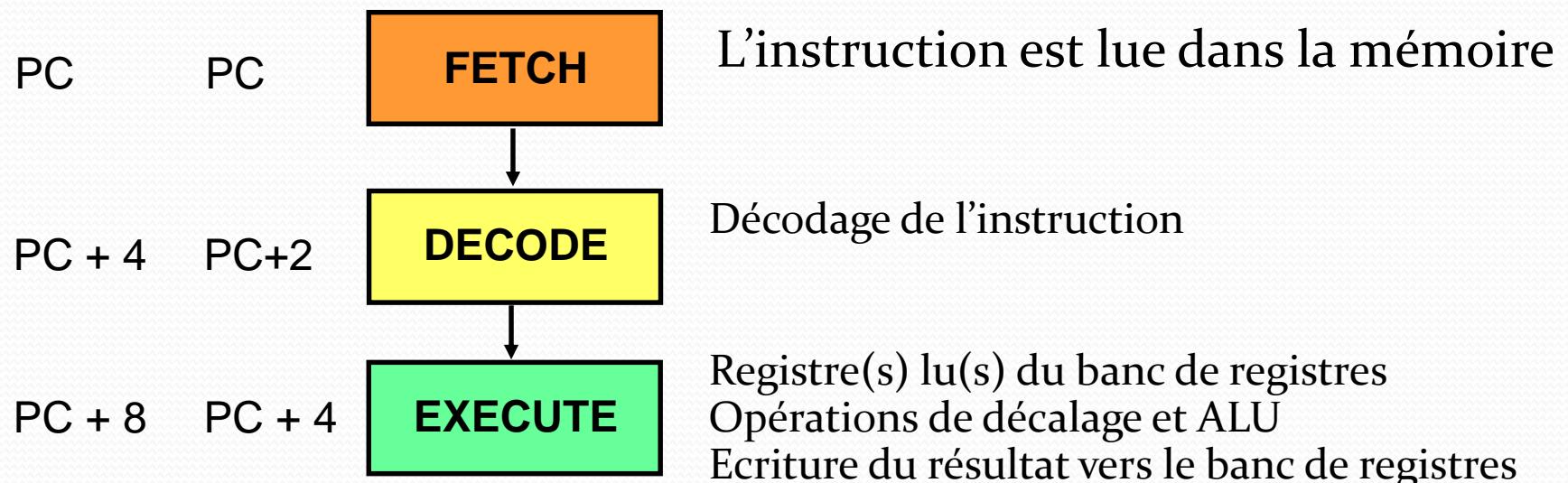


UAL: Unité Arithmétique & Logique

Le Pipeline d'Instructions

La famille ARM7 utilise un pipeline à 3 étages pour augmenter la vitesse du flot d'instructions dans le microprocesseur.

Mode: ARM Thumb



Le PC pointe sur l'instruction en cours de lecture (FETCHed), et non sur l'instruction en cours d'exécution.

Exemple: Pipeline Optimal

Cycle	1	2	3	4	5	6
Instruction						
ADD	Fetch	Decode	Execut			
SUB		Fetch	Decode	Execut		
MOV			Fetch	Decode	Execut	
AND				Fetch	Decode	Execut
ORR					Fetch	Decode
EOR						Execut
CMP					Fetch	Decode
RSB						Fetch

- il faut 6 cycles pour exécuter 6 instructions (CPI “Cycles Per Instruction”)=1
- Toutes les opérations ne jouent que sur des registres (1 cycle)

Exemple: Pipeline avec LDR

Cycle	1	2	3	4	5	6
Instruction						
ADD	Fetch	Decode	Execut			
SUB		Fetch	Decode	Execut		
LDR			Fetch	Decode	Execut	Data
MOV				Fetch	Decode	
AND					Fetch	
ORR						Fetch

[mem]=>tampon
tampon=>registre

- L'instruction LDR lit une donnée en mémoire et la charge dans un registre
- il faut 6 cycles pour exécuter 4 instructions. CPI = 1,5

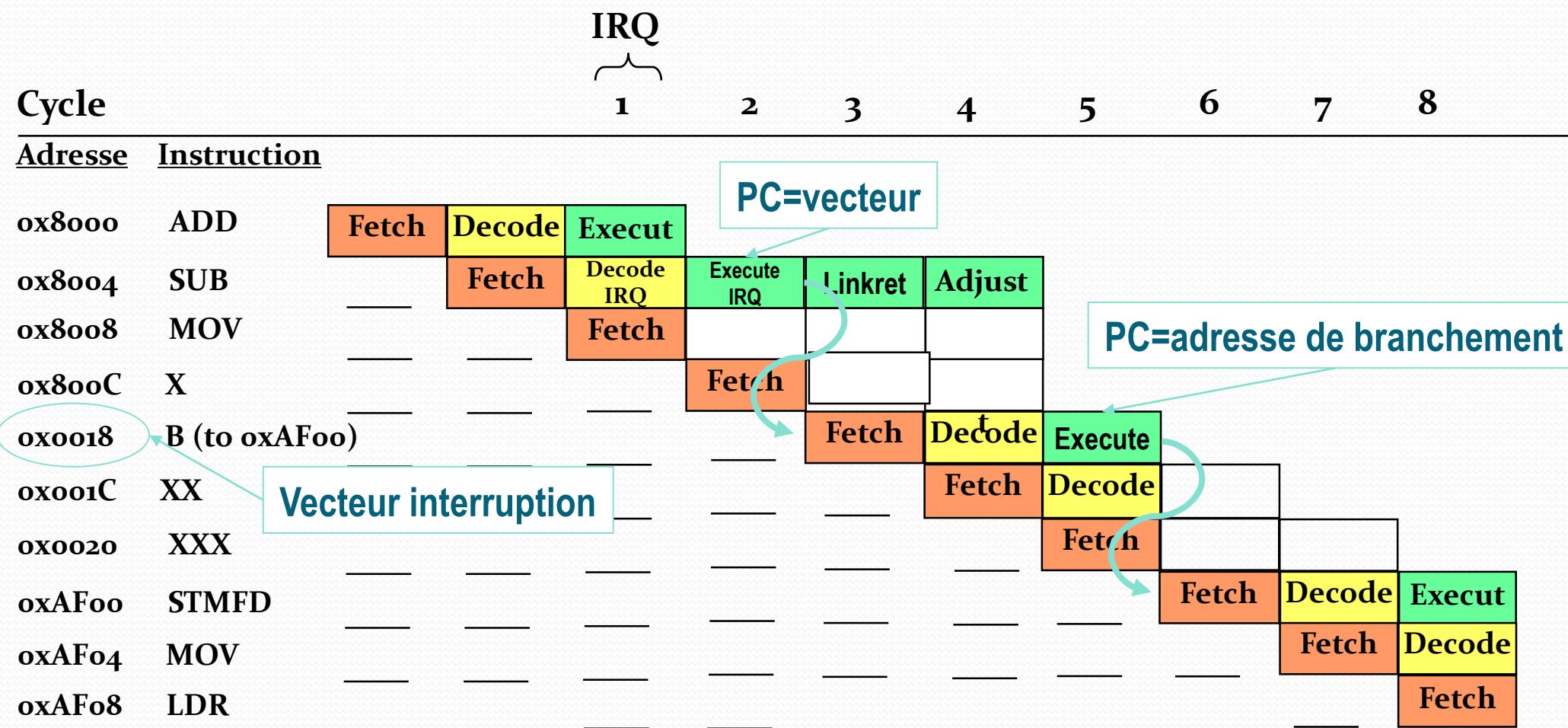
Exemple: Pipeline avec Saut

<u>Cycle</u>		1	2	3	4	5
<u>Adresse</u>	<u>Instruction</u>					
0x8000	BL	Fetch	Decode	Execute	Linkre	Adjust
0x8004	X	Fetch	Decode	t		
0x8008	XX	Fetch	Fetch			
0x8FEC	ADD	Fetch	Fetch	Decode	Execut	
0x8FF0	SUB	Fetch	Fetch	Decode	Execut	
0x8FF4	MOV	Fetch	Fetch	Fetch	Decode	Fetch

LR=PC LR=LR-4

- L'instruction BL effectue un appel de sous-programme
- Le pipeline est cassé et 2 cycles sont perdus

Exemple: Pipeline avec Interruption



* Latence minimum pour le service de l'interruption IRQ = 7 cycles

Les Modes du Microprocesseur

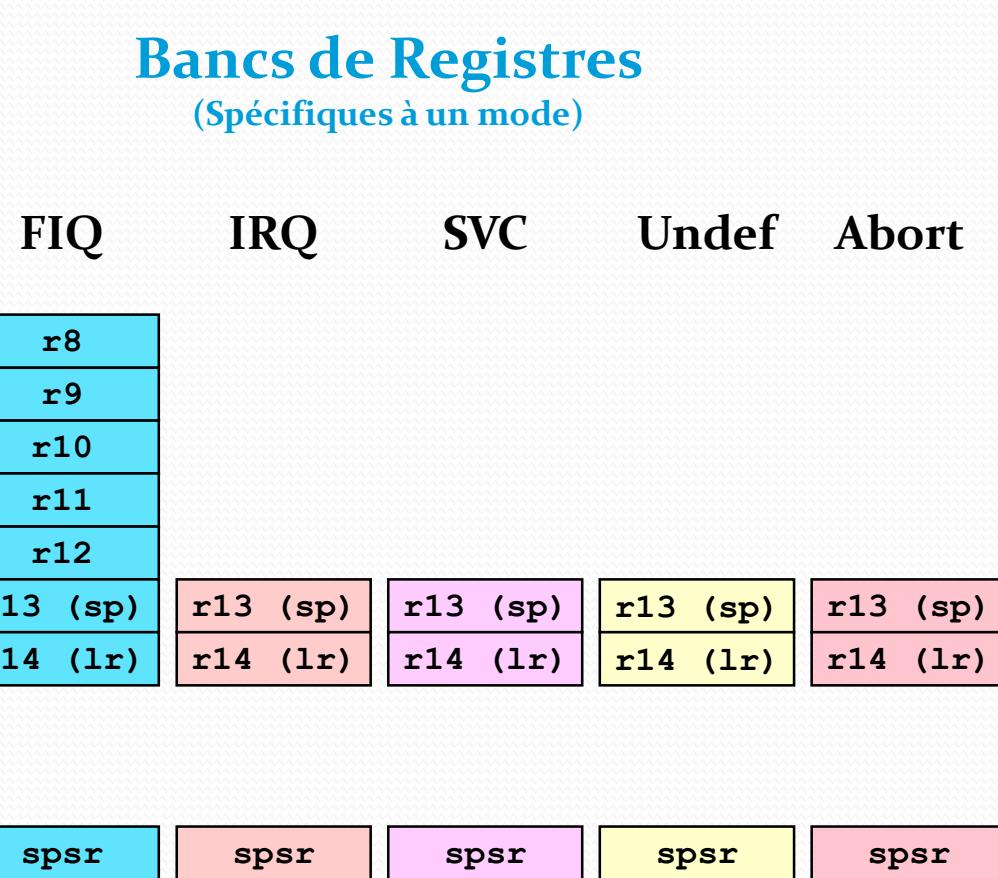
- Un microprocesseur ARM a 7 modes opératoires de base :
 - **User** : mode sans privilège où la plupart des tâches s'exécutent
 - **FIQ** : on y entre lors d'une interruption de priorité haute (rapide)
 - **IRQ** : on y entre lors d'une interruption de priorité basse (normale)
 - **Supervisor** : on y entre à la réinitialisation et lors d'une interruption logicielle (SWI “SoftWare Interrupt”)
 - **Abort** : utilisé pour gérer les violations d'accès mémoire
 - **Undef** : utilisé pour gérer les instructions non définies (“undefined”)
 - **System** : mode avec privilège utilisant les mêmes registres que le mode User

Les Registres

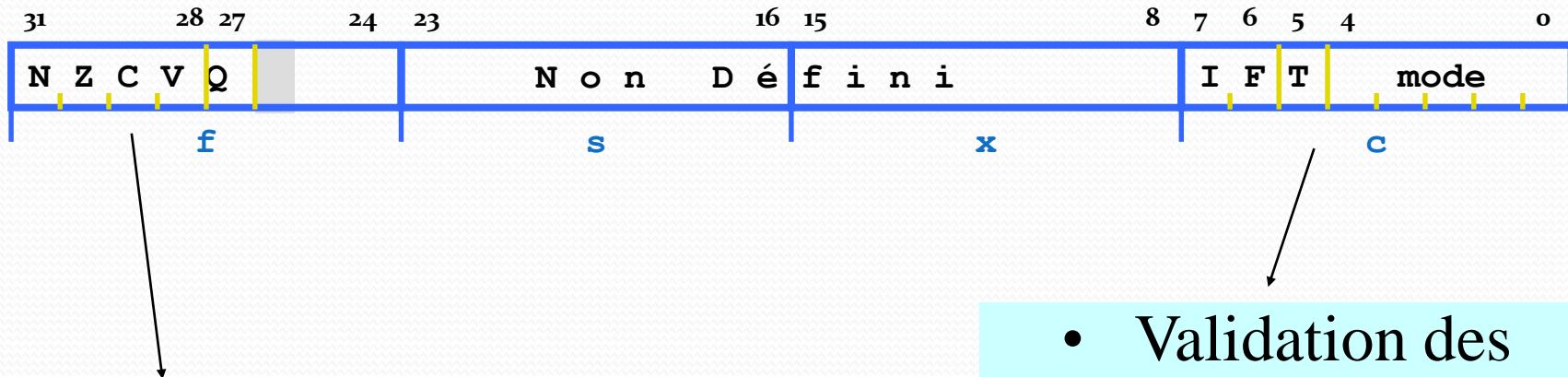
Registres actifs

Mode
Utilisateur

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)



Les Registres d'État CPSR et SPSR



- **Indicateurs conditionnels**

- N = Résultat **Négatif**
- Z = Résultat nul (**Zéro**)
- C = Retenue (**Carry**)
- V = Débordement (**oVerflow**)
- Q = débordement avec mémoire

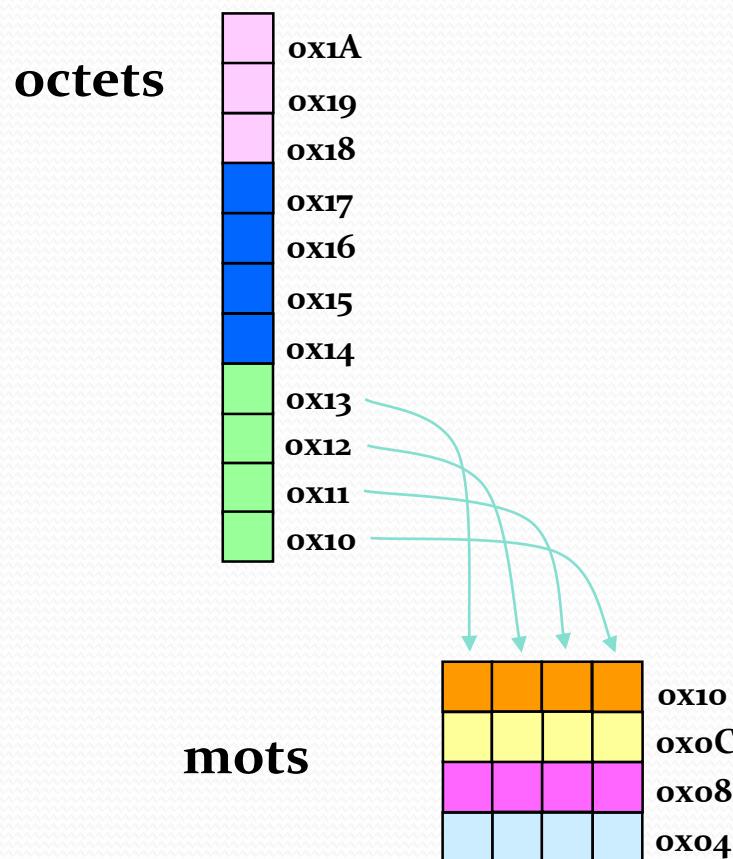
- Validation des interruptions
 - I = 1 dévalide IRQ.
 - F = 1 dévalide FIQ.
- Mode Thumb
 - T
- Indicateurs de mode
 - Indiquent le mode actif :

Accès à la Mémoire et aux E/S

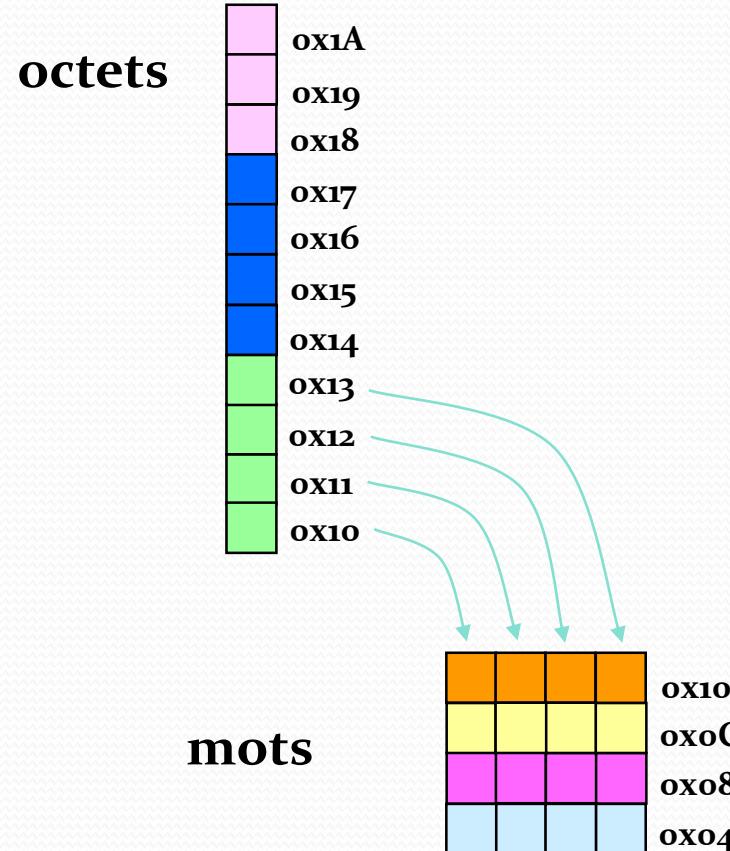
- 2 instructions d'accès :
 - LOAD (LDR) et STORE (STR)
- L'adressage mémoire se fait sur 32 bits
 - => 4 Go.
- Type des données :
 - octets
 - demi-mots (16 bits)
 - mots (32 bits)
- Les mots doivent être alignés sur des adresses multiples de 4 et les demi-mots, de 2.
- Les E/S sont dans la « mappe » mémoire

Organisation de la mémoire

La mémoire peut être vue comme une ligne d'octets repliée en mots.
2 façon d'organiser 4 octets en mot :



« Little Endian »



« Big Endian »

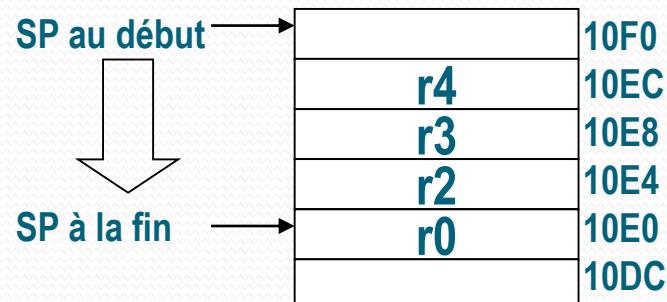
Jeu d'Instructions ARM(1)

- Toutes les instructions ont 32 bits
 - La plupart des instructions s'exécutent en un seul cycle
 - Les instructions peuvent être exécutées conditionnellement
 - Mettre à jour les indicateurs conditionnels
 - Architecture Load/Store
 - Instructions de traitement de données
 - SUB r0,r1,#5 ; r0= r1-5
 - ADD r2,r3,r3,LSL #2 ; r2=R3+4*r3=5*r3
 - ANDS r4,r4,#0x20 ; r4=r4 ET 0x20
 - ADDEQ r5,r5,r6 ; r5=r5+r6 si Z
 - Instructions spécifiques d'accès à la mémoire
 - LDR r0, [r1], #4 ; r0 = MEM[r1] et r1+=4
 - STRNEB r2, [r3] ; Stockage en octet si Z faux

Jeu d'Instructions ARM(2)

- Instructions spécifiques d'accès à la mémoire

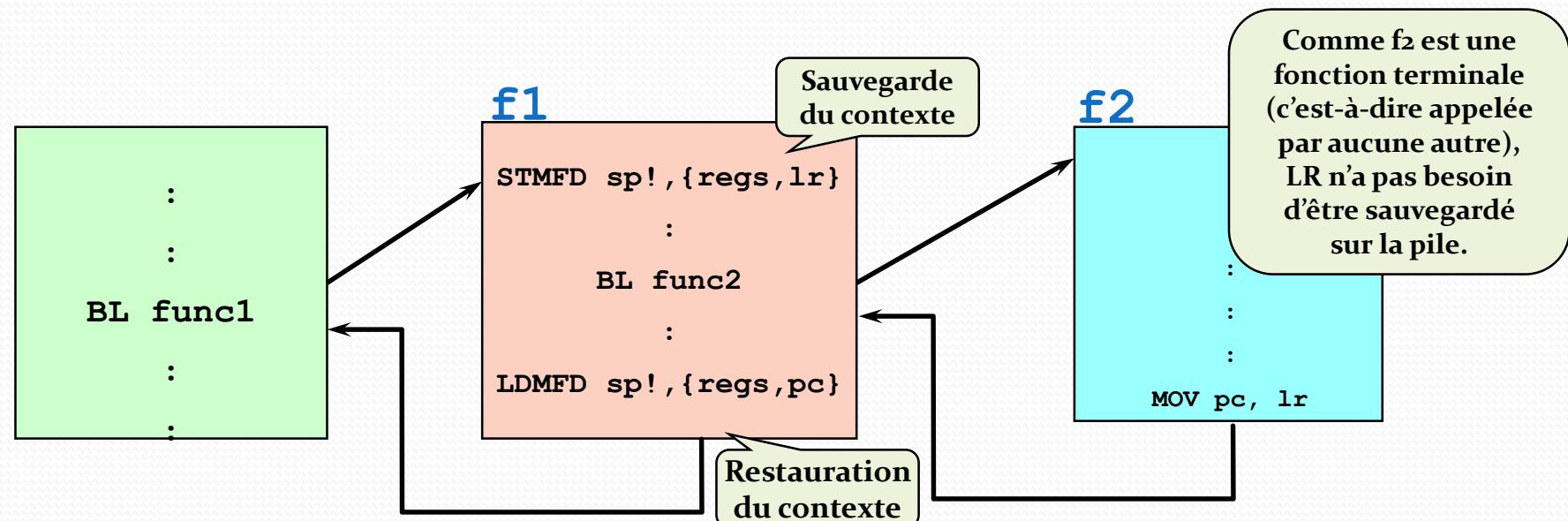
- LDR** $r0, [r1], \#4$; $r0 = \text{mem}(r1), r1 = r1 + 4$
 - STRNEB** $r2, [r3, r4]$; $\text{mem}(r3 + r4) = r2$
 - LDRSH** $r5, [r6, \#8] !$; $r5 = \text{mem}(r6 + 8), r6 = r6 + 8$
 - STMFD** $sp!, \{r0, r2-r4\}$; transferts multiples
 - ; empilage : $\text{mem}(sp+i) = \text{liste de registres}$
- Si Z=0 En octet
- En 16 bits
Avec extension de signe
sur les 16MSBs
- r6=r6+8 en fin d'exécution



Opération réciproque de dépileage :
LDMFD $sp!, \{r0, r2-r4\}$

Jeu d'Instructions ARM(3)

- Branchement et sous programmes :
- B <étiquette>
 - Calculé par rapport au PC. Étendue du branchement: ± 32 Mbyte.
- BL <subroutine>
 - Stocke l'adresse de retour dans le registre LR
 - Le retour se fait en rechargeant le registre LR dans le PC
 - Pour les fonctions non terminales, LR devra être sauvegardé



Exécutions conditionnelles

- La plupart des instructions peuvent être exécutées conditionnellement aux indicateurs Z,C,V,N
- **CMP** **r0 ,#8 ; r0=8?**
- **BEQ** **fin ; si oui (Z=1) PC=fin**
- **ADD** **r1,r1,#4 ;**
- Équivalent à
- **CMP** **r0 ,#8 ; r0=8?**
- **ADDNE** **r1,r1,#4 ; si non (Z=0)**

+ petit et
+ rapide

Conditions courantes : **EQ, NE,PL,MI,CS,VS**

=0, $\neq 0$, ≥ 0 , < 0 , carry set, débordement

Exemple: Séquence d'Instructions

```
LDR    R0, [R8, 0x10]  
ADD    R1, R0, R4, LSL #2  
STR    R1, [R8, 0x14]
```

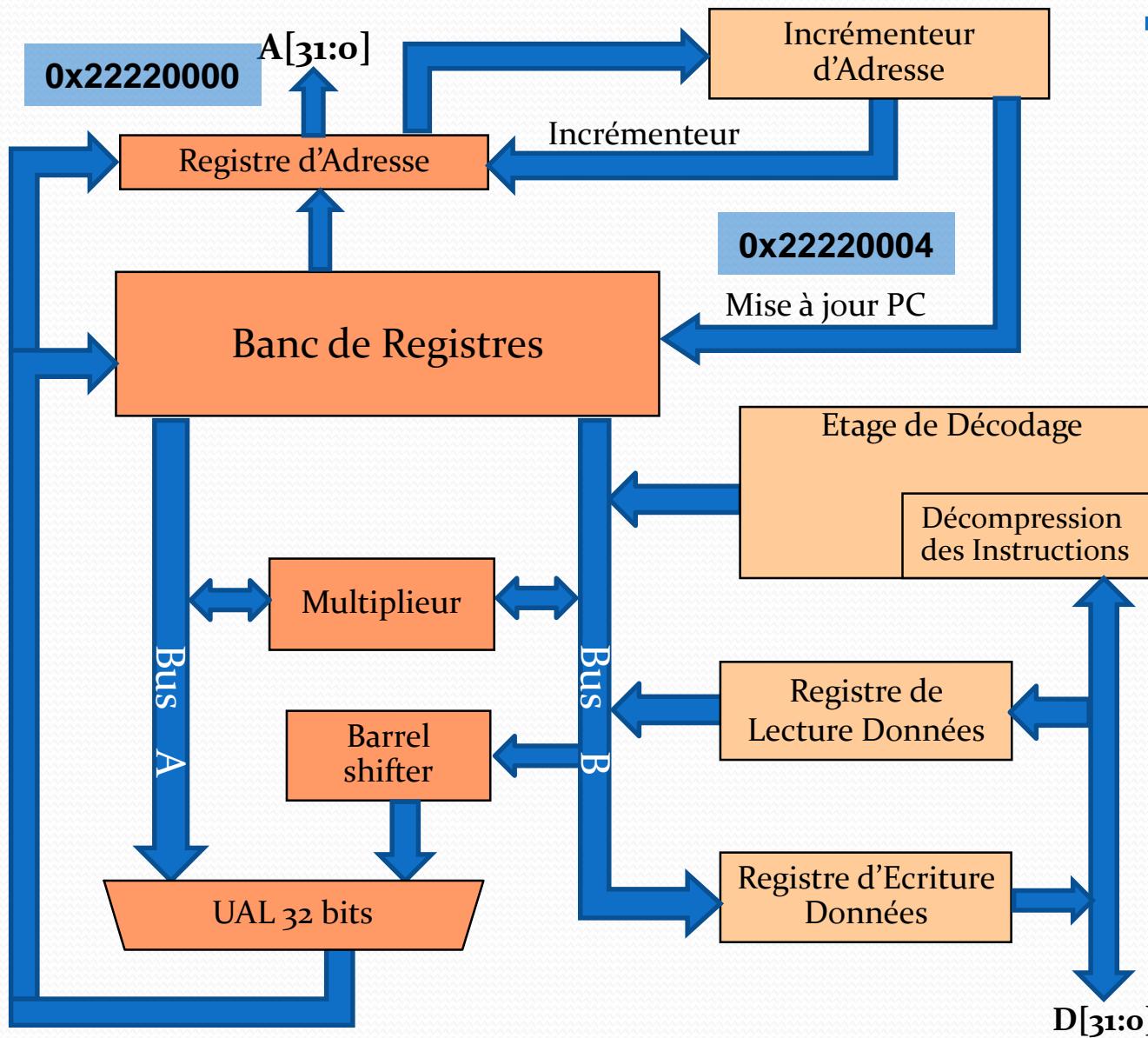
charger le mot de l'adresse [R8+0x10] dans R0
 $R1 \leftarrow R0 + (R4 \ll 2)$
ranger le contenu de R1 à l'adresse [R8+0x14]

- Conditions initiales :

```
PC = 0x22220000  
R4 = 0x00000721  
R8 = 0x55551000  
[0x55551010] = 0x00000834
```

- Les diagrammes suivants supposent que les instructions précédentes s'exécutent en un cycle mais ne montrent pas leur comportement.

Séquence d'Instructions : Cycle 1



LDR	R0, [R8, 0x10]
ADD	R1, R0, R4, LSL #2
STR	R1, [R8, 0x14]

Cycle 1

- Lecture de l'instruction LDR

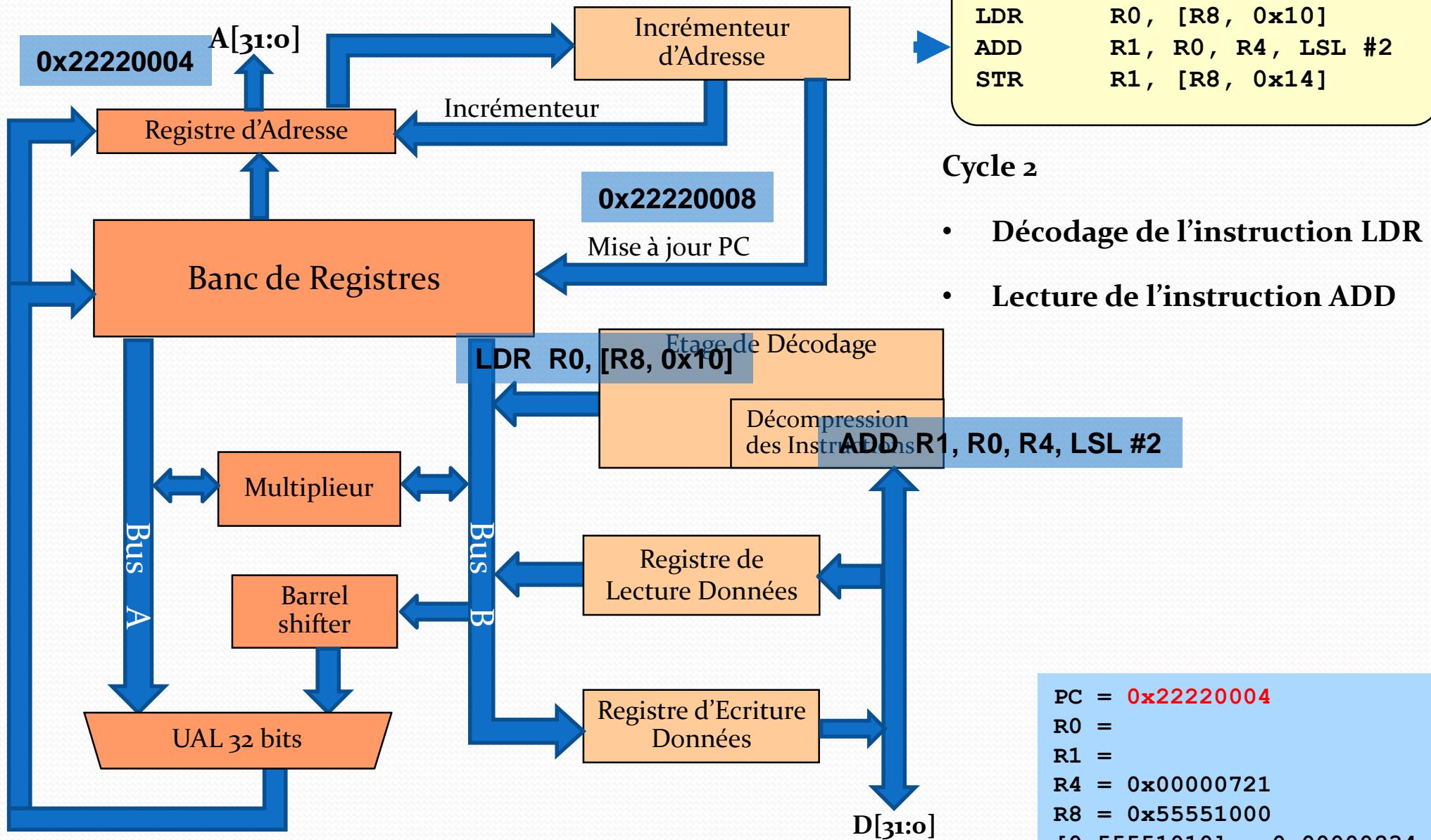
LDR R0, [R8, 0x10]

Conditions initiales :

PC = 0x22220000
R0 =
R1 =
R4 = 0x00000721
R8 = 0x55551000
[0x55551010] = 0x00000834

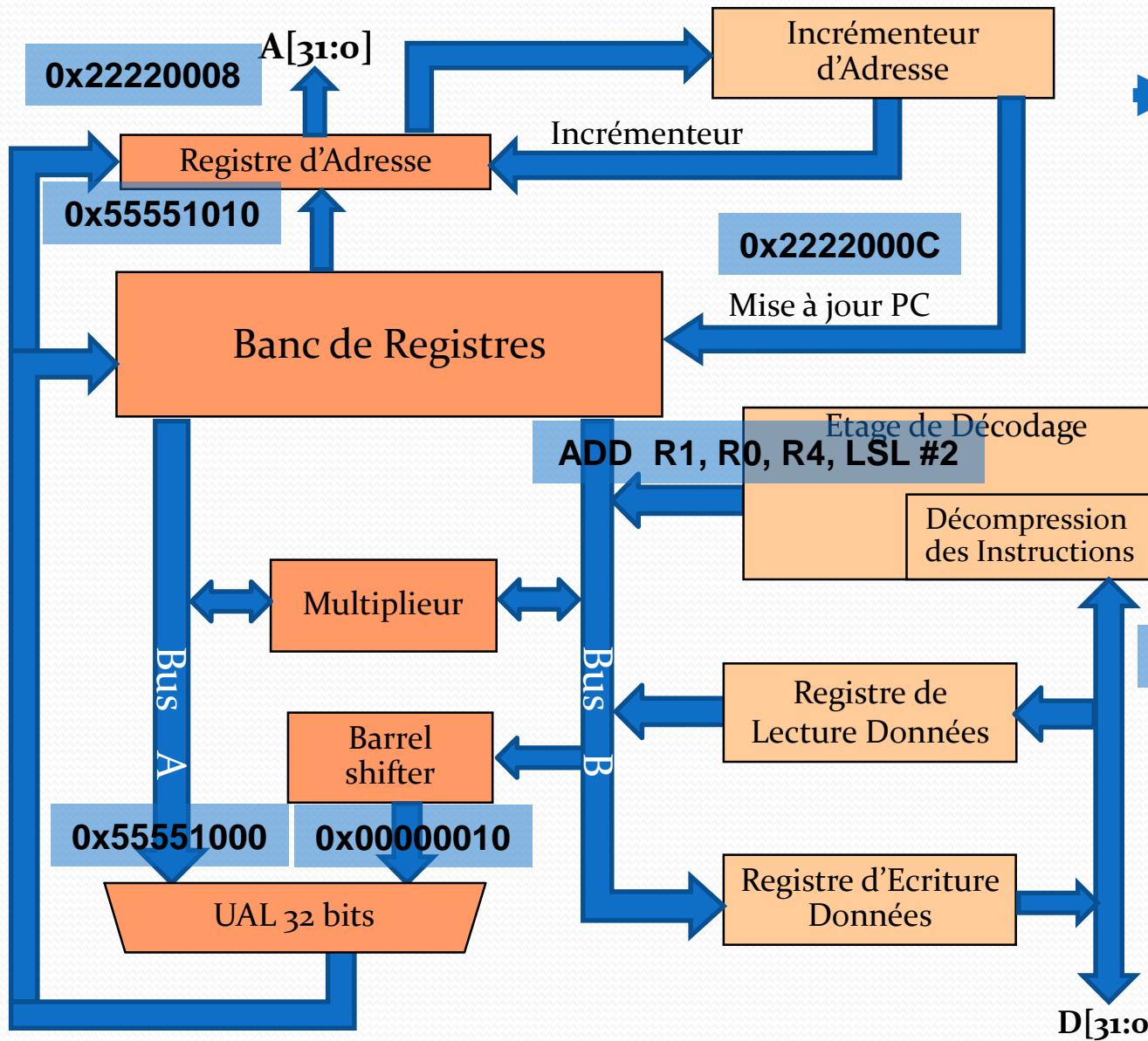
UAL: Unité Arithmétique & Logique

Séquence d'Instructions : Cycle 2



UAL: Unité Arithmétique & Logique

Séquence d'Instructions : Cycle 3



LDR R0, [R8, 0x10]
 ADD R1, R0, R4, LSL #2
 STR R1, [R8, 0x14]

Cycle 3

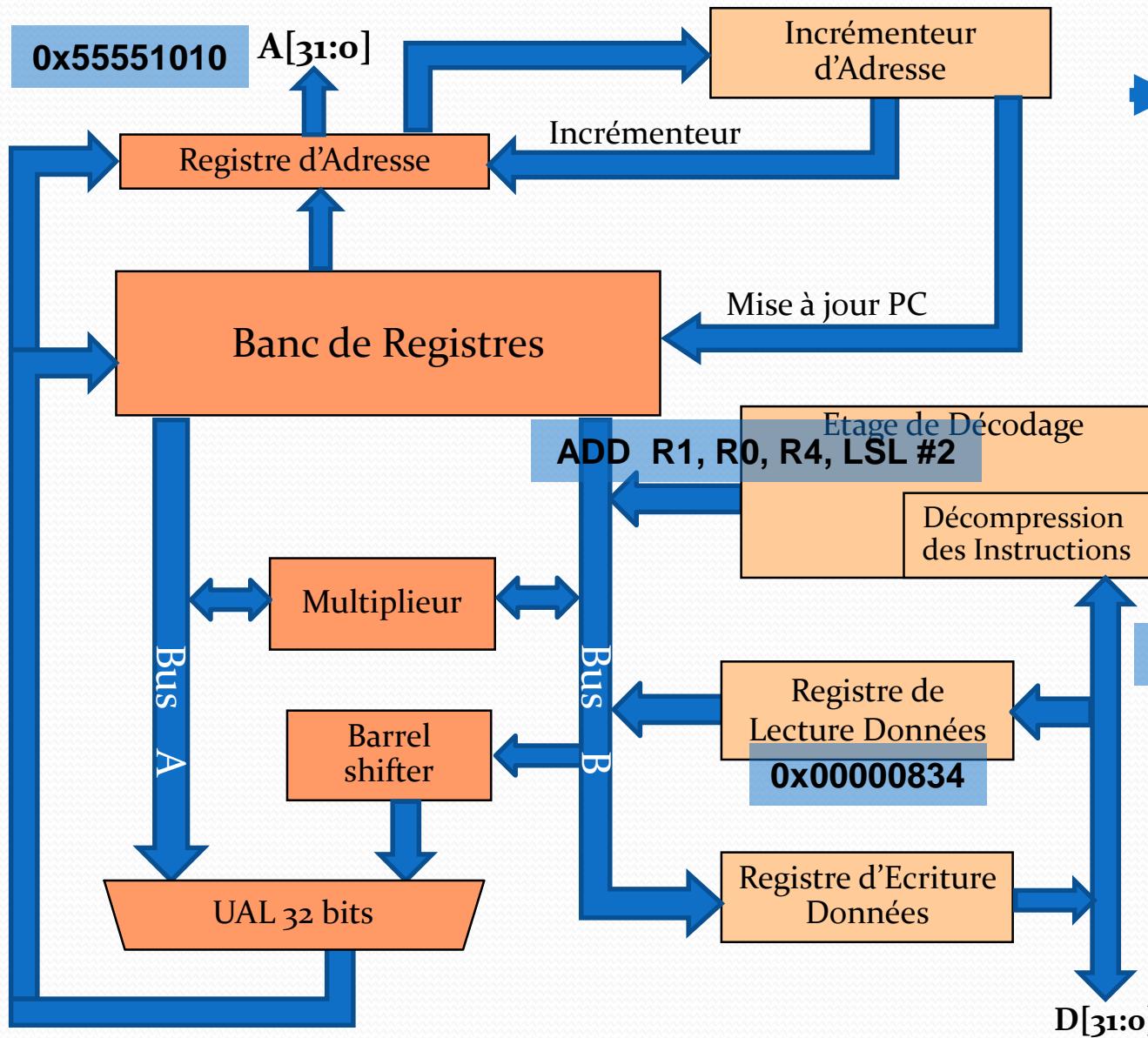
- 1^{er} cycle d'exécution de LDR
- Calcul de l'adresse en mémoire de données
- Décodage de l'instruction ADD
- Lecture de l'instruction STR

STR R1, [R8, #14]

PC = 0x22220008
 R0 =
 R1 =
 R4 = 0x00000721
 R8 = 0x55551000
 [0x55551010] = 0x00000834

UAL: Unité Arithmétique & Logique

Séquence d'Instructions : Cycle 4



LDR R0, [R8, 0x10]
 ADD R1, R0, R4, LSL #2
 STR R1, [R8, 0x14]

Cycle 4

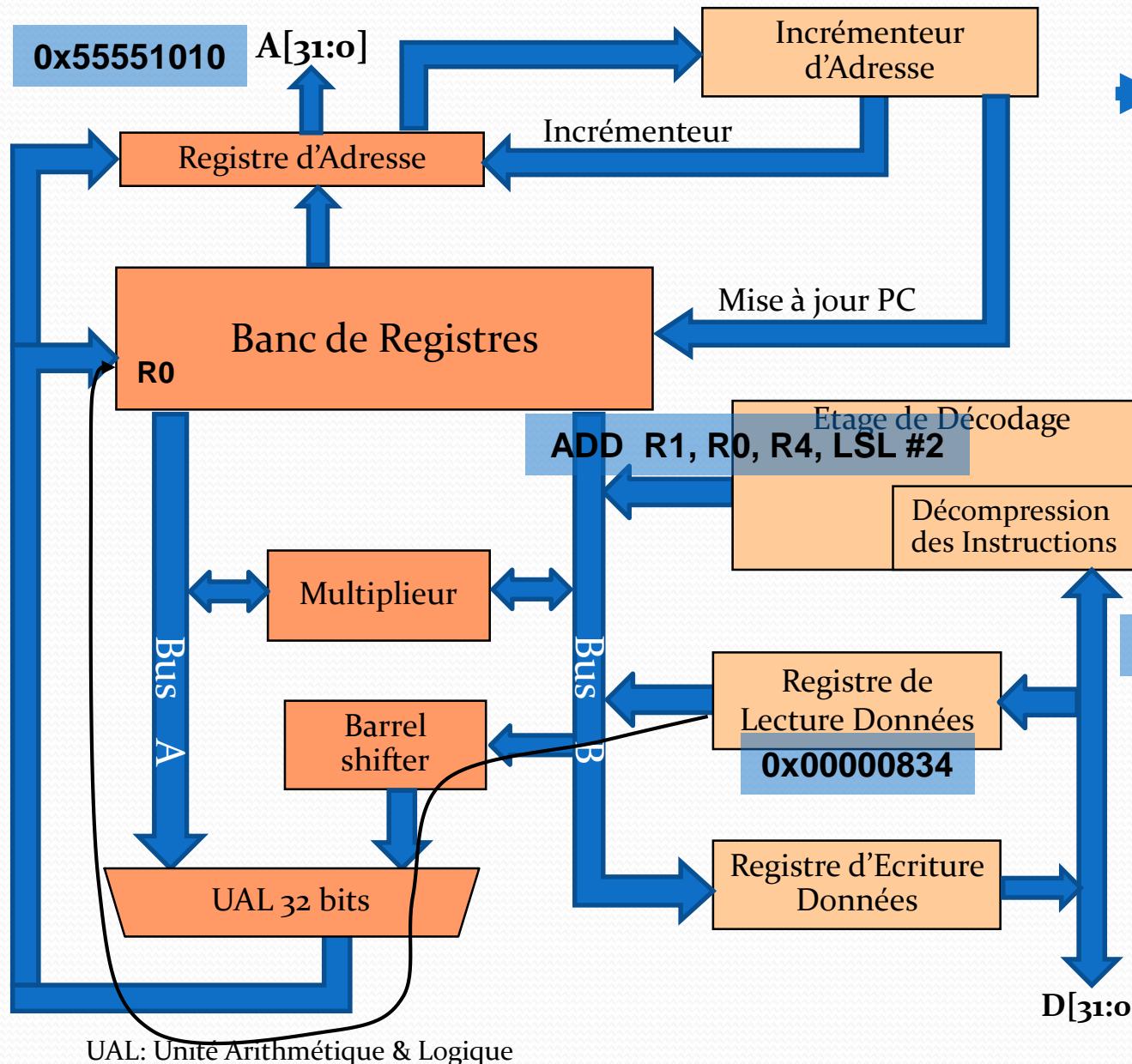
- 2^{eme} cycle d'exécution de LDR
- Lecture de la mémoire de données

STR R1, [R8, #14]

PC = 0x2222000C
 R0 =
 R1 =
 R4 = 0x00000721
 R8 = 0x55551000
 [0x55551010] = 0x00000834

UAL: Unité Arithmétique & Logique

Séquence d'Instructions : Cycle 5



LDR R0, [R8, 0x10]
 ADD R1, R0, R4, LSL #2
 STR R1, [R8, 0x14]

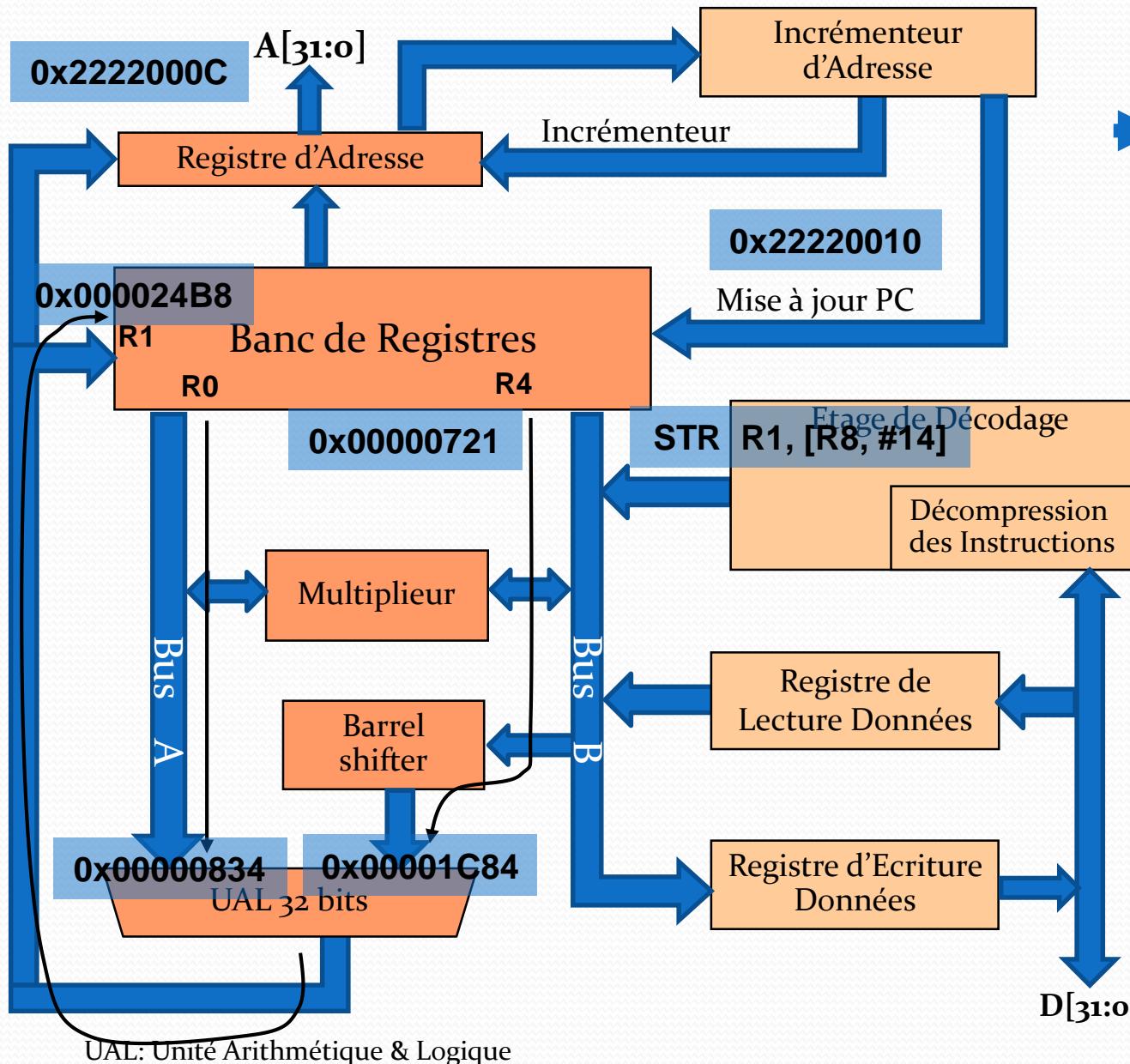
Cycle 5

- 3^{eme} cycle d'exécution de LDR
- Transfert de la donnée dans le registre-destination

STR R1, [R8, #14]

PC = 0x2222000C
 R0 = 0x00000834
 R1 =
 R4 = 0x00000721
 R8 = 0x55551000
 [0x55551010] = 0x00000834

Séquence d'Instructions : Cycle 6



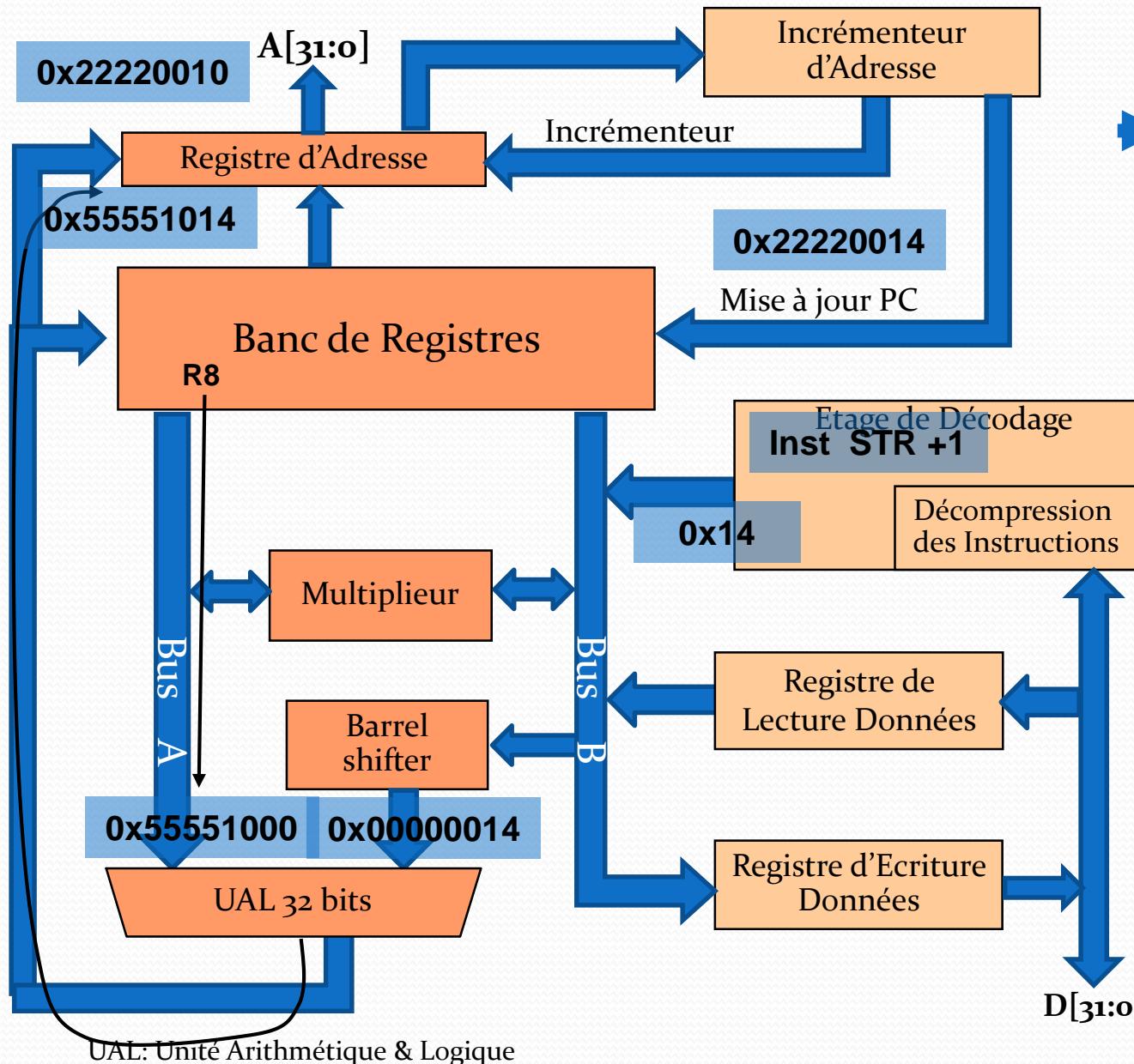
LDR R0, [R8, 0x10]
 ADD R1, R0, R4, LSL #2
 STR R1, [R8, 0x14]

Cycle 6

- Exécution de ADD
- Décodage de l'instruction STR
- Lecture de l'instruction suivant STR

PC = 0x22220010
 R0 = 0x00000834
 R1 = 0x000024B8
 R4 = 0x00000721
 R8 = 0x55551000
 $[0x55551010] = 0x00000834$

Séquence d'Instructions : Cycle 7

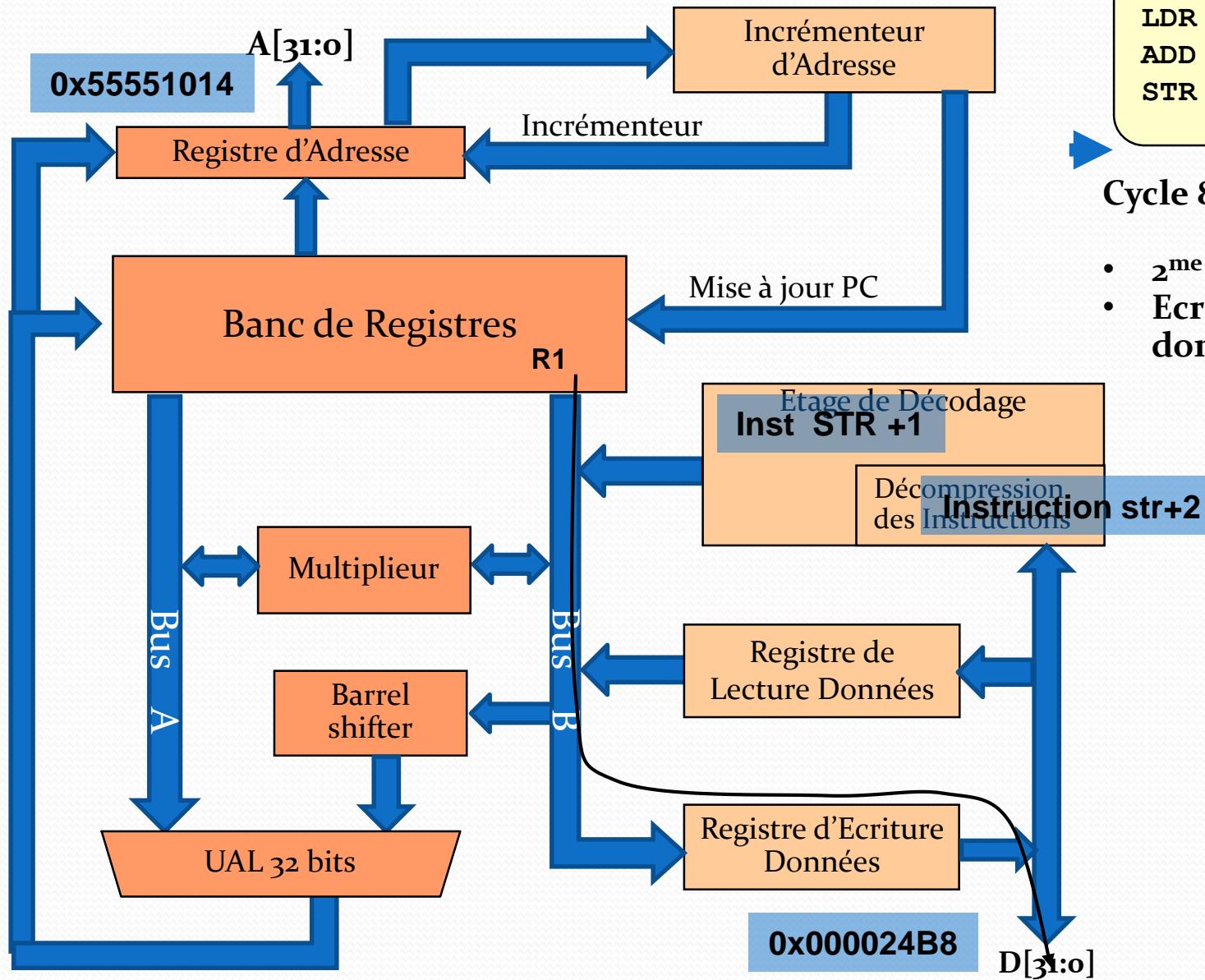


Cycle 7

- 1^{er} cycle d'exécution de STR
- Calcul de l'adresse en mémoire de données
- Décodage de l'instruction suivant STR
- Lecture de l'instruction STR+2

PC = **0x22220014**
R0 = **0x00000834**
R1 = **0x000024B8**
R4 = **0x00000721**
R8 = **0x55551000**
[0x55551010] = **0x00000834**

Séquence d'Instructions : Cycle 8



LDR R0, [R8, 0x10]
 ADD R1, R0, R4, LSL #2
 STR R1, [R8, 0x14]

Cycle 8

- 2^{me} cycle d'exécution de STR
- Ecriture en mémoire de données

PC = 0x22220014
 R0 = 0x00000834
 R1 = 0x000024B8
 R4 = 0x00000721
 R8 = 0x55551000
 [0x55551010] = 0x00000834
 [0x55551014] = 0x000024B8

UAL: Unité Arithmétique & Logique

Jeu d'instructions ARM

- On peut classer les instructions en trois grandes catégories :
 1. Traitement et manipulation des données :
 - Arithmétiques et logiques
 - Tests et comparaisons
 2. Transfert de données depuis et vers la mémoire
 3. Contrôle de flot
 - Branchements

Opérations arithmétiques et logiques

Opération sur 3 registres

OPE r_dest, r_s1, r_s2

Exemples

AND r0, r1, r2  $r0 = r1 \& r2$

ADD r5, r1, r5  $r5 = r1 + r5$

Opérations arithmétiques et logiques

Les instructions :

ADD ro, r1, r2	$ro = r1 + r2$	Addition
ADC ro, r1, r2	$ro = r1 + r2 + C$	Addition avec retenue
SUB ro, r1, r2	$ro = r1 - r2$	Soustraction
SBC ro, r1, r2	$ro = r1 - r2 - C + 1$	Soustraction
RSB ro, r1, r2	$ro = r2 - r1$	Soustraction inverse
RSC ro, r1, r2	$ro = r2 - r1 - C + 1$	Soustraction inverse avec retenue
AND ro, r1, r2	$ro = r1 \& r2$	Et binaire
ORR ro, r1, r2	$ro = r1 + r2$	Ou binaire
EOR ro, r1, r2	$ro = r1 - r2$	Ou exclusif binaire
BIC ro, r1, r2	$ro = r1 - r2 - C + 1$	Met à 0 les bites de r1 indiqués par les 1 de r2

Opérations de déplacement de données entre registres

Opération sur 2 registres

OPE r_dest, r_s1

Les instructions :

MOV r0, r1

$r0 = r1$

Déplacement

MVN r0, r1

$r0 = \sim r1$

Déplacement et négation

Opérations de décalage

Opération sur 3 registres

OPE r_dest, r_s1, r_s2

Exemples

LSL r0, r1, r2



$r0 = r1 \ll r2[7..0]$

ASR r3, r4, r5

$r3 = r4 \gg r5[7..0]$

Les instructions :

LSL	Décalage logique vers la gauche
LSR	Décalage logique vers la droite
ASL	Décalage arithmétique vers la gauche
ASR	Décalage arithmétique vers la droite
ROR	Décalage circulaire vers la droite

Remarque :

Ajouter le suffixe "S" au mnémonique de l'instruction pour mettre à jour les indicateurs (N,Z,C,V) du PSR

Exemple : ANDS or MOVS

Opérations de comparaison

Opération sur 2 registres

OPE r_s1, r_s2

Exemples

CMP ro, r1



PSR \leq ro - r1

TEQ ro, r1

PSR \leq ro (+) r1

Remarque :

Ces instruction ne modifient que les indicateurs (N,Z,C,V) du PSR, par contre le résultat n'est pas gardé

Les instructions :

CMP ro, r1 PSR \leq ro - r1 Comparer

CMN ro, r1 PSR \leq ro + r1 Comparer à l'inverse

TST ro, r1 PSR \leq ro & r1 Tester les bits indiqués par r1

TEQ ro, r1 PSR \leq ro \wedge r1 Tester l'égalité bit à bit

Opérandes Immédiats

- Un des opérandes source peut être un immédiat.
 - Un immédiat est une valeur constante qui est encodée dans une partie de l'instruction.
- On doit les procéder du symbole '#'.
 - Il peut être décimal, héxadécimal (ox...) ou octal (o).

Exemples

MOV r0, #ox20

CMP r0, #32

ADD r0, r1,#1

- En mode ARM, les instructions sont codées sur 32 bits et seulement 12 bits peuvent être utilisés pour coder l'immédiat



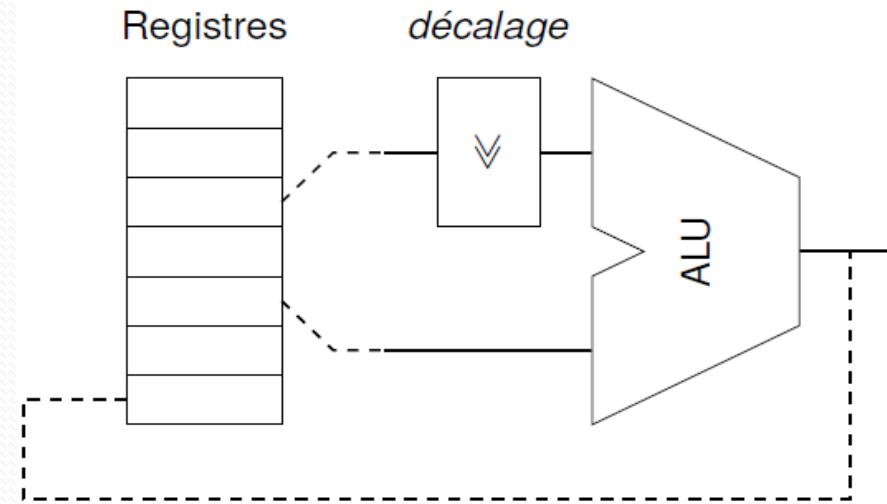
- 8 bit (o -> 0xFF)
- 4 bits pour un décalage circulaire (valeurs paires de 0 à 30).

Opérandes Immédiats

Exemples :

ADD r0, r1, #100	(0x64 << 0)
ADD r0, r1, #0xFFoo	(0xFF << 8)
MOV r0, #0x3FC	(0xFF << 2)
MOV r0, #FooooooF	(0xFF << 28)
ADD r0, r1, #0x103	Interdit !

Combiner une opération avec un décalage



- Le barrel shifter peut être utilisé en même temps que l'ALU.
- Toute opération peut être accompagnée du décalage du seconde opérande.

Exemples

ADD r0, r1, r2, LSL #4



$$r_0 = r_1 + r_2 \times 16$$

ADD r0, r1,r2, LSL r3



$$r_0 = r_1 + r_2 \times 2^{r_3}$$

LSL rd, rs, #i

équivalent à

MOV rd, rs, LSL #i

La multiplication

Les instructions :

MUL ro, r1, r2	$ro = r1 * r2$	multiplication
MLA ro, r1, r2, r3	$ro = r1 + r2 * r3$	Mult. Et accumulation
MLS ro, r1, r2, r3	$ro = r1 - r2 * r3$	Mult. Et soustraction
UMULL ro, r1, r2, r3	$\{r1, ro\} = r2 * r3$	Multi 64 bits non signée
SMULL ro, r1, r2, r3	$\{r1, ro\} = r2 * r3$	Multi 64 bits signée
UMALL ro, r1, r2, r3	$\{r1, ro\} += r2 * r3$	Multi 64 bits non signée
SMALL ro, r1, r2, r3	$\{r1, ro\} += r2 * r3$	Multi 64 bits signée

Transferts de données

- Deux instructions de transfert de données entre la mémoire et les registres
 - LDR : charger un registre avec une donnée en mémoireOn doit les procéder du symbole '#'.
STR : enregistrer la valeur du registre en mémoire

Exemples

LDR ro, [r1] (ro = RAM[r1])

STR ro, [r1] (RAM[r1] = ro)

- LDR / STR : mots de 32 bits
 - LDRH / STRH : mots de 16 bits
 - LDRB / STRB : mots de 8 bits

Modes d'adressage

- Adressage indirect
 - LDR ro, [r₁] $(ro = RAM[r_1])$
- Adressage indirect avec déplacement (Offset)
 - LDR ro, [r₁, #8] $(ro = RAM[r_1+8])$
 - LDR ro, [r₁, r₂] $(ro = RAM[r_1+r_2])$
- Adressage indirect avec déplacement et pré-incrémantation
 - LDR ro, [r₁, #8]! $(r_1=r_1 + 8 \text{ puis } ro = RAM[r_1])$
- Adressage indirect avec déplacement et post-incrémantation
 - LDR ro, [r₁], #8 $(ro = RAM[r_1] \text{ puis } r_1 = r_1 + 8)$

Transferts multiples (la pile)

- En plus des instructions LDR et STR le jeu d'instruction ARM propose les instructions LDM et STM pour les transferts multiples.
- Il existe 4 suffixes possibles pour les instructions LDM et STM
 - IA (Increment After); IB (Increment Before);
 - DA (Decrement After); DB (Decrement Before);

Exemples

LDMIA ro, {r₁, r₂, r₃}

(r₁ = RAM[ro])

(r₂ = RAM[ro + 4])

(r₃ = RAM[ro + 8])

STMIA ro, {r₁-r₃}

(RAM[ro] = r₁)

(RAM[ro+4] = r₂)

(RAM[ro+8] = r₃)

Transferts multiples (la pile)

Conventions

- Le registre r13 (sp) est le pointeur de pile (stack pointer)
- Le pointeur de pile contient l'adresse de la dernière donnée empilée
- Avant chaque empilement le pointeur de pile doit être décrémenté
- Plus simplement PUSH (empiler) et POP (dépiler)

PUSH {r1 - r5}

ou STMFD sp!, {r1 - r5}

ou STMDB sp!, {r1 - r5}

POP {r1-r5}

ou LDMFD sp!, {r1 -

r5}
r5}

ou LDMIA sp!, {r1-

Branchement

Il existe deux instructions de branchement :

- B adresse Aller à l'adresse
 - BX registre Aller à l'adresse pointé par le registre
 - BL étiquette Aller à la fonction intitulée étiquette

Exemple :

- BL triple ; plus loin dans ton code
 -
 - Triple
 - PUSH {R1,LR}
 - ADD R1, Ro, Ro
 - ADD R1, R1, Ro
 - MOV Ro, R1
 - POP {R1,LR}
 - BX LR

Exécution conditionnelle des instructions

- L'exécution des instructions peut être rendue conditionnelle en rajoutant les suffixes suivant:

EQ	Equal	$Z == 1$
NE	Not equal	$Z == 0$
CS/HS	Carry set / unsigned higher or same	$C == 1$
CC / LO	Carry clear / unsigned lower	$C == 0$
MI	Minus negative	$N == 1$
PL	Plus / positive or zero	$N == 0$
VS	Overflow	$V == 1$
VC	No overflow	$V == 0$
HI	Unsigned higher	$C == 1 \text{ and } Z == 0$
LS	Unsigned lower or same	$C == 0 \text{ and } Z == 1$
GE	Signed greater than or equal	$N == V$
LT	Signed less than	$N != V$
GT	Signed greater than	$Z == 0 \text{ and } N == V$
LE	Signed less than or equal	$Z == 1 \text{ and } N != V$

Exécution conditionnelle des instructions

Exemples

CMP r0, r1

comparer r0 et r1

SUBGE r0, r0, r1

si $r0 \geq r1$ alors $r0 = r0 - r1$

SUBLT r0, r1, r0

si $r0 < r1$ alors $r0 = r1 - r0$

SUBS r0, r1, r2

$r0 = r1 - r2$

BEQ adress

aller à l'adresse si le résultat est nul

Mode Thumb

- Jeu d'instructions codé sur 16 bits
 - Optimisé pour la densité de code à partir de code écrit en langage C
 - Augmente les performances pour des espaces mémoires réduits
 - Sous ensemble des fonctionnalités du jeu d'instructions ARM
- Le cœur a deux modes d'exécution : ARM et Thumb
 - On passe d'un mode à l'autre en utilisant l'instruction **BX**

31

0

ADDS r2, r2, #1

Pour la plupart des instructions générées par le compilateur:

- L'exécution conditionnelle n'est pas utilisée
- Les registres Source et Destination sont identiques
- Seul les premiers registres sont utilisés
- Les constantes sont de taille limitée
- Le registre à décalage n'est pas utilisé au sein d'une même instruction

15

0

ADD r2, #1

Instruction en mode Thumb (16 bits)