

# Introduction à l'architecture ARM



# Petit historique

- Tandis qu'Intel produit des microprocesseurs 8086 et 80286, des employés d'une compagnie anglaise inspirés par les travaux de l'heure travaillent à la conception d'un ordinateur
- En 1985, la compagnie ACORN produit un premier processeur qui résulte de ces travaux: le Acorn RISC Machine (ARM)
- La complexité du ARM est réduite par choix et par faute de moyens et le jeu d'instructions est simplifié.
- Alors que le 8086 gagne en popularité, ACORN et Apple collaborent pour fonder la compagnie ARM Ltd en 1990
- ARM et Intel feront évoluer leurs produits en parallèle dans des marchés différents

# ARM

- ARM Holdings:
  - développe des architectures de micro-processeurs et des jeux d'instructions
  - ne construit aucun micro-processeur comme tel! La compagnie licencie la technologie à d'autres qui les fabriquent à leur façon
    - clients: Apple, Nvidia, Samsung, Texas Instruments, etc...
- Micro-processeurs ARM
  - Supportent 32 et 64 bits
  - L'architecture **la** plus utilisée au monde
    - 100 milliards de processeurs produits en 2017

# Processeurs ARM

- Plusieurs versions de processeurs, utilisées partout!
  - ARM7TDMI(-S): Nintendo DS, Lego NXT
  - ARM946E-S: Canon 5D Mark ii (caméra)
  - ARM1176JZ(F)-S: Raspberry Pi
  - Cortex-A9: Apple iPhone 4S, iPad2
  - Cortex-A15: Nexus 10
- Beaucoup d'autres!
  - [http://en.wikipedia.org/wiki/List\\_of\\_applications\\_of\\_ARM\\_cores](http://en.wikipedia.org/wiki/List_of_applications_of_ARM_cores)



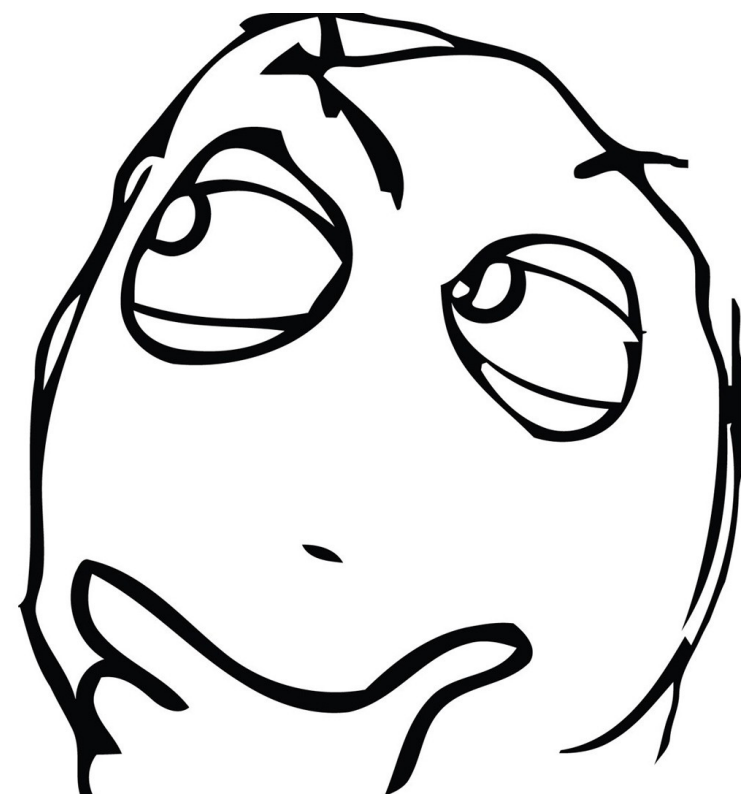


# Architecture des ARM (32 bits)

- Dans le cours: ARM7TDMI (architecture 32 bits populaire)
  - Registres de 32 bits
  - Instructions de 32 bits
  - Adresses de 32 bits
- Architecture RISC pour laquelle tout passe par des registres
- Accès à combien de mémoire?
  - $2^{32}$  octets (4GB) de mémoire

# 3 Affaires à Retenir Mentalement™ (ARM)

- ARM™ #1 : Organisation de la mémoire
- ARM™ #2 : Instructions et PC
- ARM™ #3 : Registres



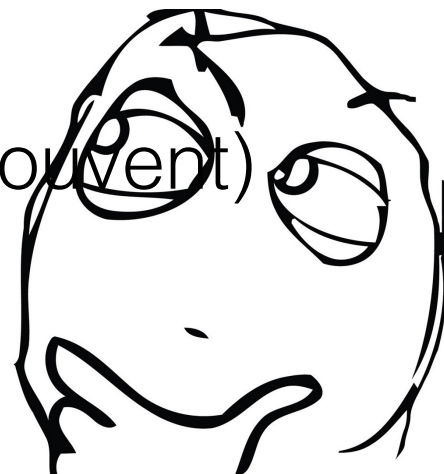
# ARM™ #1: Organisation de la mémoire

## TP1

## ARM

- Taille des emplacements en mémoire: **16** bits
  - Chaque emplacement de 16 bits possède une adresse
- Taille du bus de données et des registres: **16** bits
- Taille d'une instruction: **16** bits
- Pas de *endian-ness*

- Taille des emplacements en mémoire: **8** bits
  - Chaque emplacement de 8 bits possède une adresse
- Taille du bus de données et des registres: **32** bits
- Taille d'une instruction: **32** bits
- *Little endian* (le plus souvent)



# Question

Comment stocker un mot de 32 bits (4 octets) à l'adresse 0x0, sachant que chaque adresse stocke 8 bits (1 octet)?

exemple: 0x12345678?

# Question

Comment stocker un mot de 32 bits (4 octets) à l'adresse 0x0, sachant que chaque adresse stocke 8 bits (1 octet)?

exemple: 0x12345678?

Nous allons avoir besoin de 4 adresses au total, soit 0x0, 0x1, 0x2 et 0x3.

| Adresse (A) | A+0 | A+1 | A+2 | A+3 |
|-------------|-----|-----|-----|-----|
| 0x0         | 78  | 56  | 34  | 12  |
| 0x4         |     |     |     |     |

OU

| Adresse (A) | A+0 | A+1 | A+2 | A+3 |
|-------------|-----|-----|-----|-----|
| 0x0         | 12  | 34  | 56  | 78  |
| 0x4         |     |     |     |     |

# *Big vs Little Endian*

- La taille minimum d'une donnée stockée en mémoire est habituellement 1 octet
- Les données ayant plusieurs octets peuvent être stockées de 2 façons:
  - *Little endian* : **l'octet le moins significatif** est placé à la **plus petite** adresse dans la mémoire (ex: Intel x86)
  - *Big endian* : **l'octet le moins significatif** est placé à la **plus haute** adresse dans la mémoire (ex: Motorola 68000)



# *Big vs Little Endian*

Comment stocker un mot de 32 bits (4 octets) à l'adresse 0x0?  
exemple: 0x12345678?

*Little Endian*

| Adresse (A) | A+0 | A+1 | A+2 | A+3 |
|-------------|-----|-----|-----|-----|
| 0x0         | 78  | 56  | 34  | 12  |
| 0x4         |     |     |     |     |

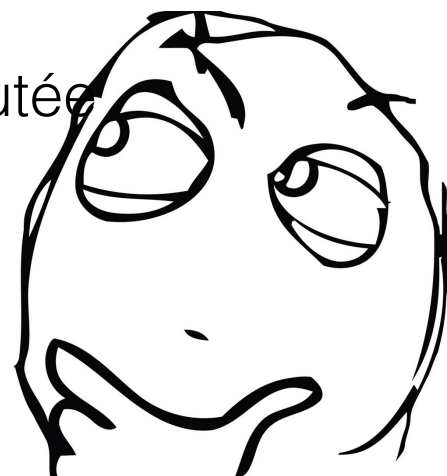
*Big Endian*

| Adresse (A) | A+0 | A+1 | A+2 | A+3 |
|-------------|-----|-----|-----|-----|
| 0x0         | 12  | 34  | 56  | 78  |
| 0x4         |     |     |     |     |



# ARM™ #2: Instructions et PC

- Combien d'adresses mémoire a-t-on besoin pour stocker une instruction?
  - chaque octet possède une adresse
  - une instruction possède 4 octets (32 bits)
  - donc une instruction nécessite 4 adresses mémoire
- 2 considérations importantes sur PC:
  - après chaque exécution, de combien doit-on incrémenter PC?
    - $PC = PC + 4$
  - qu'est-ce que contient PC?
    - PC = adresse de deux instructions plus loin que l'instruction exécutée
    - PC = adresse de l'instruction exécutée **+ 8**



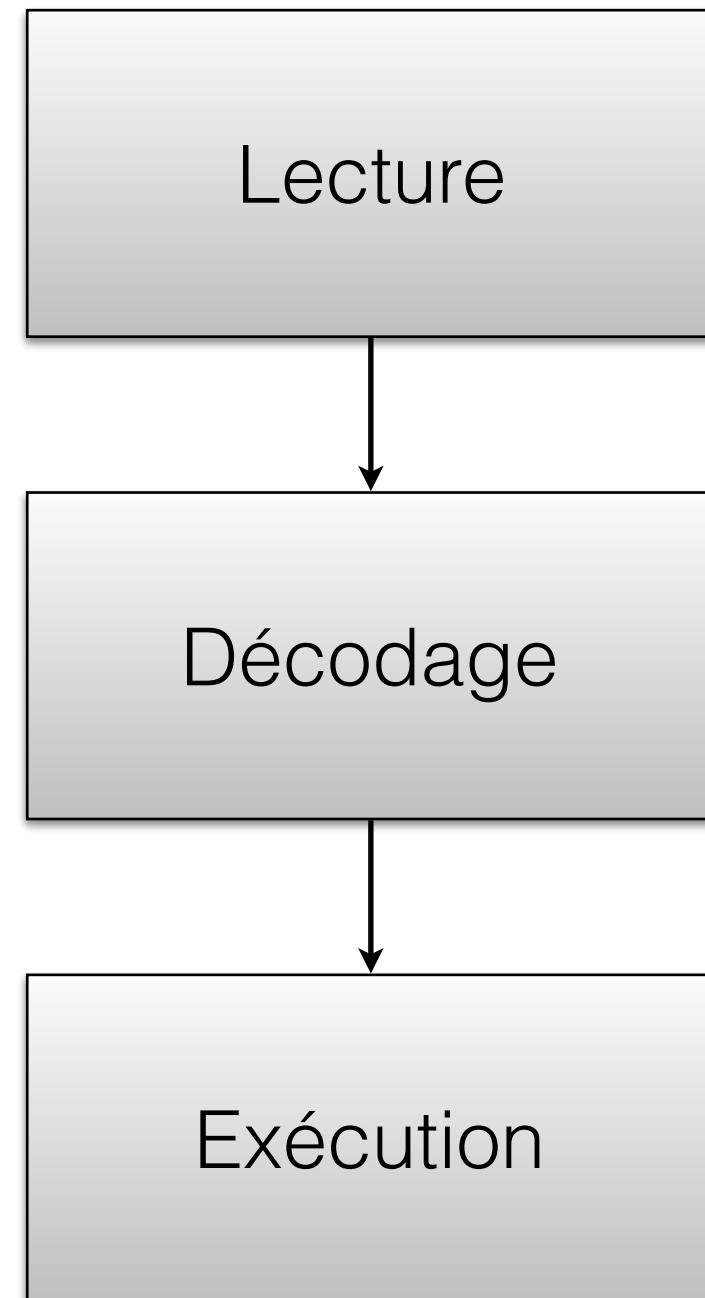


# Pipelines

- Début des années 1990
- Idée: séparer l'architecture en deux unités séparées, pouvant être exécutées simultanément
- Comme dans un restaurant:
  - *Fetch-decode* : les serveurs vont prendre les commandes des clients aux tables
  - *Execute* : les cuisiniers préparent les repas

# Pipeline ARM

- Le pipeline ARM est divisé en 3
- Que contient PC?
  - L'adresse de la prochaine instruction **à être lue**
- Donc, lors de l'exécution d'une instruction, PC indique l'adresse de **deux instructions** plus loin!



# Pipeline ARM

- Le pipeline ARM est divisé en 3
- Que contient PC?
  - L'adresse de la prochaine instruction **à être lue**
- Donc, lors de l'exécution d'une instruction, PC indique l'adresse de **deux instructions** plus loin!

|           |   | Adresse | Instruction    |
|-----------|---|---------|----------------|
| exécution | → | 0x0     | MOV R0, #0x40  |
| décodage  | → | 0x4     | LDR R0, [R0]   |
| lecture   | → | 0x8     | ADD R0, R1, R2 |
|           |   | 0xC     | ...            |

Donc, PC contient l'adresse de l'instruction courante (exécutée) **+ 8!**

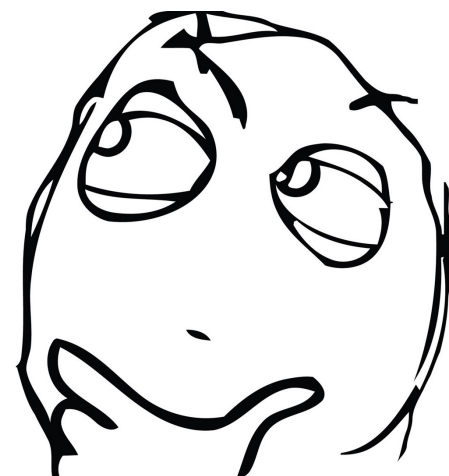
# ARM™ #3: Registres

- 16 registres de 32 bits:
  - R0 à R12: registres généraux
  - R13 à R15: registres spécifiques (ayant des fonctionnalités pré-établies)
    - R13: Pointeur de pile (*Stack Pointer* ou SP)
    - R14: Registre de liens (*Link Register* ou LR)
    - R15: Compteur de programme (*Program Counter* ou PC)
- 1 registre d'état, le *Current Program Status Register* (CPSR)
  - mémorise les résultats d'opérations arithmétiques
  - stocke le « mode d'exécution » (on y reviendra bientôt!)

## Note

Il est permis de se servir des registres spécifiques comme des registres généraux. Il faut cependant **faire très attention**, car cela peut affecter le comportement du programme de façon indésirée!

Nous vous recommandons donc **d'utiliser R0 à R12** pour vos calculs.



# 3 Affaires à Retenir Mentalement™ (ARM)

- Organisation de la mémoire
  - Chaque octet possède une adresse
  - Mots stockés en *little endian*
- Instructions et PC
  - $PC = PC + 4$
  - PC indique l'adresse de l'instruction courante **+ 8**
- Registres
  - 16 registres
    - 13 registres généraux (R0 à R12)
    - 3 registres spécifiques: SP, LR, PC
  - 1 registre d'état (CPSR)

