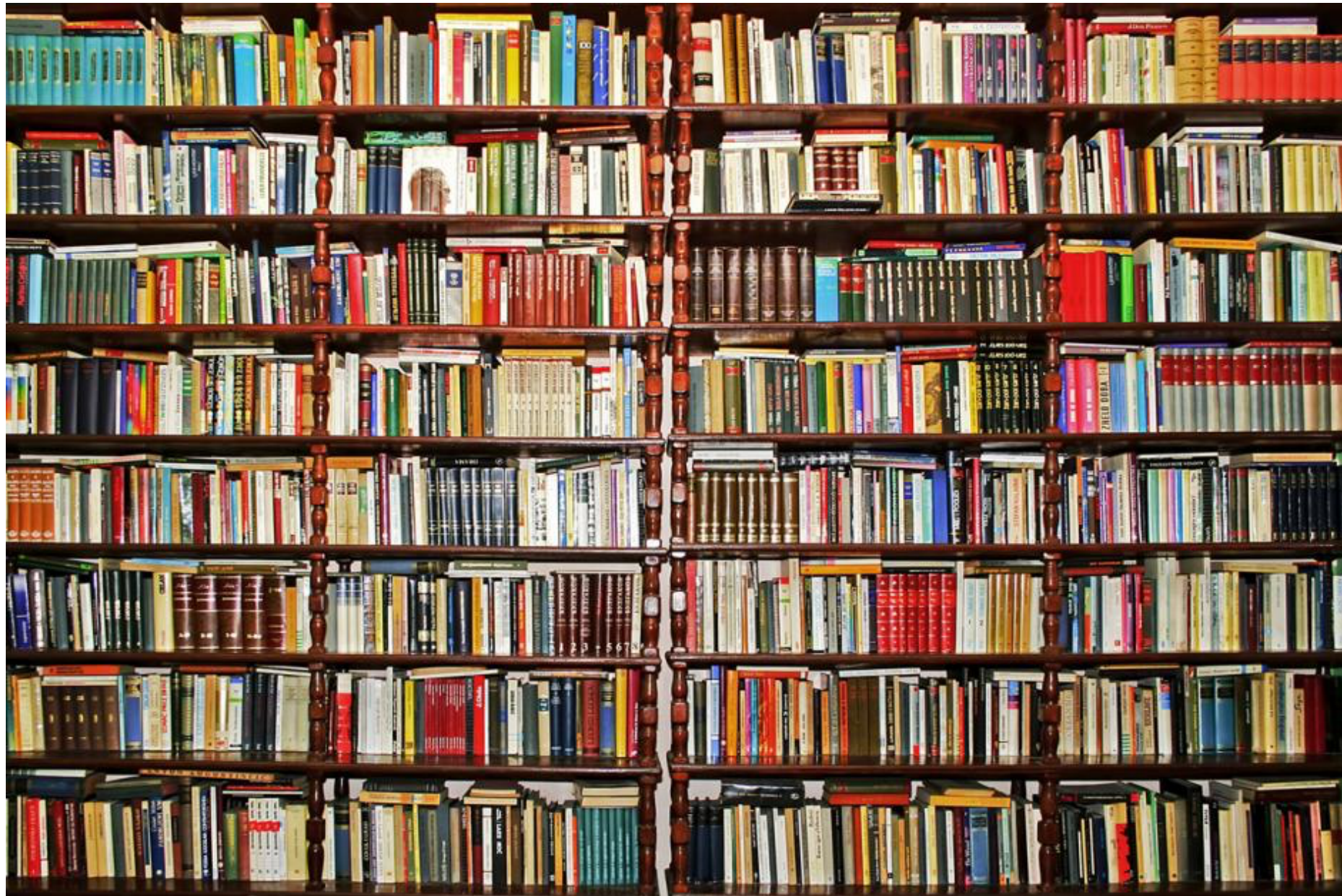


# Introduction à l'assembleur ARM: organisation de la mémoire



GIF-1001 Ordinateurs: Structure et Applications  
Jean-François Lalonde

# *Cheatsheet*: du TP1 à l'ARM

Simulateur du TP1

```
MOV Rd, Rs
MOV Rd, Const

ADD Rd, Rs
ADD Rd, Const

SUB Rd, Rs
SUB Rd, Const

LDR Rd, [Rs]
STR Rd, [Rs]

JZE Rc, Const
JZE Rc, Rs
```



ARM

```
MOV Rd, Rs
MOV Rd, Const
```

# *Cheatsheet*: du TP1 à l'ARM

Simulateur du TP1

```
MOV Rd, Rs
MOV Rd, Const

ADD Rd, Rs
ADD Rd, Const

SUB Rd, Rs
SUB Rd, Const

LDR Rd, [Rs]
STR Rd, [Rs]

JZE Rc, Const
JZE Rc, Rs
```



ARM

```
MOV Rd, Rs
MOV Rd, Const

LDR Rd, [Rs]
STR Rd, [Rs]
```

# *Cheatsheet*: du TP1 à l'ARM

Simulateur du TP1

```
MOV Rd, Rs
MOV Rd, Const

ADD Rd, Rs
ADD Rd, Const

SUB Rd, Rs
SUB Rd, Const

LDR Rd, [Rs]
STR Rd, [Rs]

JZE Rc, Const
JZE Rc, Rs
```



ARM

```
MOV Rd, Rs
MOV Rd, Const

LDR Rd, [Rs]
STR Rd, [Rs]

...nous y reviendrons!
...nous y reviendrons!
```



# Cheatsheet: du TP1 à l'ARM

## Simulateur du TP1

```
MOV Rd, Rs
MOV Rd, Const

ADD Rd, Rs
ADD Rd, Const

SUB Rd, Rs
SUB Rd, Const

LDR Rd, [Rs]
STR Rd, [Rs]

JZE Rc, Const
JZE Rc, Rs
```

```
Rd ← Rd + Rs
Rd ← Rd + Const

Rd ← Rd - Rs
Rd ← Rd - Const
```

## ARM

```
MOV Rd, Rs
MOV Rd, Const

ADD Rd, Ra, Rs
ADD Rd, Ra, Const

SUB Rd, Ra, Rs
SUB Rd, Ra, Const

LDR Rd, [Rs]
STR Rd, [Rs]

...nous y reviendrons!
...nous y reviendrons!
```

```
Rd ← Ra + Rs
Rd ← Ra + Const

Rd ← Ra - Rs
Rd ← Ra - Const
```

# Addition en assembleur ARM

- Écrivons un programme qui additionne deux valeurs stockées en mémoire:
  - La première valeur est à une adresse de base (ex: 0x40 pour le TP1, 0x1000 pour l'ARM)
  - La seconde valeur est à l'adresse mémoire suivante (après la première valeur).
  - Le résultat est écrit dans l'adresse mémoire suivante (après la seconde valeur).

Le pourquoi du 0x1000 vient dans qq minutes...

## Simulateur du TP1

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #1
LDR R2, [R0]
ADD R1, R2
ADD R0, #1
STR R1, [R0]
```

## Explications

```
Adresse de la première valeur
Lecture de la première valeur dans R0
Adresse de la deuxième valeur
Lecture de la deuxième valeur dans R1
R1 = R1 + R2
Adresse du résultat
Écriture du résultat en mémoire
```

## ARM

```
MOV R0, #0x1000
LDR R1, [R0]
ADD R0, R0, #4
LDR R2, [R0]
ADD R1, R1, R2
ADD R0, R0, #4
STR R1, [R0]
```

**Question:** Pourquoi #4 plutôt que #1?

# Addition en assembleur ARM

- Écrivons un programme qui additionne deux valeurs stockées en mémoire:
  - La première valeur est à une adresse de base (ex: 0x40 pour le TP1, 0x1000 pour l'ARM)
  - La seconde valeur est à l'adresse mémoire suivante (après la première valeur).
  - Le résultat est écrit dans l'adresse mémoire suivante (après la seconde valeur).

Le pourquoi du 0x1000 vient dans qq minutes...

## Simulateur du TP1

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #1
LDR R2, [R0]
ADD R1, R2
ADD R0, #1
STR R1, [R0]
```

## Explications

```
Adresse de la première valeur
Lecture de la première valeur dans R0
Adresse de la deuxième valeur
Lecture de la deuxième valeur dans R1
R1 = R1 + R2
Adresse du résultat
Écriture du résultat en mémoire
```

## ARM

```
MOV R0, #0x1000
LDR R1, [R0]
ADD R0, R0, #4
LDR R2, [R0]
ADD R1, R1, R2
ADD R0, R0, #4
STR R1, [R0]
```

**Question:** Pourquoi #4 plutôt que #1?

On « travaille » sur 32 bits à la fois, et chaque adresse stocke 8 bits. Donc chaque élément occupe 4 adresses mémoire.

# Démonstration

(addition)



# Exemple de programme sur le simulateur

```
SECTION INTVEC
```

```
B main
```

**Section de la  
table des vecteurs d'interruption**  
(plus de détails dans 3 semaines!)

```
SECTION CODE
```

```
main
```

```
MOV R0, #0x1000 ; Adresse de la première valeur  
LDR R1, [R0] ; Lecture de la première valeur dans R0  
ADD R0, R0, #4 ; Adresse de la deuxième valeur  
LDR R2, [R0] ; Lecture de la deuxième valeur dans R1  
ADD R1, R1, R2 ; R1 = R1 + R2  
ADD R0, R0, #4 ; Adresse du résultat  
STR R1, [R0] ; Écriture du résultat en mémoire
```

```
fin
```

```
B fin
```

**Section du code principal**

Habituellement ici que l'on écrit  
notre code.

```
SECTION DATA
```

```
; Valeurs stockées en mémoire
```

```
premiereValeur ASSIGN32 0x1
```

```
deuxiemeValeur ASSIGN32 0x2
```

```
; Résultat (on ne connaît pas sa valeur a priori)
```

```
resultat ALLOC32 1
```

**Section des données**

Habituellement ici que l'on place  
des données utilisées par le code.

# Adresses de chaque section

0x0: Adresse de début de la section INTVEC

0x7F: Adresse de fin de la section INTVEC

0x80: Adresse de début de la section CODE

0x0FFF: Adresse de fin de la section CODE

0x1000: Adresse de début de la section DATA

...

**Section de la  
table des vecteurs d'interruption**  
(plus de détails dans 2 semaines!)

## **Section du code principal**

Habituellement ici que l'on écrit  
notre code.

## **Section des données**

Habituellement ici que l'on place  
des données utilisées par le code.

# Contenu de la mémoire

Code

```
SECTION INTVEC

B main

SECTION CODE

main

MOV R0, #0x1000
LDR R1, [R0]
ADD R0, R0, #4
LDR R2, [R0]
ADD R1, R1, R2
ADD R0, R0, #4
STR R1, [R0]

fin
B fin

SECTION DATA

; Valeurs stockées en mémoire
premiereValeur ASSIGN32 0x1
deuxiemeValeur ASSIGN32 0x2

; Résultat
resultat ALLOC32 1
```

Mémoire

0x00	1E	00	00	EA											
0x10															
0x20															

⋮

0x80	01	0A	A0	E3	00	10	90	E5	04	00	80	E2	00	20	90	E5
0x90	02	10	81	E0	04	00	80	E2	00	10	80	ED	FE	FF	FF	EA
0xA0																

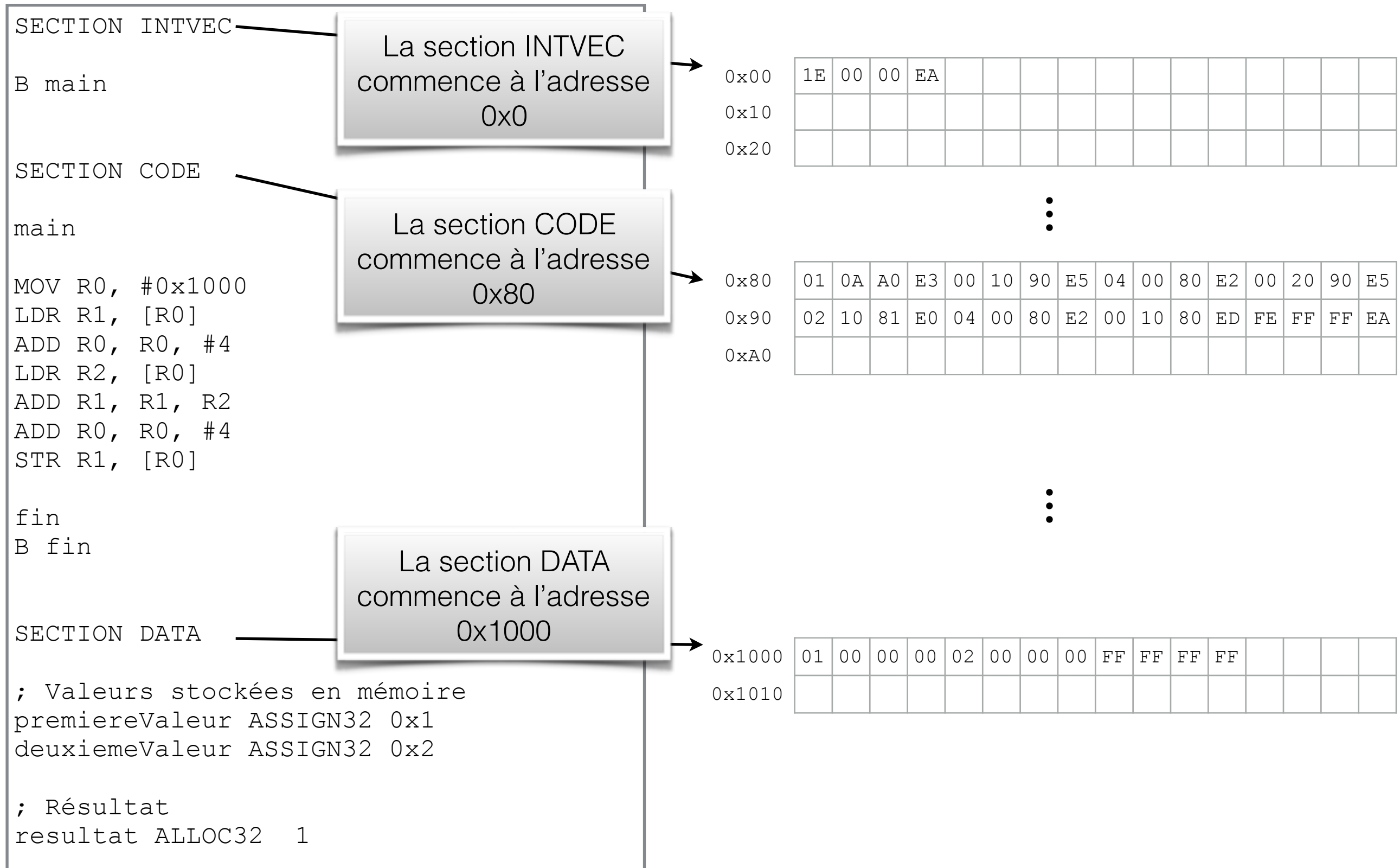
⋮

0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF				
0x1010																

# Contenu de la mémoire

Code

Mémoire



# Contenu de la mémoire

Code

```
SECTION INTVEC

B main

SECTION CODE

main

MOV R0, #0x1000
LDR R1, [R0]
ADD R0, R0, #4
LDR R2, [R0]
ADD R1, R1, R2
ADD R0, R0, #4
STR R1, [R0]

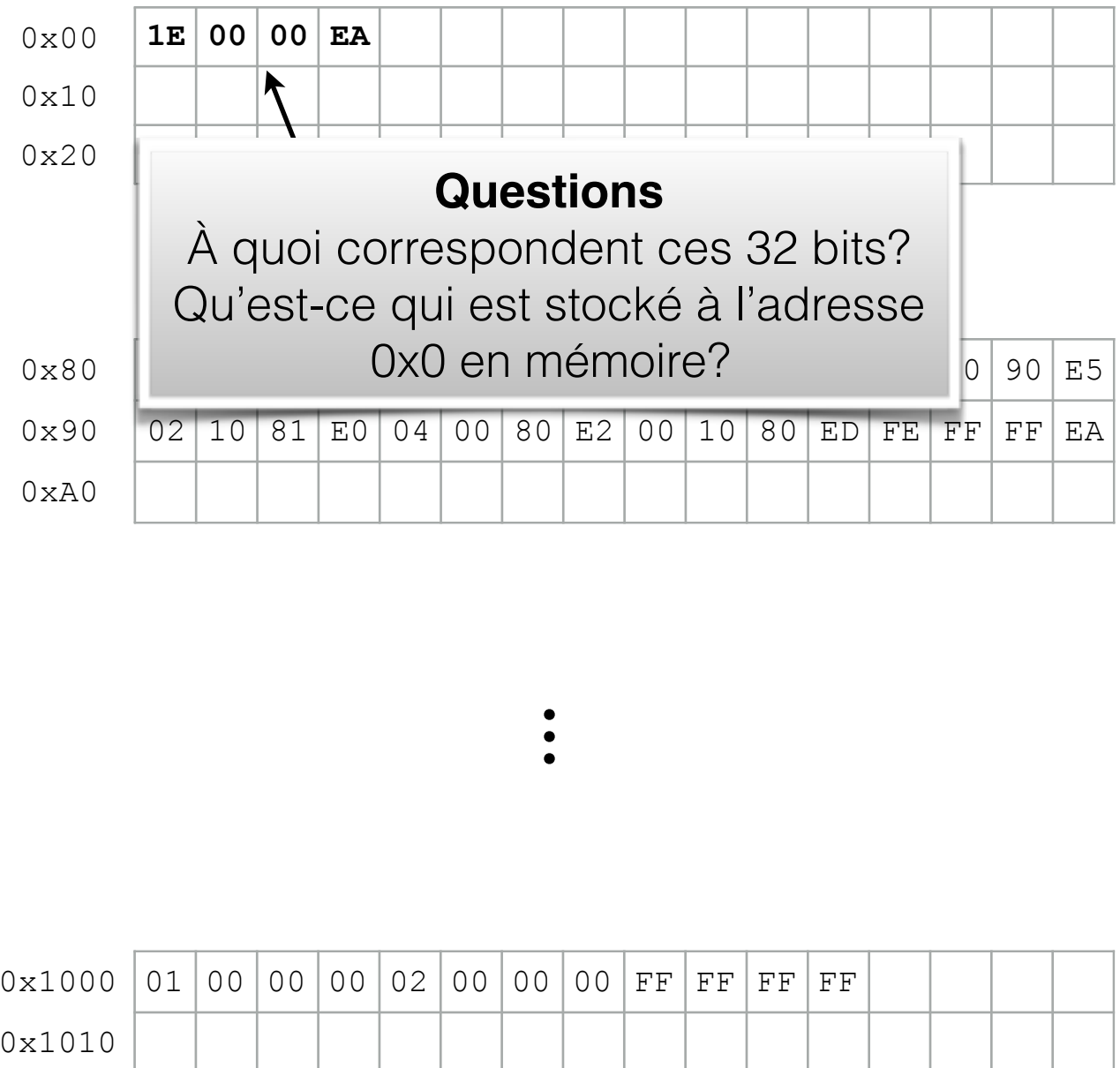
fin
B fin

SECTION DATA

; Valeurs stockées en mémoire
premiereValeur ASSIGN32 0x1
deuxiemeValeur ASSIGN32 0x2

; Résultat
resultat ALLOC32 1
```

Mémoire







# Contenu de la mémoire

Code

```
SECTION INTVEC

B main

SECTION CODE

main

MOV R0, #0x1000
LDR R1, [R0]
ADD R0, R0, #4
LDR R2, [R0]
ADD R1, R1, R2
ADD R0, R0, #4
STR R1, [R0]

fin
B fin

SECTION DATA

; Valeurs stockées en mémoire
premiereValeur ASSIGN32 0x1
deuxiemeValeur ASSIGN32 0x2

; Résultat
resultat ALLOC32 1
```

Mémoire

0x00	1E	00	00	EA											
0x10															
0x20															

⋮

0x80	<b>01</b>	<b>0A</b>	<b>A0</b>	<b>E3</b>	00	10	90	E5	04	00	80	E2	00	20	90	E5
0x90	02	10	8	E0	04	00	80	E2	00	10	80	ED	FE	FF	FF	EA
0xA0																

**Questions**  
 À quoi correspondent ces 32 bits?  
 Qu'est-ce qui est stocké à l'adresse  
 0x80 en mémoire?

0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF				
0x1010																

# Contenu de la mémoire

SECTION  
B main

Ici, main est une étiquette. Les étiquettes ne sont pas stockées en mémoire. Nous verrons leur utilité au prochain cours!

SECTION CODE

```
main
MOV R0, #0x1000
LDR R1, [R0]
ADD R0, R0, #4
LDR R2, [R0]
ADD R1, R1, R2
ADD R0, R0, #4
STR R1, [R0]
```

fin  
B fin

Le premier élément de la section CODE, soit l'instruction MOV R0, #0x1000.

SECTION

```
; Valeurs stockées en mémoire
premiereValeur ASSIGN32 0x1
deuxiemeValeur ASSIGN32 0x2

; Résultat
resultat ALLOC32 1
```

Mémoire

0x00	1E	00	00	EA																
0x10																				
0x20																				

⋮

0x80	01	0A	A0	E3	00	10	90	E5	04	00	80	E2	00	20	90	E5				
0x90	02	10	80	E0	04	00	80	E2	00	10	80	ED	FE	FF	FF	EA				
0xA0																				

## Questions

À quoi correspondent ces 32 bits?  
Qu'est-ce qui est stocké à l'adresse 0x80 en mémoire?

0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF								
0x1010																				

# Contenu de la mémoire

Code

```
SECTION INTVEC
```

```
B main
```

```
SECTION CODE
```

```
main
```

```
MOV R0, #0x1000
```

```
LDR R1, [R0]
```

```
ADD R0, R0, #4
```

```
LDR R2, [R0]
```

```
ADD R1, R1, R2
```

```
ADD R0, R0, #4
```

```
STR R1, [R0]
```

```
fin
```

```
B fin
```

```
SECTION DATA
```

```
; Valeurs stockées en mémoire
```

```
premiereValeur ASSIGN32 0x1
```

```
deuxiemeValeur ASSIGN32 0x2
```

```
; Résultat
```

```
resultat ALLOC32 1
```

Mémoire

0x00	1E	00	00	EA													
0x10																	
0x20																	

⋮

0x80	01	0A	A0	E3	00	10	90	E5	04	00	80	E2	00	20	90	E5
0x90	02	10	81	E0	04	00	80	E2	00	10	80	ED	FE	FF	FF	EA
0xA0																

⋮

0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF				
0x1010																

**Question**  
Où cette instruction est-elle stockée en mémoire?

# Contenu de la mémoire

Code

```
SECTION INTVEC
```

```
B main
```

```
SECTION CODE
```

```
main
```

```
MOV R0, #0x1000
```

```
LDR R1, [R0]
```

```
ADD R0, R0, #4
```

```
LDR R2, [R0]
```

```
ADD R1, R1, R2
```

```
ADD R0, R0, #4
```

```
STR R1, [R0]
```

```
fin
```

```
B fin
```

```
SECTION DATA
```

```
; Valeurs stockées en mémoire
```

```
premiereValeur ASSIGN32 0x1
```

```
deuxiemeValeur ASSIGN32 0x2
```

```
; Résultat
```

```
resultat ALLOC32 1
```

Mémoire

0x00	1E	00	00	EA															
0x10																			
0x20																			

⋮

0x80	01	0A	A0	E3	00	10	90	E5	04	00	80	E2	00	20	90	E5			
0x90	<b>02</b>	<b>10</b>	<b>81</b>	<b>E0</b>	04	00	80	E2	00	10	80	ED	FE	FF	FF	EA			
0xA0																			

**Question**  
Où cette instruction est-elle stockée en mémoire?

Il s'agit du 5e élément (de 32 bits, ou 4 octets) de la section CODE. Son adresse est donc  $0x80 + 4 * 4 = 0x80 + 0x10 = 0x90$

0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF							
0x1010																			

# Contenu de la mémoire

Code

```
SECTION INTVEC

B main

SECTION CODE

main

MOV R0, #0x1000
LDR R1, [R0]
ADD R0, R0, #4
LDR R2, [R0]
ADD R1, R1, R2
ADD R0, R0, #4
STR R1, [R0]

fin
B fin

SECTION DATA

; Valeurs stockées en mémoire
premiereValeur ASSIGN32 0x1
deuxiemeValeur ASSIGN32 0x2

; Résultat
resultat ALLOC32 1
```

Mémoire

0x00	1E	00	00	EA											
0x10															
0x20															

⋮

0x80	01	0A	A0	E3	00	10	90	E5	04	00	80	E2	00	20	90	E5
0x90	02	10	81	E0	04	00	80	E2	00	10	80	ED	FE	FF	FF	EA
0xA0																

## Questions

À quoi correspondent ces 32 bits?  
Qu'est-ce qui est stocké à l'adresse  
0x1000 en mémoire?

0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF				
0x1010																

# Contenu de la mémoire

Code

```
SECTION INTVEC
```

```
B main
```

```
SECTION CODE
```

```
main
```

```
MOV R0, #0x1000
```

```
LDR R1, [R0]
```

```
ADD R0, R0, #4
```

```
LDR R2, [R0]
```

```
ADD R1, R1, R2
```

```
ADD R0,
```

```
STR R1,
```

```
fin
```

```
B fin
```

Le premier élément de la section DATA,  
soit notre première valeur à additionner  
premiereValeur ASSIGN32 0x1

```
SECTION DATA
```

```
; Valeurs stockées en mémoire
```

```
premiereValeur ASSIGN32 0x1
```

```
deuxiemeValeur ASSIGN32 0x2
```

```
; Résultat
```

```
resultat ALLOC32 1
```

Mémoire

0x00	1E	00	00	EA																
0x10																				
0x20																				

⋮

0x80	01	0A	A0	E3	00	10	90	E5	04	00	80	E2	00	20	90	E5				
0x90	02	10	81	E0	04	00	80	E2	00	10	80	ED	FE	FF	FF	EA				
0xA0																				

## Questions

À quoi correspondent ces 32 bits?  
Qu'est-ce qui est stocké à l'adresse  
0x1000 en mémoire?

0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF								
0x1010																				



# Démonstration (addition)

Cette fois, analysons le contenu de la mémoire.