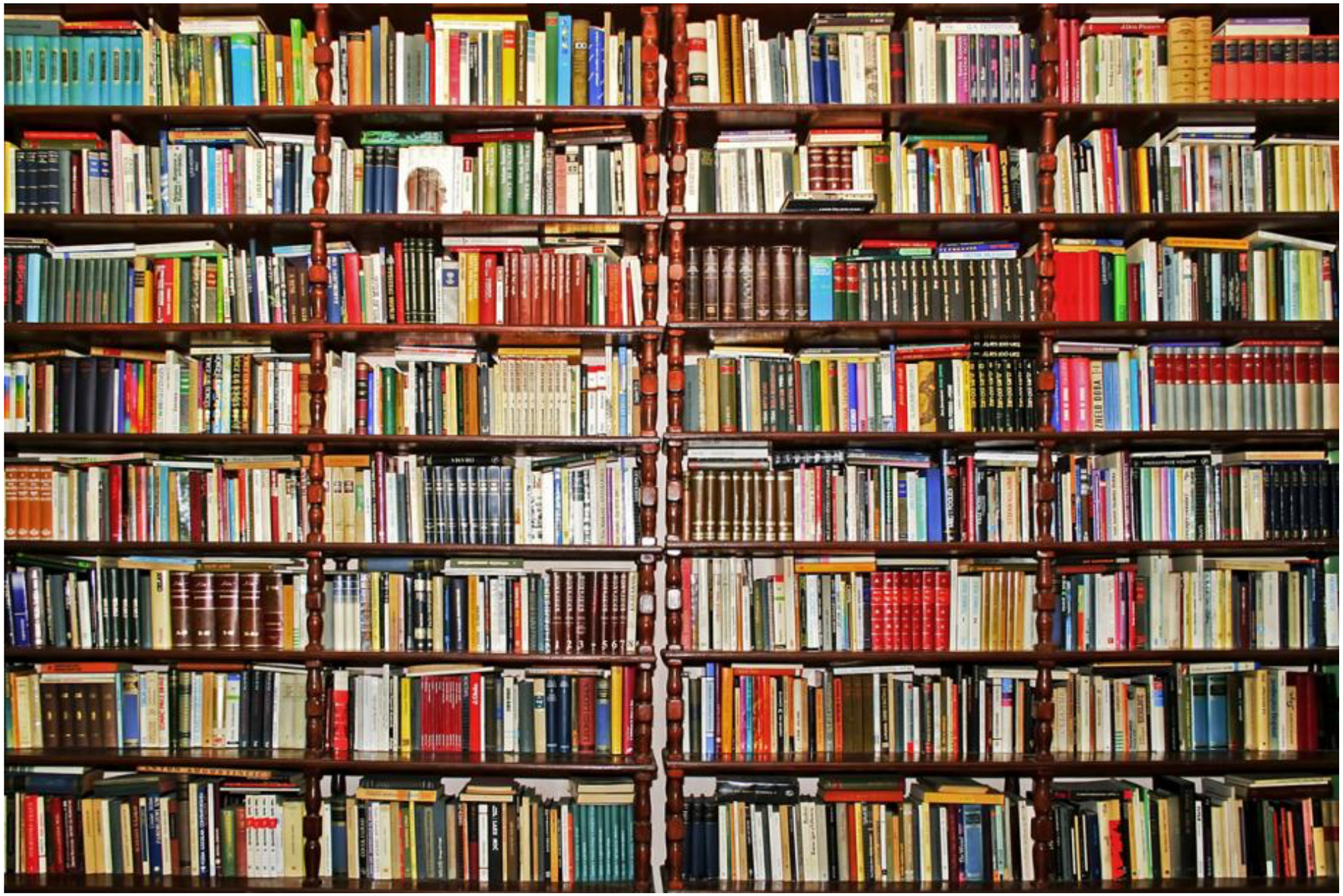
Introduction à l'assembleur ARM:
organisation de la mémoire



GIF-1001 Ordinateurs: Structure et Applications, Hiver 2019

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	Explications	ARM
`MOV R0, #0x40`	`Adresse de la première valeur`	`MOV R0, #0x1000`
`LDR R1, [R0]`	`Lecture de la première valeur dans R0`	`LDR R1, [R0]`
`ADD R0, #1`	`Adresse de la deuxième valeur`	`ADD R0, R0, #4`
`LDR R2, [R0]`	`Lecture de la deuxième valeur dans  R1`	`LDR R2, [R0]`
`ADD R1, R2`	`R1 = R1 + R2`	`ADD R1, R1, R2`
`ADD R0, #1`	`Adresse du résultat`	`ADD R0, R0, #4`
`STR R1, [R0]`	`Écriture du résultat en mémoire`	`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	Explications	ARM
`MOV R0, #0x40`	`Adresse de la première valeur`	`MOV R0, #0x1000`
`LDR R1, [R0]`	`Lecture de la première valeur dans R0`	`LDR R1, [R0]`
`ADD R0, #1`	`Adresse de la deuxième valeur`	`ADD R0, R0, #4`
`LDR R2, [R0]`	`Lecture de la deuxième valeur dans R1`	`LDR R2, [R0]`
`ADD R1, R2`	`R1 = R1 + R2`	`ADD R1, R1, R2`
`ADD R0, #1`	`Adresse du résultat`	`ADD R0, R0, #4`
`STR R1, [R0]`	`Écriture du résultat en mémoire`	`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 connait 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`
	Section de la table des vecteurs d'interruption (plus de détails dans 2 semaines!)

`0x80: Adresse de début de la section CODE`
 	
`0x0FFF: Adresse de fin de la section CODE`
	Section du code principal
	
	Habituellement ici que l'on écrit notre code.

`0x1000: Adresse de début de la section DATA`
 	
`...`
	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

```
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
```

La section INTVEC commence à l'adresse 0x0

La section CODE commence à l'adresse 0x80

La section DATA commence à l'adresse 0x1000

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																

⋮

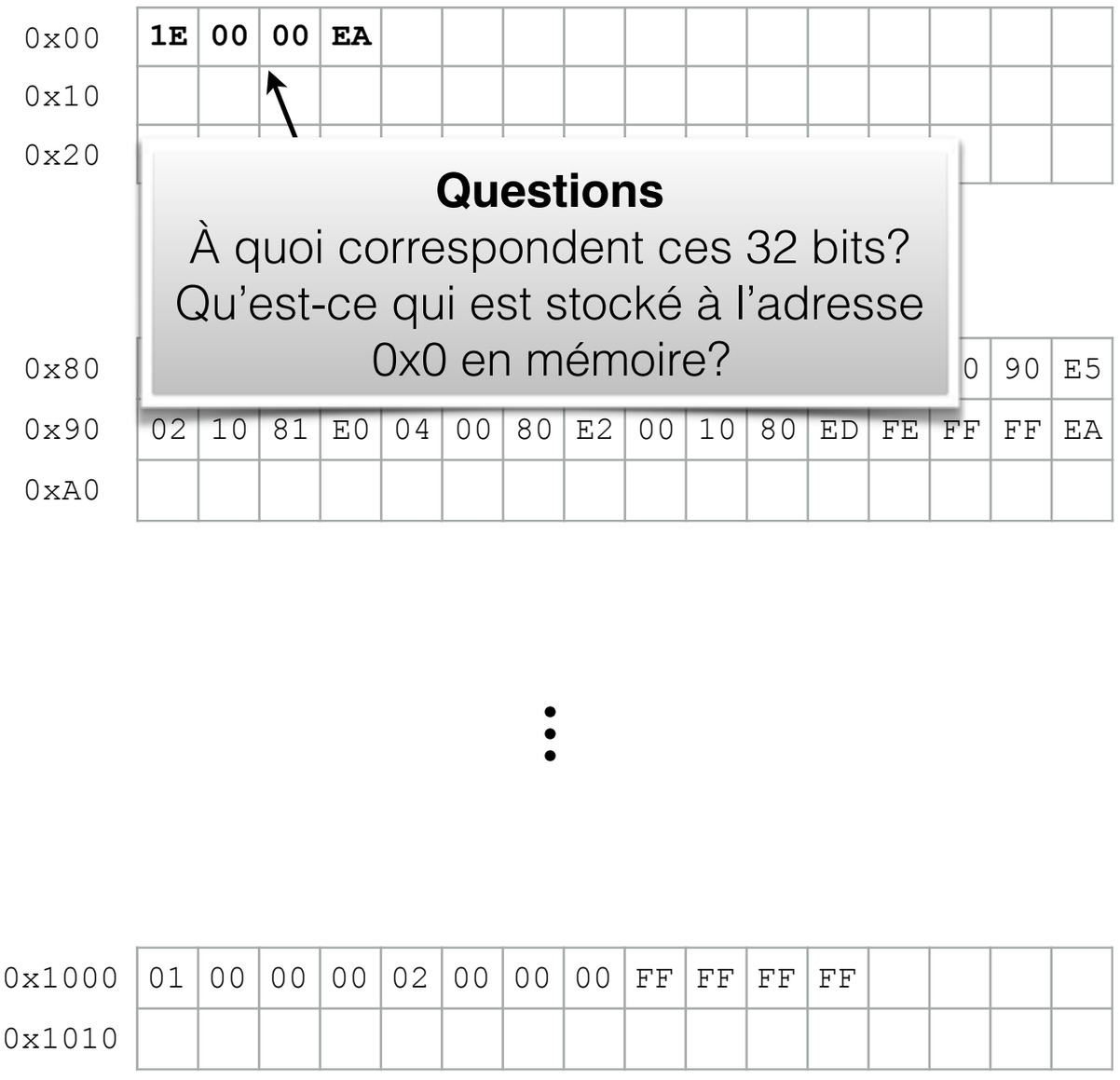
0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF				
0x1010																

Contenu de la mémoire

Code

```
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



Questions

À quoi correspondent ces 32 bits?

Qu'est-ce qui est stocké à l'adresse 0x0 en mémoire?

Contenu de la mémoire

Code

```
SECTION INTVEC

B main


SECTIO

main

MOV R0
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
```

Le premier élément de la section INTVEC, soit l'instruction `B main`.

Mémoire

0x00	**1E**	**00**	**00**	**EA**											
0x10															
0x20															
0x80													0	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 0x0 en mémoire?

⋮

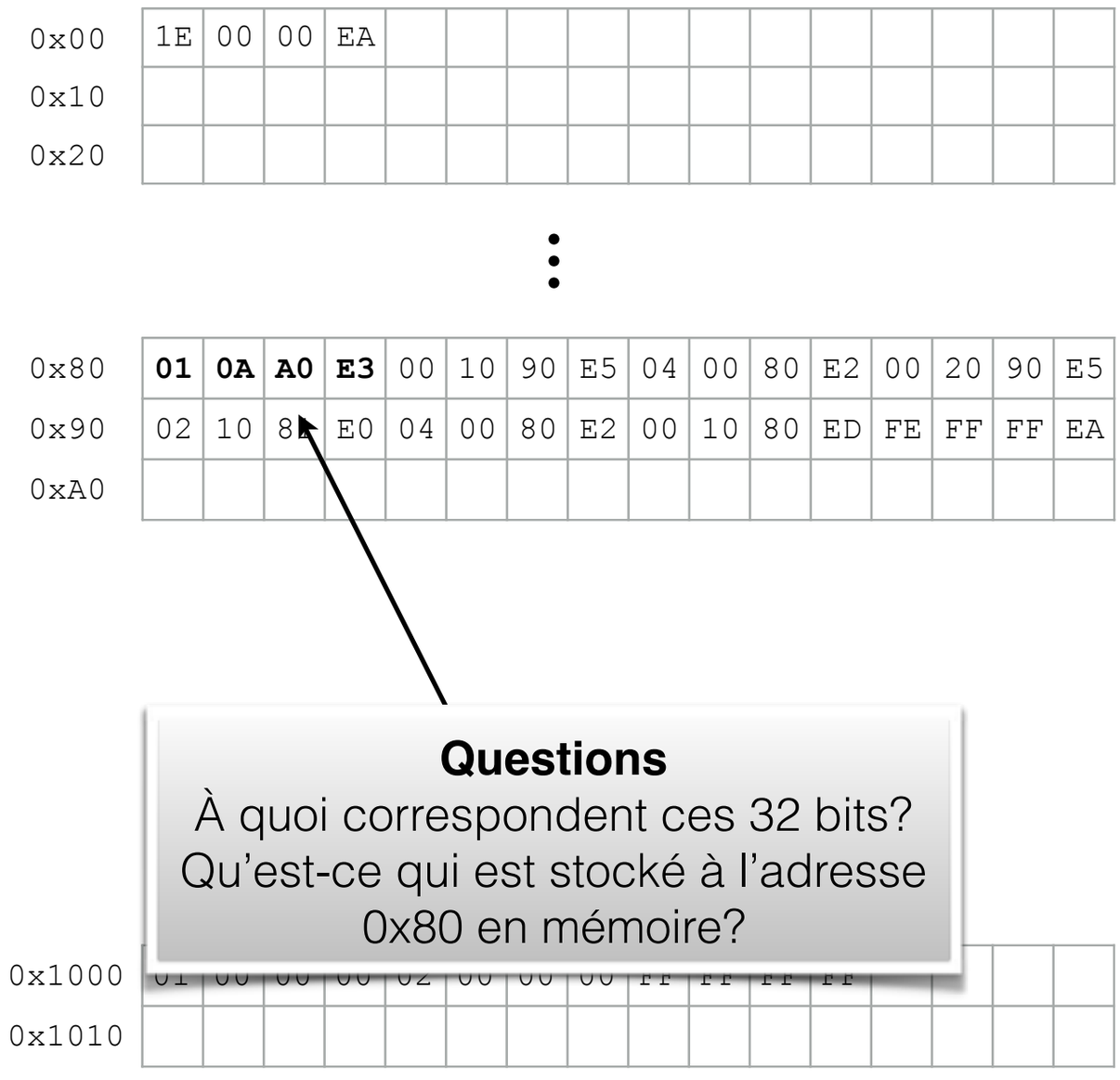
0x1000	01	00	00	00	02	00	00	00	FF	FF	FF	FF				
0x1010																

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



Contenu de la mémoire

```
SECTIO

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


SECTIO

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

; Résultat
resultat ALLOC32  1
```

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

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

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																

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

Question
Où cette instruction est-elle stockée en mémoire?

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

```
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
```

Question
Où cette instruction est-elle stockée en mémoire?

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																

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


SECTION DATA

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

; Résultat
resultat ALLOC32  1
```

Le premier élément de la section DATA, soit notre première valeur à additionner

`premiereValeur ASSIGN32 0x1`

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.