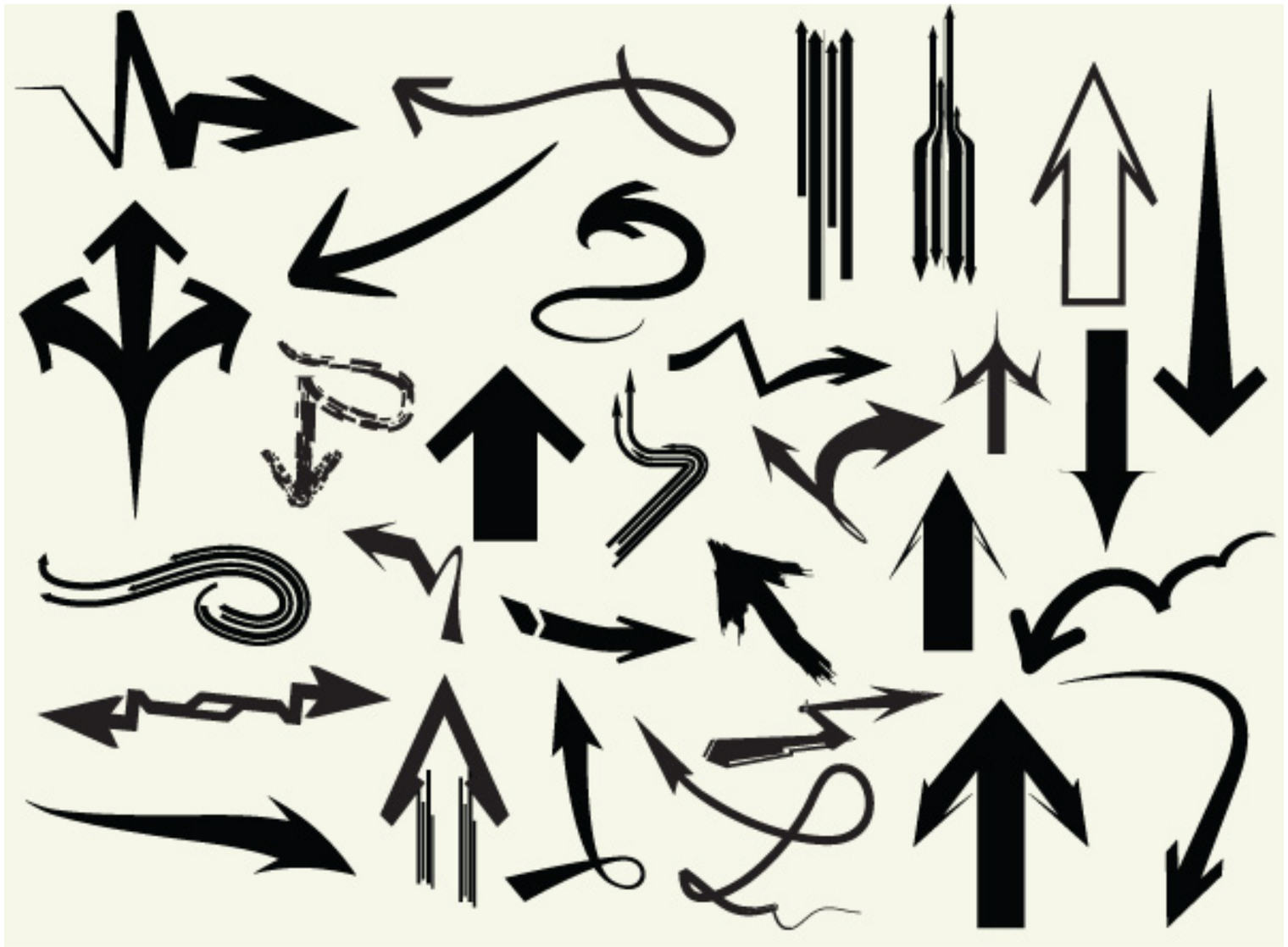
Introduction à l'assembleur ARM:
adressage



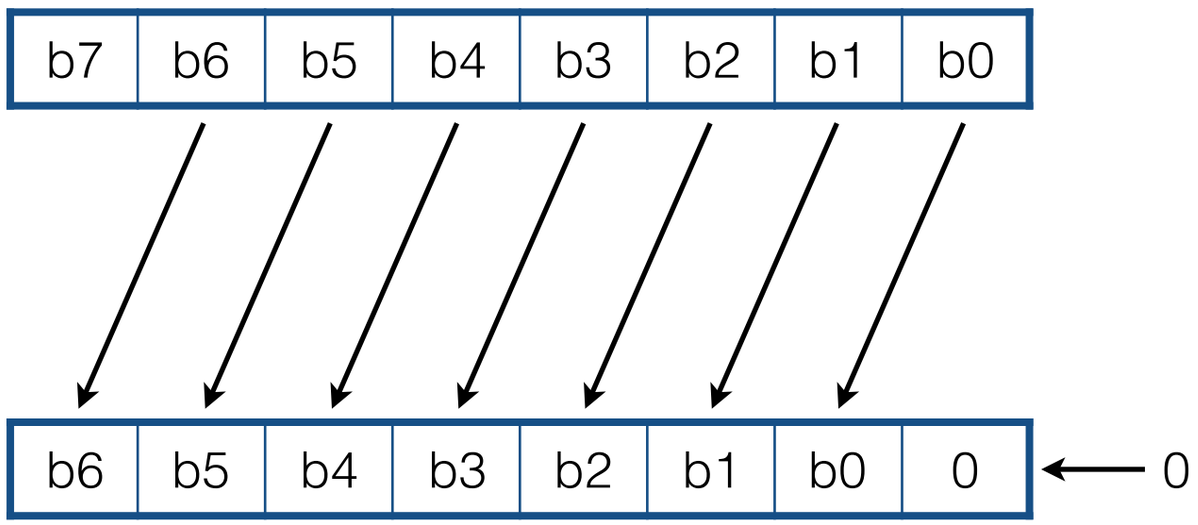
GIF-1001 Ordinateurs: Structure et Applications
Jean-François Lalonde

image

Décalage de bits

- LSL (*Logical Shift Left*):
  - décale vers la gauche
  - mets des « 0 » à droite
  - équivaut à multiplier un entier par 2

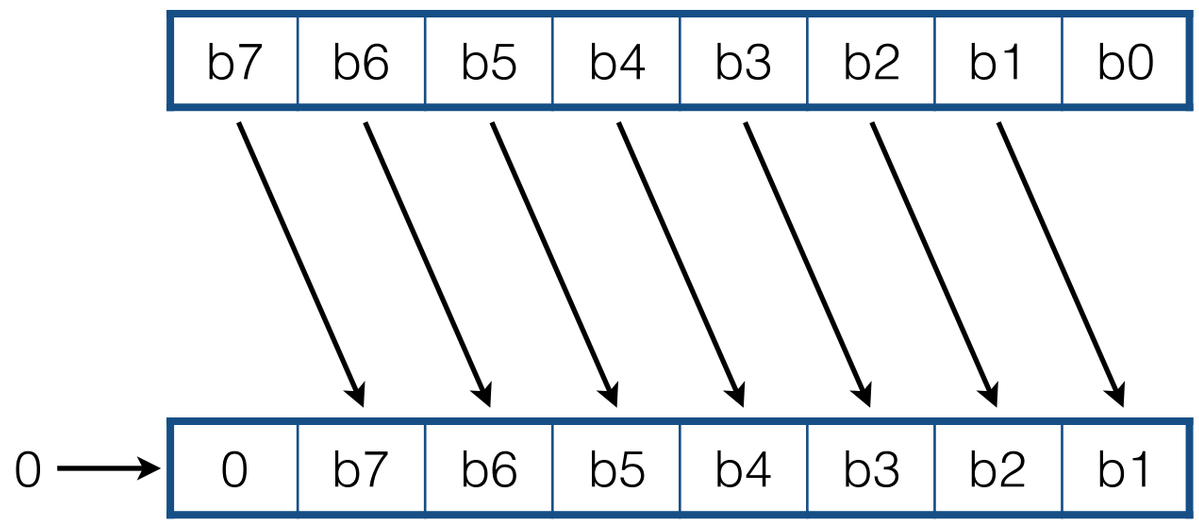
```
LSL R0, R0, #1
```



Décalage de bits

- LSR (*Logical Shift Right*):
  - décale vers la droite
  - mets des « 0 » à gauche
  - équivaut à diviser un **entier non-signé** par 2

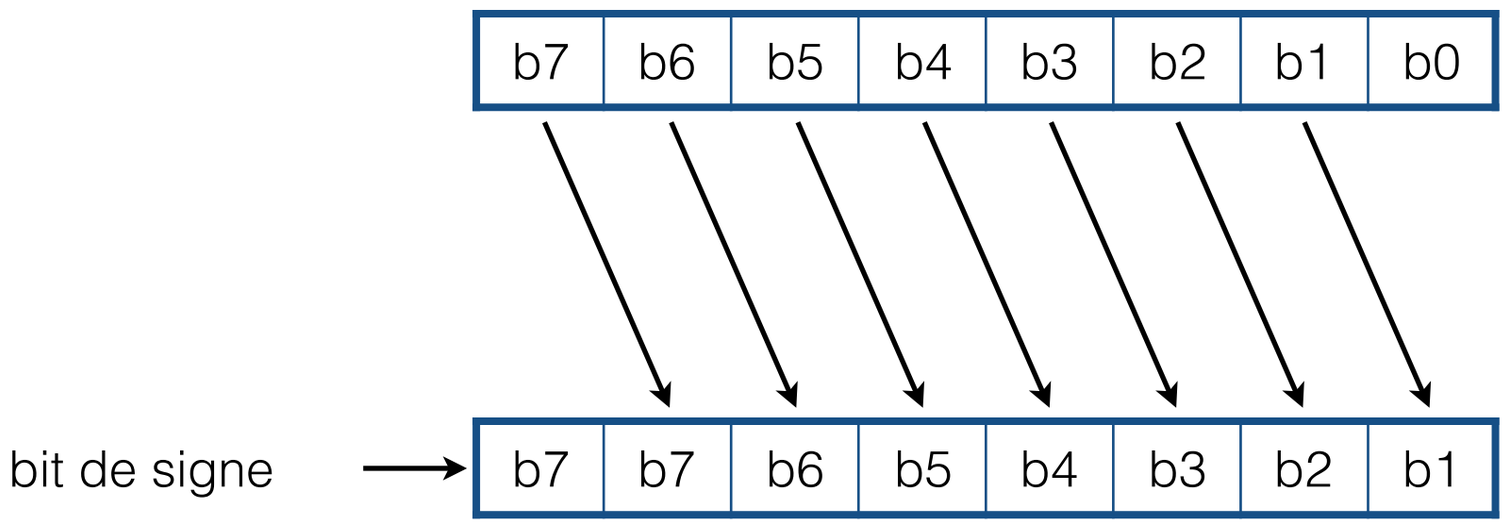
```
LSR R0, R0, #1
```



Décalage de bits

- ASR (*Arithmetic Shift Right*):
  - décale vers la droite
  - mets **le bit de signe** à gauche
  - équivaut à diviser un **entier signé** par 2

```
ASR R0, R0, #1
```



Déplacement de Données: MOV

- L'instruction

```
MOV Rn Op
```

  met l'opérande `Op` dans le registre `Rn`

- 3 choix pour l'opérande:

  1. Constante: toujours précédée du symbole #
  2. Registre
  3. Registre décalé
     - Le décalage est fait avant l'opération

- Exemples:

```
MOV R0, #1234          ; R0 ← 1234
MOV R0, R1             ; R0 ← R1
MOV R0, R1, ASR #2     ; R0 ← (R1 / 4)
```

Accès Mémoire: Load/Store

- Les accès à la mémoire se font avec deux instructions:
  - `LDR` (*LoaD Register*) lit la mémoire et met la valeur lue dans un registre.
  - `STR` (*STore Register*) met la valeur d'un registre dans la mémoire.
- Ces instructions ont le format

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

- `Rd` et `Rs` décrivent le registre de destination ou de source
- `Offset` représente un **décalage** par rapport au registre

Registre + décalage

```
LDR Rd, [Rb, Offset]
```

- Pour calculer l'adresse, on additionne `Rb` et `Offset`
  - `Rb` est le registre de base
  - `Offset` est une opérande
    1. Constante: toujours précédée du symbole #
    2. Registre
    3. Registre décalé

```
LDR R0, [R2]                  ; R0 ← Mémoire[R2]
LDR R0, [R2, #4]              ; R0 ← Mémoire[R2 + 4]
LDR R0, [R2, R3]              ; R0 ← Mémoire[R2 + R3]
LDR R0, [R1, R2, LSL #2]      ; R0 ← Mémoire[R1 + (R2 * 4)]
```

Exercice

Quel sera le contenu des registres R1, R2, R3, R4 et R5 après l'exécution de ce programme?

Mémoire

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

Code

```
MOV R0, #0x1000

LDR R1, [R0]
LDR R2, [R0, #4]
LDR R3, [R0, R2]
LDR R4, [R0, R1, LSL #2]
LDR R5, [R0, #2]
```

Solution

Quel sera le contenu des registres R1, R2, R3, R4 et R5 après l'exécution de ce programme?

Mémoire

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

Code

```
MOV R0, #0x1000

LDR R1, [R0]
LDR R2, [R0, #4]
LDR R3, [R0, R2]
LDR R4, [R0, R1, LSL #2]
LDR R5, [R0, #2]
```

R0	
R1	
R2	
R3	
R4	
R5	

Solution

Quel sera le contenu des registres R1, R2, R3, R4 et R5 après l'exécution de ce programme?

Mémoire

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

Code

```
MOV R0, #0x1000

LDR R1, [R0]
LDR R2, [R0, #4]
LDR R3, [R0, R2]
LDR R4, [R0, R1, LSL #2]
LDR R5, [R0, #2]
```

R0	0x1000
R1	
R2	
R3	
R4	
R5	

Solution

Quel sera le contenu des registres R1, R2, R3, R4 et R5 après l'exécution de ce programme?

Mémoire

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

Code

```
MOV R0, #0x1000

LDR R1, [R0]
LDR R2, [R0, #4]
LDR R3, [R0, R2]
LDR R4, [R0, R1, LSL #2]
LDR R5, [R0, #2]
```

R0	0x1000
R1	0x1
R2	
R3	
R4	
R5	

Solution

Quel sera le contenu des registres R1, R2, R3, R4 et R5 après l'exécution de ce programme?

Mémoire

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

Code

```
MOV R0, #0x1000

LDR R1, [R0]
LDR R2, [R0, #4]
LDR R3, [R0, R2]
LDR R4, [R0, R1, LSL #2]
LDR R5, [R0, #2]
```

R0	0x1000
R1	0x1
R2	0x4
R3	
R4	
R5	

Solution

Quel sera le contenu des registres R1, R2, R3, R4 et R5 après l'exécution de ce programme?

Mémoire

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

Code

```
MOV R0, #0x1000

LDR R1, [R0]
LDR R2, [R0, #4]
LDR R3, [R0, R2]
LDR R4, [R0, R1, LSL #2]
LDR R5, [R0, #2]
```

R0	0x1000
R1	0x1
R2	0x4
R3	0x4
R4	
R5	

Solution

Quel sera le contenu des registres R1, R2, R3, R4 et R5 après l'exécution de ce programme?

Mémoire

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

Code

```
MOV R0, #0x1000

LDR R1, [R0]
LDR R2, [R0, #4]
LDR R3, [R0, R2]
LDR R4, [R0, R1, LSL #2]
LDR R5, [R0, #2]
```

R0	0x1000
R1	0x1
R2	0x4
R3	0x4
R4	0x4
R5	

Démonstration

(Exercice)

Variantes de LDR/STR (*pre-indedxing*)

- On peut modifier le registre `Rb` *avant* le calcul d'accès mémoire
  - symbole « ! »

```
LDR R1, [R0, #4]!    ; R0 ← R0 + 4, R1 ← Memoire[R0]
STR R1, [R0, #4]!    ; R0 ← R0 + 4, Memoire[R0] ← R1
```

Variantes de LDR/STR (*pre-indedxing*)

- On peut modifier le registre `Rb` *avant* le calcul d'accès mémoire
  - symbole « ! »

```
LDR R1, [R0, #4]!    ; R0 ← R0 + 4, R1 ← Memoire[R0]
STR R1, [R0, #4]!    ; R0 ← R0 + 4, Memoire[R0] ← R1
```

Exemple: `LDR R1, [R0, #4]!`

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

R0	0x1000
R1	0x0

Variantes de LDR/STR (*pre-indedxing*)

- On peut modifier le registre `Rb` *avant* le calcul d'accès mémoire
  - symbole « ! »

```
LDR R1, [R0, #4]!    ; R0 ← R0 + 4, R1 ← Memoire[R0]
STR R1, [R0, #4]!    ; R0 ← R0 + 4, Memoire[R0] ← R1
```

Exemple: `LDR R1, [R0, #4]!`

0x1000	01	00	00	00	**04**	**00**	**00**	**00**	0A	00	00	00	0B	00	00	00
0x1010																

R0	**0x1004**
R1	**0x4**

Variantes de LDR/STR (*post-indedxing*)

- On peut modifier le registre `Rb` *après* le calcul d'accès mémoire
  - en dehors des []

```
LDR R1, [R0], #4 ; R1 ← Memoire[R0], R0 ← R0 + 4
STR R1, [R0], #4 ; Memoire[R0] ← R1, R0 ← R0 + 4
```

Variantes de LDR/STR (*post-indedxing*)

- On peut modifier le registre `Rb` *après* le calcul d'accès mémoire
  - en dehors des []

```
LDR R1, [R0], #4 ; R1 ← Memoire[R0], R0 ← R0 + 4
STR R1, [R0], #4 ; Memoire[R0] ← R1, R0 ← R0 + 4
```

Exemple: `LDR R1, [R0], #4`

0x1000	01	00	00	00	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

R0	0x1000
R1	0x0

Variantes de LDR/STR (*post-indedxing*)

- On peut modifier le registre `Rb` *après* le calcul d'accès mémoire
  - en dehors des []

```
LDR R1, [R0], #4 ; R1 ← Memoire[R0], R0 ← R0 + 4
STR R1, [R0], #4 ; Memoire[R0] ← R1, R0 ← R0 + 4
```

Exemple: `LDR R1, [R0], #4`

0x1000	**01**	**00**	**00**	**00**	04	00	00	00	0A	00	00	00	0B	00	00	00
0x1010																

R0	**0x1004**
R1	**0x1**

Démonstration

(Tableaux avec adressage)

Récapitulation: MOV vs LDR/STR

- MOV: déplacements **entre des registres seulement**

```
MOV R0, #0xFF          ; R0 ← 0xFF
MOV R0, R1             ; R0 ← R1
MOV R0, R1, ASR #2     ; R0 ← (R1 / 4)
```

- LDR/STR: déplacements **entre le CPU et la mémoire**

```
LDR R0, [R1]           ; R0 ← Memoire[R1]
LDR R0, [R1, #4]       ; R0 ← Memoire[R1 + 4]
LDR R0, [R1, R2]       ; R0 ← Memoire[R1 + R2]
LDR R0, [R1], #4       ; R0 ← Memoire[R1], R1 ← R1 + 4
```

```
STR R0, [R1]           ; Memoire[R1] ← R0
STR R0, [R1, #4]       ; Memoire[R1 + 4] ← R0
STR R0, [R1, R2]       ; Memoire[R1 + R2] ← R0
STR R0, [R1], #4       ; Memoire[R1] ← R0, R1 ← R1 + 4
```

Accès mémoire avec PC

- On peut aussi se servir de PC pour accéder à la mémoire

```
LDR Rd, [PC, #-4]  ; Rd ← Memoire[PC - 4]
```

- On se rappelle:
  - PC contient l'adresse de l'instruction courante **+ 8**
    - PC est « en avance »: il pointe 2 instructions plus loin.

Accès mémoire avec PC

Exemple:

instruction courante →

```
0x80    LDR R0, [PC, #-4]
0x84    MOV R1, R2
0x88    ...
```

Quelle est la valeur de PC?

0x80	04	10	1F	E5	02	10	A0	E1	0A	00	00	00	0B	00	00	00
0x90																

R0	
PC	

Accès mémoire avec PC

Exemple:

instruction courante ⟶

```
0x80    LDR R0, [PC, #-4]
0x84    MOV R1, R2
0x88    ...
```

0x80	04	10	1F	E5	02	10	A0	E1	0A	00	00	00	0B	00	00	00
0x90																

R0	
PC	**0x88**

Accès mémoire avec PC

Exemple:

instruction courante →

```
0x80    LDR R0, [PC, #-4]
0x84    MOV R1, R2
0x88    ...
```

Code binaire correspondant à `MOV R1, R2`!

0x80	04	10	1F	E5	**02**	**10**	**A0**	**E1**	0A	00	00	00	0B	00	00	00
0x90																

R0	**0xE1A01002**
PC	0x88

Démonstration

(Adressage en ARM)