Introduction à l'assembleur ARM:
utilisation de variables

```
bool arm = true;
```

GIF-1001 Ordinateurs: Structure et Applications, Hiver 2019
Jean-François Lalonde

Revenons à notre exemple d'addition

```
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
```

- Pas pratique:
  - d'avoir à connaître l'adresse des valeurs et du résultat
  - d'avoir à utiliser deux instructions (MOV puis LDR) pour les charger
- Solution?
  - L'assembleur nous permet de **donner un nom** à des adresses mémoires: ce sont les variables!

Variables assignées

- Déclarer une variable assignée (initialisée)

```
nom ASSIGNxx valeur
```

  - `nom`: le nom de la variable
  - `ASSIGNss`: `xx` indique la taille. Par exemple:
    - `ASSIGN8`: variable de 8 bits
    - `ASSIGN32`: variable de 32 bits
- `valeur`: la valeur de la variable
- exemple:

```
premiereValeur ASSIGN32 0x1
deuxiemeValeur ASSIGN32 0x2
```

Variables allouées

- Allouer une variable (réserve de l'espace mais ne lui assigne pas de valeur):

```
nom ALLOCxx nombre
```

  - `nom`: nom de la variable
  - `ALLOCxx`: `xx` indique la taille. Par exemple:
    - `ALLOC8`: variable de 8 bits
    - `ALLOC32`: variable de 32 bits
- `nombre`: le nombre d'éléments à réserver
- exemple:

```
; Résultat (on ne connait pas sa valeur a priori)
resultat ALLOC32  1
```

Assignation vs allocation

- `nom ASSIGNxx valeur`
  - assigne une *valeur* à un espace mémoire
  - parallèle avec le standard du C:
    - c'est une *définition* (on définit une valeur précise).
- `nom ALLOCxx nombre`
  - alloue un *nombre* d'espaces mémoire *sans attribuer de valeur*.
  - parallèle avec le standard du C:
    - c'est une *déclaration* (on déclare une variable sans définir de valeur).

Accéder aux variables

- Charger la **valeur** d'une variable
  - « Variante » de LDR: `LDR Rd, nomDeLaVariable`

```
; Charger la valeur de la variable premiereValeur
LDR R1, premiereValeur
```

Exemple d'addition (avec variables)

- Comment représenter notre programme d'addition en utilisant des variables?

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

```
; Résultat (on ne connait pas sa valeur a priori)
resultat ALLOC32  1
```

```
LDR R1, premiereValeur  ; charger la valeur de premiereValeur
LDR R2, deuxiemeValeur  ; charger la valeur de deuxiemeValeur
ADD R1, R1, R2          ; R1 = R1 + R2
STR R1, resultat        ; stocker R1 dans resultat
```

Démonstration

(addition, avec variables)

Accéder aux variables

- Charger la **valeur** d'une variable
  - « Variante » de LDR: `LDR Rd, nomDeLaVariable`

```
; Charger la valeur de la variable premiereValeur
LDR R0, premiereValeur
```

- Charger **l'adresse** d'une variable
  - « Variante » de LDR: `LDR Rd, =nomDeLaVariable`

```
; Charger l'adresse de la variable premiereValeur
LDR R0, =premiereValeur
```

Tableaux

- Pour déclarer un tableau:

```
nom ASSIGNxx el1 el2 el3 ...   ; Assignation
nom ALLOCxx nombreElements     ; Allocation
```

  - `nom`: nom du tableau
  - `xx` indique la taille
  - `el1 el2 el3...`: la valeur des éléments du tableau s'il s'agit d'une assignation
  - `nombreElements`: le nombre d'éléments dans le tableau s'il s'agit d'une allocation

- exemple:

```
a  ALLOC32   3                ; tableau de 3 mots de 32 bits chacun
b  ASSIGN8   0x01 0x02 0x03   ; tableau de 3 octets
```

Démonstration

(tableaux)

Tableaux et chaînes de caractères

- Les tableaux peuvent être vus comme des chaînes de variables.

- Une chaîne de caractères est un exemple de tableau d'octets

  - chaque caractère (lettre) est encodé en ASCII (0 à 255).

- Par exemple:

```
chA ASSIGN8 'Hello', 0
chB ASSIGN8 0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x00
```

- `chA` est la copie exacte de `chB`. Lorsque l'assembleur voit une chaîne entourée par des ' ', il la convertit automatiquement en un ensemble d'octets.