

# Programmation en assembleur du LC3

## 1 Programmation en assembleur du LC-3

- Introduction à l'architecture du LC-3
- Allure générale d'un programme
- Les instructions que l'on va utiliser
- Les instructions de chargement et rangement
- Les instructions de branchement
- Un exemple de programme

## Plan

## 1 Programmation en assembleur du LC-3

- Introduction à l'architecture du LC-3
- Allure générale d'un programme
- Les instructions que l'on va utiliser
- Les instructions de chargement et rangement
- Les instructions de branchement
- Un exemple de programme

## Introduction

Le LC-3 est un processeur développé dans un but pédagogique par Yale N. Patt et J. Patel dans [*Introduction to Computing Systems : From Bits and Gates to C and Beyond*, McGraw-Hill, 2004].

Des sources et exécutables sont disponibles à l'adresse :  
<http://highered.mcgraw-hill.com/sites/0072467509/>

Nous allons nous baser sur le LC-3 pour illustrer les points suivants :

- Comment programme-t-on en assembleur ?
- Comment mettre en place des routines ?
- Comment mettre en place une pile d'exécution de programme ?

Ce cours est inspiré de celui d'Olivier Carton (Université Paris Diderot - Paris 7), disponible à l'adresse : <http://www.liafa.jussieu.fr/~carton>

## La mémoire centrale et les registres du LC-3

La mémoire est organisée par **mots de 16 bits**, avec un **adressage sur 16 bits** : adresses de  $(0000)_H$  à  $(FFFF)_H$ .

Les registres :

- **8 registres généraux 16 bits** : R0, ..., R7. Toutefois,
  - ▶ R6 est utilisé spécifiquement pour la gestion de la pile d'exécution,
  - ▶ R7 est utilisé pour stocker l'adresse de retour d'un appel de fonction.
- quelques registres spécifiques 16 bits :
  - ▶ PC (*Program Counter*) et IR (*Instruction Register*)
  - ▶ PSR (*Program Status Register*) plusieurs drapeaux binaires,
- Les bits **N,Z,P** du PSR indiquent si la dernière valeur rangée dans un registre général est strictement négative, zéro ou strictement positive.

## 1 Programmation en assembleur du LC-3

- Introduction à l'architecture du LC-3
- Allure générale d'un programme
- Les instructions que l'on va utiliser
- Les instructions de chargement et rangement
- Les instructions de branchement
- Un exemple de programme

## Constantes

Il existe deux types de constantes.

- Les *chaînes de caractères* qui apparaissent uniquement après la directive **.STRINGZ** : elles sont délimitées par deux caractères " et implicitement terminées par le caractère nul.
- Les *entiers relatifs* en hexadécimal sont précédés d'un **x** ; sinon, ce sont des décimaux (le préfix **#** est optionnel). Ils peuvent apparaître comme
  - ▶ opérandes immédiats des instructions (attention à la taille des opérandes),
  - ▶ paramètres des directives **.ORIG**, **.FILL** et **.BLKW**.

Exemple :

```
.ORIG x3000      ; Constante entière en base 16
AND R2,R1,2      ; Constante entière en base 10
ADD R6,R5,#-1    ; Constante entière négative en base 10
.STRINGZ "Chaîne" ; Constante chaîne de caractères
```

## Forme générale d'un programme

Un programme source écrit en langage d'assemblage est un fichier texte qui comporte une suite de lignes. Sur chaque ligne, on trouve :

- soit une *instruction* (**ADD**, **LD**, **BR**,...);
- soit une *macro* (**GETC**, **OUT**,...);
- soit une *directive d'assemblage* (**.ORIG**, **.BLKW**,...);

Une ligne peut être précédée par une *étiquette* pour référencer son adresse.

```
.ORIG x3000          ; directive pour le début de programme
; partie dédiée au code
loop:   GETC          ; macro marquée par une étiquette
        LD R1,cmpzero   ; instruction
        ...
        HALT          ; macro
; partie dédiée aux données
cmpzero: .FILL xFFD0  ; étiquette et directive
        .END           ; directive
```

## Directives d'assemblage

### ● **.ORIG adresse**

Spécifie l'adresse à laquelle doit commencer le bloc d'instructions qui suit.

### ● **.END**

Termine un bloc d'instructions.

### ● **.FILL valeur**

Réserve un mot de 16 bits et le remplit avec la valeur constante donnée en paramètre.

### ● **.STRINGZ chaîne**

Réserve un nombre de mots égal à la longueur de la chaîne de caractères plus un caractère nul (code ASCII 0) et y place la chaîne.

### ● **.BLKW nombre**

Cette directive réserve le nombre de mots de 16 bits passé en paramètre.

# Les interruptions prédefinies : « appels système »

L'instruction TRAP appelle un gestionnaire d'interruptions mis en place par le petit système d'exploitation du LC-3 : il s'agit donc d'un appel système. Chaque appel système est identifié par une constante sur 8 bits.

Dans l'assembleur du LC-3, on peut utiliser les macros suivantes :

instruction	macro	description
TRAP x00	HALT	termine un programme (rend la main à l'OS)
TRAP x20	GETC	lit au clavier un caractère ASCII et le place dans l'octet de poids faible de R0
TRAP x21	OUT	écrit à l'écran le caractère ASCII placé dans l'octet de poids faible de R0
TRAP x22	PUTS	écrit à l'écran la chaîne de caractères pointée par R0
TRAP x23	IN	lit au clavier un caractère ASCII, l'écrit à l'écran, et le place dans l'octet de poids faible de R0

# Plan

## 1 Programmation en assembleur du LC-3

- Introduction à l'architecture du LC-3
- Allure générale d'un programme
- Les instructions que l'on va utiliser
- Les instructions de chargement et rangement
- Les instructions de branchement
- Un exemple de programme

## Les instructions du LC-3

Les instructions du LC-3 se répartissent en trois classes.

### 1 Instructions arithmétiques et logiques : ADD, AND, NOT.

### 2 Instructions de chargement et rangement :

- ▶ LD, ST : *load* et *store*
- ▶ LDR, STR : comme LD et ST mais avec un adressage relatif
- ▶ LEA : *load effective address*

### 3 Instructions de branchement :

- ▶ BR : branch
- ▶ JSR : jump subroutine
- ▶ TRAP : interruption logicielle
- ▶ RET : de retour de routine

Il y a aussi NOP, pour *no operation*, qui ne fait rien... Pour simplifier les choses, d'autres instructions du jeu d'instruction du LC3 sont ici omises.

## Récapitulatif des instructions du LC-3

syntaxe	action	NZP	codage															
			opcode				arguments											
			F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
NOT DR,SR	DR <- not SR	*	1	0	0	1			DR	SR	1	1	1	1	1	1	1	
ADD DR,SR1,SR2	DR <- SR1 + SR2	*	0	0	0	1			DR	SR1	0	0	0	SR2				
ADD DR,SR1,Imm5	DR <- SR1 + SEXT(Imm5)	*	0	0	0	1			DR	SR1	1			Imm5				
AND DR,SR1,SR2	DR <- SR1 and SR2	*	0	1	0	1			DR	SR1	0	0	0	SR2				
AND DR,SR1,Imm5	DR <- SR1 and SEXT(Imm5)	*	0	1	0	1			DR	SR1	1			Imm5				
LEA DR,label	DR <- PC + SEXT(PCoffset9)	*	1	1	1	0			DR					P_Coffset9				
LD DR,label	DR <- mem[PC + SEXT(PCoffset9)]	*	0	0	1	0			DR					P_Coffset9				
ST SR,label	mem[PC + SEXT(PCoffset9)] <- SR		0	0	1	1			SR					P_Coffset9				
LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*	0	1	1	0			DR		BaseR			Offset6				
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR		0	1	1	1			SR		BaseR			Offset6				
BR[n][z][p] label	Si (cond) PC <- PC + SEXT(PCoffset9)		0	0	0	0	n	z	p					P_Coffset9				
NOP	No Operation		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
RET (JMP R7)	PC <- R7		1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	
JSR label	R7 <- PC ; PC <- PC + SEXT(PCoffset11)		0	1	0	0	1							P_Coffset11				

Notez par exemple que l'addition (ADD) se décline de deux façons :

- ADD DR, SR1, **SR2**
- ADD DR, SR1, **Imm5**

La colonne NZP indique les instructions qui mettent à jour les drapeaux NZP.

## 1 Programmation en assembleur du LC-3

- Introduction à l'architecture du LC-3
- Allure générale d'un programme
- Les instructions que l'on va utiliser
- Les instructions de chargement et rangement
- Les instructions de branchement
- Un exemple de programme

## LDR et STR

On ne va décrire que LDR, mais le principe est symétrique pour STR.

- syntaxe : **LDR DR,baseR,Offset6**
- action :  $DR \leftarrow Mem[baseR + Sext(PCOffset6)]$

Avec LDR et STR, on peut accéder à toutes les adresses de la mémoire, ce qui permet de lever les limitations de LD/ST. On utilise ces instructions pour :

- manipuler des données sur la pile, comme les variables locales des fonctions ;
- accéder aux éléments d'un tableau ou d'une chaîne de caractères.

**baseR** est utilisé comme un **pointeur**. Par contre, **comment initialiser baseR ?**

## LD et ST

On ne va décrire que LD, mais le principe est symétrique pour ST.

- syntaxe : **LD DR,label**  
→ le label servira à désigner une adresse mémoire AdM.
- action :  $DR \leftarrow Mem[PC + Sext(PCOffset9)]$ 
  - ▶ valeur de PC après son incrémentation lors du chargement ;  
si adl est l'adresse de l'instruction,  $PC = adl + 1$ .
  - ▶ PCOffset9 est un décalage en complément à 2 sur 9 bits.
  - ▶ La case mémoire chargée est donc celle à l'adresse :  
 $adM = adl + 1 + Sext(PCOffset9)$ ,  $adl - 255 \leq adM \leq adl + 256$ .

C'est l'assembleur qui va se charger du calcul de PCOffset9 : le programmeur en langage d'assemblage utilise des labels. Notez que la distance entre une instruction LD et la case mémoire dont elle peut charger le contenu est limitée.

## LEA

LEA permet de charger une adresse dans un registre général.

- syntaxe : **LEA DR,label**
- action :  $DR \leftarrow PC + Sext(PCOffset9)$

L'adresse est calculée comme pour LD, mais seule l'adresse est chargée dans DR.

Cette instruction est utile pour :

- charger l'adresse d'un tableau dans un registre,
- charger l'adresse de la base de la pile d'exécution.

On se sert donc de LEA pour initialiser un pointeur.

## 1 Programmation en assembleur du LC-3

- Introduction à l'architecture du LC-3
- Allure générale d'un programme
- Les instructions que l'on va utiliser
- Les instructions de chargement et rangement
- Les instructions de branchement
- Un exemple de programme

On réalise avec BR des *branchements inconditionnels ou conditionnels*.

Trois drapeaux N, Z et P (majuscules) du PSR sont mis à jour dès qu'une nouvelle valeur est chargée dans l'un des registres généraux :

- N passe à 1 si cette valeur est strictement négative,
- Z passe à 1 si cette valeur est zéro,
- P passe à 1 si cette valeur est strictement positive.

L'instruction BR contient trois bits n, z et p (minuscules) :

- syntaxe : BR[n][z][p] label
- action : si cond alors PC  $\leftarrow$  PC+Sext(PCOffset9)  
avec cond =  $(\bar{n} \bar{z} \bar{p}) + (n = N) + (z = Z) + (p = P)$

L'assembleur se charge de calculer PCOffset9 d'après le label fourni : il faut néanmoins garder à l'esprit le fait que la distance du saut est limitée...

## 1 Programmation en assembleur du LC-3

- Introduction à l'architecture du LC-3
- Allure générale d'un programme
- Les instructions que l'on va utiliser
- Les instructions de chargement et rangement
- Les instructions de branchement
- Un exemple de programme

On veut écrire un programme qui range à l'adresse désignée par res la longueur d'une chaîne de caractères se trouvant à l'adresse string. Point de départ :

```
.ORIG x3000
...
; Ici viendra notre code
HALT ; Pour mettre fin à l'exécution
string: .STRINGZ "Hello World"
res: .BLKW #1
.END
```

On va traduire le pseudo-code suivant :

```
R0 <- string; // R0 pointe vers le début de la chaîne
R1 <- 0; // Le compteur R1 est initialisé à 0
while((R2 <- Mem[R0]) != '\0') {
    R0 <- R0+1; // Incrémentation du pointeur
    R1 <- R1+1; // Incrémentation du compteur
}
res <- R1; // Rangement du résultat
```

Cela donne finalement le programme suivant :

```
.ORIG x3000
LEA R0,string ; Initialisation du pointeur R0
AND R1,R1,0 ; Le compteur R1 est initialisé à 0
loop: LDR R2,R0,0 ; Chargement dans R2 du caractère pointé par R0
BRz end ; Test de sortie de boucle
ADD R0,R0,1 ; Incrémentation du pointeur
ADD R1,R1,1 ; Incrémentation du compteur
BR loop
end: ST R1,res
HALT
; Chaîne constante
string: .STRINGZ "Hello World"
res: .BLKW #1
.END
```