TD Chapitre 06 : L'Assembleur

Objectifs

- Comprendre les deux passes de l'assembleur
- Construire une table des symboles
- Encoder des instructions manuellement
- Utiliser les directives et le literal pool

Durée estimée : 1h30

Exercice 1 : Les Deux Passes

Objectif : Comprendre pourquoi deux passes sont nécessaires

1.1 Analyse du problème

Analysez ce code :

```
    B end
    MOV R0, #1
    ADD R0, R0, #2
end:
    HALT
```

Pourquoi l'assembleur ne peut-il pas traduire `B end` en un seul passage ?

▸ **Solution**

1.2 Table des symboles

Construisez la table des symboles pour ce code :

```
.text
start:
    MOV R0, #0
loop:
    ADD R0, R0, #1
    CMP R0, #10
    B.LT loop
end:
    HALT
```

Rappel : Chaque instruction fait 4 octets. Adresse de départ = 0x0000.

▸ **Solution**

Exercice 2 : Encodage d'Instructions

Objectif : Encoder des instructions en binaire

2.1 Format ALU avec immédiat

Encodez `ADD R1, R2, #10` en binaire.

Format : `[cond:4][001][opcode:4][S:1][Rn:4][Rd:4][imm12:12]`

Champ	Valeur
cond	1110 (AL)
class	001 (imm)
opcode	0100 (ADD)
S	?
Rn	?
Rd	?
imm12	?

▸ **Solution**

2.2 Format ALU avec registre

Encodez `SUB R3, R4, R5` en binaire.

Format : `[cond:4][000][opcode:4][S:1][Rn:4][Rd:4][00000000][Rm:4]`

▸ **Solution**

2.3 MOV avec immédiat

Encodez `MOV R0, #255` en binaire.

Indice : MOV utilise opcode 1101 et Rn = 0000.

▸ **Solution**

Exercice 3 : Branchements

Objectif : Calculer les offsets de branchement

3.1 Offset de branchement

Pour ce code :

```
0x0000: B loop
0x0004: ADD R0, R0, #1
0x0008: loop:
0x0008: SUB R1, R1, #1
```

Calculez l'offset encodé dans `B loop`.

Rappel : offset = (cible - PC - 8) / 4

▸ **Solution**

3.2 Branchement arrière

Pour ce code :

```
0x0000: loop:
0x0000: ADD R0, R0, #1
0x0004: CMP R0, #10
0x0008: B.LT loop
```

Calculez l'offset encodé dans `B.LT loop`.

▸ **Solution**

Exercice 4 : Directives

Objectif : Comprendre les directives d'assemblage

4.1 Section .data

Quelle est la taille et le contenu en mémoire de :

```
.data
x:      .word 42
y:      .word 0x1234
msg:    .asciz "Hi"
```

▸ **Solution**

4.2 Alignement

Pourquoi `.align 2` est-il important après une chaîne ?

```
.data
msg:    .asciz "A"
        .align 2
val:    .word 123
```

▸ **Solution**

Exercice 5 : Literal Pool

Objectif : Comprendre le mécanisme des grandes constantes

5.1 Pseudo-instruction

Que génère l'assembleur pour :

```
LDR R0, =0xDEADBEEF
ADD R0, R0, #1
HALT
```

▸ **Solution**

5.2 Quand utiliser =

Pour chaque cas, indiquez si `=` est nécessaire :

Instruction	= nécessaire ?
`MOV R0, #10`	?
`MOV R0, #1000`	?
`LDR R0, =0xFFFFFFFF`	?
`ADD R1, R1, #255`	?

▸ **Solution**

Exercice 6 : Programme Complet

Objectif : Analyser un programme assembleur complet

Code

```
.text
.global _start
_start:
    LDR R0, =array      ; Adresse du tableau
    MOV R1, #0          ; Somme
    MOV R2, #5          ; Compteur
loop:
    LDR R3, [R0]        ; Charger élément
    ADD R1, R1, R3      ; Ajouter à la somme
    ADD R0, R0, #4      ; Élément suivant
    SUB R2, R2, #1      ; Décrémenter compteur
    CMP R2, #0
    B.NE loop
    HALT

.data
array:  .word 1, 2, 3, 4, 5
```

Questions

1. Combien d'instructions dans `.text` ?
2. Quelle est la table des symboles ?
3. Que vaut R1 à la fin ?

▸ **Solution**

Récapitulatif

Compétences validées

- [ ] Expliquer les deux passes de l'assembleur
- [ ] Construire une table des symboles
- [ ] Encoder une instruction en binaire
- [ ] Calculer un offset de branchement
- [ ] Utiliser les directives et le literal pool

Prochaine étape

TP Chapitre 06 : Utiliser l'assembleur sur le simulateur web

Référence : Livre Seed, Chapitre 06 - Assembleur