

# TD Chapitre 04 : Architecture Machine

## Objectifs

- Comprendre le modèle RISC Load/Store
- Maîtriser les registres et leurs rôles
- Écrire des programmes en assembleur A32
- Utiliser les conditions et branchements

Durée estimée : 1h30

## Exercice 1 : Registres et Rôles

**Objectif :** Identifier les registres spéciaux

### 1.1 Association

Associez chaque registre à son rôle :

Registre	Rôle
R0-R3	?
R13 (SP)	?
R14 (LR)	?
R15 (PC)	?

Choix : Pile, Arguments/Retours, Instruction courante, Adresse de retour

► Solution

## 1.2 Question conceptuelle

Pourquoi le PC (R15) est-il accessible comme un registre normal ?

► Solution

## Exercice 2 : Instructions Arithmétiques

**Objectif :** Écrire des opérations de base

### 2.1 Traduire en assembleur

Traduisez ces opérations C en assembleur A32 :

```
int a = 5;          // R0 = 5
int b = 3;          // R1 = 3
int c = a + b;      // R2 = R0 + R1
int d = c - 1;      // R3 = R2 - 1
```

► Solution

## 2.2 Multiplication

nand2c a une instruction MUL. Traduisez :

```
int x = 7;           // R0 = 7
int y = 4;           // R1 = 4
int z = x * y;      // R2 = R0 * R1
```

► Solution

## Exercice 3 : Load/Store

**Objectif :** Comprendre l'accès mémoire RISC

### 3.1 Schéma Load/Store

Complétez le schéma pour incrémenter une variable en mémoire :



► Solution

### 3.2 Écrire le code

Écrivez le code assembleur pour incrémenter la valeur à l'adresse dans R1 :

► Solution

## Exercice 4 : Conditions et Branchements

**Objectif :** Utiliser CMP et B.cond

### 4.1 Traduire un if-else

Traduisez en assembleur :

```
if (R0 == R1) {  
    R2 = 1;  
} else {  
    R2 = 0;  
}
```

► Solution

## 4.2 Boucle while

Traduisez en assembleur :

```
int i = 0;          // R0
int sum = 0;         // R1
while (i < 5) {
    sum = sum + i;
    i = i + 1;
}
```

► Solution

## Exercice 5 : Codes de Condition

**Objectif :** Choisir le bon code de condition

### 5.1 Signé vs Non-signé

Après `CMP R0, R1`, quel code utiliser ?

Situation	Signé	Non-signé
$R0 < R1$	?	?
$R0 \geq R1$	?	?
$R0 == R1$	?	?

► Solution

## 5.2 Piège des comparaisons

R0 = 0xFFFFFFFF, R1 = 0x00000001

Après `CMP R0, R1` :

- En **signé** :  $R0 < R1$  ? (car  $-1 < 1$ )
- En **non-signé** :  $R0 > R1$  ? (car  $4294967295 > 1$ )

Quel branchement est pris : B.LT ou B.HI ?

► Solution

## Exercice 6 : Fonctions

**Objectif :** Utiliser BL et le retour

### 6.1 Appel simple

Complétez le code pour appeler `double_val` qui multiplie R0 par 2 :

```
main:  
    MOV R0, #5  
    ; Appeler double_val  
    ; ...  
    ; Ici R0 devrait valoir 10  
    HALT  
  
double_val:  
    ADD R0, R0, R0      ; R0 = R0 * 2  
    ; Retourner  
    ; ...
```

#### ► Solution

## 6.2 Sauvegarder LR

Si `func_a` appelle `func_b`, que se passe-t-il avec LR ?

► Solution

## Exercice 7 : MMIO

**Objectif :** Accéder aux périphériques

### 7.1 Allumer un pixel

L'écran commence à `0x00400000`. Chaque bit = 1 pixel.

Pour allumer le pixel (0, 0) :

```
; À compléter
LDR R0, =?          ; Adresse écran
MOV R1, #?          ; Bit 7 = premier pixel
STRB R1, [R0]
```

► Solution

## 7.2 Lire le clavier

Le clavier est à `0x00402600`. Comment lire la touche pressée ?

▶ Solution

## Récapitulatif

### Compétences validées

Après ce TD, vous devez savoir :

- [ ] Identifier les registres spéciaux (SP, LR, PC)
- [ ] Écrire des opérations arithmétiques en A32
- [ ] Utiliser le modèle Load/Store
- [ ] Choisir le bon code de condition
- [ ] Appeler une fonction avec BL et retourner

### Prochaine étape

➡ TP Chapitre 04 : Programmer sur le simulateur A32

📘 Référence : Livre Seed, Chapitre 04 - Architecture