GIF-1001 Ordinateurs: Structure et Applications
Solutions: ARM—variables et accès mémoire

---

1. À l'aide de la datasheet du coeur ARM ou d'un simulateur, quels bits formeraient l'instruction `ARM MOV R3, #4`?

   Solution: 0xE3A03004

2. Supposons des nombres signés sur 8 bits. Sous quelles conditions les additions suivantes change-t-elles les drapeaux C et V de l'ALU (si elles le peuvent!):
   (a) Nombre Positif + Nombre Positif

   Solution: C est toujours 0. V est 1 si le bit MSB de la somme est 1 (Somme supérieure à 127), 0 sinon.

   (b) Nombre Positif + Nombre Négatif

   Solution: V est toujours 0. Le bit carry est 1 si le résultat de l'opération est positif, 0 sinon.

   (c) Nombre Négatif + Nombre Négatif

   Solution: C est toujours 1. V est 1 si le bit MSB de la somme est 0 (Somme inférieure à -128), 0 sinon.

3. Pourquoi faut-il 4 bits pour désigner le registre source ou destination d'une opération dans le coeur ARM?

   Solution: Parce que le cœur ARM a 16 registres.

4. Supposons que R0 = 11111111h; R2 = 22222222h; R3 = 33333333h; R4 = 44444444h. Le contenu de quelles adresses se retrouvera dans R5 suite aux instructions suivantes:
   (a) `LDR R5, [R0]`

   Solution: `R5 = Mem[11111111h]`

   (b) `LDR R5, [R0, R1, ASL #2]`

   Solution: `R5 = Mem[33333331h]`

   (c) `LDR R5, [R0, #4]!`

Solution: `R5 = Mem[11111115h]`

5. Quelle(s) instruction(s) pourrions-nous utiliser pour mettre les deux bits LSB de R0 à 0? pour les mettre à 1? Les autres bits ne doivent pas changer et la valeur initiale des bits est inconnue.

Solution: Pour les mettre à 0: `AND R0, R0, #-4`

Pour les mettre à 1: `ORR R0, R0, #3`

6. Écrivez un programme qui multiplie par 2 un entier signé placé en mémoire sur 32 bits. Supposez que ce nombre commence à l'adresse 02000000h et qu'il finisse à l'adresse 02000003h (Little Endian).

Solution:

```
AdresseDeVar DC32 0x02000000
...
LDR R0, AdresseDeVar ; Adresse de la variable
LDR R1, [R0]         ; R1 = Mem[Adresse de la variable]
MOV R1, R1, ASL \#1  ; Multiplication par 2
STR R1, [R0]         ; Stocker le résultat
```

7. Écrivez un programme permettant d'inverser le signe d'un nombre. Supposez que le nombre à inverser est dans `R1`.

Solution: Une solution possible est de multiplier le nombre par -1.

```
MOV R1, #-1
MUL R2, R2, R1
```

8. Dites si les instructions de déplacement ou d'accès à la mémoire qui suivent sont légales ou illégales :
   (a) `MOV R1, MaVar`

   Solution: Illégal

   (b) `MOV R1, #4`

   Solution: Légal

   (c) `MOV #4, R1`

Solution: Illégal

(d) `LDR R1, [MonAdresse]`

Solution: Illégal

(e) `LDR R1, [R0]`

Solution: Légal

(f) `LDR R1, [R0, #4]`

Solution: Légal

(g) `LDR R1, [R0, MonOffset]`

Solution: Illégal

9. À quoi sert le "!" dans l'instruction LDR ou STR ?

Solution: Changer la valeur du registre désignant l'adresse de base avant l'opération.

10. Retrouve-t-on le nom des variables et des fonctions dans la mémoire du microprocesseur ? Si oui, comment ? Si non, pourquoi ?

Solution: Non, le microprocesseur travaille avec des adresses seulement.

11. Quelle directive assembleur permet de déclarer des tableaux de constantes?

Solution: DC

12. : Vous voulez effectuer l'opération $a = b + c + d$. Sachant que a, b, c et d sont des variables déclarées avec DS, donnez une séquence d'instructions permettant de faire cette addition.

Solution:

```
LDR R0, =a
LDR R1, =b
LDR R2, =c
LDR R3, =d
```

```
LDR R4, [R1]
LDR R5, [R2]
LDR R6, [R3]
ADD R4, R4, R5
ADD R4, R4, R6
STR R4, [R0]
```

13. Comment change-t-on les drapeaux de l'ALU. À quoi servent-ils ?

Solution: Presque toutes les instructions ARM peuvent changer les drapeaux de l'ALU avec l'option S. Les drapeaux changent en fonction du résultat de l'opération arithmétique ou logique. Ils servent à évaluer des conditions (exemple : si $a > 8$).

14. Expliquez pourquoi, lorsqu'on décale des bits vers la droite de 1, il faut habituellement copier le bit le plus significatif à gauche ?

Solution: Pour conserver le signe d'un nombre négatif.

15. (Pour aller plus loin) Si on suppose un microprocesseur ARM ayant deux registres seulement et un ordinateur ayant une mémoire RAM qui commence à l'adresse 0x20000000. En assumant que toutes les variables sont réutilisées plus loin, comment seraient assemblées les opérations mathématiques suivantes :

```
int A,B,C,D,E // Les variables sont sur 32bits
A = 3
B = 4
C = A + B
D = A + B + C + 5
```

Solution: La première chose à faire est d'assigner une adresse à A,B,C,D et E, comme le font l'assembleur et l'éditeur de lien. Choisissons arbitrairement que A aura l'adresse 0x20000000, B aura l'adresse 0x20000004, C aura l'adresse 0x20000008, D aura l'adresse 0x2000000C et E aura l'adresse 0x200000010.

Ensuite les adresses des variables seront placées dans la mémoire ROM comme des constantes parce qu'on ne peut pas mettre une adresse 32 bits à l'intérieur d'une instruction 32 bits. Le compilateur mettra l'adresse en ROM avant ou après le code.

Enfin, il faut faire le code. L'assembleur choisira des registres pour effectuer les différentes opérations. Les registres seront utilisés pour entreposer des constantes, des adresses ou des résultats de calculs.

```
B Debut
AdresseDeA DC32 0x20000000
AdresseDeB DC32 0x20000004
AdresseDeC DC32 0x20000008
AdresseDeD DC32 0x2000000C
AdresseDeE DC32 0x20000010

Debut:
;A = 3
MOV R0,#3
LDR R1, AdresseDeA ;Compilé comme LDR R1, [PC, #16]
STR R0, [R1]
;B = 3
MOV R0, #4
LDR R1, AdresseDeB ;Compilé comme LDR R1, [PC, #24]
STR R0, [R1]

;C = A+B; sans tenir compte des opérations précédentes
LDR R0, AdresseDeA
LDR R0, [R0]
LDR R1, AdresseDeB
LDR R1, [R1]
ADD R0, R0, R1
LDR R1, AdresseDeC
STR R0, [R1]


;D = A+B+C+5; sans tenir compte des opérations précédentes
MOV R0, 5
LDR R1, AdresseDeC
LDR R1, [R1]
ADD R0, R0, R1
LDR R1, AdresseDeB
LDR R1, [R1]
ADD R0, R0, R1
LDR R1, AdresseDeA
LDR R1, [R1]
ADD R0, R0, R1
LDR R1, AdresseDeD
STR R0, [R1]
```