Architecture des ordinateurs
TD2

I. Convertir les nombres de binaire pur en décimal

00010010 = $2^4 + 2^1$ = 18

10101111 = $2^7 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0$ = 175

11110000 = $2^7 + 2^6 + 2^5 + 2^4$ = 240

00001111= $2^3 + 2^2 + 2^1 + 2^0$= 15

10101100 = $2^7 + 2^5 + 2^3 + 2^2$ = 172

II. La valeur en décimal de ces nombres codés en complément à deux

Si l'on doit transformer un nombre en son complément à deux « de tête », un bon moyen est de garder tous les chiffres depuis la droite jusqu'au premier 1 (compris) puis d'inverser tous les suivants.

Les nombres sont déjà en complément à 2. Le premier bit est le bit de signe.

00010010 = $2^4 + 2^1$ = 18

10101111 =-($01010001) = -81$

11110000 =-(00010000) = -16

00001111= 15

10101100 = $-(01010100)$=-84

III. Représentation hexadécimale de ces nombres

Méthode facile : prendre des paquets de 4 bits puis se reporter au tableau de conversion binaire-hexadécimal (tableau de $\mathbf{2^4}$ lignes)

A	B	C	D	Hexa
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

Dans ce cas, Le nombre en héxa reste inchangé que ça soit signé ou pas. Ce qui diffère c'est la valeur décimale.

0001 0010 = 0x12

1010 1111 = 0xAF

1111 0000 =0xF0

0000 1111 = 0x0F

1010 1100 = 0xAC

IV. Représentation des nombres de base 10 vers la base 16 puis en binaire 8 bit

Décimal	**Hexadécimal**	**Binaire**
14	**0x0E**	**00001110**
38	**0x26**	**00100110**
364	**0x016C**	**0000 0001 0110 1100 (dépassement)**
-13	**0xF3**	**11110011**
-5	**0xFB**	**11111011**
-200	**0x38**	**00111000**

Appliquer le complément à deux dans le cas d'un signe négatif puis mettre en hexadécimale.

V. Indiquez l'état des indicateurs NVZV pour les opérations suivantes (réalisées avec alu 8 bits)

N=résultat négatif (résultat inférieur à zero) ; Z= résultat nul ; V = débordement signe du résultat diffère de celle des opérandes sur entier signé ; C=retenue binaire sur entier non signé

Opération	N	Z	V	C
12+5	0	0	0	0
12-5	0	0	0	0
4-6	1	0	0	1
120+32	1	0	1	0
0x70+0x20	1	0	1	0
0x90-0x20	0	0	1	0
130-4	0	0	1	0
200+70	0	0	0	1

VI. Représentation en complément à 2 sur 16 bits et 32 bits

/2 sur 8bits	/2 sur 16 bis	/2 sur 32 bits	Base 10
0x12	0x0012	0x00000012	18
0x90	0xFF90	0xFFFFFF90	-112
0xFF	0xFFFF	0xFFFFFFFF	-1

En complément à deux, le bit de signe est infini à gauche.

VII. Coder des chiffres en virgules flottantes

Convertir le chiffre décimal en binaire puis pour le convertir en virgule flottante de la forme 1, mantisse*2êxp ; il faudra placer la virgule après le premier 1 du nombre converti.

5,75 = **1***2^2 + **0***2^1 + **1***2^0 ++ **1***2^-1 + **1***2^-2 = 101,11=**1,0111 2^2**
-11,7 =-(1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 ++ 1*2^-1 + 0*2^-2 + 1*2^-3 + 1*2^-4 + 0*2^-5 + 0*2^-6 + 1*2^- 7+ 1*2^-8) = - 1011,10110011 = - 1,01110110011 2^3

VIII. Pourquoi l'addition de deux nombres en virgule flottante requière plus de transistors qu'une addition de nombres entiers

Car il demande plus d'opération élémentaire que l'addition de nombres entiers

TD3

Q1. Que contient R0 après :

a.	**mov**	**r0, #0x4**	**R0 contient la valeur 0x4**
b.	**mov**	**r1,#0x4**	r1 reçoit la valeur 0x4
	add	**r0,r0,r1**	**R0 contient la valeur 0x4 (valeur de r1+r0)**
c.	**mov**	**r0,#0x5a**	r0 contient la valeur 0x5a
	mov	**r1,#0x0f**	r1 contient la valeur 0x0f
	eor	**r0,r0,r1**	**eor différent à 1 identique à 0 donc r0= 0x55**
d.	**mov**	**r1,#20**	r1 reçoit la valeur 20
	add	**r0,r1,#10**	**r0 contient (r1+10=20+10) la valeur 30**
e.	**mov**	**r0,#0x12**	r0 contient la valeur 0x12
	sub	**r0,r0,#12**	r0 contient la valeur (0x12-12)
	add	**r0,r0,r0**	**r0 contient (r0+r0) 0xC**
f.	**ldr**	**r0,=0x1234**	met l'adresse de la valeur 0x1234 dans R0
	mov	**r0,r0 lsl 2**	**décale la valeur dans r0 vers la gauche de 2 position et déplace le résultat dans r0** **r0 contient r0*4=0x48D0**

LDR/STR: déplacements entre le CPU et la mémoire

MOV: déplacements entre des registres seulement

g.	**ldr**	**r0,=0x12345678**	mettre la valeur dans r0
	ldr	**r1,=0x87654321**	mettre la valeur dans r1
	and	**r0,r0,r1**	(1and1=1 le reste 0) r0=0x02244220
	sub	**r0,r0,r0**	**r0=0**
h.	**ldr**	**r0,=0x12345678**	
	mov	**r0,r0 ror #4**	**r0=0x23456781 (ramener le plus à gauche à droite 4 fois)**
i.	**mov**	**r0,#10**	
	add	**r0,r0,r0 lsr #1**	**r0=0xffff (décaler r0 vers la droite revient à diviser par 2 sa valeur puis l'additionner avec r0)**

Q2. Ecrire un programme simple pour réaliser :

a. R8=0x3

```
mov     r8,#0x3
```

b. R7=10*5

```
mov     r0,#10
mov     r1,#5
mul     r7,r0,r1
```

c. R6 = R9 – 6

```
sub     r6,r9,#6
```

d. R5=6-r9

```
 mov    r0,#6
sub     r5,r0,r9
```

Q3. Le rôle du registre 15

Le registre 15 est un pointeur d’instruction qui indique l'emplacement de la prochaine instruction à exécuter.

Le registre r4 est un registre général

TD4

Q1 :

Addition de deux nombres 64 bits avec un processeur ARM 32 bits :

La première valeur est placée R0/R1 et la deuxième valeur R2/R3. La méthode à appliquer est d'additionner les deux parties 32 bits de poids faibles en activant les flags (ADDS) suivi de l'addition des 2 parties 32bits poids fort +la retenu de l'addition précédente (ADC)

Solution :

```
        R0      R1
+
        R2      R3
-------------------------------------
        R4      R5

ADDS    R5, R3, R1      ; active les flags
ADC     R4, R0, R2      ; additionne avec la retenue
```

Q2

Faire un programme permettant de lire 10 cases mémoires et de multiplier leur valeur par 2.

a. Sans boucle

```
        TAB     EQU     0x80000         ; pseudo instruction initialisation de la variable

        LDR     R10, =TAB               ;  mettre l'adresse 0X80000 dans le registre R10

        LDR     R0 , [R10]              ; mettre le contenu de la case mémoire dont l'adresse est R10 dans  R0

        MOV     R0,R0 LSL #1            ; décalage à gauche 1 fois =multiplication par 2  puis mettre dans R0

        STR     R0, [R10]               ; Ecrire le résultat dans la case mémoire d'adress R10
```

Suite

```
        LDR     R1 , [R10, #4] !        ; mettre le contenu de la case mémoire dont l'adresse est R10 +4 (adresse suivante) dans  R1  Préindexé avec écriture adressage indirect

        MOV     R1,R1 LSL #1            ; décalage à gauche 1 fois =multiplication par 2  puis mettre dans R1

        STR     R1, [R10]               ; écrire le résultat dans la case mem d'adresse R10+4
```

Etc jusqu'a R9

b. Avec boucle :

```
TAB     EQU     0X80000                 ; initialisation d'une constante
Main                                    ; label
        LDR     R10, =TAB               ;  mettre l'adresse 0X80000 dans le registre R10
        MOV     R0,#10                  ; compteur
BCL     LDR     R1 , [R10]              ; mettre le contenu de la case mémoire dont l'adresse est R10
        MOV     R1,R1 LSL #1            ; décalage à gauche 1 fois =multiplication par 2  puis mettre dans R0
        STR     R1, [R10]               ; Ecrire le résultat dans la case mémoire d'adress R10
        ADD     R10 ,R10,#4.            ; passer à l'adresse suivante (ajout de 4 octet )
        SUBS    R0,R0,#1                ; décompte compteur
        BNE     BCL                     ; temp que R0 diff de 0 continuer la boucle
        END     Main                    ; fin programme
```

Q3

Codage

ADD r0,r1,r2

ADD R0, R1, R2 => OP Rd,Rn,Opérande2

1110 00 0 0100 0 0001 0000 000000000010 => 0xE0810002

ADDGT R0, R1, R2

1100 00 0 0100 0 0001 0000 000000000010 => 0xC0810002

ADDLTS R0,R1,R2

1011 00 0 0100 1 0001 0000 000000000010 => 0xB0910002

MOV R0,#5 ici rot = 0 cas utilisation opérande immediat

1110 00 1 1101 0 0000 0000 0000 00000101 => 0xE3A00005

Erreur MOV R0,#5 ror 4 ; rot=2^2n dans ce cas n =1

1110 00 1 1101 0 0000 0000 0001 00000101

B +1000 ;faire un branchement à l'adresse PC(adresse instruction suivante)+1000
àoffset=deplacement/4 = offset 250

Cond 101 L (si link) offset

1110 1 0 1 0 00000000000000000011111010 = > 0xEA0000FA

Q4

Input: R0 = valeur entre 0 et 15 (décimal)
Output: R1 = le code ASCII

Table de correspondance
Exemple soit r0=3

Décimal	Adresse	Contenue en HEXA	ASCII
0	TAB_ASCII + 0	0x30	0
1	TAB_ASCII +1	0x31	1
2	.. +2	0x32	2
3	…	0x33	3
4	…	0x34	4
5	…	0x35	5
6	…	0x36	6
7	…	0x37	7
8	…	0x38	8
9	…	0x39	9
10	…	0x41	A
11	…	0x42	B
12	…	0x43	C
13	…	0x44	D
14	…	0x45	E
15	TAB_ASCII +15	0x46	F

Tableau de correspondance (le but est de codé les chiffres de 0 à 15 et les déchiffrer par le code ASCII).

Chaque case dans le tableau est codée sur un octet donc la sortie désirée se trouve dans la case

>> ----------

@TAB_CONV (l'adresse de la première case du tableau)

+

Le nombre entre 0 et 15 (il sera dans R0 pour notre programme)

<< ----------

Programe :

```
MAIN        LDR     R0, =3
            BL      CONV_ASCII      ;       l'adresse ici c'est @BL
            Next_instructions       ;       l'adresse de cette instruction est @BL + 4
;       Here  "BL" Branch and Link instruction, memorize the address of the next instruction in the register LR "link register" so when le sous program ends it returns to Next_instructions -> @BL+4 car les instructions sont codées sur 32 bits donc 4 octets
```

Sous Programe :

```
CONV_ASCII  LDR     R10, =TAB_ASCII
            LDR     R1, [R10, R0]   ;       means  ADD R10, R10, R0
                                    ;              LDR     R1, [R10]

            MOV     PC, LR          ;       return to link register LR
```