ARCHITECTURE DES ORDINATEURS
Corrigé PARTIEL Octobre 2008
Tous documents autorisés – 2H

Pour toutes les questions, on utilise le jeu d'instructions ARM.

PARTIE 1 : Exécution d'instructions

On considère que les registres du processeur contiennent les huit chiffres hexadécimaux suivants :

R0	4444 AAAA
R1	8765 4321
R2	FFFF 0000
R3	DBCA 1234
R4	FFFF FFFF
R5	1234 5678
R15	F000 0000

Figure 1

Q 1) Donner le contenu des registres R6 à R11 (sous forme de huit chiffres hexadécimaux) après exécution des instructions suivantes.

a) ADD R6, R1, R0 R6 = CBA9EDCB
b) SUB R7, R1, R5 R7 = 7530ECA9
c) ADD R8, R1,R0 LSL #8 R8 = 87654321 + 44AAAA00 = CC0FED21
d) RSB R9, R1, R2 ASR #4 R9 = FFFFF0000 - 87654321 =789AACDF
e) ORR R10, R3,R5 R10=DBFE 567C
f) AND R11, R3,R5 R11=1200 1230
g) EOR R12,R3,R5 R12=C9FE 444C

Q 2) Donner le contenu du registre R15 après exécution des instructions suivantes, en supposant que R15 contient initialement F000 0000 (adresse de l'instruction CMP) pour chacune des instructions.

a) CMP R4, R5 suivie de BGT +4 R4<R5 Faux R15 = F000 0004 +4 = F000 0008_H
b) CMP R2,R4 suivie de BLT +10 R2<R4 Vrai R15 = F000 0004+40_{10} = F000002C_H
c) CMP R4,# -1 suivie de BNE +10 R4 = -1 Faux R15 = F000 0004 +4 = F000 0008_H
d) CMP R3, #0 suivie de BLT +2 R3 <0 Vrai R15= F000 0004 + 8 = F000 000C_H

Q 3) Donner le contenu des registres et des cases mémoires concernées après exécution des instructions suivantes, en supposant que pour chaque instruction, on part de la même configuration initiale qui est celle de la figure 2

a) LDR R3, [R1 + 8] R3 = Mem (R1+8) = Mem (FFFF 0008) = 2222 4444
b) LDR R4, [R1+R2] R4 = Mem (R1+R2) = Mem (FFFF 000C) = 45547667
c) LDR R5, [R1], 10_H R5 = Mem (R1) = 1234 FDB9 ; R1=R1+10 = FFFF 0010H
d) LDR R6, [R1, 14_H] ! R6 = Mem (R1+14) = Mem (FFFF 0014) = ABCD EF98;

R1 = FFFF 0014_H

Adresse mémoire	Contenu		Registre	Contenu
$FFFF0000_H$	1234 FDB9		R0	
$FFFF0004_H$	1357 9864		R1	FFFF 0000
$FFFF0008_H$	2222 4444		R2	0000 000C
$FFFF000C_H$	4554 7667		R3	
$FFFF0010_H$	CCCC EEEE		R4	
$FFFF0014_H$	ABCD EF98		R5	
$FFFF0018_H$	7755 3311		R6	

Figure 2

PARTIE 2 : Programmation assembleur

Soit le contenu des cases mémoires suivantes

Adresse mémoire	Contenu
$F0000000_H$	Partie basse de A
$F0000004_H$	Partie haute de A
$F0000008_H$	Partie basse de B
$F000000C_H$	Partie haute de B
$F0000010_H$	Partie basse de S
$F0000014_H$	Partie haute de S
$F0000018_H$	Retenue

Soient deux nombres A et B **non signés** de 64 bits dont les parties basses et hautes sont rangées en mémoire. On veut calculer la somme S sur 64 bits et ranger les résultats en mémoire. On range également en mémoire la retenue de sortie : 0 (si résultat est représentable sur 64 bits) , 1 si retenue de sortie ($65^{ème}$ bit).
On rappelle que le registre code condition contient les bits N (négatif), Z (zéro), C (retenue) et V (débordement). Les conditions pour les retenues sont CS (quand la retenue est à 1) et CC quand la retenue est à 0. Par exemple, le branchement conditionnel BCS est pris lorsque le bit retenue est à 1 et non pris quand le bit retenue est à 0. Ces conditions s'appliquent à toutes les instructions.

Q 4) Ecrire le programme assembleur ARM qui effectue la somme S = A+B, en supposant que le registre R1 contient initialement F000 0000H.

Pour les instructions LDR et STR, préciser si le déplacement indiqué est décimal ou hexadécimal. Par exemple, 12_{10} ou 12_H

Pour l'addition sur 64 bits, il y a retenue sortant de la somme des bits de poids faible si la somme ne tient pas sur 32 bits (détecté soit via ADDS et la condition BC, ou si la somme sur 32 bits est inférieure à l'une des deux parties basses).
Pour la somme des poids forts, il y a retenue de sortie ($65^{ème}$ bit) dans deux cas

- si la somme des 32 bits de poids fort ne tient pas sur 32 bits (détecté soit via ADDS et la condition BC, ou si la somme sur 32 bits est inférieure à l'une des deux parties basses).
- Si la somme des 32 bits de poids fort est égal à $FFFFFFFF_H$ et qu'il y a une retenue d'entrée.

```
MOV R0,#0
LDR R2, [R1,0]  // Partie basse de A
LDR R3, [R1,8]  //Partie basse de  B
ADDS R2,R2, R1 // Somme parties basses et positionnement de Rcc
STR R2, [R1,10H] // Partie basse de S
LDR R2, [R1,4] // partie haute de A
LDR R3, [R1,C] // partie haute de B
ADDSBC R2,R2,#1 // addition de la retenue si retenue sur parties basses
MOVBC R0, #1 // Retenue de sortie dans le cas FFFFFFFF+1
ADDS R2,R2,R3 //Addition des partie haute et positionnement de Rcc
STR R2, [R1,14H] // Partie haute de S
MOVBC R2, #1 // retenue de sortie si somme des parties hautes sur 33 bits
ADD R2,R2,R0 // La retenue de sortie est à 1 si R0 =1 ou R2=1 (cas mutuellement exclusifs)
STR R2, [R1,18H]
```

Autre version

```
MOV R0,#0
LDR R2, [R1,0]  // Partie basse de A
LDR R3, [R1,8]  //Partie basse de  B
ADD R2,R2, R1 // Somme parties basses et positionnement de Rcc
STR R2, [R1,10H] // Partie basse de S
CMP R2,R3
LDR R2, [R1,4] // partie haute de A
LDR R3, [R1,C] // partie haute de B
ADDLT R2,R2, #1 //Si R2<R3 lors de CMP R2,R3, alors il y a retenue sortant des parties basses.
ADD R2,R2,R3 //Addition des partie haute et positionnement de Rcc
STR R2, [R1,14H] // Partie haute de S
CMP R2,R3 // Si R2 <R3, il y a retenue sortante
MOVLE R0, #1 // retenue de sortie si somme des parties hautes sur 33 bits
STR R0, [R1,18H]
```

PARTIE 3 : Désassemblage

Q 5) Donner le programmes C correspondant au programme assembleur P1 qui travaille sur un tableau X[N], implanté à partir de l'adresse 1000 0000H : Que fait ce programme ? Qu'obtient-on dans la mémoire aux adresses 100 et 104 en fin d'exécution du programme ?

```
Min = X[0];
```

```
Max = X[0];
Imin =0;
Imax = 0;
For (i=1; i< N; i++)
{
if (X[i] <Min){
        Min= X[i];
        Imin = i;}
if (X[i] >Max){
        Max= X[i];
        Imax} = i;}}
```

Ce programme calcule l'indice de l'élément min et l'indice de l'élément max du programme. L'indice Imin est rangé à l'adresse 100 et Imax à l'adresse 104

Q 6) Donner la version du programme qui n'utilise pas les transferts conditionnels (MOVLT et MOVGT), mais des branchements.

```
        ADR R3, 10000000H
        MOV R4, #0
        LDR R0, [R3],#4
        MOV R1,R0
        MOV R2, R0
        MOV R5,R4
        MOV R6,R4
LOOP :  ADD R4,R4,#1
        LDR R0,[R3], #4
        CMP R0,R1
        BGE Suite1
        MOV R1,R0
        MOV R5,R4
        B Suite2
Suite1  CMP R0,R2
        BLE Suite2
        MOV R2,R0
        MOV R6,R4
Suite2 : CMP R4,#999
        BLT LOOP
        MOV R0,#0
        STR R5, [R0+100H]
        STR R6, [R0+104H]
```

PARTIE 4 : Procédures

On rappelle que la suite de Fibonacci est obtenue de la manière suivante. Les deux premières valeurs sont 0 et 1. Chaque valeur suivante est obtenue par addition des deux valeurs précédentes.

Q 7) Ecrire une procédure ARM FIBO qui constitue un tableau contenant la suite des nombres de Fibonacci en passant par registre l'adresse de début et le nombre d'éléments de la suite (0 et 1 sont compris dans le nombre d'éléments.

Le programme appelant correspondant est

```
LDR R1, Pointeur                 // adresse de FIBO (0) ;
LDR  R2, N                       // adresse de la case mémoire contenant la valeur de N
BL FIBO

FIBO
        CMP R2,#0
        MOV LE R15,R14           // N négatif ou nul : sortie
        ADD R6,R1,R2 LSL#2  // Adresse de FIB [N], qui suit le dernier élément du tableau
        CMP R2,#1
        MOV R3,#0
        STR R3, [R1], 4  // X[0] = 0
        MOVEQ R15,R14
        CMP R2, #2
        MOV R4= #1
        STR R4, [R1], 4  // X[1] = 1
        MOVEQ R15,R14
LOOP    ADD R5,R4,R3
        STR R5, [R1], 4  // X[i] =X[i-1]+X[i-2]
        MOV R3,R4
        MOV R4,R5
        CMP R1,R6  // Fin ?
        BLT LOOP
        MOV R15,R14
```

ANNEXE : jeu d'instructions ARM (simplifié)

Le jeu d'instructions ARM a 16 registres (R0 à R15) de 32 bits. R15=CP et R14=Registre de lien et R13 = Pointeur de pile. R0 est un registre normal (non câblé à 0)

Le format général des instructions est : Si condition, INST Rd, Rn, Opérande 2

- opérande 2 = décalage de Rm, comme décalage logique gauche (LSL), décalage logique droite (LSR) ou décalage arithmétique droite (ASR)..
- ou opérande 2 = rotation d'un immédiat

Les instructions mémoire sont les suivantes

Instruction	Signification	Action
LDR	Chargement mot	Rd ← Mem_{32}(AE)
LDRB	Chargement octet	Rd ← Mem_8 (AE)

STR	Rangement mot	$Mem_{32}(AE) \leftarrow Rd$
STRB	Rangement octet	$Mem_{8}(AE) \leftarrow Rd$

Les modes d'adressage sont résumés dans la table ci-dessous.

Mode	Assembleur	Action
Déplacement 12 bits, Pré-indexé	[Rn, #deplacement]	Adresse = Rn + déplacement
Déplacement 12 bits, Pré-indexé avec mise à jour	[Rn, #deplacement] !	Adresse = Rn + déplacement Rn ← Adresse
Déplacement 12 bits, Post-indexé	[Rn], #deplacement	Adresse = Rn Rn ← Rn + déplacement
Déplacement dans Rm Préindexé	[Rn, ± Rm, décalage]	Adresse = Rn ± [Rm] décalé
Déplacement dans Rm Préindexé avec mise à jour	[Rn, ± Rm, décalage] !	Adresse = Rn ± [Rm] décalé Rn ← Adresse
Déplacement dans Rm Postindexé	[Rn], ± Rm, décalage	Adresse = Rn Rn ← Rn ± [Rm] décalé
Relatif		Adresse = CP + déplacement

Les instructions pour évaluer les conditions et les utiliser (dans toutes les instructions) sont données ci-dessous, avec les instructions de branchement et d'appel de procédure.

CMP , TST	CMP Rs1, Rs2 TST Rs1, Rs2	Rs1-Rs2 → RCC Rs1 and Rs2 → RCC
Instructions arithmétiques avec suffixe S : ADDS, SUBS, etc	SUBS Rd, Rs1, Rs2	Rd ← Rs1 – Rs2 Positionne Rcc
Bcond (LT, LE, GT, GE, EQ, NE…)	Bcond, déplacement	Si cond, alors CP ← NCP +déplacement
BL	BL déplacement	R14 ← NCP CP ← NCP + déplacement
BLcond	BLcond déplacement	Si cond, alors { R14 ← NCP CP ← NCP +déplacement}