ARCHITECTURE DES ORDINATEURS
PARTIEL Octobre 2005 (CORRIGÉ)

PARTIE 1 : REPRESENTATION DES NOMBRES

Soit la représentation flottante 16 bits correspondant à la figure 1. L'interprétation est similaire à celle des flottants IEEE simple et double précision. S est le bit de signe. L'exposant est biaisé avec un excès 15. La valeur 0 de la partie exposant est réservée pour la représentation de 0 (partie fractionnaire nulle) et des nombres dénormalisés (partie fractionnaire non nulle). La valeur 31 est réservée pour l'infini (partie fractionnaire nulle) et NaN (partie fractionnaire non nulle). Pour 0<PE<31, un nombre N correspond à $(-1)^S$ x (1,fraction) x $2^{(PE-15)}$ où PE est la partie exposant.
Avec cette représentation, le plus grand nombre positif représentable est 65 504, le plus petit nombre positif normalisé représentable est 2^{-14} et le plus petit nombre positif dénormalisé représentable est 2^{-24}.

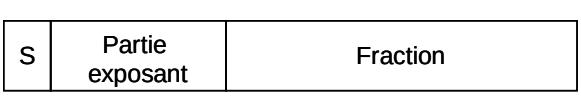

Figure 1 : flottants 16 bits

Q1) Donnez les valeurs décimales ou l'interprétation pour les flottants 16 bits suivants

A) 5000H +1*2(20-15) = 2^5 = 32
B) 4B80H +(1+1/2+1/4+1/8)*2 (18-15) = (15/8) *8 = 15
C) DE00H - (1,5) * 2 (23-15) = - 1,5 * 256 = - 384
D) 0200H 0,5 *2(-15) = 2^{-15}
E) FC00H - ∞

Q2) Donnez la représentation hexadécimale en flottants 16 bits des nombres

A) +47
47 = (47/32=*32 =(1+15/32)*25 = (1+1/4+1/8+1/16+1/32)*25
= 0 10100 011110000 = 51E0H
B) -65
-65 =-(65/64)*26 = -(1+1/64)26= 1 10101 0000010000 = D410H

Q3) Donner les résultats en hexadécimal pour les opérations sur les nombres flottants 16 bits :

A) 5000H + 4B80H = 32+15 = 47 = $51E0_H$
B) 5000H * 4B80H = 32 * 15 = 15/8*23 * 25 = +(1+1/2+1/4+1/8)*2(23-15) = 5F80H
C) 5000H + DE00H = 32 – 384 = -352 = -(1+1/4+1/8)*2^8=-(1+1/4+1/8)*$2^{(23-15)}$ = $DD80_H$
1 10110 0100000000 = D900H
D) 5C00H + FC00H = - ∞ + 32 = - ∞ = FC00H

PARTIE 2 : EXECUTION D'INSTRUCTIONS

Dans cette partie, on utilise les instructions ARM décrites en annexe.

On suppose que les registres R0 à R5 ont les contenus suivants, exprimés en hexadécimal :

R0	1234 4321
R1	9876 ABCD
R2	FEDC 3210
R3	0000 000C
R4	8200 0000
R5	A000 0000

Q4) Donner les valeurs des registres modifiés après exécution des instructions suivantes

a) AND R6, R0,R1 R6 = 10340301
b) ORR R7, R0, R1 R7 = 9A76 EBED
c) EOR R8, R0, R1 R8 = 8A42 E8EC

Q5) Donner les valeurs des registres modifiés après exécution des instructions suivantes. On indiquera les cas de débordement

a) ADD R9, R0,R2 R9 = 1110 7531 OK (P+N=> P)
b) ADD R10, R4,R5 R10 = 2200 0000 Débordement (N+N=> P)
c) SUB R11, R5, R3 R11 = 9FFF FFF4 OK (N-P => N)
d) ADD R12, R0, R1 ASR # 4 // ASR est un décalage arithmétique à droite

- R12 = 1234 4321 + F987 6ABC = 0BBB ADDD OK (P+N => P)

e) ADD R13, R0, R0 LSL #2 // LSL est un décalage logique à gauche

- R13 = 1234 4321 + 48D1 0C84 =5B05 4FA5 OK (P+P=>P)

Q6) Donner l'instruction ou la suite d'instructions pour multiplier le contenu du registre R0 par

a) la constante 17

- 17 = 16 + 1 ADD R0, R0, R0 LSL#4

b) la constante 15

- 15 = 16 – 1 RSB R0, R0, R0 LSL #4

c) la constante 255

- 255 = 256-1
- RSB R0, R0, R0 LSL#8

Soit une zone mémoire à partir de l'adresse A000 0000

Adresse (hexadécimal)	Contenu mot 32 bits (hexadécimal)
A000 0000	98 76 54 32
A000 0004	BC DE F0 12
A000 0008	AA AA CC CC
A000 000C	99 22 33 AA
A000 0010	1A 2B 3C 4D

Q7) Dans l'hypothèse d'une implantation « little endian », donner le contenu des octets d'adresse

- A000 0003 98
- A000 0004 12
- A000 0005 F0
- A000 0006 DE

Q8) Donner le contenu des registres ou des cases mémoire modifiées après exécution des instructions suivantes. **On suppose maintenant l'ordre « big endian ».**
On suppose que les instructions s'exécutent les unes après les autres

a) LDR R6, [R5 ,#4] R6 = BCDEF012
b) LDRSB R7, [R5+R3] R7 = FFFFFF99
c) LDRH R8, [R5, #C] R8= 00009922
d) LDR R9, [R5], #8 R9 = 98765432 R5 = A000 0008
e) LDR R10, [R5, #4] ! R10 = 992233AA R5 = A000 000C
f) LDR R11, [R5, #2] ! **Accès non aligné**
g) STR R1,[R4], #C Mem32 [8200 0000] = 9876 ABCD R4 = 8200 000C
h) STRB R1, [R4-R3] Mem8 [8200 0000] = CD

Q9) Que fait le code ci-dessous

```
MOV R0, #0 ;
LDR R1, [R5], #4
ADD R0, R0, R1
LDR R1, [R5, #4] !
ADD R0, R0, R1
LDR R1, [R5, #4] !
ADD R0, R0, R1
LDR R1, [R5, #4] !
ADD R0, R0, R1
LDR R1, [R5, #4] !
ADD R0, R0, R1
```

R0 = MEM32 [A0000000] + MEM32 [A0000008] + MEM32 [A000000C] + MEM32 [A0000010] + MEM32 [A0000014]

Annexe : Jeu d'instructions ARM

On rappelle que le processeur ARM a 15 registres de 32 bits. Les immédiats sont signés.

Instruction	Assembleur	Effet
ADD	ADD Ri, Rj, Rk	Ri ← Rj+Rk
	ADD Ri, Rj, #N	Ri ← Rj+N
	ADD Ri, Rj, Rk Décalage #immédiat	Ri ← Rj+Rk décalé de N positions
SUB	SUB Ri, Rj, Rk	Ri ← Rj - Rk
	SUB Ri, Rj, #N	Ri ← Rj - N
	SUB Ri, Rj, Rk Décalage #immédiat	Ri ← Rj - Rk décalé de N positions
RSB	RSB Ri, Rj, Rk	Ri ← Rk - Rj
	RSB Ri, Rj, #N	Ri ← N - Rj
	RSB Ri, Rj, Rk Décalage #immédiat	Ri ← Rk décalé de N positions -Rj

AND (et logique)	AND Ri, Rj, Rk AND Ri, Rj, #N AND Ri, Rj, Rk Décalage #immédiat	Ri ← Rj and Rk Ri ← Rj and N Ri ← Rj and Rk décalé de N positions
ORR (ou logique)	ORR Ri, Rj, Rk ORR Ri, Rj, #N ORR Ri, Rj, Rk Décalage #immédiat	Ri ← Rj or Rk Ri ← Rj or N Ri ← Rj or Rk décalé de N positions
EOR (ou exclusif)	EOR Ri, Rj, Rk EOR Ri, Rj, #N EOR Ri, Rj, Rk Décalage #immédiat	Ri ← Rj xor Rk Ri ← Rj xor N Ri ← Rj xor Rk décalé de N positions

Table 1 : Opérations arithmétiques et logiques utilisées

Les instructions mémoire utilisées sont données dans la table 2. L'adresse mémoire est donnée par le mode d'adressage indiqué dans la table 3.

Instruction	Effet	Commentaire
LDR	Rn ← Mem32 (Adresse)	Chargement mot
LDRH	Rn ← extension zéro, Mem16 (Adresse)	Chargement demi mot non signé
LDRSH	Rn ← extension signe, Mem16 (Adresse)	Chargement demi mot signé
LDRB	Rn ← extension zéro, Mem8 (Adresse)	Chargement octet non signé
LDRSB	Rn ← extension signe, Mem8 (Adresse)	Chargement octet signé
STR	Mem32 (Adresse) ← Rn	Rangement mot
STRH	Mem16 (Adresse) ← Rn[15:0]	Rangement demi mot
STRB	Mem8 (Adresse) ← Rn[7:0]	Rangement octet

Table 2 : Instructions mémoire

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 + décalage (Rm)
Déplacement dans Rm Préindexé avec mise à jour	[Rn, ± Rm, décalage] !	Adresse = Rn + décalage (Rm) Rn ← Adresse
Déplacement dans Rm Postindexé	[Rn], ± Rm, décalage	Adresse = Rn Rn ← Rn + décalage (Rm)

Table 3 : Modes d'adressage.