Université Paris Sud D. Etiemble
Licence d'Informatique

ARCHITECTURE DES ORDINATEURS
PARTIEL Octobre 2012 (CORRIGE)
Tous documents autorisés – Calculettes autorisées.

PARTIE 1 : JEU D'INSTRUCTIONS ARM

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	1111 2222
R1	8888 AAAA
R2	FEDC 3210
R3	0000 000C
R4	A800 0000
R5	C000 0000

Q 1) Donner les valeurs des registres modifiés après exécution des instructions suivantes. On indiquera les cas où le résultat est incorrect (débordement arithmétique)

a) ADD R6, R0,R2 R6 =0FED5432 (OK : positif + négatif)
b) ADD R7, R4,R5 R7 = 68000000 (Débordement : négatif + négatif donne positif)
c) SUB R8, R5, R3 R8 = BFFFFF4 (OK : négatif – positif donne négatif)
d) ADD R9, R0, R1 ASR # 4 // R9 =11112222+ F8888AAA= 0999ACCC (OK : positif + négatif donne positif)
e) ADD R10, R0, R0 LSL #2 // R10 = 11112222+44448888=5555AAAA (OK positif +positif donne positif)

Q 2) Donner l'instruction ou la suite d'instructions pour multiplier le contenu du registre R0 par

a) la constante 15 : RSB R0, R0, LSL #4
b) la constante 65 : ADD R0, R0, R0 LSL #6

Soit une zone mémoire à partir de l'adresse C000 0000

Adresse (hexadécimal)	Contenu mot 32 bits (hexadécimal)
C000 0000	56 78 9A BC
C0000 0004	11 22 44 33
C000 0008	98 76 54 32
C000 000C	FF 00 EE 11
C000 0010	A1 B2 C3 D4

Q 3) Donner le contenu des registres ou des cases mémoire modifiées après exécution des instructions suivantes. On suppose l'ordre « little endian ».

NB : l'ordre « little endian » range le mot 0x0A0B0C0D dans l'ordre suivant dans les 4 premiers octets de la mémoire :

Octet d'adresse 3 : 0A
Octet d'adresse 2 : 0B
Octet d'adresse 1 : 0C
Octet d'adresse 0 : 0D

L'octet d'adresse 0 est donc 0D
Le mot de 16 bits d'adresse 0 est donc 0C0D
Le mot de 32 bits d'adresse 0 est donc 0A0B0C0D

a) LDRSB R11, [R5 ,#4] R11 = MEM(C000 0004) = 00 00 00 33
b) LDRH R12, [R5,R3] R12 = MEM8 (C000 000C) = 00 00 EE 11
c) LDR R13, [R5, #12] R13 = MEM16 (C000 000C) = FF 00 EE 11
d) LDR R14, [R5], #4 R14= MEM(C000 0000) = 56 78 9A BC ; R5 = A000 0004
e) LDR R6, [R5, #8] ! R6 = MEM(C000 0008) = 98 76 54 32; R5 = A000 0008
f) LDR R7, [R5, #6] ! Accès non aligné
g) STR R1,[R5], #8 MEM(C000 0000) = 8888 AAAA; R5 = C000 0008
h) STRB R1, [R5,-R3] MEM8 (BFFF FFF4) = AA

Q 4) Le programme suivant effectue un traitement sur le contenu de deux tableaux d'entiers signés X[N] et Y[N] et place le résultat dans R0. Que fait le programme ? (Donner le programme C correspondant)

```
        MOV R0,#0
        ADR R4, X // place l'adresse de X[0] dans R4
        ADR R5, Y //place l'adresse de Y[0] dans R5
        MOV R6, #N
Boucle : LDR R1,[R4], #4
        LDR R2, [R5],#4
        SUB R1,R1,R2
        CMP R1,#0
        RSBLT R1,R1,#0
        ADD R0,R0,R1
        SUBS R6,R6,#1
        BGT Boucle
```

Le programme effectue la somme des valeurs absolues des différences des éléments de même indice des tableaux X et Y.

```
int  vabsdif (a,b)
if (a>b) return (a-b);
        else return (b-a);
Ou
#define vabsdif(a,b)  (a>b) ? (a-b) : (b-a)

Int X[N], Y[N], I, S;
For (i=0; i<N, i++)
        S+= vabsdif (X[i], Y[i]);
```

PARTIE 2 : JEU D'INSTRUCTIONS NIOS II

Dans cette partie, on utilise le jeu d'instructions du NIOS II

Q 5) Ecrire l'instruction ou la suite d'instructions qui place la constante 0x7FFFFFFF dans le registre R3

```
ORHI R3,R0,0x7FFF      //R3 reçoit 0x7FFF 0000
ORI R3,R3, 0xFFFF      //Ou logique entre 0x7FFF 0000 et 0x0000 FFFF
```

Q 6) On suppose que le registre R2 contient un nombre flottant simple précision X. Ecrire un programme NIOS qui met dans R1 le nombre flottant simple précision égal à la valeur absolue de X.

Le format flottant simple précision est rappelé en annexe (Figure 1)

Il faut mettre à zéro le bit de signe (bit 31). On utilise un masque 0x7FFFFFFF et on fait un ET logique entre R2 et le masque.

```
ORHI R3,R0,0x7FFF
ORI R3,R3, 0xFFFF
AND R1,R2,R3
```

Q 7) Ecrire un programme NIOS qui compte le nombre de bits à 1 dans le registre R2 et met le résultat dans R1.

```
        XOR R1,R1,R1
        ADDI R3,R0,32
Boucle : ANDI R4,R2,1
        ADD R1,R1,R4
        ADDI R3,R3,-1
        SRLI R2,R2,1
        BGT R3,R0,Boucle
```

Encore mieux !

```
        XOR R1,R1,R1
Boucle : ANDI R3,R2,1  //Test du bit de poids faible de R2
        ADDI R1,R1,R3
        SRLI R2,R2,1    // Décalage logique d'une position à droite
        BNE R2,R0, Boucle // Arrêt lorsqu'il n'y a plus de 1 dans R2
```

Annexe : Jeu d'instructions ARM

On rappelle que le processeur ARM a 15 registres de 32 bits. Les immédiats sont signés.
R15 est le compteur de programme.

Instruction	Assembleur	Effet
MOV	MOV Ri,Rj	Ri ← Rj

ADD	ADD Ri, Rj, Rk
	ADD Ri, Rj, #N
	ADD Ri, Rj, Rk Décalage #N
		Ri ← Rj+Rk
		Ri ← Rj+N
		Ri ← Rj+(Rk décalé de N positions)

SUB	SUB Ri, Rj, Rk
	SUB Ri, Rj, #N
	SUB Ri, Rj, Rk Décalage #N
		Ri ← Rj - Rk
		Ri ← Rj - N
		Ri ← Rj – (Rk décalé de N positions)

RSB	RSB Ri, Rj, Rk RSB Ri, Rj, #N RSB Ri, Rj, Rk Décalage #N	Ri ← Rk - Rj Ri ← N - Rj Ri ← (Rk décalé de N positions) -Rj
AND (et logique)	AND Ri, Rj, Rk AND Ri, Rj, #N AND Ri, Rj, Rk Décalage #N	Ri ← Rj and Rk Ri ← Rj and N Ri ← Rj and (Rk décalé de N positions)

SUBS	SUBS Ri, Rj, Rk
	SUBS Ri, Rj, #N
	SUBS Ri, Rj, Rk Décalage #N
		Ri ← Rj - Rk et Rcc positionné
		Ri ← Rj - N et Rcc positionné
		Ri ← Rj – (Rk décalé de N positions) et Rcc positionné

ORR (ou logique)	ORR Ri, Rj, Rk ORR Ri, Rj, #N ORR Ri, Rj, Rk Décalage #N	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 #N	Ri ← Rj xor Rk Ri ← Rj xor N Ri ← Rj xor (Rk décalé de N positions)
CMP	CMP Ri,Rj CMP Ri, #N	Compare Ri et Rj (ou Ri et #N) et positionne le registre code condition
BGT	BGT adresse cible	Branchement si le résultat de la comparaison (CMP) était > ou si le résultat de SUBS était différent de 0.

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. Mem32 signifie un accès mémoire à un mot de 32 bits. Mem16 signifie un accès mémoire à un demi-mot (16 bits). Mem8 signifie un accès octet.

Dans le cas d'un accès Mem16 et Mem8, il est précisé si le registre de 32 bits est complété à gauche par des 0 (extension zéro) ou par le signe du demi-mot ou de l'octet lu (extension signe).

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.

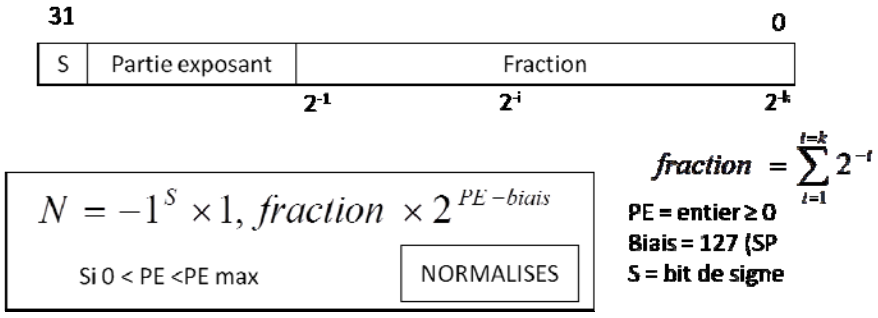

Figure 1 : Rappel sur le format flottant simple précision.