ARCHITECTURE DES ORDINATEURS
PARTIEL Octobre 2010 (2 H)
Tous documents autorisés

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	1221 4334
R1	9889 AAAA
R2	FEDC 3210
R3	0000 000C
R4	8200 0000
R5	A000 0000

Table 1

Q 1) Donner les valeurs des registres modifiés après exécution des instructions suivantes. On indiquera les cas de débordement (overflow).

a) ADD R6, R0,R2
b) ADD R7, R4,R5
c) SUB R8, R5, R3
d) ADD R9, R0, R1 ASR # 4 // ASR est un décalage arithmétique à droite
e) ADD R10, R0, R0 LSL #2 // LSL est un décalage logique à gauche

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

a) la constante 17
b) la constante 7
c) la constante 139

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

Adresse (hexadécimal)	Contenu mot 32 bits (hexadécimal)
A000 0000	12 34 56 78
A000 0004	FE DC BA 98
A000 0008	AA AA CC CC
A000 000C	99 22 33 AA
A000 0010	1A 2B 3C 4D

Table 2

Q 3) Dans l'hypothèse d'une implantation « little endian », donner le contenu des octets d'adresse

- A000 0003
- A000 0004
- A000 0005
- A000 0006

Q 4) Donner le contenu des registres ou des cases mémoire modifiées après exécution des instructions suivantes. On suppose maintenant l'ordre « big endian ». Pour chaque instruction, les registres ont les valeurs contenues dans la table 1

a) LDR R11, [R5 ,#4]
b) LDRSB R12, [R5,R3]
c) LDRH R13, [R5, #12]
d) LDR R14, [R5], #8
e) LDR R6, [R5, #4] !
f) LDR R7, [R5, #2] !
g) STR R1,[R4], #12
h) STRB R1, [R4,-R3]

Q 5) Donner le code C correspondant au programme assembleur AR M suivant, qui travaille sur un tableau d'entiers T[N]. Que contiennent les variables X, Y et Z à la fin de l'exécution du programme.

```
        ADR R1, T       // R1 reçoit l'adresse du tableau T.
        MOV R2, #N
        MOV R3,#0
        MOV R4, #0
Boucle : LDR R0, [R1] #4
        CMP R0, #0
        ADDLT R3,R3,#1
        ADDEQ R4,R4, #1
        BGT Boucle
        ADR R0, X
        STR R3, [R0]
        ADR R0, Y
        STR R4,[R0]
        ADD R3,R3,R4
        RSB R3,R3,#N
        ADR R0,Z
        STR R3,[R0]
```

PARTIE 2 : JEU D'INSTRUCTIONS NIOS II

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

Q 6) Ecrire le programme assembleur NIOS qui calcule la factorielle d'un nombre N >0 contenu dans R1 et qui place le résultat dans R3

On rappelle que n ! = n * (n-1) * (n-2) * (n-3) * …* 3 * 2 * 1.

Q 7) Modifier le programme de la question 6 pour que le résultat obtenu dans R3 soit 0 dans l'hypothèse où N ! n'est pas représentable sur 32bits en complément à 2.

Q 8) Que fait le programme assembleur NIOS suivant ? On indiquera la valeur de R2 en fin d'exécution.

```
        ADD R1, R0, R0
Boucle :ANDI R3, R2, 1
        ADD R1,R1,R3
        SRLI R2,R2,1
        BNE R2,R0, Boucle
```

Annexe : Jeu d'instructions ARM

On rappelle que le processeur ARM a 16 registres de 32 bits (R15 étant le compteur de programme). Les immédiats sont signés.

Instruction	Assembleur	Effet

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)
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)

Table 3 : Opérations arithmétiques et logiques utilisées

Les instructions mémoire utilisées sont données dans la Table 4. L'adresse mémoire est donnée par le mode d'adressage indiqué dans la Table 5. 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 4 : 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 5 : Modes d'adressage.