Université Paris Sud D. Etiemble
Licence d'informatique 2010-2011

ARCHITECTURE DES ORDINATEURS
PARTIEL Octobre 2010 (CORRIGÉ)

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

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 R6 = 10FD 7544 (OK positif + négatif)
b) ADD R7, R4,R5 R7 = 2200 000 (débordement car négatif + négatif = positif)
c) SUB R8, R5, R3 R8 = 9FFF FFF4 (OK car négatif – positif = négatif)
d) ADD R9, R0, R1 ASR # 4 // R9 = 0BA9 DDDE (OK car positif + négatif = positif)
e) ADD R10, R0, R0 LSL #2 // R10 = 5AA6 5004 (OK car positif + positif = positif)

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

a) la constante 17 : ADD R0, R0, LSL #4
b) la constante 7 : RSB R0, R0, R0 LSL #3
c) la constante 139
   139 = 119 + 20 = (17*7) + 16 +4
   ADD R1, R0, R0 LSL #4 R1 = 17 R0
   RSB R1, R1, R1 LSL #3 R1= 7*17 R0
   ADD R1, R1, R0 LSL #4 R1 = 119 R0 + 16 R0 = 135 R0
   ADD R1,R1, R0 LSL #2 R1= 135 R0 + 4 R0 = 139 R0.

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

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

- A000 0003 12
- A000 0004 98
- A000 0005 BA
- A000 0006 DC

Q 4) Donner le contenu des registres ou des cases mémoire modifiées après exécution des instructions suivantes. On suppose maintenant l'ordre « big endian ».

a) LDR R11, [R5 ,#4] R11 = MEM(A000 0004) = FE DC BA 98
b) LDRSB R12, [R5,R3] R12 = MEM8 (A000 000C) = FF FF FF 99
c) LDRH R13, [R5, #12] R13 = MEM16 (A000 000C) = 0000 9922
d) LDR R14, [R5], #8 R14= MEM(A000 0000) = 12 34 56 78 ; R5 = A000 0008
e) LDR R6, [R5, #4] ! R6 = MEM(A000 0004) = FE DC BA 98 ; R5 = A000 0004
f) LDR R7, [R5, #2] ! Accès non aligné
g) STR R1,[R4], #12 MEM(8200 0000) = 9889 AAAA ; R4 = 8200 000C
h) STRB R1, [R4,-R3] MEM8 (81FF FFF4) = AA

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
        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, X
        STR R4,[R0]
        ADD R3,R3,R4
        RSB R3,R3,#N
        ADR R0,Z
        STR R3,[R0]
```

```
i=0; X=0; Y=0;
do{
        If (T[i] <0) X++;
        If (T[i] = =0) Y++;}
While (T[i-1] >0°.
Z= N- X- Y;
```

Le programme itère dans la boucle tant que T[i] >0.
Il sort de la boucle au premier T[i] négatif ou nul.

- dans le premier cas, X=1, Y = 0 et Z = N-1
- dans le second cas X=0, Y =1 et Z=N-1

Si tous les éléments du tableau sont positifs, le programme va provoquer un « core dump » car il n'y a pas de test de fin de tableau.

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.

```
        ADD R3, R1, R0      // R3 <= N
        ADDI R2, R1,-1              //R2 <=N-1
Boucle : MUL R3, R3, R2     // F <= N *N-1 à la première itération
        ADDI R2,R2, -1      // décrémentation de N
        BGT R2,R0, Boucle   // tant que N>0
```

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.

```
        ADD R3, R1, R0      // R3 <= N
        ADDI R2, R1,-1              //R2 <=N-1
Boucle : MUL R3, R3, R2     // F <= N *N-1 à la première itération
        ADDI R2,R2, -1      // décrémentation de N
        BLT R3,R0, Suite    //si R3 négatif, débordement
        BGT R2,R0, Boucle   // tant que N>0
        BEQ R0, R0, Fin
Suite :  ADD R3,R0, R0      // cas débordement
Fin
```

Q 8) Que fait le programme assembleur NIOS suivant ? On indiquera la valeur de R1 en fin d'exécution.

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

Compte le nombre de uns qui étaient initialement dans R2 et met le résultat dans R1.

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 ADD Ri, Rj, Rk Décalage #N	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 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.

ADD	R	ADD rd, rs, rt	rd <= rs + rt (signé)
ADDI	I	ADDI rt, rs, IMM	rt <= rs + SIMM (signé)
ADDIU	I	ADDIU rt, rs, IMM	rt <= rs + SIMM (le contenu des registres est non signé)
ADDU	R	ADDU rd, rs, rt	rd <= rs + rt (le contenu des registres est non signé)
AND	R	AND rd, rs, rt	rd <= rs and rt
ANDI	I	ANDI rt, rs, IMM	rt <= rs and ZIMM
BEQ	I	BEQ rs,rt, déplac.	si rs = rt, branche à ADBRANCH
BGEZ	I	BGEZ rs,déplac.	si rs ≥ 0, branche à ADBRANH
BGEZAL	I	BGEZAL rs, déplac.	adresse de l'instruction suivante dans R31 si rs ≥ 0, branche à ADBRANH
BGTZ	I	BGTZ rs,déplac.	si rs > 0, branche à ADBRANH
BLEZ	I	BLEZ rs,déplac.	si rs ≤ 0, branche à ADBRANH
BLTZ	I	BLTZ rs,déplac.	si rs < 0, branche à ADBRANH
BLTZAL	I	BLTZAL rs, déplac.	adresse de l'instruction suivante dans R31 si rs < 0, branche à ADBRANH
BNEQ	I	BNEQ rs,rt, déplac.	si rs ≠ rt, branche à ADBRANCH
J	J	J destination	Décale l'adresse destination de 2 bits à gauche, concatène aux 4 bits de poids fort de CP et saute à l'adresse obtenue
JAL	J	JAL destination	Même action que J . Range adresse instruction suivante dans R31
JALR	R	JALR rs, rd	Saute à l'adresse dans rs. Range adresse instruction suivante dans rd
JR	R	JR rs	Saute à l'adresse dans rs
LUI	I	LUI rt, IMM	Place IMM dans les 16 bits de poids fort de rt. Met 0 dans les 16 bits de poids faible de rt
LW	I	LW rt, déplac.(rs)	rt <= MEM [rs + IMM]
OR	R	AND rd, rs, rt	rd <= rs or rt
ORI	I	ANDI rt, rs, IMM	rt <= rs or ZIMM
SLL	R	SLL rd, rt, nb	Décale rt à gauche de nb bits et range dans rd
SLT	R	SLT rd, rs, rt	rd <= 1 si rs < rt avec rs signé et 0 autrement
SLTI	I	SLTI rt, rs, IMM	rt <= 1 si rs < SIMM avec rs signé et 0 autrement
SLTIU	I	SLTIU rt, rs, IMM	rt <= 1 si rs < ZIMM avec rs non signé et 0 autrement
SLTU	R	SLTU rt, rs, r	rd <= 1 si rs < rt avec rs et rt non signés et 0 autrement
SRA	R	SRA rd, rt, nb	Décaler (arithmétique) rt à droite de nb bits et ranger dans rd
SRL	R	SRL rd, rt, nb	Décaler (logique) rt à droite de nb bits et ranger dans rd.
SUB	R	SUB rd, rs, rt	rd <= rs - rt (signé)
SUBU	R	SUBU rd rs, rt	rd <= rs - rt(non signé)
SW	I	SW rt, déplac.(rs)	rt => MEM [rs + IMM]
XOR	R	XOR rd, rs, rt	rd <= rs xor rt
XORI	I	XORI rt, rs, IMM	rt <= rs xor ZIMM

Figure 1 : Jeu d'instructions