Hes·so — Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

5.7

Microprocesseurs 1 & 2: Travail écrit no 1.

Nom : 　　**Prénom :** 　　**Classe : I/2**
Date : 23.11.2010

Problème n° 1 (architecture générale)

Pour une organisation de la mémoire est en « **little-endian** », représenter (en hexadécimal pour les entiers et en caractère ascii pour les strings) dans le tableau ci-dessous les variables suivantes :

Adresse :	variable :	taille/type :	valeur :	
0xa00000f8	var1	.long 32	00 82485a$_{16}$	→ 0x 0082485a
0xa0000111	var2	.byte 8	192$_{10}$	→ 0x C0
0xa0000160	var3	.asciz	"hello world"	
0xa0000122	var4	.short 16	417$_8$	→ 0x016F
0xa0000134	var5	.word 32	-2$_{10}$	→ 0x FFFFFFFE

-2$_{10}$ → 2: 0b000010 → ...11111110$_{2c}$ = -2$_{10}$

31–24	23–16	15–8	7–0	
				0xa00000F0
				0xa00000F4
0x 00	0x82	0x48	0x5a	0xa00000F8
				0xa00000FC
				0xa0000100
				0xa0000104
				0xa0000108
				0xa000010C
		0xC0		0xa0000110
				0xa0000114
				0xa0000118
				0xa000011C
0x 01	0x 6F			0xa0000120
				0xa0000124
				0xa0000128
				0xa000012C
				0xa0000130
0xFF	0xFF	0xFF	0xFE	0xa0000134
				0xa0000138
				0xa000013C
				0xa0000140
				0xa0000144
				0xa0000148
				0xa000014C
				0xa0000150
				0xa0000154
				0xa0000158
				0xa000015C
'wol'	'ol'	'l'	'he'	0xa0000160
o	w	'd'	'o'	0xa0000164
[illegible]	d	l	✓	0xa0000168
				0xa000016C
				0xa0000170

0x0 ✓

Microprocesseurs 1 & 2: Travail écrit no 1.

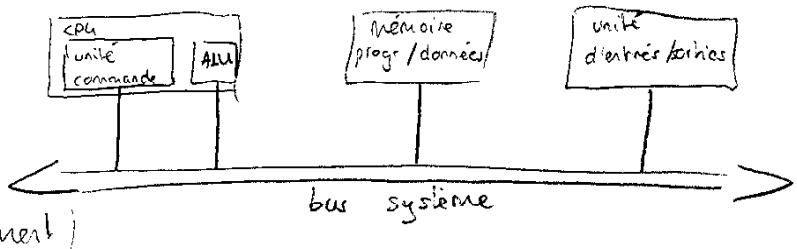
7 **Problème n° 2** (architecture interne)

a) Citer les 6 composants principaux de la structure interne des processeurs ARM.

5 Barrel Shifter ✓
MAU ✓
ALU ✓
Bank Register ✓
IR ✓ (Decode) ✓
Read / Write Register ✓
Adress Register + increment

b) Citer l'architecture utilisée par les processeurs ARM:

1 Von Neumann →
Risc



c) Citer le mode (de fonctionnement)

1 Ce processeur fonctionne sous le mode "Load & Store", c'est à dire qu'il traite uniquement les données depuis ses registres: Pour chaque opération nécessitant d'intéragir avec des données en mémoire, on charge dans un registre (LDR), on traite, puis on remet en mémoire (STR)

d) Citer les étapes d'exécution des instructions par les processeurs ARM

2 ARM 7 (3 étapes)
Fetch
↓
Decode
↓
Execute

ARM 5 (5 étapes)
Fetch
↓
Decode / Read Register
↓
Shift / ALU
↓
Data Memory Access
↓
Write

✓

Microprocesseurs 1 & 2: Travail écrit no 1.

10

Problème n° 3 (traitement numérique des nombres)

a) Prévoir l'état des flags Z, C, N et V ainsi que le résultat contenu dans le registre R0 suite à l'exécution des instructions assembleur suivantes :

Remarque : toutes les opérations sont faites avec des registres de 8 bits au lieu de 32 bits

2

```
1. ldr   r0, =252
   cmp   r0, #-4
```

1111 1100 − 1111 1100 = 0000 0000

Z= 1 C= 1 N=0 V= 0 R0(signé)= −4 R0(non signé) = 252

2

```
2. mov   r0, #129
   adds  r0, #137
```

1 1000 0001 + 1000 1001 = (1) 0000 1010

0111 1111 (+127) + 0111 0111 (+119)

Z= 0 C= 1 N=0 V= 1 R0(signé)= 10 R0(non signé) = 10

2

```
3. mov   r0, #-1
   subs  r0, #127
```

1111 1111 − 0111 1111 = 1000 0000

Z= 0 C= 1 N= 1 V=0 R0(signé)= −128 R0(non signé) = 128

b) Représenter en hexadécimal sur 32 bits (simple précision) les valeurs réelles suivantes et donner le développement :

(pour rappel : exposant est codé sur 8 bits avec un biais de 127)

1,5

a) 35,1875 : 100011,0011 · 2° = 1,00011001100110011 · 2^5

E = 127 + S = 132 → 10000100

M =

S = 0

→ 0100 0010 0000 1111 1010 1001 1000 0000 = 0x420FA980

2

b) 101 / 4096 : 1100101 · 2^{-12} = 1,100101 · 2^{-6}

E = 127 − 6 = 121 → 0111 1001

M =

S = 0

→ 0011 1100 1100 1010 0000 0000 0000 0000 = 0x3CCA0000

Microprocesseurs 1 & 2: Travail écrit no 1.

Problème n° 4 (Mode d'adressage)

Pour le code assembleur et la représentation de la mémoire (little-endian / 8-bits) et l'état des registres du processeur ci-dessous, donner le résultat des opérations (état des registres, état de la mémoire):

Mémoire (little-endian / 8 bits)	
0xa0002000	0x25
0xa0002001	0x83
0xa0002002	0x75
0xa0002003	0x84
0xa0002004	0x87
0xa0002005	0x25
0xa0002006	0x73
0xa0002007	0xc2
0xa0002008	0x00
0xa0002009	0x10
0xa000200a	0x00
0xa000200b	0xa0
0xa000200c	0x F6
0xa000200d	0x FF
0xa000200e	
0xa000200f	

	Registres (avant)	Registres (après)
R0	0xa000'0100	0x0000'3000
R1	0x0000'1000	0x0000'0001
R2	0x0000'0020	0x0000'0020
R3	0xff00'ff00	0xffff'ff87
R4	0xa000'2000	0xa000'200c
R5	0x0000'0001	0x0000'0002
R6	0x0000'000c	0x0000'0003
R7	0xffff'fff6	0x0000'0004
R8	0xa000'2008	0xa000'1fe8
R9	0x0000'0100	0xa000'1000
R10	0x0000'0000	0x0000'0005
R11	0xa000'0100	0x0000'0006
R12	0x0000'0000	0x0000'0007

0000 000c 0010 0000 0000 0000

0. `0xa000'1000: backup: .long 1,2,3,4,5,6,7,8`

2 **1. `add r0, r1, r2,lsl #8`** r0 = r1 + lsl8(R2)

r1 0x0000 1000
lsl8(R2) 0x0000 2000
R0 = 0x0000 3000

1 **2. `ldrsb r3, [r4,#4]`**

valeur à l'addr [R4 +4] → r3 (byte)

0xa000'2004 0x87 → 0xffff'ff87

2 **3. `strh r7, [r4,r6]!`**

stock la valeur de R7, à l'adresse [R4 + R6] et met à jour R4
16bits

0xa000 200c

2 **4. `ldr r9, [r8],-r9,lsr #3`**

1) charge la valeur à l'adresse R8 dans R9
2) post-index : r8 = r8 - lsr3(R9) → 0xa000 2008 − 0x0000 0020 = r8 = 0xa000 1fe8

lsr3(R9)
R9 0x0100
0000 0001 0000 0000
0000 0000 0010 0000
0x0020

2 **5. `ldr r9, =backup`**
`ldmia r9, {r1,r5-r7,r10-r12}`

r9 := &backup

ldmia : incr. after

backup
1 R1
2 R5
3 R6
4 R2
5 R10
6 R11
7 R12
8

Microprocesseurs 1 & 2: Travail écrit no 1.

10

Problème n° 5 (Programmation en assembleur)

Coder en langage assembleur ARM l'algorithme suivant :

```
#define  SIZE  200
char  str[] = "Un message ascii a transferer sans les caracteres speciaux";
char  msg[SIZE+1];
short len = 0;
void main() {
  char c;
  do {
    c = str[len];
    if (c == 0) break;
    len++;
    if ((c <= '\037') || (c >= '\x80')) c = '?';
    msg[len] = c;
  } while (len < SIZE);     ou    len <= SIZE-1
  msg[len+1] = 0; msg[0] = len;
}
---en assembleur--------
SIZE = 200
str: .asciz  "Un message ascii a transferer sans les caracteres speciaux"
msg: .byte   SIZE+1
len: .short  0

main:
    ldr   r0, =str      @ str
    ldr   r1, =msg      @ msg
    ldr   r6, =len      @ len
    ldr   r2, [r6]
    @r3=c

loop: ldrb  r3, [r0, r2]      @ c = str[len]
    cmp   r3, #0              @ if (c == 0)
    beq   end                 @ break
    add   r2, #1              @ len++
    cmp   r3, #'\037'
    bls   spec                @ if (c <= '\037')
    cmp   r3, #'\x80'
    bhs   spec                @ (c >= '\x80')
    beq   spec                @ (c == '\x80')
next: strb  r3, [r1, r2]      @ msg[len] = c
    *
spec: mov   r3, #'?'          @ c = '?'
    b     next
end:  mov   r4, #0            @ msg[len+1] = 0
    add   r5, r2, #1
    strb  r4, [r1, r5]
    strb  r2, [r1]            @ msg[0] = len
    strh  r2, [r6]            @ store len
```

plus assez de place...

```
*   cmp   r2, #SIZE-1
    bls   loop
    b     end
```