Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 : Travail écrit no 1.

4,5

[redacted]

Classe : T-2/I-2
Date : 21.11.2016

8

Problème n° 1 (Programmation en assembleur)

Lors du développement d'une application logicielle, la nécessité de recourir à des fonctions en langage assembleur pour des raisons de performances, c'est fait ressentir. Il a donc été décidé de créer un module contenant les fonctions « copy_words » et « stats» ci-dessous.

```
static long nb_calls = 0;                // variable locale au module
static void stats() { nb_calls++; }      // fonction locale au module

void copy_words (long* dst, long* src, long nb_words) {
    while (nb_words > 0) { nb_words--; dst[nb_words] = src[nb_words]; }
    stats();
}
```

a) Ecrivez, en langage assembleur ARM, toutes les directives nécessaires à l'allocation des ressources et qui permettent un assemblage et un linkage corrects du module et de ses fonctions.

--- en assembleur --------

```
.global  ~~stats;~~
         copy_words;

.data .align 8
nb_calls #0;   .long 0
.bss .align 8
nb_calls;
```

1,5

.text
.align 4

b) Implémentez en assembleur la fonction « stats ».

--- en assembleur --------

2

```
stats:
        ldr r5, =nb_calls;
        ldr r6, [r5];       // r6 = nb_calls
        add r6, r6, #1;     // r6++
        str r6, [r5];       // on stock r6 dans la variable [r5]
        bx  lr;
```

Implémentez en assembleur la fonction « copy_words ». Pour indication, les arguments « dst », « src », qui sont des pointeurs (références) vers des tableaux de « long », et « nb_words » sont passés à la fonction par les registres « R0 », « R1 » et respectivement « R2 ». r0 = dst

--- en assembleur --------

4

```
copy_words:  push {lr}       -3   2f.
1:
        sub r2, r2, #1;  // nb_word--
        ldr r3, [r1, r2, lsl #2]   // dst [nb_word]
        str r3, [r0, r2, lsl #2]   // src [nbord] = r3
2:
        cmp r2, #0;      // while >0 ...
        bne ~~copy_words~~ 1b
        b   stats
        ~~bx lr~~
        pop {pc}
```

Systèmes Embarqués 1 : Travail écrit no 1.

Problème n° 2 (Mode d'adressage)

a) Implémentez les 2 instructions assembleur permettant de sauver le contenu des registres R4 à R12 de la structure « regs » ci-dessous

```
struct {uint32_t r4,r5,r6,r7,r8,r9,r10,r11,r12;} regs;
```

```
str   r0, =regs
stmia r0, {r4-r12}
```

b) Donnez l'instruction assembleur permettant de restaurer sur la pile les registres r4, r5, r6, r7 et pc (les instructions push et pop ne peuvent pas être utilisées).

```
stmfd sp!{r4,r5,r6,r7,pc}   → ldmfd sp!{r4-r7,pc}
```

sauver — restaurer

c) Pour le code assembleur et la représentation de la mémoire (**Little-Endian / 8-bits**) et l'état des registres du processeur ci-dessous, donnez le résultat des opérations (état des registres, état de la mémoire):

Mémoire (little-endian / 8 bits)		(après)
0x80002100	0x34	
0x80002101	0xf5	
0x80002102	0x89	
0x80002103	0xc9	
0x80002104	0x25	
0x80002105	0x94	
0x80002106	0xa5	
0x80002107	0xc2	
0x80002108	0xba	
0x80002109	0x53	
0x8000210a	0x41	0x[illegible]f8
0x8000210b	0x87	

Registres (avant)		Registres (après)
R0	0x0000'0004	
R1	0x0000'3000	
R2	0x0000'000c	0x0000'0300
R3	0x0000'12f8	
R4	0x0000'0002	
R5	0x8000'210a	0x8000'210C
R6	0xffff'fff2	
R7	0x8000'5101	0xFFFF'FFAB
R8	0x8000'20fc	0x8000 20f[illegible]e
R9	0x0302'0100	
R10	0x0403'0200	
R11	0x0504'0300	
R12	0x8000'2008	
SP	0x8000'5110	

1. `lsr r2, r1, r0`

r2 = r1 shifté de r0 → r1 = 0011 0000 0000 0000 (3 0 0 0) → 0000 0011 0000 0000 (0 3 0 0)

2. `strb r3, [r5], r4`

address r5 = r3

ensuite r5 = r5 + r4

3. `ldrsh r7, [r8, #2]!`

① r8 = r8 + 2 On a pas la mémoire de 0x8000'20FE, je suppose qu'elle contient 0xAB

② ~~r8 = address = r7~~ r7 = [r8]

Systèmes Embarqués 1 : Travail écrit no 1.

Problème n° 3 (traitement numérique des nombres)

a) Prévoyez l'état des flags Z, C, N et V ainsi que le résultat contenu dans le registre R2 (en décimal) 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

```
1. ldr    r2, =129
   adds   r2, #127
```

1000 0001
1 0000 0000

Z= 1 C= 1 N= 0 V= 1 R2(non signé)= 0 R2 (signé) = 0

```
2. ldr    r2, #-4
   cmp    r2, #252
```

1111 1100
+ 0000 0100
1 0000 0000

Z= 1 C= 1 N= 0 V= 1 R2(non signé)= 0 252 R2(signé) = 0 -4

```
3. ldr    r2, #-7
   subs   r2, #6
```

1111 1001
+ 1111 1010
1 1111 0011

Z= 0 C= 1 N= 1 V= 0 R2(non signé)= 243 R2(signé) = -13

b) Représentez en hexadécimal sur 32 bits (simple précision) la valeur réelle ci-dessous et donnez le développement (pour rappel : exposant est codé sur 8 bits avec un biais de 127)

-33,125 / 32 :

$100001,001 \cdot 2^0$
$1,00001001 \cdot 2^5 \cdot 2^{-5}$
$1,00001001 \cdot 2^0$

S = 1
E = 127 + 0 = 128

=> 1 0000 0000 0000 1001 000 0000 0000 0000

c) Citez les fanions (flags) utilisés pour tester les conditions des nombres non signés et indiquez l'équation logique sur les fanions pour les opérations « eq » et « ls ».

eq => Z, C et V
ls => Z, V

Systèmes Embarqués 1 : Travail écrit no 1.

6

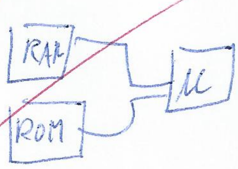
Problème n° 4 (architecture générale)

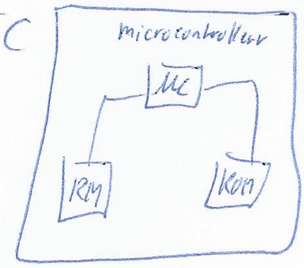
a) Quelle est la différence principale dans l'organisation mémoire d'une application logicielle entre un système embarqué et un système on chip

système embarqué = micro-controlleur = tout dans le même IC

SOC = élément séparé

0,5





b) Citez les 4 architectures de microprocesseurs selon la classification de Flynn (les abréviations)

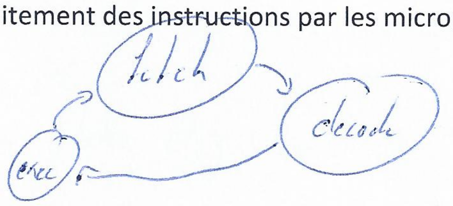
0 ~~Von Neumann Harvard~~ SISD/SIMD/MISD/MIMD

c) Quelle est la différence principale entre l'architecture Von Neumann et l'architecture Harvard

1 bus séparé entre unité de commande – mémoire programme et ALU – mémoire de donnée dans Harvard

d) Citez les 3 cycles principaux du traitement des instructions par les microprocesseurs

1 fetch
decode
exec



e) Pour une organisation de la mémoire en « **Little-Endian** », représentez (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 :
0x80002104	msg:	.asciz	"Bye"
0x8000210a	v1:	.short	-3_{10} = 0000 0000 0000 0011 => 1111 1101 = ffd
0x80002109	v2:	.byte	127_{10} 0111 1111 = 1000 0001 = ~~81~~ 7f
0x8000210c	v3:	.short	$543e_{16}$
0x80002100	v4:	.long	103_8 = 72(10) = 0100 1000 = 1011 0111 = 1011 1000 = ~~b8~~ 43 = ~~0 ff ff~~ b8

	7 … 0
0x80002100	0x b8 — 43
0x80002101	0x ff — 00
0x80002102	0x ff — 00
0x80002103	0x ff — 00
0x80002104	~~0x By~~ — B
0x80002105	~~0x e0~~ — y
0x80002106	e
0x80002107	0
0x80002108	
0x80002109	0x ~~81~~ 7f
0x8000210a	0x fd
0x8000210b	0x ff
0x8000210c	0x 3e
0x8000210d	0x 54
0x8000210e	

3,5

Systèmes Embarqués 1 : Travail écrit no 1.

Problème n° 5 (architecture interne)

a) L'architecture interne du µP ARM Cortex-A8 est réalisée de 7 à 8 composantes principales. Citez et décrivez à l'aide d'une phrase la fonction de trois de ces unités :

- Instruction decode Unit
- Instruction fetch Unit
- Instruction exec Unit

- CP15 - MMU
- L1 I-Cache
- L1 D-Cache
- L2 System memory cache
- µP bus memory system
- NEON media processor

b) Indiquez la fonction/l'usage des différents registres ci-dessous

1. R13 : SP = Stack Pointer
2. R14 : Link register LR
3. R15 : Program Counter PC
4. CPSR : Current Program Status Register

c) Quelle différence fondamentale existe-t-il entre mode de fonctionnement « user » et « supervisor » ?

User est un mode normal qui n'a pas accès au registre privilégié (code programme)

superviser est un mode activé lors de l'execution de code propre à l'OS

d) Citez les caractéristiques principales de l'architecture des µP ARM et plus spécialement du Cortex-A8

~~Adder~~
~~ALU~~
~~Unité de commande~~
~~Mémoire donnée~~
~~Mémoire programme~~

- µP RISC
- load store

e) Citez le mécanisme implémenté par le µP ARM Cortex-A8 pour augmenter ses performances lors de l'exécution d'une instruction pipeline