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

Systèmes Embarqués 1 : Travail écrit no 1.

Nom : Castella **Prénom :** Simon 5.7 **Classe : T-2/I-2**

Date : 09.11.2015

8

Problème n° 1 (Programmation en assembleur)

a) Implementez en langage assembleur ARM le code ci-dessous. Le module assembleur contiendra toutes les directives permettant un assemblage et un linkage correcte du module et de sa fonction. Pour indication, la valeur retournée par la fonction « fnct1 » est réalisée en plaçant cette valeur dans le registre R0.

```
static long sum = 10;
int fnct1() {
    sum += 10;
    return sum;
}
--- en assembleur --------
```

```
.data
sum: .long 10
.text
:fnct1
    ldr r0,=sum
    ldr r1,[r0]
    add r1,r1,#10
    str r1,[r0]
    bx lr
```

b) Implementez en langage assembleur ARM la fonction "fnct2" ci-dessous. Pour information, les deux variables "parity" et "msg" sont externes et ne doivent pas être déclarées dans le code assembleur.

```
extern char parity;
extern char msg[10];

void fnct2() {
   parity = 0;
   for (int i=10; i>0; i--) {
         parity ^= msg[i-1];
   }
}
--- en assembleur --------
```

```
:fnct2
    ldr r0,=parity
    ldrb r0,[r0]
    mov r0,#0
    mov r1,#10
    ldr r2,=msg
    b test
:loop
    ldrb r4,[r2,#-1]!
    EOR R0,r4,r0
    sub r1,#1
:test
    cmp r1,#0
    bne loop
    ldr r1,=Parity
    strb r0,[r1]
    bx lr
```

Systèmes Embarqués 1 : Travail écrit no 1.

9

Problème n° 2 (Mode d'adressage)

a) Donnez les 2 instructions assembleur permettant de restaurer le contenu des registres R0 à R12 de la structure « s » ci-dessous

```
struct S {uint32_t r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12;} s;
```

ldr r0,=S
1,1 ldmia r0, {r0-r12}

ldmia r0 {r0-r12}

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

2 stmfd sp! {r4,r5,r6,lr}

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	0x F8
0x80002109	0x53	0x 12 ✓
0x8000210a	0x41	
0x8000210b	0x87	

	Registres (avant)	Registres (après)
R0	0x0000'0001	0xFFFFFF~~F~~8
R1	0x0038'3004	0x0038'3004 ✓
R2	0x0000'000c	
R3	0x0000'12f8	0x0000'12BA
R4	0x0000'0002	
R5	0x8000'2104	0x8000'2106 ✓
R6	0x8000'2108	0x8000'210a ✓
R7	0x8000'5101	
R8	0x8000'3008	
R9	0x0302'0100	
R10	0x0403'0200	
R11	0x0504'0300	
R12	0x8000'2008	
SP	0x8000'5110	

1. `add r1,r1,r0,lsl #8` r0 << 8 1'000'0000

2

2. `ldrsb r0, [r5, r4]!`

1,5
1010 0101
0101 1010
0101 1011
5 b

3. `strh r3, [r6],#2`

2
0x8000'210a
mais après -> itt
post-incrément

Systèmes Embarqués 1 : Travail écrit no 1.

10

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 (<u>en décimal</u>) suite à l'exécution des instructions assembleur suivantes :

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

1. `ldr r2, =129`
   `subs r2, #3`

0 111 1111
1000'0001
0000'0011
0111 1110

V +++ −
−++ −
−−+ +
+−− −

1,5

Z=0 C=1 N=0 V=0 1 R2(non signé)= 126 R2 (signé) = 126

2. `mov r2, #251`
   `cmp r2, #-5`

1111 1011
1111 1011

2

Z=1 C=1 N=0 V=0 R2(non signé)= 257 R2(signé) = −5

3. `mov r2, #-7`
   `adds r2, #6`

−++
1111'1001
0000'0110
1111 1111

2

Z=0 C=0 N=1 V=0 R2(non signé)= 255 R2(signé) = −1

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)

-65,625 / 16 :

S=1 65,625= 1000001.101 = 1,0000011101 . 2^6 . 2^{-4}

E=127+2=129 10000001

= C0834000

2

1100 0000 1000 0011 0100 0000 0000

C 0 8 3 4 0 00

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

8 10

signé	non-signé
N Z V	C Z
ne $\overline{Z}=1$	ne $\overline{Z}=1$
lo N=1 V=0 Z=0	lo Z=0 && C=0

2

Systèmes Embarqués 1 : Travail écrit no 1.

9

Problème n° 4 (architecture générale)

a) Citez les 6 segments principaux d'une application une fois chargée dans la mémoire principale d'un microprocesseur, soit les 4 sections contenues dans le code et les 2 lors de l'exécution, et indiquez ce qu'elles contiennent

. row data ✓ -> Constante
. bss ✓ -> variable non initialisée
. data ✓ -> variable initialisée
. text ✓ -> code
~~fichier .s code -> binaire~~

2

- heap : donnée dynamique
stack : ...

b) Citez les 2 architectures fondamentales des microprocesseurs SISD selon la classification de Flynn

-> Neuman
-> Harward

1

c) Citez les 3 cycles principaux du traitement de l'information par les microprocesseurs

Fetch -> Decode -> Execute

1

d) 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 :
0x80002102	text:	.ascii	"hello"
0x80002107	code:	.byte	-5_{10}
0x8000210c	crc:	.short	517_{10}
0x80002108	val:	.long	$1b543e_{16}$
0x80002100	parity:	.short	1003_8

11111011 -> FF|FF|FF|FB
F B

001 000 000 011
0010 0000 0011
0 2 0 3

Adresse	7 … 0
0x80002100	0x 03
0x80002101	0x 02 ✓
0x80002102	H
0x80002103	E
0x80002104	L
0x80002105	L ✓
0x80002106	O
0x80002107	0x ~~FFFFFF~~ FB ✓
0x80002108	0x 3E
0x80002109	0x 54
0x8000210a	0x 1B
0x8000210b	0x 00 ✓
0x8000210c	0x 05
0x8000210d	0x 02 ✓
0x8000210e	

5

<- (normalement le Ø de la chaîne de caractère)

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

Systèmes Embarqués 1 : Travail écrit no 1.

8

Problème n° 5 (architecture interne)

a) Citez ou dessinez les composants principaux de l'architecture interne du µP ARM Cortex-A8

- ALU -> Processeur arithmétique Neon
- Banque de registre
- Barrel shifter
- Decode unit, Fetch unit, Execute unit
- Unit commande

1,5

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

i. R13 : SP -> stack pointer
ii. R14 : lr -> adresse de retour.
iii. R15 : pc -> program counter -> prochaine ligne d'exection
iv. CPSR : Flag N, C, V, Z, ... -> Màj quand on fait cmp subs
v. SPSR : ~~CPSR de la stack~~ (sauvegarde du CPSR) adds, ...

c) Décrivez succinctement l'utilité des modes de fonctionnement ci-dessous

i. User : Mode utilisateur. C'est le mode que nous utilisons au Labo
ii. Superviseur : Propre à l'OS -> Code que l'os doit exécuter
iii. IRQ : Mode servant les interruptions

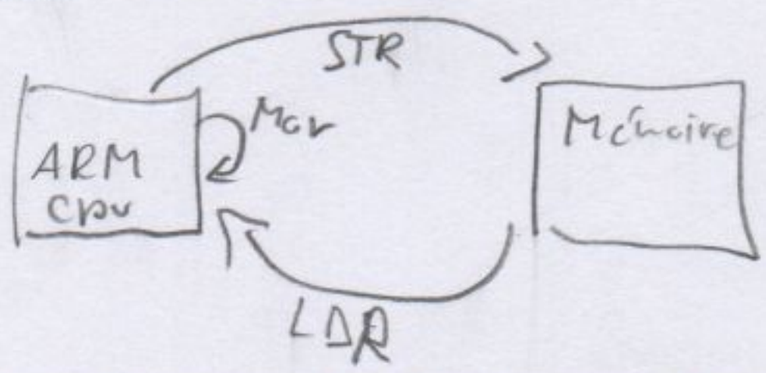
1

* Exécution des actions de l'utilisateur de la machine

d) Décrivez le principe de fonctionnement du µP ARM Cortex-A8

load and store

2



e) Décrivez succinctement le principe de fonctionnement du pipelining

F = Fetch
D = Decode
E = Execute

1,1