UNIVERSITE GRENOBLE ALPES
Licence de Sciences et technologie - Parcours MIN, BIN, INF, IAG

Contrôle Continu UE INF401 : Introduction aux Architectures Logicielles et Matérielles

Mars 2018, durée 1 h 30
Document : 1 A4 R/V manuscrit autorisé ; calculettes et téléphones portables interdits.
La plupart des questions sont indépendantes, si vous avez du mal avec l'une, passez à la suivante.
Le barème est donné à titre indicatif.

1 Numération en complément à 2 (5 points)

Pour cet exercice, **donnez tous les détails de calcul et justifiez vos réponses** (poser chaque opération avec les opérandes, les retenues, le résultat apparent et les indicateurs).

Questions :

(a) Donner les valeurs décimales des 2 entiers relatifs suivants codés sur 16 bits en complément à 2 : $(0000\ 0111\ 1001\ 1100)_2$ et $(FFBA)_{16}$. **(2 points)**

(b) Donner la représentation binaire et hexadécimale des 3 entiers relatifs suivants (codage en complément à 2 sur 1 octet) : $(+45)_{10}$, $(-45)_{10}$ et $(+77)_{10}$. **(1.5 point)**

(c) Poser chacune des 3 additions binaires suivantes sur 1 octet, donner le résultat binaire et apparent (codage complément à 2 sur 1 octet) et la valeur des indicateurs (Z, N, C et V) : $(-45)_{10} + (-45)_{10}$, $(+45)_{10} + (-77)_{10}$ et $(+77)_{10} + (+77)_{10}$. **(1,5 point)**

2 Programmation en ARM (10 points)

Un nombre entier A naturel est un *palindrome binaire sur x* bits si sa représentation en binaire sur x bits, $a_0, a_1, \ldots a_{x-1}$, est un palindrome, c'est-à-dire $a_0, a_1, \ldots a_{x-1} = a_{x-1}, a_{x-2}, \ldots a_0$. Par exemple, 129 est un palindrome binaire sur 8 bits, en effet $(129)_{10} = (10000001)_2$, alors que 94 n'est pas un palindrome, en effet $(94)_{10} = (01011110)_2$.

Le but de cet exercice est d'afficher parmi les valeurs d'un tableau d'entiers naturels sur 8 bits celles qui sont des palindromes binaires (sur 8 bits).

Dans la suite, on considère la zone `.data` suivante :

```
    .data
msg:    .asciz "palindromes :"
tab:
    .byte 94
    .byte 7
    .byte 129
    .byte 34
    .byte 58
    .byte 36
```

De plus, vous devrez compléter la zone `.text` suivante :

```
    .text
    .global main
main:

        @ à compléter

        bal exit
ptr_tab: .word tab
ptr_msg: .word msg
```

Questions.

(a) Rappeler sur combien d'octets est stockée une chaîne de x caractères déclarée avec la directive `.asciz` **(0.5 point)**

(b) En supposant que la zone `.data` est stockée en mémoire à partir de l'adresse hexadécimale $(AFF4)_{16}$, quelle est l'adresse du premier élément du tableau `tab` ? **(1 point)**

On considère le morceau de code `ARM` suivant :

```
1. mov r2,#94 @ une valeur du tableau, n.b. 94 s'écrit en binaire : 0101 1110
2. mov r3,r2
3. mov r4,r2
4. mov r5,#0

5. tp: cmp r5,#4
6.     bge fin

7.     mov r6,r3, lsr #7
8.     and r6,r6,#1

9.     and r7,r4,#1

10.    cmp r6,r7
11.    bne fin

12.    mov r3,r3,lsl #1
13.    mov r4,r4,lsr #1
14.    add r5,r5,#1
15.    bal tp

16.    fin:
```

(c) Donnez dans un tableau les valeurs des registres $r3$, $r4$, $r5$, $r6$ et $r7$ juste après l'exécution de la ligne 10, à chaque fois que cette dernière est exécutée. **(3 points)**

(d) Que peut-on dire de la valeur de $r5$ à la fin du code lorsque le nombre initialement dans $r2$ est un palindrome binaire ? **(0.5 point)**

(e) En utilisant le code précédent et les indications qui suivent, traduisez en `ARM` l'algorithme suivant. **(5 points)**

```
EcrChaine "palindromes :"
Pour i=0; i<6;i++
    si tab[i] est un palindrome binaire alors
       EcrNdecimal8 tab[i]
    fin si
Fin pour
```

Indications :

- Pour la variable i vous utiliserez le registre $r9$.
- Nous rappelons aussi que `EcrChaine` et `EcrNdecimal8` suivent la convention adoptée dans le fichier `es.s` utilisé en TP (un rappel de ces conventions est donné en annexe du sujet).
- Lorsque vous utilisez des morceaux du code de la question (c), ne recopiez pas tout, indiquez seulement le début et la fin, ajoutez "... (voir question c)" entre les deux, par exemple, si vous voulez reprendre de la ligne 5 à 11 :

```
début_de_votre_code
5. tp: cmp r5, #4
   ... (voir question c)
11. bne fin
suite_de_votre_code
```

3 Représentation des nombres réels (5 points)

L'exercice repose sur une adaptation de la norme IEEE 754 définissant une représentation des nombres réels en binaire similaire à la notation scientifique des réels (ex. : 2.99e8), mais en binaire. La norme IEEE 754 est décrite par wikipédia ainsi (conseil : en première approche, une lecture très rapide [en diagonale] est suffisante, vous pourrez revenir au texte si nécessaire plus tard) :

L'IEEE 754 est une norme pour la représentation des nombres à virgule flottante en binaire. Elle est la norme la plus employée actuellement pour le calcul des nombres à virgule flottante dans le domaine informatique, avec les CPU et les FPU. La norme définit les formats de représentation des nombres à virgule flottante (signe, mantisse, exposant, nombres dénormalisés) et valeurs spéciales (infinis et NaN), en même temps qu'un ensemble d'opérations sur les nombres flottants. [...] La version 1985 de la norme IEEE 754 définit un format simple précision (32 bits : 1 bit de signe S, 8 bits d'exposant E (-126 à 127), 23 bits de mantisse M, avec bit 1 implicite) SEEE EEEE EMMM MMMM ... MMMM ; [...] L'exposant -127 (qui est décalé vers la valeur 0) est réservé pour zéro et les nombres dénormalisés, tandis que l'exposant 128 (décalé vers 255) est réservé pour coder les infinis et les NaN. Un nombre flottant normalisé a une valeur v donnée par la formule suivante :

$$v = s \times 2^e \times m.$$

- $s = +/-1$ représente le signe (selon le bit de signe) ;
- e est l'exposant avant son décalage de 127 ;

— $m = 1 + mantisse \times 2^{-23}$ représente la partie significative (en binaire), d'où $1 \leq m < 2$ (mantisse codant la partie décimale de la partie significative, comprise entre 0 et 1)

Par exemple pour 0 01111100 01000000000000000000000 : le signe est nul, l'exposant est 124 - 127 = -3, et la partie significative est 1,01 en binaire donc 1,25 en décimal ($1\times2^0+0\times2^{-1}+1\times2^{-2}$) ; le nombre représenté est donc $+1,25 \times 2^{-3}$ soit +0,15625.

Les nombres dénormalisés suivent le même principe, sauf que $e = -126$ et $m = 0 + mantisse \times 2^{-23}$ (attention : pour le calcul, on veillera à prendre $e = -126$ et non -127, ceci afin de garantir la continuité de cette représentation avec la représentation normalisée, puisque $m = 0+mantisse\times2^{-23}$ et non plus $m = 1 + mantisse \times 2^{-23}$).

(d'après wikipedia)

Pour cet exercice, nous utiliserons une représentation simplifiée des nombres à virgule flottante sur 1 octet en nous inspirant de la norme IEEE 754 ; entre autres, il n'y aura pas de cas particuliers pour les petits et grands exposants, *i.e.* il n'y aura pas de nombres dénormalisés.

Un octet [SEEE EMMM] représentera le nombre ayant pour valeur

$$signe \times 2^{exposant} \times chiffres\_significatifs$$

soit :

$$(-1)^S \times 2^{(EEEE)_2-8} \times (1 + (MMM)_2 \times 2^{-3})$$

Exemple : [0100 1100] est positif (S=0), son exposant vaut $(1001)_2 - 8 = 9 - 8 = 1$, ses chiffres significatifs représentent $(1.100)_2$; sa valeur est donc $(11)_2 = 3$.

Questions :

(a) Donner la valeur de [0101 0010] et la représentation de -0.625. **(2 points)**

(b) La valeur 0 ne peut pas être représentée, vrai/faux ? Pourquoi ? Décrire la solution employée pour la norme IEEE 754 pour représenter le 0, comment s'appliquerait-elle pour la représentation de cet exercice. **(1.5 point)**

(c) Dénombrer le nombre total de valeurs positives représentables par le format décrit dans cet exercice), parmi elles combien sont plus grandes ou égales à 1, et parmi elles combien sont entières ? **(1,5 points)**

(Bonus) Parmi les valeurs entières positives, combien sont paires ? **(1 points hors barème)**

ANNEXES

A Instructions du processeur ARM

Code	Nom	Explication du nom	Opération	Remarque
0000	`AND`	AND	et bit à bit	
0001	`EOR`	Exclusive OR	ou exclusif bit à bit	
0010	`SUB`	SUBstract	soustraction	
0011	`RSB`	Reverse SuBstract	soustraction inversée	
0100	`ADD`	ADDition	addition	
0101	`ADC`	ADdition with Carry	addition avec retenue	
0110	`SBC`	SuBstract with Carry	soustraction avec emprunt	
0111	`RSC`	Reverse Substract with Carry	soustraction inversée avec emprunt	
1000	`TST`	TeST	et bit à bit	pas `rd`
1001	`TEQ`	Test EQuivalence	ou exclusif bit à bit	pas `rd`
1010	`CMP`	CoMPare	soustraction	pas `rd`
1011	`CMN`	CoMpare Not	addition	pas `rd`
1100	`ORR`	OR	ou bit à bit	
1101	`MOV`	MOVe	copie	pas `rn`
1110	`BIC`	BIt Clear	et not bit à bit	
1111	`MVN`	MoVe Not	not (complément à 1)	pas `rn`
	`Bxx`	Branchement		xx = condition Cf. table ci-dessous
	`BL`	Branchement à un sous-programme		adresse de retour dans r14=LR
	`LDR`	"load"		
	`STR`	"store"		

L'opérande source d'une instruction `MOV` peut être une valeur immédiate notée `#5` ou un registre noté `Ri`, `i` désignant le numéro du registre. Il peut aussi être le contenu d'un registre sur lequel on applique un décalage de `k` bits ; on note `Ri, DEC #k`, avec `DEC` $\in$ {`LSL`, `LSR`, `ASR`, `ROR`}.

B Codes conditions du processeur ARM

La table suivante donne les codes de conditions arithmétiques `xx` pour l'instruction de branchement `Bxx`.

cond	mnémonique	signification	condition testée
0000	EQ	égal	Z
0001	NE	non égal	$\overline{Z}$
0010	CS/HS	$\geq$ dans N	C
0011	CC/LO	$<$ dans N	$\overline{C}$
0100	MI	moins	N
0101	PL	plus	$\overline{N}$
0110	VS	débordement	V
0111	VC	pas de débordement	$\overline{V}$
1000	HI	$>$ dans N	$C \wedge \overline{Z}$
1001	LS	$\leq$ dans N	$\overline{C} \vee Z$
1010	GE	$\geq$ dans Z	$(N \wedge V) \vee (\overline{N} \wedge \overline{V})$
1011	LT	$<$ dans Z	$(N \wedge \overline{V}) \vee (\overline{N} \wedge V)$
1100	GT	$>$ dans Z	$\overline{Z} \wedge ((N \wedge V) \vee (\overline{N} \wedge \overline{V}))$
1101	LE	$\leq$ dans Z	$Z \vee (N \wedge \overline{V}) \vee (\overline{N} \wedge V)$
1110	AL	toujours	

C Fonctions d'entrée/sortie

Nous rappelons les principales fonctions d'entrée/sortie du fichier `es.s`.

Les fonctions d'affichages :

— `bl EcrHexa32` affiche le contenu de `r1` en hexadécimal.
— `bl EcrZdecimal32` affiche le contenu de `r1` en décimal sous la forme d'un entier relatif de 32 bits.
— `bl EcrZdecimal16` affiche le contenu de `r1` en décimal sous la forme d'un entier relatif de 16 bits.
— `bl EcrZdecimal8` affiche le contenu de `r1` en décimal sous la forme d'un entier relatif de 8 bits.
— `bl EcrNdecimal32` affiche le contenu de `r1` en décimal sous la forme d'un entier naturel de 32 bits.
— `bl EcrNdecimal16` affiche le contenu de `r1` en décimal sous la forme d'un entier naturel de 16 bits.
— `bl EcrNdecimal8` affiche le contenu de `r1` en décimal sous la forme d'un entier naturel de 8 bits.
— `bl EcrChaine` affiche la chaîne de caractères dont l'adresse est dans `r1`.

Les fonctions de saisie clavier :

— `bl Lire32` récupère au clavier un entier 32 bits et le stocke à l'adresse contenue dans `r1`.
— `bl Lire16` récupère au clavier un entier 16 bits et le stocke à l'adresse contenue dans `r1`.
— `bl Lire8` récupère au clavier un entier 8 bits et le stocke à l'adresse contenue dans `r1`.
— `bl LireCar` récupère au clavier un caractère et stocke son code ASCII à l'adresse contenue dans `r1`.

D Table des codes ascii en décimal

32	␣	33	!	34	"	35	#	36	$	37	%	38	&	39	’
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	‘	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	del