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

Contrôle Continu UE INF451 : Introduction aux Architectures Logicielles et Matérielles

9 Mars 2017, durée 1 h 30
Documents, calculettes, téléphones portables interdits
Le barême est donné à titre indicatif

1 Numération (4 points)

Pour cet exercice, **toutes les réponses devront être justifiées** (poser chaque opération avec les opérandes, les retenues, le résultat apparent et les indicateurs).

(a) Poser chacune des additions binaires sur 8 bits suivantes : $(+122)_{10} + (-33)_{10}$ et $(+122)_{10} + (+42)_{10}$,
   1. **(1 point)** pour le résultat apparent sur 8 bits (codage en complément à deux),
   2. **(1 point)** pour les indicateurs Z, N, C et V.

(b) Poser chacune des soustractions binaires sur 8 bits suivantes : $(+122)_{10} - (+42)_{10}$ et $(-33)_{10} - (+122)_{10}$,
   1. **(1 point)** pour le résultat apparent sur 8 bits (codage en complément à deux),
   2. **(1 point)** pour les indicateurs Z, N, C et V.

   Les soustractions seront réalisées par addition du complémentaire.

2 Questions de cours (4 points)

(c) Supposons que l'on dispose d'une mémoire de 2^{32} octets. Sachant qu'il faut pouvoir adresser tous les octets, combien de bits, au minimum, sont nécessaires pour stocker une adresse de cette mémoire ? **(0,25 point)**

(d) Citez les trois entités principales du modèle Von Neumann. **(0,75 point)**

(e) Rappelez ce qu'est une mémoire volatile. Donnez un exemple de mémoire volatile. **(0,75 point)**

(f) Citez trois types de mémoires centrales. **(0,75 points)**

(g) Expliquez comment les bus sont utilisés pour permettre au processeur de faire une lecture en mémoire. **(1,5 point)**

3 Programmation en ARM (12 points)

3.1 Comprendre un programme en ARM (4 points)

```
 1: .data
 2: X:  .byte 8
 3: Y:  .byte 5
 4: .text
 5: .global main
 6: main:
 7: ldr r2,ptrX
 8: ldrb r2,[r2]
 9: ldr r3,ptrY
10: ldrb r3,[r3]
11: mov r1,r2
12: bl EcrNdecimal32
13: mov r1,r3
14: bl EcrNdecimal32
15: add r2,r2,r3
16: sub r3,r2,r3
17: sub r2,r2,r3
18: mov r1,r2
19: bl EcrNdecimal32
20: mov r1,r3
21: bl EcrNdecimal32
22:     bal exit
23: ptrX: .word X
24: ptrY: .word Y
```

(h) Expliquez l'effet des instructions des lignes 7 et 8. **(1 point)**

(i) Quelle serait la valeur de `r2` après l'exécution des lignes 7-8 si on remplaçait la ligne 8 par `ldrh r2,[r2]`. **(1 point)**

(j) Quelles valeurs (et dans quel ordre) sont affichées en ligne 19 et 21. **(2 points)**

3.2 Ecrire un programme en ARM (8 points)

On considère le code à trous suivant :

```
.data
P:      .asciz "palindrome"
NP:     .asciz "pas un palindrome"
.balign 4
TAB:    .word 4
        .word 7
        .word 7
        .word 5
        .word 7
        .word 7
        .word 4
.text   .global main
main:
        @ partie a completer en question (o)
bal exit
ptrP:   .word P
ptrNP:  .word NP
ptrTAB: .word TAB
```

(k) Rappelez comment la chaîne "palindrome" est stockée en mémoire (faites un dessin). **(0,5 point)**

(l) Quelle est la signification de `.balign 4` et pourquoi est-ce nécessaire dans cette zone `.data` ? **(0,5 point)**

(m) Quel calcul arithmétique est effectué par l'instruction `mov r3,r1,lsl #2` ? **(0,5 point)**

(n) En supposant que `TAB` est stocké à partir de l'adresse `0x2A8C`, quelle est l'adresse de `TAB[2]` ? **(0,5 point)**

(o) Complétez le code à trous pour réaliser l'algorithme ci-dessous. Pour les variables i, n et m, de type entier naturel, vous utiliserez les registres `r1`, `r9` et `r2`, respectivement. Nous rappelons aussi que `EcrChaine` suit la convention adoptée dans le fichier `es.s` utilisé en TP (un rappel de ces conventions est donné en annexe du sujet). **(6 points)**

```
i := 0
n := 7
m := n/2

tant que i < m et TAB[i] = TAB[n-i-1] faire
    i := i+1
fin tant que

si i = m alors
    EcrChaine "palindrome"
sinon
    EcrChaine "pas un palindrome"
fin si
```

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