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

Contrôle Continu UE INF401 : Architectures des ordinateurs

Mars 2020, durée 1 h 30
Document : 1 A4 R/V personnel 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, sur 21 !, est donné à titre indicatif.

1 Numération en complément à 2 (6 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) Déterminer le nombre de bits minimum pour représenter chacun des nombres suivants puis donner la représentation binaire et hexadécimale de ces entiers relatifs avec le nombre de bits choisi précédemment (codage en complément à 2) : $(+1337)_{10}$, $(-1337)_{10}$. **(2 points)**

Réponse : Il faut l'intervalle [-2048,+2047] donc 12 bits

$(+1337)_{10} = (0101\ 0011\ 1001)_2 = (5\ 3\ 9)_{16}$

Complément à 1 : $(1010\ 1100\ 0110)_2 = (A\ C\ 6)_{16}$
Complément à 2 : $(1010\ 1100\ 0111)_2 = (A\ C\ 7)_{16}$
D'où $(-1337)_{10} = (1010\ 1100\ 0111)_2$

(b) Donner les valeurs décimales des 2 entiers relatifs suivants codés sur 16 bits en complément à 2 : $(0000\ 1100\ 0100\ 0001)_2$ et $(FF8D)_{16}$. **(2 points)**

Réponse : $(0000\ 1100\ 0100\ 0001)_2$ donne en décimal 3137
$(FF8D)_{16}$ est négatif donc il faut le complément à 2 pour avoir sa valeur entière : $(FF8D)_{16} = (1111\ 1111\ 1000\ 1101)_2$
Complément à 1 : $(0000\ 0000\ 0111\ 0010)_2$
Complément à 2 : $(0000\ 0000\ 0111\ 0011)_2 = (64 + 32 + 16 + 2 + 1)_{10}$
D'où $(FF8D)_{16} = \S(-115)_{10}$

(c) Poser chacune des opérations 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 (ou E pour la vraie soustraction) et V) : $(+47)_{10} + (-77)_{10}$, $(-57)_{10} - (+77)_{10}$ (vraie soustraction) et $(-47)_{10} - (+57)_{10}$ (soustraction par addition du complémentaire) . **(2 points)**

```
+47 = 0010 1111
Complement a 1 : 1101 0000
-47 = 1101 0001

+57 = 0011 1001
```

```
complement a 1 : 1100 0110
-57 = 1100 0111

+77 = 0100 1101
complement a 1 : 1011 0010
-77 = 1011 0011


+47 + -77
   00111 111
    0010 1111
    1011 0011
    1110 0010
C=0,N=1,V=0,Z=0

-57 - +77  (vraie soustraction)
   01111 000
    1100 0111
    0100 1101
    0111 1010

E=0,N=0,V=1,Z=0

-47 - +57
   11000 111
    1101 0001
    1100 0111
    1001 1000
C=1,N=1,V=0,Z=0
```

2 Programmation en ARM (11 points)

2.1 A lire attentivement :

Nous rappelons que le code ASCII du caractère A se note 'A'. De plus en langage d'assemblage `ARM`, la valeur immédiate représentant le code ASCII du caractère 'A' se note #'A'.

Nous rappelons qu'en langage d'assemblage `ARM` (comme dans la plupart des langages) une chaîne de n caractères est représentée par un tableau d'au moins $n+1$ octets : les n premiers correspondent aux codes ASCII des caractères de la chaîne et le $n+1^{\text{ème}}$ est l'octet 0. Par exemple la chaîne "Bonjour" peut être représentée par un tableau de 8 cases (indicée de 0 à 7), contenant successivement les codes ASCII 'B', 'o', 'n', 'j', 'o', 'u', 'r' et enfin l'octet 0.

2.2 Code César

Le but de cet exercice est de décoder, puis afficher, un message secret chiffré suivant la technique dite du « code de César ». Pour cela, vous devrez compléter le code donné ci-dessous. Il n'est pas nécessaire de connaitre la technique du code de César pour faire l'exercice, l'algorithme est donné.

```
.data
              @ 1234567890123456
SECRET: .asciz "KXKT A'PGRWX !!!"
        .balign 4
X:      .word 15

        .text .global main
main:
            @ partie manquante
      b exit

ptr_SECRET: .word SECRET
ptr_X:      .word X
```

Questions :

(d) Rappelez l'effet de la directive `.balign 4` **(0,5 point)**

Réponse : Cette directive corrige l'alignement : la prochaine adresse utilisée dans le segment sera la première adresse libre qui est multiple de 4.

(e) En supposant que l'adresse de chargement du segment `.data` est `0x6000` (en hexadécimal), donnez et justifiez l'adresse de la variable `X`. **(1 point)**

Réponse : La chaîne SECRET contient 16 caractères, donc, en tenant compte du caractère de fin de chaîne, elle occupe un tableau de 17 octets, c-à-d, de l'adresse `0x6000` à l'adresse `0x6011`. Donc la premère adresse libre est `0x6012` mais celle-ci n'est pas multiple de 4. Donc la correction d'alignement donne l'adresse `0x6014`.

Le but est maintenant de traduire en `ARM` l'algorithme suivant :

```
 1: EcrChaine(SECRET)
 2: i:=0
 3: tant que SECRET[i] != 0 faire
 4:       c := SECRET[i]
 5:       si c >= 'A' et c <= 'Z' alors
 6:            c := c - X
 7:            si c < 'A' alors
 8:                 c := c + 'Z' - 'A' + 1
 9:            fin si
10:       fin si
11:       SECRET[i] := c
12:       i=i+1
```

```
13: fin tant que
14: EcrChaine(SECRET)
```

(f) Donnez le code ARM des instructions pour la ligne 1 (pour EcrChaine, vous appliquerez les conventions de es.s utilisée en TP notamment, *cf.* annexe). **(1 point)**

```
ldr r1,ptr_SECRET @ r1:=&SECRET
bl EcrChaine
```

(g) En supposant que la variable c est représentée par le registre r3, donnez le code ARM des instructions de la ligne 8. **(1 point)**

```
add r3,r3,#'Z'
sub r3,r3,#'A'
add r3,r3,#1
```

(h) En supposant que la variable c est représentée par le registre r3 et que la variable i est représentée par le registre r0, donnez le code ARM des instructions en lignes 4 et 11. Vous utilisez le registre r1 pour les calculs intermédiaires éventuels. **(1 point)**

Ligne 4 :

```
ldr r1, ptr_SECRET @ r1 := &SECRET
ldrb r3, [r1,r0]   @ r3 := SECRET[i]
```

Ligne 11 :

```
ldr r1, ptr_SECRET @ r1 := &SECRET
strb r3, [r1,r0]   @ SECRET[i] := r3
```

(i) En supposant que la variable c est un entier relatif représenté par le registre r3, donnez le code ARM du « si » en lignes 7-9. Ne recopiez pas le code ARM de la ligne 8, inscrivez simplement à la place @ ici ligne 8. **(1 point)**

```
cmp r3, #'A'
bcs finsi
@ ici ligne 8
finsi:
```

(j) Donnez le code ARM permettant de stocker dans le registre r2 la valeur de la variable X. Vous utilisez le registre r1 pour les calculs intermédiaires éventuels. **(1 point)**

```
ldr r1,ptr_X @ r1 := &X
ldr r2,[r1]  @ r2 := mem[r1]
```

(k) En supposant que la variable c est un entier relatif représenté par le registre r3 et que la valeur de X est dans le registre r2, donnez le code des instructions en lignes 5-10. Pour les lignes 7-9, écrivez simplement @ ici lignes 7-9. **(1,5 point)**

```
cmp r2,#'A'
bcc finsi2
cmp r2,#'Z'
bhi finsi2
sub r3,r3,r2
@ ici lignes 7-9
finsi2:
```

(l) Donnez le code de la boucle « tant que » (lignes 2-13). Vous utiliserez le registre `r0` pour la variable `i` et le registre `r1` pour stocker l'adresse de chaîne `SECRET`. Vous utilisez le registre `r4` pour les calculs intermédiaires éventuels. Ne répeter pas le code des lignes 4-11 (qui a déjà été demandé), à la place inscrivez simplement `@ ici lignes 4-11`. **(2 points)**

```
mov r0,#0
ldr r1,ptr_SECRET
tantque: ldrb r4,[r1]
         cmp r4,#0
         beq fintantque
         @ ici lignes 4-11
         add r0,r0,#1
         b tantque
fintantque:
```

(m) Quels sont les messages affichés par le programme ? **(1 point)**

Réponse : « KXKT A'PGRWX!!! » puis « VIVE L'ARCHI!!! ».

3 Codage base64. (4 points)

« En informatique, base64 est un codage de l'information utilisant 64 caractères, choisis pour être disponibles sur la majorité des systèmes. Défini en tant qu'encodage MIME dans le RFC 2045, il est principalement utilisé pour la transmission de messages (courrier électronique et forums Usenet) sur Internet. Il est par ailleurs défini en propre dans le RFC 4648.

Description : Un alphabet de 65 caractères est utilisé pour permettre la représentation de 6 bits par un caractère. Le 65e caractère (signe « = ») n'est utilisé qu'en complément final dans le processus de codage d'un message.

Ce processus de codage consiste à coder chaque groupe de 24 bits successifs de données par une chaîne de 4 caractères. On procède de gauche à droite, en concaténant 3 octets pour créer un seul groupement de 24 bits (8 bits par octet). Ils sont alors séparés en 4 nombres de seulement 6 bits (qui en binaire ne permettent que 64 combinaisons). Chacune des 4 valeurs est enfin représentée (codée) par un caractère de l'alphabet retenu. (Table ci-dessous.)

Ainsi 3 octets quelconques sont remplacés par 4 caractères, choisis pour être compatibles avec tous les systèmes existants.

Chaque valeur (chaque groupe de 6 bits) est utilisée comme index dans la table ci-dessous. Le caractère correspondant est indiqué dans la colonne codage. »

Valeur	Codage	Valeur	Codage	Valeur	Codage	Valeur	Codage
0	000000 A	17	010001 R	34	100010 i	51	110011 z
1	000001 B	18	010010 S	35	100011 j	52	110100 0
2	000010 C	19	010011 T	36	100100 k	53	110101 1
3	000011 D	20	010100 U	37	100101 l	54	110110 2
4	000100 E	21	010101 V	38	100110 m	55	110111 3
5	000101 F	22	010110 W	39	100111 n	56	111000 4
6	000110 G	23	010111 X	40	101000 o	57	111001 5
7	000111 H	24	011000 Y	41	101001 p	58	111010 6
8	001000 I	25	011001 Z	42	101010 q	59	111011 7
9	001001 J	26	011010 a	43	101011 r	60	111100 8
10	001010 K	27	011011 b	44	101100 s	61	111101 9

```
11 001011 L          28 011100 c          45 101101 t          62 111110 +
12 001100 M          29 011101 d          46 101110 u          63 111111 /
13 001101 N          30 011110 e          47 101111 v
14 001110 O          31 011111 f          48 110000 w       (complément) =
15 001111 P          32 100000 g          49 110001 x
16 010000 Q          33 100001 h          50 110010 y
```

extrait de l'article base64 issu de Wikipedia.

Partie A. Codage (3 points).

Décrire par un schéma la méthode de codage base64.

Effectuer le codage des 3 premiers octets du premier exemple à suivre et des 6 octets de l'exemple suivant :

— Un fichier texte (ascii) commençant par "Il etait une fois ..."

Réponse :

Les 6 premiers octets : "Il eta" : 0x49 0x6C 0x20 0x65 0x74 0x61

Les 6 premiers octets en binaire : 0100 1001 0110 1100 0010 0000 0110 0101 0111 0100 0110 0001

Les 6 premiers octets en binaire redécoupage en paquets de 6 : 010010 010110 110000 100000 011001 010111 010001 100001

Les 6 premiers octets en base64 : SWwgZXRh

— Un fichier binaire commençant par les mots 32 bits 0xABCDEF01 0x23456789 ...

Réponse :

Les 6 premiers octets : AB CD EF 01 23 45

Les 6 premiers octets en binaire : 1010 1011 1100 1101 1110 1111 0000 0001 0010 0011 0100 0101

Les 6 premiers octets en binaire redécoupage en paquets de 6 : 101010 111100 110111 101111 000000 010010 001101 000101

Les 6 premiers octets en base64 : q83vASNF

Partie B. Analyse (1 point).

Répondez en quelques lignes aux questions suivantes :

— Quelles sont les avantages et inconvénients d'un codage base64 ?

Réponse :

- Avantage : Codage en ASCII donc portable (pas de problème compatibilité car l'ASCII est standard), beaucoup d'outils de mails (par exemple) utilise uniquement l'ASCII
- Inconvénient : Le code est 1/3 plus gros que l'original

— Si la séquence de données à coder n'a pas un nombre d'octets multiple de 3, est-ce un problème ?

Réponse : Le problème est que l'on code par paquets de 3 octets, dont il faut un nombre de donnée multiple de 3.

Si ce n'est pas le cas : il nous reste à la fin soit 1 ou 2 octets à coder.

— Si il nous en reste 1 : on code « normalement » les 6 premiers bits, on complète les 2 bits restants avec 4 zéros (à droite), on code ce nouveau paquet de 6 et enfin on termine avec deux caractères spéciaux « = » (ce qui distinguera ce cas particulier lors du décodage)

— Si il nous en reste 2 : on code « normalement » les 12 premiers bits (c-à-d en deux paquets de 6), on complète les 4 bits restants avec deux zéros (à droite) et enfin on termine avec un caractère spécial « = » (ce qui distinguera ce cas particulier lors du décodage)

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`	Branch	branchement conditionnel	xx = condition Cf. table ci-dessous
	`BL`	Branch and Link	appel sous-programme	adresse de retour dans r14=LR
	`LDR`	LoaD Register	lecture mémoire	
	`STR`	STore Register	écriture mémoire	

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

Nous rappelons 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'affichages du fichier `es.s` :

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

Nous rappelons les principales fonctions de saisie clavier du fichier `es.s` :

- `bl Lire32` (resp. `Lire16`, `Lire8`) récupère au clavier un entier 32 bits (resp. 16 bits, 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 et hexadécimal

0x20	32	␣	33	!	34	"	35	#	0x24	36	$	37	%	38	&	39	'
0x28	40	(	41	)	42	*	43	+	0x2C	44	,	45	-	46	.	47	/
0x30	48	0	49	1	50	2	51	3	0x34	52	4	53	5	54	6	55	7
0x38	56	8	57	9	58	:	59	;	0x3C	60	<	61	=	62	>	63	?
0x40	64	@	65	A	66	B	67	C	0x44	68	D	69	E	70	F	71	G
0x48	72	H	73	I	74	J	75	K	0x4C	76	L	77	M	78	N	79	O
0x50	80	P	81	Q	82	R	83	S	0x54	84	T	85	U	86	V	87	W
0x58	88	X	89	Y	90	Z	91	[	0x5C	92	\	93	]	94	^	95	_
0x60	96	`	97	a	98	b	99	c	0x64	100	d	101	e	102	f	103	g
0x68	104	h	105	i	106	j	107	k	0x6C	108	l	109	m	110	n	111	o
0x70	112	p	113	q	114	r	115	s	0x74	116	t	117	u	118	v	119	w
0x78	120	x	121	y	122	z	123	{	0x7C	124	|	125	}	126	~	127	del