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

Contrôle Continu UE INF451 : Architectures des ordinateurs

Mars 2021, 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.
Tant que possible, indiquez bien tous les détails et justifiez vos réponses.
Le barème est donné à titre indicatif.

1 Numération et opération en binaire et en complément à 2 (6 points)

(a) Déterminer le nombre de bits minimum pour représenter ensemble les valeurs suivantes puis donner la représentation binaire de ces entiers relatifs avec le nombre de bits choisi globalement (codage en complément à 2) : $(+121)_{10}$, $(-323)_{10}$. **(2 points)**

Réponse : Il faut l'intervalle [-512,+511] donc 10 bits

$(+121)_{10} = (00\ 0111\ 1001)_2 = 79_{16}$

$(+323)_{10} = (01\ 0100\ 0011)_2$
Complément à 1 : $(10\ 1011\ 1100)_2$
Complément à 2 : $(-323)_{10} = (10\ 1011\ 1101)_2 = 2BD_{16}$

(b) Donner les valeurs binaires et décimales des 2 entiers relatifs suivants codés sur 16 bits en complément à 2 : $(2021)_{16}$ et $(FA57)_{16}$. **(2 points)**

Réponse : $(2021)_{16} = (0010\ 0000\ 0010\ 0001)_2$ donne en décimal 8225
$(FA57)_{16} = (1111\ 1010\ 0101\ 0111)_2$ donne en décimal -1 449
$C2(FA57)_{16} = 0000\ 0101\ 1010\ 1001_2 : -(5 * 256 + 10 * 16 + 9) = -(1024 + 256 + 169)$

(c) Poser chacune des opérations suivantes sur 1 octet, effectuer l'opération en binaire (sur 1 octet, avec les retenues) et donner une interprétation de l'opération pour la représentation binaire usuelle et pour le codage en complément à 2 (sur 1 octet) ainsi que la valeur des indicateurs (Z, N, C et V) : $(0111\ 1011)_2 + (1110\ 1010)_2$ et $(0110\ 1111)_2 - (1101\ 1110)_2$ (soustraction par addition du complémentaire). Pour la soustraction, détailler la méthode employée. **(2 points)**
Réponse :

```
im2ag-turing:[~]: add 8 123 234
          Bit numbers
          7654 3210                              if natural     if signed

          0111 1011 left   :  0x      7b -->        123 or        +123
        + 1110 1010 right  :  0x      ea -->        234 or         -22
   C=1 == 1111 0100 < c0=0   (in carries)
   V=0  ^ ---- ----
 Z=0 N=0->0110 0101  =     :  0x      65 -->        101 or        +101
```

```
im2ag-turing:[~]: subc2 8 111 222
          Bit numbers
          7654 3210                              if natural     if signed

          0110 1111 left   :  0x      6f -->        111 or        +111
        + 0010 0001 right  :  0x      21 -->         33 or         +33
   C=0 != 1101 1111 < c0=1   (in carries)
   V=1  ^ ---- ----
 Z=0 N=1->1001 0001  =     :  0x      91 -->        145 or        -111
```

On a choisi ici d'utiliser le complément à 1 comme opérande plus 1 en retenue initiale. Il est possible de prendre le complément à 2 0010 0010 comme opérande et une retenue initiale nulle.

2 Programmation en ARM (8 points)

Le but de cet exercice est de calculer la $i^{\text{ème}}$ ligne du triangle de Pascal. Cette ligne sera stockée dans un tableau de **mots** représentant des entiers naturels. Ce tableau sera appelé `TAB`. L'algorithme vous permettant de résoudre ce problème est donné ci-dessous.

```
 1. EcrChaine("Entrez un nombre'')
 2. X:=Lire32()
 3. si X <= 100 alors
 4.    pour i de 0 à X-1 faire
 5.         TAB[i] := 1
 6.         j:=i
 7.         tant que j >= 2 faire
 8.              j :=  j - 1
 9.              TAB[j] := TAB[j] + TAB[j-1]
10.         fin tant que
11.    fin pour
12. fin si
```

Dans l'algorithme, les variables `X, i` et `j` sont des entiers naturels sur 1 mot, les fonctions `EcrChaine` et `Lire32` correspondent aux fonctions du fichier `es.s` étudié en cours et en TP. Le fonctionnement des fonctions d'`es.s` est rappelé en annexe du sujet.

Vous allez maintenant traduire l'algorithme en assembleur `ARM` en vous basant sur le squelette de code donné ci-dessous.

```
  .data
M: .asciz "Entrez un nombre"
  .bss
X: .skip 4
TAB: .skip 400
  .text
```

```
  .global main
main:

            @ partie manquante

  b exit

ptr_X: .word X
ptr_M: .word M
ptr_TAB: .word TAB
```

Questions :

(a) Rappelez la taille d'un mot en octets. **(0,5 point)**

Réponse : 4 octets.

(b) Rappelez la différence entre les segments `.data` et `.bss` **(0,5 point)**

Réponse :
`.data` : données initialisés. `.bss` : données non-initialisés.

(c) Donnez le code `ARM` des instructions pour la ligne 1 (pour `EcrChaine`, vous appliquerez les conventions de `es.s` utilisées en TP notamment, *cf.* annexe). **(0,5 point)**

```
ldr r1,ptr_M
bl EcrChaine
```

(d) En supposant que l'adresse de la variable `X` est stockée dans le registre `r1`, donnez le code `ARM` pour récupérer la valeur de la variable `X` dans le registre `r2` puis exécuter le test en ligne 3 (ne traduisez pas le bloc d'instruction des lignes 4-11). Si le test est faux, le branchement se fera vers l'étiquette `finCode`. **(1 point)**

```
ldr r2,[r1]
cmp r2,#100  @ si r2 > 100
bhi finCode
```

(e) En supposant que l'adresse du tableau `TAB` est stockée dans le registre `r3`, que le registre `r0` représente la variable `i`, que le registre `r6` représente la variable `j`, donnez le code `ARM` des instructions des lignes 5 et 9. Utilisez les registres `r4`, `r5` et `r7` pour les calculs intermédiaires éventuels. **(2 points)**

```
mov r5,r0,lsl #2  @ offset    ou calcul d'adresse :     add  r5,r3,r0,lsl #2
mov r4,#1                                               mov  r4,#1
str r4,[r3,r5]    @ tab[i] := 1                         str  r4,[r5]
```

(f) En supposant que l'adresse du tableau `TAB` est stockée dans le registre `r3`, que le registre `r0` représente la variable `i`, que le registre `r6` représente la variable `j`, donnez le code `ARM` des lignes 7-10 (hors ligne 9). Ne recopiez pas le code `ARM` de la ligne 9, inscrivez simplement à la place `@ ici ligne 9`. Introduire et utiliser une étiquette `apresTantQue` pour sortir du tant que. Utilisez les registres `r4`, `r5` et `r7` pour les calculs intermédiaires éventuels. **(2 points)**

```
      mov r6,r0
                                      @ avec condition suivant le corps
tq:                                   tq:   b  cond
cond:   cmp r6,#2
        bcc ftq   @ ou blo
corps:  sub r6,r6,#1              corps:   @ code du corps identique

        mov r4,r6,lsl #2
        sub r5,r4,#4

        ldr r7,[r3,r4]
        ldr r5,[r3,r5]

        add r5,r5,r7
        str r5,[r3,r4]
        b tq                       cond:    cmp r6,#2
ftq:                                        bcs corps  @ ou bhs
                                  fintq:
```

(g) En supposant que le registre `r0` représente la variable `i` et que la valeur de la variable `X` est stockée dans le registre `r2`, donnez le code `ARM` du « pour » en lignes 4-11. Ne recopiez pas le code `ARM` des lignes 5-10, inscrivez simplement à la place `@ ici lignes (5-10)`. Introduire et utiliser une étiquette `apresPour` pour sortir du pour. **(1.5 point)**

```
mov r0,#0
                                @ avec condition suivant corps
pour: cmp r0,r2                           b condpour
      bcs fin  @ ou bhs         corpspour:

      @ ici lignes 5-10                   @ ici lignes 5-10

      add r0,r0,#1                        add r0,r0,#1
      b pour                    condpour: cmp r0,r2
fin:                                      bcc  corpspour  @ ou blo
```

3 Codage NégaBinaire (6 points)

"*Le système négabinaire (base -2) est un système de numération positionnel non standard utilisé dans l'ordinateur expérimental polonais BINEG, construit en 1957-59. Il possède la propriété inha-*

bituelle d'avoir les nombres négatifs et positifs représentés sans un bit de signe" source Wikipédia.

Le négabinaire (base -2), c'est comme le binaire (base 2), les chiffres possibles sont aussi les bits 0 et 1. La différence ? en binaire, un nombre $(N_3N_2N_1N_0)_2$ a pour valeur $\sum_i N_i \times 2^i$; en négabinaire, un nombre $(N_3N_2N_1N_0)_{-2}$ a pour valeur $\sum_i N_i \times (-2)^i$.

Questions :

(a) Donnez les valeurs décimales des 16 premiers nombres binaires et négabinaires sur 4 bits. Faites un tableau à trois colonnes (1 : combinaisons binaires sur 4 bits [dans l'ordre], 2 : valeurs associées en binaire habituel, 3 : valeurs associées en négabinaire). Encadrez les lignes où la valeur binaire est égale à la valeur négabinaire. **(2 points)**
Réponse :

code	bin	negabin	code	bin	negabin	code	bin	negabin	code	bin	negabin
00**0**0	**0**	**0**	**0**0**0**1	**1**	**1**	0010	2	-2	0011	3	-1
01**0**0	**4**	**4**	**0**1**0**1	**5**	**5**	0110	6	2	0111	7	3
1000	8	-8	1001	9	-7	1010	10	-10	1011	11	-9
1100	12	-4	1101	13	-3	1110	14	-6	1111	15	-5

Les valeurs sont identiques si les chiffres de rangs pairs sont nuls.

(b) Donnez le plus grand nombre sur 8 bits, en binaire, en héxadécimal et en décimal dont la valeur en binaire est égale à la valeur en négabinaire **(1 points)**.

(c) Dénombrez le nombre de demi-mots de 16 bits où la valeur binaire est égale à la valeur négabinaire. Expliquez votre calcul. **(1 points)**.

Réponses :

Le plus grand nombre sur 8 bits de valeurs identiques est celui dont tous les bits de rangs impairs sont nuls et tous ceux de rang pairs non nuls :

$\mathbf{0}1\mathbf{0}1\mathbf{0}1\mathbf{0}1_2 = 55_{16} = 5 * 16 + 5 = 85$

Prendre toutes les combinaisons possibles des 8 bits de rang pairs, en mettant 0 pour tous les bits de rangs impairs, soit $2^8 = 256$ *nombres de valeur identique dans les deux conventions.*

Le négabinaire (base -2), c'est comme le binaire (base 2), pour trouver le codage d'un nombre il "*suffit*" d'utiliser le même algorithme de divisions successives : en binaire, des divisions successives par 2 ; en négabinaire, des divisions successives par -2. Attention, cependant, ne pas se tromper dans les divisions par -2 des nombres impairs (rappel, le reste ne peut prendre que les 2 valeurs possibles 0 et 1) : 7 divisé par -2 vaut -3 reste 1 ; -9 divisé par -2 vaut 5 reste 1.

Questions :

(d) Appliquer l'algorithme des divisions successives par -2 pour 22 et donner la décomposition négabinaire de 22 sur 8 bits. **(1 points)**

(e) Appliquer l'algorithme des divisions successives par -2 pour -34 et donner la décomposition négabinaire de -34 sur 8 bits. **(1 points)**

Réponse :

```
 22 = -2 * -11 reste 0
-11 = -2 * 6   reste 1
  6 = -2 * -3  reste 0
 -3 = -2 *  2  reste 1
  2 = -2 * -1  reste 0
 -1 = -2 *  1  reste 1
  1 = -2 *  0  reste 1

0 110 1010     64 -32 -8 -2 = 32-10 = 22

-34 = -2 * 17  reste 0
 17 = -2 * -8  reste 1
 -8 = -2 *  4  reste 0
  4 = -2 * -2  reste 0
 -2 = -2 *  1  reste 0
  1 = -2 *  0  reste 1

00 10 0010 = -32 -2 = -34
```

ANNEXES

A 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`.

B Principales 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	
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
	`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`}.

C 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	