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

Contrôle Continu UE INF401 : Architectures des ordinateurs

Mars 2019, 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) 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) : $(+5019)_{10}$, $(-5019)_{10}$. **(1,5 point)**

(b) Donner les valeurs décimales des 2 entiers relatifs suivants codés sur 16 bits en complément à 2 : $(0001\ 0011\ 0101\ 0100)_2$ et $(FEED)_{16}$. **(2 points)**

(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) : $(+23)_{10} + (-13)_{10}$, $(+43)_{10} - (+23)_{10}$ (vraie soustraction) et $(-13)_{10} - (+43)_{10}$ (soustraction par addition du complémentaire) . **(1,5 point)**

2 Programmation en ARM (10 points)

Le but de cet exercice est de programmer en ARM le *crible d'Erathostène* qui permet de calculer les nombres premiers inférieurs à une borne N (ici on a choisi $N = 111$). Pour cela on considère les zones `.bss` et `.text` suivantes :

```
      .bss
T:    .skip 444

      .text .global main
main:

      @ partie à compléter

      b exit
ptrT: .word T
```

La première partie de l'algorithme consiste à initialiser le tableau de 111 mots appelé `T` avec la boucle suivante :

```
pour i de 0 à 110 faire
     T[i] := 1
fin pour
```

(a) Traduisez en `ARM` la boucle d'initialisation précédente. En particulier, vous stockerez l'adresse du tableau `T` dans le registre $r0$ et le registre $r1$ représentera l'indice i (i est un entier naturel). Vous utiliserez les registres $r2$ et $r3$ pour les calculs intermédiaires. **(2 points)**

Ensuite, l'algorithme consiste à affecter `T`$[i]$ à 0 si le nombre i n'est pas premier. On sait que 0 et 1 ne sont pas premiers, donc :

```
T[0] := 0
T[1] := 0
```

(b) Traduisez en `ARM` les deux affectations précédentes en supposant que l'adresse de `T` est toujours dans le registre $r0$. Vous utiliserez le registre $r2$ pour stocker la constante 0. **(0,5 point)**

Il faut ensuite calculer la partie entière inférieure de la racine carrée de N ($\lfloor\sqrt{N}\rfloor$). Ci-dessous, on vous donne le code `ARM` qui calcule dans le registre $r2$ la valeur $\lfloor\sqrt{111}\rfloor$.

```
       mov r2,#1
tq:    mul r3,r2,r2
       cmp r3,#111
       bhi finTq
       add r2,r2,#1
       b tq
finTq: sub r2,r2,#1
```

(c) Traduisez en langage algorithmique (ou en langage `C`) le code `ARM` précédent. **(1 point)**

Le crible procède ensuite par élimination : il marque à 0 les cases `T`$[i]$ du tableau de tous les indices i entre 2 et $N-1$ multiples d'un entier entre 2 et $\lfloor\sqrt{N}\rfloor$.

```
pour i de 2 à partie entière inférieure de la racine carrée de 111 faire
     j := 2 * i
     tant que j < 111 faire
          T[j] := 0
          j := j + i
     fin tant que
fin pour
```

(d) Traduisez en `ARM` l'algorithme précédent en supposant que l'adresse de `T` est toujours dans le registre $r0$ et que $\lfloor\sqrt{111}\rfloor$ est toujours stocké dans le registre $r2$. Le registre $r1$ représentera l'indice i, le registre $r3$ représentera l'indice j. Vous utiliserez les registres $r4$ et $r5$ pour les calculs intermédiaires. Pour plus de clarté dans le code, vous dessinerez un cadre autour de la traduction du code du « tant que ». **(4 points)**

Enfin, on affiche les nombres premiers inférieurs à 111 :

```
pour i de 2 à 110 faire
     si T[i] = 1 alors
        EcrNdecimal32(i)
     fin si
fin pour
```

(e) Traduisez en `ARM` l'algorithme précédent en supposant que l'adresse de `T` est toujours dans le registre $r0$. Le registre $r1$ représentera l'indice i. Vous utiliserez le registre $r3$ pour les calculs intermédiaires. Nous rappelons aussi que la fonction `EcrNdecimal32` suit la convention adoptée dans le fichier `es.s` utilisé en TP (un rappel de ces conventions est donné en annexe du sujet). **(2,5 points)**

3 Représentation des nombres avec détection d'erreur par une nouvelle règle de parité. (5 points)

Lors d'un transfert de données, chacun des bits transite sur une liaison série :

— après conversion en valeur de signal électrique (exemple RS232 : $\pm$ 12 Volts)

— avec une faible probabilité d'être inversé durant le transfert.

Pour détecter les erreurs de transmissions, il est d'usage d'ajouter de la redondance dans l'information transmise, les codages habituels (binaire pur, complément à deux) ne seront donc pas utilisés dans cet exercice.

Question préliminaire [1 point] : donner le **nombre** de codes binaires possibles avec $n = 4$ bits et $n = 8$ bits.

Pour cet exercice, nous utiliserons une représentation des nombres originale qui permet la détection d'erreur. Cette représentation des nombres devra permettre de représenter tous les entiers relatifs.

Définition : une séquence de n (pair) bits est α_n si elle contient autant de bits à 0 que de bits à 1.

Exemples :

— Une alternance de 0 et de 1 de longueur n paire est α_n.

— Une séquence composée uniquement de 0 ou uniquement de 1 n'est pas α_n, de même que celles qui ont plus de 0 que de 1 (ou l'inverse).

Chaque entier relatif sera codé sur n (pair) bits par un code α_n. En réception, toute combinaison de n bits autre que α_n sera interprétée comme une erreur de transmission. L'objectif est de détecter toute erreur de transmission affectant un seul bit et/ou d'obtenir une valeur moyenne du signal électrique (par exemple nulle) indépendante de l'entier émis.

Partie A. Sur 4 bits [2 points].

Énumérer dans la première colonne d'un tableau à deux colonnes les k codes α_4.

Déterminer quel intervalle de k entiers relatifs peut être encodé (donner un exemple).

Définir dans la seconde colonne du tableau une association bijective entre les codes α_4 et les entiers de votre intervalle.

Expliquer brièvement votre choix d'intervalle d'entiers et de bijection.

Partie B. Sur 8 bits [2 points].

Définition des propriétés γ_n et δ_n : une séquence de n (n pair) bits est

— γ_n si ses $n/2$ bits de gauche forment une séquence $\alpha_{n/2}$

— δ_n si ses $n/2$ bits de droite forment une séquence $\alpha_{n/2}$

On utilise à présent un codage sur $n = 8$ bits et une restriction des entiers aux seuls codes à la fois α_8 et γ_8 (ce qui de fait implique d'être aussi aussi δ_8).

Question 1 : Donner le **nombre** de codes binaires à la fois α_8 et γ_8 et proposer un intervalle d'entiers relatifs représentable de cette manière.

Question 2 : Dans le cas général, toujours sur 8 bits, sans le respect des propriétés γ_8 ou δ_8 pouvez-vous dénombrer ou donner une approximation du nombre total de configurations α_8.

(Bonus) Plutôt que de rejeter les configurations non autorisées, le système pourrait-il corriger les erreurs détectées ? **Justifiez votre réponse. (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