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

Contrôle continu UE INF241 : Introduction aux Architectures Logicielles et Matérielles

9 Mars 2015, durée 1 h 30
Documents, calculettes, téléphones portables interdits
Le barême est donné à titre indicatif

1 Numération (5 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 suivantes : $(+120)_{10} + (-36)_{10}$ et $(+120)_{10} + (+45)_{10}$,
   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 suivantes : $(+120)_{10} - (+45)_{10}$ et $(-36)_{10} - (+120)_{10}$,
   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.

(c) **(1 point)** On souhaite stocker en mémoire l'entier naturel 265. Combien de bits sont, au minimum, nécessaires pour stocker ce nombre ? Combien d'octets sont, au minimum, nécessaires pour stocker ce nombre ? Justifiez vos réponses.

2 Codage d'une instruction en ARM (3 points)

Nous nous intéressons au codage binaire d'une instruction de calcul `ARM` de deux registres opérandes `rn` et `rm` vers un registre cible `rd`. Pour une telle instruction, il est possible d'appliquer un décalage sur la valeur du deuxième registre opérande `rm` avant d'effectuer le calcul. Par exemple, `ADD r1, r2, r3, lsl #2` ajoute la valeur de `r2` à la valeur de `r3` décalée au préalable de 2 positions sur la gauche, et stocke le résultat dans `r1`. Ainsi, cela revient à affecter `r1` avec `r2` + `r3` × 4.

Une telle instruction est encodée selon le schéma suivant (figure 1) :

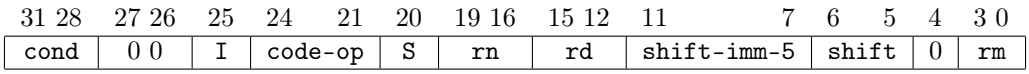

FIGURE 1 – Codage d'une instruction arithmétique ou logique

— `cond` correspond au code de la condition (expression booléenne des indicateurs Z, N, C et V) à vérifier pour exécuter l'instruction,
— `I` doit être positionné à 0 pour signifier que le second opérande n'est pas une valeur immédiate,
— `code-op` encode la nature de l'opération (mnénomique de l'instruction : par exemple, le code de `ADD`),

— `S` vaut 1 si on souhaite mettre à jour les indicateurs avec le résultat de l'opération,
— `rn` est le numéro du registre premier opérande,
— `rm` est le numéro du registre deuxième opérande,
— `rd` est le numéro du registre où est stocké le résultat,
— `shift` indique la nature du décalage : `LSL`, `LSR`, `ASR` et `ROR` sont respectivement encodés en 00, 01, 10 et 11, enfin
— `shift-imm-5` code le nombre de décalages à effectuer.

Les codes des conditions et des opérations sont donnés dans les annexes A et B.

Questions :

(d) Donner le code de l'instruction `ADDLT r1,r8,r4,LSL #28` en binaire. **(2 points)**
(e) Donner le code de l'instruction `ADDLT r1,r8,r4,LSL #28` en hexadécimal. **(1 point)**

3 Programmation en ARM (12 points)

3.1 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, contenant successivement les codes ASCII 'B', 'o', 'n', 'j', 'o', 'u', 'r' et enfin l'octet 0.

3.2 Conversion entier naturel vers hexadécimal

Le but de cet exercice est de stocker dans une chaîne de caractères la représentation hexadécimale d'un **entier naturel codé sur 4 octets** saisi au clavier, puis d'afficher cette chaîne. Par exemple, si l'utilisateur saisit le nombre 1690, le programme affichera la chaîne de caractères "69A".

Mémoire : Nous utiliserons dans la suite la zone `.data` suivante :

```
.data
Q: .asciz "entrer un entier naturel"
R: .asciz "conversion hexadecimal :"
```

Questions :

Nous allons avoir besoin d'une variable `E` sur 4 octets où nous allons stocker l'entier saisi au clavier. De plus, le résultat de la conversion sera stocké dans un tableau `H` de 9 caractères.

(f) Déclarer `E` et `H` dans une zone de données non initialisées. **(1 point)**
(g) Combien faut-il de chiffres hexadécimaux (au minimum) pour représenter un nombre stocké dans un octet ? **(0,25 point)**
(h) Justifier pourquoi un tableau d'exactement 9 caractères est nécessaire et suffisant pour stocker la représentation hexadécimale de `E`. **(0,75 point)**

(i) Donner le code `ARM` réalisant la saisie clavier suivante **(1 point)** :

```
afficher "entrer un entier naturel"
saisir E
r1 <- E
```

(j) Le résultat de quelle opération arithmétique effectuée sur la valeur de `r1` est stocké dans `r5` suite à l'exécution des instructions `ARM` suivantes **(1 point)** :

```
mov r3,r1
mov r1,r1,lsr #4
mov r4,r1,lsl #4
sub r5,r3,r4
```

(k) Traduiser en `ARM` l'algorithme suivant (si vous utilisez du code donné dans vos réponses précédentes, contentez-vous de l'indiquer clairement, ne perdez pas de temps à recopier) **(4 points)** :

```
x <- E
n <- 0
tantque x != 0 faire
        y <- x modulo 16
        x <- x / 16
        si y < 10 alors
           H[n] <- y+'0'
        sinon
           H[n] <- y-10+'A'
        fin si
        n <- n + 1
fin tantque
H[n] <- 0
```

Pour les variables x, n et y vous utiliserez les registres $r1$, $r2$ et $r3$, respectivement ; par contre `E` et `H` sont les variables déclarées lors de la question (f).

L'algorithme précédent stocke la représentation hexadécimale de `E` dans `H`, mais à l'envers ! L'algorithme suivant renverse la chaîne stockée dans `H`, puis l'affiche.

```
i <- 0
tantque i < n/2 faire
        z <- H[i]
        H[i] <- H[n-i-1]
        H[n-i-1] <- z
        i <- i + 1
fin tantque
afficher "conversion hexadecimal :"
afficher H
```

(l) Traduiser l'algorithme précédent en `ARM`. Pour les variables i et z vous utiliserez les registres $r3$ et $r1$, respectivement. Nous rappelons que pour n nous utilisons déjà $r2$ et que `H` a été déclarée lors de la question (f). **(4 points)**

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