IFT209 – Programmation système
Université de Sherbrooke

Examen périodique

Enseignant : Michael Blondin
Date : mardi 26 février 2019

Prénom : _____ Nom : _____

CIP : _____ Signature : _____

Directives :

- Vous devez répondre aux questions dans le **cahier de réponses**, *pas* sur ce questionnaire ;
- **Aucun matériel additionnel** (notes de cours, fiches récapitulatives, etc.) n'est permis ;
- **Aucun appareil électronique** (calculatrice, téléphone, tablette, ordinateur, montre intelligente, etc.) n'est permis ;
- Vous devez donner **une seule réponse** par sous-question ;
- Le langage d'assemblage utilisé dans l'examen est celui de l'**architecture ARMv8** tel qu'utilisé en classe ; un sommaire de cette architecture est présenté en **annexe** de ce questionnaire ;
- L'examen comporte **6 questions** sur **4 pages** valant un total de **50 points** ;
- La correction est notamment basée sur la **clarté** et l'**exactitude** de vos réponses, ainsi que sur la **démarche** pour les questions qui en requièrent une.

Question 1 : questions en rafale

(a) Quel est le plus grand entier *signé* pouvant être représenté sur n bits ? 1 pt

(b) Quel est le plus petit nombre de bits nécessaire afin de représenter l'entier *non signé* $2FEDCB_{16}$? 1 pt

(c) Dans le *pire cas*, combien de bits faut-il pour stocker le produit de deux entiers *non signés* de n bits ? 1 pt

(d) Combien d'*octets* devez-vous minimalement allouer afin de stocker une matrice, de m lignes et n colonnes, dont les éléments sont des demi-mots ? 1 pt

(e) Si le compteur d'instruction contient $FF08_{16}$, que contient le registre `x30` après l'exécution de `bl printf` ? 1 pt

(f) L'architecture ARMv8 est-elle de type RISC ou CISC ? 1 pt

(g) Nommez *deux* unités du processeur d'une architecture de von Neumann. 1 pt

Question 2 : systèmes de numération

Effectuez les conversions des entiers *non signés* suivants, en laissant une trace de votre démarche. Vos conversions ne doivent *pas* utiliser de bases intermédiaires. Vous ne pouvez donc *pas* passer par la base 10 en (b) et (c).

(a) 43 de la base 10 vers la base 2 2 pts

(b) 10110011101 de la base 2 vers la base 16 2 pts

(c) A0F817 de la base 16 vers la base 4 2 pts

Question 3 : entiers signés

Soit $a = 1101011$ et $b = 100100$, des entiers *signés* représentés sous complément à deux.

(a) Effectuez la soustraction $a - b$ en laissant une trace de votre démarche. 3 pts

(b) Donnez la valeur en base 10 du résultat de la soustraction précédente. 2 pts

(c) Supposons que le registre $\texttt{x}_{19}$ contienne a (étendu sur 64 bits), et que le registre $\texttt{x}_{20}$ contienne b (étendu sur 64 bits). Donnez la valeur des codes de condition N (négatif), Z (zéro) et V (débordement) après l'exécution de l'instruction « `cmp x19, x20` ». Justifiez brièvement votre réponse pour la valeur de V. 3 pts

(d) Dites si $a > b$. Justifiez brièvement votre réponse. 2 pts

Question 4 : mémoire et accès aux données

Rappelons que l'architecture ARMv8 utilise le format « little-endian » (petit-boutiste). Considérons le contenu suivant de sa mémoire principale :

adresse	contenu
0	01_{16}
1	$A4_{16}$
2	BC_{16}
3	48_{16}
4	$5F_{16}$
5	11_{16}
6	FF_{16}
7	43_{16}
⋮	⋮

(a) Quelle est la valeur hexadécimale du mot stocké à l'adresse 2 ? 2 pts

(b) Le mot stocké à l'adresse 2 est-il à une adresse alignée ? Justifiez votre réponse. 2 pts

(c) Quelle est la valeur des registres $\texttt{x}_{19}$ et $\texttt{x}_{20}$ après l'exécution de ces instructions : 3 pts

```
mov   x19, 2
ldr   x20, [x19], 3
sub   x19, x19, 1
ldrb  w20, [x19, 2]
```

Question 5 : programmation en langage d'assemblage

Complétez le code partiel ci-bas afin de :

(a) Lire un entier *non signé* n de 64 bits ; 3 pts

(b) Calculer $f(n)$, où la fonction f est définie par : 4 pts

$$f(n) = \begin{cases} n - 1 & \text{si } n \text{ est impair,} \\ n \div 3 & \text{sinon ;} \end{cases}$$

(c) Afficher $f(n)$. 3 pts

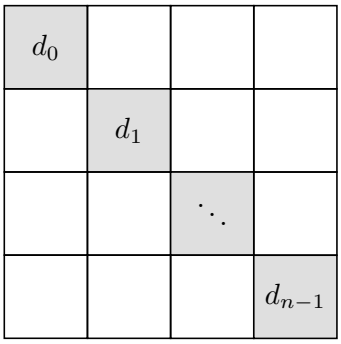
Code partiel :

```
.global main

main:
        // Lire n

        // Calculer f(n)

        // Afficher f(n)

        bl      exit

/*
    données ici
                */
```

Question 6 : tableaux et programmation structurée

Considérons une matrice B de taille $n \times n$ dont les éléments sont des entiers *signés* de 64 bits. Supposons que B soit stockée dans un tableau à l'adresse a de la mémoire principale. Supposons également que le registre `x0` contienne a et que le registre `x1` contienne n.

(a) À quelle adresse se situe le $i^{\text{ème}}$ élément de la diagonale principale de B ? Autrement dit, à quelle adresse se situe l'élément d_i de la matrice $n \times n$ suivante ? 2 pts



(b) Complétez le code du *sous-programme* suivant afin qu'il retourne le produit des éléments de la diagonale principale de B ; autrement dit, afin qu'il retourne $d_0 \cdot d_1 \cdots d_{n-1}$: 8 pts

```
diag:
        SAVE
        /*
            code ici
                        */
        RESTORE
        ret
```

Annexe :

Sommaire de l'architecture ARMv8

Registres

- Chaque registre $\texttt{x}_\texttt{n}$ possède 64 bits: $b_{63}b_{62}\cdots b_1b_0$
- Notation: $\texttt{x}_\texttt{n}\langle i\rangle \stackrel{\text{def}}{=} b_i$, $\texttt{x}_\texttt{n}\langle i,j\rangle \stackrel{\text{def}}{=} b_ib_{i-1}\cdots b_j$, $\texttt{r}_\texttt{n}$ réfère au registre $\texttt{x}_\texttt{n}$ ou $\texttt{w}_\texttt{n}$
- Chaque registre $\texttt{w}_\texttt{n}$ possède 32 bits et correspond à $\texttt{x}_\texttt{n}\langle 31,0\rangle$
- Le compteur d'instruction `pc` n'est pas accessible
- Conventions:

Registres	Nom	Utilisation
$\texttt{x}_0$ – $\texttt{x}_7$	—	registres d'arguments et de retour de sous-programmes
$\texttt{x}_8$	`xr`	registre pour retourner l'adresse d'une structure
$\texttt{x}_9$ – $\texttt{x}_{15}$	—	registres temporaires sauvegardés par l'appelant
$\texttt{x}_{16}$ – $\texttt{x}_{17}$	$\texttt{ip}_0$ – $\texttt{ip}_1$	registres temporaires intra-procéduraux
$\texttt{x}_{18}$	`pr`	registre temporaire pouvant être réservé par le système
$\texttt{x}_{19}$ – $\texttt{x}_{28}$	—	registres temporaires sauvegardés par l'appelé
$\texttt{x}_{29}$	`fp`	pointeur vers l'ancien sommet de pile (*frame pointer*)
$\texttt{x}_{30}$	`lr`	registre d'adresse de retour (*link register*)
$\texttt{x}_{zr}$	`sp`	registre contenant la valeur 0, ou pointeur de pile (*stack pointer*)

Arithmétique (entiers)

- Les codes de condition sont modifiés par `cmp`, `adds`, `subs` et `negs`
- À cette différence près, `adds`, `adcs`, `subs` et `negs` se comportent respectivement comme `add`, `adc`, `sub` et `neg`
- Instructions, où i est une valeur immédiate de 12 bits et j est une valeur immédiate de 6 bits:

Code d'op.	Syntaxe	Effet	Exemple
`cmp`	`cmp rd, rm`	compare $\texttt{r}_\texttt{d}$ et $\texttt{r}_\texttt{m}$	`cmp x19, x21`
	`cmp rd, i`	compare $\texttt{r}_\texttt{d}$ et i	`cmp x19, 42`
	`cmp rd, rm, decal j`	compare $\texttt{r}_\texttt{d}$ et $\texttt{r}_\texttt{m}$ *decal* j	`cmp x19, x21, lsl 1`
`add`	`add rd, rn, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} + \texttt{r}_\texttt{m}$	`add x19, x20, x21`
	`add rd, rn, i`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} + i$	`add x19, x20, 42`
	`add rd, rn, rm, decal j`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} + (\texttt{r}_\texttt{m}$ *decal* $j)$	`add x19, x20, x21, lsl 1`
`adc`	`adc rd, rn, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} + \texttt{r}_\texttt{m} + \texttt{C}$	`adc x19, x20, x21`
`sub`	`sub rd, rn, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} - \texttt{r}_\texttt{m}$	`sub x19, x20, x21`
	`sub rd, rn, i`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} - i$	`sub x19, x20, 42`
	`sub rd, rn, rm, decal j`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} - (\texttt{r}_\texttt{m}$ *decal* $j)$	`sub x19, x20, x21, lsl 1`
`neg`	`neg rd, rm`	$\texttt{r}_\texttt{d} \leftarrow -\texttt{r}_\texttt{m}$	`neg x19, x21`
	`neg rd, rm, decal j`	$\texttt{r}_\texttt{d} \leftarrow -(\texttt{r}_\texttt{m}$ *decal* $j)$	`neg x19, x21, lsl 1`
`mul`	`mul rd, rn, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} \cdot \texttt{r}_\texttt{m}$	`mul x19, x20, x21`
`udiv`	`udiv rd, rn, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} \div \texttt{r}_\texttt{m}$ (non signé)	`udiv x19, x20, x21`
`sdiv`	`sdiv rd, rn, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} \div \texttt{r}_\texttt{m}$ (signé)	`sdiv x19, x20, x21`
`madd`	`madd rd, rn, rm, ra`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{a} + (\texttt{r}_\texttt{n} \cdot \texttt{r}_\texttt{m})$	`madd x19, x20, x21, x22`
`msub`	`msub rd, rn, rm, ra`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{a} - (\texttt{r}_\texttt{n} \cdot \texttt{r}_\texttt{m})$	`msub x19, x20, x21, x22`

Accès mémoire

- Les instructions `ldrsw`, `ldrsh` et `ldrsb` se comportent respectivement comme `ldr` (4 octets), `ldrh` et `ldrb` à l'exception du fait que les bits excédentaires reçoivent le bit de signe de la donnée chargée, plutôt que des zéros
- Instructions, où a est une adresse et $\text{mem}_b[a]$ réfère aux b octets à l'adresse a de la mémoire principale:

Code d'op.	Syntaxe	Effet	Exemple
`mov`	`mov rd, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{m}$	`mov x19, x21`
	`mov rd, i`	$\texttt{r}_\texttt{d} \leftarrow i$	`mov x19, 42`
`ldr`	`ldr xd, a`	charge 8 octets: $\texttt{x}_\texttt{d}\langle 63,0\rangle \leftarrow \text{mem}_8[a]$	`ldr x19, [x20]`
	`ldr wd, a`	charge 4 octets: $\texttt{x}_\texttt{d}\langle 31,0\rangle \leftarrow \text{mem}_4[a]$; $\texttt{x}_\texttt{d}\langle 63,32\rangle \leftarrow 0$	`ldr w19, [x20]`
`ldrh`	`ldrh wd, a`	charge 2 octets: $\texttt{x}_\texttt{d}\langle 15,0\rangle \leftarrow \text{mem}_2[a]$; $\texttt{x}_\texttt{d}\langle 63,16\rangle \leftarrow 0$	`ldrh w19, [x20]`
`ldrb`	`ldrb wd, a`	charge 1 octet: $\texttt{x}_\texttt{d}\langle 7,0\rangle \leftarrow \text{mem}_1[a]$; $\texttt{x}_\texttt{d}\langle 63,8\rangle \leftarrow 0$	`ldrb w19, [x20]`
`str`	`str xd, a`	stocke 8 octets: $\text{mem}_8[a] \leftarrow \texttt{x}_\texttt{d}\langle 63,0\rangle$	`str x19, [x20]`
	`str wd, a`	stocke 4 octets: $\text{mem}_4[a] \leftarrow \texttt{x}_\texttt{d}\langle 31,0\rangle$	`str w19, [x20]`
`strh`	`strh wd, a`	stocke 2 octets: $\text{mem}_2[a] \leftarrow \texttt{x}_\texttt{d}\langle 15,0\rangle$	`str w19, [x20]`
`strb`	`strb wd, a`	stocke 1 octet: $\text{mem}_1[a] \leftarrow \texttt{x}_\texttt{d}\langle 7,0\rangle$	`strb w19, [x20]`

Conditions de branchement

- Codes de condition: `N` (négatif), `Z` (zéro), `C` (report), `V` (débordement)
- Conditions de branchement:

Entiers non signés

Code	Signification	Codes de condition
`eq`	$=$	$\mathtt{Z}$
`ne`	$\neq$	$\neg\mathtt{Z}$
`hs`	$\geq$	$\mathtt{C}$
`hi`	$>$	$\mathtt{C} \wedge \neg\mathtt{Z}$
`ls`	$\leq$	$\neg\mathtt{C} \vee \mathtt{Z}$
`lo`	$<$	$\neg\mathtt{C}$

Entiers signés

Code	Signification	Codes de condition
`eq`	$=$	$\mathtt{Z}$
`ne`	$\neq$	$\neg\mathtt{Z}$
`ge`	$\geq$	$\mathtt{N} = \mathtt{V}$
`gt`	$>$	$\neg\mathtt{Z} \wedge (\mathtt{N} = \mathtt{V})$
`le`	$\leq$	$\mathtt{Z} \vee (\mathtt{N} \neq \mathtt{V})$
`lt`	$<$	$\mathtt{N} \neq \mathtt{V}$
`vs`	débordement	$\mathtt{V}$
`vc`	pas de débordement	$\neg\mathtt{V}$
`mi`	négatif	$\mathtt{N}$
`pl`	non négatif	$\neg\mathtt{N}$

Branchement

- Instructions de branchement, où j est une valeur immédiate de 6 bits:

Code d'op.	Syntaxe	Effet	Exemple
`b.`	`b.cond etiq`	branche à `etiq:` si *cond*	`b.eq main100`
`b`	`b etiq`	branche à `etiq:`	`b main100`
`cbz`	`cbz rd, etiq`	branche à `etiq:` si $\mathtt{r_d} = 0$	`cbz x19 main100`
`cbnz`	`cbnz rd, etiq`	branche à `etiq:` si $\mathtt{r_d} \neq 0$	`cbnz x19 main100`
`tbz`	`tbz rd, j, etiq`	branche à `etiq:` si $\mathtt{r_d}\langle j \rangle = 0$	`tbz x19 main100`
`tbnz`	`tbnz rd, j, etiq`	branche à `etiq:` si $\mathtt{r_d}\langle j \rangle \neq 0$	`tbnz x19 main100`
`bl`	`bl etiq`	branche à `etiq:` et $\mathtt{x_{30}} \leftarrow \mathtt{pc} + 4$	`bl printf`
`blr`	`blr xd`	branche à $\mathtt{x_d}$ et $\mathtt{x_{30}} \leftarrow \mathtt{pc} + 4$	`blr x20`
`br`	`br xd`	branche à $\mathtt{x_d}$	`br x20`
`ret`	`ret`	branche à $\mathtt{x_{30}}$ (retour de sous-prog.)	`ret`

Adressage

- Modes d'adressages, où k est une valeur immédiate de 7 bits:

Nom	Syntaxe	Adresse	Effet	Exemple
adresse d'une étiquette	`adr xd, etiq`	—	$\mathtt{x_d} \leftarrow$ adresse de `etiq:`	`adr x19, main100`
indirect par registre	`[xd]`	$\mathtt{x_d}$	—	`[x20]`
indirect par registre indexé	`[xd, xn]`	$\mathtt{x_d} + \mathtt{x_n}$	—	`[x20, x21]`
	`[xd, k]`	$\mathtt{x_d} + k$	—	`[x20, 1]`
	`[xd, xn, decal k]`	$\mathtt{x_d} + (\mathtt{x_n}\ \mathit{decal}\ k)$	—	`[x20, x21, 1]`
ind. par reg. indexé pré-inc.	`[xd, k]!`	$\mathtt{x_d} + k$	$\mathtt{x_d} \leftarrow \mathtt{x_d} + k$ avant calcul	`[x20, 1]!`
ind. par reg. indexé post-inc.	`[xd], k`	$\mathtt{x_d} + k$	$\mathtt{x_d} \leftarrow \mathtt{x_d} + k$ après calcul	`[x20], 1`
relatif	`etiq`	adresse de `etiq`	—	`main100`

Logique et manipulation de bits

▶ Instructions, où i est une valeur immédiate de 12 bits et j est une valeur immédiate de 6 bits:

Code d'op.	Syntaxe	Effet	Exemple

and	`and rd, rn, rm`
	`and rd, rn, i`
	`and rd, rn, rm, decal j`
		$r_d \leftarrow r_n \wedge r_m$
		$r_d \leftarrow r_n \wedge i$
		$r_d \leftarrow r_n \wedge (r_m \; decal \; j)$
			`and x19, x20, x21`
			`and x19, x20, 42`
			`and x19, x20, x21, lsl 1`

orr	`orr rd, rn, rm`
	`orr rd, rn, i`
	`orr rd, rn, rm, decal j`
		$r_d \leftarrow r_n \vee r_m$
		$r_d \leftarrow r_n \vee i$
		$r_d \leftarrow r_n \vee (r_m \; decal \; j)$
			`orr x19, x20, x21`
			`orr x19, x20, 42`
			`orr x19, x20, x21, lsl 1`

eor	`eor rd, rn, rm`
	`eor rd, rn, i`
	`eor rd, rn, rm, decal j`
		$r_d \leftarrow r_n \oplus r_m$
		$r_d \leftarrow r_n \oplus i$
		$r_d \leftarrow r_n \oplus (r_m \; decal \; j)$
			`eor x19, x20, x21`
			`eor x19, x20, 42`
			`eor x19, x20, x21, lsl 1`

mvn	`mvn rd, rn`	$r_d \leftarrow \neg r_n$	`mvn x19, x20`
lsl	`lsl xd, xn, j`	décalage de j bits vers la gauche: $x_d\langle 63, j\rangle \leftarrow x_n\langle 63-j, 0\rangle$; $x_d\langle j-1, 0\rangle \leftarrow 0$.	`lsl x19, x20, 1`
lsr	`lsr xd, xn, j`	décalage de j bits vers la droite: $x_d\langle 63-j, 0\rangle \leftarrow x_n\langle 63, j\rangle$; $x_d\langle 63, 64-j\rangle \leftarrow 0$	`lsr x19, x20, 1`
ror	`ror xd, j`	rotation de j bits vers la droite: $x_d \leftarrow x_n\langle j-1, 0\rangle \; x_n\langle 63, j\rangle$	`ror x19, 1`

Autres instructions

Code d'op.	Syntaxe	Effet	Exemple
csel	`csel rd, rn, rm, cond`	si *cond*: $r_d \leftarrow r_n$, sinon: $r_d \leftarrow r_m$	`csel x19, x20, x21, eq`

Données statiques

Segments de données

Pseudo-instruction	Contenu
`.section ".text"`	instructions
`.section ".rodata"`	données en lecture seule
`.section ".data"`	données initialisées
`.section ".bss"`	données non-initialisées

Données

`.align`	k	donnée suivante stockée à une adresse divisible par k
`.skip`	k	réserve k octets
`.ascii`	s	chaîne de caractères initialisée à s
`.asciz`	s	chaîne de caractères initialisée à s suivi du carac. nul
`.byte`	v	octet initialisé à v
`.hword`	v	demi-mot initialisé à v
`.word`	v	mot initialisé à v
`.xword`	v	double mot initialisé à v
`.single`	f	nombre en virg. flottante simple précision initialisé à f
`.double`	f	nombre en virg. flottante double précision initialisé à f

Entrées/sorties (haut niveau)

▶ Affichage: `printf(&format, val1, val2, ...)`

▶ Lecture: `scanf(&format, &var1, &var2, ...)`

▶ Spécificateurs de format:

Famille	Format	Type
Nombres sur 32 bits	`%d`	entier décimal signé
	`%u`	entier décimal non signé
	`%X`	entier hexadécimal non signé
	`%f`	nombre en virgule flottante
Nombres sur 64 bits	`%ld`	entier décimal signé
	`%lu`	entier décimal non signé
	`%lX`	entier hexadécimal non signé
	`%lf`	nombre en virgule flottante
Caractères	`%c`	caractère (1 octet)
	`%s`	chaîne de caractères