IFT209 – Programmation système

Université de Sherbrooke

Examen périodique

Enseignant: Michael Blondin
Date: jeudi 27 février 2020
Durée: 110 min. (08h30 à 10h20)

Directives:

- Vous devez répondre aux questions dans le **cahier de réponses**, *pas* sur ce questionnaire;
- **Une seule feuille (recto seulement)** de notes manuscrites au format 8½" × 11" est permise;
- **Aucun matériel additionnel** (notes de cours, fiches récapitulatives, etc.) n'est permis;
- **Aucun appareil électronique** (calculatrice, téléphone, tablette, ordinateur, etc.) n'est permis;
- Vous devez donner **une seule réponse** par sous-question;
- L'examen comporte **6 questions** sur **4 pages** valant un total de **50 points**;
- La correction se base sur la **clarté**, l'**exactitude** et la **concision** de vos réponses, ainsi que sur la **justification** pour les questions qui en requièrent une;
- À moins d'avis contraire, le langage d'assemblage utilisé est celui de l'**architecture ARMv8** tel qu'utilisé en classe; un sommaire de cette architecture est présenté en **annexe**.

Question 1: systèmes de numération

(a) 2 pts

(b) 2 pts

(c) 2 pts

Question 2: architecture des ordinateurs

(a) 3 pts

(b) 2 pts

Question 3: mémoire et accès aux données

(a) 2 pts

(b) 2 pts

(c) 4 pts

Question 4: entiers signés et circuits logiques

(a) 3 pts

(b) 4 pts

(c) 4 pts

Question 5: tableaux

(a) 1 pt

(b) 2 pts

(c) 7 pts

Question 6: programmation structurée en langage d'assemblage

(a) 3 pts

(b) 7 pts

Annexe:

Sommaire de l’architecture ARMv8

Registres.

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

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

Arithmétique (entiers).

- Les codes de condition sont modifiés par `cmp`, `adds`, `adcs`, `subs`, `sbcs` et `negs`
- À cette différence près, `adds`, `adcs`, `subs`, `sbcs` et `negs` se comportent respectivement comme `add`, `adc`, `sub`, `sbc` 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 $\mathtt{r}_d$ et $\mathtt{r}_m$	`cmp x19, x21`
	`cmp rd, i`	compare $\mathtt{r}_d$ et i	`cmp x19, 42`
	`cmp rd, rm, decal j`	compare $\mathtt{r}_d$ et $\mathtt{r}_m$ *decal* j	`cmp x19, x21, lsl 1`
`add`	`add rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + \mathtt{r}_m$	`add x19, x20, x21`
	`add rd, rn, i`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + i$	`add x19, x20, 42`
	`add rd, rn, rm, decal j`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + (\mathtt{r}_m$ *decal* $j)$	`add x19, x20, x21, lsl 1`
`adc`	`adc rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n + \mathtt{r}_m + \mathtt{C}$	`adc x19, x20, x21`
`sub`	`sub rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - \mathtt{r}_m$	`sub x19, x20, x21`
	`sub rd, rn, i`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - i$	`sub x19, x20, 42`
	`sub rd, rn, rm, decal j`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - (\mathtt{r}_m$ *decal* $j)$	`sub x19, x20, x21, lsl 1`
`sbc`	`sbc rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n - \mathtt{r}_m - 1 + \mathtt{C}$	`sbc x19, x20, x21`
`neg`	`neg rd, rm`	$\mathtt{r}_d \leftarrow -\mathtt{r}_m$	`neg x19, x21`
	`neg rd, rm, decal j`	$\mathtt{r}_d \leftarrow -(\mathtt{r}_m$ *decal* $j)$	`neg x19, x21, lsl 1`
`mul`	`mul rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n \cdot \mathtt{r}_m$	`mul x19, x20, x21`
`udiv`	`udiv rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n \div \mathtt{r}_m$ (non signé)	`udiv x19, x20, x21`
`sdiv`	`sdiv rd, rn, rm`	$\mathtt{r}_d \leftarrow \mathtt{r}_n \div \mathtt{r}_m$ (signé)	`sdiv x19, x20, x21`
`madd`	`madd rd, rn, rm, ra`	$\mathtt{r}_d \leftarrow \mathtt{r}_a + (\mathtt{r}_n \cdot \mathtt{r}_m)$	`madd x19, x20, x21, x22`
`msub`	`msub rd, rn, rm, ra`	$\mathtt{r}_d \leftarrow \mathtt{r}_a - (\mathtt{r}_n \cdot \mathtt{r}_m)$	`msub x19, x20, x21, x22`

Accès mémoire.

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

Code d'op.	Syntaxe	Effet	Exemple
`mov`	`mov rd, rm`	$\mathtt{r_d} \leftarrow \mathtt{r_m}$	`mov x19, x21`
	`mov rd, i`	$\mathtt{r_d} \leftarrow i$	`mov x19, 42`
`ldr`	`ldr xd, a`	charge 8 octets: $\mathtt{x_d}\langle 63,0\rangle \leftarrow \mathrm{mem}_8[a]$	`ldr x19, [x20]`
	`ldr wd, a`	charge 4 octets: $\mathtt{x_d}\langle 31,0\rangle \leftarrow \mathrm{mem}_4[a]$; $\mathtt{x_d}\langle 63,32\rangle \leftarrow 0$	`ldr w19, [x20]`
`ldrh`	`ldrh wd, a`	charge 2 octets: $\mathtt{x_d}\langle 15,0\rangle \leftarrow \mathrm{mem}_2[a]$; $\mathtt{x_d}\langle 63,16\rangle \leftarrow 0$	`ldrh w19, [x20]`
`ldrb`	`ldrb wd, a`	charge 1 octet: $\mathtt{x_d}\langle 7,0\rangle \leftarrow \mathrm{mem}_1[a]$; $\mathtt{x_d}\langle 63,8\rangle \leftarrow 0$	`ldrb w19, [x20]`
`str`	`str xd, a`	stocke 8 octets: $\mathrm{mem}_8[a] \leftarrow \mathtt{x_d}\langle 63,0\rangle$	`str x19, [x20]`
	`str wd, a`	stocke 4 octets: $\mathrm{mem}_4[a] \leftarrow \mathtt{x_d}\langle 31,0\rangle$	`str w19, [x20]`
`strh`	`strh wd, a`	stocke 2 octets: $\mathrm{mem}_2[a] \leftarrow \mathtt{x_d}\langle 15,0\rangle$	`str w19, [x20]`
`strb`	`strb wd, a`	stocke 1 octet: $\mathrm{mem}_1[a] \leftarrow \mathtt{x_d}\langle 7,0\rangle$	`strb w19, [x20]`
`ldp`	`ldp xd, xn, a`	charge 16 octets: $\mathtt{x_d}\langle 63,0\rangle \leftarrow \mathrm{mem}_8[a]$, $\mathtt{x_n}\langle 63,0\rangle \leftarrow \mathrm{mem}_8[a+8]$	`ldp x19, x20, [sp]`
`stp`	`stp xd, xn, a`	stocke 16 octets: $\mathrm{mem}_8[a] \leftarrow \mathtt{x_d}\langle 63,0\rangle$, $\mathrm{mem}_8[a+8] \leftarrow \mathtt{x_n}\langle 63,0\rangle$	`stp x19, x20, [sp]`

Conditions de branchement.

- Codes de condition: `N` (négatif), `Z` (zéro), `C` (report), `V` (débordement)
- `C` indique aussi l'*absence* d'emprunt lors d'une soustraction
- 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, 1, main100`
`tbnz`	`tbnz rd, j, etiq`	branche à `etiq:` si $\mathtt{r_d}\langle j\rangle \neq 0$	`tbnz x19, 1, 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`	—	$x_d \leftarrow$ adresse de `etiq:`	`adr x19, main100`
indirect par registre	`[xd]`	x_d	—	`[x20]`
indirect par registre indexé	`[xd, xn]`	$x_d + x_n$	—	`[x20, x21]`
	`[xd, k]`	$x_d + k$	—	`[x20, 1]`
	`[xd, xn, decal k]`	$x_d + (x_n \; decal \; k)$	—	`[x20, x21, lsl 1]`
ind. par reg. indexé pré-inc.	`[xd, k]!`	$x_d + k$	$x_d \leftarrow x_d + k$ avant calcul	`[x20, 1]!`
ind. par reg. indexé post-inc.	`[xd], k`	x_d	$x_d \leftarrow x_d + k$ après calcul	`[x20], 1`
relatif	`etiq`	adresse de `etiq`	—	`main100`

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`
`mvn`	`mvn rd, rn`	$r_d \leftarrow \neg r_n$ (négation de chacun des bits)	`mvn x19, x20`

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