IFT209 – Programmation système

Université de Sherbrooke

Examen périodique

Enseignant: Michael Blondin
Date: samedi 20 février 2021
Durée: 110 min.

Directives:

- Vous devez répondre aux questions dans le **cahier de réponses**, *pas* sur ce questionnaire;
- **Une seule feuille (recto verso)** de notes manuscrites au format $8\frac{1}{2}'' \times 11''$ 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 **5 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

Effectuez les conversions des nombres *non signés* suivants, en laissant une trace de votre démarche. Si une base intermédiaire est requise, ne passez *pas* par la base 10.

(a) 86 de la base 10 vers la base 2 — 2 pts

(b) 27415 de la base 8 vers la base 16 — 2 pts

(c) 87 + 5/8 de la base 10 vers la base 2 — 2 pts

Considérons une extension du système binaire où le bit le plus à gauche représente le signe: 1 signifie « positif » et 0 signifie « négatif ». Par exemple, dans ce système 111 représente 3, et 0101 représente -5.

(d) Quel est le *plus petit* nombre représentable sur n bits dans ce système? Justifiez brièvement. — 2 pts

Question 2: architecture des ordinateurs

Considérons l'architecture fictive Pico209. Elle possède quatre registres d'usage général de 8 bits: x_0, x_1, x_2 et x_3. Son jeu d'instructions est constitué des cinq instructions suivantes, chacune codée sur 16 bits, où i désigne une valeur immédiate de 4 bits signés:

instruction	**effet**	**format du code machine**	
		1er octet	2ème octet
`add xd, xn, xm`	$x_d \leftarrow x_n + x_m$	`00 00 00 00`	`__ __ __ 00` (d, n, m)
`inc xd`	$x_d \leftarrow x_d + 1$	`00 00 00 01`	`__ 11 00 00` (d)
`mov xd, i`	$x_d \leftarrow i$	`00 00 00 10`	`__ 11 ____` (d, i)
`jeq xd, xn, i`	additionne $2i$ au compteur d'instructions si $x_d = x_n$	`00 00 00 11`	`__ __ ____` (d, n, i)
`jmp i`	additionne $2i$ au compteur d'instructions	`00 00 01 00`	`00 11 ____` (i)

Rappel: le compteur d'instructions, aussi dénoté « program counter » ou PC, est le registre interne qui contient l'adresse de la ligne de code à exécuter par l'unité de contrôle.

(a) Le tableau ci-dessous illustre un programme Pico209 et son code machine. Remplissez les deux instructions manquantes ainsi que la portion de code machine manquante: 6 pts

programme	**code machine** (en hexadécimal)
`mov x1, 0`	`0x0270`
`mov x2, 0`	`0x????`
`jeq x0, x1, 4`	`0x0314`
`???? ??????`	`0x0050`
`???? ??????`	`0x01B0`
`jmp -3`	`0x043D`
`// fin du programme`	

(b) Quel est le plus grand saut vers l'avant qu'on peut accomplir avec un branchement `jmp`, en terme de nombre de lignes de code (c.-à-d. en nombre d'instructions)? Justifiez brièvement. 1 pt

Question 3: mémoire et accès aux données

Rappelons que, dans notre contexte, l'architecture ARMv8 utilise le format petit-boutiste (« little-endian »). Supposons que la mémoire principale contienne ces données:

adresse	contenu
⋮	⋮
$\texttt{0DE6}_{16}$	$\texttt{10}_{16}$
$\texttt{0DE7}_{16}$	$\texttt{00}_{16}$
$\texttt{0DE8}_{16}$	$\texttt{EB}_{16}$
$\texttt{0DE9}_{16}$	$\texttt{0D}_{16}$
$\texttt{0DEA}_{16}$	$\texttt{01}_{16}$
$\texttt{0DEB}_{16}$	$\texttt{02}_{16}$
$\texttt{0DEC}_{16}$	$\texttt{C4}_{16}$
$\texttt{0DED}_{16}$	$\texttt{FE}_{16}$
$\texttt{0DEE}_{16}$	$\texttt{66}_{16}$
⋮	⋮

(a) Quelle est la valeur hexadécimale du *mot* stocké à l'adresse $\texttt{0DEA}_{16}$? 2 pts

(b) L'adresse $\texttt{0DEA}_{16}$ respecte-t-elle les contraintes d'*alignement* pour l'adressage d'un octet? d'un demi-mot? d'un mot? d'un double mot? Justifiez. 2 pts

(c) Supposons que l'adresse numérique associée à l'étiquette « `donnees:` » soit $\texttt{0DE8}_{16}$, et que les registres soient initialisés à 0. Décrivez l'évolution du contenu de $\texttt{x}_{19}$ et $\texttt{x}_{20}$ après l'exécution de *chacune* de ces instructions: 3 pts

```
instruction1:   adr   x19, donnees
instruction2:   ldrh  w20, [x19], 3
instruction3:   ldrb  w19, [x19]
instruction4:   ldrb  w19, [x20, x19]
```

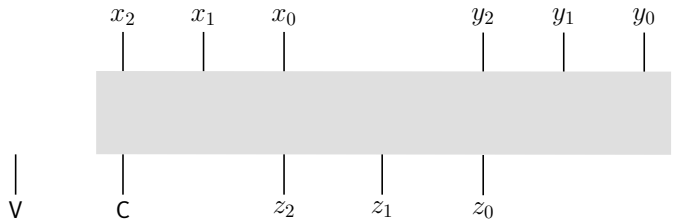
Question 4: entiers signés et circuits logiques

(a) Donnez la représentation binaire signée des nombres -28 et 19 sur huit bits, puis calculez $-28-19$ sur huit bits. Laissez une trace. Indiquez s'il y a débordement et/ou report. 3 pts

(b) Le registre $\texttt{x}_{19}$ vaut 1 après l'exécution du programme ci-dessous. Pourquoi? 2 pts

```
mov     x19, -1
adds    x19, x19, 1
adc     x19, x19, x19
```

(c) Le circuit ci-dessous prend en entrée deux entiers signés x et y de 3 bits, et retourne leur somme z ainsi que le bit de report `C` (« **c**arry »). Ajoutez des portes logiques afin que la sortie `V` (« o**v**erflow ») indique s'il y a eu débordement. Comme les détails internes ne sont pas illustrés, vos portes n'ont accès qu'à x, y, z et `C`. 3 pts



Remarque: si vous avez de la difficulté à dessiner les pictogrammes utilisés en classe, écrivez simplement le nom des portes.

Question 5: tableaux

Considérons une matrice **B** de n lignes et n colonnes dont les éléments sont des entiers signés de 64 bits, dont la dernière ligne et la dernière colonne contiennent uniquement des zéros, et dont les autres éléments sont non nuls. Voici un exemple d'une telle matrice:

1	−1	1	−4	0
4	2	−2	−3	0
1	1	1	−1	0
2	3	−1	1	0
0	0	0	0	0

On traverse **B** de la façon suivante. On débute à l'indice $(0, 0)$. Si l'élément actuel vaut $x > 0$, alors on se déplace de x éléments vers la droite; si l'élément actuel vaut $x < 0$, alors on se déplace de $|x|$ éléments vers le bas; sinon on termine. On suppose qu'on n'a jamais à quitter **B** et qu'il n'y a pas de boucle infinie. Dans l'exemple ci-dessus, on termine en quatre déplacements (via les éléments colorés).

Supposons que **B** soit stockée en mémoire (ligne à ligne) dans un tableau bidimensionnel qui débute à l'adresse a.

(a) Combien d'*octets* sont nécessaires afin de stocker **B**? 1 pt

(b) Si le premier élément de **B** vaut -3, à quelle *adresse* doit-on se déplacer? 2 pts

(c) Complétez le code suivant afin de calculer le nombre de déplacements requis afin de traverser **B**. 7 pts

```
// on suppose que
//  x19 = a  (adresse du tableau 2D qui contient B)
//  x20 = n  (n > 0)

/************************
      CODE À COMPLÉTER
************************/

// x28 doit contenir le nombre de déplacements afin
//                                      de traverser B selon les règles
```

Question 6: programmation structurée en langage d'assemblage

Considérons une alarme qui sonne du lundi au vendredi durant les cinq premières minutes de chaque heure (et qui cesse de sonner à la cinquième minute). Une journée est représentée par $j \in \{1, 2, \ldots, 7\}$ où 1 correspond à dimanche et 7 à samedi. Le temps est représenté par le nombre t de secondes depuis minuit le jour même. Par exemple, 01:02:05 est représenté par $t = 3600 + 120 + 5 = 3725$. On désire écrire un programme qui lit une journée j et un temps t, puis qui indique si l'alarme sonne (1) ou non (0). Voici des exemples d'entrées et sorties:

Entrée	$j = 1, t = 67$ (dimanche 00:01:07)	$j = 2, t = 0$ (lundi 00:00:00)	$j = 2, t = 299$ (lundi 00:04:59)	$j = 2, t = 300$ (lundi 00:05:00)	$j = 3, t = 3725$ (mardi 01:02:05)
Sortie	0	1	1	0	1

(a) Complétez l'*appel* au *sous-programme* `alarme` ci-dessous. 3 pts

(b) Complétez le *corps* du *sous-programme* `alarme` ci-dessous. 7 pts

```
.global  main
.include "macros_save_restore.s"        // Accès aux macros SAVE et RESTORE

main:                                   // main()
      // Lire journée j et temps t      // {
      adr     x0, fmtLecture            //
      adr     x1, temp                  //
      bl      scanf                     //   scanf("%lu", &temp)
      ldr     x19, temp                 //   j = temp
                                        //
      adr     x0, fmtLecture            //
      adr     x1, temp                  //
      bl      scanf                     //   scanf("%lu", &temp)
      ldr     x20, temp                 //   t = temp
                                        //
      // L'alarme sonne?                //
      /**** (a) CODE À COMPLÉTER ****/  //   b = alarme(j, t)
                                        //
      adr     x0, fmtSortie             //
      mov     x1, x21                   //
      bl      printf                    //   printf("%lu\n", b)
                                        //
      // Quitter                        //
      mov     x0, 0                     //   exit(0)
      bl      exit                      // }

// Entrées: journée j dans {1, 2, 3, 4, 5, 6, 7} (entier non signé de 64 bits)
//          temps t en secondes depuis minuit    (entier non signé de 64 bits)
// Sortie:  1 si l'alarme sonne au temps t de la journée j, 0 sinon
alarme:
      /**** (b) CODE À COMPLÉTER ****/

.section ".bss"
            .align  8
temp:       .skip   8

.section ".rodata"
fmtLecture:   .asciz  "%lu"
fmtSortie:    .asciz  "%lu\n"
```

Annexe:

Sommaire de l’architecture ARMv8

Registres.

- Chaque registre $\mathtt{x}_n$ possède 64 bits: $b_{63}b_{62}\cdots b_1b_0$
- Notation: $\mathtt{x}_n\langle i\rangle \coloneqq b_i$, $\mathtt{x}_n\langle i,j\rangle \coloneqq b_ib_{i-1}\cdots b_j$, $\mathtt{r}_n$ 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`

Sera couvert après la relâche:

Logique et manipulation de bits.

- Les instructions `lsl`, `lsr`, `asr` et `ror` possèdent également une variante de 32 bits utilisant les registres w_d, w_n et w_m (dans ce cas, les 32 bits de poids fort sont mis à 0)
- 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
`mvn`	`mvn rd, rn`	$r_d \leftarrow \neg r_n$	`mvn x19, x20`
`and`	`and rd, rn, rm`	$r_d \leftarrow r_n \wedge r_m$	`and x19, x20, x21`
	`and rd, rn, i`	$r_d \leftarrow r_n \wedge i$	`and x19, x20, 4`
	`and rd, rn, rm, decal j`	$r_d \leftarrow r_n \wedge (r_m \; decal \; j)$	`and x19, x20, x21, lsl 1`
`orr`	`orr rd, rn, rm`	$r_d \leftarrow r_n \vee r_m$	`orr x19, x20, x21`
	`orr rd, rn, i`	$r_d \leftarrow r_n \vee i$	`orr x19, x20, 4`
	`orr rd, rn, rm, decal j`	$r_d \leftarrow r_n \vee (r_m \; decal \; j)$	`orr x19, x20, x21, lsl 1`
`eor`	`eor rd, rn, rm`	$r_d \leftarrow r_n \oplus r_m$	`eor x19, x20, x21`
	`eor rd, rn, i`	$r_d \leftarrow r_n \oplus i$	`eor x19, x20, 4`
	`eor rd, rn, rm, decal j`	$r_d \leftarrow r_n \oplus (r_m \; decal \; j)$	`eor x19, x20, x21, lsl 1`
`bic`	`bic rd, rn, rm`	$r_d \leftarrow r_n \wedge \neg r_m$	`bic x19, x20, x21`
	`bic rd, rn, i`	$r_d \leftarrow r_n \wedge \neg i$	`bic x19, x20, 4`
	`bic rd, rn, rm, decal j`	$r_d \leftarrow r_n \wedge \neg(r_m \; decal \; j)$	`bic x19, x20, x21, lsl 1`
`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`
`asr`	`asr xd, xn, j`	décalage arithmétique de j bits vers la droite: $x_d\langle 63-j, 0\rangle \leftarrow x_n\langle 63, j\rangle$; $x_d\langle 63, 64-j\rangle \leftarrow x_n\langle 63\rangle$	`asr x19, x20, 1`
`ror`	`ror xd, xn, j`	décalage circulaire de j bits vers la droite: $x_d \leftarrow x_n\langle j-1, 0\rangle \; x_n\langle 63, j\rangle$	`ror x19, xn, 1`

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 64 bits	`%ld`	entier décimal signé
	`%lu`	entier décimal non signé
	`%lX`	entier hexadécimal non signé
	`%lf`	nombre en virgule flottante
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 16 bits	`%hd`	entier décimal signé
	`%hu`	entier décimal non signé
	`%hX`	entier hexadécimal non signé
Caractères	`%c`	caractère (1 octet)
	`%s`	chaîne de caractères