IFT209 – Programmation système

Université de Sherbrooke

Examen périodique

Enseignant: Michael Blondin
Date: jeudi 24 février 2022
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 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 **8 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) 14F5 de la base 16 vers la base 8 — 2 pts

1 4 F 5

0001 0100 1111 0101

1 2 3 6 5

(b) 77 de la base 10 vers la base 2 — 2 pts

1001101 (car $77 = 64 + 8 + 4 + 1$)

(c) 111010 de la base 2 vers la base 10 — 2 pts

58 (car $32 + 16 + 8 + 2 = 58$)

Considérons les nombres fractionnaires binaires de la forme « $\overbrace{x_3\,x_2\,x_1\,x_0}^{\text{4 bits}}\,,\,\overbrace{y_1\,y_2\,y_3}^{\text{3 bits}}$ ».

(d) Quel est le plus grand nombre strictement inférieur à 1 représentable dans ce système? Donnez votre réponse à l'aide d'une fraction (par exemple: « 25 / 32 »). Justifiez. — 2 pts

Plus il y a de bit à 1, plus le nombre est grand. Par contre, la partie entière doit être à zéro, sinon le nombre excède 1. Ainsi, le plus grand nombre est $1/2 + 1/4 + 1/8 = 7/8$.

Question 2: architecture des ordinateurs

Considérons l'architecture fictive GIGA209 au format grand-boutiste (« big-endian »). Elle possède 16 registres d'usage général de 32 bits: r_0, r_1, …, r_{15}. Son jeu d'instructions est constitué des quatre 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
`sub rd, rn, rm`	$r_d \leftarrow r_n - r_m$	0000 ____ (d)	____ (n) ____ (m)
`dadd rd, i`	$r_d \leftarrow 2 \cdot r_d + i$	0100 ____ (d)	1111 ____ (i)
`jzr rd, i`	additionne $2i$ au compteur d'instructions si $r_d = 0$	1000 ____ (d)	0000 ____ (i)
`load rd, i`	charge dans r_d les 4 octets situés à une distance de $2i$	1111 1111	____ (d) ____ (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 GIGA209 et son code machine. Remplissez les deux instructions manquantes ainsi que les deux portions de code machine manquantes: 6 pts

programme	code machine (en hexadécimal)
`sub r15, r15, r15`	0x0FFF
`sub r14, r14, r14`	0x0EEE
`sub r0, r0, r0`	0x0000
`load r1, 6`	0xFF16
`dadd r14, 1`	0x????
`jzr r1, 6`	0x8106
`?? ????????`	0x40F1
`sub r1, r1, r14`	0x????
`?? ????????`	0x8F0D
`// donnée figée`	0x0000
`//          en mémoire`	0x000A
`// fin du programme ici`	

programme	**code machine** (en hexadécimal)
`dadd r14, 1`	`0x4EF1`
`dadd r0, 1`	`0x40F1`
`sub r1, r1, r14`	`0x011E`
`jzr r15, -3`	`0x8F0D`

(b) Rappelons qu'une architecture RISC est caractérisée par un jeu d'instructions constitué: 2 pts

— de *peu* d'instructions;
— d'instructions relativement *simples*;
— d'instructions qui *ne combinent pas les accès mémoire* à d'autres types d'opérations;
— d'instructions dont la traduction vers le code machine est de *taille fixe*.

Comme les deux premières propriétés sont subjectives, supposons qu'elles sont satisfaites par GIGA209. Dites si GIGA209 est une architecture RISC. Justifiez.

Oui:
— Toutes les instructions sont codées sur 16 bits;
— La seule instruction mémoire est `load` et elle n'effectue aucune opération additionnelle.

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
⋮	⋮
$\texttt{0FCC}_{16}$	$\texttt{01}_{16}$
$\texttt{0FCD}_{16}$	$\texttt{42}_{16}$
$\texttt{0FCE}_{16}$	$\texttt{1F}_{16}$
$\texttt{0FCF}_{16}$	$\texttt{BC}_{16}$
$\texttt{0FD0}_{16}$	$\texttt{04}_{16}$
$\texttt{0FD1}_{16}$	$\texttt{02}_{16}$
$\texttt{0FD2}_{16}$	$\texttt{CD}_{16}$
$\texttt{0FD3}_{16}$	$\texttt{0F}_{16}$
$\texttt{0FD4}_{16}$	$\texttt{00}_{16}$
⋮	⋮

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

BC1F_{16}

(b) L'adresse $\texttt{0FCE}_{16}$ 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

L'adresse $\texttt{0FCE}_{16} = \cdots 1110_2$ est un multiple de 2, mais pas de 4 ou 8. Elle respecte donc seulement les contraintes pour un octet et un demi-mot.

(c) Supposons que l'adresse numérique associée à l'étiquette « `donnees:` » soit $\texttt{0FCE}_{16}$, 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: 4 pts

```
instruction1:   adr   x19, donnees
instruction2:   ldrb  w20, [x19, 2]
instruction3:   ldrh  w19, [x19, x20]
instruction4:   ldr   w20, [x19], 1

// Rappel: ldr xd (64 bits), ldr wd (32 bits), ldrh wd (16 bits), ldrb wd (8 bits)
```

$\texttt{x}_{19}$	$\texttt{x}_{20}$
0x0FCE	0x00
0x0FCE	0x04
0x0FCD	0x04
0x0FCE	0x04BC1F42

Question 4: entiers signés et circuits logiques

(a) Donnez la représentation binaire signée des nombres -25 et 12 sur six bits, puis calculez $-25 - 12$ sur six bits. Laissez une trace. Indiquez s'il y a débordement et/ou report. Justifiez. *3 pts*

On a $-25 = -32 + 4 + 2 + 1 = 100111$ et $12 = 8 + 4 = 001100$. Calculons -12:

$$001100 \xrightarrow{\text{complément}} 110011 \xrightarrow{+1} 110100.$$

On obtient:

$$\begin{array}{rr} & {\scriptstyle 1\ \ 1}\ \ \ \ \\ + & 100111 \\ & 110100 \\ \hline & 011011 \end{array}$$

Il y a report car il y a une retenue à la fin. Il y a aussi débordement car l'addition de deux nombres négatifs donne une somme positive.

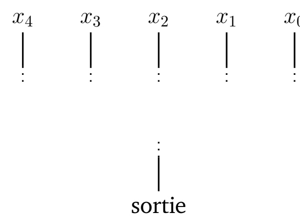
(b) Effectuez le produit de ces deux nombres signés de $n = 4$ bits: 1011×0101. Utilisez l'algorithme « simple », c.-à-d. celui qui étend les nombres sur $2n$ bits, fait les calculs non signés, puis tronque le résultat sur $2n$ bits. *2 pts*

11100111 obtenu ainsi:

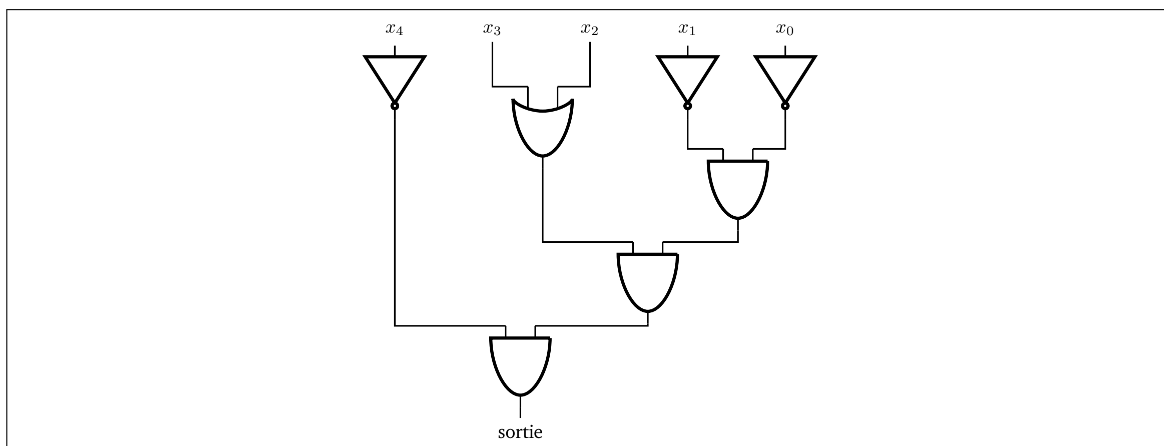
$$\begin{array}{rrl} & 11111011 & \text{(-5)} \\ \times & 00000101 & \text{(5)} \\ \hline & 11111011 & \\ + & 11111011 & \\ \hline & 10011100111 & \text{(-25)} \end{array}$$

Rappel: vous avez implémenté cet algorithme au labo 2.

(c) Complétez ce circuit afin qu'il détermine si un entier signé de 5 bits est un multiple de 4 strictement positif: *3 pts*



Remarque: dans le doute quant à la lisibilité, utilisez le nom des portes plutôt que leur pictogramme.



Question 5: programmation en langage d'assemblage

Le cours « IFT902 – Théorie des questions d'examen » est offert à la session d'hiver (janvier à avril inclusivement) de chaque année impaire depuis l'an 2020, donc aux hivers 2021, 2023, etc. Nous cherchons à implémenter un programme qui, étant donné une date, indique la prochaine date à laquelle IFT902 sera donné. Si le cours est offert à la date entrée, alors la date en entrée est simplement retournée en sortie.

Dans notre contexte, une *date* est représentée par un entier non signé qui indique le nombre de mois écoulés depuis janvier 2020. Voici des exemples d'entrées et sorties:

Entrée	0 (janv. 2020)	4 (mai 2020)	12 (janv. 2021)	13 (févr. 2021)	15 (avr. 2021)	16 (mai 2021)
Sortie	12 (janv. 2021)	12 (janv. 2021)	12 (janv. 2021)	13 (févr. 2021)	15 (avr. 2021)	36 (janv. 2023)

Complétez le code du programme ci-dessous afin d'accomplir la tâche décrite. 8 pts

```
.global main

// Usage des registres:
//   x19 -- entree,  x28 -- sortie
main:                                   // main()
    // Lire une date                    // {
    adr     x0, fmtEntree               //
    adr     x1, temp                    //
    bl      scanf                       //   scanf(fmtEntree, &temp)
    ldr     x19, temp                   //   entree = temp
                                        //
    /**** CODE ARMv8 À COMPLÉTER ****/  //   /* calcul de la sortie */
                                        //
    // Affichage                        //
    adr     x0, fmtSortie               //
    mov     x1, x28                     //
    bl      printf                      //   printf(fmtSortie, sortie)
                                        //
    // Quitter                          //
    mov     x0, 0                       //
    bl      exit                        //   exit(0)
                                        // }
.section ".bss"
            .align  8
temp:       .skip   8

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

Ma solution

```
    mov     x22, 12         //
                            //
    udiv    x20, x19, x22   // annee = entree / 12
    mul     x21, x20, x22   //
    sub     x21, x19, x21   // mois  = entree % 12
                            //
    tbnz    x20, 0, impair  // if (annee % 2 == 0)
pair:                       // {
    add     x20, x20, 1     //   annee += 1
    mov     x21, 0          //   mois   = 0
    b       calcul          // }
                            // else
impair:                     // {
    cmp     x21, 4          //   if (mois >= 4)
    b.lo    calcul          //   {
    add     x20, x20, 2     //     annee += 2
    mov     x21, 0          //     mois   = 0
                            //   }
calcul:                     // }
    mul     x28, x22, x20   //
    add     x28, x28, x21   // sortie = 12*annee + mois
```

Adaptation d'une solution étudiante

```
    mov     x20, 12         // debut = 12
    mov     x21, 15         // fin   = 15
                            //
boucle:                     // while (true) {
    cmp     x19, x20        //   if (entree < debut) {
    b.lo    retour_debut    //     sortie = debut
                            //     break
                            //   }
    cmp     x19, x21        //   else if (entree <= fin)
    b.ls    retour_entree   //     sortie = entree
                            //     break
                            //   }
    add     x20, x20, 24    //   debut += 24
    add     x21, x21, 24    //   fin   += 24
    b       boucle          // }
retour_debut:               //
    mov     x28, x20        //
    b       affichage       //
retour_entree:              //
    mov     x28, x19        //
affichage:                  //
```

Question 6: tableaux

Considérons un tableau **B** de m lignes et n colonnes où n est impair, et où les éléments sont des entiers non signés de 16 bits. Nous disons que **B** est *en forme de T* si:

- il contient 1 dans chaque élément de sa première ligne et de sa colonne du centre;
- il contient 0 partout ailleurs.

Par exemple, ce tableau est en forme de T:

1	1	1	1	1
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

Attention: il ne s'agit que d'un exemple; répondez aux questions par rapport à m et n, pas 4 et 5.

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

(a) Si **B** est en forme de T, alors combien d'*octets* de **B** sont nuls? 1,5 pts

$$\underbrace{2mn}_{\text{\# octets}} - \underbrace{(m+n-1)}_{\text{\# octets non nuls}}$$

(b) Quelle est l'*adresse* du premier élément de la colonne du centre de **B** (par ex. celui hachuré ci-dessus)? 1,5 pts

Sol. 1: $a + 2 \cdot (n \div 2)$

Sol. 2: $a + 2 \cdot ((n-1)/2)$

Sol. 3: $a + (n-1)$

(c) Supposons que tous les éléments de **B** soient 0. Complétez le code suivant afin de mettre **B** en forme de T (donc, stocker 1 aux éléments appropriés). 7 pts

```
// on suppose que
//  x19 = a  (adresse du tableau 2D qui contient B)
//  x20 = m  (m > 0)
//  x21 = n  (n > 0 et impair)
//  B ne contient que des 0

/*************************
  CODE ARMv8 À COMPLÉTER
*************************/

// B doit maintenant être en forme de T
```

Une explication de la solution se trouve en ligne.

```
        mov     x22, 1                //
                                      //
        // Remplir la ligne           //
        mov     x23, x19              // ptr  = a
        mov     x24, x21              // iter = n
    boucle_ligne:                     // do {
        strh    w22, [x23], 2         //   *ptr = 1; ptr += 2
        sub     x24, x24, 1           //   iter--
        cbnz    x24, boucle_ligne     // } while (iter != 0)
                                      //
        // Remplir la colonne         //
        sub     x23, x21, 1           //
        add     x23, x19, x23         // ptr  = a + (n - 1)
        mov     x24, x20              // iter = m
    boucle_col:                       // do {
        strh    w22, [x23]            //   *ptr = 1
        add     x23, x23, x21         //
        add     x23, x23, x21         //   ptr += 2*n
        sub     x24, x24, 1           //   iter--
        cbnz    x24, boucle_col       // } while (iter != 0)
```

Annexe:

Sommaire de l'architecture ARMv8

Registres.

- Chaque registre $\texttt{x}_\texttt{n}$ possède 64 bits: $b_{63}b_{62}\cdots b_1 b_0$
- Notation: $\texttt{x}_\texttt{n}\langle i\rangle := b_i$, $\texttt{x}_\texttt{n}\langle i,j\rangle := b_i b_{i-1}\cdots b_j$, $\texttt{r}_\texttt{n}$ réfère au registre $\texttt{x}_\texttt{n}$ ou $\texttt{w}_\texttt{n}$
- Chaque sous-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`, `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 $\texttt{r}_\texttt{d}$ 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`
`sbc`	`sbc rd, rn, rm`	$\texttt{r}_\texttt{d} \leftarrow \texttt{r}_\texttt{n} - \texttt{r}_\texttt{m} - 1 + \texttt{C}$	`sbc x19, x20, x21`
`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.

- `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 $\mathtt{x}_\mathtt{d}$ où les bits excédentaires sont le bit de signe de la donnée chargée, plutôt que des zéros
- Instructions, où a est une adresse et $\text{mem}_b[a]$ réfère aux b octets à l'adresse a de la mémoire principale:

Code d'op.	Syntaxe	Effet	Exemple
`mov`	`mov rd, rm`	$\mathtt{r}_\mathtt{d} \leftarrow \mathtt{r}_\mathtt{m}$	`mov x19, x21`
	`mov rd, i`	$\mathtt{r}_\mathtt{d} \leftarrow i$	`mov x19, 42`
`ldr`	`ldr xd, a`	charge 8 octets: $\mathtt{x}_\mathtt{d}\langle 63, 0\rangle \leftarrow \text{mem}_8[a]$	`ldr x19, [x20]`
	`ldr wd, a`	charge 4 octets: $\mathtt{x}_\mathtt{d}\langle 31, 0\rangle \leftarrow \text{mem}_4[a]$; $\mathtt{x}_\mathtt{d}\langle 63, 32\rangle \leftarrow 0$	`ldr w19, [x20]`
`ldrh`	`ldrh wd, a`	charge 2 octets: $\mathtt{x}_\mathtt{d}\langle 15, 0\rangle \leftarrow \text{mem}_2[a]$; $\mathtt{x}_\mathtt{d}\langle 63, 16\rangle \leftarrow 0$	`ldrh w19, [x20]`
`ldrb`	`ldrb wd, a`	charge 1 octet: $\mathtt{x}_\mathtt{d}\langle 7, 0\rangle \leftarrow \text{mem}_1[a]$; $\mathtt{x}_\mathtt{d}\langle 63, 8\rangle \leftarrow 0$	`ldrb w19, [x20]`
`str`	`str xd, a`	stocke 8 octets: $\text{mem}_8[a] \leftarrow \mathtt{x}_\mathtt{d}\langle 63, 0\rangle$	`str x19, [x20]`
	`str wd, a`	stocke 4 octets: $\text{mem}_4[a] \leftarrow \mathtt{x}_\mathtt{d}\langle 31, 0\rangle$	`str w19, [x20]`
`strh`	`strh wd, a`	stocke 2 octets: $\text{mem}_2[a] \leftarrow \mathtt{x}_\mathtt{d}\langle 15, 0\rangle$	`str w19, [x20]`
`strb`	`strb wd, a`	stocke 1 octet: $\text{mem}_1[a] \leftarrow \mathtt{x}_\mathtt{d}\langle 7, 0\rangle$	`strb w19, [x20]`
`ldp`	`ldp xd, xn, a`	charge 16 octets: $\mathtt{x}_\mathtt{d}\langle 63, 0\rangle \leftarrow \text{mem}_8[a]$, $\mathtt{x}_\mathtt{n}\langle 63, 0\rangle \leftarrow \text{mem}_8[a+8]$	`ldp x19, x20, [sp]`
`stp`	`stp xd, xn, a`	stocke 16 octets: $\text{mem}_8[a] \leftarrow \mathtt{x}_\mathtt{d}\langle 63, 0\rangle$, $\text{mem}_8[a+8] \leftarrow \mathtt{x}_\mathtt{n}\langle 63, 0\rangle$	`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 $\mathtt{r}_\mathtt{d} = 0$	`cbz x19 main100`
`cbnz`	`cbnz rd, etiq`	branche à `etiq:` si $\mathtt{r}_\mathtt{d} \neq 0$	`cbnz x19 main100`
`tbz`	`tbz rd, j, etiq`	branche à `etiq:` si $\mathtt{r}_\mathtt{d}\langle j\rangle = 0$	`tbz x19, 1, main100`
`tbnz`	`tbnz rd, j, etiq`	branche à `etiq:` si $\mathtt{r}_\mathtt{d}\langle j\rangle \neq 0$	`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 $\mathtt{x}_\mathtt{d}$ et $\mathtt{x}_{30} \leftarrow \mathtt{pc} + 4$	`blr x20`
`br`	`br xd`	branche à $\mathtt{x}_\mathtt{d}$	`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`

Les manipulations de bits ne seront couvertes qu'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, val`$_1$`, val`$_2$`, ...)`
- Lecture: `scanf(&format, &var`$_1$`, &var`$_2$`, ...)`
- 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