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

1010110 obtenu par $86 = 64 + 16 + 4 + 2$ ou par:

$$
\begin{aligned}
86 \div 2 &= 43 \text{ reste } 0 \\
43 \div 2 &= 21 \text{ reste } 1 \\
21 \div 2 &= 10 \text{ reste } 1 \\
10 \div 2 &= 5 \text{ reste } 0 \\
5 \div 2 &= 2 \text{ reste } 1 \\
2 \div 2 &= 1 \text{ reste } 0 \\
1 \div 2 &= 0 \text{ reste } 1
\end{aligned}
$$

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

2	7	4	1	5
010	111	100	001	101

Regroupé par 4 bits: 2 F 0 D

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

$$\underbrace{1010111}_{(a)+1},\overbrace{101}^{5/8=4/8+1/8=1/2+1/8}$$

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

Le plus petit nombre est $0\underbrace{11\cdots1}_{n-1 \text{ fois}}$ dont la valeur est $-1 \cdot \underbrace{11\cdots1_2}_{n-1 \text{ fois}} = -(2^{n-1}-1) = -2^{n-1}+1$.

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`	

programme	code machine (en hexadécimal)
`mov x2, 0`	`0x02B0`
`add x1, x1, x0`	`0x0050`
`inc x2`	`0x01B0`

(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

Comme l'opérande `i` est signé, sa plus grande valeur est $0111 = 7$. Comme chaque instruction est codée sur deux octets, il y a `i` sauts. Le plus grand saut avant est donc de 7 instructions.

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

Le mot (4 octets) stocké à l'adresse $\texttt{0DEA}_{16}$ est:

$\texttt{01}_{16}$	$\texttt{02}_{16}$	$\texttt{C4}_{16}$	$\texttt{FE}_{16}$

Puisque le format est « little-endian », sa valeur est FEC40201_{16}.

(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

> Oui pour un octet et un demi-mot, et non pour un mot et un double mot.
>
> En effet, $0DEA_{16} = \cdots 1010_2$. Puisque le bit de poids faible vaut 0, l'adresse est un multiple de 2, et bien sûr de 1. Puisque le deuxième bit vaut 1, ce n'est pas un multiple de 4 (ou de 8).

(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]
```

$\texttt{x}_{19}$	$\texttt{x}_{20}$
0x0DE8	0x0000
0x0DEB	0x0DEB
0x0002	0x0DEB
0x00FE	0x0DEB

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

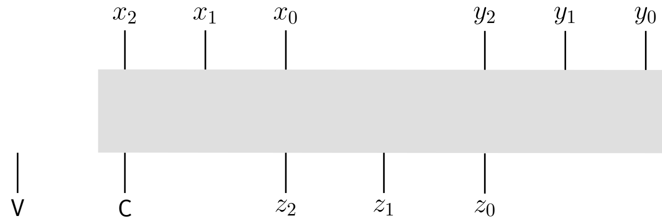
> On a $-28 = -32 + 4 = 100100$ et $19 = 16 + 2 + 1 = 010011$. En étendant les bits de signe, on obtient $-28 = 11100100$ et $19 = 00010011$. Calculons -19 par complément à deux:
>
> $$00010011 \xrightarrow{\text{complément}} 11101100 \xrightarrow{+1} 11101101.$$
>
> On obtient:
>
> $$\begin{array}{r} {\scriptstyle 111\ \ 11} \\ 11100100 \\ +\ 11101101 \\ \hline 11010001 \end{array}$$
>
> Il y a report car il y a une retenue à la toute fin, mais pas débordement car l'addition de deux nombres négatifs donne une somme négative comme attendu.

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

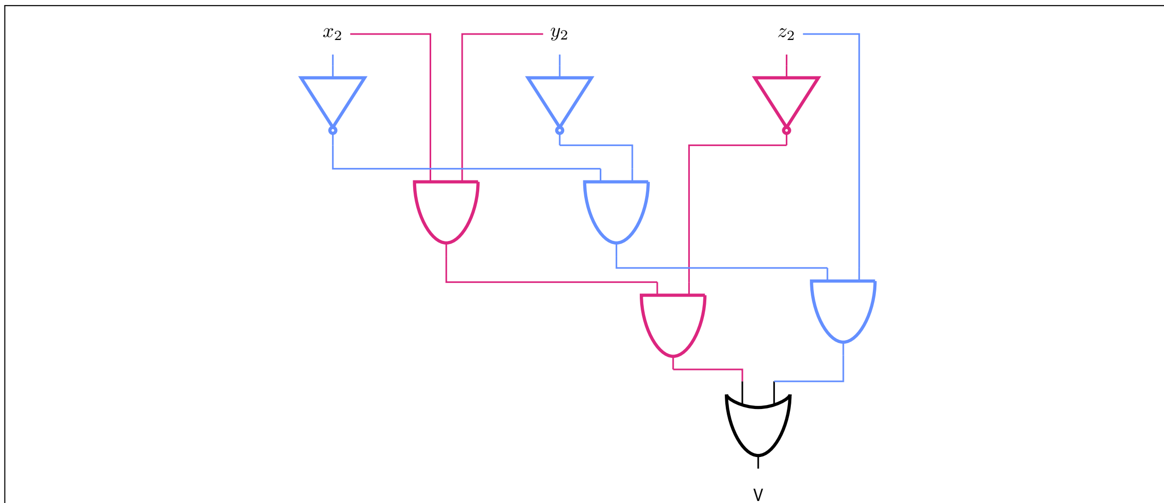
```
mov     x19, -1
adds    x19, x19, 1
adc     x19, x19, x19
```

> La première instruction assigne $\texttt{x}_{19} \leftarrow -1 = \overbrace{11\cdots1}^{64 \text{ fois}}$. La deuxième instruction additionne 1, ce qui mène à $\texttt{x}_{19} = 00\cdots0 = 0$ avec report. Comme « `adds` » met les codes de condition à jour, on a $\texttt{C} = 1$. La dernière instruction assigne donc $\texttt{x}_{19} \leftarrow \texttt{x}_{19} + \texttt{x}_{19} + \texttt{C} = 0 + 0 + 1 = 1$.

(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.



(Il y a débordement ssi on additionne deux nombres négatifs et qu'on obtient un nombre positif, ou vice-versa. Donc V $= ((x_2 \land y_2) \land \neg z_2) \lor ((\neg x_2 \land \neg y_2) \land z_2)$, ce qui correspond au circuit ci-dessus. Alternativement, on pourrait utiliser l'expression $(x_2 = y_2) \land (x_2 \neq z_2)$ qui s'exprime avec quatre portes: $\neg(x_2 \oplus y_2) \land (x_2 \oplus z_2)$.)

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

$8n^2$ (8 octets par élément, $n \cdot n$ éléments)

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

$a + 3 \cdot 8n$ (adresse de base + saut de 3 lignes, chaque ligne contient $8n$ octets)

(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
```

Solution sans calcul d'index

```
        mov   x28, 0        // nb_deplacements = 0
        mov   x21, 8        //
boucle:                     //
        ldr   x22, [x19]    //
        cmp   x22, 0        // while (mem[a] != 0) {
        b.eq  fin           //
        b.gt  deplacer      //   saut = mem[a]
bas:                        //
        neg   x22, x22      //   if (saut < 0) {
        mul   x22, x22, x20 //     saut = -saut * n
deplacer:                   //   }
        mul   x22, x21, x22 //
        add   x19, x19, x22 //   a += 8 * saut
        add   x28, x28, 1   //   nb_deplacements++
        b     boucle        // }
fin:                        //
```

Solution avec calcul d'index

```
        mov   x28, 0          // nb_deplacements = 0
        mov   x21, 8          //
        mov   x22, 0          // i = 0
        mov   x23, 0          // j = 0
boucle:                       //
        mul   x24, x22, x20   //
        add   x24, x24, x23   //
        mul   x24, x21, x24   // index = 8*(i*n + j)
                              //
        ldr   x25, [x19, x24] //
        cmp   x25, 0          // while (mem[a + index] != 0) {
        b.eq  fin             //
                              //   saut = mem[a + index]
        add   x28, x28, 1     //   nb_deplacements++
        b.lt  bas             //
droite:                       //   if (saut > 0) {
        add   x23, x23, x25   //     j += saut
        b     boucle          //   }
bas:                          //   else {
        sub   x22, x22, x25   //     i -= saut
        b     boucle          //   }
fin:                          // }
```

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"
```

```
                mov     x0, x19         //
                mov     x1, x20         //
                bl      alarme          //
                mov     x21, x0         // b = alarme(j, t)
```

```
    alarme:                             // alarme(j, t)
                SAVE                    // {
                mov     x19, x0         //
                mov     x0, 0           //   b = 0
                                        //
                cmp     x19, 1          //
                b.eq    alarme_ret      //
                cmp     x19, 7          //
                b.eq    alarme_ret      //   if (j != 1 && j != 7) {
                                        //
                mov     x20, 3600       //
                udiv    x21, x1, x20    //
                mul     x21, x21, x20   //
                sub     x1, x1, x21     //     secondes_depuis_heure = t % 3600
                                        //
                cmp     x1, 300         //
                b.hs    alarme_ret      //     if (secondes_depuis_heure < 5*60) {
                mov     x0, 1           //       b = 1
    alarme_ret:                         //     }
                RESTORE                 //   }
                ret                     //   return b
                                        // }
```