

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 **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) 14F5 de la base 16 vers la base 8 2 pts
- (b) 77 de la base 10 vers la base 2 2 pts
- (c) 111010 de la base 2 vers la base 10 2 pts

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

- (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

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, \dots, 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	
		1 ^{er} octet	2 ^{ème} octet
<code>sub rd, rn, rm</code>	$r_d \leftarrow r_n - r_m$	0 0 0 0 <u> </u> <i>d</i>	<u> </u> <u> </u> <u> </u> <i>n</i> <i>m</i>
<code>dadd rd, i</code>	$r_d \leftarrow 2 \cdot r_d + i$	0 1 0 0 <u> </u> <i>d</i>	1 1 1 1 <u> </u> <i>i</i>
<code>jzr rd, i</code>	additionne $2i$ au compteur d'instructions si $r_d = 0$	1 0 0 0 <u> </u> <i>d</i>	0 0 0 0 <u> </u> <i>i</i>
<code>load rd, i</code>	charge dans r_d les 4 octets situés à une distance de $2i$	1 1 1 1 1 1 1 1	<u> </u> <u> </u> <i>d</i> <i>i</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)
<code>sub r15, r15, r15</code>	0x0FFF
<code>sub r14, r14, r14</code>	0x0EEE
<code>sub r0, r0, r0</code>	0x0000
<code>load r1, 6</code>	0xFF16
<code>dadd r14, 1</code>	0x????
<code>jzr r1, 6</code>	0x8106
<code>?? ????????</code>	0x40F1
<code>sub r1, r1, r14</code>	0x????
<code>?? ????????</code>	0x8F0D
<code>// donnée figée</code>	0x0000
<code>// en mémoire</code>	0x000A
<code>// fin du programme ici</code>	

- (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.

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
⋮	⋮
0FCC ₁₆	01 ₁₆
0FCD ₁₆	42 ₁₆
0FCE ₁₆	1F ₁₆
0FCF ₁₆	BC ₁₆
0FD0 ₁₆	04 ₁₆
0FD1 ₁₆	02 ₁₆
0FD2 ₁₆	CD ₁₆
0FD3 ₁₆	0F ₁₆
0FD4 ₁₆	00 ₁₆
⋮	⋮

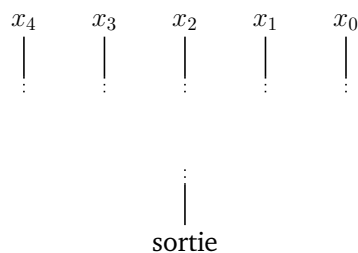
- (a) Quelle est la valeur hexadécimale du *demi-mot* stocké à l'adresse 0FCE₁₆? 2 pts
- (b) L'adresse 0FCE₁₆ respecte-t-elle les contraintes d'*alignement* pour l'adressage d'un octet? d'un demi-mot? d'un mot? d'un double mot? Justifiez. 2 pts
- (c) Supposons que l'adresse numérique associée à l'étiquette « *donnees:* » soit 0FCE₁₆, et que les registres soient initialisés à 0. Décrivez l'évolution du contenu de x₁₉ et x₂₀ 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)

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
- (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
- 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"
```

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

Annexe:

Sommaire de l'architecture ARMv8

Registres.

- ▶ Chaque registre x_n possède 64 bits: $b_{63}b_{62} \dots b_1b_0$
- ▶ Notation: $x_n \langle i \rangle := b_i$, $x_n \langle i, j \rangle := b_i b_{i-1} \dots b_j$, r_n réfère au registre x_n ou w_n
- ▶ Chaque sous-registre w_n possède 32 bits et correspond à $x_n \langle 31, 0 \rangle$
- ▶ Le compteur d'instruction pc n'est pas accessible
- ▶ Conventions:

Registres	Nom	Utilisation
$x_0 - x_7$	—	registres d'arguments et de retour de sous-programmes
x_8	xr	registre pour retourner l'adresse d'une structure
$x_9 - x_{15}$	—	registres temporaires sauvegardés par l'appelant
$x_{16} - x_{17}$	ip ₀ - ip ₁	registres temporaires intra-procéduraux
x_{18}	pr	registre temporaire pouvant être réservé par le système
$x_{19} - x_{28}$	—	registres temporaires sauvegardés par l'appelé
x_{29}	fp	pointeur vers l'ancien sommet de pile (<i>frame pointer</i>)
x_{30}	lr	registre d'adresse de retour (<i>link register</i>)
x_{31}	sp	registre contenant la valeur 0, ou pointeur de pile (<i>stack pointer</i>)

Arithmétique (entiers).

- ▶ Les codes de condition sont modifiés par **cmp**, **adds**, **adcs**, **subs**, **sbc** et **negs**
- ▶ À cette différence près, **adds**, **adcs**, **subs**, **sbc** 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 r_d et r_m	cmp x19, x21
	cmp rd, i	compare r_d et i	cmp x19, 42
	cmp rd, rm, decal j	compare r_d et r_m <i>decal j</i>	cmp x19, x21, lsl 1
add	add rd, rn, rm	$r_d \leftarrow r_n + r_m$	add x19, x20, x21
	add rd, rn, i	$r_d \leftarrow r_n + i$	add x19, x20, 42
	add rd, rn, rm, decal j	$r_d \leftarrow r_n + (r_m \text{ decal } j)$	add x19, x20, x21, lsl 1
adc	adc rd, rn, rm	$r_d \leftarrow r_n + r_m + C$	adc x19, x20, x21
sub	sub rd, rn, rm	$r_d \leftarrow r_n - r_m$	sub x19, x20, x21
	sub rd, rn, i	$r_d \leftarrow r_n - i$	sub x19, x20, 42
	sub rd, rn, rm, decal j	$r_d \leftarrow r_n - (r_m \text{ decal } j)$	sub x19, x20, x21, lsl 1
sbc	sbc rd, rn, rm	$r_d \leftarrow r_n - r_m - 1 + C$	sbc x19, x20, x21
neg	neg rd, rm	$r_d \leftarrow -r_m$	neg x19, x21
	neg rd, rm, decal j	$r_d \leftarrow -(r_m \text{ decal } j)$	neg x19, x21, lsl 1
mul	mul rd, rn, rm	$r_d \leftarrow r_n \cdot r_m$	mul x19, x20, x21
udiv	udiv rd, rn, rm	$r_d \leftarrow r_n \div r_m$ (non signé)	udiv x19, x20, x21
sdiv	sdiv rd, rn, rm	$r_d \leftarrow r_n \div r_m$ (signé)	sdiv x19, x20, x21
madd	madd rd, rn, rm, ra	$r_d \leftarrow r_a + (r_n \cdot r_m)$	madd x19, x20, x21, x22
msub	msub rd, rn, rm, ra	$r_d \leftarrow r_a - (r_n \cdot r_m)$	msub x19, x20, x21, x22

Accès mémoire.

- **ldrsb**, **ldrsh** et **ldrsb** se comportent respectivement comme **ldr** (4 octets), **ldrh** et **ldrb** à l'exception du fait qu'ils effectuent un chargement dans 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 $\text{mem}_b[a]$ réfère aux b octets à l'adresse a de la mémoire principale:

Code d'op.	Syntaxe	Effet	Exemple
mov	mov rd, rm	$r_d \leftarrow r_m$	mov x19, x21
	mov rd, i	$r_d \leftarrow i$	mov x19, 42
ldr	ldr xd, a	charge 8 octets: $x_d \langle 63, 0 \rangle \leftarrow \text{mem}_8[a]$	ldr x19, [x20]
	ldr wd, a	charge 4 octets: $x_d \langle 31, 0 \rangle \leftarrow \text{mem}_4[a]$; $x_d \langle 63, 32 \rangle \leftarrow 0$	ldr w19, [x20]
ldrh	ldrh wd, a	charge 2 octets: $x_d \langle 15, 0 \rangle \leftarrow \text{mem}_2[a]$; $x_d \langle 63, 16 \rangle \leftarrow 0$	ldrh w19, [x20]
ldrb	ldrb wd, a	charge 1 octet: $x_d \langle 7, 0 \rangle \leftarrow \text{mem}_1[a]$; $x_d \langle 63, 8 \rangle \leftarrow 0$	ldrb w19, [x20]
str	str xd, a	stocke 8 octets: $\text{mem}_8[a] \leftarrow x_d \langle 63, 0 \rangle$	str x19, [x20]
	str wd, a	stocke 4 octets: $\text{mem}_4[a] \leftarrow x_d \langle 31, 0 \rangle$	str w19, [x20]
strh	strh wd, a	stocke 2 octets: $\text{mem}_2[a] \leftarrow x_d \langle 15, 0 \rangle$	str w19, [x20]
strb	strb wd, a	stocke 1 octet: $\text{mem}_1[a] \leftarrow x_d \langle 7, 0 \rangle$	strb w19, [x20]
ldp	ldp xd, xn, a	charge 16 octets: $x_d \langle 63, 0 \rangle \leftarrow \text{mem}_8[a]$, $x_n \langle 63, 0 \rangle \leftarrow \text{mem}_8[a+8]$	ldp x19, x20, [sp]
stp	stp xd, xn, a	stocke 16 octets: $\text{mem}_8[a] \leftarrow x_d \langle 63, 0 \rangle$, $\text{mem}_8[a+8] \leftarrow x_n \langle 63, 0 \rangle$	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	=	Z
ne	≠	¬Z
hs	≥	C
hi	>	C ∧ ¬Z
ls	≤	¬C ∨ Z
lo	<	¬C

Entiers signés		
Code	Signification	Codes de condition
eq	=	Z
ne	≠	¬Z
ge	≥	N = V
gt	>	¬Z ∧ (N = V)
le	≤	Z ∨ (N ≠ V)
lt	<	N ≠ V
vs	débordement	V
vc	pas de débordement	¬V
mi	négatif	N
pl	non négatif	¬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 <i>cond</i>	b.eq main100
b	b etiq	branche à etiq :	b main100
cbz	cbz rd, etiq	branche à etiq : si $r_d = 0$	cbz x19 main100
cbnz	cbnz rd, etiq	branche à etiq : si $r_d \neq 0$	cbnz x19 main100
tbz	tbz rd, j, etiq	branche à etiq : si $r_d \langle j \rangle = 0$	tbz x19, 1, main100
tbnz	tbnz rd, j, etiq	branche à etiq : si $r_d \langle j \rangle \neq 0$	tbnz x19, 1, main100
bl	bl etiq	branche à etiq : et $x_{30} \leftarrow \text{pc} + 4$	bl printf
blr	blr xd	branche à x_d et $x_{30} \leftarrow \text{pc} + 4$	blr x20
br	br xd	branche à x_d	br x20
ret	ret	branche à 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]	$x_d + (x_n \text{ decal } k)$	—	[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	$r_d \leftarrow r_n \wedge (r_m \text{ decal } j)$	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	$r_d \leftarrow r_n \vee (r_m \text{ decal } j)$	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	$r_d \leftarrow r_n \oplus (r_m \text{ decal } j)$	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	$r_d \leftarrow r_n \wedge \neg (r_m \text{ decal } j)$	bic x19, x20, x21, lsl 1
lsl	lsl xd, xn, j	décalage de j bits vers la gauche: $x_d(63, j) \leftarrow x_n(63 - j, 0)$; $x_d(j - 1, 0) \leftarrow 0$	lsl x19, x20, 1
lsr	lsr xd, xn, j	décalage de j bits vers la droite: $x_d(63 - j, 0) \leftarrow x_n(63, j)$; $x_d(63, 64 - j) \leftarrow 0$	lsr x19, x20, 1
asr	asr xd, xn, j	décalage arithmétique de j bits vers la droite: $x_d(63 - j, 0) \leftarrow x_n(63, j)$; $x_d(63, 64 - j) \leftarrow x_n(63)$	asr x19, x20, 1
ror	ror xd, xn, j	décalage circulaire de j bits vers la droite: $x_d \leftarrow x_n(j - 1, 0) \ x_n(63, j)$	ror x19, xn, 1

Données statiques.

Segments de données		Données	
Pseudo-instruction	Contenu	.align <i>k</i>	donnée suivante stockée à une adresse divisible par <i>k</i>
.section ".text"	instructions	.skip <i>k</i>	réserve <i>k</i> octets
.section ".rodata"	données en lecture seule	.ascii <i>s</i>	chaîne de caractères initialisée à <i>s</i>
.section ".data"	données initialisées	.asciz <i>s</i>	chaîne de caractères initialisée à <i>s</i> suivi du carac. nul
.section ".bss"	données non-initialisées	.byte <i>v</i>	octet initialisé à <i>v</i>
		.hword <i>v</i>	demi-mot initialisé à <i>v</i>
		.word <i>v</i>	mot initialisé à <i>v</i>
		.xword <i>v</i>	double mot initialisé à <i>v</i>
		.single <i>f</i>	nombre en virg. flottante simple précision initialisé à <i>f</i>
		.double <i>f</i>	nombre en virg. flottante double précision initialisé à <i>f</i>

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
Nombres sur 16 bits	%f	nombre en virgule flottante
	%hd	entier décimal signé
	%hu	entier décimal non signé
Caractères	%hX	entier hexadécimal non signé
	%c	caractère (1 octet)
	%s	chaîne de caractères