

IFT209 – Programmation système
Université de Sherbrooke
Examen final

Enseignant: Michael Blondin
Date: jeudi 27 avril 2023
Durée: 3 heures

Directives:

- Répondez aux questions dans le **cahier de réponses**, pas sur ce questionnaire;
- **Une feuille (recto verso)** de notes au format $8\frac{1}{2}'' \times 11''$ est permise, et les fiches en **annexe**;
- **Aucun matériel additionnel** (notes de cours, examens antérieurs, etc.) n'est permis;
- **Aucun appareil électronique** (calculatrice, téléphone, tablette, ordinateur, etc.) n'est permis;
- Donnez **une seule réponse** par sous-question;
- L'examen comporte **5 questions** sur **6 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 est présenté à l'**annexe B**;
- La question 5 utilise le langage d'assemblage du **NES** tel qu'utilisé en classe; un sommaire est présenté à l'**annexe C**.

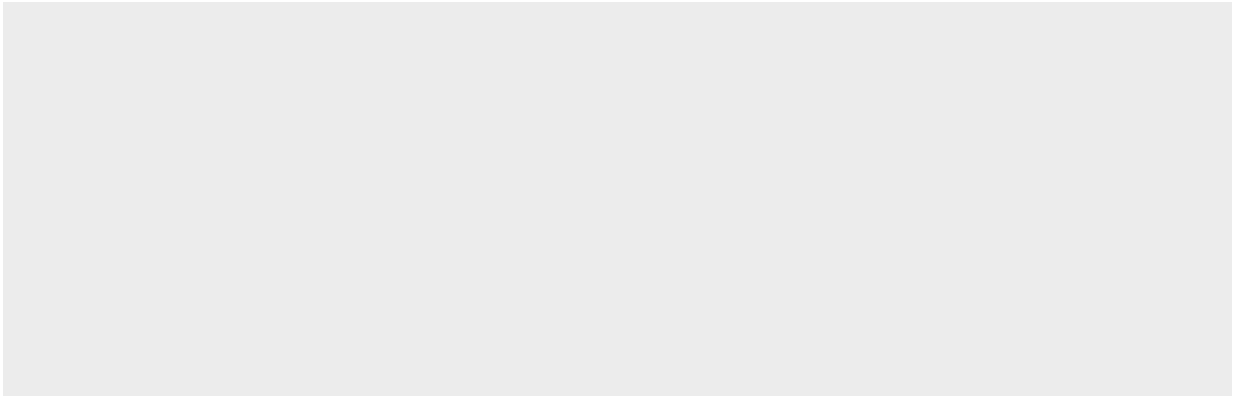
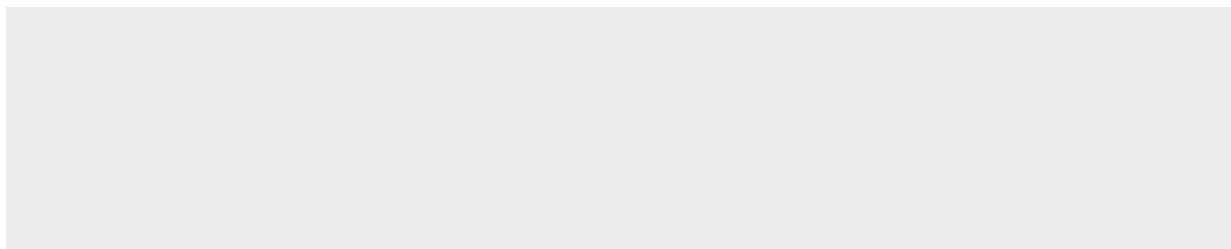
Question 1: valeurs booléennes et chaînes de bits

(a)

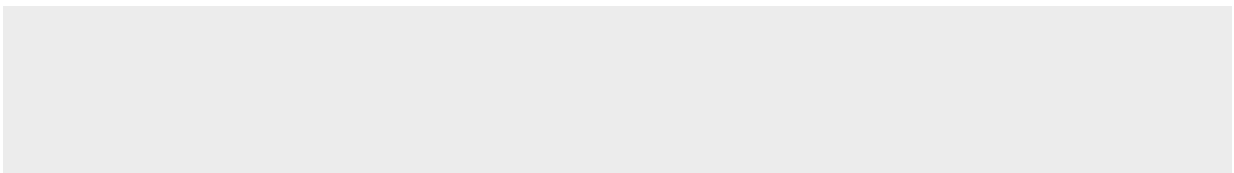
3 pts

(b)

5 pts

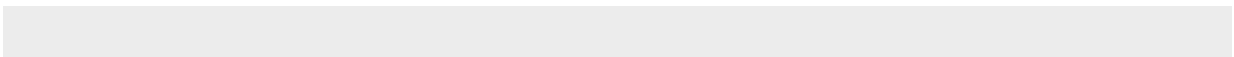
**Question 2: chaînes de caractères**

(a)



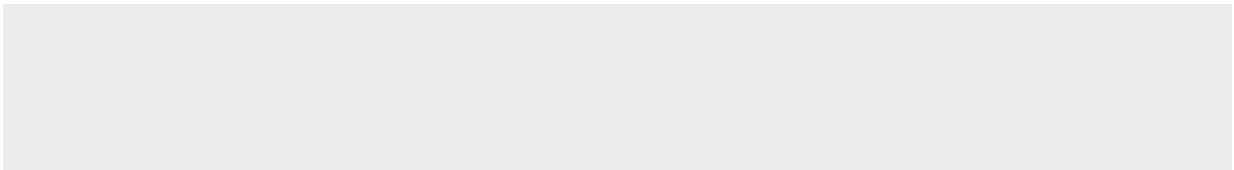
2 pts

(b)



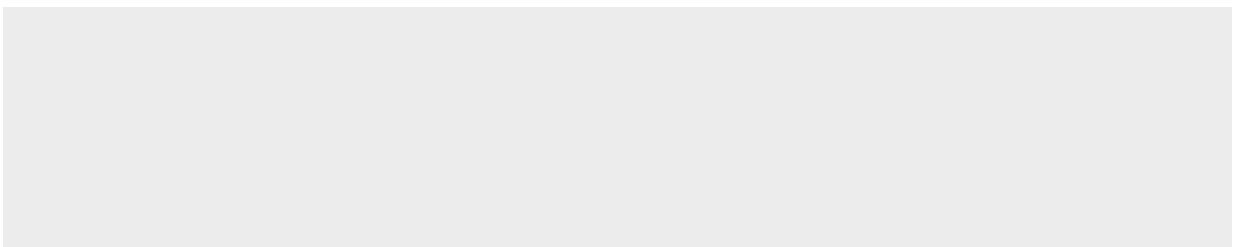
2 pts

(c)

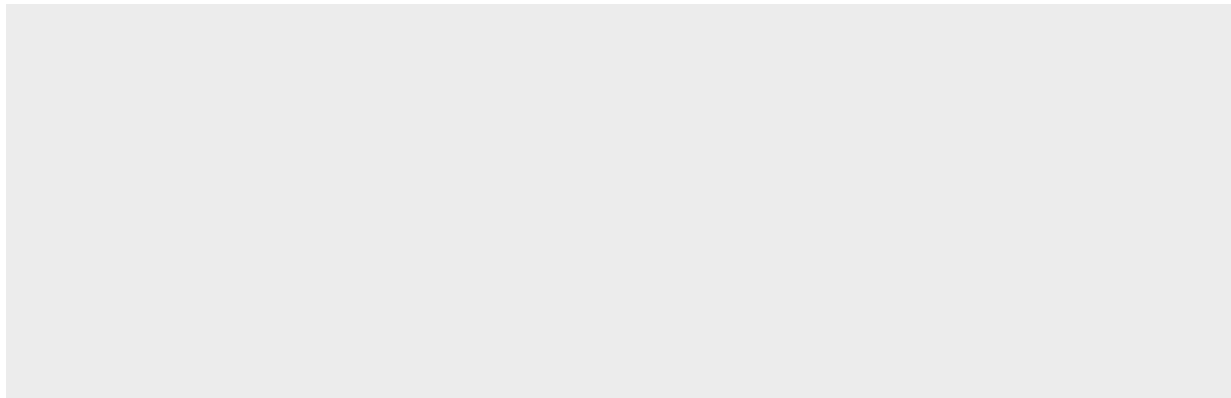


2 pts

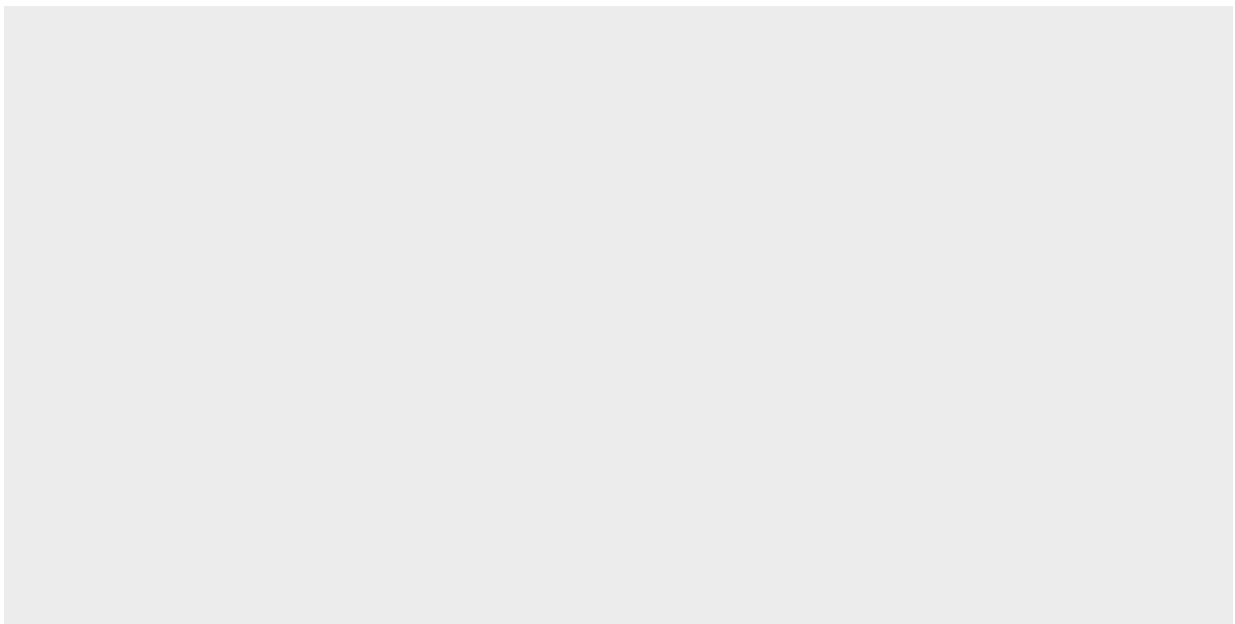
(d)



6 pts

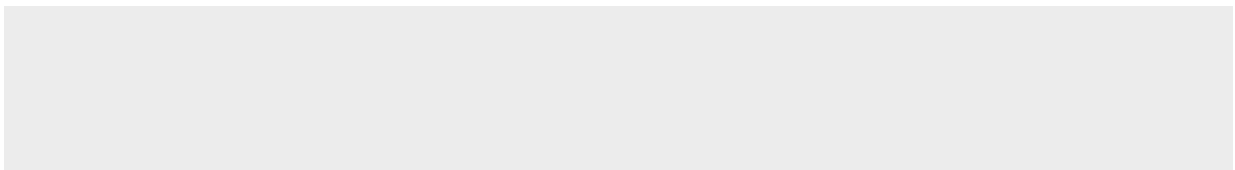
Question 3: sous-programmes et mémoire

(a)



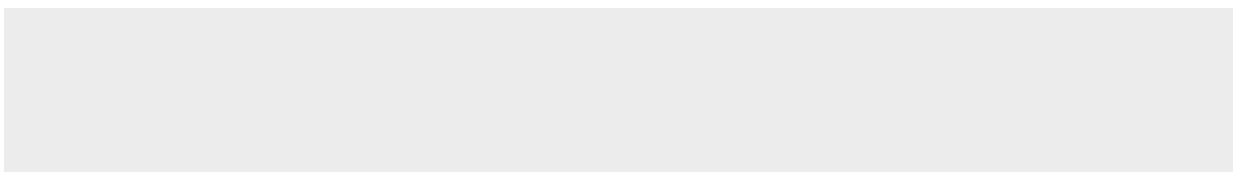
5 pts

(b)



2,5 pts

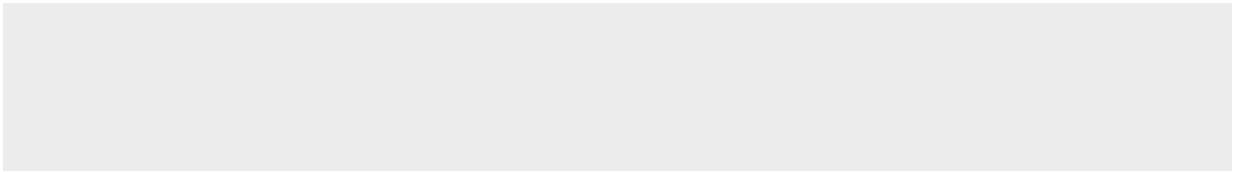
(c)



2,5 pts

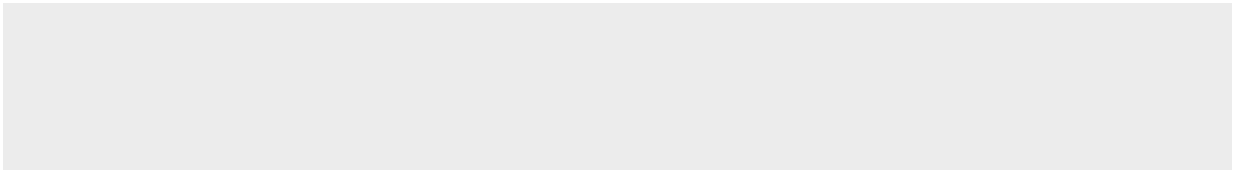
Question 4: nombres en virgule flottante

(a)



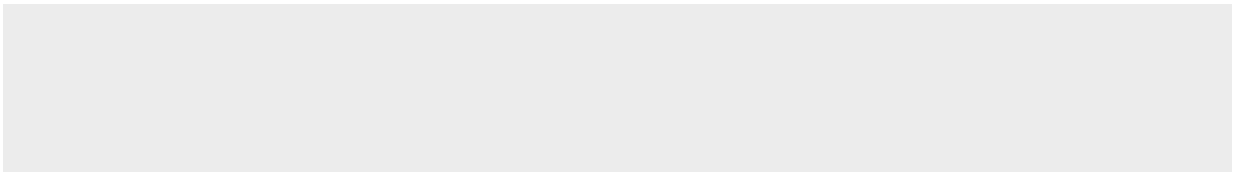
5 pts

(b)

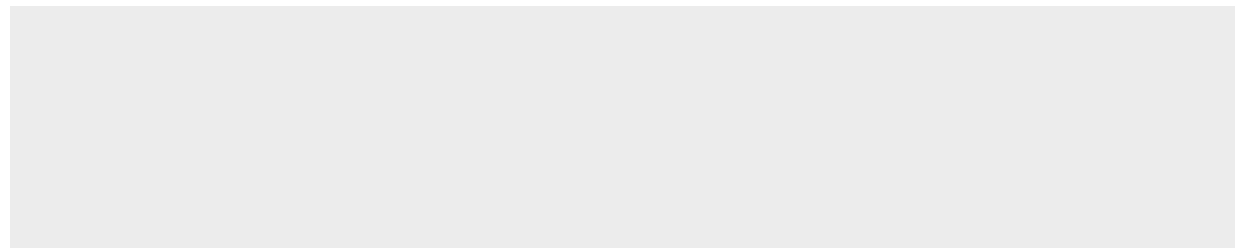


2,5 pts

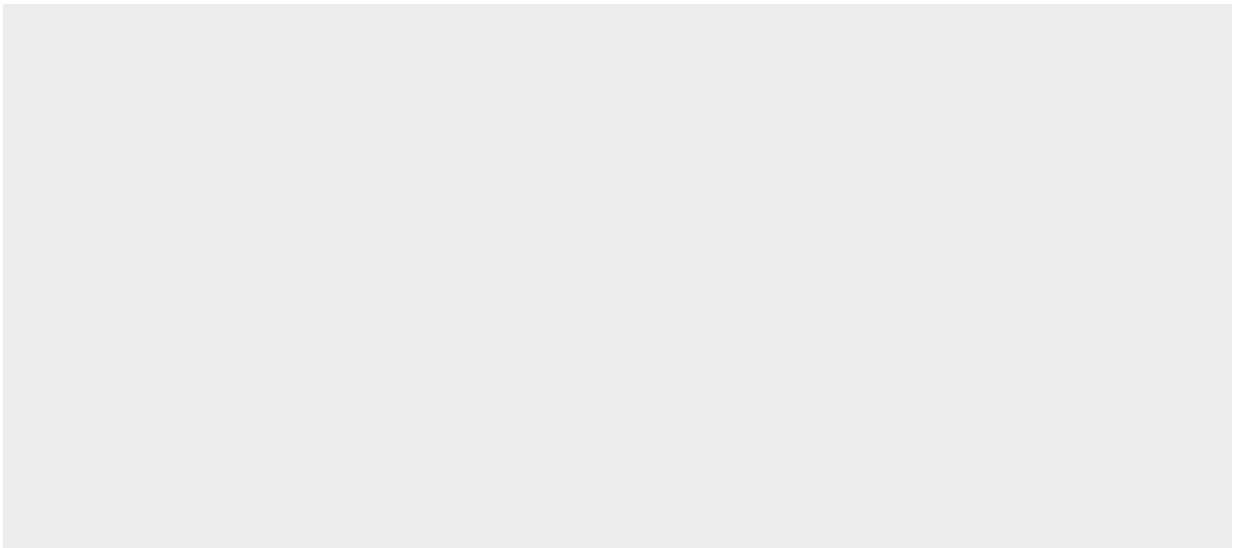
(c)



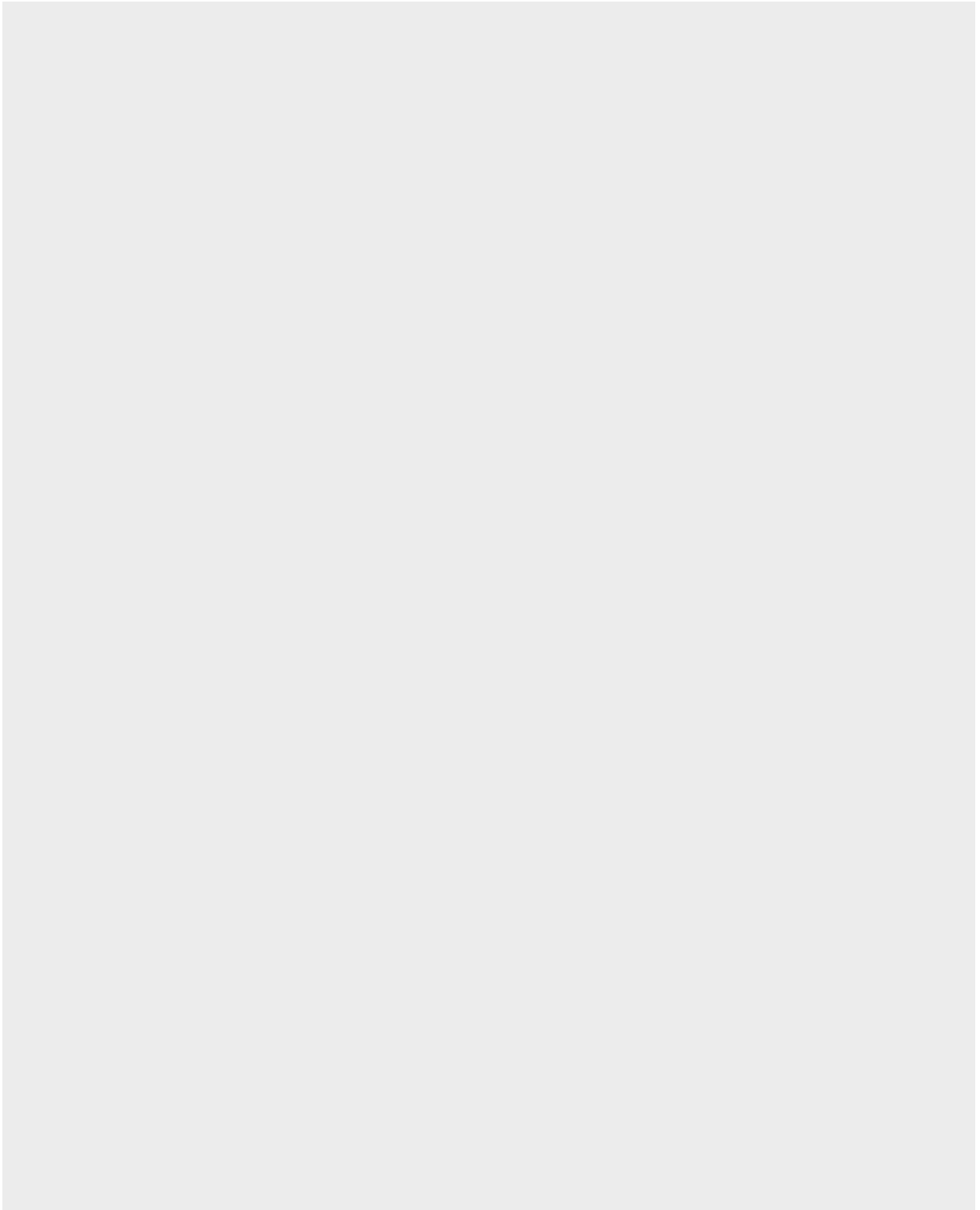
2,5 pts

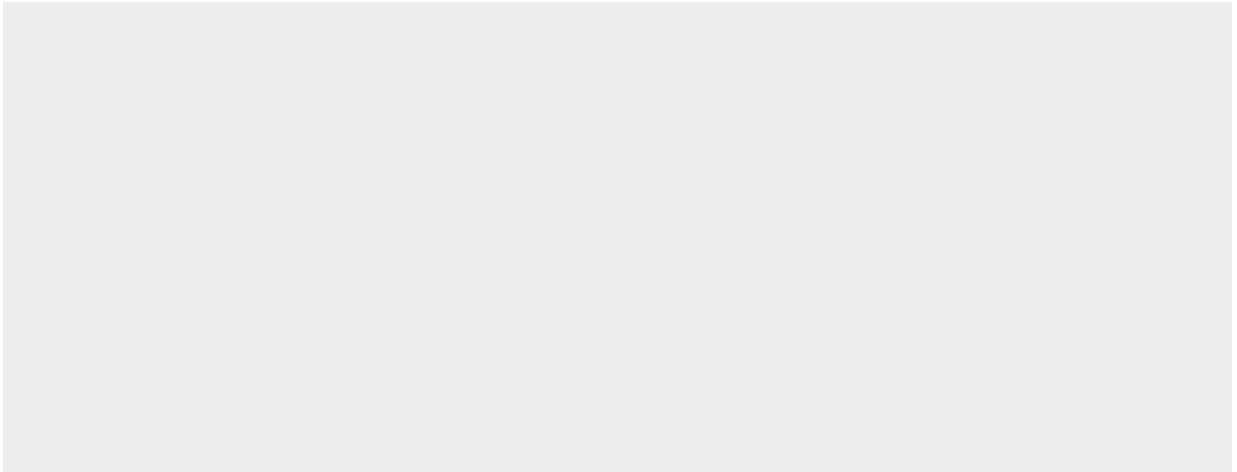
Question 5: entrées/sorties

(a)

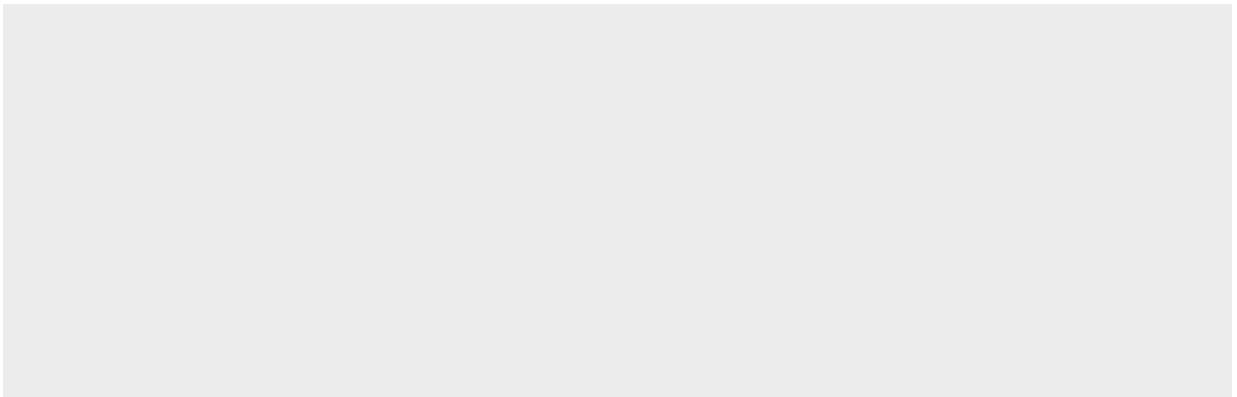


7 pts





(b)



3 pts

Annexe A:

Fiches récapitulatives

8. Programmation structurée

Séquence

- Composition séquentielle d'instructions
- Une instruction de haut niveau peut nécessiter plusieurs instructions de bas niveau; par ex. « `x19 *= 7` » devient:

```
mov    x20, 7
mul    x19, x19, x20
```

Sélection

- Exécution conditionnelle d'instructions (`if`, `switch`, ...)
- Implémentation: branchements avant:

```
if (cond(xd, xn)) {
    // code si
}
else {
    // code sinon
}

si:      cmp    xd, xn
        b.-cond  sinon
        // code si
        b      fin
sinon:   // code sinon
fin:
```

- Conditions multiples: obtenues avec plusieurs sélections

Itération

- Exécution répétée d'instructions (`while`, `do while`, `for`, ...)
- Implémentation: branchements arrière, et parfois avant:

```
while (cond(xd, xn)) {
    // code
}

boucle:
    cmp    xd, xn
    b.-cond  fin
    // code
    b      boucle
fin:
```

Sous-programmes

- Permettent de modulariser le code en sous-routines
- Registres partagés par programme et sous-programmes
- Arguments: passés par valeur ou adresse dans x_0-x_7 (en ordre)
- Appel: « `bl sprog` » assigne $x_{30} \leftarrow pc+4$ et branche à `sprog`:
- Retour: « `ret` » branche vers l'adresse de retour x_{30}
- Sauvegarde: l'appelé doit rétablir les registres x_{19} à x_{30}

9. Valeurs booléennes et chaînes de bits

Valeurs booléennes

- Correspond à un bit: 1 = vrai, 0 = faux
- Représentation: sur un octet, puisque bits non adressables

Opérateurs logiques

- Opérations: $\neg$, $\wedge$, $\vee$, $\oplus$ « bit à bit » étendues aux chaînes:

mnv x19, x20	and x19, x20, x21	orr x19, x20, x21	eor x19, x20, x21
0 ... 1 0 1	0 ... 1 0 1	0 ... 1 0 1	0 ... 1 0 1
1 ... 0 1 0	1 ... 1 0 0	1 ... 1 0 0	1 ... 1 0 0
1 ... 0 1 0	0 ... 1 0 0	1 ... 1 0 1	1 ... 0 0 1

- Échange de valeurs: se fait sans registre temporaire avec `eor`

Décalages logiques et arithmétiques

- Décale les bits de j positions vers la gauche/droite:

```
11000101 3 bits vers la gauche → 00101000    lsl xd, xn, 3
11000101 3 bits vers la droite → 00011000    lsr xd, xn, 3
```

- Bit de signe copié lors d'un décalage arithmétique à droite:

```
11000101 3 bits vers la droite → 11111000    asr xd, xn, 3
```

- Multiplication/division: par 2^k correspond à un décalage de k bits vers la gauche/droite

Décalages circulaires

- Comme un décalage logique, mais les bits « perdus » sont ré-insérés de l'autre côté:

```
11000101 3 bits vers la gauche → 00101110    n'existe pas sur ARMv8
11000101 3 bits vers la droite → 10111000    ror xd, xn, 3
```

Masquage

- Permet d'isoler certains bits à manipuler:

sélection	$r \wedge m$	met à 0 les bits de r non spécifiés par m
activation	$r \vee m$	met à 1 les bits de r spécifiés par m
désactivation	$r \wedge \neg m$	met à 0 les bits de r spécifiés par m
basculement	$r \oplus m$	inverse les bits de r spécifiés par m

10. Chaînes de caractères

Généralités

- Caractère: symbole représenté par une chaîne de bits
- Chaîne de caractères: suite finie de caractères, normalement terminée par un caractère nul

ASCII

- Représente 128 caractères codés sur 7 bits
- Lettre minuscule mise en majuscule en assignant le 6^{ème} bit de poids faible à 0, par ex. $a = 1100001_2$ et $A = 1000001_2$

ISO 8859-1 (Latin-1)

- Représente 256 caractères codés sur 8 bits
- Caractères 0 à 127: ASCII
- Caractères 128 à 255: lettres accentuées et autres caractères

UTF-8

- Représente > 1 000 000 caractères sur 1 à 4 octets
- Caractères 0 à 127: ASCII
- Caractères 128 à 255: ISO 8859-1, mais codés différemment
- Format général:

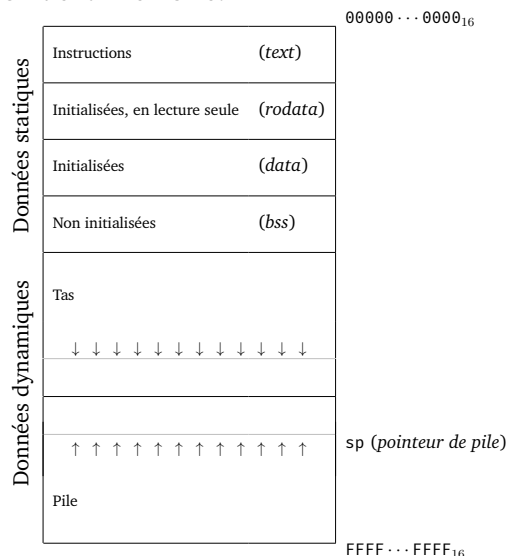
# bits	plage de codes ¹		format binaire des octets			
	début	fin	octet 1	octet 2	octet 3	octet 4
7	000000 ₁₆	00007F ₁₆	0*****	—	—	—
11	000080 ₁₆	0007FF ₁₆	110*****	10*****	—	—
16	000800 ₁₆	00FFFF ₁₆	1110****	10*****	10*****	—
21	010000 ₁₆	10FFFF ₁₆	11110***	10*****	10*****	10*****

- Exemples:

car.	code	codage
a	1100001 ₂	01100001 ₂
é	000 11101001 ₂	11000011 10101001 ₂
ヶ	00110000 10110001 ₂	11100011 10000010 10110001 ₂
𐀀	00001 00100100 00001101 ₂	11110000 10010010 10010000 10001101 ₂

11. Sous-programmes et mémoire

Disposition de la mémoire.



Tas.

- ▶ Contient les données allouées dynamiquement: structures de données, objets, etc.

Pile d'exécution.

- ▶ Stocke les données temporaires lors d'appel de sous-prog.
- ▶ Données empilées à l'appel et dépilées au retour
- ▶ *Pointeur de pile*: sp contient l'adresse du sommet de la pile
- ▶ *Empiler*: décrémenter sp + stocker avec **stp** xd, xn, a
- ▶ *Dépiler*: incrémenter sp + charger avec **ldp** xd, xn, a

Récursion.

- ▶ *Implémentée par*: appels de sous-prog. + usage de la pile
- ▶ *Récursion trop profonde*: erreur car la pile est bornée
- ▶ *Solution (partielle)*: empiler le moins de données possibles

12. Nombres en virgule flottante

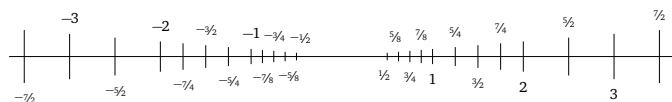
Représentation.

- ▶ *Nombre en virgule flottante*:

$$\underbrace{\pm}_{\text{signe}} \underbrace{d_0 d_1 d_2 \dots d_{n-1}}_{\text{mantisse en base } \beta} \times \underbrace{\beta^e}_{\text{exposant base}}$$

- ▶ *Normalisé*: si $d_0 \neq 0$

- ▶ Représente différents ordres de grandeur:



14. Entrées/sorties

Mécanismes d’entrée/sortie.

- *Attente active*: interrogation continue d’un registre d’état jusqu’à un événement (ex. *VBANK*)
- *Interruption*: signal lancé vers le processeur lors d’un événement (ex. NMI, RESET, IRQ)

Interruptions.

- *Gestionnaire*: sous-routine qui traite une interruption
- *Table d’interruptions*: contient l’adresse des gestionnaires
- *Traitement*: sauvegarder l’état du processeur; appeler le gestionnaire; restaurer l’état
- *Priorité*: valeur numérique assignée à une interruption
- *Gestion des priorités*: interruption ignorée si une interruption de priorité > est en cours; gestionnaire en exécution mis en attente si une interruption de priorité ≥ est lancée
- *Non masquable*: top priorité, ne peut pas ignorer (ex. RESET)

Accès direct à la mémoire (DMA).

- *DMA*: permet au processeur d’initier un accès mémoire et de laisser un contrôleur effectuer le transfert de données
- *Sur le NES*: envoi des tuiles mem[0x0200, 0x02FF] vers la mémoire de *sprites* via DMA:

```
lda #$02
sta $4014
```

Appels système.

- *Accès E/S*: empêché par le système d’exploitation (sécurité)
- *Appel système*: service offert par le noyau du système d’exploitation; appelé via une interruption logicielle
- *Exemples UNIX + ARMv8*:

code	appel système
64	write(flux, chaine, #octets)
63	read(flux, tampon, #octets)

Flux d’entrée standard = 0
Flux de sortie standard = 1

```
// Afficher chaine
mov x8, 64
mov x0, 1
adr x1, chaine
mov x2, 10
svc 0
```

Annexe B:

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 $mem_b[a]$ réfère aux b octets à l'adresse a de la mémoire principale:

Code d'op.	Syntaxe	Effet	Exemple
mov	mov rd, rm mov rd, i	$r_d \leftarrow r_m$ $r_d \leftarrow i$	mov x19, x21 mov x19, 42
ldr	ldr xd, a ldr wd, a	charge 8 octets: $x_d \langle 63, 0 \rangle \leftarrow mem_8[a]$ charge 4 octets: $x_d \langle 31, 0 \rangle \leftarrow mem_4[a]$; $x_d \langle 63, 32 \rangle \leftarrow 0$	ldr x19, [x20] ldr w19, [x20]
ldrh	ldrh wd, a	charge 2 octets: $x_d \langle 15, 0 \rangle \leftarrow mem_2[a]$; $x_d \langle 63, 16 \rangle \leftarrow 0$	ldrh w19, [x20]
ldrb	ldrb wd, a	charge 1 octet: $x_d \langle 7, 0 \rangle \leftarrow mem_1[a]$; $x_d \langle 63, 8 \rangle \leftarrow 0$	ldrb w19, [x20]
str	str xd, a str wd, a	stocke 8 octets: $mem_8[a] \leftarrow x_d \langle 63, 0 \rangle$ stocke 4 octets: $mem_4[a] \leftarrow x_d \langle 31, 0 \rangle$	str x19, [x20] str w19, [x20]
strh	strh wd, a	stocke 2 octets: $mem_2[a] \leftarrow x_d \langle 15, 0 \rangle$	strh w19, [x20]
strb	strb wd, a	stocke 1 octet: $mem_1[a] \leftarrow x_d \langle 7, 0 \rangle$	strb w19, [x20]
ldp	ldp xd, xn, a	charge 16 octets: $x_d \langle 63, 0 \rangle \leftarrow mem_8[a]$, $x_n \langle 63, 0 \rangle \leftarrow mem_8[a+8]$	ldp x19, x20, [sp]
stp	stp xd, xn, a	stocke 16 octets: $mem_8[a] \leftarrow x_d \langle 63, 0 \rangle$, $mem_8[a+8] \leftarrow x_n \langle 63, 0 \rangle$	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	$\neq$	$\neg Z$
hs	$\geq$	C
hi	>	$C \wedge \neg Z$
ls	$\leq$	$\neg C \vee Z$
lo	<	$\neg C$

Entiers signés

Code	Signification	Codes de condition
eq	=	Z
ne	$\neq$	$\neg Z$
ge	$\geq$	$N = V$
gt	>	$\neg Z \wedge (N = V)$
le	$\leq$	$Z \vee (N \neq V)$
lt	<	$N \neq V$
vs	débordement	V
vc	pas de débordement	$\neg V$
mi	négatif	N
pl	non négatif	$\neg N$

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 $x_{30} \leftarrow pc + 4$	bl printf
blr	blr xd	branche à x_d et $x_{30} \leftarrow pc + 4$	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 <i>cond</i> : $r_d \leftarrow r_n$, sinon: $r_d \leftarrow r_m$	csel x19, x20, x21, eq

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: $x_d \langle 63, j \rangle \leftarrow x_n \langle 63 - j, 0 \rangle$; $x_d \langle j - 1, 0 \rangle \leftarrow 0$	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

Registres (nombres en virgule flottante).

- Possède 32 registres double précision (64 bits) de la forme d_n
- Chaque registre d_n possède un sous-registre simple précision (32 bits) s_n
- v_n réfère au registre d_n ou s_n
- Conventions:

Registres	Utilisation
$d_0 - d_7$	registres d'arguments et de retour de sous-programmes
$d_8 - d_{15}$	registres sauvegardés par l'appelé
$d_{16} - d_{31}$	registres sauvegardés par l'appelant

Manipulation et arithmétique (nombres en virgule flottante).

- Les conditions de branchement sont les mêmes que pour les entiers et sont déterminées à partir de codes de condition mis à jour par **fcmp**

Code d'op.	Syntaxe	Effet	Exemple
ldr	ldr d_n, a	charge un nombre en virgule flottante double précision de l'adresse a vers d_n (8 octets)	ldr $d8, [x19]$
	ldr s_n, a	charge un nombre en virgule flottante simple précision de l'adresse a vers s_n (4 octets)	ldr $s8, [x19]$
str	str d_n, a	stocke un nombre en virgule flottante double précision de d_n vers l'adresse a (8 octets)	str $d8, [x19]$
	str s_n, a	stocke un nombre en virgule flottante simple précision de s_n vers l'adresse a (4 octets)	str $s8, [x19]$
fmov	fmov vd, vm	$V_d \leftarrow V_m$	fmov $d8, d9$
	fmov vd, i	$V_d \leftarrow i$	fmov $d8, 1.5$
fcmp	fcmp vd, vm	compare v_d et v_m	fcmp $d8, d9$
	fcmp vd, i	compare v_d et i	fcmp $d8, 0.0$
fadd	fadd vd, vn, vm	$V_d \leftarrow V_n + V_m$	fadd $d8, d9, d10$
fsub	fsub vd, vn, vm	$V_d \leftarrow V_n - V_m$	fsub $d8, d9, d10$
fmul	fmul vd, vn, vm	$V_d \leftarrow V_n \cdot V_m$	fmul $d8, d9, d10$
fdiv	fdiv vd, vn, vm	$V_d \leftarrow V_n / V_m$	fdiv $d8, d9, d10$
fsqrt	fsqrt vd, vn	$V_d \leftarrow \sqrt{V_n}$	fsqrt $d8, d9$
fabs	fabs vd, vn	$V_d \leftarrow V_n $	fabs $d8, d9$
ucvtf	ucvtf vd, rn	convertit l'entier non signé dans r_n vers un nombre en virgule flottante dans v_d (selon le mode d'approximation configuré dans le registre de contrôle FPCR)	ucvtf $d8, x19$ ucvtf $d8, w19$ ucvtf $s8, x19$ ucvtf $s8, w19$
scvtf	scvtf vd, rn	convertit l'entier signé dans r_n vers un nombre en virgule flottante dans v_d (selon le mode d'approximation configuré dans le registre de contrôle FPCR)	scvtf $d8, x19$ scvtf $d8, w19$ scvtf $s8, x19$ scvtf $s8, w19$
fcvt	fcvt vd, vn	convertit le nombre en virgule flottante dans v_n vers un nombre en virgule flottante d'une autre précision dans v_d	fcvt $d8, s9$

Appels système.

- x_8 : code numérique du service
- x_0 à x_5 : arguments
- **svc 0**: appel du service

Données statiques.

Segments de données		Données	
Pseudo-instruction	Contenu		
.section ".text"	instructions	.align k	donnée suivante stockée à une adresse divisible par k
.section ".rodata"	données en lecture seule	.skip k	réserve k octets
.section ".data"	données initialisées	.ascii s	chaîne de caractères initialisée à s
.section ".bss"	données non-initialisées	.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
	%c	caractère (1 octet)
Caractères	%s	chaîne de caractères

Annexe C:

Sommaire de l'architecture du NES

Registres.

- Possède 4 registres d'un octet
- Registre interne: p (*registre d'état*), contient des états et codes de conditions dont *report/emprunt* (1 octet)
- Registre interne: pc (*compteur d'instruction*), contient l'adresse de la prochaine instruction (2 octets)

Nom	Utilisation principale
a	accumulateur, utilisé comme opérande et valeur de retour des opérations arithmétiques et logiques
x	utilisé comme compteur ou comme index pour l'adressage indexé
y	utilisé comme compteur ou comme index pour l'adressage indexé
s	pointeur de pile (pointe vers $0100_{16} + s$)

Valeurs immédiates.

- #: valeur numérique, sans #: adresse
- \$: valeur hexadécimale
- %: valeur binaire
- Exemples:

expression	valeur
#5	5_{10}
#\$FF	FF_{16}
##00010011	00010011_2
\$FF	adresse FF_{16}

Modes d'adressage.

Nom.	Syntaxe	Adresse	Exemple
absolu	i	i	<code>lda \$D010</code>
indexé par x	i, x eti _q , x	$i + x$ $eti_q + x$	<code>lda \$D010, x</code> <code>lda tab, x</code>
indexé par y	i, y eti _q , y	$i + y$ $eti_q + y$	<code>lda \$D010, y</code> <code>lda tab, y</code>

Accès mémoire.

- Instructions, où $mem_1[a]$ dénote l'octet situé à l'adresse a de la mémoire principale:

Code d'op.	Syntaxe	Effet	Exemple
<code>lda</code>	<code>lda #i</code> <code>lda adr</code>	$a \leftarrow i$ $a \leftarrow mem_1[adr]$	<code>lda #42</code> <code>lda var</code>
<code>ldx</code>	<code>ldx #i</code> <code>ldx adr</code>	$x \leftarrow i$ $x \leftarrow mem_1[adr]$	<code>ldx #42</code> <code>ldx var</code>
<code>ldy</code>	<code>ldy #i</code> <code>ldy adr</code>	$y \leftarrow i$ $y \leftarrow mem_1[adr]$	<code>ldy #42</code> <code>ldy var</code>
<code>sta</code>	<code>sta adr</code>	$mem_1[adr] \leftarrow a$	<code>sta var</code>
<code>stx</code>	<code>stx adr</code>	$mem_1[adr] \leftarrow x$	<code>stx var</code>
<code>sty</code>	<code>sty adr</code>	$mem_1[adr] \leftarrow y$	<code>sty var</code>
<code>txa</code>	<code>txa</code>	$a \leftarrow x$	<code>txa</code>
<code>tax</code>	<code>tax</code>	$x \leftarrow a$	<code>tax</code>
<code>tya</code>	<code>tya</code>	$a \leftarrow y$	<code>tya</code>
<code>tay</code>	<code>tay</code>	$y \leftarrow a$	<code>tay</code>
<code>txs</code>	<code>txs</code>	$s \leftarrow x$	<code>txs</code>
<code>tsx</code>	<code>tsx</code>	$x \leftarrow s$	<code>tsx</code>
<code>pha</code>	<code>pha</code>	empile a sur la pile	<code>pha</code>
<code>pla</code>	<code>pla</code>	dépile le premier octet de la pile vers a	<code>pla</code>

Arithmétique.

Code d'op.	Syntaxe	Effet	Exemple
adc	adc #i adc adr	$a \leftarrow a + i + report$ $a \leftarrow a + mem_1[adr] + report$	lda #1 adc var
sbc	sbc #i sbc adr	$a \leftarrow a - i - emprunt$ $a \leftarrow a - mem_1[adr] - emprunt$	sbc #1 sbc var
clc	clc	$report \leftarrow 0$ (utile avant adc)	clc
sec	sec	$emprunt \leftarrow 0$ (utile avant sbc)	sec
inx	inx	$x \leftarrow x + 1$	inx
iny	iny	$y \leftarrow y + 1$	iny
inc	inc adr	$mem_1[adr] \leftarrow mem_1[adr] + 1$	inc var
dec	dec adr	$mem_1[adr] \leftarrow mem_1[adr] - 1$	dec var

Logique.

Code d'op.	Syntaxe	Effet	Exemple
asl	asl adr	décalage logique de $mem_1[adr]$ d'un bit à gauche (directement en mémoire)	asl var
lsr	lsr adr	décalage logique de $mem_1[adr]$ d'un bit à droite (directement en mémoire)	lsr var
and	and #i and adr	$a \leftarrow a \wedge i$ $a \leftarrow a \wedge mem_1[adr]$	and #%00100011 and var
ora	ora #i ora adr	$a \leftarrow a \vee i$ $a \leftarrow a \vee mem_1[adr]$	ora #%00100011 ora var
eor	eor #i eor adr	$a \leftarrow a \oplus i$ $a \leftarrow a \oplus mem_1[adr]$	eor #%00100011 eor var

Comparaisons et branchements.

Code d'op.	Syntaxe	Effet	Exemple
cmp	cmp #i cmp adr	compare a et i compare a et $mem_1[adr]$	cmp #0 cmp var
cpx	cpx #i cpx adr	compare x et i compare x et $mem_1[adr]$	cpx #0 cpx var
cpy	cpy #i cpy adr	compare y et i compare y et $mem_1[adr]$	cpy #0 cpy var
beq	beq etiq	branche à etiq : si =	beq boucle
bne	bne etiq	branche à etiq : si $\neq$	bne boucle
jmp	jmp etiq	branche à etiq :	jmp boucle
jsr	jsr etiq	branche au sous-programme etiq : et empile l'adresse de retour	jsr func
rts	rts	branche à l'adresse de retour d'un sous-programme	rts
rti	rti	branche à l'adresse de retour d'une interruption	rti