

IFT209 – Programmation système  
Université de Sherbrooke

## Examen final

Enseignant: Michael Blondin  
Date: vendredi 19 avril 2024  
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 **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 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)

(i)

3 pts

(ii)

2 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)

7 pts

(b)

3 pts

**Question 4: nombres en virgule flottante**

(a)

5 pts

(b)

(i)

2,5 pts

(ii)

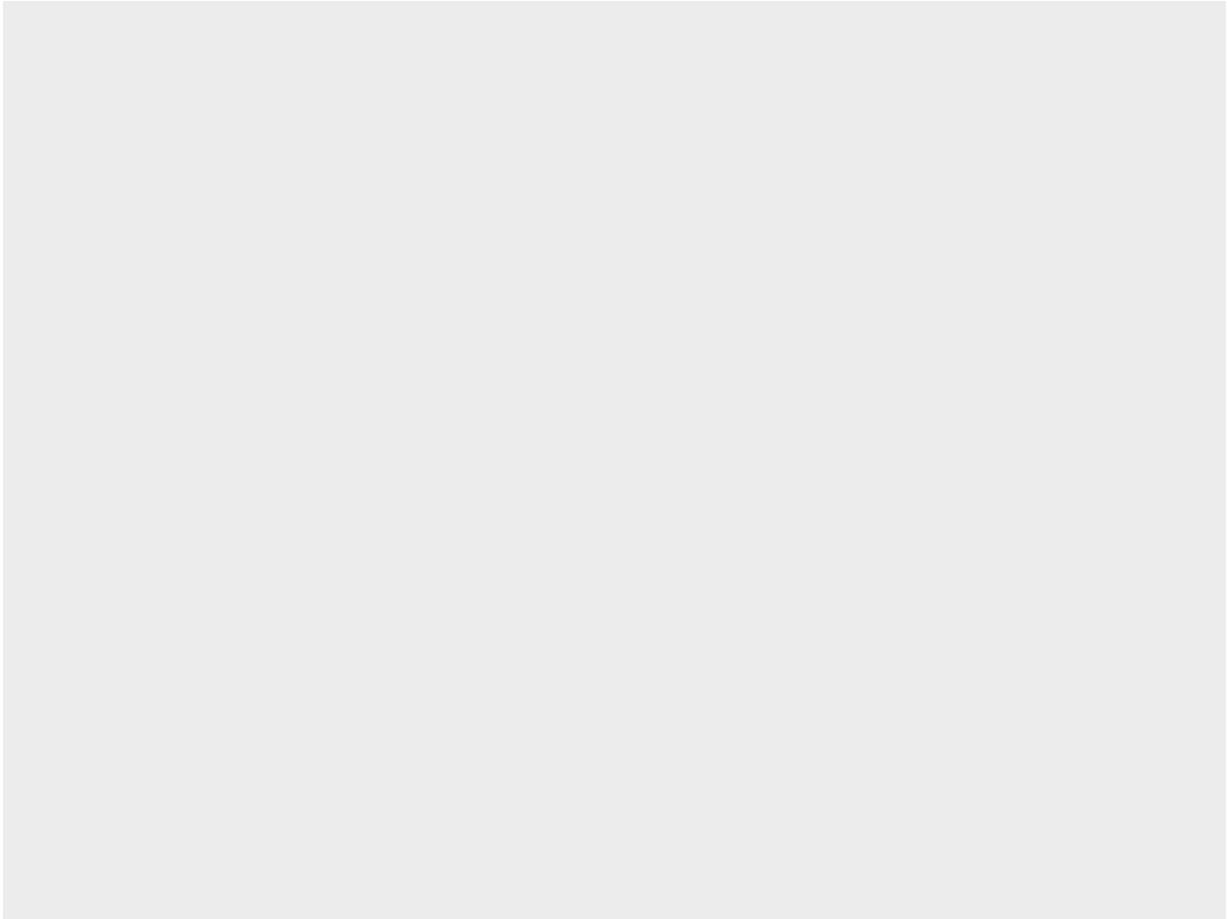
2,5 pts

**Question 5: entrées/sorties**

(a)

6 pts

(b)



4 pts

## **Annexe A:**

### Fiches récapitulatives



## 12. Nombres en virgule flottante

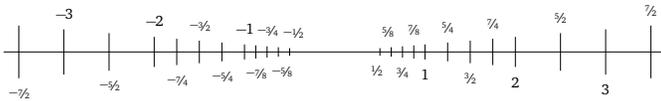
### Représentation.

- ▶ *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:



### Arithmétique.

- ▶ *Addition:* (1) mettre exposants en commun; (2) additionner mantisses; (3) normaliser; (4) arrondir
- ▶ *Multiplication:* (1) additionner exposants; (2) multiplier mantisses; (3) normaliser; (4) arrondir

### Précision.

- ▶ *Approximations de nombres réels:*
  - (a) arrondir (égalité: dernier chiffre pair): 1,9565 → 1,956
  - (b) troncation: 1,5416 → 1,541
- ▶ *Erreur relative:*  $\text{err}(x) := \frac{x-\bar{x}}{x}$  où  $\bar{x}$  est l'approximation
- ▶ *Borne pour mode (a):*  $|\text{err}(x)| \leq \underbrace{(\beta/2) \cdot \beta^{-n}}_{\varepsilon \text{ machine}}$

### Norme IEEE 754.

format	signe	exposant	mantisse
simple	1 bit	8 bits	23 bits (+1 bit caché)
double	1 bit	11 bits	52 bits (+1 bit caché)

- ▶ *Repr. avec biais:* 1000011011100...0 = -1,11 × 2<sup>13-127</sup>
- ▶ ±0 (s0...00...0); ±∞ (s1...10...00); NaN (s1...1e0...01)

### ARMv8.

- ▶ *Registres:* d<sub>n</sub> (64 bits) et s<sub>n</sub> (32 bits)
- ▶ *Instructions:* **ldr**, **str**, **fmov**, **fcmp**, **fadd**, **fmul**, **fsqrt**, etc.

## 13. Introduction aux entrées/sorties : NES

### Architecture.

- ▶ *Pas RISC:* possible de manipuler la mémoire directement
- ▶ *Processeurs:* proc. principal + proc. d'images (PPU)
- ▶ *Mémoire principale:* primaire + registres d'E/S + programme
- ▶ *Mémoire vidéo:* stocke les tuiles et palettes de couleurs

### Jeu d'instructions.

- ▶ *Registres:* a (accumulateur), x (index), y (index), s (pile)
- ▶ *Valeurs imm.:* # (numérique), \$ (hexadécimal), % (binaire)
- ▶ *Accès mémoire:* **lda**, **ldx**, **ldy** (chargement d'octet); **sta**, **stx**, **sty** (stockage d'octet); **txa**, **tax**, **tya**, etc. (copie)
- ▶ *Arithmétique:* **adc** (addition avec report); **sbc** (soustraction avec emprunt); **inc**, **inx**, **iny**, **dec** (inc/décrémentation)
- ▶ *Logique:* **asl** (<< 1), **lsr** (>> 1), **and** (^), **ora** (v), **eor** (⊕)
- ▶ *Contrôle:* **cmp**, **cpx**, **cpy** (comparaison); **beq**, **bne** (branch. conditionnel), **jmp** (branch. incond.), **jsr**/**rts** (sous-prog.)

### Tuiles.

- ▶ *Images:* constituées de tuiles de 8 × 8 pixels
- ▶ *Tuiles:* stockées dans la cartouche, transférées vers le PPU
- ▶ *Tuile:* spécifiée par 4 octets (y, i, a, x): position verticale y, numéro de tuile i, attributs a, position horizontale x
- ▶ *Attributs:* 8 bits pour réflexions, profondeur et couleurs

### Sorties (graphiques).

- ▶ L'affichage se fait lors du rafraîchissement vertical
- ▶ *Sortie:* stocker tuiles de 0X00<sub>16</sub> à 0XFF<sub>16</sub> en mém. principale
- ▶ *Affichage:* transférer au PPU en écrivant #\$0X à \$4014

### Entrées (manettes).

- ▶ *Entrée:* protocole de communication via port de manettes
- ▶ *Demande de lecture:* envoyer #1, puis #0, via \$4016
- ▶ *Lecture:* lire bit de poids faible à \$4016 pour chaque bouton

## 14. Entrées/sorties

### Mécanismes d'entrée/sortie.

- ▶ *Attente active:* interrogation continue d'un registre d'état jusqu'à un événement (ex. VBLANK)
- ▶ *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)	<b>mov</b> x8, 64
63	read(flux, tampon, #octets)	<b>mov</b> x0, 1
		<b>adr</b> x1, chaine
		<b>mov</b> x2, 10
		<b>svc</b> 0
	Flux d'entrée standard = 0	
	Flux de sortie standard = 1	

## **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 <sub>0</sub> - ip <sub>1</sub>	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
<b>cmp</b>	<b>cmp</b> rd, rm	compare $r_d$ et $r_m$	<b>cmp</b> x19, x21
	<b>cmp</b> rd, i	compare $r_d$ et $i$	<b>cmp</b> x19, 42
	<b>cmp</b> rd, rm, decal j	compare $r_d$ et $r_m$ <i>decal j</i>	<b>cmp</b> x19, x21, <b>lsl</b> 1
<b>add</b>	<b>add</b> rd, rn, rm	$r_d \leftarrow r_n + r_m$	<b>add</b> x19, x20, x21
	<b>add</b> rd, rn, i	$r_d \leftarrow r_n + i$	<b>add</b> x19, x20, 42
	<b>add</b> rd, rn, rm, decal j	$r_d \leftarrow r_n + (r_m \text{ decal } j)$	<b>add</b> x19, x20, x21, <b>lsl</b> 1
<b>adc</b>	<b>adc</b> rd, rn, rm	$r_d \leftarrow r_n + r_m + C$	<b>adc</b> x19, x20, x21
<b>sub</b>	<b>sub</b> rd, rn, rm	$r_d \leftarrow r_n - r_m$	<b>sub</b> x19, x20, x21
	<b>sub</b> rd, rn, i	$r_d \leftarrow r_n - i$	<b>sub</b> x19, x20, 42
	<b>sub</b> rd, rn, rm, decal j	$r_d \leftarrow r_n - (r_m \text{ decal } j)$	<b>sub</b> x19, x20, x21, <b>lsl</b> 1
<b>sbc</b>	<b>sbc</b> rd, rn, rm	$r_d \leftarrow r_n - r_m - 1 + C$	<b>sbc</b> x19, x20, x21
<b>neg</b>	<b>neg</b> rd, rm	$r_d \leftarrow -r_m$	<b>neg</b> x19, x21
	<b>neg</b> rd, rm, decal j	$r_d \leftarrow -(r_m \text{ decal } j)$	<b>neg</b> x19, x21, <b>lsl</b> 1
<b>mul</b>	<b>mul</b> rd, rn, rm	$r_d \leftarrow r_n \cdot r_m$	<b>mul</b> x19, x20, x21
<b>udiv</b>	<b>udiv</b> rd, rn, rm	$r_d \leftarrow r_n \div r_m$ (non signé)	<b>udiv</b> x19, x20, x21
<b>sdiv</b>	<b>sdiv</b> rd, rn, rm	$r_d \leftarrow r_n \div r_m$ (signé)	<b>sdiv</b> x19, x20, x21
<b>madd</b>	<b>madd</b> rd, rn, rm, ra	$r_d \leftarrow r_a + (r_n \cdot r_m)$	<b>madd</b> x19, x20, x21, x22
<b>msub</b>	<b>msub</b> rd, rn, rm, ra	$r_d \leftarrow r_a - (r_n \cdot r_m)$	<b>msub</b> 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
<b>mov</b>	<b>mov</b> rd, rm <b>mov</b> rd, i	$r_d \leftarrow r_m$ $r_d \leftarrow i$	<b>mov</b> x19, x21 <b>mov</b> x19, 42
<b>ldr</b>	<b>ldr</b> xd, a <b>ldr</b> wd, a	charge 8 octets: $x_d \langle 63, 0 \rangle \leftarrow mem_8[a]$ charge 4 octets: $x_d \langle 31, 0 \rangle \leftarrow mem_4[a]$ ; $x_d \langle 63, 32 \rangle \leftarrow 0$	<b>ldr</b> x19, [x20] <b>ldr</b> w19, [x20]
<b>ldrh</b>	<b>ldrh</b> wd, a	charge 2 octets: $x_d \langle 15, 0 \rangle \leftarrow mem_2[a]$ ; $x_d \langle 63, 16 \rangle \leftarrow 0$	<b>ldrh</b> w19, [x20]
<b>ldrb</b>	<b>ldrb</b> wd, a	charge 1 octet: $x_d \langle 7, 0 \rangle \leftarrow mem_1[a]$ ; $x_d \langle 63, 8 \rangle \leftarrow 0$	<b>ldrb</b> w19, [x20]
<b>str</b>	<b>str</b> xd, a <b>str</b> wd, a	stocke 8 octets: $mem_8[a] \leftarrow x_d \langle 63, 0 \rangle$ stocke 4 octets: $mem_4[a] \leftarrow x_d \langle 31, 0 \rangle$	<b>str</b> x19, [x20] <b>str</b> w19, [x20]
<b>strh</b>	<b>strh</b> wd, a	stocke 2 octets: $mem_2[a] \leftarrow x_d \langle 15, 0 \rangle$	<b>str</b> w19, [x20]
<b>strb</b>	<b>strb</b> wd, a	stocke 1 octet: $mem_1[a] \leftarrow x_d \langle 7, 0 \rangle$	<b>strb</b> w19, [x20]
<b>ldp</b>	<b>ldp</b> xd, xn, a	charge 16 octets: $x_d \langle 63, 0 \rangle \leftarrow mem_8[a]$ , $x_n \langle 63, 0 \rangle \leftarrow mem_8[a + 8]$	<b>ldp</b> x19, x20, [sp]
<b>stp</b>	<b>stp</b> xd, xn, a	stocke 16 octets: $mem_8[a] \leftarrow x_d \langle 63, 0 \rangle$ , $mem_8[a + 8] \leftarrow x_n \langle 63, 0 \rangle$	<b>stp</b> 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			Entiers signés		
Code	Signification	Codes de condition	Code	Signification	Codes de condition
<b>eq</b>	=	Z	<b>eq</b>	=	Z
<b>ne</b>	≠	¬Z	<b>ne</b>	≠	¬Z
<b>hs</b>	≥	C	<b>ge</b>	≥	N = V
<b>hi</b>	>	C ∧ ¬Z	<b>gt</b>	>	¬Z ∧ (N = V)
<b>ls</b>	≤	¬C ∨ Z	<b>le</b>	≤	Z ∨ (N ≠ V)
<b>lo</b>	<	¬C	<b>lt</b>	<	N ≠ V
			<b>vs</b>	débordement	V
			<b>vc</b>	pas de débordement	¬V
			<b>mi</b>	négatif	N
			<b>pl</b>	non négatif	¬N

## Branchement.

- Instructions de branchement, où  $j$  est une valeur immédiate de 6 bits:

Code d'op.	Syntaxe	Effet	Exemple
<b>b.</b>	<b>b.cond</b> etiq	branche à <b>etiq</b> : si <i>cond</i>	<b>b.eq</b> main100
<b>b</b>	<b>b</b> etiq	branche à <b>etiq</b> :	<b>b</b> main100
<b>cbz</b>	<b>cbz</b> rd, etiq	branche à <b>etiq</b> : si $r_d = 0$	<b>cbz</b> x19 main100
<b>cbnz</b>	<b>cbnz</b> rd, etiq	branche à <b>etiq</b> : si $r_d \neq 0$	<b>cbnz</b> x19 main100
<b>tbz</b>	<b>tbz</b> rd, j, etiq	branche à <b>etiq</b> : si $r_d \langle j \rangle = 0$	<b>tbz</b> x19, 1, main100
<b>tbnz</b>	<b>tbnz</b> rd, j, etiq	branche à <b>etiq</b> : si $r_d \langle j \rangle \neq 0$	<b>tbnz</b> x19, 1, main100
<b>bl</b>	<b>bl</b> etiq	branche à <b>etiq</b> : et $x_{30} \leftarrow pc + 4$	<b>bl</b> printf
<b>blr</b>	<b>blr</b> xd	branche à $x_d$ et $x_{30} \leftarrow pc + 4$	<b>blr</b> x20
<b>br</b>	<b>br</b> xd	branche à $x_d$	<b>br</b> x20
<b>ret</b>	<b>ret</b>	branche à $x_{30}$ (retour de sous-prog.)	<b>ret</b>

## Adressage.

- Modes d'adressages, où  $k$  est une valeur immédiate de 7 bits:

Nom	Syntaxe	Adresse	Effet	Exemple
adresse d'une étiquette	<b>adr</b> xd, etiq	—	$x_d \leftarrow$ adresse de <b>etiq</b> :	<b>adr</b> 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, <b>lsl</b> 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
<b>csel</b>	<b>csel</b> rd, rn, rm, cond	si <i>cond</i> : $r_d \leftarrow r_n$ , sinon: $r_d \leftarrow r_m$	<b>csel</b> x19, x20, x21, <b>eq</b>

## 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
<b>mvn</b>	<b>mvn</b> rd, rn	$r_d \leftarrow \neg r_n$	<b>mvn</b> x19, x20
<b>and</b>	<b>and</b> rd, rn, rm	$r_d \leftarrow r_n \wedge r_m$	<b>and</b> x19, x20, x21
	<b>and</b> rd, rn, i	$r_d \leftarrow r_n \wedge i$	<b>and</b> x19, x20, 4
	<b>and</b> rd, rn, rm, decal j	$r_d \leftarrow r_n \wedge (r_m \text{ decal } j)$	<b>and</b> x19, x20, x21, <b>lsl</b> 1
<b>orr</b>	<b>orr</b> rd, rn, rm	$r_d \leftarrow r_n \vee r_m$	<b>orr</b> x19, x20, x21
	<b>orr</b> rd, rn, i	$r_d \leftarrow r_n \vee i$	<b>orr</b> x19, x20, 4
	<b>orr</b> rd, rn, rm, decal j	$r_d \leftarrow r_n \vee (r_m \text{ decal } j)$	<b>orr</b> x19, x20, x21, <b>lsl</b> 1
<b>eor</b>	<b>eor</b> rd, rn, rm	$r_d \leftarrow r_n \oplus r_m$	<b>eor</b> x19, x20, x21
	<b>eor</b> rd, rn, i	$r_d \leftarrow r_n \oplus i$	<b>eor</b> x19, x20, 4
	<b>eor</b> rd, rn, rm, decal j	$r_d \leftarrow r_n \oplus (r_m \text{ decal } j)$	<b>eor</b> x19, x20, x21, <b>lsl</b> 1
<b>bic</b>	<b>bic</b> rd, rn, rm	$r_d \leftarrow r_n \wedge \neg r_m$	<b>bic</b> x19, x20, x21
	<b>bic</b> rd, rn, i	$r_d \leftarrow r_n \wedge \neg i$	<b>bic</b> x19, x20, 4
	<b>bic</b> rd, rn, rm, decal j	$r_d \leftarrow r_n \wedge \neg (r_m \text{ decal } j)$	<b>bic</b> x19, x20, x21, <b>lsl</b> 1
<b>lsl</b>	<b>lsl</b> xd, xn, j	décalage de $j$ bits vers la gauche: $x_d \langle 63, j \rangle \leftarrow x_n \langle 63 - j, 0 \rangle$ ; $x_d \langle j - 1, 0 \rangle \leftarrow 0$	<b>lsl</b> x19, x20, 1
<b>lsr</b>	<b>lsr</b> xd, xn, j	décalage de $j$ bits vers la droite: $x_d \langle 63 - j, 0 \rangle \leftarrow x_n \langle 63, j \rangle$ ; $x_d \langle 63, 64 - j \rangle \leftarrow 0$	<b>lsr</b> x19, x20, 1
<b>asr</b>	<b>asr</b> xd, xn, j	décalage arithmétique de $j$ bits vers la droite: $x_d \langle 63 - j, 0 \rangle \leftarrow x_n \langle 63, j \rangle$ ; $x_d \langle 63, 64 - j \rangle \leftarrow x_n \langle 63 \rangle$	<b>asr</b> x19, x20, 1
<b>ror</b>	<b>ror</b> xd, xn, j	décalage circulaire de $j$ bits vers la droite: $x_d \leftarrow x_n \langle j - 1, 0 \rangle x_n \langle 63, j \rangle$	<b>ror</b> 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
<b>ldr</b>	<b>ldr</b> $d_n, a$	charge un nombre en virgule flottante double précision de l'adresse $a$ vers $d_n$ (8 octets)	<b>ldr</b> $d8, [x19]$
	<b>ldr</b> $s_n, a$	charge un nombre en virgule flottante simple précision de l'adresse $a$ vers $s_n$ (4 octets)	<b>ldr</b> $s8, [x19]$
<b>str</b>	<b>str</b> $d_n, a$	stocke un nombre en virgule flottante double précision de $d_n$ vers l'adresse $a$ (8 octets)	<b>str</b> $d8, [x19]$
	<b>str</b> $s_n, a$	stocke un nombre en virgule flottante simple précision de $s_n$ vers l'adresse $a$ (4 octets)	<b>str</b> $s8, [x19]$
<b>fmov</b>	<b>fmov</b> $v_d, v_m$	$v_d \leftarrow v_m$	<b>fmov</b> $d8, d9$
	<b>fmov</b> $v_d, i$	$v_d \leftarrow i$	<b>fmov</b> $d8, 1.5$
<b>fcmp</b>	<b>fcmp</b> $v_d, v_m$	compare $v_d$ et $v_m$	<b>fcmp</b> $d8, d9$
	<b>fcmp</b> $v_d, i$	compare $v_d$ et $i$	<b>fcmp</b> $d8, 0.0$
<b>fadd</b>	<b>fadd</b> $v_d, v_n, v_m$	$v_d \leftarrow v_n + v_m$	<b>fadd</b> $d8, d9, d10$
<b>fsub</b>	<b>fsub</b> $v_d, v_n, v_m$	$v_d \leftarrow v_n - v_m$	<b>fsub</b> $d8, d9, d10$
<b>fmul</b>	<b>fmul</b> $v_d, v_n, v_m$	$v_d \leftarrow v_n \cdot v_m$	<b>fmul</b> $d8, d9, d10$
<b>fdiv</b>	<b>fdiv</b> $v_d, v_n, v_m$	$v_d \leftarrow v_n / v_m$	<b>fdiv</b> $d8, d9, d10$
<b>fsqrt</b>	<b>fsqrt</b> $v_d, v_n$	$v_d \leftarrow \sqrt{v_n}$	<b>fsqrt</b> $d8, d9$
<b>fabs</b>	<b>fabs</b> $v_d, v_n$	$v_d \leftarrow  v_n $	<b>fabs</b> $d8, d9$
<b>ucvtf</b>	<b>ucvtf</b> $v_d, r_n$	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)	<b>ucvtf</b> $d8, x19$ <b>ucvtf</b> $d8, w19$ <b>ucvtf</b> $s8, x19$ <b>ucvtf</b> $s8, w19$
<b>scvtf</b>	<b>scvtf</b> $v_d, r_n$	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)	<b>scvtf</b> $d8, x19$ <b>scvtf</b> $d8, w19$ <b>scvtf</b> $s8, x19$ <b>scvtf</b> $s8, w19$
<b>fcvt</b>	<b>fcvt</b> $v_d, v_n$	convertit le nombre en virgule flottante dans $v_n$ vers un nombre en virgule flottante d'une autre précision dans $v_d$	<b>fcvt</b> $d8, s9$

## 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		
<code>.section ".text"</code>	instructions	<code>.align</code> $k$	donnée suivante stockée à une adresse divisible par $k$
<code>.section ".rodata"</code>	données en lecture seule	<code>.skip</code> $k$	réserve $k$ octets
<code>.section ".data"</code>	données initialisées	<code>.ascii</code> $s$	chaîne de caractères initialisée à $s$
<code>.section ".bss"</code>	données non-initialisées	<code>.asciz</code> $s$	chaîne de caractères initialisée à $s$ suivi du carac. nul
		<code>.byte</code> $v$	octet initialisé à $v$
		<code>.hword</code> $v$	demi-mot initialisé à $v$
		<code>.word</code> $v$	mot initialisé à $v$
		<code>.xword</code> $v$	double mot initialisé à $v$
		<code>.single</code> $f$	nombre en virg. flottante simple précision initialisé à $f$
		<code>.double</code> $f$	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	<code>%ld</code>	entier décimal signé
	<code>%lu</code>	entier décimal non signé
	<code>%lX</code>	entier hexadécimal non signé
	<code>%lf</code>	nombre en virgule flottante
Nombres sur 32 bits	<code>%d</code>	entier décimal signé
	<code>%u</code>	entier décimal non signé
	<code>%X</code>	entier hexadécimal non signé
Nombres sur 16 bits	<code>%f</code>	nombre en virgule flottante
	<code>%hd</code>	entier décimal signé
	<code>%hu</code>	entier décimal non signé
Caractères	<code>%hX</code>	entier hexadécimal non signé
	<code>%c</code>	caractère (1 octet)
	<code>%s</code>	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</i>	<i>i</i>	<code>lda \$D010</code>
indexé par x	<i>i, x</i> <i>eti, x</i>	$i + x$ $eti + x$	<code>lda \$D010, x</code> <code>lda tab, x</code>
indexé par y	<i>i, y</i> <i>eti, y</i>	$i + y$ $eti + y$	<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	$a \leftarrow a + i + report$	lda #1
	adc adr	$a \leftarrow a + mem_1[adr] + report$	adc var
sbc	sbc #i	$a \leftarrow a - i - emprunt$	sbc #1
	sbc adr	$a \leftarrow a - mem_1[adr] - emprunt$	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	$a \leftarrow a \wedge i$	and #%00100011
	and adr	$a \leftarrow a \wedge mem_1[adr]$	and var
ora	ora #i	$a \leftarrow a \vee i$	ora #%00100011
	ora adr	$a \leftarrow a \vee mem_1[adr]$	ora var
eor	eor #i	$a \leftarrow a \oplus i$	eor #%00100011
	eor adr	$a \leftarrow a \oplus mem_1[adr]$	eor var

## Comparaisons et branchements.

Code d'op.	Syntaxe	Effet	Exemple
cmp	cmp #i	compare a et $i$	cmp #0
	cmp adr	compare a et $mem_1[adr]$	cmp var
cpx	cpx #i	compare x et $i$	cpx #0
	cpx adr	compare x et $mem_1[adr]$	cpx var
cpy	cpy #i	compare y et $i$	cpy #0
	cpy adr	compare y et $mem_1[adr]$	cpy var
beq	beq etiq	branche à <b>etiq:</b> si =	beq boucle
bne	bne etiq	branche à <b>etiq:</b> si $\neq$	bne boucle
jmp	jmp etiq	branche à <b>etiq:</b>	jmp boucle
jsr	jsr etiq	branche au sous-programme <b>etiq:</b> 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