Travail pratique #1
INF1600 - Architecture du processeur

Trimestre : automne 2018

Équipier1: Nanor Janjikian (1901777)
Équipier2 : Yasmina Abou-Nakkoul (1897266)

Équipe: 11 (groupe B1)

Presenté à :
Ulrich Dah-Achinanon

École polytechnique de Montréal
8 octobre 2018

Exercice 1 : Révision de logique et arithmétique numérique

1. Les valeurs décimales pour les nombres entiers signés suivants :

Pour convertir les nombres suivants en décimal, nous devons nous rappeler que le complément à deux consiste à renverser le nombre bit par bit et à ajouter un 1. De plus, nous devons vérifier le signe de chaque nombre.

a. 1100 1101 (binaire) :

- Tout d'abord, nous renversons les bits : $(00110010)_2$
- Nous ajoutons un 1 : $(00110011)_2$
- Ensuite, nous observons que le nombre binaire est négatif : $-(00110011)_2$
- Nous multiplions chaques bits par son exposant de 2 : $-(2^5+2^4+2^1+2^0)$ **= -51**

b. 0110 1011 (binaire) :

- Nous observons que le nombre binaire est positif
- Nous multiplions chaques bits par son exposant de 2 : $(2^0+2^1+2^3+2^5+2^6)$ **= 107**

c. 5726 (octal) :

- 1 chiffre est représenté par 3 bits pour transformer le nombre en binaire : $(101\ 111\ 010\ 110)_2$
- Le nombre est négatif et nous renversons les bits : $-(010000101001)_2$
- Nous ajoutons un 1 : $(0100\ 0010\ 1010)_2$
- Nous multiplions chaques bits par son exposant de 2 : $-(2^1+2^3+2^5+2^{10})$ **= -1066**

d. FADE (hexadécimal, 16 bits) :

- 1 chiffre est représenté par 4 bits pour transformer le nombre en binaire : $(1111\ 1010\ 1101\ 1110)_2$
- Le nombre est négatif et nous renversons les bits : $-(0000\ 0101\ 0010\ 0001)_2$
- Nous ajoutons un 1 : $(0000\ 0101\ 0010\ 0010)_2$
- Nous multiplions chaques bits par son exposant de 2 : $-(2^1+2^5+2^8+2^{10})$ **= -1314**

e. 1000 0000 (binaire) :

- Tout d'abord, nous renversons les bits : $(01111111)_2$
- Nous ajoutons un 1 : $(10000000)_2$
- Ensuite, nous observons que le nombre binaire est négatif : $-(10000000)_2$
- Nous multiplions chaques bits par son exposant de 2 : $-(2^7)$ **= -128**

2. Dans le tableau suivant, nous allons indiquer les bases possibles pour les numéros.

Tableau 1 : Les bases possibles pour les numéros :

ID	Numéros	BIN	OCT	DEC	HEX
a.	2586			X	X
b.	0000 0000	X	X	X	X
c.	11111	X	X	X	X
d.	514		X	X	X

e.	A626				X

X : bases possibles pour les numéros

3. En langage C :

$y = x \,\&\, (5 << 4)$

L'instruction de la ligne précédente décale le nombre binaire de 5, soit $(101)_2$, de 4 bits vers la gauche. Ensuite, elle compare tous les bits de *x* avec $(1010000)_2$: 5<<4. La porte ET « & » fait une comparaison bit à bit entre les deux entiers binaires. En effet, si l'un des 2 bits comparés équivaut à 0, alors le bit de retour sera 0. Toutefois, si les 2 bits sont à 1, le bit de retour sera 1. Finalement, le résultat obtenu par cette comparaison sera mis dans la variable *y*.

4. La représentation binaire signée (complément à deux) 16-bit et en hexadécimal 16-bit des nombres entiers en base de décimale suivants :

a. -9876 :

- En base de 2 :- $(0010\ 0110\ 1001\ 0100)_2$
- Nous inversons les bits : $(1101\ 1001\ 0110\ 1011)_2$
- Nous ajoutons un 1 : $(1101\ 1001\ 0110\ 1100)_2$
- Nous faisons l'association des bits avec les valeurs hexadécimales : $(D96C)_{16}$

b. -64 :

- En base de 2 : - $(0000\ 0000\ 0100\ 0000)_2$
- Nous inversons les bits : $(1111\ 1111\ 1011\ 1111)_2$
- Nous ajoutons un 1 : $(1111\ 1111\ 1100\ 0000)_2$
- Nous faisons l'association des bits avec les valeurs hexadécimales : $(FFC0)_{16}$

c. 12345 :

- En base de 2 : $(0011\ 0000\ 0011\ 1001)_2$
- Nous faisons l'association des bits avec les valeurs hexadécimales : $(3039)_{16}$

5. Effectuons les opérations arithmétiques suivantes sur 8 bits et indiquons s'il y a un débordement signé, en plus de fournir le résultat au format hexadécimal.

a. 8B + 6A :

- 8B : 1000 1011
- 6A : 0110 1010

→ 8B + 6A = $(1000\ 1011)_2 + (0110\ 1010)_2 = (1111\ 0101)_2 = (F5)_{16}$

La réponse (F5 = -11) de l'opération n'a pas de débordement parce que les deux opérandes ont des signes opposés.

b. 52 + 49 :

- 52 : 0101 0010
- 49 : 0100 1001

→ 52 + 49 = $(0101\ 0010)_2 + (0100\ 1001)_2 = (1001\ 1011)_2 = (9B)_{16}$

La réponse (9B = -101) de l'opération a un débordement parce que les deux opérandes ont des signes positifs et le résultat est négatif. En effet, l'addition de deux nombres positifs doit avoir un résultat positif.

6. a. La valeur décimale de l'entier non signé en big-endian :

Les octets 2 à 8 de la séquence sont: C2, BB, 38 et A0.

- La valeur décimale de l'entier non signé en **big-endian** est donc la conversion de $(C2BB38A0)_{16} \rightarrow (3267049632)_{10}$.

b. La valeur décimale de l'entier non signé en little-endian :

- La valeur décimale de l'entier non signé en **little-endian** est la conversion de $(A038BBC2)_{16} \rightarrow (2688072642)_{10}$.

Exercice 2 : Disque dur

Tableau 2 : Caractéristiques d'un disque dur

Vitesse de rotation		5400 RPM
Zone	Pistes/Zone	Secteurs/Piste
1	624	792
2	1424	780
3	1680	760
4	1815	720

a) L'espace total sur le disque dur (512 B/secteur) :

D'un côté, pour calculer l'espace totale sur le disque dur, nous pouvons faire le calcul de l'espace de chaque zone pour ensuite faire l'addition et trouver l'espace total. D'un autre côté, nous pouvons faire le calcul de l'espace total sur le disque directement en additionnant le produit des zones entre Pistes/Zone et Secteurs/Piste.

$$[(624 * 792 + 1424 * 780 + 1680 * 760 + 1815 * 720) * 512\,] / 2^{30} = 2\ GiB$$

b) Le taux de lecture moyenne :

Tout d'abord, nous devons trouver le nombre moyen de secteurs par piste.

$$(792 + 780 + 760 + 720) / 4 = 763\ secteurs/piste\ en\ moyenne$$

Le taux de lecture :

$$5400 / 60 * 763 * 512 * 8 / 2^{20} = 268,24\ Mb/s$$

c) taux de lecture moyenne effective si le disque dur est connecté avec un bus PCIe de vitesse 4000 Mb/s.

Le disque dur étant connecté au bus PCIe de vitesse 4000 Mb/s, nous indique que l'interface n'est pas limitante par rapport à la vitesse de lecture moyenne du disque parce que son taux de transfert est très supérieur du taux de lecture moyenne (4000 Mb/s > 268 Mb/s). Alors, le taux de transfert reste le même, soit de 268 Mb/s.

d) **Changeriez-vous les résultats précédents si l'information sur le nombre de surfaces était disponible ?**

Si le nombre de surfaces est plus grand que 1, il sera possible de lire plusieurs pistes en même temps. Ceci veut dire que le taux de transfert est proportionnel au nombre de surfaces qu'on peut lire à la fois. Le résultat pourrait effectivement changer si l'information sur le nombre de surfaces était disponible.

Exercice 3 : Description RTN La notation RTN

Écrivez en notation RTN abstraite les instructions hypothétiques suivantes :

Nous savons que :

op<4..0>	:=	IR<31..27>
a<4..0>	:=	IR<26..22>
b<4..0>	:=	IR<21..17>
c<4..0>	:=	IR<16..12>
k<16..0>	:=	IR<16..0>

1. SUBMUL Ra, Rb, k :

- Cette instruction soustrait le contenu du registre numéro b du registre numéro a puis multiplie le tout par huit et met le résultat dans le registre numéro a
- Précisez la valeur de la constante k
- Le code d'opération de cette instruction est 5

(IR<31..27> = 5) → R[IR<16..12>] ← ((R[IR<26..22>] - R[IR<21..17>])*R[IR<16..0>]);

La valeur de la constante k est 8 : IR<16..0>.

2. DECREM Ra, Rb :

- Cette instruction décrémente les registres numéro a et b et même temps
- Le code d'opération de cette instruction est 13

(IR<31..27> = 13) → R[IR<16..12>] ← (R[IR<26..22>] -1) : R[IR<21..17>] ← (R[IR<21..17>] -1);

Exercice 4 : Architecture d'un microprocesseur

Soit la microarchitecture du processeur 32-bit simplifié suivant :

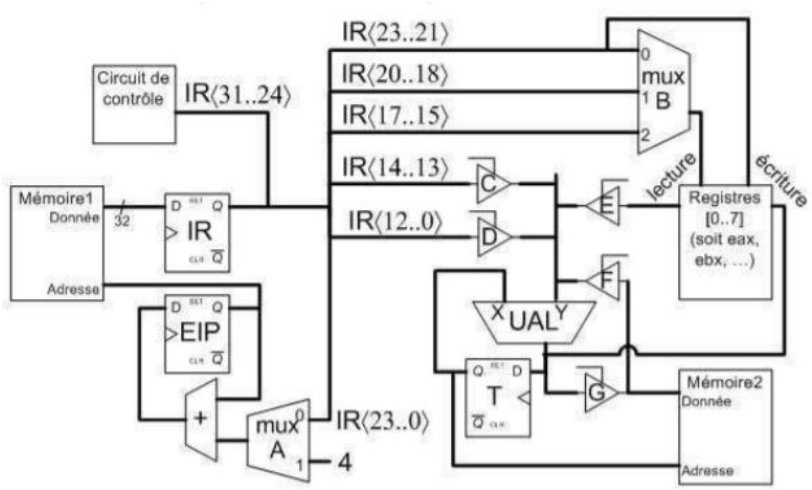

Image 1. Microarchitecture du processeur 32-bit

1. Soit l'instruction : r1 ← Memoire2 [r3] + r3 :

a) Écrivez un encodage possible (en hexadécimal) de l'instruction. Inventez l'opcode au besoin.

Tableau 3 : Encodage possible de la première instruction

Bits	31..24	23..21	20..18	17..15	14..13	12..0
Valeur	0001 1011	001	011	000	00	0 0000 0000 0000
Commentaires	opcode (arbitraire)	r1	r3	Inutile	Inutile	Inutile

Encodage possible de l'instruction: en big-endian, $(0001\ 1011\ 0010\ 1100\ 0000\ 0000\ 0000\ 0000)_2$ équivaut à : $(1B2C0000)_{16}$. En little-endian, l'instruction serait 0x00002C1B, soit l'opposé de l'encodage en big-endian.

b) Écrivez le RTN concret des macro-instructions permettant d'exécuter l'instruction de haut niveau avec la microarchitecture proposée.

T ← R[IR<20...18>];
T ← Mémoire2[T];
R[IR<23...21>] ← T + R[IR<20...18>];

c) Sous forme de tableau : pour chaque micro instruction trouvée en b), écrivez la liste des valeurs des signaux de contrôle qui en permettent l'exécution. Les noms des signaux de contrôle sont A, B, C, D, E, F, G, UAL, ecrireEIP, ecrireT et ecrireRegistre (correspondant aux éléments de mêmes noms sur le circuit). À noter que le signal B est

sur 2 bits et que l'UAL est sur 8 bits. Vous n'avez pas à effectuer la recherche d'instruction; supposez qu'elle se trouve déjà dans le registre IR.

Tableau 4 : Signaux de contrôle pour chaque micro-instruction

	SIGNAUX DE CONTRÔLE										
Micro instructions	A*	B (bin)	C	D	E	F	G	UAL (hex)	ecrireEIP	ecrireT	ecrireRegistre
T ← R[IR<20...18>]	0	0b01	0	0	1	0	0	0x0A	0	1	0
T ← Mémoire2[T]	0	0b01	0	0	0	1	0	0x0A	0	1	0
R[IR<23...21>] ← T + R[IR<20...18>]	0	0b01	0	0	1	0	0	0x4A	0	0	1

* : Nous avons mis le signal de A à zéro parce que nous voulons activer IR<23..0>. Toutefois, nous savons qu'en mettant le A à 1, nous allons déplacer l'opérande d'instruction de +4.

d) Simulation de l'instruction (r1 ← Memoire2[r3] + r3) avec le logiciel Electric :

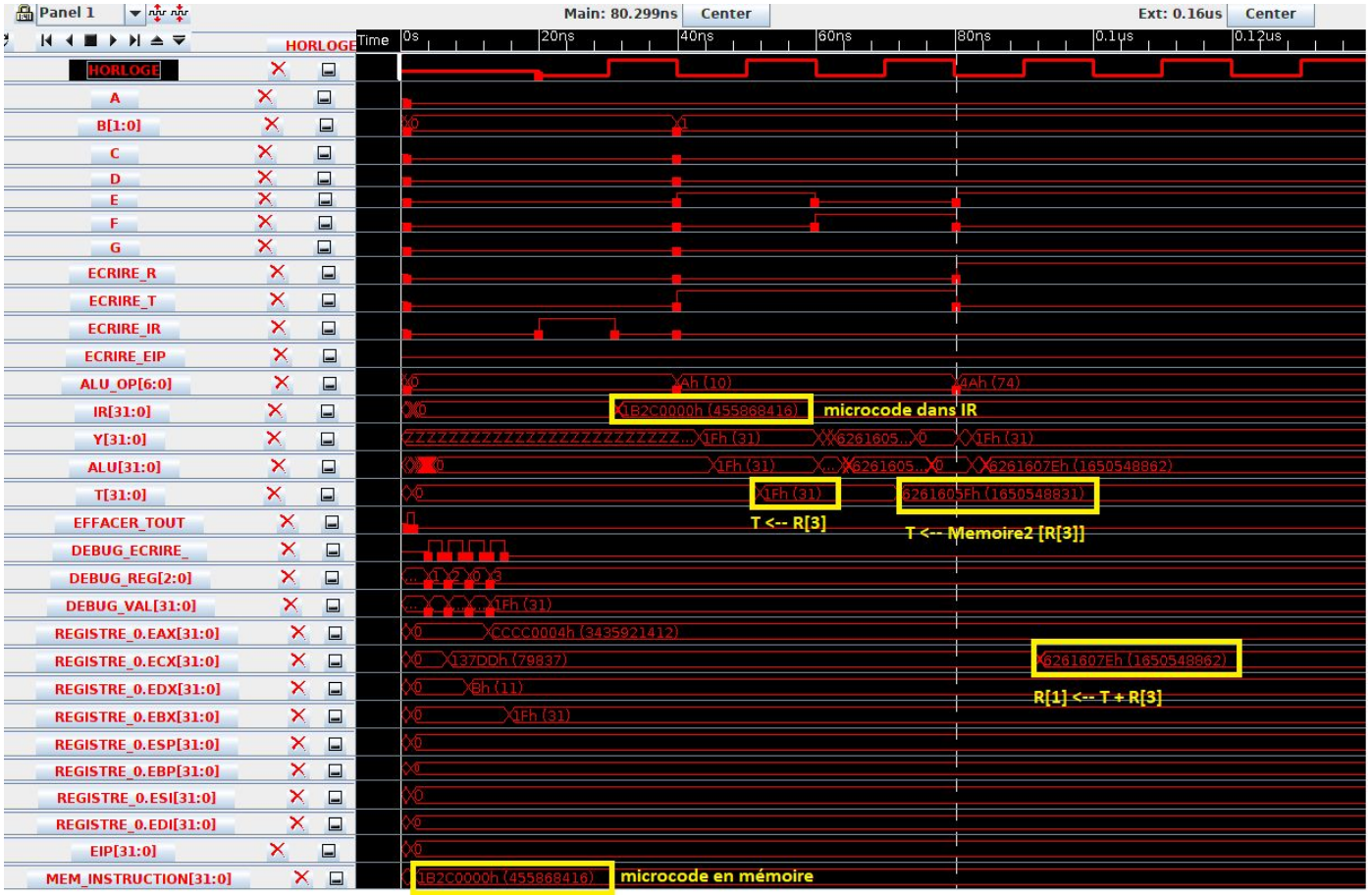

Image 2. Simulation de la première instruction

Nous observons dans l'image 2 que les micro-instructions ont agit tel que prévu. Premièrement, nous avons chargé l'instruction en mémoire et ensuite dans le registre IR. A la suite, nous avons lu la valeur du registre r3, soit 0x1F, et nous l'avons écrite dans T. Nous sommes allées lire la valeur à l'adresse r[3] de la mémoire pour l'écrire dans T. Finalement, nous avons écrit le résultat de l'opération T + r[3] dans r[1], soit ECX.

2. Soit l'instruction : r1 ← (Memoire2 [r3] + 0x23) >> r2 :

a) **Écrivez un encodage possible (en hexadécimal) de l'instruction. Inventez l'opcode au besoin.**

Tableau 5 : Encodage possible de la deuxième instruction

Bits	31..24	23..21	20..18	17..15	14..13	12..0
Valeur	0010 1100	001	011	010	00	0 0000 0010 0011
Commentaires	opcode (arbitraire)	r1	r3	r2	Inutile	0x23

Encodage possible de l'instruction: en big-endian, $(0010\ 1100\ 0010\ 1101\ 0000\ 0000\ 0010\ 0011)_2$ équivaut à : $(2C2D0023)_{16}$. En little-endian, l'instruction serait 0x23002D2C, soit l'opposé de l'encodage en big-endian.

b) Écrivez le RTN concret des macro-instructions permettant d'exécuter l'instruction de haut niveau avec la microarchitecture proposée.

T ← R[IR<20...18>];
T ← Mémoire2[T];
T ← T + R[IR<12...0>];
R[IR<23...21>] ← T >> R[IR<17...15>];

c) Sous forme de tableau : pour chaque micro instruction trouvée en b), écrivez la liste des valeurs des signaux de contrôle qui en permettent l'exécution. Les noms des signaux de contrôle sont A, B, C, D, E, F, G, UAL, ecrireEIP, ecrireT et ecrireRegistre (correspondant aux éléments de mêmes noms sur le circuit). À noter que le signal B est sur 2 bits et qu'UAL est sur 8 bits. Vous n'avez pas à effectuer la recherche d'instruction ;supposez qu'elle se trouve déjà dans le registre IR.

Tableau 6 : Signaux de contrôle pour chaque micro-instruction

	SIGNAUX DE CONTRÔLE										
Micro instructions	A	B(bin)	C	D	E	F	G	UAL(hex)	ecrireEIP	ecrireT	ecrireRegistre
T ← R[IR<20...18>]	0	0b01	0	0	1	0	0	0x0A	0	1	0
T ← Mémoire2[T]	0	0b01	0	0	0	1	0	0x0A	0	1	0
T ← T + R[IR<12...0>]	0	0b01	0	1	0	0	0	0x4A	0	1	0
R[IR<23...21>] ← T >> R[IR<17...15>]	0	0b10	0	0	1	0	0	0x11	0	0	1

d) Simulation de l'instruction (r1 ← (Memoire2 [r3] + 0x23) >> r2) avec le logiciel Electric

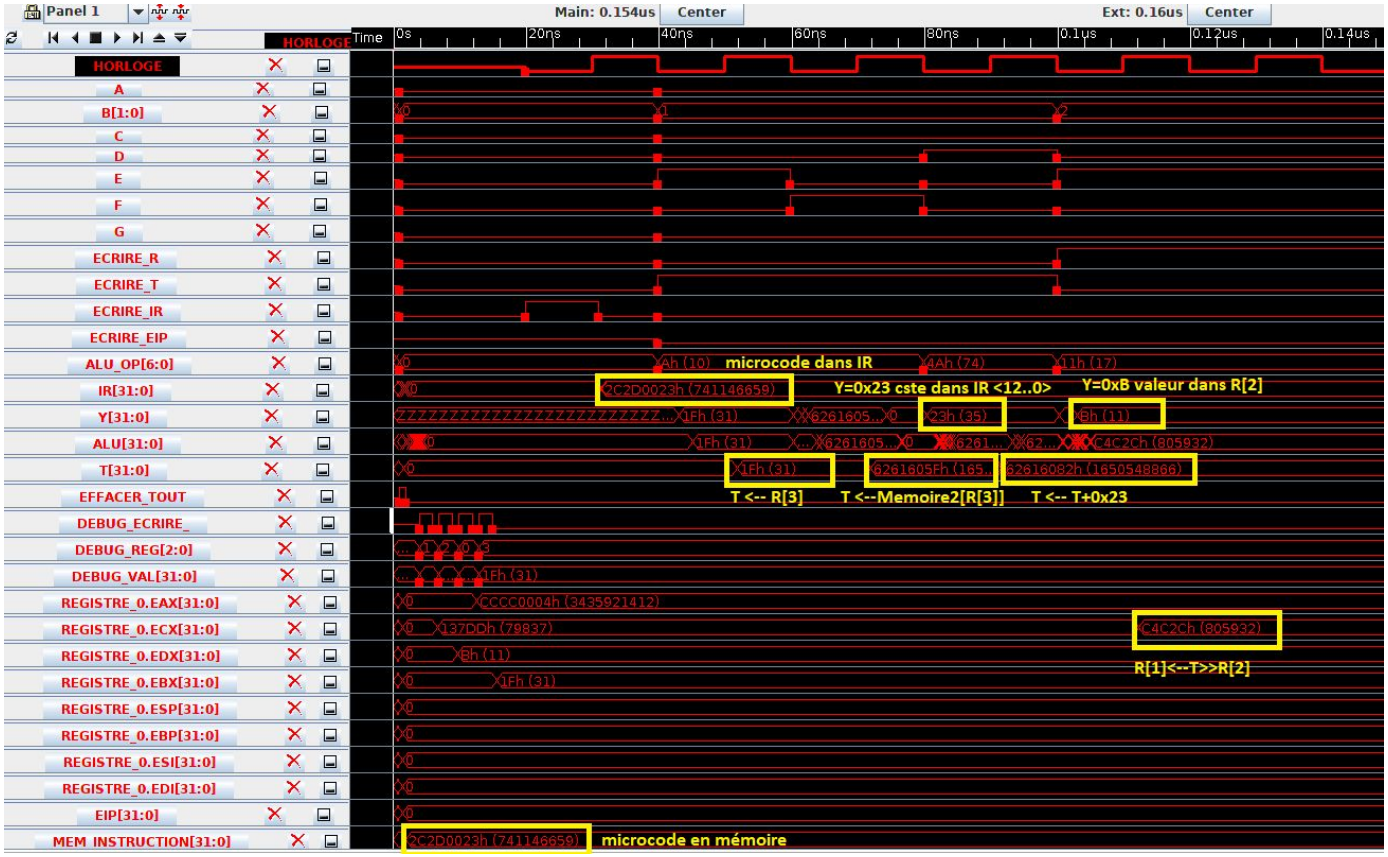

Image 3. Simulation de la deuxième instruction

Nous observons dans l'image 3 que les micro-instructions ont agit tel que prévu. Premièrement, nous avons chargé l'instruction en mémoire pour l'écrire dans IR. Ensuite, nous lisons la valeur de r[3], soit 0x1F pour l'écrire dans T. Nous sommes allées lire la valeur qui se trouve à l'adresse r[3] dans la mémoire pour l'écrire dans T. Nous rendons la constante 0x23 disponible sur le bus et nous l'ajoutons dans T. A la suite, nous lisons la valeur de r[2], soit 0xB pour effectuer un décalage du nombre de bits sur la valeur de T. Finalement, le résultat est mis dans r[1], soit ECX.