INF1600 - Architecture des micro-ordinateurs
TP2 - Architecture à deux bus et introduction à l'assembleur IA-32

Trimestre : automne 2018

Équipier 1: Nanor Janjikian (1901777)
Équipier 2 : Yasmina Abou-Nakkoul (1897266)

Équipe: 11 (groupe B1)

Présenté à :
Ulrich Dah-Achinanon

École Polytechnique de Montréal
5 novembre 2018

Exercice 1 : Architecture avec microcodes

1. Recherche d'instruction :

RTN Concret	15	14	13	12	11	10	9	8	7	6*	5*	4	3	2	1	0	hexa
MA ← PC;	0	0	1	1	X	0	0	X	0	1	1	0	0	X	0	0	**0x3060**
MD ← M[MA] : PC ← PC + 4;	0	1	1	0	1	1	0	0	1	1	0	0	0	X	0	0	**0x6CC0**
IR ← MD;	1	0	0	0	X	0	1	0	0	1	1	0	0	X	0	0	**0x8260**

* : Pour utiliser la fonction (+4) directement de l'UAL, les bits 6 et 5 à 10 (2 en décimal)
X : bits dont la valeur n'a pas d'influence. Toutefois, pour traduire en hexadécimal, nous les mettons à 0.

2. Exécution d'une instruction générique :

Soit, (IR<31..27> = opcode) →
R[IR<26..22>] ← R[IR<21..17>] oper M[R[IR<16..12>] + IR<11..0>];

RTN Concret	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	hexa
A ← R[IR<16..12>];	0	0	0	0	X	0	0	X	0	1	1	0	1	1	1	0	**0x006E**
MA ← A + IR<11..0>;	0	0	0	1	X	0	0	X	0	0	1	0	0	X	0	1	**0x1021**
MD ← M[MA] : A ← R[IR<21..17>];	0	0	0	0	1	1	0	0	1	1	1	0	1	0	1	0	**0x0CEA**
R[IR<26..22>] ← A oper MD;	1	0	0	0	X	0	1	X	0	0	0	1	0	X	0	0	**0x8210**

<u>Pour A ← R[IR<16..12>] :</u>
Bit 1 : 1 pour écrire dans A
Bit 2 : 1 pour sélectionner rc
Bit 3 : 1 parce qu'on lit dans la banque de registres
Bits 6 et 5 : pour laisser passer la valeur à travers l'UAL, mettre 11 (3 en décimal)

Pour MA← A + IR<11..0> :
Bit 12 : 1 pour écrire dans MA
Bits 6 et 5 : pour faire l'addition avec l'UAL, mettre 01 (1 en décimal)
Bit 0 : 1 parce qu'on utilise la constante de l'instruction

Pour MD ← M[MA] :A ← R[IR<21..17>] :
Bit 10 : 1 pour écrire dans MD
Bit 11 : 1 parce que la donnée vient de la mémoire
Bit 7 : 1 parce qu'on fait un accès mémoire (MD)
Bit 8 : 0 parce qu'on lit dans la mémoire
Bits 6 et 5 : pour laisser passer la valeur à travers l'UAL, mettre à 11 (3 en décimal)
Bit 2 : 1 pour sélectionner rb
Bit 1 : 1 pour écrire dans A
Bit 3 : 1 parce qu'on lit dans la banque de registres

Pour R[IR<26..22>] ← A oper MD :
Bits 6 et 5 :pour faire l'opération précisée par l'opcode, mettre à 00 (0 en décimal)
Bit 9 : 1 parce qu'on lit MD
Bit 4 : 1 parce qu'on écrit dans la banque de registres
Bit 15 est à 1 seulement pour le dernier microcode parce qu'il est le dernier.
Bit 14 est à 0 parce que l'écriture dans PC n'est pas effectué.

3. Simulations :

L'exécution à l'adresse 8 de la mémoire est une addition d'après tp2mem.txt . Dans les deux captures d'écran que nous avons prises de la simulation sur *Electric*, nous avons suivi les étapes décrites dans les exercices 1 et 2. Nous commençons par charger l'instruction dans le registre d'instructions, puis au prochain front montant, nous chargeons dans l'accumulateur A, la valeur dans rc. Nous additionnons ensuite A à la valeur écrite dans R[IR<11..0>], soit 0x107 et nous la décrivons dans MA. Nous effectuons ensuite deux instructions dans le même cycle d'horloge, soit le stockage de la valeur en mémoire à l'adresse MA, dans MD et l'entrée dans A, de la valeur de R[IR<21..17>]. Finalement, l'instruction d'addition (opération de l'UAL = 0x4A) est effectuée et son résultat est stocké dans R[1], soit ECX dans *Electric*.

Figure 1: Première image de la simulation de l'instruction générique

Figure 2: Deuxième image de la simulation de l'instruction générique

4. L'opération NAND :

Description de op[6:0] :

- op[0] à op[3] représentent les sorties de la table de vérité de l'opération NAND.

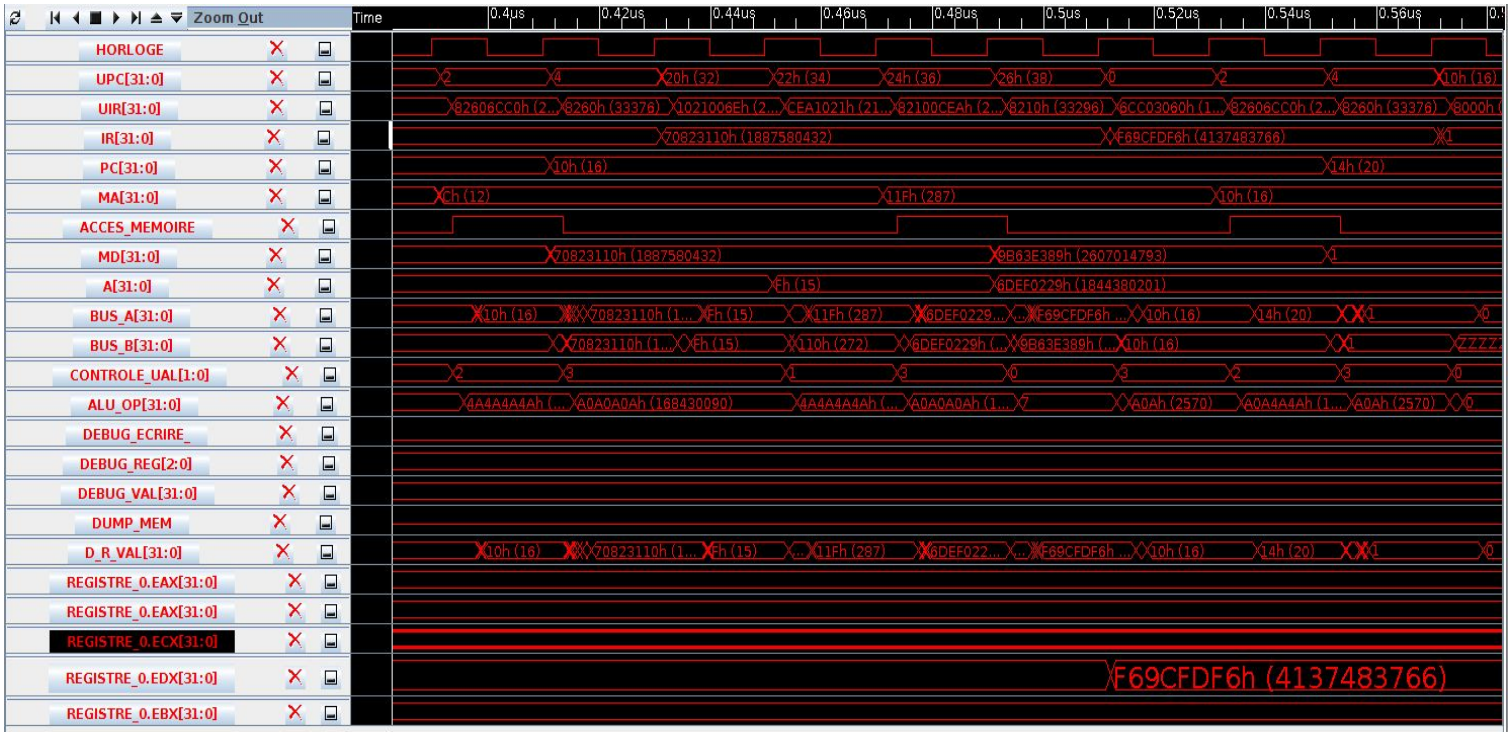
La voici :

A	B	F
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

Donc op[0] = 1, op[1] = 1, op[2] = 1 et op[3] = 0.
op[4] = 0, op[5] est un carry dans l'addition, et comme on ne l'utilise pas, nous le mettons à 0. op[6] sert à une des entrées du AND (la même addition que celle ou le op[5] est un carry). Comme nous ne voulons pas faire d'addition, nous mettons op[6] à 0.

La valeur de op[6:0] = 0000111.

Voici la simulation de l'opération NAND :



5. Compréhension :

a) Les données des deux derniers octets servent à définir en partie l'adresse du registre rc de l'instruction et la valeur immédiate dans (IR<11..0>). Dans le cas de l'instruction 0x55555555, comme nous ne savons pas ce que fait le code d'opération 0xA (écrit dans le RTN comme oper), nous pouvons assumer que l'instruction pourrait être commutative et nous pouvons nous permettre d'échanger les opérandes, soit les registres rb et rc. Une instruction sur 32 bits qui aurait exactement le même effet d'exécution serait celle-ci: 0x556AA555. Cette instruction est obtenue si on échange les registres rb et rc. Ces derniers seront tous les deux lus, nous sélectionnons le registre lu avec un mutex.

b) On peut faire en parallèle les accès mémoire et les calculs lorsque l'architecture contient deux bus. Cela a été un avantage dans le TP lors de l'exécution d'une instruction générique, soit MD ← M[MA] : A ← R[IR<21..17>];, de le RTN concret. Un accès à la mémoire et une lecture d'un

registre qu'on écrit dans l'accumulateur sont effectués, dans un seul cycle processeur.

c) Non, les instructions semblent un peu moins flexibles que celles du processeur étudié à l'exercice 4 du TP1. Dans le TP1, l'utilisation de l'adresse du registre ra sert à lire et écrire dans ce registre. Cette utilisation permet d'obtenir deux autres possibilités de lecture (par exemple, r[1] ← r[1] + r[2] * r[3] est possible). Dans ce TP, ra se trouve à l'adresse IR<26..22> qui sert à l'écriture dans le registre. Ainsi, afin d'additionner une valeur à ce qui se trouve dans le registre ra, il faut mettre son adresse dans un autre registre (rb ou rc). Finalement, il y a avait deux constantes dans l'instruction du TP1 qu'on pouvait concaténer pour obtenir une valeur sur 15 bits. Dans ce TP, il n'y a qu'une constante sur 12 bits. La seule flexibilité que le TP2 a de plus que le TP1 est que ce dernier ne permettait une banque de registre de 8 bits seulement, alors que ce TP nous offre 32 bits de flexibilité.