École Polytechnique de Montréal

Département de génie informatique et génie logiciel

INF1600

Architecture des micro-ordinateurs

TP2 : Architecture à deux bus et introduction à l'assembleur IA-32

Soumis par

Tamer Arar, 1828776

Éric Chao, 1827022

Section de labo #1

16 octobre 2016

Contenu	**Points du cours**
Exercice 1 : a + b + c + d + e	0,4 + 0,8 + 0,8 + 0,6 + 0,4
Exercice 2	1,5
Exercice 3	1,5
Français écrit erroné	jusqu'à -0,6
Format de remise erroné	jusqu'à -1
Retard	-0,025 par heure

Exercice 1

Questions

1.

Tableau I. Séquence de microcodes correspondant à une recherche d'une instruction selon l'architecture donnée.

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	0	0	0	0	0	1	1	0	0	0	0	0	0x3060
MD <- M[MA] : PC <- PC + 4;	0	1	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0x6CC0
IR <- MD ;	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0x8260

2.

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

Tableau II. Microcodes d'exécution d'une instruction selon le RTN abstrait donné.

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	0	0	0	0	0	1	1	0	1	1	1	0	0x006E
MA <- A + IR<11..0>;	0	0	0	1	0	0	0	0	0	0	1	0	0	0	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 R[IR<16..12>;	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0x8210

3.

D’après les fichiers donnés pour la simulation dans Electric, mais plus précisément grâce au opcode dans le fichier *tp2topalu.txt*, on peut prévoir que *oper* sera un additionneur.

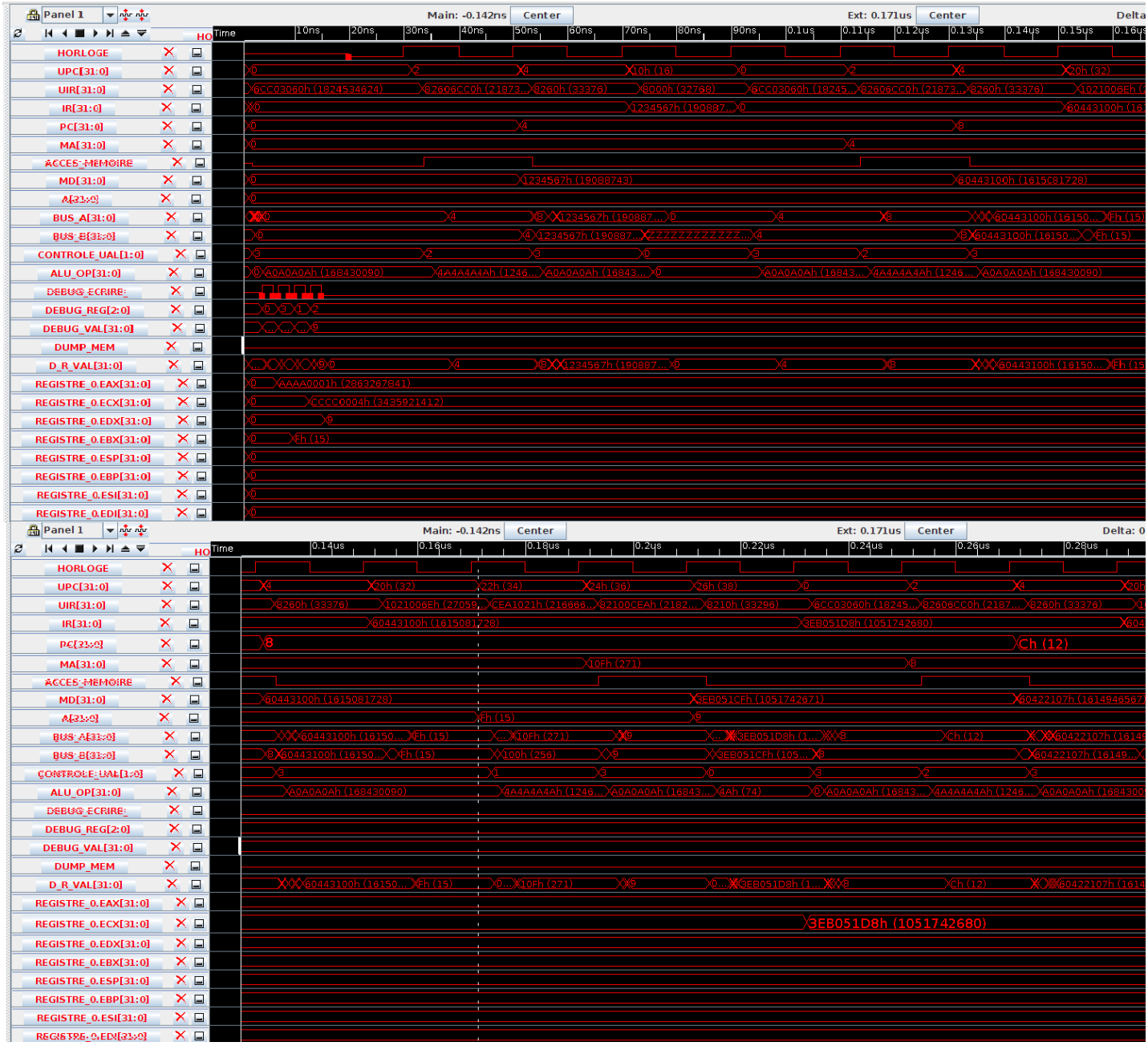

Figure 1 : Captures d’écran de la simulation de la question 3 sur le programme Electric.

On peut remarquer que le RTN concret fixé a bien été réalisé dans la simulation. On voit que la valeur du registre *rc* (EBX sur la figure) est bien transférée au registre *A* vers 0,17µs. Ensuite, une addition est faite avec la constante sur IR<11..0>, qui s'avère à être 0x100 et le résultat, 0x10F, est bel et bien transféré dans MA[31 :0] vers 0,19µs. Pendant que *A* reçoit ce qui se trouve dans le registre *rb* (EDX sur la figure), MD reçoit l'élément adressé par M[MA] soit 0x3EB051CF vers 0,21µs. Pour finir, après la troisième instruction, c'est-à-dire après l'adresse 8 du fichier *tp2mem.txt*, le résultat obtenu est 0x3EB051D8 dans le registre *ra* (ECX sur la figure). En effet, il y a bien eu une addition.

4.

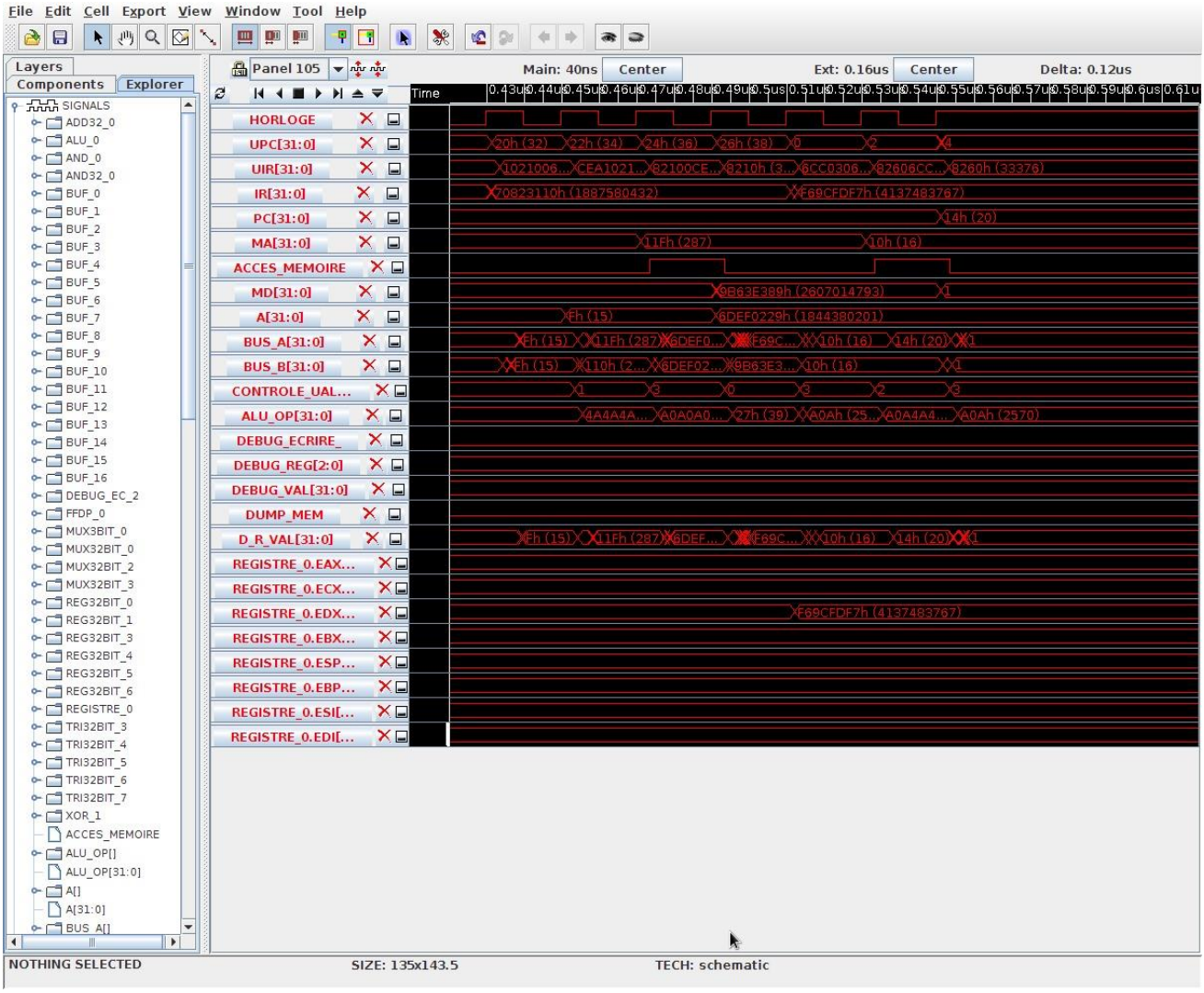

Figure 2 : Captures d'écran de la simulation de la question 4 sur le programme Electric.

La valeur de op[6:0] doit être 010 0111 soit 0x27 en hexadécimal. En effet, comme nous pouvons le voir dans cette image lorsque l'opcode de NAND (27h) est reçu par L'UAL représenté par ALU_OP dans l'image, nous pouvons remarquer que le résultat qui est écrit dans le registre EDX est le résultat d'une opération NAND du registre MD et du registre A comme nous l'avons précisé dans notre microprogramme.

Preuve :

Opération NAND

1001 1011 0110 0011 1110 0011 1000 1001 (9B63E389)

0110 1101 1110 1111 0000 0010 0010 1001 (6DEF0229)

1111 0110 1001 1100 1111 1101 1111 0110 (F69CFDF6)

Du fait qu'il y avait une retenue venant d'une opération précédente, la valeur 1 est ajouté au résultat ce qui donne

1111 0110 1001 1100 1111 1101 1111 0111 (F69CFDF7) comme affiché dans l'image dans le registre EDX

La valeur 1 est justement montré dans le registre MD : 9B63E389 est poursuivi d'un 1.

5.

a)

Puisque notre constante est sur 12 bits, le dernier octet et les 4 bits le précédent représente cette constante. Aussi, les premiers 4 bits font partie du registre *rc* (sur 5 bits), on ne doit pas les changer. On peut, donc, avoir l'instruction suivante qui aura le même effet d'exécution : 0x05555EEE

b) Une architecture à deux bus réduit le nombre de micro-instructions total que l'on doit faire pour terminer une tâche et de plus, nous n'avons pas besoin d'attendre la disponibilité du bus pour pouvoir stocker notre donnée dans le registre en stockant la donnée temporairement dans le registre C dans une architecture à un bus. Nous pouvons directement stocker notre donnée dans le registre en le faisant passer par le deuxième bus. Dans notre cas, lorsque nous faisions une opération, la valeur était directement transmise dans le registre qui devait la contenir ce qui nous permettait de sauver une action du processeur par opération dans L'UAL, donc deux au total dans notre microprogramme.

c) Non, cela revient au même, car l'UAL prendra aussi, dans cette architecture, 2 cycles pour effectuer une opération logique. Il ne peut pas recevoir 2 entrées en même temps. Une entrée

doit obligatoirement passer par le Registre A, afin de laisser au résultat, une fois généré, une chance d'accéder au bus A.

Exercice 2

flds c	Pile = c
flds d	Pile = d, c
flds e	Pile = e, d, c
faddp	st0 = e+d
faddp	st0 = (e+d)+c
flds g	Pile = (e+d+c), g
fmulp	st0 = (e+d+c)*g
fstps a	a = (e+d+c)*g ----------
flds b	Pile = b
flds f	Pile = f, b
fsubp	st0 = f-b
flds d	Pile = d, (f-b)
flds e	Pile = e, d, (f-b)
fmulp	st0 = d*e
fdivrp	st0 = (f-b)/(d*e)
flds a	Pile = ((e+d+c)*g),((f-b)/(d*e))
fmulp	st0 = ((e+d+c)*g)*((f-b)/(d*e))
fstps a	a = ((e+d+c)*g)*((f-b)/(d*e)) ----------
flds g	Pile = g
flds d	Pile = d, g
fdivrp	st0 = g/d
flds b	Pile = b, (g/d)
faddp	st0 = b + (g/d)

```
flds a          Pile = (((e+d+c)*g)*((f-b)/(d*e))), (b + (g/d))
fsubp           st0 = (((e+d+c)*g)*((f-b)/(d*e))) - (b + (g/d))
fstps a         a = (((e+d+c)*g)*((f-b)/(d*e))) - (b + (g/d)) ----------
```

Exercice 3

```
        mov $0, %esi
        mov $10, %edi
again:
        cmp %edi, %esi
        ja done
        mov a, %eax
        mov b, %ebx
        mov e, %edx
        add %edx, %eax
        sub %ebx, %eax
        mov %eax, a

if:
        mov e, %edx
        add $1, %edx
        cmp %edx, %esi
        jna next
        mov b, %ebx
        mov c, %ecx
        add $2, %ecx
```

```
        cmp %ecx, %ebx
        jne else
next:
        mov a, %eax
        mov c, %ecx
        mov e, %edx
        add %ecx, %eax
        sub %edx, %eax
        mov %eax, a

        mov b, %ebx
        add $2, %ebx
        mov %ebx, b

        add $1, %esi
        jmp again
else:
        mov a, %eax
        mov c, %ecx
        add d, %eax
        sub %ecx, %eax
        mov %eax, a

        mov c, %ecx
        sub $1, %ecx
        mov %ecx, c
```

```
        add $1, %esi
        jmp again

done:
        ret
```