Laboratoire 4
Organisation Informatique de Base

CEG 2536 - Architecture des ordinateurs I

Université d'Ottawa

Professeur : Michel Saydé

Noms et numéros des étudiants :
Gbegbe Decaho Jacques 300094197
Fatme Adb-Ali 300047012

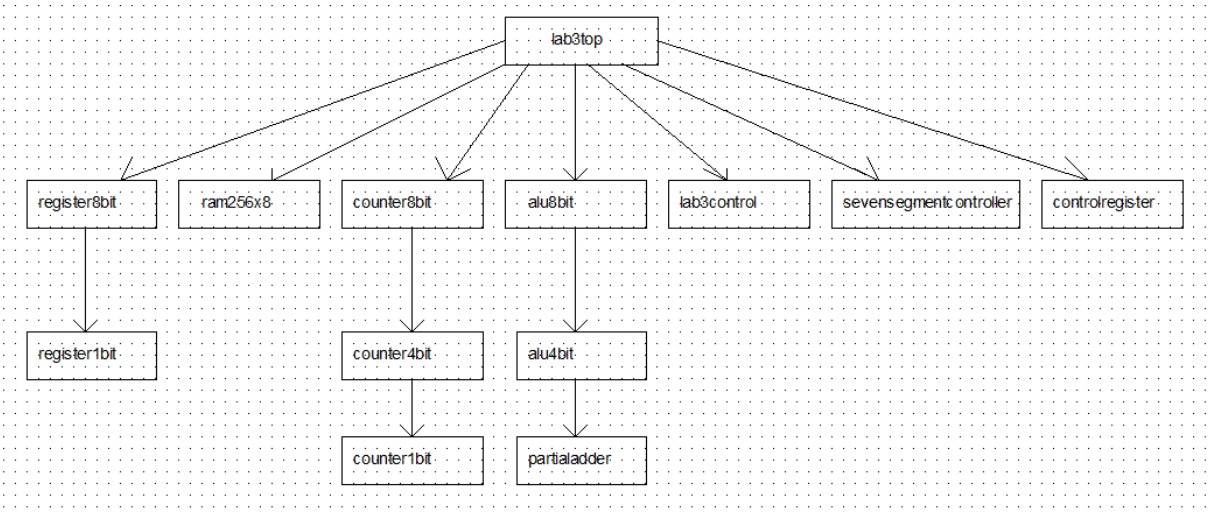
Date de soumission :

Liste des figures :

Liste des tableaux :

Prelab - Matériel

1) Schéma hiérarchique des fichiers



2) On peut vérifier par analyse de ces fichiers que seul un régistre placera sa sortie sur le bus de données à la fois car c'est un seul régistre à la fois qui peut placer sa sortie sur un bus à un cycle donnée car l'interface entre les régistres et le bus est un multiplexeur avec N entré et 1 seule sortie.
Un seul des régistres peut être une sortie du multiplexeur à un instant donné, donc un seul régistre sera chargé sur le bus à cet instant donné.

3) Les signaux de réinitialisation du régistre sont asynchrone car le signal d'effacement des régistres ici utilisé est donné comme une entrée DFF au lieu son clair ce qui efface l'entrée au lieu d'utiliser l'effacement DFF. D'où la sortie n'est évaluée que lorsque D est réevaluée et ne sera effacée qu'au prochain cycle d'horloge.

4) Lorsqu'un load et une réinitialisation sont envoyés simultanément à un régistre, la bascule sera réinitialisée. Ceci est dû au fait que l'entrée DFF dépend toujours du fait que la réinitialisation est asynchrone, elle ne suit donc pas le sens de l'horloge contrairement au load qui est synchronisé avec l'horloge.

5) Le régistre est directement relié à la mémoire parce que seule une des nombreuses valeurs stockés dans la mémoire ne peut sortir d'elle, c'est le régistre d'adresse qui indique à la mémoire laquelle des valeurs stockées doit sortir.

6) Le compteur de programme, le régistre de données, et l'accumulateur sont mis en œuvre en tant que compteurs car les compteurs sont simples à charger car plus efficaces à utiliser qu'un circuit logique séparé par incrémente le régistre.

7) Parmi les commandes (reset, incrément, et load) des compteurs, celui qui a la plus grande priorité est <Incrément> à côté de reset et load. L'effacement ne se produit efficacement que si load et increment sont tous les deux à 0.
Load a la priorité la plus basse, il ne produit d'effets que si Increment est égal à 0 et reset à IN'.

8) Non, il n'est pas possible de lire une valeur directement de la mémoire de l'accumulateur. Car la seule entrée vers l'accumulateur provient de l'ALU, celle-ci a la capacité de charger une valeur directement de DR vers AC, mais elle n'a pas d'entrée directe depuis le bus. Ce chargement de valeurs de l'ALU vers l'accumulateur nécessite 2 cycles d'horloge et n'est donc pas direct.

9) Analyse et détermination de la table de vérité décrivant les 8 opérations pouvant êtres sélectionnés par 3 lignes de commandes.
Les opérations de décalage sont arithmétiques.

A partir du tableau 5

S2	S1	S0	Opérations
0	0	0	AC ← AC+DR
0	0	1	AC ← AC+DR'+1
0	1	0	AC ← ashl AC
0	1	1	AC ← ashr AC
1	0	0	AC ← AC Λ DR
1	0	1	AC ← AC V DR
1	1	0	AC ← DR
1	1	1	AC ← AC'

Tableau 1 : Table d'opération d'ALU

10) Design de l'unité de contrôle

Signaux de commande	Expressions	Expressions RTL	Signaux de commande	Expressions	Expressions RTL
1.Memwrite	M[AR] ← AC SC ← 0 M[AR] ← DR	T9Y4+T10Y6	12.ALU_Sel2		T5X1IR1+T9Y3+T9Y0
2. AR_Load	AR ←PC AR ← PC AR ← OUTA AR ← M[AR] AR ← M[AR]	T2+T5IR'6+T0+T6IR'6+T7X2	13.ALU_Sel1		T9Y3+T5X1(IR1+IR2+IR3)
3.PC_Load	PC ← AR	T8Y5	14.ALU_Sel0		T9Y2+T5X1(IR3+IR1)
4.PC_Inc	PC ← PC + 1 PC ← PC + 1 PC ← PC + 1 PC ← PC + 1	(IR6'.T5.STOP')+(T11+T12).y6.STOP'.(DR0+DR1+DR2+DR3+DR4+DR5+DR6+DR7)'+(T2.STOP')	15.BusSel2		T0+T9Y4
5.DR_Load	DR ← M[AR] DR ← M[AR] DR ← M[AR] DR ← M[AR]	T8*(Y0+Y1+Y2+Y3+Y6)	16.BusSel1		T2+T5+T10Y6+T0

	DR ← M[AR]				
6.DR_Inc	DR ← DR + 1	T9Y6	17.BusSel0		T10Y6+T8Y5+T9Y4
7.IR_Load	IR ← M[AR]	T3	18.SC_Clear	SC ← 0	T5X1+T9(Y0+Y1+Y2+Y3+Y4)+T8Y5+T12Y6
8.AC_Clear	AC ← 0	T5X1IR0	19.Halt	S ← 1	T5X1IR5
9.AC_Load	AC ← AC’ AC ← ashl AC AC ← ashr AC AC ← AC (AND) DR AC ← AC + DR AC ← AC - DR AC ← DR	T5X1(IR1+IR2+IR3) + T9(Y0+Y1+Y2+Y3)			
10.AC_Inc	AC ← AC + 1	T5X1IR4			
11.OUTD_Load	OUTD ← M[AR]	T1			

tableau 2 : Design de l’unité de contrôle

Procedure - Matériel

- quartus II
- Carte Altera

6. Construire et tester l'unité de contrôle

1. Exécution automatique

★ Schéma des diagrammes

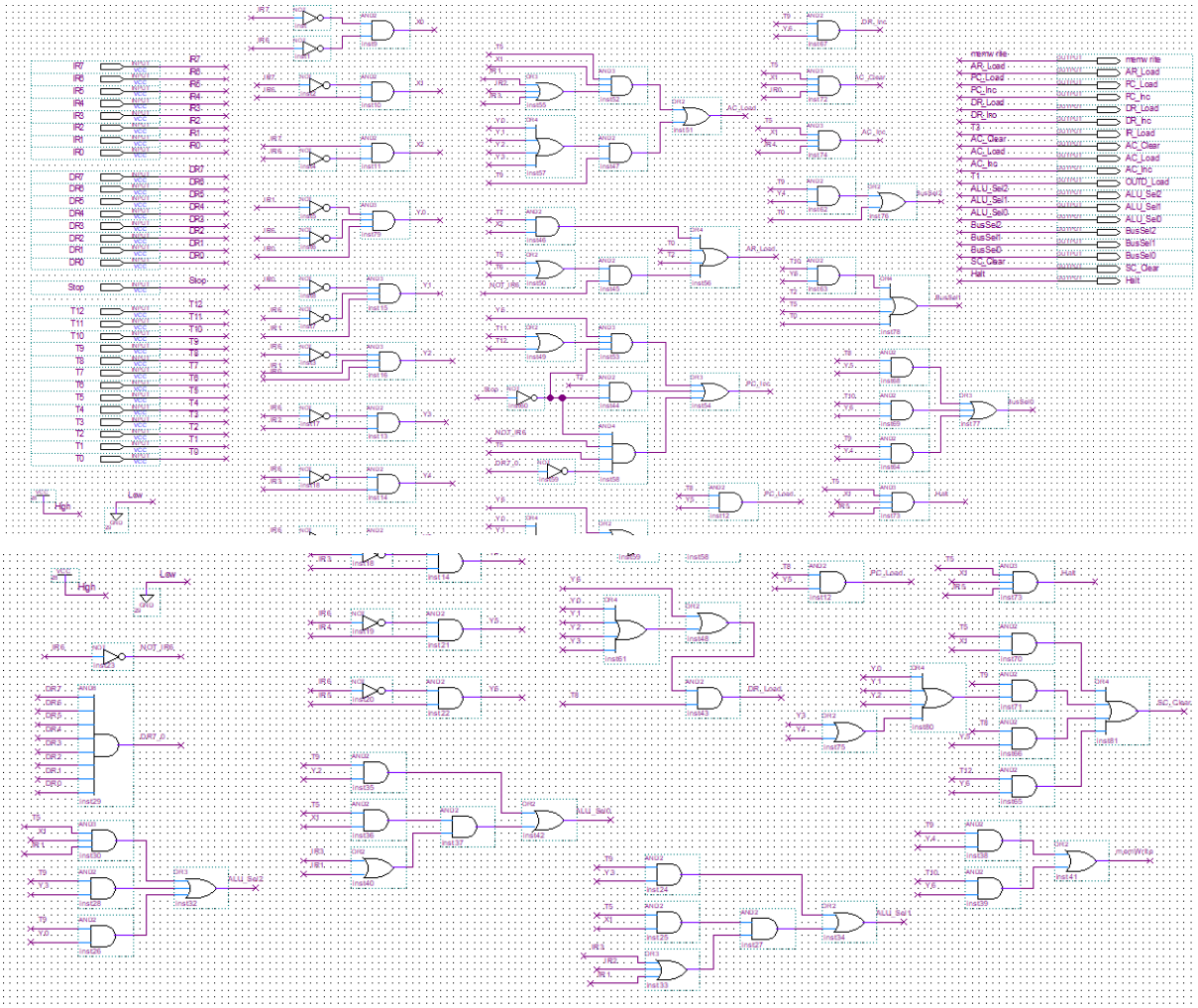


Figure 1 :Diagramme de l'unité de contrôle lab3top

★ Compilations

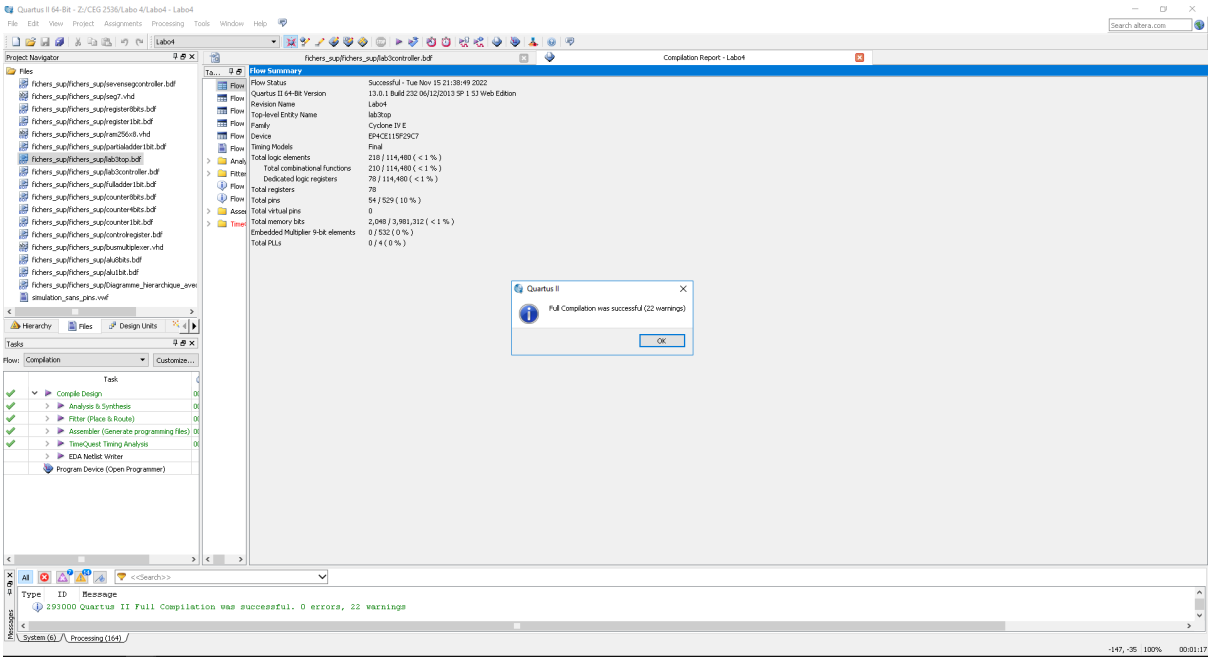


figure 2 : compilations

★ Simulations

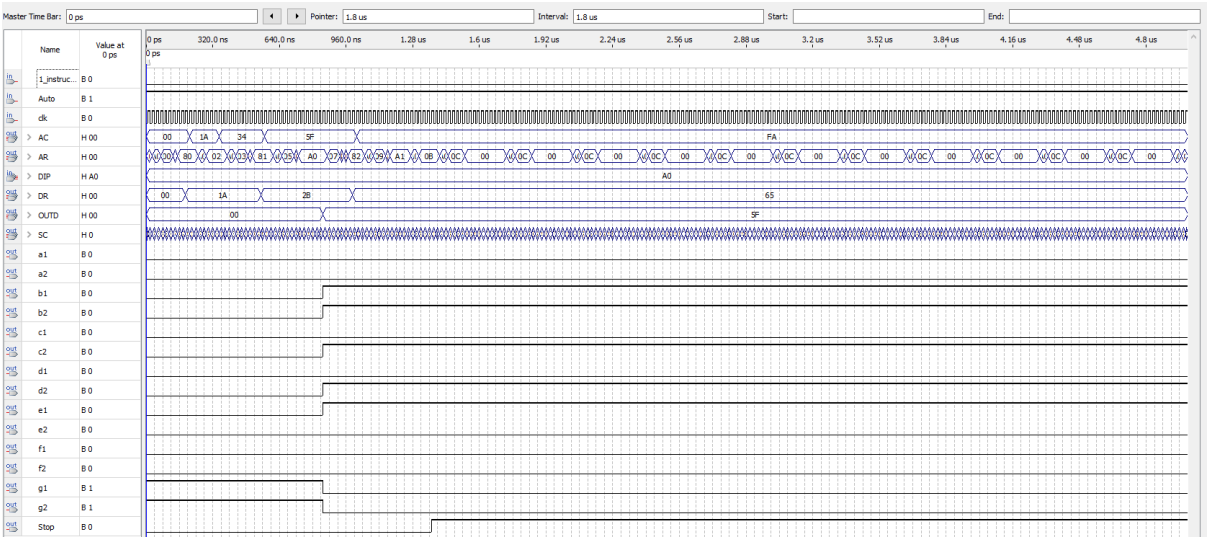


Figure 3 : Simulation de lab3top lorsque DIP pointe vers A0

Lorsque DIP pointe vers A0 la sortie obtenue est OUTD = 5F

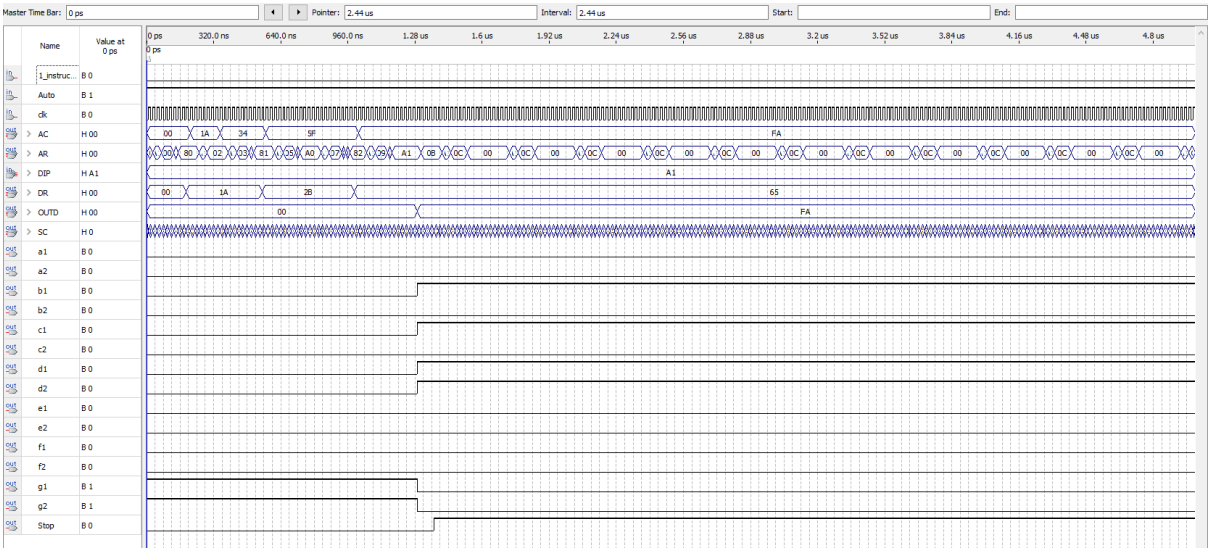
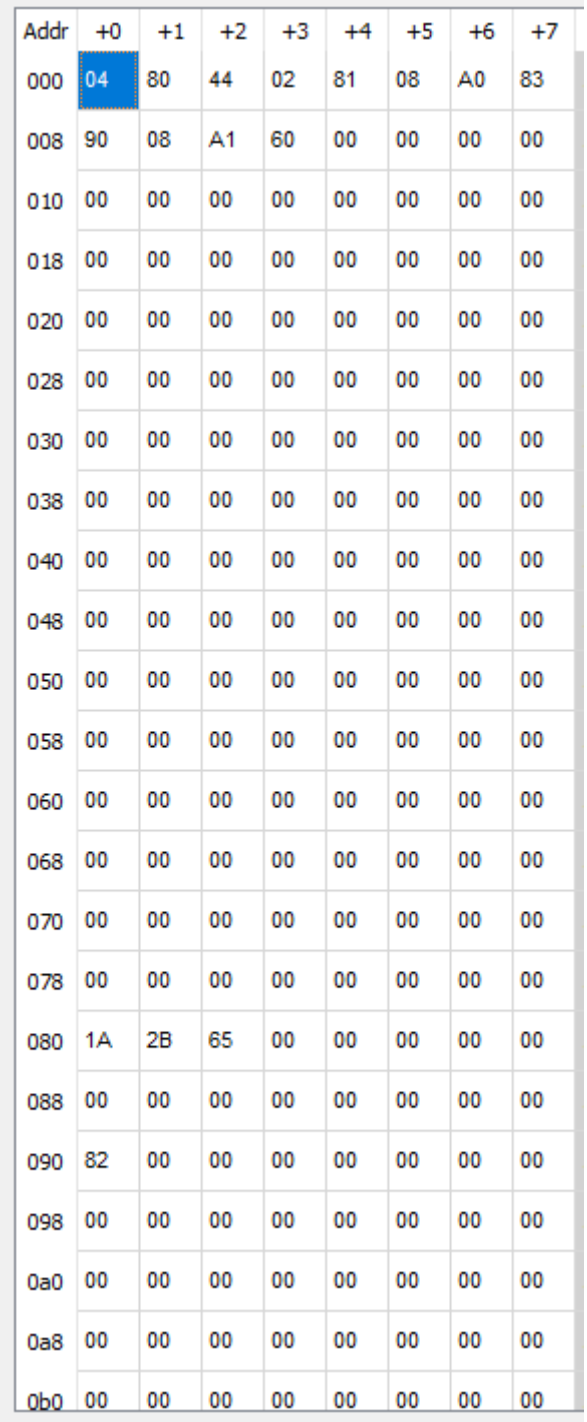


Figure 4 : Simulation de lab3top lorsque DIP pointe vers A1

Lorsque DIP pointe vers A1 la sortie obtenue est OUTD = FA

7. Prélab - Logiciel

7.1 Analyse de programme



Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	04	80	44	02	81	08	A0	83
008	90	08	A1	60	00	00	00	00
010	00	00	00	00	00	00	00	00
018	00	00	00	00	00	00	00	00
020	00	00	00	00	00	00	00	00
028	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00
038	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00
048	00	00	00	00	00	00	00	00
050	00	00	00	00	00	00	00	00
058	00	00	00	00	00	00	00	00
060	00	00	00	00	00	00	00	00
068	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00
078	00	00	00	00	00	00	00	00
080	1A	2B	65	00	00	00	00	00
088	00	00	00	00	00	00	00	00
090	82	00	00	00	00	00	00	00
098	00	00	00	00	00	00	00	00
0a0	00	00	00	00	00	00	00	00
0a8	00	00	00	00	00	00	00	00
0b0	00	00	00	00	00	00	00	00

Figure 5 : memorycontent8 (.mif fichier qui représente le programme en code machine)

1. Analysons le programme

1.Charger le compteur	Chargée de A0 dans AC
2.Complément du compteur	Complément AC
3.Sauvegarder le compteur dans la mémoire de AC	Sauvegarder le complément du compteur dans A0
4. Incrémenter le compteur et le sauvegarder	Incrémenter la valeur de A0 par 1. S'arrêter si Compteur =0
5. brunch to address 20	Si le compteur !=0, exécuter l'instruction se trouvant dans l'adresse 20
6.Charger (indirect) la mémoire de A1	X est chargé dans AC
7.Ajouter x=x+y (indirect)	Ajout indirect de la valeur de A2 (Y) dans AC (X)
8.Stocker z←x+y	Stocker AC(X+Y) dans M[A3] = Z
9.Charger x	Charger X dans AC
10.Incrémenter X	Incrémenter AC
11.Stocker A1←X	Stocker X obtenu dans A1
12.Incrémenter	Incrémenter X encore
13.Stocker A2←Y	Stocker la valeur de X dans A2
14.Inc	Incrémenter X dans AC encore
15.Stocker A3←Z	Stocker AC dans Z encore et program brunch pour vérifier la 4e condition

Tableau 3 : tableau d'analyse du code de la partie 7.1.1

2. Pseudo-code du programme

```
int compteur = 1, n = 144, x = 1, y = x + 1;
System.out.print (“ ” + n)
```

```
while (i <= n)
    System.out.println(x + " , ");

    int z = x + y ;
    x = y ;
    y = z ;

    compteur ++;
```

3. Le programme calcul une séquence de Fibonacci ou le nombre total est ajouté. Il compte jusqu'à 144 qui est la 10e valeur.
4. Il est pratique d'utiliser les instructions de référence de la mémoire avec une adresse indirecte parce que le pointeur peut être facilement incrémenter sans avoir à créer de nombreux sous-programmes.

7.2. Conception de programme

1. programme au format .mif

   Ce programme à pour instruction d'ajouter consécutivement une série de numéro faisant partie d'une séquence de nombres hexadécimales.

1. Pseudocode :

Address	Contents	Code	Function
00	04	LDA	Load A2 into AC
01	A2	A2	
02	42	CMA	Complément AC
03	08	STA	Store AC into A2
04	A2	A2	
05	20	ISZ	Increment A2. if A2 was 0, skip next instruction
06	A2	A2	

07	10	BUN	Brunch to 0A
08	0A	0A	
09	60	HLT	Halt
0A	84	LDA (indirect)	Load value in memory referenced by A1 in AC
0B	A1	A1	
0C	02	ADD	Add A0 to AC
0D	A0	A0	
0E	08	STA	Store AC into A0
0F	A0	A0	
10	42	CMA	Complément AC (if AC was 00, becomes FF)
11	08	STA	Store AC into A3
12	A3	A3	
13	20	ISZ	Increment A3, if A3 was 0, skip next instruction
14	A3	A3	
15	10	BUN	Brunch to 1C
16	1C	1C	
17	84	LDA (indirect)	Load value in memory referenced by A1 in AC
18	A1	A1	
19	08	STA	Store AC into A3

1A	A3	A3	
1B	60	HLT	Halt
1C	04	LDA	Load A1 into AC
1D	A1	A1	
1E	50	INC	Increment AC
1F	08	STA	Store AC into A1
20	A1	A1	
21	10	BUN	Brunch to 05
22	05	05	
80	21		Values from question
81	B5		
82	37		
83	08		
84	5C		
85	84		
86	A1		
87	1D		
88	72		
89	FF		
8A	F6		
8B	43		
8C	03		
8D	A9		
8E	D4		
8F	19		

90	31		
91	D9		
92	47		
93	82		
94	14		
95	52		
96	07		
97	CA		
98	04		
A0	00		
A1	80		
A2	19		
A3	00		

Tableau 4 : Tableau du Programme d'addition en langage assembleur sous forme de table

La table de cette séquence est:

Ajout	+0	+1	+2	+3	+4	+5	+6	+7
80	21	B5	37	08	5C	84	A1	1D
88	72	FF	F6	43	03	A9	D4	19
90	31	D9	47	82	14	52	07	CA
98	04							
a0	PTR	SUM	Dernier ajout	variable temp				

Tableau 5 : table de séquence

2.Écrire un programme qui peut multiplier un nombre non signé de 4 bits

Address	Contents	Code	Function
00	04	LDA	Load A0 into AC
01	A0	A0	
02	42	CMA	Complement AC
03	08	STA	Store AC into A3
04	A3	A3	
05	41	CLA	Clear AC
06	20	ISZ	Increment A3. If A3 was 0, skip next instruction
07	A3	A3	
08	10	BUN	Branch to 0C
09	0C	0C	
0A	10	BUN	Branch to 10
0B	10	10	
0C	02	ADD	Add A1 to AC (AC becomes A1+A0)
0D	A1	A1	
0E	10	BUN	
0F	06	06	
10	08	STA	Store AC into A2
11	A2	A2	
12	60	HLT	Halt program

A0	3		First Operand
A1	5		Second Operand
A2	00		Running sum
A3	00		Temp - Used as counter (without changing X)

Tableau 6 :code en langage assembleur d'un programme pouvant multiplier un nombre non signé de 4 bits.

★ Simulation des tableaux

8. Procédure - Logiciel

(Nous l'avons présentés à notre AE**)**

Conclusion et Discussion

Pour commencer, nous avons d'abord analysé la structure de base d'un ordinateur. Puis nous avons conçu une unité de commande avec une mémoire d'ordinateur ayant une capacité de 256 mots de 8 bits pouvant stocker des programmes et des données. Nous avons aussi écrit un programme simple en code machine en utilisant des opcodes. De plus, nous avons fait la simulation sur la carte altera, le tout en utilisant Quartus II. Pour finir, nous avons comparé les résultats obtenus sur la carte Altera aux résultats attendus.

Nous avons alors constaté que notre résultat attendu était inversé sur la carte Altera mais nous avons été rassurés par le TA que c'est un problème commun pour tous les étudiants de ce laboratoire. De plus, nous avons éprouvé quelques difficultés avec la simulation. Enfin, nous avons éprouvé le plus de difficultés lorsqu'il fallait écrire notre programme en opcodes car nous n'étions pas encore entièrement familiarisé avec cette matière mais nous avions quand même réussi à le faire à temps.

En conclusion, nous pouvons dire que ce laboratoire était quand même réussi, malgré les petites complications, et qu'il nous a bien permis de mettre en pratique ce que nous venions d'apprendre en classe.