LAB 1: Introduction à la programmation de microprocesseur
CEG3536 - Architecture des ordinateurs II
University of Ottawa

Professor : **Mohamed Ali Ibrahim**
AE : Hamidou Nouhoum Cisse
Céline Girard

Rapport de laboratoire écrit par :
Brian Makos|300194563
Scott Makos|300194574

Date du Laboratoire :

Septembre 23 2022

Partie Théorique:

Introduction : Ce laboratoire comporte sur la creation d'un module Delay qui crée un délai qui peut être agrandi avec une incrémentation. De plus, Nous aurons aussi a concevoir un programme avec le moniteur D-Bug12et d'analyser le mode d'adressage utilisé à chaque instruction du programme. (CONTEXT MAYBE). Ce laboratoire nous familiarise avec les modes d'adressage ainsi et nous donne l'expérience de coder un programme assembleur.

Definition du probléme :

Partie 1 : Afin de pouvoir resoudre ce problème, nous aurons besoin d'analyser chaque étape exécuter du code donnée.Une fois terminé, nous aurons besoin d'explique le but du code, lui apporter quelque changement et analyser le code suite au changement.

Partie 2 : Cetter section du laboratoire demande le débogage du sous-programme *isCodeValid* ainsi que l'analyse du fichier *alarmSimul.asm*.

Partie 3 : Pour cette partie du laboratoire, nous aurons besoin de programmer un module *Armed*, celui-ci contient aussi un module de delai qui peut incrementer. Cependant, le programme aura aussi besoin de pouvoir executer certaines taches lorsqu'il est en délai.

Description des Entrée/Sortie :

Partie 1 : Pour cette section, le code va imprimer « : » après exécuter la commande « JSR 0,X »(quand *putchar* est exécuter avant) Par la suite, nous commençons un boucle (accumulateur de valeur 3) quand LDAA #3 est executer par la suite. Ce code, va imprimer une espace, suivi par le caractère choisis par l'utilisateur lors de l'appelle de *getchar*. L'espace et le caractère sera imprimer 3 fois jusqu'à ce que le L'accumulateur atteint 0. Le code qu'il faut ajouter est un autre appelle a *getchar* dans la boucle comme on perd la valeur de celle-ci lors du 2^{ième} boucle

Partie 2/Partie 3 : Cette partie du laboratoire est un programme d'alarme. Une fois que, *isCodeValid* est déboguer, et le code du Partie 3 est écrit, celui-ci devrait avoir un delai de 10sec entre l'écran « Alarming » et « Armed »

Comme entrées, nous avons les codes pour activer l'alarme. Comme sortie nous aurons un delai d'un certain temps en ms entre l'écran « Alarming » et « Armed ».

Partie 1 – Utiliser le moniteur D-Bug-12

a. Avec le moniteur D-Bug12, assemblez le programme (avec la commande ASM du D-Bug12) suivant à l'adresse $2000. (Note : tapez le point « . » pour sortir du mode ASM.) En entrant le programme, complétez b ci-dessous. Après la vérification que le programme est bien en mémoire, démarrez le programme avec la commande G 2000. Notez que l'instruction PSHA est stockée à l'adresse $200F. Cette adresse est utilisée dans l'instruction BNE pour la création d'une boucle.

Adresse	Mnémonique	Code opér.	Commentaires
$2000	LDD	#$3A	Charger code pour ":" dans B
	LDX	$EE86	Chargez le vecteur pour le sous-programme *putchar*
	JSR	0,X	Imprimer au terminal le contenu de B
	LDX	$EE84	Chargez le vecteur pour la routine *getchar*
	JSR	0,X	Placez un nouveau caractère dans B
	LDAA	#3	Initialise le compteur de boucle
$200F	PSHA		Sauvez le compteur sur la pile
	PSHB		Sauvez le contenu de B sur la pile
	LDD	#$20	Chargez B avec une espace
	LDX	$EE86	Chargez le vecteur pour le sous-programme *putchar*
	JSR	0,X	Imprimer au terminal
	PULB		Récupérer le caractère original
	LDX	$EE86	Chargez le vecteur pour le sous-programme *putchar*
	JSR	0,X	Imprimer au terminal
	PULA		Récupérer le compteur
	DECA		Décrémenter le compteur de boucle.
	BNE	$200F	Si compteur <> 0, répéter.
	SWI		Retourner au moniteur

b. À mesure que vous entrez les instructions à l'assembleur, enregistrez, dans le tableau cidessous, le code machine dans la colonne « Contenu » ainsi que son adresse dans la colonne « Adresse ». Voyez la première entrée comme exemple. À mesure que vous entrez les instructions, essayez de prédire le code machine avant de l'assembler. Dans votre rapport diviser l'instruction machine dans les parties code d'opération et opérande pour les instructions suivantes. Expliquez le mode d'adressage utilisé par chaque instruction.

- LDX $EE86

    - Cet commande prend l'information dans RAM et le store dans le registre X. $EE86 est une commande pour la fonction putchar
- JSR 0,X
    - Cette commande "Jump to Subroutine" prend la fonction dans le registre X et le place sur le stack. Cette commande (sub-routine) est ensuite executer
- BNE $200F
    - "branch not equal" cette commande saute au adresse $200F si la condition « not equal 0 » est atteint.

Tableau 1.1 : List Instructions, leurs adresse et leur contenu trouver lors de l'execution des instructions données dans le terminal du logiciel miniIDE.

Adresse	Contenu	Instruction	Description
2000	CC003A	LDD #$3A	Charger code pour ":" dans B
2003	FEEE86	LDX $EE86	Chargez le vecteur pour le sous- programme putchar
2006	1500	JSR 0,X	Imprimer au terminal le contenu de B
2008	FEEE84	LDX $ EE84	Chargez le vecteur pour la routine getchar
200B	1500	JSR 0,X	Placez un nouveau caractère dans B
200D	8603	LDAA #3	Initialise le compteur de boucle
200F	36	PSHA	Sauvez le compteur sur la pile
2010	37	PSHB	Sauvez le contenu de B sur la pile
2011	CC0020	LDD #$20	Chargez B avec une espace
2014	FEEE86	LDX $EE86	Chargez le vecteur pour le sous-programme putchar
2017	1500	JSR 0,X	Imprimer au terminal
2019	33	PULB	Récupérer le caractère original
201A	FEEE86	LDX $EE86	Chargez le vecteur pour le sous-programme putchar
201D	1500	JSR 0,X	Imprimer au terminal
201F	32	PULA	Récupérer le compteur
2020	43	DECA	Décrémenter le compteur de boucle
2021	26EC	BNE $200F	Si compteur <> 0, répéter
2023	3F	SWI	Retourner au moniteur

c. Dans votre rapport, expliquez ce que fait le programme? Utilisez une fonction C dans votre explication.

Ce programme commence en imprimant la valeur du code ASCII qui est stocké lors de la commande LDD dans l'adresse 2000 et l'imprime grâce a la fonction putchar (ou LDX $EE86) qui est executer avec la commande JSR 0,X. Par la suite, La commande getchar (LDX $EE84) est executer avec JSR 0,X et l'utilisateur entre un character. Par la suite, la commande LDAA (load acummulator) est executer et dependant de la valeur hexadecimal associer a cette commande, il y aura a loop qui execute justqu'a ce que cette valeur est decrementer à 0 (grâce a DECA (decremente accumualteur) et BNE (branch not equal to 0)). Lors de chaque itération de ce loop, le programme PUSH la valeur de l'utilisateur et l'accumulateur sur le stack pour être stocké. Ensuite, le programme va imprimer la valeur ASCII associer a la commande LDD dans l'adressage 2011 (grâce a putchar). Par la suite, le programme PULL la valeur de B (la valeur choisis par l'utilisateur) afin qu'il puisse être imprimmer avec putchar. Après, le programme décrémente l'accumulateur et le loop continue (Justqu'à ce que la valeur de l'accumulateur = 0). Ensuite, la commande SWI est executer après le loop, et on retourne au moniteur.

partie2.c > partie2(int)

```c
#include<stdio.h>
#include <time.h>


void partie2(int countOfLoops){

    char b = ";" ;
    char userInput;
    char c;

    printf(b);
    scanf("%d", &userInput);
    int Top=-1, inp_array[2];

    int loopExit = 3;

    while (loopExit != 0);
    {

        c = b;
        b = " ";
        printf(b);

        b = c;
        printf(b);

        loopExit--;

    }




}

```

d. Changez le message guide « : » à « > » ainsi que le caractère de séparation d'un espace « » au point-virgule «;».

; = “$3B”, > = “$3E”

e. Changez le compteur de boucle pour imprimer exactement 15 fois le caractère tapé. Montrez le fonctionnement de votre programme à votre AE.
   - LDAA #$0F

f. Dans votre rapport, donnez votre nouveau code en vous servant du même tableau que dans b ci-dessus.

Tableau 1.1 : List Instructions, leurs adresse et leur contenu trouver lors de l’execution des instructions données dans le terminal du programme miniIDE et la modifications des valeurs associer aux intructions dans les adressage 2000, 200D et 2011

Adresse	Contenu	Instruction	Description
2000	CC003E	LDD #$3E	Charger code pour “>” dans B
2003	FEEE86	LDX $EE86	Chargez le vecteur pour le sous- programme putchar
2006	1500	JSR 0,X	Imprimer au terminal le contenu de B
2008	FEEE84	LDX $ EE84	Chargez le vecteur pour la routine getchar
200B	1500	JSR 0,X	Placez un nouveau caractère dans B
200D	860F	LDAA #$0F	Initialise le compteur de boucle
200F	36	PSHA	Sauvez le compteur sur la pile
2010	37	PSHB	Sauvez le contenu de B sur la pile
2011	CC003B	LDD #$20	Chargez B avec une ;

2014	FEEE86	LDX $EE86	Chargez le vecteur pour le sous-programme putchar
2017	1500	JSR 0,X	Imprimer au terminal
2019	33	PULB	Récupérer le caractère original
201A	FEEE86	LDX $EE86	Chargez le vecteur pour le sous-programme putchar
201D	1500	JSR 0,X	Imprimer au terminal
201F	32	PULA	Récupérer le compteur
2020	43	DECA	Décrémenter le compteur de boucle
2021	26EC	BNE $200F	Si compteur <> 0, répéter
2023	3F	SWI	Retourner au moniteur

Partie 2 – L'assembleur

Tableau 2.1 : Les instructions tracer lors de l'execution des étapes g) et h) dans la partie 2 du laboratoire

D						
A	B	X	Y	SP	PC	CCR
00	00	0000	0000	3C00	0400	1001 0000
00	00	0000	0000	2000	0403	1001 0000
00	00	0000	0000	1FFE	0427	1001 0000
00	00	0000	0000	1FFE	0429	1001 0000
00	00	0000	0000	1FFE	042C	1001 0000
0C	9C	0000	0000	1FFE	0450	1001 0000
0C	9C	0000	0000	1FFE	0456	1001 0000
0C	9C	0000	0000	1FFE	045C	1001 0000

h) Tracez deux autres instructions (elles devront être l'instruction MOVW). Quels registres changent? Quel mode d'adressage est utilisé par ces instructions. Vérifiez l'effet dans la mémoire avec la commande D-Bug12 pour examiner le contenu de mémoire (MD). Inscrivez vos observations dans votre rapport.
   - MOVW bouge une certaine data de un mémoire a un autre.

i) Dans votre rapport, décrivez comment vous avez trouvez le bogue et comment vous l'avez corrigé.

Nous avons trouvé la bogue majoritairement avec un peu de pensée critique et de l'essai et erreur. D'abord, lorsqu'on exécutait le code, nous avons observer que chaque valeur qui passait dans notre terminal armait l'alarme. Cela dit on a pu conclure que dans les conditions de notre partie ''if'' n'était pas fonctionnel. Nous avons pris le temps de comprendre chaque morceau de ce code et avec de la recherche nous avons trouvez que BEQ voulait dire ''branch if equal'' dans ce même article on a trouvé qu'il existait aussi BNE ''branch if not equal''. On savait que les conditions requises par notre code ne fonctionnaient pas et que toute les code alarmes fonctionnait plutôt que juste 0000. Donc, on a décider de changer BEQ à BNE et on a trouvez aussitôt avec la prochaine exécution de notre code qu'il fonctionnait après ce changement.

Partie 3 – Le développement du module Delay

```
; Stack Usage:
        OFFSET 0  ; to setup offset into stack
PDLY_VARSIZE:
PDLY_PR_Y    DS.W 1 ; preserve Y
PDLY_PR_X    DS.W 1 ; preserve X
PDLY_PR_B    DS.B 1 ; preserve B
PDLY_RA      DS.W 1 ; return address

polldelay: pshb
   pshx             ;on place x sur un stack
   pshy             ;on place y sur un stack

   ldx #10000       ;10 secondes

   ldy #delayCount  ;on place la valeur de delayCount dans y
   ldaa #FALSE      ;accumulator = FALSE


        ; Complete this routine

delayLoop:          ;loop de delay 1 ms commence ici
   nop
   nop
   nop
   nop
   dex
   bne delayLoop    ;on retourne pour un autre loop de 1 ms si condition n'est pas atteint

   ;si condition atteint on continue


   ldy delayCount   ;Ici on décrement delayCount. Si la condition de BNE est on pass a returnEtExit pour
   dey
   sty delayCount
   bne returnAndExit
   ldaa #True




;restore registers and stack
returnAndExit:
   puly             ;on pull les valeurs de y,x et b du stack
   pulx
   pulb
   rts              ;return du subroutine
```

Figure 3.1 : Image de notre code pour le sous-programme pollDelay

```
setDelay:

    ; Complete this subroutine
    std delayCount ;On met la valeur de delayCount dans l'accumulateur
    rts
```

Figure 3.2 : Image de notre code pour le sous programme setDelay

Conculsion :

Dans ce laboratoire, nous avons analyser et modifier un programme en code assembleur, déboger un programme en code assembler et créer un sous-programme en assembleur. Chaque composant de ce lab a été un succès comme nous avons pu démontrer au TA chaque aspect du lab sans problem. Deplus, nous avons approfondi nos connaisance sur le code assembleur.

Difficultés/Défis :

La plus grande difficulté de ce laboratoire était d'apprendre ce que faisait chaque ligne du code assembleur, puisque ce lab était la première fois qu'on a travailler en assembleur.