École de Génie Électrique et Informatique
Université d'Ottawa

uOttawa

CEG 3536– Architecture des ordinateurs II
Automne 2023

Laboratoire 1 :

Introduction à la programmation de microprocesseur

Soumis par :
Elam Olame **Mugabo** 300239792
Decaho **Gbegbe** 300094197

Professeur : Mohamed Ali **Ibrahim**, PhD.ing
Assistant à l'Enseignement : Joel **Simweray**

Date de soumission : 7 octobre 2023

1. **INTRODUCTION :**

   Le microprocesseur est la pièce principale d'un ordinateur. C'est un processeur miniaturisé qui tient dans un seul circuit intégré. Il gère l'exécution des instructions de l'ordinateur. Son langage natif est la langage machine ou code machine.

   Le langage machine est le seul compréhensible par la machine mais est peu ergonomique pour un être humain. D'où, il existe le langage assembleur qui est le langage de plus bas niveau qui représente le langage machine sous une forme lisible par un humain.

2. **OBJECTIFS :**

- Devenir familier avec le moniteur Debug-12.
- Introduire la programmation d'assembleur.
- Assembler et exécuter un logiciel assembleur.
- Apprendre les principes de base du débogage.
- Appliquez les différents modes d'adressages.
- Concevoir et implémenter un module de logiciel.

3. **COMPOSANTES DU LABORATOIRE:**

- Windows PC
- Carte entraîneur Dragon 12 Plus

4. EXPERIENCE :

A. Partie 1 - Utiliser le moniteur D-Bug-12

Cette section consistait à assembler un programme dans le moniteur D-bug12 a une adresse spécifique : $ 2000. On a donc entré les instructions ligne par ligne à cette adresse après avoir entré dans le terminal la commande, “asm $2000”, qui permet de stocker des instructions à des adresses spécifiques.

Voici la simulation de notre travail :

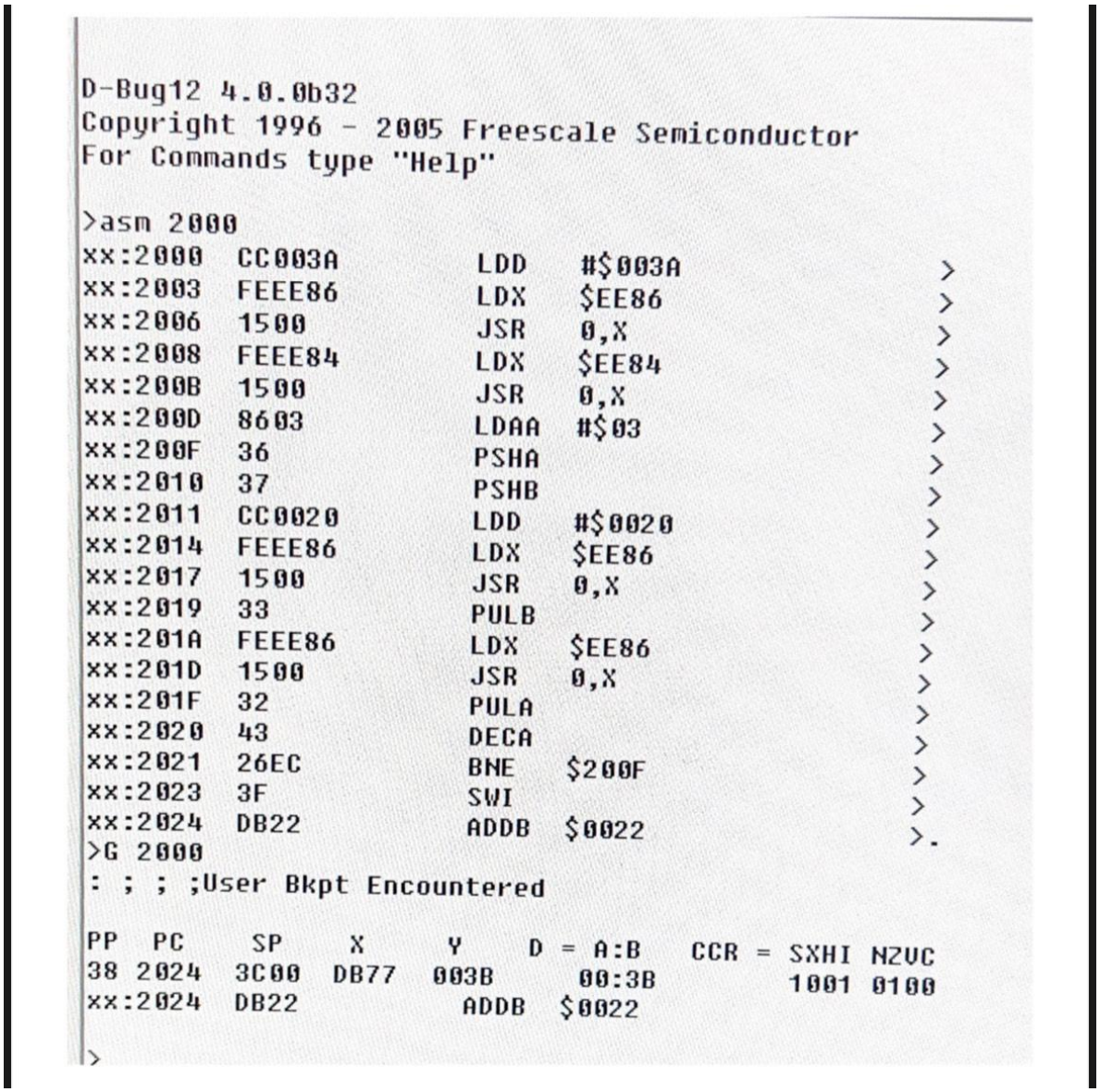


Figure 1 : Simulation sur Debug avant modification du code

Adresse	**Contenu**	**Instruction**	**Description**
2000	C63A	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

Tableau 1 : Code avant la modification de certaines commandes

Après cela, il nous a été demandé de modifier certaines commandes dont :

- Changer le message guide « : » à « > » ainsi que le caractère de séparation d'un espace « » au point-virgule «;».
- Changer le compteur de boucle pour imprimer exactement 15 fois le caractère tapé.

Voici ci-dessous la simulation qui en résulte :

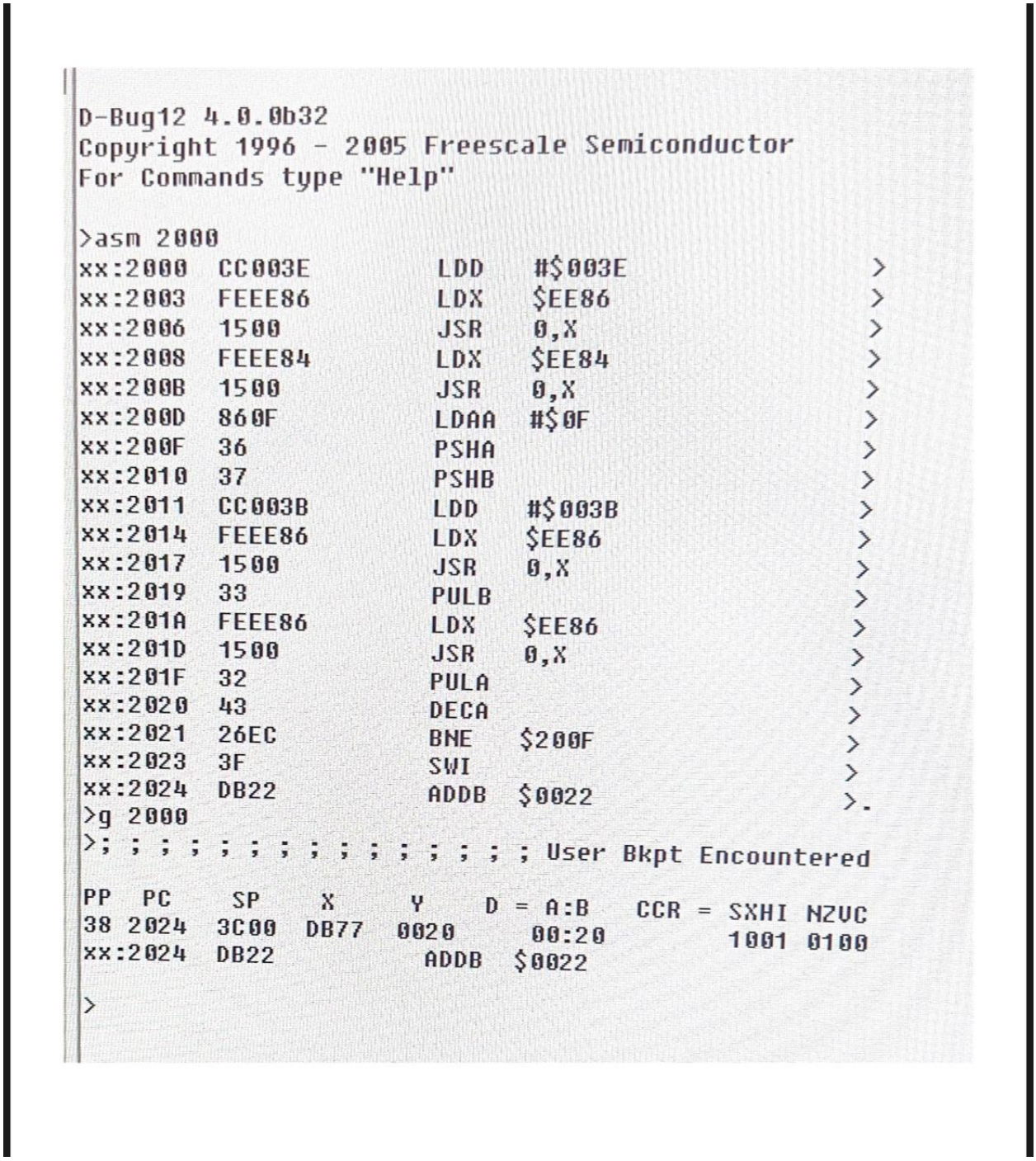


Figure 2 : Simulation sur Debug après modification du code

Adresse	**Contenu**	**Instruction**	**Description**
2000	C63A	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 #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	CC003B	LDD #$3B	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

Tableau 2 : Code après la modification de certaines commandes

Il y avait quelques petites questions auxquelles nous devions répondre dont :

b. Mode d'adressage utilisé par chaque instruction:

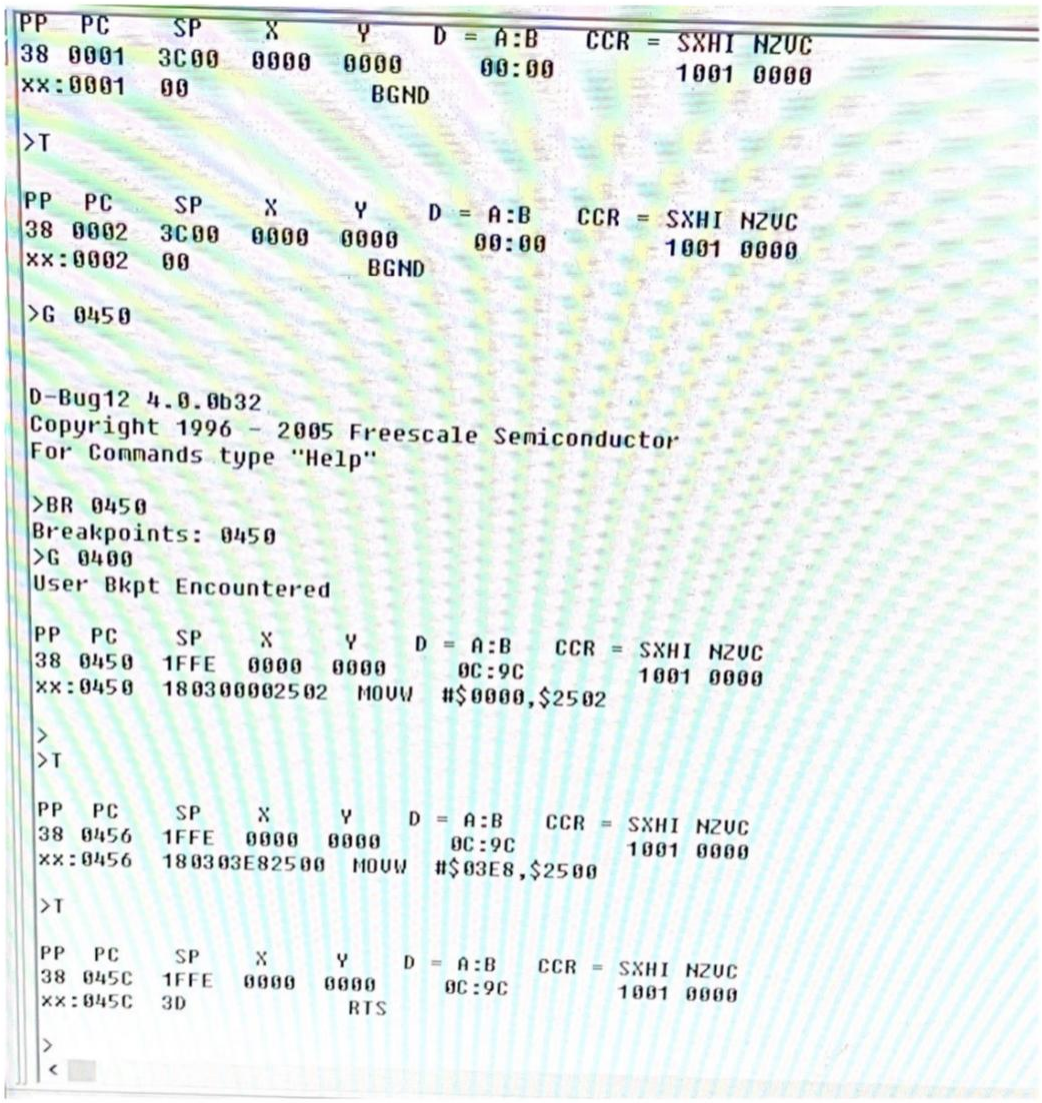
- LDX $EE86 est le mode d'adressage direct avec comme opérande et Op code ci dessous:
  - Opérande: EE86
  - OP code: FE
  - Le contenu est alors FEEE86
- JSR 0,X consiste à l'adressage indexé.
  - L'opérande est: 00
  - L'OPCode est: 15
  - Le contenu est donc 1500.
- BNE $200F consiste à l'adressage relatif avec:
  - Operande: EC
  - Opcode: 26

c. Explication Avec fonction C:

```
#include <stdlib.h>
#include <stdio.h>
void main(){
```

```
char k = getchar(); /* declaration du caractere K comme variable demander une entrée de l'utilisateur */

printf("\n"); // mettre le caractère espace
while(k != EOF) // condition quand K est différent de EOF est repeter
{
putchar(k); // afficher le caractere k
k = getchar();
}
printf("End Of Program"); // autrement afficher fin de program
}
```

Partie II - L’assembleur



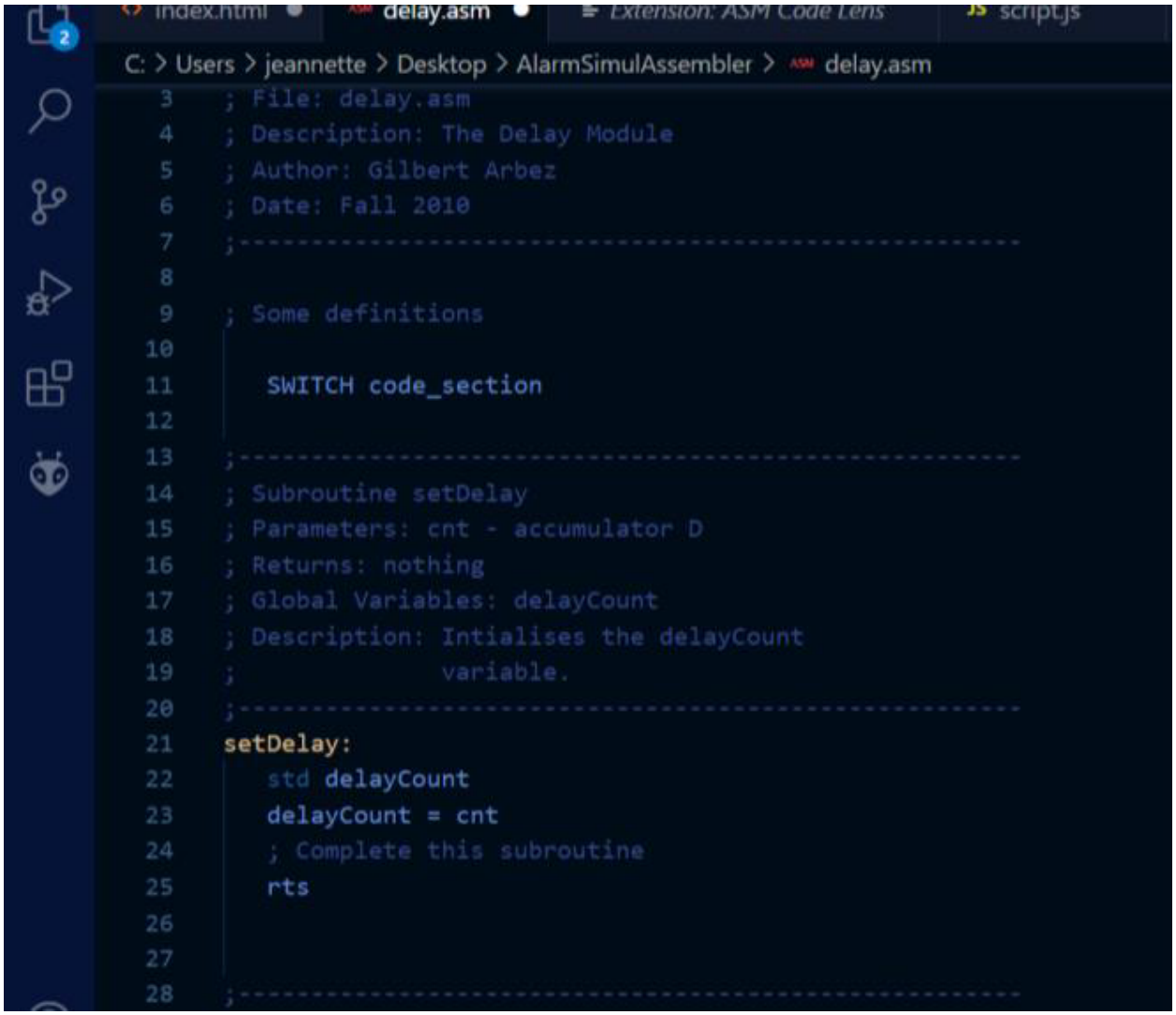
D						
A	**B**	**X**	**Y**	**SP**	**PC**	**CCR**
00	00	0000	0000	3c00	0400	1001 0000
00	00	0000	0000	2000	0403	1001 0000
00	00	0000	0000	1FFE	0427	1001 0000
00	00	0000	0000	1FFE	0429	1001 0000
Le Breakpoint (BR 0450)						
0C	9C	0000	0000	1FFE	0450	1001 0000
0C	9C	0000	0000	1FFE	0456	1001 0000
0C	9C	0000	0000	1FFE	045C	1001 0000
0C	9C	0000	0000	2000	0405	1001 0000
07	9B	0000	0000	2000	0408	1001 0000

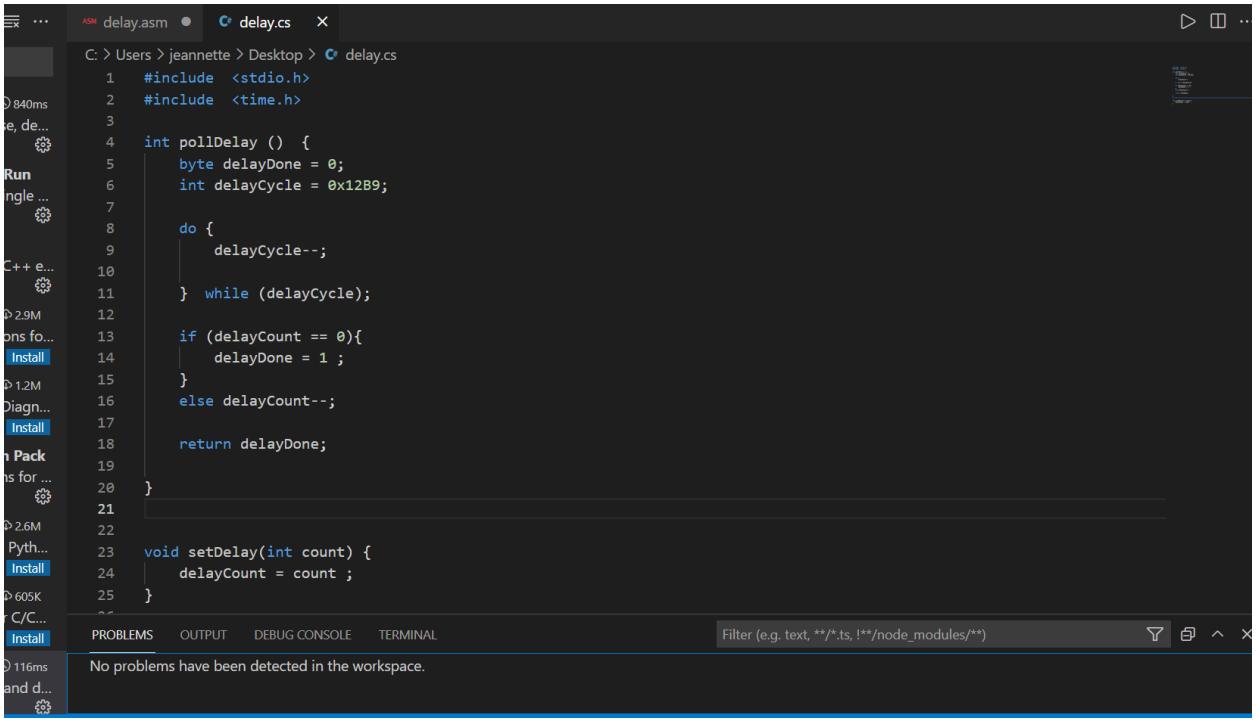
- Au niveau du **breakpoint** le registre qui changent est PC, c'est un mode d'adressage immédiat qui est utilisé par les instructions.
- La commande "rts" permet de dépiler la valeur du compteur de programme de la pile. La ramenant ainsi à la boucle principale du code.
- L'adresse 045c est l'adressage de "rts".
- on a remarqué une instruction erronée dans isCodeValid (beq) qui n'existe pas. Nous avons remplacé cette instruction par "bne" car il est utilisé pour actionner le début ou la fin d'une condition ou boucle.

```
; Subroutine: isCodeValid
; Parameters: alarmCode stored in register D
; Local Variables
;   ptr - pointer to array - in register X
;   cnt, retval - on the stack.
; Returns: TRUE/FALSE - Returned in accumulator A
; Description: Checks to see if alarm code is in the
;              alarmCodes array.
;-------------------------------------------------------

; Stack usage
        OFFSET 0
CDV_ALARMCODE   DS.W 1 ; alarmCode
CDV_CNT     DS.B 1 ; cnt
CDV_RETVAL DS.B 1 ; retval
CDV_VARSIZE:
CDV_PR_X    DS.W 1 ; preserve x register
CDV_RA      DS.W 1 ; return address

isCodeValid: pshx
    leas -CDV_VARSIZE,SP
    std CDV_ALARMCODE,SP
    ; int *ptr;  // pointer to alarmCodes
    movb #NUMCODES,CDV_CNT,SP ; byte cnt = 5;
    movb #FALSE,CDV_RETVAL,SP ; byte retval = FALSE;
    ldx #alarmCodes           ; ptr = alarmCodes;
cdv_while                     ; do
    ldd 2,X+                  ; {
    cpd CDV_ALARMCODE,SP      ;    if(*ptr++ == alarmCode)
    bne cdv_endif             ;    {
    movb #TRUE,CDV_RETVAL,SP  ;      retval = TRUE;
    bra cdv_endwhile          ;      break;
cdv_endif:                    ;    }
    dec CDV_CNT,SP            ;  cnt--;
    bne cdv_while             ; } while(cnt != 0);
cdv_endwhile:
    ldaa CDV_RETVAL,SP        ;  return(retval);
    ; restore registers and stack
    leas CDV_VARSIZE,SP
    pulx
    rts
```

Partie III - Développement du module Delay

Dans cette troisième section, nous mettons en œuvre le module de retard. L'objectif est de créer un délai en multiples de 1 ms. Pour ce faire, nous devons suivre un pseudocode tel qu'indiqué ci-dessous. Le programme invoque une fonction qui détermine la durée du retard. La valeur diminue progressivement jusqu'à atteindre 0.





Conclusion :

En guise conclusion, nous sommes heureux d'avoir fait ce laboratoire parce que les objectifs fixés ont pu être atteints étant donné que tous les théorèmes et principes en lien avec ce laboratoire ont été démontrés. Cette expérience nous a permis d'en connaître un peu plus sur le fonctionnement du moniteur Debug-12, nous avons pu assembler et exécuter un logiciel assembleur. Nous avons eu une connaissance sur les principes de base du débogage et nous avons appliqué les différents modes d'adressages. Pour finir, nous avons pu concevoir et implémenter un module de logiciel.