



MINI PROJET
REALISATION D'UN CALCULATRICE

Section : Génie électronique

Groupe : IE4

Elaboré par :

Tarek Bouchkati

Marwa Said

SOMMAIRE

INTRODUCTION GENERALE.....	1
----------------------------	---

CHAPITRE I : Aspect théorique de la calculatrice

I.INTRODUCTION.....	2
II.CAHIER DE CHARGE.....	2
III.LES OUTILS DE TRAVAIL.....	3
III.1.Les instructions de base.....	4
III.1.1.De base et d'affectation.....	4
III.1.2.Arithmétique et logique.....	5
III.1.3.Décalage et rotation.....	5
IV.PRTOTYPE DE LA CALCULATRICE.....	6
V.CONCLUSION.....	7

CHAPITRE II : Etude fonctionnel de la calculatrice

I.INTRODUCTION.....	8
II.LES PROCEDURES D'ECRITURE ET DE LECTURE.....	8
II.1.La procédure ScanInt.....	8
II.2.La procédure ScanHex.....	10
II.3.La procédure PrintInt.....	11
III.LES PROCEDURES ARITHMETIQUES.....	12
III.1.La procédure addition.....	12
III.2.La procédure soustraction.....	13
III.3.La procédure de multiplication.....	14
III.4.La procédure de division.....	15
III.5.La procédure de PGCD.....	16

III.6.La procédure prog_pgcd.....	17
III.7.La procédure PPCM.....	18
III.8.La procédure puissance	19
IV.CONCLUSION.....	19

CHAPITRE III : La programmation en langage assembleur de la calculatrice

I.INTRODUCTION.....	20
II.LES CODES SOURCES.....	20
III.LES TRACES D'EXECUTION.....	27
IV.CONCLUSION.....	28
PERSPECTIVES.....	29

INTRODUCTION GENERALE

Un microprocesseur est un objet capable de traiter, de stocker et de restituer de l'information. C'est la partie centrale qui permet le traitement de l'information. Ce circuit intégré programmable met en évidence une architecture bien définie qui englobe une unité arithmétique et logique, une unité de contrôle, des registres il communique avec son environnement via trois bus : données, adresses et commande.

L'objectif de notre séance de cours micro processeur est de comprendre à travers le microprocesseur 8086 le fonctionnement général de ces derniers et d'avoir les outils de programmation nécessaire en langage assembleur.

Après avoir se familiariser avec le langage de programmation on a eu recours à aboutir les objectifs de notre séance de cours et réaliser une calculatrice programmer en langage à base du micro-processeur 8086.

Dans le premier chapitre nous présentons le cahier de charge du travail demandé ainsi que les outils nécessaires à la réalisation de notre travail désirée.

Dans le deuxième chapitre on met en œuvre les différents organigrammes de chaque procédure afin de faciliter la tâche de programmation.

Finalement on aboutit au listing des programmes utilisés et présenté des exemples de trace d'exécution.

CHAPITRE I
ASPECT THEORIQUE DE LA
CALCULATRICE

I.INTRODUCTION :

Dans ce chapitre nous allons intéresser de définir le cahier de charge demandés pour la réalisation de la calculatrice.

II.CAHIER DE CHARGE :

Il s'agit de réaliser un calculatrice qui permet de faire l'addition, la soustraction, la multiplication et la division, de plus on a choisi de calculer le plus grand et le plus petit multiple commun de deux nombres données (PGCD et PPCM). Considérons que les données introduites ne doivent pas dépasser les 4 digits hexadécimal.

Pour cela il faudra faire :

- Afficher un menu principal permettant de choisir le type d'opération comme le montre la figure suivant :

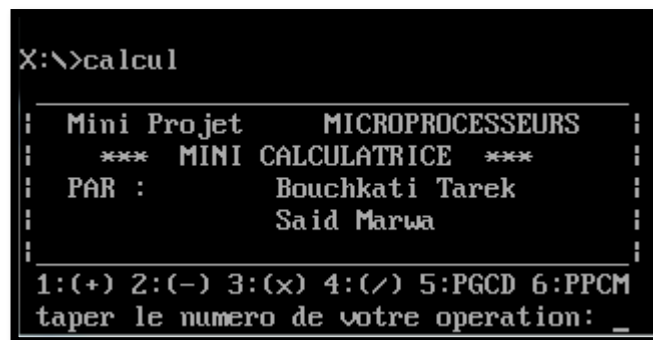


FIGURE1 : Photo du menu de la calculatrice

- Introduire un caractère correspond à l'opération désirée.
- Effectuer l'opération et afficher le résultat.

```

X:\>calcul

: Mini Projet      MICROPROCESSEURS :
: *** MINI CALCULATRICE ***         :
: PAR :      Bouchkati Tarek        :
:      Said Marwa                   :
:                                   :
1:(+) 2:(-) 3:(x) 4:(/) 5:PGCD 6:PPCM
taper le numero de votre operation: 1
  n1 = 0001
  n2 = 0003
  n1 + n2 = 00000004
> quitter (y/n)? _

```

FIGURE2 : Photo de l'addition de deux nombres quelconques

III. Les outils de travail :

Pour pouvoir réaliser notre projet de calculatrice on a utilisé le compilateur TASM, il présente un acronyme du turbo assembler compilateur. C'est un assembleur de la famille $\times 86$. Le programme que l'on désire traduire en langage machine (assembler) doit être placé dans un fichier texte avec l'extension **.ASM**.

Les fichiers .EXE ou .COM sont directement exécutables. Un utilitaire spécial (chargeur) du système d'exploitation (Ex: DOS), est responsable de la lecture du fichier exécutable, de son implantation en mémoire principale, puis du lancement du programme.

Les programmes .EXE sont limités que par la mémoire disponible dans l'ordinateur contrairement aux programmes .COM qui ne peuvent pas utiliser plus d'un segment dans la mémoire. Leur taille est ainsi limitée à 64 Ko.


III.1.Les instructions de base :

Il existe plusieurs catégories d'instruction: les instructions de base et d'affectation, les instructions arithmétiques et logiques, les instructions de décalage et de rotation, les instructions de comparaison et les instructions de rupture de séquence.

III.1.1.De base et d'affectation :

Les instructions d'affectation permettent de faire des transferts de données entre les registres et la mémoire.

 Chargement d'une valeur dans un registre

 Déplacement d'une valeur depuis un emplacement mémoire dans un registre, et inversement.

Le tableau ci -après résume les différents instructions de base utiliser pour programmer notre calculatrice en langage assembleur :

USAGE	INSTRUCTIONS	Rôle
Générale	-MOV	-transfert d'un ou plusieurs octets.
	-PUSH	-chargement de la pile.
	-POP	-Déchargement de la pile.
	-XCHG	-Echanger un ou plusieurs octets.
	-XLAT	-Translation d'octet.

III.1.2.Arithmétiques et logique :

La réalisation de la calculatrice est basée sur l'utilisation des instructions de division de multiplication d'addition et de soustraction des valeurs de deux registres ou mémoire et chargement du résultat dans un registre ou mémoire. Ces instructions sont présentées dans le tableau suivant :

USAGE	INSTRUCTION	RÔLE
Addition	-ADD -ADC -INC	-Addition. -Addition avec retenue. -Incrémentement de +1.
Soustraction	-SUB -DEC	-soustraction. -Décrémentement de -1.
Multiplication	-MUL	-Multiplication d'un ou plusieurs octets non signé.
Division	-DIV	-Division d'un ou plusieurs octets non signés.

III.1.3.Déclages et rotation :

De plus on a utilisé les instructions de décalages et de rotation suivantes :

USAGE	INSTRUCTIONS	RÔLE
Décalages	-SHL -SAL	-Décalage logique à gauche. -Décalage arithmétique à gauche.
Rotation	-ROR -RCR -RCL	-Rotation à droite. -Rotation à gauche avec bit de retenue. -Rotation à droite avec bit de retenue.

On a aussi bien employées dans notre programme les instructions de comparaison et de conditions, de lecture et d'écriture afin de pouvoir gérer notre programme d'une manière adéquate et faciliter le travail.

IV. Prototype de la calculatrice :

L'idée de la réalisation de notre calculatrice est assez simple il consiste à faire entrer deux opérateurs et un opérande de notre choix.

L'organigramme suivant illustre l'idée générale de travail.

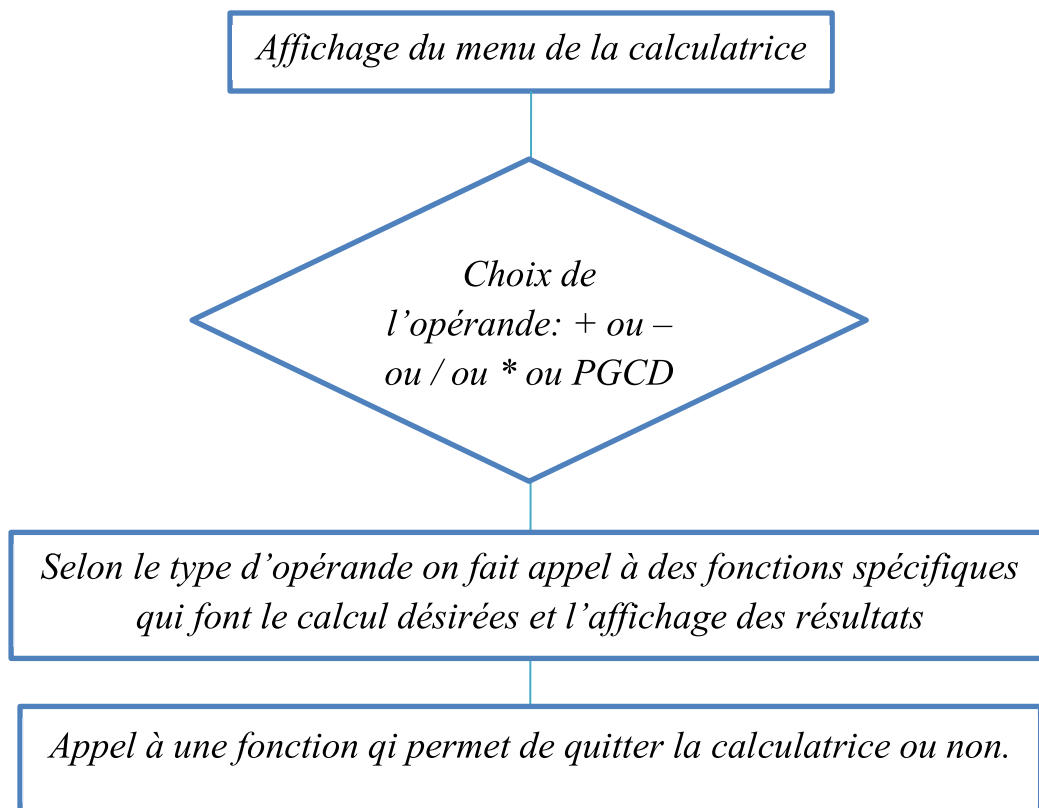


FIGURE 3 : Organigramme généralisé de la calculatrice

V. CONCLUSION :

D'après l'étude de l'aspect théorique de la réalisation de la calculatrice, on a pu avoir les outils nécessaires pour réaliser le travail.

CHAPITRE II

ETUDE FONCTIONNEL DE LA

CALCULATRICE

I.INTRODUCTION :

Dans ce chapitre nous allons découvrir les différentes fonctions utilisées pour la réalisation pratique de la calculatrice ainsi que l'explication de chacune de ces fonctions afin de mieux comprendre l'enchaînement du travail réalisé et banalisé la plus possible le langage de programmation.

II.LES PROCEDURE D'ECRITURE ET DE LECTURE :

Le travail demandé consiste en premier lieu d'afficher le menu de calculatrice afin de pouvoir choisir le type d'opération et saisir les numéros entrés au clavier puis afficher directement les résultats sur l'écran. Ce travail paraît simple pour l'utilisateur mais en langage de programmation il est primordial de décrire à la machine les instructions à faire.

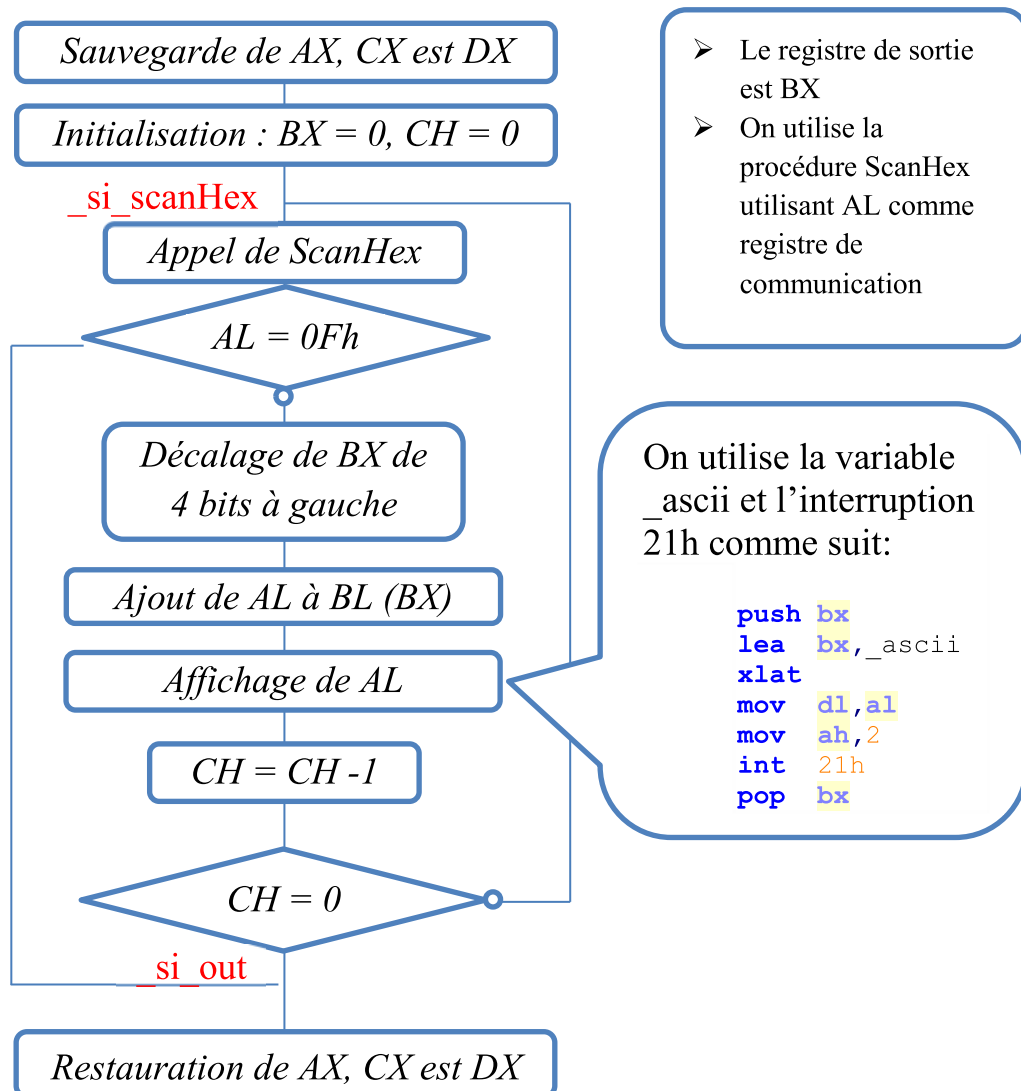
Pour ceci il est nécessaire que la machine comprenne quelle doit écrire un message ou saisir un nombre.

Ce travail est fait via deux fonctions qu'on crée qui sont : **scanInt** et **printInt**.

II.1.La procédure scanInt :

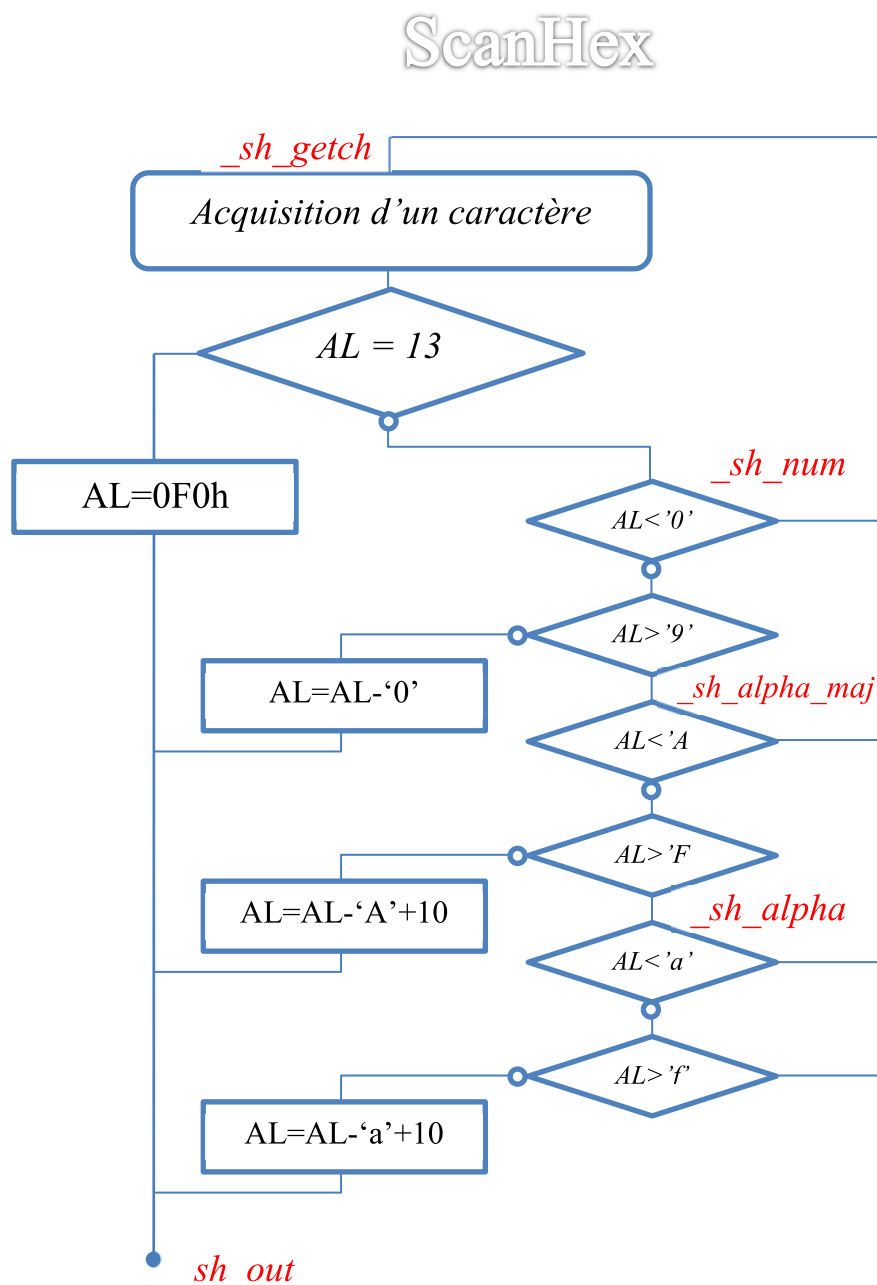
La fonction de lecture scanInt consiste à lire les données entrées par le clavier.

ScanInt

**FIGURE 4 : Organigramme de la fonction ScanInt**

II.2. La procédure ScanHex :

La fonction scanInt fait appel à une fonction autre ScanHex qui assure la conversion ascii hexadécimale des caractères saisie. L'organigramme suivant décrit le principe de cette fonction.



- Le résultat est placé dans le registre AL.
- Utilisation de l'interruption 8 pour l'acquisition d'un caractère sans écho

FIGURE 5 : Organigramme de la procédure ScanHex

II.3. La procédure PrintInt :

Afin de pouvoir afficher les résultats sur l'écran on a créé une procédure printInt qui permet d'écrire un mot en hexadécimale. L'organigramme ci-après détaille le principe de fonctionnement de cette procédure:

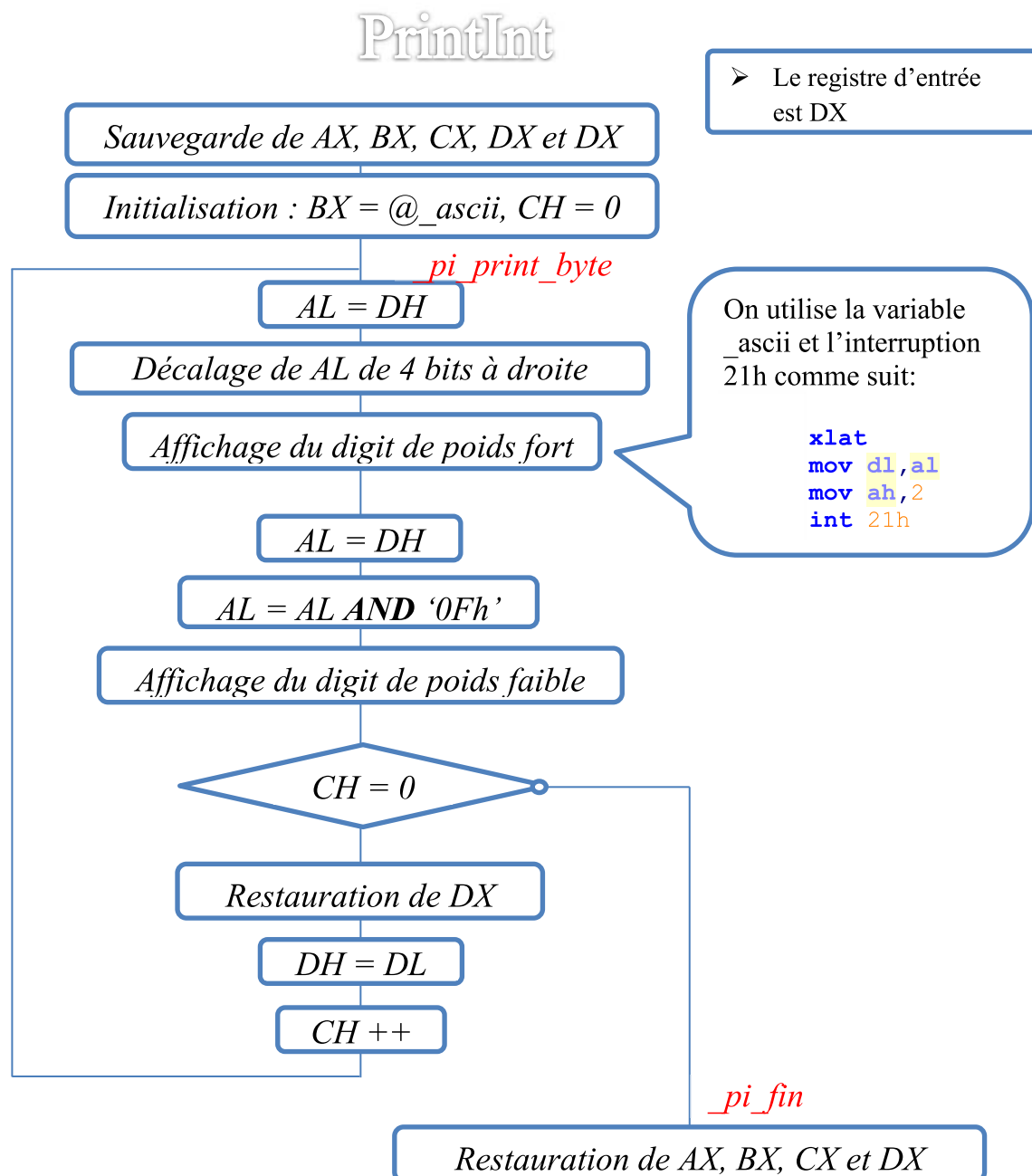


FIGURE 6 : Organigramme de la procédure PrintInt

III.LES PROCEDURE ARETHMITIQUE :

La calculatrice est basée sur l'utilisation des opérations arithmétiques telles que l'addition la soustraction, la multiplication et la division.

III.1.Laprocedure addition :

L'addition est une simple opération arithmétique qui peut être modélisé par l'organigramme suivant :

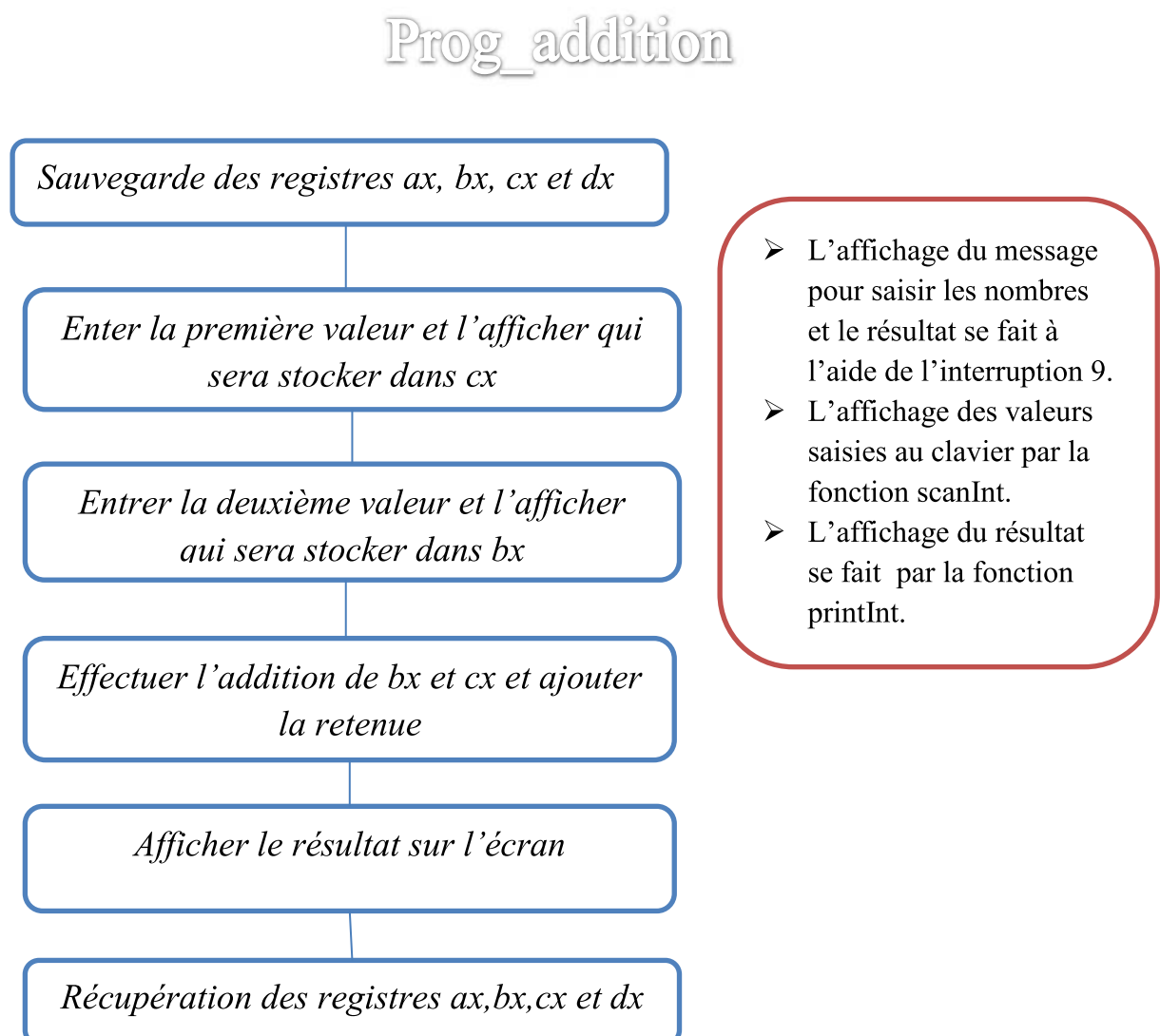


FIGURE 7 : Organigramme de la procédure addition

III.2.La procédure soustraction :

La soustraction est l'opération qui calcule la différence entre deux nombres.

Ces nombres saisis par le clavier seront stocker dans des registres bien défini et on peut les soustraire à l'aide d'un ensemble d'instructions bien définie qu'on peut les décrire par l'organigramme suivant :

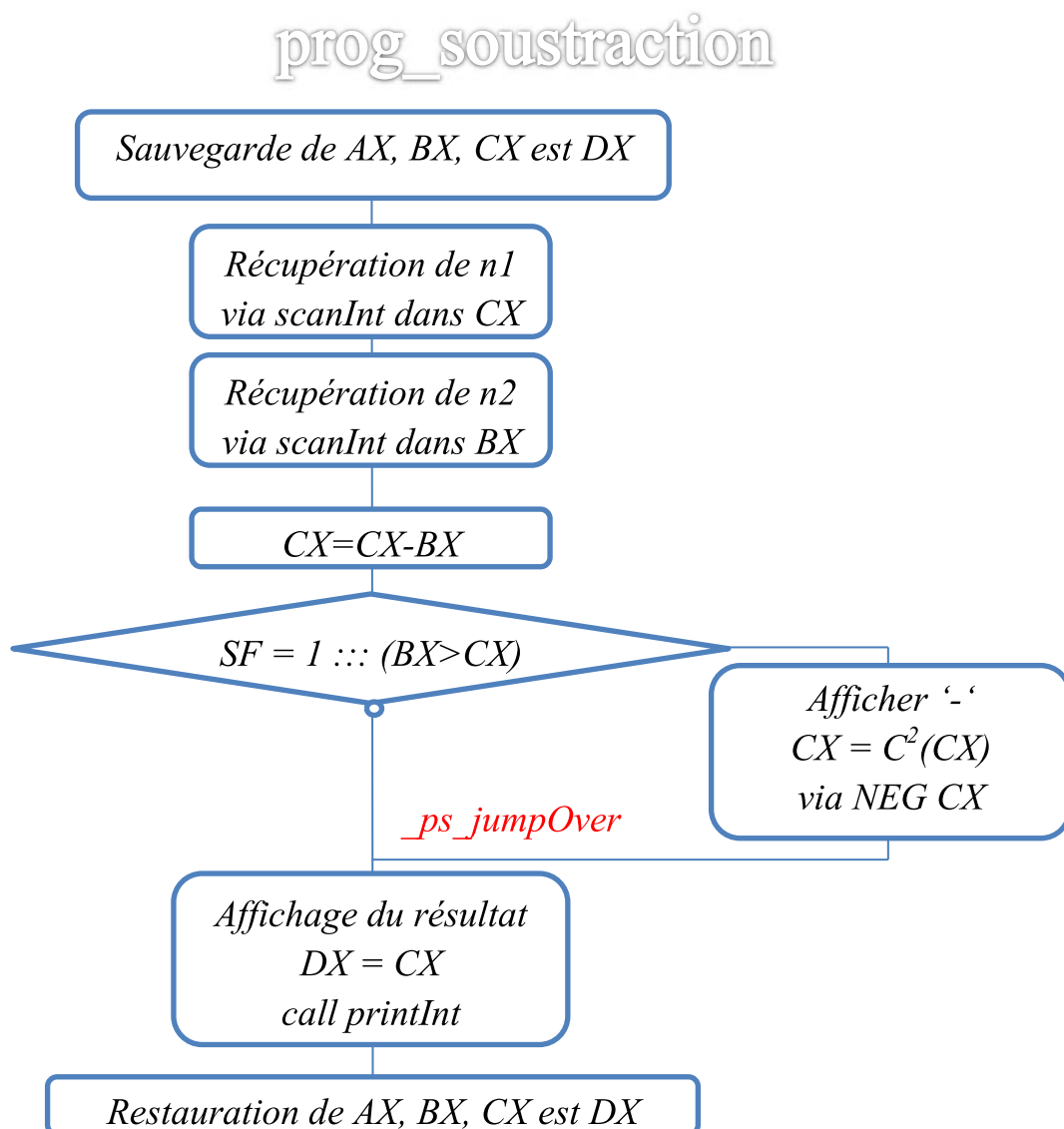


FIGURE 8 : Organigramme de la procédure soustraction

III.3.La procédure de multiplication :

La multiplication est l'une des opérations élémentaires de base qui permet de calculer le produit de deux nombres entré par l'utilisateur. On a procédé cette opération de la façon suivante :

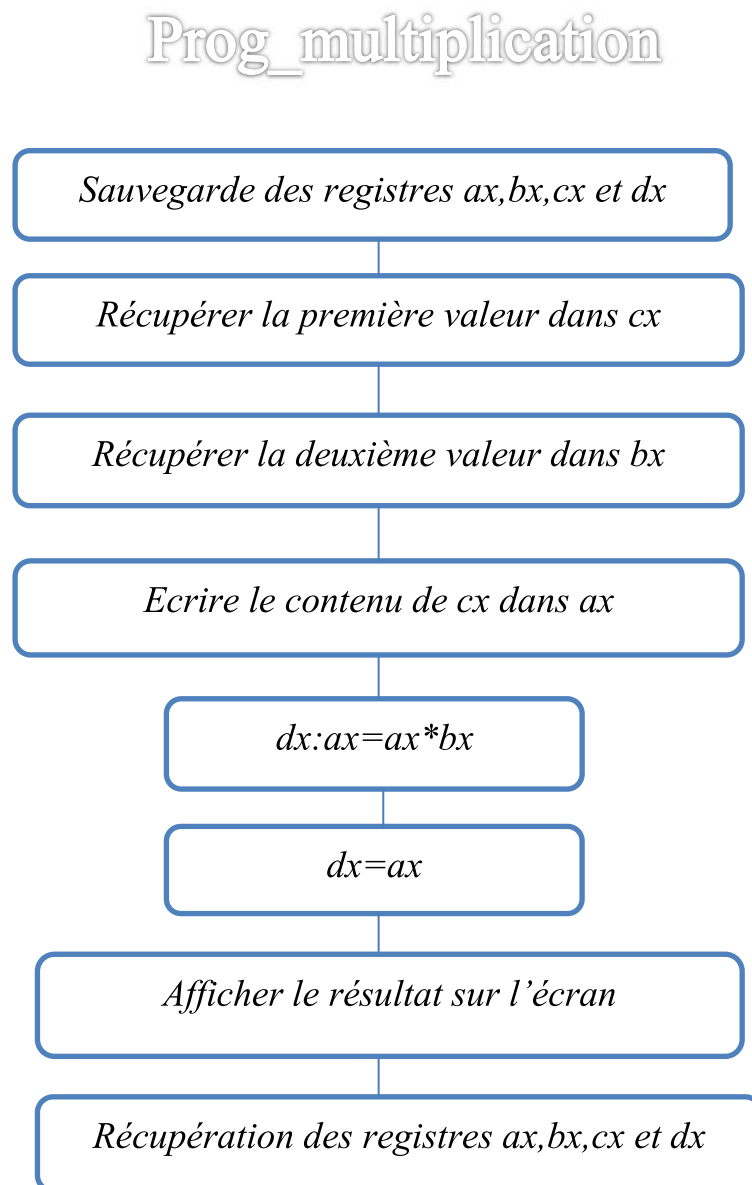


FIGURE 9 : Organigramme de la procédure multiplication

III.4. La procédure de la division :

La division est une opération qui à deux entiers naturels appelés dividende et diviseur, associe deux autres entiers appelés quotient et reste. En langage de programmation on stocke les valeurs dans des registres spécifiés et on utilise des instructions bien définies.

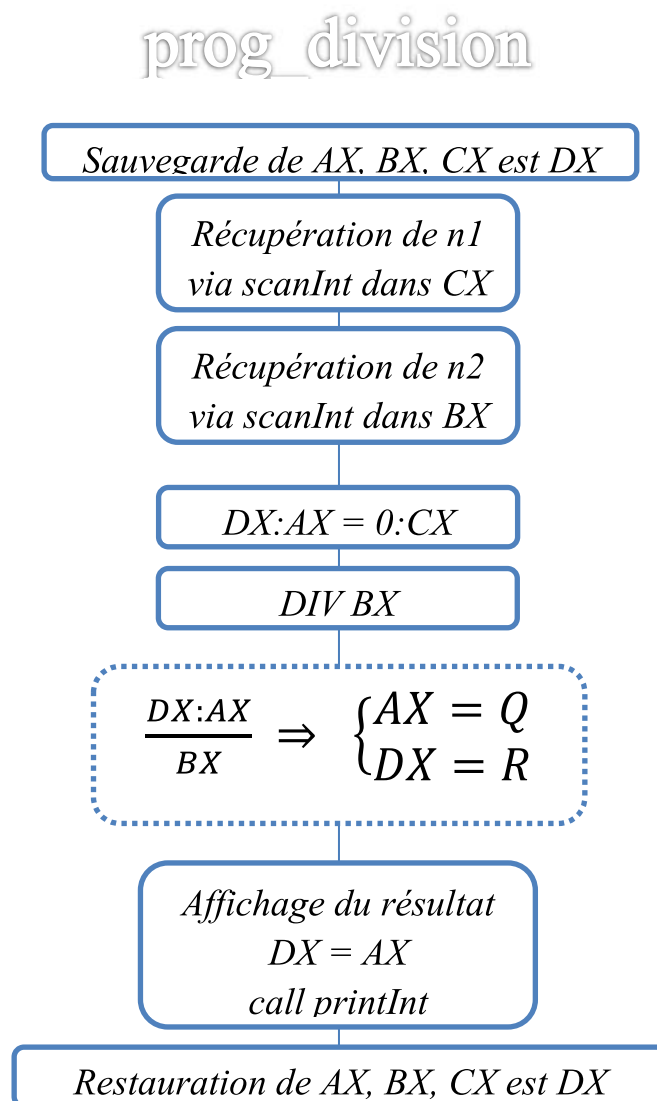


FIGURE 10 : Organigramme de la procédure de division

III.5.La procédure de PGCD :

Le pgcd ou comme son nom indique est le plus grand diviseur commun, consiste à calculer le plus grand diviseur des deux entiers naturels qui divise simultanément ces deux entiers. Pour pouvoir écrire un programme en assembleur qui assure ce calcul on a procédé par la méthode d'Euclide expliquer ci-après :

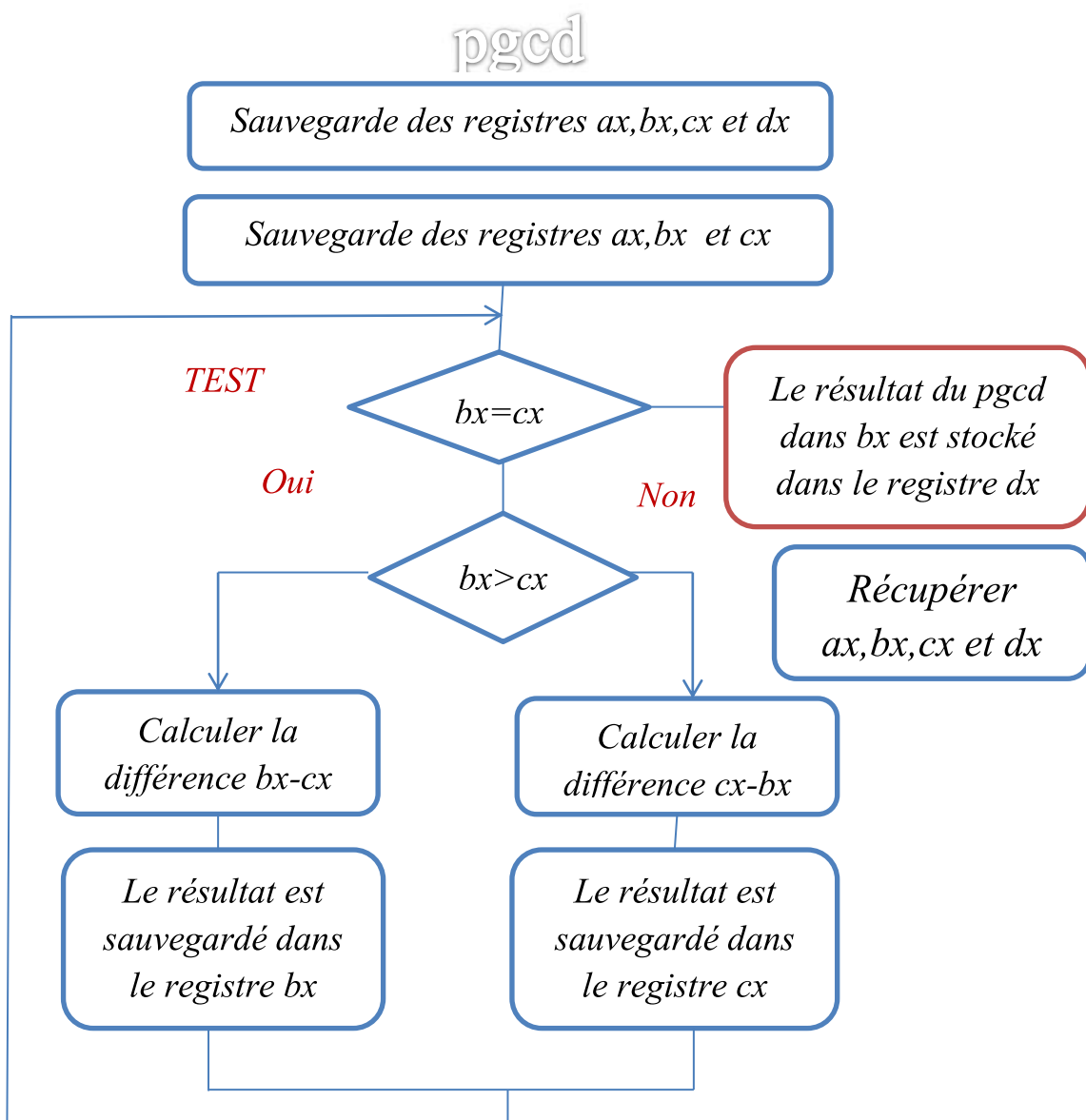
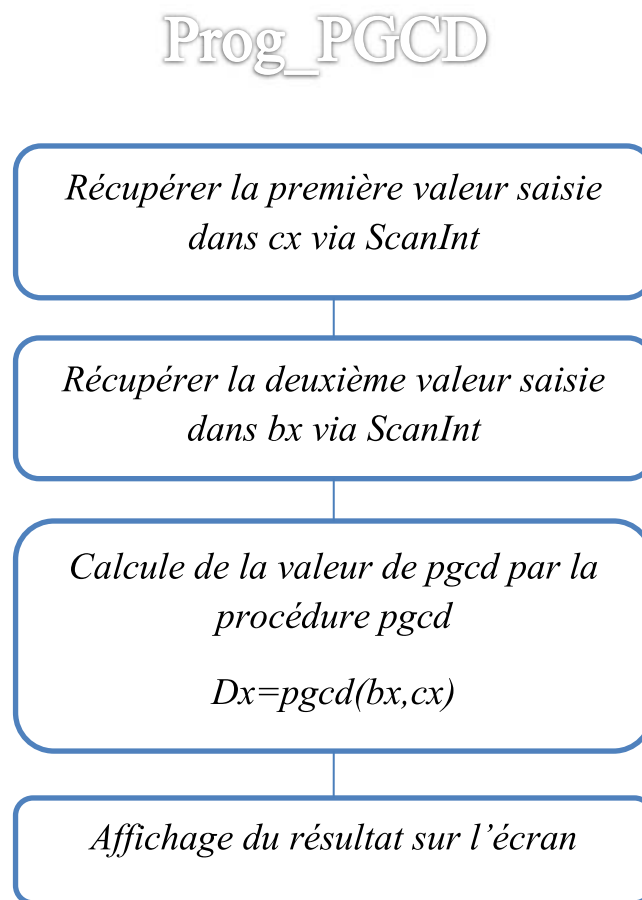


Figure 11: Organigramme permettant de calculer le PGCD selon méthode d'Euclide

III.6.La procédure prog_pgcd :

**FIGURE 12 : Organigramme de la procédure prog_pgcd**

III.7.Laprocédue PPCM :

Le PPCM c'est le plus petit multiple commun qui consiste à donner le plus petits diviseur deux entiers naturels.

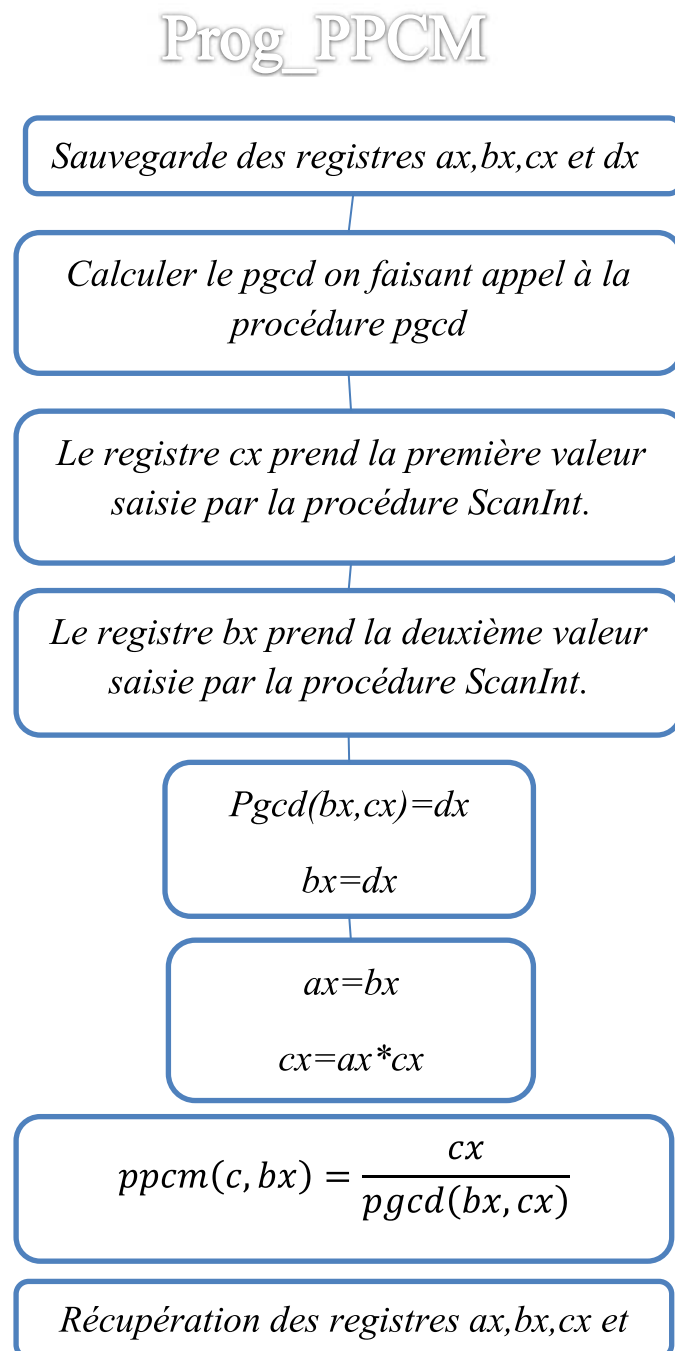


FIGURE 13 : Organigramme de la procédure PPCM

III.8.La procédure puissance :

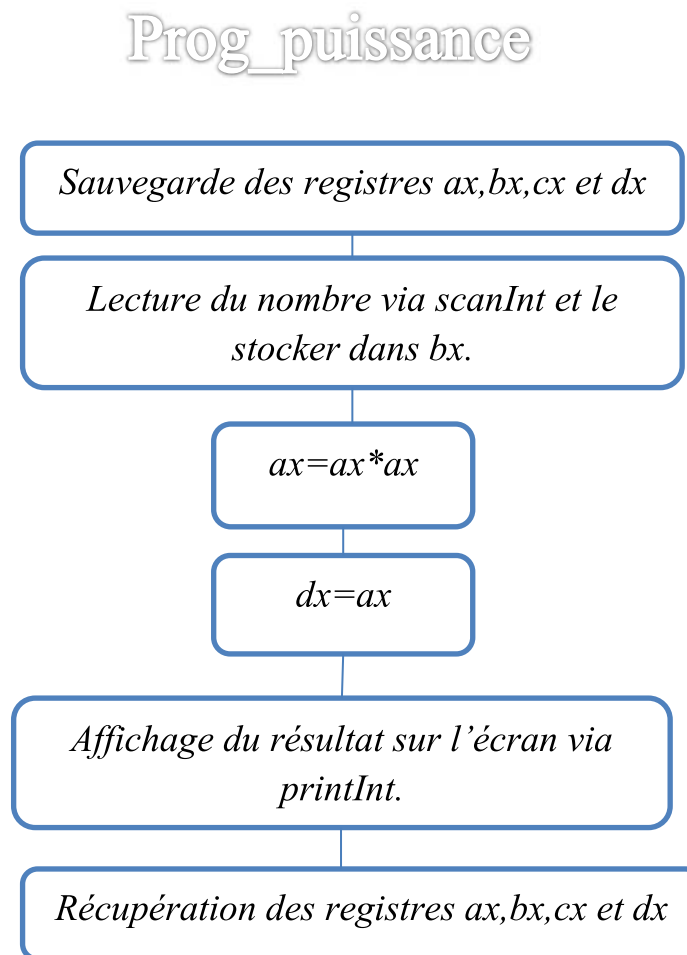


FIGURE14 : Organigramme de la procédure puissance

IV.CONCLUSION :

Ce chapitre nous à introduire à la partie programmation en donnant une explication sur les idées de chaque fonction à exploiter dans le programme par l'intermédiaire des organigrammes déjà présenté précédemment.

CHAPITRE III

LA PORGRAMMTION EN LANGAGE ASSEMBLEUR DE LA CALCULATRICE

I.INTRODUCTION :

Finalement on a achevé la partie la plus importante dans notre projet qui s'articule autour l'implémentation pratique de tout ce qu'on avait vu précédemment afin de réussir à construire un programme fonctionnel et exécutable en utilisant les outils de langages de programmation assembleur .Le chapitre comme son nom indique il contient les codes programmes de chacune des procédures décrit par le chapitre II en détaillant le langages de programmation utilisé.

II.LES CODES SOURCES :

La procédure« wrLn »

wrLn proc near	⇒ La procédure« wrLn » permet d'effectuer un saut à la ligne
push ax	
push dx	
mov dl,10	⇒ 10 = code ascii du retour à la ligne
mov ah,2	
int 21h	
pop dx	
pop ax	
ret	
wrLn endp	

La procédure « scanHex »

scanHex proc near	⇒ « scanHex » retourne dans AL la valeur du caractèrehexadécimal ou F0h
assume cs:code	⇒ dans le cas de la touche entrée
_sh_getch:	⇒ section de la saisie
mov ah,8	⇒ utilisation de l'interruption 8 pour l'acquisition d'un caractère sans écho
int 21h	
cmp al,13	⇒ comparer AL à la valeur du retour chariot (13)
jne _sh_num	⇒ si AL != 13 en effectue un saut au test numérique
mov al,0F0h	⇒ si AL = 13 on affecte 0Fh à AL
jmp _sh_out	On saute à la fin de la procédure
_sh_num:	⇒ section du test numérique
cmp al,'0'	⇒ On compare AL à '0' = 30h
jb _sh_getch	⇒ Si AL < '0' on retourne à la section de saisie
cmp al,'9'	⇒ On compare AL à '9' = 39h
ja _sh_alpha_maj	⇒ Si AL > '9' on passe à l'étape du test du majuscule
sub al,'0'	⇒ Nous sommes dans l'intervalle '0' ... '9'.Alors on retranche de AL le code ascii de '0'et on saute à la fin de la procédure
jmp _sh_out	
_sh_alpha_maj:	⇒ Section du test Majuscule
cmp al,'A'	⇒ On compare AL à 'A' = 41h
jb _sh_getch	⇒ Si AL < 'A' on retourne à la section de saisie
cmp al,'F'	⇒ On compare Al à 'F' = 46h
ja _sh_alpha	⇒ Si AL > 'F' on passe à l'étape de test du minuscule

```

sub al,'A'-10
jmp _sh_out
_sh_alpha:
cmp al,'a'
jb _sh_getch
cmp al,'f'
ja _sh_getch
sub al,'a'-10
jmp _sh_out
_sh_out:
ret
scanHex endp

```

- ⇒ Nous sommes dans l'intervalle 'A'...'F'. Alors on retranche de AL le code ascii de 'A' et on rajoute +10 et on saute à la fin de la procédure
- ⇒ Section du test Minuscule
 - ⇒ On compare AL à 'a'=61h
 - ⇒ Si AL < 'a' on retourne à la section de saisie
 - ⇒ On compare AL à 'f'=66h
 - ⇒ Si AL > 'f' on retourne à la section de saisie
 - ⇒ Nous sommes dans l'intervalle 'a'...'f'. Alors on retranche de AL le code ascii de 'a' et on rajoute +10 et on saute à la fin de la procédure
- ⇒ Section de la sortie de la procédure

La procédure « scanInt »

```

scanInt proc near
push ax
push cx
push dx
mov bx,0
mov ch,4
_si_scanHex:
call scanHex;
cmp al,0F0h
je _si_out
mov cl,4
sal bx,cl
add bl,al

push bx
lea bx,_ascii
xlat
mov dl,al
mov ah,2
int 21h
pop bx
dec ch
cmp ch,0
jne _si_scanHex
_si_out:
pop dx
pop cx
pop ax
ret
scanInt endp

```

- ⇒ « scanInt » retourne dans BX la valeur de l'entier tapé
- ⇒ On sauvegarde AX, BX et DX
- ⇒ On initialise le registre de sortie BX à la valeur 0
- ⇒ On initialise CH à 4 étant le nombre maximal des digits
- ⇒ Section de saisie d'un seul digit
- ⇒ On appelle la procédure « scanHex » qui retourne dans AL la valeur à traiter
- ⇒ On teste si la touche tapée est la touche entrée
- ⇒ Si oui on sort de la procédure
- ⇒ Sinon on ajoute la valeur tapée dans BX
- ⇒ BX : xxxx.xxxx.xxxx.xxxx => xxxx.xxxx.xxxx.0000
- ⇒ AL : 0000.aaaa => BX : xxxx.xxxx.xxxx.aaaa
- ⇒ Affichage du caractère correspondant à la valeur retournée par « scanHex »
- ⇒ Sauvegarde de BX
- ⇒ On pointe l'adresse de « _ascii » par BX
- ⇒ AL <= _ascii[AL] : conversion
- ⇒ Affichage du contenu de AL
- ⇒ Récupération de BX
- ⇒ On décrémente CH (on manque un caractère)
- ⇒ On teste si on a tapé les 4 caractères
- ⇒ Si CH != 0 c.à.d. il reste des digits à taper on retourne à la section de saisie
- ⇒ Section de sortie
- ⇒ Restauration des registres DX, CX et AX

La procédure « printInt »

```

printInt proc near
push ax
push bx
push cx
push dx

lea bx,_ascii
mov ch,0

```

- ⇒ « printInt » affiche le mot passé dans le registre DX
- ⇒ Sauvegarde de AX, BX, CX et DX
- RQ : on sauvegarde deux fois DX *
- ⇒ On pointe « _ascii » par le registre BX
- ⇒ Initialisation de CH à 0 dans le but d'afficher octet par octet

<code>_pi_print_byte:</code>	⇒ Section d'affichage de l'octet située dans DH
<code>mov al,dh</code>	⇒ AL <= DH
<code>mov cl,4; on</code>	⇒ Préparer CL à un décalage de 4
<code>shr al,cl</code>	⇒ AL : HHHH.LLLLL ⇒ 0000.HHHH
<code>xlat</code>	⇒ Conversion
<code>mov dl,al</code>	⇒ Affichage
<code>mov ah,2</code>	
<code>int 21h</code>	
<code>mov al,dh</code>	⇒ AL <= DH
<code>and al,0fh</code>	⇒ AL : HHHH.LLLL ⇒ 0000.LLLL
<code>xlat</code>	⇒ Conversion
<code>mov dl,al</code>	⇒ Affichage
<code>mov ah,2</code>	
<code>int 21h</code>	
<code>cmp ch,0</code>	⇒ On compare CH à 0
<code>jnz _pi_fin</code>	⇒ Si CH != 0 au sort de la procédure
	⇒ Si CH=0
<code>pop dx</code>	⇒ Restauration de DX *
<code>mov dh,dl</code>	⇒ On met DL dans DH
<code>inc ch</code>	⇒ On modifie la valeur de CH par incrémentation
<code>jmp _pi_print_byte</code>	⇒ On se branche à la section d'affichage d'un octet
<code>_pi_fin:</code>	⇒ Section de la sortie de la procédure
<code>pop dx</code>	⇒ Restauration de DX, CX, BX et AX
<code>pop cx</code>	
<code>pop bx</code>	
<code>pop ax</code>	
<code>ret</code>	
<code>printInt endp</code>	

La procédure « prog_addition »

<code>prog_addition proc near</code>	⇒ « prog_addition » garantie l'addition de deux mots
<code>push ax</code>	⇒ Sauvegarde de AX, BX, CX et DX
<code>push bx</code>	
<code>push cx</code>	
<code>push dx</code>	
<code>lea dx,_qn1</code>	⇒ Affichage du message _qn1 = ' n1 = '
<code>mov ah,9</code>	⇒ DX pointe le message
<code>int 21h</code>	⇒ AH <= 9
<code>call scanInt</code>	⇒ Appel de l'interruption 9
<code>mov cx,bx</code>	⇒ Appel de la procédure « scanInt »
<code>call wrLn</code>	⇒ Sauvegarde de l'entier saisi dans CX
	⇒ Retour à la ligne
<code>lea dx,_qn2</code>	⇒ Affichage du message _qn2 = ' n2 = '
<code>mov ah,9</code>	⇒ DX pointe le message
<code>int 21h</code>	⇒ AH <= 9
<code>call scanInt</code>	⇒ Appel de l'interruption 9
<code>call wrLn</code>	⇒ Appel de la procédure « scanInt » l'entier saisi sera dans BX
	⇒ Retour à la ligne
<code>lea dx,_r_add</code>	⇒ Affichage du message _r_add = ' n1 + n2 = '
<code>mov ah,9</code>	
<code>int 21h</code>	
<code>mov dx,0</code>	⇒ ADDITION :
<code>add cx,bx</code>	⇒ Initialiser DX à 0 : DX étant la partie haute du résultat

```

adc dx,0
call printInt
mov dx,cx
call printInt
call wrLn

pop dx
pop cx
pop bx
pop ax
ret
prog_addition endp

```

- ⇒ $CX \leq CX + BX$
- ⇒ Ajout de carry dans DX
- ⇒ Affichage de DX
- ⇒ Affichage de CX
- ⇒ Retour à la ligne
- ⇒ Restauration de DX, CX, BX et AX

La procédure « prog_soustraction »

```

prog_soustraction proc near
  push ax
  push bx
  push cx
  push dx

  lea dx,_qn1
  mov ah,9
  int 21h
  call scanInt
  mov cx,bx
  call wrLn

  lea dx,_qn2
  mov ah,9
  int 21h
  call scanInt
  call wrLn

  lea dx,_r_sub
  mov ah,9
  int 21h

  sub cx,bx
  jns _ps_jumpOver

  mov dl,'-
  mov ah,2
  int 21h
  neg cx
  _ps_jumpOver:
  mov dx,cx
  call printInt
  call wrLn

  pop dx
  pop cx
  pop bx
  pop ax
  ret
prog_soustraction endp

```

- ⇒ « prog_soustraction » garantie la soustraction de deux mots
- ⇒ Sauvegarde de AX, BX, CX et DX
- ⇒ Affichage du message _qn1 = ' n1 = '
- ⇒ Appel de la procédure « scanInt »
 - ⇒ Sauvegarde de l'entier saisi dans CX
- ⇒ Retour à la ligne
- ⇒ Affichage du message _qn2 = ' n2 = '
- ⇒ Appel de la procédure « scanInt » l'entier saisi sera dans BX
- ⇒ Retour à la ligne
- ⇒ Affichage du message _r_sub = ' n1 - n2 = '
- ⇒ SOUSTRACTION
 - ⇒ $CX \leq CX - BX$
 - ⇒ Si le résultat de soustraction est négatif alors on saute le traitement de cas négatif
 - ⇒ Sinon ($BX > CX$)
 - ⇒ Affichage de la signe (-)
 - ⇒ On complémente CX
 - ⇒ Déclaration du saut
 - ⇒ Affichage de CX
 - ⇒ Retour à la ligne
- ⇒ Récupération de DX, CX, BX et AX

La procédure « prog_multiplication »

prog_multiplication proc near	⇒ « prog_multiplication » garantie la multiplication de deux mots
push ax	⇒ Sauvegarde de AX, BX, CX et DX
push bx	
push cx	
push dx	
 lea dx,_qn1	⇒ Affichage du message _qn1 = ' n1 = '
mov ah,9	
int 21h	
call scanInt	⇒ Appel de la procédure « scanInt »
mov cx,bx	⇒ Sauvegarde de l'entier saisi dans CX
call wrLn	⇒ Retour à la ligne
 lea dx,_qn2	⇒ Affichage du message _qn2 = ' n2 = '
mov ah,9	
int 21h	
call scanInt	⇒ Appel de la procédure « scanInt » l'entier saisi sera dans BX
call wrLn	⇒ Retour à la ligne
 lea dx,_r_mul	⇒ Affichage du message _r_mul = 'n1 * n2 = '
mov ah,9	
int 21h	
 mov ax,cx	⇒ MULTIPLICATION
mul bx	⇒ AX <= CX
call printInt	⇒ DX :AX <= AX * BX
mov dx,ax	⇒ Affichage de DX
call printInt	⇒ Affichage de AX
call wrLn	⇒ Retour à la ligne
 pop dx	⇒ Restauration de DX, CX, BX et AX
pop cx	
pop bx	
pop ax	
ret	
prog_multiplication endp	

La procédure « prog_division »

prog_division proc near	⇒ « prog_division » garantie la division de deux mots
push ax	⇒ Sauvegarde de AX, BX, CX et DX
push bx	
push cx	
push dx	
 lea dx,_qn1	⇒ Affichage du message _qn1 = ' n1 = '
mov ah,9	
int 21h	
call scanInt	⇒ Appel de la procédure « scanInt »
mov cx,bx	⇒ Sauvegarde de l'entier saisi dans CX
call wrLn	⇒ Retour à la ligne
 lea dx,_qn2	⇒ Affichage du message _qn2 = ' n2 = '
mov ah,9	
int 21h	

call scanInt	⇒ Appel de la procédure « scanInt » l'entier saisi sera dans BX
call wrLn	⇒ Retour à la ligne
lea dx,_r_div	⇒ Affichage du message _r_div = 'n1 / n2 = '
mov ah,9	
int 21h	
mov dx,0	⇒ DIVISION
mov ax,cx	⇒ Initialiser DX à 0
div bx	⇒ AX <= CX
mov dx,ax	⇒ AX = (DX :AX) DIV (BX) & DX = (DX :AX) MOD (BX)
call printInt	⇒ Affichage de AX
call wrLn	⇒ Retour à la ligne
pop dx	⇒ Restauration de DX, CX, BX et AX
pop cx	
pop bx	
pop ax	
ret	
prog_division endp	

La procédure« pgcd »

pgcd proc near	⇒ « pgcd » : calcul le pgcd de BX et CX et le met dans DX
push ax	⇒ Sauvegarde de AX, BX et CX
push bx	
push cx	
_pgcd_cmp:	⇒ Section de comparaison
cmp bx,cx	⇒ On compare BX à CX
je _pgcd_end	⇒ S'in sont égaux Donc on à trouvé le PGCD alors on sort de la procédure
cmp bx,cx	⇒ On compare BX à CX
ja _pgcd_over	⇒ Si BX > CX alors on saute au traitement correspondant
sub cx,bx	⇒ Sinon c.à.d. BX < CX ==> CX <= CX - BX
jmp _pgcd_cmp	⇒ On revient à la section de comparaison
_pgcd_over:	⇒ Traitement du cas BX > CX
sub bx,cx	⇒ BX <= BX - CX
jmp _pgcd_cmp	⇒ On revient à la section de comparaison
_pgcd_end:	⇒ Fin de la procédure :
mov dx,bx	⇒ On met le résultat dans DX
pop cx	⇒ Restauration des registre CX, BX et AX
pop bx	
pop ax	
ret	
pgcd endp	

La procédure« prog_pgcd »

prog_pgcd proc near	⇒ « prog_pgcd » garantie l'exécution de l'opération du PGCD
push ax	⇒ Sauvegarde de AX, BX, CX et DX
push bx	
push cx	
push dx	
lea dx,_qn1	⇒ Affichage du message _qn1 = ' n1 = '
mov ah,9	
int 21h	
call scanInt	⇒ Appel de la procédure « scanInt »
mov cx,bx	⇒ Sauvegarde de l'entier saisi dans CX

call wrLn	⇒ Retour à la ligne
lea dx,_qn2	⇒ Affichage du message _qn2 = ' n2 = '
mov ah,9	
int 21h	
call scanInt	⇒ Appel de la procédure « scanInt » l'entier saisi sera dans BX
call wrLn	⇒ Retour à la ligne
lea dx,_r_pgcd	⇒ Affichage du message _r_pgcd = ' PGCD(n1,n2) = '
mov ah,9	
int 21h	
call pgcd	⇒ PGCD
call printInt	⇒ Appel de la procédure PGCD qui fait l'opération : $DX \leftarrow \text{pgcd}(BX, CX)$
call wrLn	⇒ Affichage de DX
	⇒ Retour à la ligne
pop dx	⇒ Restauration de DX, CX, BX et AX
pop cx	
pop bx	
pop ax	
ret	
prog_pgcd endp	

La procédure « prog_ppcm »

prog_ppcm proc near	⇒ « prog_ppcm » garantie l'exécution de l'opération de PPCM
push ax	⇒ Sauvegarde de AX, BX, CX et DX
push bx	
push cx	
push dx	
lea dx,_qn1	⇒ Affichage du message _qn1 = ' n1 = '
mov ah,9	
int 21h	
call scanInt	⇒ Appel de la procédure « scanInt »
mov cx,bx	⇒ Sauvegarde de l'entier saisi dans CX
call wrLn	⇒ Retour à la ligne
lea dx,_qn2	⇒ Affichage du message _qn2 = ' n2 = '
mov ah,9	
int 21h	
call scanInt	⇒ Appel de la procédure « scanInt » l'entier saisi sera dans BX
call wrLn	⇒ Retour à la ligne
lea dx,_r_ppcm	⇒ Affichage du message _r_ppcm = ' PPCM(n1,n2) = '
mov ah,9	
int 21h	
call pgcd	⇒ PPCM
mov ax,bx	⇒ $DX = \text{PPCM}(BX, CX)$
mov bx,dx	⇒ $AX \leftarrow BX$
mul cx	⇒ $BX \leftarrow DX : \text{PGCD}$
div bx	⇒ $DX : AX \leftarrow AX * CX$
mov dx,ax	⇒ $AX = (DX : AX) \text{ DIV } (BX)$
call printInt	⇒ Affichage de $AX = \text{PPCM}(BX, CX) = BX.CX / \text{PGCD}(BX, CX)$
call wrLn	⇒ Retour à la ligne
pop dx	⇒ Récupération de DX, CX, BX et AX

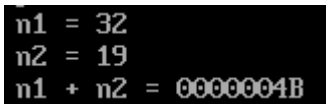
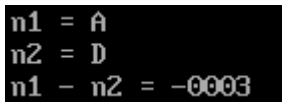
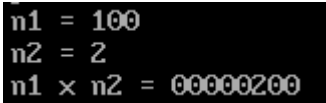
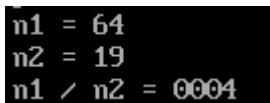
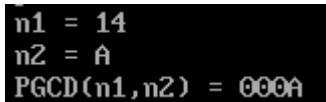

```

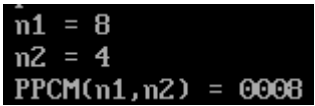
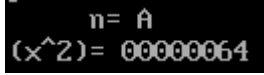
pop cx
pop bx
pop ax
ret
prog_ppcm endp

```

III. Les traces d'exécution :

Dans cette partie du chapitre 3 on va présenter des exemples de traces d'exécution de chaque fonction.

<p>On a effectué un exemple de trace d'exécution pour la procédure addition :</p> <p> $n1 = (50)_{10} = (32)_{16}$ $n2 = (25)_{10} = (19)_{16}$ $n1 + n2 = (75)_{10} = (4B)_{16}$ </p>	
<p>Un exemple d'exécution pour la procédure soustraction :</p> <p> $n1 = (10)_{10} = (A)_{16}$ $n2 = (13)_{10} = (D)_{16}$ $n1 - n2 = -3$ </p>	
<p>La procédure multiplication nous permet de multiplier deux nombres entrés en hexadécimaux :</p> <p> $n1 = (256)_{10} = (100)_{16}$ $n2 = (2)_{10} = (2)_{16}$ $n1 * n2 = (512)_{10} = (200)_{16}$ </p>	
<p>Une trace d'exécution pour la procédure division pour les deux entiers suivants :</p> <p> $(64)_{16} = (100)_{10}$ $(19)_{16} = (25)_{10}$ $n1 / n2 = (4)_{10} = (4)_{16}$ </p>	
<p>Pour la procédure PGCD on a calculé le pgcd des deux nombres suivants :</p> <p> $n1 = (14)_{16} = (20)_{10}$ $n2 = (A)_{16} = (10)_{10}$ $\text{pgcd}(n1, n2) = (10)_{10} = (A)_{16}$ </p>	






Pour le PPCM on a : $n1=(8)_{10}=(8)_{16}$ $n2=(4)_{10}=(4)_{16}$ $PPCM(n1,n2)=(8)_{10}=(8)_{16}$	 <pre> n1 = 8 n2 = 4 PPCM(n1, n2) = 0008 </pre>
On finalement calculer la puissance de 2 pour : $n=(10)_{10}=(A)_{16}$ $n^2=(100)_{10}=(64)_{16}$	 <pre> n = A (x^2) = 00000064 </pre>

IV.CONCLUSION:

Ce chapitre nous a détaillé la partie programmation de la réalisation de la calculatrice en langage assembleur afin d'éclaircir les procédures utilisés dans le corps du programme principale.

PERSPECTIVES

Malgré qu'on a réussi à achever notre but et effectuer une calculatrice ou on peut faire l'addition, la soustraction, la division, la multiplication, la puissance de deux et calculer le PGCD et le PPCM. Notre travail reste critiquant face aux améliorations et les perspectives qu'on peut l'effectuer. Pour cela on peut citer quelques progrès de ce projet qui lui donne une meilleur lisibilité des résultats ou au niveau de la saisie des valeurs :

-  On constate qu'au niveau de l'affichage des zéros persistent qui gêne la clarté de l'affichage.
-  La division n'est pas à virgule flottante.
-  La saisie des nombres est en hexadécimale peut être améliorée en une saisie en décimale.
-  Au niveau de saisie on peut ajouter la possibilité de supprimer les digits tapés.
-  On a détecté un bogue au niveau de saisie lorsqu'on tape « entrée » tout au début de saisie.