PARIS DESCARTES IUT

Département Informatique

TP UE21-M2101
Architecture et programmation des mécanismes de base d'un système informatique
1ère année – Semestre 2
Version 2017-2018

Responsable du cours : Philippe Darche

Version du samedi 18 mars 2017

Introduction

Le module UE21-M2101 *Architecture et programmation des mécanismes de base d'un système informatique* du Programme Pédagogique National (PPN) 2013 du Diplôme Universitaire de Technologie (DUT) Informatique a pour objectif de **savoir développer des applications simples mettant en oeuvre les mécanismes de bas niveau d'un système informatique**. Il complète le module M1101 du semestre précédent. Ce module se compose de 7 cours et de 7 séances de TP. Il sera divisé au niveau du cours en deux parties, l'assemblage et l'architecture des ordinateurs.
Le cours doit être appris et le texte du TP doit être lu et préparé avant d'arriver en séance, cette dernière ne servant qu'à corriger les exercices, qu'à manipuler sur machinen ainsi qu'à répondre aux questions que vous vous posez. Les supports de cours et les textes des TP sont sur le disque commun aux étudiants de la formation dans le répertoire dont le nom réseau est :

\\SRVSAUV\INFO\COMMUN\DUT 1ere année\M2101_Architecture_des_ordinateurs

Ce répertoire est accessible directement par le poste de travail (disque H:\COMMUN\ DUT 1ere année\ M2101_Architecture_des_ordinateurs). Une liste des acronymes et une bibliographie s'y trouvent aussi.

Règles typographiques : Les mots en anglais sont en italique dans la police courante. Pour la partie « langage d'assemblage », les commandes à taper sont en police « `courier new` ».

Illustration : L'ordinateur UNIVAC I de la société Remington Rand (1951).

TP 1 : découverte de la chaîne de développement

Objectifs pédagogiques

L'objet de ce TP est d'analyser le listing d'un programme assemblé. Les notions de segment, d'identificateurs et d'adressage à l'intérieur de celui-ci sont abordées. L'environnement de développement est installé et présenté.

Exercice 1 : analyse d'un fichier affiche.asm assemblé en version détaillée (.lst)

Rappel d'ASR1. Soit le fichier affiche.lst donné en annexe.

Q1. Rappelez la chaîne de développement d'un programme écrit en langage évolué C. Que contient un fichier objet ? A quoi sert un éditeur de liens ? A quel niveau de langage se situe le langage d'assemblage ?

Q2. Qu'est-ce qu'un segment ? Comment le CPU repère le segment en mémoire ?

Q3. Qu'indiquent les directives STACK 100h, DATASEG et CODESEG du fichier source (extension .asm) ou listing (extension .lst) de l'annexe ?

Q4. Que représente les première, seconde, troisième et quatrième colonnes du fichier listing généré par l'option /l de l'assembleur tasm ?

Q5. A quoi sert la directive #PRAGMA dans le langage C ? A quoi servent les directives d'assemblage en début de fichier et réparties dans l'ensemble de ce fichier ?

Q6. Qu'est-ce qu'un identificateur ? Qu'est-ce qu'une table des symboles (*symbol table*) ? Où se trouve-t-elle dans le fichier listing ?

Q7. Que deviennent les identificateurs après assemblage ?

Q8. A quel segment appartient la chaîne de caractère `chaine` ? Quelle est son adresse de rangement dans ce segment ? Quelle est son encombrement mémoire (octet) ? A quoi sert la variable suivante avec le caractère $? A quoi sert le caractère $ dans l'expression mathématique ?

Q9. A quel segment appartient l'instruction `mov ah,9` ? Quelle est son adresse de rangement dans ce segment ? Quelle est son encombrement mémoire (octet) ? Quelle est l'adresse de l'instruction suivante ?

Q10. A partir des valeurs d'adresse trouvées dans les deux questions précédentes, que pouvez-vous déduire concernant le mode d'adressage des identificateurs ? Quel est l'intérêt de ce mode d'adressage dans un contexte de multiprogrammation ?

Q11. A partir du format des adresses, déterminez la taille maximale d'un segment pour le MPU 8086 ?

Exercice 2 : assemblage du premier programme (affiche.asm)

Q1. Assemblez le programme affiche.asm. Réalisez l'édition de liens et exécutez le programme. Quel est le rôle des options ?

Annexe

```
Turbo Assembler    Version 3.1          24/01/10 17:56:28       Page 1
affiche.asm


      1                             ; nom du programme : affiche.asm
      2                             ; fonction : affichage d'une chaîne de caractères
      3
      4                                   IDEAL
      5                                   DOSSEG
      6     0000                          MODEL TINY
      7                                   P8086
      8
      9     0000                          STACK 100h
     10
     11     0100                          DATASEG
     12     0000  48 65 6C 6C 6F 20 57+   chaine DB "Hello World   !"
     13           6F 72 6C 64 20 21
     14     000D  24                      fin_chaine   DB "$"
     15           =000E                   longueur =   $ - chaine
     16
     17     000E                          CODESEG
     18
     19     0000  B8 0000s          debut:  mov ax,@data
     20     0003  8E D8                     mov ds,ax
     21     0005  8E C0                     mov es,ax
     22
     23     0007  B8 000E                   mov ax,longueur
     24
     25                             ; initialisation des paramètres entrants
     26     000A  B4 09                     mov ah,9
     27     000C  BA 0000r                  mov dx,OFFSET chaine
     28                             ; appel système pour l'affichage
     29     000F  CD 21                     int 21h
     30
     31                             ; appel système pour la fin de programme
     32     0011  B4 4C                     mov ah,4ch
     33     0013  CD 21                     int 21h
     34                                   END debut


Turbo Assembler    Version 3.1          24/01/10 17:56:28       Page 2
Symbol Table



Symbol Name                Type       Value

??DATE                     Text       "24/01/10"
??FILENAME                 Text       "affiche "
??TIME                     Text       "17:56:28"
??VERSION                  Number     030A
@32BIT                     Text       0
@CODE                      Text       DGROUP
@CODESIZE                  Text       0
@CPU                       Text       0101H
@CURSEG                    Text       _TEXT
@DATA                      Text       DGROUP
@DATASIZE                  Text       0
@FILENAME                  Text       AFFICHE
@INTERFACE                 Text       00H
```

```
@MODEL                          Text       1
@STACK                          Text       DGROUP
@WORDSIZE                       Text       2
CHAINE                          Byte       DGROUP:0000
DEBUT                           Near       DGROUP:0000
FIN_CHAINE                      Byte       DGROUP:000D
LONGUEUR                        Number 000E

Groups & Segments               Bit Size Align    Combine Class

DGROUP                          Group
  STACK                         16  0100 Para     Stack   STACK
  _DATA                         16  000E Word     Public  DATA
  _TEXT                         16  0015 Word     Public  CODE
```

TP 2 : écriture et exécution pas-à-pas du premier programme

Objectifs pédagogiques

L'objetif de ce TP est d'écrire, d'assembler rt d'analyser l'exécution pas à pas d'un programme. Le programme de débogage Turbo Debugger (TD) est mis en œuvre. Avant de commencer, la notion de registre est abordé.

Exercice 1 : exécution d'une instruction

Q1. Est-ce qu'un microprocesseur peut arrêter son cycle d'exécution lors de l'exécution d'une instruction ?

Q2. A quoi sert le registre des indicateurs (binaires) ?

Q3. A quel moment le registre des indicateurs (binaires) est-il renseigné ? L'est-il toujours ? Dans la négative, donnez un exemple.

Exercice 2 : écriture du premier programme

Q1. En prenant comme modèle le fichier modele.asm, écrivez le programme TP_add2.asm qui calcule l'expression suivante : RM = M1 + M2. Ces trois variables seront définies dans le segment de données. Proposez une initialisation. Quelles sont les questions préalables à se poser ? Exécutez-le pas-à-pas.

Q2. Complétez le programme précédent pour qu'il calcule l'expression suivante : R = M1 + M2 + M3. Il y aura deux versions de programme (TP_add3_1.asm et TP_add3_2.asm) avec les initialisations suivantes :

```
M1    DW 65534
M2    DW 1
M3    DW 1
```

et

```
M1    DW 32766
M2    DW 1
M3    DW 1
```

Exécutez-les pas-à-pas. Que remarquez-vous au niveau des indicateurs. Interprétez les résultats.

Exercice 3 : lecture d'une documentation constructeur

Soit le jeu d'instruction du microprocesseur 8086/8088 de la société Intel.

Q1. Cherchez dans la documentation les informations se référant à l'instruction `add` (8086_documentation_1_scan.pdf). Quels sont les indicateurs qui risquent d'être modifiés suite à son exécution ? Quels sont ses modes d'adressage ? Pour chacun d'eux, précisez le nombre de cycles d'horloge nécessaires si le mode d'adressage concerné est utilisé par cette instruction.

Q2. Avec l'instruction `std`, quels sont les indicateurs qui risquent d'être modifiés suite à son exécution ? Expliquez cette valeur.

Q3. A partir des deux questions précédentes, tirez une conclusion sur la valeur d'un indicateur.

TP 3 : le transfert de contrôle

Partie 1 : les sauts conditionnels

Objectifs pédagogiques

L'objet de ce TP est d'étudier certains mécanismes de base du microprocesseur et des langages de programmation. Il a aussi comme but que vous deveniez autonome au niveau de la chaîne de développement (*i.e.* assemblage et éditions de lien) d'un programme.

Question de cours : les sauts

Q1. A quoi sert une instruction de saut conditionnel ? Donnez un exemple d'instruction.

Q2. Sur quel(s) évènement(s) ou état(s) se réalise le débranchement ?

Q3. Existe-t-il d'autres types de saut ? Donnez un exemple d'instruction.

Exercice 1 : les sauts conditionnels

Lorsqu'un programme est demandé, faites l'analyse du problème, l'écriture, l'assemblage et l'exécution de ce dernier.

Q1. La structure de contrôle si_alors. Réalisez un programme qui teste si la valeur d'une variable `nombre` est supérieure à 5 et qui affiche à l'écran le message « valeur supérieure à 5». Quelle est ou quelles sont les meilleures valeurs de test pour la variable `nombre` ?

Q2. La structure de contrôle si_alors_sinon. Modifiez le programme précédent pour qu'il raffine l'affichage en indiquant à l'écran les messages « valeur supérieure à 5», « valeur égale à 5 » ou « valeur inférieure à 5 » selon le cas.

Généralités sur l'appel système

L'utilisateur d'un système informatique se voit proposer, tout du moins en ce qui concerne les mini-ordinateurs et les micro-ordinateurs, un ensemble de routines qui lui permettent de disposer des fonctionnalités du système d'exploitation à l'intérieur d'un programme utilisateur. L'accès à ces primitives est normalisé, c'est-à-dire que le passage des paramètres et l'appel s'effectuent selon une convention pré-définie. Dans le cas du DOS PC, l'appel s'effectue via le mécanisme des interruptions et le passage des paramètres se fait par les registres.

L'interruption DOS N° 33

Il existe une interruption réservée aux appels système qui porte le numéro 33 (ou 21h). La fonction N° 9 de cette interruption permet d'afficher une chaîne de caractères à l'écran. La syntaxe est la suivante :

- INT 33 ou INT 21h

Le passage des paramètres doit suivre la convention suivante :

- le registre de données AH (groupe registres généraux) doit contenir le numéro de la fonction,
- DX, le registre de données, contiendra l'adresse logique du début de la chaîne à afficher,
- la chaîne à afficher doit se terminer par le caractère "$".

Remarques

- N’oubliez pas les caractères de tabulations lors de l’affichage.
- Les données seront rangées en mémoire, il n’y aura donc pas d’initialisation au clavier.

Annexe

```
; nom du programme : affiche.asm
; fonction : affichage d'une chaîne de caractères

        IDEAL
        DOSSEG
        MODEL TINY
        P8086

        STACK 100h

        DATASEG
msg     DB "Hello World !"
fin_msg DB "$"
; longueur = $ - msg ; pour calcul automatique d'une longueur de chaine

        CODESEG

debut:  mov ax,@data
        mov ds,ax
        mov es,ax

; initialisation des paramètres entrants
        mov ah,9
        mov dx,OFFSET msg
; appel fonction système d'affichage
        int 21h

; appel système fin de programme
        mov ah,4ch
        int 21h
        END debut
```

TP 4 : le transfert de contrôle (suite)

Partie 1 : les sauts conditionnels (suite)

Objectifs pédagogiques

L'objet de ce TP est d'étudier certains mécanismes de base du microprocesseur et des langages de programmation. Le fonctionnement de l'instruction MUL est détaillé. L'instruction `loop` est introduite. Nous nous intéressons, pour finir, à la valeur de l'adresse relative de saut ou offset lors d'un saut conditionnel.

Q1. La structure de contrôle répéter_jusqu'à. Réalisez un programme qui calcule la factorielle d'un nombre n et qui range le résultat dans une variable **résultat**. Vous utiliserez l'instruction `mul` au format octet. N'oubliez pas les cas extrêmes. Quelle est la meilleure valeur de test pour n ?
La démarche de développement sera d'écrire une première version sans se préoccuper des valeurs particulières. La seconde les prendra en compte.

Rappel : fac(n) = n! = n × (n - 1) × (n - 2) × ..... × 1 avec 0! = 1

1°) Remplissez le tableau Q4-1.

n	n!
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tableau Q4-1 : Quelques valeurs de fac(n)

2°) Analysez le problème en remplissant le tableau Q4-2.

Variables	Valeur minimale	Valeur maximale	Format minimum (bits)	Format retenu pour le programme
n				
résultat				

Tableau Q4-2 : Etude du format des variables

Q1.1. Quelles sont les valeurs des adresses de saut ? Expliquez-les.

Q2. La structure de contrôle tant_que. Modifiez le programme de factorielle précédent pour qu'il utilise le saut conditionnel `jcxz`. Une dernière version utilisera l'instruction `loop`.

Q3. A quelle catégorie de saut appartient l'instruction `loop` ?

Q4. Même problème que précédemment avec la structure de contrôle **pour_var_allant_de_x_à_y**. Réalisez un programme qui calcule la factorielle d'un nombre n et qui range le résultat dans une variable résultat. Vous utiliserez l'instruction `mul`. Trois versions sont à coder : comptage et décomptage avec instruction de saut conditionnel.

Q5. facultatif. Test des indicateurs. Réalisez un programme qui additionne deux mots binaires. Il devra ensuite afficher la valeur de la retenue et du dépassement de capacité. Application numérique au format n = 8 bits : 0100 0000 + 0100 0000. Interprétez le résultat à partir de ces indicateurs. Autre calcul : 0100 0000 + 1100 0000.

Q6. facultatif. Test des indicateurs. Réalisez un programme qui soustrait deux mots binaires. Il devra ensuite indiquer la valeur de la retenue et du dépassement de capacité. Application numérique au format n = 8 bits : 0100 0000 - 1100 0000. Interprétez le résultat à partir de ces indicateurs.

TP 5 : le transfert de contrôle (suite)

Partie 2 : les appels de sous-programme

Objectifs pédagogiques

L'objet de ce TP est d'étudier certains mécanismes de base du microprocesseur et des langages de programmation. La séance est consacré à l'étude de la notion de sous-programme en réalisant un progarmme de multiplication par additions successives. L'étudiant devra être autonome en préparation du DST.

Exercice 1 : appel de sous-programme

Q1. Quelle(s) différence(s) faites-vous entre une fonction et une procédure ? Répondez en remplissant le tableau Q5-1

Caractéristiques	Fonction	Procédure
Exemple de langage		
Nombre de paramètres entrants		
Nombre de paramètres sortants		

Tableau Q5-1 : Comparaison entre fonction et procédure

Q2. Comment implémente-t-on une fonction ou une procédure (concepts de langage de haut niveau) en langage d'assemblage ?

Q3. Comment peut-on réaliser, en langage d'assemblage, un débranchement de séquence d'exécution sans utiliser les instructions de saut (in)conditionnel ?

Q4. Quelles sont les types et les modes de passage (*i.e.* quelle mémoire) pour passer des paramètres à un sous-programme ? Donnez les avantages de chaque mode de passage.

Q5. Quelles sont les instructions qui sont impliquées lors de l'utilisation d'un sous-programme (*i.e.* passage de paramètre(s) et déroutement) ?

Q6. Qu'est-ce que le contexte d'exécution dans le cadre d'un appel à un sous-programme ? Que contient-il au minimum ? Quel est l'avantage de sauvegarder que ce contexte minimal ?

Q7. Les instructions `call` et `ret` modifient-elles les indicateurs ? Etait-ce prévisible ?

Q8. Ecrire un programme de multiplication de deux entiers naturels X et Y par additions successives avec rangement dans une variable résultat R. X et Y $\in$ [0, 255]. Le travail devra être incrémental. Une première version réalisera l'opération demandée sans optimisation (R = 0 + X + ··· + X, Y fois). Une deuxième version résolvera les cas où les variables X ou Y sont nuls. La dernière minimisera la valeur du compteur de boucle.

Q9. Transformez le programme de multiplication de deux entiers naturels par additions successives de la question précédente pour qu'il devienne un sous-programme. Le passage de paramètres sera de type « par valeur » et « par registre ».

TP 6 : le transfert de contrôle (fin)

Partie 2 (suite) : la notion de pile

Objectifs pédagogiques

L'objet de ce TP est d'étudier certains mécanismes de base du microprocesseur et des langages de programmation. La séance est consacré à l'étude la pile. La séance consiste à modifier un programme existant pour un passage de paramètre par la pile. La dynamique de la pile sera visualisée avec le débogueur TD.

Exercice 1 : appel de sous-programme

Q1. Quel est le format de l'opérande des instructions de manipulation de la pile push et pop ?

Q2. Transformez le programme de multiplication de la dernière question du TP précédent pour que le passage de paramètres soit de type « par valeur » et « par la pile ».

Q3. Calculez l'Adresse Effective (AE) lors de l'exécution de l'instruction call (*cf.* fichier listing) ?

Q4. Observez avec Turbo Debugger en mode pas-à-pas l'évolution du pointeur de pile (cadre pile en bas à droite de la fenêtre CPU) à chaque empilement, dépilement, appel et retour de sous-programme.

Exercice 2 : appel de sous-programme

Q1. Transformez le programme de l'exercice précédent pour que le passage de paramètres soit de type « par adresse » et « par la pile » **pour les paramètres entrants**. Le passage du paramètre sortant reste par valeur et par registre.

Exercice 3 : les indicateurs du registre d'état

Q1. Ecrivez un programme qui affiche la valeur des indicateurs du registre d'état (CCR pour *Code Condition Register*), une par ligne, suite à l'opération de FFFFh + 8000h.

TP 7 : analyse d'un problème et écriture du programme correspondant

Objectifs pédagogiques

L'objet de ce TP est de renforcer les notions de langage d'assemblage abordées dans les séances précédentes. Il permet de se préparer au DST.

Exercice 1 : calcul de plusieurs factorielles de n et rangement dans un tableau

Objectifs :

L'analyse et l'écriture du programme factorielle ont été faites pendant le TP 4. A partir de ces résultats, la version en sous-programme est à écrire. Ecrivez un programme qui appelle 10 fois un sous-programme calculant la factorielle de x (x variant de 0 à 9) et qui les range dans le tableau déclaré ci-dessus dans l'ordre croissant des adresses. Le passage des paramètres se fera **par la pile et par valeur**. La déclaration d'un tableau dans le segment de données sera la suivante :

TAB DW 20 DUP (0)

Répondez aux deux questions suivantes :

- nombre de paramètre(s) entrant(s) ?
- nombre de paramètre(s) sortant(s) ?

Exercice 2 : calcul de la suite de Fibonancci

Les "nombres de Fibonacci" sont définis comme suit :

Fib(0) = 1,

Fib(1) = 1,

Fib(n) = Fib(n-1) + Fib(n-2) pour $n \geq 2$

avec, pour notre problème, n entier, $\in \{0, \cdots, 12\}$. Cela revient donc, pour $n \geq 2$, à calculer la somme des deux précédentes valeurs calculées pour obtenir la suivante.

Q1. Quelle est la représentation numérique que vous allez choisir ?

Q2. Analysez le problème en remplissant le tableau suivant :

Variables	Valeur min.	Valeur max.	Format min. (bit)	Format retenu pour le programme (bit)
n				
Fib(n)				

Tableau Q7-1 : Etude du format des variables

Q3. Déclarez deux variables n et Fibn initialisées respectivement à 12 et 0. Ecrire un programme en pseudo-code, puis le programme correspondant prêt à être assemblé qui calcule Fib(n) **de manière itérative** (par opposition à récursive) et qui range le résultat en mémoire dans la variable Fibn.

Q4. Transformez le programme précédent pour qu'il devienne un sous-programme. Le passage de paramètres

se fera « par valeur » et « par registre ».

Q5. Maintenant, n entier, $\in \{0, ..., 13\}$. Soit la déclaration d'un tableau dans le segment de données :

```
TAB DW 14 DUP (0)
```

Ecrire un programme qui appelle 14 fois un sous-programme calculant Fib(n) (n variant de 0 à 13) et qui les range dans le tableau déclaré ci-dessus dans l'ordre croissant des adresses mémoires. Le passage des paramètres se fera par **registre** et par **valeur**. Le sous-programme sera une adaptation du programme précédant.

Q6. Indiquez le nombre de paramètres entrants et sortants, leur type, les registres utilisés et les variables auxquelles ils correspondent.

TP 8 (préparation du DST) : les débranchements et la notion de pile

Objectifs pédagogiques

La séance est consacré à l'étude de la pile. Elle consiste à écrire un programme principal (*i.e.* sans fonction) puis à le transformer en sous-programme avec un passage de paramètres par valeur et par registre, puis par la pile. La dynamique de la pile sera visualisée avec le débogueur TD.exe.

Exercice 1 : écriture d'un programme à partir d'un pseudo-code

Q1. Soit le pseudo-code suivant :

```
début
    n = 3;
    a = 0;
    c = n
    if (c > 0) {
    b = 2;
    do {
    a = a + b;
    b = b + 2;
    c = c – 1;
    }
    while (c != 0);
    }
    resultat = a;
fin
```

sachant que n et resultat sont deux variables mémoires de type `unsigned int` (format 16 bits). Ecrivez le programme correspondant.

Q2. Transformez le programme de la question précédente pour que le passage de paramètres soit de type « par valeur » et « par registre ».

Q3. Transformez le programme de la question précédente pour que le passage de paramètres soit de type « par valeur » et « par la pile ».

Q4. Que fait ce programme ?

Q5. Calculez l'Adresse Effective (AE) lors de l'exécution de l'instruction call (*cf.* fichier listing) ?

Q6. Observez avec Turbo Debugger en mode pas-à-pas l'évolution du pointeur de pile (cadre pile en bas à droite de la fenêtre CPU) à chaque empilement, dépilement, appel et retour de sous-programme.

TP 9 : arithmétique (supplément)

Objectifs pédagogiques

L'objet de ce TP est de renforcer les notions de langage d'assemblage des séances précédentes, en particulier les débranchements par sauts conditionnel et le passage de paramètres d'un sous-programme.

Exercice 1 : la division euclidienne

Il s'agit d'implémenter une division euclidienne par soustractions successives. L'équation de base avec le dividende N, le diviseur D, le quotient Q et le reste R, est :

$$N = D \times Q + R$$

avec N, Q, R $\in \mathbb{N}$ et D $\in \mathbb{N}^*$.

Q1. Quelle est la condition d'arrêt de la boucle qui réalise la soustraction ?

Q2. Donnez la définition des données avec leur valeur initiale.

Q3. Ecrivez maintenant le code en vous aidant d'un pseudo-code.

Exercice 2 : calcul d'une suite de puissances de 2

On décide de calculer les puissances de 2.

$y = 2^x$ avec $x \in \{0, ..., 15\}$

Q1. Quelle est la représentation numérique que vous choisissez ?

Q2. Analysez le problème en remplissant le tableau suivant :

Variables	Valeur min.	Valeur max.	Format min.	Format retenu pour le problème
x				
y				

Tableau Q9-1 : Etude du format des variables

Q3. Déclarez deux variables x et y initialisées respectivement à 15 et 0. Ecrire un programme prêt à être assemblé qui calcule 2^x et qui range le résultat en mémoire dans la variable y.

Q4. Soit la déclaration d'un tableau dans le segment de données :

```
TAB DW 16 DUP (0)
```

Ecrire un programme qui appelle 16 fois un sous-programme calculant les puissances de 2 croissantes et qui les rangent dans le tableau déclaré ci-dessus dans l'ordre croissant des adresses. Le passage des paramètres se fera par registre et par valeur.

Répondre aux quatre questions suivantes :

nombre de paramètre(s) entrant(s) ?
nombre de paramètre(s) sortant(s) ?
registre(s) paramètre entrant ?
registre(s) paramètre sortant ?

Exercice 3 : multiplication signée de type octet

La représentation de travail sera la représentation module et signe. Pour les données à multiplier, le format de travail sera n = 9 bits, 1 bit est dédié au signe et 8 bits à la valeur absolue. Pour faciliter les manipulations, la donnée sera cadrée sur 16 bits (figure 1). Pour le résultat, le format sera n = 17, cadré sur 24 bits selon le même principe.

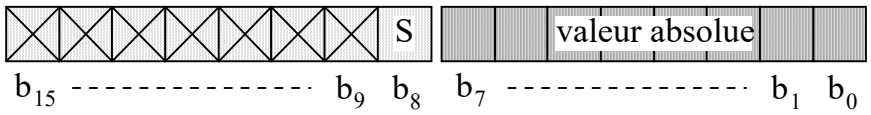

Figure Q9-1 : Représentation module et signe

Q1. Analysez le problème en remplissant le tableau suivant, les valeurs étant exprimées en base dix :

Variable	Valeur min.	Valeur max.
A		
B		
R = A × B		

Tableau Q9-2 : Etude du format des variables

Q2. Soit, en assembleur, la déclaration des données suivantes :

```
signe_A   DB 00000001b
A         DB 255
signe_B   DB 0
B         DB 0FFh
```

Q2.a. Ecrivez un pseudo-code pour résoudre la multiplication de deux nombres A et B, dans la représentation décrite ci-dessus, **par additions successives**. Dissociez bien le traitement du signe de celui de la valeur absolue. N'oubliez pas d'optimiser votre programme (valeur du compteur de boucle, etc.).

Q2.b. Ecrire le programme prêt à être assemblé, **correspondant à votre pseudo-code**, et rangeant le résultat, préalablement déclaré, en mémoire derrière les déclarations de A et B.

TP 10 : la manipulation des chaînes de caractères (supplément)

Objectifs pédagogiques

L'objet de ce TP est d'étudier la manipulation des chaînes de caractères.

Exercice 1 : la manipulation des chaînes de caractères

Les micro-processeurs de la famille 80X86 possèdent des instructions spécialisées dans la manipulation des chaînes de caractères. Pour un transfert unitaire de type octet, voici les instructions concernées :

- LODSB (chargement d'un élément de la chaîne de caractères),
- STOSB (transfert d'un élement vers une chaîne de caractères),
- MOVSB (transfert d'une chaîne),
- CMPSB (comparaison de deux chaînes de caractères),
- SCASB (scrutation et test d'une chaîne de caractères).

Démarche de travail

Le travail demandé est le suivant :

- définition des variables (type, initialisation, etc.),
- analyse du problème sous la forme d'un pseudo-code,
- écriture du programme,
- assemblage, test et déverminage.

pour tester si une phrase est un palindrome.

Rappel

Un palindrome est une phrase qui se lit à l'endroit, comme à l'envers, sans tenir compte des espaces, de l'accentuation des caractères et des signes de ponctuations. Soit la variable chaînei rangée en zone mémoire N° i. L'algorithme non optimisé devra suivre les étapes suivantes :

- afficher la chaîne de caractères N° 1 à l'écran,
- transférer cette chaîne de caractères de la zone 1 vers la zone 2,
- supprimer les blancs et les signes de ponctuation,
- afficher la zone mémoire N° 2 à l'écran,
- transférer de nouveau celle-ci de la zone 2 vers la zone 3, mais cette fois-ci en l'inversant,
- afficher la chaîne de caractères de la zone 3 à l'écran,
- vérifier que la phrase est un palindrome en comparant la chaîne de la zone 2 avec celle de la zone 3,
- afficher un texte indiquant le résultat du test.

Remarques

- N'oubliez pas les caractères de tabulations,
- Les données seront rangées en mémoire, il n'y aura donc pas d'initialisation au clavier.

Complément

1°) Travail complémentaire :

- rechercher une sous-chaîne dans la chaîne précédente,
- affichage du texte : "chaîne trouvée ou non trouvée",
- dans le cas d'une détection positive, remplacer cette sous-chaîne par des blancs,
- afficher la phrase,

- supprimer les blancs,
- afficher de nouveau la phrase.

2°) Amélioration de l'algorithme de détection du palindrome : utilisez la même chaîne de caractères à l'aide deux pointeurs, l'un pointant le début de la phrase et s'incrémentant et l'autre pointant la fin de la phrase et se décrémentant.

Note

En ne tenant pas compte des espaces, de l'accentuation et de la forme capitale des caractères, de l'apostrophe et des signes de ponctuations, voici quelques phrases pour tester votre programme :

- Serge lava le grès.
- Et il a été alité.
- Karine alla en Irak.
- Il a sali.
- Narine alla en Iran.
- Un ému a son os au menu.
- Alec, si le minaret te ranime, lis cela.
- L'avaleur lama a mal rue laval.
- Eric notre valet alla te laver ton cire.
- Un lit a tort.