

Architecture (ASR1 et ASR2)

TP N° 6

1 Objectifs :

- Maîtriser la pile, les sous-programmes, le passage de paramètres, la sauvegarde de paramètres.
- Utiliser la pile lors d'appels de programmes imbriqués.

2 Organisation du TP :

Le travail effectué durant cette séance sera à rendre à votre responsable de TP par email impérativement à la fin de la séance. Le sujet de l'email sera [2012][Architecture][TP6] Nom Prénom .

Le TP comporte plus d'exercices que vous ne pourrez normalement en faire pendant la séance. Ces exercices supplémentaires-vous sont donnés à titre d'entraînement et votre responsable de TP pourra vous les corriger si vous les lui rendez. Attention, corriger vos exercices ne veut pas dire qu'il vous fournira une correction type des exercices ! Il vous indiquera des pistes pour résoudre vos problèmes et pour vous aider à trouver une solution correcte qui fonctionne.

3 Exercice 1 : Le PGCD (Obligatoire)

Ecrivez le programme qui calcule le PGCD de 2 nombres entiers positifs sur 8 bits. Les nombres seront lus au clavier. On notera que l'instruction `a%b` retourne le reste de la division de `a` par `b`. On pensera à afficher le résultat de manière lisible.

Le programme se décompose de la manière suivante :

- Lecture des nombres saisis au clavier par l'utilisateur.
- Calcul du pgcd.
- Affichage d'un message donnant le résultat.

Voici l'algorithme pour le calcul du pgcd :

```
function pgcd(a,b) {  
    if (b==0) {  
        return a  
    } else {  
        return pgcd(b,a%b)  
    }  
}
```

Les fonctions de lecture clavier et d'affichage écran seront dans un fichier `io.asm`. La fonction de calcul du pgcd sera dans un fichier `math.asm`. Le reste du programme sera dans un fichier `tp6ex1.asm`

4 Exercice 2 : Palindrome (Obligatoire)

Ecrivez un programme qui vérifie si une chaîne saisie par un utilisateur (celle-ci se terminera par `$`) est bien un palindrome (le `$` ne sera pas compté comme faisant partie de la chaîne).

Le programme se décompose de la manière suivante :

- Lecture de la chaîne de caractères (taille max 64 octets) saisie au clavier par l'utilisateur.
- Vérification si la chaîne est bien un palindrome
- Affichage d'un message contenant la chaîne et indiquant si celle-ci est bien un palindrome.

Comme dans l'exercice précédent les fonctions d'entrées/sorties seront dans le fichier `io.asm` et on séparera le programme principal (`tp6ex2.asm`) de la fonction palindrome (`string.asm`).

Architecture (ASR1 et ASR2)

TP N° 6

5 Exercice 3 : Factorielle (Conseillé)

Le programme calcule $n!$ avec un nombre compris entre (0 et 8) qui sera lu au clavier. On pensera à afficher le résultat de manière lisible et à effectuer les calculs sur 16 bits.

```
function facto(a){
    if (a < 1) {
        return 1
    } else {
        return a*facto(a-1)
    }
}
```

6 Exercice 4 : Calcule la somme (Optionnel)

Le programme calcule la somme des n premiers nombres entiers naturels (codés sur 16 bits) d'un tableau. On supposera disposer d'un tableau sous la forme d'une variable globale. Ce tableau sera initialisé à l'avance de 20 valeurs différentes. L'algorithme récursif est le suivant :

```
Int16 SomNb (unsigned int16 n) {
    if ( n > 1 ) then
        return ( SomNb(n-1) + TAB(n-1)) ;
    else
        return TAB(0) ;
}
```

On respectera les règles de passage de paramètres suivantes :

- Le nombre d'entiers à sommer sera passé par la pile (passage par valeur) ;
- Le résultat sera également passé par valeur et se trouvera sur le sommet de pile lors du retour de chaque appel (il faut laisser un emplacement libre sur la pile juste avant l'appel).

7 Exercice 5 : Variables, Registres et Pile (Conseillé)

Ecrire un programme qui fait la somme de 5 entiers et respecte les contraintes ci-dessous (Regardez les fonctions de l'interruption 21H du DOS pour les différentes entrées/sorties) :

7.1 Utilisation de variables

- Chaque entier est dans une variable définie sur 8 bits (et initialisés à la création) dont la valeur ne sera pas modifiée durant tout le programme. Une seule lecture sera autorisée pour chaque variable.
- Le résultat de la somme sera sauvegardé dans une variable 16 bit (non initialisés à la création) nommée RES_SOMME. Une seule écriture sera autorisée dans résultat.

7.2 Utilisation des registres et lecture au clavier

- Chaque entier sera lu au clavier.
- Le résultat de la somme sera sauvegardé dans une variable 16 bit (non initialisés à la création) nommée RES_SOMME. Une seule écriture sera autorisée dans résultat.
- Aucune autre variable ne sera utilisée.
- Les instructions LOOP et CMP sont interdites.

7.3 Utilisation de LOOP et affichage à l'écran

- Chaque entier sera lu au clavier.
- Le résultat de la somme sera affiché à l'écran.
- Aucune variable ne sera utilisée
- On utilisera une boucle (LOOP) pour faire les différentes lectures

Architecture (ASR1 et ASR2)

TP N° 6

8 Exercice 6 : Moyenne olympique (Conseillé)

Le but de cet exercice est d'écrire un programme faisant la moyenne olympique de notes. La moyenne olympique d'une liste de nombres est la moyenne de la liste obtenue en lui retirant son plus grand et son plus petit élément.

Le programme devra respecter les consignes suivantes :

- Les notes seront comprises entre 0 et 20 et seront lues au clavier.
- Le programme devra gérer un nombre de notes compris entre 3 et 99.
- La lecture d'une note devra se faire dans une procédure *Lecture*.
- La recherche de la note MAX devra se faire dans une procédure *Max*.
- La recherche de la note MIN devra se faire dans une procédure *Min*.
- Le résultat devra être affiché à l'aide d'une procédure *Affichage*.

9 Exercice 7 : Chaines de caractères, Branchements et filtre (Optionnel)

Ecrire un programme en assembleur qui va utiliser l'interruption 21H du DOS et qui affiche la chaîne de caractères : « Est-ce que nous sommes vendredi ? (o/n) » et attend un caractère de réponse.

- Si vous tapez 'o', le programme affichera « Bon Week-end ! »
- Si vous tapez 'n', le programme affichera « Bonne semaine ! »
- Sinon il affichera « Erreur ...Veuillez répondre correctement ! »

Le programme devra générer les réponses en minuscule (o/n) et en majuscule (O/N) de manière identique.

10 Exercice 8 : Tri par insertion (Optionnel)

Ecrire le programme¹ suivant en assembleur :

```
#define MAX 10
void insertion(int t[MAX]) {
    /* Spécifications externes : Tri du tableau t par insertion séquentielle */
    int i,p,j;
    double x;
    for (i = 1; i < MAX; i++) {
        /* stockage de la valeur en i */
        x = t[i];
        /* recherche du plus petit indice p inférieur à i tel que t[p] >= t[i] */
        for(p = 0; t[p] < x; p++)
            /* p pointe une valeur de t supérieure à celle en i */
            /* décalage avant des valeurs de t entre p et i */
            for (j = i-1; j >= p; j--) {
                t[j+1] = t[j];
            }
        /* insertion de la valeur stockée à la place vacante */
        t[p] = x;
    }
}
```

¹ Une mise en œuvre simple du tri par insertion sur un tableau de nombres réels en C :
http://fr.wikipedia.org/wiki/Tri_par_insertion

Architecture (ASR1 et ASR2)

TP N° 6

- Vous déclarez MAX comme une constante et t comme un tableau de 10 octets initialisé.

11 Exercice 9 : Cryptographie (Optionnel)

Nous allons étudier un programme de cryptage de type DES (Data Encryption Standard) mais en beaucoup plus simple. Pour plus de détail sur DES consulter cette page : <http://www.commentcamarche.net/crypto/des.php3>

Détail de l'algorithme à écrire :

