Feuille de travaux pratiques n° 4
Assembleur MIPS

Pour tous les exercices, on commencera par écrire l'algorithme demandé en C/C++ et l'on traduira ensuite ce code en suivant les schémas vus en cours.

Programmes simples

Exercice 4.1

1. Récupérer sur *madoc* le programme `leap.asm` et le tester dans *MARS* ;
2. Corriger le programme pour qu'il détermine correctement les années bissextiles postérieures à 1582.

Exercice 4.2

1. Récupérer sur *madoc* le programme `dow.asm` et le tester dans *MARS* ;
2. Traduire le programme assembleur en un programme C/C++.

Exercice 4.3

Écrire un programme demandant un entier positif sur 32 bits à l'utilisateur, et lui indiquant s'il est plus grand ou plus petit qu'un nombre à découvrir stocké dans le segment de données. Le programme devra afficher le nombre de coups ayant été nécessaires pour trouver la bonne valeur.

Exercice 4.4

Écrire un programme affichant à l'écran le nombre de « `e` » que contient une chaîne de caractères stockée dans le segment de données.

Fonctions et procédures

Exercice 4.5

En utilisant l'algorithme d'Euclide, écrire la fonction `int pgcd(int a, int b)` en assembleur.
Rappel :

$$\text{pgcd}(a,b) = \begin{cases} a & \text{si } b == 0, \\ \text{pgcd}(b, \text{rem}(a,b)) & \text{sinon} \end{cases}$$

avec la précondition $a \geqslant b$.
On testera la fonction avec plusieurs jeux d'essais.

Exercice 4.6

Soit la fonction `fibo` définie de la façon suivante :

$$\begin{cases} \text{fibo}(0) & = 0, \\ \text{fibo}(1) & = 1, \\ \text{fibo}(n) & = \text{fibo}(n-1) + \text{fibo}(n-2), \quad \forall n \geqslant 2. \end{cases}$$

Écrire un programme MIPS pour cette fonction en utilisant les conventions classiques d'appels de procédures.

Exercice 4.7

Écrire en assembleur MIPS la procédure `void minmax4(int a, int b, int c, int d, int *m,int *M)` prenant en entrée quatre entiers sur 32 bits et retournant dans les deux derniers paramètres passés par adresse le minimum et le maximum de ces quatre nombres.
Pour cela, on écrira d'abord une procédure `void minmax2(int u, int v, int *mini, int *Maxi)`, semblable à `minmax4()`, mais ne prenant en entrée que deux valeurs entières. C'est ensuite `minmax2()` que l'on invoquera pour obtenir le résultat de `minmax4()`.

Exercice 4.8

Traduire en assembleur MIPS le programme C++ ci-dessous implémentant la méthode du crible d'Eratosthène pour déterminer la liste des nombres premiers entre 0 et 100. On prendra soin d'utiliser un tableau d'*octets* pour coder la variable `premier` en assembleur.

```
1  #include <iostream>
2  #include <stdint.h>
3
4  int main(void)
5  {
6    const uint32_t N = 100;
7    bool premier[N+1];
8
9    // Initialisation: par défaut, tous les entiers sauf 0 et 1
10   // sont considérés premiers.
11   premier[0] = premier[1] = false;
12   for (uint32_t i = 2; i <= N; ++i) {
13     premier[i] = true;
14   }
15
16   // Criblage
17   for (uint32_t i = 2; i <= N/2; ++i)
18     if (premier[i]) // 'i' est premier: élimination des multiples
19       for (uint32_t j = i*i; j <= N; j+=i)
20         premier[j] = false;
21
22   // Affichage
23   for (uint32_t i = 2; i <= N; ++i)
24     if (premier[i])
25       std::cout << i << " ";
26   std::cout << std::endl;
27
28   return 0;
29 }
```

Exercice 4.9 (Fonction `carambole()`)

1. Écrire en assembleur MIPS la fonction récursive `int carambole(int a,int b)` définie de la façon suivante (les règles s'appliquent dans leur ordre d'apparition et sont mutuellement exclusives) :

$$\text{carambole}(x,y) = \begin{cases} y+1 & \text{si } x = 0 \\ \text{carambole}(x-1,1) & \text{si } y = 0 \\ \text{carambole}(x-1,\text{carambole}(x,y-1)) & \text{sinon} \end{cases}$$

   On respectera les conventions de passage de paramètres. On considèrera qu'un `int` est codé sur 32 bits.

2. Écrire une fonction `main()` testant la fonction `carambole()` avec les couples $(2,22)$ et $(3,5)$, et affichant le résultat à l'écran sous la forme suivante :

```
carambole(2,22) = 47
carambole(3,5) = 253
```