Debug

Nous utilisons pour ce test un petit programme d'un seul module.
Testons-le avec Turbo C++ Version 3.0

tstSomme.c

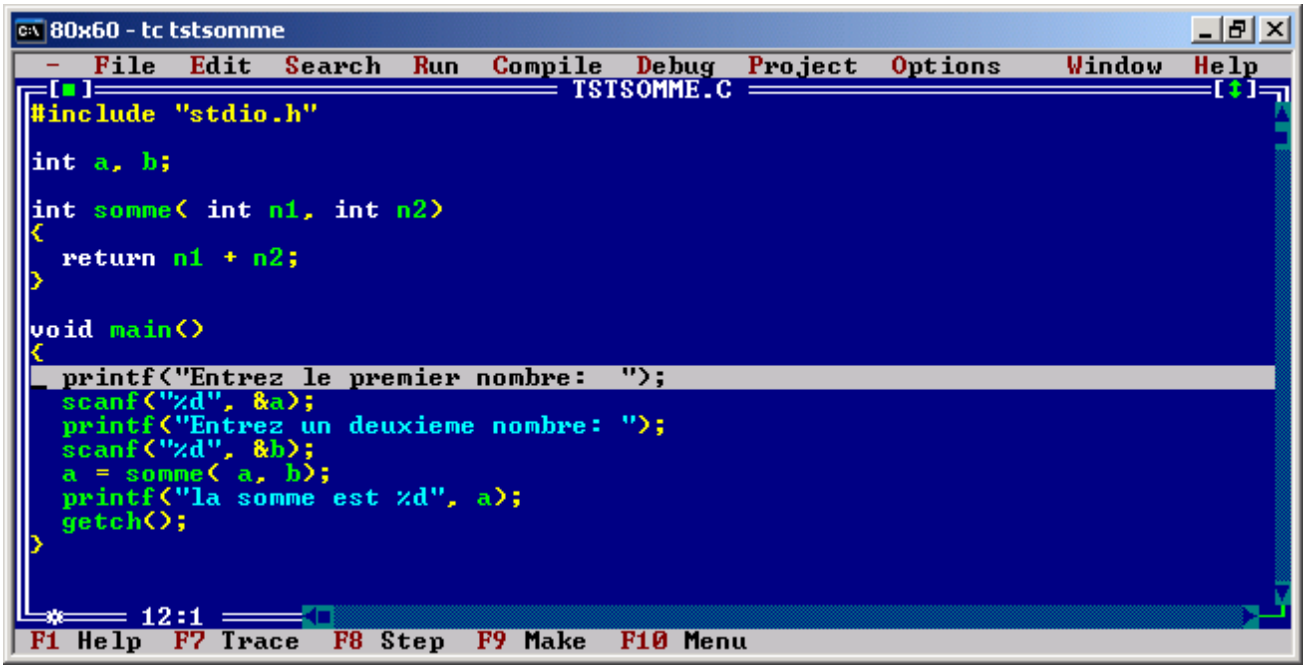
```
#include "stdio.h"

int a, b;

int somme( int n1, int n2)
{
  return n1 + n2;
}

void main()
{
  printf("Entrez le premier nombre:  ");
  scanf("%d", &a);
  printf("Entrez un deuxieme nombre: ");
  scanf("%d", &b);
  a = somme( a, b);
  printf("la somme est %d", a);
  getch();
}
```

Demander la production de l'exécutable (**C**ompile > **M**ake)
L'environnement de développement est à la fois un éditeur et un debugger. Tout se passe comme si on était en présence d'un langage interprété. La seule différence est qu'il a fallu demander la construction de l'exécutable avant de pouvoir lancer la première instruction.
Lançons l'exécution du programme pas à pas de manière à voir se dérouler les instructions une à une. (**R**un > **T**race into F7)
Les trois raccourcis intéressants sont **F7** pour avancer instruction par instruction et **ALT-F5** pour visualiser l'écran "utilisateur" autrement dit celui ou tourne l'application.



Agrandissez votre fenêtre pour pouvoir y afficher plus d'informations :

Option > **E**nvironment > **P**reference… > cochez l'option « **Screen size » ( ) 43/50 lines** > [**OK**]

Une autre fonction intéressante de debug est d'afficher les données en cours d'exécution. Ouvrez pour ce faire la fenêtre de « Watch » **W**indow > **W**atch
Testez la commande **D**ebug > **W**atches > **A**dd watch... ou son raccourci **Ctrl F7** pour saisir les noms des variables que vous voudriez voir évoluer.

Testez le programme pas à pas avec la touche F7 puis au moment d'entrer dans la fonction somme voyez l'effet de la commande **Ctrl F3** (= **D**ebug > **C**all stack…) Elle permet de retracer les appels de fonction en indiquant en outre les valeurs passées comme paramètres.

Debug avec turbo debugger

Le turbo debugger est un programme complémentaire lui aussi produit par Borland et qui permet le test de programmes conçus dans différents langages. Nous l'utiliserons pour des programme écrit s en C ou en assembleurs.

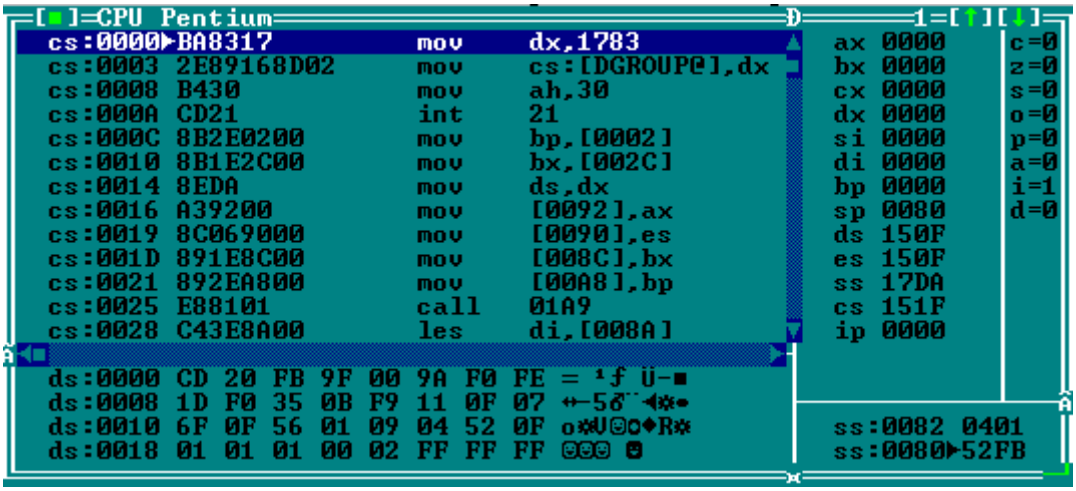
Le Turbo Debugger est un programme qui nécessite la présence de deux exécutables : TD.exe et RTM.exe. Il faut donc que ces fichiers se trouvent dans l'un des répertoires spécifié par la variable d'environnement PATH. La dernière condition pour que le debugger puisse se référer au code source est d'utiliser une version du linker pas trop ancienne. Nous utilisons la version 7 de TLink.exe au lieu de la version initiale de TC V3.0

Voici la commande en ligne à taper pour avoir en une seule commande la compilation et l'édition de lien :

```
TCC -v TstSomme.c
```

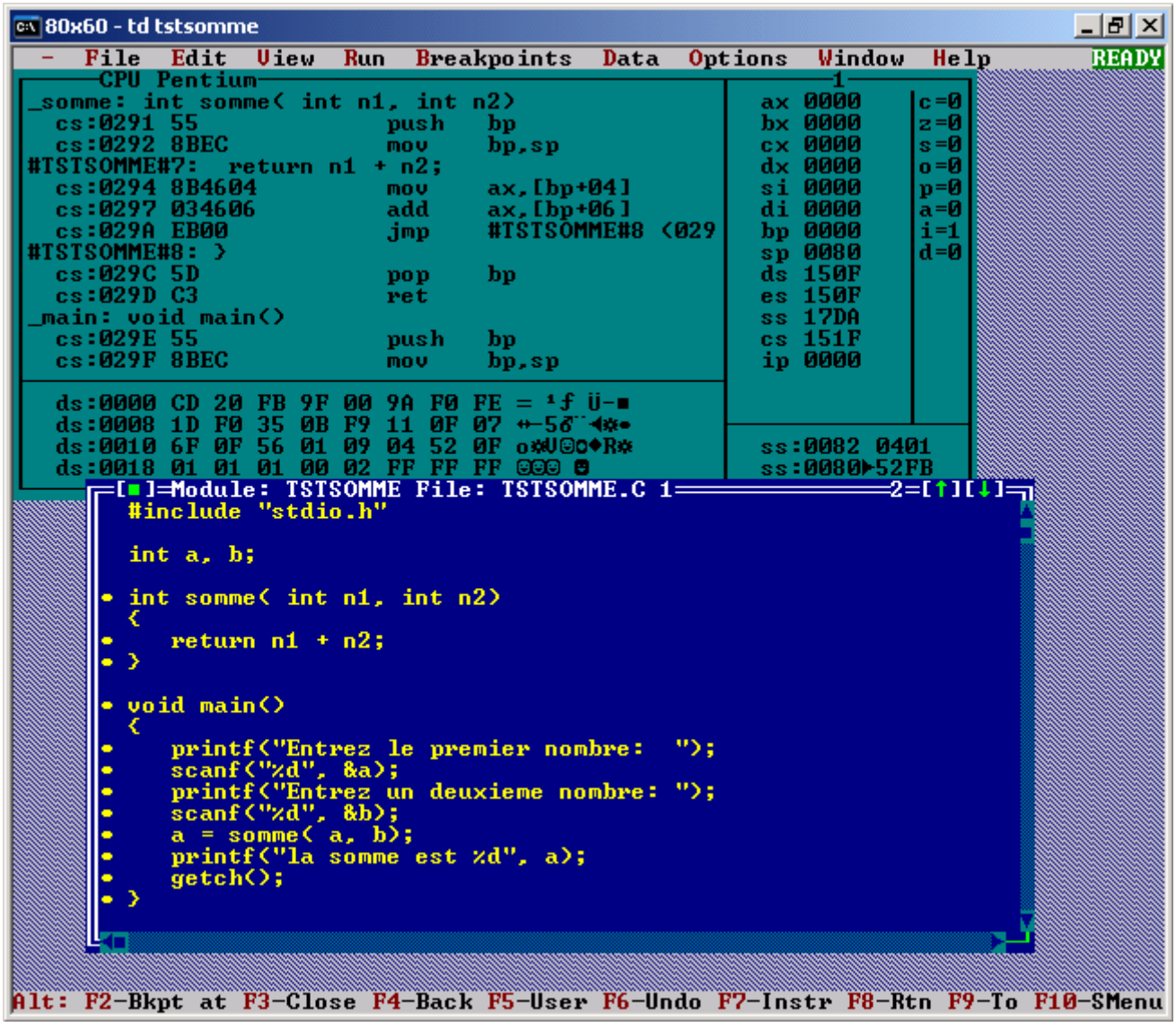
Lancer le debugger TD : `TD TstSomme`

Au départ, une seule fenêtre, dite fenêtre "CPU", affiche le segment de code de l'application. On y voit les instructions en assembleur et leurs codes en hexadécimal. Plus bas dans la même fenêtre figure une partie du segment de données tandis qu'à droite on peut voir les états des registres.



Pas facile évidemment de s'y retrouver dans ce charabia !

Il est heureusement possible de faire apparaître le code source, pressez la touche F3 Demandons l'affichage du module source F3 (ou ALT-View > Module ...) puis sélectionner le module TSTSOMME
Une seconde fenêtre affiche le code source en langage C. Redimensionner cette fenêtre et placer la de sorte à voir simultanément le code source en C et la fenêtre « CPU »



Si on lance le programme pas à pas [**F7**] on s'aperçoit qu'il faut beaucoup d'instructions avant d'entrer dans le module main( ). Plaçons le curseur dans le module source sur la ligne main() et demander l'exécution jusqu'à cet endroit (**R**UN > **G**o to cursor **F4**)

A chaque fois que l'on presse la touche F7 on avance maintenant d'une ligne en code C. Allons ainsi jusqu'au début de la fonction somme( ).

Voyons maintenant en détail et en assembleur comment se passe la fonction somme. Cliquez pour cela dans la fenêtre qui affiche le code machine de cette fonction. La touche F7 fera avancer instruction par instruction.

Remarquer comment les paramètres sont passés sur la pile. Le pointeur BP sert à remonter dans la pile pour y reprendre les arguments passés à la fonction. En fin de fonction le résultat, un entier, est simplement retourné par l'accumulateur AX. Remarquer aussi l'instruction en `CS:203,` ce saut est inutile, c'est le résultat d'une compilation qui n'est pas optimisée.