

Liste d'exercices 1 : les nombres entiers signés (1 séance)

Notions à considérer pour pouvoir faire les exercices

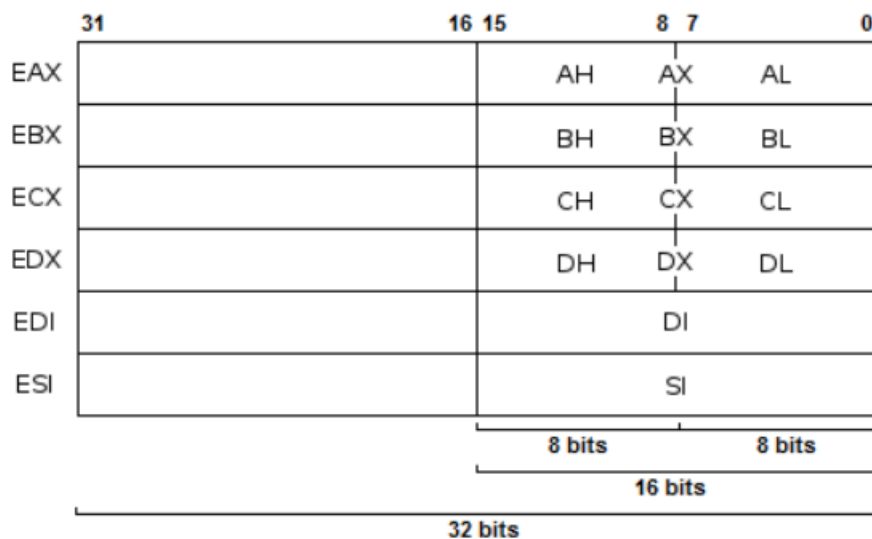
Structure d'un programme contenant de l'assembleur :

```
... // endroit où déclarer les variables

int main(void)
{
    _asm
    {
        ... // endroit où placer les instructions en assembleur
    }

    return 0;
}
```

Organisation des registres généraux :



Ces registres vont servir à conserver temporairement des résultats.

Il existe encore 2 registres généraux : ESP et EBP, mais on les utilisera plus tard.

Types de données pour les nombres entiers signés :

Types	Tailles	Plages de valeurs
char	8 bits	$[-2^7, \dots, +2^7 - 1]$
short	16 bits	$[-2^{15}, \dots, +2^{15} - 1]$
int	32 bits	$[-2^{31}, \dots, +2^{31} - 1]$

$-2^7 = -128$; $+2^7 - 1 = +127$; $-2^{15} = -32768$; $+2^{15} - 1 = +32767$; $-2^{31} = -2147483648$; $+2^{31} - 1 = +2147483647$.

Instructions sur entiers :

- *MOV opd, ops* effectue $opd = ops$
- *ADD opd, ops* effectue $opd = opd + ops$
- *SUB opd, ops* effectue $opd = opd - ops$
- *IMUL opd, ops* effectue $opd = opd * ops$

Contraintes imposées par le processeur :

- Les opérandes *ops* et *opd* des instructions à 2 opérandes mentionnées ci-dessus doivent avoir la même taille (par exemple, *mov eax, bl* est impossible).
- L'opérande destination *opd* doit pouvoir stocker le résultat de l'exécution de l'instruction. Il ne peut donc être une valeur numérique (par exemple, *mov 5, eax* est impossible).
- Un seul des 2 opérandes peut être un emplacement en mémoire (par exemple, *mov i, j* est impossible).

Contraintes imposées par le langage C :

- Les opérations sont réalisées sur des entiers de 32 bits.
- On ne modifie pas les variables figurant à droite de l'opérateur d'affectation $=$ (par exemple, pour $i = j * k + 2$, les variables j et k ne peuvent être modifiées).
- Les opérateurs n'ont pas tous la même priorité.

<i>Priorités</i>	<i>Opérateurs en C</i>	<i>Instructions en assembleur</i>
+++	*	imul
++	+, -	add, sub
+	=	mov

Si plusieurs opérateurs avec la même priorité se trouvent dans une instruction d'affectation en C et que des parenthèses ne sont pas utilisées, alors l'application de ces opérateurs se fait simplement de gauche à droite.

Exemple :

`int i, j;`

<i>Langage C</i>	<i>Assembleur</i>
<code>i = j - 2 * i + 5;</code>	<code>mov ebx, 2 imul ebx, i mov eax, j sub eax, ebx add eax, 5</code>

	mov i, eax
--	------------

On a commencé par faire la multiplication. Ensuite, en allant de gauche à droite, on a fait la soustraction, puis l'addition. Enfin, on a affecté le résultat à la variable i.

Exemple :

`int i, j;`

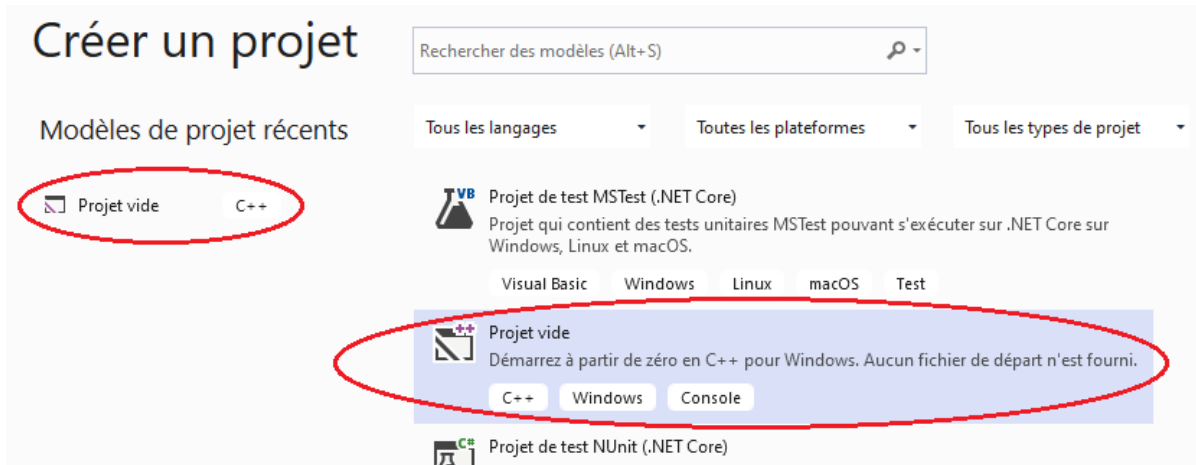
<i>Langage C</i>	<i>Assembleur</i>
<code>i = j * 2 - i * 5;</code>	<pre> mov eax, j imul eax, 2 mov ebx, i imul ebx, 5 sub eax, ebx mov i, eax </pre>

On a commencé par faire les 2 multiplications. Ensuite, on a fait la soustraction. Enfin, on a affecté le résultat à la variable i.

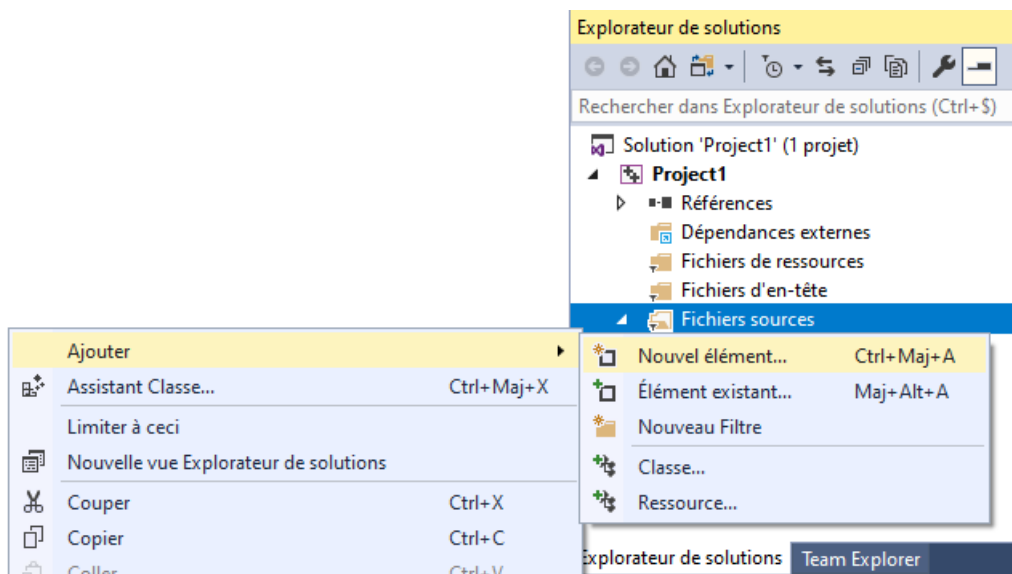
Exercices

Exercice 1 : créez un projet dans Visual Studio :

1. Sélectionnez, dans le menu Fichier, l'option Nouveau → Projet.
2. Dans la fenêtre Créer un projet, sélectionnez, à gauche, Projet vide C++ et, à droite, Projet vide.

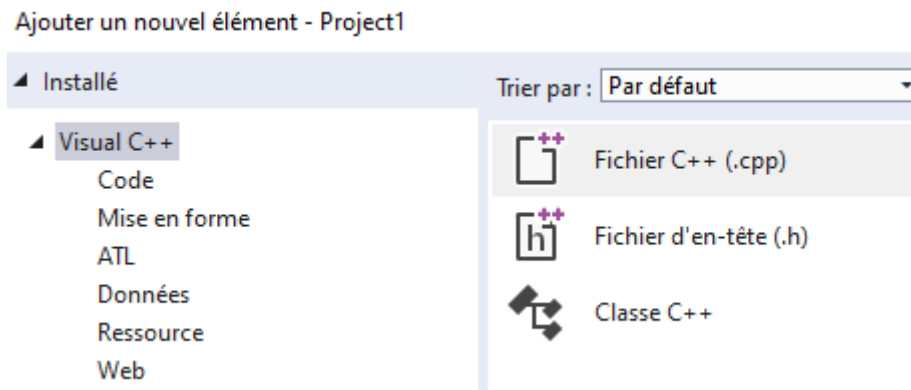


3. Dans la fenêtre Configurer votre nouveau projet, entrez le nom du projet, indiquez l'emplacement où placer ce projet (dans le dossier créé à l'étape 5 de la section 2.2 qui est d:\projet_vs dans notre exemple), puis cliquez sur Créer.
4. La fenêtre Explorateur de solutions affiche la structure en dossiers du projet. Pour ajouter un fichier source dans ce projet, cliquez avec le bouton droit de la souris sur le dossier Fichiers sources et sélectionnez l'option Ajouter → Nouvel élément.



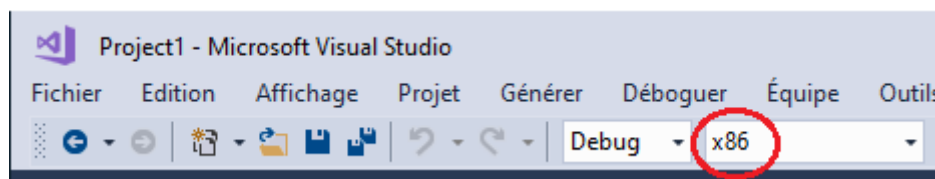
Si la fenêtre Explorateur de solutions n'est pas visible, alors, pour l'afficher, il faut sélectionner l'option Explorateur de solutions dans le menu Affichage.

5. Dans la fenêtre Ajouter un nouvel élément, sélectionnez Visual C++ à gauche et Fichier C++ (.cpp) à droite, entrez le nom du fichier source, donnez-lui l'extension .c, puis cliquez sur le bouton Ajouter.

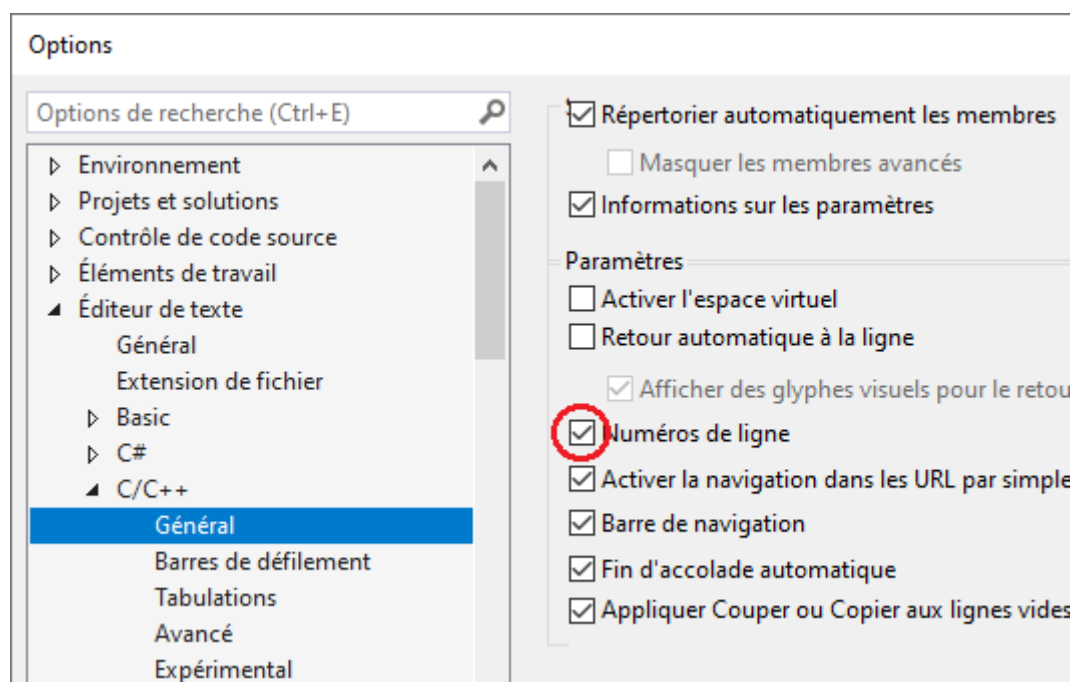


Maintenant, un fichier source ne contenant encore rien est attaché au projet.

6. Vérifiez, dans la barre des icônes, que l'option **x86** est sélectionnée dans la boîte combo. Ceci indique au compilateur que le code à générer, lors de la compilation, sera du code 32 bits.



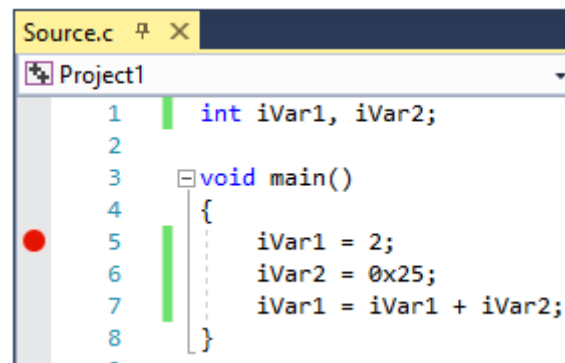
7. Si vous souhaitez afficher le numéro des lignes dans le fichier source, sélectionnez Options dans le menu Outils. Dans la fenêtre Options, sélectionnez à gauche Éditeur de texte, C/C++, et Général, puis cochez Numéros de ligne. Ensuite, cliquez sur le bouton Ok.



Exercice 2 : testez les 2 exemples d'usage du débogueur suivants :

Exemple 1 :

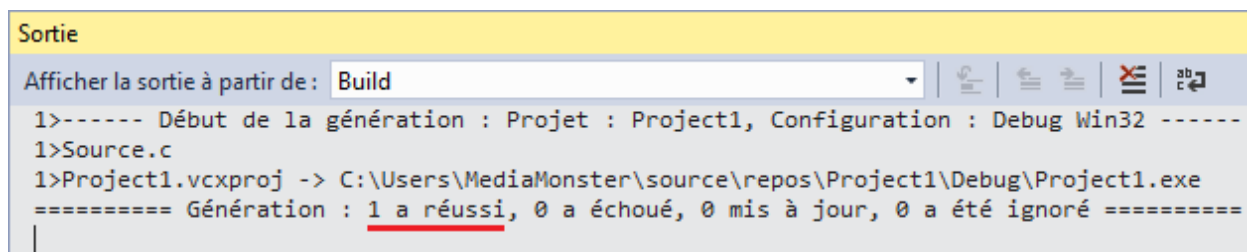
1. Dans le fichier source ajouté au projet, encodez le programme suivant et placez un point d'arrêt sur l'instruction `iVar1 = 2` :



```
Source.c  [icon] X
Project1
1  int iVar1, iVar2;
2
3  void main()
4  {
5      iVar1 = 2;
6      iVar2 = 0x25;
7      iVar1 = iVar1 + iVar2;
8  }
```

On place le point d'arrêt (point rouge) en cliquant dans la zone grisée au début de la ligne 5. Le **point d'arrêt** est l'endroit où l'exécution du programme va devoir, temporairement, être interrompue, afin de permettre au programmeur de visualiser la valeur stockée dans certaines variables et/ou dans certains registres du processeur.

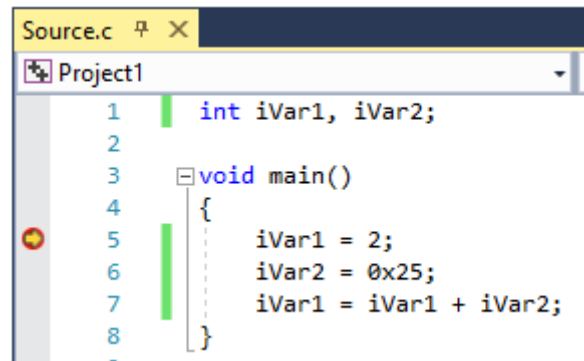
2. Générer le fichier exécutable en sélectionnant l'option Générer la solution dans le menu Générer. On sait si la génération du fichier exécutable a réussi en observant le contenu de la fenêtre Sortie :



```
Sortie
Afficher la sortie à partir de : Build
1>----- Début de la génération : Projet : Project1, Configuration : Debug Win32 -----
1>Source.c
1>Project1.vcxproj -> C:\Users\MediaMonster\source\repos\Project1\Debug\Project1.exe
===== Génération : 1 a réussi, 0 a échoué, 0 mis à jour, 0 a été ignoré =====
|
```

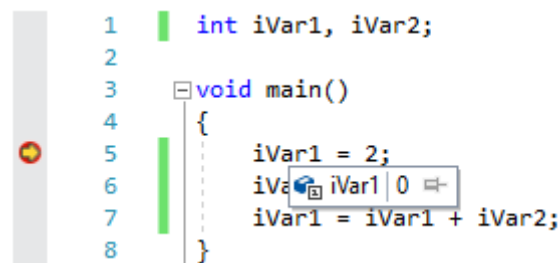
3. Pour démarrer l'exécution du programme dans le débogueur, sélectionnez l'option Démarrer le débogage dans le menu Débogueur, ou appuyez sur la touche F5, ou encore cliquez sur la flèche verte dans la barre d'icônes.

L'exécution du programme est interrompue sur le point d'arrêt. L'instruction désignée par une flèche jaune est `iVar1 = 2`. Cette instruction est la **prochaine** instruction à exécuter dès la poursuite de l'exécution du programme.



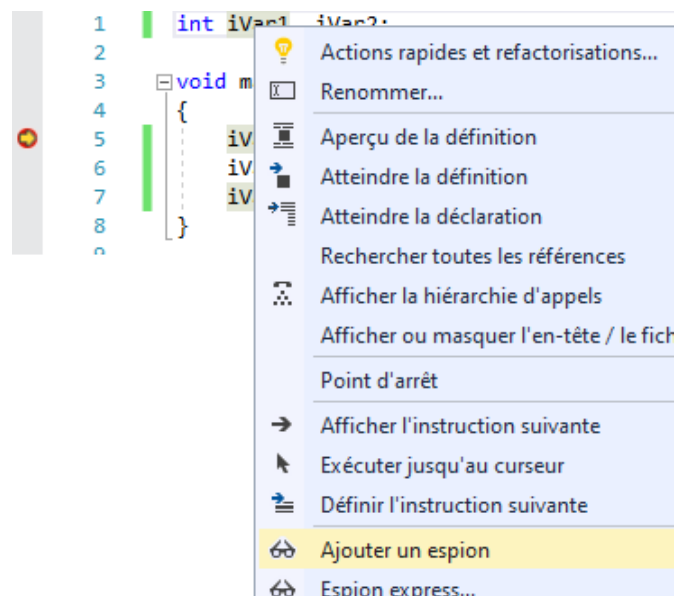
```
Source.c  [icon] X
Project1
1  int iVar1, iVar2;
2
3  void main()
4  {
5      iVar1 = 2;
6      iVar2 = 0x25;
7      iVar1 = iVar1 + iVar2;
8  }
```

4. Pour visualiser le contenu d'une variable ou d'un registre, déplacez le pointeur de la souris sur cette variable sans cliquer dessus. Une petite fenêtre alors apparaît temporairement dans laquelle figure la valeur que stocke cette variable.



```
1  int iVar1, iVar2;
2
3  void main()
4  {
5      iVar1 = 2;
6      iVar1 = iVar1 + iVar2;
7      iVar1 = iVar1 + iVar2;
8  }
```


5. Il est possible de choisir explicitement les variables dont on veut suivre l'évolution du contenu. Pour ce faire, cliquez avec le bouton droit sur la variable concernée, par exemple iVar1, puis sélectionnez l'option Ajouter un espion.




```
1  int iVar1, iVar2;
2
3  void main()
4  {
5      iVar1 = 2;
6      iVar1 = iVar1 + iVar2;
7      iVar1 = iVar1 + iVar2;
8  }
```

- Actions rapides et refactorisations...
- Renommer...
- Aperçu de la définition
- Atteindre la définition
- Atteindre la déclaration
- Rechercher toutes les références
- Afficher la hiérarchie d'appels
- Afficher ou masquer l'en-tête / le fichier
- Point d'arrêt
- Afficher l'instruction suivante
- Exécuter jusqu'au curseur
- Définir l'instruction suivante
- Ajouter un espion**
- Espion express...

Une fenêtre appelée **Espion 1** apparaît. Elle montre la valeur courante en **décimal** de la variable.

Espion 1	
Nom	Valeur
 iVar1	0

On peut aussi ajouter explicitement le nom d'une variable dans la fenêtre Espion 1.

Espion 1	
Nom	Valeur
 iVar1	0
iVar2	



6. Dans la fenêtre du fichier source, appuyez à 2 reprises sur F10 pour exécuter les 2 instructions suivantes : iVar1 = 2 et iVar2 = 0x25. La flèche jaune se trouve à présent sur la ligne 7, tandis que la fenêtre Espion 1 montre le nouveau contenu des variables iVar1 et iVar2.

Source.c

Project1

```
1 int iVar1, iVar2;
2
3 void main()
4 {
5     iVar1 = 2;
6     iVar2 = 0x25;
7     iVar1 = iVar1 + iVar2;
8 }
```

Espion 1

Nom	Valeur
 iVar1	2
 iVar2	37



7. Appuyez sur F10 une seule fois. Maintenant que l'addition a été réalisée, la fenêtre Espion 1 montre que iVar1 stocke la valeur 39.

Source.c

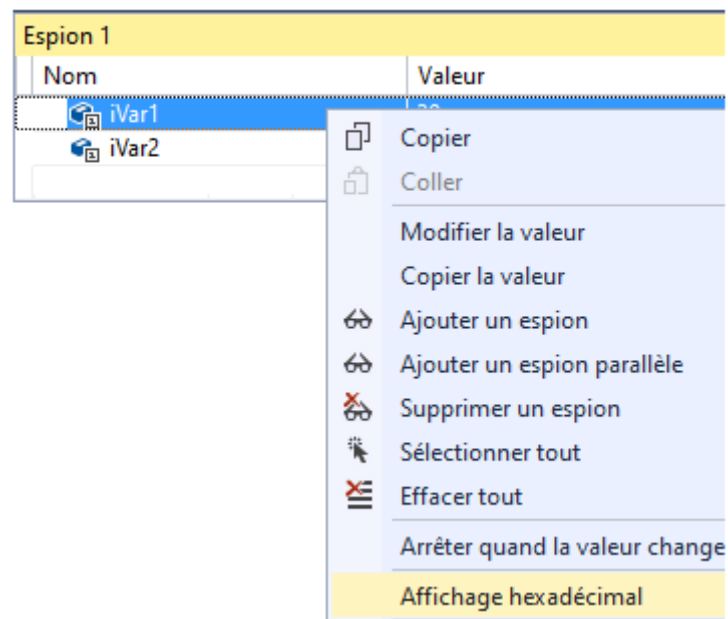
Project1

```
1 int iVar1, iVar2;
2
3 void main()
4 {
5     iVar1 = 2;
6     iVar2 = 0x25;
7     iVar1 = iVar1 + iVar2;
8 }
```

Espion 1

Nom	Valeur
 iVar1	39
 iVar2	37

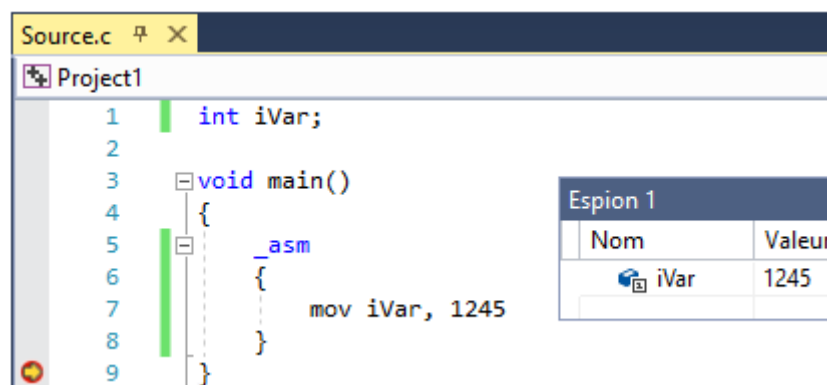
8. Pour visualiser en **hexadécimal** le contenu de variables et/ou de registres, cliquez avec le bouton droit dans la fenêtre Espion 1 et sélectionnez l'option Affichage hexadécimal.



Les valeurs sont maintenant affichées en hexadécimal :

Espion 1	
Nom	Valeur
iVar1	0x00000027
iVar2	0x00000025

Exemple 2 : entrez le contenu suivant dans le fichier source. Ensuite, placez le point d'arrêt sur la ligne 9. Puis, démarrez l'exécution du programme dans le débogueur de sorte à avoir la flèche jaune sur le point d'arrêt.



Après avoir ajouté la variable iVar dans la fenêtre Espion 1, on voit que sa valeur est bien 1245.

Quelques remarques :

- Il n'est pas nécessaire de placer ; à la fin d'une instruction en assembleur.
- On ne peut mettre qu'une seule instruction en assembleur par ligne.

- Il n'est pas nécessaire de respecter la casse pour le nom des registres et le nom des instructions. Par contre, il faut la respecter pour le nom des variables et des constantes.

Exercice 3 : écrivez et testez la séquence d'instructions en assembleur qui correspond à chaque instruction d'affectation en langage C suivante (en commentaire en vert figure la valeur à obtenir au final) :

```
int i = -36;
int j = 25;

• i = 143;           // i = 143
• i = j;             // i = 25
• i = i + j;         // i = -11
• i = i + j - 8;     // i = -19
• i = i - j + 8;     // i = -53
• i = i + j * 8;     // i = 164
• i = i * j + 8;     // i = -892
• i = 2 - i * j + 8; // i = 910
• i = j * 0x12 + i + i * 5 + 9; // i = 243
• j = j - j * i + j + i * 0x12; // j = 302
• j = i + j * 18 - i * j * 178; // j = 160614
• i = 0x12 - i + i * 5 - i * j - 8; // i = 766
• i = 0x1AB * j + i - 35 * i - j - 8 - 0x22 - i * i - 5; // i = 10531
```