ca-4. アセンブリ言語

（コンピュータ・アーキテクチャ演習）

URL: https://www.kkaneko.jp/cc/ca/index.html

金子邦彦

アウトライン

4-1 アセンブリ言語とは

4-2 逆アセンブル

4-3 Visual Studio のデバッガー

4-4 変数のアドレスの確認

4-1 アセンブリ言語とは

4-1 アセンブリ言語とは

- アセンブリ言語とは人間が理解しやすいように，マシン語（機械語）を翻訳した言語
- アセンブリ言語の命令と，マシン語の命令が，ほぼ 1 対 1 に対応

```
4D 5A 90 00 03 00 00 00-04 00 00 00 FF FF 00 00
B8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 10 01 00 00
0E 1F BA 0E 00 B4 09 CD-21 B8 01 4C CD 21 54 68
69 73 20 70 72 6F 67 72-61 6D 20 63 61 6E 6E 6F
74 20 62 65 20 72 75 6E-20 69 6E 20 44 4F 53 20
6D 6F 64 65 2E 0D 0D 0A-24 00 00 00 00 00 00 00
0F 93 F3 1B 4B F2 9D 48-4B F2 9D 48 4B F2 9D 48
24 84 36 48 48 F2 9D 48-50 6F 01 48 49 F2 9D 48
```

実行型ファイルの例

```
E:¥>ndisasm WINWORD.EXE | more
00000000  4D                dec bp
00000001  5A                pop dx
00000002  90                nop
00000003  0003              add [bp+di],al
00000005  0000              add [bx+si],al
00000007  0004              add [si],al
00000009  0000              add [bx+si],al
0000000B  00FF              add bh,bh
0000000D  FF00              inc word [bx+si]
0000000F  00B80000          add [bx+si+0x0],bh
00000013  0000              add [bx+si],al
00000015  0000              add [bx+si],al
00000017  004000            add [bx+si+0x0],al
0000001A  0000              add [bx+si],al
0000001C  0000              add [bx+si],al
0000001E  0000              add [bx+si],al
00000020  0000              add [bx+si],al
00000022  0000              add [bx+si],al
00000024  0000              add [bx+si],al
00000026  0000              add [bx+si],al
00000028  0000              add [bx+si],al
0000002A  0000              add [bx+si],al
```

アセンブリ言語の例

Pentium 系列プロセッサでのアセンブリ言語の主な命令

種類	命令	意味
データ転送と実効アドレス	**MOV**	データ転送　※　ロード，ストア，プッシュ，ポップ
	LEA	実効アドレスのロード
算術演算	**ADD**	加算
	SUB	減算
	IMUL	乗算
	IDIV	除算
	SAR, SAL	算術シフト
論理演算	**AND**	論理積
	OR	論理和
	SHR, SHL	論理シフト
比較	**CMP**	比較
	TEST	AND による比較
ジャンプ（分岐）	**JMP**	無条件ジャンプ（無条件分岐）
	J??	条件ジャンプ（条件分岐）
サブルーチン	**CALL**	サブルーチン呼び出し（サブルーチンコール）
	RET	サブルーチンからの復帰

4-2 逆アセンブル

逆アセンブル

マシン語を**アセンブリ言語**に翻訳すること

ビルド

逆アセンブル

```
#include "stdafx.h"
#include <math.h>

int main()
{
    double d = 2.0;
    printf("%f, %f ¥n", d, sqrt(d));
    return 0;
}
```

```
00000000  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ........@.......
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00  00 00 00 00 F0 00 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ........!..L.!Th
00000050  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
00000080  FF FA 97 C6 BB 9B F9 95  BB 9B F9 95 BB 9B F9 95  ................
00000090  FD CA 24 95 BB 9B F9 95  FD CA 26 95 BA 9B F9 95  ..$.......&.....
000000a0  FD CA 19 95 A8 9B F9 95  FD CA 18 95 BC 9B F9 95  ................
000000b0  66 64 32 95 B9 9B F9 95  BB 9B F8 95 83 9B F9 95  fd2.............
000000c0  B6 C9 18 95 B9 9B F9 95  B6 C9 22 95 BA 9B F9 95  ..........".....
000000d0  B6 C9 27 95 BA 9B F9 95  52 69 63 68 BB 9B F9 95  ..'.....Rich....
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000f0  50 45 00 00 4C 01 07 00  A8 13 37 55 00 00 00 00  PE..L.....7U....
00000100  00 00 00 00 E0 00 02 01  0B 01 0C 00 00 3C 00 00  .............<..
```

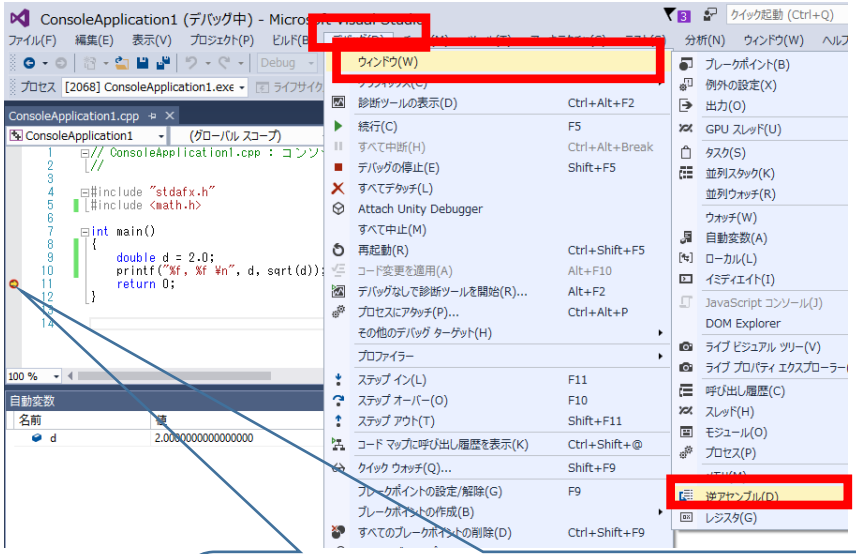
```
00F117BC  lea         edi,[ebp-0D0h]
00F117C2  mov         ecx,34h
00F117C7  mov         eax,0CCCCCCCCh
00F117CC  rep stos    dword ptr es:[edi]
    double d = 2.0;
00F117CE  movsd       xmm0,mmword ptr [__real@4000000000000000
00F117D6  movsd       mmword ptr [d],xmm0
    printf("%f, %f ¥n", d, sqrt(d));
00F117DB  sub         esp,8
00F117DE  movsd       xmm0,mmword ptr [d]
00F117E3  movsd       mmword ptr [esp],xmm0
00F117E8  call        _sqrt (0F112A3h)
00F117ED  fstp        qword ptr [esp]
00F117F0  sub         esp,8
00F117F3  movsd       xmm0,mmword ptr [d]
00F117F8  movsd       mmword ptr [esp],xmm0
00F117FD  push        offset string "%f, %f ¥n" (0F16B30h)
00F11802  call        _printf (0F11325h)
00F11807  add         esp,14h
    return 0;
```

C言語などの
プログラミング言語

マシン語

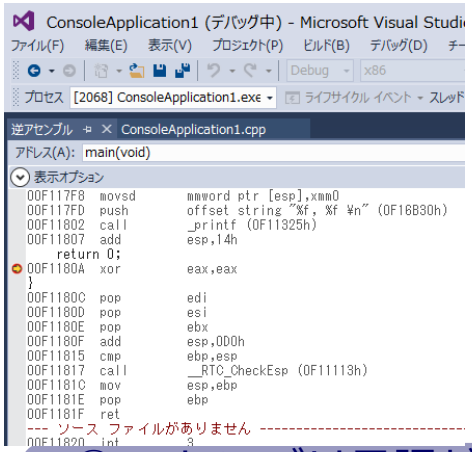
アセンブリ言語

Visual Studioでの逆アセンブル手順

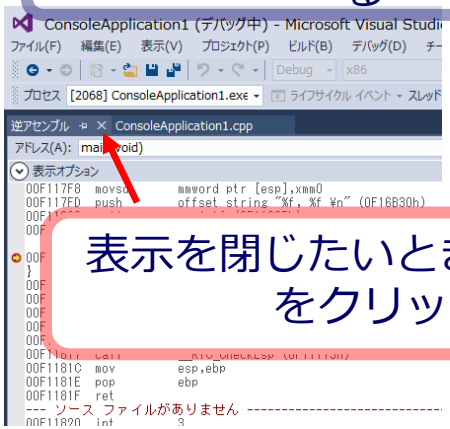


デバッガーを起動済みで，プログラムの実行が中断しているときに・・・

① 「デバッグ」
→「ウインドウ」
→「**逆アセンブル**」



② アセンブリ言語が表示される



表示を閉じたいときは，「x」をクリック

逆アセンブルの結果の例

```
00F117BC  lea           edi,[ebp-0D0h]
00F117C2  mov           ecx,34h
00F117C7  mov           eax,0CCCCCCCCh
00F117CC  rep stos      dword ptr es:[edi]
    double d = 2.0;
00F117CE  movsd         xmm0,mmword ptr [__real@4000000000000000
00F117D6  movsd         mmword ptr [d],xmm0
    printf("%f, %f ¥n", d, sqrt(d));
00F117DB  sub           esp,8
00F117DE  movsd         xmm0,mmword ptr [d]
00F117E3  movsd         mmword ptr [esp],xmm0
00F117E8  call          _sqrt (0F112A3h)
00F117ED  fstp          qword ptr [esp]
00F117F0  sub           esp,8
00F117F3  movsd         xmm0,mmword ptr [d]
00F117F8  movsd         mmword ptr [esp],xmm0
00F117FD  push          offset string "%f, %f ¥n" (0F16B30h)
00F11802  call          _printf (0F11325h)
00F11807  add           esp,14h
    return 0;
```

元のマシン語での
アドレス

命令

アセンブル

- アセンブリ言語を
- マシン語に翻訳すること
- （逆アセンブルの逆）

```
00000000  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ........@.......
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00  00 00 00 00 F0 00 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ........!..L.!Th
00000050  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
00000080  FF FA 97 C6 BB 9B F9 95  BB 9B F9 95 BB 9B F9 95  ................
00000090  FD CA 24 95 B8 9B F9 95  FD CA 26 95 BA 9B F9 95  ..$.......&.....
000000a0  FD CA 19 95 A8 9B F8 95  FD CA 18 95 BC 9B F8 95  ................
000000b0  66 64 32 95 B9 9B F8 95  BB 9B F8 95 83 9B F8 95  fd2.............
000000c0  B6 C9 18 95 B9 9B F9 95  B6 C9 22 95 BA 9B F9 95  ..........".....
000000d0  B6 C9 27 95 BA 9B F9 95  52 69 63 68 BB 9B F9 95  ..'.....Rich....
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000f0  50 45 00 00 4C 01 07 00  A8 13 37 55 00 00 00 00  PE..L.....7U....
00000100  00 00 00 00 E0 00 02 01  0B 01 0C 00 00 3C 00 00  .............<..
```

マシン語による
プログラム

```
00F117BC  lea         edi,[ebp-0D0h]
00F117C2  mov         ecx,34h
00F117C7  mov         eax,0CCCCCCCCh
00F117CC  rep stos    dword ptr es:[edi]
    double d = 2.0;
00F117CE  movsd       xmm0,mmword ptr [__real@4000000000000000]
00F117D6  movsd       mmword ptr [d],xmm0
    printf("%f, %f ¥n", d, sqrt(d));
00F117DB  sub         esp,8
00F117DE  movsd       xmm0,mmword ptr [d]
00F117E3  movsd       mmword ptr [esp],xmm0
00F117E8  call        _sqrt (0F112A3h)
00F117ED  fstp        qword ptr [esp]
00F117F0  sub         esp,8
00F117F3  movsd       xmm0,mmword ptr [d]
00F117F8  movsd       mmword ptr [esp],xmm0
00F117FD  push        offset string "%f, %f ¥n" (0F16B30h)
00F11802  call        _printf (0F11325h)
00F11807  add         esp,14h
    return 0;
```

アセンブリ言語による
プログラム

4-3 Visual Studio のデバッガー

デバッガー

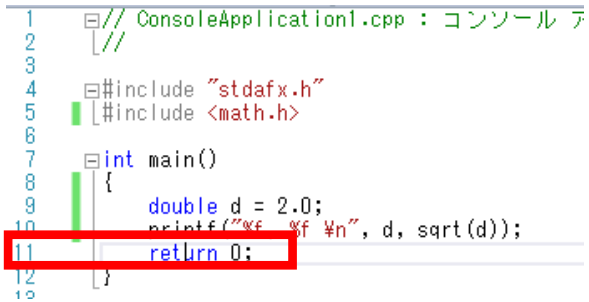
- デバッガーとは、プログラムの不具合（バグ）の発見や修正を支援するソフトウェアのこと

※ **バグ**を自動的に取り除いてくれるわけではない

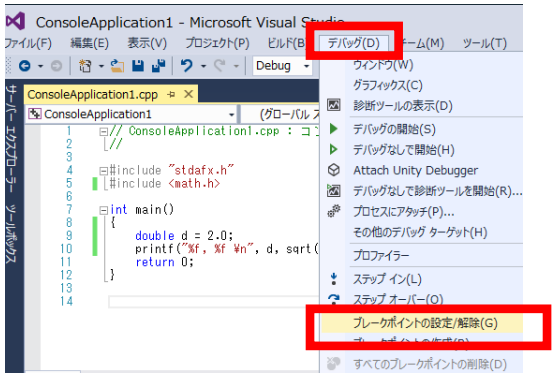
主な機能

- ◆ **ブレークポイント**機能：　プログラムの実行中断
- ◆ **トレース**機能：　プログラム実行中に**変数**の値などを表示

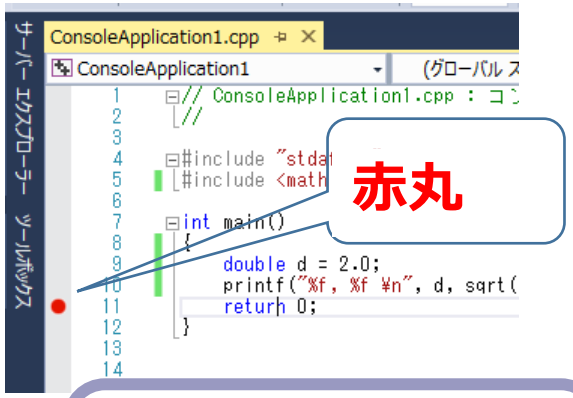
Visual Studio でのブレークポイント設定手順



① 「**return 0;**」の行をマウスでクリック



② 「デバッグ」→「**ブレークポイントの設定/解除**」

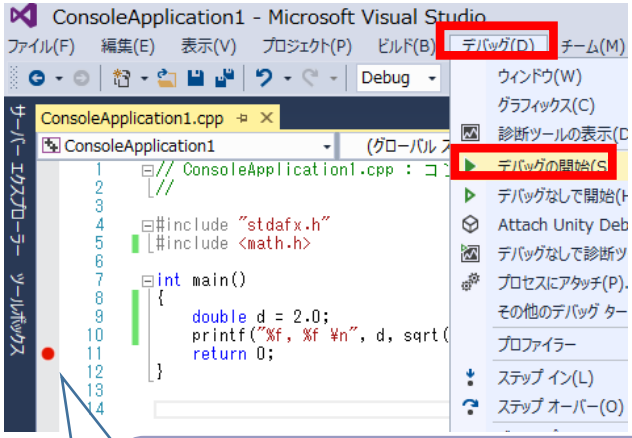


③ ブレークポイントが設定されるので**確認**

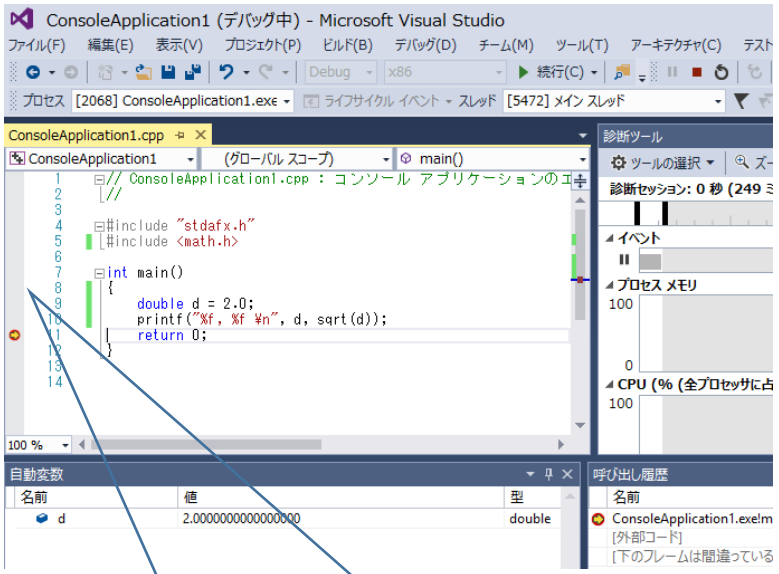
ブレークポイントとは、プログラムの実行を中断させたい行．複数設定可

ブレークポイントを解除したいときは赤丸をクリック
※ 同様の操作で、**ブレークポイント**の設定もできる

Visual Studio でのデバッガー起動手順



① 「デバッグ」
→ 「**デバッグの開始**」



ブレークポイントが設定された状態で、**デバッガー**を起動しようとしている

形が変化！　これは、プログラムの実行が、ここで中断していることを示す

デバッガーのステップオーバー機能

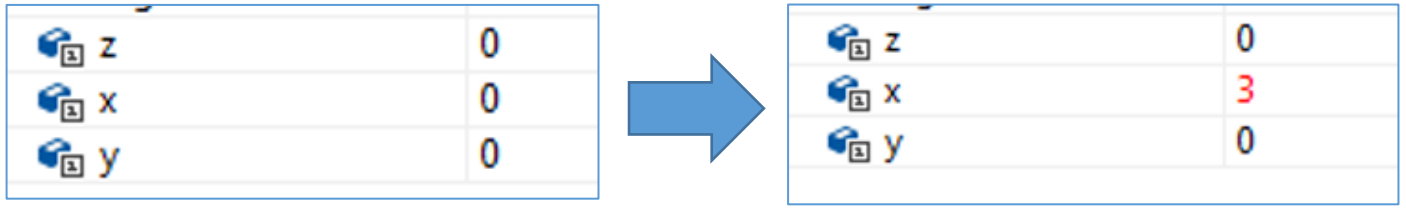
デバッガーのステップオーバー機能とは、

ブレークポイントで、プログラム実行が中断しているときに、プログラム実行を１行進めること

ステップオーバー機能の用途例

- 変数の値の変化

  ※ 変数名と値の対応表



変数 x の値が 0 から 3 に変化

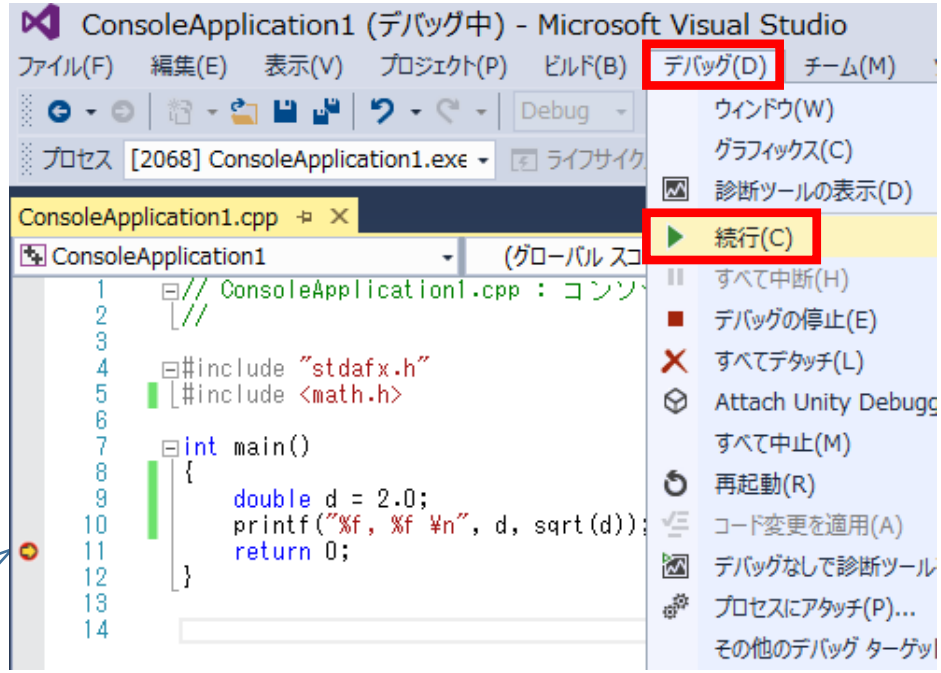
- メモリ中のデータの変化

  ※ ダンプリスト形式

- プロセッサのレジスタ内のデータの変化

  ※ レジスタ名と値の対応

ブレークポイントで中断されたプログラム実行を再開する手順



実行がここで
中断している

「デバッグ」→
「**続行**」

変数の値の変化

```
int main()
{
    static int x, y, z;
    x = 3;
    y = 4;
    z = x + y;
    return 0;
}
```

この行で，x の値は 3 に変化

この行で，y の値は 4 に変化

この行で，z の値は 7 に変化

※「static int x, y, z;」の「static」は、変数 x, y, z のメモリへの格納法を
コントロールするキーワード

演習

- Visual Studio を起動しなさい
- Visual Studio で、Win32 コンソールアプリケーション用プロジェクトを新規作成しなさい

プロジェクトの「名前」は何でもよい

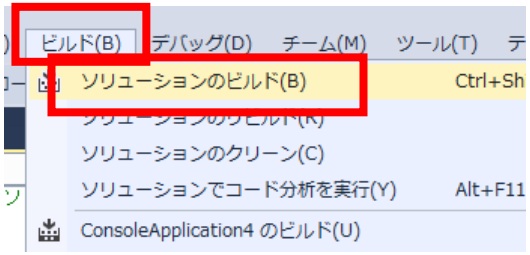
- Visual Studioのエディタを使って，ソースファイルを編集しなさい

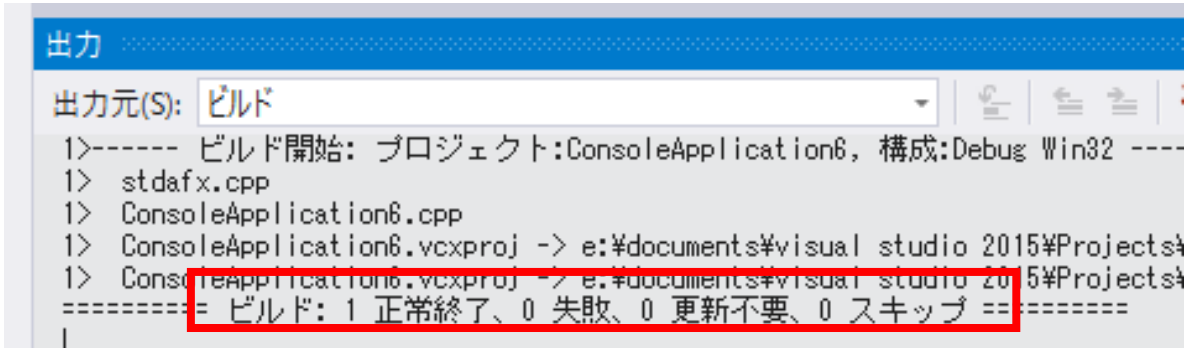
```
int main()
{
    static int x, y, z;
    x = 3;
    y = 4;
    z = x + y;
    return 0;
}
```

４行追加

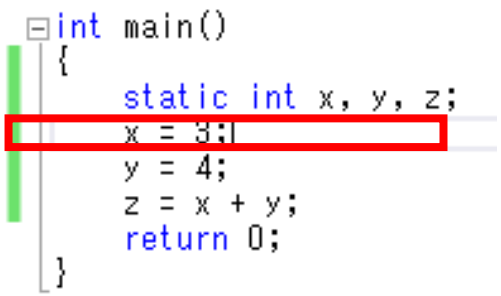
- ビルドしなさい．ビルドのあと「１　正常終了，０　失敗」の表示を確認しなさい

→　表示されなければ，プログラムのミスを自分で確認し，修正して，ビルドをやり直す



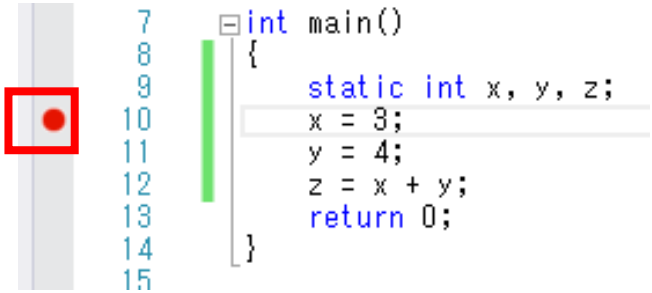


- Visual Studioで「x=3;」の行に，ブレークポイントを設定しなさい



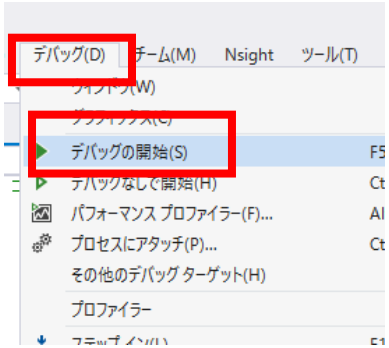
① 「**x=3;**」の行をマウスでクリック

② 「デバッグ」→「**ブレークポイントの設定/解除**」



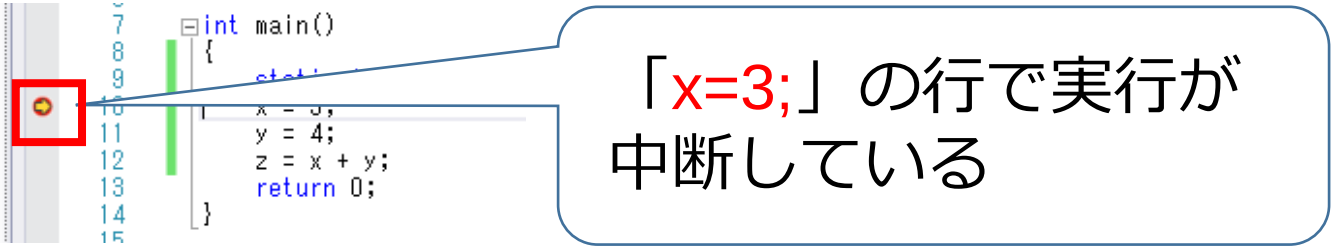
③ ブレークポイントが設定されるので確認．
赤丸がブレークポイントの印

- Visual Studioで，デバッガーを起動しなさい.

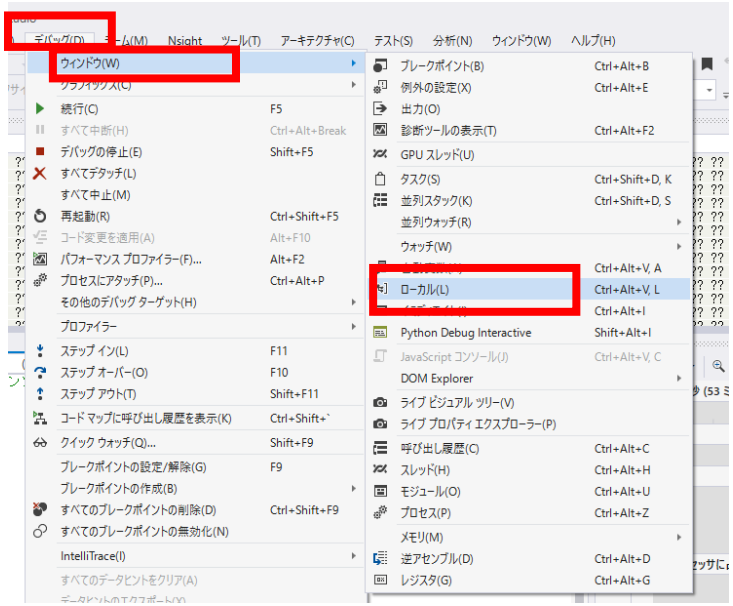


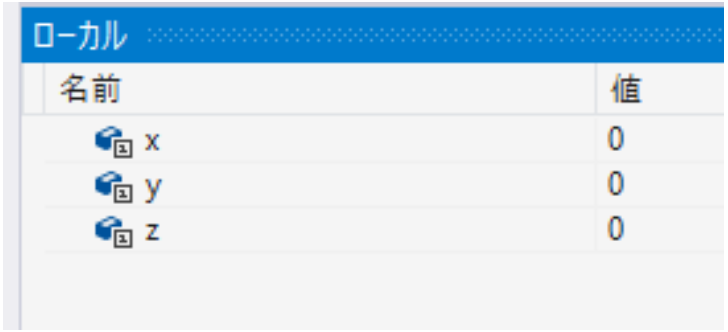
「デバッグ」
→ 「**デバッグ開始**」

- 「x=3;」の行で，実行が中断することを確認しなさい
- あとで使うので，中断したままにしておくこと



- 「x=3;」の行で，実行が中断した状態で，変数の値を表示させなさい．手順は次の通り．

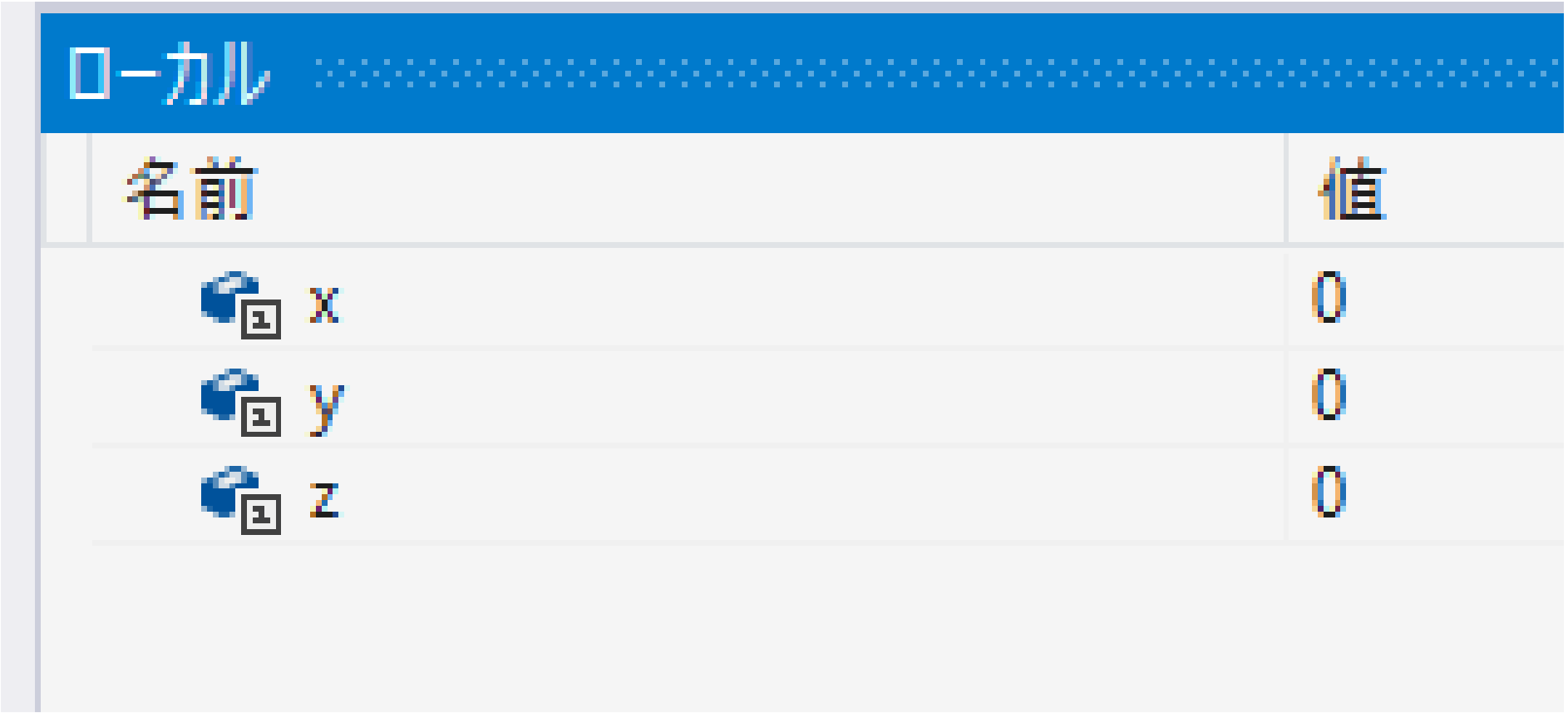




① 「デバッグ」
→ 「ウインドウ」
→ 「**ローカル**」

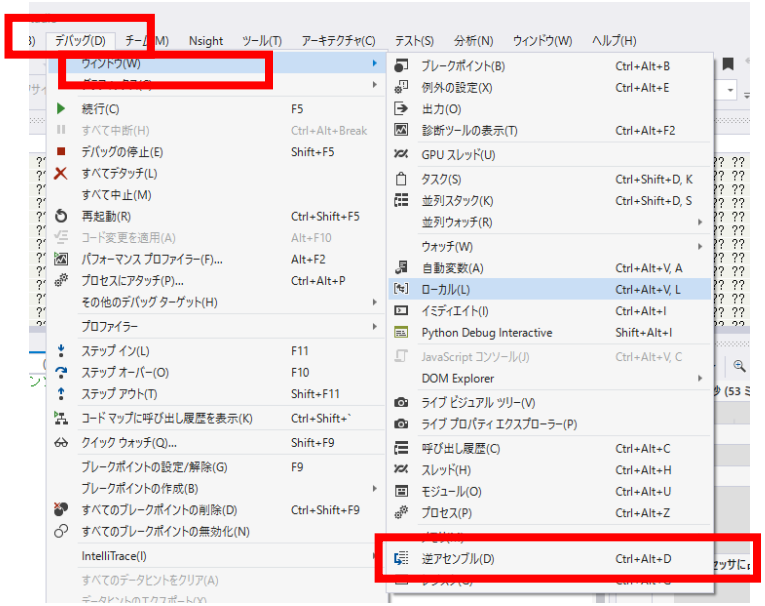
② 変数名と値の対応表が表示される

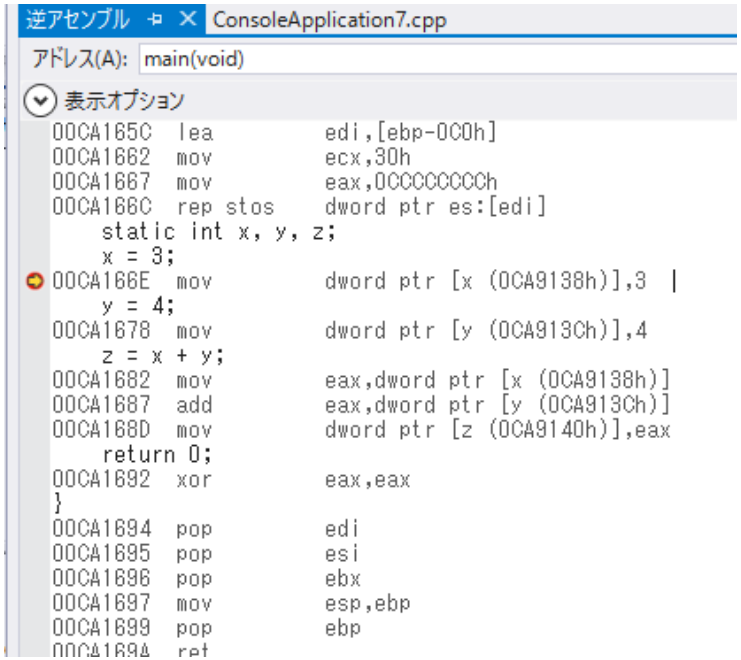
※ 次ページに拡大図



「x=3;」は
未実行であることを確認！

- 「x=3;」の行で，実行が中断した状態で，逆アセンブルを行いなさい.

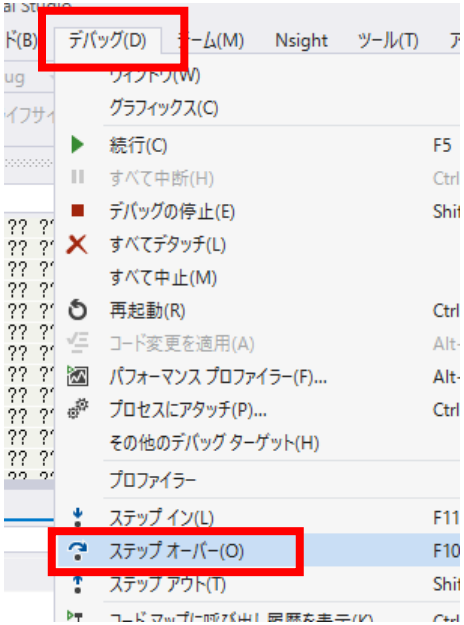


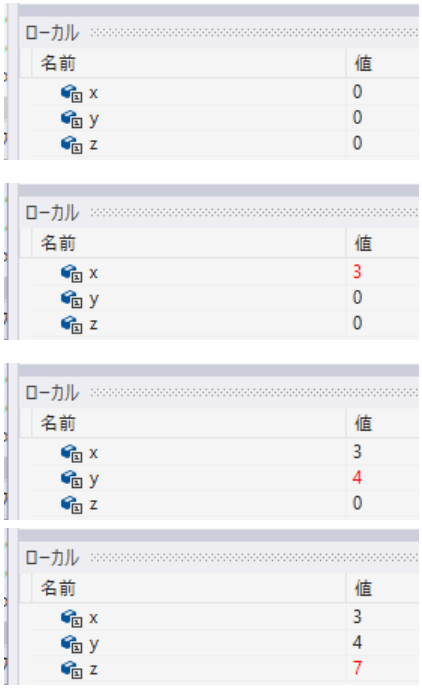


① 「デバッグ」→「ウインドウ」→「**逆アセンブル**」

② 逆アセンブルの結果が表示される

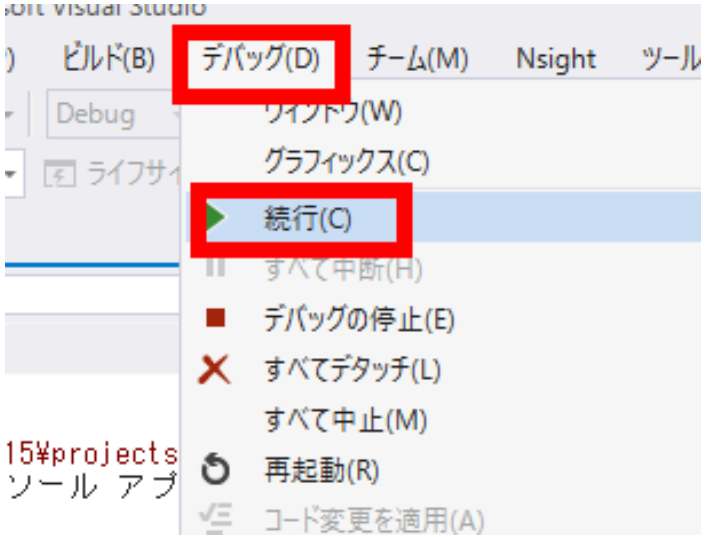
- ステップオーバーの操作を１回ずつ行いながら，変数 x, y, z の値の変化を確認しなさい．





「デバッグ」
→「**ステップオーバー**」
（あるいは F10 キー）

- 最後に，プログラム実行の再開の操作を行いなさい。これで，デバッガーが終了する。



「デバッグ」
→ 「**続行**」

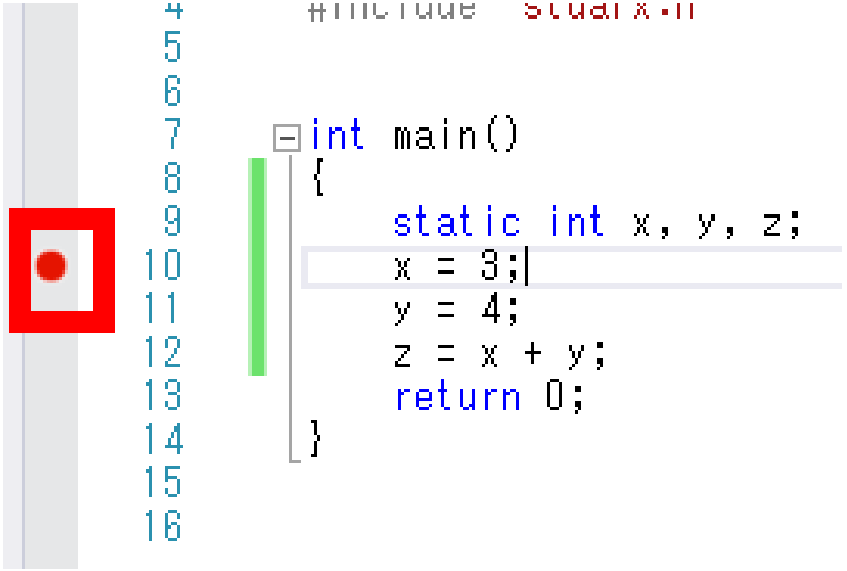
4-4 変数のアドレスの確認

- ソースファイルをそのまま使う.

```
1   // ConsoleApplication2.cpp :
2   //
3
4   #include "stdafx.h"
5
6
7   int main()
8   {
9       static int x, y, z;
10      x = 3;
11      y = 4;
12      z = x + y;
13      return 0;
14  }
15
```

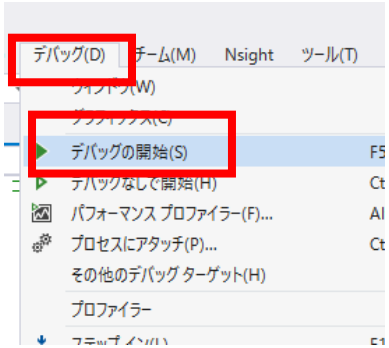
←そのまま使う

- ブレークポイントの設定もそのまま使う.
- 「x = 3;」の行にブレークポイントが設定されていること再確認しておく



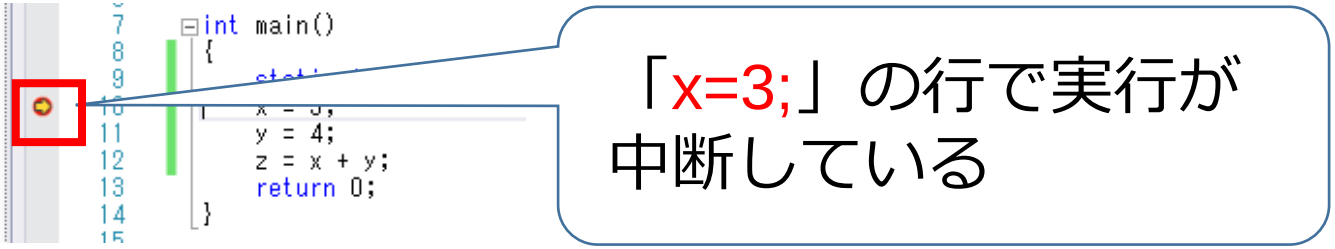
赤丸がブレークポイントの印

- Visual Studioで，デバッガーを起動しなさい.



「デバッグ」
→ 「**デバッグ開始**」

- 「x=3;」の行で，実行が中断することを確認しなさい
- あとで使うので，中断したままにしておくこと



- 「x = 3;」の行で、実行が中断した状態で、逆アセンブルを行ってみなさい.

① 「デバッグ」→「ウインドウ」→「**逆アセンブル**」

```
00281660  rep stos    dword ptr es:[edi]
    static int x, y, z;
    x = 3;
0028166E  mov         dword ptr [x (0289138h)],3
    y = 4;
00281678  mov         dword ptr [y (028913Ch)],4
    z = x + y;
00281682  mov         eax,dword ptr [x (0289138h)]
00281687  add         eax,dword ptr [y (028913Ch)]
0028168D  mov         dword ptr [z (0289140h)],eax
    return 0;
```

② **逆アセンブルの結果**が表示される

逆アセンブルの結果の見方

元の Visual C++ プログラム

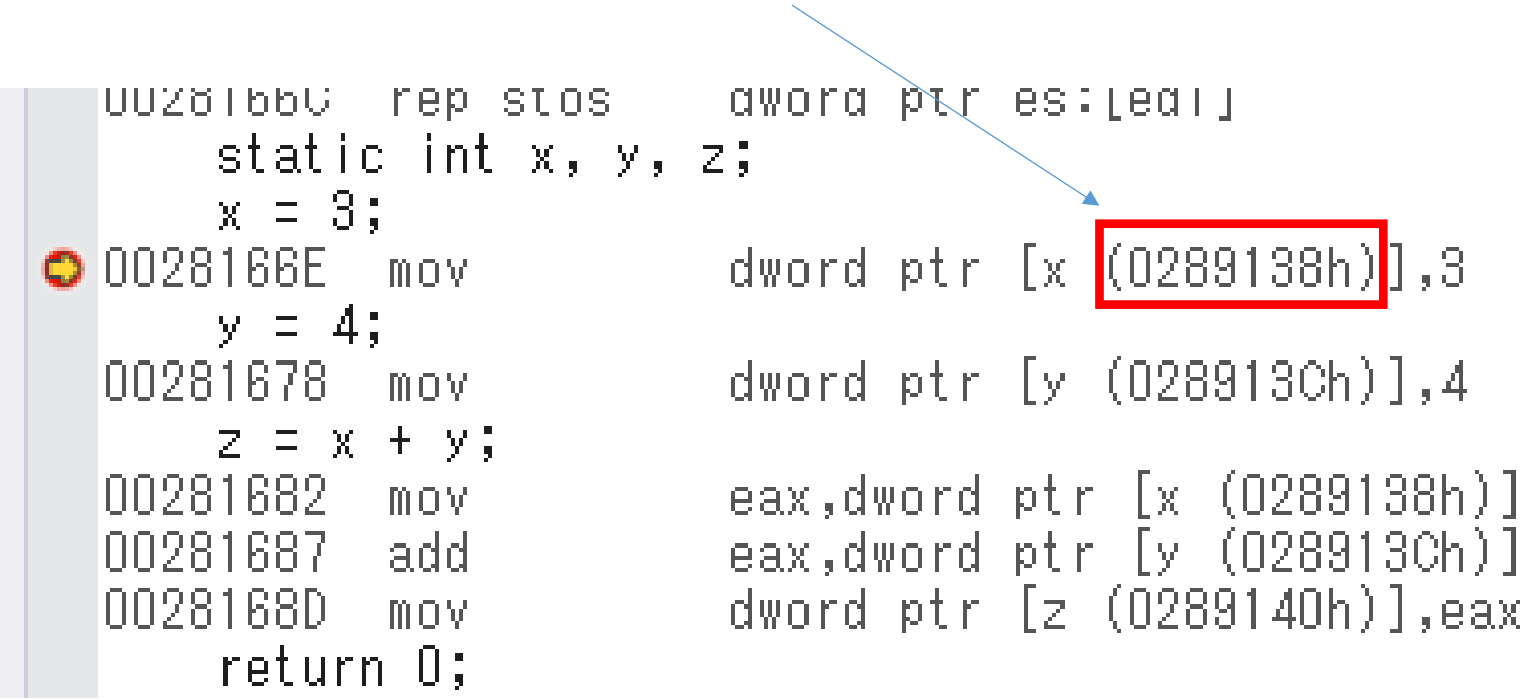
```
00281BBC  rep stos    dword ptr es:[edi]
    static int x, y, z;
    x = 3;
0028166E  mov         dword ptr [x (0289138h)],3
    y = 4;
00281678  mov         dword ptr [y (028913Ch)],4
    z = x + y;
00281682  mov         eax,dword ptr [x (0289138h)]
00281687  add         eax,dword ptr [y (028913Ch)]
0028168D  mov         dword ptr [z (0289140h)],eax
    return 0;
```

プログラムで読み書きする
メモリのアドレス

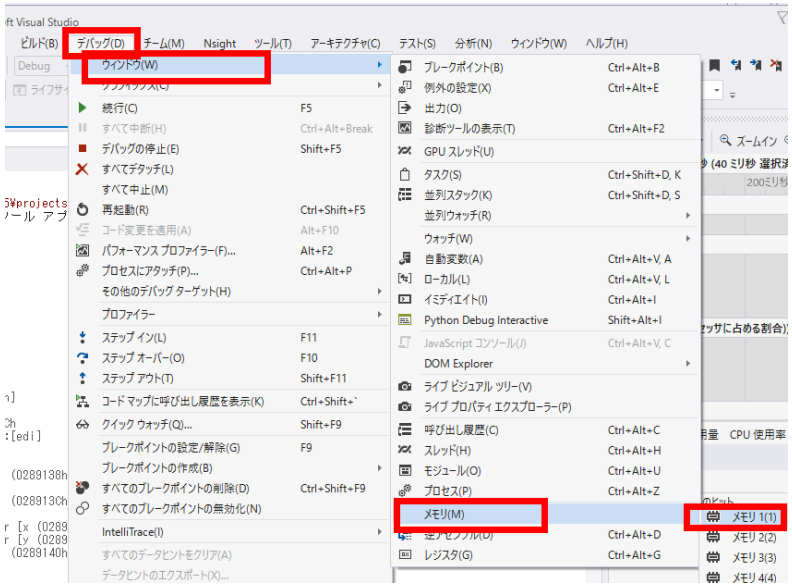
Visual C++ に対応した
アセンブリ言語プログラム

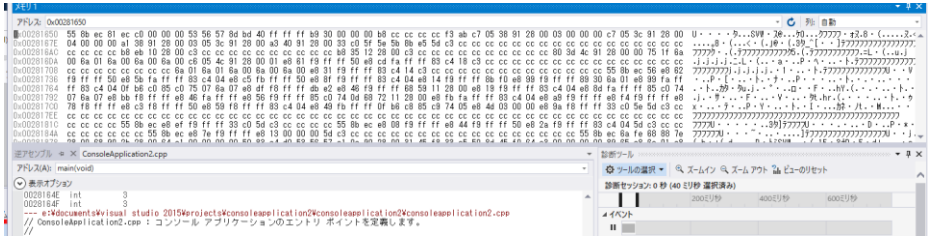
- 「x = 3;」の直下に、x のアドレスが表示されているので確認しなさい．後で使う。

例） アドレスは **0289138h**



- 「x = 3;」の行で、実行が中断した状態で、メモリの中身を表示させなさい．手順は次の通り．

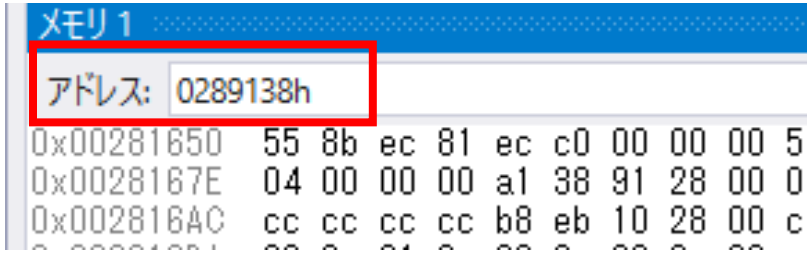


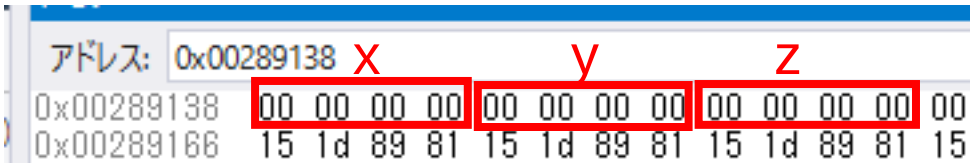


② メモリの中身がダンプリスト形式で表示される

①「デバッグ」
→「ウインドウ」
→ 「**メモリ**」→「**メモリ１**」

- 「**アドレス**」のところに「0289138h」のように記入。「0289138h」は、手順 6 で調べた値を転記. 末尾に「h」





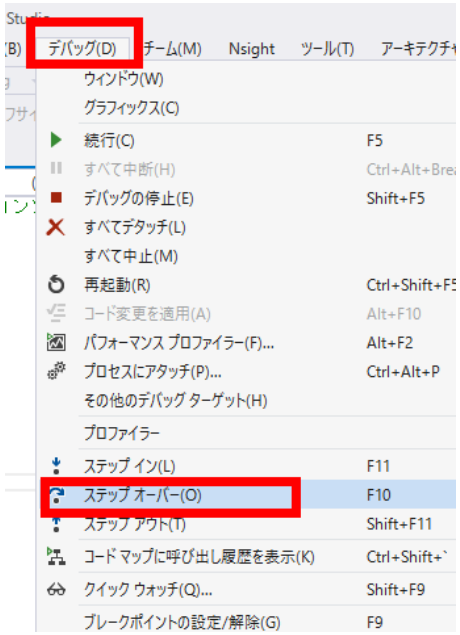
③　アドレスのところに、x のアドレスを書き、末尾に「h」を付けて Enter キーを押す

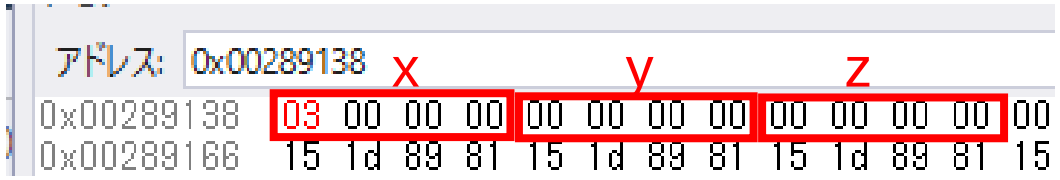
変数 x, y, z の値はすべて 0

④ メモリの中身が表示される

「x = 3;」は未実行で　あることを確認！

- １回だけステップオーバーの操作を行い、変数 x の値が 3 に変化することを確認しなさい.



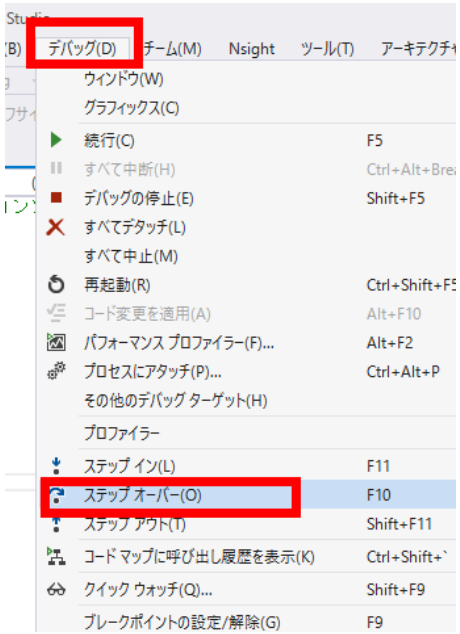


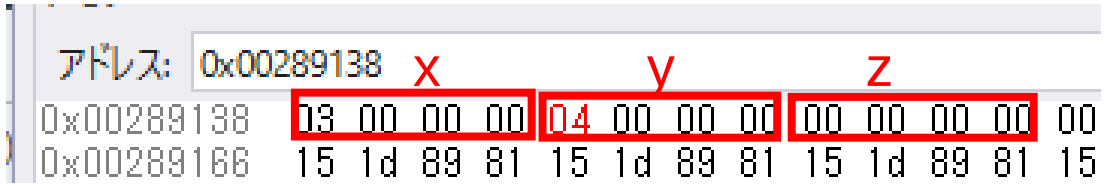
変数 x の値が 3 に変化！

「デバッグ」
→「**ステップオーバー**」
（あるいは **F10 キー**）

「**x = 3;**」が実行された

- もう１回だけステップオーバーの操作を行い、変数 y の値が 4 に変化することを確認しなさい.



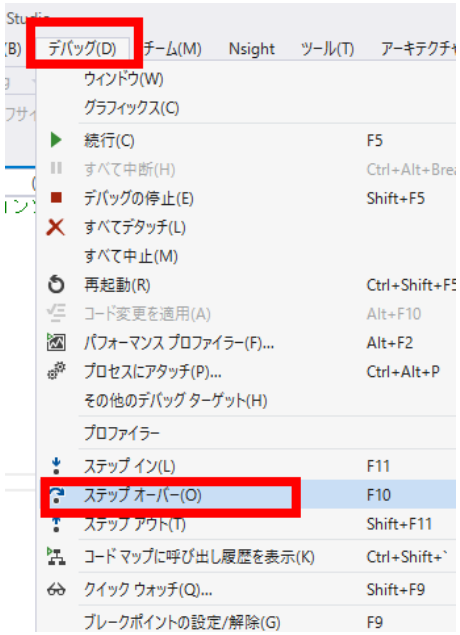


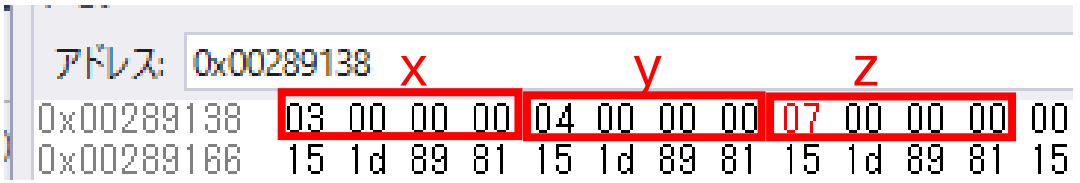
変数 y の値が 4 に変化！

「デバッグ」
→「**ステップオーバー**」
（あるいは **F10 キー**）

「**y = 4;**」が実行された

- さらに３回、ステップオーバーの操作を行い、変数 z の値が 7 に変化することを確認しなさい.



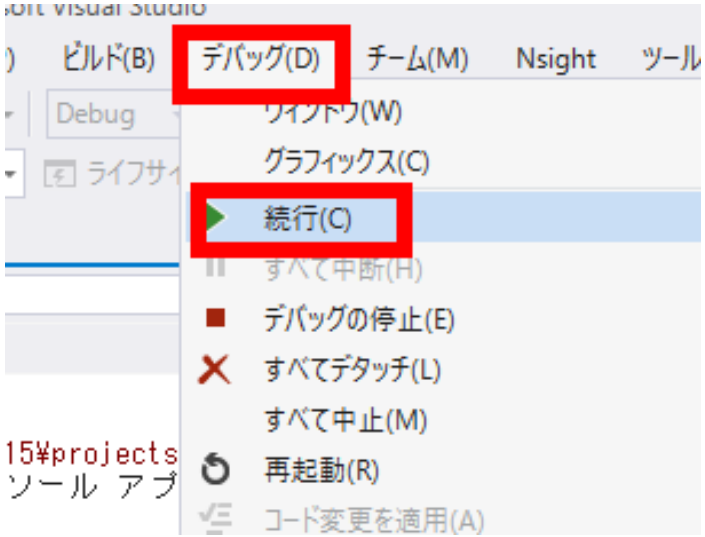


変数 z の値が 7 に変化！

「デバッグ」
→「**ステップオーバー**」
（あるいは **F10 キー**）

「**z = x + y;**」が実行された

- 最後に、プログラム実行の再開の操作を行いなさい。これで、デバッガが終了する。



「デバッグ」
→「**続行**」

アセンブリ言語の中のメモリアドレス

- プログラム

```
    x = 3;
00CA166E  mov         dword ptr [x (0CA9138h)],3
    y = 4;
00CA1678  mov         dword ptr [y (0CA913Ch)],4
    z = x + y;
00CA1682  mov         eax,dword ptr [x (0CA9138h)]
00CA1687  add         eax,dword ptr [y (0CA913Ch)]
00CA168D  mov         dword ptr [z (0CA9140h)],eax
    return 0;
```

変数 x のメモリアドレス

変数 y のメモリアドレス

変数 x のメモリアドレス

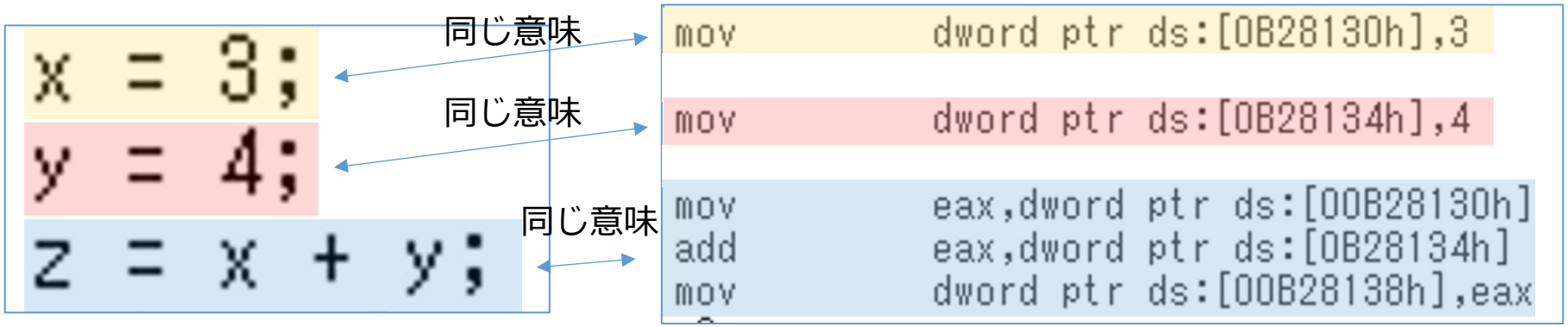
変数 y のメモリアドレス

変数 z のメモリアドレス

C++ 言語とアセンブリ言語

Visual C++ の
プログラム

アセンブリ言語



命令

命令が対象とする相手である
オペランド

C++ 言語とアセンブリ言語

Visual C++ の
プログラム

```
x = 3;
y = 4;
z = x + y;
```

アセンブリ言語

```
mov     dword ptr [x (0CA9138h)],3
mov     dword ptr [y (0CA913Ch)],4
mov     eax,dword ptr [x (0CA9138h)]
add     eax,dword ptr [y (0CA913Ch)]
mov     dword ptr [z (0CA9140h)],eax
```

x のアドレス

y のアドレス

x のアドレス

y のアドレス

z のアドレス

C++ 言語とアセンブリ言語

Visual C++ の
プログラム

```
x = 3;
y = 4;
z = x + y;
```

アセンブリ言語

変数 x に 3 をセット

```
mov     dword ptr [x (0CA9138h)],3
```

変数 y に 4 をセット

```
mov     dword ptr [y (0CA913Ch)],4
```

変数 x を読み出す

```
mov     eax,dword ptr [x (0CA9138h)]
add     eax,dword ptr [y (0CA913Ch)]
mov     dword ptr [z (0CA9140h)],eax
```

変数 y を読み出して
足し算

結果を
変数 z に書き込む