# ca-5. レジスタ

（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

CC BY NC SA

# アウトライン


5-1 プロセッサの仕組み

5-2 レジスタ

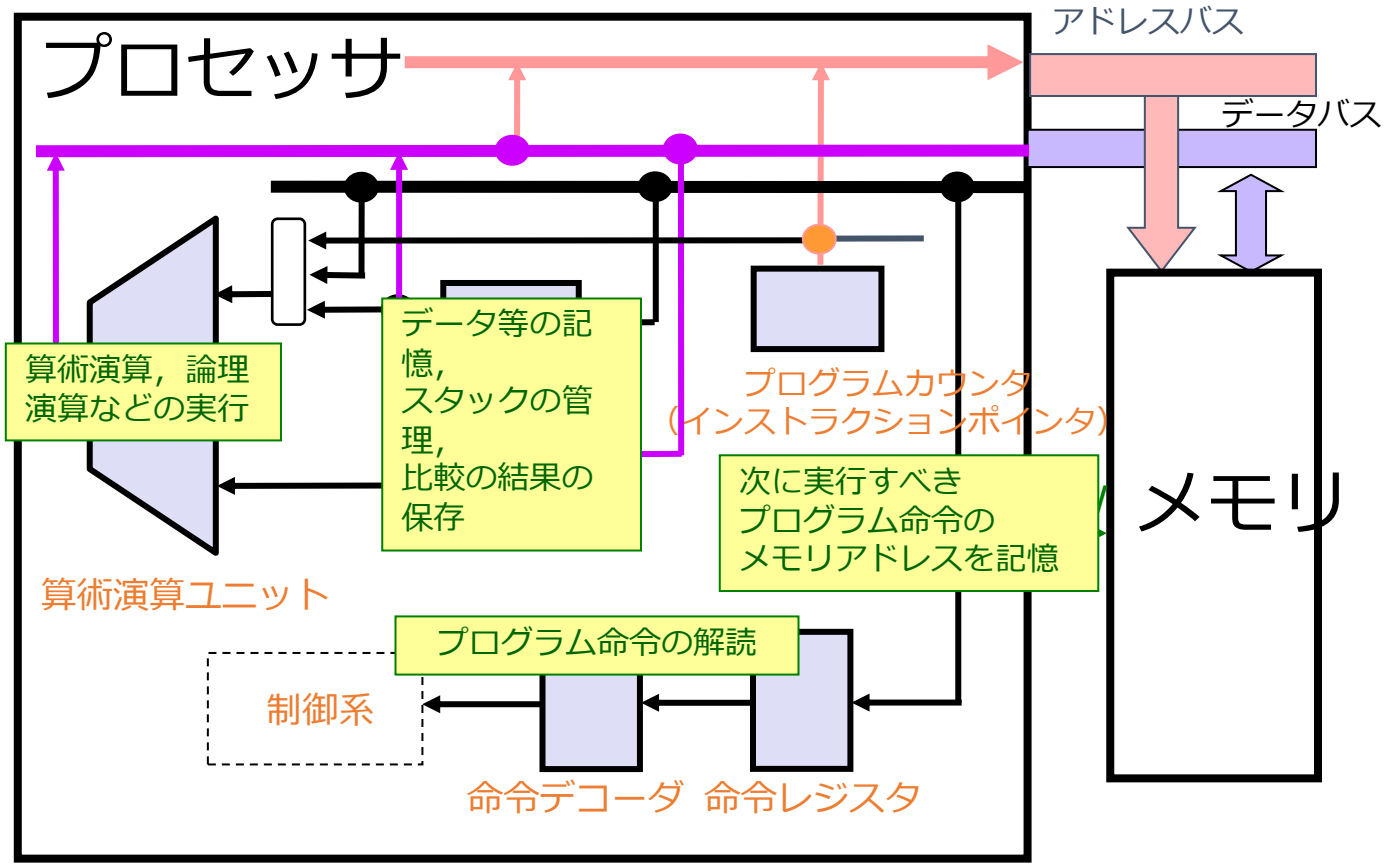
5-3 Pentium系列プロセッサのレジスタ

5-4 レジスタを使うプログラムの例

5-5 Visual Studio でレジスタ表示


# 5-1 プロセッサの仕組み

# プロセッサ（CPU）の仕組み



# プロセッサ


- メモリにあるプログラムやデータが読み出されて，プロセッサで処理される
- プロセッサがメモリに書き込みを行うこともある


# 5-2 レジスタ

# 5-2 レジスタとは

- レジスタは，プロセッサの内部にあるデータやプログラムの格納場所

- レジスタには名前（レジスタ名）がある

# データ格納場所の種類

| レジスタ | キャッシュメモリ | 一般のメモリ | ハードディスク |
|---|---|---|---|
| CPUの**内部** | CPUの**内部** | CPUの**外** | CPUの**外** |
| **レジスタ名** | **アドレス** | **アドレス** | セクタ番号，シリンダ番号 |
| 超高速 | 高速 | 低速 | 超低速 |
| 極小サイズ | 小サイズ | 大サイズ | 超大サイズ |


# 5-3 Pentium系列プロセッサのレジスタ

# Pentium系列プロセッサのレジスタ

- 代表的なものは

| | |
|---|---|
| • EAX, EBX, ECX, EDX,<br>• ESI, EDI, | 汎用レジスタ |
| • EBP, | ベースポインタ |
| • ESP, | スタックポインタ |
| • EFLAGS, | フラグレジスタ |
| • CS, DS, ES, SS, FS, GS, | セグメントレジスタ |
| • EIP | プログラムカウンタ |

名前　　種類

**レジスタ**の名前がいろいろあるので，
種類分けする

# Pentium系列プロセッサのレジスタの大きさ

**Pentium 系列プロセッサ**では,
**レジスタ**のサイズは**32ビット**または**16ビット**

代表的なものは

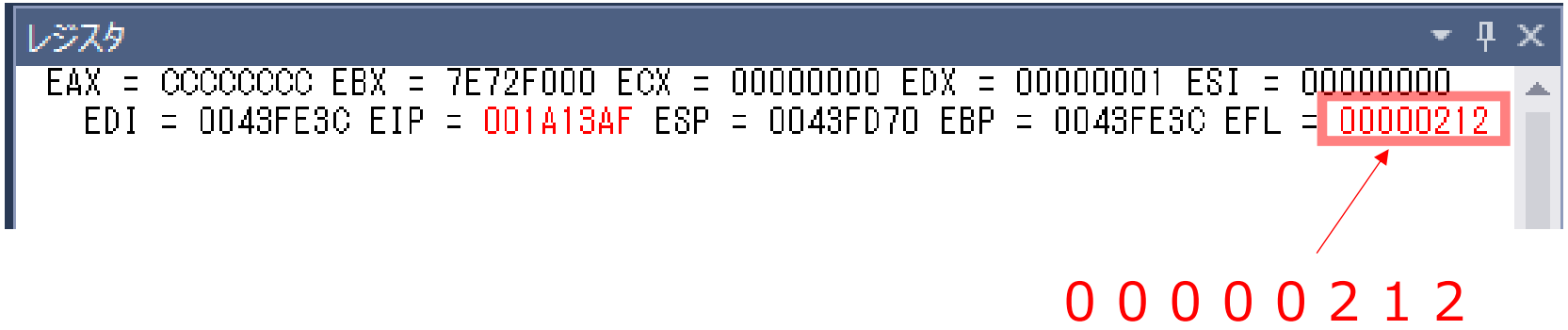
- EAX, EBX, ECX, EDX,
- ESI, EDI,
- EBP,
- ESP,

それぞれ３２ビット長
（例） 04001234H　※ 16進8桁

- EFLAGS,
- CS, DS, ES, SS, FS, GS

それぞれ１６ビット長
（例） 3000H ※ 16進4桁

# フラグレジスタ eflags (縮めて EFL)




0 0 0 0 0 2 1 2

| フラグの名前 | ID | VIP | VIF | AC | VM | RF | | NT | IOPL | | OF | DF | IF | TF | SF | ZF | | AF | | PF | | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 値 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

32 ビット長

# 5-4 レジスタを使うプログラムの例

# レジスタを使う例

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int a, b;
    a = 10;
    b = a + 20;
    printf("%d, %d¥n", a, b);
    return 0;
}
```

Visual Studio の C++ プログラム

```
    int a, b;
    a = 10;
012A13DE  mov         dword ptr [a],0Ah
    b = a + 20;
012A13E5  mov         eax,dword ptr [a]
012A13E8  add         eax,14h
012A13EB  mov         dword ptr [b],eax
```

アセンブリ言語のプログラム

**b = a + 20**

```
mov     eax,dword ptr [a]
add     eax,14h
mov     dword ptr [b],eax
```

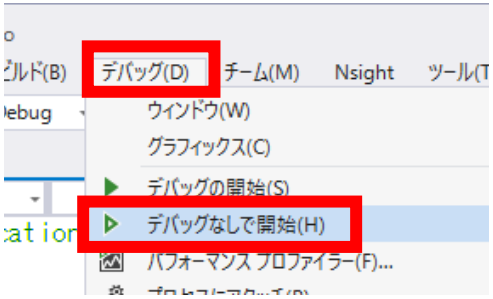
# レジスタの値の変化

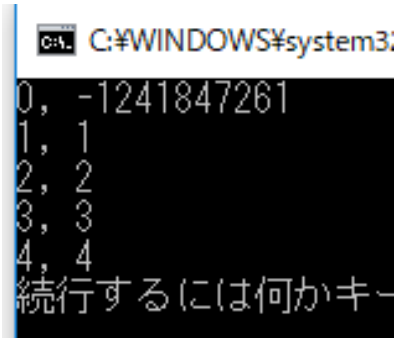
次のプログラムで，レジスタの値が**変化**する

```
int i;
for (i = 0; i < 5; i++) {
  printf("%d¥n", i);
}
```

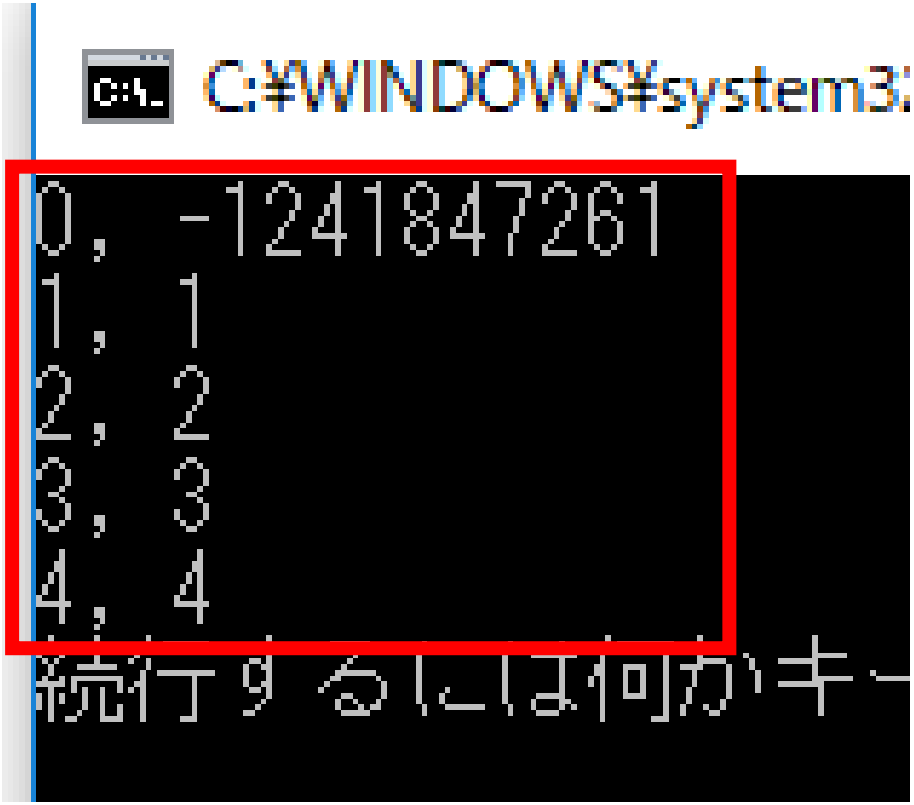
# レジスタの値の変化を見る

```
int main()
{
    int i, x;
    for (i = 0; i < 5; i++) {
        _asm {
            mov x, eax;
        };
        printf("%d, %d¥n", i, x);
    }
    return 0;
}
```




レジスタ eax の値の変化が右側に表示される




結果の例.

レジスタ eax は，**最初は初期化されていない**（以前の値が残っている）

```
    for (i = 0; i < 5; i++) {
002E1D08  mov         dword ptr [i],0
002E1D0F  jmp         main+3Ah (02E1D1Ah)
002E1D11  mov         eax,dword ptr [i]
002E1D14  add         eax,1
002E1D17  mov         dword ptr [i],eax
002E1D1A  cmp         dword ptr [i],5
002E1D1E  jge         main+5Ah (02E1D3Ah)
        _asm {
            mov x, eax;
002E1D20  mov         dword ptr [x],eax
        };
        printf("%d, %d¥n", i, x);
002E1D23  mov         eax,dword ptr [x]
002E1D26  push        eax
```

**逆アセンブル**してみると，
**「i < 5」かどうかを調べる**のに
レジスタ eax を使っている．

**最初の表示の時点**では，**まだ**「i < 5」
かどうかを**調べていない**ので，
**レジスタ eax** は**初期化されていない**

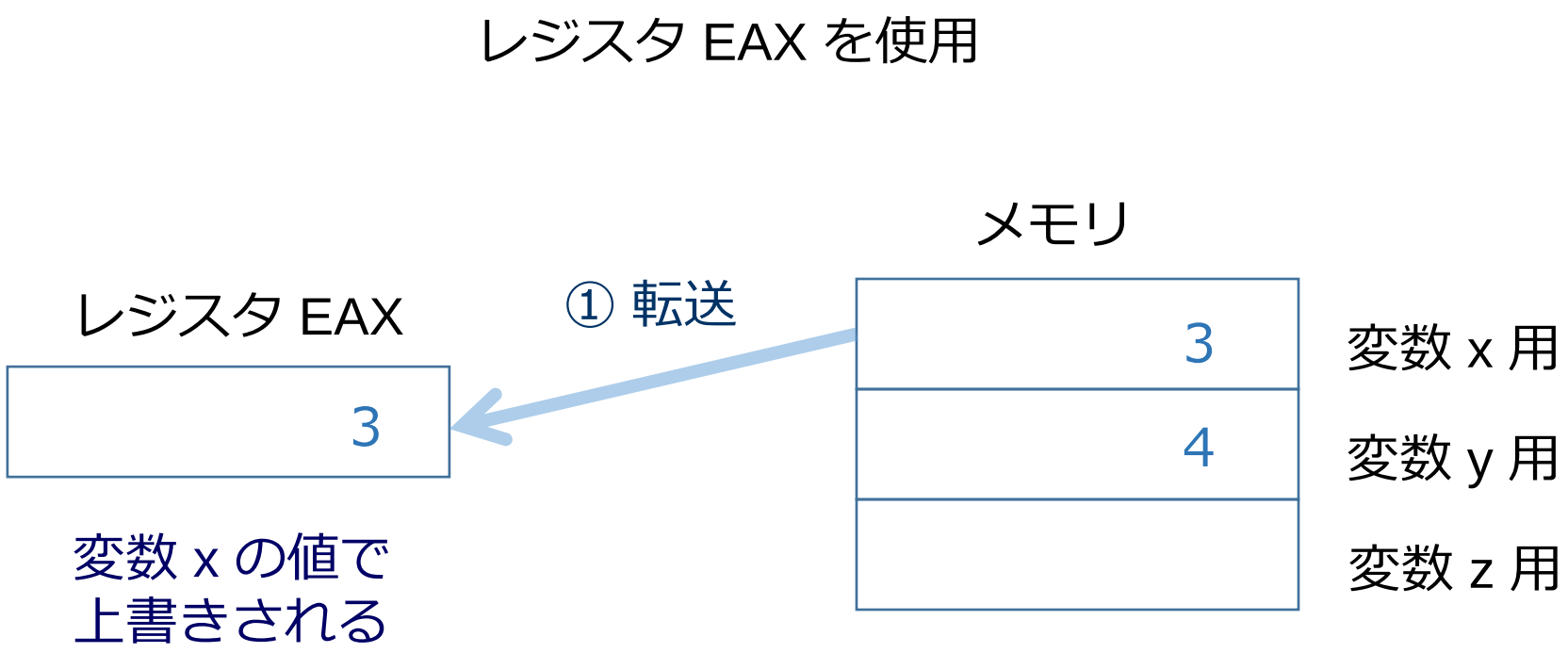
# 5-5 Visual Studio でレジスタ表示

# レジスタを使っているのは？

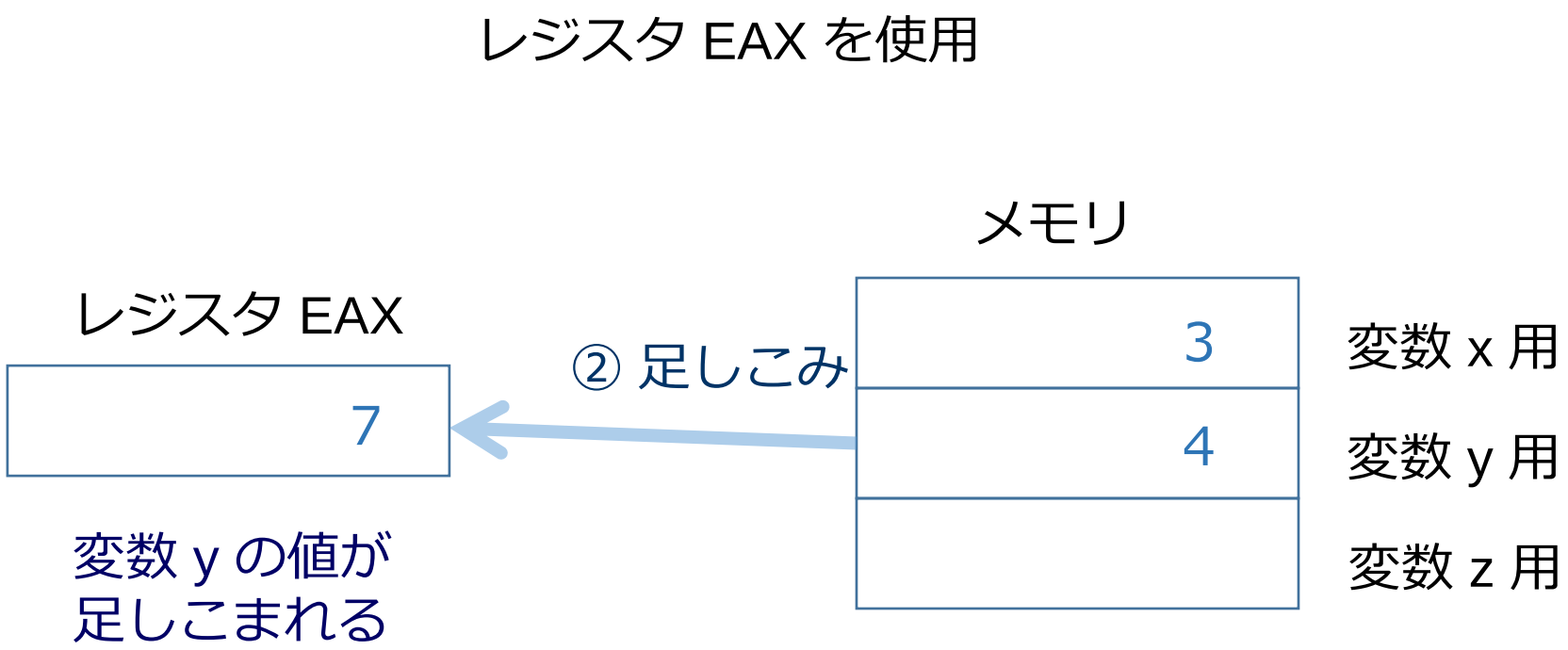
```
x = 3;
y = 4;
z = x + y;
```

ここで，レジスタを使用！

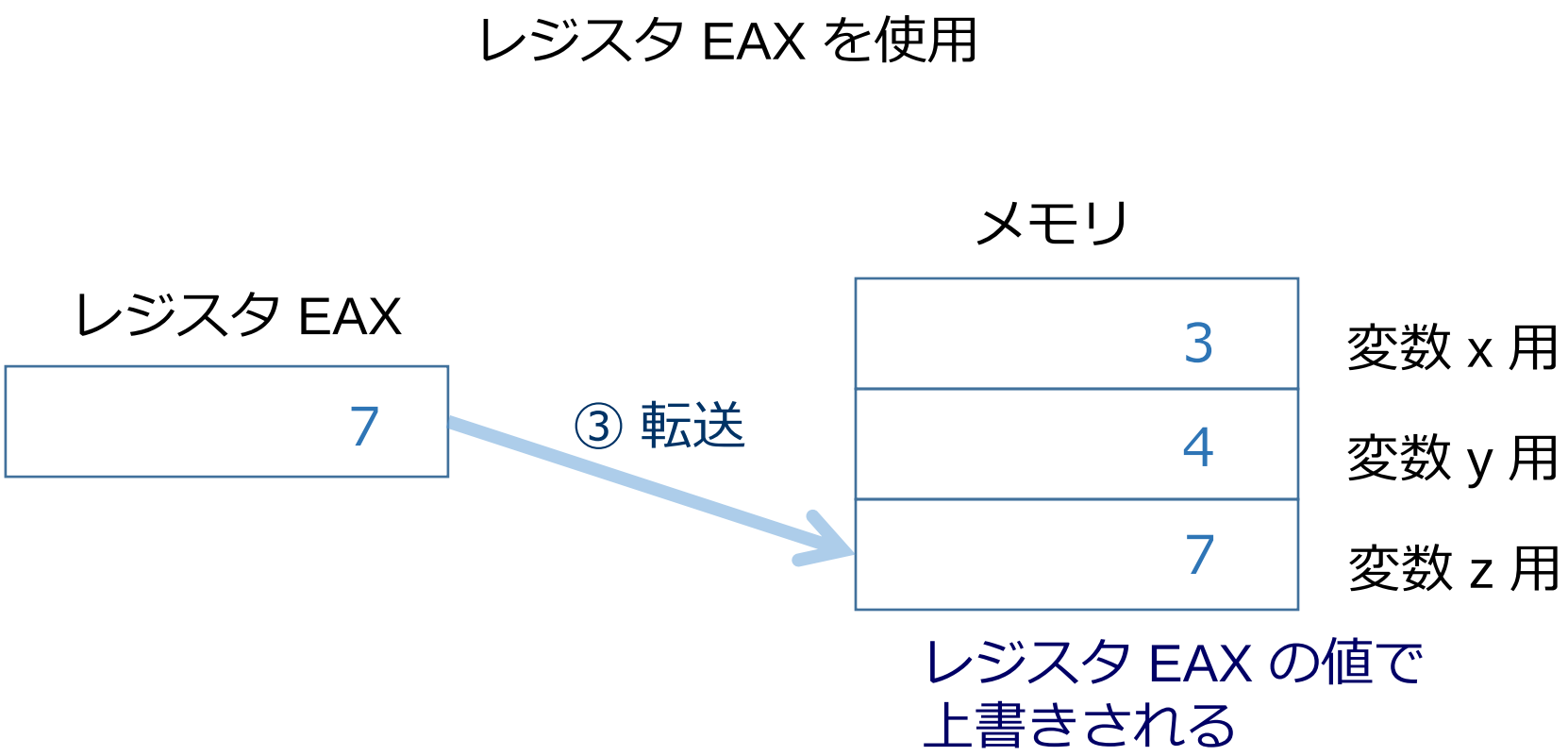
# 「z = x + y;」部分で行われていること



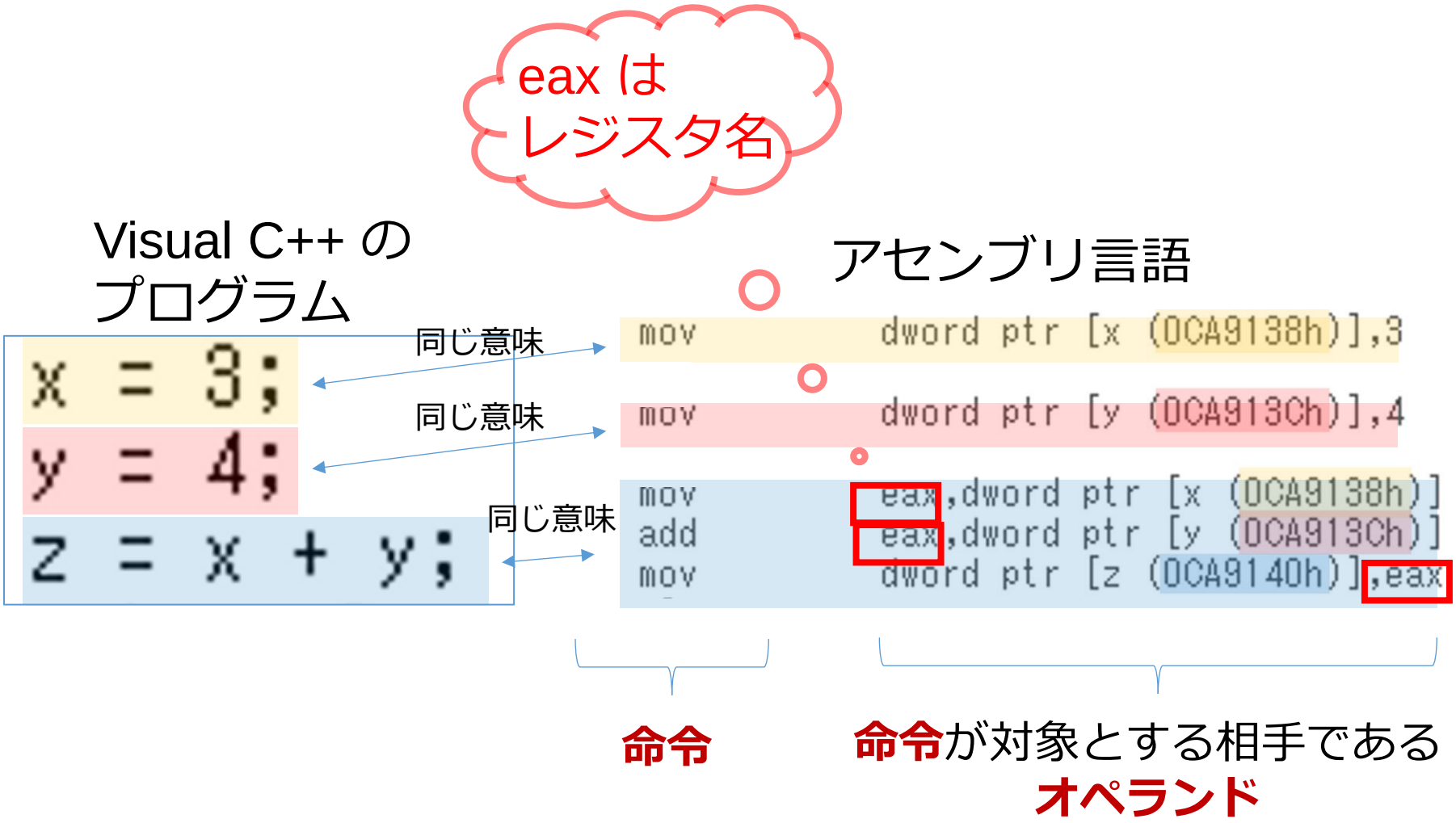
# 「z = x + y;」部分で行われていること



# 「z = x + y;」部分で行われていること



# レジスタを使っているのは？



# 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

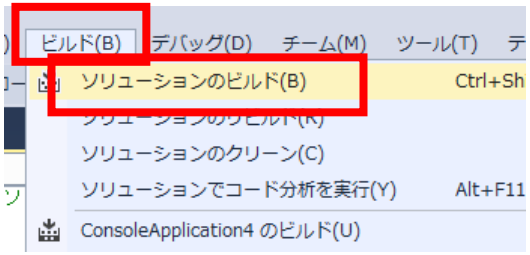
- Visual Studioのエディタを使って，ソースファイルを編集しなさい

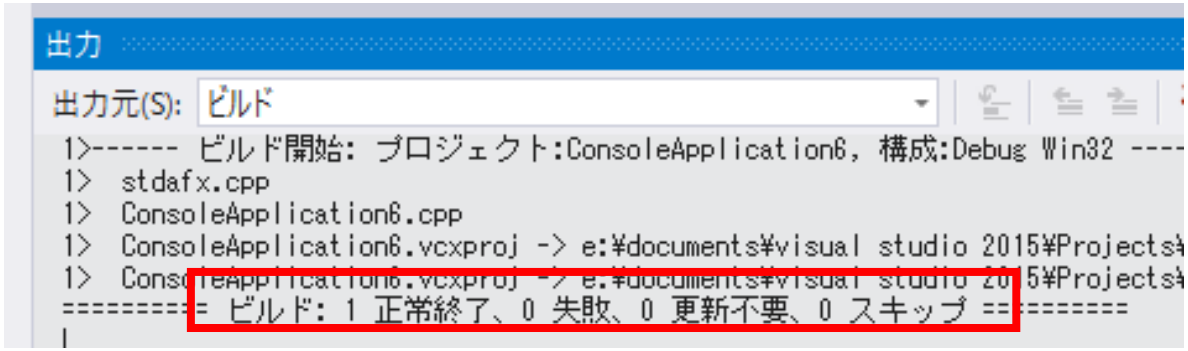
```
int main()
{
    static int x, y, z;
    x = 3;
    y = 4;
    z = x + y;
    return 0;
}
```

４行追加

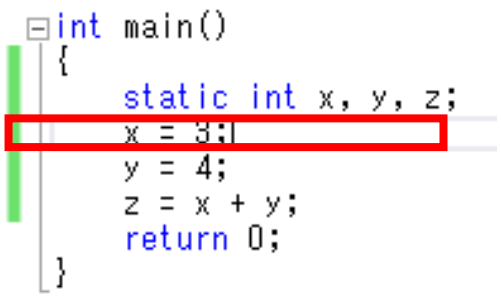
- ビルドしなさい．ビルドのあと「１　正常終了，０　失敗」の表示を確認しなさい

→　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



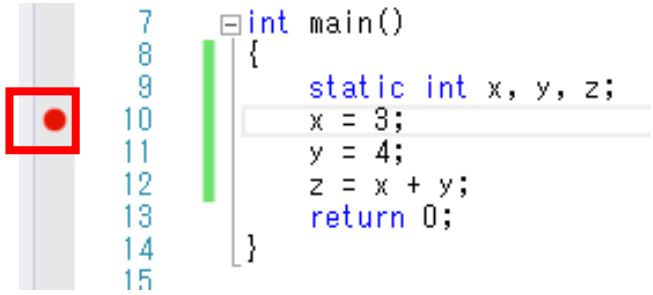



- Visual Studioで「x=3;」の行に，ブレークポイントを設定しなさい



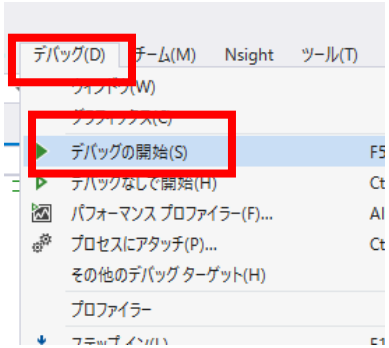

① 「**x=3;**」の行をマウスでクリック


② 「デバッグ」→「**ブレークポイントの設定/解除**」



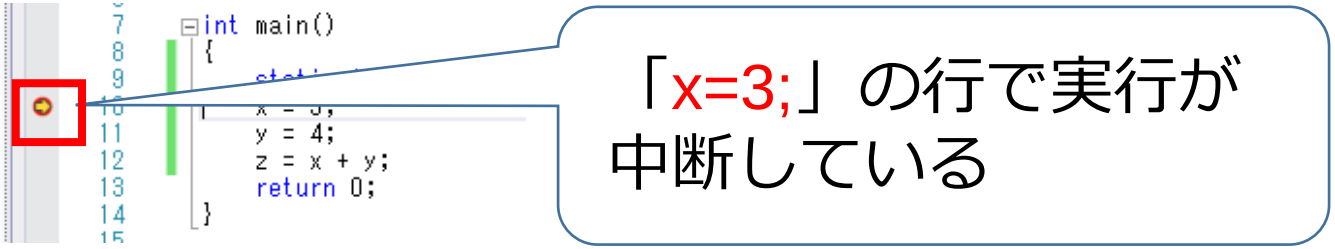

③ ブレークポイントが設定されるので確認．
赤丸がブレークポイントの印

- Visual Studioで，デバッガーを起動しなさい.

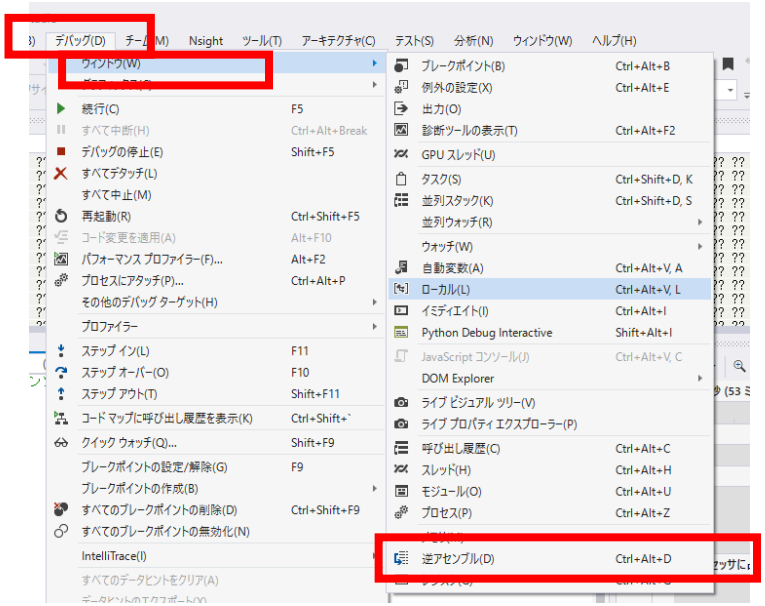


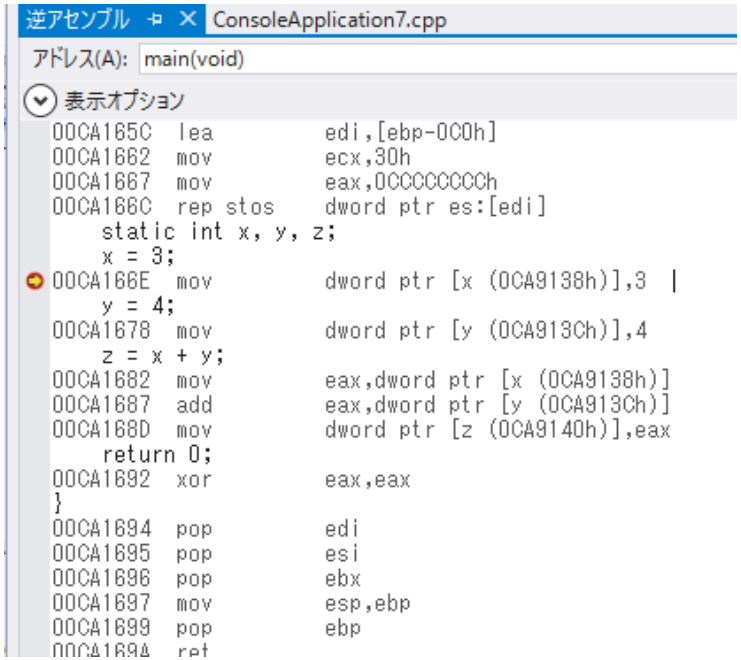

「デバッグ」
→ 「**デバッグ開始**」

- 「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，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

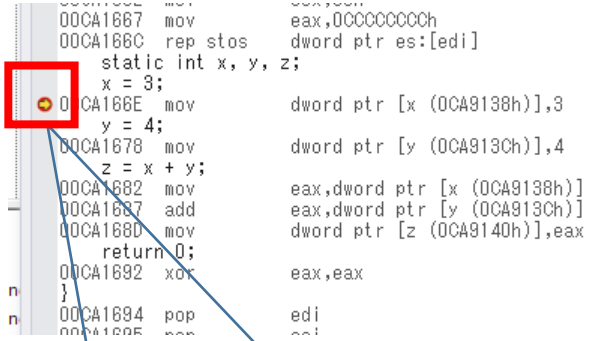
- 逆アセンブルの結果で，**レジスタ名 eax** を確認しなさい

```
00CA1660  rep stos    qword ptr es:[edi]
    static int x, y, z;
    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;
00CA1692  xor         eax,eax
```

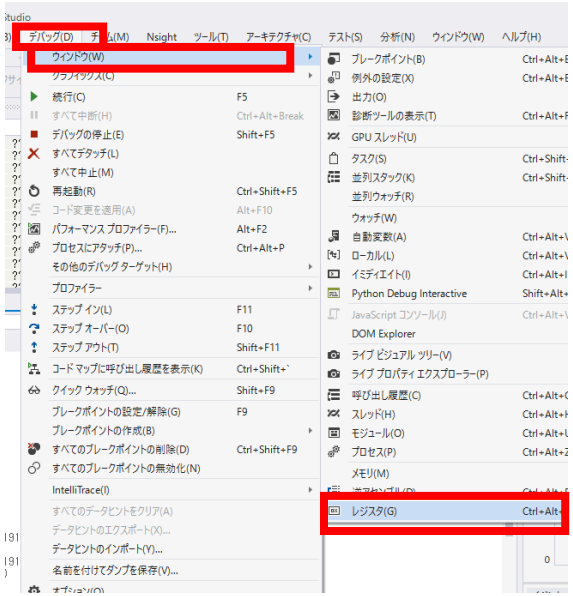
レジスタ名 eax の確認！

元の C++ プログラム

- 「x = 3;」の行で，実行が中断した状態で，レジスタの中身を表示させなさい．手順は次の通り．



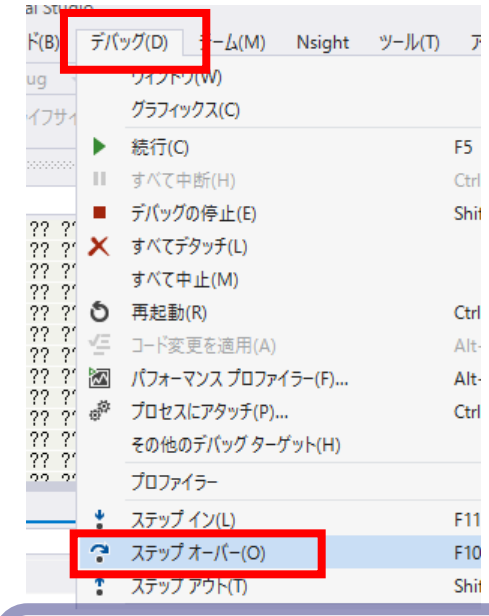

デバッガーを**起動済み**で，プログラムの実行が中断しているときに・・・



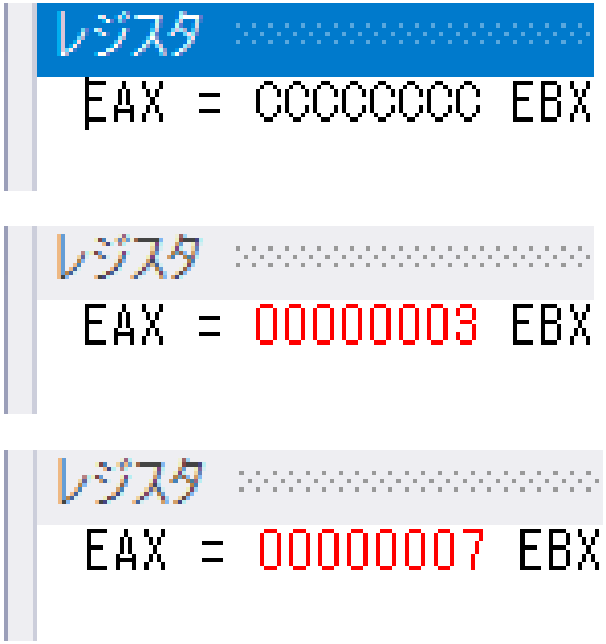

② レジスタが表示される．

① 「デバッグ」
→「ウインドウ」→「**レジスタ**」

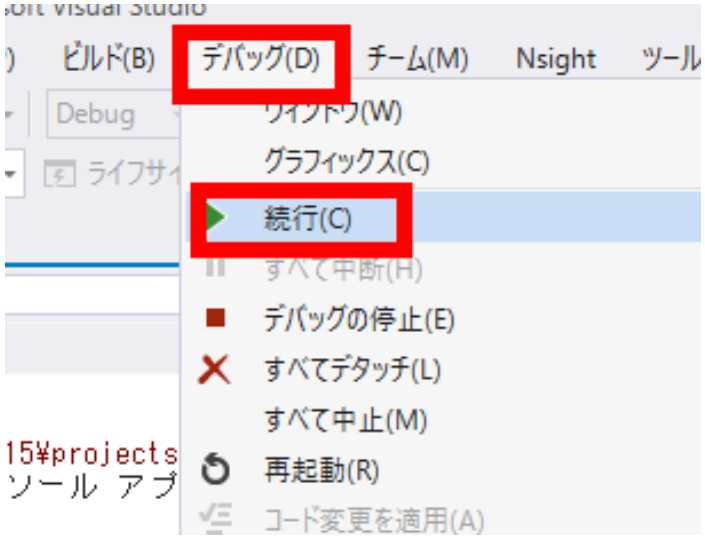
- ステップオーバーの操作を１回ずつ行いながら，レジスタ eax の値の変化を確認しなさい．




「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**」