

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



アウトライン



5-1 プロセッサの仕組み

5-2 レジスタ

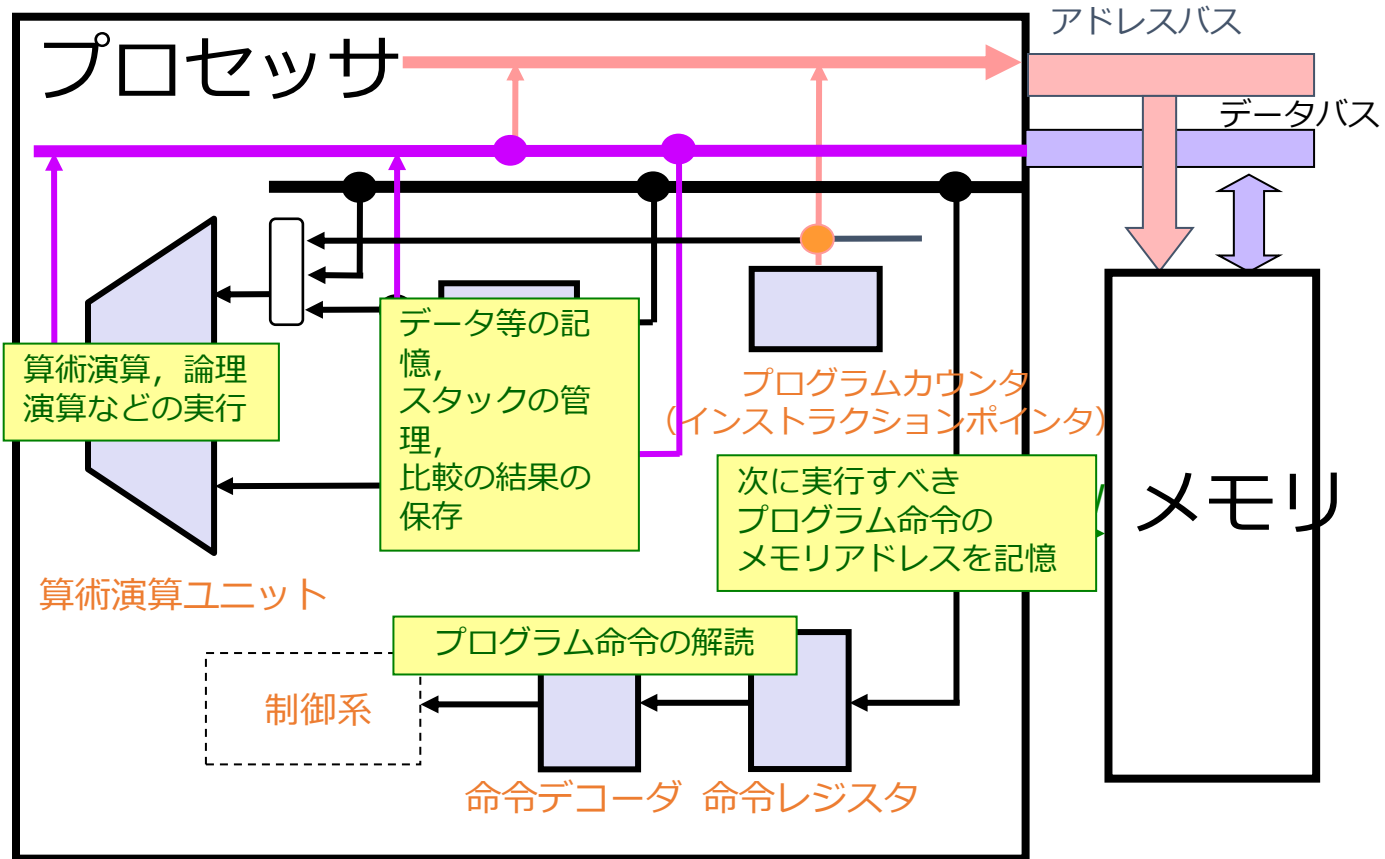
5-3 Pentium系列プロセッサのレジスタ

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 <u>内部</u>	CPUの <u>内部</u>	CPUの <u>外</u>	CPUの <u>外</u>
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,

汎用レジスタ

• ESI, EDI,

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

それぞれ 32ビット長
(例) 04001234H ※ 16進8桁

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)



```
レジスタ
EAX = CCCCCCCC EBX = 7E72F000 ECX = 00000000 EDX = 00000001 ESI = 00000000
EDI = 0043FE3C EIP = 001A19AF ESP = 0043FD70 EBP = 0043FE3C EFL = 00000212
```

0 0 0 0 0 2 1 2

フラグの名前	I	V	V	A	V	R	N	IOP	O	D	I	T	S	Z	A	P	C							
	D	P	F	C	M	F	T	L	F	F	F	F	F	F	F	F	F							
値	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;
}
```

```
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

Visual Studio の C++ 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



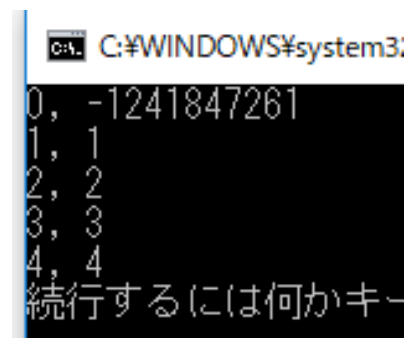
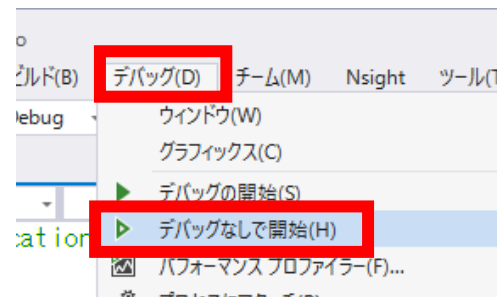
次のプログラムで、レジスタの値が変化する

```
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;
}
```



```
C:\WINDOWS\system32
0, -1241847261
1, 1
2, 2
3, 3
4, 4
続行するには何かキー
```

レジスタ `eax` の値の変化が右側に表示される


```
C:\ C:\WINDOWS\system32  
0, -1241847261  
1, 1  
2, 2  
3, 3  
4, 4  
続行するには何かキー
```

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]  
002F1D26 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;
```

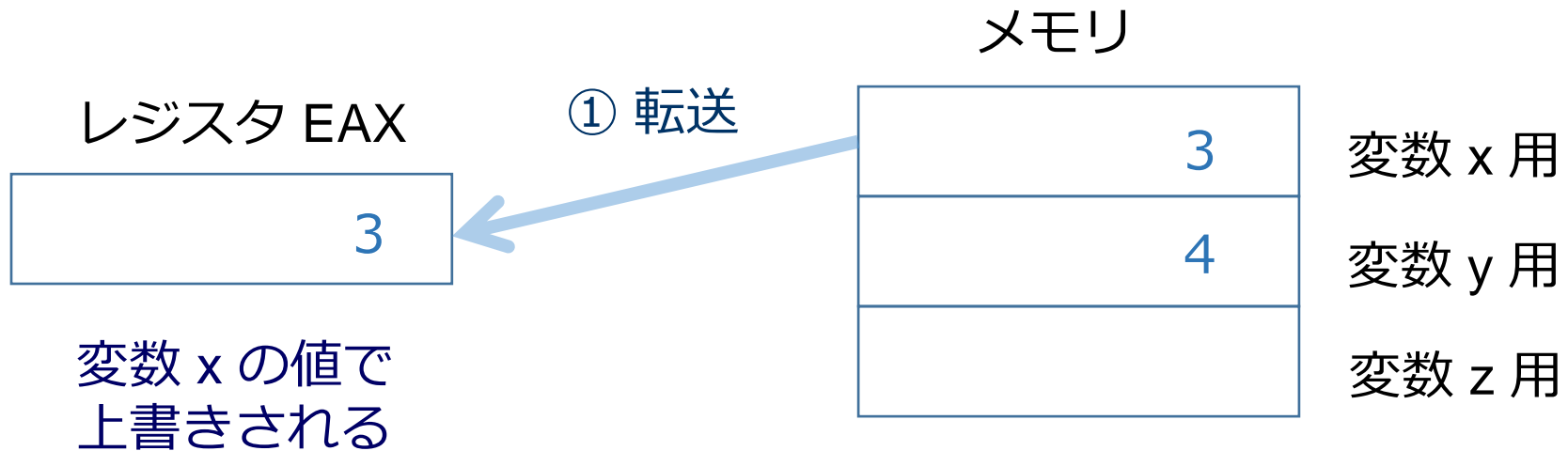
```
z = x + y;
```

ここで、レジスタを使用！

「 $z = x + y;$ 」部分で行われていること



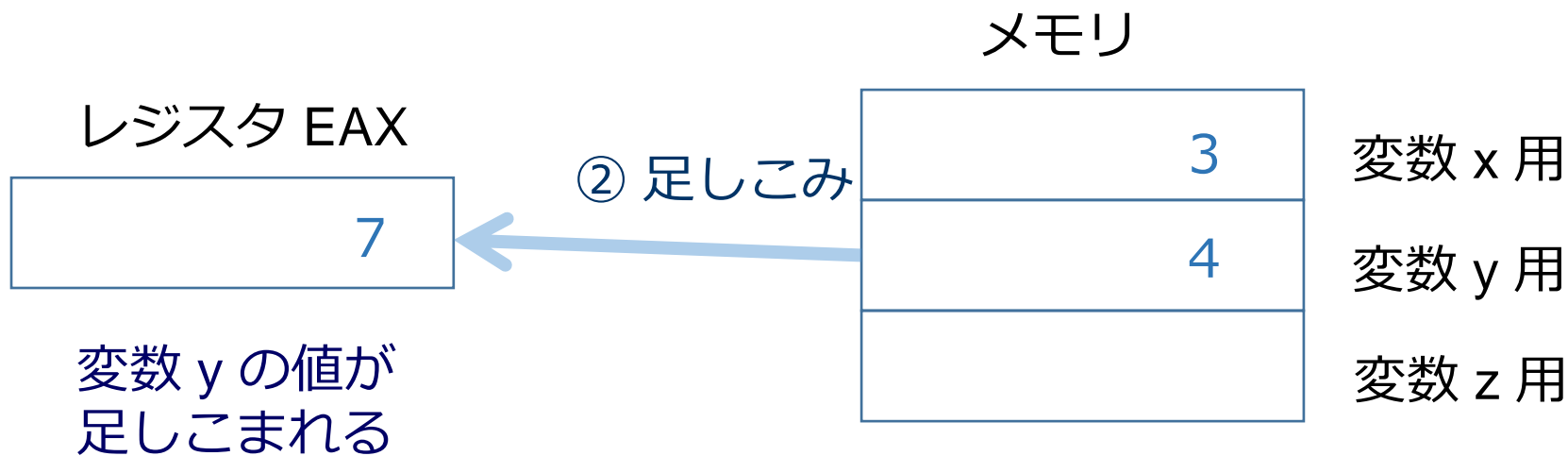
レジスタ EAX を使用



「z = x + y;」部分で行われていること



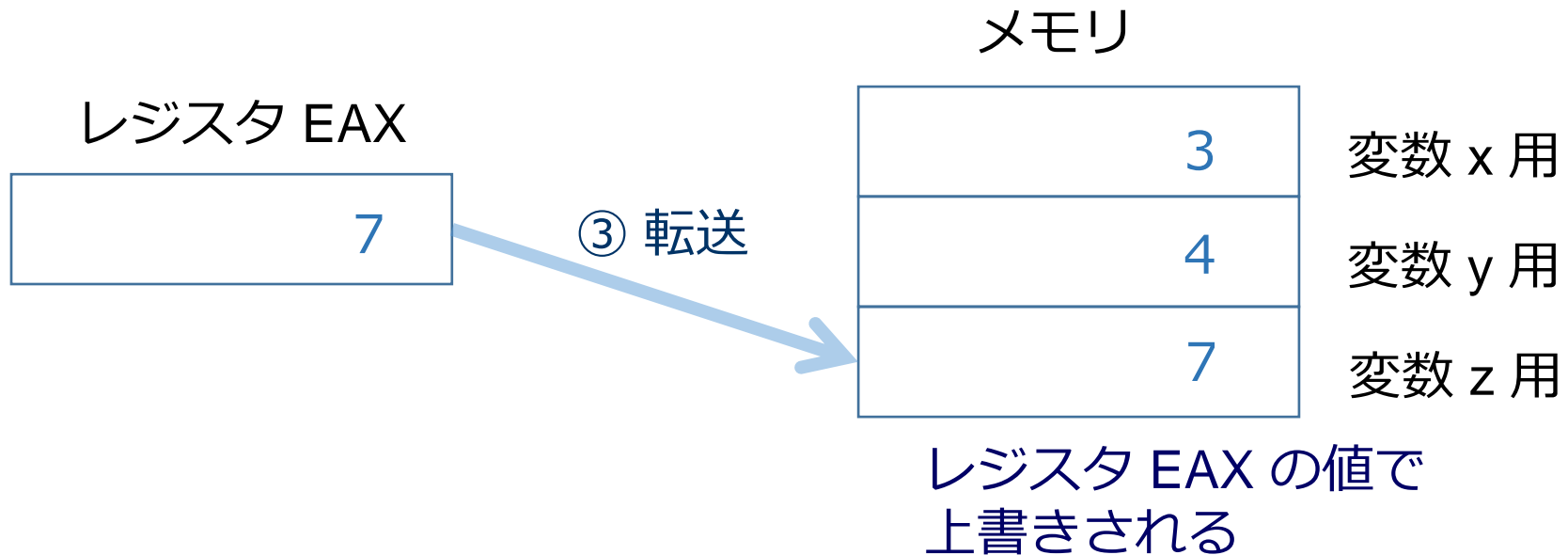
レジスタ EAX を使用



「z = x + y;」 部分で行われていること



レジスタ EAX を使用



レジスタを使っているのは？

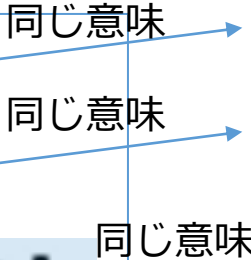
eax は
レジスタ名

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
```



命令

命令が対象とする相手である
オペランド

- 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

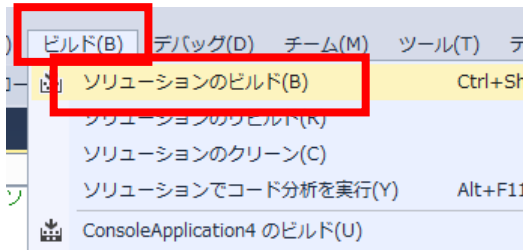
- Visual Studioのエディタを使って、ソースファイルを編集しなさい

```
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

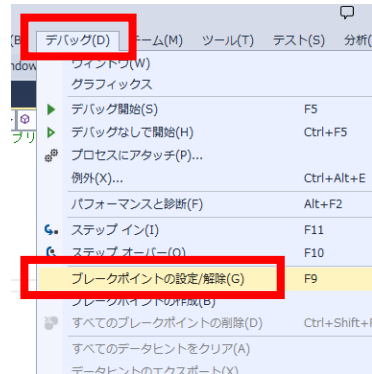
→ 表示されなければ、プログラムのミスを自分で確認し、修正して、ビルドをやり直す



```
出力
出力元(S): ビルド
1>----- ビルド開始: プロジェクト:ConsoleApplication6, 構成:Debug Win32 -----
1> stdafx.cpp
1> ConsoleApplication6.cpp
1> ConsoleApplication6.vcxproj -> e:\documents\visual studio 2015\Projects\
1> ConsoleApplication6.vcxproj -> e:\documents\visual studio 2015\Projects\
=====
= ビルド: 1 正常終了, 0 失敗, 0 更新不要, 0 スキップ =
=====
```

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

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



```
7
8
9
10
11
12
13
14
15
```

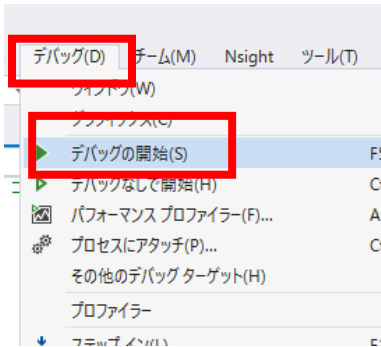
```
int main()
{
    static int x, y, z;
    x = 3;
    y = 4;
    z = x + y;
    return 0;
}
```

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」

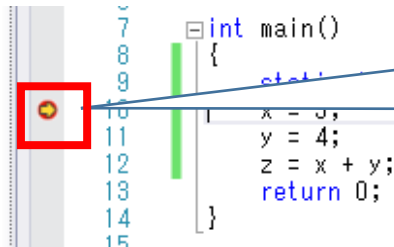
③ ブレークポイントが設定されるので
確認.
赤丸がブレークポイントの印

- Visual Studioで、デバッガーを起動しなさい。



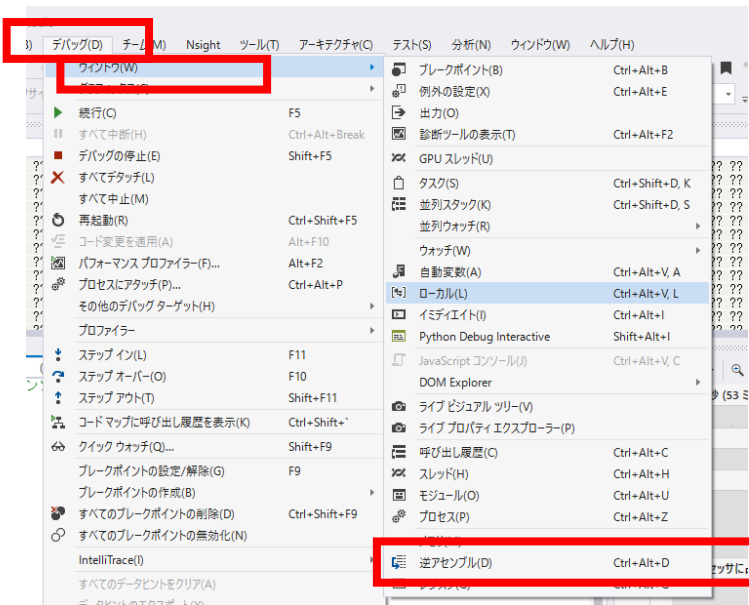
「デバッグ」
→ 「デバッグ開始」

- 「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
中断している

- 「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。



```
逆アセンブル -> ConsoleApplication7.cpp
アドレス(A): main(void)
表示オプション
00CA185C lea     edi,[ebp-000h]
00CA1862 mov     ecx,30h
00CA1867 mov     eax,0CCCCCCCCh
00CA186C rep stos dword ptr es:[edi]
        static int x, y, z;
        x = 3;
00CA186E mov     dword ptr [x (0CA9138h)],3 |
        y = 4;
00CA1878 mov     dword ptr [y (0CA913Ch)],4
        z = x + y;
00CA1882 mov     eax,dword ptr [x (0CA9138h)]
00CA1887 add     eax,dword ptr [y (0CA913Ch)]
00CA188D mov     dword ptr [z (0CA9140h)],eax
        return 0;
00CA1892 xor     eax,eax
}
00CA1894 pop     edi
00CA1895 pop     esi
00CA1896 pop     ebx
00CA1897 mov     esp,ebp
00CA1899 pop     ebp
00CA189A ret
```

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 dword 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
```

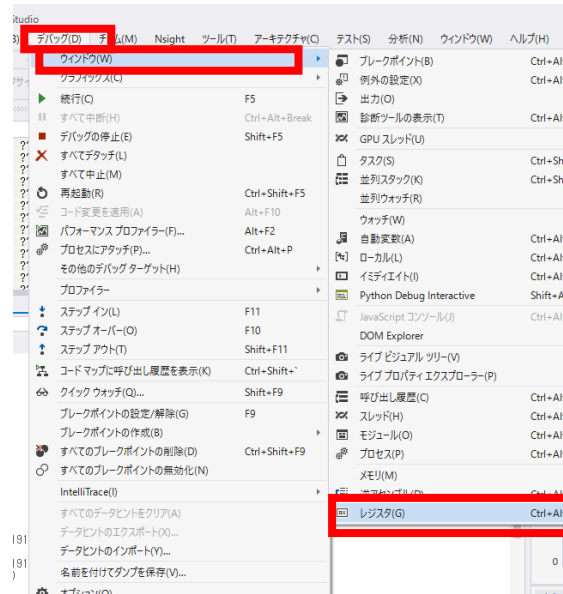
レジスタ名 **eax** の確認！

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。

```
0001887 mov     eax,0CCCCCCCCh
000188C rep stos dword ptr es:[edi]
static int x, y, z;
x = 3;
000186E mov     dword ptr [x (0CA9138h)],3
y = 4;
0001878 mov     dword ptr [y (0CA913Ch)],4
z = x + y;
0001882 mov     eax,dword ptr [x (0CA9138h)]
0001887 add     eax,dword ptr [y (0CA913Ch)]
000188D mov     dword ptr [z (0CA9140h)],eax
return 0;
0001892 xor     eax,eax
}
0001894 pop     edi
0001895 ret
```

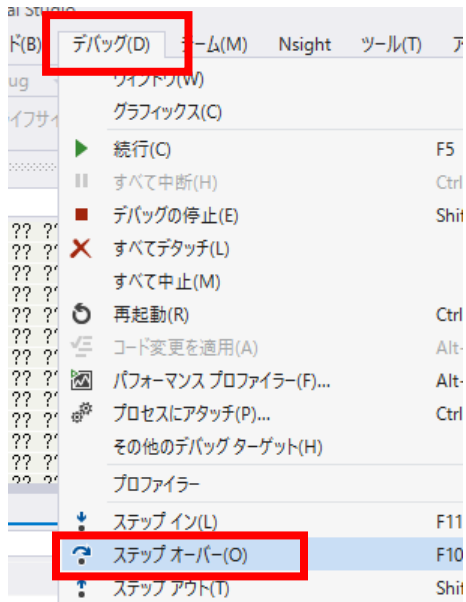
デバッガを起動済みで，プログラムの実行が中断しているときに・・・



② レジスタが表示される。

① 「デバッグ」
→ 「ウィンドウ」 → 「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。



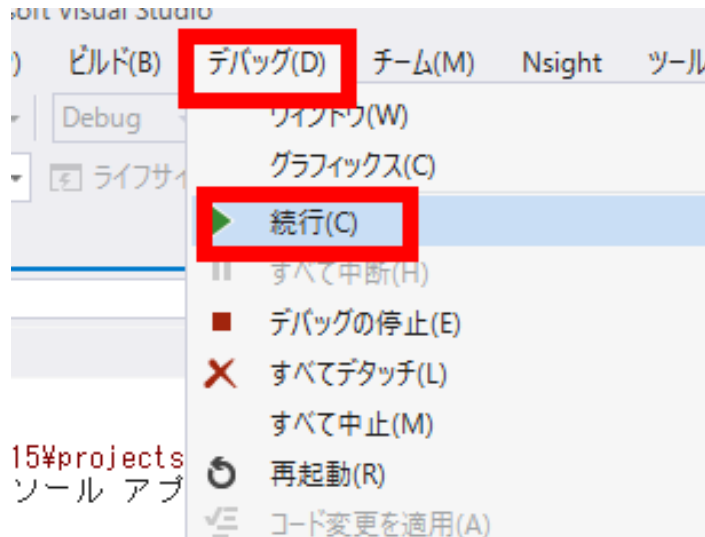
「デバッグ」
→ 「ステップオーバー」
(あるいは F10 キー)

レジスタ
EAX = 00000000 EBX

レジスタ
EAX = 00000003 EBX

レジスタ
EAX = 00000007 EBX

- 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」