

# ca-8. 算術演算命令

(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



8-1 算術演算の例

8-2 算術演算命令

# 8-1 算術演算の例

# 足し算 add の例



```
int main()
{
    int a;
    _asm {
        mov a, 100;
        add a, 200;
    }
    printf("a = %d", a);
    return 0;
}
```

## アセンブリ言語のプログラム

<pre>mov a, 100;</pre>	a に 100 をセット
<pre>add a, 200;</pre>	a に 200 を足す

## 実行結果の例

```
C:\WINDOWS\system
a = 300 続行するに
```

- 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

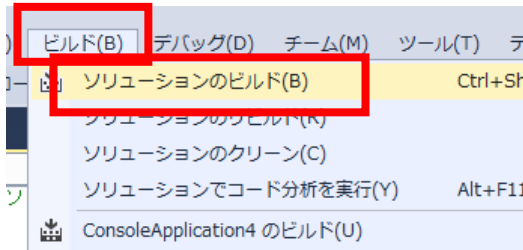
- Visual Studioのエディタを使って、ソースファイルを編集しなさい

```
int main()  
{  
    int a;  
    _asm {  
        mov a, 100;  
        add a, 200;  
    }  
    printf("a = %d", a);  
    return 0;  
}
```

6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

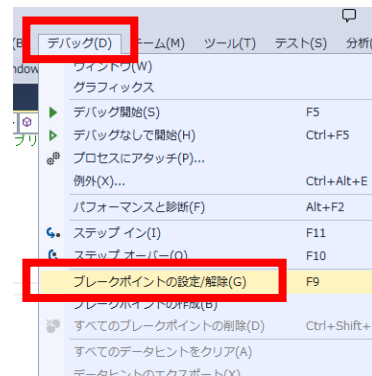
→ 表示されなければ, プログラムのミスを自分で確認し, 修正して, ビルドをやり直す



```
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「printf」の行に、ブレークポイントを設定しなさい

```
int main()
{
    int a;
    _asm {
        mov a, 100;
        add a, 200;
    }
    printf( a = %d , a);
    return 0;
}
```



```
7 int main()
8 {
9     int a;
10    _asm {
11        mov a, 100;
12        add a, 200;
13    }
14    printf("a = %d", a);
15    return 0;
16 }
```

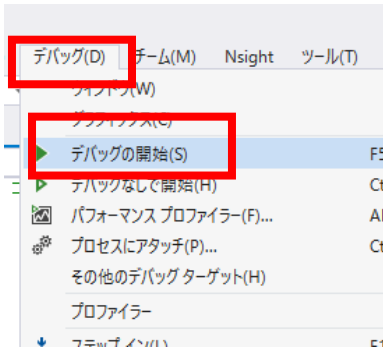
① 「printf」の行をマウスでクリック

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



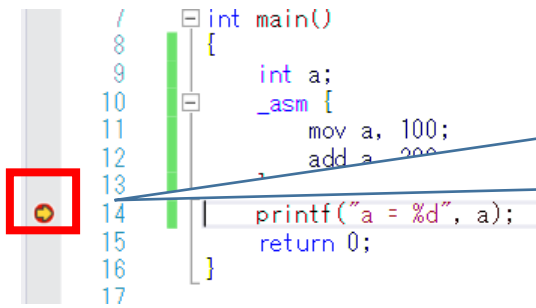
- Visual Studioで、デバッガーを起動しなさい。



「デバッグ」  
→ 「デバッグ開始」

- 「printf」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

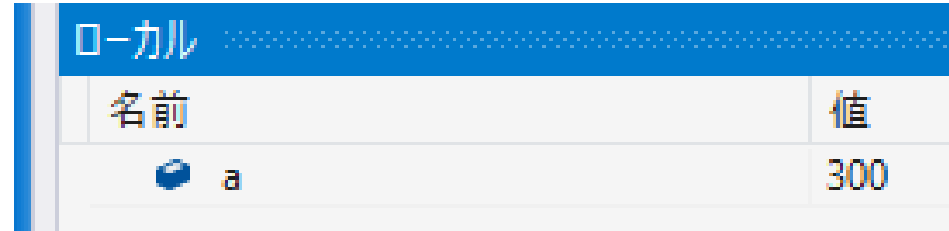
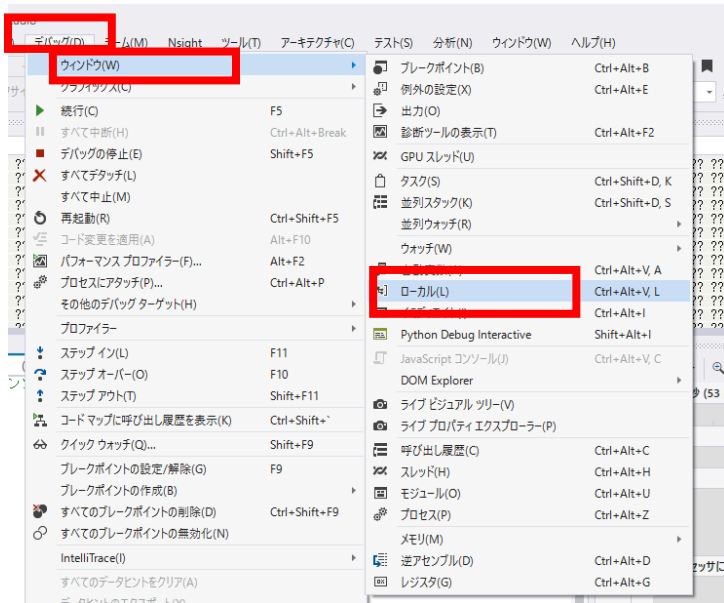
```
8 | int main()  
9 | {  
10 |     int a;  
11 |     _asm {  
12 |         mov a, 100;  
13 |         add a, 200;  
14 |     }  
15 |     printf("a = %d", a);  
16 |     return 0;  
17 | }
```



The image shows a code editor window with a C program. A red square highlights a breakpoint icon on the left margin of line 14, which contains the printf statement. The code is as follows:  
8 | int main()  
9 | {  
10 | int a;  
11 | \_asm {  
12 | mov a, 100;  
13 | add a, 200;  
14 | }  
15 | printf("a = %d", a);  
16 | return 0;  
17 | }

「printf」の行で実行が  
中断している

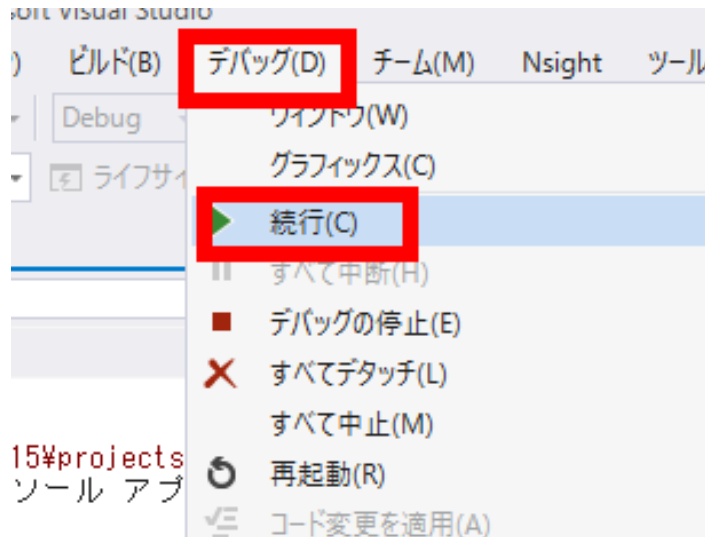
- 「printf」の行で、実行が中断した状態で、変数の値を表示させなさい。手順は次の通り。



② 変数名と値の対応表が表示される

① 「デバッグ」  
→ 「ウィンドウ」  
→ 「ローカル」

- 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」

- 次のように書き替えて，同じ手順を繰り返さないさい。そして，変数  $a$  の値を確認しなさい

```
int main()
{
    int a;
    _asm {
        mov a, 30;
        add a, 20;
    }
    printf("a = %d", a);
    return 0;
}
```

ローカル	
名前	値
a	50

**add 加算**

- 次のように書き替えて，同じ手順を繰り返さないさい。そして，変数 a の値を確認しなさい

```
int main()
{
    int a;
    _asm {
        mov a, 30;
        sub a, 20;
    }
    printf("a = %d", a);
    return 0;
}
```

ローカル	
名前	値
a	10

**sub 加算**

- 次のように書き替えて，同じ手順を繰り返さないさい。そして，変数 a の値を確認しなさい

```
int main()
{
    int a;
    _asm {
        mov a, 30;
        imul eax, a, 20;
        mov a, eax;
    }
    printf("a = %d", a);
    return 0;
}
```

名前	値
a	50

**imul 乗算**  
**次ページに解説**

```
int main()
```

```
{
```

```
int a;      アセンブリ言語の  
_asm {     プログラム
```

```
mov a, 30;  
imul eax, a, 20;  
mov a, eax;
```

- ① a に 30 をセット
- ② a × 20 の結果を,  
**レジスタ** eax にセット
- ③ a に**レジスタ** eax の  
値をセット

```
printf("a = %d", a);
```

```
return 0;
```

```
}
```

## 8-2 算術演算命令



# 算術演算命令とは



- 数に関する各種の演算を行う命令

加算

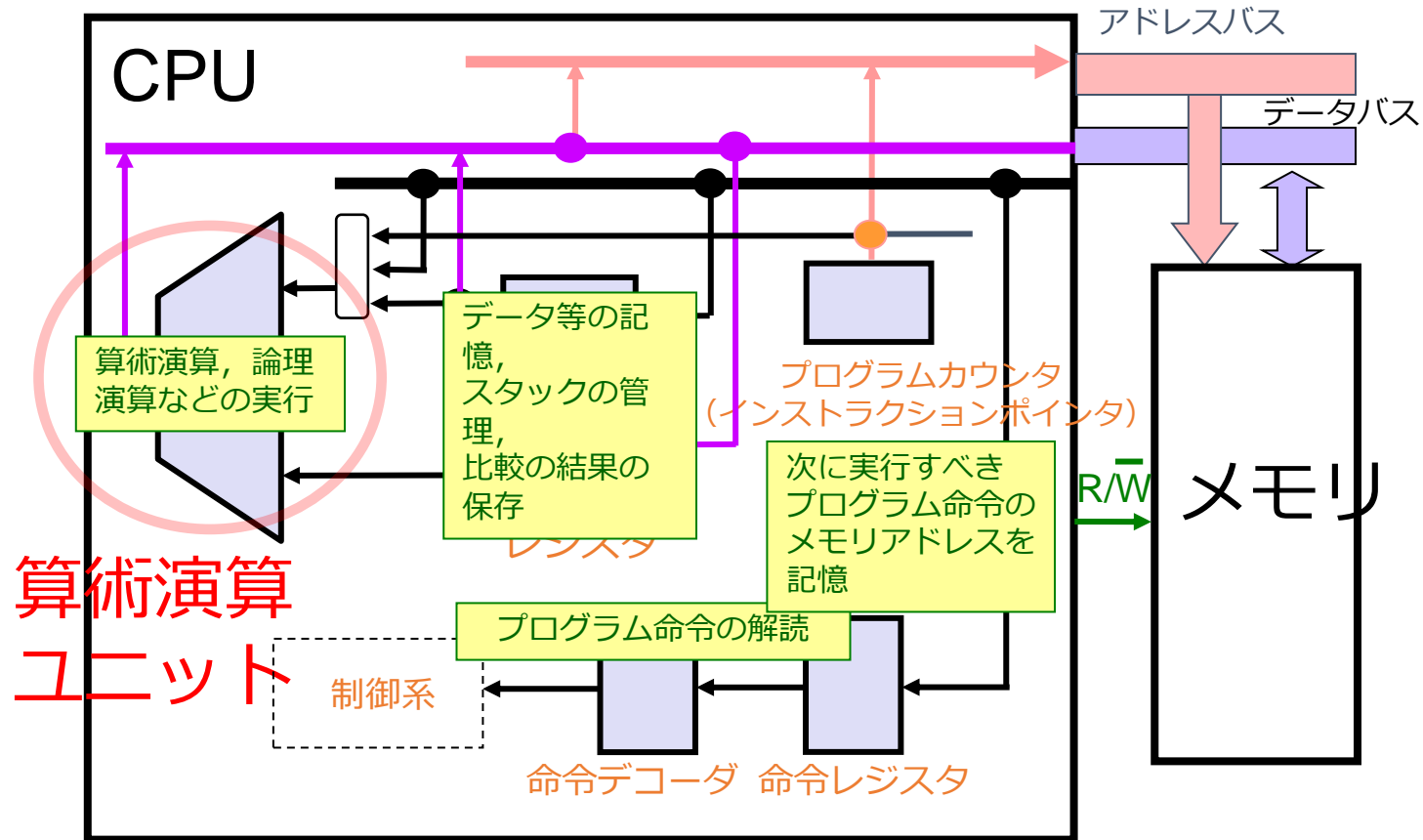
減算

乗算

除算

算術シフト など

# プロセッサの中の算術演算ユニット



# Pentium 系列プロセッサでの算術演算の例



Visual C++	アセンブリ言語
<code>a = a + 100;</code>	<code>add eax,64h</code>
<code>a = a - 100;</code>	<code>sub eax,64h</code>
<code>a = a * 100;</code>	<code>imul eax,dword ptr ds:[0C08130h],64h</code>
<code>a = a / 100;</code>	<code>mov ecx,64h</code> <code>idiv eax,ecx</code>

a は整数の変数

# Pentium 系列プロセッサでの算術演算の例



Visual C++	アセンブリ言語
<code>a = a + 100;</code>	<code>add eax, 64h</code>
<code>a = a - 100;</code>	<code>sub eax, 64h</code>
<code>a = a * 100;</code>	<code>imul eax, dword ptr ds:[0C08130h], 64h</code>
<code>a = a / 100;</code>	<code>mov ecx, 64h</code> <code>idiv eax, ecx</code>

a は整数の変数

# 加算



## Visual C++ の プログラム

## アセンブリ言語

```
int main()
{
    int a;
    a = 200;
    a = a + 100;
    printf("a = %d", a);
    return 0;
}
```

同じ意味

```
mov     eax,dword ptr [a]
add     eax,64h
mov     dword ptr [a],eax
```

# 減算



## Visual C++ の プログラム

## アセンブリ言語

```
int main()
{
    int a;
    a = 200;
    a = a - 100;
    printf("a = %d", a);
    return 0;
}
```

同じ意味

```
mov     eax,dword ptr [a]
sub     eax,64h
mov     dword ptr [a],eax
```

## Visual C++ の プログラム

## アセンブリ言語

```
int main()
{
    int a;
    a = 50;
    a = a * 40;
    printf("a = %d", a);
    return 0;
}
```

同じ意味

```
imul     eax,dword ptr [a],28h
mov      dword ptr [a],eax
```

IMUL は**独特**.  
第2オペランドと第3オペランドを  
乗算して, 第1オペランドに格納

## Visual C++ の プログラム

## アセンブリ言語

```
int main()
{
    int a;
    a = 300;
    a = a / 5;
    printf("a = %d", a);
    return 0;
}
```

同じ意味

```
mov     eax,dword ptr [a]
cdq
mov     ecx,5
idiv   eax,ecx
mov     dword ptr [a],eax
```

Pentium 系列プロセッサで  
の除算は、**割った余りを扱う**  
ための準備が必要で、  
プログラムが長くなる



- 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

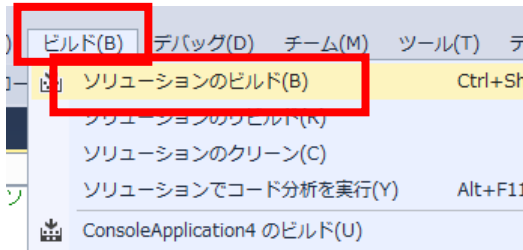
- Visual Studioのエディタを使って、ソースファイルを編集しなさい

```
int main()
{
    int a;
    a = 200;
    a = a + 100;
    printf("a = %d", a);
    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

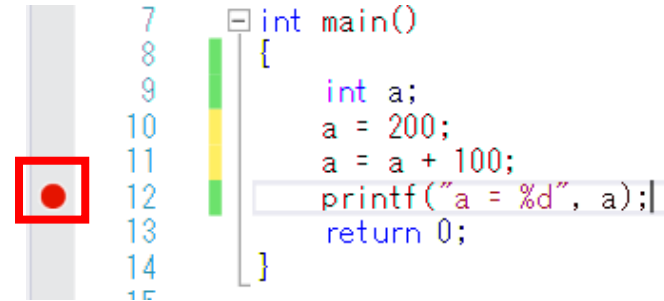
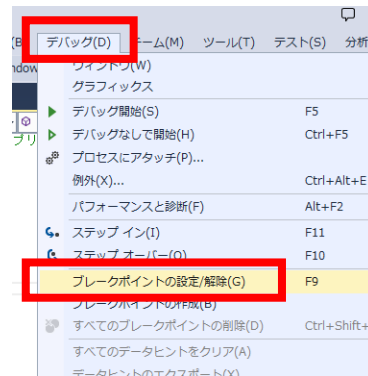


```
出力
出力元(S): ビルド
1>----- ビルド開始: プロジェクト:ConsoleApplication6, 構成:Debug Win32 -----
1> stdafx.cpp
1> ConsoleApplication6.cpp
1> ConsoleApplication6.vcxproj -> e:\documents\visual studio 2015\Projects\
1> ConsoleApplication6.vcxproj -> e:\documents\visual studio 2015\Projects\
=====
= ビルド: 1 正常終了, 0 失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

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

※ 設定されていないときは、下のように操作して設定する

```
int main()
{
    int a;
    a = 200;
    a = a + 100;
    printf("a = %d", a);
    return 0;
}
```

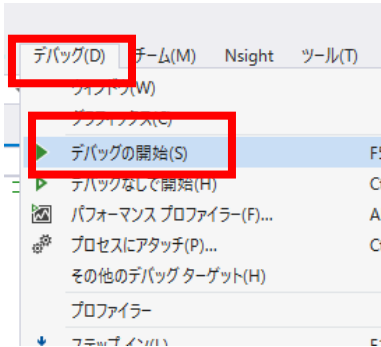


① 「printf」の行をマウスでクリック

② 「デバッグ」→「ブレークポイントの設定/解除」

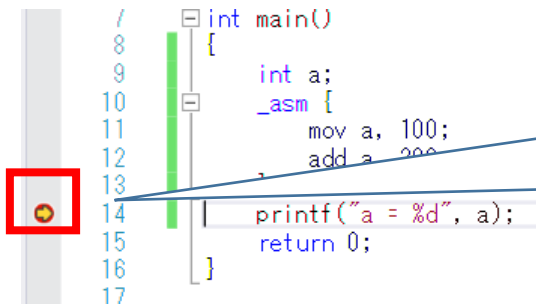
③ ブレークポイントが設定されるので確認.  
赤丸がブレークポイントの印

- Visual Studioで、デバッガーを起動しなさい。



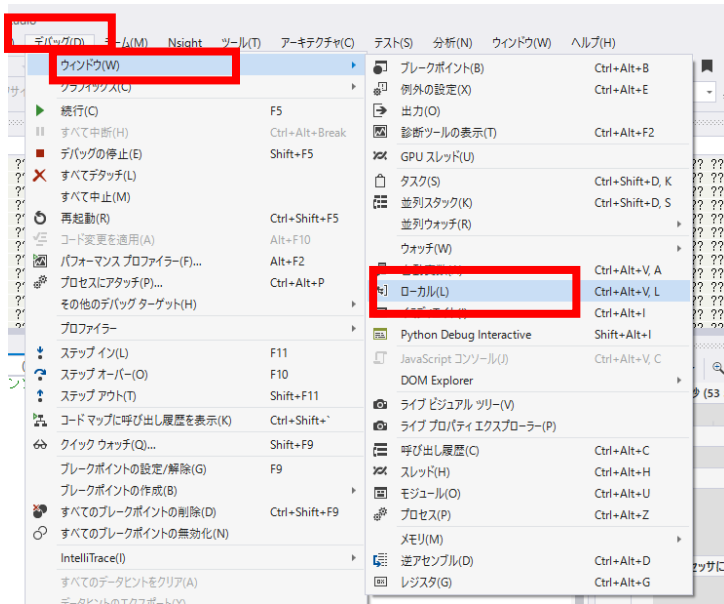
「デバッグ」  
→ 「デバッグ開始」

- 「printf」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



「printf」の行で実行が  
中断している

- 「printf」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。



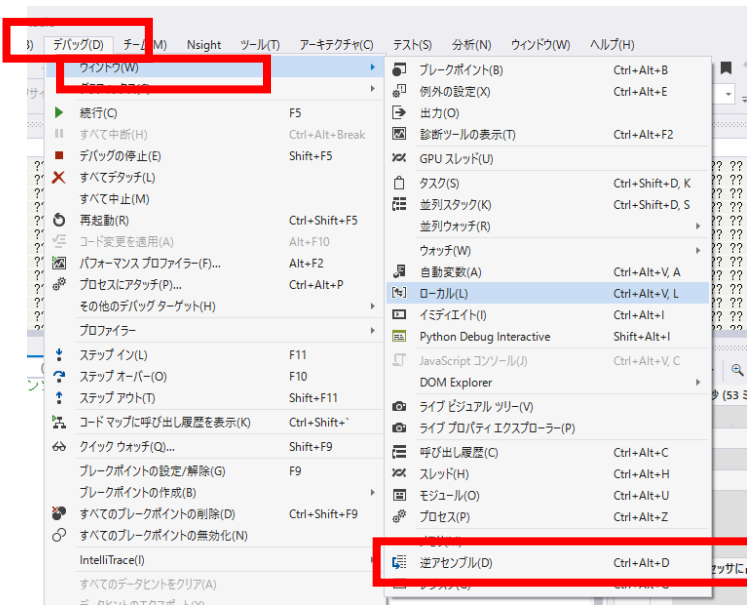
The image shows a variable view window titled 'ローカル' (Local). It contains a table with two columns: '名前' (Name) and '値' (Value). The table has one row with the variable name 'a' and the value '300'.

名前	値
a	300

① 「デバッグ」  
→ 「ウィンドウ」  
→ 「ローカル」

② 変数名と値の対応表が表示される

- 「printf」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 - ConsoleApplication9.cpp
アドレス(A): main(void)
表示オプション
00CC1799 push     ebx
00CC179A push     esi
00CC179B push     edi
00CC179C lea     edi,[ebp-00Ch]
00CC17A2 mov     ecx,39h
00CC17A7 mov     eax,0CCCCCCCch
00CC17AC rep stos dword ptr es:[edi]
        int a;
        a = 200;
00CC17AE mov     dword ptr [a],00C8h
        a = a + 100;
00CC17B5 mov     eax,dword ptr [a]
00CC17B8 add     eax,64h
00CC17BB mov     dword ptr [a],eax
        printf("a = %d", a);
00CC17BE mov     eax,dword ptr [a]
00CC17C1 push   eax
00CC17C2 push   offset string "a = %d" (00C8B30h)
00CC17C7 call   _printf (00C1318h)
00CC17CC add     esp,8
        return 0;
00CC17CF xor     eax,eax
    }
00CC17D1 pop     edi
00CC17D2 pop     esi
00CC17D3 pop     ebx
00CC17D4 add     esp,00Ch
00CC17DA cmp     ebp,esp
00CC17DC call   __RTC_CheckEsp (00C110Eh)
00CC17E1 mov     esp,ebp
```

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、「a = a + 100」のところにある「add」を確認しなさい

```
00001785  mov     eax,dword ptr [a]
00001788  add     eax,64h
000017BB  mov     dword ptr [a],eax
printf("a = %d", a);
```



- 次のように書き替えて，同じ手順を繰り返さない。
- 逆アセンブルで「a = a - 10」のところの sub を確認しなさい。

```
int main()
{
    int a;
    a = 200;
    a = a - 100;
    printf("a = %d", a);
    return 0;
}
```

```
01101784 mov     eax,dword ptr [a],0x00000000
a = a - 100;
01101785 mov     eax,dword ptr [a]
01101788 sub     eax,64h
0110178B mov     dword ptr [a],eax
```

- 次のように書き替えて，同じ手順を繰り返さない。
- 逆アセンブルで「a = a \* 40」のところの imul を確認しなさい。

```
int main()  
{  
    int a;  
    a = 50;  
    a = a * 40;  
    printf("a = %d", a);  
    return 0;  
}
```

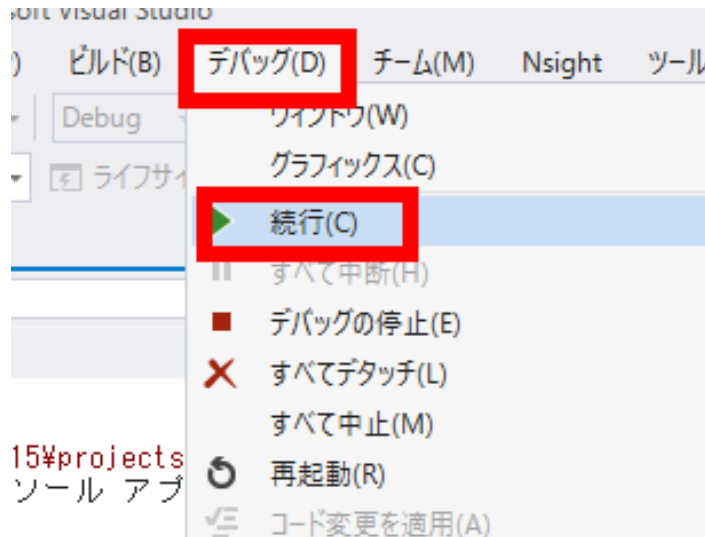
```
000D17A8 mov     eax, dword ptr [a], 020  
    a = a * 40;|  
008D17B5 imul   eax, dword ptr [a], 28h  
008D17B9 mov     dword ptr [a], eax  
    printf("a = %d", a);
```

- 次のように書き替えて，同じ手順を繰り返さない。
- 逆アセンブルで「 $a = a / 5$ 」のところの `idiv` を確認しなさい。

```
int main()
{
    int a;
    a = 300;
    a = a / 5;
    printf("a = %d", a);
    return 0;
}
```

```
00CE17B5  mov     eax,dword ptr [a]
00CE17B8  cdq
00CE17B9  mov     ecx,5
00CE17BE  idiv   eax,ecx
00CE17C0  mov     dword ptr [a],eax
printf("a = %d", a):
```

- 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」