

# 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 {
```

アセンブリ言語のプログラム

```
    mov a, 100;
    add a, 200;
```

a に 100 をセット  
a に 200 を足しこむ

```
}
```

```
printf("a = %d", a);
return 0;
```

```
}
```

実行結果の例

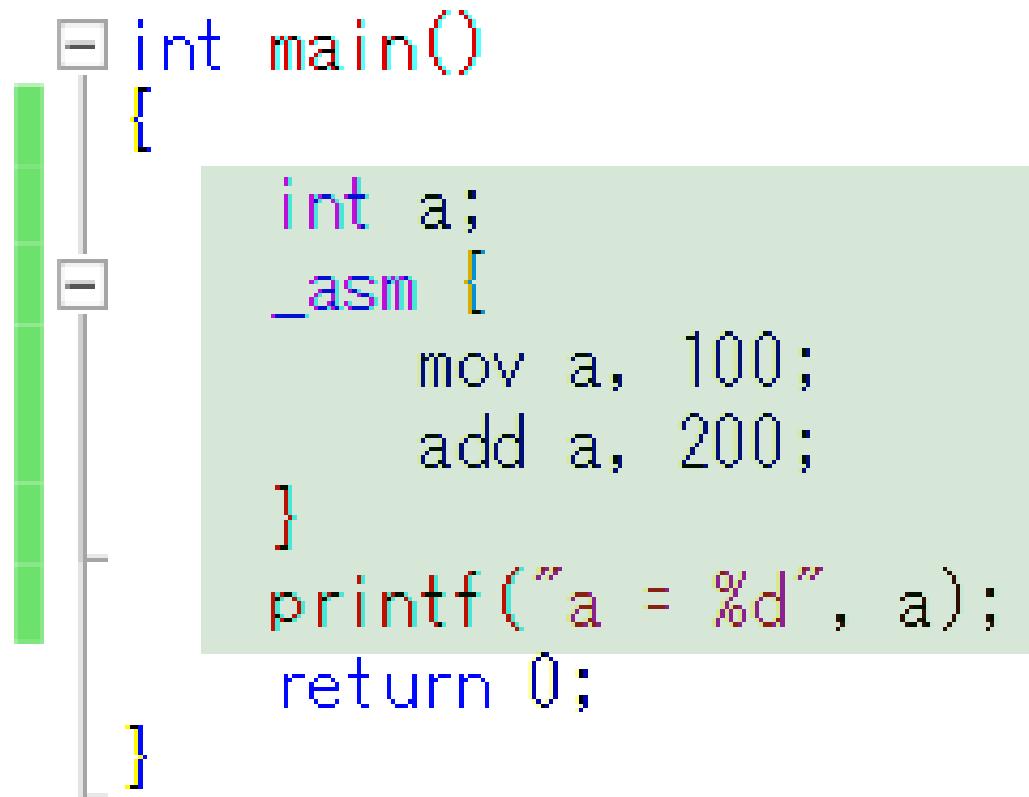
```
C:\WINDOWS\system
a = 300 続行するに(
```

# 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

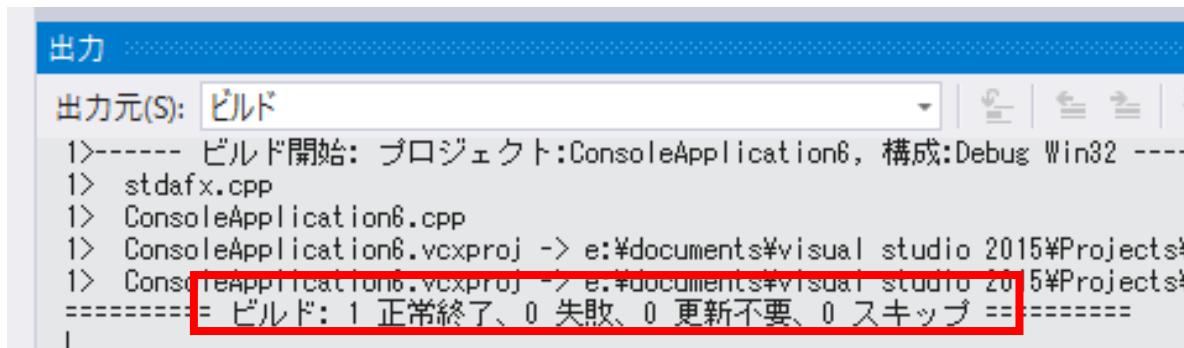
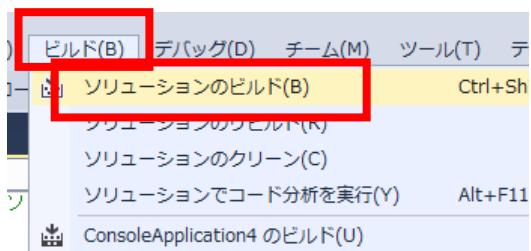


```
int main()
{
    int a;
    _asm {
        mov a, 100;
        add a, 200;
    }
    printf("a = %d", a);
    return 0;
}
```

A screenshot of the Visual Studio code editor. On the left, there is a vertical green bar with three white square markers. The first marker is above the opening brace of the main function, the second is above the opening brace of the assembly block, and the third is below the closing brace of the assembly block. The assembly block itself is highlighted with a light green background. The code is written in C, defining a main function that declares an integer 'a', contains an assembly block with two instructions (mov a, 100; add a, 200;), prints the value of 'a' using printf, and returns 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

A screenshot of the Visual Studio Output window titled '出力'. The dropdown menu shows '出力元(S): ビルド'. The window displays the build log:

```
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\ConsoleApplication6\ConsoleApplication6.vcxproj -> e.\#documents\visual studio 2015\Projects\ConsoleApplication6\ConsoleApplication6.vcxproj
===== ビルド: 1 正常終了, 0 失敗, 0 更新不要, 0 スキップ =====
```

The last line of the log, '===== ビルド: 1 正常終了, 0 失敗, 0 更新不要, 0 スキップ =====', is highlighted with a red box.

- 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;
}
```



① 「printf」の行をマウスでクリック

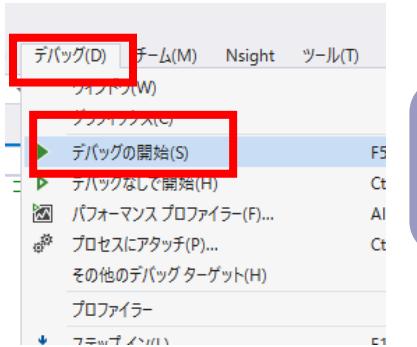
② 「デバッグ」 → 「ブレークポイントの設定/解除」

```
7
8
9
10
11
12
13
14
15
16
17
```

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

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」

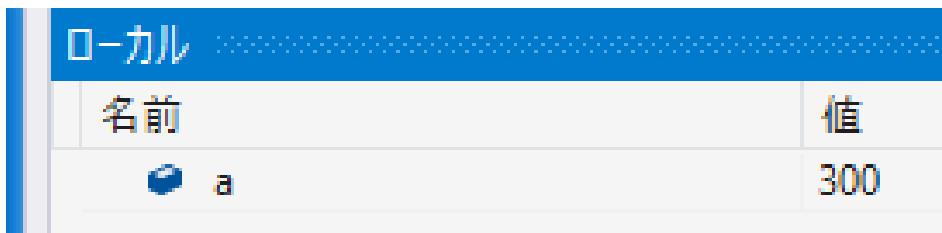
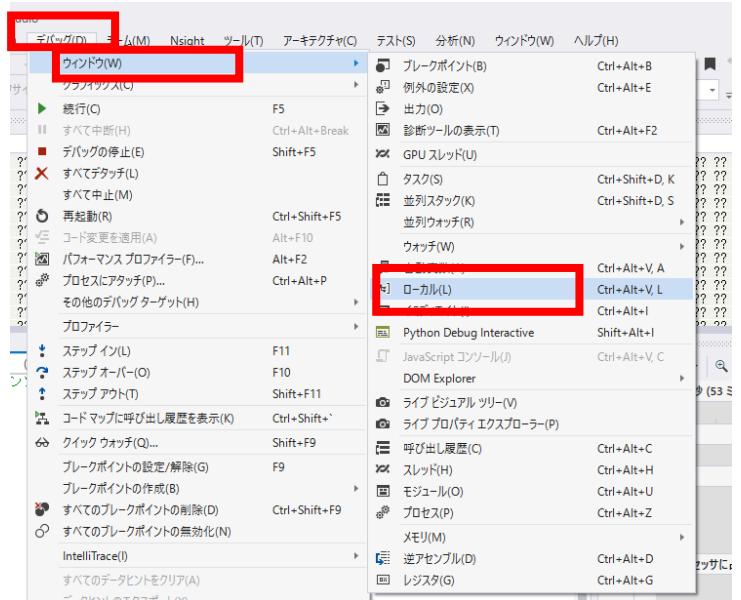
- 「printf」の行で、実行が中断することを確認しなさい
- あとで使うので、中断したままにしておくこと

```

7
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

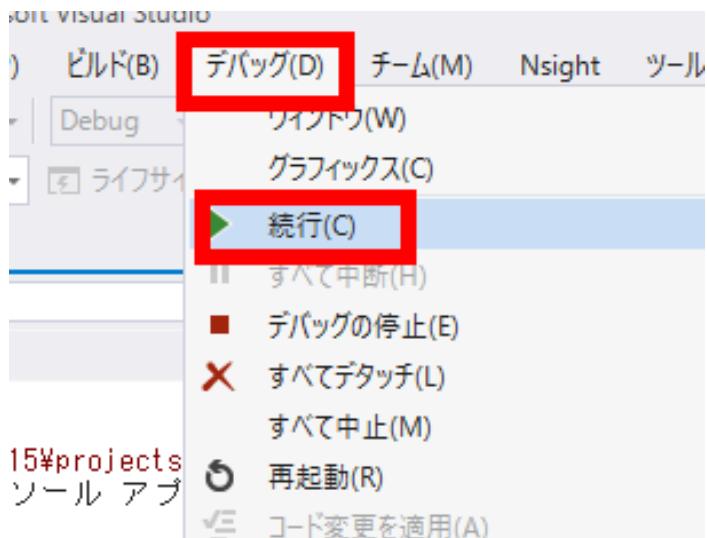
- 「printf」の行で、実行が中断した状態で、変数の値を表示させなさい。手順は次の通り。



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、同じ手順を繰り返しなさい。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、同じ手順を繰り返しなさい。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;
}
  
```

| 名前 | 値  |
|----|----|
| a  | 10 |

**sub 加算**

- 次のように書き替えて、同じ手順を繰り返しなさい。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;
    _asm {
        mov a, 30;
        imul eax, a, 20;
        mov a, eax;
    }
    printf("a = %d", a);
    return 0;
}
```

アセンブリ言語の  
プログラム

- 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



## 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

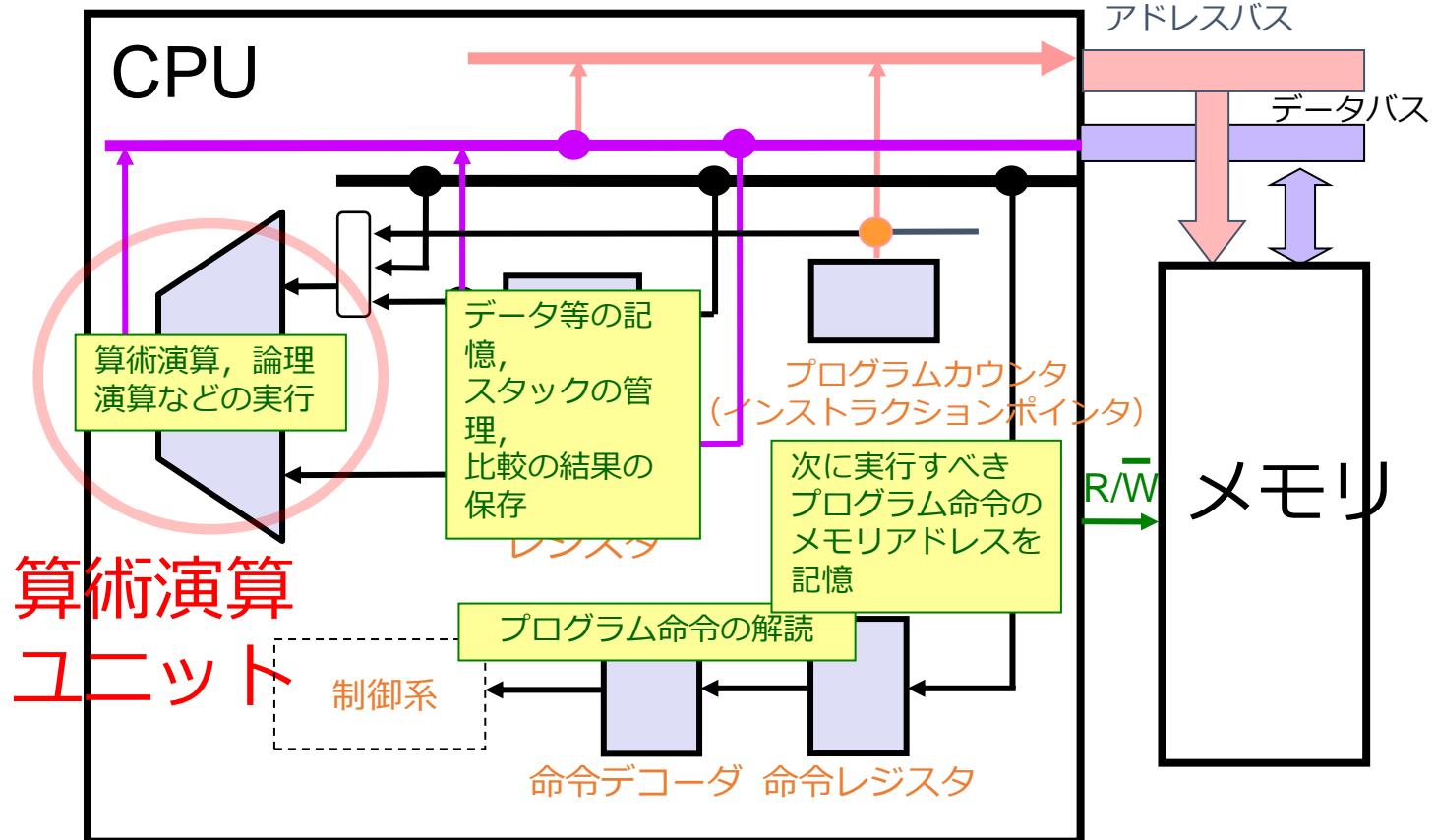
減算

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                              |
|--------------|--------------------------------------|
| a = a + 100; | add eax,64h                          |
| a = a - 100; | sub eax,64h                          |
| a = a * 100; | imul eax,dword ptr ds:[0C08130h],64h |
| a = a / 100; | mov ecx,64h<br>idiv eax,ecx          |

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                                |
|--------------|--|
| a = a + 100; | add eax, 64h                           |
| a = a - 100; | sub eax, 64h                           |
| a = a * 100; | imul eax, dword ptr ds:[0C08130h], 64h |
| a = a / 100; | mov ecx, 64h<br>idiv eax, ecx          |

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; // This line is highlighted with a red box
    printf("a = %d", a);
    return 0;
}
```

|     |                   |
|-----|-------------------|
| MOV | eax,dword ptr [a] |
| add | eax,64h           |
| MOV | dword ptr [a],eax |

同じ意味

# 減算



## Visual C++ の プログラム

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

## アセンブリ言語

|     |                   |
|-----|-------------------|
| mov | eax,dword ptr [a] |
| sub | eax,64h           |
| mov | dword ptr [a],eax |

同じ意味

# 乗算



Visual C++ の  
プログラム

アセンブリ言語

```
int main()
{
    int a;
    a = 50;
    a = a * 40; // 第2オペランド
    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

# 除算



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

|                |                   |
|----------------|-------------------|
| ~ ~ ~          | eax,dword ptr [a] |
| mov            |                   |
| cdq            |                   |
| mov            | ecx,5             |
| <b>idiv</b>    | eax,ecx           |
| mov            | dword ptr [a],eax |
| +f("%d", a); - |                   |

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

# 演習



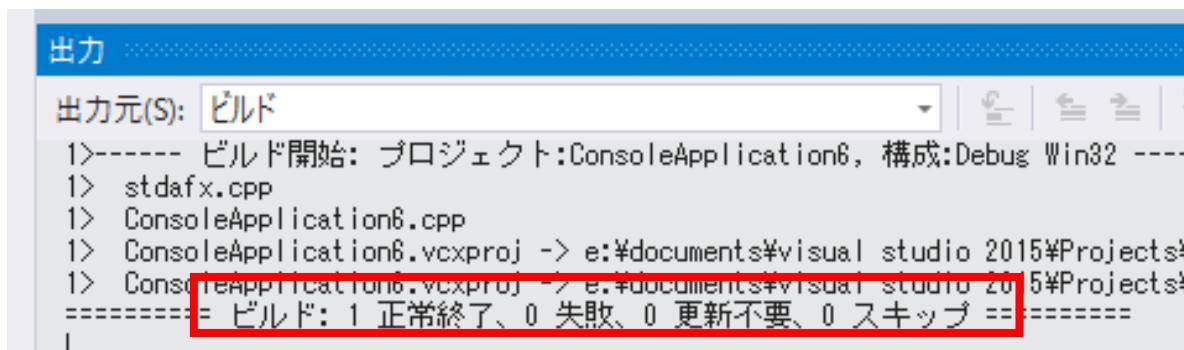
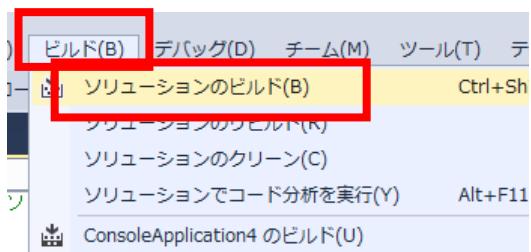
- Visual Studio を起動しなさい
- Visual Studio で, Win32 コンソールアプリケーション用プロジェクトを新規作成しなさい  
プロジェクトの「名前」は何でもよい

- 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()
{
    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

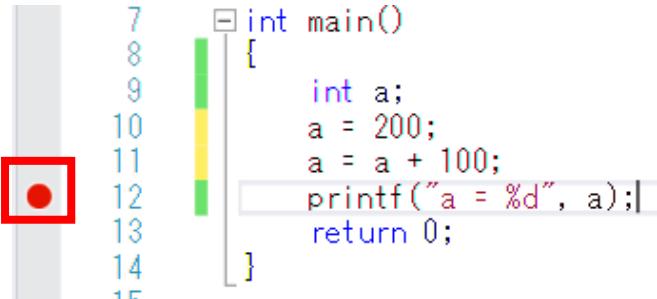


```
出力
出力元(S): ビルド
1>----- ビルド開始: プロジェクト:ConsoleApplication6, 構成:Debug Win32 -----
1> stdafx.cpp
1> ConsoleApplication6.cpp
1> ConsoleApplication6.vcxproj -> e:\documents\visual studio 2015\Projects\ConsoleApplication6\ConsoleApplication6.vcxproj -> e.\documents\visual studio 2015\Projects\ConsoleApplication6\ConsoleApplication6.vcxproj
===== 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;
}
```

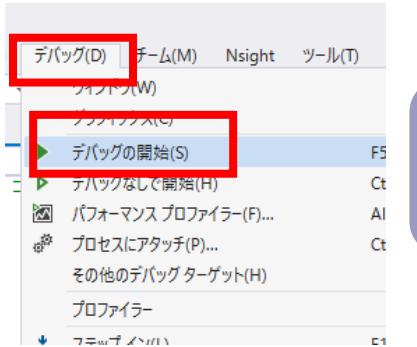


① 「printf」の行をマウスでクリック

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」

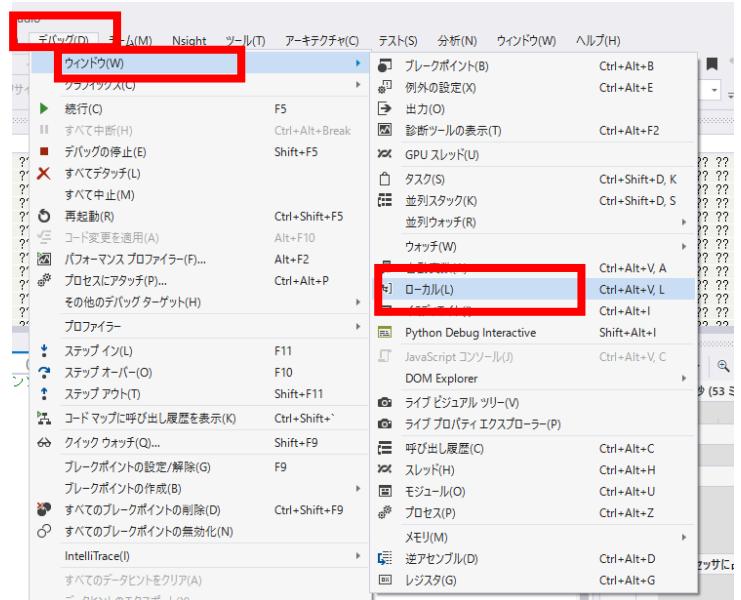
- 「printf」の行で、実行が中斷することを確認しなさい
- あとで使うので、中斷したままにしておくこと

```

7
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  
中斷している

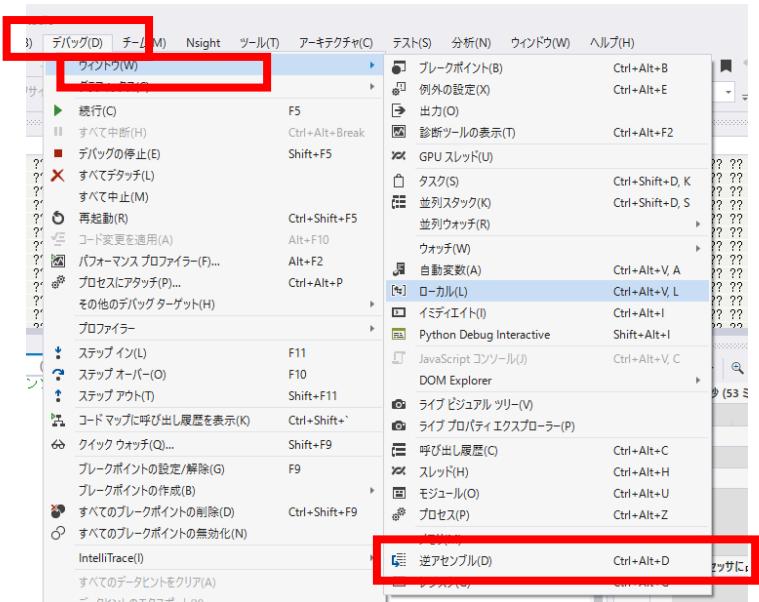
- 「printf」の行で、実行が中断した状態で、変数の値を表示させなさい。手順は次の通り。



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」

- 「printf」の行で、実行が中断した状態で、逆アセンブルを行なさい。



逆アセンブル ➔ X 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-0CCh]
00CC17A2 mov     ecx,33h
00CC17A7 mov     eax,0CCCCCCCCCh
00CC17AC rep stos dword ptr es:[edi]
int a;
a = 200;
00CC17AE mov     dword ptr [a],008h
a = a + 100;
00CC17B5 mov     eax,dword ptr [a]
00CC17B8 add     eax,64h
00CC17B9 mov     dword ptr [a],eax
printf("a = %d", a);
00CC17BE mov     eax,dword ptr [a]
00CC17C1 push    eax
00CC17C2 push    offset string "a = %d" (00C6B30h)
00CC17C7 call    _printf (0CC1316h)
00CC17C8 add     esp,8
return 0;
00CC17CF xor     eax,eax
}
00CC17D1 pop     edi
00CC17D2 pop     esi
00CC17D3 pop     ebx
00CC17D4 add     esp,0CCh
00CC17D8 cmp     ebp,esp
00CC17DC call    __RTC_CheckEsp (0CC110Eh)
00CC17E1 mov     esp,ebp

```

① 「デバッグ」 → 「ウィンドウ」 → 「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

```
000017B4    mov    a, a + 100;  
000017B5    mov    eax, dword ptr [a]  
000017B8    add    eax, 64h  
000017BB    mov    dword ptr [a], eax  
              printf("a = %d", a);
```

- 次のように書き替えて、同じ手順を繰り返しなさい。
- 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;
}
  
```

|  | OPCODE   | Mnemonic | Op. # | Op. Type          | Op. Description |
|--|----------|----------|-------|-------------------|-----------------|
|  |          | mov      |       |                   |                 |
|  | 011C17B5 | mov      |       | dword ptr [a]     | a = a - 100;    |
|  | 011C17B8 | sub      |       |                   | eax,64h         |
|  | 011C17BB | mov      |       | dword ptr [a],eax |                 |

- 次のように書き替えて、同じ手順を繰り返しなさい。
- 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;
}
```

|   |   |
|---|---|
| <pre>00000100  IMUL    dw a, eax           a = a * 40;  008D17B5  imul    eax,dword ptr [a],28h 008D17B9  mov     dword ptr [a],eax               nprintf("a = %d", a);</pre> | <pre>00000100  IMUL    dw a, eax           a = a * 40;  008D17B5  imul    eax,dword ptr [a],28h 008D17B9  mov     dword ptr [a],eax               nprintf("a = %d", a);</pre> |
|---|---|

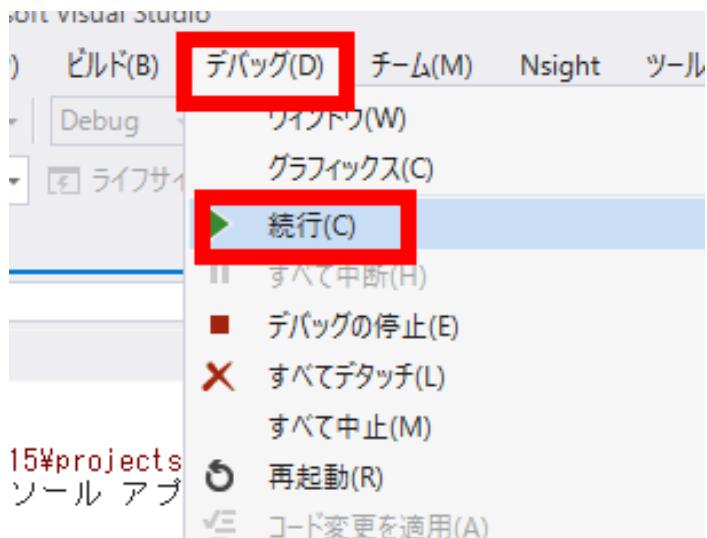
- 次のように書き替えて、同じ手順を繰り返しなさい。
- 逆アセンブルで「 $a = a / 5$ 」のところの idiv を確認しなさい。

```

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

| 逆アセンブル        | 意味                   | アセンブル |
|---------------|----------------------|-------|
| $a = a / 5;$  |                      |       |
| 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なさい。これで、デバッガーが終了する。



「デバッグ」  
→ 「続行」