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

a に 100 をセット

a に 200 を足しこむ

実行結果の例

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

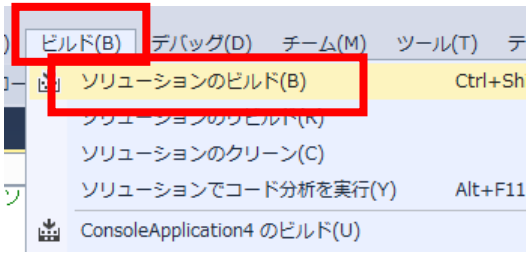
- Visual Studioのエディタを使って，ソースファイルを編集しなさい

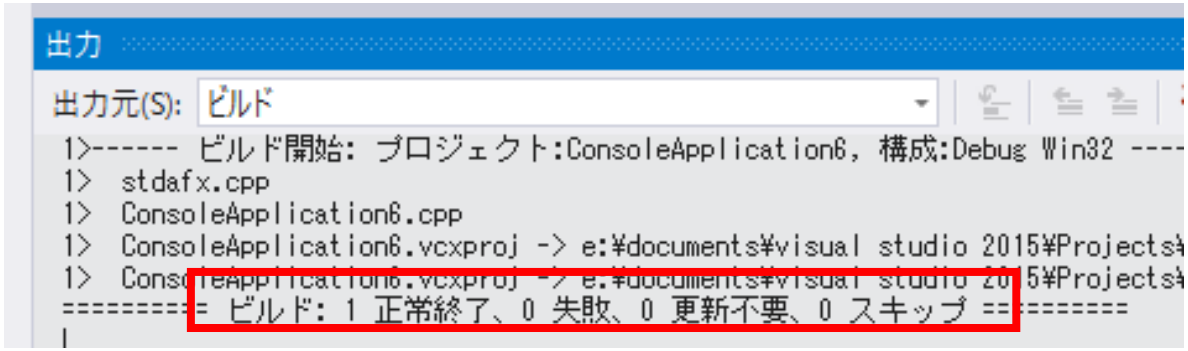
```
int main()
{
    int a;
    _asm {
        mov a, 100;
        add a, 200;
    }
    printf("a = %d", a);
    return 0;
}
```

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





- 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;
}
```

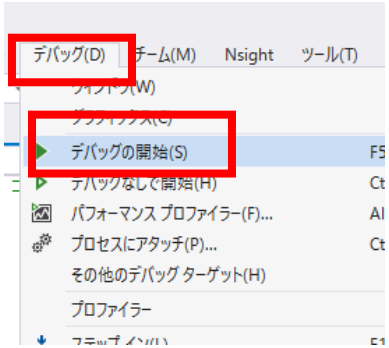

```
 7  int main()
 8  {
 9      int a;
10      _asm {
11          mov a, 100;
12          add a, 200;
13      }
14      printf("a = %d", a);
15      return 0;
16  }
```

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**」

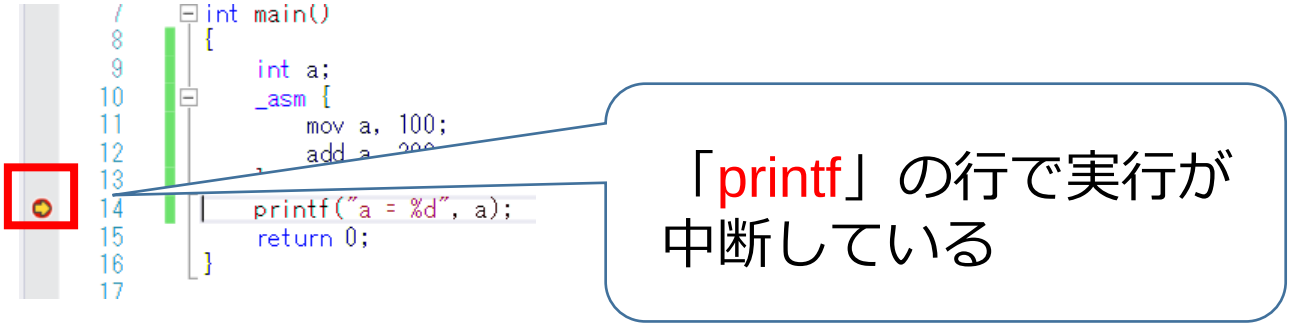
③ ブレークポイントが設定されるので確認．赤丸がブレークポイントの印

- Visual Studioで，デバッガーを起動しなさい.

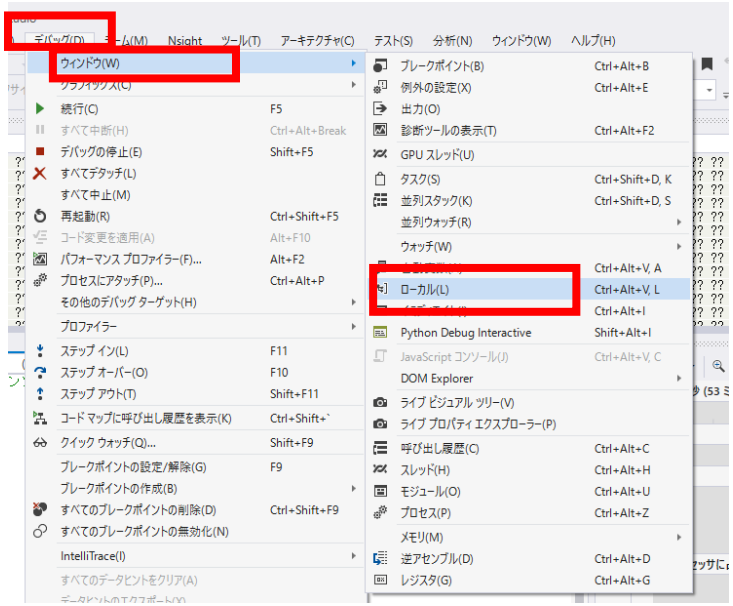


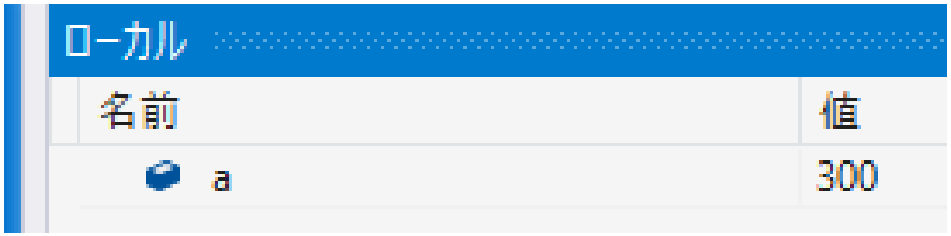
「デバッグ」
→ 「**デバッグ開始**」

- 「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



- 「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．

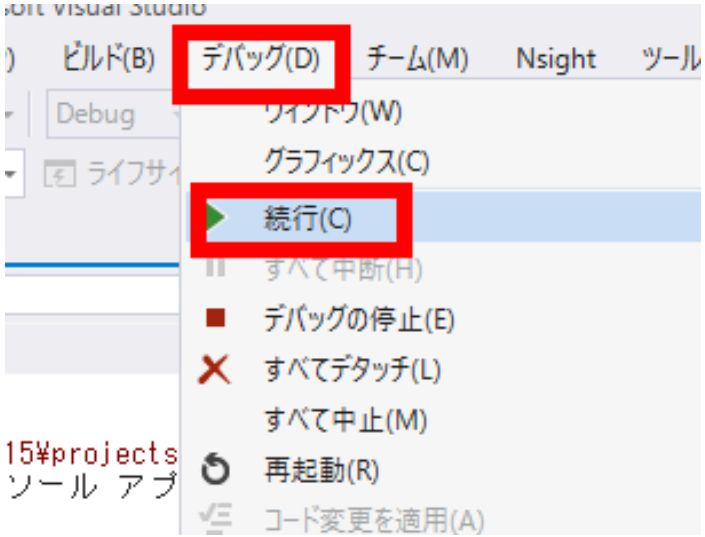




① 「デバッグ」
→ 「ウインドウ」
→ 「**ローカル**」

② 変数名と値の対応表が表示される

- 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;
}
```

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;
}
```

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;
}
```

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

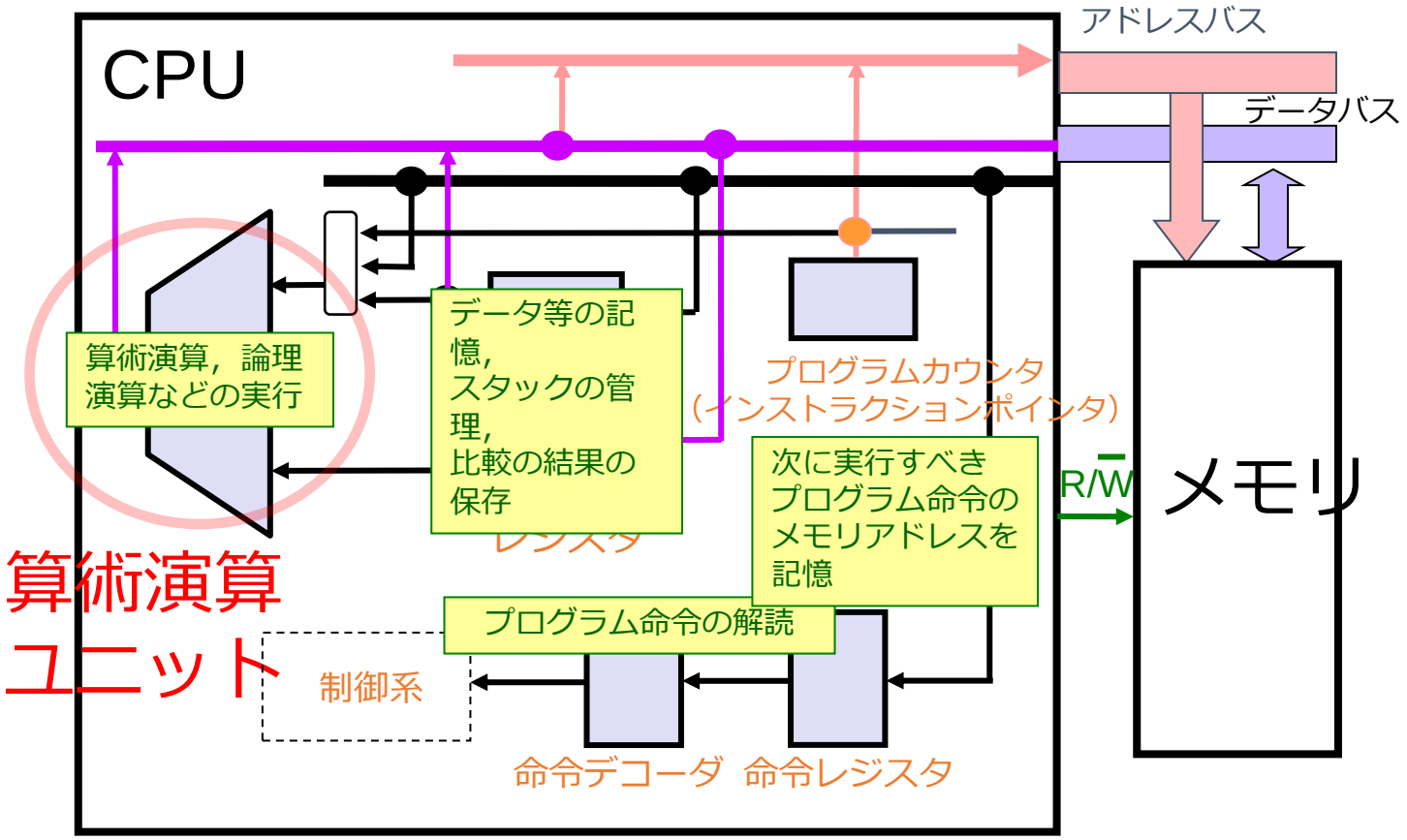
③ a に**レジスタ** eax の値をセット

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 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 idiv eax,ecx

a は整数の変数

加算

Visual C++ の
プログラム

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

同じ意味

アセンブリ言語

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

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;
}
```

同じ意味

アセンブリ言語

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

乗算

Visual C++ の
プログラム

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

同じ意味

アセンブリ言語

```
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

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

同じ意味

アセンブリ言語

```
    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

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

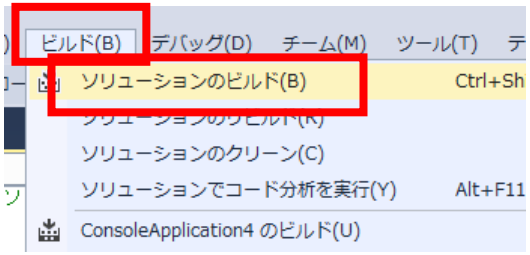
- Visual Studioのエディタを使って，ソースファイルを編集しなさい

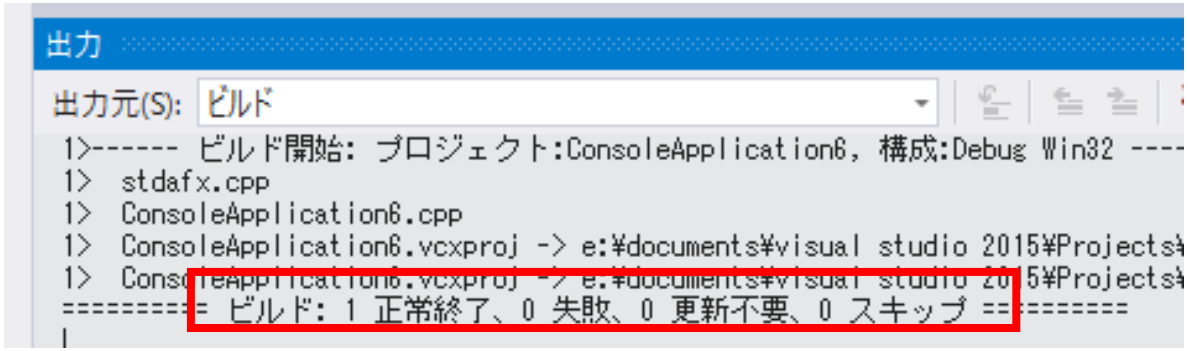
```
int main()
{
    int a;
    a = 200;
    a = a + 100;
    printf("a = %d", a);
    return 0;
}
```

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

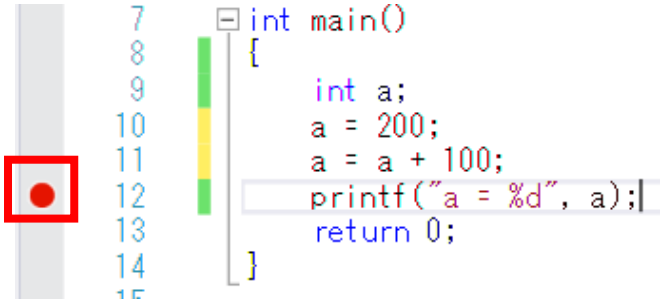




- Visual Studioで「printf」の行に，ブレークポイントを設定しなさい

※ **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;
}
```

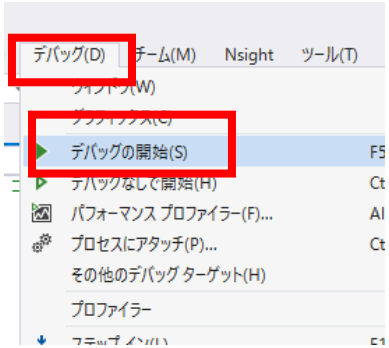


① 「**printf**」の行をマウスでクリック

② 「デバッグ」→「**ブレークポイントの設定/解除**」

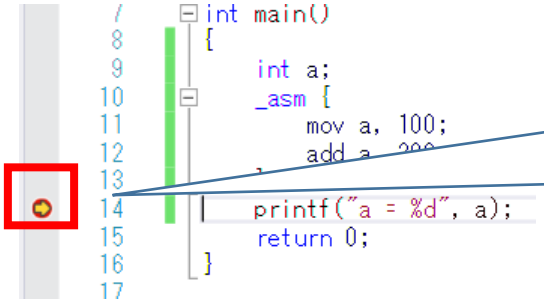
③ ブレークポイントが設定されるので確認．赤丸がブレークポイントの印

- Visual Studioで，デバッガーを起動しなさい.



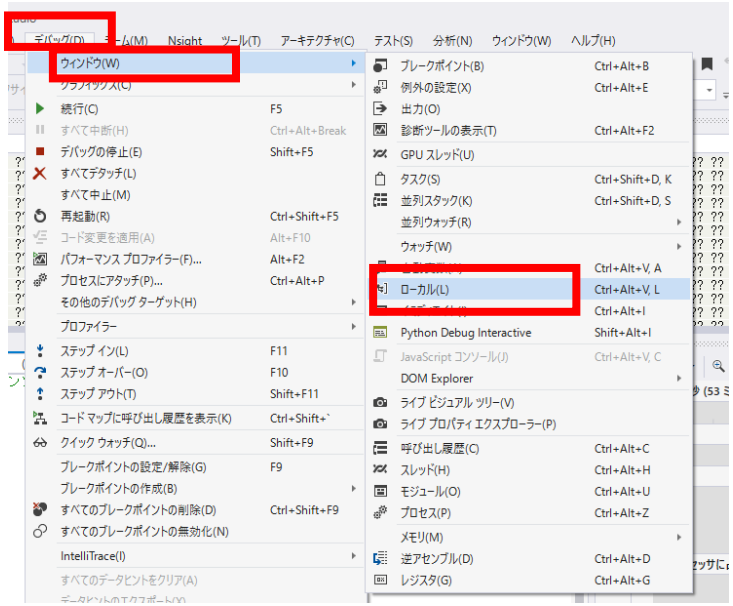
「デバッグ」
→ 「**デバッグ開始**」

- 「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

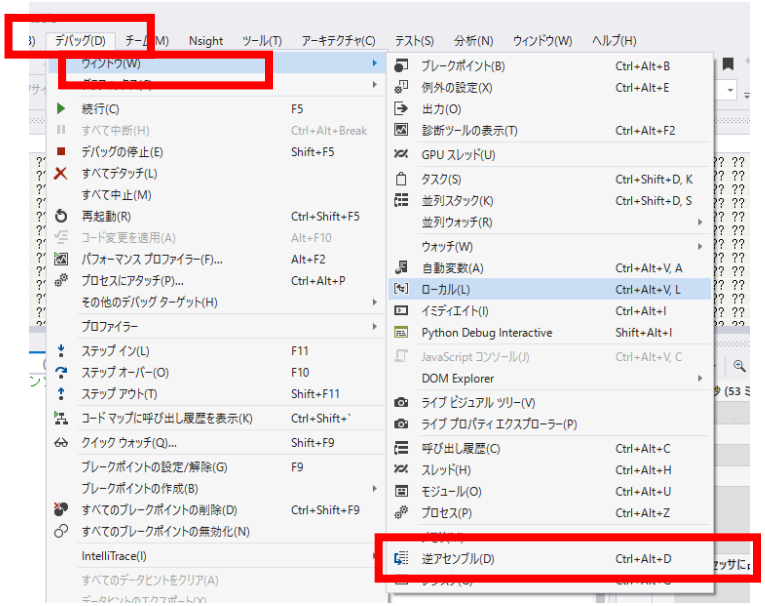
- 「printf」の行で，実行が中断した状態で，変数の値を表示させなさい．手順は次の通り．

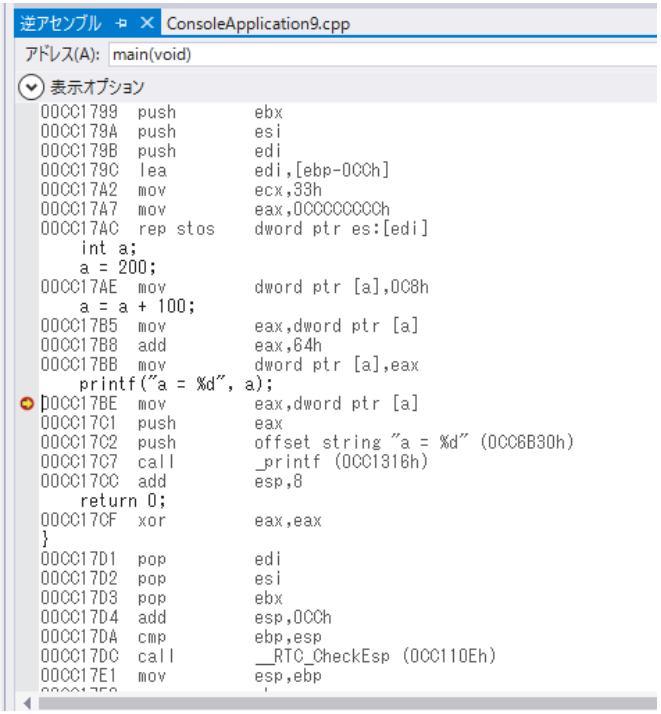


① 「デバッグ」
→ 「ウインドウ」
→ 「**ローカル**」

② 変数名と値の対応表が表示される

- 「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.





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

```
00CC17AE  mov         dword ptr [a],0Ah
    a = a + 100;
00CC17B5  mov         eax,dword ptr [a]
00CC17B8  add         eax,64h
00CC17BB  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;
}
```

```
    a = a - 100;
011C17B5  mov         eax,dword ptr [a]
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;
}
```

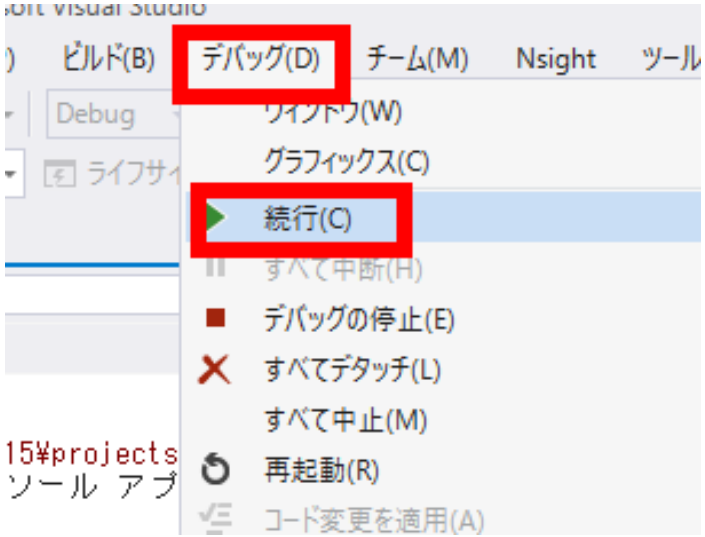
```
    a = a * 40;
008D17B5  imul        eax,dword ptr [a],28h
008D17B9  mov         dword ptr [a],eax
```

- 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 / 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;
}
```

```
    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いなさい．これで，デバッガーが終了する．



「デバッグ」
→ 「**続行**」