

ca-7. データ転送命令と アドレッシングモード

(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



7-1 データ転送命令

7-2 アドレッシングモード

7-3 配列

7-4 C/C++ の配列は、メモリにどのように格納されているか

7-1 データ転送命令

命令セット



種類	命令	意味
データ転送と実効アドレス	MOV	データ転送 ※ ロード, ストア, プッシュ, ポップ
	LEA	実効アドレスのロード
算術演算	ADD	加算
	SUB	減算
	IMUL	乗算
	IDIV	除算
	SAR, SAL	算術シフト
論理演算	AND	論理積
	OR	論理和
	SHR, SHL	論理シフト
比較	CMP	比較
	TEST	AND による比較
ジャンプ (分岐)	JMP	無条件ジャンプ (無条件分岐)
	J??	条件ジャンプ (条件分岐)
サブルーチン	CALL	サブルーチン呼び出し (サブルーチンコール)
	RET	サブルーチンからの復帰

プログラムの例



```
#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;
}
```

```
C:\> C:\WINDOWS\system32\cmd.exe
```

```
10, 30
```

```
続行するには何かキーを押してください . . .
```

C++ 言語とアセンブリ言語



Visual C++ の プログラム

```
int a, b;  
a = 10;  
b = a + 20;  
printf("%d, %d\n", a, b);  
return 0;
```



アセンブリ言語

```
--- e:\documents\visual studio 2013\projects\ConsoleApplication1\ConsoleApplication1.cpp ---  
#include "stdafx.h"  
  
int _tmain(int argc, _TCHAR* argv[])  
{  
00F313C0  push    ebp  
00F313C1  mov     ebp,esp  
00F313C3  sub     esp,0D8h  
00F313C9  push    ebx  
00F313CA  push    esi  
00F313CB  push    edi  
00F313CC  lea    edi,[ebp-0D8h]  
00F313D2  mov     ecx,36h  
00F313D7  mov     eax,0CCCCCCCCh  
00F313DC  rep stos dword ptr es:[edi]  
    int a, b;  
    a = 10;  
00F313DE  mov     dword ptr [a],0Ah  
    b = a + 20;  
00F313E5  mov     eax,dword ptr [a]  
00F313E8  add     eax,14h  
00F313EB  mov     dword ptr [b],eax  
    printf("%d, %d\n", a, b);  
00F313EE  mov     esi,esp  
00F313F0  mov     eax,dword ptr [b]  
00F313F3  push    eax  
00F313F4  mov     ecx,dword ptr [a]  
00F313F7  push    ecx  
00F313F8  push    0F35858h  
00F313FD  call   dword ptr ds:[0F39114h]  
00F31403  add     esp,0Ch  
00F31406  cmp     esi,esp  
00F31408  call   __RTC_CheckEsp (0F31140h)  
    return 0;  
00F3140D  xor     eax,eax  
}  
00F3140F  pop     edi  
00F31410  pop     esi  
00F31411  pop     ebx  
00F31412  add     esp,0D8h  
00F31418  cmp     ebp,esp  
00F3141A  call   __RTC_CheckEsp (0F31140h)  
00F3141F  mov     esp,ebp  
00F31421  pop     ebp  
00F31422  ret
```

命令 オペランド

データ転送命令とは

「データ転送」せよの命令

- ・ **メモリ**からデータを読み出して、**レジスタ**に書き込み
- ・ **レジスタ**からデータを読み出して、**メモリ**に書き込み
- ・ **レジスタ**からデータを読み出して、別の**レジスタ**に書き込みなど

```
--- e:\documents\visual studio 2013\projects\consoleapplicat
#include "stdafx.h"

int _tmain(int argc, _TOCHAR* argv[])
{
00F31300 push    ebp
00F31301 mov     ebp,esp
00F31303 sub     esp,0008h
00F31309 push    ebx
00F3130A push    esi
00F3130B push    edi
00F3130C lea   edi,[ebp-0008h]
00F3130E mov     ecx,30h
00F3130F mov     eax,0CCCCCCCCh
00F31310 mov     dword ptr [eax],eax

    int a, b;
    a = 10;
00F313DE mov     dword ptr [a],0A
    b = a + 20;
00F313E5 mov     eax,dword ptr [a]
00F313E8 add     eax,14h
00F313EB mov     dword ptr [b],eax

    printf("a=%d, b=%d\n", a, b);
00F313EE mov     esi,esp
00F313F0 mov     eax,dword ptr [b]
00F313F3 push   eax
00F313F4 mov     ecx,dword ptr [a]
00F313F7 push   ecx
00F313F8 push   0F358508h
00F313FD call   dword ptr ds:[0F31114h]
00F31403 add     esp,0Ch
00F31406 cmp     esi,esp
00F31408 call   _RTC_CheckEsp (0F31140h)
    return 0;
00F31409 mov     eax,ebx
00F3140F pop     edi
00F31410 pop     esi
00F31411 pop     ebx
00F31412 add     esp,0008h
00F31418 cmp     ebp,esp
00F3141A call   _RTC_CheckEsp (0F31140h)
00F3141F mov     esp,ebp
00F31421 pop     ebp
00F31422 ret
}

```

```
int a, b;
a = 10;
00F313DE mov     dword ptr [a],0A
b = a + 20;
00F313E5 mov     eax,dword ptr [a]
00F313E8 add     eax,14h
00F313EB mov     dword ptr [b],eax
printf("a=%d, b=%d\n", a, b);
```

メモリからレジスタ eax へ

レジスタ eax からメモリへ

データ転送命令

mov 命令 が使用されている

7-2 アドレッシングモード

アドレッシングモードのバリエーション



Visual C++ の
プログラム

アセンブリ言語

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

```
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
```

x のアドレス
y のアドレス
x のアドレス
y のアドレス
z のアドレス

メモリへの書き込み,
メモリからの読み出し
を行う

アドレッシングモード

アドレッシングモードのバリエーション



Visual C++ の
プログラム

アセンブリ言語

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

変数 x に 3 をセット

```
mov     dword ptr [x (0CA9138h)], 3
```

変数 y に 4 をセット

```
mov     dword ptr [y (0CA913Ch)], 4
```

```
mov     eax, dword ptr [x (0CA9138h)]  
add     eax, dword ptr [y (0CA913Ch)]  
mov     dword ptr [z (0CA9140h)], eax
```

値を扱う
アドレッシングモード

アドレッシングモードのバリエーション



Visual C++ の
プログラム

アセンブリ言語

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

```
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 (UCA9140h)],eax
```

結果を
変数 z に書き込む

レジスタを扱う
アドレッシングモード

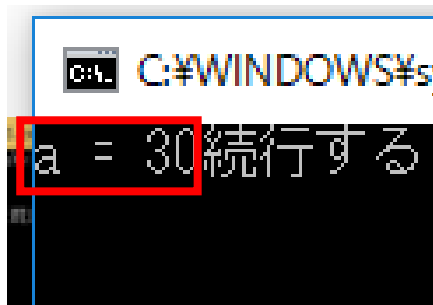
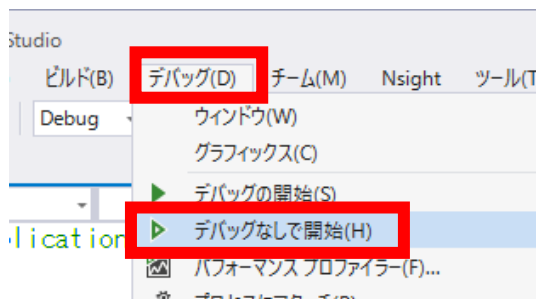
演習



次のプログラムを Visual Studio で実行し、結果を確認しなさい

```
int main()
{
    int a;
    _asm {
        mov eax, 10;
        add eax, 20;
        mov a, eax;
    };
    printf("a = %d", a);
    return 0;
}
```

- ← ①レジスタEAXに値 1 0 をセット
- ← ②レジスタEAXに 2 0 を足しこむ
- ← ③レジスタEAXの値を、変数aのアドレスに書き込む



「a = 30」

が表示されたら成功

7-3 配列

C/C++ での配列と繰り返し



```
≡ int main()
  {
    static int x[5] = { 8, 6, 4, 2, 3 };
    static int y[5] = { 0, 0, 0, 0, 0 };
    int i;
    ≡ for (i = 0; i < 5; i++) {
      y[i] = x[i] * 10;
    }
    return 0;
  }
```

繰り返す処理

i の値は 0, 1, 2, 3, 4
と変化し, 全部済んだら終わる

- 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

```
#include "stdafx.h"
```

```
int main()
```

```
{
```

```
    static int x[5] = { 8, 6, 4, 2, 3 };
```

```
    static int y[5] = { 0, 0, 0, 0, 0 };
```

```
    int i;
```

```
    for (i = 0; i < 5; i++) {
```

```
        y[i] = x[i] * 10;
```

```
    }
```

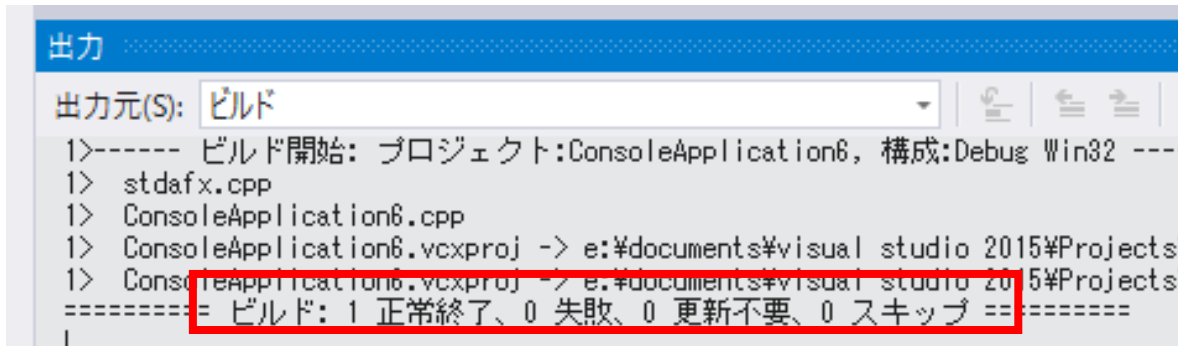
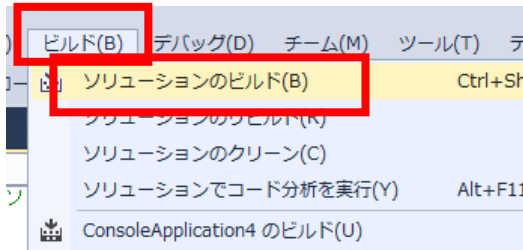
```
    return 0;
```

```
}
```

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「int i;」の行に，ブレークポイントを設定しなさい

```
int main()
{
    static int x[5] = { 8, 6, 4, 2, 3 };
    static int y[5] = { 0, 0, 0, 0, 0 };
    int i;
    for (i = 0; i < 5; i++) {
        y[i] = x[i] * 10;
    }
    return 0;
}
```



```
7
8
9
10
11
12
13
14
15
16
17
18
```

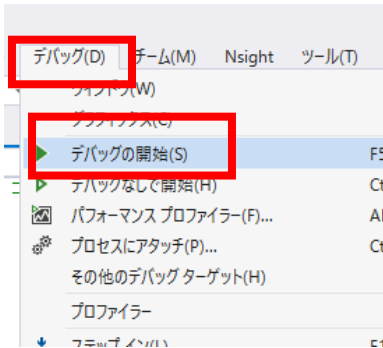
```
int main()
{
    static int x[5] = { 8, 6, 4, 2, 3 };
    static int y[5] = { 0, 0, 0, 0, 0 };
    int i;
    for (i = 0; i < 5; i++) {
        y[i] = x[i] * 10;
    }
    return 0;
}
```

① 「int i;」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。



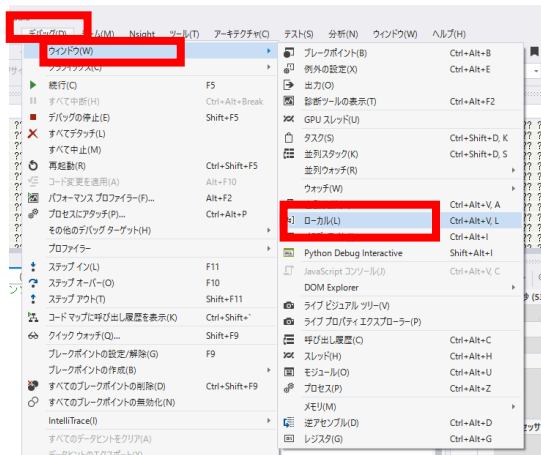
「デバッグ」
→ 「デバッグ開始」

- 「int i;」 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

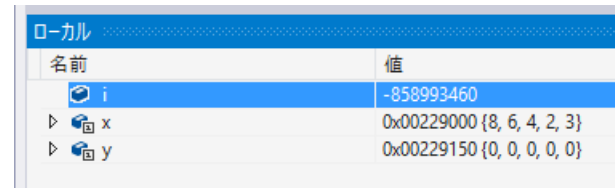


「int i;」 の行で実行が
中断している

- 「**int i;**」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」






名前	値
i	-858993460
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {0, 0, 0, 0, 0}

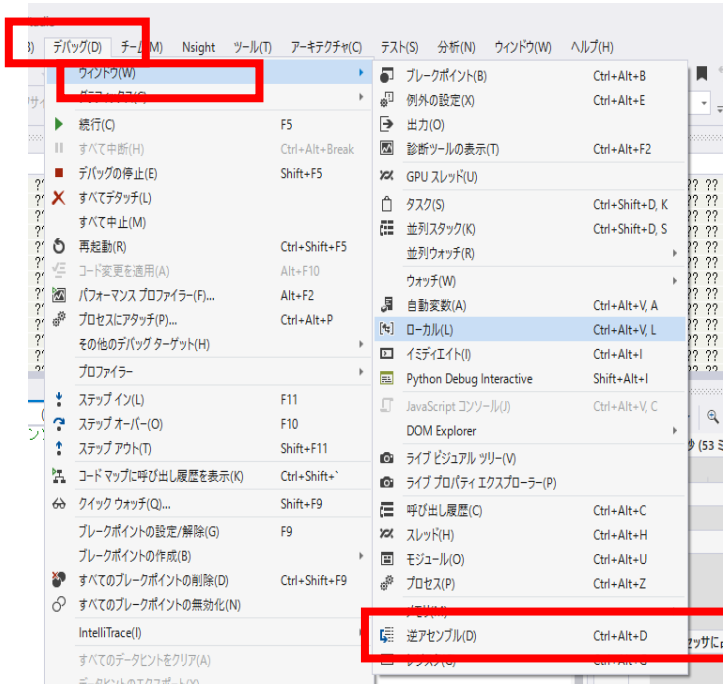
② 変数名と値の対応表が表示される

※ 次ページに拡大図

ローカル

名前	値
 i	-858993460
▷  x	0x00229000 {8, 6, 4, 2, 3}
▷  y	0x00229150 {0, 0, 0, 0, 0}

- 「**int i;**」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。



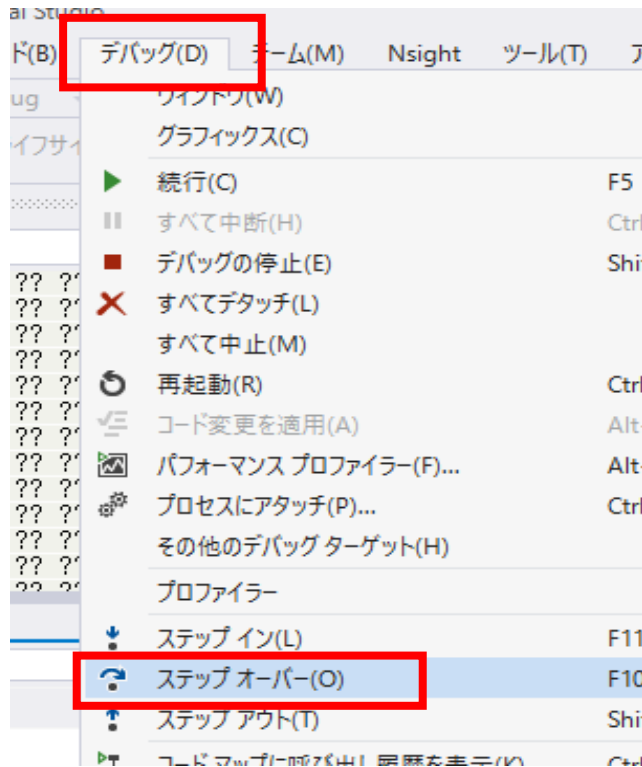
```
static int x[5] = { 8, 6, 4, 2, 3 };
static int y[5] = { 0, 0, 0, 0, 0 };
int i;
for (i = 0; i < 5; i++) {
0022168E mov     dword ptr [i],0
00221675 jmp     main+30h (0221680h)
00221677 mov     eax,dword ptr [i]
0022167A add     eax,1
0022167D mov     dword ptr [i],eax
00221680 cmp     dword ptr [i],5
00221684 jge     main+4Dh (022169Dh)
        y[i] = x[i] * 10;
00221686 mov     eax,dword ptr [i]
00221689 imul   ecx,dword ptr x (0229000h)[eax*4],0Ah
00221691 mov     edx,dword ptr [i]
00221694 mov     dword ptr y (0229150h)[edx*4],ecx
        }
00221698 jmp     main+27h (0221677h)

return 0;
0022169D var     eax,ebx
```

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

- ステップオーバーの操作を 1 回ずつ行いながら、変数 i , x , y の値の変化を確認しなさい。



名前	値
i	-858993460
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {0, 0, 0, 0, 0}

名前	値
i	0
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {80, 0, 0, 0, 0}

名前	値
i	1
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {80, 60, 0, 0, 0}

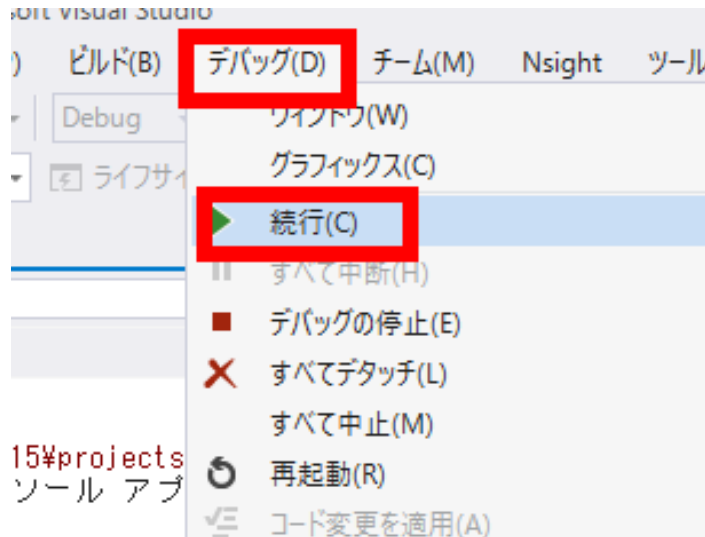
名前	値
i	2
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {80, 60, 40, 0, 0}

名前	値
i	3
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {80, 60, 40, 20, 0}

名前	値
i	4
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {80, 60, 40, 20, 30}

「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**」

7-4 C/C++ の配列は、
メモリにどのように格納され
ているか

数値
(10進数)

16進数

80 →

50	00	00	00
----	----	----	----

60 →

3c	00	00	00
----	----	----	----

40 →

28	00	00	00
----	----	----	----

20 →

14	00	00	00
----	----	----	----

30 →

1e	00	00	00
----	----	----	----

```
メモリ  
アドレス: 0x00229150  
0x00229150 50 00 00 00 3c 00 00 00 28 00 00 00 14 00 00 00 1e 00 00 00  
0x00229175 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

ダンプリストの例

リトルエンディアンで,

4バイトの数値に**コード化**した場合の例

7-4 配列



- C 言語や, C++ 言語の配列は, 同じ型の要素の並び.
- コード化されて, メモリに格納される時,
要素が順にメモリに格納される
各要素のサイズは同じ

- 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

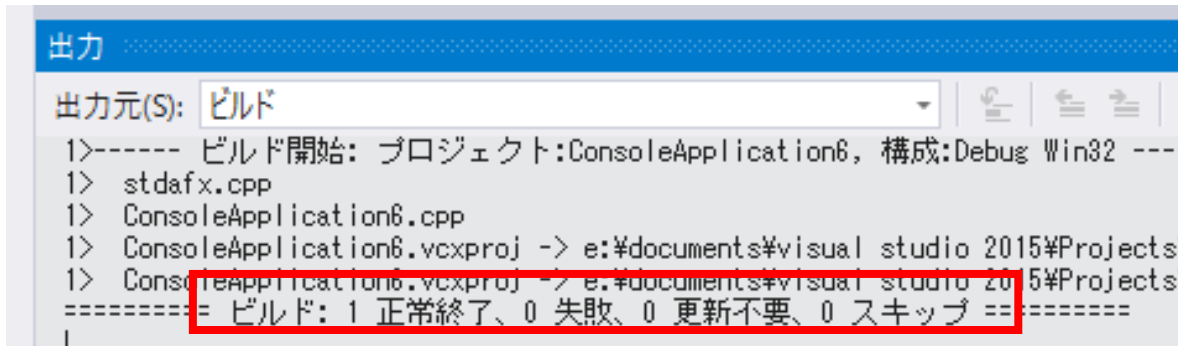
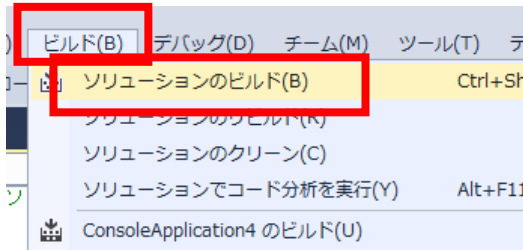
- 先ほどのプログラムをそのまま使う

```
#include "stdafx.h"
```

```
int main()  
{  
    static int x[5] = { 8, 6, 4, 2, 3 };  
    static int y[5] = { 0, 0, 0, 0, 0 };  
    int i;  
    for (i = 0; i < 5; i++) {  
        y[i] = x[i] * 10;  
    }  
    return 0;  
}
```

- 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

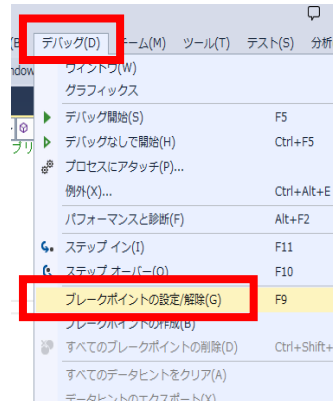
→ 表示されなければ、プログラムのミスを自分で確認し、修正して、ビルドをやり直す



- Visual Studioで「int i;」の行に、ブレークポイントを設定していること

```
int main()
{
    static int x[5] = { 8, 6, 4, 2, 3 };
    static int y[5] = { 0, 0, 0, 0, 0 };
    int i;
    for (i = 0; i < 5; i++) {
        y[i] = x[i] * 10;
    }

    return 0;
}
```



```
int main()
{
    static int x[5] = { 8, 6, 4, 2, 3 };
    static int y[5] = { 0, 0, 0, 0, 0 };
    int i;
    for (i = 0; i < 5; i++) {
        y[i] = x[i] * 10;
    }

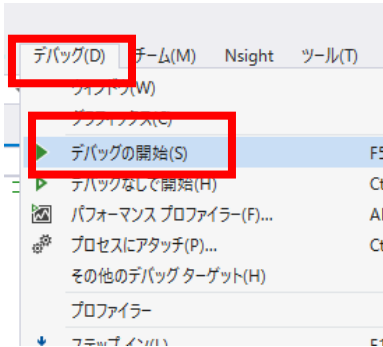
    return 0;
}
```

① 「int i;」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」

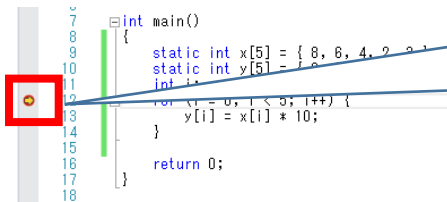
③ ブレークポイント
が設定されるので確認。
赤丸がブレークポイントの印

- Visual Studioで、デバッガーを起動しなさい。



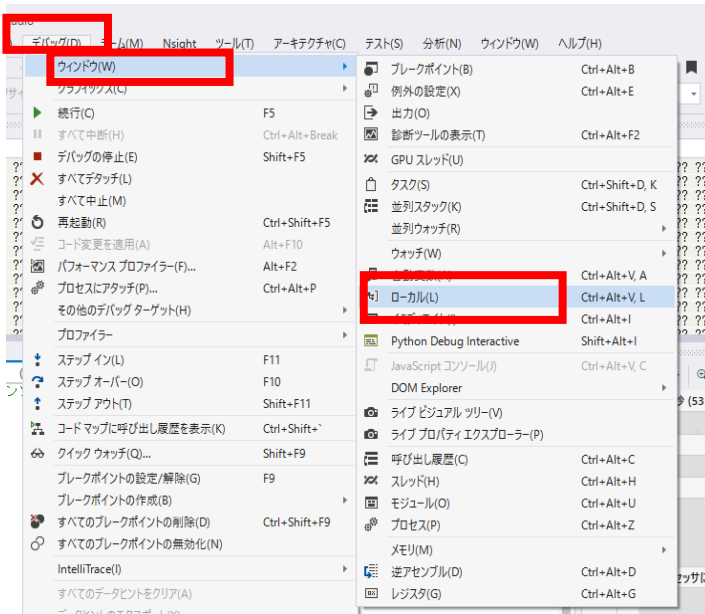
「デバッグ」
→ 「デバッグ開始」

- 「int i;」 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



「int i;」 の行で実行が
中断している

- 「**int i;**」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。



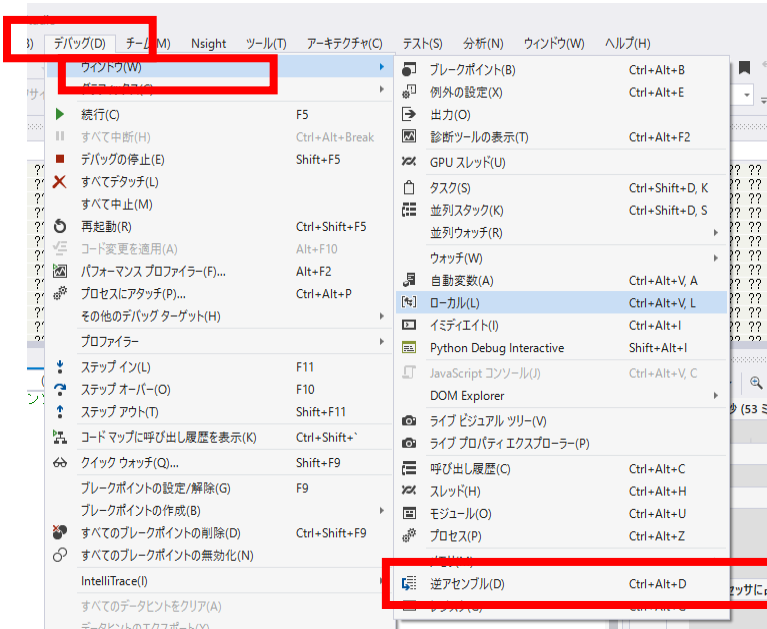
ローカル	
名前	値
i	-858993460
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {0, 0, 0, 0}

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

※ 次ページに拡大図

- 「**int i;**」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。



```
static int x[5] = { 8, 6, 4, 2, 3 };
static int y[5] = { 0, 0, 0, 0, 0 };
int i;
for (i = 0; i < 5; i++) {
0022166E mov     dword ptr [i],0
00221675 jmp     main+30h (0221680h)
00221677 mov     eax,dword ptr [i]
0022167A add     eax,1
0022167D mov     dword ptr [i],eax
00221680 cmp     dword ptr [i],5
00221684 jge     main+4Dh (022169Dh)
        y[i] = x[i] * 10;
00221686 mov     eax,dword ptr [i]
00221689 imul  ecx,dword ptr x (0229000h)[eax*4],0Ah
00221691 mov     edx,dword ptr [i]
00221694 mov     dword ptr y (0229150h)[edx*4],ecx
}
0022169B jmp     main+27h (0221677h)

return 0;
0022169D var     ebx,ebx
```

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

- ローカルウィンドウで配列 y の先頭アドレスを調べなさい

名前	値
i	-858993460
x	0x00229000 {8, 6, 4, 2, 3}
y	0x00229150 {0, 0, 0, 0, 0}

「0x」が付いているのは16進数

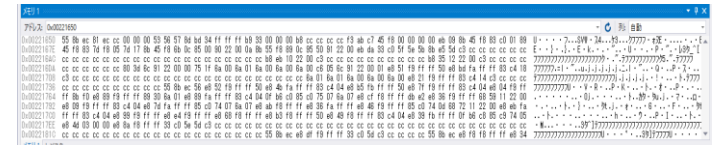
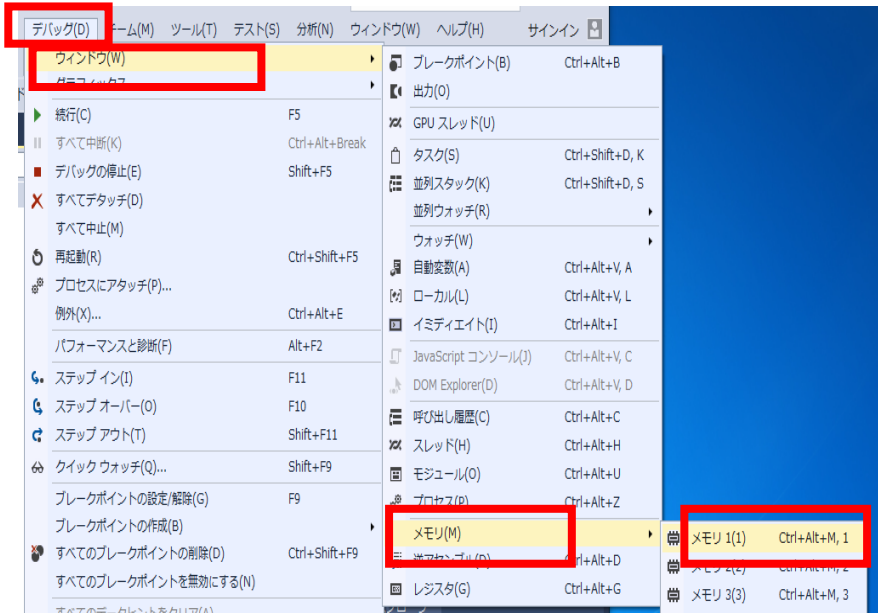
y の先頭アドレスは、起動のたびに
変化する可能性がある。
注意！

- プログラムの中で配列 y の先頭アドレスがあることを確認しなさい

```
static int y[5] = { 0, 0, 0, 0, 0 };
int i;
for (i = 0; i < 5; i++) {
0022168E mov dword ptr [i],0
00221675 jmp main+30h (0221680h)
00221677 mov eax,dword ptr [i]
0022167A add eax,1
0022167D mov dword ptr [i],eax
00221680 cmp dword ptr [i],5
00221684 jge main+4Dh (022169Dh)
        y[i] = x[i] * 10;
00221686 mov eax,dword ptr [i]
00221689 imul ecx,dword ptr x (0229000h)[eax*4],0Ah
00221691 mov edx,dword ptr [i]
00221694 mov dword ptr y (0229150h)[edx*4],ecx
}
```

「h」が付いているのは
16進数

・ダンプリストを表示させなさい。



② メモリの中身がダンプリスト形式で表示される

① 「デバッグ」
→ 「ウィンドウ」
→ 「メモリ」 → 「メモリ 1」

- 配列 y の先頭アドレスを,
メモリウインドウの「アドレス」のところに
書き写して, Enter キーを押す

「0x00229150」のように
頭に 0x を付ける

ローカル	名前	値
	i	-858993460
▷	x	0x00229000 {8, 6, 4,
▷	y	0x00229150 {0, 0, 0,



メモリ 1	
アドレス	値
0x00221650	55 8b ec 81 ec cc 00
0x0022167E	45 f8 83 7d f8 05 7d
0x002216AC	cc cc cc cc cc cc cc
0x002216DA	cc cc cc cc cc cc 80

① 配列 y の先頭ア
ドレス

- 10進数にすると, 80, 60, 40, 20, 30 である

16進数 10進数

メモリ 1	アドレス: 0x00229150
0x00229150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
メモリ 1	アドレス: 0x00229150
0x00229150	50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
メモリ 1	アドレス: 0x00229150
0x00229150	50 00 00 00 00 3c 00 00 00 00 00 00 00 00 00 00 00 00 00 00
メモリ 1	アドレス: 0x00229150
0x00229150	50 00 00 00 00 3c 00 00 00 28 00 00 00 00 00 00 00 00 00 00
メモリ 1	アドレス: 0x00229150
0x00229150	50 00 00 00 00 3c 00 00 00 28 00 00 00 14 00 00 00 00 00 00
メモリ 1	アドレス: 0x00229150
0x00229150	50 00 00 00 00 3c 00 00 00 28 00 00 00 14 00 00 00 1e 00 00 00

50

80

3c

60

28

40

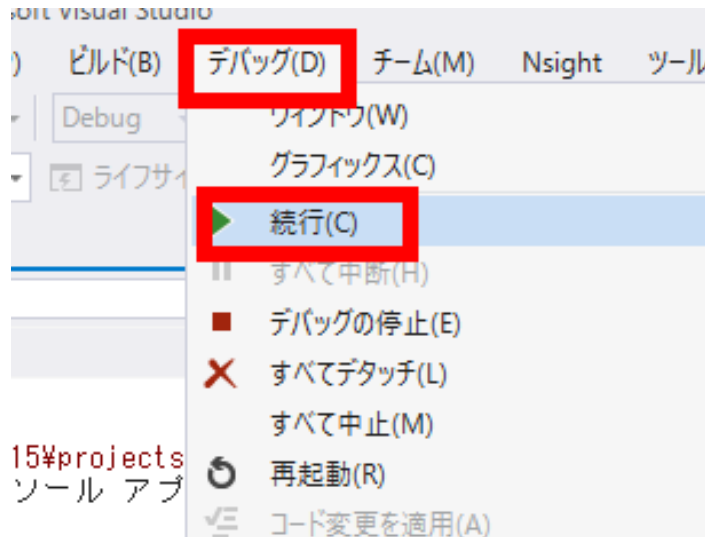
14

20

1e

30

- 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」

配列の要素に値を格納するプログラム



レジスタ EAX に 値
4 をセットせよ

EAX

4

レジスタ EAX の 値
を 3倍 した値を
ECX にセットせよ

ECX

12

レジスタ ECX の 値 と
0EC8130h を足したアドレス
に 4 を書き込め！

アセンブリ言語

```
00EC139E  mov     eax,4
00EC13A3  imul   ecx,eax,0
00EC13A6  mov     dword ptr [ecx+0EC8130h],1

00EC13B0  mov     eax,4
00EC13B5  shl    eax,0
00EC13B8  mov     dword ptr [eax+0EC8130h],2

00EC13C2  mov     eax,4
00EC13C7  shl    eax,1
00EC13C9  mov     dword ptr [eax+0EC8130h],3

00EC13D3  mov     eax,4
00EC13D8  imul   ecx,eax,3
00EC13DB  mov     dword ptr [ecx+0EC8130h],4
```

```
0x00EC8130  01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00
0x00EC8140  00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
```

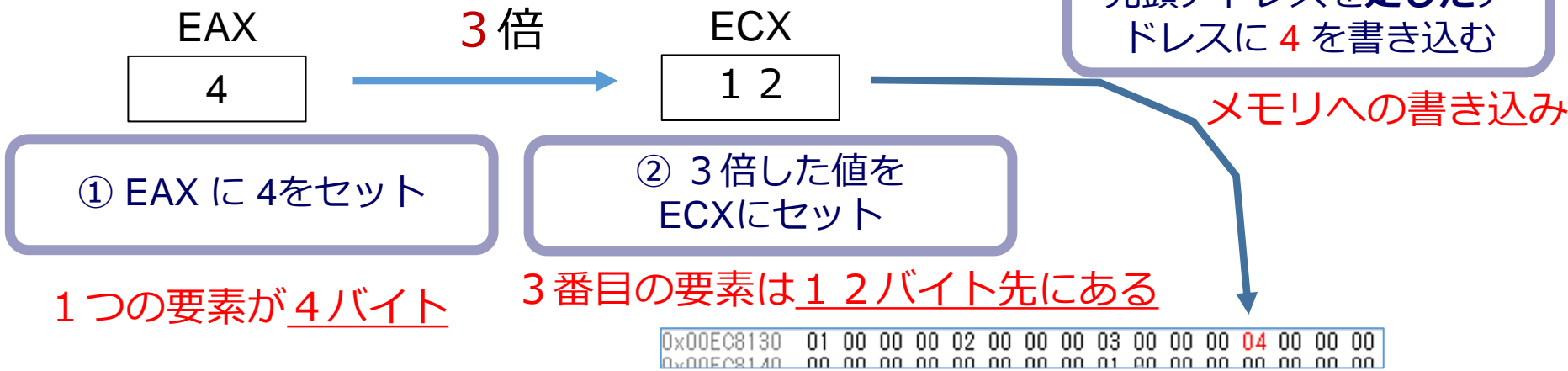
0EC8130h は
配列 a の先頭アドレス

配列の要素に値を格納するプログラム



Visual C++ では : $a[3] = 4$

```
mov     eax,4
imul   ecx,eax,3
mov     dword ptr [ecx+0EC8130h],4
```



種々のアドレッシングモード



<code>b = a + 200;</code>	<code>mov eax,dword ptr ds:[00348130h] add eax,0c8h</code>	0c8h
<code>b = a + x;</code>	<code>mov eax,dword ptr ds:[00258130h] add eax,dword ptr ds:[258138h]</code>	変数 x のアドレス
<code>b = a + y[5];</code>	<code>mov eax,4 imul ecx,eax,5 mov edx,dword ptr ds:[1198130h] add edx,dword ptr ds:[ecx+1198138h]</code>	y[5]があるアドレス
<code>b = a + y[i];</code>	<code>mov eax,dword ptr ds:[00048160h] mov ecx,dword ptr ds:[48130h] add ecx,dword ptr ds:[eax*4+48138h]</code>	y[i]があるアドレス