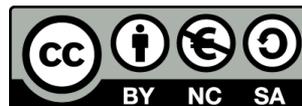


ca-9. 数の扱い

(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



9-1 浮動小数点数の扱い

9-2 2の補数

9-3 算術シフト

9-4 論理シフト

9-1 浮動小数点数の扱い

```
// ConsoleApplication5.cpp : コンソールアプリケーションのソースファイル。
//
#include "stdafx.h"
#include <stdlib.h>

int _tmain(int argc, _TCHAR* argv[])
{
    int x, y;
    x = 20;
    y = x * x;
    printf("%d\n", y);
    system("pause");

    return 0;
}
```

対応

```
int x, y;
x = 20;
00EF13DE mov     dword ptr [x],14h
y = x * x;
00EF13E5 mov     eax,dword ptr [x]
00EF13E8 imul   eax,dword ptr [x]
00EF13EC mov     dword ptr [y],eax
printf("%d\n", y);
00EF13EF mov     esi,esp
00EF13F1 mov     eax,dword ptr [y]
00EF13F4 push   eax
00EF13F5 push   0EF5858h
00EF13FA call   dword ptr ds:[0EF9118h]
00EF1400 add     esp,8
00EF1403 cmp     esi,esp
00EF1405 call   __RTC_CheckEsp (0EF1145h)
system("pause");
00EF140A mov     esi,esp
00EF140C push   0EF585Ch
00EF1411 call   dword ptr ds:[0EF9110h]
00EF1417 add     esp,4
00EF141A cmp     esi,esp
00EF141C call   __RTC_CheckEsp (0EF1145h)

return 0;
00EF1421 xor     eax,eax
}
```

```
// ConsoleApplication5.cpp : コンソール
//
#include "stdafx.h"
#include <stdlib.h>

int _tmain(int argc, _TCHAR* argv[])
{
    double x, y;
    x = 20;
    y = x * x;
    printf("%f\n", y);
    system("pause");

    return 0;
}
```

対応

int を double に変更

```
double x, y;
x = 20;
000A13DE movsd xmm0,mmword ptr ds:[0A5868h]
000A13E6 movsd mmword ptr [x],xmm0
y = x * x;
000A13EB movsd xmm0,mmword ptr [x]
000A13F0 mulsd xmm0,mmword ptr [x]
000A13F5 movsd mmword ptr [y],xmm0
printf("%f\n", y);
000A13FA mov esi,esp
000A13FC sub esp,8

000A1420 push 0A3B5Ch
000A1425 call dword ptr ds:[0A9110h]
000A142B add esp,4
000A142E cmp esi,esp
000A1430 call __RTC_CheckEsp (0A1145h)

return 0;
000A1435 xor eax,eax
```

使用されるレジスタが
xmm0 になるなど、
いくつかの変化

9-4 2の補数

2の補数



- 2の補数は、負の整数も扱いたいときに便利
- 2の補数では、最上位ビットが符号ビット

0 → 正の整数または0

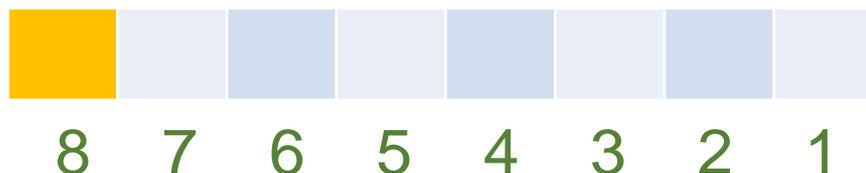
1 → 負の整数

10進数の2	0	0	0	0	0	1	0
10進数の1	0	0	0	0	0	0	1
10進数の0	0	0	0	0	0	0	0
10進数の-1	1	1	1	1	1	1	1
10進数の-2	1	1	1	1	1	1	0

2の補数での符号ビット



8ビットの整数データの場合



2の補数では、**最上位ビット**は符号ビット

0 → 正の数, 0

1 → 負の数

正なのか負なのかの区別を使う

① ウェブブラウザを起動する

② python tutor を使いたいのので、次の URL を開く

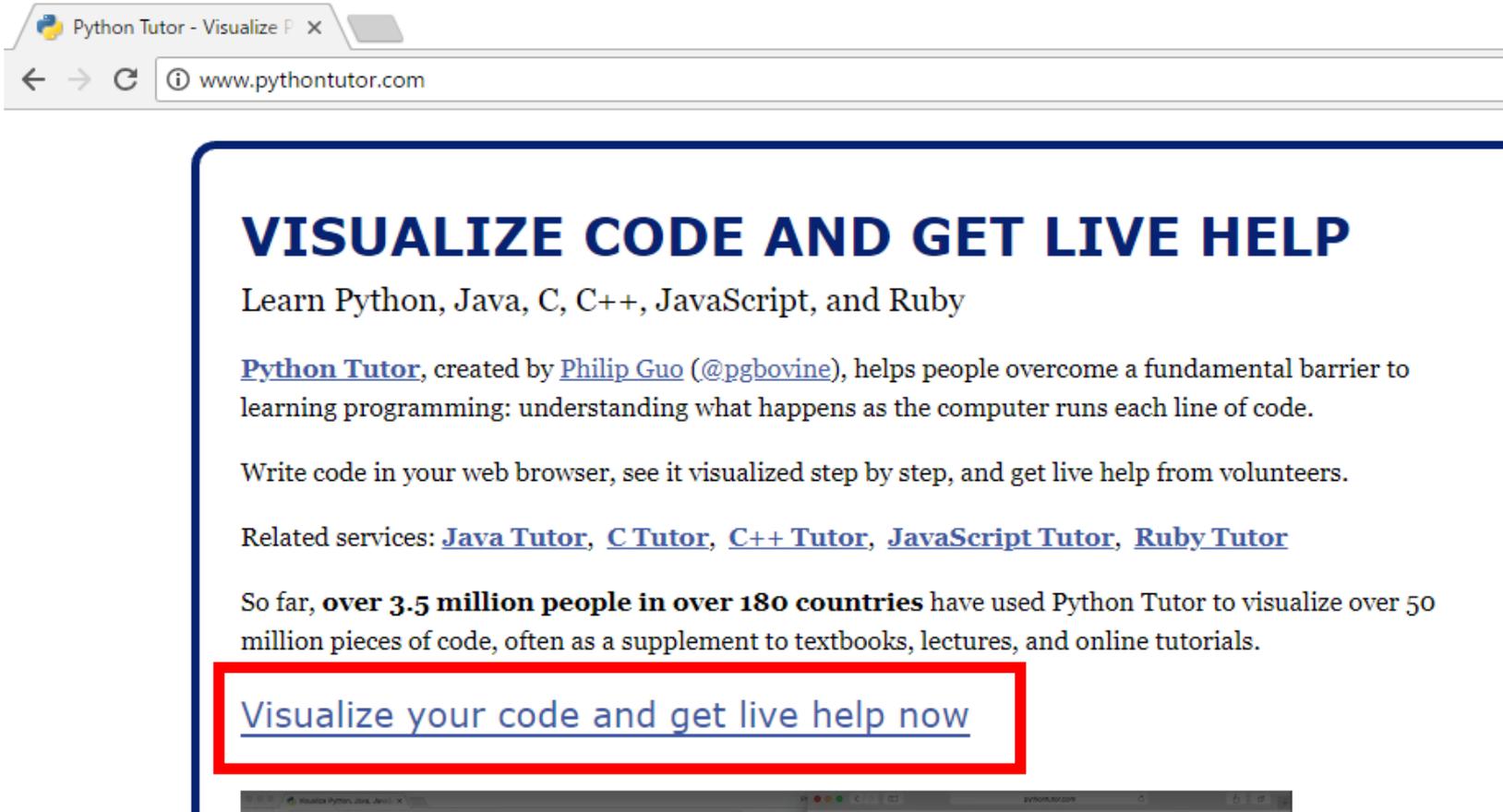
<http://www.pythontutor.com/>

※ Internet Explorer でうまく動かない場合があります

→ うまく動かないときは Google Chrome を試してください

※ 日本語モードはないので、英語で使う

③ 「Visualize your code and live help now」 をク



The image shows a browser window with the URL www.pythontutor.com. The page content includes the heading "VISUALIZE CODE AND GET LIVE HELP", a sub-heading "Learn Python, Java, C, C++, JavaScript, and Ruby", and a paragraph describing the service. A red box highlights the link "Visualize your code and get live help now".

Python Tutor - Visualize P x

← → ↻ ⓘ www.pythontutor.com

VISUALIZE CODE AND GET LIVE HELP

Learn Python, Java, C, C++, JavaScript, and Ruby

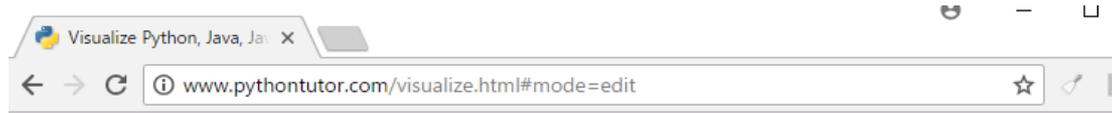
[Python Tutor](#), created by [Philip Guo \(@pgbovine\)](#), helps people overcome a fundamental barrier to learning programming: understanding what happens as the computer runs each line of code.

Write code in your web browser, see it visualized step by step, and get live help from volunteers.

Related services: [Java Tutor](#), [C Tutor](#), [C++ Tutor](#), [JavaScript Tutor](#), [Ruby Tutor](#)

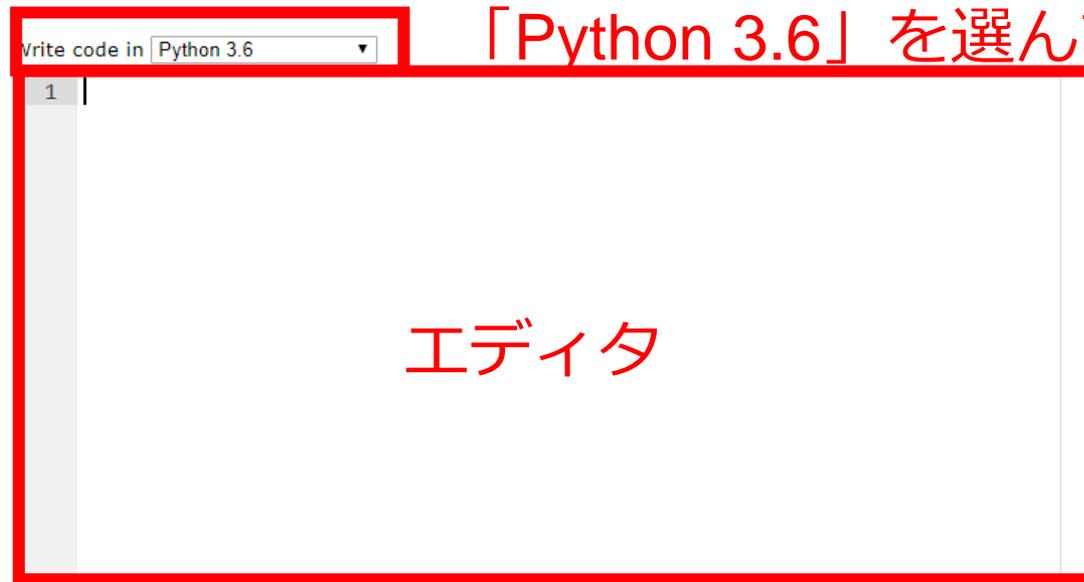
So far, **over 3.5 million people in over 180 countries** have used Python Tutor to visualize over 50 million pieces of code, often as a supplement to textbooks, lectures, and online tutorials.

[Visualize your code and get live help now](#)



言語を選べる.

「Python 3.6」を選んでおく



Help improve this tool by completing a [short user survey](#)
Keep this tool free by making a [small donation](#) (PayPal, Patreon, credit/debit card)

Visualize Execution

Live Programming Mode

実行のためのボタン

Advanced instructions: [setting breakpoints](#) | [hiding variables](#) | [live programming](#)

① 次のように書きなさい

Write code in

```
1 print( format( -2 & 0xff, 'b' ) )  
2 print( format( -1 & 0xff, 'b' ) )  
3 print( format( 0 & 0xff, 'b' ) )  
4 print( format( 1 & 0xff, 'b' ) )  
5 print( format( 2 & 0xff, 'b' ) )  
6 print( format( 3 & 0xff, 'b' ) )  
7
```

```
print( format( -2 & 0xff, 'b' ) )  
print( format( -1 & 0xff, 'b' ) )  
print( format( 0 & 0xff, 'b' ) )  
print( format( 1 & 0xff, 'b' ) )  
print( format( 2 & 0xff, 'b' ) )  
print( format( 3 & 0xff, 'b' ) )
```

② 「Visualize Execution」 をクリック

Write code in Python 3.6

```
1 print( format( -2 & 0xff, 'b' ) )
2 print( format( -1 & 0xff, 'b' ) )
3 print( format( 0 & 0xff, 'b' ) )
4 print( format( 1 & 0xff, 'b' ) )
5 print( format( 2 & 0xff, 'b' ) )
6 print( format( 3 & 0xff, 'b' ) )
7
8
9
10
```

Help improve this tool by completing a [short user survey](#).
Keep this tool free by making a [small donation](#) (PayPal, Patreon,

Visualize Execution

Live Programming Mode

③ 「Last」 ボタンをクリック

Python 3.6

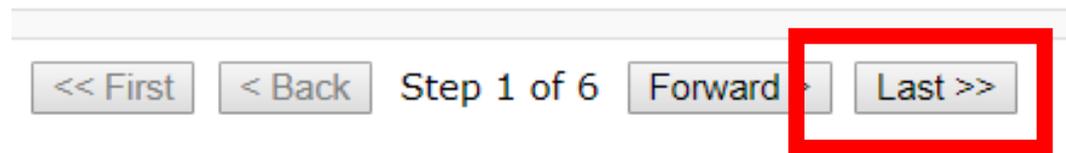
```
→ 1 print( format( -2 & 0xff, 'b' ) )  
2 print( format( -1 & 0xff, 'b' ) )  
3 print( format( 0 & 0xff, 'b' ) )  
4 print( format( 1 & 0xff, 'b' ) )  
5 print( format( 2 & 0xff, 'b' ) )  
6 print( format( 3 & 0xff, 'b' ) )
```

[Edit this code](#)

as just executed

o execute

ode to set a breakpoint; use the Back and Forward buttons to jump there.



④ 結果を確認

Print output (drag lower right corner to resize)

```
11111110  -2
11111111  -1
0          0
1          1
10         2
11         3
```

10進数

9-3 算術シフト

算術シフト



整数を2倍したい, 4倍したい, 1/2倍したい,
1/4倍したい

数 10 (十進数) 00001010 (二進数)



20 (十進数) 00010100 (二進数)

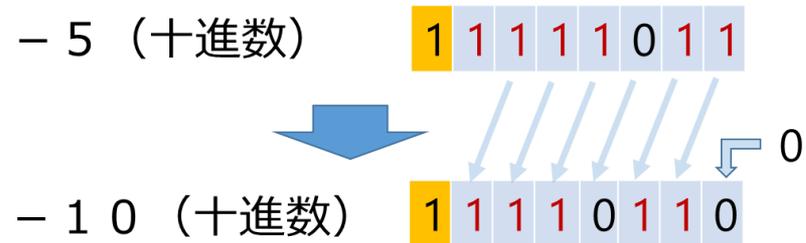
ここでは,
最上位ビットは
符号ビット
(+か-かの記録)

整数の2倍 = 最上位ビット以外のビットを左に1つ
ずらす (左シフト)

算術シフトの左と右

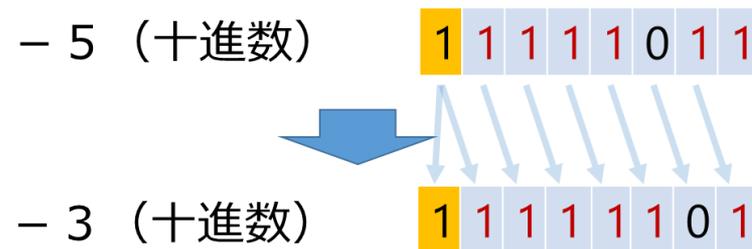
• 算術左シフト

- 最上位ビットは変化しない
- 最上位ビット以外を左にシフト。できた空きには0を詰める



• 算術右シフト

- 最上位ビットは変化しない
- 最上位ビット以外を右にシフト。できた空きには最上位ビットの値を詰める



算術左シフトによる2倍



10 (十進数) 00001010 (二進数)



20 (十進数) 00010100 (二進数)

-5 (十進数) 11111011 (二進数) ※ 2の補数



-10 (十進数) 11110110 (二進数) ※ 2の補数

最上位ビット以外を左に1つずらす (左シフト) .

最下位ビットに0を入れる

算術左シフトによる4倍



10 (十進数) 00001010 (二進数)



40 (十進数) 00101000 (二進数)

-5 (十進数) 11111011 (二進数) ※ 2の補数



-20 (十進数) 11101100 (二進数) ※ 2の補数

最上位ビット以外を左に2つずらす (左シフト) .

最下位2つに, 「00」を入れる

算術右シフトによる 1 / 2 倍

10 (十進数) 0 0 0 0 1 0 1 0 (二進数)



5 (十進数) 0 0 0 0 0 1 0 1 (二進数)

-5 (十進数) 1 1 1 1 1 0 1 1 (二進数) ※ 2の補数



-3 (十進数) 1 1 1 1 1 1 0 1 (二進数) ※ 2の補数

最上位ビット以外を右に1つずらす (右シフト) .

最上位から2番目のビットには, 最上位ビットを入れる

- 整数を2倍, 4倍, \dots , $1/2$ 倍, $1/4$ 倍 \dots , したいときに使う
- **2の補数**では, 符号ビットがある. 符号ビットの部分はそのまま残すのが, **算術シフト**

算術右シフト sar の例



算術右シフトは 1/2倍, 1/4倍, . . .

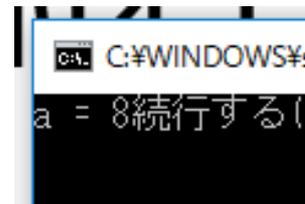
```
int main()
{
    int a;
    _asm {
        mov a, 32;
        sar a, 2;
    };
    printf("a = %d", a);
    return 0;
}
```

アセンブリ言語のプログラム

```
mov a, 32;
sar a, 2;
```

a に 32 をセット
a を 2ビット算術右シフト

実行結果の例



算術左シフト sal の例



算術左シフトは 2倍, 4倍, . . .

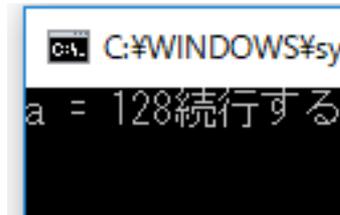
```
int main()
{
    int a;
    _asm {
        mov a, 32;
        sal a, 2;
    };
    printf("a = %d", a);
    return 0;
}
```

アセンブリ言語のプログラム

```
mov a, 32;
sal a, 2;
```

a に 32 をセット
a を 2ビット算術左シフト

実行結果の例



- 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

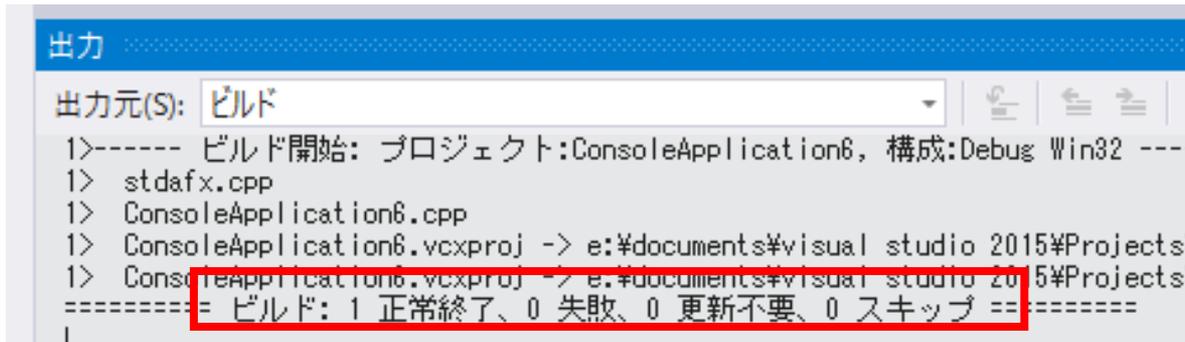
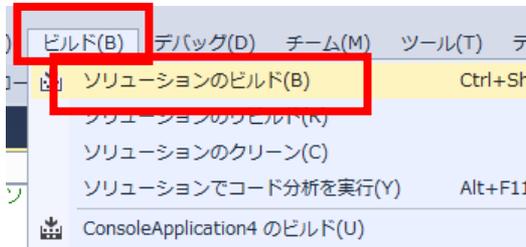
```
int main()  
{
```

```
    int a;  
    a = 300;  
    a = a / 2;  
    printf("a = %d", a);  
    return 0;          追加
```

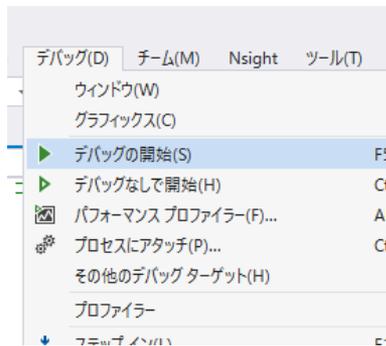
```
}  
_
```

- 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、ブレークポイントを設定しなさい
- 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」

- **ブレークポイント**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

- 逆アセンブルで「 $a = a / 2$ 」のところを確認しなさい。

idiv ではなく sar になっている (算術シフト)

```
int main()
{
    int a;
    a = 300;
    a = a / 2;
    printf("a = %d", a);
    return 0;
}
```

```
000417AE  mov     dword ptr [a],12Ch
          a = a / 2;
000417B5  mov     eax,dword ptr [a]
000417B8  cdq
000417B9  sub     eax,edx
000417BB  sar     eax,1
000417BD  mov     dword ptr [a],eax
          printf("a = %d", a);
```

1ビットのシフト

- 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「 $a = a / 4$ 」のところを確認しない。

```
int main()
{
    int a;
    a = 300;
    a = a / 4;
    printf("a = %d", a);
    return 0;
}
```

```
a = a / 4;
417B5  mov     eax,dword ptr [a]
417B8  cdq
417B9  and     edx,3
417BC  add     eax,edx
417BE  sar    eax,2
417C1  mov     dword ptr [a],eax
printf("a = %d", a):
```

2ビットのシフト

- 次のように書き替えて，今度は，変数 a の値を確認しなさい

```
int main()
{
    int a;
    _asm {
        mov a, 30;
        sar a, 3;
    }
    printf("a = %d", a);
    return 0;
}
```

ローカル	
名前	値
 a	3

**3ビットのシフト
(8で割って，あまりを切り捨て)**

- 次のように書き替えて、変数 a の値を確認しなさい

```
int main()
{
    int a;
    _asm {
        mov a, 30;
        sal a, 3;
    }
    printf("a = %d", a);
    return 0;
}
```

ローカル	
名前	値
a	240

**3ビットのシフト
(8倍)**

Pentium 系列プロセッサの 算術演算命令の例



種類	命令	意味
算術演算	add	加算
	sub	減算
	imul	乗算
	idiv	除算
	sar, sal	算術シフト

9-4 論理シフト

論理シフトの例



2ビット目が1か0かを調べたい

他のビットは無視したい

論理右シフト



0

1 0 0 0 1 0 1 0



0 1 0 0 0 1 0 1

0 0 0 0 0 0 0 1

論理積



0 0 0 0 0 0 0 1

※ もし、元の2ビット目が0だったときは、
この値は「0 0 0 0 0 0 0 0」