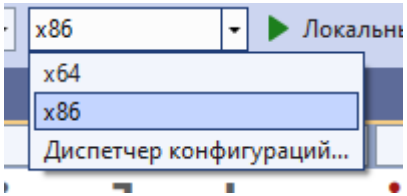
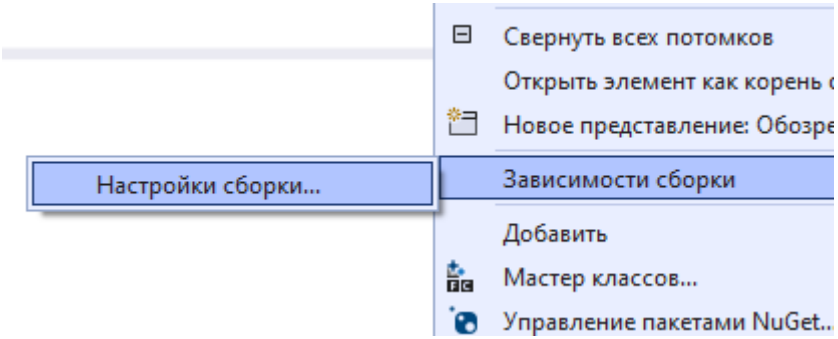
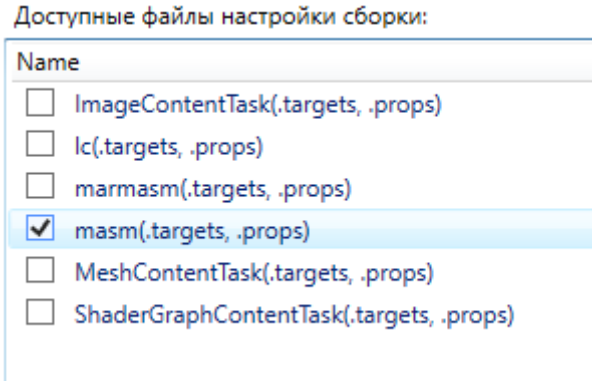
Логические команды

Запуск кода

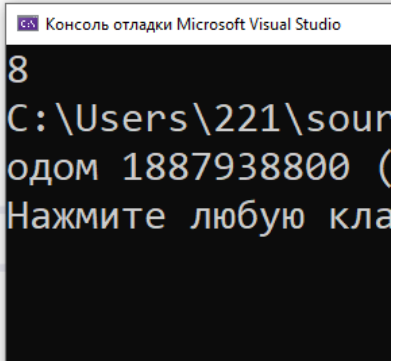

x86
Локальны
x64
x86
Диспетчер конфигураций...


Свернуть всех потомков
Открыть элемент как корень
Новое представление: Обозре
Настройки сборки...
Зависимости сборки
Добавить
Мастер классов...
Управление пакетами NuGet...


Доступные файлы настройки сборки:
Name
ImageContentTask(.targets, .props)
Ic(.targets, .props)
marmasm(.targets, .props)
masm(.targets, .props)
MeshContentTask(.targets, .props)
ShaderGraphContentTask(.targets, .props)

Запуск кода

```
#include <iostream>
using namespace std;
int main()
{
    int a = 5, b = 3, sum;
    __asm {
        mov eax, a;
        mov ebx, b;
        add eax, ebx;
        mov sum, eax;
    }
    cout << sum;
}
```



AND

Инструкция AND выполняет поразрядное логическое умножение

```
add destination, source
//destination = destination and source
```

AND

Операция возвращает 1, если соответствующие разряды обоих операндов равны 1

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

```
10110101
10001101
------------
10000101
```

```
1010101
0101111
------------
0000101
```

AND

Пример:

```
__asm {
   mov eax,12
   and eax, 6 //EAX and 6 = 4
}
```

OR

Инструкция OR выполняет поразрядное логическое сложение

```
or destination, source
//destination = destination or source
```

OR

Операция возвращает 1, если хотя бы один из соответствующих разрядов обоих операндов равен 1

A	B	A and B
0	0	0
0	1	1
1	0	1
1	1	1

```
10110101
10001101
----------
10111101
```

```
1010101
0101111
----------
1111111
```

OR

Пример:

```
__asm {
   mov eax,12
   or eax, 6 //EAX or 6 = 14
}
```

XOR

Инструкция XOR выполняет поразрядную операцию исключающего ИЛИ

```
xor destination, source
//destination = destination xor source
```

XOR

Операция возвращает 1, если соответствующие разряды обоих операндов не равны друг другу

A	B	A and B
0	0	0
0	1	1
1	0	1
1	1	1

```
10110101
10001101
------------
00111000
```

```
1010101
0101111
------------
1111010
```

XOR

Пример:

```
__asm {
   mov eax,12
   xor eax, eax //EAX xor EAX = 0
}
```

NOT

Инструкция NOT выполняет поразрядное отрицание и принимает один параметр

```
not destination
//destination = not destination
```

NOT

Если разряд равен 1, то он меняется на 0, и наоборот

A	not A
0	1
1	0

```
10110101
------------
01001010
```

```
1010101
------------
0101010
```

NOT

Пример:

```
__asm {
   mov eax,12
   not eax //not EAX= 4294967283
}
```

NEG

Кроме обычной инверсии в ассемблере есть арифметическое отрицацие, которое выполняет инструкция **NEG**.

```
neg destination
//destination = - destination
//-destination = destination
```

NEG

Значение операнда будет умножаться на -1. Таким образом, мы сможем получить из положительного числа отрицательное, а из отрицательного - положительное.

NEG

Пример:

```
__asm {
   mov eax,12
   neg eax //not EAX = -12
   neg eax //not EAX = 12
}
```

Команды сдвига

Сдвиг влево

Для сдвига влево применяется инструкция **shl**, которая имеет следующий синтаксис:

```
shl dest, count
```

Сдвиг влево

Пример:

```
__asm {
   mov eax, 2 //10 = 2
   shl eax, 1 //10 << 1 = 100 = 4
}
```

Сдвиг вправо

Для сдвига вправо предназначена инструкция shr, которая работает во многом аналогично инструкции shl:

```
shr dest, count
```

Сдвиг вправо

Пример:

```
__asm {
   mov eax, 2 //10 = 2
   shr eax, 1 //10 >> 1 = 1 = 1
}
```

Вращение

Операции вращения влево и вправо ведут себя так же, как операции сдвига влево и вправо, за тем исключением, что бит, смещенный с одного конца, помещается обратно в другой конец. Для вращения предусмотрены две инструкции: **rol** (поворот влево) и **ror** (поворот вправо).

ROL

Синтаксис:

```
rol dest, count
```

Пример:

```
mov al, 131 // в AL число 131 или 10000011
rol al, 2 // вращаем число в AL на 2 разряда влево
// 10000011 << 2 = 00001110 = 14
```

ROL

Синтаксис:

```
ror dest, count
```

Пример:

```
mov al, 131 // в AL число 131 или 10000011
ror al, 2 // вращаем число в AL на 2 разряда вправо
//10000011 >> 2 = 11100000 = 224
```