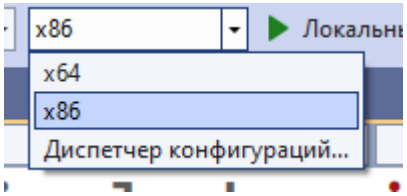
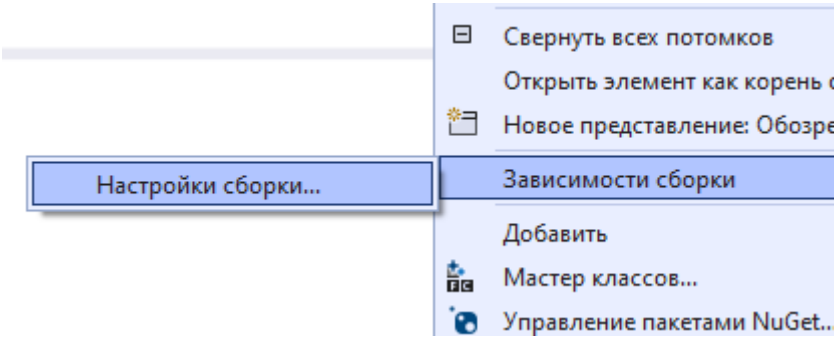
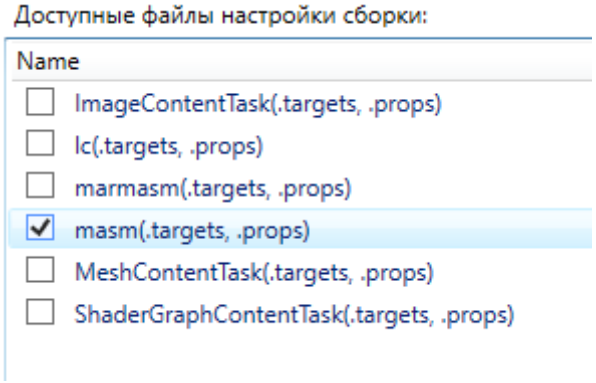
Цепочечные команды

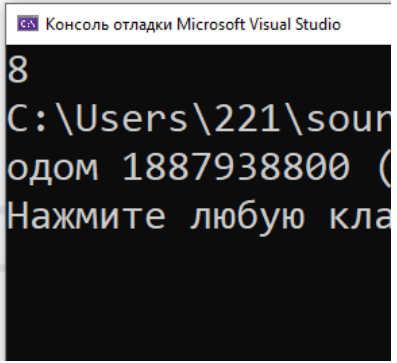
http://i-assembler.ru/25/Lesson/Lesson11/Les_11.htm#cmps


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)
lc(.targets, .props)
marmasm(.targets, .props)
masm(.targets, .props)
MeshContentTask(.targets, .props)
ShaderGraphContentTask(.targets, .props)

```
#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;
}
```



Цепочечные команды

Специальные инструкции, предназначенные для эффективной обработки блоков данных, таких как строки и массивы. Они часто используются в ассемблерных вставках на C++ для оптимизации критических участков кода.

Размеры данных

Размер (Bit)	Размер (Byte)	Название	Пример регистра	Пример команд
8	1	Byte	AL,BL,CL,DL	MOVSB, STOSB
16	2	Word	AX,BX,CX,DX	MOVSW, STOSW
32	4	Dword	EAX,EBX,ECX,EDX	MOVSD, STOSD
64	8	Qword	RAX,RBX..	MOVSQ
80	10	Tbyte	ST(0)	FSTP

Типы данных в C++

Размер (Bit)	Размер (Byte)	Тип C++
8	1	char
16	2	short
32	4	int
64	8	long long

Копирование данных

Команда MOVS используется для копирования строки.

Синтаксис: MOVS op1,op2

Ограничения: Операнды должны иметь один и тот же размер

```
char str1[] = "Hello";
char str2[] = "world";
__asm{
    mov ecx, 5;//rep повторится 5 раз
    //5 раз копирует символ
    //из str1 в str2
    rep movs str2, str1;
    //копирует str2 в eax
    lea eax, str2;
    //добавляет eax в стек
    push eax;
    //вызов printf для вывода str2
    call printf;
    //удаление eax из стека
    pop eax;
}
```

Копирование данных

Команда MOVSB используется для копирования строк байтов

Синтаксис: MOVSB

Ограничения: Нет

Копирование данных

Команда MOVSW используется для копирования строк слов

Синтаксис: MOVSW

Ограничения: Нет

Сравнение цепочек

cmps (CoMPare String) — сравнить строки;

cmpsb (CoMPare String Byte) — сравнить строку байт;

cmpsw (CoMPare String Word) — сравнить строку слов;

cmpsd (CoMPare String Double word) — сравнить строку двойных слов.

Алгоритм работы

Алгоритм работы команды cmps заключается в последовательном выполнении вычитания (элемент цепочки-источника — элемент цепочки-получателя) над очередными элементами обеих цепочек.

Принцип выполнения вычитания командой cmps аналогичен команде сравнения cmp. Она, так же, как и cmp, производит вычитание элементов, не записывая при этом результата, и устанавливает флаги zf, sf и of.

Алгоритм работы

Чтобы заставить команду cmps выполняться несколько раз, то есть производить последовательное сравнение элементов цепочек, необходимо перед командой cmps определить префикс повторения.

С командой cmps можно использовать префикс повторения repe/repz или repne/repnz

Алгоритм работы

Чтобы заставить команду cmps выполняться несколько раз, то есть производить последовательное сравнение элементов цепочек, необходимо перед командой cmps определить префикс повторения.

С командой cmps можно использовать префикс повторения repe/repz или repne/repnz

Алгоритм работы

repe или repz — если необходимо организовать сравнение до тех пор, пока не будет выполнено одно из двух условий:

- достигнут конец цепочки (содержимое ecx/cx равно нулю);
- в цепочках встретились разные элементы (флаг zf стал равен нулю);

repne или repnz — если нужно проводить сравнение до тех пор, пока:

- не будет достигнут конец цепочки (содержимое ecx/cx равно нулю);
- в цепочках встретились одинаковые элементы (флаг zf стал равен единице).

```
char str1[] = "Hello";
char str2[] = "Hello";
int res;
_asm {
    lea esi, str1;
    lea edi, str2;
    mov ecx, 5;
    repe cmps;
    je found;
    mov res, 0;
    jmp end1;
found:
    mov res, 1;
end1:
```

Алгоритм работы

Причина прекращения операции сравнения	Команда условного перехода, реализующая переход по этой причине
операнд_источник > операнд_приемник	jg
операнд_источник = операнд_приемник	je
операнд_источник <> операнд_приемник	jne
операнд_источник < операнд_приемник	jl
операнд_источник <= операнд_приемник	jle
операнд_источник >= операнд_приемник	jge

Операция сканирования цепочек

Команды, реализующие эту операцию-примитив, производят поиск некоторого значения в области памяти. Логически эта область памяти рассматривается как последовательность (цепочка) элементов фиксированной длины размером 8, 16 или 32 бит.

Искомое значение предварительно должно быть помещено в регистр al/ax/eax. Выбор конкретного регистра из этих трех должен быть согласован с размером элементов цепочки, в которой осуществляется поиск.

Операция сканирования цепочек

Система команд микропроцессора предоставляет программисту четыре команды сканирования цепочки.

Выбор конкретной команды определяется размером элемента:

scas адрес_приемника (SCAning String) — сканировать цепочку;

scasb (SCAning String Byte) — сканировать цепочку байт;

scasw (SCAning String Word) — сканировать цепочку слов;

scasd (SCAning String Double Word) — сканировать цепочку двойных слов.

Команда scas

scas адрес_приемника

Команда имеет один операнд, обозначающий местонахождение цепочки в дополнительном сегменте

Команда scas

префиксы repe/repz или repne/repnz:

repe или repz — если нужно организовать поиск до тех пор, пока не будет выполнено одно из двух условий:

- **достигнут конец цепочки (содержимое ecx/cx равно 0);**
- **в цепочке встретился элемент, отличный от элемента в регистре al/ax/eax;**

repne или repnz — если нужно организовать поиск до тех пор, пока не будет выполнено одно из двух условий:

- **достигнут конец цепочки (содержимое ecx/cx равно 0);**
- **в цепочке встретился элемент, совпадающий с элементом в регистре al/ax/eax.**

```
char str1[] = "Hello";
char _ch = '1';
_asm {
    cld
    lea edi, str1;//строка
    mov al, _ch;
    mov ecx, 5;
    repne scas str1;//выход при первом обнаружении
    je found;
    mov eax, 0;
    push eax;
    call print;
    pop eax;
    jmp end1
found:
    mov eax, 1;
    push eax;
    call print;
    pop eax;
end1:
}
```