

# Практика №1

Разработка программ на языке  
Ассемблера

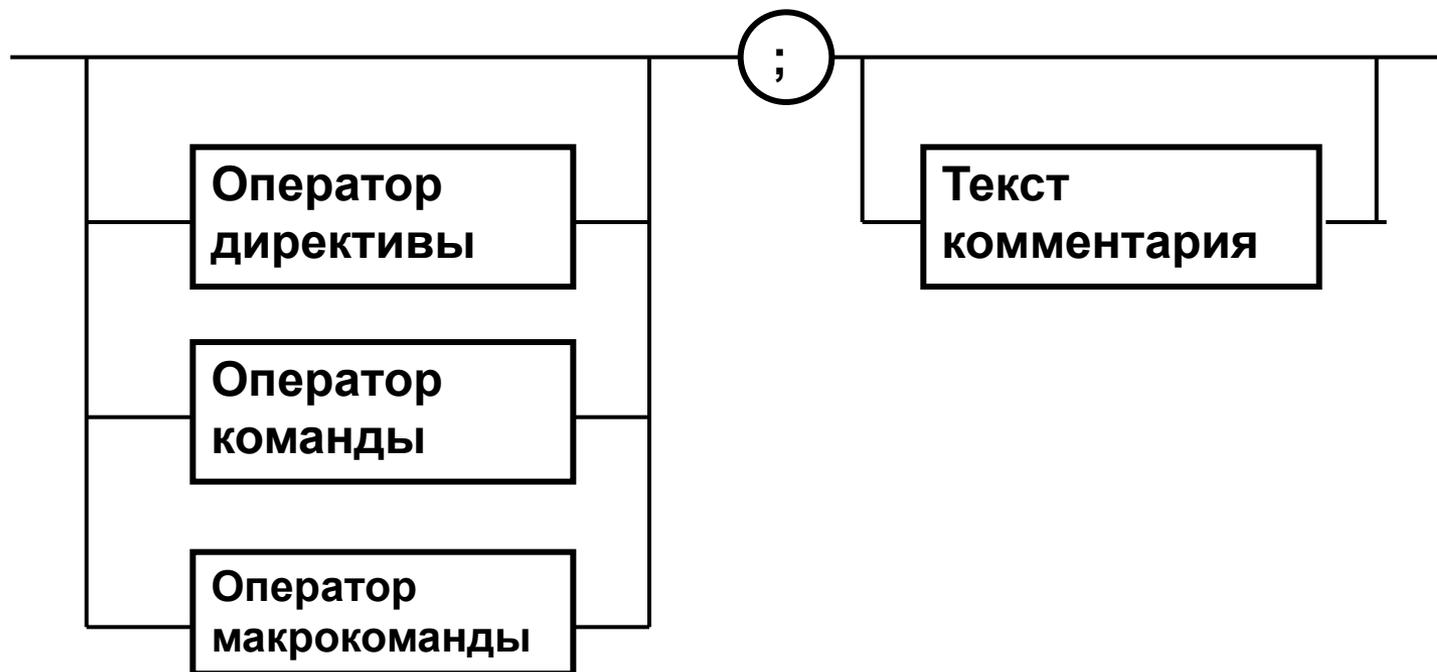
# Язык Ассемблер

- Является символическим аналогом машинного языка: программа отражает все особенности архитектуры процессора: организацию памяти, способы адресации операндов, правила использования регистров и т.д.
- Программа на Ассемблере представляет собой совокупность блоков памяти, называемых сегментами памяти.
- Программа состоит из предложений Ассемблера.

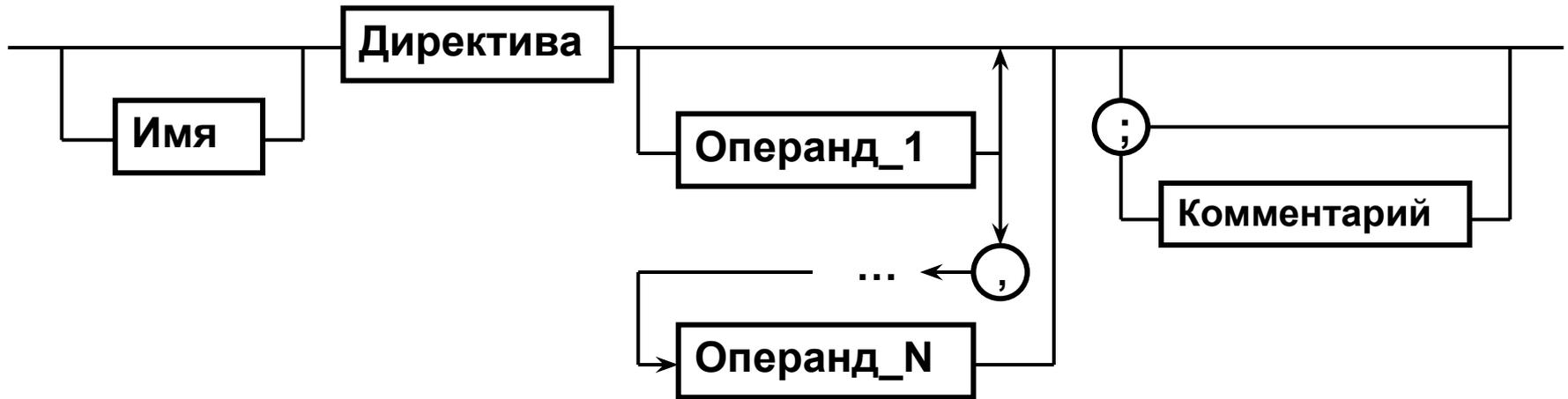
# Предложения Ассемблера

- Команды (или инструкции) – символические аналоги машинных команд.
- Макрокоманды – оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями.
- Директивы – указания транслятору на выполнение некоторых действий.
- Строки комментариев – текст, игнорирующийся транслятором.

# Формат предложения



# Формат директив



# Подготовка и запуск программ на Ассемблере

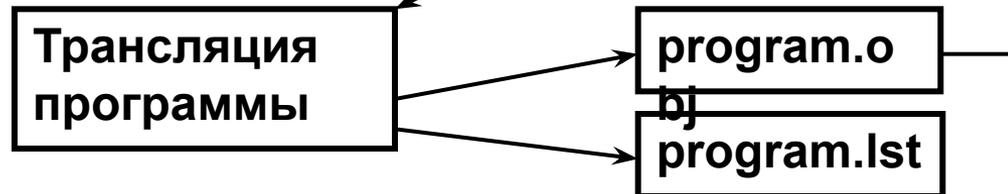
1. Разработка программа на «чистом» ассемблере
2. Использование ассемблерных вставок на ЯВУ

# Процесс разработки программ на Ассемблере

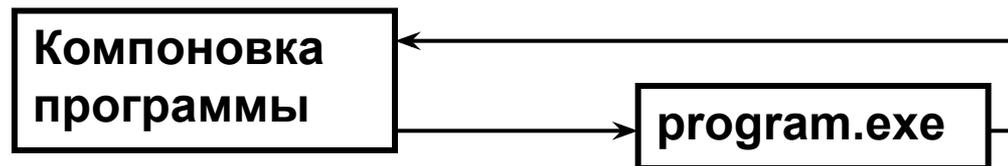
1. Ввод исходного текста программы



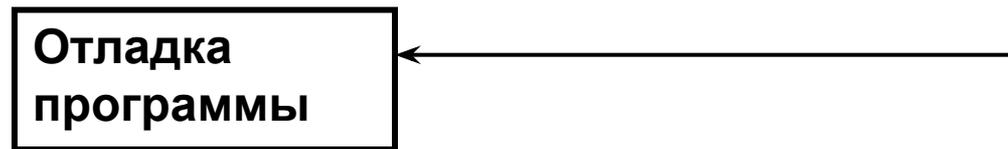
2. Создание объектного модуля



3. Создание загрузочного модуля



4. Отладка программы



**.686P**

**.MODEL FLAT, STDCALL**

**.STACK 4096**

**.DATA**

MB OK EQU 0

MSG TITLE DB "Native ASM",0

MSG TEXT DB "Программа на чистом Ассемблере!",0

HW DD ?

**EXTERN MessageBoxA@16:NEAR**

**.CODE**

**START:**

**PUSH** MB OK

**PUSH** OFFSET MSG\_TITLE

**PUSH** OFFSET MSG\_TEXT

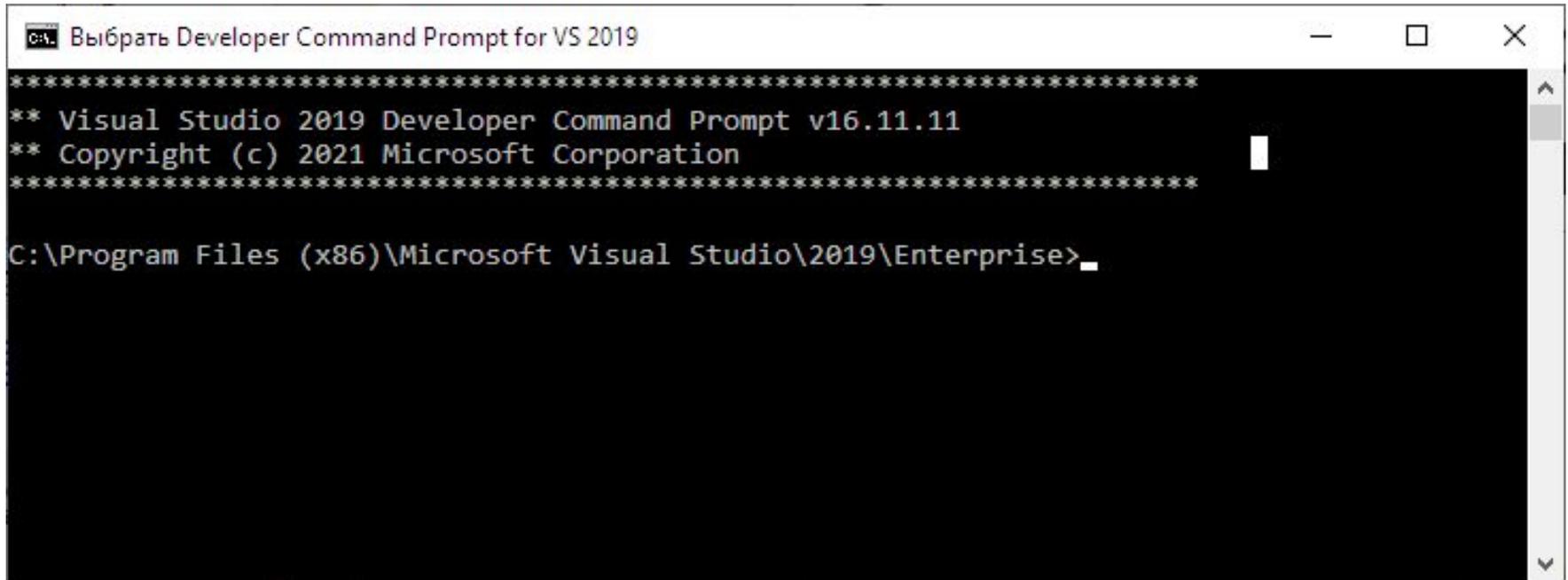
**PUSH** HW

**CALL** MessageBoxA@16

**RET**

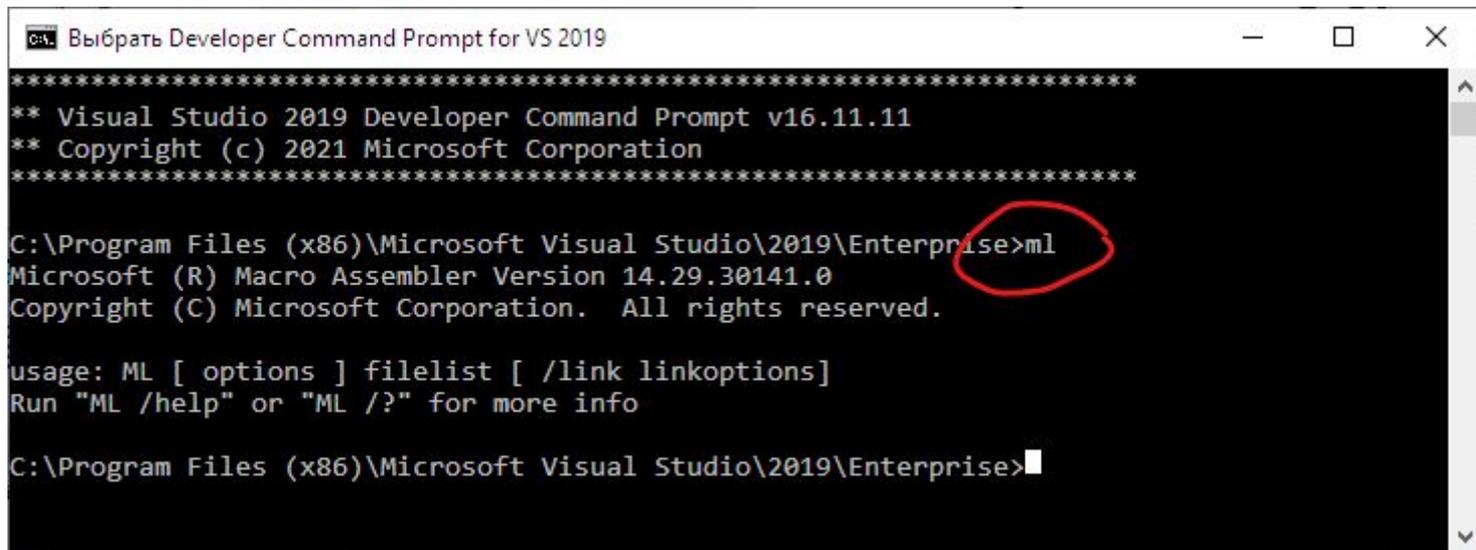
**END START**

# Visual Studio Developer Command Prompt

A screenshot of a Windows command prompt window titled "Выбрать Developer Command Prompt for VS 2019". The window has a black background and white text. The text inside the window reads: "\*\*\*\*\*  
\*\* Visual Studio 2019 Developer Command Prompt v16.11.11  
\*\* Copyright (c) 2021 Microsoft Corporation  
\*\*\*\*\*  
C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise>". A white cursor is positioned at the end of the last line.

```
CA:  Выбрать Developer Command Prompt for VS 2019
*****
** Visual Studio 2019 Developer Command Prompt v16.11.11
** Copyright (c) 2021 Microsoft Corporation
*****
C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise>
```

# ml.exe - Microsoft (R) Macro Assembler Version



```
Выбрать Developer Command Prompt for VS 2019
*****
** Visual Studio 2019 Developer Command Prompt v16.11.11
** Copyright (c) 2021 Microsoft Corporation
*****
C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise>ml
Microsoft (R) Macro Assembler Version 14.29.30141.0
Copyright (C) Microsoft Corporation. All rights reserved.

usage: ML [ options ] filelist [ /link linkoptions ]
Run "ML /help" or "ML /?" for more info

C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise>
```

`ml /c prog1.asm`

# Линковка программы

```
link /SUBSYSTEM:WINDOWS prog1.obj user32.lib
```

# Использование ассемблерных вставок (C++)

*asm-block:*

*\_\_asm assembly-instruction ;<sub>opt</sub>*

*\_\_asm { assembly-instruction-list } ;<sub>opt</sub>*

*assembly-instruction-list:*

*assembly-instruction ;<sub>opt</sub>*

*assembly-instruction ; assembly-instruction-list ;<sub>op</sub>*

t

# Пример ассемблерной вставки и операторов ввода/вывода

```
#include <iostream>
using namespace std;

int main()
{
    int n = 0;
    short r = 0;
    cout << "Input number:";
    cin >> n;
    _asm
    {
        xor ax, ax
        mov ecx, n

        mov bx, 0
        mov ax, 1

    startLoop:
        mov r, ax
        xchg ax, bx
        add ax, bx
        loop startLoop

        mov r, ax
    }

    cout << "Result:" << r << endl;
    system("pause");
    return 0;
}
```

# Синтаксис языка Ассемблер

Предложения Ассемблера формируются из *лексем*.

Лексема - синтаксически неразделимая последовательность допустимых символов языка, имеющие смысл для транслятора.

Лексемами являются:

- Идентификаторы – последовательности допустимых символов, использующиеся для обозначения таких объектов программы, как коды операций, имена переменных и названия меток.
- Цепочки символов – последовательности символов, заключенные в одинарные или двойные кавычки.
- Целые числа в двоичной, десятичной или шестнадцатеричной системах счисления:

# Формат команд и макрокоманд



- Имя метки – идентификатор, значением которого является адрес первого байта того предложения исходного текста программы, которое он обозначает.
- Имя – идентификатор, отличающий данную директиву от других директив.
- Код операции или директива – это мнемоническое обозначения соответствующей машинной команды, макрокоманды или директивы транслятора.
- Операнды – части команды, макрокоманды или директивы ассемблера, обозначающие объекты, над которыми производятся действия.

# Синтаксис языка Ассемблер

Допустимыми символами при написании текста программ являются:

- все латинские буквы;
- цифры;
- знаки: ?, @, \$, \_, &;
- разделители: , . [ ] ( ) < > { } + / \* % ! ' " ? \ =  
# ^

# Запись числовых констант

10000011**b** – двоичная система счисления

129, 123**d** – десятичная система счисления

74**q** – восьмеричная система счисления

2A**h** – шестнадцатеричная система счисления (константа должна начинаться с цифры)

# Простые “типы данных”

Обозначения:

- ? – показывает, что значение не определено;
- Значение инициализации – значение элемента данных, которое будет занесено после загрузки программы;
- Выражение – итеративная конструкция;
- Имя – некоторое символическое имя метки или ячейки данных.

Типы данных:

- db – 1 байт
- dw – 2 байта
- dd – 4 байта
- dq – 8 байт
- df – 6 байт
- dp – 6 байт
- dt – 10 байт

# Команды пересылки данных общего назначения

**mov** <операнд назначения>, <операнд-источник>

**xchg** <операнд1>, <операнд2>

# mov – основная команда пересылки данных

- Схема команды:

mov приемник, источник

- Назначение:

пересылка данных между регистрами или регистрами и памятью.

- Алгоритм работы:

копирование второго операнда в первый операнд.

- Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

# Инструкции сложения ADD и вычитания SUB

Команда **ADD** требует двух операндов, как и команда MOV:

**ADD o1, o2**

Команда **ADD** складывает оба операнда и записывает результат в o1, предыдущее значение которого теряется.

Точно так же работает команда вычитания — **SUB**:

**SUB o1, o2**

Результат, o1-o2, будет сохранен в o1, исходное значение o1 будет потеряно.

```
mov ax, 8      ; заносим в AX число 8
mov cx, 6      ; заносим в CX число 6
mov dx, cx     ; копируем CX в DX, DX = 6
add dx, ax     ; DX = DX + AX
```

Команда **ADD** сохранит результат  $DX + AX$  в регистре  $DX$ , а исходные значения  $AX$  и  $CX$  останутся нетронутыми.

# Команды инкрементирования INC и декрементирования DEC

Эти команды предназначены для инкрементирования и декрементирования.

Команда **INC** добавляет, а **DEC** вычитает единицу из единственного операнда.

Допустимые типы операнда — такие же, как у команд ADD и SUB, а формат команд таков:

```
INC o1      ;o1 = o1 + 1
DEC o1      ;o1 = o1 - 1
```

**Ни одна из этих инструкций не изменяет флаг CF.**

```
add al,1    ;AL = AL + 1
inc al      ;AL = AL + 1
Inc number  ;number = number+1
```

# Команда MUL

Команда **MUL** может быть записана в трех различных форматах — в зависимости от операнда:

**MUL r/m8**

**MUL r/m16**

**MUL r/m32**

В 8-разрядной форме операнд может быть любым 8-битным регистром или адресом памяти. Второй операнд всегда хранится в AL. Результат (произведение) будет записан в регистр AX.

**(r/m8) \* AL -> AX**

В 16-разрядной форме операнд может быть любым 16-битным регистром или адресом памяти. Второй операнд всегда хранится в AX. Результат сохраняется в паре DX:AX.

**(r/m16) \* AX -> DX:AX**

В 32-разрядной форме второй операнд находится в регистре EAX, а результат записывается в пару EDX:EAX.

**(r/m32) \* EAX -> EDX:EAX**

# Команда MUL

**Пример 1:** умножить значения, сохраненные в регистрах BH и CL, результат сохранить в регистр AX:

```
mov al, bh      ;AL = BH — сначала заносим в AL второй операнд
mul cl          ;AX = AL * CL — умножаем его на CL
```

Результат будет сохранен в регистре AX.

**Пример 2:** вычислить  $486^2$ , результат сохранить в DX:AX:

```
mov ax, 486    ; AX = 486
mul ax         ; AX * AX -> DX:AX
```

**Пример 3:** вычислить диаметр по радиусу, сохраненному в 8-битной переменной radius, результат записать в 16-битную переменную diameter:

```
mov al, 2      ; AL = 2
mul radius     ; AX = radius * 2
mov diameter, ax ; diameter <- AX
```

# Команда DIV

Подобно команде **MUL**, команда **DIV** может быть представлена в трех различных форматах в зависимости от типа операнда:

**DIV r/m8**

**DIV r/m16**

**DIV r/m32**

Операнд служит делителем, а делимое находится в фиксированном месте (как

в случае с **MUL**) в 8-битной форме переменный операнд (делитель) может быть любым 8-битным регистром или адресом памяти. Делимое содержится в **AX**. Результат сохраняется так: частное — в **AL**, остаток — в **AH**.

**AX / (r/m8) -> AL, остаток -> AH**

В 16-битной форме операнд может быть любым 16-битным регистром или адресом памяти. Второй операнд всегда находится в паре **DX:AX**. Результат сохраняется в паре **DX:AX** (**DX** — остаток, **AX** — частное).

**DX:AX / (r/m16) -> AX, остаток -> DX**

В 32-разрядной форме делимое находится в паре **EDX:EAX**, а результат записывается в пару **EDX:EAX** (частное в **EAX**, остаток в **EDX**).

**EDX:EAX / (r/m32) -> EAX, остаток -> EDX**