**Надо сделать:** Получить оценку

1. Запускаем Microsoft Visual Studio 2010, выбираем Файл – Создать – Проект… :

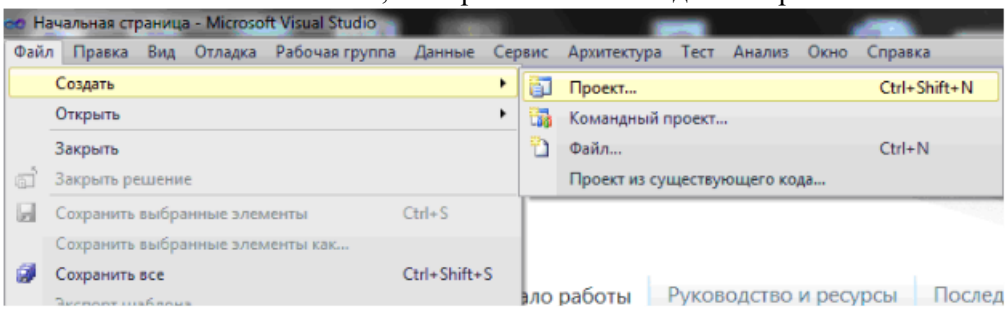


Рис. 3.1. Создание нового проекта

2. В открывшемся окне выбираем Другие языки – Visual C++ - Win32 – Консольное приложение Win32

Проекту необходимо задать имя и указать расположение. Выбранный тип проекта позволяет создавать приложение-«обертку» для нашего ассемблерного кода, используя только API-функции Windows.

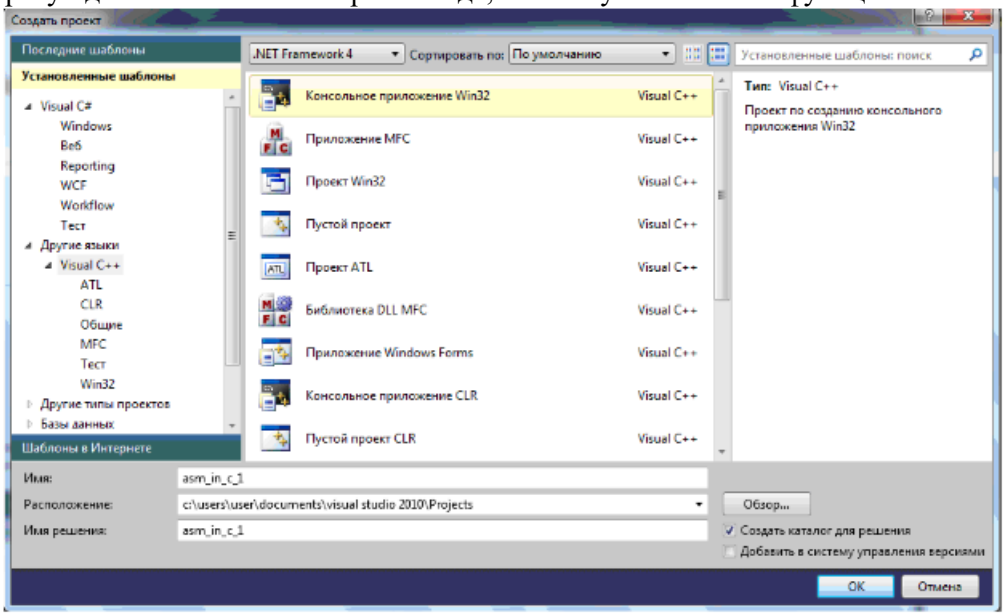


Рис. 3.2. Окно создания нового проекта

3. В открывшемся диалоговом окне необходимо нажать Далее, затем выбрать тип приложения Консольное приложение и отметить галочку Пустой проект, затем нажать Готово.

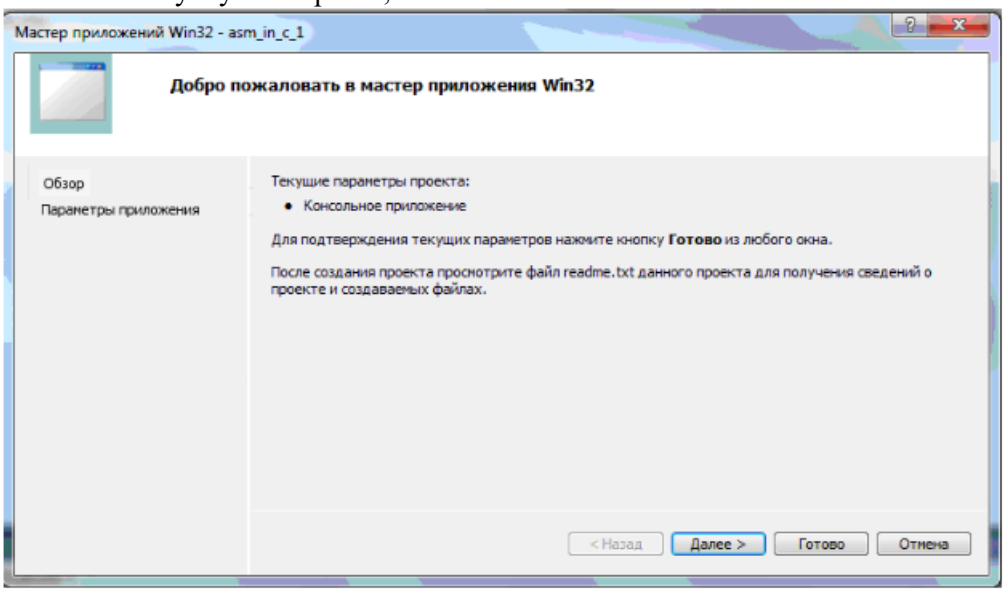


Рис. 3.3 Окно мастера приложений

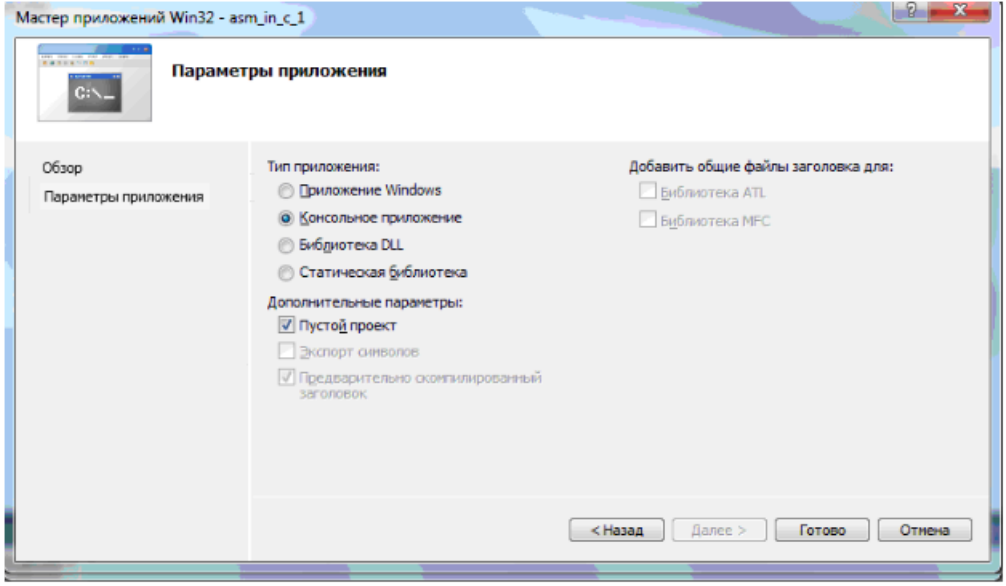


Рис. 3.4. Выбор типа приложения

4. После того, как проект создан, в Обозревателе решений выбираем Файлы исходного кода – Добавить – Создать элемент … (Обозреватель решений доступен во вкладке Вид).

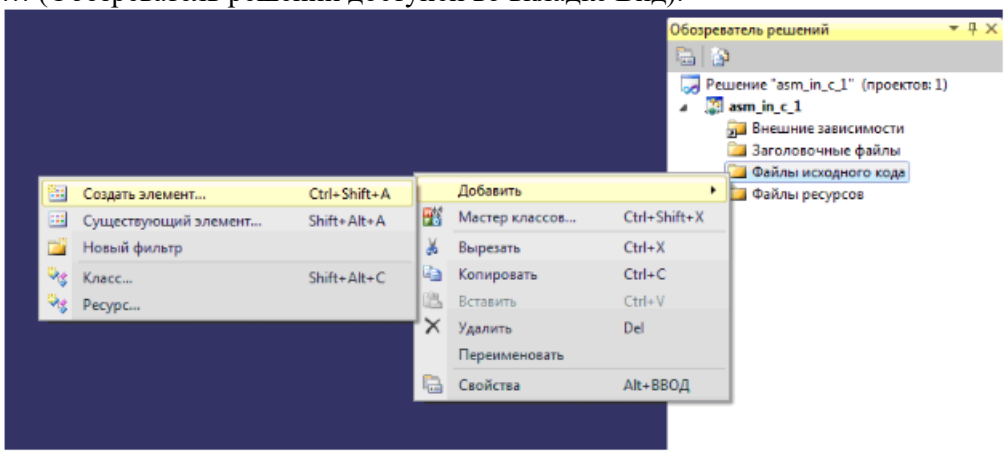


Рис. 3.5. Создание нового элемента в проекте

5. В открывшемся диалоговом окне выбираем Файл C++(.cpp). Поскольку мы предполагаем использовать лишь один файл в нашем проекте, назовем его также, как и проект.

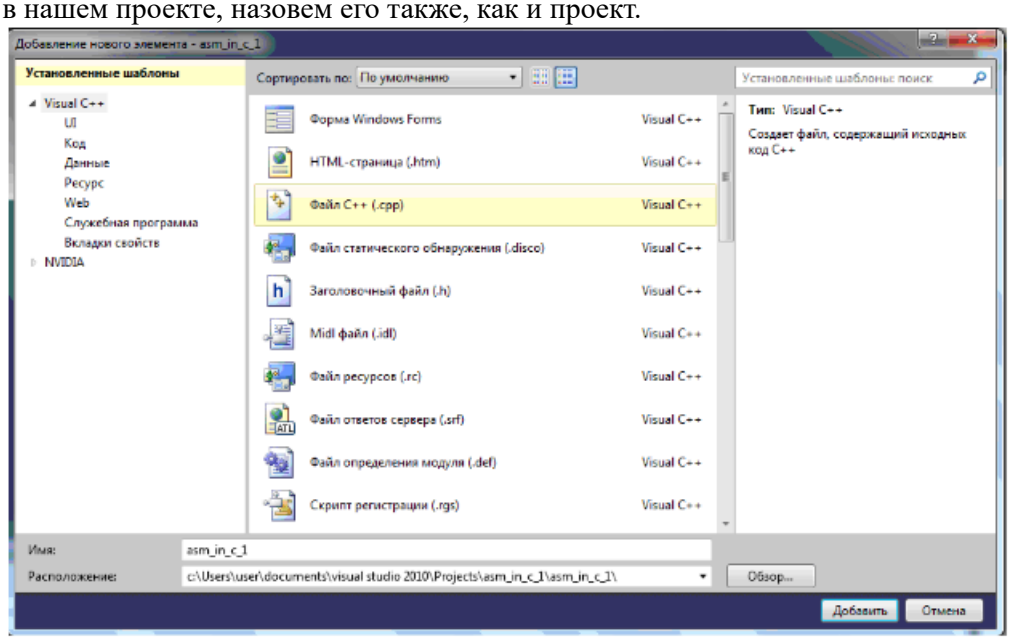


Рис. 3.6. Выбор типа создаваемого элемента

6. Напишем элементарную программу.

```
/* подключаемые заголовочные файлы */
#include <stdio.h> // необходим для работы printf
#include <conio.h> // необходим для работы _getch();

/* объявления функций */
int add(int, int);  // складывает два целых числа
int sub(int, int);  // вычитает из первого целого второе
```

```
int prov(int, int); // в случае, если первое число больше второго, выполняет sub,
// иначе выполняет add

/* глобальные переменные */
int i1, i2;

void main() // основная функция. Тип void означает, что эта функция ничего
     // не возвращает
{
i1 = 10; // объявляем локальные переменные
i2 = 20;
printf("%d\n", prov(i1, i2)); // выводим результат функции prov
// запись в кавычках определяет формат вывода:
// %d означает, что будет выведено целое число,
// \n означает "конец строки"
_getch(); // ждет ввода любого символа с клавиатуры и возвращает его,
   // используется для того, чтобы консоль не закрывалась после выполнения
   // программы в режиме отладки
}

/* реализация функций */
int prov(int a, int b)
{
int res;
if (a>b)
res = sub(a, b);
else
res = add(a, b);
return res;
}

int add(int a, int b)
{
return a+b;
}

int sub(int a, int b)
{
return a-b;
}
```

Функция **printf** будет использоваться нами постоянно, поэтому есть смысл сказать о ней несколько слов. Она является стандартной библиотечной функцией языка Си и выводит информацию на консольное устройство (текстовый экран). В общем случае формат этой функции можно представить так:

```
printf(формат, список переменных)
```

Параметр формат представляет собой строку, ограниченную кавычками, в которой, в частности, должны указываться типы переменных, содержащиеся в параметре список переменных. Строка printf(»%d», a) означает, что будет выведено значение переменной a, причем тип задан как %d , т. е. целый со знаком и 32-битный по умолчанию. Кроме %d можно также использовать %u - целый без знака, %x - 32-битное число в шестнадцатеричной системе счисления, %s – строка, %d – тип double. Типы в формате должны соответствовать переменным в списке. В строке формата можно указывать комментарий, который будет выведен вместе со значениями:

```
printf("%d больше чем %d", a, b)
```

7. Выберем Отладка – Начать отладку.

Очевидно, что результат работы программы 30.

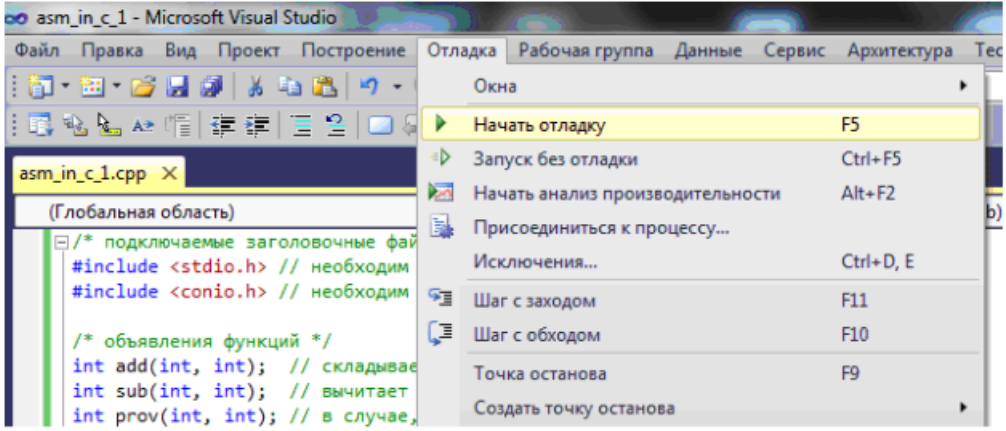


Рис. 3.7. Запуск отладчика

Попробуйте закомментировать строки #include <conio.h> и _getch() и запустить программу в режиме отладки. Консоль закроется сразу после выполнения программы. Теперь попробуйте выбрать Запуск без отладки. В этом случае будем выведено сообщение **Для продолжения нажмите любую клавишу…** .

**Начало программирования на ассемблере**

Для использования команд ассемблера в программах на языке С применяется ключевое слово *asm. Есть два способа использовать это ключевое слово в программе: употреблять* asm для каждой команды процессора или использовать фигурные скобки для обозначения блока ассемблерных команд. Таким образом, фрагмент

```
__asm
{
MOV EAX, EDX;
OR EAX, EBX;
}
```

полностью эквивалентен фрагменту

```
__asm MOV EAX, EDX;
__asm OR EAX, EBX;
```

**Пример простейшей программы на Си с использованием ассемблерной вставки:**

```
/* подключаемые заголовочные файлы */
#include <stdio.h> // необходим для работы printf
#include <conio.h> // необходим для работы _getch();
/* глобальные переменные */
int a;
/* главная функция */
void main()
{
a=10;
__asm {
ADD a, 10; // прибавляем к а 10
SUB a, 2;  // вычитаем из а 2
};
printf("%d ",a);
_getch();
}
```

**Кратко о регистрах**

Микропроцессоры Intel включают регистры общего назначения, регистр флагов, сегментные регистры, управляющие регистры, системные адресные регистры, а также отладочные регистры. Особо следует отметить регистр EIP, который называют указателем команд. В нем всегда содержится адрес команды относительно начала сегмента. К данному регистру нет прямого доступа, но косвенно многие команды изменяют его содержимое (команды передачи управления).

Рассмотрим **рабочие регистры**. Их надо использовать, когда очень важно быстро обратиться к данным. По возможности следует так писать программы, чтобы большую часть вычислений проводить с загруженными один раз в эти регистры данными.

| EAX | EBX | ECX | EDX |
|---|---|---|---|
| E-часть AX | E-часть BX | E-часть CX | E-часть DX |
| E-часть AH AL | E-часть BH BL | E-часть CH CL | E-часть DH DL |

| | |
|---|---|
| EAX (сокращение от Accumulator) | EAX=16+AX=16+AH+AL |
| EBX (сокращение от Base) | EBX=16+BX=16+BH+BL |

ECX (сокращение от Counter) ECX=16+CX=16+CH+CL

EDX (сокращение от Data) EDX=16+DX=16+DH+DL

Названия регистров можно писать и строчными буквами. EAX, EBX, ECX и EDX - 32-битные регистры данных центрального процессора (от 386-го и по сей день). В эти регистры помещаются необходимые значения, и в них же чаще всего оказываются результаты вычислений. В 32 битах можно хранить hex-число от 00 00 00 00h до FF FF FF FFh. И это всё, что может там храниться. На назначения (Accumulator, Base, Counter, Data) пока можно не обращать внимания. Для записи числа в регистр, в переменную или в память используется оператор MOV:

**Команда MOV**

| | |
|---|---|
| Происхождение | от англ. слова move - движение, перемена места |
| Формат | mov приемник, источник |
| Действие | Копирует содержимое источника в приёмник |
| Примечание | MOV не может передавать данные между двумя адресами оперативной памяти (для этой цели существуют команды MOVS) |

При использовании ассемблерных вставок с C++ в качестве одного из элементов (источника или приемника) может выступать переменная, объявленная средствами C++. Переписывать значение из одной переменной в другую нельзя.

Пример

```
mov  ah,09   // поместить значение 9 в регистр AH
mov  dx,010D // поместить значение 010Dh в DX
```

Получив первую инструкцию, процессор выполнит инициализацию своего 8-битного регистра AH значением 9, после чего регистр AH будет содержать только байт 09. При выполнении второй команды процессор поместит в свой 16-битный регистр DX значение 010Dh, после чего регистр DX будет содержать только эти два байта 01 и 0Dh. Важно понять следующее: так как регистр DX состоит из DH и DL, то можно сказать, что после выполнения второй строки кода программы в регистре DH окажется значение 01, а в регистре DL окажется значение 0Dh.

```
     DH  DL
DX
=    01  0D
```

Важно! В 16-битный регистр (AX,BX,CX,DX) нельзя положить значение больше двух байт (FFFFh), а в 8-битный (AH,AL, BH,BL, CH,CL, DH,DL) нельзя положить больше байта, то есть FFh. Допустим:

```
EAX=  99884433
AX=   4433
AH=   44
AL=   33
```

Важно понять, что физически есть только 4 байта (99 88 44 33h). По отдельности можно обращаться к AX за значением 4433h, или к AH за 44h, или к AL за 33h. Но 9988h находится в E-части, а у неё нет собственного имени, она не является подрегистром. Вы не можете прочитать или загрузить 2 старших байта такого регистра, не обратившись ко всему регистру. Пример:

```
mov  EAX, 0FFFFFFFFh  // Так правильно, и EAX будет равен FFFFFFFF
mov  EAX, 01FFFFFFFFh // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  EAX, 0           // Так правильно, и EAX станет равен 00000000
mov  AX,  0FFFFh      // Так правильно, и EAX будет равен 0000FFFF
mov  AX,  1FFFFh      // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  AX,  0           // Так правильно, и AX станет равен 0000
mov  AH,  111h        // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  AL,  100h        // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  AL,  0BBh        // Так правильно, и EAX станет равен 000000BB
mov  AH,  AL          // так правильно, и EAX станет равен 0000BBBB
```

К старшей части EAX отдельно обращаться можно, например, при помощи команд сдвига битов (будет рассмотрено подробно позже):

```
shl EAX,10h           // Сделает теперь регистр EAX равным BBBB0000
shr EAX,10h           // А эта команда сделает его обратно равным 0000BBBB
```

Арифметические операции

Для выполнения арифметических операций используются команды ADD (сложение двух чисел), I (прибавление единицы), SUB (вычитание), DEC (вычитание из числа единицы), IMUL (умножение), Dıv (деление). В этой работе рассмотрим команды сложения и вычитания.

**Команда ADD**

| | |
|---|---|
| Происхождение | от англ. слова add - прибавлять, присоединять |
| Формат | add приёмник, источник |
| Действие | приёмник = приёмник + источник |

| | |
|---|---|
| Примечание | если нужно увеличить всего на один, лучше использовать команду INC |

**Пример**

```
add   AH,AL    // прибавить к AH содержимое AL
```

**Команда SUB**

| | |
|---|---|
| Происхождение | от англ. слова sub, subtraction - вычитание |
| Формат | sub приёмник, источник |
| Действие | приёмник = приёмник - источник |
| Примечание | если нужно отнимать всего один, лучше использовать команду DEC |

**Пример**

```
sub   AH,AL    // вычесть из AH содержимое AL
```

**Задание**

1. Найдите сумму чисел, находящихся в регистрах EAX, EBX, ECX, накапливая ее в регистре EDX. Содержимое регистров EAX, EBX, ECX не меняйте.
2. Найдите разность суммы чисел, находящихся в регистрах EAX, EBX, и числа из регистра ECX. Результат – в регистре EDX. Содержимое регистров EAX, EBX, ECX не меняйте.
3. Сложите два вектора с целочисленными координатами (a1,a2) и (b1,b2).
4. Найдите разность двух векторов с целочисленными координатами (a1,a2) и (b1,b2).

1. Запускаем Microsoft Visual Studio 2010, выбираем Файл – Создать – Проект… :

Рис. 3.1. Создание нового проекта

2. В открывшемся окне выбираем Другие языки – Visual C++ - Win32 – Консольное приложение Win32

Проекту необходимо задать имя и указать расположение. Выбранный тип проекта позволяет создавать приложение-«обертку» для нашего ассемблерного кода, используя только API-функции Windows.

Рис. 3.2. Окно создания нового проекта

3. В открывшемся диалоговом окне необходимо нажать Далее, затем выбрать тип приложения Консольное приложение и отметить галочку Пустой проект, затем нажать Готово.

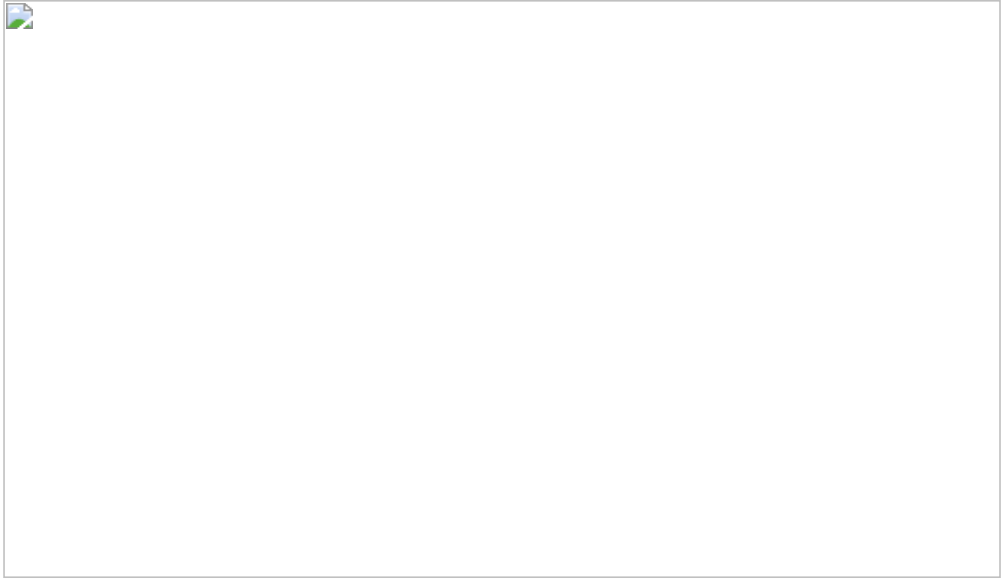

Рис. 3.3 Окно мастера приложений

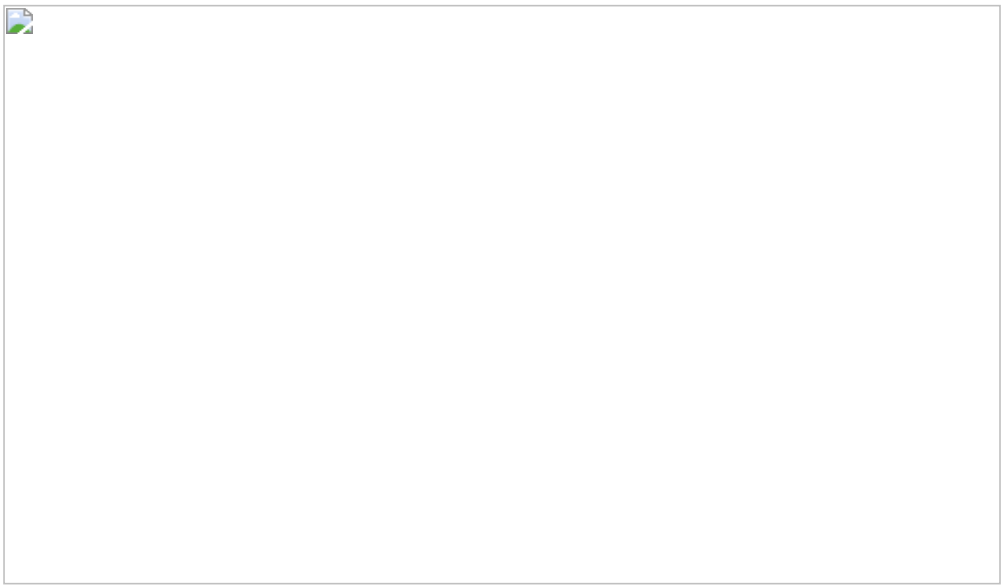

Рис. 3.4. Выбор типа приложения

4. После того, как проект создан, в Обозревателе решений выбираем Файлы исходного кода – Добавить – Создать элемент … (Обозреватель решений доступен во вкладке Вид).

Рис. 3.5. Создание нового элемента в проекте

5. В открывшемся диалоговом окне выбираем Файл C++(.cpp). Поскольку мы предполагаем использовать лишь один файл в нашем проекте, назовем его также, как и проект.

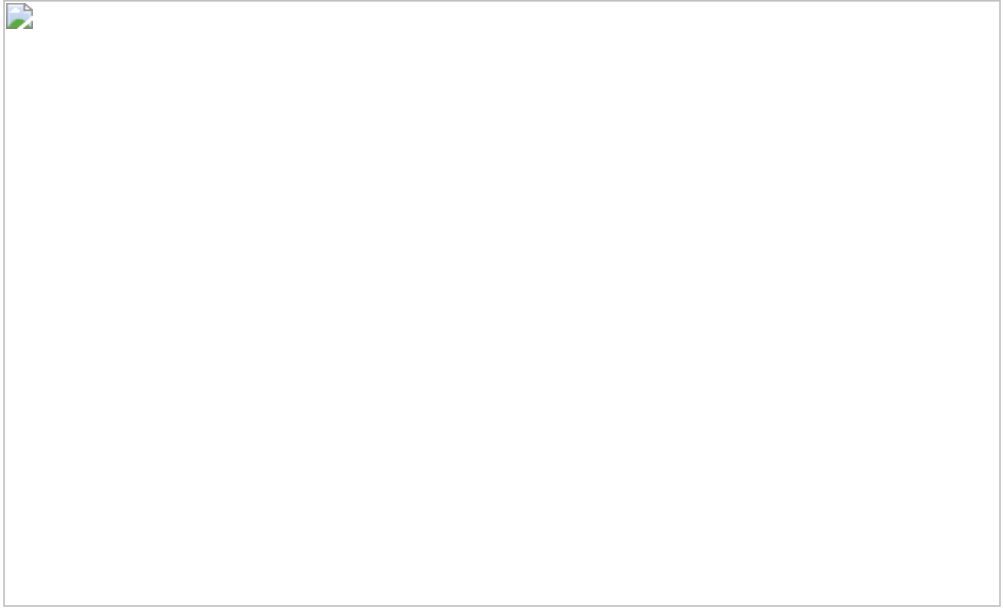

Рис. 3.6. Выбор типа создаваемого элемента

6. Напишем элементарную программу.

```
/* подключаемые заголовочные файлы */
#include <stdio.h> // необходим для работы printf
#include <conio.h> // необходим для работы _getch();

/* объявления функций */
int add(int, int);  // складывает два целых числа
int sub(int, int);  // вычитает из первого целого второе
int prov(int, int); // в случае, если первое число больше второго, выполняет sub,
                    // иначе выполняет add

/* глобальные переменные */
int i1, i2;

void main() // основная функция. Тип void означает, что эта функция ничего
            // не возвращает
{
       i1 = 10; // объявляем локальные переменные
       i2 = 20;
       printf("%d\n", prov(i1, i2)); // выводим результат функции prov
                                     // запись в кавычках определяет формат вывода:
                                     // %d означает, что будет выведено целое число,
                                     // \n означает "конец строки"
       _getch(); // ждет ввода любого символа с клавиатуры и возвращает его,
                 // используется для того, чтобы консоль не закрывалась после выполнения
                 // программы в режиме отладки
}

/* реализация функций */
int prov(int a, int b)
{
       int res;
       if (a>b)
             res = sub(a, b);
       else
             res = add(a, b);
       return res;
}

int add(int a, int b)
{
       return a+b;
}

int sub(int a, int b)
{
```

```
	return a-b;
}
```

Функция **printf** будет использоваться нами постоянно, поэтому есть смысл сказать о ней несколько слов. Она является стандартной библиотечной функцией языка Си и выводит информацию на консольное устройство (текстовый экран). В общем случае формат этой функции можно представить так:

```
printf(формат, список переменных)
```

Параметр формат представляет собой строку, ограниченную кавычками, в которой, в частности, должны указываться типы переменных, содержащиеся в параметре список переменных. Строка printf(»%d», a) означает, что будет выведено значение переменной a, причем тип задан как %d , т. е. целый со знаком и 32-битный по умолчанию. Кроме %d можно также использовать %u - целый без знака, %x - 32-битное число в шестнадцатеричной системе счисления, %s – строка, %d – тип double. Типы в формате должны соответствовать переменным в списке. В строке формата можно указывать комментарий, который будет выведен вместе со значениями:

```
printf("%d больше чем %d", a, b)
```

7. Выберем Отладка – Начать отладку.

Очевидно, что результат работы программы 30.

Рис. 3.7. Запуск отладчика

Попробуйте закомментировать строки #include <conio.h> и _getch() и запустить программу в режиме отладки. Консоль закроется сразу после выполнения программы. Теперь попробуйте выбрать Запуск без отладки. В этом случае будем выведено сообщение **Для продолжения нажмите любую клавишу…** .

**Начало программирования на ассемблере**

Для использования команд ассемблера в программах на языке С применяется ключевое слово *asm. Есть два способа использовать это ключевое слово в программе: употреблять* asm для каждой команды процессора или использовать фигурные скобки для обозначения блока ассемблерных команд. Таким образом, фрагмент

```
__asm
{
	MOV EAX, EDX;
	OR EAX, EBX;
}
```

полностью эквивалентен фрагменту

```
__asm MOV EAX, EDX;
__asm OR EAX, EBX;
```

**Пример простейшей программы на Си с использованием ассемблерной вставки:**

```
/* подключаемые заголовочные файлы */
#include <stdio.h> // необходим для работы printf
#include <conio.h> // необходим для работы _getch();
/* глобальные переменные */
int a;
/* главная функция */
void main()
{
	a=10;
	__asm {
		ADD a, 10; // прибавляем к a 10
		SUB a, 2;  // вычитаем из a 2
	};
	printf("%d ",a);
	_getch();
}
```

**Кратко о регистрах**

Микропроцессоры Intel включают регистры общего назначения, регистр флагов, сегментные регистры, управляющие регистры, системные адресные регистры, а также отладочные регистры. Особо следует отметить регистр EIP, который называют указателем команд. В нем всегда содержится адрес команды относительно начала сегмента. К данному регистру нет прямого доступа, но косвенно многие команды изменяют его содержимое (команды передачи управления).

Рассмотрим **рабочие регистры**. Их надо использовать, когда очень важно быстро обратиться к данным. По возможности следует так писать программы, чтобы большую часть вычислений проводить с загруженными один раз в эти регистры данными.

| EAX | EBX | ECX | EDX |
|---|---|---|---|
| E-часть AX | E-часть BX | E-часть CX | E-часть DX |
| E-часть AH AL | E-часть BH BL | E-часть CH CL | E-часть DH DL |

| | |
|---|---|
| EAX (сокращение от Accumulator) | EAX=16+AX=16+AH+AL |
| EBX (сокращение от Base) | EBX=16+BX=16+BH+BL |
| ECX (сокращение от Counter) | ECX=16+CX=16+CH+CL |
| EDX (сокращение от Data) | EDX=16+DX=16+DH+DL |

Названия регистров можно писать и строчными буквами. EAX, EBX, ECX и EDX - 32-битные регистры данных центрального процессора (от 386-го и по сей день). В эти регистры помещаются необходимые значения, и в них же чаще всего оказываются результаты вычислений. В 32 битах можно хранить hex-число от 00 00 00 00h до FF FF FF FFh. И это всё, что может там храниться. На назначения (Accumulator, Base, Counter, Data) пока можно не обращать внимания. Для записи числа в регистр, в переменную или в память используется оператор MOV:

**Команда MOV**

| | |
|---|---|
| Происхождение | от англ. слова move - движение, перемена места |
| Формат | mov приемник, источник |
| Действие | Копирует содержимое источника в приёмник |
| Примечание | MOV не может передавать данные между двумя адресами оперативной памяти (для этой цели существуют команды MOVS) |

При использовании ассемблерных вставок с C++ в качестве одного из элементов (источника или приемника) может выступать переменная, объявленная средствами C++. Переписывать значение из одной переменной в другую нельзя.

Пример

```
mov  ah,09   // поместить значение 9 в регистр AH
mov  dx,010D // поместить значение 010Dh в DX
```

Получив первую инструкцию, процессор выполнит инициализацию своего 8-битного регистра AH значением 9, после чего регистр AH будет содержать только байт 09. При выполнении второй команды процессор поместит в свой 16-битный регистр DX значение 010Dh, после чего регистр DX будет содержать только эти два байта 01 и 0Dh. Важно понять следующее: так как регистр DX состоит из DH и DL, то можно сказать, что после выполнения второй строки кода программы в регистре DH окажется значение 01, а в регистре DL окажется значение 0Dh.

| | DH | DL |
|---|---|---|
| DX = | 01 | 0D |

Важно! В 16-битный регистр (AX,BX,CX,DX) нельзя положить значение больше двух байт (FFFFh), а в 8-битный (AH,AL, BH,BL, CH,CL, DH,DL) нельзя положить больше байта, то есть FFh. Допустим:

```
EAX= 99884433
AX=  4433
AH=  44
AL=  33
```

Важно понять, что физически есть только 4 байта (99 88 44 33h). По отдельности можно обращаться к AX за значением 4433h, или к AH за 44h, или к AL за 33h. Но 9988h находится в E-части, а у неё нет собственного имени, она не является подрегистром. Вы не можете прочитать или загрузить 2 старших байта такого регистра, не обратившись ко всему регистру. Пример:

```
mov  EAX, 0FFFFFFFFh  // Так правильно, и EAX будет равен FFFFFFFF
mov  EAX, 01FFFFFFFFh // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  EAX, 0           // Так правильно, и EAX станет равен 00000000
mov  AX,  0FFFFh      // Так правильно, и EAX будет равен 0000FFFF
mov  AX,  1FFFFh      // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  AX,  0           // Так правильно, и AX станет равен 0000
```

```
mov  AH,  111h        // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  AL,  100h        // Так НЕправильно. Значение больше регистра,
                         // данной операции быть не может
mov  AL,  0BBh        // Так правильно, и EAX станет равен 000000BB
mov  AH,  AL          // так правильно, и EAX станет равен 0000BBBB
```

К старшей части EAX отдельно обращаться можно, например, при помощи команд сдвига битов (будет рассмотрено подробно позже):

```
shl EAX,10h           // Сделает теперь регистр EAX равным BBBB0000
shr EAX,10h           // А эта команда сделает его обратно равным 0000BBBB
```

Арифметические операции

Для выполнения арифметических операций используются команды ADD (сложение двух чисел), INC (прибавление единицы), SUB (вычитание), DEC (вычитание из числа единицы), IMUL (умножение), DIV (деление). В этой работе рассмотрим команды сложения и вычитания.

**Команда ADD**

Происхождение от англ. слова add - прибавлять, присоединять

Формат add приёмник, источник

Действие приёмник = приёмник + источник

Примечание если нужно увеличить всего на один, лучше использовать команду INC

**Пример**

```
add   AH,AL    // прибавить к AH содержимое AL
```

**Команда SUB**

Происхождение от англ. слова sub, subtraction - вычитание

Формат sub приёмник, источник

Действие приёмник = приёмник - источник

Примечание если нужно отнимать всего один, лучше использовать команду DEC

**Пример**

```
sub   AH,AL    // вычесть из AH содержимое AL
```

**Задание**

1. Найдите сумму чисел, находящихся в регистрах EAX, EBX, ECX, накапливая ее в регистре EDX. Содержимое регистров EAX, EBX, ECX не меняйте.
2. Найдите разность суммы чисел, находящихся в регистрах EAX, EBX, и числа из регистра ECX. Результат – в регистре EDX. Содержимое регистров EAX, EBX, ECX не меняйте.
3. Сложите два вектора с целочисленными координатами (a1,a2) и (b1,b2).
4. Найдите разность двух векторов с целочисленными координатами (a1,a2) и (b1,b2).

Добавить ответ на задание

## Состояние ответа

| | |
|---|---|
| **Состояние ответа на задание** | Ответы на задание еще не представлены |
| **Состояние оценивания** | Не оценено |

Отдел электронных образовательных ресурсов и сетевого обучения

Структура университета

Телефон : +7 (4722) 39-22-51 (по вопросам ЭИОС). По вопросам справок: +7 (4722) 38-05-17 (МФЦ БелГАУ)

Эл.почта : help@belgau.ru