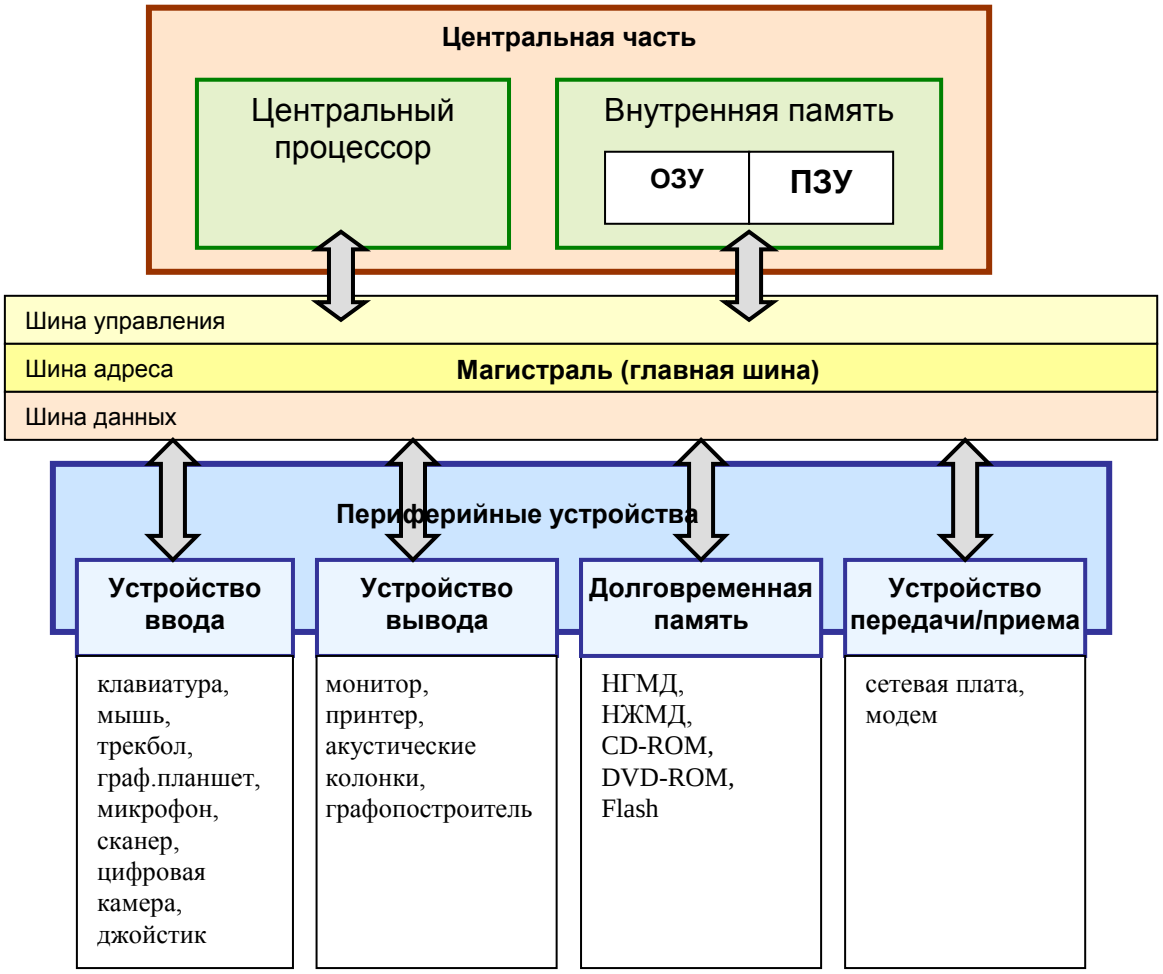
Лекция №1
Тема " Архитектура компьютера. Алгоритмический язык ассемблер"

Архитектура компьютера



Немного истории

1971 год – появился первый микропроцессор (процессор, реализованный в виде одной интегральной схемы). Его создала фирма Intel.
Автором первого 4-х разрядного микропроцессора был американский инженер Эдвард Хофф.

1981 год – первые персональные компьютеры корпорации IBM, которые получили название IBM PC. Они использовали 16-разрядный микропроцессор с 8-разрядной внешней шиной Intel 8088.

В дальнейшем в персональных компьютерах использовались процессоры 8086, которые являлись полностью 16 разрядными.

> Примечание
>
> С этих пор имя процессора стало нарицательным, о программах, использующих возможности процессора 8086, говорят, что они работают в режиме 86-го процессора.

1 февраля 1982 год – процессор 80286, в котором был реализован принципиально новый режим работы, получивший название *защищенного* режима. Однако процессор 80286 мог работать и в режиме процессора 8086, который стали называть *реальным*.

В дальнейшем появились процессоры 80386, 80486 и различные варианты процессора Pentium. Все они могут работать и в реальном, и в защищенном режимах.

Принципиальные различия реального и защищенного режимов

	Реальный режим	Защищенный режим
Способ обращения к оперативной памяти компьютера	Возможность адресовать память лишь в пределах 1 Мбайта.	Использует другой механизм, позволяющий обращаться к памяти объемом до 4 Гбайт.
Поддержание режима многозадачности	Не поддерживает.	Реализовано в самом микропроцессоре. Поддерживает работу в этом режиме и защищает задачи друг от друга.

Центральный процессор – программно-управляемое электронное устройство, построенное на одной или нескольких БИС (или СБИС) и предназначенное для цифровой обработки информации.

БИС (большая интегральная схема) и СБИС (сверхбольшая интегральная схема) представляют собой электронные схемы, реализованные в виде полупроводниковых кристаллов.

Интегральные схемы различаются по степени интеграции – коэффициент функциональной интеграции:

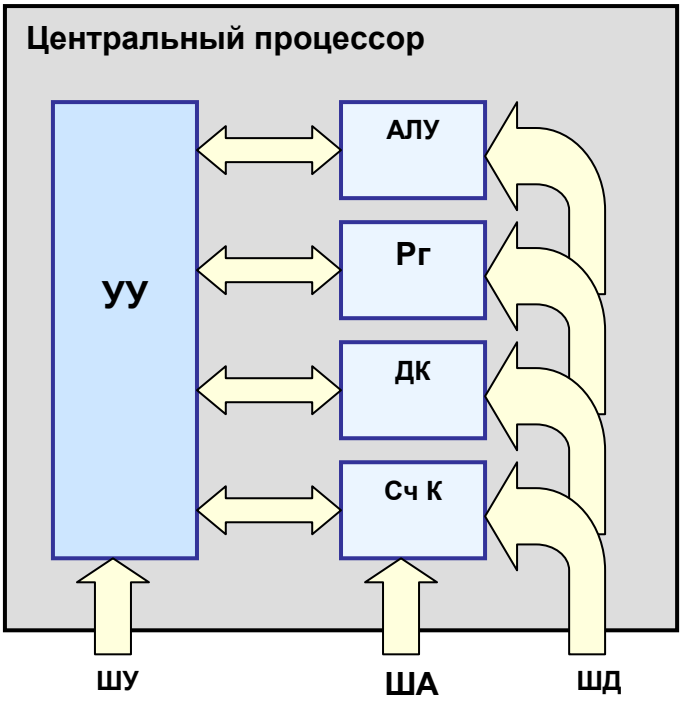
$$K_Ф = \lg N_Э$$

где $N_Э$ – количество логических элементов ИЛИ-НЕ или И-НЕ, расположенных на кристалле.

По величине $K_Ф$ различают:

- МИС (малые интегральные схемы) – Кф =< 1,
- СИС (средние интегральные схемы) – 1 < Кф =< 2,
- БИС (большие интегральные схемы) – 2 < Кф =< 3,
- СБИС (сверхбольшие интегральные схемы) – Кф > 3.

Функции центрального процессора



1. Выборка (чтение) команд программы из ОП.
2. Декодирование команд.
3. Выборка (чтение) из ОП данных, необходимых для выполнения операций, закодированных в командах.
4. Выполнение арифметических, логических и других операций, закодированных в командах.
5. Управление пересылкой информации между своими внутренними регистрами, основной памятью и портами ввода-вывода.
6. Отработка сигналов от устройств ввода-вывода, в том числе реализация прерываний.
7. Управление основными узлами компьютера и координация их взаимодействия.

Регистры - это небольшой специальный вид памяти, имеющийся в распоряжении процессора; служат для кратковременного хранения информации.

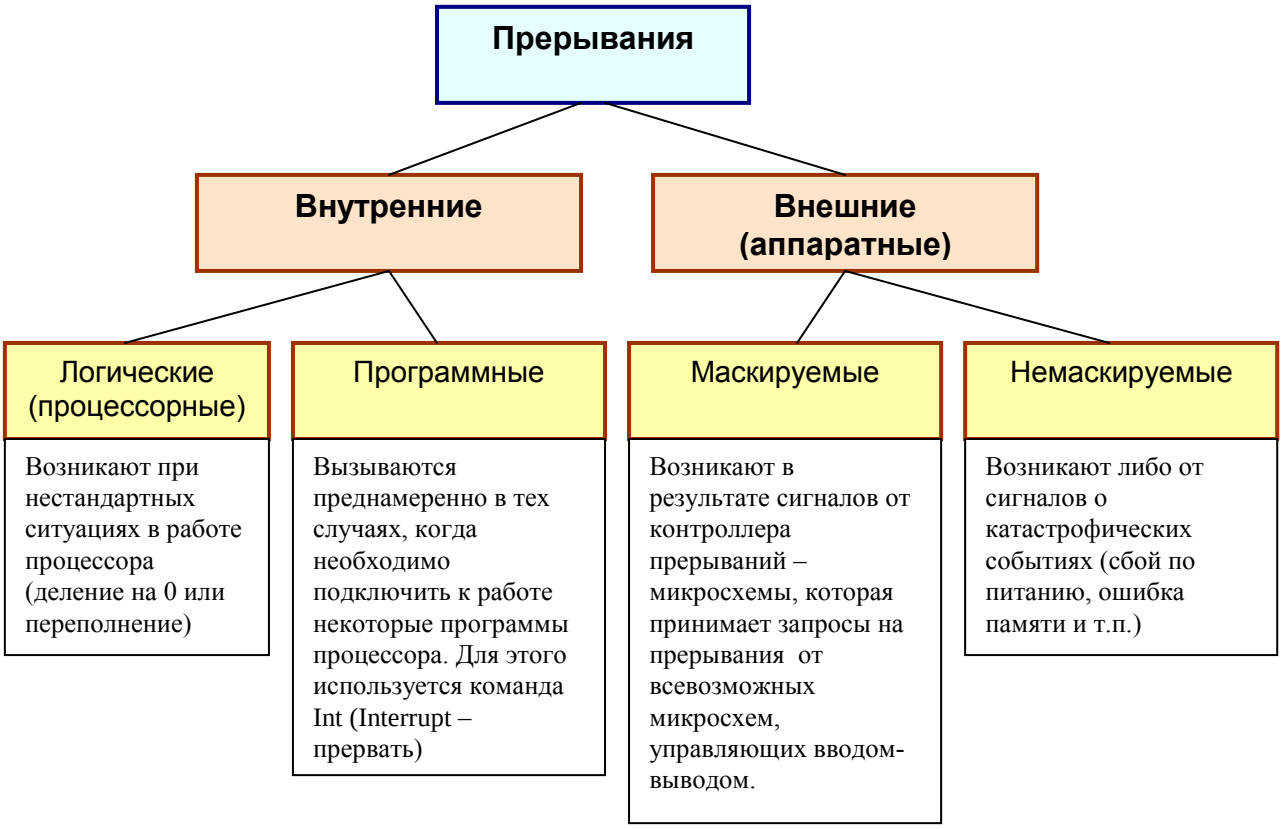
Архитектура – это обобщенное представление системы с точки зрения существующих в ней информационных потоков и способов их обработки.

Архитектура процессора характеризуется:

1. Структура адресного пространства.
2. Состав, имена и назначение программно-доступных регистров.
3. Классы прерываний, особенности инициирования и обработки прерываний.
4. Способы машинного представления (форматы) данных, обрабатываемых процессором.
5. Способы машинного представления (форматы) команд и режимы адресации.
6. Логическая структура и исполнительный цикл процессора.
7. Система команд.

Система прерываний

Прерывание – это прекращение выполнения текущей последовательности команд для обработки некоторого события, происшедшего либо в самом процессоре, либо вне его.



Механизм обработки прерываний

Каждому прерыванию присвоен уникальный номер, а с каждым номером связана вполне определенная программа – **программа обработки прерывания**.

Все программы обработки прерываний постоянно находятся в оперативной памяти, и в любой момент каждая из них может быть запущена.

Адрес оперативной памяти, где находится программа обработки прерывания, определяется по **таблице векторов прерываний**, которая постоянно находится в оперативной памяти.

Исполнительный цикл процессора

В 1978 году появился процессор Intel 8086. В архитектуре процессора было реализовано нескольких новых технических решений. Одно из них связано с организацией исполнительного цикла.

В традиционных процессорах:

1. Выбор очередной машинной команды из основной памяти по адресу, указанному в регистре счетчика команд (IP).
2. Декодирование команды. Чтение операндов из основной памяти (если они нужны).
3. Выполнение команды, включая запись результата в память.
4. Изменение значения счетчика.

Откуда процессор знает, какие именно операции нужно выполнять?

Процессор извлекает данные из памяти, и эти данные указывают ему, что нужно делать. Такие данные называются *инструкции*.

Откуда процессор знает, какую инструкцию нужно выполнить следующей?

С помощью внутреннего указателя, который указывает на то место в памяти, где хранится значение следующей выполняемой инструкции. Когда из памяти считывается и выполняется следующая инструкция, указатель перемещается на следующую инструкцию.

Команды микропроцессора

- **Команды перемещения данных** Перемещение данных между памятью, регистрами и внешними устройствами.

- **Команды преобразования**. Выполняют действия сложения, вычитания, умножения, арифметический сдвиг, конъюнкцию и т.д.

Общий принцип программирования микропроцессоров:

1. Поместить данные из памяти или внешнего устройства в регистр.

2. Выполнить преобразование данных.

3. Записать результат в память или вывести на внешнее устройство.

Структура адресного пространства

Адресное пространство процессора это множество ячеек основной памяти, к которым может обращаться процессор для записи туда информации или для чтения информации оттуда.

Основная память компьютера представляет собой *электронное устройство*, состоящее из большого числа двоичных запоминающих элементов, а также схем управления ими.

Байт - это наименьшая адресуемая ячейка памяти.

Разряды байта нумеруются справа налево от 0 до 7:

7	6	5	4	3	2	1	0

старшие разряды *младшие разряды*

Слово (word) – это два соседних байта. Размер слова - 16 разрядов. Они нумеруются, если рассматривать слово как единое целое, справа от 0 до 15. Адресом слова считается по определению адрес первого байта (с меньшим адресом).

A		**A + 1**	
15	8	7	0

Старший байт Младший байт

Двойное слово (double word) - это четыре соседних байта или, что тоже самое, два соседних слова. Размер двойного слова - 32 разряда, они нумеруются справа налево от 0 до 31. Адрес двойного слова - адрес первого из его байтов (с наименьшим адресом).

A		**A+1**		**A+2**		**A+3**	
31	24	23	16	15	8	7	0

Что такое ассемблер?

В **1949 году** была введена в эксплуатацию английская машина с хранимой программой EDSAC - конструктор Морис Уилкс из Кембриджского университета. Эта машина содержала 3000 электронных ламп и в 6 раз была производительней своих предшественниц. В этой машине Морис Уилкс ввел впервые систему мнемонических обозначений для машинных команд - *ассемблер*. С годами ассемблер превратился в самостоятельный язык программирования.

Ассемблер - язык программирования, название которого происходит от английского слова *assemble* (сборка).

Программы для машин первого поколения представляли собой непосредственно *машинный код*. Программирование в машинных кодах очень трудоемко, так как многие машинные коды зависят от относительного положения в памяти. Это означает, что если вы изменили одну команду в программе на машинном коде, то вам придется модифицировать кроме того еще и другие.

Для того чтобы упростить программирование придумали символическое обозначение. Программы для перевода таких команд в машинные коды и получили название ассемблеров. Позже ассемблером стали называть набор команд процессора, а в настоящее время это полноценный язык программирования, позволяющий использовать переменные, процедуры, метки, макросы и многое другое.

Программы на языке ассемблер переводятся в машинный код с помощью программы-транслятора. Основной отличительной особенностью языка ассемблера от языков высокого уровня состоит в том, что операторы C и Pascal обычно переводятся в целые наборы машинных кодов, а команды ассемблера непосредственно преобразуются в соответствующий машинный код.

Когда следует использовать ассемблер?

1. Обеспечение максимальной скорости выполнения некоторых функций.
Максимальная скорость обеспечивается за счет того, что в стандартных операторах высокоуровневых языков выполняется значительное количество подготовительных действий.

2. Управление нестандартными устройствами.
Управление нестандартными устройствами осуществляется обычно на языке ассемблера, так как развитие внешних оболочек (операционных сред, драйверов и т.д.) идет значительно медленнее, чем развитие самих устройств.

Пример

Инициализировать режим графики, поставить точку в центре экрана, дождаться нажатия на любую клавишу, закрыть режим графики и завершить программу.

```
USES Graph,Crt;
VAR d,m :Integer;
     BEGIN
          d:=0;
          Initgraph (d, m, 'c:\bp\bgi');

          PutPixel (320, 240, White);

          Repeat   Until KeyPressed;
          CloseGraph
     END.
```

Пример

та же программа, но на ассемблере

```
  DOSSEG              ; Включение стандартного порядка сегментов
  MODEL SMALL         ; Выбор модели памяти
  STACK 100h          ; Резервирует пространство для стека программы
                      ; 100h - 256 байтов под сегмент стека
  CODESEG             ; Начало программы
  mov     ax,@Data    ; Установить в DS адрес
  mov     ds,ax       ; сегмента данных DOSSEG

  mov     ah,0
  mov     al,12h      ; Устанавливает режим графики
  int     10h         ; Прерывание 10h отвечает за видеосистему

  mov     ah,0ch
  mov     al,15
  mov     bh,0
  mov     cx,320
  mov     dx,240
  int     10h         ; Ставит точку на экране

  mov     ah,0
  int     16h         ; Прерывание 16h управляет клавиатурным вводом

  mov     ah,0
  mov     al,3
  int     10h         ; Устанавливает текстовый режим

  mov     ah,4ch      ; Функция DOS: выход из программы
  int     21h         ; Вызов DOS. Останов программы
  END
```

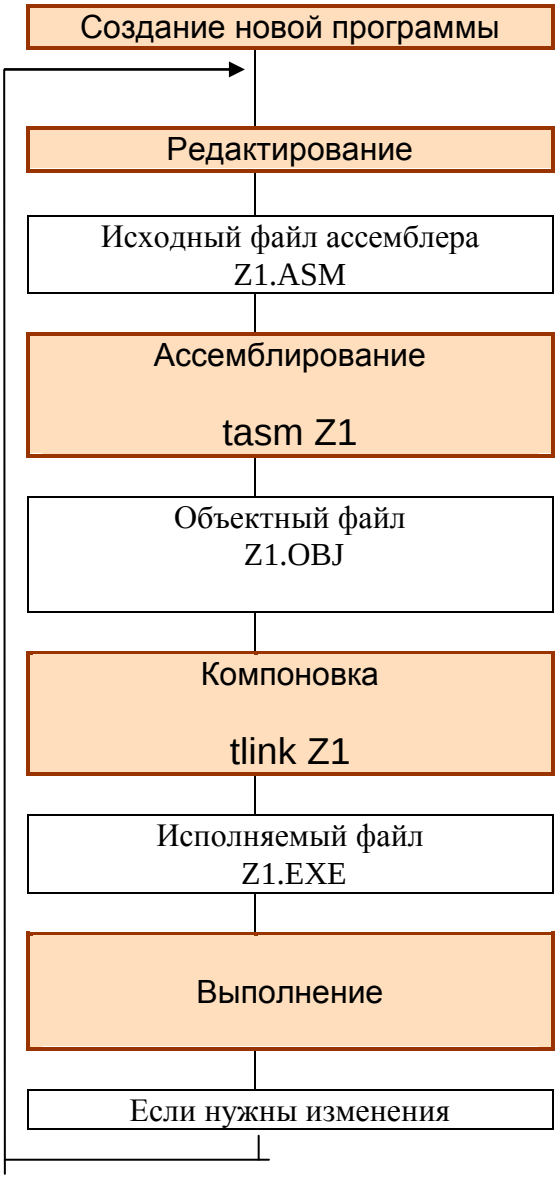
Результаты создания исполняемых файлов на Pascal и Assembler

Том в устройстве C не имеет метки
Серийный номер тома: 1655-0CCE
Содержимое каталога C:\ELENA\ASM\SAMPL

.	<КАТАЛОГ>		02.01.97	5:24 .
..	<КАТАЛОГ>		02.01.97	5:24 ..
PAS1	PAS	166	02.01.97	4:12
Z1	**EXE**	**551**	**02.01.97**	**1:59**
PAS1	**EXE**	**20 448**	**02.01.97**	**4:13**
Z1	ASM	326	02.01.97	1:59

4 файл(а,ов) 21 491 байт
2 каталог(а,ов) 281 411 584 байт свободно

Создание исполняемой программы



считывает файл Z1.ASM и, если программа введена правильно, создает файл Z1.OBJ

Содержит объектный код, еще не готовый для непосредственного исполнения

Считывает Z1.OBJ и создает исполняемый файл Z1.EXE

Это позволяет разделить большую программу на небольшие фрагменты, ассемблировать их, создавая отдельные объектные файлы, а затем скомпоновать с помощью одной команды.

Отдельные фрагменты могут иметь общие данные и вызывать подпрограммы, описанные в других модулях.

Анализ программы

Для того чтобы посмотреть машинные коды, получившиеся в результате компиляции, необходимо выполнить компиляцию с ключом L.

tasm /L Z1.asm

В результате будет создан файл Z1.LST, в котором приведена вся информация о транслируемой программе.

Некоторым командам (например, STACK) не соответствуют машинные коды. Эти команды управляют работой компилятора.

В конце листинга приведены сегменты программы. Они получили имена STACK, _DATA, _TEXT и объединены в группу DGROUP.

```
Turbo Assembler  Version 2.0          02/01/97 08:31:39          Page 1
asm1.asm

   1                                    DOSSEG
   2    0000                            MODEL SMALL
   3    0000                            STACK 100h
   4    0000                            CODESEG
   5    0000  B8 0000s                  mov  ax,@Data
   6    0003  8E D8                     mov  ds,ax
   7    0005  B4 00                     mov  ah,0
   8    0007  B0 12                     mov  al,12h
   9    0009  CD 10                     int  10h
  10    000B  B4 0C                     mov  ah,0ch
  11    000D  B0 0F                     mov  al,15
  12    000F  B7 00                     mov  bh,0
  13    0011  B9 0140                   mov  cx,320
  14    0014  BA 00F0                   mov  dx,240
  15    0017  CD 10                     int  10h
  16    0019  B4 00                     mov  ah,0
  17    001B  CD 16                     int  16h
  18    001D  B4 00                     mov  ah,0
  19    001F  B0 03                     mov  al,3
  20    0021  CD 10                     int  10h
  21    0023  B4 4C                     mov  ah,4ch
  22    0025  CD 21                     int  21h
  23                                    END
```

```
Turbo Assembler  Version 2.0          02/01/97 08:31:39          Page 2
Symbol Table

Symbol Name                Type     Value

??DATE                     Text     "02/01/97"
??FILENAME                 Text     "asm1      "
??TIME                     Text     "08:31:39"
??VERSION                  Number 0200
@CODE                      Text     _TEXT
@CODESIZE                  Text     0
@CPU                       Text     0101H
@CURSEG                    Text     _TEXT
@DATA                      Text     DGROUP
@DATASIZE                  Text     0
@FILENAME                  Text     ASM1
@MODEL                     Text     2
@WORDSIZE                  Text     2

Groups & Segments          Bit Size Align  Combine Class

DGROUP                                     Group
  STACK                    16  0100 Para     Stack     STACK
  _DATA                    16  0000 Word     Public    DATA
_TEXT                      16  0027 Word     Public    CODE
```

Обработка ошибок

Существуют основные два вида ошибок:

1. Ошибки являются критичными для программы. Даже если будет создан объектный код, он не будет компоноваться и исполняться, т.е. TLINK не скомпонует его в исполняемый файл.

2. Предупреждения не являются критичными. Результирующий объектный код, вероятно, будет компоноваться, но исполняться он может некорректно.

Если Turbo Assembler находит ошибку, он выводит сообщение об этом после соответствующего номера строки, взятого в скобки.

Перенаправление результатов ассемблирования:

```
tasm  Z1.asm > err.txt

tasm  Z1.asm > prn
```

Пример сообщений об ошибках:

Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file: Z1.asm	*ассемблирован файл Z1.asm*
Error asm1.asm(2) Illegal instruction	*сообщения об ошибках*
Error messages: 1	
Warning messages: None	*предупреждающие сообщения*
Passes: 1	
Remaining memory: 425k	*остается памяти*

Внимание!

Одна ошибка может повлечь за собой сообщения об ошибках в других строках.