|  |  |  |
| --- | --- | --- |
| Кашаев  Марат  Хамзяевич | Лабораторная работа №1  Основы ассемблера |  |
| ПРИ-141 |

**Цель:** Ознакомиться с основами программирования с использованием ассемблерных вставок.

**Теоретическая часть:**

**Кратко о регистрах**

Микропроцессоры Intel включают регистры:

* общего назначения
* флаги
* сегментные
* управляющие
* системные адресные
* отладочные

Особо можно отметить регистр EIP. Содержит величину смещения относительно начала программы на линию кода, которая будет исполняться следующей.

Таблицы рабочих регистров

|  |  |  |  |
| --- | --- | --- | --- |
| EAX (32bit) | | | |
|  | | AX (16bit) | |
|  |  | AH (8bit) | AL (8bit) |

EAX (Accumulator): EAX = AX + AH + AL

|  |  |  |
| --- | --- | --- |
| EBX (32bit) | | |
|  | BX (16bit) | |
|  | BH (8bit) | BL (8bit) |

EBX (Base): EBX = BX + BH + BL

|  |  |  |
| --- | --- | --- |
| ECX (32bit) | | |
|  | CX (16bit) | |
|  | CH (8bit) | CL (8bit) |

ECX (Counter): ECX = CX + CH + CL

|  |  |  |
| --- | --- | --- |
| EDX (32bit) | | |
|  | DX (16bit) | |
|  | DH (8bit) | DL (8bit) |

EDX (Data): EDX = DX + DH + DL

**Операции**

\_\_asm {

**mov** **eax,** 3; //Константу в eax

**mov** **ebx,** **eax**; //eax в ebx

**mov** **eax,** 4; //Константу в eax

**add** **eax,** **ebx**; // eax = eax + ebx

**sub** **eax,** **ebx**; // eax = eax - ebx

**inc** **eax**; //eax = eax + 1

**dec** **eax**; //eax = eax - 1

}

**Умножение**

Для умножения без знака предназначена команда MUL. У этой команды один операнд – второй множитель, который должен находится в регистре или в памяти. Место первого и результат задаётся неявно и зависит от размера операнда:

|  |  |  |
| --- | --- | --- |
| Результат | Множитель | Размер операнда |
| AX (16bit) | AL (8bit) | Byte |
| DX:AX | AX(16bit) | Word |

**mul** **bl** ;AX = AL \* BL

**mul** **ax** ;DX:AX = AX \* AX

Разрядность результата получается в 2 раза больше разрядности операндов. Запись DX:AX означает, что старшее слово результата будет находится в DX, а младшее в AX.

**Умножение со знаком**

Для умножения чисел со знаком предназначена команда IMUL. Эта команда имеет три формы, различающиеся количеством операндов:

* С одним операндом — форма, аналогичная команде MUL. В качестве операнда указывается множитель. Местоположение другого множителя и результата определяется по таблице.
* С двумя операндами — указываются два множителя. Результат записывается на место первого множителя. Старшая часть результата в этом случае игнорируется. Кстати, эта форма команды не работает с операндами размером 1 байт.
* С тремя операндами — указывается положение результата, первого и второго множителя. Второй множитель должен быть непосредственным значением. Результат имеет такой же размер, как первый множитель, старшая часть результата игнорируется. Это форма тоже не работает с однобайтными множителями.

**imul** **cl** ;AX = AL \* CL

**imul** **si** ;DX:AX = AX \* SI

**imul** **bx,ax** ;BX = BX \* AX

**imul** **cx,-**5 ;CX = CX \* (-5)

**imul** **dx,bx,**134h ;DX = BX \* 134h

*\*CF = OF = 0, если произведение помещается в младшей половине результата, иначе CF = OF = 1. Для второй и третьей формы команды CF = OF = 1 означает, что произошло переполнение.*

**Деление чисел без знака**

Деление целых двоичных чисел — это всегда деление с остатком! По аналогии с умножением, размер делителя, частного и остатка должен быть в 2 раза меньше размера делимого. Деление чисел без знака осуществляется с помощью команды DIV. У этой команды один операнд — делитель, который должен находиться в регистре или в памяти. Местоположение делимого, частного и остатка задаётся неявно и зависит от размера операнда:

|  |  |  |  |
| --- | --- | --- | --- |
| Частное | Остаток | Делимое | Размер делителя |
| AL | AH | AX | Byte |
| AX | DX | DX:AX | Word |

При выполнении команды DIV может возникнуть прерывание:

* Если делитель равен нулю;
* Если частное не помещается в отведённую под него разрядную сетку (например, если при делении слова на байт частное больше 255).

**div** **cl** ;AL = AX / CL, остаток в AH

**div** **di** ;AX = DX:AX / DI, остаток в DX

**Деление чисел со знаком**

Для деления чисел со знаком предназначена команда IDIV. Единственным операндом является делитель. Местоположение делимого и частного определяется также, как для команды DIV. Эта команда тоже генерирует прерывание при делении на ноль или слишком большом частном.

*Перевод исходного файла в ассемблерный код:*

*\*gcc: src.c – S*

*\*intel: cl /FAs src.c*

|  |  |  |
| --- | --- | --- |
| Кашаев  Марат  Хамзяевич | Лабораторная работа №2  Вещественные числа и операции над ними |  |
| ПРИ-141 |

**Цель**: Получить представление о представлении числе с плавающей запятой.

# **Теоретическая часть:**

Представление десятичных чисел с плавающей запятой в двоичной форме выглядит следующим образом:

При 32 битовом представлении, первый бит отдаются под знак, следующие 8 знаков отдается под порядок, причем нуль смещён в 127, оставшаяся часть (23 бита) отдана под мантиссу.

Обобщенный алгоритм преобразования вещественного десятичного числа в двоичное число с плавающей точкой формата IEEE:

1. Представить целую часть вещественного числа в двоичном виде
2. Преобразовать дробную часть в двоичный формат
3. Записать полученную дробную часть после десятично точки. Если мантисса меньше выделенного места, заполнить разряды «вправо» нулями
4. Нормализовать полученное двоичное число, определив значение показателя степени
5. Сдвинуть показатель степени и представить в двоичном виде
6. Записать значение характеристики в соответствующие биты формата перед нормализованной мантиссой, отбросив единицу целой части
7. Положить 0/1 в качестве знака в старший разряд

Другие форматы данных, обрабатываемые сопроцессором:

* Целые числа
  + Двоичные 16, 21, 64 бита
  + Упакованные целые числа (BCD) – числа максимальной длина 18 упакованные десятичных цифр (9 байт)
* Специальные числовые значения
  + Деморализованные вещественные числа
  + Нуль
  + Положительные и отрицательные значения бесконечности
  + Нечисла
  + Неопределенности и неподдерживаемые форматы

Для представления специальных значений зарезервированы минимальная 000…00 и максимальная 111…11 характеристики.

**Нуль** представляется с нулевой характеристикой и нулевой мантиссой: (32 нуля)

**Деморализованные вещественные числа** – числа на границе с нулем. Имеют нулевую характеристику и ненулевую мантиссу

**Бесконечности** – кодируются с максимальным значением характеристики и мантиссой равной единице

**Нечисла** – кодируются с максимальным значением характеристики и неединичной мантиссой

**Неопределенность –** результат недействительной операции

**Неподдерживаемые форматы –** остальные битовые наборы

**Программная модель сопроцессора**

С точки зрения программиста представляет собой совокупность регистров, каждый из которых имеет свое функциональное назначение. В программной модели сопроцессора можно выделить три группы регистров:

* Регистры стека
* Служебные регистры
* Регистры данных

**Регистры стека**

Физические номера регистров R0 – R7. Сопроцессор оперирует логическими номерами регистров St(0) – St(1):

1 бит – знак, 15 бит – порядок, 64 бит – мантисса.

**Служебные регистры**

**Status Word Register (SWR, Регистр состояния сопроцессора)** – отражает текущее состояние сопроцессора после выполнения команды.

1) 6 флагов исключительных ситуаций:

* Бит 0 – недействительная операция (IE);
* Бит 1 – ненормализованный операнд (DE);
* Бит 2 – деление на ноль (ZE);
* Бит 3 – переполнение (OE);
* Бит 4 – антипереполнение (UE);
* Бит 5 – потеря точности (PE);

2) Бит sf (Stack Fault) – ошибка работы стека сопроцессора

* SF=1 C1 = 1 – переполнение стека
* SF=1 C1 = 0 – чтение из пустого стека

3) Бит es (Error Summary) – суммарная ошибка сопроцессора. Устанавливается в единицу, если возникает любая из шести перечисленных выше ситуаций.

4) 4 бита C0 – C3 Condition Code (коды условия) – отражают результат выполнения операции. Интерпретация битов: бит 8 – C0 (CF); бит 9 – C1; бит 10 – C2 (PF); бит 14 – С3 (ZF).

5) Биты 11 – 13 – трех-битовое поле top – содержит указатель регистра текущей вершины стека.

6) Бит 15 – бит занятости сопроцессора (B).

**Control Word Register (CWR, Регистр управления)** – определяет особенности обработки численных данных и состоит из:

1) шести масок исключений – предназначены для исключений. Если какая-то из шести масок установлена в 1, то это исключение будет обрабатываться самим сопроцессором. Если 0, то при возникновении соответствующего исключения будет возбуждено прерывание. Биты 0...5 – маска особых случаев – используется для запрета прерывания в случае, если:

* бит 0 –недействительная операция (IM=0);
* бит 1 –ненормализованный операнд (DM=1)
* бит 2 – деление ненулевого числа на ноль (ZM=1);
* бит 3 –переполнение (OM=1);
* бит 4 – антипереполнение (исчезновение порядка UM=1);
* бит 5 – потеря точности (PM=1).

б) поле управления точностью (PC) – предназначено для выбора длины мантиссы. Биты 8-9 – 00 – 24 бита; 10 – 53 бита; 11 – 64 бита. По умолчанию значение поля устанавливается pc=11;

в) поля управления округлением (RC) – позволяет управлять процессом округления чисел во время работы сопроцессора. Необходимость округления может возникнуть, если в процессе выполнения команды получается непредставимый результат, например, периодическая дробь. Установка поля округления позволит выполнить округление в нужную сторону. Биты 10-11 задают режим округления.

г) бит 12 – интерпретация бесконечности.

**Tag Word Register (TWR, Регистр тегов)** – используется для контроля состояния каждого из регистров R0 – R7. Регистр отводит 2 бита на каждый регистр стека:

* 00 – в регистре не нулевое действительное число;
* 01 – в регистре истинный ноль;
* 10 – в регистре не число или бесконечность;
* 11 – регистр пуст.

**Регистры указателей**

Регистр указатель команды **ipr (Instruction Point Register)** содержит адрес команды, вызвавшей особый случай, и 11 бит команды.

Регистр указатель данных **dpr (Data Point Register)** содержит адрес операнда команды, вызвавшей особый случай.

**Система команд сопроцессора**

Мнемоническое обозначение команд:

* первая буква всегда F
* вторая буква определяет тип операнда в памяти
  + I – операции с целыми числами
  + B – операции с десятичными числами
  + При отсутствии, операции выполняются с вещественными числами
* предпоследняя или последняя R (reverse) указывает обратный порядок для операций вычитания или деления
* последняя буква P идентифицирует команду, последним действием которой ­– извлечение из стека.

**Команды передачи данных**

Предназначены для организации обмена между регистрами стека, вершиной стека и ячейками памяти. Для каждого из трех типов данных определена своя группа команд. Главная функция – преобразование в единое представление в виде расширенного вещественного числа. Все операции сохранения данных в память выполняют обратное преобразование.

**1. Команды загрузки чисел в ST(0):**

FLD <источник> – вещ. числе из памяти или стека

FILD – целых числе из памяти

FBLD – десятичных числе из памяти

**2. Команды сохранения без извлечения из стека**

FST <приемник> – копирует вещ. число из ST(0) в память

FIST – копирует целое число из ST(0) в память

**3. Команды записи в память с извлечением из стека**

FSTP <приемник> ­– перемещает знач. ST(0) в mem (вещ.)

FISTP – перемещь. знач. ST(0) в mem (целое)

FBSTP – перемещь. знач. ST(0) в mem (десятич.)

**4. Команда обмена**

FXCH [<приемник>] – меняется местами ST(0) и приемник, если приемник не указан, то меняются местами ST(0) и ST(1)

**5. Команды загрузки констант**

FLDS – загрузка нуля

FLD1 – загрузка единицы

FLDPI – загрузка pi

FLDL2T – загрузка двоичного логарифма десяти

FLDLG2 – загрузка десятичного логарифма двух

FLDLN2 – загрузка натурального логарифма двух

**Арифметические команды**

Форматы основных арифметических команд:

Fop [ST(1), ST] – стековая (всегда с извлечением P)

Fop ST(i), ST или Fop ST, ST(i) – регистровая

FopP ST(i), ST – регистровая с извлечением

Fop [ST,] <операнд2> – вещественный операнд в памяти

Fop [ST,] <операнд2> – целочисленный операнд в памяти

где op = ADD <операнд1> = <операнд1> + <операнд2>

SUB <операнд1> = <операнд1> – <операнд2>

SUBR <операнд1> = <операнд2> – <операнд1>

MUL <операнд1> = <операнд1> \* <операнд2>

DIV <операнд1> = <операнд1> / <операнд2>

DIVR <операнд1> = <операнд2> / <операнд1>

Дополнительные арифметические команды

FSQRT – извлечение квадратного корня

FSCALE – масштабирование ST(0) ←ST(0) \* 2ST(1)

FPREM – вычисляет частичный остаток ST(0) ← ST(0) - q\*ST(1), q – целая часть результат ST(0) / ST(1)

FPNDINT – округление до целого

FXTRACT – расцепляет число на порядок, который заменяет число в ST(1) и мантиссу, которая помещается в ST(0)

FABS – получение модуля числа

FCHS – изменение знака числа

**Команды сравнения**

Команды этой группы выполняют сравнения значений числа в вершине стека и операнда, указанного в команде.

FCOM [<операнд>] – сравнение чисел

FCOMP [<операнд>] – сравнение чисел и одно извлечение

FCOMPP [<операнд>] – сравнение чисел и два извлечения

FICOM <операнд> – сравнение c целым числом

FICOMP <операнд> – сравнение c целым и извлечение

FUCOM <регистр> – сравнение c регистром

FUCOMP <регистр> – сравнение c регистром и извлечение

FUCOMPP <регистр> – сравнение c рег. и два извлечения

FTST – сравнение с нулем и замена источника на нуль

FXAM – анализ операнда

В результате работы, в регистре состояния устанавливаются значения битов кодов условия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Условие | С3 | С2 | С1 | С0 |
| ST(0) > <операнда2> или ST(1) | 0 | 0 | X | 0 |
| ST(0) < <операнда2> или ST(1) | 0 | 0 | X | 1 |
| ST(0) = <операнда2> или ST(1) | 1 | 0 | X | 0 |
| Не сравнимы | 1 | 1 | X | 1 |

Чтобы передать результаты из FPU в CPU нужно записать биты условия в регистр процессора eflags.

Для этого:

1. FSTSW AX – сохранить слово состояния в регистре AX

2. Значения нужных бит извлекаются и анализируется основным процессором. Осуществимо двумя способами:

SAHF – копирует C3 в ZF, С2 в PF, C0 в CF

TEST AX, <константа кода>

|  |  |  |  |
| --- | --- | --- | --- |
| Результат | Константа | Переход | B=TOP?0<[X]\*8 |
| ST > операнд | 4500h | JZ | 01000101 [0]\*8 |
| ST < операнд | 0100h | JNZ | 00000001 [0]\*8 |
| ST = операнд | 4000h | JNZ | 01000000 [0]\*8 |
| Не сравнимы | 0400h | JNE | 00000100 [0]\*8 |

**Команды трансцендентных функций**

\*Значения всех операндов тригонометрических функций задаются в радианах.

**1. Команды тригонометрические**

FPTAN – ST(1) = tg(ST), ST [-263, 264]

FPATAN – ST = arctg(ST(1)/ST(0)), операнды извлекаются

FSIN – ST = sin(ST), где ST [-263, 264]

FCOS – ST = con(ST), где ST [-263, 264]

FSINCOS – ST(1) = sin(ST), где ST[-263, 264], ST = cos(ST)

**2. Команды логарифмически и показательные**

F2XM1 – ST = 2ST-1, где ST [-1, 1]

FYL2X – ST = ST(1) \* log2ST, где ST > 0

FYL2XP1 – ST = ST(1) \* log2(ST+1),

где ST

**Команды управления сопроцессором**

Предназначены для управления сопроцессором. Могут начинаться с FN (без ожидания и без проверки особых случаев) или F (с ожиданием).

FNSTCW (FSTW) – записать содержимое CW в ram

FLDCW – загрузить CW из ram

FNSTSW (FSTSW) – записать SW в ram

FNSTSW AX (FSTSW AX) – записать CW в AX

FNCLEX (FCLEX) – сбросить флаги особых случаев в SW и биты ES, B

FNINIT (FINIT) – инициализировать сопроцессор:

* управляющий регистр – бесконечность со знаком, округление к ближайшему, расширенная точность, все особые случаи замаскированы
* регистр состояния – B=0 (бит занятости сброшен), код условия не определен, ST=ES=0, флаги особых случаев установлены в нуль
* регистр тегов – все поля регистра тегов содержат значение 11 (пустой регистр))

FNSTENV (FSTENV) – записать в память среду (содержимое всех регистров, кроме численных, в предопределенном формате)

FLDENV – загрузить среду

FNSAVE (FSAVE) – записать полное состояние (дополнительно сохраняет содержимое численных регистров)

FRSTOR – восстановить полное состояние

FINCSTP – увеличить указатель стека TOP на 1

FDECSTP – уменьшить указатель стека TOP на 1

FFREE – освободить регистр

FNOP – нет операции (не производит никаких действий)

FSETPM – установить защищенный режим работы (переводит сопроцессор в защищенный режим работы)

|  |  |  |
| --- | --- | --- |
| Кашаев  Марат  Хамзяевич | Лабораторная работа №3  Операторы выбора. Условные переходы и ветвления |  |
| ПРИ-141 |

**Цель**: Получить представление о реализации операторов выбора на уровне ассемблера.

# **Теоретическая часть:**

JMP ­– передача управления в другую точку, не сохраняя информацию для возврата.

Существуют:

* short (переход) – адрес в пределах [-127, 128]
* near (переход) – перехода в том же сегменте, что JMP
* far (переход) – адрес в другом сегменте
* переход с переключением задач

По сути JMP модифицирует IP

|  |  |  |
| --- | --- | --- |
| Команда | Переход, если | Условие |
| JZ/JE | нуль или равно | ZF=1 |
| JNZ/JNE | не нуль или не равно | ZF=0 |
| JC/JNAE/JB | есть переполнение/не выше и не равно/ниже | CF=1 |
| JNC/JAE/JNB | нет переполнения/выше или равно/не ниже | CF=0 |
| JP | число единичных бит чётное | PF=1 |
| JNP | число единичных бит нечетное | PF=0 |
| JS | знак равен 1 | SF=1 |
| JNS | знак равен 0 | SF=0 |
| JO | есть переполнение | OF=1 |
| JNO | нет переполнения | OF=0 |
| JA/JNBE | выше / не ниже и не равно | CF=0 ZF=0 |
| JNA/JBE | не выше / ниже или равно | CF=1 ZF=1 |
| JG/JNLE | больше / не меньше и не равно | ZF=0 SF=OF |
| JGE | больше или равно / не меньше | SF=OF |
| JL/JNGE | меньше / не больше и не равно | SF!=OF |
| JLE/JNG | меньше или равно / не больше | ZF=1 SF!=OF |
| JCXZ | содержимое CX равно нулю | CX=0 |

Сравнения «выше» и «ниже» относятся к беззнаковым числам, а «больше» и «меньше» — к числам со знаком. Для беззнаковых чисел признаком переполнения будет флаг CF, а соответствующими командами перехода JC и JNC. Для чисел со знаком о переполнении можно судить по состоянию флага OF, поэтому им соответствуют команды перехода JO и JNO. Команды переходов не изменяют значения флагов.

BOUND <индекс> <границы> - проверяет, не выходит ли значение первого операнда (регистр), взятое как число со знаком, за границы, указанные во втором операнде (переменная).

**Логические операции**

|  |  |
| --- | --- |
| Команда | Назначение |
| and приемник, источник | и |
| or приемник, источник | или |
| xor приемник, источник | исключающее или |
| not приемник | инверсия |
| test приемник, источник | логическое сравнение |

|  |  |  |
| --- | --- | --- |
| Кашаев  Марат  Хамзяевич | Лабораторная работа №4  Функции, определенные программистом. |  |
| ПРИ-141 |

**Цель**: Получить практику реализации функций и процедур на уровне ассемблера.

# **Теоретическая часть:**

  Обычно процедуры предназначены для выполнения каких-то отдельных, законченных действий программы и поэтому их иногда называют подпрограммами. Для работы с процедурами предназначены команды CALL и RET. С помощью команды CALL выполняется вызов процедуры. Эта команда работает почти также, как команда безусловного перехода (JMP), но с одним отличием — одновременно в стек сохраняется текущее значение регистра IP. Это позволяет потом вернуться к тому месту в коде, откуда была вызвана процедура. В качестве операнда указывается адрес перехода, который может быть непосредственным значением (меткой), регистром (кроме сегментных) или ячейкой памяти, содержащей адрес.

Возврат из процедуры выполняется командой RET. Эта команда восстанавливает значение из вершины стека в регистр IP.

Передача параметров с процедуру:

По значению: mov ax, word ptr value

По ссылке: mov ax, offset value

Держать в регистрах

Через стек:

push par1

push par2

call proc

add sp, 4

push bp

mov bp, sp

mov ax [bp+4]

mov bx [bp+8]

|  |  |  |
| --- | --- | --- |
| Кашаев  Марат  Хамзяевич | Лабораторная работа №5  Циклы. |  |
| ПРИ-141 |

**Цель**: Получить представление о реализации циклов на уровне ассемблера.

# **Теоретическая часть:**

Циклы - повторяющееся выполнение последовательных команд. Для создания цикла необходимо знание использование меток. Вместо меток компилятор везде будет подставлять адрес текущей команды.

label: mov ax, 1024

Команда LOOP

У этой команды один операнд - имя метки, на которую осуществляется переход. В качестве счетчика используется регистр CX. Команда LOOP выполняет декремент CX, а затем проверяет его значение. Если содержимое CX не равно нулю, то осуществляется переход на метку, иначе управление переходит к следующей после LOOP команде.

При необходимости вложенного цикла необходимо сохранить значение CX перед началом вложенного и восстановить после его завершения.

mov cx, 16

lp1:

mov bx, cx

mov cx, 16

lp2:

mov cx, bx

loop lp1

|  |  |  |
| --- | --- | --- |
| Кашаев  Марат  Хамзяевич | Лабораторная работа №6  Одномерные массивы. |  |
| ПРИ-141 |

**Цель**: Получить практику работы с одномерными массивами на уровне ассемблера.

# **Теоретическая часть:**

Массив (в некоторых языках программирования также таблица, ряд, матрица) — тип или структура данных в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом.

Для одномерных(1D) числовых массивов есть некий набор операций:

1.инициализировать массив (ввести с клавиатуры, случайными числами, копированием из другого массива,

константой, функцией)

2. Найти сумму (произведение)массива (массивов)

3. Найти минимум(максимум) массива

4.найти элемент в массиве (последовательный, бинарный поиск)

найти элемент в массиве удовлетворяющий условию

5. Скалярное произведение массивов

6. Сортировка массивов.

7. Сравнение массивов.

Большая часть учебных задач на начальном этапе обычно сводится к комбинации данных шаблонов(алгоритмов).

**Строки**

Работа со строками в основном аналогична работе с массивами, если не добиваться максимально коротких или быстрых программ.

Есть множество интереснейших алгоритмов (Кнут, Ахо-Ульман и др.).

Использование строковых команд MOVS, SCANS, CMPS, LODS, STOS элегантно, но

1) требует аккуратности и

2) удобно для достаточно длинных (8-16-32 в зависимости от алгоритма) строк

Максимальный размер строки обрабатываемый одной строчной командой == 2 Гб.

Для более коротких строк быстрее и проще написать "массивное" решение (через массивы).

Некоторые алгоритмы - выделение слова, распознавание слова могут быть основаны (или не основаны по Павловской) на КА(конечных автоматах).

Алгоритм выделения слов, распознавания слов - см. Практикум по С. Подбельской. Или Павловская, задача 5.2

Построение/использование - КА - Юров, практикум, гл.2.

**Пример**. Ввести строку, перевернуть ее а затем вывести.

Ассемблерный код основан на особенностях представления строки в С - строка заканчивается символом '\0' (==0 ).

Исходный код из книги после переделки(Dos16--> Win32) Не работал. Заработала 4-я версия., когда от исходного кода осталось мало.