Лабораторная работа 1

Основы программирования на Ассемблере

Цель работы

1.Знакомство со структурой **ПЭВМ IBM PC.**
2.Использование регистров, доступных программисту.
3.Знакомство со структурой программы.
4.Написание элементов программы.

ОБЩИЕ СВЕДЕНИЯ О ПЭВМ

Язык Ассемблер разработан для максимального использования конкретной специфики компьютера. Следовательно, для того, чтобы написать программу на Ассемблере, важно знать архитектуру компьютера.

ВНУТРЕННИЕ РЕГИСТРЫ

Микропроцессор IBM PC содержит три группы регистров данных и адресов, 16-битовый указатель команд **IP (Instruction Pointer)** и 16-битовый регистр флагов.

Регистры работы с данными

К первой группе относятся 4 регистра общего назначения: **AX, BX, CX, DX**. Их можно рассматривать как четыре 16-битовых или восемь 8-битовых регистров. Эти регистры образованы из 8-битовых регистров: регистр **AX** состоит из двух байтовых регистров **AL и AH,** регистр **BX из BL, BH, CX из CL, CH, и DX из DL и DH**. Здесь **L и H** означают младшие (**L**ow-order) и старшие (**H**igh-order) байты 16-битовых регистров. Например, регистры **AL** и **AH** образуют соответственно младший и старший байты регистра **AX**. Всеми этими регистрами можно пользоваться при программировании, но следует учитывать, что ряд команд использует их неявным образом, в частности:

регистр **AX**, аккумулятор (**accumulator**), используется при умножении и делении слов, в операциях ввода-вывода и в некоторых операциях над строками; регистр AL используется при выполнении аналогичных операций над байтами.

регистр **BX**, базовый регистр (**base register**), часто используется при адресации данных в памяти;

регистр **CX**, счетчик (**count register**), используется как счетчик числа повторений цикла и в качестве номера позиции элемента данных при операциях над строками. Регистр **CL** используется как счетчик при операциях сдвига и циклического сдвига на несколько битов;

регистр **DX**, регистр данных (**data register**), используется при умножении и делении слов. Кроме того, в операциях ввода-вывода он используется как номер порта и для хранения номера строки и столбца для выводимого символа.

Регистры сегментов

Оперативная память разделена на сегменты. Процессор может иметь дело одновременно с четырьмя сегментами. Начальные адреса этих сегментов содержатся в его четырех регистрах сегментов. Эти регистры выполняют следующие функции:

регистр сегмента команд **CS (code segment)** указывает на сегмент, содержащий текущую исполняемую программу;

регистр сегмента стека **SS (stack segment**) указывает на текущий сегмент стека;

регистр сегмента данных **DS (data segment)** указывает на текущий сегмент данных;

регистр дополнительного сегмента **ES (extra segment)** указывает на текущий дополнительный сегмент.

Регистры указателей и индексов

Для вычисления адреса команды в сегменте команд процессор извлекает номер сегмента памяти из регистра **CS**, а смещение - из регистра **IP**. Подобным образом за счет выбора номера сегмента из соответствующего регистра сегмента, а смещение - из другого регистра осуществляется доступ к данным других сегментов. Для доступа к сегменту данных микропроцессор извлекает номер сегмента из регистра **DS**, а смещение - из регистра **BX** или индексного регистра (**SI** или **DI**). Для доступа к сегменту стека процессор извлекает номер сегмента из регистра **SS**, а смещение - из регистра указателя (**SP** или **BP**). Выбирая номер сегмента из регистра **ES**, процессор может также получить доступ к дополнительному сегменту.

Флаги

Очень важным является регистр, в котором содержатся признаки последней выполненной операции, именуемый флагами. В 16-битовом регистре флагов размещены девять флагов. Это:

1. флаг переноса **CF (carry flag) (**Бит 0) изменяется во многих операциях;
2. флаг четности **PF (parity flag)** (Бит 2); PF=1, если в результате операции количество битов в состоянии 1 четно.
3. флаг вспомогательного переноса **AF (auxiliary carry flag**). (Бит 4);
4. флаг нуля **ZF (zero flag),** (Бит 6) равен 1, если результат операции равен 0;
5. флаг знака **SF (sign flag)** (Бит 7) SF = 1, если результат отрицательный;
6. флаг трассировки **TF (trap flag),** (Бит 8);
7. флаг прерывания **IF (interrupt enable flag),** (Бит 9) разрешает прерывания от внешних устройств. Если IF=0 прерывания запрещены;
8. флаг направления **DF (direction flag).** (Бит 10);
9. флаг переполнения **OF (overflow flag).** (Бит 11).

Для проверки состояния флагов используются команды условного перехода.

Простейшая программа на Ассемблере

;Первая программа на Ассемблере

```
; Данные программы
DATA   SEGMENT                    ;Начало сегмента данных
  HELLO      DB  'Здравствуйте!$'      ;Данные программы
DATA   ENDS                       ;Конец сегмента данных
;
;Код программы
CODE SEGMENT                      ;Начало сегмента кодов
  ASSUME CS: CODE, DS: DATA
  START:   MOV AX, DATA           ;Занесение адреса сегмента
           MOV  DS, AX            ;данных в регистр сегмента данных
           MOV  AH, 9             ;Функция вывода строки сообщения
           MOV  DX, OFFSET HELLO  ;Смещение HELLO в регистр DX
           INT  21H               ;Вызов программы ввода/вывода
           MOV  AH, 4CH           ;Функция выхода из программы
           INT 21H                ;Вызов программы ввода/вывода
CODE ENDS                         ;Конец сегмента кодов
;
END START          ;последний оператор текста программы с меткой
                   ;начала выполнения программы.
```

Текст программы на Ассемблере можно набрать в редакторе текстов, например, в **NotePad** (в папке **Accessories (Стандартные)**).

Подготовка программы состоит из 3 шагов (Рис. 1):

1. Подготовка исходного текста программы (Например, **HELLO.ASM**)
2. Создание файла **.OBJ**,
3. Создание файла **.EXE**,
4. Выполнение **EXE** файла.

Создание программы

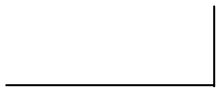

Рисунок 1. Этапы создания программы на Ассемблере

После набора текста программы **HELLO.ASM** в **NotePad** откройте окно **MS-DOS Prompt** и туда, с помощью мышки из окна **Проводника** и папки где находится исходный текст, втащите название файла **TASM** или **MASM** и, после пробела, также втащите туда же название файла с программой. Нажмите ввод. Если ассемблирование пройдет без ошибок, будет создан объектный файл с расширением **.OBJ**. В окне **MS-DOS Prompt** таким же образом запустите редактор связей **TLINK** или **LINK** с названием объектного файла. Будет создан выполняемый файл **.EXE**, который можно запустить на выполнение.

Запустить Ассемблер **MASM** или **TASM** можно также из **Norton Commander** или **FAR.**

ПРАВИЛА ЗАПИСИ ПРОГРАММЫ НА АССЕМБЛЕРЕ

Программа, как правило, состоит из 3 сегментов: сегмента стека, сегмента данных и сегмента кодов (программы).

Каждый сегмент начинается с названия сегмента и ключевого слова **segment.** Вторым оператором в сегменте кодов должен быть записан оператор **assume**, который устанавливает соответствие между регистрами сегментов и названиями сегментов. Сегмент заканчивается оператором с названием сегмента и слова **ends.**

Сегмент кодов обычно содержит основную процедуру и несколько процедур, и для каждой из них записывается название (например, **myproc1**); после названия в начале подпрограммы пишется слово **proc,** а конце - **endp**. Каждая процедура заканчивается словом **ret**.

```
myproc1 proc
        .
        .
ret
myproc1 endp.
```

Данные обычно записываются в специальном сегменте данных, пример которого дан ниже.

```
mydata  segment
      x     db    ?
      y     dw    ?
mydata  ends
```

Программа, состоящая из указанных сегментов и включающая в себя основные директивы, а также некоторые дополнительные конструкции, необходимые для создания работающей программы, представлена на рисунке:

```
;MODEL.ASM
```

```
SSEG SEGMENT STACK    ;Начало сегмента стека
DB 256 DUP (?)        Указана длина стека
SSEG ENDS             ;Конец сегмента стека
;
DSEG SEGMENT          ;Начало сегмента данных
        CATS DB 'A'   ;Ваши данные
        RATS DB 'B'   ;Ваши данные
        BATS DB 'C'   ;Ваши данные
DSEG ENDS             ;Конец сегмента данных
;
CSEG SEGMENT          ;Начала сегмента кодов
ASSUME CS:CSEG,DS:DSEG,SS:SSEG  ; указание названий сегментов регистрам сегментов
START PROC  ;Начало главной процедуры START
PUSH DS  ;Работа программы начинается
PUSH AX  ;с установки AX=0
MOV BX, DSEG  ;Задание базового адреса
MOV DS, BX    ; сегмента данных
CALL MAIN   ;Вызов процедуры MAIN
RET
START ENDP  ;конец процедуры START

MAIN PROC NEAR        ;Начало процедуры MAIN
        MOV AH, CATS  ;Ваша программа
        MOV BH, RATS  ;Ваша программа
        MOV CH, BATS  ;Ваша программа
        MOV DL, CATS  ;Занесение буквы A в DL для вывода
        MOV AH, 2     ;Задание функции вывода символа
        INT 21H       ;Обращение к системной программе
        RET           ;Организация возврата
MAIN ENDP             ;Конец процедуры MAIN
;
CSEG ENDS   ;конец сегмента кодов
END START   ;указываем имя первой выполняемой процедуры
```

Текст программы заканчивается оператором **END**, в котором указывается имя процедуры или оператора, с которого начинается выполнение программы. Оператор **END** указывает Ассемблеру, где завершить трансляцию программы.

Описание данных

Все данные и переменные, используемые в программе, должны быть описаны явно в сегменте данных.

Синтаксис оператора описания данных:

[имя переменной] Dx список значений

[имя переменной] является необязательным параметром и указывает на имя данных, по которому можно к ним обращаться.

Тип данных определяется параметром D**x**. Возможны следующие значения **DB, DW, DD, DQ, DT.**

DB определяет данные длиной байт. Например:

MY_BYTE DB 233 ;переменная MY_BYTE занимает 1 байт и значение 233

MY_LIST DB 15, 19, 23, 22 ;список байтовых данных под именем MY_LIST

X DB ? ;переменная X длиной 1 байт, значение ее не определено

Y DB 10 DUP (?) ;переменная Y длиной 10 байт, значение ее не определено

STRING DB 'ASSEMBLER' ;строка из символов ASCII

DW определяет данные длиной слово. Например:

MY_WORD DW 3453 ;переменная MY_WORD занимает слово, значение 34533

Формат команды Ассемблера

Общий вид команды:

[метка :] КОП [Операнд 1] [,Операнд 2] ;Комментарий

Элементы команды, указанные в квадратных скобках, в различных командах могут отсутствовать.

Пересылка данных

К наиболее простым и часто используемым командам Ассемблера относится команда **MOV** (переслать). Она используется для того, чтобы

1) данные длиной 1 байт или 2 байта переслать:

а) из регистра в регистр;

б) из регистра в память;

в) из памяти в регистр;

2) данные, непосредственно записанные в команде, переслать в регистр или по заданному адресу в память.

Комментарий помещают после точки с запятой. Можно привести следующие примеры команды **MOV**:

MOV AL, BL *;переслать содержимое регистра BL в регистр AL;*

MOV DS, AX *;переслать содержимое регистра AX в регистр DS;*

MOV AH, 3 *;переслать число 3 в регистр AH*

MOV CX, 256 *;переслать число 256 в регистр CX.*

MOV BL, 00110101B *;переслать двоичную константу в регистр BL;*

MOV CX, 0A6BCH *;переслать шестнадцатеричную константу в регистр CX.*

MOV DL, 'A' ;*занести в регистр DL символ A в коде ASCII*

Вывод символа на экран по прерыванию типа 21H

Ввод/вывод символов в/из компьютера выполняется в кодах ASCII (American Standard Code for Information Interchange). Для вывода символа на экран используется непосредственное обращение к функции ввода-вывода системной программы:

MOV AH, 2 *указываем системе, что хотим вывести символ*

MOV DL,'C' *в регистр DL заносим выводимый символ в коде ASCII*

INT 21H *вызываем системную программу*

В регистр **AH** загружается код функции, указывающий операционной системе, что должен быть выведен символ, содержащийся в регистре **DL**. Выводимый символ заносится в регистр **DL**, по команде **INT 21H** происходит прерывание, и управление

передается системной программе ДОС. Одновременно с выводом символа на экране курсор перемещается на одну позицию вправо. Затем управление возвращается прикладной программе, и начинается выполнение команды, следующей за командой **INT 21H.**

Ввод символа с клавиатуры по прерыванию типа 21H

Для ввода символа с клавиатуры используется непосредственное обращение к функции ввода/вывода системной программы:

MOV AH, 1 *указываем системе, что хотим ввести сим**вол***

INT 21H *вызываем системную программу*

Программа ждет нажатия клавиши для ввода символа. После нажатия символ в коде ASCII попадает в регистр AL.

Организация циклов

В программе часто необходимо выполнять многократно некоторые фрагменты программы. Для этого в Ассемблере имеется удобное решение организации циклов. Количество повторений цикла заносится в регистр **CX (только в CX!),** а команда **LOOP** обеспечивает выполнение заданного количества повторений. Как правило, команда **LOOP** ставится в конце повторяющейся последовательности команд программы. Формат команды:

Loop метка

Команда вычитает единицу из содержимого регистра **CX** и, если содержимое неравно нулю, передает управление на метку. Если регистр **CX** становится равен нулю, выполняется следующая за **LOOP** команда. В регистр **CX** заносится количество повторений цикла.

Ассемблеры MASM и TASM

MASM является двухпроходным Ассемблером. Это означает, что исходный текст программы он просматривает два раза. За первый проход он определяет все переменные в программе и строит таблицу их адресов. Таблица содержит имена переменных, адреса, в которых они находятся и тип данных. Во время второго прохода Ассемблер заменяет каждую команду на машинный код и определяет длину каждой команды.

TASM является однопроходным ассемблером. Он создает машинный код и таблицу адресов за один проход.

Листинг программы

```
_Microsoft (R) Macro Assembler Version 5.10              9/3/96 17:54:04
0000                            sseg segment stack
 0000  0100[                    db 256 dup (?)
          ??
                               ]

 0100                           sseg ends
 0000                           dseg segment
```

```
0000  61                  cats db 'a'
0001  62                  rats db 'b'
0002  63                  bats db 'c'
0003                      dseg ends
                          ;
0000                      cseg segment
                          assume cs:cseg, ds:dseg, ss:sseg
0000                      start proc far
0000  1E                  push ds                ;
0001  50                  push ax                ;
0002  BB ---- R           mov bx,dseg
0005  8E DB               mov ds,bx
0007  E8 000B R           call main
000A  CB                  ret
000B                      start endp
000B                      main proc near
000B  8A 26 0000 R        mov ah,cats
000F  8A 3E 0001 R        mov bh,rats
0013  8A 2E 0002 R        mov ch,bats
0017  8A 16 0000 R        mov dl,cats
001B  B4 02               mov ah,2
001D  CD 21               int 21h
001F  CB                  retf
0020                      main endp
0020                      cseg ends
                          end start
```

Рассмотрим листинг подробнее. Слева в каждой строке листинга стоит четырехзначное число. Это значение смещения для команды, текст которой напечатан в той же строке справа между значением смещения и исходной командой записано машинной представление этой команды. В сегменте данных (ему присвоено имя DSEG) описаны три переменные с именами CATS, RATS, BATS. Их машинные коды 61, 62, 63 и для них выделены три байта памяти (сегмент заканчивается значением 3). В сегменте кодов (его имя CSEG) в левой части указаны смещения адресов соответствующих команд. По их значениям можно определить длины машинных команд.

Работа с отладчиком DEBUG

Отлаживать программу удобнее с отладчиком. Отладчик **DEBUG** позволяет отслеживать выполнение программы по шагам, просматривать содержимое регистров, запускать программу с заданных точек.

Запустите программу **DEBUG** с выполняемым файлом **.EXE.** После появления знака – (тире) наберите команду**.**

Команды DEBUG:

-R высветить содержимое регистров

-T выполнить одну команду

-G[адрес] выполнить программу до указанного адреса

-Q выйти из **DEBUG**

-P выполнить процедуру (фрагмент программы, начинающийся с **CALL** или **INT).**

Выполнение лабораторной работы

1. Текст программы на Ассемблере наберите в редакторе текстов, например, **NotePad**. Файл должен иметь расширение **.ASM**

2. Для трансляции программы на ассемблере выполните программу **MASM** или **TASM** с указанием имени файла с вашей программой, которая имеет расширение **.ASM**. На экране появится сообщение:

Source filename [.ASM]: (наберите имя-файла программы)

Object filename [имя-файла.OBJ]: (нажмите ввод)

Source listing {NUL.LST]: (укажите имя-файла листинга или нажмите ввод)

Cross-reference [NUL.CRF]: (нажмите ввод)

Результатом работы является объектный модуль, который записывается в файл с расширением **.OBJ**.

3. Если нет ошибок (сообщение **Severe errors 0**) выполните программу **LINK,** функция которой сформировать выполняемый модуль с расширением **.EXE**, подключив к нему необходимые стандартные программы. Появится сообщение:

Object modules [.OBJ]: (наберите имя-файла с расширением .OBJ)

Runfile [имя-файла.EXE]: (нажмите ввод)

Libraries [.LIB]: (нажмите ввод)

4. Запустите на выполнение полученный **.EXE** файл

5. Подготовьте отчет по лабораторной работе. Отчет должен содержать вариант задания, исходный текст программы с комментариями каждой команды, листинг программы, пояснения к листингу.

Варианты заданий

1. Напишите программу вывода инициалов вашего имени и фамилии, после каждой буквы поставьте точки.
2. Выведите на экран двоичное представление символа. Программа вводит символ в коде ASCII с клавиатуры, выводит на экран его двоичное представление (например, двоичное представление буквы V – это 01010110).
3. Напишите программу сложения пяти чисел 1, 2, 3, 4, 5. Ведите двоичное число, представляющее сумму этих чисел.
4. Напишите программу ввода с клавиатуры двух символов, вычисления их суммы и вывода двоичного представления суммы на экран.

Контрольные вопросы

1. Что такое оперативная память?
2. Какие регистры имеются в IBM PC?
3. Какова длина и назначение регистров?
4. Что находится в регистре сегмента?
5. Что делает команда пересылки?
6. Что такое сегмент?
7. Из каких частей состоит программа на Ассемблере?
8. Что такое процедура?
9. Какая информация содержится в листинге программы?
10. Как вычислить смещение адреса по листингу программы?
11. Куда вводится символ при вводе по прерыванию INT 21H?
12. Откуда выводится символ при выводе по прерыванию INT 21H?
13. Как организуется цикл в программе?