Торгашин Роман Геннадьевич
преподаватель
Государственное бюджетное профессиональное образовательное учреждение Воронежской области «Борисоглебский техникум промышленных и информационных технологий»
г. Борисоглебск Воронежской области

ИСПОЛЬЗОВАНИЕ ОТЛАДЧИКА DEBUG.EXE ПРИ ИЗУЧЕНИИ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

При изучении архитектуры микропроцессорных систем необходимо наглядно иллюстрировать алгоритмы работы этих систем и давать студентам базовые знания по программированию на ассемблере. Например, таковы требования ФГОС для курса МДК.02.01 “Микропроцессорные системы» специальности 09.02.01 «Компьютерные системы и комплексы».

В числе изучаемых архитектур мы уделяем внимание одной из наиболее распространенных - x86 (IA-32). Стенды для изучения микропроцессоров такой архитектуры существуют в продаже, но цена колеблется от 15 до 150 тысяч рублей. Их приобретение может оказаться серьезной нагрузкой на бюджет учебного заведения, а предлагаемая комплектация избыточна. К счастью, все современные персональные компьютеры либо используют процессоры x86, либо позволяют их эмулировать.

На первоначальном этапе изучения мы используем *бесплатную* утилиту **debug.exe,** входящую в состав MS-DOS и эмулятора командной строки Windows. Эта утилита, несмотря на свои скромные возможности, позволяет отслеживать: выполнение инструкций, изменение данных в памяти и содержимого регистров - поэтому ее удобно использовать для изучения базовых понятий.

Первое, с чем знакомятся студенты – кодирование команд на машинном языке, размещение команд в памяти и методы адресации данных. Для этого им предлагаются следующие задания:

Ввести программу в виде последовательности машинных инструкций с использованием непосредственной адресации данных[2].

Таблица 1 Код для задания по использованию непосредственной адресации

Машинная инструкция	Символьный код	Комментарий
B82301	MOV AX, 0123	Занести значение 0123 в AX
052500	ADD AX, 0025	Прибавить к значению AX 0025H
8BD8	MOV BX, AX	Занести значение AX в BX
03D8	ADD BX ,AX	Прибавить значение из AX к BX
8BCB	MOV CX, BX	Занести значение BX в CX
2BCB	SUB CX, AX	Отнять значение AX от CX
2BC0	SUB AX, AX	Отнять AX от AX (очистить AX)
EBEE	JMP 100	Перейти к началу программы

Заметим, что поскольку студент только начинает изучение материала - на данном этапе предлагается готовый код программы. В принципе, для выполнения задания достаточно привести машинные инструкции, но символьный код и описание выполняемых команд обеспечивают лучшее понимание происходящего и готовят к изучению ассемблера.

После ввода машинных инструкций студент пошагово выполняет программу, контролируя и анализируя изменения регистров.

```
-e cs:100 B8 23 01 05 25 00 8B D8 03 D8 8B CB 2B CB 2B C0 EB EE
-d cs:100 114
0B8F:0100  B8 23 01 05 25 00 8B D8-03 D8 8B CB 2B CB 2B C0   .#..%.......+.+.
0B8F:0110  EB EE F6 06 03                                    .....
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B8F  ES=0B8F  SS=0B8F  CS=0B8F  IP=0100   NV UP EI PL NZ NA PO NC
0B8F:0100 B82301        MOV     AX,0123
-t

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B8F  ES=0B8F  SS=0B8F  CS=0B8F  IP=0103   NV UP EI PL NZ NA PO NC
0B8F:0103 052500        ADD     AX,0025
-t

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B8F  ES=0B8F  SS=0B8F  CS=0B8F  IP=0106   NV UP EI PL NZ NA PE NC
0B8F:0106 8BD8          MOV     BX,AX
-
```

Рисунок 1 Программа с непосредственной адресацией

На рисунке 1 показано, как выглядит экран после ввода программы, просмотра регистров и выполнения двух шагов.

Частой ошибкой при выполнении этого задания становится неверное вычисление смещения. Если студент вводит не все команды за один раз, а последовательно, ему нужно вычислять правильное смещение для каждой команды. Проблема возникает при смещении большем 109. По привычке студенты вычисляют в десятичной системе счисления, а не в шестнадцатеричной и получают смещение 110 вместо 10A.

Уже на этом этапе необходимо обратить внимание студентов на то что, в конце программы есть команда зацикливания выполнения. Следует объяснить им, что процессор не может «остановиться». Пока подается питание и тактовые импульсы – он вычисляет адрес следующей команды и выполняет ее. Без зацикливания или явного перехода к другой программе (выхода в ОС) он будет перебирать адреса до тех пор, пока не достигнет границы имеющейся области памяти. Это может привести к исключению или к тому, что будет выполнена команда, не относящаяся к данной программе.

Далее студенты знакомятся с прямой адресацией и использованием предопределенных данных.

Для этого вводят, начиная со смещения 0200h, данные 23 01 25 00 00 00 2A 2A 2A, а, начиная со смещения 0100h, код A1 00 02 03 06 02 02 A3 04 02 EB F4.[2]

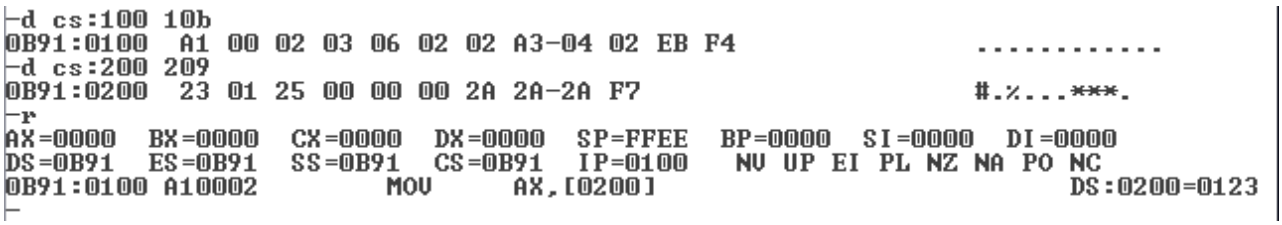

Рисунок 2 Работа с прямой адресацией в debug

На рисунке 2 показано состояние памяти и регистров при выполнении этого задания.

Далее утилиту debug мы используем при изучении обработки прерываний. В этих заданиях студенты вводят команды ассемблера, что дает им возможность более «плавно» перейти к изучению этого языка.

Все задания на этом этапе основаны на вызове прерываний DOS при помощи команды INT. Сначала студентам предлагается готовая последовательность команд:

```
MOV AH, 2A
INT 21
JMP 100
```

Важно обратить внимание студентов на то, что при пошаговом выполнении программ с использованием команды INT изменяется последовательность использования инструкций debug. В остальных случаях для выполнения очередной команды можно использовать **(t)race**, но при выполнении INT это приводит к тому, что процессор переходит к подпрограмме обработки прерывания, определенной в DOS.

После такого перехода студенту приходится выполнять трассировку пока не будет закончена обработка прерывания или вручную переопределять адрес сегмента и смещения в регистрах процессора и запускать программу заново. Такая ошибка имеет положительное значение с точки зрения обучения, так как наглядно иллюстрирует работу алгоритмов обработки прерываний. Поэтому ее можно допустить намеренно.

Для того чтобы выполнения заданий не занимало слишком много времени стоит рекомендовать студентам использовать инструкцию **(p)roceed**, которая заставляет debug выполнить команду без входа в подпрограмму.

После выполнения команд студенты читают и расшифровывают полученные данные из регистров процессора. В данном случае использована инструкция для получения текущей даты INT 21h с кодом функции Fn 2Ah.

Таблица 2 Размещение результатов выполнения прерывания 21h(2Ah)

Регистр	Информация
AL	День недели, где 0 - воскресенье
CX	Год
DH	Месяц
DL	День месяца

Следующее задание: вывести на экран текст используя INT 21h(09h). Оно иллюстрирует, кроме обработки прерываний, использование заранее определенных символьных данных.

Таблица 3 Код программы вывода текста с использованием INT 21h(09h)[2]

Инструкция	Комментарий
MOV AH, 09	Подготовка кода функции
MOV DX, 109	Смещение, в котором определены выводимые байты
INT 21	Вызов прерывания 21 для функции 2A
JMP 100	Возврат в начало программы
DB 'Hello All!','$'	Определение строки для вывода. '$' - окончание строки

При выполнении этого задания нужно обратить внимание студентов на значение символа '$' в конце строки данных.

Дальнейшие задания не содержат готового ассемблерного кода – студентам предлагается самостоятельно изменить существующую программу:

1. Получить дату и время, используя инструкцию INT 21h(2Ch)
2. Реализовать ввод символа с клавиатуры с использованием INT 16h(10h)
3. Самостоятельно изучить спецификацию соответствующего прерывания DOS и получить следующие данные (по вариантам):
   a. INT 12H - размер используемой памяти
   b. INT 21H - номер версии DOS
   c. INT 21H - информация о свободной памяти диска.

Следующим этапом изучения архитектуры x86 и ассемблера является использование эмулятора **emu8086**. В эту программу интегрирован Flat Assembler и реализована работа с различными виртуальными устройствами доступными через порты эмулируемого процессора. В некоторых устройствах реализована обратная связь, что позволяет изучить реальные алгоритмы обмена данными между MCU и периферией. API, который использует программа для работы с виртуальными устройствами, описан в документации, что позволяет

разрабатывать собственные виртуальные устройства. В состав дистрибутива emu8086 входит также набор примеров кода, которые можно использовать при изучении принципов программирования на FASM или как основу при разработке практических работ.

На момент написания этой статьи бессрочная лицензия на неограниченное число рабочих мест стоит 24.869 р., годовая — 4.383 р., бессрочная на одного человека — 417 р. Существует бесплатный аналог данной программы — **i8086emu**, который распространяется по лицензии GPL. Но он менее удобен и его работа нестабильна. Мы сталкивались со случаями, когда эмулятор зависал при выполнении кода примера входящего в состав дистрибутива.

Однако опыт использования этого emu8086 выходит за рамки данной статьи.

Список используемых источников

1. Питер Абель. Ассемблер. Язык и программирование для IBM PC – К: Век+, 2003.

2. CODENET. Прерывания DOS и BIOS – URL: http://www.codenet.ru/progr/dos/