Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Космический»

КАФЕДРА «Прикладная математика, информатика и вычислительная техника»

РАЗРАБОТКА ПРОСТЕЙШЕЙ ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ МИКРОПРОЦЕССОРА INTEL 8086

Методические указания по выполнению лабораторной работы для студентов специальности 09.03.01

Студент гр. К3-22М _____ А. С. Митрохин
(Подпись, дата)

Руководитель _____ А. В. Чернышов
(Подпись, дата)

2020 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящие методические указания включают лабораторную работу, рекомендуемую для студентов, впервые знакомящихся с языками ассемблера и архитектурой x86 в частности.

В пособии приводится теоретический минимум, необходимый для получения следующих начальных навыков:

— Разработка простейших программ на языке ассемблера для микропроцессора Intel 8086.

— Трансляция и компоновка (линковка) программы с использованием Turbo Assembler (TASM) и Turbo Linker в операционной системе MS DOS.

— Выполнение отладки разработанной программы с применением утилит Borland Turbo Debugger и debug.

Студентам предлагается выполнить индивидуальное задание, которое позволит на практике применить и закрепить полученные знания.

Теоретические сведения

Intel 8086 (в 1980-85 годах известный как iAPX 86/10) — первый 16-битный микропроцессор компании Intel. Разрабатывался с весны 1976 года и выпущен 8 июня 1978 года. Несмотря на то, что в 2018 году процессор отметил 40-летний юбилей, реализованная в процессоре архитектура набора команд стала основой широко известной архитектуры x86, активно применяющейся на сегодняшний день.

Современные процессоры этой архитектуры сохраняют возможность выполнять все команды этого набора, поэтому изучение языка ассемблера для этого процессора можно использовать в качестве исходной точки для изучения архитектуры x86.

Порядок следования байтов в ОП в архитектуре x86 носит название Little-endian. Это значит, что младший байт последовательности следует в её начале (запись с «младшего конца), а старший – в конце. Например, слово (AB CD) в little-endian будет записано как (CD BA). К слову, в произведении Джонатана Свифта «Путешествия Гулливера» описываются воюющие государства Лилипутия и Блефуску из-за споров по поводу того, с какого конца следует разбивать варёные яйца. Тех, кто считает, что их нужно разбивать с тупого конца, в произведении называют Big-endians («тупоконечники» в русском переводе).

На рисунке 1 представлена регистровая память изучаемого микропроцессора. Как можно заметить, регистры Intel 8086 имеют 16 разрядов, потому он и называется 16-битным.

Рис. 1 Регистры процессора Intel 8086

Регистры разделены на 3 группы: общего назначения (AX, BX, CX, DX), указателей (SP, DP, DI, SI), сегментных (CS, DS, SS, ES), и вне категорий находятся указатель команд (IP), указывающий адрес текущей инструкции относительно сегмента CS, и, наконец, регистр флагов процессора.

Регистры общего назначения предназначены для хранения операндов арифметико-логических инструкций, а также адресов или отдельных компонентов адресов ячеек памяти. Именно они будут использоваться при разработке простой программы на языке ассемблера

Практическая часть

Рассмотрим следующий пример. Требуется разработать программу на языке ассемблера Intel 8086, решающую математическое выражение $d := a + b - c$, где указанные переменные заданы словом. Для проверки работоспособности следует произвести отладку программы.

1 Разработка первой программы на языке ассемблера

Intel 8086 весьма стар, и современные операционные системы не поддерживают его работу. Однако из этой ситуации можно выйти, используя эмулятор операционной системы MS DOS dosemu, который можно запустить из эмулятора терминала в операционной системе Linux.

Запустив dosemu, вы окажетесь в командной строке, в которую можно вводить команды MS DOS.

Перейдите в раздел диска D, введя «d:”. В эмуляторе раздел D: привязан к вашему домашнему каталогу, и ваши написанные исходные файлы при сохранении будут находиться именно там.

Теперь вам необходимо запустить входящий в стандартный пакет программ MS DOS текстовый редактор EDIT. Для этого введите edit <filename.asm>, где filename.asm – имя исходного файла вашей будущей программы.

Примечание. Если возникает ситуация, когда необходимо найти файл, а вы не помните его название, но знаете расширение, можно воспользоваться командой DIR. Например, при вводе ***dir *.asm*** *консоль выведет все файлы в текущем каталоге, название которых оканчивается на «.asm».*

В случае успеха откроется окно приветствия программы, и теперь можно приступать к написанию исходного текста программы

1.1 Исполняемая часть программы

Исполяемый код программы должен храниться в сегменте кода (инструкций). Для его объявления следует написать директивы, указанные в листинге 1:

Листинг 1 – Объявление сегмента

```
1. code segment
2. code ends
```

segment является директивой объявления семгента с указанным именем. В нашем случае, сегмент называется **code**. Для объявления конца сегмента используется директива **ends**, перед которой так же необходимо указать название завершаемого сегмента

Для хранения данных, как правило, создаётся отдельный сегмент, но в данной лабораторной работе это будет проигнорировано. Все данные и инструкции будут находиться в одном сегменте для упрощения разработки. Опишем черновик логики программы, который можно увидеть на листинге 2:

Листинг 2 – Логика программы

```
1. code segment
2.    mov ax, a   ;ax = a
3.    add ax, b   ;ax = ax + b
4.    sub ax, c   ;ax = ax - c
5.    mov d, ax   ;d = ax
6.    hlt         ;Остановить процессор
7. code ends
```

Рассмотрим написанный код. Он содержит 4 разные инструкции:

mov – команда пересылки. Она позволяет записать значение второго операнда, исполняющей роль отправителя, в первый — приёмника. Приёмником может являться регистр или адрес ячейки памяти. Источником могут быть как регистр, так и ячейка памяти, а также непосредственное значение.

В первой инструкции происходит запись значения переменной **a** в регистр **ax**, в четвёртой — отправление значения **ax** в переменную **d.**

add – команда сложения двух операндов, сумма записывается в первый операнд. Во второй инструкции происходит сложение **ax** и значения b, результат будет помещён в **ax**.

sub – команда вычитания второго операнда из первого, результат хранится в первом операнде. В третьей инструкции происходит вычитание c из ax с помещением результата в **ax**.

Команда **hlt** является инструкцией остановки работы процессора, однако внешнее прерывание восстановит его работу. В данном случае этой командой будет завершена работа программы.

Исходя из описания инструкций, получаем следующий алгоритм:

1. Записать в регистр **ax** значение переменной **a**.

2. Прибавить к значению **ax** значение переменной **b**, тем самым складывая a и b.

3. Вычесть из регистра **ax** значения **c**. Теперь в **ax** хранится результат выражения **a + b – c**.

4. Записать в переменную **d** значение **ax.**

1.2 Указание данных

Следующим этапом для написания программы будет являться указание самих величин переменных, потому что без значений a, b и c произвести вычисления не удастся.

Объявление переменной может быть выполнено двумя способами.

Первый способ: объявление переменных в качестве меток. Этот способ является наиболее распространённым в языках ассемблера разных процессоров, поэтому следует начать с него

Общий формат имеет вид: **var: define_directive value**, где var – имя метки, define_directive – директива объявления, value – значение.

Рассмотрим подробнее директивы объявления. В процессоре Intel8086 их всего 3: db, dw, dd. Их особенности указаны в таблице 1.

Таблица 1. Директивы объявления данных

Имя директивы	**Число резервируемых байт**	**Диапазон значений**
db (define byte – объявить байт)	1	-128..127
dw (define word – объявить слово)	2	-32767..32768
dd (define double word – объявить двойное слово)	4	-2147483648..2147483647

Осталось выбрать подходящую для программы директиву. Размер операндов в одной инструкции должен всегда совпадать, иначе транслятор выведет соответствующую ошибку. Исходя из вышеизложенного, размер регистра ax имеет 16 разрядов, соответственно, наше значение должно быть так же двухбайтовым, следовательно, необходимо использовать директиву dw. В листинге 3 указана программа с объявленными переменными.

Листинг 3

```
1.   code segment
2.      mov ax, a
3.      add ax, b
4.      sub ax, c
5.      mov d, ax
6.      hlt
7.   a: dw 6
8.   b: dw 3
9.   c: dw 7
10.   d: dw ?
11.  code ends
```

Вы наверняка заметили, что в значении переменной d указан вопросительный знак. Это означает, что для вас не имеет значения, какое значение будет иметь переменная d, т. к. туда мы только запишем результат.

Если попробовать провести трансляцию программы, то транслятор выведет ряд ошибок:

— «Недосягаемые» переменные a, b, c ,d

— Неизвестный размер a, b, c, d

— Нет точки входа программы

Первая ошибка связана с тем, что транслятор будет пытаться искать данные в сегменте данных, которого в программе просто нет, поэтому необходимо явно задать сегмент нахождения переменных в регистре кода, например, **cs:a.**

Вторая ошибка связана с тем, что наши переменные представлены в виде меток, и вместо значения в a, b, c и d пишутся их адреса в памяти. В связи с этим, нужно указать, сколько байт должно быть взято от начала адреса метки. Для этого используется директива.

type ptr, где **type** – размер данных (**byte**, **word**, **dword** для 1, 2 и 4 байт соответственно).

Третья ошибка говорит о том, что не задана метка, откуда начинает работу программа, и директивы, которая указывает конец программы и точку входа в неё. Директива конца программы имеет формат **end start_label**, где **start_label** — точка входа в программу. Исправленный файл представлен в листинге 4

Листинг 4 – Компилируемый исходный файл

```
1.    code segment
2.    start:
3.       mov ax, word ptr cs:a
4.       add ax, word ptr cs:b
5.       sub ax, word ptr cs:c
6.       mov word ptr cs:d, ax
```

```
7.    a: dw 6
8.    b: dw 3
9.    c: dw 7
10.   d: dw ?
11.  code ends
12.      end start
```

Программа готова и теперь можно приступать к её компиляции и отладке.

Однако в компиляторе TASM можно использовать другой метод: объявлять переменные именно как переменные, и это позволит избежать явного указания типа переменной?

Листинг 5 – Объявление данных в формате переменных

```
1.    code segment
2.    start:
3.       mov ax, cs:a
4.       add ax, cs:b
5.       sub ax, cs:c
6.       mov cs:d, ax
7.    a dw 6
8.    b dw 3
9.    c dw 7
10.  d dw ?
11.  code ends
12.     end start
```

Примечание. В процессе изучения можно заметить строгое разделение понятия «директива» и «инструкция». ***Директива*** *– команда, предназначенная для транслятора с целью указать на особенности обработки ис-*

ходного кода и не транслируется в машинный код. ***Инструкция*** *– исполняемая процессором команда, имеющая свой код.*

2 Трансляция и компоновка

Для трансляции необходимо ввести следующую инструкцию в командную строку, находясь в каталоге, где лежит ваш исходный файл:

z:\tasm\tasm filename.asm

Если запустить программу без аргумента, то автоматически будет выведена справка, из которой можно узнать дополнительные опции трансляции. Например, для включения отладочной информации следует использовать флаг **–zi.**

Если в выведенном сообщении в графе Error Messages указано None, то вы всё сделали верно, то транслятор сформирует объектный файл filename.obj и можно приступить к компоновке:

Z:\tasm\tlink filename.obj

Если запустить программу без аргумента, то автоматически будет выведена справка, из которой можно узнать дополнительные опции линковки. Например, для включения отладочной информации следует использовать флаг **/v.**

Компоновщик выведет сообщение warning: no stack, сообщающий об отсутствии сегмента кода в программе. Т.к. мы его и не создавали, сообщение можно смело игнорировать. Теперь мы получили исполняемый файл. Однако попытка запуска ни к чему хорошему не приведёт, т. к. исполняемый файл должен иметь особую обвязку для поддержки запуска операционной системой. При этом, даже если программа запустилась, разработчик ничего не увидит, т. к. нет никакого вывода на экран. В связи с этим, необходимо использовать отладчик.

3 Отладка программы

3.1 Отладка с помощью программы DEBUG

Debug.exe — программа-отладчик, разработанная для операционной системы MS DOS и используемая для отладки исполняемых файлов. Под более поздние версии операционных систем (Windows NT и старше) работает через эмулятор MS-DOS и имеет ограниченные возможности. До Windows XP включительно, отладчик debug.exe являлся стандартным компонентом системы.

Для входа в отладчик введите команду **debug filename.exe**

Вы увидите знак дефиса, после которого необходимо ввести символ-команду. Для начала вызовем справку, введя символ «**?».**

Теперь вы видите развёрнутый список возможных команд. Ниже приведены ключи, которые будут далее использованы при отладке

— D [диапазон] – вывести дамп памяти в заданном диапазоне

— R – вывести значения регистров и следующую исполняемую команду

— U – дизассемблировать исполняемый файл

— T – выполнить трассировку программы (без параметров выполнит 1 шаг)

— Q – выйти из отладчика

Для начала, введите команду **u.** Дизассемблированный исполняемый файл выводит листинг в формате *исполняемый адрес, машинный код, дизассемблированная команда. Отладчику всё равно, что дизассемблировать, и даже наши данные будут представлены в виде какой-либо команды.*

Получим дамп памяти от начала программы: **d cs:0000.** Отладчик выведет набор байт в шестнадцатеричном коде. Попробуйте найти смещение от сегмента, от которого начинаются данные, используя выведенный дизассемблированный код и дамп памяти. Обратите внимание, что на месте перемен-

ной **d** находится неизвестное значение. Так как в исходном коде не указано значение переменной **d**, там будет храниться мусор из оперативной памяти.

Выведем значения регистров процессора ключом **r**. Отладчик показывает значения всех регистров процессора. Теперь пошагово выполните программу, наблюдая за изменениями значения регистров. После отображения команды HLT, повторно выведите дамп памяти со значениями переменных и убедитесь, что программа выполнена верно.

После завершения отладки выйдите из отладчика, введя **q**

3.2 Отладка с помощью Borland Turbo Debugger

Turbo Debugger, в отличие от символьного отладчика debug, имеет полноценный графический интерфейс. Его управление осуществляется исключительно из клавиатуры, поэтому для навигации по меню необходимо нажать F10, а затем с помощью стрелок перемещаться внутри него.

Запустим исполняемый файл в этом отладчике, введя команду

Z:\td\td filename.exe

При запуске отладчик сообщит об отсутствии символьной таблицы и просит нажать **ESC** для назначения. При закрытии окна предупреждения вы увидите дизассемблированный файл в центре окна, а также значения регистров справа и дамп памяти внизу.

Для отслеживания изменений значений в оперативной памяти с помощью нажатий на Tab сделайте активным окно дампа. Нажмите Ctrl+G, и вам предложат ввести значение, от которого начнётся вывод значений. Введите вычисленное ранее значение смещения для выдачи значений переменной и нажмите Enter.

Для выполнения пошаговой отладки нажимайте на клавишу F8 и наблюдайте за процессом. Если вы выполните команду HLT, то отладчик сообщит об остановке работы программы и сбросит значения регистров и оперативной памяти. В связи с этим, для отката в исходное состояние зайдите во вкладку Run и выполните Program Reset (или нажмите Ctrl+F2).

3.3 Использование отладочной информации в Turbo Debugger

Отладочная информация позволяет облегчить жизнь разработчику при тестировании работы программы. Первым делом, выполните трансляцию и компоновку, используя флаги из раздела 2.

Теперь при запуске файла в Turbo Debugger вы будете видеть свой исходный код, однако, как можете заметить, исчезли окна значений регистров и дампа памяти. Для отображения перейдите на вкладку View и найдите вкладку Registers. Окно отобразится вовсе не так, как удобно разработчику, но его всегда можно переместить, однократно нажав на Scroll Lock, а затем с помощью стрелок отправить в нужную часть экрана. Для принятия нового положения нажмите Enter.

Проделайте аналогичные действия с окном Dump. Теперь для переключения между активными окнами нажимайте F6. Если вы не можете вернуть нужное окно, то вы можете повторно запросить его отображение через View.

Если вы хотите увидеть дизассемблированную версию файла, то во вкладке View откройте CPU. Если вы хотите закрыть окно, нажмите F3.

Одним из преимуществ такой отладки является возможность легко отслеживать значения ваших данных. Если они заданы как переменные, то вы можете открыть окно Variables и увидеть их значения. Однако, если ваши переменные заданы как метки, то единственным способом является переход в дампе памяти с помощью Ctrl+G, в качестве аргумента используя название метки. К слову, то же самое можно делать, если они объявлены как стандартные переменные.

Попробуйте отслеживать ваши переменные при пошаговой отладке.

Индивидуальное задание

Согласно номеру варианта, напишите программу вычисления математического выражения с записью результата по адресу заданной переменной на языке ассемблера Intel 8086, используя размер переменных и метод их задания, согласно таблице 2, и проведите отладку способами из раздела 3.

Таблица 2

Номер варианта	Математическое выражение	Размер переменной	Метод задания данных
1	a := a + b + 3	Байт	Метка
2	d := -a + b – c	Байт	Переменная
3	a := 2a	Слово	Метка
4	d := a – 5 – c	Слово	Переменная
5	c := a + 2b	Байт	Метка
6	c := 2a - b	Байт	Переменная
7	c := b – a	Слово	Метка
8	a := 3a – b	Слово	Переменная
9	c := a – 2b	Байт	Метка
10	b := 4a	Байт	Переменная
11	a := a + 13 + c	Слово	Метка
12	d := -a + b – c	Слово	Переменная
13	c := a + b	Байт	Метка
14	d := a – (-17) – c	Байт	Переменная
15	c := a + 2b	Слово	Метка
16	b := 2a + 1	Слово	Переменная
17	c := b – a	Байт	Метка
18	a := 3a – b	Байт	Переменная
19	c := a – 2b	Слово	Метка
20	b := 4a	Слово	Переменная
21	a := 2a	Байт	Метка
22	a := 16 + b + c	Байт	Переменная
23	d := -a + b - c	Слово	Метка

Таблица 2 (продолжение)

Номер варианта	Математическое выражение	Размер переменной	Метод задания данных
24	a := 2a	Слово	Переменная
25	d := a – b – 1	Байт	Метка
26	c := a + 2b	Байт	Переменная
27	c := 2a + b	Слово	Метка
28	c := b - a	Слово	Переменная
29	a := 3a – b	Байт	Метка
30	c := a – 2b	Байт	Переменная
31	b := 4a	Слово	Метка
32	a := 4 + b + c	Слово	Переменная
33	d := -a + b - c	Байт	Метка
34	a := 2a	Байт	Переменная
35	d := a – b – c	Слово	Метка
36	c := a + 2b	Слово	Переменная
37	c := 2a + b	Байт	Метка
38	c := b - a	Байт	Переменная
39	a := 3a – b	Слово	Метка
40	b := 2a	Слово	Переменная

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 8086 [Электронный ресурс] // Википедия URL: https://ru.wikipedia.org/wiki/8086
2. Полный набор команд процессора 8086 [Электронный ресурс] // URL: http://www.avprog.narod.ru/progs/emu8086/8086_instruction_set.html
3. Скэнлон Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера // Радио и связь. 1991, С. 336
4. Брэй Б. «Микропроцессоры Intel: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4. Архитектура, программирование и интерфейсы» // БХВ-Петербург. 2005. С. 1328