МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образование

учреждение высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова»

(ФГБОУ ВО «ИжГТУ имени М. Т. Калашникова»)

Институт непрерывного профессионального образования

Лабораторная работа №1

«Введение в ассемблер»

Выполнил студент группы: 5-19-1 од

Тепляков Георгий Константинович

Принял: преподаватель

Ситников Василий Владимирович

Ижевск

2018

**Цель работы**

Ознакомиться с машинно-ориентированным языком программирования низкого уровня-ассемблером. Научиться составлять программы на данном языке, компилировать их с помощью компилятора FASM и отлаживать программы, используя отладчик Turbo Debugger.

**Задание**

Написать простую программу на 20-25 строк с комментариями (программа может не выполнять ничего существенного, но она не должна зацикливаться и вешать систему).

Краткие теоретические сведения и понятия

**Описание языков Ассемблера**

Все языки программирования разделяются по уровням: низкий и высокий. Любой из синтаксической системы «семейки» Ассемблера отличается тем, что объединяет сразу некоторые достоинства наиболее распространенных и современных языков. С другими их роднит и то, что в полной мере можно использовать систему компьютера.

Отличительной особенностью компилятора является простота в использовании. Этим он отличается от тех, которые работают лишь с высокими уровнями. Если взять во внимание любой такой язык программирования, Ассемблер функционирует вдвое быстрее и лучше. Для того чтобы написать в нем легкую программу, не понадобится слишком много времени.

**О структуре языка**

Если говорить в общем о работе и структуре функционирования языка, можно точно сказать, что его команды полностью соответствуют командам процессора. То есть Ассемблер использует мнемокоды, наиболее удобные человеку для записи.

В отличие от других языков программирования, Ассемблер использует вместо адресов для записи ячеек памяти определенные метки. Они с процессом выполнения кода переводятся в так называемые директивы. Это относительные адреса, которые не влияют на работу процессора (не переводятся в машинный язык), а необходимы для распознавания самой средой программирования. Для каждой линейки процессора существует своя система команд. При таком раскладе правильным будет любой процесс, в том числе и переведенный машинный код.

**Язык Ассемблера имеет несколько синтаксисов**

Плюсы языка

Наиболее важным и удобным приспособлением языка Ассемблера станет то, что на нем можно написать любую программу для процессора, которая будет весьма компактной. Если код оказывается огромным, то среда программирования некоторые процессы перенаправляет в оперативную память. При этом они все выполняют достаточно быстро и без сбоев, если конечно, ими управляет квалифицированный программист.

Драйвера, операционные системы, BIOS, компиляторы, интерпретаторы и т. д. – это все программа на языке Ассемблера. При использовании дизассемблера, который совершает перевод из машинного в компьютерный язык, можно запросто понять, как работает та или иная системная задача, даже если к ней нет пояснений. Однако такое возможно лишь в том случае, если программы легкие. К сожалению, в нетривиальных кодах разобраться достаточно сложно.

Минусы языка

К сожалению, начинающим программистам (и зачастую профессионалам) трудно разобрать язык. Ассемблер требует подробного описания необходимой команды. Из-за того, что нужно использовать машинные команды, растет вероятность ошибочных действий и сложность выполнения.

Для того чтобы написать даже самую простую программу, программист должен быть квалифицированным, а его уровень знаний достаточно высоким. Средний специалист, к сожалению, зачастую пишет плохие коды.

Если платформа, для которой создается программа, обновляется, то все команды необходимо переписывать вручную – этого требует сам язык. Ассемблер не поддерживает функцию автоматического регулирования работоспособности процессов и замену каких-либо элементов.

**Команды языка**

Как уже было сказано выше, для каждого процессора имеется свой набор команд. Простейшими элементами, которые распознаются любыми типами, являются следующие коды:

* Пересылка данных осуществляется при помощи mov и т. д.
* Команды, связанные с арифметикой: sub, imul и др.
* Побитовые и логические функции можно реализовать при помощи or, and и т. п. Именно эти основы языка Ассемблера позволяют ему быть схожим с другими.
* Для того чтобы осуществить переход от одной команды к другой, следует прописать такие операторы: djnz, cfsneq, cjne. Неопытному программисту может показаться, что это просто набор букв, однако это неверно.
* In и out применяются в том случае, если возникла необходимость ввода в порт (или вывода из него).
* К управляющим командам относят int. Благодаря ему можно прекратить выполнение каких-либо процессов в пользу основного действия.

**Использование директив**

Программирование микроконтроллеров на языке (Ассемблер это позволяет и прекрасно справляется с функционированием) самого низкого уровня в большинстве случаев заканчивается удачно. Лучше всего использовать процессоры с ограниченным ресурсом. Для 32-разрядной техники данный язык подходит отлично. Часто в кодах можно заметить директивы. Что же это? И для чего используется?

Для начала необходимо сделать акцент на том, что директивы не переводятся в машинный язык. Они регулируют выполнение работы компилятором. В отличие от команд, эти параметры, имея различные функции, отличаются не благодаря разным процессорам, а за счет другого транслятора. Среди основных директив можно выделить следующие:

* Всем известные макросы.
* Имеются также директивы, которыми обладает высший язык. Ассемблер отлично «считывает» их и выполняет.
* Функции контролирования и управления режимами компилятора.
* Распознавание констант и переменных.
* Регулирование работы программ, которые находятся в оперативной памяти.

**Происхождение названия**

Благодаря чему получил название язык – "Ассемблер"? Речь идет о трансляторе и компиляторе, которые и производят зашифровку данных. С английского Assembler означает не что иное, как сборщик. Программа не была собрана вручную, была использована автоматическая структура. Более того, на данный момент уже у пользователей и специалистов стерлась разница между терминами. Часто Ассемблером называют языки программирования, хотя это всего лишь утилита.

Из-за общепринятого собирательного названия у некоторых возникает ошибочное решение, что существует единый язык низкого уровня (или же стандартные нормы для него). Чтобы программист понял, о какой структуре идет речь, необходимо уточнять, для какой платформы используется тот или иной язык Ассемблера.

**Макросредства**

Языки Ассемблера, которые созданы относительно недавно, имеют макросредства. Они облегчают как написание, так и выполнение программы. Благодаря их наличию, транслятор выполняет написанный код в разы быстрее. При создании условного выбора можно написать огромный блок команд, а проще воспользоваться макросредствами. Они позволят быстро переключаться между действиями, в случае выполнения условия или невыполнения.

При использовании директив макроязыка программист получает макросы Ассемблера. Иногда он может широко использоваться, а иногда его функциональные особенности снижаются до одной команды. Их наличие в коде облегчает работу с ним, делает его более понятным и наглядным. Однако следует все равно быть внимательным – в некоторых случаях макросы, наоборот, ухудшают ситуацию.

Компилятор FASM fasm (сокращение от flat assembler) — свободно распространяемый многопроходной ассемблер, написанный Томашем Грыштаром. fasm самодостаточен, изначально компилировался TASM’ом и стал способен собираться из собственных исходных кодов начиная с версии 0.90 (4 мая 1999). fasm обладает небольшими размерами и очень высокой скоростью компиляции, имеет богатый и ёмкий макро-синтаксис, позволяющий автоматизировать множество рутинных задач. Поддерживаются как объектные форматы, так и форматы исполняемых файлов. Это позволяет в большинстве случаев обойтись без компоновщика. В остальных случаях нужно использовать сторонние компоновщики, поскольку таковой вместе с fasm не распространяется.

Компиляция программы в fasm состоит из 2 стадий: препроцессирование и ассемблирование.

Препроцессирование на стадии препроцессора раскрываются все макросы, символические константы, обрабатываются директивы препроцессора.

В отличие от стадии ассемблирования, препроцессирование выполняется только 1 раз. Смешивание стадий ассемблирования и препроцессирования — распространённая ошибка начинающих.

Ассемблирование на стадии ассемблирования определяются адреса меток, обрабатываются условные директивы, раскрываются циклы и генерируется собственно программа. Fasm — многопроходной ассемблер, что позволяет ему делать некоторую оптимизацию, например, генерирование короткого перехода на метку вместо длинного. Во время прохода компилятор не всегда может вычислить выражение в условных директивах. В этом случае он делает какой-нибудь выбор и пытается скомпилировать дальше. Благодаря тому, что адреса меток, вычисленные на N-ном проходе, используются на N+1-проходе, этот процесс обычно сходится.

**Отладчик Turbo Debugger**

Отладчик Turbo Debugger представляет собой набор инструментальных средств, позволяющий отлаживать программы на уровне исходного текста и предназначенный для программистов, использующих семейство компиляторов Borland. В пакет отладчика входят набор выполняемых файлов, утилит, справочных текстовых файлов и примеров программ.

Turbo Debugger позволяет вам отлаживать программы для Microsoft Windows, Windows 32s, Windows NT и DOS. Многочисленные перекрывающие друг друга окна, а также сочетание спускающихся и раскрывающихся меню обеспечивают быстрый, интерактивный пользовательский интерфейс. Интерактивная, контекстно-зависимая справочная система обеспечит вас подсказкой на всех стадиях работы. Кроме того, Turbo Debugger полный набор средств отладки:

* Вычисление любых выражений языка Си, C++, Pascal и Assemb ler.
* Полное управление выполнением программы, включая программ ную анимацию.
* Доступ на нижнем уровне к регистрам процессора и системной памяти.
* Полные средства проверки данных.
* Развитые возможности задания точек останова и регистрации.
* Трассировка сообщений Windows, включая точки останова по сообщениям.
* Обратное выполнение.
* Поддержка удаленной отладки, в том числе для Windows.
* Полная поддержка объектно-ориентированного программирова ния, включая просмотр классов и проверку объектов.
* Макрокоманды в виде последовательности нажатий клавиш, ус коряющие выполнение команд.
* Копирование и вставка между окнами и диалогами.
* Контекстно-зависимые меню.
* Возможность отладки больших программ.
* Диалоговые окна, позволяющие вам настроить параметры от ладчика.
* Возможность отладчик 16- и 32-разрядных программ Windows (для 32-разрядной отладки имеется отладчик TD32).
* Обработка исключительных ситуаций операционной системы, а также С и С++.
* Сохранение сеанса.
* Поддержка нитей для мультинитевого программирования Win dows NT.
* Возможность подключения готовых к выполнению в Windows процессов.
* Возможность выбора для элементов, выводимых в Turbo Debug ger, национального порядка сортировки.

|  |
| --- |
| use16 ;Генерировать 16-битный код  org 100h ;Программа начинается с адреса 100h  mov ax,255 ;поместить 255 в регистр AX  inc ax ;увеличить содержимое AX на 1  nop ;ничего не делает  mov bx,ax ;поместить содержимое BX в AX  mov al,[a] ;Загружаю значение а в AL  inc al ;al = al + 1  mov ah,[b] ;Загружаю значение c в AH  add al,ah ;al = al + ah  mov [c],al ;сохраняем результат в с  mov al,[q] ;Загружаю значение q в AL  mov ah,[p] ;Загружаю значение p в AH  dec ah ;ah = ah - 1  sub al,ah ;al = al - ah  mov [s],al ;сохраняем результат в с  mov dx,m ;в DX адрес строки  mov ah,9 ;номер функции DOS.  int 21h ;обращение у функции DOS.  mov ax,4C00h ;\  int 21h ;/ Завершение программы  a db 2 ; \  b db 5 ; |Объявление значений  c db ? ; /  q db 7 ; \  p db 2 ; |Объявление значений  s db ? ; /  m db 'Hello !$' |

Код программы

Результат выполнения программы

Для того, чтобы написать программу, нам надо запустить fasmw.exe. Откроется окошко, в которое можно набивать код:

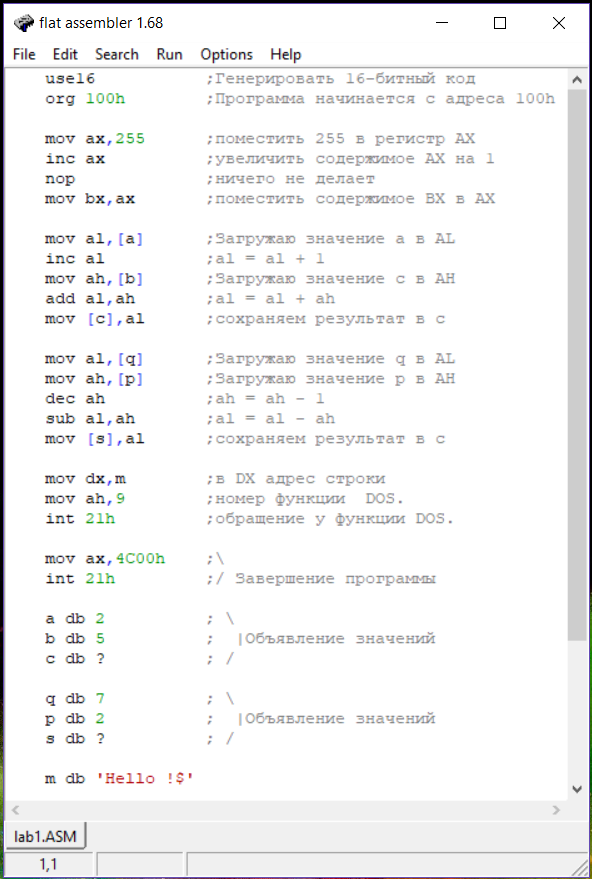


Рис. 1 – вид программы Fasm assembler c написанным кодом

Чтобы откомпилировать программу надо выбрать меню Run->Compile. FASM предложит сохранить файл, а затем скомпилирует. То есть переведет текст, набранный нами, в машинный код и сделает его программой.

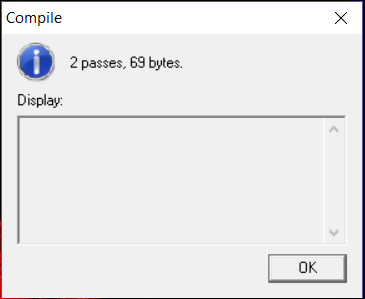


Рис. 2 – окно окончания компиляции

Теперь можно запустить отладчик Turbo Debugger, чтобы открыть программу надо зайти File->Open, после чего в строке «File name» нужно указать путь до программы с разрешением COM и нажать «ок», после чего откроется программа в которой можно увидеть, как выполняется программа.

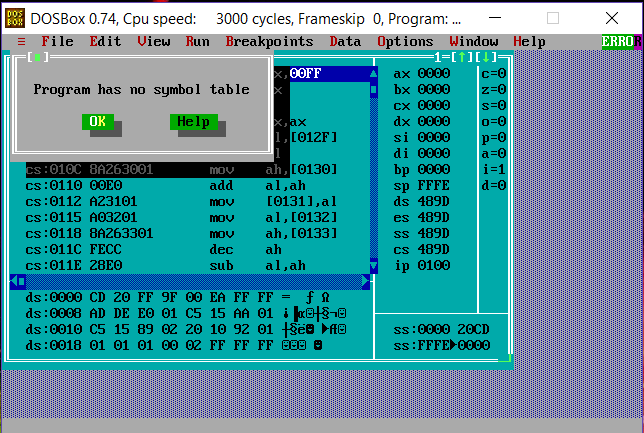


Рис 3 - Сообщение означает, что в исполняемом файле нет специальных данных для отладки

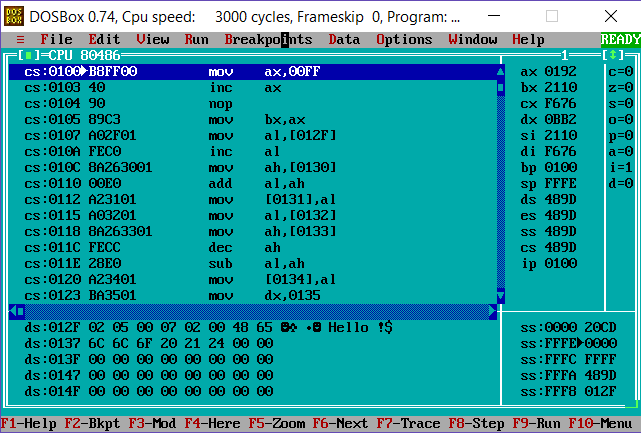


Рис 4 - Вид программы в отладчике Turbo Debugger

**Вывод**

В ходе выполнения данной лабораторной работы была написана программа, которая складывает и вычитает числа, а также выводит строку «Hello». Я ознакомился с работой компилятора FASM и работой с окладчиком Turbo Debugger.