Московский государственный технический университет
имени Н.Э. Баумана

==========

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Г.С. Иванова, Т.Н. Ничушкина

ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ MASM32 В СРЕДЕ RADASM С ИСПОЛЬЗОВАНИЕМ 32-РАЗРЯДНОГО ОТЛАДЧИКА OLLYDBG

Методические указания по выполнению
лабораторных работ № 1-2 по дисциплине
Машинно-зависимые языки и основы компиляции

Москва

УДК 004.432

ББК 32.973.26-018.1

И 21

Рецензент: Новик Наталья Владимировна, к.т.н., доцент кафедры ИУ7

Иванова Г.С., Ничушкина Т.Н.

Программирование на ассемблере MASM32 в среде RADAsm с использованием 32-разрядного отладчика OlleDBG: Методические указания по выполнению лабораторных работ № 1–2 по дисциплине Машинно-зависимые языки и основы компиляции для студентов 2 курса кафедры ИУ6 (бакалавры). М.: МГТУ имени Н.Э. Баумана, 2015. 38 с.

Издание содержит теоретический материал для изучения и описание процесса выполнения двух лабораторных работ, посвященных первичному знакомству с программированием на ассемблере MASM32 в настраиваемой среде программирования на ассемблерах RADAsm. Определены цели лабораторных работ, задания, последовательность их выполнения и требования к отчету. Приведено описание основных возможностей среды и отладчика OlleDBG. Рассмотрены форматы представления данных и способы их адресации. Приведены примеры простейших программ, демонстрирующие особенности программирования на ассемблере вычислений.

Для студентов 2 курса кафедры ИУ6 МГТУ имени Н.Э. Баумана, обучающихся по программе бакалавриата направления «Информатика и вычислительная техника», профиль Вычислительные машины, комплексы, системы и сети.

Рекомендовано учебно-методической комиссией факультета «Информатика и системы управления» МГТУ им. Н.Э. Баумана

Учебное издание

Иванова Галина Сергеевна

Ничушкина Татьяна Николаевна

Программирование на ассемблере MASM32 в среде RADAsm с использованием 32-разрядного отладчика OlleDBG.

Методические указания по лабораторным работам 1-2 по дисциплине Машинно-зависимые языки и основы компиляции

Оглавление

Предисловие

Язык ассемблера или ассемблер (англ. assembly language) — язык низкого уровня с командами, обычно соответствующими командам процессора (так называемым машинным командам). Это соответствие позволяет отнести язык к группе машинно-зависимых, к которой относятся также машинные языки.

В отличие от машинных языков языки ассемблера используют мнемонические (буквенные) обозначения команд, а также символическую адресацию для команд и данных. Это существенно облегчает процесс написания программ, непосредственно преобразуемых в машинные коды.

Кроме того язык ассемблера использует также макрокоманды, которым при переводе в машинный язык соответствует группа машинных команд, что сокращает размер программы, позволяя не расписывать последовательности машинных команд, реализующие часто встречающиеся действия.

Изучение основ программирования на языке ассемблера является необходимой частью обучения специалистов в области проектирования, как аппаратных, так и программных средств вычислительной технике, поскольку позволяет отчетливо понимать, как работает вычислительная машина в процессе выполнения программы.

В целом лабораторный практикум по дисциплине обеспечивает базовые знания для формирования следующих профессиональных компетенций:

– способность разрабатывать технические задания на разработку и выбор информационного, программного, аппаратного и эргономического обеспечения информационно-вычислительных систем (СПК-2);

– способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии (СПК-3);

– способность применять современные программно-методические комплексы исследования и автоматизированного проектирования объектов профессиональной деятельности (СПК-7);

– способность сопрягать аппаратные и программные средства в составе информационных и автоматизированных систем (СПК-9).

Первые две работы выполняются по специальным методическим указаниям и обеспечивают знакомство со средой и языком.

Лабораторная работа № 1. Изучение среды и отладчика ассемблера

Цель работы: изучение процессов создания, запуска и отладки программ на ассемблере в среде программирования RADAsm с использованием 32-разрядного отладчика OlleDBG.

Объем работы: 2 часа.

Введение

Masm32 – специализированный пакет программирования на 32-разрядном языке ассемблера IA-32. Являясь продуктом фирмы Microsoft, этот язык максимально приспособлен для создания Windows-приложений на ассемблере. Кроме транслятора, компоновщика и необходимых библиотек пакет Masm32 включает сравнительно простой текстовый редактор и некоторые инструменты, предназначенные для облегчения программирования на ассемблере. Однако базовый набор инструментов не содержит 32-х разрядного отладчика и предполагает работу в командном режиме, что не очень удобно.

В лабораторных работах для создания программ на языке ассемблера будет использоваться специализированная интегрированная среда программирования на ассемблерах RADAsm, которая помимо других ассемблеров позволяет программировать на языке ассемблера Masm32. Точнее будет использоваться специально настроенный экземпляр среды – «сборка» RADAsm + OlleDBG, где OlleDBG – 32-х разрядный отладчик, отображающий в шестнадцатеричной системе счисления содержимое всех сегментов программы (сегментов кода, данных и стека) и позволяющий выполнять трассировку программы, включать в нее точки останова, а так-

же просматривать содержимое памяти в процессе пошагового выполнения программы. Наиболее важные настройки приведены в Приложении А.

1 Начало работы со средой

Программная среда активизируется запуском программы RADAsm.exe.

После вызова программы на экране появляется окно среды RADAsm, в котором обычно высвечивается последняя программа, отлаживаемая в среде во время предыдущего сеанса работы (рисунок 1.1).

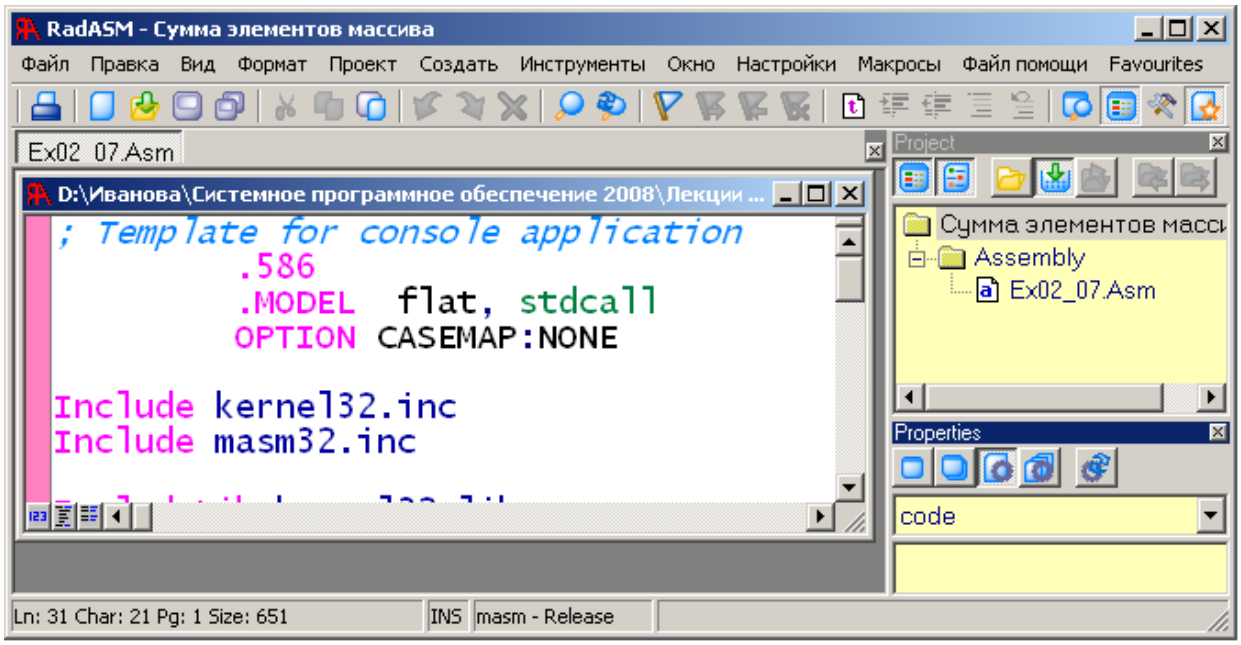

Рисунок 1.1 – Окно интегрированной среды RADAsm

Помимо стандартных компонентов окно среды содержит меню, линейку инструментов, строку состояния, окно программы и два служебных окна: окно навигации Project и окно свойств Properties. Работа со средой осуществляется с помощью команд меню или соответствующих им комбинаций «горячих» клавиш.

Для создания нового проекта необходимо выбрать пункт меню **Файл/Новый проект**, после чего на экране появится первое окно четырехоконного Мастера создания проекта (рисунок 1.2).

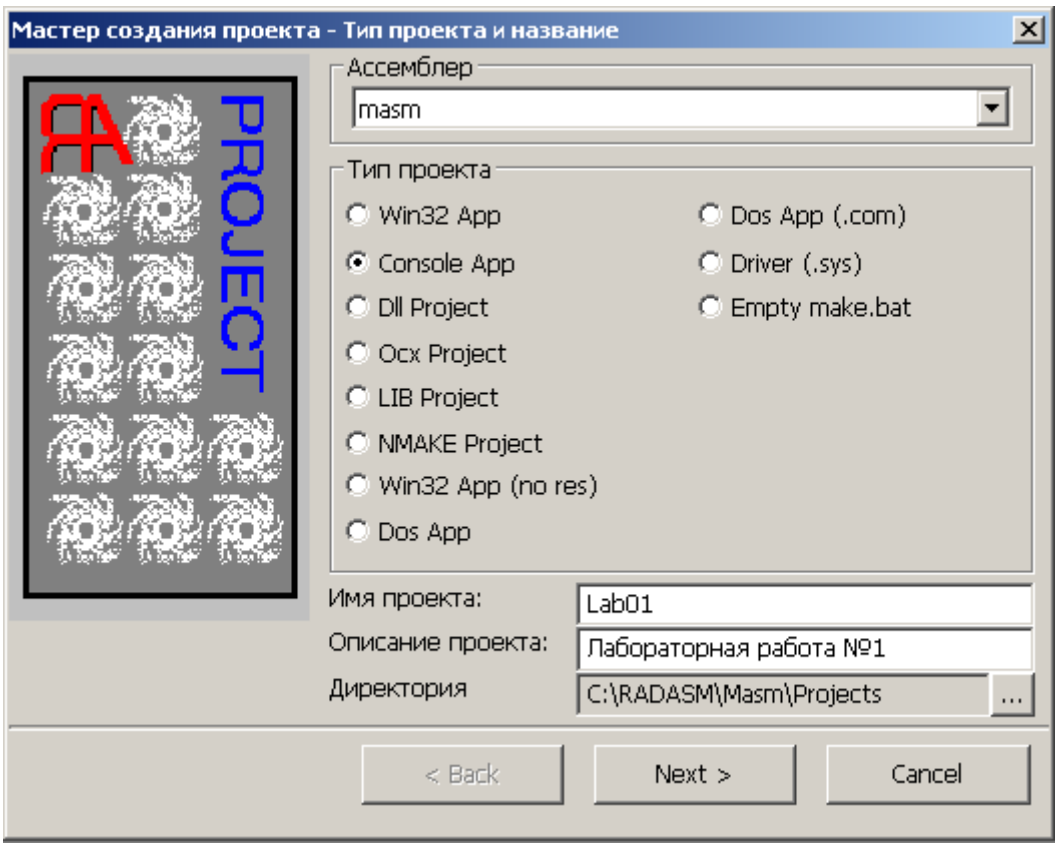

Рисунок 1.2 – Окно мастера создания нового проекта

В этом окне необходимо выбрать тип проекта – в нашем случае Console App (консольное приложение), а также ввести его имя, например, Lab01, описание, например, «Лабораторная работа № 1», и путь к создаваемой средой новой папке с именем проекта.

На следующем шаге в специальном окне Мастера выбирается шаблон проекта – conapp.tpl – специально созданный для лабораторных работ шаблон консольного приложения.

Предпоследнее окно Мастера предлагает выбрать типы создаваемых файлов – выбираем Asm (исходные файлы ассемблера), и папки – выбираем папку Bak, используемую для размещения предыдущих копий файлов.

Последнее окно Мастера определяет доступные для работы с проектом пункты меню запуска приложения **Создать** и выполняемые команды. В данной сборке выполняемые команды уже настроены, поэтому в нем можно ничего не менять, хотя использоваться будут не все пункты созданного меню, а только следующие: **Assemble** (транслировать или, точнее, ас-

семблировать), **Link** (компоновать), **Run** (выполнить) и **Run w/Debug** (выполнить в подключенном отладчике).

В результате будет получена заготовка консольного приложения Windows. Просмотреть эту заготовку можно, дважды щелкнув левой клавишей мыши по файлу Lab01.asm в окне навигатора Project, расположенном справа вверху:

```
; Template for console application  - комментарий
        .586            ; подключение набора команд
Pentium
        .MODEL  flat, stdcall ; модель памяти и
                        ; конвенция о передаче парамет-
ров
        OPTION CASEMAP:NONE  ; опция различия строч-
ных
                                ;  и прописных букв
Include kernel32.inc   ; подключение описаний процедур
и
Include masm32.inc     ; констант

IncludeLib kernel32.lib ; подключение библиотек
IncludeLib masm32.lib

        .CONST  ; начало раздела констант
MsgExit  DB    "Press Enter to Exit",0AH,0DH,0

        .DATA  ;раздел инициализированных переменных

        .DATA? ;раздел неинициализированных перемен-
ных
inbuf    DB    100 DUP (?)
        .CODE  ; начало сегмента кода
Start:
;                                       Место, куда
;    Add you statements                 добавляется код
;
        Invoke StdOut,ADDR MsgExit   ; вывод сообще-
ния
        Invoke StdIn,ADDR inbuf,LengthOf inbuf
                                        ; ввод строки
        Invoke  ExitProcess,0       ; завершение про-
граммы
```

```
End    Start                ; конец модуля
```

Заготовку, как в других средах программирования, можно запустить на выполнение. Она содержит вызовы процедур, обеспечивающих вывод на экран запроса «Press Enter to Exit» (Нажмите клавишу Enter для выхода) и ввод строки. Это сделано для того, чтобы задержать автоматическое закрытие окна консоли при завершении программы до нажатия клавиши Enter.

2 Запуск заготовки приложения

Для запуска заготовки программы на выполнение необходимо осуществить:

- трансляцию **Создать/Assemble**,
- компоновку **Создать/Link**,
- запуск на выполнение **Создать/Run**.

В процессе трансляции (ассемблирования) исходная программа на ассемблере преобразуется в ее двоичный эквивалент. Если трансляция проходит нормально, то в окне Output, которое появляется под окном программы, выводится текст следующего вида:

```
C:\Masm32\Bin\ML.EXE /c /coff /Cp /nologo
                                  /I"C:\Masm32\Include"
"Lab01.asm"
Assembling: Lab01.asm

Make finished.
Total compile time 78 ms
```

Примечание – Окно Output появляется и закрывается. Чтобы повторно посмотреть результаты, необходимо установить курсор мыши на верхнюю часть строки состояния под окном текста программы (нижнюю рамку окна Output).

Первая строка сообщения об ассемблировании – вызов ассемблера:

C:\Masm32\Bin\ML.EXE – полное имя файла транслятора ассемблера masm32 (путь + имя), за которым следуют опции:

/c – заказывает ассемблирование без автоматической компоновки,

/coff – определяет формат объектного модуля Microsoft (coff),

/Cp – означает сохранение регистра строчных и прописных букв всех идентификаторов программы,

/nologo – осуществляет подавление вывода сообщений на экран в случае успешного завершения ассемблирования,

/I"C:\Masm32\Include" – определяет местонахождение вставляемых (.inc) файлов,

и параметр **"Lab01.asm"** – задает имя обрабатываемого файла.

Остальные строки – сообщение о начале и завершении процесса ассемблирования и времени выполнения этого процесса.

Результатом нормального завершения ассемблирования является создание файла, содержащего объектный модуль программы, – файла Lab01.obj.

Если при ассемблировании обнаружены ошибки, то объектный модуль не создается и после сообщения о начале ассемблирования идут сообщения об ошибках, например:

```
Lab01.asm(26) : error A2006: undefined symbol : EAY
```

что означает, что ассемблер не распознал символическое имя EAY, встретившееся в строке 26 исходного файла.

В сообщении указывается:

- номер строки исходного текста (в скобках),
- номер ошибки, под которым она описана в документации,

- возможная причина.

После исправления ошибок процесс ассемблирования повторяют, пока не будет получен синтаксически корректный текст программы.

Следующий этап – компоновка программы. На этом этапе к объектному (двоичному) коду программы добавляются объектные коды используемых подпрограмм. При этом в тех местах программы, где происходит вызов процедур, указывается их относительный адрес в модуле. Сведения о компоновке также выводятся в окно Output:

```
C:\Masm32\Bin\LINK.EXE /SUBSYSTEM:CONSOLE /RELEASE
     /VERSION:4.0 /LIBPATH:"C:\Masm32\Lib"
     /OUT:"Lab01.exe" "Lab01.obj"
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Make finished.
Total compile time 109 ms
```

Первая строка вывода также является командной строкой вызова компоновщика

C:\Masm32\Bin\LINK.EXE – полное имя компоновщика, за которым следуют опции:

/SUBSYSTEM:CONSOLE – подключить стандартное окно консоли,

/RELEASE – создать реализацию (а не отладочный вариант),

/VERSION:4.0 – минимальная версия компоновщика,

/LIBPATH:"C:\Masm32\Lib" – путь к файлам библиотек,

/OUT:"Lab01.exe" – имя результата компоновки – загрузочного файла

и параметр **"Lab01.obj"** – имя объектного файла.

В следующих строках также возможны сообщения об ошибках. В основном в лабораторных работах вы будете получать сообщение о неразрешенных внешних ссылках, например:

```
Lab01.obj:error LNK2001:unresolved external symbol
_ExitProcess@4
Lab01.exe : fatal error LNK1120: 1 unresolved exter-
nals
Make error(s) occured.
```

Как правило, такое сообщение говорит о наличии в программе вызовов процедур, для которых в указанных библиотеках не найдены соответствующие двоичные коды. Так в данном примере не подключена библиотека, в которой находится процедура **`ExitProcess`**, код которой таким образом оказался недоступен.

После устранения ошибок компоновки программу необходимо перетранслировать и заново скомпоновать.

Если процессы трансляции и компоновки прошли нормально, то программу можно запустить на выполнение. При этом открывается окно консоли, в которое выводится строка запроса (рисунок 1.3).

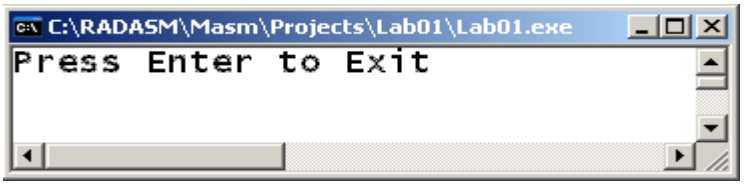

Рисунок 1.3 – Окно консоли

Окно закрывается при нажатии клавиши Enter.

3 Создание простейшей программы

Для изучения возможностей отладчика необходимо ввести программу, которая выполняет несколько простых команд, например, вычисляет результат следующего выражения:

$$X=A+5-B$$

Данные для программы зададим константами, поместив их описание в раздел инициированных данных:

```
            .DATA
A           SDWORD  -30
B           SDWORD  21
```

Для результата вычислений – переменной X – необходимо зарезервировать место, поместив описание соответствующей неинициализированной переменной в раздел неинициализированных данных:

```
            .DATA?
X           SDWORD  ?
```

Размещение неинициализированной переменной в этом разделе не обязательно, но более грамотно, поскольку неинициализированные переменные в образе исполняемой программы на диске не хранятся, а размещаются при загрузке на выполнение.

Фрагмент кода программы, выполняющей сложение и вычитание, необходимо поместить в сегменте кодов после метки Start:

```
Start:  mov     EAX,A ; поместить число в регистр EAX
        add     EAX,5;сложить EAX и 5, результат в EAX
        sub     EAX,B ;вычесть B, результат в EAX
        mov     X,EAX ; сохранить результат в памяти
```

Поскольку программа ничего не выводит, результат операции следует посмотреть в отладчике.

4 Просмотр выполнения программы в отладчике

Для запуска программы в режиме отладки следует последовательно инициировать следующие пункты меню:

- трансляцию **Создать/Assemble**,

- компоновку **Создать/Link**,
- запуск на выполнение в отладчике **Создать/Run w/Debug**.

После запуска на экране появляется окно отладчика OlleDBG (рисунок 1.4).

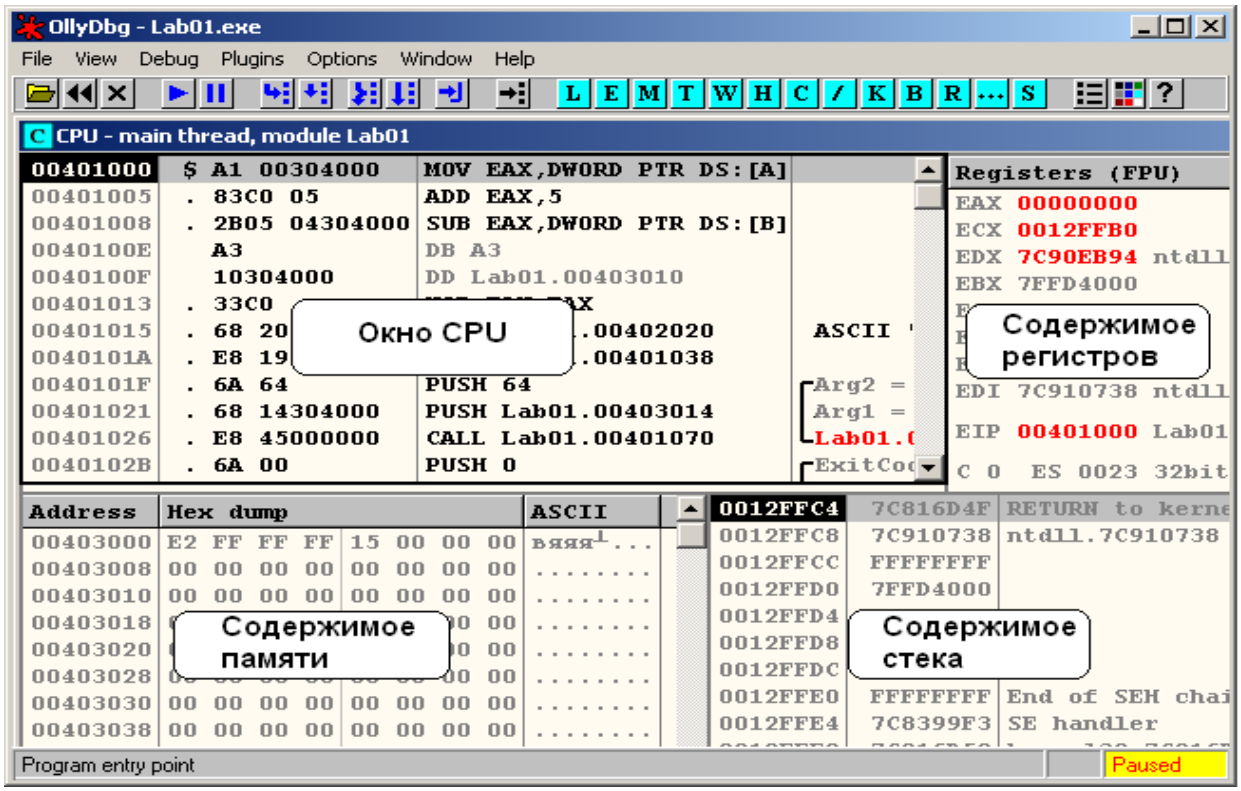

Рисунок 1.4 – Окно OlleDBG

В окне можно выделить четыре области:

- дочернее окно **CPU** – окно процессора, в котором высвечиваются адреса команд, их шестнадцатеричные коды, результаты дисассемблирования и результаты выделения параметров процедур;
- область **Registers** – окно содержимого регистров процессора,
- окно данных, в котором высвечиваются адреса памяти (**Address**) и ее шестнадцатеричный (**Hex dump**) и символьный (**ASCII**) дампы;
- окно стека, в котором показывается содержимое вершины стека.

В исходный момент времени курсор находится в окне CPU, т.е. программа готова к выполнению. Для выполнения команды отладчика используют следующие комбинации клавиш:

F7 – выполнить шаг с заходом в тело процедуры;

F8 – выполнить шаг, не заходя в тело процедуры.

Адрес начала программы **00401000h**. К сожалению, отладчик не всегда правильно дисассемблирует двоичный код программы, и какие-то команды могут остаться не распознанными. Так на рисунке 4 видно, что команда **mov X,EAX** оказалась не распознанной. Вместо нее в отладчике указан код, соответствующий определению данных: DB A3 и т.д. Однако выполнение программы будет осуществляться правильно.

Адрес начала раздела инициированных данных – **00403000h**. Каждое значение занимает столько байт, сколько резервируется директивой. В отладчике каждый байт представлен двумя шестнадцатеричными цифрами. Кроме того, используется обратный порядок байт, т.е. младший байт числа находится в младших адресах памяти (перед старшим). Если шестнадцатеричная комбинация соответствует коду символа, то он высвечивается в следующем столбце (**ASCII**), иначе в нем высвечивается точка.

Не инициированные данные располагаются после инициированных с адреса, кратного 16.

При каждом выполнении команды мы можем наблюдать изменение данных в памяти и/или в регистрах, отслеживая процесс выполнения программы и контролируя правильность промежуточных результатов.

Так после выполнения первой команды число A копируется в регистр EAX, который при этом подсвечивается красным. При записи в регистр порядок байт меняется на прямой, при котором первым записан старший байт.

5 Описание данных в программе на ассемблере

Все данные, используемые в программах на ассемблере, обязательно должны быть описаны.

Директива описания данных имеет следующий формат:

[<Имя>] <Директива>[<Константа>DUP(]<Список инициализаторов>[)]

где <Имя> – имя поля данных, которое может не присваиваться;

<Директива> – команда, объявляющая тип описываемых данных (см. таблицу 1);

<Константа> DUP – используются при описании повторяющихся данных, тогда константа определяет количество повторений;

<Список инициализаторов> – последовательность инициализирующих констант, указанных через запятую, или символ «?», если инициализирующее значение не определяется.

Таблица 1 – Директивы определения данных

Директива	**Описание типа данных**
BYTE	8-разрядное целое без знака
SBYTE	8-разрядное целое со знаком
WORD	16-разрядное целое без знака или ближний указатель RM
SWORD	16-разрядное целое со знаком
DWORD	32-разрядное целое без знака или ближний указатель
SDWORD	32-разрядное целое со знаком

Таблица 1 – Директивы определения данных (Окончание)

Директива	**Описание типа данных**
FWORD	48-разрядное целое или дальний указатель
QWORD	64-разрядное целое
TBYTE	80-разрядное целое
Real4	32-х разрядное короткое вещественное
Real8	64-х разрядное длинное вещественное
Real10	80-ти разрядное расширенное вещественное

Примечание – В качестве директив также могут использоваться:

- **DB –** определить байт,
- **DW –** определить слово,

- **DD –** определить двойное слово (4 байта),
- **DQ –** определить четыре слова (8 байт),
- **DT –** определить 10 байт,

однако при их использовании знаковые и беззнаковые, целые и вещественные типы не различаются, поэтому директивы считаются устаревшими.

В качестве инициализаторов при описании данных применяются:

- целые константы [<знак>]<целое> [<основание системы счисления>],

например:

  - -43236, 236**d** – целые десятичные числа,
  - 23**h**, 0AD**h** – целые шестнадцатеричные числа (если шестнадцатеричная константа начинается с буквы, то перед ней указывается 0),
  - 0111010**b** – целое двоичное;

- вещественные константы [<знак>] <целое> . [**E**|**e** [<знак>] <целое>], например: -2., 34**E**-28;
- символы в кодировке ASCII (MS DOS) или ANSI (Windows) в апострофах или кавычках, например: 'A' или ″A″;
- строковые константы в апострофах или кавычках, например, 'ABCD' или ″ABCD″.

Примеры определения данных различных типов приведены далее.

6 Задание

1. Запустите RADAsm, создайте файл проекта по шаблону консольного приложения. Внимательно изучите структуру программы и зафиксируйте текст с комментариями в отчете.

2. Запустите шаблон на выполнение и просмотрите все полученные сообщения. Убедитесь, что текст программы и настройки среды не содержат ошибок.

3. Добавьте директивы определения данных и команды сложения и вычитания, описанные в разделе 3 настоящих методических указаний. Найдите в отладчике внутреннее представление исходных данных, зафиксируйте его в отчете и поясните.

Проследите в отладчике выполнение набранной вами программы и зафиксируйте в отчете результаты выполнения каждой добавленной команды (изменение регистров, флагов и полей данных).

4. Введите следующие строки в раздел описания инициированных данных и определите с помощью отладчика внутренние представление этих данных в памяти. Результаты проанализируйте и занесите в отчет.

```
val1    BYTE   255
chart   WORD   256
lue3    SWORD  -128
alu     BYTE   ?
v5      BYTE   10h
        BYTE   100101B
beta    BYTE   23,23h,0ch
sdk     BYTE   "Hello",0
min     SWORD  -32767
ar      DWORD  12345678h
valar   BYTE   5 DUP (1, 2, 8)
```

5. Определите в памяти следующие данные:

а) целое число 25 размером 2 байта со знаком;

б) двойное слово, содержащее число -35;

в) символьную строку, содержащую ваше имя (русскими буквами и латинскими буквами).

Зафиксируйте в отчете внутреннее представление этих данных и дайте пояснение.

6. Определите несколькими способами в программе числа, которые во внутреннем представлении (в отладчике) будут выглядеть как **25 00 и 00 25**. Проверьте правильность ваших предположений, введя соответствующие строки в программу. Зафиксируйте результаты в отчете.

7. Замените директивы описания знаковых данных на беззнаковые:

```
A       DWORD   -30
B       DWORD   21
X       DWORD   ?
```

Запустите программу и прокомментируйте результат.

8. Добавьте в программу переменную F1=65535 размером слово и переменную F2= 65535 размером двойное слово. Вставьте в программу команды сложения этих чисел с 1:

```
        add     F1,1
        add     F2,1
```

Проанализируйте и прокомментируйте в отчете полученный результат (обратите внимание на флаги).

7 Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы и таблицы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Содержание отчета по лабораторной работе № 1 приведено в тексте задания на работу.

Отчет должны иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента.

8 Контрольные вопросы

1. Дайте определение ассемблеру. К какой группе языков он относится?

2. Как создать заготовку программы на ассемблере? Из каких частей она состоит?

3. Как запустить программу на ассемблере на выполнение? Что происходит с программой на каждом этапе обработки?

4. Назовите основные режимы работы отладчика. Как осуществить пошаговое выполнение программы и просмотреть результаты выполнения машинных команд.

5. В каком виде отладчик показывает положительные и отрицательные целые числа? Как будут представлены в памяти числа:

```
A   Word 5,-5
```
?

Как те же числа будут выглядеть после загрузки в регистр AX?

6. Что такое «разрядная сетка»? Как ограничения разрядной сетки влияют на представление чисел в памяти компьютера?

7. Каким образом в ассемблере программируются выражения? Составьте фрагмент программы для вычисления C=A+B, где A, B и C – целые числа формата BYTE.

Литература

1. Г.С. Иванова. Презентации курса лекций Машинно-зависимые языки и основы компиляции. М.: МГТУ им. Н.Э. Баумана, 2015.

2. Г.С. Иванова, Т.Н. Ничушкина. Основы программирования на ассемблере IA-32. Учебное пособие: Электронное учебное издание. М.: МГТУ им. Н.Э. Баумана, 2010. Адрес доступа: http://e-learning.bmstu.ru/moodle/mod/resource/ view.php?id=35.

Лабораторная работа № 2. Программирование целочисленных вычислений

Цель работы: изучение форматов машинных команд, команд целочисленной арифметики ассемблера и программирование целочисленных вычислений.

Объем работы: 2 часа.

Введение

Структура команд ассемблера отображает формат соответствующих машинных команд, поэтому очень важно разобраться, какие компоненты входят в машинную команду и какие способы адресации данных можно использовать при написании программы на ассемблере.

1 Форматы машинных команд IA-32

Размер машинной команды процессора IA-32 может меняться от 1 до 15 байт. Команда имеет следующую структуру (рисунок 2.1):

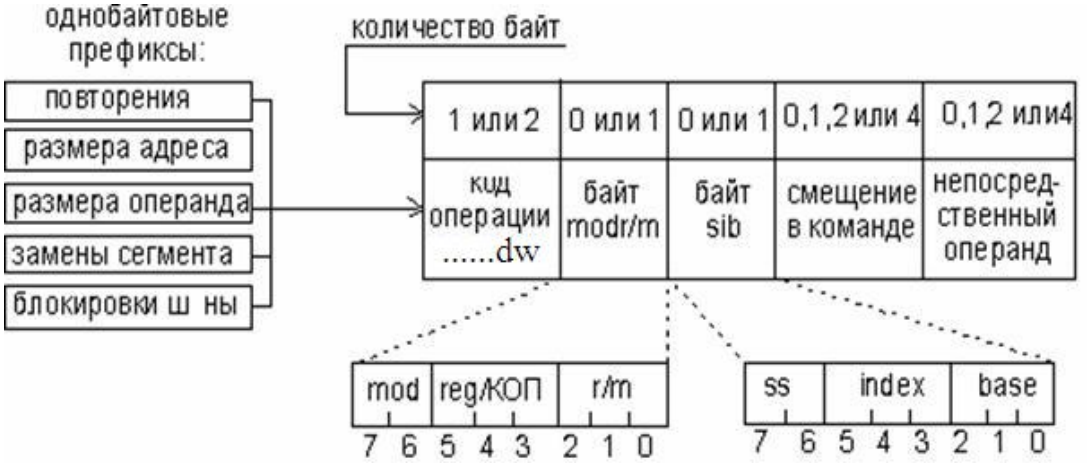

Рисунок 2.1 – Структура машинной команды IA32

На рисунке 1 использованы следующие обозначения:

префикс повторения – используется только для строковых команд;

префикс размера адреса (67h) – применяется для изменения размера смещения: 16 бит при 32-х разрядной адресации;

префикс размера операнда (66h) – указывается, если вместо 32-х разрядного регистра для хранения операнда используется 16-ти разрядный;

префикс замены сегмента – используется при адресации данных любым сегментом кроме DS;

d – направление обработки, например, пересылки данных:

1 – в регистр, 0 – из регистра;

w – размер операнда: 1 – операнды - двойные слова, 0 – операнды - байты;

mod – режим: 00 - Disp=0 – смещение в команде 0 байт;

01 - Disp=1 – смещение в команде 1 байт;

10 - Disp=2 – смещение в команде 2 байта;

11 - операнды-регистры.

Регистры кодируются в зависимости от размера операнда:

	w=1		w=0	
reg	**000**	**EAX**	**000**	**AL**
(r)	**001**	**ECX**	**001**	**CL**
	010	**EDX**	**010**	**DL**
	011	**EBX**	**011**	**BL**
	100	**ESP**	**100**	**AH**
	101	**EBP**	**101**	**CH**
	110	**ESI**	**110**	**DH**
	111	**EDI**	**111**	**BH**

Если в команде используется двухбайтовый регистр (например, AX), то перед командой добавляется префикс изменения длины операнда (66h).

Различают два вида команд, обрабатывающих операнд в памяти:

- команды без байта sib (см. таблицу 2);
- команды, содержащие байт sib (см. таблицу 3).

Различаются эти команды по содержимому поля m (r/m): если m≠100, то байт sib в команде отсутствует и используется таблица 2.

Таблица 2 – Схемы адресации памяти в отсутствии байта Sib

Поле r/m	Эффективный адрес второго операнда		
	mod = 00B	mod = 01B	mod = 10B
000B	EAX	EAX+Disp8	EAX+Disp32
001B	ECX	ECX+Disp8	ECX+Disp32
010B	EDX	EDX+Disp8	EDX+Disp32
011B	EBX	EBX+Disp8	EBX+Disp32
100B	*Определяется Sib*	*Определяется Sib*	*Определяется Sib*
101B	Disp32[1]	SS:[EBP+Disp8]	SS:[EBP+Disp32]
110B	ESI	ESI+Disp8	ESI+Disp32
111B	EDI	EDI+Disp8	EDI+Disp32

Таблица 3 – Схемы адресации памяти при наличии байта Sib

Поле base	Эффективный адрес второго операнда		
	mod = 00B	mod = 01B	mod = 10B
000B	EAX+ss*index	EAX+ss*index +Disp8	EAX+ss*index +Disp32
001B	ECX+ss*index	ECX+ss*index +Disp8	ECX+ss*index +Disp32
010B	EDX+ss*index	EDX+ss*index +Disp8	EDX+ss*index +Disp32
011B	EBX+ss*index	EBX+ss*index +Disp8	EBX+ss*index +Disp32
100B	SS:[ESP+ss*index]	SS:[ESP+ ss*index]+Disp8	SS:[ESP+ ss*index] +Disp32
101B	Disp32[1]+ss*index	SS:[EBP+ss*index +Disp8]	SS:[EBP+ss*index +Disp32]
110B	ESI+ss*index	ESI+ss*index +Disp8	ESI+ss*index +Disp32
111B	EDI+ss*index	EDI+ss*index +Disp8	EDI+ss*index +Disp32

ss – масштаб; *Index* – индексный регистр; *Base* – базовый регистр;

[1] – особый случай – адрес операнда не зависит от содержимого EBP, а определяется только смещением в команде (прямая адресация).

Примеры:

1) `mov EBX,ECX`

100010DW Mod Reg Reg
10001001 11 001 011
8 9 C B

2) `mov BX,CX`

префикс1 100010DW Mod Reg Reg
01100110 10001001 11 001 011
6 6 8 9 C B

3) **`mov ECX,DS:6[EBX]`**

100010DW Mod Reg Reg Смещение младший байт
10001011 01 001 011 00000110
4 B 4 B 0 6

4) **`mov CX,DS:6[EBX]`**
префикс 100010DW Mod Reg Reg Смещение младший байт
01100110 10001011 01 001 011 00000110
6 6 8 B 4 B 0 6

5) **`mov CX,ES:6[EBX]`**
префикс1 префикс2 100010DW Mod Reg Reg Смещение м. байт
01100110 00100110 10001011 01 001 011 00000110
6 6 2 6 8 B 4 B 0 6

6) **`mov ECX,6[EBX+EDI*4]`**
100010DW Mod Reg Mem SS Ind Base Смещение младший байт
10001011 01 001 100 10 111 011 00000110
8 B 4 C B B 0 6

Примечание. При выполнении лабораторной работы следует иметь в виду, что команды mov, один из операндов которых размещен в регистрах AL|AX|EAX, а второй задан смещением в памяти, имеют особый формат. Это связано с тем, что указанные команды унаследованы от более старого прототипа – 16-ти разрядного процессора 8080. Эти команды не содержат байта адресации и имеют коды операции $A0_{16}$ .. $A3_{16}$, два последних бита которых также расшифровываются, как D и W. При этом D=1 соответствует «из регистра», а D=0 – «в регистр», W имеет те же значения, что и в предыдущем случае. Сразу за кодом операции этих команд следуют 4-х байтовые (с учетом 32-х разрядной адресации) смещения.

Примеры:

1) **`mov AL,A ;`** при адресе A соответствующем 403000

101000DW Смещение 32 разряда
10100000 00000000 00110000 01000000 00000000
A 0 0 0 3 0 4 0 0 0

2) **`mov B,AX ;`** при адресе B соответствующем 403004

префикс1 `101000DW` Смещение 32 разряда

01100110 10100011 00000100 00110000 01000000 00000000
6 6 A 0 0 4 3 0 4 0 0 0

2 Команды целочисленной арифметики IA-32

Процессоры семейства IA-32 поддерживают арифметические операции над однобайтовыми, двухбайтовыми и четырехбайтовыми целыми числами.

Размер операндов при этом определяется:

- объемом регистра, хранящего число – если хотя бы один операнд находится в регистре;
- размером числа, заданным директивой определения данных;
- специальными описателями, например, BYTE PTR (байт), WORD PTR (слово) и DWORD PTR (двойное слово), если ***ни один операнд не находится в регистре и размер операнда отличен от размера, указанного директивой определения данных***.

1. Команда пересылки данных – пересылает число размером 1, 2 или 4 байта из источника в приемник:

`mov` Приемник, Источник

Допустимые варианты:

```
mov   reg, reg
mov   mem, reg
mov   reg, mem
mov   mem, imm
mov   reg, imm
mov   r/m16, sreg
mov   sreg, r/m16
```

Здесь и далее используются следующие условные обозначения:

reg – содержимое указанного регистра данных;

mem – содержимое памяти по указанному адресу;

imm – непосредственно заданный в команде операнд (литерал);

r/m16 – содержимое указанного 16-ти разрядного регистра или 16-ти разрядное содержимое памяти;

sreg – содержимое сегментного регистра.

Примеры команд:

а) **mov AX,BX**

б) **mov ESI,1000**

в) **mov 0[DI],AL**

г) **mov AX,code ;** code – адрес начала сегмента

mov DS,AX

2. Команда перемещения и дополнения нулями **–** при перемещении значение источника помещается в младшие разряды, а в старшие заносятся нули. Формат команды:

movzx Приемник, Источник

Допустимые варианты:

movzx r16/r32, r/m8

movzx r32, r/m16

Примеры команд:

а) **movzx EAX,BX**

б) **movzx SI,AH**

3. Команда перемещения и дополнения знаковым разрядом **–** команда выполняется аналогично и имеет формат, совпадающий с форматом предыдущей команды, но в старшие разряды заносятся знаковые биты:

movsx Приемник, Источник

4. Команда обмена данных

XCHG Операнд1, Операнд 2

Допустимые варианты:

```
xchg    reg,reg
xchg    mem,reg
xchg    reg,mem
```

5-6. Команды записи слова или двойного слова в стек и извлечения из стека

```
PUSH    imm16 / imm32 / r16 / r32 / m16 / m32
POP     r16 / r32 / m16 / m32
```

Если в стек помещается 16-ти разрядное значение, то значение ESP:= ESP-2, если помещается 32 разрядное значение, то ESP := ESP-4.

Если из стека извлекается 16-ти разрядное значение, то значение ESP := ESP+2, если помещается 32 разрядное значение, то ESP := ESP+4.

Примеры:

```
push    SI
pop     word ptr [EBX]
```

8-9. Команды сложения – складывает операнды, а результат помещает по адресу первого операнда. В отличие от ADD команда ADC добавляет к результату значение бита флага переноса CF.

ADD Операнд1, Операнд2

ADC Операнд1, Операнд2

Допустимые варианты:

```
add    reg,reg
add    mem,reg
add    reg,mem
add    mem,imm
add    reg,imm
```

10-11. Команды вычитания – вычитает из первого операнда второй и результат помещает по адресу первого операнда. В отличие от SUB команда SBB вычитает из результата значение бита флага переноса CF. Допустимые варианты те же, что и у сложения.

SUB Операнд1, Операнд2

SBB Операнд1, Операнд 2

13-14. Команды добавления/вычитания единицы

```
INC     reg/mem
DEC     reg/mem
```

Примеры:

```
inc     AX
dec   byte ptr 8[EBX,EDI]
```

15-16. Команды умножения

MUL <Операнд2>

IMUL <Операнд2>

Допустимые варианты:

```
mul/imul   r|m8   ; AX= AL*<Операнд2>
mul/imul   r|m16  ; DX:AX= AX*<Операнд2>
mul/imul   r|m32  ; EDX:EAX= EAX*<Операнд2>
```

В качестве второго операнда нельзя указать непосредственное значение!!!

Регистры первого операнда в команде не указываются. Местонахождение и длина результата операции зависит от размера второго операнда.

Пример:

```
mov    AX,4
imul    word ptr A ;   DX:AX:=AX*A
```

17-19. Команды «развертывания» чисел – операнды в команде не указываются. Операнд и его длина определяются кодом команды и не могут быть изменены. При выполнении команды происходит расширение записи числа до размера результата посредством размножения знакового разряда.

Команды часто используются при программировании деления чисел одинаковой размерности для обеспечения удвоенной длины делимого

```
CBW  ; байт в слово AL -> AX
CWD  ; слово в двойное слово AX -> DX:AX
CDQ        ; двойное слово в учетверенное EAX -> EDX:EAX
CWDE       ; слово в двойное слово AX -> EAX
```

20-21. Команды деления

```
DIV  <Операнд2>

IDIV <Операнд2>
```

Допустимые варианты:

```
div/idiv  r|m8   ;  AL= AX:<Операнд2>, AH - остаток
div/idiv  r|m16  ;  AX= (DX:AX):<Операнд2>, DX - остаток
div/idiv  r|m32  ;  EAX= (EDX:EAX):<Операнд2>, EDX - остаток
```

В качестве второго операнда нельзя указать непосредственное значение!!!

Пример:

```
mov   AX,40
cwd
idiv    word ptr A;   AX:=(DX:AX):A
```

3 Пример линейной программы

Разработать приложение, вычисляющее X = (A+B)(B-1)/(D+8).

Ниже показан только текст, который добавляется к шаблону.

```
          .DATA
A         SWORD 25
B         SWORD -6
D         SWORD 11

          .DATA?
X         SWORD ?

          .CODE
Start:    mov    CX,D
          add    CX,8; CX:=D+8
          mov    BX,B
```

```
dec     BX   ; BX:=B-1
mov     AX,A
add     AX,D; AX:=A+D
imul    BX   ; DX:AX:=(A+D)*(B-1)
idiv    CX   ; AX:=(DX:AX):CX
mov     X,AX
. . .
```

4 Организация ввода-вывода

Библиотека MASM32.lib содержит специальные подпрограммы ввода-вывода консольного режима:

1. Процедура ввода:

StdIn PROC lpszBuffer:DWORD, bLen:DWORD

Первый операнд – адрес буфера ввода, второй – размер буфера ввода (до 128 байт). В буфере ввода введенная строка завершается символом конца строки (13, 10).

2. Процедура замены символов конца строки нулем:

StripLF PROC lpszBuffer:DWORD ; буфер ввода

3. Функция преобразования завершающейся нулем строки в число:

atol proc lpszBuffer:DWORD ; результат – в EAX

Пример программирования ввода:

```
        .DATA
zapros  DB    'Input value:',13,10,0 ; запрос
buffer  DB    10 dup ('0')     ; буфер ввода

        .CODE
        . . .
vvod:   Invoke StdOut,ADDR zapros
        Invoke StdIn,ADDR buffer,LengthOf buffer
```

```
            Invoke StripLF,ADDR buffer
```

; Преобразование в SDWORD

```
            Invoke atol,ADDR buffer ;
```
результат в EAX

```
            . . .
```

4. Процедура вывода *завершающейся нулем* строки в окно консоли:

StdOut PROC lpszBuffer:DWORD ; буфер вывода, зав. нулем

5. Процедура преобразования числа в строку:

dwtoa PROC public dwValue:DWORD, lpBuffer:PTR BYTE

Пример программирования вывода:

```
            .DATA
result      DWORD ?          ;  поле результата
string      DB    13,10,'Result ='  ;  заголовок вывода
resstr      DB    16 dup (?)        ;  выводимое число
            .CODE
            . . .
```

; Преобразование

```
            Invoke dwtoa,result,ADDR resstr
```

; Вывод

```
            Invoke StdOut,ADDR string
            . . .
```

Версия программы раздела 3 с вводом-выводом:

; Template for console application

```
            .586
            .MODEL  flat, stdcall
            OPTION CASEMAP:NONE

Include kernel32.inc
```

```
Include masm32.inc

IncludeLib kernel32.lib
IncludeLib masm32.lib

          .CONST
MsgExit   DB    13,10,"Press Enter to Exit",0AH,0DH,0

          .DATA
B         SWORD -6
D         SWORD 11
X         SWORD ?
fX        SWORD 0     ; старшее слово результата
Zapros    DB    13,10,'Input A',13,10,0
Result    DB    'Result='
ResStr    DB    16 DUP (' '),0

          .DATA?
A         SWORD ?
fA        SWORD ?    ; старшее слово переменной A
Buffer    DB    10 DUP (?)
inbuf     DB    100 DUP (?)

          .CODE
Start:    Invoke StdOut,ADDR Zapros
          Invoke StdIn,ADDR Buffer,LengthOf Buffer
          Invoke StripLF,ADDR Buffer
; Преобразование в SDWORD
          Invoke atol,ADDR Buffer ;результат в EAX
```

```
        mov     DWORD PTR A,EAX
; Вычисления
        mov     CX,D
        add     CX,8; CX:=D+8
        mov     BX,B
        dec     BX   ; BX:=B-1
        mov     AX,A
        add     AX,D; AX:=A+D
        imul    BX   ; DX:AX:=(A+D)*(B-1)
        idiv    CX   ; AX:=(DX:AX):CX
        mov     X,AX
; Преобразование
        Invoke dwtoa,X,ADDR ResStr
; Вывод
        Invoke StdOut,ADDR Result

        XOR     EAX,EAX
        Invoke StdOut,ADDR MsgExit
        Invoke StdIn,ADDR inbuf,LengthOf inbuf

        Invoke ExitProcess,0
        End     Start
```

5 Задание

1. Разработать программу, вычисляющую заданное выражение. Просмотреть в отладчике и зафиксировать в отчете ход выполнения вычислений (покомандно). Убедиться в правильности программы.

2. Посмотреть в отладчике форматы 3-4 команд **mov** и расшифровать двоичные коды этих команд, используя материалы теоретической части.

Варианты заданий

1. $a = (b^2-(c+1)*d)/b$
2. $c = a/c - k + (d+1)*5$
3. $b = a*j - j^2/(k+2)$
4. $a = a*(a+b/4)/(k-1)$
5. $d = 3*a*x/[5*(b-5)]$
6. $a = a*x-3*(b+3/k)$
7. $a = a^3/3 -c*(x+3)$
8. $d= (k-5)^2/4+2*k$
9. $d = a*x/2 – (a+b)/2$
10. $a = (b^2-2*b)/(3a+b)$
11. $b = (a^2-b^2)/2+a*(k+1)$
12. $e = (a-c)^2+2*a*c/k$
13. $g = b^2(f-2)/k-(n+r)*m$
14. $f = (a^2-b)/(y+a)+(a-1)^2$
15. $s = q^3- 2*a*q+a^2/q$
16. $n = q^2/3 - a*d +5$
17. $m = a*c^2- b * a/c+a/b$
18. $x = a*y*(b-a)/4 + a^2-2$
19. $n = a*x^2- b*y/a+x/(y+a)$
20. $k = (l-a)^2/c+k-l+c/2$
21. $s = (a-b^2)/(y-a)+a^2 - c$
22. $b = (m-5)*(m+2) + m+a/2$
23. $c = (a+b)/d - d^2* a - b$
24. $a = b*(c-d) - c/(d-1)$
25. $q = a^2/2 - b^3/(4- a + b$
26. $s = a*b/2 – k + a/2 -b$

6 Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы и таблицы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Отчет должны иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента.

Отчет должен содержать:

1) текст задания;

2) текст программы с комментариями;

3) вручную посчитанные тесты;

4) результаты, полученные при выполнении программы;

5) выводы.

7 Контрольные вопросы

1. Что такое машинная команда? Какие форматы имеют машинные команды процессора IA32? Чем различаются эти форматы?

2. Назовите мнемоники основных команд целочисленной арифметики. Какие форматы для них можно использовать?

3. Сформулируйте основные правила построения линейной программы вычисления заданного выражения.

4. Почему ввод-вывод на языке ассемблера не программируют с использованием соответствующих машинных команд? Какая библиотека используется для организации ввода-вывода в данной лабораторной?

5. Расскажите, какие процедуры использую для организации ввода вывода. Какие операции выполняет каждая процедура?

Литература

1. Иванова Г.С. Презентации курса лекций Машинно-зависимые языки и основы компиляции. М.: МГТУ им. Н.Э. Баумана, 2015.

2. Иванова Г.С., Ничушкина Т.Н. Основы программирования на ассемблере IA-32. Учебное пособие: Электронное учебное издание. М.: МГТУ им. Н.Э. Баумана, 2010. Способ доступа: http://e-learning.bmstu.ru/moodle/mod/resource/view.php?id=35.

3. Самарев Р.С. Создание схем алгоритмов средствами Microsoft Visio и OpenOffice Draw. Методические указания по выполнению лабораторных

работ: Электронное учебное издание. – М.: МГТУ им. Н.Э. Баумана, 2010. – http://e-learning.bmstu.ru/moodle/mod/resource/view.php?id=35.

Приложение А. Наиболее важные настройки среды RADAsm

1. Настройка путей

Для того, чтобы настроить пути автовызова среды используется пункт меню **Настройки/Установить пути** (см. рисунок П.1).

При этом:

$A – путь к каталогу, содержащему Masm32;

$R – путь к каталогу установки RADAsm;

$E – путь к каталогу, содержащему папку отладчика OlleDBG;

$B – путь к каталогу обрабатывающих программ Masm32;

$H – путь к каталогу, содержащему файлы справок;

$I – путь к каталогу подставляемых файлов Masm32 (.inc);

$L – путь к каталогу автовызова двоичных образов стандартных подпрограмм (функций API и т.п.);

$M – путь к каталогу, содержащему файлы макросов;

$P – путь к каталогу, в котором по умолчанию создаются проекты;

$T – путь к каталогу шаблонов новых проектов.

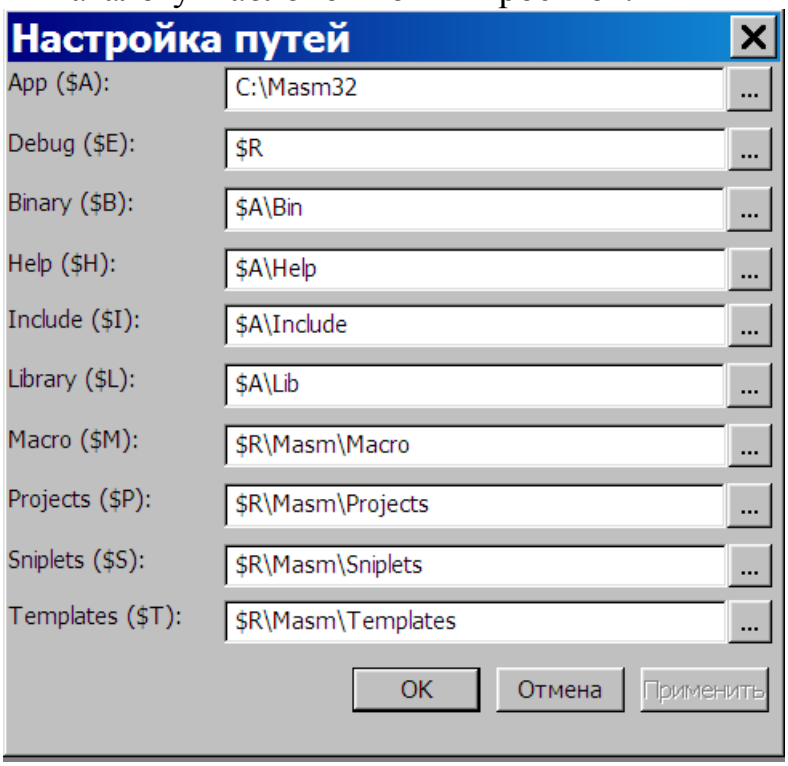

Рисунок П.1 – Настройки путей

Если к проекту с помощью INCLUDE подключаются другие файлы, не заданные явно в проекте, то путь к ним указывается полностью.

2. Формирование командных строк для вызова ассемблера, компоновщика или отладчика

Для формирования командных строк используется специальный формат, который затем в среде распознается и транслируется в команду. Настройки выполняются при создании проекта (на последнем шаге) или при вызове пункта меню **Проект/Настройки проекта** (см. рисунок П.2).

Помеченные команды доступны для вызова:

Compile RC – компилятор ресурсов (окон, кнопок и т.п.).

Assemble – ассемблирование.

Link – компоновка.

Build – сборка.

Go – компиляция ресурсов, ассемблирование, сборка и запуск.

Run – выполнение.

Rub w/Debug – выполнение в отладчике.

Go all – компиляция ресурсов, ассемблирование всех модулей, сборка и запуск.

Assemble modules – ассемблирование модулей, оформленных в отдельных файлах.

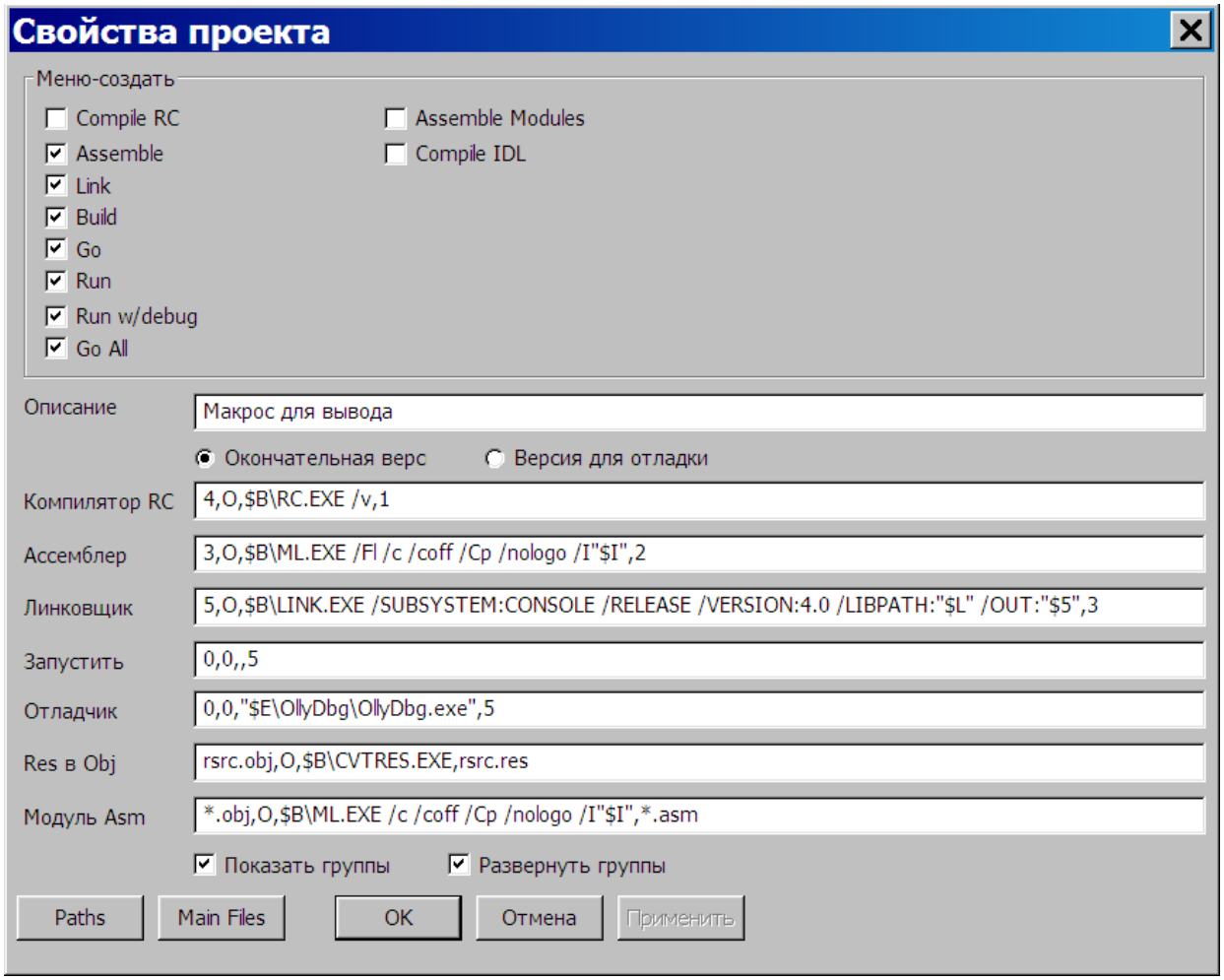

Рисунок П.2 – Настройки командных строк

Для записи команды используется специальный формат:

<Номер или имя создаваемого или пересоздаваемого файла>,

<Указание окна вывода результата>,

<Имя вызываемой программы с опциями>,

<Номер или имя исходного файла>

В соответствии с принятой автором среды системой все типы файлов, используемых в проекте, имеют номера, например:

0=.rap – файл проекта RADAsm;

1=.rc – файл ресурсов;

2=.asm – исходная программа на ассемблере;

3=.obj – объектный модуль (результат ассемблирования);

4=.res – результат компиляции файла ресурсов;

5=.exe – исполняемый файл;

7=.dll – файл динамически загружаемой библиотеки;

8=.txt – текстовый файл;

9=.lib – файл библиотеки;

10=.mak – командный файл;

11=.hla – файл ассемблера высокого уровня;

12=.com – исполняемый com файл.

Второй параметр – указание окна вывода результатов:

O – сообщения об ошибках выводятся в окно Output RADAsm (если указано OT, то туда же выводится командная строка);

C – сообщения об ошибках выводятся в окно Console.

Далее идет имя обрабатывающей программы с опциями и имя исходного модуля. Более подробно правила записи командных строк приведены в справке среды RADAsm.

Содержание