Министерство образования Республики Беларусь

«ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. ЕВФРОСИНИИ ПОЛОЦКОЙ»

Факультет информационных технологий

Кафедра технологий программирования

Методические указания для выполнения лабораторной работы №1 по курсу «Конструирование программного обеспечения»

«Знакомство со средой программирования для языка низкого уровня. Разработка, трансляция, компоновка, выполнение и отладка на примере простой программы»

Полоцк, 2022 г.

ЦЕЛЬ РАБОТЫ

Целью работы является освоение инструментальных средств создания программ на языке ассемблера.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Архитектура компьютера

Структурные схемы компьютеров мало чем отличаются у разных моделей компьютеров. У всех компьютеров есть оперативная память, процессор, внешние устройства. Различными являются способы, средства и используемые ресурсы, с помощью которых компьютер функционирует как единый механизм. Совокупность функциональных программно-управляемых свойств компьютера называют архитектурой.

Архитектура ЭВМ – это абстрактное представление ЭВМ, которое отражает структурную, схемотехническую и логическую организацию. Приступить к изучению языка ассемблера любого компьютера имеет смысл только после выяснения того, какая часть из вышеперечисленных аспектов архитектуры оставлена видимой и доступной для программирования. Это так называемая программная модель компьютера. В приложении 1 приведена программная модель микропроцессора Intel (Pentium III). Как видно из приведенной модели, ее основу составляют регистры общего назначения, состояния и ряд других. В разных моделях компьютеров одной и той же фирмы-производителя количество и разрядность регистров могут существенно изменяться. Но в качестве базовой модели для языка программирования микропроцессоров Intel используется 14-регистровая система, на которой остановимся далее подробнее.

Универсальные регистры *AX*, *BX*, *CX*, *DX* имеют разрядность 16 бит (2 байта) и могут использоваться для временного хранения любых данных, при этом можно работать с каждым регистром целиком, а можно отдельно с каждым его байтом. Старшие байты *РОН* имеют имена *AH*, *BH*, *DH*, *CH*, а младшие – *AL*, *BL*, *DL*, *CL*. Регистры *AL*, *AH* образуют соответственно младший и старший байты условного регистра *AX*.

Кроме того, каждый из РОН может использоваться как специальный регистр при выполнении некоторых команд программы:

– регистр AX, аккумулятор, 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, а также при преобразовании десятичных чисел и выполнении над ними арифметических операций;

– регистр AH используется при умножении и делении байтов;

– регистр BX, базовый регистр, часто используется при адресации данных в памяти;

– регистр CX, счетчик, 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, используется при умножении и делении слов. Кроме этого используется в операциях ввода-вывода как номер порта.

В микропроцессорах Intel программы и данные хранятся в отдельных областях памяти - сегментах, с объемом до 64 КБ (килобайт). Одновременно микропроцессор может иметь дело с 4 сегментами, начальные адреса которых содержатся в ***сегментных регистрах*** CS, DS, SS, ES. Эти регистры выполняют следующие функции:

– регистр сегмента команд CS указывает на сегмент, содержащий текущую исполняемую программу. Для вычисления адреса следующей подлежащей исполнению команды процессор складывает значение CS умноженное на 16 с указателем команд IP;

– регистр сегмента стека SS указывает на текущий сегмент стека – области памяти предназначенной для временного хранения данных и адресов;

– регистр сегмента данных DS указывает на текущий сегмент данных, который обычно содержит используемые в программе переменные;

– регистр дополнительного сегмента ES указывает на текущий дополнительный сегмент, который используется при выполнении операций над строками.

Регистры смещений *IP, SP, BP, SI, DI* используются для хранения относительных адресов ячеек памяти внутри сегментов (иначе говоря, смещений относительно начала сегментов):

– регистр IP хранит смещение адреса текущей команды программы;

– регистр SP указывает на вершину стека – смещение относительно начала стека;

– в регистр BP записывается начальный адрес поля памяти, непосредственно отведенного под стек;

– регистры SI и DI предназначены для хранения адресов индексов источника и приемника данных при операциях над строками и другими структурами данных.

Регистр флагов FL представляет собой 16-битовый регистр, где фиксируется информация о текущем состоянии процессора (рис. 1).

15 0

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Рис. 1. Регистр флагов

Флаг ***OF*** называется флагом переполнения и OF=1 свидетельствует о наличии ошибки в операциях над числами со знаком.

Флаг направления ***DF*** используется в командах работы со строками. При DF=1 регистр индекса, используемый в командах работы со строками, увеличивается на 1 при каждом следующем выполнении команды, при DF=0 – регистр индекса на 1 уменьшается.

Флаг прерывания ***IF*** обычно он устанавливается в 1 и такое его значение позволяет исполняемой программе пользователя реагировать на прерывания. Однако, когда вызывается программа обработки прерывания, флаг IF устанавливается в 0, чтобы никакие другие прерывания не могли помешать текущей обработке прерывания.

Флаг ***TF*** называется флагом трассировки, при его значении, равном 1, разрешается выполнение программы по шагам.

Флаг знака ***SF*** устанавливается в 1, если в результате выполнения операции над числами со знаком, получается отрицательное число.

Флаг нуля ***ZF*** устанавливается в 1, если результатом операции является нулевое значение.

Флаг вспомогательного переноса ***AF*** используется в двоично-десятичной арифметике. Этот флаг устанавливается в 1, если результат такой операции не является десятичной цифрой.

Флаг четности ***PF*** устанавливается в 1, если результат операции имеет четное количество битов, равных 1, в двоичном представлении результата.

Флаг ***CF*** называется флагом переноса и в него заносится перенос (или заем) из знакового (старшего) разряда числа.

!ВАЖНО!

Нужно уяснить, что не все команды программы на Ассемблере устанавливают в 0 или в 1 флаги. Причем, выполнение тех или иных команд связано с установкой конкретных флагов. Обратите внимание на это обстоятельство при изучении команд ассемблера.

Процедуры формирования программы

У большинства существующих реализаций ассемблера нет интегрированной среды, подобной Turbo Pascal или Turbo C. Поэтому для выполнения функций по

вводу кода программы, ее трансляции, редактированию и отладке необходимо использовать отдельные служебные программы.

Последовательность процедур формирования программы на языке ассемблера и совокупность порождаемых файлов показана на рисунке 2.

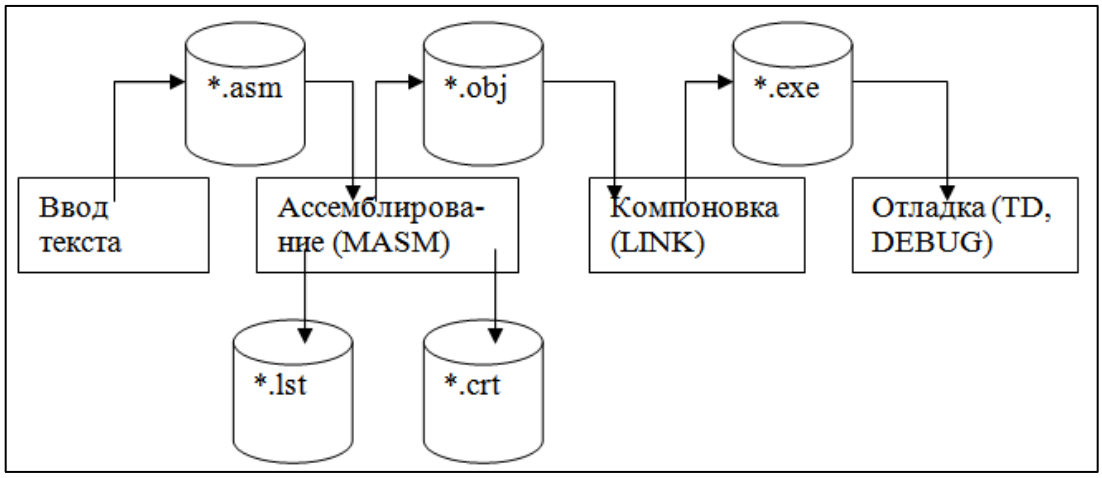

Рис. 2. Процесс обработки ассемблер-программы

В процессе формирования программы на языке ассемблера выделено 4 этапа:

- ввод исходного кода программы текстовым редактором,
- трансляция программы,
- создание загрузочного модуля,
- отладка программы.

Начальной процедурой создания программы на языке Ассемблера является ввод исходного текста программы в файл с расширением ***.asm***. При этом может быть использован любой текстовый редактор, сохраняющий текст в виде стандартных кодов ASCII, например, редактор *NC* или блокнот. Основное требование к редактору, заключается в том, чтобы он (редактор) не вставлял посторонних символов (специальных символов форматирования).

Следующим шагом формирования программы является компиляция, которая носит специфическое название ассемблирование. Этот этап может быть выполнен программами *ASM, MASM* или *TASM* (сложность программ-компиляторов растет в указанной последовательности). Результатом выполнения этого этапа является программа в машинных кодах с расширением ***.obj***, или, иначе, объектная программа, уже “понятная” микропроцессору. Естественно, перевод состоится лишь в том случае, если исходный текст программы не содержит ошибок. Одновременно с объектным файлом могут быть созданы файлы листинга (****.lst***) и перекрестных ссылок (****.crf***). Рекомендуется файл листинга создавать обязательно, поскольку при наличии ошибок в листинге описывается характер ошибки сразу после ошибочной команды, что значительно упрощает внесение исправлений, особенно на этапе обучения.

Файл листинга содержит код ассемблера исходной программы, машинный (объектный) код каждой команды и ее смещение в кодовом сегменте (значение регистра IP). Кроме того, сообщения о найденных синтаксических ошибках в программе помещаются непосредственно после ошибочной команды (бывают исключения, когда ошибка не в самой команде, а ранее нее, но эти ситуации встречаются редко). Строки в файле листинга имеют следующий формат:

```
<глубина_вложенности>, <номер_строки>, <смещение>, <машинный_код>,
<исходный код>
```

где ***<глубина_вложенности>*** – уровень вхождения программного блока (программы, модуля, макроопределения, процедуры) в файл,

<номер_строки> – номер строки в файле листинга, он фигурирует в сообщениях об ошибках, но не обязательно совпадает с номером команды в исходном тексте,

<смещение> – смещение в байтах текущей команды относительно начала сегмента кода,

<машинный_код> – машинное представление команды ассемблера, записанной правее в той же строке в поле <исходный код>, а <исходный код> является не чем иным, как записанной пользователем командой языка ассемблер.

Однако объектная программа еще не является законченной и исполняемой, т.к. в ней определены не все адреса (программа не является “перемещаемой”) и не объединены части (блоки) программы, которые могут транслироваться отдельно с целью более простой отладки. Преобразование объектной программы в исполняемую (компоновка) выполняется загрузчиком (редактором связей) LINK либо TLINK (в зависимости от используемой программы ассемблирования: для *ASM, MASM – LINK*, для *TASM – TLINK*).

Чтобы проверить работоспособность созданной программы и увидеть результаты ее работы (если не использован вывод на дисплей), применяют программу отладчик. Тестирование и отладка исполняемой программы выполняется отладчиком TD или DEBUG.

Отладчик ***td.exe***, разработанный фирмой Borland International представляет собой оконную среду отладки программ на уровне исходного текста на языках Pascal, C, ассемблер. Основные возможности отладчика, наиболее широко используемые студентами – это:

– выполнение трассировки программы в прямом направлении, при котором за 1 шаг выполняется одна машинная инструкция;

– просмотр и изменения состояния аппаратных ресурсов микропроцессора во время командного выполнения программы.

Управлять работой отладчика можно с помощью системы меню двух типов:

– глобального, находящегося в верхней части экрана и постоянно доступного. Вызов меню осуществляется нажатием клавиши F10;

– локального, учитывающего особенности окон и становящегося активным щелчком правой мыши или нажатием клавиш Alt+F10.

Специфика программы на ассемблере в том, что делать выводы о правильности ее функционирования можно, отслеживая работу на уровне микропроцессора, обращая внимание на то, как изменяется состояние ресурсов микропроцессора и компьютера в целом. Общее поведение программы позволяет просмотреть режим безусловного выполнения, который вызывается нажатием клавиши F9. Однако для детального изучения работы программы рекомендуется применять режим выполнения программы по шагам, для вызова которых выбираются пункты меню ***Run -> Trace into*** (прерывание или внутренняя процедура будут выполняться по шагам) или ***Run -> Step over*** (вызов процедуры или прерывание отрабатываются как одна обычная команда). При этом используется окно ***CPU***, вызов которого осуществляется через глобальное меню командой ***View -> CPU***. Окно ***CPU*** состоит из 5 подчиненных окон:

– окно с исходной программой в машинных кодах,

– ***Register*** – окно регистров микропроцессора, отражающее текущее состояние регистров,

– окно флагов, отражающее состояние флагов микропроцессора в соответствии с таблицей 1;

– окно стека, в котором отражается содержимое области памяти, отведенной для стека,

– окно дампа оперативной памяти Dump, отражающее содержимое области памяти по адресу, указанному в левой части окна. В окне можно увидеть содержимое произвольной области памяти, для этого нужно в локальном меню, вызываемом по щелчку правой кнопки мыши, выбрать нужную команду.

Таблица 1 – Обозначения и значения флагов

Имя флага	**Обозначение флага**	**Установлен**	**Сброшен**
Флаг переполнения	O	1	0
Флаг направления	D	1	0
Флаг прерывания	I	1	0
Флаг знака	S	1(<0)	0(>0)
Флаг нуля	Z	1	0

Флаг вспомогательного переноса	A	1	0
Флаг четности	P	1	0
Флаг переноса	C	1	0

Рекомендуемый порядок работы с отладчиком:

a) вызвать на выполнение ***td.exe.***;

b) выбрать файл исполняемой программы, набрав комбинации клавиш ***FILE ->OPEN*** и имя Вашей программы в окне запроса. После ответа OK на сообщение об отсутствии символьной таблицы в окно ***CPU*** загружается программа с нулевого адреса относительно начала сегментного регистра кодов (для приведенного в конце описания лабораторной работы примера это будет команда PUSH DS);

c) выбрать режим пошагового выполнения ***Run -> Step over***. В окне ***CPU*** появляется окрашенный треугольник между относительным адресом команды и машинным кодом команды. Он показывает очередную команду, которая будет выполнена процессором после нажатия функциональной клавиши F8. Изменения, которые происходят в сегментных регистрах после выполнения команды, отмечаются белым цветом соответствующей строки в окне регистров. Пошаговый процесс выполнять до тех пор, пока не появится сообщение об окончании программы (с ключевым словом terminated);

d) после выполнения команд, связанных с изменением содержимого ячеек памяти, нужно просматривать эти изменения командой ***VIEW -> DUMP***. При отсутствии мыши скрыть окно дампа памяти можно нажатием функциональной клавиши F6.

Структура программы

Программа на языке ассемблера представляет собой последовательность операторов, описывающих выполняемые операции. Оператором (строкой) исходной программы может быть или команда, или псевдооператор (директива) ассемблера. Команды выполняются в процессе решения задачи на компьютере, а директивы – в процессе ассемблирования (трансляции) программы. Следовательно, в отличие от команд директивы сообщают ассемблеру (транслятору), что ему делать с командами и данными, которые вводятся в программе. Ниже в таблице 2 перечисляются наиболее часто используемые директивы ассемблера.

Таблица 5 – Синтаксис и функции псевдооператоров (директив)

Псевдооператор	**Формат и Функция**
Определения данных DB	[имя] DB выражение [,…….] определяет переменную или присваивает ячейке памяти начальное значение. Резервирует 1 или более байт (при наличии запятых)
DW	[имя] DW выражение [,…….] аналогично предыдущему резервирует двухбайтовые слова
DD	[имя] DD выражение [,…….] Резервирует 4-х байтовые двойные слова
Определения сегмента или процедуры SEGMENT	Имя_cer SEGMENT [тип_выравнивания (подгонки)] [тип_связи] [‘класс’] . . . . Имя_cer ENDS Определяет границы сегмента программы. Обязательно содержит начало описания Имя_cer SEGMENT и окончание описания Имя_cer ENDS
ASSUME	ASSUME регистр_cer: Имя_cer [,…….] Или ASSUME регистр_cer: NOTHING Сообщает Ассемблеру, какой регистр сегмента связан с соответствующим сегментом программы. Оператор ASSUME регистр_cer: NOTHING отменяет действие всех предыдущих операторов ASSUME для данного регистра
PROC	Имя PROC [NEAR] или Имя PROC FAR . . . . RET имя ENDP Присваивает имя последовательности операторов. Должно иметь начало PROC и окончание ENDP
Псевдооператор	Формат и Функция
Управление трансляцией END	END [метка точки входа] Отмечает конец исходной программы

Внешние ссылки PUBLIC	PUBLIC идентификатор Делает определенный ранее идентификатор доступным другим модулям программы, которые впоследствии должны быть присоединены к данному модулю
EXTERN	EXTERN имя: тип [, ….] Указывает, что имя определено в другом модуле программы
INCLUDE	INCLUDE файл вставляет содержимое указанного файла в текущий файл исходной программы
Определение идентификаторов EQU	Имя EQU текст или Имя EQU числовое_выражение Постоянно присваивает идентификатору имя текст или числовое_выражение Имя = числовое_выражение Числовое_выражение присваивается идентификатору имя, но может быть переприсвоено

Обязательные требования к структуре ASM-программы следующие:

– программа может использовать четыре сегмента памяти, начальные адреса которых должны быть загружены в регистры микропроцессора CS,DS, ES, SS, а сами сегменты в явном виде определены в программе в виде операторных скобок

– ***имя_сегмента segment***

– . . . . . .

– ***имя_сегмента ends***,

например,

```
DSEG SEGMENT PARA PUBLIC ‘DATA’
SOURCE DB 10,20,30,40
DEST DB 4 DUP(?)
DSEG ENDS;
```

– в программе должно быть указание, какие сегментные регистры закрепляются за используемыми именами регистров, например:

```
ASSUME CS:CSEG, DS:DSEG, SS:STACK.
```

При исполнении программы адреса сегментных регистров CS, SS, ES в соответствии с вышеприведенными указаниями загружаются автоматически;

– сегмент данных DS в EXE-программе не может быть загружен автоматически, поскольку он используется программой-загрузчиком LINK для формирования начального адреса служебной области памяти – префикса программного сегмента (PSP), непосредственно предшествующего любой исполняемой программе. PSP – это группа служебных слов в оперативной памяти, формируемая для каждой загружаемой программы пользователя и занимающая обычно 256 байт (100H байт), именно адрес этой области записывается в регистр DS. Поэтому в самом начале исполняемой программы этот сегмент инициируется принудительно: сначала в стек записывается адрес возврата к служебной области в виде 2-ух слов – содержимого регистра DS и нулевого смещения; затем в регистр DS записывается адрес сегмента данных исполняемой программы, например, как показано ниже:

```
PUSH DS ; поместить в стек адрес PSP
SUB AX,AX ; обнулить регистр AX
PUSH AX ; поместить в стек смещение адреса возврата=0
MOV AX,DSEG ; инициировать адрес сегмента данных
MOV DS,AX ; загрузить адрес в регистр DS;
```

– в исходной программе обязательно должна быть определена метка для первой команды, с которой начнется выполнение программы. Это может быть собственно метка или имя процедуры, как показано в приведенной ниже программе. Имя этой метки обязательно должно быть указано в конце программы в качестве операнда директивы ***END***, например,

```
END OUR_PROG;
```

– обеспечение выхода из программы, например, используя функцию 4C прерывания 21H, как показано ниже:

```
MOV AX,4C00H
INT 21H
```

или оформив основную программу как процедуру с атрибутом FAR и стандартным выходом из процедуры RET, как показано в приведенной ниже программе.

Пример программы на Ассемблере

```
TITLE EX_PROG
PAGE ,132
STACK SEGMENT PARA STACK ‘STACK’
```

```
DB 64 DUP('STACK ') ; Область стека
STACK ENDS
DSEG SEGMENT PARA PUBLIC 'DATA'
SOURCE DB 10,20,30,40 ; эта таблица будет скопирована
DEST DB 4 DUP(?) ; в эту таблицу в обратном порядке
DSEG ENDS
SUBTTL ОСНОВНАЯ ПРОГРАММА
PAGE
CSEG SEGMENT PARA PUBLIC 'CODE'
ASSUME CS:CSEG, DS:DSEG, SS:STACK
OUR_PROG PROC FAR
;занести в стек такие начальные значения, чтобы программа
; могла возвратить управление отладчику
PUSH DS ; поместить в стек номер блока адреса возврата
SUB AX,AX ; обнулить регистр AX, тоже можно сделать командой MOV AX,0
PUSH AX ; поместить в стек значение адреса возврата=0
; инициировать адрес сегмента данных
MOV AX,DSEG
MOV DS,AX
; присвоить элементам таблицы DEST нулевые начальные значения
MOV DEST,0 ; обнуление 1-ого байта
MOV DEST+1,0 ; обнуление 2-ого байта
MOV DEST+2,0 ; обнуление 3-его байта
MOV DEST+3,0 ; обнуление 4-ого байта
; скопировать таблицу SOURCE в таблицу DEST в обратном порядке, в
качестве промежуточной ячейки пересылки использовать регистр AL
MOV AL, SOURCE
MOV DEST+3,AL
MOV AL, SOURCE+1
MOV DEST+2,AL
MOV AL, SOURCE+2
MOV DEST+1,AL
MOV AL, SOURCE+3
MOV DEST,AL
RET ; возврат управления отладчику db
OUR_PROG ENDP
CSEG ENDS
END OUR_PROG
```

Порядок выполнения работы

1. Набрать приведенную в тексте программу на ассемблере с использованием редактора текста.

2. Оттранслировать программу в объектный код.

3. Скомпоновать программу (получить исполнимый файл). Изучить листинг программы.

4. Провести отладку программы и проверить получаемые результаты.

5. Внести в программу следующие изменения: задать исходную таблицу SOURCE из 5 двухбайтовых шестнадцатеричных переменных и скопировать эту новую таблицу в DEST.

6. В сегменте данных определить переменные, заполнив их следующими значениями:

- 5 байтов A, B, C, D, E;
- 5 двухбайтовых слов AA, BB, CC, DD, EE;
- 5 двойных слов AAAA, BBBB, CCCC, DDDD, EEEE;

7. Получить исполнимый файл программы с данными пункта 6 и изучить дамп памяти данных с целью уяснения механизма выравнивания.

Содержание отчета

Отчет должен включать:

а) титульный лист;

б) формулировку цели работы;

в) описание результатов выполнения пунктов 3-7:

- листинги программ;
- результаты выполнения программ;

г) выводы, согласованные с целью работы.

Контрольные вопросы

1. Какие группы регистров выделяются в микропроцессоре и каковы особенности их использования?

2. Какую функцию в микропроцессоре выполняет регистр флагов?

3. Как используется регистр команд IP?

4. Какие шаги необходимо выполнить для получения из программы на языке ассемблера исполняемого модуля?

5. Прокомментируйте содержание листинга программы.

6. В каких окнах и в каком виде отображается состояние микропроцессора при отладке программ с применением отладчика td.exe?