Государственный комитет Российской Федерации по высшему образованию

КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. А.Н.ТУПОЛЕВА

Кафедра прикладной математики и информатики

А.Ю.АЛЕКСАНДРОВ, А.Н.КОЗИН, В.М.ТРЕГУБОВ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
" ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ АССЕМБЛЕРА"
Часть 1

Учебное пособие

Казань 1996

Лабораторная работа N 1

СТРУКТУРА ПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕРА

1.Цель работы. Целью лабораторной работы является получение знаний об основных правилах записи и структуре программ на языке Ассемблера для ПЭВМ типа IBM PC, а также приобретение практических навыков по составлению простейших программ на языке Ассемблера, их выполнению на ЭВМ и отладке с использованием интерактивных отладчиков.

2. Основные теоретические сведения.

Программы на языке Ассемблера представляют собой последовательность операторов, описывающих выполняемые операции.

Операторы в свою очередь подразделяются на команды и директивы (или псевдооператоры, псевдокоманды).

Каждая команда порождает одну или несколько команд в машинных кодах. Директивы не порождают машинных команд, а лишь информируют транслятор с языка Ассемблера об определенных действиях.

Формат команды языка Ассемблера в общем случае содержит до 4-х полей:

[метка:] команда [поле операндов] [;комментарий]

Обязательным является только поле команды. По крайней мере один пробел между полями обязателен. Максимальная длина строки - 132 символа.

Метка используется для указания адреса команды и служит для присвоения символического имени первому байту области памяти, в котором будет записан машинный эквивалент команды. Метки могут содержать:

- буквы от A до Z, причем Ассемблер не делает разницы между строчными и прописными буквами;
- цифры от 0 до 9;
- спец.символы: '?', '.'(только первый символ), '@'(коммерческое эт), '$', '_'.

Первым символом в метке должна быть буква или спец.символ. Максимальная длина метки - 31 символ.

Поле команды содержит имя команды, например, MOV - команда пересылки, ADD - команда сложения и другие.

Операнды определяют:

- начальные значения данных;
- элементы, над которыми выполняется действие.

Например, команда

MOV ax, 4 заносит в регистр

AX значение 4.

В поле операнда, если оно присутствует, может быть один или два операнда. Между собой операнды отделяются запятой. В командах с двумя операндами первый представляет собой приемник, а второй - источник. Источник никогда не изменяется, приемник изменяется почти всегда.

Комментарий начинается с символа ';' в любом месте программы. Все символы справа от символа ';' до конца строки воспринимаются Ассемблером как комментарий.

Директивы могут иметь до 4-х полей:

[метка] директива [операнд] [;комментарий]

Наиболее употребительные директивы можно разделить на две группы: директивы данных и директивы управления листингом. Директивы данных в свою очередь делятся на пять групп:

- директивы определения идентификаторов;
- директивы определения данных;
- директивы внешних ссылок;
- директивы определения сегмента/процедуры;

- директивы управления трансляцией.

Существует две директивы определения идентификаторов: EQU и '='. EQU присваивает имя выражению постоянно, а '=' - временно и его можно переопределить.

Существуют следующие директивы определения данных:

- DB - определить байт;
- DD - определить двойное слово;
- DW - определить слово;
- DT - определить 10 байт.

Обычно DB и DW резервируют память под переменные, а DD и DT - под адреса. В любом случае можно указать начальное значение или знак '?' только для резервирования памяти. Операция DUP позволяет повторять одно и тоже значение несколько раз. Например, директива

PEREM DB 4 DUP(?)

резервирует четыре байта в памяти под переменную PEREM.

Если перечислять значения переменной через запятую, то тем самым определяется таблица данных. Например, директива

TABL DW 5,7,9

выделяет область памяти с именем TABL длиной три слова и заносит в первое слово значение 5, во второе значение 7, в третье слово - значение 9.

Директивы определения сегмента делят исходную программу на сегменты. В программе на языке Ассемблера возможно 4 вида сегментов:

- сегмент данных;
- сегмент стека;
- сегмент команд;
- дополнительный сегмент.

По крайней мере один сегмент - сегмент команд - должен присутствовать в каждой программе.

Существует две директивы определения сегмента: SEGMENT и ENDS, а также директива ASSUME, которая устанавливает для Ассемблера соответствие между конкретными сегментами и сегментными регистрами. Например, директива

DATASG SEGMENT PARA 'DATA' определяет начало

сегмента с именем DATASG, а директива

DATASG ENDS конец этого

сегмента. Директива

ASSUME CS:CODESG,DS:DATASG

указывает Ассемблеру, что сегмент CODESG будет адресоваться с помощью сегментного регистра CS, а сегмент DATASG - с помощью сегментного регистра DS.

Директивы PROC и ENDP определяют процедуру. Процедура может иметь атрибут дистанции NEAR или FAR. Процедура с атрибутом NEAR может быть вызвана только из того сегмента команд, в котором она определена, а процедура с атрибутом FAR может быть вызвана и из другого сегмента команд. Например директива

FUN PROC NEAR

определяет начало процедуры FUN с атрибутом дистанции NEAR, а директива

FUN ENDP

конец этой процедуры. Программа на Ассемблере имеет атрибут FAR.

Директивы внешних ссылок PUBLIC и EXTRN делают возможным использование переменных и процедур, определенных в разных файлах.

Директива управления трансляцией END отмечает конец исходного модуля, поэтому она должна присутствовать в каждом модуле.

Директивы управления листингом PAGE и TITLE могут быть использованы для определения формата вывода распечаток программ и выдачи заголовков.

Таким образом, типовая структура программы на языке Ассемблера для ПЭВМ типа IBM PC выглядит следующим образом:

TITLE заголовок программы

```
                PAGE 60,132 ;---------------------------------------------------- ; Определение сегмента
стека
STACKSG  SEGMENT PARA STACK 'STACK'
                DB          64 DUP(?) ; область стека, не менее 32 слов
STACKSG  ENDS ;---------------------------------
--------------------
; Определение сегмента данных DATASG   SEGMENT PARA 'DATA'
; здесь поместить, если необходимо, директивы определения ; данных
                ...
DATASG ENDS ;-----------------------------------
-----------------
; Определение сегмента команд (основная программа)
CODESG   SEGMENT PARA 'CODE'
ASSUME   CS:CODESG, DS:DATASG, SS:STACKSG
ENTRY        PROC FAR
; Инициализировать программу
                PUSH  DS                    ; сохранить в стеке адрес возврата
                SUB   AX,AX                 ; обнулить регистр AX
                PUSH  AX                    ; занести в стек нулевое
                                            ; смещение для адреса возврата
; Инициализировать адрес сегмента данных
                MOV   AX,DATASG ; занести адрес сегмента
                MOV   DS,AX                 ; данных в регистр DS
; Программа
                ...
                RET                         ; вернутся в DOS
ENTRY        ENDP
CODESG   ENDS
              END   ENTRY
```

В данном фрагменте показана группа команд PUSH DS ... MOV DS,AX, которая является стандартной для программ на Ассемблере и оеспечивает возврат управления в DOS после выполнения программы. Команда RET обеспечивает выход из программы и передачу управления DOS.

Пример программы на языке Ассемблера приведен на рис. 1. В программе определены 3 сегмента:

STACKSG - сегмент стека. Оператор DB отводит под стек 64 байта (32 слова), не присваивая им начальных значений. Стек будет содержать адрес возврата в DOS.

DATASG - сегмент данных. В этом сегменте с помощью директивы определения данных DW определяется переменная PP, которой присваивается начальное значение - 0123h.

CODESG - сегмент команд. В этом сегменте содержатся выполняемые команды программы.

Директивы SEGMENT - ENDS определяют начало и конец сегмента.

Директива ASSUME назначает регистр CS для адресации сегмента команд, регистр DS - сегмента данных, регистр SS - сегмента стека.

Директива PROC определяет имя ENTRY как точку входа в основную программу из DOS (начало программы). Следующая группа команд PUSH DS ... MOV DS,AX является стандартной, команда RET обеспечивает выход из программы и передачу управления DOS. Директивы ENDP, ENDS, END заканчивают программу.

```
TITLE       FIRST
STACKSG  SEGMENT PARA STACK 'STACK'
                DB          64 DUP(?)
STACKSG  ENDS
DATASG      SEGMENT PARA 'DATA'
PP              DW    0123h
DATASG      ENDS
CODESG      SEGMENT PARA 'CODE'
```

```
ASSUME    CS:CODESG, DS:DATASG, SS:STACKSG
ENTRY     PROC  FAR
          PUSH    DS
          SUB     AX,AX
          PUSH    AX
          MOV     AX,DATASG
          MOV     DS,AX
          MOV     AX,PP     ; записать 0123 в AX
          ADD     AX,0025h  ; прибавить 25 к AX
          MOV     BX,AX     ; переслать AX в BX
          ADD     BX,AX     ; прибавить AX к BX
          MOV     CX,BX     ; переслать BX в CX
          SUB     CX,AX     ; вычесть AX из CX
          SUB     AX,AX     ; очистить AX
          RET
ENTRY     ENDP
CODESG    ENDS
          END   ENTRY
```

Рис.1. Пример программы на языке Ассемблера

3. Выполнение программ на Ассемблере

3.1. Построение исполнимого файла

Для создания программ на Ассемблере необходимо выполнить следующие действия:

1) С использованием какого-либо текстового редактора ввести текст программы в ЭВМ и запомнить его в файле с расширением .ASM (например, PROG.ASM).

2) Выполнить трансляцию программы. Для этого необходимо набрать команду

tasm prog,prog,prog /z

В результате будет создан объектный файл под именем имя_программы.OBJ, файл листинга имя_программы.LST, файл перекрестных ссылок имя_программы.CRF.

3) Построить исполняемый .EXE файл. Для этого вызвать компоновщик TLINK командой

tlink имя_программы.OBJ

В результате на диске будет построен исполняемый файл имя_программы.EXE и листинг распределения памяти имя_программы.MAP.

Просмотр листинга программы можно осуществить стандартными средствами персонального компьютера в файле имя_программы.LST.

Кроме листинга программы можно получить:

- листинг таблицы перекрестных ссылок;
- листинг распределения памяти.

В таблице перекрестных ссылок указывается номер строки, в которой определен каждый идентификатор, и номера тех строк, в которых есть ссылки на него.

Листинг распределения памяти содержит сводные сведения о сегментах программы и находится в файле имя_программы.MAP. Для каждого сегмента указывается адрес начала и конца, длина сегмента в байтах, его имя и категория.

3.2. Выполнение программы

Выполняемый .EXE файл можно вызвать как из DOS, так и выполнить под управлением отладчика TD (Turbo Debbuger).

Вызов программы из DOS осуществляется стандартным образом.

Последовательность действий при отладке программ под управлением интерактивного отладчика TD (Turbo Debbuger) приведена в приложении.

4. Задание на лабораторную работу

1. Изучить структуру программы на Ассемблере и основные требования к ней.

2. Разработать простую программу обмена данными между двумя одномерными таблицами, причем данные в таблице-"приемнике" должны размещаться в порядке, обратном таблице-"источнику". Для этого:

1) С использованием директивы DB определить в сегменте данных две таблицы с именами TABL1 и TABL2 по 4 байта каждая для таблицы-"источника" и таблицы-"приемника" соответственно. При этом таблица TABL1 должна иметь начальные значения.

2) Обнулить таблицу TABL2 командами:

```
MOV TABL2,0          ; обнулить первый байт
MOV TABL2+1,0 ; обнулить второй байт и т.д.
```

3) Скопировать таблицу TABL1 в TABL2 последовательностью команд:

```
MOV AL,TABL1          ; переслать в регистр AL первый
                      ; байт таблицы TABL1
MOV TABL2+3,AL   ; переслать в четвертый байт
                      ; таблицы TABL2 содержимое
                      ; регистра AL
```

и т.д.

3. С помощью текстового редактора ввести программу в ЭВМ и записать ее на диск. Оттранслировать программу и построить исполнимый файл. Распечатать листинг программы и изучить его структуру.

4. Выполнить программу под управлением отладчика TD. Провести пошаговую трассировку программы, распечатывая на каждом шаге содержимое сегмента данных. Изучить изменение указателя стека SP, командного указателя IP и сегмента данных от шага к шагу выполнения программы.

5. Завершить выполнение программы под управлением отладчика TD.

Лабораторная работа N2

КОМАНДЫ ПЕРЕСЫЛКИ И КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

1.Цель работы. Целью лабораторной работы является получение знаний о режимах адресации МП i8088, основных командах пересылки, командах перехода и организации циклов, а также приобретение практических навыков по применению команд пересылки и команд перехода при программировании на языке Ассемблера.

2. Основные теоретические сведения.

1. Режимы адресации микропроцессора i8088.

Микропроцессор i8088 (МП) может использовать различные методы (или режимы) для получения тех данных, которыми должна оперировать программа. Эти методы называются режимами адресации. Ассемблер определяет режим на основе формата операнда в исходной программе.

Можно выделить семь различных режимов адресации:

- регистровая;
- непосредственная;
- прямая;
- косвенная регистровая;
- по базе;
- прямая с индексированием;
- по базе с индексированием.

Простейшими режимами адресации являются первые два, поскольку в этом случае МП получает значение операнда из регистра или непосредственно из команды. При остальных пяти режимах МП должен вычислить адрес ячейки памяти, затем прочитать из нее значение операнда.

При регистровой адресации МП извлекает операнд из регистра или загружает его в регистр. Например,

```
MOV AX,CX ; пересылка содержимого регистра CX
          ; в регистр AX
```

Непосредственная адресация позволяет указать 8-ми или 16-битовое значение константы в качестве второго операнда команды. Например,

```
MOV CX,500 ; загрузить в регистр CX  значение 500
```

Для описания других режимов адресации введем понятие исполнительного адреса.

Исполнительным адресом называется смещение операнда относительно начала сегмента, в котором находится операнд.

При прямой адресации исполнительный адрес является составной частью команды. МП добавляет этот адрес к сдвинутому на 4 бита (т.е. умноженому на 16) содержимому регистра сегмента данных DS и получает 20-битовый физический адрес операнда. Обычно прямая адресация применяется, если операндом служит метка. Например,

```
MOV AX,TABLE ; загрузка в регистр AX содержимого
             ; ячейки памяти с меткой TABLE
```

При этом надо иметь в виду, что байты в памяти располагаются в обратной последовательности: старшие байты - по старшим адресам, младшие байты - по младшим адресам, т.е. если по адресу TABLE хранится значение AA, а по адресу TABLE+1 значение BB, то в результате приведенной операции регистр AX будет содержать значение BBAA.

При косвенной регистровой адресации исполнительный адрес операнда содержится в базовом регистре BX, регистре указателя базы BP или в индексном регистре SI или DI. Косвенные регистровые операнды необходимо заключать в квадратные скобки. Например,

```
MOV AX,[BX] ; загрузить в регистр AX содержимое ячейки,
            ; исполнительный адрес,  которой находится в
            ; регистре BX
```

Чтобы поместить смещение ячейки TABLE в регистр BX можно использовать операцию OFFSET. Например,

```
MOV BX,OFFSET TABLE ;  поместить в регистр BX
                    ; смещение  ячейки TABLE
```

Данный способ адресации эффективен в тех случаях, когда необходимо получать доступ к последовательности ячеек, начиная с данного адреса.

При адресации по базе исполнительный адрес вычисляется путем сложения содержимого регистра BX или BP и сдвига, измеряемого в байтах. Например, команды выполняют одно и тоже действие:

```
MOV AX,[BX]+4  ; загрузить  в  регистр AX содержимое
MOV AX,[BX+4]  ; по адресу, отстоящему на 4 байта
               ; от ячейки, исполнительный адрес
MOV AX,4[BX]   ; которой находится в регистре BX
```

При прямой адресации с индексированием исполнительный адрес вычисляется как сумма значений сдвига и индексного регистра SI или DI. Этот тип адресации удобен для доступа к элементам таблицы, когда сдвиг указывает на начало таблицы, а индексный регистр - на ее элемент. Например,

```
MOV DI,2              ; загрузить в AL третий элемент
MOV AL,TABLE[DI]      ; таблицы TABLE
```

При адресации по базе с индексированием исполнительный адрес вычисляется как сумма значений базового и индексного регистров и, возможно, сдвига. Этот метод удобен при адресации двухмерных массивов, когда базовый регистр содержит начальный адрес массива, а значение сдвига и индексного регистра определяют смещение по строке и столбцу. Например, допустимые форматы команды

```
MOV AX,[BX+2+DI]
MOV AX,[BX+2][DI]
MOV AX,[BX][DI+2]
```

При всех способах адресации (кроме тех, где используется регистр BP) исполнительный адрес вычисляется относительно регистра сегмента DS. Для регистра BP регистром сегмента служит регистр сегмента стека SS. Для изменения сегментных регистров, принятых по умолчанию, можно использовать операцию изменения префикса сегмента. Например, команда

```
MOV ES:[BX], DX
```

пересылает слово из регистра DX в ячейку памяти в сегменте,
адресуемом текущим содержимым сегментного регистра ES со смещением, находящемся в регистре BX.

2. Команды пересылки данных

Команды пересылки данных осуществляют обмен информацией между регистрами, ячейками памяти.

Команды пересылки данных делятся на 4 группы:

- команды общего назначения;
- команды ввода/вывода;
- команды пересылки адреса;
- команды пересылки флагов.

Рассмотрим наиболее распространенные из них.

Команда MOV.

Это основная команда общего назначения. Она позволяет переслать:

- байт или слово между регистрами или между регистром и ячейкой памяти;
- непосредственно адресуемое значение в регистр или ячейку памяти.

Формат команды:

MOV приемник,источник

Примеры:

```
MOV AX,TABLE  ;  переслать  из  памяти  в  регистр
MOV TABLE,AX  ;  переслать из регистра в память
MOV  CL,25          ;  переслать в регистр CL значение 25
```

Запрещается:

- непосредственная пересылка данных из одной ячейки памяти в другую. Для такой пересылки необходимо использовать промежуточный регистр общего назначения;
- загружать непосредственно адресуемый операнд в регистр сегмента. Такую пересылку необходимо делать через промежуточный регистр общего назначения;
- непосредственно пересылать значение одного регистра сегмента в другой;
- использовать регистр CS в качестве приемника.

Команда LEA.

Это команда загрузки исполнительного адреса. Она пересылает относительный адрес (смещение) ячейки памяти в 16-битовый регистр общего назначения, регистр указателя или индексный регистр. Формат команды:

LEA регистр,память

Примеры:

```
LEA BX,TABLE[DI] ; если регистр DI содержит 5, то в
                 ; регистре BX будет смещение ячейки
                 ; TABLE+5  в сегменте, адресуемом
                 ; текущим значением регистра DS
```

Команды PUSH и POP.

Команда PUSH помещает содержимое регистра или ячейки памяти размером в слово в вершину стека. Формат команды:

PUSH источник

Примеры:

```
PUSH SI
PUSH CS
PUSH TABLE[BX][DI]
```

Команда POP извлекает слово с вершины стека и помещает его в ячейку памяти или регистр. Формат команды:

POP приемник

Примеры:
POP AX

Под вершиной стека понимается ячейка памяти в сегменте стека, смещение которой содержится в указателе SP. SP всегда указывает на слово, помещенное в стек последним. Команда PUSH уменьшает значение SP на 2, а команда POP - увеличивает на 2.

Команды PUSHF и POPF.

Команда PUSHF помещает содержимое регистра флагов в вершину стека. Формат команды

PUSHF

Команда POPF извлекает слово с вершины стека и помещает его в регистр флагов. Формат команды:

POPF

3.Команды передачи управления

Процессор выполняет команды в том порядке, в каком они записаны в программе. Возможность изменения такого порядка обеспечивают команды передачи управления.

Команда безусловного перехода JMP передает управление в указанную точку того же или другого программного сегмента. Формат команды:

JMP метка

Если переход необходим только при выполнении некоторого условия и не осуществляется в противном случае, то такой переход называется условным. Условный переход обычно реализуется в два шага: сначала сравниваются некоторые величины, в результате чего формируются флаги процессора, а затем в зависимости от значений флагов происходит условный переход.

Рассмотрим сначала команду сравнения CMP. Команда имеет формат:

CMP op1,op2

Действие команды можно условно изобразить так:

op1 - op2 --> флаги процессора

Требования к операндам op1 и op2 аналогичны требованиям к операндам команды MOV.

Команды условного перехода имеют единый формат: команда метка

Команда осуществляет переход в указанную точку программы при выполнении условия, заданного мнемоникой команды. Если условие не выполняется, то переход не осуществляется а выполняется команда, следующая за командой условного перехода. Переход может осуществляться как вперед, так и назад в диапазоне +127 ... -128 байт. Ниже приводятся некоторые из команд условного перехода:

Команда	Перейти, если	Условие перехода
	Для любых чисел	
JE	op1=op2	ZF=1
JNE	op1<>op2	ZF=0

	Для чисел без знака	
JA	op1>op2	CF=0 и ZF=0
JAE	op1>=op2	CF=0
JB	op1<op2	CF=1
JBE	op1<=op2	CF=1 или ZF=1
	Для чисел со знаком	
JG	op1>op2	SF=OF и ZF=0
JGE	op1>=op2	SF=OF
JL	op1<op2	SF<>OF
JLE	op1<=op2	SF<>OF или ZF=1
	Команды, которые могут использоватся без предварительного сравнения	
JZ (JNZ)		ZF=1 (ZF=0)
JS (JNS)		SF=1 (SF=0)
JC (JNC)		CF=1 (CF=0)
JO (JNO)		OF=1 (OF=0)
JCXZ	содержимое регистра CX = 0	ZF=1 (ZF=0)

Примеры:

```
; 1   ...
        CMP  AL,BL
        JE   M1          ;переход на метку M1, если AL=BL
        ...
M1:   ...

; 2
        MOV CX,10
CIKL:
        ...              ; тело цикла, выполняемого 10 раз
        DEC CX           ; CX=CX-1
        JCXZ FIN  ; переход на метку FIN, если CX=0
        JMP CIKL ; иначе, переход на CIKL FIN: ...
```

Команды управления циклами

Для повторения некоторой группы команд (тела цикла) N раз (N>0) в процессоре i8088 предусмотрена команда LOOP. Формат команды: LOOP метка

Команда должна находиться в конце цикла. Ее действие можно условно описать так:

CX:=CX-1; if CX<>0 then goto метка

Пример:

```
        MOV CX,N ; CX=N (N>0) CIKL: ... ; --
        ...              ; |тело цикла, выполняемого 10 раз
        ...              ; --
        LOOP CIKL
```

Команда имеет ряд особенностей:

1) В качестве счетчика может использоваться только регистр CX, поэтому при организации вложенных циклов необходимо его сохранять и восстанавливать по ходу выполнения;

2) Поскольку LOOP ставится в конце цикла, тело выполнится хотя бы один раз (даже при CX=0);

3) Как и для команд условных переходов, расстояние от метки начала тела цикла до команды LOOP не должно превышать 127 байтов (30-40 команд).

Пример программирования вложенных циклов:

```
                MOV CX,N    ;
```

```
CIKL1:      PUSH CX     ;  _____
            ...         ;| Тело внешнего цикла
            MOV CX,M    ;| -------------------
CIKL2:      ...         ;| | Внутренний цикл |
            ...         ;| |                 |
            LOOP CIKL2  ;| -------------------
            ...         ;| Счетчик внешнего  цикла сохраняется в
            POP CX      ;| стеке, а перед LOOP восстанавливается
            LOOP CIKL1  ; _____
```

Для организации циклов с возможностью досрочного выхода из цикла можно использовать команды LOOPE (цикл по счетчику и пока равно) и LOOPNE (цикл по счетчику и пока не равно).

Действие команд можно представить как:

LOOPE: CX:=CX-1; if (CX<>0) and (ZF=1) then goto метка

LOOPNE: CX:=CX-1; if (CX<>0) and (ZF=0) then goto метка

В остальном команды аналогичны команде LOOP.

Команда LOOPE обычно используется для поиска первого элемента последовательности, отличного от заданного. LOOPNE- для поиска первого элемента, имеющего заданную величину.

3. Пример программы

Дана строка из четырех символов. Необходимо осуществить круговую перестановку символов строки.

```
STACKSG SEGMENT PARA STACK
        DB 64 DUP(?)
STACKSG  ENDS
DATASG   SEGMENT PARA 'DATA'
STR1     DB '1234'
STR2     DB 4 DUP(' ')
DATASG   ENDS
CODESG   SEGMENT PARA 'CODE'
ASSUME   CS:CODESG,DS:DATASG,SS:STACKSG
ENTRY    PROC FAR
; Стандартная часть
        PUSH  DS
        SUB   AX,AX
        PUSH  AX
        MOV   AX,DATASG
        MOV   DS,AX
;
        MOV  DX,4           ; общее количество перестановок
; Переслать первый символ из STR1 в STR2
M1:
        LEA   DI,STR1   ; загрузить в DI смещение первого байта из STR1
        LEA   SI,STR2   ; загрузить в SI смещение первого байта из STR3
        MOV   CX,3
        MOV   AL,[DI]   ; переслать в AL первый байт из STR1
        MOV   [SI]+3,AL ; переслать в последний байт из STR2 содержимое AL
        INC  DI                 ; DI=DI+1 - следующий символ из STR1
;
; переслать остаток строки STR1 в STR2
M2:
        MOV   AL,[DI]       ;   в AL следующий символ из STR1
```

```
            MOV   [SI],AL          ;  переслать AL в очередной (с первого) байт в STR2
            INC   DI               ;  DI=DI+1 - следующий символ из STR1
            INC   SI               ;  SI=SI+1 - следующий символ из STR2
            LOOP  M2               ;  идти на M2
;
; переслать STR2 в STR1
            LEA  DI,STR1
            LEA  SI,STR2
            MOV CX,4 M3:
            MOV   AL,[SI]
            MOV   [DI],AL
            INC   DI
            INC   SI
            LOOP  M3
;
            DEC  DX
            CMP  DX,00             ; Все перестановки сделаны?
            JNE  M1                ; Нет - идти на M1
            RET
ENTRY   ENDP
CODESG ENDS
            END   ENTRY
```

4. Задание на лабораторную работу

1. Дана строка из 15 символов. Разработать программу, осуществляющую круговую перестановку части букв исходной строки.

Индивидуальные задания.

1) с первого символа по восьмой;
2) с третьего символа по десятый;
3) с первого символа по пятнадцатый через один;
4) со второго символа по десятый через два;
5) с первого по двенадцатый по два символа;
6) с третьего символа по четырнадцатый по три символа;
7) с первого по пятнадцатый по пять символов.

2. Провести пошаговую трассировку программы под управлением отладчика TD.

Лабораторная работа N3

ПОБИТОВАЯ ОБРАБОТКА НА ЯЗЫКЕ АССЕМБЛЕРА

1.Цель работы. Целью лабораторной работы является получение знаний о логических командах и командах сдвига, а также приобретение практических навыков по применению команд побитовой обработки при программировании на языке Ассемблера.

2. Основные теоретические сведения.

Команды побитовой обработки данных манипулируют группами битов в регистрах или ячейках памяти.

Существует 3 группы команд побитовой обработки:

- логические команды;
- команды сдвига;
- команды циклического сдвига.

2.1. Логические команды

К логическим командам относятся команды AND, OR, XOR, TEST, NOT. Формат первых четырех команд:

команда приемник,источник

Формат команды NOT:

NOT приемник

Логические команды обрабатывают один байт или слово в регистре или в памяти и устанавливают флаги CF, OF, PF, SF, ZF (флаг AF не определен). При этом обработка байта (слова) осуществляется непосредственно бит за битом.

Логические операции часто используются для установки в двоичные разряды необходимых значений. Существуют следующие способы получения подобных результатов:

- для установки в данный разряд 0 необходимо логически умножить разряд на 0 (команда AND);
- для установки в данный разряд 1 необходимо логически сложить разряд с 1 (команда OR).

Например, предположим, что требуется, не меняя остальные 7 битов байта, присвоить пятому байту значение 1. Для этого достаточно логически сложить исходный байт с байтом, содержащим 00100000.

Рассмотрим действия логических команд:

AND: Если оба из сравниваемых битов приемника и источника равны единице, то результирующий бит равен единице; во всех остальных случаях результат равен нулю.

OR: Если хотя бы один из двух сравниваемых битов приемника и источника равен единице, то результирующий бит приемника равен единице; если сравниваемые биты равны нулю, то результат равен нулю.

XOR: Если один из сравниваемых битов равен нулю, а другой равен единице, то результирующий бит равен единице; если оба бита одинаковы, то результат равен нулю.

NOT: Устанавливает обратное значение битов в байте или слове, регистре или ячейке памяти.

TEST: Действует как AND, устанавливает флаги, но не меняет биты.

Примеры,

```
MOV  AL,11000101B
MOV  BH,01011100B
AND  AL,BH              ; устанавливает в AL 010001000
OR   CL,CL              ; устанавливает флаги CF и ZF
```

2.2. Команды сдвига

Команды сдвига осуществляют сдвиг 8- или 16-битового содержимого регистров на одну или несколько позиций влево или вправо. Формат:

команда приемник,счетчик

В этих командах содержимое приемника сдвигается на величину, задаваемую счетчиком. При этом счетчик может быть цифрой 1 или значением без знака в регистре CL.

Особенностью команд сдвига является то, что они помещают во флаг переноса CF значение бита, сдвинутого за один из концов приемника.

Команды сдвига делятся на команды логического, арифметического и циклического сдвига.

2.2.1. Команды арифметического сдвига

SAL / SAR - сдвинуть арифметически влево / вправо.

Команда SAL не сохраняет знак операнда, но заносит 1 во флаг переполнения CF и 0 в освобождающийся бит. Команда SAR сохраняет знак операнда, занося его при каждом сдвиге в старший бит операнда. Например,

```
MOV  BL,10110100B
MOV  AL,BL       ; пусть CF=1
SAL  AL,1        ; AL=01101000, CF=1
MOV  AL,BL
SAR  AL,1        ; AL=11011010, CF=0
```

2.2.2. Команды логического сдвига:

SHL / SHR - сдвинуть логически влево / вправо.

Команда SHL идентична команде SAL. Команда SHR аналогична SAR, но сдвигает операнд вправо, занося 0 в старший бит операнда. Примеры,

```
MOV  BL,10110100B
MOV  AL,BL       ; пусть CF=1
SHL  AL,1        ; AL=01101000, CF=1
MOV  AL,BL
SHR  AL,1        ; AL=01011010, CF=0
```

Поскольку сдвиг операнда на 1 бит удваивает значение операнда, а сдвиг на 1 бит вправо уменьшает его вдвое, то команды сдвига можно использовать в качестве команд быстрого умножения и деления.

Примеры,

```
MOV  CL,2
SHL  AX,CL                ; умножить число без знака на 4
```

2.3. Команды циклического сдвига:

ROL / ROR - сдвинуть циклически влево / вправо.

При выполнении этих команд вышедший за пределы операнда бит входит в него с другого конца и помещается во флаг переноса. Примеры,

```
MOV  BL,10110100B  ; пусть CF=1
MOV  AL,BL
ROL  AL,1                 ; AL=01101001, CF=1
```

RCL / RCR - сдвинуть циклически вместе с флагом переноса влево / вправо.

При выполнении этих команд осуществляется сдвиг, в противоположный конец операнда помещается значение флага CF и затем в CF помещается вышедший за пределы операнда бит.

Примеры,

```
MOV  BL,10110100B  ; пусть CF=1
MOV  AL,BL
RCR  AL,1                 ; AL=11011010, CF=0
```

2.4. Простейшие способы ввода символов с клавиатуры и вывода символов на экран.

Для вывода символов на экран в текущей позиции курсора, необходимо:

- определить область данных для вывода при этом последним символом области вывода должен быть знак $;

- занести в область данных для вывода необходимые данные;
- установить в регистре AH значение 09;
- занести в регистр DX адрес области данных для вывода;
- указать команду прерывания INT 21H.

Пример.

Для вывода на экран текста 'Введите данные' необходимы следующие команды:

```
STR   DB 'Введите данные','$'
  ...
        MOV  AH,09
        LEA  DX,STR
        INT  21H
```

Для ввода последовательности символов с клавиатуры необходимо:

- определить область памяти в сегменте данных для вводимых символов, причем длина этой области должна быть на 2 байта больше максимальной длины строки с учетом того, что последний введенный символ всегда будет 'RETURN' (код 0DH). Первый байт содержит максимальное число символов, а второй байт будет содержать число реально введенных символов до нажатия клавиши 'RETURN';
- установить в регистре AH значение 10;
- поместить в регистр DX адрес области памяти;
- указать команду прерывания INT 21H.

Пример.

Для ввода с клавиатуры строки максимальной длиной в 50 символов необходимы следующие команды:

```
 STR DB 50,52 DUP(?)
  ...
       MOV  AH,10
       LEA  DX,STR
       INT  21H
```

3. Пример программы

Следующая программа вводит строку символов латинского алфавита и заменяет все строчные буквы на прописные, а прописные - на строчные. Результат выводится на экран. Например, строка TablE будет заменена на строку tABLe и выведена на экран.

```
STACKSG   SEGMENT PARA STACK 'stack'
          DB 64 DUP(?)
STACKSG   ENDS
DATASG    SEGMENT PARA 'DATA'
STR       DB 10,12 DUP(' ')
DATASG    ENDS
CODESG SEGMENT PARA 'CODE'
ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
ENTRY   PROC FAR

        PUSH  DS
        SUB   AX,AX
        PUSH  AX
        MOV   AX,DATASG
        MOV   DS,AX

; Ввод исходной строки
        MOV  AH,10
        LEA  DX,STR
        INT  21H
```

```
; Блок замены символа 'A' на 'a'
            MOV  SI,2
            SUB  CX,CX                ; CX=CX-CX
            MOV  CL,STR+1             ; длина строки
M1:          MOV  AL,STR[SI]   ; занести в AL первый символ
            CMP   AL,'A'      ;  < 'A' ?
            JB       NEXT1
            CMP   AL,'z'      ;  > 'z' ?
            JA       NEXT1
            CMP   AL,'a'      ;  < 'a' ?
            JB       MARK1
;
            XOR  AL,00100000B             ; заменить
            MOV  STR[SI],AL               ; занести новый символ в строку
            JMP  SHORT NEXT1
MARK1:  OR   AL,00100000B   ; заменить
            MOV  STR[SI],AL               ; занести новый символ в строку
NEXT1:  INC  SI                            ; перейти к следующему символу (SI=SI+1)
            LOOP M1
; Печать результирующей строки
PRINT1:
            MOV   Ah,09h
            LEA    DX,STR
            SUB    BX,BX
            MOV   BL,STR+1
            MOV   STR[BX+2],'$'
            MOV   STR,0Ah
            MOV   STR+1,0Dh
            INT     21H

             RET
ENTRY     ENDP
CODESG  ENDS
            END   ENTRY
```

4. Варианты заданий на лабораторную работу

1. Ввести с терминала строку символов. Вывести на экран ее двоичное представление.
2. Ввести с терминала строку символов. Вывести на экран количество единичных битов третьего символа строки.
3. Дано двоичное число. Вывести на экран его шестнадцатеричное представление.
4. Дано двоичное число. Вывести на экран его восьмеричное представление.
5. Ввести с терминала строку символов. Вывести на экран количество нулевых битов второго символа строки.
6. Разработать программу быстрого умножения числа длиной двойное слово.
7. Разработать программу быстрого деления числа без знака длиной двойное слово.

ПРИЛОЖЕНИЕ

РАБОТА С ОТЛАДЧИКОМ Turbo Debugger (TD)

В процессе отладки часто бывает так, что программа, успешно пройдя этап трансляции и компоновки, либо работает не так, как ожидалось, либо вообще не работает. Это означает, что с точки зрения правил языка программирования, программа написана правильно (в ней нет синтаксических ошибок), но алгоритм ее в чем-то неверен. Для отладки такой программы следует воспользоваться программой-отладчиком.

Последовательность действий при работе с отладчиком TD

1. Загрузить программу под управлением отладчика, введя с клавиатуры в командной строке (TD prog.exe). На экране появится информационный кадр отладчика, показанный на рис.1.

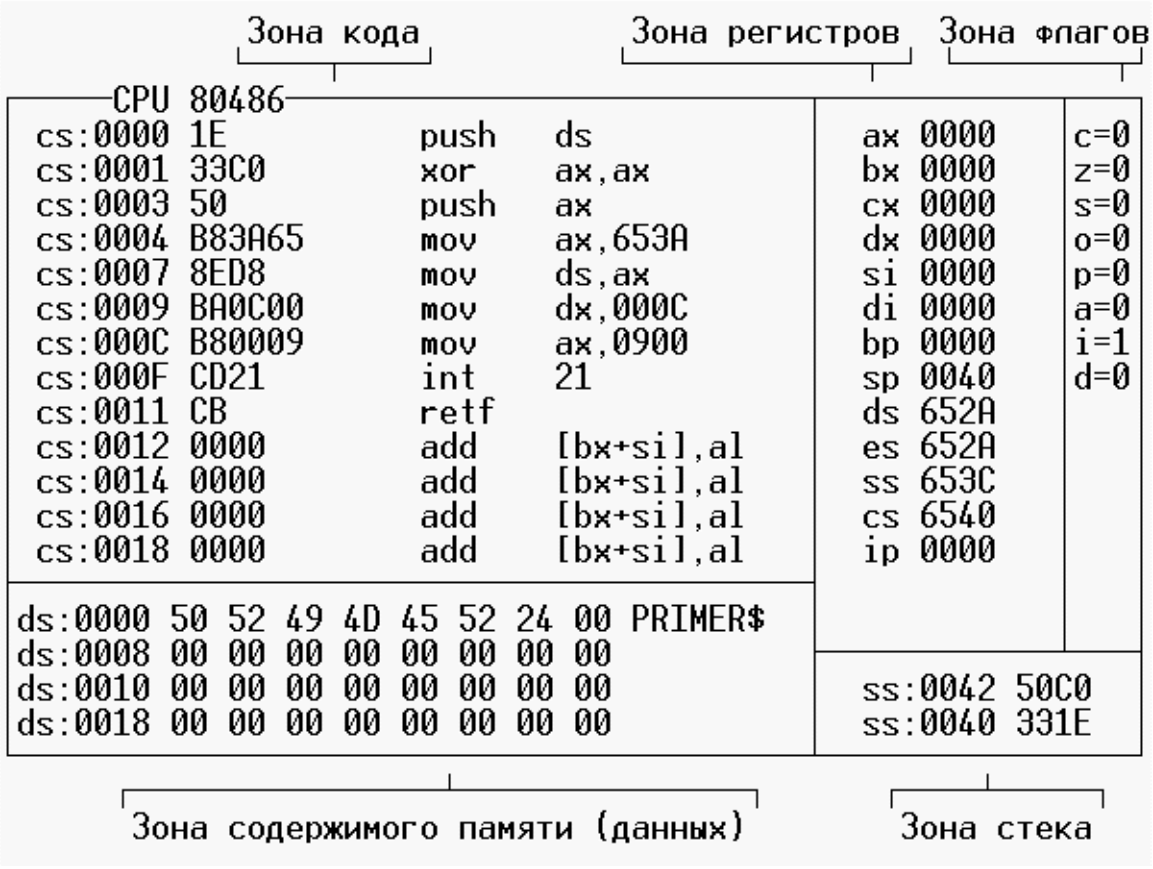

Рис.1. Информационный кадр отладчика TD

Действия при работе с отладчиком:

а) В ответ на сообщение "Program has no symbol table" следует нажать клавишу <Enter>;

б) Увеличить размер кадра до размеров экрана, нажав на <F5>;
в) В режиме пошаговой отладки выполнить инициализацию программы и сегмента данных (<F7> или <F8> - выполнение одной команды);
г) Используя <Tab> или <Shift>-<Tab>, перейти в зону данных;
д) Нажать <Ctrl>-<G> и ввести в поле ввода начальный адрес интересуемой области памяти (сегмента данных) DS:0 . В зоне появится содержимое сегмента данных;
е) Используя <Tab> или <Shift>-<Tab> , перейти в зону кода;
ж) Выполнить программу в пошаговом режиме;
з) Для выхода из отладчика в DOS, нажать <Alt>-<X>.

Если в процессе работы программа осуществляет вывод на экран, то просмотреть его можно, нажав на <Alt>-<F5>.

ЛИТЕРАТУРА

Основная литература

1. Скэнлон Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера: Пер. с англ.- М.:Радио и связь, 1991.-336 с.
2. Абель П. Язык Ассемблера для IBM PC и программирования:
Пер. с англ.- М.:Высшая школа, 1992.-477 с.

Дополнительная литература

1. Рудаков П.И., Финогенов К.Г. Программируем на языке ассемблера
IBM PC: Части 1 и 2.-М.:"Энтроп", 1995.-164, 162 с.
2. Финогенов К.Г. Самоучитель по системным функциям MS-DOS.-М.:
"МП Малип", 1993.-264 с.
3. Пильщиков В.Н. Программирование на языке ассемблера IBM PC.-
М.:"ДИАЛОГ МИФИ", 1994.-288 с.
4. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT, AT: Пер. с англ.- М.:Финансы и статистика, 1992.-544 с.