Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«Севастопольский государственный университет»

Институт информационных технологий и управления в технических системах

М Е Т О Д И Ч Е С К И Е У К А З А Н И Я

«Программирование на языке Ассемблера IBM PC»

по дисциплине

« Системное программирование»

для студентов очной и заочной форм обучения
направления 09.03.01–«Информатика и вычислительная техника»

Севастополь

2016

УДК 681.3.06

Методические указания «Программирование на языке Ассемблера IBM PC» по дисциплине «Системное программирование» для студентов очной и заочной форм обучения направления 09.03.01– «Информатика и вычислительная техника» / Сост. Тертычный А.И., Шалимова Е.М. – Севастополь: Издательство СевГУ, 2016. – 40 с.

Целью методических указаний является оказание помощи студентам в изучении дисциплины. Методические указания содержат краткое изложение основных теоретических положений: форматов записи команд и директив, примеры решения задач, а также задачи для самостоятельного решения и список рекомендованной литературы.

Методические указания рассмотрены и утверждены на заседании кафедры информационных технологий и компьютерных систем, протокол № 9 от 19.05. 2016 года.

Рецензент Карлусов В.Ю., к. т. н., доцент.

Ответственный за выпуск Брюховецкий А.А., зав. кафедрой ИТиКС, к. т. н., доцент.

СОДЕРЖАНИЕ

1. ПРЕДСТАВЛЕНИЕ ДАННЫХ В КОМПЬЮТЕРЕ IBM PC. ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ

Цель занятия: изучить форматы представления данных в компьютере IBM PC и директивы определения данных в языке Ассемблер.

1.1. Основные теоретические сведения

В языке Ассемблера IBM PC определены следующие типы данных: целое число, символ, строка. В общем случае под целое число можно отвести любое число байтов, однако система команд ПК поддерживает только числа размером в байт, слово (два байта) и частично размером в двойное слово.

1.1.1. Знаковые и беззнаковые данные

В ПК делается различие между целыми числами без знака (неотрицательными) и со знаком. Это объясняется тем, что в ячейках одного и того же размера можно представить больший диапазон беззнаковых чисел, чем неотрицательных знаковых чисел, и если известно заранее, что некоторая числовая величина является неотрицательной, то выгоднее рассматривать ее как беззнаковую.

Для изображения положительных и отрицательных чисел в процессорах i8086/88 применяется дополнительный код. Самый левый бит числа используется для представления знака: 0 – число положительное, 1 – отрицательное.

Таким образом, байт в беззнаковом пространстве может содержать числа в диапазоне от 0 до 255, а знаковый байт – в диапазоне от -128 до +127. Двухбайтное беззнаковое число может содержать значения от 0 до 65536, а знаковое – от -32768 до +32767.

Каждое хранимое в памяти целое число может трактоваться программистом как знаковое или беззнаковое, примеры различной трактовки приведены в таблице 1.1.

Таблица 1.1 – Примеры интерпретации знаковых и беззнаковых данных

Содержимое байта	Беззнаковая интерпретация	Знаковая интерпретация
11111100	252	–4
00000101	5	5
11110110	246	–10
10001001	137	–119
01111001	121	121

1.1.2. Представление данных в оперативной памяти

Программы, выполняемые процессором, и данные, с которыми работают программы, находятся в оперативной памяти компьютера. Все байты памяти пронумерованы. Номер последнего байта определяется объемом оперативной памяти. Номер байта называется ***адресом***.

Числа размером в слово или двойное слово хранятся в памяти в "перевернутом" виде: младший байт имеет меньший адрес. Например, число 130 (0082h) в виде слова хранится в памяти как **| 82 | 00 |**

1.1.3. Директивы определения данных

В языке Ассемблера имеются предложения для задания значений переменных и резервирования памяти. Они называются директивами определения данных (таблица 1.2). Отметим, что директивы DQ и DT используются редко.

Таблица 1.2 – Директивы определения данных

Директива	Размер области памяти
DB	байт
DW	слово (два байта)
DD	двойное слово (четыре байта)
DQ	восемь байтов
DT	десять байтов

В простейшем случае в директивах DB, DW или DD определяется одна константа, которой дается имя для последующих ссылок на нее. По этой директиве ассемблер формирует машинное представление константы и записывает его в соответствующую область памяти. Адрес этой области становится значением имени: все вхождения имени в программу ассемблер будет заменять на этот адрес.

Для задания значений переменных используются числа в системах счисления с основаниями 10, 16, 8 или 2. Десятичные числа записываются в обычном виде; шестнадцатиричное число оканчивается буквой h (если число начинается с "цифры" A, B, ..., F, то вначале обязателен незначащий 0), восьмиричное число − буквами q или o, двоичное число − буквой b.

Константы-символы определяются директивой DB, в которой указывается либо код символа (целое от 0 до 255), либо сам символ в кавычках (одинарных или двойных).

Примеры директив представлены в таблице 1.3.

Таблица 1.3 – Примеры директив определения данных

Директива	Назначение
`A DB 162`	определить байт со значением 162 и именем A
`B DB 0A2h`	определить байт с таким же значением, но с именем B
`C W -1`	определить слово со значением -1 и именем C
`D DW 0FFFFh`	определить слово с таким же значением, но с именем D
`C1 DB '*'`	определить символ со значением 2Ah (код *) и именем C1
`C1 DB 02Ah`	директива, эквивалентная предыдущей
`X DD -1`	определить двойное слово со значением -1 и именем X

С помощью любой из директив **DB, DW и DD** можно определить переменную, не задавая начального значения. В этом случае в правой части директивы указывается вопросительный знак:

```
F DW ?
```

В одной директиве можно определить сразу несколько констант и/или переменных одного и того же размера, для чего их надо перечислить через запятую. Они размещаются в соседних ячейках памяти:

```
G DB 200, -5, 10h, ?, 'F'
```

По этой директиве резервируются 5 смежных байтов памяти. Первые три байта и пятый байт инициализируются указанными в директиве константами, в четвертый байт начальное значение не заносится. Имя, указанное в директиве, считается именующим первую из констант. Для ссылок на остальные используются выражения вида <***имя***> + <***целое***>. Таким образом, для доступа к байту с числом -5 надо указать выражение **G+1**, для доступа к байту с **10h** – выражение **G+2** и т.д. Д

Если в директиве **DB** перечислены только символы, например:

```
S DB 'a','+','b'
```

тогда эту директиву можно записать короче, заключив все эти символы в одни кавычки:

```
S DB 'a+b'
```

Если в директиве используется несколько одинаковых констант (переменных), то можно воспользоваться конструкцией повторения **k DUP(a,b,...,c)** , которая эквивалентна повторенной k раз последовательности **a,b,...,c**.

Например, директивы

```
V1 DB 0,0,0,0,0
V2 DW ?,?,?,?,?,?,?,?,?,'a',1,2,1,2,1,2,1,2
```

с помощью конструкции повторения можно представить в виде:

```
V1 DB  5 DUP(0)
V2 DW  9 DUP(?), 'a', 4 DUP(1,2)
```

1.2. Примеры решения задач

Пример1.

Определить размер памяти и ее содержимое для каждой из директив:

```
A DB 3
B DW 1
C DW 1, 3000H
D DD 12345678H
```

Решение приведено в таблице 1.4.

Таблица 1.4 – Результат действия директив

Директива	Размер памяти	Содержимое
`A DB 3`	1 байт	`03`
`B DW 1`	1 слово (2 байта)	`01 00`
`C DW 1, 3000H`	2 слова (4 байта)	`01 00 00 30`
`D DD 12345678H`	4 байта (двойное слово)	`78 56 34 12`

Пример2.

Выписать директивы, определяющие переменные с возможными интервалами значений A[10–230], M[-24–215].

Решение.

Возможное значение переменной A находится в диапазоне [0–255], что соответствует представлению беззнакового числа размером в 1 байт, для объявления которого используется директива **DB.** Возможное значение переменной B соответствует интервалу знаковых чисел размером в слово [-32768 – 32767], для объявления которого используется директива **DW.**

Таким образом,

```
A   DB    ?
B   DW    ?
```

1.3. Задачи для самостоятельного решения

1. Для каждой из указанных директив выписать эквивалентную ей директиву, где начальное значение переменной записано в 16-ричном виде.

а) A DB 17 б) B DB 15 в) V DB 255
г) G DB 150 д) D DB -17 е) E DB -1

2. Для каждой из указанных директив выписать две эквивалентные ей директивы, в первой из которых начальное значение переменной записано в виде десятичного числа со знаком, а во второй - без знака.

а) A DB 0Ah б) B DB 0A5h в) V DW 7Fh
г) G DB 80h д) D DB 101b е)E DW 0FFFEh
з) Z DW 80h

3. A DW 1020h

B DD 10203040h

Указать в 16-ричном виде значения байтов с адресами: A и A+1; B, B+1, B+2 и B+3.

4. Описать переменную D размером в двойное слово с начальным значением 2^{16}.

5. Записать более простым способом директиву

C DB '5'+1

6. Описать переменную-слово X, начальным значением которой является:
а) адрес этой же переменной;
б) адрес следующего слова памяти;
в) адрес предыдущего слова памяти.

7. A DB 0,1,2

B DB 3,4,5,6

Указать значения байтов с адресами: A+1, B+2, A+4 и B-1.

8. Выписать директивы определения переменных:
а) R[100:2330], Y[-24:121], масив 8 чисел в интервале [-3400:215].
б) L[18:63000], Q[-2540:2015], масив 5 чисел в интервале [-128:115].
в) T[5:252], Z[-4134:-5215], масив 8 чисел в интервале [24:256].

9. Выписать все возможные варианты (кроме тех, где указываются коды букв) описания символьного массива S, начальными значениями элементов которого являются первые три большие буквы русского алфавита.

10. Описать массив X из 85 элементов-слов со следующими начальными значениями:
а) первые 40 элементов имеют значение 10, следующие 20 элементов - значение '*', остальные - без начального значения;
б) средний элемент имеет значение 1, а все остальные - значение 0.

2. ПРОГРАММИРОВАНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

Цель занятия: изучить команды языка Ассемблер IBM PC для реализации арифметических операций, приобрести навыки программирования арифметических выражений.

2.1. Основные теоретические сведения

2.1.1. Регистр флагов

Флаг – это бит, принимающий значение 1 ("флаг установлен"), если выполнено некоторое условие, и значение 0 ("флаг сброшен") в противном случае. В процессоре i8086/88 используется 9 флагов, каждому из них присвоено определенное имя Все флаги собраны в регистре флагов, каждый флаг – это один из разрядов регистра, часть разрядов не используется, рисунок 2.1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	**x**	**x**	**x**	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**x**	**AF**	**x**	**PF**	**x**	**CF**

Рисунок 2.1−Регистр флагов

Некоторые флаги принято называть флагами условий; они автоматически меняются при выполнении команд и фиксируют те или иные свойства их результата (например, равен ли он нулю). Другие флаги называются флагами состояний; они меняются из программы и оказывают влияние на дальнейшее поведение процессора (например, блокируют прерывания).

Флаги условий:

CF (carry flag) - флаг переноса. Принимает значение 1, если при сложении целых беззнаковых чисел появилась единица переноса из самого левого бита, или если при вычитании чисел без знака первое из них было меньше второго. В командах сдвига в CF заносится бит, вышедший за разрядную сетку. CF фиксирует также особенности команды умножения;

OF (overflow flag) - флаг переполнения. Устанавливается в 1, если при сложении или вычитании целых чисел со знаком получился результат, по модулю превосходящий допустимую величину ;

ZF (zero flag) - флаг нуля. Устанавливается в 1, если результат команды оказался равным 0;

SF (sign flag) - флаг знака. Устанавливается в 1, если в операции над знаковыми числами получился отрицательный результат;

PF (parity flag) - флаг четности. Равен 1, если младший байт результата содержит четное количество двоичных единиц;

AF (auxiliary carry flag) - флаг дополнительного переноса. Фиксирует особенности выполнения операций над двоично-десятичными числами.

Флаги состояний:

DF (direction flag) - флаг направления. Устанавливает направление просмотра строк в строковых командах, при DF=0 строки просматриваются
"вперед" (от начала к концу), при DF=1 - в обратном направлении;

IF (interrupt flag) - флаг прерываний. При IF=0 процессор перестает реагировать на поступающие к нему прерывания, при IF=1 блокировка прерываний снимается;

TF (trap flag) - флаг трассировки. При TF=1 после выполнения каждой команды процессор делает прерывание (с номером 1), чем можно воспользоваться при отладке программы.

2.1.2. Арифметические команды

Процессор i8086/88 реализует четыре базовые арифметические операции над числами в двоичном и двоично-десятичном форматах.

Двоичное представление удобно в том случае, если данные определены в самой программе. Арифметические операции для двоичных 8– и 16–битовых знаковых и беззнаковых операндов приведены в таблице 2.1.

Команды сложения и вычитания для знаковых и беззнаковых данных выполняются одними и теми же командами по одним и тем же алгоритмам. Команды сложения и вычитания возможны для целых чисел размером в слово и байт. Специальных команд для сложения и вычитания двойных слов нет, эти операции реализуются через команды сложения и вычитания слов.

Таблица 2.1 – Арифметические команды

Мнемоника команды	Описание команды
ADD (сложить) **ADC** (сложить с переносом) **INC (**инкремент)	приемник+источник → приемник приемник+ cf+источник→ приемник приемник+1→ приемник
SUB (вычесть) **SBB** (вычесть с переносом**)** **DEC (**декремент)	приемник–источник → приемник приемник–cf–источник→приемник приемник–1→ приемник
MUL (умножить без знака) **IMUL** (умножить со знаком)	AL*источник(8) → AX AX*источник(16)→ DX,AX Команда MUL, но операнды знаковые
DIV (разделить без знака) **IDIV**(разделить со знаком)	AX/источник(8) → AL DX,AX/источник(16) → AX Команда DIV, но операнды знаковые
NEG (изменить знак)	–приемник → приемник

Сложение и вычитание беззнаковых чисел производится по модулю 2^8 для байтов и 2^{16} для слов. Это означает, что если в результате сложения появилась единица переноса, не вмещающаяся в разрядную сетку, то она отбрасывается. Рассмотрим сложение байтов 128 и 130 :

128		1	0	0	0	0	0	0	0
	+								
130		1	0	0	0	0	0	1	0
258	**1**	0	0	0	0	0	0	1	0

При сложении возникла единица переноса из старшего разряда, для которой нет места в разрядной сетке. Таким образом, в результирующем байте остается число 00000010, которое и объявляется результатом сложения. Ошибка здесь не фиксируется, но флаг переноса CF устанавливается в 1 (если переноса не было, то

CF=0). Обнаружить такое искажение суммы можно только последующим анализом флага CF.

Умножение и деление знаковых и беззнаковых чисел выполняются по разным алгоритмам, разными машинными командами. При умножении байтов(слов) первый сомножитель должен находиться в регистре AL (AX), результатом же умножения является слово (двойное слово), которое заносится в регистр AX (регистры DX и AX). Тем самым при умножении сохраняются все цифры произведения.

При делении байтов (слов) первый операнд (делимое) должен быть словом (двойным словом) и обязан находиться в регистре AX (регистрах DX и AX). Результатом деления являются две величины размером в байт (слово) - частное и остаток от деления; частное записывается в регистр AL (AX), а остаток - в регистр AH (DX).

Во многих случаях исходные данные вводятся программой с клавиатуры или из файла в виде символов, т.е. каждый байт содержит одну цифру. Примером неупакованного десятичного формата служит ASCII–код, в котором каждый символ представляется одним байтом. Представление цифр в ASCII – коде дано в таблице 2.2.

Таблица 2.2 – Представление цифр в ASCII – коде.

Цифра	**Код**	**Цифра**	**Код**
0	30H	5	35H
1	31H	6	36H
2	32H	7	37H
3	33H	8	38H
4	34H	9	39H

С помощью команд, приведенных в таблице 2.1, можно выполнять арифметические операции непосредственно над числами в ASCII – формате, но результат таких операций требует коррекции. Для коррекции результата используются следующие команды:

AAA – коррекция для сложения ASCII – кодов;

AAD – коррекция для деления ASCII – кодов;

AAM – коррекция для умножения ASCII – кодов;

AAS – коррекция для вычитания ASCII – кодов.

Эти команды записываются без операндов и выполняют коррекцию содержимого регистра AX.

Рассмотрим пример. Предположим, что регистры AX и BX содержат шестнадцатеричные значения 0038 (код символа 8) и 0034 (код символа 4) соответственно. Сложение и коррекция выполняются следующими командами:

```
ADD  AL,BL   ; сложение 34H и 38H
AAA      ; коррекция  для сложения ASCII–кодов
```

Команда AAA проверяет правую шестнадцатеричную цифру (4 бита) в регистре AL. Если эта цифра находится между цифрами A и F или флаг переноса AF равен 1, то к содержимому регистра AL прибавляется 6, а к содержимому регистра AH прибавляется 1. В нашем примере в регистре AX получаются следующие значения:

после команды ADD: 006C ;
после команды AAA: 0102 .

Для получения окончательного ASCII–представления необходимо поставить тройки на место левых шестнадцатеричных цифр. Это можно сделать, к примеру, следующим образом:

```
OR   AX,3030H   ;результат 3132
```

При программировании арифметических выражений иногда число-байт необходимо расширить до слова, т.е. нужно получить такое число с тем же значением, но размером в слово. Существует два способа такого расширения – без знака и со знаком. В любом случае исходное число-байт попадает во второй байт слова, а первый байт заполняется по-разному. При расширении без знака в него записываются нулевые биты, а при расширении со знаком в первый байт записываются нули, если расширяемое число положительное, и записываются восемь двоичных единиц в противном случае:

Число-байт	**Беззнаковое расширение**	**Знаковое расширение**
12	00 12	00 12
88	00 88	FF 88

Аналогичным образом выполняется расширение слова до двойного слова.

В языке Ассемблер есть безадресные команды:

CBW (convert byte to word) и
CWD (convert word to double),

которые выполняют ***знаковое*** расширение соответственно регистра AL до AX и AX до пары регистров <DX,AX>, которые в этом случае рассматриваются как один длинный 32-битный регистр.

2.1. Примеры решения задач

Пример1.

Пусть после выполнения команды **ADD AX, BX** содержимое регистра **AX** в шестнадцатеричном виде **F1 02** . Определить значение флагов **ZF, SF, PF,** соответствующих результату.

Решение.

ZF=0, т.к. результат операции не равен 0.

Представим содержимое **AX** в двоичной системе счисления.

Старший байт	**Младший байт**
1111 0001	0000 0010

Таким образом,
SF=1, т.к. результат операции отрицательный: самый левый бит равен1.
PF=0, т.к. младший байт результата содержит нечетное число единиц.

Пример 2.

Пусть A,B,C,D –переменные из диапазона [-50;50]. Разработать программу для вычисления значения выражения $Y=C^2+(B+D)*A$.

Решение.

Поскольку все переменные не превышают по размеру байт (в знаковый байт умещаются числа из диапазона [-128;127]), то для резервирования памяти воспользуемся директивой DB. Для резервирования памяти под Y, вычислим максимально и минимально возможные значения выражения:

$Ymax=50*50+(50+50)*50=7500,$

$Ymin=(0)^2+(-50-50)*50= -5000.$

Таким образом, для размещения знаковой переменной Y требуется отвести 2 байта. Следовательно, воспользуемся директивой DW.
Определим порядок выполнения операций в заданном выражении:

Операция и операнды	**C**	*****	**C**	**+**	**(B**	**+**	**D)**	*****	**A**
Номер		**3**		**4**		**1**		**2**	

Оценим размер памяти для хранения промежуточных результатов и значения выражения:
- результат суммы B+D занимает байт;
- произведение (B+D)*A требует резервирования слова;
- C^2 также займет слово.

```
PR1     SEGMENT
        ASSUME  CS:PR1,DS:PR1
        ORG     100H
START:
; ----------------------------------выполнение сложения B+D
XOR AX,AX               ; очищаем содержимое  AX
MOV AL, B               ; помещаем в AL переменную B
ADD AL,D                 ; выполняем сложение AL с D
; ----------------------------------выполнение умножения на A
IMUL B                  ; умножаем содержимое AL на B
; ----сохраняем результат вычисления второго слагаемого в регистр BX
MOV BX, AX
;----------------------------------возводим в квадрат  C
XOR AX,AX               ; очищаем содержимое  AX
MOV AL, C               ; помещаем в AL значение C
IMUL C                  ; умножаем содержимое AL на C
;--------------------вычисляем Y как сумму двух сложных слагаемых
ADD AX,BX
MOV Y, AX
INT       20H

A DB 10                         ; операнд A
B DB 20                         ; операнд B
C DB 30                         ; операнд C
D DB 40                         ; операнд D

Y DW ?                          ; переменная результата
PR1     ENDS
        END       START
```

2.3. Задачи для самостоятельного решения

1. Указать значения регистра CL (в виде как знакового, так и беззнакового десятичных чисел) и флагов CF, OF, SF и ZF после выполнения следующей пары команд:

а) MOV CL,128
ADD CL,128

б) MOV CL,-10
ADD CL,-40

в) MOV CL,246
ADD CL,216

г) MOV CL,40
SUB CL,100

2. Пусть

X DB ?

Требуется записать в регистр CL значение переменной X, увеличенное на 2. Определить, какой из следующих двух фрагментов правильно решает эту задачу:

а) MOV CL,X
ADD CL,2

б) MOV CL,X+2

3. Указать исходное значение регистра AL (любое из возможных), при котором после выполнения команды ADD AL,2 флаги имели бы следующие значения:

а) CF=1, OF=0, SF=0 б) CF=0, OF=1, SF=1

4. Пусть

X DD ?
Y DD ?
Z DD ?
W DD ?,?

Реализовать следующие операции над "длинными" числами:

а) Z=X+Y (считать, что сумма укладывается в двойное слово);
б) Z=X-Y (считать X≥Y);
в) W=X*Y (считать X и Y беззнаковыми числами).

5. A DB ?
B DW ?

Среди перечисленных команд указать те, что записаны с ошибкой:

а) ADD BX,'*' б) SUB AL,400 в) ADC BH,BL
г) SBB AX,BL д) ADD BX,B е) SUB CX,A
ж) ADC B,'B' з) CBW AL и) DEC A+1
к) DIV 100 л) SUB A,BYTE PTR B

6. Пусть T - переменная размером в двойное слово, а H, M и S - байтовые переменные. Считая, что прошло T секунд ($0 \le T < 86400$) от начала суток, определить, сколько полных часов (H), минут (M) и секунд (S) прошло к этому моменту времени.

7. Выписать фрагмент программы для вычисления значения выражения $x^3 - a*b$, если все переменные знаковые длиной в 1 байт.

8. Пусть

```
N DB ?   ; число без знака
K DW ?
```

Вычислить: K=N*(N+1)/2

9. Пусть в регистре BH находится код некоторой большой латинской буквы. Требуется записать в этот регистр код соответствующей малой латинской буквы. Определить, какой из следующих фрагментов правильно решает эту задачу:

а)
```
SUB BH,'A'
ADD BH,'a'
```

б)
```
ADD BH,'a'-'A'
```

в)
```
MOV BH, BH-'A'+'a'
```

10. Пусть

```
N DW ?   ; 1≤N≤365
WD DB ?
```

Записать в WD номер дня недели (1 - понедельник, ..., 7 - воскресенье), на который приходится N-й день года, считая, что 1 января этого года - понедельник.

3. ОРГАНИЗАЦИЯ ВЕТВЛЕНИЙ И ЦИКЛОВ

Цель занятия: изучить команды языка Ассемблер IBM PC для организации ветвлений и циклов и приобрести практические навыки разработки программ разветвляющейся и циклической структуры.

3.1. Основные теоретические сведения

Для реализации ветвящихся и циклических вычислительных процессов используются команды, которые обеспечивают переход к той или иной ветви вычислительного процесса. Физический адрес текущей выполняемой команды зависит от значений двух регистров: сегментного регистра **CS** и счётчика адреса **IP** и вычисляется по формуле:

$$A_{физ} = (CS*16 + IP)_{mod\ 2^{20}}$$

Следовательно, для осуществления перехода необходимо в один или оба эти регистра занести новые значения. Будем называть переход ***близким,*** если меняется ***только*** значение регистра **IP**, если же при переходе меняются значения двух регистров, то такой переход будем называть ***дальним*** (межсегментным) переходом.

В зависимости от способа изменения значения регистра переход может быть ***относительным*** или ***абсолютным.*** При относительном переходе происходит знаковое сложение содержимого регистра с некоторой константой, например,

$$IP = (IP + Const)_{mod\ 2^{16}}$$

При абсолютном переходе происходит просто присваивание соответствующему регистру нового значения, например,

```
CS = Const
```

Для сегментного регистра **CS** реализован только абсолютный переход, в то время как для счётчика адреса **IP** возможен как абсолютный, так и относительный переход.

В свою очередь, относительные переходы по величине той константы, которая прибавляется к значению счётчика адреса `IP`, могут быть ***короткими*** или ***длинными***. При коротком переходе величина этой знаковой константы (`i8`) не превышает по размеру одного байта (т.е. лежит в диапазоне от –128 до +127):

$$IP = (IP + i8)_{mod\ 2^{16}},$$

а при длинном переходе эта константа имеет размер слова:

$$IP = (IP + i16)_{mod\ 2^{16}}$$

Величина, используемая при абсолютном переходе для задания нового значения любого из этих регистров, может быть ***прямой*** или ***косвенной***. Прямая величина является просто числом (непосредственный адрес), а косвенная является *адресом* некоторой области памяти или регистром, откуда и будет извлекаться необходимое число, например,

```
IP = [m16]
```

Здесь в регистр **IP** заносится число, содержащееся в двух байтах памяти по адресу **m16**, т.е. это ***близкий длинный абсолютный косвенный*** переход.

Так как команды перехода предназначены только для передачи управления в другое место программы, то они ***не меняют*** флагов**.**

Команды перехода можно разделить на две группы:

1) команды безусловного перехода;
2) команды условного перехода**.**

3.1.1. Команды безусловного перехода

В языке Ассемблер все команды безусловного перехода обозначаются одинаково:

JMP op

Однако в зависимости от типа операнда, ассемблер формирует разные машинные команды. Возможные типы операнда указаны в таблице 3.1.

Таблица 3.1 – Типы операнда команды **JMP**.

op	Способ выполнения	Вид перехода
i8	**IP = (IP + i8)$_{mod\ 2^{16}}$**	Близкий относительный короткий
i16	**IP = (IP + i16)$_{mod\ 2^{16}}$**	Близкий относительный длинный
r16	**IP = [r16]**	Близкий абсолютный косвенный
m16	**IP = [m16]**	Близкий абсолютный косвенный
m32	**IP = [m32], CS = [m32+2]**	Дальний абсолютный косвенный
seg:off	**IP = off, CS = seg**	Дальний абсолютный прямой

Здесь **seg:off** – это мнемоническое обозначение двух операндов в формате **i16**, разделённых двоеточием.

Для указания близкого относительного перехода в команде обычно записывается метка команды, на которую необходимо выполнить переход.

Приведём примеры команд безусловного перехода:

JMP BX; близкий абсолютный косвенный переход, op=r16

JMP word ptr [BX]; близкий абсолютный косвенный переход, op=m16

JMP L . близкий относительный прямой переход . .

3.1.2. Команды условного перехода

Все команды условного перехода выполняются по схеме:

если <условие перехода>, то перейти к L

и производят ***близкий короткий относительный*** переход, если выполнено условие перехода, в противном случае продолжается последовательное выполнение команд программы.

Все команды условного перехода, кроме одной, вычисляют условие перехода, анализируя один или несколько флагов из регистра флагов, и лишь одна команда условного перехода вычисляет условие перехода, анализируя значение регистра **CX**.

Команды условного перехода в языке Ассемблер имеют следующую форму записи

J<мнемоника> ссылка

Мнемоника перехода (от одной до трёх букв) связана со значением анализируемых флагов (или регистра **CX**) или со способом формирования этих флагов. Чаще всего программисты формируют флаги, проверяя отношение между операндами (op1 и op2) с помощью команд вычитания или сравнения.

В таблице 3.2 приведена мнемоника команд условного перехода. Некоторые команды имеют разную мнемонику, но выполняются одинаково, такие команды указаны в одной строке таблицы.

Таблица 3.2– Мнемоника команд условного перехода

КОП	Условие перехода	
	Логическое условие перехода	Результат (rez) команды вычитания или соотношение операндов op1 и op2 команды сравнения
1	2	3
JE **JZ**	ZF = 1	rez = 0 или op1 = op2 (результат = 0, операнды равны)
JNE **JNZ**	ZF = 0	rez <> 0 или op1 <> op2 Результат <> 0, операнды не равны
JG **JNLE**	(SF=OF) **and** (ZF=0)	rez > 0 или op1 > op2 Знаковый результат > 0, op1 **больше** op2
JGE **JNL**	SF = OF	rez >= 0 или op1 >= op2 Знаковый результат >= 0, т.е. op1 **больше или равен** (**не меньше**) op2
JL **JNGE**	SF <> OF	rez < 0 или op1 < op2 Знаковый результат < 0, т.е. op1 **меньше** (**не больше или равен**) op2
JLE **JNG**	(SF<>OF) **or** (ZF=1)	rez <= 0 или op1 <= op2 Знаковый результат <= 0, т.е. op1 **меньше или равен**(**не больше**) op2
JA **JNBE**	(CF=0) **and** (ZF=0)	rez > 0 или op1 > op2 Беззнаковый результат > 0, т.е. op1 **выше** (**не ниже или равен**) op2
JAE **JNB** **JNC**	CF = 0	rez >= 0 или op1 >= op2 Беззнаковый результат >= 0, т.е. op1 **выше или равен** (**не ниже**) op2
JB **JNAE** **JC**	CF = 1	rez < 0 или op1 < op2 Беззнаковый результат < 0, т.е. op1 **ниже** (**не выше или равен**) op2
JBE **JNA**	(CF=1) **or** (ZF=1)	rez >= 0 или op1 >= op2 Беззнаковый результат >= 0, т.е. op1 **ниже или равен** (**не выше**) op2
JS	SF = 1	Знаковый бит разультата (7-й или 15-ый, в зависимости от размера) равен единице
JNS	SF = 0	Знаковый бит результата равен нулю
JO	OF = 1	Флаг переполнения равен единице
JNO	OF = 0	Флаг переполнения равен нулю

Продолжение таблицы 3.2.

1	2	3
JP **JPE**	PF = 1	Флаг чётности равен единице
JNP **JPO**	PF = 0	Флаг чётности равен нулю
JCXZ	CX = 0	Значение регистра CX равно нулю

Рассмотрим пример. Пусть надо реализовать переход на метку **L**, если **X>Y.** Соответствующий фрагмент для знаковых **X,Y** приведен ниже:

```
        MOV AX,X
        CMP AX,Y
        JG  L
              . . .
    L:  . . .
```

Этот фрагмент является корректным, если расстояние между меткой и командой условного перехода помещается в байт. В противном случае переход следует разбить на два перехода, например так, как это показано в следующем фрагменте:

```
        MOV    AX,X
        CMP    AX,Y
        JLE    L1
        JMP    L
      L1:
              . . .
     L:
```

3.1.3. Команды цикла

Для организации цикла с параметром в Ассемблере можно использовать специальные ***команды цикла***. Команды цикла, по сути, тоже являются командами условного перехода и, как следствие, реализуют только ***близкий короткий относительный*** переход. Команда цикла имеет формат

```
LOOP ссылка ; ссылка заменится на операнд i8
```

При выполнении команды **LOOP** сначала содержимое регистра **CX** уменьшается на 1, затем проверяется значение в регистре **CX**. Если

значение не равно 0, то переход по ссылке, иначе – к следующей команде. Регистр CX по сути является параметром цикла и называется счётчиком цикла.

Приведем схему организации цикла.

```
   MOV  CX,N; N–число повторений цикла
L:    . . . ;
                тело цикла
       . . .  ;
   LOOP L
```

Отметим, что такая схема организации цикла подходит только для случая CX >0. Если возможен вариант,что число повторений может быть нулевым, то при CX =0 надо сделать обход цикла:

```
MOV      CX,N
   JCXZ  L1  ;при CX=0 перейти к L1
L:       . . .; тело цикла
         . . .;
   LOOP  L
L1:            . . .
```

Кроме команды **LOOP** существуют следующие команды для организации циклов:

LOOPZ / LOOPE ссылка
LOOPNZ / LOOPNE ссылка

В этих командах проверяется еще значение флага **ZF**. В командах **LOOPZ / LOOPE** переход по ссылке выполняется, если **CX≠0 *и* ZF =1**, а в команде **LOOPNZ / LOOPNE**, если **CX≠0 *и* ZF =0.**

3.2. Примеры решения задач

Пример 1.

Выписать фрагмент программы для вычисления значения **X** по формуле

$$X := \begin{cases} (A+1)*(A-1), & \text{при } A>2 \\ 4, & \text{при } A=2 \\ (A+1) \bmod 7, & \text{при } A<2 \end{cases}$$

Решение.

Пусть A и X– целые, длиной в слово, тогда для резервирования памяти используем директивы **DW** . Кроме того, введем переменную err – признак корректности выполнения арифметических операций, равный 1, если возникло переполнение и равный 0 в противном случае.

```
A  DW ?
X  DW ?
err      DB 0

   MOV   AX,A  ; AX := A
   MOV   BX,AX ; BX := A
   INC   AX    ; AX := A+1
   JO    Error
   CMP   BX,2  ; Сравнение A и 2
   JLE   L1
   ; Ветвь1 вычисления X
   DEC   BX    ; BX := A-1
   JO    Error
   IMUL  BX    ; (DX,AX):=(A+1)*(A-1)
   JO    Error ;  Произведение  (A+1)*(A-1)  не
помещается в ax

L: MOV   X,AX  ; Результат, ветви 1,2, или 3
JMP      Endprog
L1:      JL    L2
   ; Ветвь2 вычисления X
   MOV   AX,4
   JMP   L;    На вывод результата
L2:      MOV   BX,7
; Ветвь3 вычисления X
   CWD       ; (DX,AX):= длинное (A+1)
   IDIV  BX;   DX:=(A+1) mod 7, ax:=(A+1) div 7
   MOV   AX,DX
   JMP   L;    На вывод результата
Error: MOV     err, 1
Endprog:
           ...
```

Пример 2.

Задан массив из 12 целых чисел, возможные значения лежат в интервале 7 ÷ 240. Привести фрагмент программы для вычисления суммы элементов массива, значение которых больше 115.

Решение.

Для хранения значения элемента массива достаточно одного байта, а для хранения значения суммы необходимо слово (два байта), так как сумма может принимать значения от 0 до 1380. Элементы массива и значение суммы являются беззнаковыми числами. Для хранения элементов массива выделим область памяти M длиной 12 байтов, для подсчета суммы будем использовать регистр BX.

Для организации обращения к различным элементам массива необходимо использовать режим адресации с возможностью изменения исполнительного адреса в процессе выполнения программы. Это возможно реализовать при косвенной, базовой или индексной адресации. Выберем косвенную адресацию и регистр SI в качестве регистра-модификатора.

```
M   DB   23,160,176,8,97,54,23,34,62,115,48,116  ; объявление
                    ; массива чисел
    XOR   BX,BX     ; обнуление суммы
    LEA   SI,M      ; загрузка адреса 1-го элемента массива
    MOV   CX,12     ;  загрузка числа повторений цикла
L1: MOV AL,[SI]          ;  загрузка значения очередного
элемента
    CMP   AL,115  ; сравнение значения элемента с числом 115
     JBE   L2  ; переход  по меньше или равно
    MOV AH,0        ; расширение беззнакового числа до слова
    ADD   BX,AX     ; прибавление к сумме значения элемента
L2: INC   SI        ; вычисление адреса следующего элемента
    LOOP   L1       ; реализация циклического процесса
    INT 20H         ; команда завершения вычислений
```

3.3. Задачи для самостоятельного решения

1. Задан массив из 23 целых чисел в интервале -700 ÷ 21000. Напишите программу на языке Ассемблера, которая вычисляет сумму элементов Xi, значение которых меньше 3150.

2. Задан массив из 44 целых чисел в интервале -6000 ÷ 9000. Напишите программу на языке ассемблера, которая определяет количество пар соседних элементов, в которых произведение элементов больше 4500.

3. Задан массив из 25 целых чисел в интервале 16 ÷ 253. Напишите программу на языке ассемблера, которая определяет количество элементов в массиве кратных 11.

4. X DB 206 ; (-50)
Определить, будет ли сделан переход на метку MET при выполнении следующих команд:

```
а) CMP X,210      б) CMP X,210     в) CMP X,-40    г) CMP X,-40
JA MET              JE LAB           JG MET           JL MET
                    JB MET
```

5. Пусть X и Y - знаковые байтовые переменные, а L - метка.
Реализовать следующие условные переходы:
а) if (X>2) or (Y<10) then goto L;
б) if (X>2) and (Y<10) then goto L

6. Пусть X, Y, Z и W - знаковые переменные-слова. Реализовать следующее присваивание:
а) Y:=abs(X) б) Z:=min(X,Y) в) W:=max(X+10,2*Y,8-Z)

7. В регистре BL находится число от 0 до 15. Записать в BL код соответствующей шестнадцатиричной цифры как символа (в качестве "буквенных" цифр использовать большие латинские буквы от 'A' до 'F').

8. N DW ? ; 0≤N≤999
Определить, содержит ли десятичная запись числа N цифру 5. Ответ - 1 (содержит) или 0 - записать в регистр DL.

9. Пусть Y - переменная-слово со значением от 1 до 2100. Определить, является ли год Y високосным, и в регистр CL записать 1, если является, и 0 в противном случае. (По "новому стилю" високосными считаются года, кратные 4, однако из всех годов, кратных 100, високосными являются лишь кратные 400: 1700, 1800 и 1900 - невисокосные, 2000 - високосный).

10. N EQU 100
X DB N DUP(?) ; числа со знаком
Y DB ?

Считая, что элементы массива X упорядочены по возрастанию, определить, есть ли в X элемент, равный Y, и записать ответ 1 (есть) или 0 а в регистр AX.
При решении задачи использовать метод бинарного поиска.

4. ОБРАБОТКА БИТОВЫХ ДАННЫХ

Цель занятия: изучить команды языка Ассемблер IBM PC для обработки битовых данных, приобрести навыки использования логических команд и команд сдвига.

4.1. Основные теоретические сведения

Процессор выполняет обработку только 8- и 16-битовых данных. В задачах системного программирования возникает необходимость обработки данных, имеющих другой размер, например 3 или 12 битов. Для реализации таких операций применяются логические команды и команды сдвига. В Ассемблере предусмотрены 5 логических команд и восемь команд сдвига (таблица 4.1), которые выполняются над 8- и 16-битовыми операндами.

Таблица 4.1 – Логические команды и команды сдвига

Мнемоника команды	Описание команды
Логические команды: **AND**(логическое И) **OR**(логическое ИЛИ) **XOR**(исключающее ИЛИ) **NOT**(инверсия) **TEST**(проверка)	 приемник**&**источник → приемник приемник|источник → приемник приемник ⊕ источник → приемник ~ операнд → операнд приемник&источник
Команды линейного сдвига: **SHL**(логический сдвиг влево) **SHR**(логический сдвиг вправо) **SAL**(арифметический сдвиг влево) **SAR**(арифметический сдвиг вправо)	} операнд сдвигается, освободившийся разряд заполняется 0, выдвигаемый бит заносится в CF операнд сдвигается с сохранением значения знакового бита, выдвигаемый бит – в CF
Команды циклического сдвига: **ROL**(циклический сдвиг влево) **ROR**(циклический сдвиг вправо) **RCL**(циклический сдвиг влево через перенос) **RCR** (циклический сдвиг вправо через перенос)	 } операнд сдвигается, освободившийся разряд заполняется значением выдвигаемого разряда, выдвигаемый бит – в CF } сдвигаются биты операнда, выдвигаемый бит – в CF, освободившийся разряд заполняется прежним значением CF

Команды **OR, AND, XOR** выполняют операции логического (***побитового***) сложения, умножения и сложения по модулю 2 двух операндов. Результат замещает первый операнд (приемник); второй операнд (источник) не изменяется. Команда **NOT** инвертирует все разряды операнда. Команда **TEST** выполняется как команда **AND**, но результат не записывается в приемник. Взаимодействие битов для команды **OR** отражено в таблице 4.2.

Таблица 4.2– Правила выполнения логических команд

Приемник	Источник	OR	AND	XOR	NOT
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Логические команды (кроме **NOT)** воздействует на флаги **OF, SF, ZF, PF** и **CF,** при этом флаги **CF,OF** всегда сбрасываются в 0.

Команда **SHL** (shift logical left – сдвинуть логически влево) и **SHR** (shift logical вправо) сдвигают числа без знака. Команда **SHR** аналогична команде **SHL** но сдвигает операнд не влево, а вправо. При каждом сдвиге операнда команда **SHR** заносит 0 в вакантный старший бит этого операнда (бит 7 при сдвиге байта, бит 15 при сдвиге слова).

Команды **SAL** (shift arithmetic left – сдвинуть арифметически влево) и **SAR** (shift arithmetic right – сдвинуть арифметически вправо) сдвигают числа со знаком. Команда **SAR** сохраняет знак операнда, репродуцируя его при выполнении сдвига. Команда **SAL** не сохраняет знак, но заносит 1 во флаг переполнения **OF** в случае изменения знака операнда. При каждом сдвиге операнда команда **SAL** заносит 0 в вакантный нулевой бит обрабатываемого операнда.

Поскольку сдвиг операнда на ***n*** битов влево означает умножение операнда на 2^n, а сдвиг на ***n*** битов вправо означает деление операнда на 2^n, то команды сдвига целесообразно использовать в качестве команд быстрого умножения и деления. Например:

```
MOV DL, 5   ; DL = 5  = 00000101
SHL DL, 1   ; DL = 10 = 00001010
MOV CL, 1   ; CL = 1
SHL DL, CL  ; DL = 20 = 00010100
```

Логические команды и команды сдвига используются для анализа, выделения и формирования данных, представленных последовательностью битов в байте или слове.

4.2 Примеры решения задач

Пример 1.

В памяти имеется переменная, содержащая координаты точки на плоскости в формате 010YYYY01P1XXXX1, где Y и X – смещения в виде 4-х разрядных двоичных чисел, P – флаг наличия данных. Разработать программу для преобразования данных в формат YYYYXXXX и формирования признака наличия данных 47H.

Решение. Исходные данные составляют 16 битов, для их представления выделим переменную S размером в слово. Для представления результата выделим две переменные D и M размером в байт каждая .

```
PR1         SEGMENT
            ASSUME  CS:PR1,DS:PR1
            ORG     100H
START:
MOV  M,0              ; установка признака отсутствия данных
; ---------------------------------проверка наличия данных
TEST S,40H            ; проверка значения флага P
JZ   LE               ; переход  при P=0
; -----------------------------------выделение значения X
MOV  BX,S             ; загрузка в BX данных
AND  BX,011110B       ; выделение в BX значения X
; -----------------------------------выделение значения Y
MOV  AX,S             ; загрузка в AX данных
AND  BX,1E00H                 ; выделение в AX значения Y
; -----------------------------------формирование выходных данных
SHL  BX,1             ; сдвиг значения X на позицию 0-3 битов
MOV  CL,5             ; загрузка в CL числа сдвигов
SHL  AX,CL            ; сдвиг значения Y на позицию 4-7 битов
OR   BL,AL            ; добавление значения Y к значению X
; -----------------------------------запись результата
MOV  D,BL             ; запись координат
MOV  M, 47H           ; запись признака данных
```

```
LE:  INT  20H

S DW 0100110011110111    ; исходные данные
D DB ?                   ; преобразованные данные
L DB ?                   ; признак наличия данных
PR1         ENDS
            END       START
```

4.3. Задачи для самостоятельного решения

Замечание:

в задачах 1-3 переменные определены следующим образом:

A DB ?
B DB ?
X DW ? ; число со знаком

Рассматривая A и B как логические переменные, принимающие лишь значения 0 ("ложь") и 0FFh ("истина"), реализовать следующие присваивания.

1. A = A and not B or not A and B
2 A = B or (X>2) and A
3. A = A>=B (команды условного перехода не использовать)

***Замечание*:**

в задачах 4-7 реализовать указанную операцию.

4. Сделать переход на метку L, если 4 средних бита регистра AL - это 1001b;
5. Сделать переход на метку L, если 2 правых бита регистра AL равны 2 правым битам регистра BL;
6. Сделать переход на метку L, если равны 3 левых и 3 правых бита регистра AX;
7. Записать в регистр CL байт, составленный из 4 левых битов регистра AL и 4 правых битов регистра BL;

8. X DD ? ; число со знаком

Не используя команды условного перехода, записать в регистр AL знаковый бит числа X, т.е. 1, если X<0, и 0 иначе.

9. Пусть X и Y - беззнаковые переменные-слова. Не используя команды умножения и деления, реализовать следующее присваивание:
а) Y=4*X - X div 8 + X mod 16
б) Y:=35*X

10. X DW ? ; число без знака
Реализовать условный переход на метку L, если X - четное число, четырьмя способами - соответственно с помощью команд DIV, AND, TEST и SHR. Какой из этих способов лучше и почему?

5. ОБРАБОТКА СТРОКОВЫХ ДАННЫХ

Цель занятия: изучить команды языка Ассемблер IBM PC для обработки строк и приобрести практические навыки их применения.

5.1. Основные теоретические сведения

В ПК под строкой понимается последовательность соседних байтов или слов. В связи с этим все строковые команды имеют две разновидности – для работы со строками из байтов (в мнемонику операций входит буква B) и для работы со строками из слов (в мнемонику входит буква W). Максимальная длина строки составляет 64 Кб.

В языке Ассемблер IBM PC имеется пять команд для обработки строковых данных. Все команды обработки строк являются ***безадресными***:

MOVSB/MOVSW – переслать байт/слово из одной области памяти (источник) в другую (приемник);

LODSB/LODSW – загрузить из памяти (источник) байт/слово в регистр **AL/AX**;

STOSB/STOSW – записать содержимое регистра **AL/AX** в память (приемник);

CMPSB/CMPSW – сравнить содержимое двух областей памяти (приемник и источник) длиной байт (слово),

SCASB/SCASW – сравнить содержимое регистра **AL(AX)** с содержимым области памяти (приемник).

В описаниях команд источник – это ячейка памяти с адресом **DS:SI**, а приемник – ячейка памяти с адресом **ES:DI**.

Каждая из этих команд выполняется только над одним элементом строки. Однако одновременно происходит автоматическая настройка на следующий или предыдущий элемент строки: после выполнения операции значение регистра **SI** и/или **DI** увеличивается (при **DF=0**) или уменьшается (при **DF=1**) на 1 (для байтовых строк) или на 2 (для строк из слов).

Инициализация значений регистров и флага **DF** должна быть выполнена до начала операции над строкой. Флаг **DF** установить в 0 можно с помощью команды **CLD**, а в 1 –с помощью команды **STD**.

Если сегментный регистр **DS** уже имеет нужное значение, тогда загрузить регистр **SI** можно с помощью команды

LEA SI,<начальный/конечный адрес строки>

Если надо загрузить оба регистра DS и SI, тогда можно воспользоваться командой

LDS SI,m32

которая в регистр **SI** заносит первое слово, а в регистр **DS** - второе слово из двойного слова, имеющего адрес **m32** (таким образом, по адресу **m32+2** должен храниться сегмент, а по адресу **m32** - смещение начального или конечного элемента строки). Начальную загрузку регистров **ES и DI** обычно осуществляют одной командой

LES DI,m32 ; аналогична команде **LDS**

Обработка последовательности элементов (слов или байтов) одной командой может быть обеспечена использованием префикса **REP.** Предварительно в регистр **CX** необходимо поместить число таких повторений, т.е. число обрабатываемых элементов. Это позволяет указанным командам обрабатывать строки различной длины.

Префикс **REP** указывается непосредственно перед командой. Например,

REP MOVSB

При выполнении этой команды происходит уменьшение на 1 значения в регистре **CX** до 0.

При использовании команд **CMPS и SCAS** возможно применение модификаций префикса **REP**:

REPZ/REPE – повторять операцию пока флаг **ZF** показывает "равно или ноль". Прекратить операцию, если **ZF** "не равно или не ноль" или при **CX=**0;

REPNE/REPNZ – повторять операцию пока флаг **ZF** показывает "не равно или не ноль". Прекратить операцию при флаге **ZF**, указывающем на "равно или ноль" или при **CX=0.**

5.2 Примеры решения задач

Пример 1.

Задана строка из 300 символов. Привести фрагмент программы, в котором из строки копируется в новую строку второе предложение. Считать, что предложения разделены символом '.'.

Решение**.**

Пусть A и B– области памяти для хранения исходной и результирующей строк соответственно.

Для копирования предложения будем использовать команду **MOVSB,** для этого необходимо задать число копируемых элементов N в регистре **CX**, адрес первого копируемого элемента – в регистре **SI**, адрес первого элемента строки приемника – в **DI**(рисунок 5.1).

Для поиска второго предложения воспользуемся командой поиска **SCASB.** Для поиска в строке в регистр **CX** заносим 300 (число элементов в строке), в аккумулятор **AL** записываем код символа '.', а в регистр **DI** адрес начала поиска (**DI_0**).

После первого поиска регистр **DI** будет указывать на элемент, следующий за символом '.' (**DI_1**), при этом значение в регистре **DI** совпадает со значением регистра **SI** для копирования. Затем, выполнив команду поиска второй раз, получим новое значение в регистре **DI** ((**DI_2**). Для определения числа копируемых символов находим разность **DI_2** – **SI**.

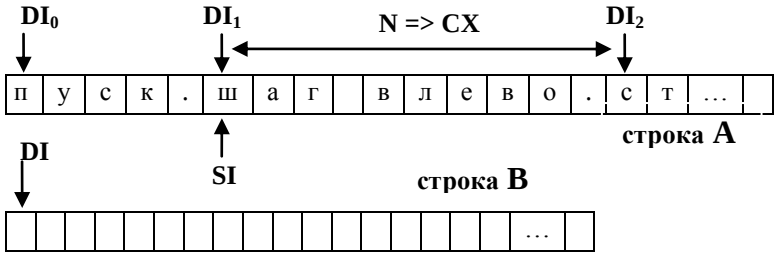

Рисунок 5.1 – Схема решения задачи

Представим фрагмент программы .

```
A    DB   'пуск.шаг влево.стоп.',279 dup ' '  ; объявление
B    DB   299 DUP ' '        ; строк
   CLD              ; установка направления обработки строки
                    ; от младших адресов к старшим (DF = 0)
   LEA  DI,A        ; загрузка адреса начала строки
   MOV  CX,300      ; загрузка числа элементов строки
   MOV AL,'.'       ;  загрузка кода искомого элемента
REPNE SCASB         ;  обработка элементов строки пока
                    ; «не равно», т.е поиск символа
```

```
   JNE   LE        ; переход по «не равно» на метку LE,
                   ; если символ '.' в строке не обнаружен
   MOV SI,DI       ; запись адреса второго предложения
REPNE SCASB        ; продолжение поиска (поиск 2-й точки)
   JNE   LE        ; переход по «не равно» на метку LE,
                   ; если символ '.' в строке не обнаружен
   SUB DI,SI       ; вычисление числа копируемых элементов N
   MOV CX,DI       ; запись N в регистр CX
   LEA  DI,B       ; загрузка адреса строки B
REP MOVSB          ; пересылка (копирование)
                   ; 2-го предложения
LE:   INT 20H      ; завершение вычислений
```

5.3. Задачи для самостоятельного решения

1. Задана строка из 520 символов. Удалить семь символов в строке после первого символа '='.

2. Задана строка из 342 символов. Вставить в строку вместо первого символа '%' подстроку '0246897531' .

3. Задана строка из 483 символов. Определить адрес первого символа в строке из заданных ('@$#&%*+=!').

4.Заданы две строки A и S. Если в строку A входит пробел, тогда сделать значением переменной B подстроку из 15 начальных символов строки S.

5. Заданы две строки A и B. Если строки A и B равны (состоят из равного числа попарно равных символов), тогда в регистр AL записать 1, а иначе - 0.

6. В строке из 300 символов символы '*' заменить символами '!'.

7. Задана строка A из 100 слов. Найти **слово** '**ФФ**'. Если слово найдено, то переслать его в регистр BX, а его номер –в DL.

8. Пусть
  str1 DB 'ASDFGFGHHJ gggUUrr'
  str2 DB 40 dup (' ')
Выполнить пересылку str1 в str2 слева направо.

9. Пусть
  str1 DW 'ASDFGFGHHJ gggUUrr'
  str2 DW 40 dup (' ')
Выполнить пересылку str1 в str2 справа налево.

10. Задана строка из 200 символов. Определить адрес первого вхождения подстроки 'AB' в строку и занести его в регистр BX.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Абель П. Язык Ассемблера для IBM PC и программирования**.**– М.: Высшая школа, 2006.–447с.

2. Пильщиков В.Н. Программирование на языке Ассемблера IBM PC. – М.: Диалог – МИФИ, 2014. – 288 с.

3. Баула В. Г. Архитектура ЭВМ и язык ассемблера Части 1-3. – М.: Изд-во факультета ВМиК МГУ,2006.–117с.

4. Пильщиков В.Н. Упражнения по языку ассемблера MASM (учебное пособие). – М.: Изд-во факультета ВМиК МГУ,2006.–40с.

Заказ №_____от «___»_____2016 ____Тираж_____экз.
Изд-во СевГУ