

## Лабораторная работа 2

### Адресация и ввод/вывод в программе на Ассемблере

#### Основные задачи

1. Основные операции языка Ассемблер.
2. Изучение способов адресации.
3. Изучение операций ввода/вывода
3. Разработка программы на Ассемблере

Работа с данными, находящимися в оперативной памяти, это целиком забота программиста. Для определения адреса данных необходимо хорошо знать способы адресации оперативной памяти. Адрес оперативной памяти состоит из двух частей: адреса сегмента и смещения относительно начала сегмента. Адрес начала сегмента находится в регистре сегмента, значение смещения в одном из регистров, как правило, это регистры BX, DI, SI, SP, BP.

#### Способы адресации операндов

В программах на **Assembler** применяются следующие типы адресации операндов: регистровая, прямая, непосредственная, косвенная, базовая, индексная, базово-индексная.

**Регистровая** адресация подразумевает использование в качестве операнда регистра, например:

```
push DS  
mov BP,SP
```

При **прямой** адресации один операнд представляет собой адрес памяти, второй - регистр:

```
move DATA, AX
```

**Непосредственная** адресация применяется, когда операнд длиной в байт или слово находится в ассемблерной команде:

```
mov AX, 4Ch
```

При использовании **косвенной** адресации исполнительный адрес формируется исходя из сегментного адреса в одном из сегментных регистров и смещения в регистрах BX, BP, SI или DI, например:

```
mov AL, [BX]; база находится в регистре DS, смещение в регистре BX
```

```
mov AH, [SI]; база - в DS, смещение - в SI
```

```
mov AX,[DI]; база в DS, смещение - в DI
```

```
mov AX, ES: [DI]; база - в ES, смещение - в DI
```

```
mov DX, [BP]; база - в SS, смещение - в BP
```

В случае применения **базовой** адресации исполнительный адрес является суммой значения смещения и содержимого регистра BP или BX, например:

```
mov AX, [BP+6]; база - SS, смещение - содержимое BP, которое складывается
```

с 6

```
mov [BX+Delta], AX; база - DS, смещение - содержимое BX+смещение Delta
```

```
mov AX, [BP]+4; база - SS, смещение - содержимое BP+4
```

```
mov DX, 8[BX]; база - DS, смещение - содержимое BX+8
```

При индексной адресации исполнительный адрес определяется как сумма значений указанного смещения и содержимого регистра SI или DI так же, как и при базовой адресации, например:

*mov DX, [SI+5]; база - DS, смещение - SI+5*

*mov ES:[DI]+6,AL ;база - ES, смещение - DI+6*

Базово-индексная адресация подразумевает использование для вычисления исполнительного адреса суммы содержимого базового и индексного регистров, а также смещения, находящегося в операторе, например:

*mov BX, [BP][SI]*

*mov ES:[BX+DI],AX*

*mov Array[BX][SI],12h*

*mov AX,[BP+6+DI]*

*mov Array [BP+BX]; ошибка - два базовых регистра*

*mov Array [DI+SI]; ошибка - два индексных регистра*

### Флаги

Девять флагов размещены в регистре флагов. Флаги имеют следующие значения:

1. Бит 0, флаг переноса **CF (carry flag)**. Изменяется во многих операциях.
2. Бит 2, флаг четности **PF (parity flag)**.
3. Бит 4, вспомогательный флаг переноса **AF (auxiliary carry flag)**.
4. Бит 6, флаг нуля **ZF (zero flag)**, равен 1, если результат операции равен 0.
5. Бит 7, флаг знака **SF (sign flag)**. SF равен 1, если результат отрицательный.
6. Бит 8, флаг трассировки **TF (trap flag)**,
7. Бит 9, флаг прерывания **IF (interrupt enable flag)**, разрешает прерывания от внешних устройств. Если IF=0 прерывания запрещены.
8. Бит 10, флаг направления **DF (direction flag)**.
9. Бит 11, флаг переполнения **OF (overflow flag)**.

### Обработка массивов

Обработка массивов обычно выполняется в цикле. Кроме того в программе часто необходимо выполнять многократно некоторые фрагменты программы. Для этого в Ассемблере имеется удобное решение организации циклов. Количество повторений цикла заносится в регистр **CX**, а команда **LOOP** обеспечивает выполнение заданного количества повторений. Как правило, команда **LOOP** ставится в конце повторяющейся последовательности команд программы.

#### *Loop метка*

Команда вычитает единицу из содержимого регистра **CX** и, если содержимое не равно нулю, передает управление на метку. Если регистр **CX** становится равен нулю, выполняется следующая за **LOOP** команда. В регистр **CX** заносится количество повторений цикла.

Например: сложить десять чисел, которые находятся в таблице с именем **TAB**.

*Mydata segment*

*Tab DB 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ;значения элементов массива*

*Mydata ends*

*Mycode segment*

...

<i>MOV CX, 10</i>	<i>;количество повторений</i>
<i>MOV SI, 0</i>	<i>;ноль в индексный регистр</i>
<i>MOV BL, 0</i>	<i>сумма будет в BL</i>
<i>BEGIN: ADD BL, TAB[SI]</i>	<i>;добавляем элемент к сумме</i>
<i>INC SI</i>	<i>;переходим к следующему элементу</i>
<i>LOOP BEGIN</i>	<i>;возврат на начало цикла</i>
<i>...</i>	
<i>Mycode ends</i>	

## Команды управления программой

### Команда сравнения

#### *CMP операнд1, операнд2*

Команда вычитает операнд2 из операнда1 и устанавливает регистры флагов. Операнды не меняются. За ней обычно идет команда условного перехода.

### Команды условного перехода

В Ассемблере имеется группа команд условного перехода, которые проверяют состояние флагов в регистре флагов:

Команда проверки состояния флага переноса

**JC** перейти если флаг CF установлен в единицу (например, при сдвиге бита из регистра)

**JZ** перейти, если флаг ZF установлен в единицу (например, результат операции равен нулю)

**JS** перейти, если флаг SF установлен в единицу (проверяет только старший бит байта или слова)

**JP** перейти, если флаг PF установлен в единицу (в результате операции количество единиц четно)

**JO** перейти, если флаг OF установлен в единицу (операция завершилась переполнением)

### Переходы при работе с беззнаковыми данными

Флаги, на которые  
реагирует команда

<i>JE/JZ переход если результат равен нулю</i>	<i>ZF</i>
<i>JNE/JNZ переход если не равно./не нуль</i>	<i>ZF</i>
<i>JA/JNBE если выше/ не ниже или равно</i>	<i>ZF,CF</i>
<i>JAE/JNB если выше или равно/ не ниже</i>	<i>CF</i>
<i>JB/JNAE если ниже/не выше или равно</i>	<i>CF</i>
<i>JBE/JNA если ниже или равно/не выше</i>	<i>CF, AF</i>

### Переходы при работе со знаковыми данными:

<i>JE/JZ если равно/нуль</i>	<i>ZF</i>
<i>JNE/JNZ если не равно/не нуль</i>	<i>ZF</i>
<i>JG/JNLE больше/не меньше или равно</i>	<i>ZF,SF,OF</i>
<i>JGE/JNL больше или равно/не меньше</i>	<i>SF,OF</i>
<i>JL/JNGE меньше/не больше или равно</i>	<i>SF,OF</i>
<i>JLE/JNG меньше или равно/не больше</i>	<i>ZF,SF,OF</i>

### Команда безусловного перехода JMP.

#### *JMP метка*

Передает управление на метку.

### Команды логических операций

Устанавливают флаги

**PF, SF, ZF**

**Команда AND источник, приемник (логическое умножение).** Побитно выполняет операцию логического умножения над источником и приемником. Если оба обрабатываемых бита равны 1, результат 1, иначе 0. Как правило операция применяется для обнуления заданных в операнде битов или для выделения части битов кода.

Например, необходимо убрать признак кода ASCII в введенной с клавиатуры цифре. Тогда

**MOV AH, 1**

Введем с клавиатуры

**INT 21H**

Цифру по прерыванию 21H в регистр AL

**AND AL, 00001111B** Удалим признак кода ASCII (т.е. старшие 4 бита)

**Команда OR источник, приемник (логическое сложение).** Побитно складывает биты в операндах. Если хотя бы один из пары обрабатываемых битов равен 1, то результат 1, иначе 0. Обычно используется для установки заданных битов в 1.

Например, надо добавить признак кода ASCII в выводимую на экран цифру. Тогда:

**MOV DL, 7**

В регистре DL получаем двоичную семерку

**OR DL, 00110000B**

и добавляем к ней признак ASCII (т.е. в старшие 4 бита)

**Команда XOR источник, приемник (сложение по модулю 2).** Побитно складывает биты операндов (но переноса в старший бит нет). Если оба бита одинаковых, результат 0, иначе 1. Обычно используется для изменения заданных битов в обратное состояние. Например, изменим значения старших 4 битов на обратные. Тогда:

**MOV AL, 7BH**

В AL комбинация 01111011

**XOR AL, 11110000B**

После операции в AL 10001011

**Команда TEST источник, приемник.** Аналогична команде **AND**, но не изменяет операнды, а только флаги **ZF, SF, PF**. Идущая после этой команды команда условного перехода определяет каков был результат.

### Команды сдвига

**Команда SHR dest, 1 или SHR dest, CL** (Сдвиг вправо). Последний сдвигаемый бит помещается во флаг **CF**

**Команда SHL dest, 1 или SHL dest, CL** (Сдвиг влево). Последний сдвигаемый бит помещается во флаг **CF**.

Команды сдвига обычно используется для анализа состояния байта или слова.

### **Варианты заданий**

Для реализации заданий можно воспользоваться программой ввода/вывода, приведенной в приложении 1.

1. Задана последовательность символов, заканчивающаяся точкой. Подсчитать количество битов в состоянии 0 для каждого символа. Создать новый массив, который содержит эти значения для каждого символа. Выведите на экран символ, его двоичное представление и количество битов в состоянии 1.

2. Задан массив из 50 слов. Первое число массива содержит действительное количество чисел в массиве. В каждом слове находятся два числа: одно в битах 14, 13, 12, 11, 10, второе в битах 8, 7, 6, 5, 4, 3. Если бит 1 в слове не равен 0, то умножить эти два числа и поместить произведение в новый массив. Выведите на экран числа и произведения.

3. Задан массив из 30 слов. В каждом слове находятся два числа: одно в битах 14, 13, 12, 11, 10, второе в битах 8, 7, 6, 5, 4, 3. Если бит 14 и бит 13 в первом числе совпадают с битами 8 и 7 второго числа, то поместить их в два разных массива.

4. Задан массив из 40 слов. В каждом слове находятся два числа: одно в битах 12, 11, 10, 9, 8, второе в битах 7, 6, 5, 4, 3, 2. Если бит 9 и бит 8 в первом числе совпадают с битами 3 и 2 второго числа, то поместить первое число в новый массив.

5. Задано слово и массив из 32 чисел форматом слово. Каждому биту слова соответствует пара чисел из массива: первому биту слова соответствуют числа 1 и 2 из массива, второму биту - числа 3 и 4, и т.д. Если бит слова равен 1, умножить соответствующую пару чисел и, если произведение больше слова, поместить его в новый массив.

6. Задана таблица:

TABL DB 0, 2, 4, 9, 0, 12, 8

DB 5, 12, 7, 0, 8, 0, 5

DB 0, 5, 14, 6, 15, 9, 0

DB 13, 5, 18, 45, 3, 9, 11

Подсчитать количество нулей в каждой строке и запомнить их в массив. Заменить нули на число FFH. Выведите количество нулей в каждой строке на экран. Если нулей в строке не обнаружено, то вывести на экран символ ?.

7. Задан массив из 50 байтов. Первый байт содержит следующую информацию: биты 7, 6, 5, 4, 3, содержат число, означающее реальное количество элементов массива, которое необходимо обработать, во втором байте: биты 7, 6, 5 константу C1, биты 4, 3, 2, 1, 0 константу C2. Если числа массива (начиная с третьего) больше 230, умножить его на C1, иначе на C2. Все произведения (слова!) поместить в новый массив.

8. Задан массив из 40 слов. В каждом слове находятся два числа: одно в битах 14, 13, 12, 11, 10, второе в битах 8, 7, 6, 5, 4. Если бит 11 и бит 10 в первом числе совпадают с битами 5 и 4 второго числа, то числа поместить в два разных массива.

9. Задано слово и массив из 16 чисел форматом слово. Каждому биту слова соответствует число из массива. Если бит в слове равен 1, то соответствующее слово из массива разделить на 10. Ненулевые остатки от деления поместить в новый массив и вывести на экран.

10. 6. Задана таблица:

TAB DB 5, 2, 0, 9, 11, 16, 7

DB 5, 12, 7, 0, 8, 0, 9

DB 5, 0, 14, 6, 15, 9, 31

DB 13, 5, 18, 45, 3, 9, 45

Числа больше 10 заменить на 9. Вывести на экран количество таких чисел в каждой строке и запомнить их в новый массив

11. Задан текст из 100 символов, содержащий слова произвольной длины. Слова в тексте разделены пробелами. Создать новый массив, в который поместить количества букв в каждом слове. Вывести на экран количество слов в тексте.

12. Задан текст из 100 символов, содержащий слова произвольной длины. Слова в тексте разделены пробелами. Создать новый массив, в который поместить количества букв в каждом слове. Вывести на экран все слова текста, каждое с новой строки.

### **Выполнение лабораторной работы**

1. Напишите программу на Ассемблере в соответствии с заданием.
2. Выполните ассемблирование с помощью MASM.
3. Распечатайте листинг программы
4. Распечатайте результаты работы программы
5. Подготовьте отчет по лабораторной работе.

### **Контрольные вопросы**

1. Что такое адрес оперативной памяти?
2. Из каких составных частей состоит адрес?
3. Какие методы адресации предоставляет Ассемблер?
4. Для чего используются флаги и какими средствами они проверяются?
5. Как организуется цикл?
6. Как извлечь элемент массива?
7. Что такое условный переход?
8. Каково назначение индексного регистра?