Лабораторная работа № 2_2

Команды пересылки данных. Режимы адресации

Цель работы:

Практическое овладение навыками разработки программного кода на языке Ассемблер. Изучение команд передачи данных и использования различных способов адресации операндов. Практическое освоение основных функций отладчика TD.

Порядок выполнения работы:

1. Создать рабочую папку для текстов программ на ассемблере и записать в нее файлы tasm.exe, tlink.exe, rtm.exe и td.exe из пакета tasm, а также файл с исходным текстом программы на ассемблере, который сохранить с именем prog№.asm.
2. Создать загрузочный модуль, загрузить его в отладчик и выполнить программу в пошаговом режиме.

Содержание отчета:

1. Цель работы.
2. Постановка задачи.
3. Листинг программ.
4. Таблицы состояния регистров в ходе выполнения программ для задания 1 и 2.
5. Ответы на контрольные вопросы.
6. Вывод.

Постановка задачи:

1. Написать программу с именем Lab№.asm используя различные виды режимов адресации, сделать исполняемый файл, и проследить за работой в Турбоотладчике (фрагмент программы Приложение 1).

2. Опишите в сегменте данных следующую информацию:

```
B_TAB     db    1Ah,2Bh,3Ch,4Dh,5Eh,6Fh,7Ah,8Bh
W_TAB     dw    1A2Bh,3C4Dh,5E6Fh,7A8Bh
B_TAB1    db    0Ah,8 dup(1)
W_TAB1    dw    8 dup(1)
W_TAB2    dw     11h,12h,13h,14h,15h,16h,17h,18h
```

3. На основе работы программы в таблице 1, в графы 2 и 3 зафиксировать значение операнда приемника на каждом шаге программы.

Таблица 1

Оператор	Операнд-приемник	
	до выполнения	После выполнения
1	2	3

4. Выполнить задание варианта (Приложение_2). На основе работы программы в таблице 1, в графы 2 и 3 зафиксировать значение операнда приемника на каждом шаге программы.

Теоретическая часть

Режимы адресации

Важной особенностью машинных команд является то, что они не могут манипулировать одновременно 2-мя операндами, находящимися в оперативной памяти (ОЗУ). Это означает, что в команде только 1 операнд может указывать на ячейку ОЗУ, другой операнд должен быть либо регистром, либо непосредственным значением.

По этой причине возможны следующие сочетания операндов в команде:

а) регистр - регистр;

б) регистр - память;

в) память – регистр;

г) регистр - непосредственный операнд;

д) память - непосредственный операнд.

Для команд характерно, что при наличии двух операндов первый из них является приемником, а второй – источником. Результат операции сохраняется по первому адресу, вот почему первый операнд никогда не может быть непосредственным операндом или, иначе говоря, константой.

Смещение, которое вычисляется операционным блоком для доступа к находящемуся в памяти операнду, называется исполнительным адресом операнда. Этот исполнительный адрес показывает, на каком расстоянии (в байтах) от начала сегмента располагается искомый операнд.

В зависимости от используемого режима адресации получение исполнительного адреса может заключаться только в извлечении его как составной части исполняемой команды, а могут потребоваться дополнительные операции сложения составной части команды с содержимым других регистров.

- Способы адресации приведены в табл.1.

Таблица 1

N п/п	Адресация	Основной формат оператора	Сегмент по умолчанию	Примечание
1	Регистровая заключается в указании в команде самого регистра в котором содержится значение операнда,	Регистр 1 или 2 байта		Наиболее быстрое выполнение
2	Непосредственная - заключается в указании в команде самого значения операнда, а не его адреса;	Данное 1 или 2 байта		Применяется в операциях с константами
3	Прямая - предполагает указание в команде непосредственно исполнительного адреса;	Исполнительный адрес 2 байта	DS	Применяется для однократного обращения к памяти
4	Косвенная регистровая - в команде указывается адрес регистра или ячейки памяти, в которых хранится адрес операнда или его составляющие;	[BX] [BP] [SI] [DI]	DS SS DS DS	Применяется при работе с одномерными массивами

5	Базовая	[BX] + сдвиг [BP] + сдвиг	DS SS	Применяется для обращения к элементу структуры, нач. адрес которой в BP или BX
6	Прямая с индексированием	Сдвиг[SI] Сдвиг [DI]	DS DS	Применяется при работе с одномерными массивами, сдвиг - нач. адрес массива
7	По базе с индексированием	Сдв.[BX][SI] Сдв.[BX][DI] Сдв.[BP][SI] Сдв.[BP][DI]	DS DS SS SS	Применяется при работе с двумерными массивами

Из семи режимов адресации самыми быстрыми являются регистровая и непосредственная адресации операндов, поскольку в этом случае операционный блок микропроцессора 8x86 извлекает их либо из регистров (при регистровой адресации), либо из конвейера команд (при непосредственной адресации). В других режимах адресация выполняется дольше, потому что интерфейс шины вначале должен вычислить адрес ячейки памяти, извлечь операнд и только после этого передать его операционному блоку.

Каждое описание режима адресации, приведенное в данном разделе, сопровождается примерами его применения. В большинстве случаев для этого используется команда MOV микропроцессора 8x86.

Регистровая и непосредственная адресация

При *регистровой адресации* микропроцессор 8x86 извлекает операнд из регистра (или загружает его в регистр). Например, команда

MOV AX,CX

копирует 16-битовое содержимое регистра счетчика CX в аккумулятор AX. Содержимое регистра CX не изменяется. В данном примере микропроцессор 8x86 использует регистровую адресацию для извлечения операнда-источника из регистра CX и загрузки его в регистр-приемник AX.

Непосредственная адресация позволяет Вам указывать 8- или 16-битовое значение константы в качестве операнда-источника. Эта константа содержится в команде (куда она помещается Ассемблером), а не в регистре или в ячейке памяти. Например, команда

MOV CX,500

загружает значение 500 в регистр CX, а команда

MOV CL,-30

загружает значение - 30 в регистр CL.

Чтобы избежать трудностей, помните, что допустимые значения для 8-битовых чисел со знаком ограничены диапазоном от -128 (80H) до 127 (7FH), а допустимые значения 16-битовых чисел со знаком — диапазоном от —32768 (8000H) до 32767 (7FFFH). Максимальные значения 8-битовых чисел без знака равны соответственно 255 (OFFH) и 65535 (OFFFFH).

Расширение знакового бита непосредственных значений

Ассемблер всегда расширяет знак при пересылке непосредственных значений в операнд-приемник. Это означает, что он дублирует старший значащий бит значения источника до тех пор, пока не будут заполнены все 8 или 16 битов операнда-приемника.

Например, операнд-источник нашего первого примера, десятичное число 500, может быть записано в виде 10-битового двоичного значения 0 111 110 100. Когда Ассемблер устанавливает,

что Вы требуете загрузить это значение в 16-битовый регистр CX, то он расширяет его до 16-битового, записав перед ним шесть копий "знакового" бита (со значением 0). Поэтому в регистр CX попадает двоичное значение 0 000 000 111 110 100. Во втором примере микропроцессор 8x86 загружает в регистр CL 8-битовое двоичное представление 11 100 010 десятичного числа –30.

Режимы адресации памяти

Доступ к ячейкам памяти обеспечивается взаимодействием операционного блока и интерфейса шины микропроцессора 8x86. Когда операционному блоку требуется прочитать или записать значение операнда, находящегося в памяти, он передает значение смещения адреса интерфейсу шины. Последний добавляет это смещение к содержимому регистра сегмента (предварительно дополненному четырьмя нулями) и тем самым получает 20-битовый физический адрес, который и используется для доступа к операнду.

Исполнительный адрес

Смещение, которое вычисляется операционным блоком для доступа к находящемуся в памяти операнду, называется *исполнительным адресом* операнда. Исполнительный адрес показывает, на каком расстоянии (в байтах) располагается операнд от начала сегмента, в котором он находится. Будучи 16-битовым числом без знака, исполнительный адрес позволяет получить доступ к операндам, находящимся выше начала сегмента на расстоянии до 65535 (или 64K) байтов.

Время, затрачиваемое операционным блоком на вычисление исполнительного адреса, является одним из основных компонентов общего времени исполнения команды. В зависимости от используемого режима адресации получение исполнительного адреса может заключаться всего лишь в извлечении его как составной части команды, но иногда могут потребоваться довольно долгие манипуляции, например сложение извлеченной из команды составляющей с регистром базы и с индексным регистром. Даже если время исполнения не является критичным для Вашей программы, стоит оценивать эти временные факторы в процессе чтения следующих ниже описаний режимов адресации.

Прямая адресация

При прямой адресации исполнительный адрес является составной частью команды (так же, как значения при непосредственной адресации). Микропроцессор 8x86 добавляет этот исполнительный адрес к сдвинутому содержимому регистра сегмента данных DS и получает 20-битовый физический адрес операнда.

Обычно прямая адресация применяется, если операндом служит метка. Например, команда

MOV AX,TABLE

загружает содержимое ячейки памяти TABLE в регистр AX. На рис. 1 показана схема исполнения этой команды. Обратите внимание на то, что против ожидания микропроцессор 8x86 заполняет данные в памяти в обратном порядке. Старший байт слова *следует* за младшим байтом, а не предшествует ему. Чтобы усвоить это, запомните, что *старшая часть (старшие биты)* данных *располагается в ячейках памяти со старшими адресами.*

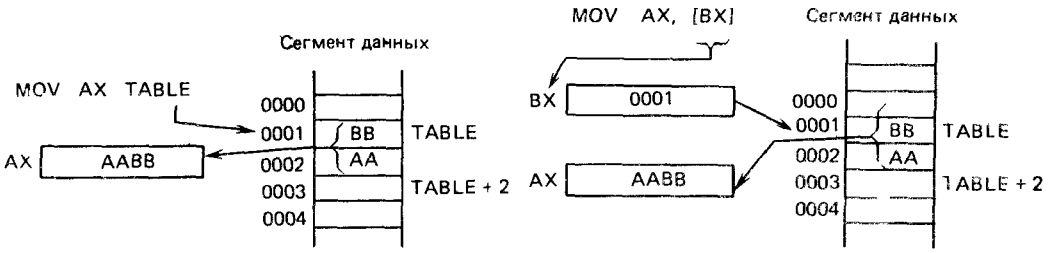

Рис. 1. Прямая адресация **Рис. 2. Косвенная регистровая адресация**

Косвенная регистровая адресация

При косвенной регистровой адресации исполнительный адрес операнда содержится в базовом регистре BX, регистре указателя базы BP или индексном регистре (SI или DI). Косвенные регистровые операнды надо заключать в квадратные скобки, чтобы отличить их от регистровых операндов. Например, команда

MOV AX,[BX]

загружает в регистр AX содержимое ячейки памяти, адресуемой значением регистра BX (рис. 2).

Как поместить смещение адреса в регистр BX? Один из методов состоит в применении команды LEA (Загрузить исполнительный адрес). Например, для загрузки слова из ячейки TABLE в регистр AX можно воспользоваться последовательностью команд

LEA BX, TABLE
MOV AX,[BX]

Эти две команды выполняют те же действия, что и одна команда

MOV AX,TABLE

с той лишь разницей, что в первом случае предыдущее содержимое регистра BX уничтожается. Если Вам нужен доступ лишь к одной ячейке памяти (в данном случае TABLE), то разумнее воспользоваться одной командой. Однако для доступа к *нескольким* ячейкам, начиная с данного базового адреса, гораздо лучше иметь исполнительный адрес в регистре. Почему? Потому что содержимым регистра можно манипулировать, не извлекая каждый раз новый адрес.

Адресация по базе

При адресации по базе Ассемблер вычисляет исполнительный адрес с помощью сложения значения сдвига с содержимым регистров BX или BP.

Регистр BX удобно использовать при доступе к структурированным записям данных, расположенным в разных областях памяти. В этом случае базовый адрес записи помещается в базовый регистр BX и доступ к ее отдельным элементам осуществляется по их сдвигу относительно базы. А для доступа к разным записям одной и той же структуры достаточно соответствующим образом изменить содержимое базового регистра.

Предположим, например, что требуется прочитать с диска учетные записи для ряда работников. При этом каждая запись содержит табельный номер работника, номер отдела, номер группы, возраст, тарифную ставку и т.д. Если номер отдела хранится в пятом и шестом бантах записи, а начальный адрес записи содержится в регистре BX, то команда

MOV AX,[BX]+4

загрузит в регистр AX номер отдела, в котором служит данный работник (рис. 3). (Сдвиг равен 4, а не 5, потому что первый байт записи имеет номер 0.)

Ассемблер позволяет указывать адресуемые по базе операнды тремя разными способами. Следующие команды эквивалентны:

```
MOV AX,[BP]+4     ;Это стандартная форма записи,
MOV AX,4[BP]      ; но сдвиг можно указать на первом месте
MOV AX,[BP+4]     ; или внутри скобок
```

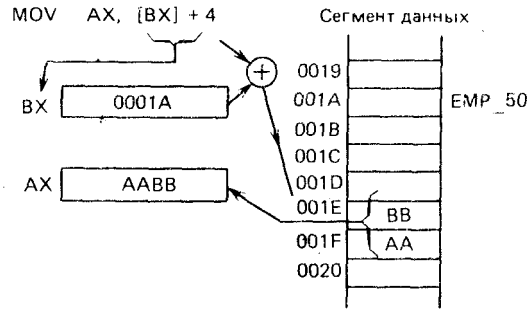

Рис. 3. Адресация по базе

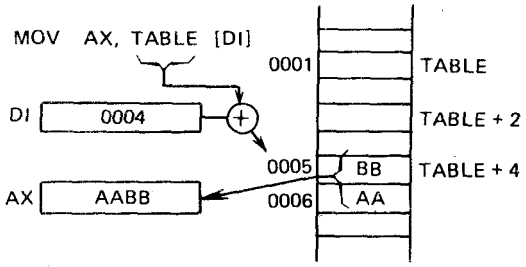

Рис. 4. Прямая адресация с индексированием

Прямая адресация с индексированием

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

Например, если B_TABLE — таблица байтов, то последовательность команд

```
MOV DI,2
MOV AL,B_TABLE[DI]
```

загрузит третий элемент таблицы в регистр AL.

В таблице слов соседние элементы отстоят друг от друга на два байта, поэтому при работе с ней надо удваивать *номер элемента* при вычислении значения индекса. Если TABLE — таблица слов, то для загрузки в регистр AX ее третьего элемента надо использовать последовательность команд

```
MOV DI,4
MOV AX,TABLE[DI]
```

(рис. 4).

Адресация по базе с индексированием

При адресации по базе с индексированием исполнительный адрес вычисляется как сумма значений базового регистра, индексного регистра и, возможно, сдвига.

Так как в этом режиме адресации складывается два отдельных смещения, то он удобен при адресации двумерных массивов, когда базовый регистр содержит начальный адрес массива, а значения сдвига и индексного регистра суть смещения по строке и столбцу.

Предположим, например, что Ваша ЭВМ следит за шестью предохранительными клапанами на химическом предприятии. Она считывает их состояния каждые полчаса и запоминает в ячейках памяти. За неделю эти считывания образуют массив, состоящий из 336 блоков (48 считываний в течение семи дней) по шесть элементов в каждом, а всего — 2016 значений.

Если начальный адрес массива загружен в регистр BX, сдвиг блока (номер считывания, умноженный на 12) - в регистре DI, а номер клапана задан в переменной VALVE, то команда

```
MOV AX ,VALVE[BX][DI ]
```

загрузит требуемое считывание состояния клапана в регистр AX. На рис. 5 изображен процесс извлечения результата третьего считывания (с номером 2) для клапана 4 из массива, у которого смещение в сегменте данных равно 100H.

Приведем несколько допустимых форматов операндов, адресуемых по базе с индексированием:

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

1. Что понимается под режимом адресации?
2. Что необходимо указать для описания метода адресации?
3. В чем заключается сущность регистрового режима адресации?
4. В чем заключается сущность косвенного регистрового режима адресации?
5. В чем заключается сущность прямого режима адресации?
6. В чем заключается сущность непосредственного режима адресации?
7. В чем заключается сущность базового режима адресации?
8. В чем заключается сущность индексного режима адресации?
9. В чем заключается сущность базового индексного режима адресации?

ПРИЛОЖЕНИЕ_1

Фрагмент программы, использующий разные виды адресации данных

```
;непосредственная (операнд-источник)
                mov   al,-3          ;расширение знака
                mov   ax,3
                mov   B_TAB,-3
                mov   W_TAB,-3
                mov   ax,2A1Bh
;регистровая
            mov   bl,al
            mov   bh,al
            sub   ax,bx
            sub   ax,ax
;прямая
            mov   ax,W_TAB
                  mov   ax,W_TAB+3
                  mov   ax,W_TAB+5
                  mov   al,byte ptr W_TAB+6
                  mov   al,B_TAB
                  mov   al,B_TAB+2
                  mov   ax,word ptr B_TAB
                  mov   es:W_TAB2+4,ax
```

! продолжение на следующей странице

```
;косвенная
                mov   bx,offset B_TAB
                mov   si,offset B_TAB+1
                mov   di,offset B_TAB+2
                mov   dl,[bx]
                mov   dl,[si]
                mov   dl,[di]
                mov   ax,[di]
                mov   bp,bx
                mov   al,[bp]                   ;какой сегмент?
                mov   al,ds:[bp]
                mov   al,es:[bx]
                mov   ax,cs:[bx]
; базовая
                mov   ax,[bx]+2         ;основная форма
                mov   ax,[bx]+4         ;проверьте допустимость других
                mov   ax,[bx+2]
                mov   ax,[4+bx]
                mov   ax,2+[bx]
                mov   ax,4+[bx]
                mov   al,[bx]+2
                mov   bp,bx                     ;другой базовый регистр
                mov   ax,[bp+2]                 ;откуда содержимое ax?
                mov   ax,ds:[bp]+2              ;попробуем переназначить
                                                ;сегментный регистр
                mov   ax,ss:[bx+2]
;индексная
                mov   si,2                      ;загрузка индекса
                mov   ah,B_TAB[si]              ;основная форма
                mov   al,[B_TAB+si]             ;проверьте другие
                mov   bh,[si+B_TAB]
                mov   bl,[si]+B_TAB
                mov   bx,es:W_TAB2[si]
                mov   di,4
                mov   bl,byte ptr es:W_TAB2[di]
                mov   bl,B_TAB[si]
;базовая индексная
                mov   bx,offset B_TAB           ;загрузка базы
                mov   al,3[bx][si]              ;основная форма
                mov   ah,[bx+3][si]
                mov   al,[bx][si+2]
                mov   ah,[bx+si+2]
                mov   bp,bx
                mov   ah,3[bp][si]              ;из какого сегмента?
                mov   ax,ds:3[bp][si]
                mov   ax,word ptr ds:2[bp][si]
```

ПРИЛОЖЕНИЕ 2

1. Задать одномерный массив, состоящий из 10 элементов.
2. Заполнить массив числовыми константами. Размер элементов массива для четных номеров варианта - байт, для нечетных номеров варианта – слово.
3. Разместить элементы массива в регистры общего назначения, используя различные способы адресации, по следующей схеме:

   - в AX – элемент массива, номер которого соответствует первой цифре Вашего дня рождения,
   - в BX – элемент массива, номер которого соответствует второй цифре Вашего дня рождения,
   - в CX – элемент массива, номер которого соответствует первой цифре Вашего месяца рождения,
   - в DX – элемент массива, номер которого соответствует второй цифре Вашего месяца рождения,
   - в SI – элемент массива, номер которого соответствует первой цифре Вашего года рождения,
   - в DI – элемент массива, номер которого соответствует второй цифре Вашего года рождения,
   - в BP – элемент массива, номер которого соответствует третьей цифре Вашего года рождения, в SP – элемент массива, номер которого соответствует четвертой цифре Вашего года рождения.
4. Назначить переменной fio Вашу фамилию, имя, отчество. Определить физические адреса заглавных букв. Разместить в регистр AL среднюю букву ФИО (значение округлить до целого).
5. По адресу равному дню и месяцу Вашего рождения (например, 23 февраля – 2302) занести год Вашего рождения, представив его как шестнадцатеричное число.
6. Определить переменную const, присвоив ей день и месяц Вашего рождения в формате описанном выше. Разместить это значение в регистре CX.
7. Поместить в переменную name уменьшительно-ласкательную форму Вашего имени. Определить адрес name.
8. В памяти сразу после name разместить символ, код которого в ASCII-кодах определить, как Ваш номер по порядку плюс 14. А затем через пробел дату Вашего рождения в формате 23021999.