Лабораторная работа № 1

Технология подготовки и выполнения программ на языке ассемблер TASM.

Цель работы

Практическое овладение навыками разработки программного кода на языке ассемблер TASM. Изучение основных возможностей 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 необходимо сохранить с именем PROG_2.asm.
2. Создать загрузочный модуль, загрузить его в отладчик и выполнить программу в пошаговом режиме.
3. Изучить возможности отладчика TD, режимы просмотра регистров, файлов, памяти.
4. Провести проверку правильности работы программы используя варианты тестовых данных.

Содержание отчета

1. Цель работы.
2. Задание с пошаговой реализацией.
3. Текст программ.
4. Окна отладчика **Dump** для **PRG_1.exe** и **PRG_2.exe** с пояснениями.
5. Вывод.

Теоретическая часть

1. Типы данных

При программировании на языке ассемблера используются данные следующих типов:

1. Непосредственные данные, представляющие собой числовые или символьные значения, являющиеся частью команды. Непосредственные данные формируются программистом в процессе написания программы для конкретной команды ассемблера.

2. Данные простого типа, описываемые с помощью набора директив резервирования памяти, позволяющих выполнить самые элементарные операции по размещению и инициализации числовой и символьной информации. При обработке этих директив ассемблер сохраняет в своей таблице символов информацию о местоположении данных (значения сегментной составляющей адреса и смещения) и типе данных, то есть единицах памяти, выделяемых для размещения данных в соответствии с директивой резервирования и инициализации данных.

3. Данные сложного типа, которые были введены в язык ассемблера с целью облегчения разработки программ. Сложные типы данных (массивы, структуры, записи, объединения) строятся на основе базовых типов. Введение сложных типов данных позволяет несколько сгладить различия между языками высокого уровня и ассемблером.

Данные простого типа:

- **Байт** - восемь последовательно расположенных битов, пронумерованных от 0 до 7, при этом бит 0 является самым младшим значащим битом;
- **Слово** - последовательность из двух байт, имеющих последовательные адреса. Размер слова - 16 бит; биты в слове нумеруются от 0 до 15. Байт, содержащий нулевой бит, называется младшим байтом, а байт, содержащий 15-й бит - старшим байтом. Микропроцессоры Intel имеют важную особенность - младший байт всегда хранится по меньшему адресу. Адресом слова считается адрес его младшего байта. Адрес старшего байта может быть использован для доступа к старшей половине слова.
- **Двойное слово** - последовательность из четырех байт (32 бита), расположенных по последовательным адресам. Нумерация этих бит производится от 0 до 31. Слово, содержащее нулевой бит, называется младшим словом, а слово, содержащее 31-й бит, - старшим словом. Младшее слово хранится по меньшему адресу. Адресом двойного слова считается адрес его младшего слова. Адрес старшего слова может быть использован для доступа к старшей половине двойного слова.
- **Учетверенное слово** - последовательность из восьми байт (64 бита), расположенных по последовательным адресам. Нумерация бит производится от 0 до 63. Двойное слово, содержащее нулевой бит, называется младшим двойным словом, а двойное слово, содержащее 63-й бит, - старшим двойным словом. Младшее двойное слово хранится по меньшему адресу. Адресом учетверенного слова считается адрес его младшего двойного слова. Адрес старшего двойного слова может быть использован для доступа к старшей половине учетверенного слова.

Очень важно уяснить себе порядок размещения данных в памяти. Он напрямую связан с логикой работы микропроцессора с данными**.**

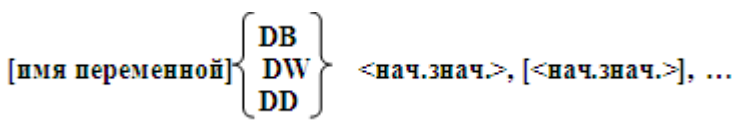
Микропроцессоры Intel требуют следования данных в памяти по принципу: младший байт по младшему адресу.

Простые типы данных

Для описания простых типов данных в программе используются специальные директивы резервирования и инициализации данных, которые по сути являются указаниями транслятору на выделение определенного объема памяти. Если проводить аналогию с языками высокого уровня, то директивы резервирования и инициализации данных являются определениями переменных. Машинного эквивалента этим директивам нет; При работе с этими директивами и с дампами памяти не забывайте о принципе <младший байт по младшему адресу>, в соответствии с которым при сохранении данных в памяти младшая часть значения всегда записывается перед старшей частью.

Директивы определения данных

В общем случае все директивы объявления данных имеют такой синтаксис:



DB (Define Byte) определить байт.

Директивой db можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона:

- для чисел со знаком -128...+127;
- для чисел без знака 0...255;
- 8-битовое относительное выражение, использующее операции HIGH и LOW;
- символьную строку из одного или более символов. Строка заключается в кавычки. В этом случае определяется столько байт, сколько символов в строке.

DW (Define Word) - определить слово.

Директивой dw можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона:

для чисел со знаком -32768...32767;

для чисел без знака 0...65535;

- выражение, занимающее 16 или менее бит, в качестве которого может выступать смещение в 16-битовом сегменте или адрес сегмента;
- 1- или 2-байтовая строка, заключенная в кавычки.

DD (Define Doubleword) (определить двойное слово)

Директивой dd можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона:

для 8086:

для чисел со знаком -32768...+32767;

для чисел без знака 0...65533;

для i386 и выше:

для чисел со знаком -2147483648...+2147483647;

для чисел без знака 0...4 294 967 295;

- относительное или адресное выражение, состоящее из 16-битового адреса сегмента и 16-битового смещения;
- строку длиной до 4 символов, заключенную в кавычки.

реже

DQ (Define Quarter word) - определить учетверенное слово

Директивой DQ можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона:

для МП i8086:

для чисел со знаком -32 768...+32 767;

для чисел без знака 0...65 535;

для МП i386 и выше:

для чисел со знаком -2 147 483 648... +2 147 483 647;

для чисел без знака 0...4 294 967 295;

- относительное или адресное выражение, состоящее из 32 или менее бит (для i80386) или 16 или менее бит (для младших моделей микропроцессоров Intel);

- константу со знаком из диапазона -2**63...2**63 - 1;
- константу без знака из диапазона 0...2**64 - 1;
- строку длиной до 8 байт, заключенную в кавычки.

DF (Define Far word) - определить указатель дальнего слова

DP (Define Pointer) /I определить указатель 48 бит

Директивами DF и DP можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона:
  для 8086:
    для чисел со знаком -32 768...+32 767;
    для чисел без знака 0...65 535;
  для i386 и выше:
    для чисел со знаком -2 147 483 648...+2 147 483 647;
    для чисел без знака 0...4 294 967 295;
- относительное или адресное выражение, состоящее из 32 или менее бит (для i80386) или 16 или менее бит (для младших моделей микропроцессоров Intel);
- адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;
- константу со знаком из диапазона -2**47...2**47 - 1;
- константу без знака из диапазона 0...2**48 - 1;
- строку длиной до 6 байт, заключенную в кавычки.

DT (Define Ten Bytes) - определить 10 байт

Директивой DT можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона:
  для 18086:
    для чисел со знаком -32 768...+32 767;
    для чисел без знака 0...65 535;
  для i386 и выше:
    для чисел со знаком -2 147 483 648... +2 147 483 647;
    для чисел без знака 0...4 294 967 295;
- относительное или адресное выражение, состоящее из 32 или менее бит (для i80386) или 16 или менее бит (для младших моделей);
- адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;
- константу со знаком из диапазона -2**79...2**79 - 1;
- константу без знака из диапазона 0...2**80 - 1;

- строку длиной до 10 байт, заключенную в кавычки;
- упакованную десятичную константу в диапазоне 0...99 999 999 999 999 999 999.

Примерный вид сегмента данных представлен ниже:

```
. data                          ;начало сегмента с именем DATA
VAR_1  DB   11000110b           ;определить переменную VAR_1 размером байт
                                ;с начальным значением 11000110b
VAR_2  DW  9FFEh;               ;определить переменную VAR_2 размером слово
                                ;с начальным значением 9FFEh
VAR_3  DW   ?                   ;определить переменную VAR_3 размером
                                ;слово не задавая ее начального значения
STRING DB 'ASSEMBLER'           ;определить строку символов STRING каждый
                                ;символ которой имеет размер байт с
                                начальным ;значением ASSEMBLER
ARRAY_1 DB 0, -3, 28, 46, 39    ;определить массив с именем MAS_1 состоящий
                                из ;пяти числовых элементов размером байт
```

Как видно из приведенного примера для задания начальных значений могут использоваться числовые константы, представленные в двоичной (Binary - **b**), шестнадцатиричной (Hexal - **h**), восьмиричной (Octal – **q**) или десятичной (Decimal – **d**) системах счисления. При этом за младшей цифрой числа должен следовать однобуквенный дескриптор системы счисления[1].

Следует знать: Если шестнадцатеричная константа начинается с буквы, то она обязательно должна быть дополнена слева незначащим нулем.

Множественная инициализация

Для резервирования памяти под массивы используется директива DUP

[имя] директива dup_выражение [,dup_выражение]

Примерный вид сегмента данных представлен ниже:

```
AREA      DW      20 dup(?)       ;резервируется  память объемом 20 слов
STRING_2  DB      10 dup('*')     ;строка заполняется кодом символа '*'
ARRAY_2   DW      256 dup(128) ;массив из 256 слов инициализируется  числом 128
          DB      3 dup(5 dup(8)); резервирует область данных, заполненную 15
                  восьмерками
```

2. Работа с отладчиком Turbo Debugger (TD). Окно Dump

В окне **Dump** (Дамп) выводится в непосредственном виде содержимое области памяти. (Это окно эквивалентно области дампа окна CPU.) Вы можете выводить данные в виде символов, шестнадцатеричных байтов, слов, двойных слов, или в любом формате с плавающей точкой. Данное окно можно использовать для просмотра в непосредственном виде некоторых данных, когда вам не требуются остальные части окна CPU. В локальном меню этого окна

[1] В обозначении десятичных чисел использование дескриптора не обязательно.

имеются команды, позволяющие вам модифицировать выводимые данные, изменять формат их вывода, а также манипулировать блоками данных.

Локальное меню окна **Dump** (**Alt+F10** или **Ctrl+F10**) содержит пункты:

- **Goto** (Переход) – выводит на экран данные по новому адресу;
- **Change** (Изменение) – изменяет байты данных по адресу курсора;
- **Display As** (Режим вывода) – задает режим вывода дампа: байтами, словами и т.п.

Для того что бы просмотреть область данных, следует выполнить следующие действия:

- открыть окно **Dump** или сделать активной одноименную панель окна **CPU**;
- с помощью пункта **GoTo** локального меню окна (или панели) выполнить позиционирование по адресу **DS:0**. Перед позиционированием необходимо обеспечить правильное значение регистра **DS**, как правило, это означает выполнение первых двух строк программы по загрузке регистра **DS** адресом сегмента данных;
- с помощью пункта **DisplayAs** локального меню окна (или панели) обеспечить просмотр данных в формате слов. Первое слово сегмента данных со смещением **0** – это переменная **massage**, далее следует 36 слов – числа массива, и далее – остальные переменные.

Контрольные вопросы

1. Как размещаются в памяти байты?
2. Как размещаются в памяти слова, двойные слова?
3. С какого байта адресуются слова, двойные слова?
4. Как размещается в памяти полный адрес переменной?

3. Задание на выполнение работы

1. Запустить программу **PRG_1.exe**, созданный ранее. Используя окно **Dump,** просмотреть как хранятся данные, описанные в программе.

Prg_1.asm. Пример использования директив резервирования и инициализации данных

```
.model small
.stack 100h
.data
message db 'Запустите эту программу в отладчике','$'
perem_1    db    0ffh
perem_2    dw    3a7fh
perem_3    dd    0f54d567ah
mas        db    10 dup (' ')
pole_1     db    5 dup (?)
adr        dw    perem_3
adr_full   dd    perem_3
numbers    db    11, 34, 56, 23
fin        db    'Конец сегмента данных программы $'
.code
start:
      mov   ax,@data
      mov   ds,ax
      mov   ah,09h
      mov   dx,offset message
      int   21h
      mov   ah, 7h
      int   21h
      mov   ax,4c00h
      int   21h
end   start
```

2. Используя текстовый редактор, отредактировать исходный модуль программы **PRG_1.asm**, заменив данные на приведенные в таблице вариантов. Проанализировать результат, используя окно **Dump.**

Варианты заданий:

1.	Pa	db	73H	10.	Pa	db	5BH
	Pb	dw	0AE21H		Pb	dw	0BA21H
	Pc	dd	38EC76A4H		Pc	dd	0FA4A32BCH
	Mas	db	10 dup(1),2,3		Mas	db	4,5,6,5 dup(0)
	Pole	db	5 dup(?)		Pole	db	6 dup(?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
2.	Pa	db	67H	11.	Pa	db	4AH
	Pb	dw	4AEFH		Pb	dw	0DEFCH
	Pc	dd	12DC4567H		Pc	dd	81ADFF06H
	Mas	db	5,6,7,8		Mas	db	5 dup(1),2,3,3 dup(4)
	Pole	db	6 dup(0)		Pole	db	6 dup(“ “)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
3.	Pa	db	4DH	12.	Pa	db	7FH
	Pb	dw	0ED56H		Pb	dw	0ACDEH
	Pc	dd	32AF8DD7H		Pc	dd	10B0A488H
	Mas	db	4,3,5, 4 dup(0)		Mas	db	3 dup(0),1,2,3, 4 dup(0)
	Pole	db	6 dup(?)		Pole	db	5 dup(32)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
4.	Pa	db	5DH	13.	Pa	db	0BCH
	Pb	dw	0A1A3H		Pb	dw	903FH
	Pc	dd	3 dup(4),5,6		Pc	dd	6CAA3E41H
	Mas	db	4,3,5, 4 dup(0)		Mas	db	1,2,3, 4 dup(4)
	Pole	db	5 dup(?)		Pole	db	5 dup(?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
5.	Pa	db	62h	14.	Pa	db	0FBH
	Pb	dw	7ED1H		Pb	dw	54ADH
	Pc	dd	0EE45DA31H		Pc	dd	0E04365FAH
	Mas	db	1,2, 6 dup(3),0		Mas	db	3 dup(0), 4 dup(1),2,3
	Pole	db	5 dup(0)		Pole	db	5 dup(?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
6.	Pa	db	0FFH	15.	Pa	db	11H
	Pb	dw	4ADEH		Pb	dw	4D2DH
	Pc	dd	0C23891F5H		Pc	dd	98ADF156H
	Mas	db	4 dup(0),1,2,3		Mas	db	5 dup(0),1,2,3
	Pole	db	3 dup(‘ ‘)		Pole	db	3 dup(?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
7.	Pa	db	0AEH	16.	Pa	db	10H
	Pb	dw	63BCH		Pb	dw	1A2DH
	Pc	dd	63BCDEF3H		Pc	dd	55AEF2C8H
	Mas	db	9,8,3 dup(0)		Mas	db	1,2,3,4,5,6
	Pole	db	5 dup(“ “)		Pole	db	5 dup(0)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
8.	Pa	db	5DH	17.	Pa	db	0BCH
	Pb	dw	0A1A3H		Pb	dw	903FH
	Pc	dd	3 dup(4),5,6		Pc	dd	6CFA3E41H
	Mas	db	4,3,5, 4 dup(0)		Mas	db	1,2,3, 4 dup(4)
	Pole	db	5 dup(?)		Pole	db	7 dup(?)

	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
9.	Pa	db	62h	18.	Pa	db	0FBH
	Pb	dw	7ED1H		Pb	dw	54ADH
	Pc	dd	0EE45DA31H		Pc	dd	0E04365FAH
	Mas	db	1,2, 6 dup(3),0		Mas	db	3 dup(0), 4 dup(1),2,3
	Pole	db	5 dup(0)		Pole	db	5 dup(?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc