МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Отчёт о практике

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Толстова Роберта Сергеевича

Проверено:

Старший преподаватель _____ Е. М. Черноусова

Саратов 2024

СОДЕРЖАНИЕ

1 Исходный код программ

1.1 Задание 1

Вариант 3: Массив заполнить натуральными числами от 1 до 10 и организовать вывод массива на экран в виде таблицы 2x5 с фиксированной шириной столбцов.

```
.model small
.stack 100h

.data
    ; Фамилия и номер группы
    info db "Tolstov Robert 251$"

    ; Массив чисел от 1 до 10
    numbers db 2, 4, 6, 8, 10, 1, 3, 5, 7, 9

    ; Формат вывода (для удобства) -- отступ для чисел
    space db "  $"

.code
main:
    ; Инициализация сегмента данных
    mov ax, @data
    mov ds, ax

    ; Выводим фамилию и номер группы
    mov ah, 09h
    lea dx, info
    int 21h

    ; Переход на новую строку
    mov ah, 02h
    mov dl, 0Ah
    int 21h
    mov dl, 0Dh
    int 21h

    ; Выводим массив чисел 2x5
    call PrintArray

    ; Завершаем программу
    mov ah, 4Ch
```

```
    int 21h

; Процедура для вывода массива чисел в виде таблицы 2x5
PrintArray proc
    ; Первый ряд чисел (числа с индексами 0-4)
    lea si, numbers
    call PrintRow

    ; Переход на новую строку
    mov ah, 02h
    mov dl, 0Ah
    int 21h
    mov dl, 0Dh
    int 21h

    ; Второй ряд чисел (числа с индексами 5-9)
    lea si, numbers+5
    call PrintRow

    ret
PrintArray endp

; Процедура для вывода одного ряда чисел
PrintRow proc
    ; Цикл вывода 5 чисел с правильными отступами
    mov cx, 5           ; Цикл на 5 элементов
PrintLoop:
    ; Загружаем число из массива
    mov al, [si]
    ; Выводим число с отступом
    call PrintNum
    ; Переход к следующему числу
    inc si
    ; Печатаем пробел между числами
    mov ah, 02h
    mov dl, 09h ; табуляция
    int 21h
    loop PrintLoop
    ret
PrintRow endp

; Процедура для вывода числа в виде строки
PrintNum proc
    mov bl, 10
```

```
    ; Если число меньше 10, просто выводим его как символ
    cmp al, bl
    jl PrintSingleDigit

    xor ah, ah ; обнуляем ah

    ; Если число двухзначное (например 10), выводим его как два
символа
    ; Разделяем на десятки и единицы
    div bl              ; AX / 10 -> AL = остаток (единицы), AH =
десятки
    mov bh, ah

    ; Выводим десятки
    add al, '0'         ; Преобразуем в символ
    mov dl, al
    mov ah, 02h
    int 21h          ; Выводим десятки

    ;xor al, al

    ; Выводим единицы
    add bh, '0'         ; Преобразуем в символ
    mov dl, bh
    mov ah, 02h
    int 21h          ; Выводим единицы

    ret

PrintSingleDigit:
    mov bl, al
    xor al, al
    cmp cx, 5
    jz PrintSingleDigitWithoutSpace
    mov ah, 02h
    mov dl, ' '
    int 21h

PrintSingleDigitWithoutSpace:
    ; Если число одноцифровое (менее 10), выводим его как символ
    add bl, '0'         ; Преобразуем в символ
    mov dl, bl
    mov ah, 02h
    int 21h
```

```
    ret
PrintNum endp

end main
```

Код первой программы

2 Контрольные вопросы

2.1 Какой командой можно выделить в памяти место под одномерный массив байтов array размерностью 20?

Для выделения в памяти места под одномерный массив байтов размерностью 20 в TASM используется директива DB (Define Byte). Эта директива позволяет определить массив байтов и задать его размер.

В данном примере выделим место под массив из 20 элементов:

```
array db 20 dup(0)
```

2.2 Опишите команды умножения на байт и на слово.

В TASM для выполнения операций умножения используются команды `MUL` и `IMUL`. Эти команды позволяют умножать значения, хранящиеся в регистрах, на операнды, которые могут быть как в памяти, так и в регистрах.

2.2.1 Умножение без знака (MUL)

Синтаксис:

```
MUL <операнд>
```

Если операнд – это байт (например, содержимое регистра BL), то команда умножает его на значение в регистре AL, а результат (двойное слово) помещается в регистр AX.

Например,

```
MOV AL, 150
MOV BL, 250
MUL BL ; AX = AL * BL
```

Если же операнд – это слово (например, содержимое регистра BX), то команда умножает его на значение в регистре AX, а результат помещается в пару регистров DX:AX.

Например,

```
MOV AX, 10000
MOV BX, 5000
MUL BX ; DX:AX = AX * BX
```

2.2.2 Умножение со знаком (IMUL)

Синтаксис:

```
IMUL <операнд>
```

Если операнд – это байт (например, содержимое регистра BL), то команда умножает его на значение в регистре AL, а результат (двойное слово) помещается в регистр AX.

Например,

```
MUL BL ; AX = AL * BL
```

Если же операнд – это слово, то команда умножает его на значение в регистре AX, а результат помещается в пару регистров DX:AX.

Например,

```
IMUL BX ; DX:AX = AX * BX (со знаком)
```

2.3 Опишите команды умножения на байт и на слово.

В TASM для выполнения операций умножения используются команды `MUL` и `IMUL`. Эти команды позволяют умножать значения, хранящиеся в регистрах, на операнды, которые могут быть как в памяти, так и в регистрах.

2.4 Пусть имеется массив: array DW 50 DUP(?). Для доступа к отдельным элементам массива используется адресное выражение array[SI]. Как называется этот способ адресации и как с его помощью будет вычисляться адрес элементов массива?

Способ адресации, используемый для доступа к элементам массива array DW 50 DUP(?) с помощью выражения array[SI], называется индексной адресацией.

2.4.1 Вычисление адресов элементов массива

При использовании индексной адресации адреса элементов массива вычисляются следующим образом:

1. Базовый адрес: Адрес начала массива хранится в сегменте данных и может быть представлен как array.
2. Индекс: Значение в регистре SI указывает на номер элемента, который вы хотите получить.
3. Размер элемента: Поскольку массив объявлен как DW (word), каждый элемент занимает 2 байта.

Формула для вычисления адреса A элемента массива будет выглядеть так:

$$A = \text{base\_address} + (S_I \cdot 2) \tag{1}$$

где:
- base_address — адрес начала массива,
- S_I — индекс элемента,
- 2 — размер одного элемента в байтах (для слова).

2.5 Каким образом осуществляется перебор элементов некоторого массива A с помощью адресного выражения A[SI], если массив состоит из байтов, слов или двойных слов?

Перебор элементов массива A с помощью адресного выражения A[SI] осуществляется следующим образом:

1. **Определение массива**:
   - Массив может состоять из байтов, слов или двойных слов. В зависимости от типа данных, размер каждого элемента будет различным:
     - **Байты**: 1 байт (8 бит)
     - **Слова**: 2 байта (16 бит)
     - **Двойные слова**: 4 байта (32 бита)

2. **Использование SI (Segment Index)**:
   - Регистры в ассемблере, такие как SI (Source Index), используются для указания на текущий элемент массива. При этом адресное выражение A[SI] позволяет получить доступ к элементам массива по индексу SI.

3. **Перебор элементов**:
   - Чтобы перебрать массив, необходимо использовать цикл, который будет изменять значение SI для доступа к каждому элементу. Например, для массива байтов это может выглядеть так:

```
mov SI, 0           ; Начинаем с первого элемента
mov CX, размер_массива ; Устанавливаем количество элементов для перебора
loop_start:
    mov AL, [A + SI] ; Загружаем элемент массива в регистр AL
    ; Здесь можно выполнять операции с элементом AL
    inc SI           ; Переходим к следующему элементу
    loop loop_start  ; Повторяем цикл, пока CX не станет 0
```

4. **Учет размера элемента**:
   - При работе с массивами слов или двойных слов необходимо учитывать размер элемента при инкрементировании SI:
     - Для слов: `inc SI` следует заменить на `add SI, 2`
     - Для двойных слов: `inc SI` следует заменить на `add SI, 4`

2.6 Для некоторого массива A каким будет результат выполнения команды mov DI, A и команды mov DI, offset A?

Команды `mov DI, A` и `mov DI, offset A` имеют разные результаты в зависимости от контекста.

1. **Команда `mov DI, A`**:
   - Эта команда загружает в регистр DI значение, которое находится по адресу A. Если A является меткой массива, то команда переместит значение элемента массива в регистр DI. Например, если A указывает на байт, то DI будет содержать значение этого байта.

2. **Команда `mov DI, offset A`**:
   - Эта команда загружает в регистр DI смещение (offset) метки A относительно начала сегмента. Смещение указывает на адрес, по которому начинается массив A в памяти. В отличие от первой команды, здесь не загружается значение элемента массива, а именно адрес.

Таким образом:
- `mov DI, A` загружает значение элемента массива (например, первый элемент), тогда как `mov DI, offset A` загружает адрес начала массива.