**Федеральное государственное образовательное учреждение   
высшего образования**

**«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»**

**Институт приборостроения, автоматизации и информационных технологий**

**Кафедра информационных систем и цифровых технологий**

Отчет к лабораторной работе № 4

«Обработка массивов. Числа Фибоначчи»

Выполнила:

Студентка группы 21ПГ

Банных Мария Алексеевна

Приняла:

доцент кафедры   
информационных систем   
и цифровых технологий  
Конюхова Оксана Владимировна  
  
доцент кафедры   
информационных систем   
и цифровых технологий  
Амелина Ольга Викторовна

Орёл 2022 год

Ответы на контрольные вопросы:

1. *Массивы и их представление в памяти компьютера.*

Как структура представления, массив является упорядоченным множеством элементов определенного типа. Упорядоченность массива определяется набором целых чисел, называемых индексами, которые связываются с каждым элементом массива и однозначно конкретизируют его расположение среди других элементом массива. Локализация конкретного элемента массива - ключевая задача при разработке любых алгоритмов, работающих с массивами.

Наиболее просто представляются одномерные массивы. Соответствующая им структура хранения — это вектор. Она однозначна и есть не что иное, как просто последовательное расположение элементов в памяти. Чтобы локализовать нужный элемент одномерного массива, достаточно знать его индекс. Так как ассемблер не имеет средств для работы с массивом как структурой данных, то для доступа к элементу массива необходимо вычислить его адрес.

Представление двумерных массивов немного сложнее. Здесь мы имеем случай, когда структуры хранения и представления различны. О структуре представления говорить излишне — это матрица. Структура хранения остается прежней — вектор. Но теперь его нельзя без специальных оговорок интерпретировать однозначно. Все зависит от того, как решил разработчик программы «вытянуть» массив — по строкам или по столбцам. Наиболее естествен порядок расположения элементов массива — по строкам. При этом наиболее быстро изменяется последний элемент индекса.

1. *Режимы адресации данных, которые могут применяться для доступа к элементам массива. Приведите примеры.*

1) Регистровый относительный. Является обобщением методов адресации, обеспечивающих вычисление эффективного адреса (EA) операнда в памяти в виде суммы базового значения адреса и «смещения» disp, указываемого в команде.

Относительную адресацию широко применяют как для адресации памяти, представленной в виде блоков (например, сегментов), так и для адресации специальных структур данных: массивов, записей и др. В зависимости от способа использования адресуемого в команде регистра различают базовый и индексный режимы адресации.

EA={BP\BX\SI\DI}+{Смещение disp}

2) Индексный. Применяется для обработки упорядоченных массивов данных; при этом каждый элемент массива определяется собственным номером. Тогда базовый адрес массива задаётся смещением disp, указываемым в команде, а значение индекса (номер элемента массива) определяется содержимым индексного регистра.

Индексная адресация удобна, если необходимо записать или считать список данных из последовательных ячеек памяти не подряд, а с некоторым шагом, указанным в индексе.

Пример: d\_s segment

mas db 3,5,1,8,9,’$’

d\_s ends

c\_s segment

assume ds:d\_s, cs:c\_s

begin:

…

mov si,0 ;в si-номер элемента массива

m1: mov ah, mas[si] ;mas- смещение

;в ah – значение элемента массива mas с

;номером в si

add si,1

jmp m1

…

c\_s ends

end begin

3) Базово-индексный. Используется для доступа к элементам массива, адресуемого указателем. Базовый адрес массива задаётся указателем базы (базовым регистром), а номер элемента массива – содержимым индексного регистра.

Пример: mov ax, bx[si]

Если в bx содержится 100, а в si находится 52, то по адресу (смещению) 152 в сегменте данных находится искомое данное.

4) Относительный базовый индексный. Используется для адресации элементов в указываемом массиве записей. Базовый адрес массива задаётся указателем базы, номер записи (т.е., элемента массива) определяется содержимым индексного регистра, а смещение в команде указывает расстояние до записи.

Пример:

d\_s segment ;опишем массив из 5 сотрудников со значениями по умолчанию

mas\_sotr worker 5 dup (<>)

d\_s ends

c\_s segment

assume ds:d\_s, cs:c\_s

begin: ;в bx – адрес начала массива сотрудников

lea bx, mas\_sotr ;в si – смещение второй записи

mov si, (type worker)\*2 ; в ax – стаж второго сотрудника

mov ax,[bx][si].standing

c\_s ends

end begin

1. *Способы описания массивов в сегменте данных.*

1) Перечислением элементов массива в поле операндов одной из директив описания данных. При перечислении элементы разделяются запятыми. К примеру, массив из 5 элементов. Размер каждого элемента 4 байта:

mas dd 1,2,3,4,5

2) Используя оператор повторения dup. К примеру, массив из 5 нулевых элементов. Размер каждого элемента 2 байта:

mas dw 5 dup (0)

Такой способ определения используется для резервирования памяти с целью размещения и инициализации элементов массива.

3) Используя директивы label и rept. Пара этих директив может облегчить описание больших массивов в памяти и повысить наглядность такого описания. Директива rept относится к макросредствам языка ассемблера и вызывает повторение указанное число раз строк, заключенных между директивой и строкой endm. К примеру, определим массив байт в области памяти, обозначенной идентификатором mas\_b. В данном случае директива label определяет символическое имя mas\_b, аналогично тому, как это делают директивы резервирования и инициализации памяти. Достоинство директивы label в том, что она не резервирует память, а лишь определяет характеристики объекта. В данном случае объект — это ячейка памяти. Используя несколько директив label, записанных одна за другой, можно присвоить одной и той же области памяти разные имена и разный тип, что и сделано в следующем фрагменте:

...

n=0

...

mas\_b label byte

mas\_w label word

rept 4

dw 0f1f0h

endm

В результате в памяти будет создана последовательность из четырех слов f1f0. Эту последовательность можно трактовать как массив байт или слов в зависимости от того, какое имя области мы будем использовать в программе — mas\_b или mas\_w.

4) Использование цикла для инициализации значениями области памяти, которую можно будет впоследствии трактовать как массив.

1. *Особенности обработки двумерных массивов в ассемблерных программах. Вычисление смещения элемента двумерного массива относительно начала сегмента данных.*

Двумерный массив нужно моделировать. На описании самих данных это почти никак не отражается — память под массив выделяется с помощью директив резервирования и инициализации памяти.

Непосредственно моделирование обработки массива производится в сегменте кода, где программист, описывая алгоритм обработки ассемблеру, определяет, что некоторую область памяти необходимо трактовать как двухмерный массив.

При обработке двумерных массивов удобно использовать базовую индексную адресацию со смещением.

Если последовательность однотипных элементов в памяти трактуется как двухмерный массив, расположенный по строкам, то адрес элемента (i, j) вычисляется по формуле

(база + количество\_элементов\_в\_строке \* размер\_элемента \* i+j)

Здесь i = 0...n–1 указывает номер строки, а j = 0...m–1 указывает номер столбца.

1. *Какие режимы адресации данных можно использовать для доступа к элементам двумерного массива? Приведите примеры.*

Организовать адресацию двухмерного массива логично, используя рассмотренную нами ранее базово-индексную адресацию. При этом возможны два основных варианта выбора компонентов для формирования эффективного адреса:

• сочетание прямого адреса, как базового компонента адреса, и двух индексных регистров для хранения индексов:

mov ax,mas[ebx][esi]

• сочетание двух индексных регистров, один из которых является и базовым и индексным одновременно, а другой — только индексным:

mov ax,[ebx][esi]

Практическая часть:

.model small

data segment

mass dw 18 dup (0, 1)

maxc dw 0

minc dw 1000

data ends

kod segment

assume DS: data, CS: kod

begin:

mov ax, data

mov ds, ax

xor ax, ax

mov cx, 16 ;16 повторений (т.к. 0 и 1 уже есть в массиве)

povtor:

mov ax, mass[si+2] ;в регистр ax поместить значение следующего элемента массива

add ax, mass[si] ;сложить следующий элемент с текущим

mov mass[si+4], ax ;в элемент после следующего поместить значение регистра ax

inc si

inc si ;увеличили индекс массива на 2

LOOP povtor

; 0 1 1 2 3 5

; 8 D 15 22 37 59

; 90 E9 179 262 3DB 63D

xor si, si

mov bx, minc

mov cx, 6 ;6 повторений, т.к. в строке 6 элементов массива

minimymi:

inc si

inc si

mov dx, mass[si+10] ;записываем в регистр dx элемент массива

mov ch, 2

mov ax, dx

div ch ;проверяем его на чётность

xor ch, ch

cmp ah, ch

JE nachalo ;если чётный, то идём в начало

cmp dx, bx ;если нечётный, то сравниваем элемент массива со значением минимума

JGE nachalo ;если больше или равно, то идём в начало

mov bx, dx ;если меньше, то присваиваем переменной minc значение регистра dx

nachalo:

LOOP minimymi

xor si, si

mov bx, maxc

mov cx, 3 ;3 повторения, т.к. имеется только 3 столбца

maksimymi:

mov ax, mass[si+6] ;записываем в регистр ax элемент массива

ROR ax, 1 ;сдвигаем вправо для проверки чётности

JC konez ;если нечётный, то идём в конец

mov dx, mass[si+6] ;если чётный, то записываем в регистр dx элемент массива

cmp dx, bx ;сравниваем его со значением максимума

JLE konez ;если меньше, то идём в конец

mov bx, dx ;если больше, то присваиваем переменной minc значение регистра dx

konez:

mov dx, cx

mov cx, 12

sm:

inc si

LOOP sm

mov cx, dx

LOOP maksimymi

mov ah, 4ch

int 21h

kod ends

end begin

end