Выполнил студент 213 группы: Тихонов Евгений

Ответы на вопросы:

1) Массивы и их представление в памяти компьютера.

Ответ: Массивы в памяти компьютера представлены как некое количество идущих подряд ячеек памяти (байтовых, двухбайтовых и т.д.). Стоит понимать что массив представляет собой «сплошной» кусок памяти без каких либо «разрывов».

2) Режимы адресации данных, которые могут применяться для доступа к элементам массива.

Ответ: Существует несколько режимов адресации к данным для доступа к ячейкам массива:

Прямая адресация памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

Базовая и индексная адресация памяти. Относительный адрес ячейки памяти находится в регистре, обозначение которого заключается в квадратные скобки. При использовании регистров BX или BP адресацию называют базовой, при использовании регистров SI или DI - индексной. При адресации через регистры BX, SI или DI в качестве сегментного регистра подразумевается DS; при адресации через BP - регистр SS. Таким образом, косвенная адресация через регистр BP предназначена для работы со стеком. Однако при необходимости можно явно указать требуемый сегментный регистр.

Базовая и индексная адресации памяти со смещением. Относительный адрес операнда определяется суммой содержимого регистра (BX, BP, SI или DI) и указанного в команде числа, которое называют смещением.

Базово-индексная адресация памяти. Относительный адрес операнда определяется суммой содержимого базового и индексного регистров. Допускается использование следующих пар:[BX][SI], [BX][DI], [BP][SI], [BP][DI]

3) Описание массивов в сегменте данных.

Ответ: В сегменте данных описание массива происходит следующим образом: Указывается имя под которым будет подразумеваться указатель на первый элемент массива, затем идет указание размера ячеек, выделяемых для хранения данных(байт, слово, двойное слово), затем идет оператор DUP, который указывает какие значения будут лежать в ячейках. Пример:

Arr DB 30 DUP(1,2)

Arr1 DB 15 DUP(14)

Arr_DW DW 4 DUP(?)

4) Особенности обработки двумерных массивов в ассемблерных программах. Вычисление эффективного адреса (EA) элемента двумерного массива.

Ответ: Двумерный массив в ассемблере представляется с помощью одномерного массива, с той поправкой, что мы условно делим наш массив на равные части, которые будут представлять из себя «строки» нашего двумерного массива. Например мы выделили 18 ячеек памяти в виде одномерного массива, мы условно делим его на 3 части(по 6 элементов в одной части) и у нас получается двумерный массив имеющий размеры 3*6(3 строки и 6 столбцов). Исходя что мы хотим обращаться к элементам двумерного массива относительно строк и столбцов нам нужно проводить некие вычисления для доступа к нужным элементам. Вычисления идут следующим образом: Начало массива + (индекс строки*длину строки) + индекс столбца. Индексация строк и столбцов начинается с 0

5) Какие режимы адресации данных можно использовать для доступа к элементам двумерного массива?

Ответ: Для адресации к элементам двумерного массива можно использовать те же самые режимы адресации что и для одномерного массива.

Программа:

Программа написана на основе ассемблера TASM.
Тестирование проводились в среде DOSBox, программой TURBO DEBUGGER.

Код программы:

```
stack_seg segment stack "stack"
    db 20 dup(?)
stack_seg ends


data_seg segment
    Arr dw 18 DUP(?)
    MinMax dw 2 DUP(0)
data_seg ends


code_seg segment "code"
    assume ss:stack_seg, ds:data_seg, cs:code_seg
    Start:
        ;set data segment
        mov ax,data_seg
        mov ds,ax
        ;find fibo nums
        mov Arr+0, 0; Arr[0] = 0
```

```
mov Arr+2, 1; Arr[1] = 1
mov CX, 16
mov si,4
fibo_loop:
    sub si, 2; i-1 indx
    mov ax, Arr[si]; ax = Arr[i-1]

    sub si, 2; i-2 indx
    add ax, Arr[si]; ax += Arr[i-2]

    add si, 4; i indx (i-2-2+4 = i)

    mov Arr[si], ax; Arr[i] = ax

    add si, 2; i++
    loop fibo_loop

;find min odd elem in 2nd row 3*6 matrix
mov CX, 6; 6 iters
mov si, 12; start from 6 index (12 because 1 cell = 2 byte)
mov MinMax+0, 10000; initialize var for min elem
find_min_loop:
    mov ax, Arr[si]
    call GetEvenBit; mov even bit to ax
    cmp AX, 0       ; if AX == 0
    JE else_finded_min; num even, skip
                    ; else go to find min
    mov ax, Arr[si]
    cmp MinMax+0, ax ; if MinMax[0] <= Arr[i]
```

```
        JBE else_finded_min; jump to else_finded_min
        if_finded_min:
            mov ax, Arr[si] ; if MinMax[0] > Arr[i]
            mov MinMax+0, ax; MinMax[0] = Arr[i]
        else_finded_min:
        add si, 2; i++
        loop find_min_loop

    ;find max even elem in 4rd column 3*6 matrix
    mov CX, 3; 3 iters
    mov si, 6; start from 3 index (6 because 1 cell = 2 byte)
    mov MinMax+2, 0; initialize var for max elem
    find_max_loop:
        mov ax, Arr[si]
        call GetEvenBit; mov even bit to ax
        cmp AX, 1       ; if AX == 1
        JE else_finded_max; num odd, skip
                        ; else go to find max
        mov ax, Arr[si]
        cmp MinMax+2, ax   ;if MinMax[1] >= Arr[i]
        JNB else_finded_max;jump to else_finded_max
        if_finded_max:
            mov ax, Arr[si] ; if MinMax[1] < Arr[i]
            mov MinMax+2, ax; MinMax[0] = Arr[i]
        else_finded_max:
        add si, 12; i+=6
        loop find_max_loop

    ; correct exit
```

```
        mov ah,4ch
        int 21h


    GetEvenBit proc near; mov first bit (even bit) to AX
…       ror AX, 1; right cycle shift
        LAHF; move last 8 FLAGS bits to AH
        and AH, 00000001b; get first bit(CF), there is our shifted bit
        mov AL, AH
        mov AH, 0
    ret
    GetEvenBit endp


code_seg ends
end Start
```