Универзитет „Св. Кирил и Методиј" во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Факултет за информатички науки и компјутерско инженерство, Скопје

Микропроцесори и микроконтролери

Лабораториска вежба број 03

РАБОТЕЊЕ СО НИЗИ И МАТРИЦИ ВО АСЕМБЛЕР 8086

Работење со низи и матрици во асемблер 8086

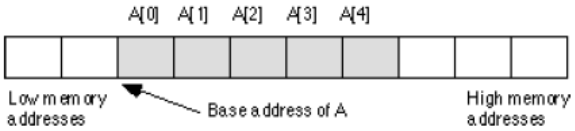
Целта на оваа вежба е изучување на наредбите за работа со низи и матрици во асемблер 8086.

Предмет на работа

1. Запознавање со низи во асемблер.
2. Запознавање со матрици во асемблер.

Запознавање со низи во асемблер

Низа може да се дефинира како секвенца од повеќе последователни елементи од ист тип. Така на пример, во виш програмски јазик низа од 10 целобројни елементи може да се дефинира како: NIZA : ARRAY[0..9] OF INTEGER; До секој елемент се пристапува според целоброен индекс. Во меморија овие низи се претставуваат како последователен простор во меморија каде се сместени соодветните елементи:



Во асемблер, за да се пристапи до секој од елементите во низата потребно е да се знае нивната адреса. Таа може да се добие според следната формула:

```
Element_Address = Base_Address + ((Index - Initial_Index) * Element_Size)
```

- Base_Address е адресата на првиот елемент во низата (A[0])
- Index е на индексот елементот што го бараме (пр. за A[1] индекс е 1)
- Initial_Index е индексот на првиот елемент во низата (најчесто нула)
- Element_Size е големината на секој од елементите изразена во бајти. Ако елементите од низата се од тип db големината е 1, а ако се од тип dw големината е 2.

Дефинирање на низи

Низите вообичаено се дефинираат во податочниот сегмент со следната синтакса:

```
ime_na_niza TIP n dup (?)
```

каде ime_na_niza е името на променливата со која ќе се пристапува до низата, TIP е основниот тип на сите елементи (db = byte, dw = word итн), n е бројот на елементи во низата a dup (?) означува да се направат n дупликати од основниот тип кои се неиницијализирани. При самата дефиниција може и да се иницијализираат низите. Еден пример е:

```
broevi db 5, 10, 15, 20, 25, 30
```

Пример за пристап до низи во меморија

Во овој пример потребно е да напишете код во асемблер кој што ќе изврши печатење на елементите од една низа на екран. Низата се пристапува преку променливата NIZA дефинирана во податочниот сегмент.

Во податочниот сегмент додадете го следниот код:

```
NIZA DB 2, 4, 1, 6, 7
```

На точното обележано место на кодот треба да го напишете следниот код:

```
        MOV BX, OFFSET NIZA
        MOV CX, 05h
        MOV AH, 02h
        MOV AL, 0h
CIKLUS:
        MOV DL, [BX]
        ADD DL, 30h
        INT 21h
        INC BX
        LOOP CIKLUS
```

Да ги појасниме сите редови еден по еден:

1.

```
        NIZA DB 2, 4, 1, 6, 7
```

Во податочниот сегмент се дефинира простор за променливата NIZA која содржи пет елементи.

2.

```
        MOV BX, OFFSET NIZA
```

Во регистарот BX се сместува адресата на почетокот на низата (поместувањето во рамките на DS сегментот, бидејќи NIZA е дефинирана во тој сегмент). Оваа линија код може да се замени со следната која го има истото значење:

```
        LEA BX, NIZA
```

Со инструкцијата LEA(Least effective Address) се копира ефективната адреса на првиот елемент во NIZA во регистерот BX.

3.

```
        MOV CX, 05h
```

Во регистарот CX внесуваме вредност 05h, колку елементи треба да се отпечатат, т.е. колку пати треба да се повтори циклусот.

```
        MOV AH, 02h
        MOV AL, 0h
```

Во регистарот AH внесуваме вредност 02h, што означува печатење при повик на INT 21h.

4.

```
CIKLUS:
        MOV DL, [BX]
        ADD DL, 30h
        INT 21h
        INC BX
        LOOP CIKLUS
```

Ја запишуваме вредноста сместена на локацијата каде покажува регистарот BX во регистарот DL. Потоа го зголемуваме регистарот DL за 30h за да се добие соодветниот ASCII код во DL (30h е ASCII код за знакот 0). Со повик на INT 21h се печати ASCII кодот од DL на екран. Потоа го зголемуваме BX да покажува на следниот елемент. Со наредбата LOOP циклусот продолжува доколку CX е поголемо од нула.

Извршете ја програмата чекор по чекор, и анализирајте ги вредностите на регистрите BX, CX и DL во секој чекор од циклусот. На која локација (физичка и логичка) се запишани податоците? Погледнете ја меморијата.

Запознавање со матрици во асемблер

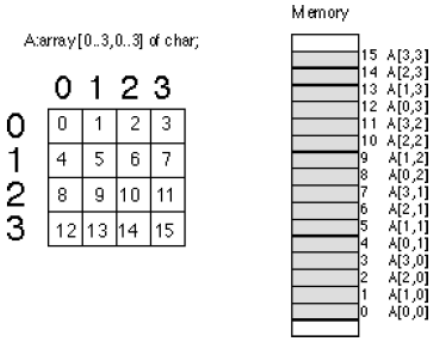
Познато е дека матриците се дводимензионални, но сепак тие треба да се зачуваат во еднодимензионалниот мемориски простор (секоја мемориска локација во податочниот сегмент има точно определена ефективна адреса). На пример, нека имаме матрица од тип char, со големина 3x3. Значи, таа има 9 елементи организирани во три редици и три колони. За тие девет елементи треба да се најде континуиран простор од девет бајти во меморијата. Секој од елементите секогаш треба да е соодветен на единствен бајт од меморијата.

За да ги претставиме деветте елементи во меморијата, ни треба функција со два влезни параметри (за редица и колона) чиј излезен резултат е поместување во еднодимензионална низа со големина од девет. Функцијата треба да е независна од големината на матрицата, а може дури и да не зависи од димензиите на низата (значи да може да се употреби врз еднодимензионални низи, матрици, тридимензионални низи итн.). Оваа постапка е всушност мапирање (хаширање). Програмерите употребуваат два различни пристапа за оваа намена, а тоа се редење на елементите по редици (row major) и редење на елементите по колони (column major). Од нив, најчесто се употребува row major .

Редење на елементите по редици

Со овој пристап последователните елементи од една низа при формирање на матрица се редат по редици. На пример, ако имаме 9 елементи, првите три ќе бидат во првата редица, (редицата со индекс 0), вторите три во втората (со индекс 1), а третите три во третата редица (со индекс 2). Според пристапот на редење на елементите по редица, откако ќе се исполни првата редица, се преминува на полнење на втората, потоа на третата итн.

Пресликувањето на матрицата наредена според редици во мемориски простор се прави на следниов начин: првата редица се пресликува во еднодимензионалниот меморискиот простор. На нејзиниот крај се надоврзува втората редица. На крајот на втората редица се надоврзува почетокот на третата редица итн.



Елементите се мапираат според следната формула:

```
Element_Address = Base_Address + (rowindex*row_size+colindex)* Element_Size
```

При тоа, за произволна матрица A секоја од променливите го означува следново:

- Base_Address – адресата на првиот елемент од матрицата (A[0][0])
- colindex – левиот индекс на елементот (на пример, кај елементот A[0][1], тоа е 0)
- row_size – број на елементи во една од редиците во матрицата (еднакво на бројот на колони по множено со големината на секој елемент во колоната, во примерот на матрицата на сликата овој број е 4)
- rowindex - десниот индекс на елементот (на пример, кај елементот A[0][1], тоа е 1)
- Element_Size – големината на произволен елемент од матрицата, изразен во бајти (ако елементот е word, Element_Size ќе биде 2, ако е byte ќе биде 1)

Ако претпоставиме дека Element_Size=1, rowindex=2, colindex=1, row_size=4, според горната формула ќе добиеме:

```
Element_Address = 0 + (2 * 4 + 1) * 1 = 9
```

Навистина елементот A[1][2], кој се наоѓа во редицата со индекс 2 и колоната со индекс 1, кога ќе се запише во меморија тој се запишува во низа со поместување 9 од почетниот елемент. Погледнете ја сликата дадена погоре.

Пример за пристап до матрици во меморија

Напишете код во асемблер кој ќе иницијализира матрица со нумеричките вредности на знаците т.е. цифрите (0-9) кои се внесуваат од тастатура. Даден е кодот со објаснувања.

Во податочниот сегмент се дефинираат индексите на елемент од матрица i и j како и самата матрица MATRICA која има 4 редици и 5 колони, па оттука има 20 елементи.

```
data segment
      i db ?
      j db ?
      MATRICA db 20 dup (?)
ends
```

Во кодниот сегмент се дефинира кодот на програмата.

```
code segment

start:
      ;се поставуваат сегментните регистри DS и ES
      MOV ax, data
      MOV ds, ax
      MOV es, ax

      MOV i, 0h ;се иницијализира на 0 бројачот по редици
RED: `
      MOV j, 0h ;се иницијализира на 0 бројачот по колони
KOLONA:

      MOV ah,01h
      INT 21h ;се чита знак од тастатура (ASCII код на цифра) и се сместува
во AL

      MOV cl, al ;AX ќе се користи понатаму па знакот се копира во CL
      SUB cl, 30h ;од CL се одзема 30h (ASCII кодот за '0') за да се добие
нумеричката вредност за внесената цифра ('5'-> 5)

      ;следно се пресметува поместувањето за матрицата со индекси i и j
      MOV ax, 4 ;4 големината на една редица
      MUL i   ;се множи 4 со i и производот се сместува во AX

      MOV dh, 0h
      MOV dl, j
      ADD  ax,  dx   ;се додава j на AX  и се добива поместувањето од
почетниот елемент на матрицата

      MOV bx, ax
      MOV MATRICA[bx], cl ;вредноста на цифрата се копира во меморија

      INC j ; се зголемува бројачот на колони
      CMP j, 4h ;се проверува дали се поминати елементите од сите колони
      JNE KOLONA ;ако не се враќаме
```

```
        INC i ;се зголемува бројачот на редици
        CMP i, 5h ;се проверува дали се поминати елементите од сите редици
        JNE RED

        MOV ax, 4c00h ;излез
        INT 21h
ends
end start
```