Two Dimensional Arrays

Muhammad Afzaal
m.afzaal@nu.edu.pk

Book Chapter

- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 9
  - Section 9.4

Arrays

- Indirect operands can be used to step through arrays

```
.data
arr DB 10h, 20h, 30h, 40h
.code
MOV si, OFFSET arr
MOV ecx, 4
MOV ax, 0
L1:
    ADD ax, [si]
    ADD si, TYPE arr
    LOOP L1
```

Two Dimensional Arrays (1/2)

- Two-dimensional array is high-level abstraction of a one-dimensional array
- Two methods to arrange rows and columns
  - **Row-Major Order**
  - Column-Major Order
- x86 ISA has two operand types, base-index and base-index-displacement which are ideal to use with arrays

Two Dimensional Arrays (2/2)

Logical arrangement

	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90

Row-Major Order

10	20	30	40	50	60	70	80	90
00	01	02	10	11	12	20	21	22

Column-Major Order

10	40	70	20	50	80	30	60	90
00	10	20	01	11	21	02	12	22

Base-Index Operands

- Base-Index operand adds the values of two registers to produce an offset
- These two registers are called Base and Index
- In 32-bit mode, any extended general purpose register may be used as base and index
- In 16-bit mode, base register must be either BX or BP and index must be SI or DI
- Syntax is `[Base+Index]`

Accessing One-Dimensional Array

```
.data
    arr DB 10h, 20h, 30h, 40h
.code
    MOV BX, OFFSET arr
    MOV SI, 2
    MOV AL, [BX+SI] ;AL=
    INC SI
    MOV AL, [BX+SI] ;AL=
```

Accessing Two-Dimensional Array (1/2)

- In row-major order
  - Row-offset is held in base register
  - Column-offset is held inside index register

```
.data
    arr DB 10h, 20h, 30h
    r_size=($-arr)
        DB 40h, 50h, 60h
        DB 70h, 80h, 90h
```

Accessing Two-Dimensional Array (2/2)

- Suppose we want to locate value at row=1 and col=2

```
.data
   arr DB 10h, 20h, 30h
   r_size=($-arr)
       DB 40h, 50h, 60h
       DB 70h, 80h, 90h
```

	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90

```
.code
   r_ind=1
   c_ind=2
   MOV BX, OFFSET arr ;BX=100
   ADD BX, r_size*r_ind;BX=100+3*1
   MOV SI, c_ind ;SI=2
   MOV AL, [BX+SI] ;AL=[103+2]
```

	10	20	30	40	50	**60**	70	80	90
Address	100	101	102	103	104	105	106	107	108

Base-Index-Displacement Operands

Read Yourself