Defining Data

Muhammad Afzaal

m.afzaal@nu.edu.pk

Book Chapter

- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 3
  - Section 3.4

Data Definition Statement (1/2)

- Assigns storage in memory for a variable
- Syntax for a data definition statement is

```
[name] directive initializer [,initializer]
```

- Name is optional and must follow the rules of naming the identifiers
- At least one initializer is required
- Question mark (`?`) can be used as initializer if uninitialized variable

Data Definition Statement (2/2)

- Directive can be any of the following

Directive	Description	Usage
DB	**D**efine **B**yte	8-bit Integer
DW	**D**efine **W**ord	16-bit Integer
DD	**D**efine **D**oubleword	32-bit Integer
DQ	**D**efine **Q**uadword	64-bit Integer
DT	**D**efine **T**enbytes	80-bit Integer

DB Directive

- Defines an 8-bit signed or unsigned variable
- The initializer must fit into 8-bits either signed or unsigned
- `name` shows the offset from the beginning of its segment
- Syntax is like this

```
        [name] DB initializer
```

- Examples are

```
val1 DB 255  ; largest unsigned value
val2 DB +127 ; largest signed value
```

DW Directive

- Defines a 16-bit signed or unsigned integer
- The initializer must fit into 16-bits either signed or unsigned
- `name` shows the offset from the beginning of its segment
- Syntax is like this

```
        [name] DW initializer
```

- Examples are

```
val1 DW 65535  ; largest unsigned value
val2 DW -32768 ; smallest signed value
```

DD Directive

- Defines a 32-bit signed or unsigned integer
- The initializer must fit into 32-bits either signed or unsigned
- `name` shows the offset from the beginning of its segment
- Syntax is like this

```
        [name] DD initializer
```

- Examples are

```
val1 DD FFFFFFFFh ;largest unsigned value
val2 DD 80000000h ;smallest signed value
```

DQ Directive

- Defines a 64-bit signed or unsigned integer
- The initializer must fit into 64-bits either signed or unsigned
- `name` shows the offset from the beginning of its segment
- Syntax is like this

```
        [name] DQ initializer
```

- Examples are

```
val1 DQ 10001010h
val2 DQ 10001010b
```

Multiple Initializers

- If multiple initializers are used in the same data definition statement
  - … its label refers only to the offset of first initializer

```
[name] Directive initializer ,initializer
```

- Also called Array
- Example is

```
vals1 DB 10, -20, 30
vals2 DW 0Ah, 10, 00111100b
```

Defining Strings

- Strings are **sequence of characters** including spaces
- Enclosed in single or double quotation marks
- As they are sequence of characters and each character occupies 1 byte, `DB` directive is used to define them
- End with a null byte
- Examples are

```
str1 DB "Hello", 0
str2 DB 'Hello', 0
```

DUP Operator

- **DUP**licates same value on many storage locations
- Useful when allocating space for string or array
- Can be used with initialized or uninitialized data
- Examples are

```
a DB 10 DUP(0) ;10 bytes all zero
b DB 10 DUP(?) ;10 bytes uninitialized
c DB 3 DUP ('hi');6 bytes 'hihihi'
```

Defining Real Number Data

- DD, DQ, DT directives can be used to define real numbers
- Examples are

```
rVal1 DD 1.2
rVal2 DQ 3.1E-190
rVal3 DT 8.9E+3036
```

Data Type	Significant Digits	Approximate Range
DD (Short Real)	6	$1.18x10^{-38}$ to $3.40x10^{38}$
DQ (Long Real)	15	$2.23x10^{-308}$ to $1.79x10^{308}$
DT (Extended-precision Real)	19	$3.37x10^{-4932}$ to $1.18x10^{4932}$

Little Endian Order

- x86 processors store and retrieve data from memory using Little Endian Order
- Least significant byte is stored at the first memory address allocated for data
- Remaining bytes are stored in the next consecutive memory locations
- Example, consider 2-bytes value 1234h
  - If placed in memory at offset 0000, 34h would be stored in first byte
  - 12h would be stored in the second byte

Big Endian Order

- Some other processors store and retrieve data from memory using Big Endian Order
- Most significant byte is stored at the first memory address allocated for data
- Remaining bytes are stored in the next consecutive memory locations
- Example, consider 2-bytes value 1234h
  - If placed in memory at offset 0000, 12h would be stored in first byte
  - 34h would be stored in the second byte