Microprocessors (0630371)
Fall 2010/2011 – Lecture Notes # 7

Defining Data

Outline of the Lecture

- **Defining BYTE and SBYTE Data**
- **Defining Strings**
- **Defining WORD and SWORD Data**
- **Defining DWORD and SDWORD Data**
- **Defining QWORD Data**
- **Defining TBYTE Data**
- **Defining Real Number Data**
- **Adding Variables to the AddSub Program**
- **Declaring Uninitialized Data**
- **Mixing Code and Data**

Defining BYTE and SBYTE Data

- The **BYTE** (define byte) and **SBYTE** (define signed byte) directives allocate storage for one or more unsigned or signed values, for example:

```
value1 BYTE 'A'      ; character constant
value2 BYTE 0        ; smallest unsigned byte
value3 BYTE 255      ; largest unsigned byte
value4 SBYTE -128    ; smallest signed byte
value5 SBYTE +127    ; largest signed byte
```

- A question mark **(?)** initializer leaves the variable uninitialized, implying it will be assigned a value at runtime:

```
value6 BYTE ?
```

- The optional name is a label marking the variable's offset from the beginning of its enclosing segment. For example, if **value7** is located at offset 0000 in the data segment, **value8** is automatically located at offset 0001:

```
Value7 BYTE 10h
Value8 BYTE 20h
```

- The **DB legacy directive** can also define an 8-bit variable, signed or unsigned:

```
val1 DB 255 ; unsigned byte
val2 DB -128 ; signed byte
```

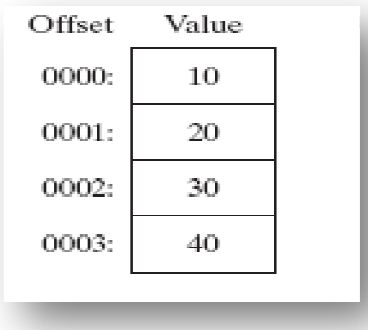
Multiple Initializers

- If multiple initializers are used in the same data definition, its label refers only to the **offset of the first initialize**, for example

```
list BYTE 10,20,30,40
```

- Assume list is located at offset 0000. If so, the value 10 is at offset 0000, 20 is at offset 0001, 30 is at offset 0002, and 40 is at offset 0003
- Not all data definitions require labels

```
list BYTE 10,20,30,40
     BYTE 50,60,70,80
     BYTE 81,82,83,84
```



- Within a single data definition, its initializers can use different radixes. Character and string constants can be freely mixed.

```
list1 BYTE 10, 32, 41h, 00100010b
list2 BYTE 0Ah, 20h, 'A', 22h
```

Defining Strings

- To define a string of characters, enclose them in single or double quotation marks.
- **null-terminated string**

```
greeting1 BYTE "Good afternoon",0
greeting2 BYTE 'Good night',0
```

- Strings are an exception to the rule that byte values must be separated by commas. Without that exception, `greeting1` would have to be defined as

```
greeting1 BYTE 'G','o','o','d'....etc.
```

- A string can be spread across multiple lines without having to supply a label for each line:

```
greeting1 BYTE "Welcome to the Encryption Demo program "
          BYTE "created by Kip Irvine.",0dh,0ah
          BYTE "If you wish to modify this program, please
     "
          BYTE "send me a copy.",0dh,0ah,0
```

- The hexadecimal codes **0Dh** and **0Ah** are alternately called end-of-line characters.

Example

```
menu BYTE "Checking Account",0dh,0ah,0dh,0ah,
     "1. Create a new account",0dh,0ah,
     "2. Open an existing account",0dh,0ah,
     "3. Credit the account",0dh,0ah,
     "4. Debit the account",0dh,0ah,
     "5. Exit",0ah,0ah,
     "Choice> ",0
```

DUP Operator

- The DUP operator allocates storage for multiple data items, using a constant expression as a counter.
- It is particularly useful when allocating space for a string or array, and can be used with initialized or uninitialized data:

```
BYTE 20 DUP(0)      ; 20 bytes, all equal to zero
BYTE 20 DUP(?)      ; 20 bytes, uninitialized
BYTE 4 DUP("STACK") ; 20 bytes: "STACKSTACKSTACKSTACK"
```

Defining WORD and SWORD Data

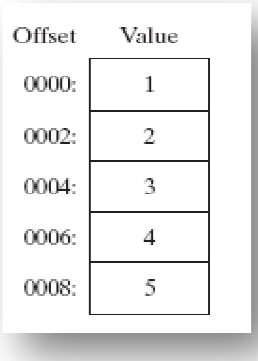
```
word1 WORD 65535    ; largest unsigned value
word2 SWORD -32768  ; smallest signed value
word3 WORD ?        ; uninitialized, unsigned
```

- The legacy DW directive can also be used:

```
val1 DW 65535       ; unsigned
val2 DW -32768      ; signed
```

- Array of Words: Create an array of words by listing the elements or using the DUP operator.

```
myList WORD 1,2,3,4,5
array WORD 5 DUP(?) ; 5 values, uninitialized
```



Offset	Value
0000:	1
0002:	2
0004:	3
0006:	4
0008:	5

Defining DWORD and SDWORD Data

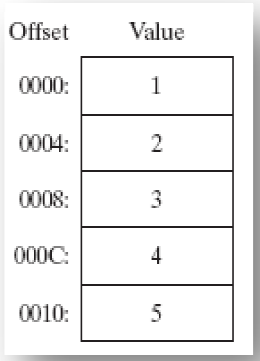
```
val1 DWORD 12345678h      ; unsigned
val2 SDWORD -2147483648   ; signed
val3 DWORD 20 DUP(?)      ; unsigned array
```

- The legacy DD directive can also be used:

```
val1 DD 12345678h         ; unsigned
val2 DD -2147483648       ; signed
```

- Array of Doublewords

```
myList DWORD 1,2,3,4,5
array DWORD 5 DUP(?) ; 5 values, uninitialized
```



Offset	Value
0000:	1
0004:	2
0008:	3
000C:	4
0010:	5

Defining QWORD Data

```
quad1 QWORD 1234567812345678h
quad1 DQ 1234567812345678h
```

Defining TBYTE Data

- This data type is primarily for the storage of binary-coded decimal numbers (packed BCD Integers in 10-byte package)
- Each byte (exept the highest) contains two BCD numbers.
- In the highest byte, the highest bit indicates the number's sign (if the highest byte =80h, the number is negative;if the highest byte =00h, the number is positive).
- The integer range is **-999,999,999,999,999,999** to **+999,999,999,999,999,999**

```
val1 TBYTE 10000000000123456789Ah
val1 DT 10000000000123456789Ah
```

- Constant initializers must be in hexa numbers

```
BCDVal1 TBYTE 800000000000001234h ; valid
BCDVal2 TBYTE -1234               ; invalid
```

Defining Real Number Data

- REAL4 defines a 4-byte single-precision real variable.
- REAL8 defines an 8-byte double-precision real,
- REAL10 defines a 10-byte double extended-precision real.

```
rVal1 REAL4 -1.2
rVal2 REAL8 3.2E-260
rVal3 REAL10 4.6E+4096
ShortArray REAL4 20 DUP(0.0)
```

- The legacy DD, DQ, and DT directives can define real numbers:

```
rVal1 DD -1.2       ; short real
rVal2 DQ 3.2E-260   ; long real
rVal3 DT 4.6E+4096  ; extended-precision real
```

Adding Variables to the AddSub Program

```
TITLE Add and Subtract, Version 2 (AddSub2.asm)
; This program adds and subtracts 32-bit unsigned
; integers and stores the sum in a variable.
INCLUDE Irvine32.inc
.data
val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
```

```
finalVal DWORD ?
.code
main PROC
mov eax,val1 ; start with 10000h
add eax,val2 ; add 40000h
sub eax,val3 ; subtract 20000h
mov finalVal,eax ; store the result (30000h)
call DumpRegs ; display the registers
exit
main ENDP
END main
```

Declaring Uninitialized Data

- The **.DATA?** directive declares uninitialized data. When definiting a large block of uninitialized data, the .DATA? directive reduces the size of a compiled program.
- the following code is declared efficiently:

```
.data
smallArray DWORD 10 DUP(0) ; 40 bytes
.data?
bigArray DWORD 5000 DUP(?) ; 20,000 bytes, not initialized
```

- The following code, on the other hand, produces a compiled program 20,000 bytes larger:

```
.data
smallArray DWORD 10 DUP(0) ; 40 bytes
bigArray DWORD 5000 DUP(?) ; 20,000 bytes
```

Mixing Code and Data

- The assembler lets you switch back and forth between code and data in your programs.

```
.code
mov eax,ebx
.data
temp DWORD ?
.code
mov temp,eax
. . .
```