Outline of the Lecture

- **Symbolic Constants**
  - **Equal-Sign Directive**
  - **Calculating the Sizes of Arrays and Strings.**
  - **EQU Directive**
  - **TEXTEQU Directive**
- **Real-Address Mode Programming**

### Symbolic Constants

- A symbolic constant (or symbol definition) is created by associating an identifier (a symbol) with an integer expression or some text.
- Symbols cannot change at run time.
- Unlike a variable definition, a symbolic constant does no use any storage.

<table>
<thead>
<tr>
<th>Uses storage?</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value changes at run time?</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Equal-Sign Directive

```plaintext
name = expression
```

- **name** is called a symbolic constant
- **expression** is a 32-bit integer (expression or constant)
- Good programming style to use symbols.
  - **Example 1 (Keyboard Definitions)**
    ```plaintext
    Esc_key = 27
    mov al, Esc_key ; good style
    ```
    Rather than
    ```plaintext
    mov al, 27 ; poor style
    ```
  - **Example 2 (Using the DUP Operator)**
    The counter used by DUP should be a symbolic constant
    ```plaintext
    Count = 5
    array DWORD COUNT DUP(0)
    ```

- May be **redefined**.
  A symbol defined with `_` can be redefined within the same program.
  ```plaintext
  COUNT = 5
  mov al, COUNT ; AL = 5
  COUNT = 10
  mov al, COUNT ; AL = 10
  COUNT = 100
  mov al, COUNT ; AL = 100
  ```

#### Calculating the Size of a Byte Array.

- Uses a constant named **ListSize** to declare the size of list:
  ```plaintext
  list BYTE 10, 20, 30, 40
  ListSize = 4
  ```
A better way to handle this situation would be to let the assembler automatically calculate `ListSize`. The `$` operator (current location counter) returns the offset associated with the current program statement:

```assembly
list BYTE 10,20,30,40
ListSize = ($ - list)
```

`ListSize` must follow immediately after `list`.

```assembly
list BYTE 10,20,30,40
var2 BYTE 20 DUP(?)
ListSize = ($ - list) ;incorrect
```

### Calculating the Size of a Word Array

- **current location counter:** `$`
  - subtract address of list
  - difference is the number of bytes
  - divide by 2 (the size of a word)

```assembly
list WORD 1000h,2000h,3000h,4000h
ListSize = ($ - list) / 2
```

### Calculating the Size of a Doubleword Array

- **current location counter:** `$`
  - subtract address of list
  - difference is the number of bytes
  - divide by 4 (the size of a doubleword)

```assembly
list DWORD 1,2,3,4
ListSize = ($ - list) / 4
```

### Calculating the Size of a string

- Rather than calculating the length of a string manually, let the assembler do it:
  ```assembly
  myString BYTE "This is a long string, containing"
  BYTE "any number of characters"
  myString_len = ($ - myString)
  ```

### EQU Directive

- The EQU directive associates a symbolic name with an integer expression or some arbitrary text.
- There are three formats:
  ```assembly
  name EQU expression
  name EQU symbol
  name EQU <text>
  ```
  - `expression` must be a valid integer expression
  - `symbol` is an existing symbol name, already defined with = or EQU.
  - `text` is any text may appear within the brackets `< . . >`

- EQU can be useful when defining a value that does not evaluate to an integer:
  ```assembly
  PI EQU <3.1416>
  ```

- associate a symbol with a character string
  ```assembly
  pressKey EQU "Press any key to continue...",0
  ```

- associate a symbol with an expression
  ```assembly
  matrix1 EQU 10 * 10
  matrix2 EQU <10 * 10>
  ```

- Cannot be redefined
TEXTEQU Directive

- Define a symbol as either an integer or text expression called a text macro.
- There are three different formats:

  ```
  name TEXTEQU <text>
  name TEXTEQU textmacro
  name TEXTEQU %constExpr
  ```

- **Example 1**

  ```
  continueMsg TEXTEQU <"Do you wish to continue (Y/N)?">.
data
  prompt1 BYTE continueMsg
  ```

- **Example 2**

  ```
  continueMsg TEXTEQU <"Do you wish to continue (Y/N)?">.
  rowSize = 5.
data
  prompt1 BYTE continueMsg
  count TEXTEQU % (rowSize * 2) ;
  move TEXTEQU <mov>
  setupAL TEXTEQU <move al,count>
  .code
  setupAL ; generates: "mov al,10"
  ```

- **TEXTEQU** Can be redefined.

The following program illustrates the definition of symbolic constants:

```assembly
TITLE Symbolic Constants (File: Constants.asm)
; Demonstration of EQU and = directives
.MODEL flat, stdcall
.STACK
INCLUDE Irvine32.inc
.data
Rows EQU 3
Cols EQU 3
Elements EQU Rows * Cols
CR EQU 10
LF EQU 13
PromptText EQU <"Press any key to continue ...",CR,LF,0>
matrix WORD Elements DUP(0)
prompt BYTE BYTE PromptText
COUNT = 10h
COUNT = 100h
COUNT = 1000h
COUNT = SIZEOF matrix
.code
main PROC
exit
main ENDP
END main
```
Real-Address Mode Programming

Generate 16-bit MS-DOS Programs

- **Advantages**
  - enables calling of MS-DOS and BIOS functions
  - no memory access restrictions

- **Disadvantages**
  - must be aware of both segments and offsets
  - cannot call Win32 functions (Windows 95 onward)
  - limited to 640K program memory

- **Requirements**
  - INCLUDE Irvine16.inc
  - Two additional instructions are inserted at the beginning of the startup procedure (main)
  - Initialize DS to the data segment using predefined MASM constant `@data`:
    
    ```assembly
    mov ax, @data
    mov ds, ax
    ```

Add and Subtract, 16-Bit Version

```
TITLE Add and Subtract, Version 2      (AddSub2.asm)
INCLUDE Irvine16.inc
.data
val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
finalVal DWORD ?
.code
main PROC
    mov ax, @data ; initialize DS
    mov ds, ax
    mov eax, val1 ; get first value
    add eax, val2 ; add second value
    sub eax, val3 ; subtract third value
    mov finalVal, eax ; store the result
    call DumpRegs ; display registers
exit
main ENDP
END main
```

Programming Exercise 1

The following exercise can be done in protected mode or real-address mode.

**Subtracting Three Integers**

Using the `AddSub.asm` program as a reference, write a program that subtracts three integers using only 16-bit registers. Insert a call `DumpRegs` statement to display the register values.