

---

# Macros and Conditional Assembly

Chapter 10  
S. Dandamudi

---

## Outline

---

- What are macros?
- Macros with parameters
- Macros vs procedures
  - \* Parameter passing
  - \* Types of parameters
  - \* Invocation mechanism
  - \* When are macros better?
- Labels in macros
- Comments in macros
- Macro operators
- List control directives
- Repeat block directives
  - \* REPT directive
  - \* WHLE directive
  - \* IRP and IRPC directives
- Conditional assembly
  - \* IF and IFE
  - \* IFDEF and IFNDEF
  - \* IFB and IFNB
  - \* IFIDN and IFDIF
- Nested macros
- Performance: Macros vs procedures

## What Are Macros?

---

- Macros provide a means to represent a block of text (code, data, etc.) by a name (*macro name*)
- Macros provide a sophisticated text substitution mechanism
- Three directives

\* =

Example: **CLASS\_SIZE = 90** (can be redefined later)

\* EQU

» Example: **CLASS\_SIZE EQU 90**

\* MACRO

1998

© S. Dandamudi

Macros: Page 3

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## What Are Macros? (cont'd)

---

- Macros can be defined with MACRO and ENDM
- Format

```
macro_name  MACRO [parameter1, parameter2, ...]
             macro body
             ENDM
```

- A macro can be invoked using

```
macro_name [argument1, argument2, ...]
```

Example:	<u>Definition</u>	<u>Invocation</u>
	<pre>multAX_by_16  MACRO                sal    AX,4                ENDM</pre>	<pre>... mov    AX,27 multAX_by_16 ...</pre>

1998

© S. Dandamudi

Macros: Page 4

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macros with Parameters

---

- Macros can be defined with parameters
  - » More flexible
  - » More useful
- Example

```
mult_by_16    MACRO    operand
               sal     operand,4
               ENDM
```

\* To multiply a byte in DL register

```
mult_by_16    DL
```

\* To multiply a memory variable **count**

```
mult_by_16    count
```

## Macros with Parameters (cont'd)

---

Example: To exchange two memory words

```
Wmxchg    MACRO    operand1, operand2
           xchg    AX,operand1
           xchg    AX,operand2
           xchg    AX,operand1
           ENDM
```

Example: To exchange two memory bytes

```
Bmxchg    MACRO    operand1, operand2
           xchg    AL,operand1
           xchg    AL,operand2
           xchg    AL,operand1
           ENDM
```

## Macros vs. Procedures

---

- Similar to procedures in some respects
  - \* Both improve programmer productivity
  - \* Aids development of modular code
  - \* Can be used when a block of code is repeated in the source program
- Some significant differences as well
  - \* Parameter passing
  - \* Types of parameters
  - \* Invocation mechanism

1998

© S. Dandamudi

Macros: Page 7

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macros vs. Procedures (cont'd)

---

### Parameter passing

- In macros, similar to a procedure call in a HLL

```
mult_by_16    AX
```
- In procedures, we have to push parameters onto the stack

```
push    AX
call   times16
```
- Macros can avoid
  - \* Parameter passing overhead
    - Proportional to the number of parameters passed
  - \* **call/ret** overhead

1998

© S. Dandamudi

Macros: Page 8

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macros vs. Procedures (cont'd)

---

### Types of parameters

- \* Macros allow more flexibility in the types of parameters that can be passed
  - Result of it being a text substitution mechanism

### Example

```
shift    MACRO    op_code,operand,count
          op_code operand,count
        ENDM
```

can be invoked as

```
shift    sal,AX,3
```

which results in the following expansion

```
sal     AX,3
```

1998

© S. Dandamudi

Macros: Page 9

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macros vs. Procedures (cont'd)

---

### Invocation mechanism

- \* Macro invocation
  - » Done at assembly time by text substitution
- \* Procedure invocation
  - » Done at run time
- Tradeoffs

Type of overhead	Procedure	Macro
Memory space	lower	higher
Execution time	higher	lower
Assembly time	lower	higher

1998

© S. Dandamudi

Macros: Page 10

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## When Are Macros Better?

---

- Useful to extend the instruction set by defining macro-instructions

<pre>times16 PROC     push BP     mov BP, SP     push AX     mov AX, [BP+4]     sal AX, 4     mov [BP+4], AX     pop AX     pop BP     ret 2 times16 ENDP</pre>	<p style="text-align: center;">Invocation</p> <pre>push count call times16 pop count</pre>
---	--

Too much overhead  
Use of procedure is impractical

## When Are Macros Better? (cont'd)

---

- Sometimes procedures cannot be used
  - » Suppose we want to save and restore BX, CX, DX, SI, DI, and BP registers
  - » Cannot use **pusha** and **popa** as they include AX as well

<pre>save_regs MACRO     push BP     push DI     push SI     push DX     push CX     push BX ENDM</pre>	<pre>restore_regs MACRO     pop BX     pop CX     pop DX     pop SI     pop DI     pop BP ENDM</pre>
---	--

## Labels in Macros

---

- Problem with the following macro definition

```
to_upper0    MACRO    ch
              cmp     ch, 'a'
              jb     done
              cmp     ch, 'z'
              ja     done
              sub     ch, 32
done:
              ENDM
```

- \* If we invoke it more than once, we will have duplicate label **done**

1998

© S. Dandamudi

Macros: Page 13

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Labels in Macros (cont'd)

---

- Solution: Use LOCAL directive
- Format: LOCAL local\_label1 [,local\_label2,...]

```
to_upper    MACRO    ch
              LOCAL   done
              cmp     ch, 'a'
              jb     done
              cmp     ch, 'z'
              ja     done
              sub     ch, 32
done:
              ENDM
```

Assembler uses labels  
??XXXX  
where XXXX is  
between 0 and FFFFH  
To avoid conflict,  
do not use labels that  
begin with ??

1998

© S. Dandamudi

Macros: Page 14

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Comments in Macros

---

- We don't want comments in a macro definition to appear every time it is expanded

» The ;; operator suppresses comments in the expansions

```
;;Converts a lowercase letter to uppercase.
to_upper MACRO ch
    LOCAL done
    ; case conversion macro
    cmp ch,'a' ;; check if ch >= 'a'
    jb done
    cmp ch,'z' ;; and if ch >= 'z'
    ja done
    sub ch,32 ;; then ch := ch - 32
done:
ENDM
```

## Comments in Macros (cont'd)

---

- Invoking the `to_upper` macro by

```
mov AL,'b'
to_upper AL
mov BL,AL
mov AH,'1'
to_upper AH
mov BH,AH
```

generates the following macro expansion



## Comments in Macros (cont'd)

---

```
17 0000 B0 62          mov     AL,'b'
18                    to_upper  AL
1  19                    ; case conversion macro
1  20 0002 3C 61          cmp     AL,'a'
1  21 0004 72 06          jb     ??0000
1  22 0006 3C 7A          cmp     AL,'z'
1  23 0008 77 02          ja     ??0000
1  24 000A 2C 20          sub     AL,32
1  25 000C                    ??0000:
26 000C 8A D8          mov     BL,AL
27 000E B4 31          mov     AH,'1'
```

1998

© S. Dandamudi

Macros: Page 17

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Comments in Macros (cont'd)

---

```
28                    to_upper  AH
1  29                    ; case conversion macro
1  30 0010 80 FC 61          cmp     AH,'a'
1  31 0013 72 08          jb     ??0001
1  32 0015 80 FC 7A          cmp     AH,'z'
1  33 0018 77 03          ja     ??0001
1  34 001A 80 EC 20          sub     AH,32
1  35 001D                    ??0001:
36 001D 8A FC          mov     BH,AH
```

1998

© S. Dandamudi

Macros: Page 18

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macro Operators

---

- Five operators

;;	Suppress comment
&	Substitute
< >	Literal-text string
!	Literal-character
%	Expression evaluate

- \* We have already seen ;; operator
- \* We will discuss the remaining four operators

## Macro Operators (cont'd)

---

### Substitute operator (&)

- \* Substitutes a parameter with the actual argument

- Syntax: **&name**

```
sort2    MACRO  cond, num1, num2
          LOCAL  done
          push   AX
          mov    AX, num1
          cmp   AX, num2
          j&cond done
          xchg  AX, num2
          mov   num1, AX
done:
          pop   AX
          ENDM
```

## Macro Operators (cont'd)

---

- To sort two unsigned numbers **value1** and **value2**, use

```
sort2 ae,value1,value2
```

generates the following macro expansion

```
push    AX
        mov    AX,value1
        cmp    AX,value2
        jae    ??0000
        xchg   AX,value2
        mov    value1,AX
??0000:
        pop    AX
```

- To sort two signed numbers **value1** and **value2**, use

```
sort2 ge,value1,value2
```

1998

© S. Dandamudi

Macros: Page 21

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macro Operators (cont'd)

---

### Literal-text string operator (< >)

\* Treats the enclosed text as a single string literal rather than separate arguments

\* Syntax: **<text>**

```
range_error1  MACRO    number,variable,range
err_msg&number  DB    '&variable: out of range',0
range_msg&number  DB    'Correct range is &range',0
                ENDM
```

- Invoking with

```
range_error1  1,<Assignment mark>,<0 to 25>
```

produces

```
err_msg1     DB    'Assignment mark: out of range',0
range_msg1   DB    'Correct range is 0 to 25',0
```

1998

© S. Dandamudi

Macros: Page 22

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macro Operators (cont'd)

---

### Literal-character operator (!)

- \* Treats the character literally without its default meaning
- \* Syntax: **!character**

```
range_error2    MACRO    number,variable,range
err_msg&number  DB    '&variable: out of range - &range',0
                ENDM
```

- Invoking with

```
range_error2    3,mark,<can!'!'t be !> 100>
```

produces

```
err_msg3    DB    'mark: out of range - can''t be > 100',0
```

- \* Without the ! operator, two single quotes will produce a single quote in the output

1998

© S. Dandamudi

Macros: Page 23

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Macro Operators (cont'd)

---

### Expression Evaluate operator (%)

- \* Expression is evaluated and its value is used to replace the expression itself
- \* Syntax: **%expression**

```
init_array    MACRO    element_size,name,size,init_value
name    D&element_size    size DUP (init_value)
                ENDM
```

- Assuming
- ```
NUM_STUDENTS    EQU    47
NUM_TESTS       EQU    7
```

Invoking with

```
init_array    W,marks,%NUM_STUDENTS*NUM_TESTS,-1
```

produces

```
marks    DW    329 DUP (-1)
```

1998

© S. Dandamudi

Macros: Page 24

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## List Control Directives

---

- Control the contents of **.LST** file
- Two directives control the source lines
  - **.LIST** Allows listing of subsequent source lines  
Default mode
  - **.XLIST** Suppresses listing of subsequent source lines
- Macro invocation call directives
  - **.LALL** Enables listing of macro expansions
  - **.SALL** Suppresses listing of all statements in macro expansions
  - **.XALL** Lists only the source statements in a macro expansion that generates code or data

1998

© S. Dandamudi

Macros: Page 25

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Repeat Block Directives

---

- Three directives to repeat a block of statements
  - \* **REPT**
  - \* **WHILE**
  - \* **IRP/IRPC**
- Mostly used to define and initialize variables in a data segment
- Each directive identifies the beginning of a block
  - \* **ENDM** indicates the end of a repeat block

1998

© S. Dandamudi

Macros: Page 26

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Repeat Block Directives (cont'd)

---

### REPT directive

- Syntax:

```
REPT expression
macro-body
ENDM
```

\* **macro-body** is repeated **expression** times

|                                                                     |                                                                                                               |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <pre>mult_16 MACRO operand REPT 4     sal operand,1 ENDM ENDM</pre> | <pre>mult_16 MACRO operand     sal operand,1     sal operand,1     sal operand,1     sal operand,1 ENDM</pre> |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|

1998

© S. Dandamudi

Macros: Page 27

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Repeat Block Directives (cont'd)

---

### WHILE directive

- Syntax:

```
WHILE expression
macro-body
ENDM
```

\* **macro-body** is executed until **expression** is false (0)

- Following code produces cubed data table

```
WHILE int_value LT NUM_ENTRIES
    DW int_value*int_value*int_value
    int_value = int_value+1
ENDM
```

1998

© S. Dandamudi

Macros: Page 28

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Repeat Block Directives (cont'd)

---

### IRP and IRPC directives

IRP - Iteration RePeat

IRPC - Iteration RePeat with Character substitution

- IRP directive

- Syntax:

```
IRP parameter, <argument1[, argument2, ...]>
    macro-body
```

```
ENDM
```

- \* Angle brackets are required

- \* Arguments are given as a list separated by commas

- » During the first iteration **argument1** is assigned to **parameter**,  
**argument2** during the second iteration, ...

---

1998

© S. Dandamudi

Macros: Page 29

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Repeat Block Directives (cont'd)

---

### IRP example

```
.DATA
IRP value, <9,6,11,8,13>
    DB    value
ENDM
```

produces

```
.DATA
DB    9
DB    6
DB    11
DB    8
DB    13
```

---

1998

© S. Dandamudi

Macros: Page 30

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Repeat Block Directives (cont'd)

---

- IRPC directive

- Syntax:

```
IRPC parameter, string
macro-body
ENDM
```

- \* **macro-body** is repeated once for each character in **string**
- \* **string** specifies
  - » the number of iterations
  - » the character to be used in each iteration
- \* During the first iteration first character of **string** is assigned to **parameter**, second character during the second iteration, ...

## Repeat Block Directives (cont'd)

---

- IRPC example

- \* To generate a sequence of DB statements in the order a, A, e, E, ...

```
defineDB MACRO value
DB value
ENDM

IRPC char, aeiou
defineDB '&char'
defineDB %'&char'-32
ENDM
```

- \* Can also use

```
IRP char, <a,e,i,o,u>
```

in place of **IRPC** statement



## Conditional Assembly

---

- Several conditional directives are available

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| IF/IFE        | Assembles if condition is true (IF)<br>or false (IFE)                                |
| IFDEF/IFNDEF  | Assembles if symbol is defined (IFDEF)<br>or undefined (IFNDEF)                      |
| IFB/IFNB      | Assembles if arguments are blank (IFB)<br>or not blank (IFNB)                        |
| IFIDN/IFDIF   | Assembles if arguments are same (IFIDN)<br>or different (IFDIF) - case sensitive     |
| IFIDNI/IFDIFI | Assembles if arguments are same (IFIDNI)<br>or different (IFDIFI) - case insensitive |

1998

© S. Dandamudi

Macros: Page 33

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Nested Macros

---

- Macros can be nested

```
shifty MACRO lr,operand,count
    IF PROC_TYPE EQ 8086
        IF count LE 4
            REPT count
                sh&lr operand,1
            ENDM
        ELSE
            mov CL,count
            sh&lr operand,CL
        ENDIF
    ELSE
        sh&lr operand,count
    ENDIF
ENDM
```

1998

© S. Dandamudi

Macros: Page 34

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Nested Macros (cont'd)

```
shift  MACRO  operand,count
      ;; positive count => left shift
      ;; negative count => right shift
      IFE count EQ 0
        IF count GT 0      ;; left shift
          shifty l,operand,count
        ELSE                ;; right shift
          ;; count negative
          shifty r,operand,-count
        ENDIF
      ENDIF
    ENDM
```

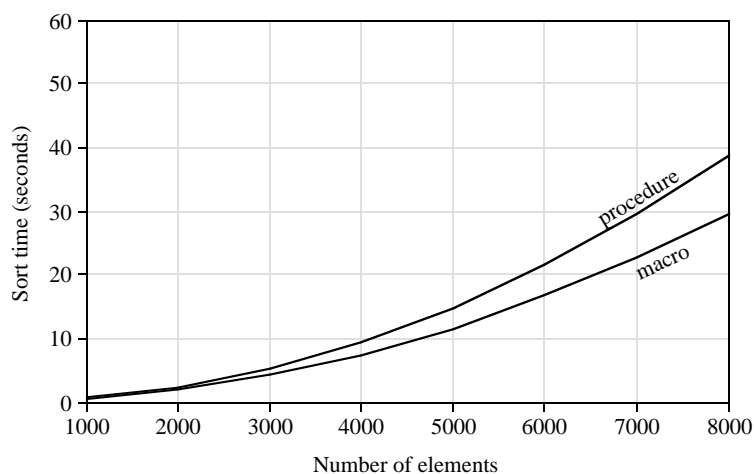
1998

© S. Dandamudi

Macros: Page 35

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Performance: Macros vs Procedures



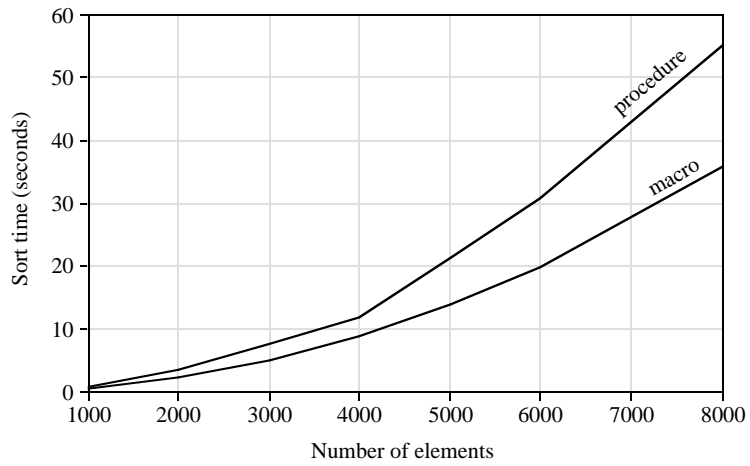
1998

© S. Dandamudi

Macros: Page 36

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

## Performance: Macros vs Procedures (cont'd)



1998

© S. Dandamudi

Macros: Page 37

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.