Assembly Programming

Chapter 10: STRUCTURE AND MACROS

CSE3030

Prof. Youngjae Kim

Distributed Computing and Operating Systems Laboratory (DISCOS)

https://discos.sogang.ac.kr

Structures

- Syntax

```
name STRUCT
    field-declarations ; identical to variable
                       ; declarations
name ENDS
```

- An Example (The COORD structure)

```
COORD STRUCT
    X WORD 10          ; offset 00
    Y WORD ?           ; offset 02
COORD ENDS
```

- Declaring of Structure Variables

```
.data
point1 COORD <5,10>
point2 COORD <>
```

  - Insert replacement initializers between brackets:
    `< . . . >`
  - Empty brackets <> retain the structure's default field initializers

Array of Structures

- An array of structure objects can be defined using the DUP operator.
- Initializers can be used.
- Example1:

```
NumPoints = 3
AllPoints COORD NumPoints DUP(<0,0>)
```

- Example2:

```
Employee STRUCT
    IdNum BYTE "000000000"
    LastName BYTE 30 DUP(0)
    Years WORD 0
    SalaryHistory DWORD 0,0,0,0
Employee ENDS
.data
RD_Dept Employee 20 DUP(<>)
accounting Employee 10 DUP(<,,,4 DUP(20000) >)
```

Use default values

Referencing Structure Variables

```
Employee STRUCT                               ; bytes
   IdNum BYTE "000000000"                     ; 9
   LastName BYTE 30 DUP(0)                    ; 30
   Years WORD 0                               ; 2
   SalaryHistory DWORD 0,0,0,0                ; 16
Employee ENDS                                 ; 57
.data
worker Employee <>
.code
mov eax,TYPE Employee                         ; 57
mov eax,SIZEOF Employee                       ; 57
mov eax,SIZEOF worker                         ; 57
mov eax,TYPE Employee.SalaryHistory           ; 4
mov eax,LENGTHOF Employee.SalaryHistory       ; 4
mov eax,SIZEOF Employee.SalaryHistory         ; 16
mov dx, worker.Years
mov worker.SalaryHistory,20000                ; first salary
mov [worker.SalaryHistory+4],30000            ; second salary
mov edx,OFFSET worker.LastName
mov esi,OFFSET worker
mov ax,(Employee PTR [esi]).Years
mov ax,[esi].Years        ; invalid operand (ambiguous)
```

Looping Through an Array of Points

- Sets the X and Y coordinates of the All Points array to sequentially increasing values (1,1), (2,2), ...

```
.data
NumPoints = 3
AllPoints COORD NumPoints DUP(<0,0>)

.code
   mov edi,0                     ; array index
   mov ecx,NumPoints             ; loop counter
   mov ax,1                      ; starting X, Y values
L1:
   mov (COORD PTR AllPoints[edi]).X, ax
   mov (COORD PTR AllPoints[edi]).Y, ax
   add edi,TYPE COORD
   inc ax
   Loop L1
```

Nested Structure

- Nested Structure is a struct that contains other structs.
- Use nested ‘{‘ (or ‘<‘) to init each COORD structure.

```
Rectangle STRUCT
    UpperLeft COORD <>
    LowerRight COORD <>
Rectangle ENDS
.code
rect1 Rectangle { {10,10}, {50,20} }
rect2 Rectangle < <10,10>, <50,20> >
```

```
COORD STRUCT
    X WORD ?
    Y WORD ?
COORD ENDS
```

- Referencing

```
mov rect1.UpperLeft.X, 10
mov esi,OFFSET rect1
mov (Rectangle PTR [esi]).UpperLeft.Y, 10

// use the OFFSET operator
mov edi,OFFSET rect2.LowerRight
mov (COORD PTR [edi]).X, 50
mov edi,OFFSET rect2.LowerRight.X
mov WORD PTR [edi], 50
```

Declaring and Using Unions

- All of the fields in a union begin at the same offset
  - differs from a structure
- Provides alternate ways to access the same data
- Syntax:

```
unionname UNION
   union-fields
unionname ENDS
```

- Example:

called variant fields

The Integer union consumes 4 bytes (equal to the largest field)

```
Integer UNION
   D DWORD 0
   W WORD 0
   B BYTE 0
Integer ENDS
.data
val1 Integer <12345678h>
val2 Integer <100h>
val3 Integer <>
```

```
mov val3.B, al
mov ax,val3.W
add val3.D, eax
```

MACROs

- A macro (also called a macro procedure) is a named block of assembly language statements.
  - Once defined, it can be invoked (called) one or more times.
  - During the assembler's preprocessing step, each macro call is expanded into a copy of the macro.
  - The expanded code is passed to the assembly step, where it is checked for correctness.

- Syntax:

```
macroname MACRO [parameter-1, parameter-2,...]
      statement-list
ENDM
```

Examples

- mNewLine

```
mNewLine MACRO                    ; define the macro
    call Crlf
ENDM
.data
    . . .
.code
mNewLine                          ; invoke the macro
```

- mPutChar

Definition:

```
mPutchar MACRO char
    push eax
    mov al,char
    call WriteChar
    pop eax
ENDM
```

Invocation:

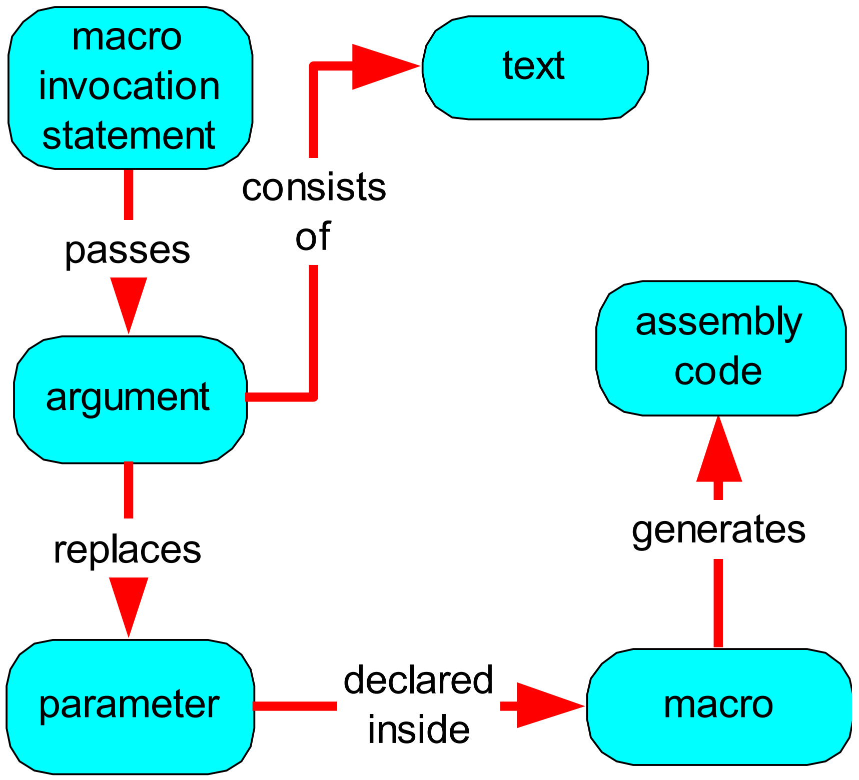
```
.code
mPutchar 'A'
```

Expansion:

```
1       push eax
1       mov al,'A'
1       call WriteChar
1       pop eax
```

Invoking Macros

- Procedures
  - Each argument matches a declared parameter.
  - Each parameter is replaced by its corresponding argument when the macro is expanded. Also, generates assembly language source code.
  - Arguments are treated as simple text by the preprocessor.



mWriteStr Macro

```
mWriteStr MACRO buffer
    push edx
    mov  edx,OFFSET buffer
    call WriteString
    pop  edx
ENDM
.data
str1 BYTE "Welcome!",0
.code
mWriteStr str1
```

Expansion:

```
mWriteStr MACRO buffer
     push edx
     mov  edx,OFFSET buffer
     call WriteString
     pop  edx
ENDM
```

```
1    push edx
1    mov  edx,OFFSET str1
1    call WriteString
1    pop  edx
```

Possible Macro Errors

- Invalid Argument

```
.code
mPutchar 1234h
```

```
1       push eax
1       mov al,1234h    ; error!
1       call WriteChar
1       pop eax
```

- Blank Argument

```
.code
mPutchar
```

```
1       push eax
1       mov al,         ; error!
1       call WriteChar
1       pop eax
```

More Examples

- mReadStr

```
mReadStr MACRO varName
    push ecx
    push edx
    mov edx,OFFSET varName
    mov ecx,(SIZEOF varName) - 1
    call ReadString
    pop edx
    pop ecx
ENDM
.data
firstName BYTE 30 DUP(?)
.code
mReadStr firstName
```

More Examples

- Macro containing code and data
- mWrite
  - The mWrite macro writes a string literal to standard output.

The LOCAL directive prevents string from becoming a global label.

```
mWrite MACRO text
    LOCAL string
    .data                       ;; data segment
    string BYTE text,0          ;; define local string
    .code                       ;; code segment
    push edx
    mov  edx,OFFSET string
    call Writestring
    pop  edx
ENDM
```

Conditional-Assembly Directives

- Checking for Missing Arguments
- Default Argument Initializers
- Boolean Expressions
- IF, ELSE, and ENDIF Directives
- The IFIDN and IFIDNI Directives
- Special Operators
- Macro Functions

Checking for Missing Arguments

- The IFB directive returns true if its argument is blank.
- Example:

```
IFB <row>                 ;; if row is blank,
  EXITM                   ;; exit the macro
ENDIF
```

- mWriteString

```
mWriteStr MACRO string
   IFB <string>
      ECHO ----------------------------------------
      ECHO * Error: parameter missing in mWriteStr
      ECHO * (no code generated)
      ECHO ----------------------------------------
      EXITM
   ENDIF
   push edx
   mov edx,OFFSET string
   call WriteString
   pop edx
ENDM
```

Default Argument Initializers

- A default argument initializer automatically assigns a value to a parameter when a macro argument is left blank.
- Example : **mWriteln** can be invoked either with or without a string argument:

```
mWriteLn MACRO text:=<" ">
    mWrite text
    call Crlf
ENDM
.code
mWriteln "Line one"
mWriteln
mWriteln "Line three"
```

Sample output:

```
Line one

Line three
```

Boolean Expressions

- The assembler permits the following relational operators to be used in constant boolean expressions containing IF and other conditional directives.

▪ LT - Less than	GT - Greater than
▪ EQ - Equal to	NE - Not equal to
▪ LE - Less than or equal to	GE - Greater than or equal to.

IF, ELSE, and ENDIF Directives

- The IF directive must be followed by a constant boolean expression.
  - An alternate block of statements can be assembled if the expression is false.

```
IF boolean-expression
   statement-list
[ELSE
   statement-list]
ENDIF
```

```
RealMode = 1            or
RealMode EQU 1          or
RealMode TEXTEQU 1
```

```
IF RealMode EQ 1
  mov ax,@data
  mov ds,ax
ENDIF
```

The IFIDN and IFIDNI Directives

- IFIDN compares two symbols and returns true if they are equal (case-sensitive)
- IFIDNI also compares two symbols, using a case-insensitive comparison
- Syntax:

```
IFIDNI <symbol>, <symbol>
   statements
ENDIF
```

- An Example :

```
mReadBuf MACRO bufferPtr, maxChars
  IFIDNI <maxChars>,<EDX>
    ECHO Warning: Second argument cannot be EDX
    ECHO **********************************
    EXITM
  ENDIF
  .
  .
ENDM
```

Special Operators

- Substitution (&): resolves ambiguous references to parameter names within a macro.

```
ShowRegister MACRO regName
.data
tempStr BYTE " &regName=",0
    . . .
.code
ShowRegister EDX   ; invoke the macro
```

```
tempStr BYTE " EDX=",0
```

- Expansion (%): expands text macros or converts constant expressions into their text representations.

```
mGotoxy MACRO X:REQ, Y:REQ
    push edx
    mov  dh,Y
    mov  dl,X
    call Gotoxy
    pop  edx
ENDM
```

imply required parameters

```
mGotoXY %(5 * 10),%(3 + 4)
The preprocessor generates
the following code:

1 push edx
1 mov dl,50
1 mov dh,7
1 call Gotoxy
1 pop edx
```

Literal-Text (< >)

- Groups one or more characters and symbols into a single text literal

```
mWrite "Line three", 0dh, 0ah    ; Wrong(3 arguments)
mWrite <"Line three", 0dh, 0ah> ; a single argument
```

Literal-Character (!)

- Forces the preprocessor to treat a predefined operator as an ordinary character.

The following declaration prematurely ends the text definition when the first > character is reached.

```
BadYValue TEXTEQU <Warning: Y-coordinate is > 24>
```

The following declaration continues the text definition until the final > character is reached.

```
BadYValue TEXTEQU <Warning: Y-coordinate is !> 24>
```

Macro Functions

- Returns an integer or string constant.
  - The value is returned by the EXITM directive.
- When calling a macro function, the argument(s) must be enclosed in parentheses
  - Example: The IsDefined macro acts as a wrapper for the IFDEF directive.

```
IsDefined MACRO symbol
    IFDEF symbol
       EXITM <-1>   ;; True
    ELSE
       EXITM <0>    ;; False
    ENDIF
ENDM
```

```
IF IsDefined( RealMode )
   mov ax,@data
   mov ds,ax
ENDIF
```

$\Updownarrow$

```
IF RealMode EQ 1
  mov ax,@data
  mov ds,ax
ENDIF
```

Defining Repeat Blocks

- WHILE Directive : repeats a statement block as long as a particular constant expression is true.
- Syntax :

```
WHILE constExpression
    statements
ENDM
```

- Example : Generates Fibonacci integers between 1 and F0000000h at assembly time.

```
.data
val1 = 1
val2 = 1
DWORD val1    ; first two values
DWORD val2
val3 = val1 + val2

WHILE val3 LT 0F0000000h
    DWORD val3
    val1 = val2
    val2 = val3
    val3 = val1 + val2
ENDM
```

```
.data
DWORD 1
DWORD 1
DWORD 2
DWORD 3
DWORD 5
 . . .
```

The values generated by this code
Can be viewed in a listing (.LST) file.

REPEAT Directive

- Repeats a statement block a fixed number of times at **assembly time**
- Syntax:

```
REPEAT constExpression
    statements
ENDM
```

an unsigned constant integer expression, determines the number of repetitions.

- The following code generates 100 integer data definitions in the sequence 10, 20, 30, . . .

```
iVal = 10
REPEAT 100
    DWORD iVal
    iVal = iVal + 10
ENDM
```

FOR Directive

- The FOR directive repeats a statement block by iterating over a comma-delimited list of symbols.
- Each symbol in the list causes one iteration of the loop.
- Syntax:

```
FOR parameter,<arg1,arg2,arg3,...>
   statements
ENDM
```

- Example: A Structure Definition

```
Window STRUCT
  FOR color,<frame,titlebar,background,foreground>
     color DWORD ?
  ENDM
Window ENDS
```

```
Window STRUCT
   frame DWORD ?
   titlebar DWORD ?
   background DWORD ?
   foreground DWORD ?
Window ENDS
```

FORC Directive

- The FORC directive repeats a statement block by iterating over a string of characters.
  - Each character in the string causes one iteration of the loop.
- Syntax:

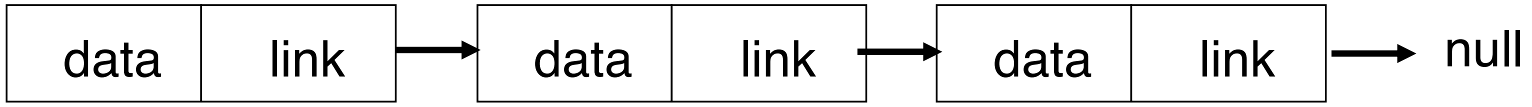
```
FORC parameter, <string>
    statements
ENDM
```

- Example:

```
FORC code,<ABCDEFG>                Group_A WORD ?
    Group_&code WORD ?             Group_B WORD ?
ENDM                               Group_C WORD ?
                             ⇨     Group_D WORD ?
                                   Group_E WORD ?
                                   Group_F WORD ?
                                   Group_G WORD ?
```

Example: Linked List

- We can use the REPEAT directive to create a singly linked list at assembly time.
  - Each node contains a pointer to the next node.
- A null pointer in the last node marks the end of the list



- Structure Definition

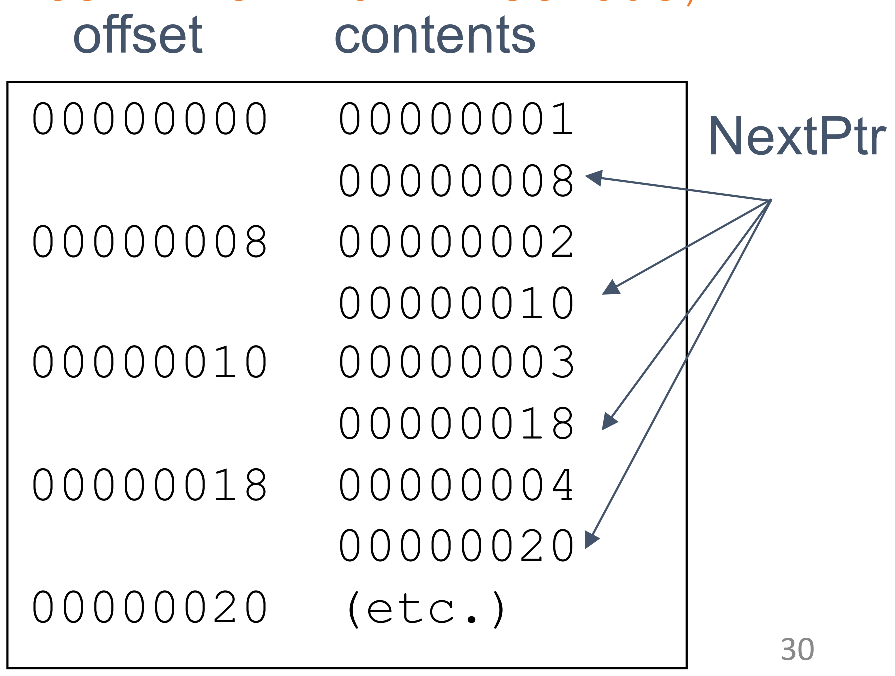
```
ListNode STRUCT
    NodeData DWORD ?    ; the node's data
    NextPtr  DWORD ?    ; pointer to next node
ListNode ENDS

TotalNodeCount = 15
NULL = 0
Counter = 0
```

Example: Linked List (Cont')

- List Generation
  - The REPEAT directive generates the nodes.
  - Each node is initialized with a counter and an address that points 8 bytes beyond the current node's location:

```
.data
LinkedList LABEL PTR ListNode
REPEAT TotalNodeCount
    Counter = Counter + 1
    ListNode <Counter, ($ + Counter * SIZEOF ListNode)>
ENDM
ListNode <0,0> ; tail node
```

The value of $ does not change—it remains fixed at the location of the LinkedList label.



```
00000000   00000001
           00000008
00000008   00000002
           00000010
00000010   00000003
           00000018
00000018   00000004
           00000020
00000020   (etc.)
```