AL Directives / Operators

CO 2103 Assembly Language

Topics

- Source Code Formatting Directives
- Symbolic Constants
- Program Data and Storage
- Naming Storage Locations
- Defining Data (Storage)
- Data-Related Operators and Directives

Source Code Formatting Directives

Directive	Description
`title`	`Title of the listing file`
`org`	`Specify the starting address (first location) in the memory to store the machine codes`
`proc`	`Begin procedure`
`endp`	`End of procedure`
`end`	`End of program assembly`
`.code`	`Mark the beginning of the code segment`
`.data`	`Mark the beginning of the data segment`
`.model`	`Specify the program memory model`
`.stack`	`Set the size of the stack segment`

Symbolic Constants

- Symbolic names associated with storage locations represent addresses
- Named constants are symbols created to represent specific values determined by an expression
- Named constants can be numeric or string
- Some named constants can be redefined
- No storage is allocated for these values
- Assigned using:
  - Equal-sign Directive
  - EQU Directive
  - TEXTEQU Directive

Equal Sign Directive

- name = expression
  - expression must be numeric
  - these symbols may be redefined at any time

```
maxint = 7FFFh
count = 1
DW count
count = count * 2
DW count
```

Equal-Sign Directive (Eg)

```
prod       = 10 * 5      ; Evaluates an expression
maxInt     = 7FFFh       ; Maximum 16-bit signed value
minInt     = 8000h       ; Minimum 16-bit signed value
maxUInt    = 0FFFFh      ; Maximum 16-bit unsigned value
string     = 'XY'        ; Up to two characters allowed
count      = 500
endvalue   = count + 1   ; Can use a predefined symbol

.386
maxLong    = 7FFFFFFFh   ; Maximum 32-bit signed value
minLong    = 80000000h   ; Minimum 32-bit signed value
maxULong   = 0FFFFFFFFh  ; Maximum 32-bit unsigned value
```

Using the Equal-Sign Directive

Statement	Assembled As
count = 5	
mov al, count	mov al, 5
mov dl, al	mov dl, al
count = 10	
mov cx, count	mov cx, 10
mov dx, count	mov dx, 10
count = 2000	
mov ax, count	mov ax, 2000

EQU Directive

- name EQU expression
  - expression can be string or numeric
  - Use < and > to specify a string EQU
  - these symbols cannot be redefined later in the program

```
sample EQU 7Fh
aString EQU <1.234>
message EQU <This is a message>
```

Program Data and Storage

- Pseudo-ops to define data or reserve storage
  - DB - byte(s)
  - DW - word(s)
  - DD - doubleword(s)
  - DQ - quadword(s)
  - DT - tenbyte(s)
- These directives require one or more operands
  - define memory contents
  - specify amount of storage to reserve for run-time data

Naming Storage Locations

- Names (labels) can be associated with storage locations

```
ANum DB -4
  DW 17
ONE
UNO DW 1
X DD ?
```

- These names are called variables

- **ANum** refers to a byte storage location, initialized to **FCh (-4)**
- The next word has no associated name
- **ONE** and **UNO** refer to the same word
- **X** is an unitialized doubleword

Defining Data (Storage)

- Numeric data values
  - 100 - decimal
  - 100B - binary
  - 100H - hexadecimal
  - '100' - ASCII
  - "100" - ASCII
- Use the appropriate DEFINE directive (byte, word, etc.)
- A list of values may be used - the following creates 4 consecutive words

```
DW 40CH,10B,-13,0
```

- A ? represents an uninitialized storage location

```
DB 255,?,-128,'X'
```

Data Allocation Directives

Mnemonic	Description	Bytes	Attribute
DB	Define byte	1	Byte
DW	Define word	2	Word
DD	Define doubleword	4	Doubleword
DF, DP	Define far pointer	6	Far pointer
DQ	Define quadword	8	Quadword
DT	Define tenbytes	10	Tenbyte

Arrays

- Any consecutive storage locations of the same size can be called an array

```
X DW 40CH,10B,-13,0
Y DB 'This is an array'
Z DD -109236, FFFFFFFFH, -1, 100B
```

- Components of **X** are at **X, X+2, X+4, X+8**
- Components of **Y** are at **Y, Y+1, …, Y+15**
- Components of **Z** are at **Z, Z+4, Z+8, Z+12**

Define Byte (DB)

- Create data having a byte attribute
  - one or more bytes may be defined together
  - can mix different data representations
  - can assign a starting value

```
.data
char1     db   'A'
signed1   db   -128
combo     db   'A',-10, 30h, 40q, 1101b
list      db   10,20,30,40
aString   db   "Hello there",0
count     db   ?                  ; uninitialized
```

DUP Operator

- Use DUP to create an array
  - each element is assigned the same value
  - Only used as an operand of a define directive

```
.data
arrayB db 30 dup(10h)    ;30 bytes initialised to 10h
array2 db 10 DUP(?)      ;10 bytes uninitialized
dw 10h dup (0)     ; 16 words initialised to 0
db 3 dup ("ABC")
db 4 dup(3 dup (0,1), 2 dup('$'))
```

Define Word (DW)

- Create data having a word attribute
  - can mix different data representations
  - can use integer expressions
  - can contain 16-bit offset of another variable

```
.data
wordVal   dw   1234h,5000h,65535, -24
signed1   db   35 * 4
aString   dw   "AB"
ptr1      dw   offset wordval
```

Word Storage

- Word, doubleword, and quadword data are stored in reverse byte order (in memory), i.e. Little Endian order

```
Directive     Bytes in Storage
DW 256        00 01
DD 1234567H   67 45 23 01
DQ 10         0A 00 00 00 00 00 00 00
X DW 35DAh    DA 35
```

*Low byte of **x** is at **x**, high byte of **x** is at **x+1***

Define Doubleword (DD)

- Create one or more 32-bit integers
  - doubleword attribute

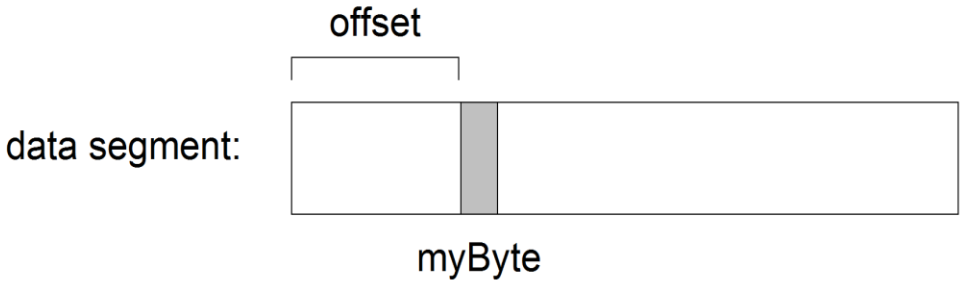
```
.data
largeVal   dd 12345678h
array      dd 100 dup(?)
smallVals dd -1,-2,-3,-4,-5
```

Data-Related Operators and Directives

- OFFSET Operator
- PTR Operator
- TYPE Operator
- LENGTHOF Operator
- SIZEOF Operator
- LABEL Directive

OFFSET Operator

- OFFSET returns the distance in bytes, of a label from the beginning of its enclosing segment
  - Protected mode: 32 bits
  - Real mode: 16 bits



OFFSET Examples

- Let's assume that the data segment begins at 00404000h:

```
.data
bVal BYTE ?
wVal WORD ?
dVal DWORD ?
dVal2 DWORD ?

.code
mov esi,OFFSET bVal  ; ESI = 00404000
mov esi,OFFSET wVal  ; ESI = 00404001
mov esi,OFFSET dVal  ; ESI = 00404003
mov esi,OFFSET dVal2; ESI = 00404007
```

Relating to C/C++

- The value returned by OFFSET is a pointer. Compare the following code written for both C++ and assembly language:

```
; C++ version:
char array[1000];
char * p = &array;
```

```
.data
array BYTE 1000 DUP(?)
.code
mov  esi,OFFSET myArray     ; ESI is p
```

PTR Operator

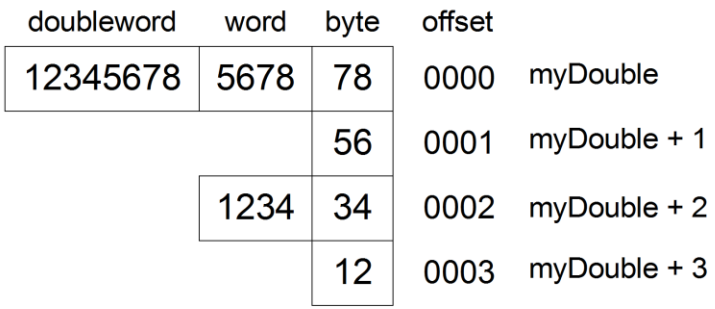
- Overrides the default type of a label (variable)
- Provides the flexibility to access part of a variable

```
.data
myDouble DWORD 12345678h
.code
mov ax,myDouble          ; error – why?
mov ax,WORD PTR myDouble          ; loads 5678h
mov WORD PTR myDouble,4321h       ; saves 4321h
```

Note Little Endian Order of data access

PTR Operator Examples

```
.data
myDouble DWORD 12345678h
```




```
mov al,BYTE PTR myDouble          ; AL = 78h
mov al,BYTE PTR [myDouble+1]      ; AL = 56h
mov al,BYTE PTR [myDouble+2]      ; AL = 34h
mov ax,WORD PTR [myDouble]        ; AX = 5678h
mov ax,WORD PTR [myDouble+2]      ; AX = 1234h
```

PTR Operator (cont)

- PTR can also be used to combine elements of a smaller data type and move them into a larger operand. The CPU will automatically reverse the bytes.

```
.data
myBytes BYTE 12h,34h,56h,78h

.code
mov ax,WORD PTR [myBytes]      ; AX = 3412h
mov ax,WORD PTR [myBytes+2]    ; AX = 5634h
mov eax,DWORD PTR myBytes      ; EAX = 78563412h
```

Your turn . . .

- Write down the value of each destination operand:

```
.data
varB BYTE 65h,31h,02h,05h
varW WORD 6543h,1202h
varD DWORD 12345678h

.code
mov ax,WORD PTR [varB+2]   ; a.
mov bl,BYTE PTR varD; b.
mov bl,BYTE PTR [varW+2]   ; c.
mov ax,WORD PTR [varD+2]   ; d.
mov eax,DWORD PTR varW     ; e.
```

TYPE Operator

- The TYPE operator returns the size, in bytes, of a single element of a data declaration.

```
.data
var1 BYTE ?
var2 WORD ?
var3 DWORD ?
var4 QWORD ?

.code
mov eax,TYPE var1    ; 1
mov eax,TYPE var2    ; 2
mov eax,TYPE var3    ; 4
mov eax,TYPE var4    ; 8
```

LENGTHOF Operator

- The LENGTHOF operator counts the number of elements in a single data declaration.

```
.data                                          LENGTHOF
byte1  BYTE 10,20,30                           ; 3
array1 WORD 30 DUP(?),0,0                      ; 32
array2 WORD 5 DUP(3 DUP(?))                    ; 15
array3 DWORD 1,2,3,4                           ; 4
digitStr BYTE "12345678",0                     ; 9

.code
mov ecx,LENGTHOF array1                        ; 32
```

SIZEOF Operator

- The SIZEOF operator returns a value that is equivalent to multiplying LENGTHOF by TYPE.

```
.data                                          SIZEOF
byte1  BYTE 10,20,30                           ; 3
array1 WORD 30 DUP(?),0,0                      ; 64
array2 WORD 5 DUP(3 DUP(?))                    ; 30
array3 DWORD 1,2,3,4                           ; 16
digitStr BYTE "12345678",0                     ; 9

.code
mov ecx,SIZEOF array1                          ; 64
```

Spanning Multiple Lines (1 of 2)

- A data declaration spans multiple lines if each line (except the last) ends with a comma. The LENGTHOF and SIZEOF operators include all lines belonging to the declaration:

```
.data
array WORD 10,20,
30,40,
50,60

.code
mov eax,LENGTHOF array  ; 6
mov ebx,SIZEOF array    ; 12
```

Spanning Multiple Lines (2 of 2)

- In the following example, array identifies only the first WORD declaration. Compare the values returned by LENGTHOF and SIZEOF here to those in the previous slide:

```
.data
array  WORD 10,20
    WORD 30,40
    WORD 50,60

.code
mov eax,LENGTHOF array  ; 2
mov ebx,SIZEOF array    ; 4
```

LABEL Directive

- Assigns an alternate label name and type to an existing storage location
- LABEL does not allocate any storage of its own
- Removes the need for the PTR operator

```
.data
dwList   LABEL DWORD
wordList LABEL WORD
intList  BYTE 00h,10h,00h,20h
.code
mov eax,dwList   ; 20001000h
mov cx,wordList  ; 1000h
mov dl,intList   ; 00h
```

Summary

- AL directives and operators are used to provide high-level structure/features in AL programs