Duration: 13 Sessions | Undergraduate Program | Winter 2023 | Department: Computer Science

Intro Assembly Lang & Op Sys

Instructor: Dr. Amin Safaei

Winter 2023

Lakehead UNIVERSITY

Duration: 120 min | Undergraduate Program | Fall 2023 | Department: Computer Science

Assembly Language Fundamentals

Instructor: Dr. Amin Safaei

Winter 2023

Lakehead UNIVERSITY

- **This set of lecture slides is made from the following textbooks:**
  - Barry B. Brey, The Intel Microprocessor: Architecture, Programming, and Interfacing, eight edition, Prentice Hall India, 2008.
  - M. A. Mazidi, R. D. McKinlay, J. G. Mazidi, 8051 Microcontroller, The: A Systems Approach
  - S. P. Dandamudi, Introduction to Assembly Language Programming For Pentium and RISC Processors
  - Kip R. Irvine, Assembly Language for x86 Processors (8th Edition)

- **The slides are picked or adapted from the set of slides provided by the following textbooks:**
  - Barry B. Brey, The Intel Microprocessor: Architecture, Programming, and Interfacing, eight edition, Prentice Hall India, 2008.
  - M. A. Mazidi, R. D. McKinlay, J. G. Mazidi, 8051 Microcontroller, The: A Systems Approach
  - S. P. Dandamudi, Introduction to Assembly Language Programming For Pentium and RISC Processors
  - Kip R. Irvine, Assembly Language for x86 Processors (8th Edition)

Duration: 120 min | Undergraduate Program | Fall 2023 | Department: Computer Science

Assembly Language Fundamentals

Instructor: Dr. Amin Safaei

Winter 2023

Lakehead UNIVERSITY

3.1 Overview

- Basic Elements of Assembly Language
- Example: Adding and Subtracting Integers
- Assembling, Linking, and Running Programs
- Defining Data
- Symbolic Constants
- 64-Bit Programming

3.1 Overview

- Basic Elements of Assembly Language
- Example: Adding and Subtracting Integers
- Assembling, Linking, and Running Programs
- Defining Data
- Symbolic Constants
- 64-Bit Programming

3.2 Basic Elements of Assembly Language

- Integer constants
- Integer expressions
- Character and string constants
- Reserved words and identifiers
- Directives and instructions
- Labels
- Mnemonics and Operands
- Comments
- Examples

3.3 Integer Constants

- Optional leading + or – sign
- binary, decimal, hexadecimal, or octal digits
- Common radix characters:
  - h – hexadecimal
  - d – decimal
  - b – binary
  - r – encoded real
- Examples: 30d, 6Ah, 42, 1101b
- Hexadecimal beginning with letter: 0A5h

```
26                      ; decimal
26d                     ; decimal
11010011b               ; binary
42q                     ; octal
42o                     ; octal
1Ah                     ; hexadecimal
0A3h                    ; hexadecimal
```

Lakehead UNIVERSITY

3.4 Integer Expressions

- Operators and precedence levels:

Operator	Name	Precedence Level
( )	parentheses	1
+, -	unary plus, minus	2
*, /	multiply, divide	3
MOD	modulus	3
+, -	add, subtract	4

- Examples:

Expression	Value
16 / 5	3
-(3 + 4) * (6 - 1)	-35
-3 + 4 * 6 - 1	20
25 mod 3	1

3.5 Character and String Constants

- Enclose character in single or double quotes
  - 'A', "x"
  - ASCII character = 1 byte
- Enclose strings in single or double quotes
  - "ABC"
  - 'xyz'
  - Each character occupies a single byte
- Embedded quotes:
  - 'Say "Goodnight," Gracie'

3.6 Reserved Words and Identifiers

- Reserved words cannot be used as identifiers
  - Instruction mnemonics, directives, type attributes, operators, predefined symbols
- Identifiers
  - 1-247 characters, including digits
  - not case sensitive
  - first character must be a letter, _, @, ?, or $

3.7 Directives

- Commands that are recognized and acted upon by the assembler
  - Not part of the Intel instruction set
  - Used to declare code, data areas, select memory model, declare procedures, etc.
  - not case sensitive
- Different assemblers have different directives
  - NASM not the same as MASM, for example

```
myVar    DWORD 26
mov      eax,myVar
```

3.7 Directives

- Commands that are recognized and acted upon by the assembler
  - Not part of the Intel instruction set
  - Used to declare code, data areas, select memory model, declare procedures, etc.
  - not case sensitive
- Different assemblers have different directives
  - NASM not the same as MASM, for example

```
myVar    DWORD 26
mov      eax,myVar
```

```
 1: .data                          ; this is the data area
 2: sum DWORD 0                    ; create a variable named sum
 3:
 4: .code                          ; this is the code area
 5: main PROC
 6:   mov eax,5                    ; move 5 to the eax register
 7:   add eax,6                    ; add 6 to the eax register
 8:   mov sum,eax
 9:
10:   INVOKE ExitProcess,0         ; end the program
11: main ENDP
```

3.8 Instructions

- Assembled into machine code by assembler
- Executed at runtime by the CPU
- We use the Intel IA-32 instruction set
- An instruction contains:
  - Label (optional)
  - Mnemonic (required)
  - Operand (depends on the instruction)
  - Comment (optional)

3.9 Labels

- Act as place markers
  - marks the address (offset) of code and data
- Follow identifer rules
- **Data label**
  - must be unique
  - example:myArray (not followed by colon)

```
count DWORD 100
array  DWORD 1024, 2048
       DWORD 4096, 8192
```

- **Code label**
  - target of jump and loop instructions
  - example: L1: (followed by colon)

```
target:
    mov   ax,bx
    ...
    jmp   target
```

```
L1: mov   ax,bx
L2::
```

Lakehead UNIVERSITY

3.10 Mnemonics and Operands

- Instruction Mnemonics
  - memory aid
  - examples: MOV, ADD, SUB, MUL, INC, DEC
- Operands
  - constant
  - constant expression
  - register
  - memory (data label)
- Constants and constant expressions are often called immediate values

```
stc                 ; set Carry flag
inc eax             ; add 1 to EAX
mov count,ebx       ; move EBX to count
imul eax, ebx, 5
```

Lakehead UNIVERSITY

3.11 Comments

- Comments are good!
  - explain the program's purpose
  - when it was written, and by whom
  - revision information
  - tricky coding techniques
  - application-specific explanations
- Single-line comments
  - begin with semicolon (;)
- Multi-line comments
  - begin with COMMENT directive and a programmer-chosen character
  - end with the same programmer-chosen character

```
COMMENT !
   This line is a comment.
   This line is also a comment.
!

COMMENT &
   This line is a comment.
   This line is also a comment.
&
```

3.12 Instruction Format Examples

- No operands
  - `stc ; set Carry flag`
- One operand
  - `inc eax ; register`
  - `inc myByte ; memory`
- Two operands
  - `add ebx,ecx ; register, register`
  - `sub myByte,25 ; memory, constant`
  - `add eax,36 * 25 ; register, constant-expression`

3.13 What's Next

- Basic Elements of Assembly Language
- **Example: Adding and Subtracting Integers**
- Assembling, Linking, and Running Programs
- Defining Data
- Symbolic Constants
- 64-Bit Programming

3.14 Example: Adding and Subtracting Integers

```
 1: ; AddTwo.asm - adds two 32-bit integers
 2: ; Chapter 3 example
 3:
 4: .386
 5: .model flat,stdcall
 6: .stack 4096
 7: ExitProcess PROTO,  dwExitCode:DWORD
 8:
 9: .code
10: main PROC
11:   mov  eax, 5      ; move 5 to the eax register
12:   add  eax, 6      ; add 6 to the eax register
13:
14:   INVOKE ExitProcess,0
15: main ENDP
16: END main
```

Showing registers and flags in the debugger:

```
EAX=00030000EBX=7FFDF000 ECX=00000101 EDX=FFFFFFFF
ESI=00000000EDI=00000000 EBP=0012FFF0 ESP=0012FFC4
EIP=00401024EFL=00000206CF=0 SF=0ZF=0OF=0
```

```
call program_1
if ErrorLevel 1 goto FailedLabel
call program_2
if ErrorLevel 1 goto FailedLabel
:SuccessLabel
Echo Great, everything worked!
```

Lakehead UNIVERSITY

3.14 Example: Adding and Subtracting Integers

```
 1: ; AddTwo.asm - adds two 32-bit integers
 2: ; Chapter 3 example
 3:
 4: .386
 5: .model flat,stdcall
 6: .stack 4096
 7: ExitProcess PROTO,  dwExitCode:DWORD
 8:
 9: .code
10: main PROC
11:   mov  eax, 5      ; move 5 to the eax register
12:   add  eax, 6      ; add 6 to the eax register
13:
14:   INVOKE ExitProcess,0
15: main ENDP
16: END main
```

Showing registers and flags in the debugger:

```
EAX=00030000EBX=7FFDF000 ECX=00000101 EDX=FFFFFFFF
ESI=00000000EDI=00000000 EBP=0012FFF0 ESP=0012FFC4
EIP=00401024EFL=00000206CF=0 SF=0ZF=0OF=0
```

```
call program_1
if ErrorLevel 1 goto FailedLabel
call program_2
if ErrorLevel 1 goto FailedLabel
:SuccessLabel
Echo Great, everything worked!
```

Lakehead UNIVERSITY

3.15 Suggested Coding Standards

- **Some approaches to capitalization**
  - Capitalize nothing
  - Capitalize everything
  - Capitalize all reserved words, including instruction mnemonics and register names
  - Capitalize only directives and operators
- **Other suggestions**
  - Descriptive identifier names
  - Spaces surrounding arithmetic operators
  - Blank lines between procedures
- **Indentation and spacing**
  - code and data labels – no indentation
  - executable instructions – indent 4-5 spaces
  - comments: right side of page, aligned vertically
  - 1-3 spaces between instruction and its operands
    - ex: movax,bx
  - 1-2 blank lines between procedures

3.16 Program Template

```
; Program Template (Template.asm)
; Program Description:
; Author:
; Creation Date:
; Revisions:
; Date:Modified by:
.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
.data
; declare variables here
.code
main PROC
 ; write your code here
 INVOKE ExitProcess,0
main ENDP
; (insert additional procedures here)
END main
```

3.13 What's Next

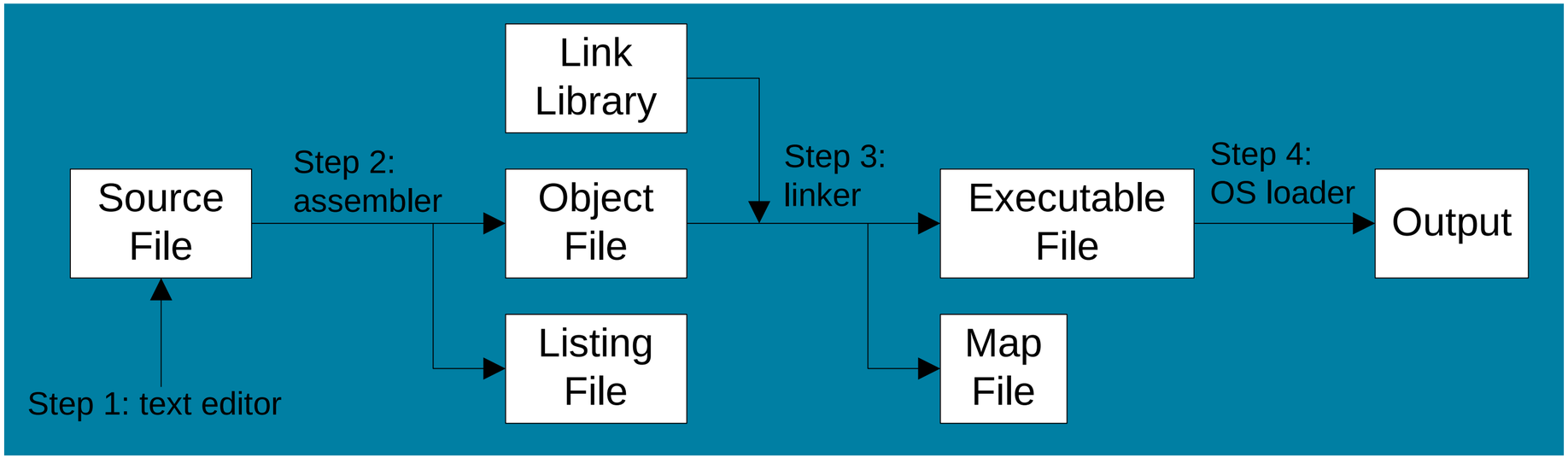
- Basic Elements of Assembly Language
- Example: Adding and Subtracting Integers
- **Assembling, Linking, and Running Programs**
- Defining Data
- Symbolic Constants
- 64-Bit Programming

3.17 Assembling, Linking, and Running Programs

- Assemble-Link-Execute Cycle
- Listing File
- Map File

3.18 Assemble-Link Execute Cycle

- The following diagram describes the steps from creating a source program through executing the compiled program.
- If the source code is modified, Steps 2 through 4 must be repeated.



3.19 Listing File

- Use it to see how your program is compiled
- Contains
  - Source code
  - Addresses
  - Object code (machine language)
  - Segment names
  - Symbols (variables, procedures, and constants)

```
 1:   ; AddTwo.asm - adds two 32-bit integers.
 2:   ; Chapter3 example
 3:
 4:   .386
 5:   .model flat,stdcall
 6:   .stack 4096
 7:   ExitProcess PROTO,dwExitCode:DWORD
 8:
 9:    00000000                          .code
10:    00000000                          main PROC
11:    00000000  B8 00000005                 mov eax,5
12:    00000005  83 C0 06                    add eax,6
13:
14:                                          invoke  ExitProcess,0
15:    00000008  6A 00                       push    +000000000h
16:    0000000A  E8 00000000 E               call    ExitProcess
17:    0000000F                          main ENDP
18:                                      END main
```

Lakehead UNIVERSITY

3.13 What's Next

- Basic Elements of Assembly Language
- Example: Adding and Subtracting Integers
- Assembling, Linking, and Running Programs
- **Defining Data**
- Symbolic Constants
- 64-Bit Programming

3.20 Defining Data

- Intrinsic Data Types
- Data Definition Statement
- Defining BYTE and SBYTE Data
- Defining WORD and SWORD Data
- Defining DWORD and SDWORD Data
- Defining QWORD Data
- Defining TBYTE Data
- Defining Real Number Data
- Little Endian Order
- Adding Variables to the AddSub Program
- Declaring Uninitialized Data

3.21 Intrinsic Data Types

- **BYTE, SBYTE**
  - 8-bit unsigned integer; 8-bit signed integer
- **WORD, SWORD**
  - 16-bit unsigned & signed integer
- **DWORD, SDWORD**
  - 32-bit unsigned & signed integer
- **QWORD**
  - 64-bit integer
- **TBYTE**
  - 80-bit integer
- **REAL4**
  - 4-byte IEEE short real
- **REAL8**
  - 8-byte IEEE long real
- **REAL10**
  - 10-byte IEEE extended real

3.22 Data Definition Statement

- A data definition statement sets aside storage in memory for a variable.
- May optionally assign a name (label) to the data
- Syntax:
  - **[name] directive initializer [,initializer] . . .**
  - value1 BYTE 10
  - All initializers become binary data in memory

3.22 Defining BYTE and SBYTE Data

- Each of the following defines a single byte of storage:
  - 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
  - value6 BYTE ? ; uninitialized byte

- MASM does not prevent you from initializing a BYTE with a negative value, but it's considered poor style.
- If you declare a SBYTE variable, the Microsoft debugger will automatically display its value in decimal with a leading sign.

3.23 Defining Byte Arrays

- Examples that use multiple initializers:
  - list1 BYTE 10,20,30,40
  - list2 BYTE 10,20,30,40
  - BYTE 50,60,70,80
  - BYTE 81,82,83,84
  - list3 BYTE ?,32,41h,00100010b
  - list4 BYTE 0Ah,20h,‘A’,22h

3.24 Defining Strings

- A string is implemented as an array of characters
  - For convenience, it is usually enclosed in quotation marks
  - It often will be null-terminated
- Examples:
  - str1 BYTE "Enter your name",0
  - str2 BYTE 'Error: halting program',0
  - str3 BYTE 'A','E','I','O','U'
  - greeting BYTE "Welcome to the Encryption Demo
  - program "
  - BYTE "created by Kip Irvine.",0

3.24 Defining Strings

- To continue a single string across multiple lines, end each line with a comma:

```
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
```

- End-of-line character sequence:
  - 0Dh = carriage return
  - 0Ah = line feed

```
str1 BYTE "Enter your name: ",0Dh,0Ah
BYTE "Enter your address: ",0
newLine BYTE 0Dh,0Ah,0
```

Lakehead UNIVERSITY

3.25 Using the DUP Operator

- Use DUP to allocate (create space for) an array or string.

  Syntax: counter DUP (argument)

- Counter and argument must be constants or constant expressions

```
var1 BYTE 20 DUP(0)         ; 20 bytes, all equal to zero
var2 BYTE 20 DUP(?)         ; 20 bytes, uninitialized
var3 BYTE 4 DUP("STACK")    ; 20 bytes:
"STACKSTACKSTACKSTACK"
var4 BYTE 10,3 DUP(0),20    ; 5 bytes
```

3.26 Defining WORD and SWORD Data

- Define storage for 16-bit integers
  - or double characters
  - single value or multiple values

```
word1WORD65535        ; largest unsigned value
word2SWORD -32768     ; smallest signed value
word3WORD?            ; uninitialized, unsigned
word4WORD"AB"         ; double characters
myList WORD1,2,3,4,5  ; array of words
arrayWORD5 DUP(?)     ; uninitialized array
```

3.27 Defining DWORD and SDWORD Data

- Storage definitions for signed and unsigned 32-bit integers:

```
val1 DWORD12345678h              ; unsigned
val2 SDWORD –2147483648          ; signed
val3 DWORD20 DUP(?)              ; unsigned array
val4 SDWORD –3,–2,–1,0,1         ; signed array
```

3.28 Defining QWORD, TBYTE, Real Data

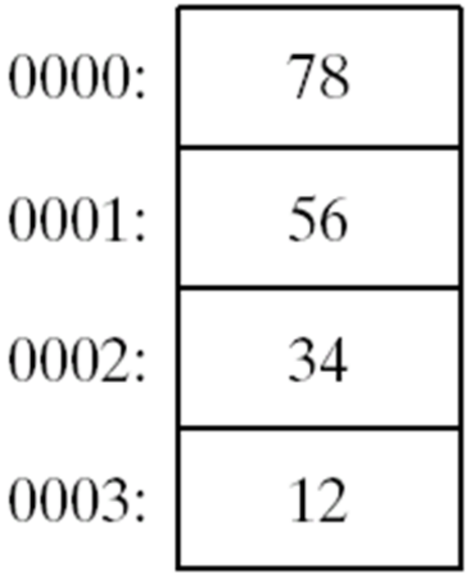
- Storage definitions for quadwords, tenbyte values, and real numbers:

```
quad1 QWORD1234567812345678h
val1TBYTE 100000000123456789Ah
rVal1 REAL4    -2.1
rVal2 REAL8 3.2E-260
rVal3 REAL10 4.6E+4096
ShortArray REAL4 20 DUP(0.0)
```

3.29 Little Endian Order

- All data types larger than a byte store their individual bytes in reverse order.
- The least significant byte occurs at the first (lowest) memory address.
- Example:

```
val1 DWORD 12345678h
```



3.30 Adding Variables to AddSub

```
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
```

3.31 Declaring Unitialized Data

- Use the .data? directive to declare an unintialized data segment:

```
.data?
```

- Within the segment, declare variables with "?" initializers:

```
smallArray DWORD 10 DUP(?)
```

- **Advantage: the program's EXE file size is reduced.**

3.13 What's Next

- Basic Elements of Assembly Language
- Example: Adding and Subtracting Integers
- Assembling, Linking, and Running Programs
- Defining Data
- **Symbolic Constants**
- 64-Bit Programming

3.32 Symbolic Constants

- Equal-Sign Directive
- Calculating the Sizes of Arrays and Strings
- EQU Directive
- TEXTEQU Directive

3.33 Equal-Sign Directive

- ***name = expression***
- expression is a 32-bit integer (expression or constant)
- may be redefined
- name is called a **symbolic constant**
- good programming style to use symbols

```
COUNT = 500
.
.
mov ax,COUNT
```

3.34 Calculating the Size of a Byte/Word/Doubleword Array

- Current location counter: $
  - subtract address of list
  - difference is the number of bytes

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

- Divide total number of bytes by 2 (the size of a word)

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

- Divide total number of bytes by 4 (the size of a doubleword)

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

3.35 EQU Directive

- Define a symbol as either an integer or text expression.
- Cannot be redefined

```
PI EQU <3.1416>
pressKey EQU <"Press any key to continue...",0>
.data
prompt BYTE pressKey
```

3.37 TEXTEQU Directive

- Define a symbol as either an integer or text expression.
- Called a text macro
- Can be redefined

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

3.13 What's Next

- Basic Elements of Assembly Language
- Example: Adding and Subtracting Integers
- Assembling, Linking, and Running Programs
- Defining Data
- Symbolic Constants
- **64-Bit Programming**

3.38 64-Bit Programming

- MASM supports 64-bit programming, although the following directives are not permitted:
  - `INVOKE, ADDR, .model, .386, .stack`
- 64-Bit Version of AddTwoSum

```
; AddTwoSum_64.asm - Chapter 3 example.
ExitProcess PROTO
.data
sum DWORD 0
.code
mainPROC
moveax,5
addeax,6
movsum,eax

movecx,0
call ExitProcess
main ENDP
END
```

- The following lines are not needed:

```
.386
.model flat,stdcall
.stack 4096
```

- INVOKE is not supported.
- CALL instruction cannot receive arguments
- Use 64-bit registers when possible

Lakehead UNIVERSITY

3.39 Summary

- Integer expression, character constant
- directive – interpreted by the assembler
- instruction – executes at runtime
- code, data, and stack segments
- source, listing, object, map, executable files
- Data definition directives:
  - `BYTE, SBYTE, WORD, SWORD, DWORD, SDWORD, QWORD, TBYTE, REAL4, REAL8, and REAL10`
  - `DUP operator, location counter ($)`
- Symbolic constant
  - `EQU and TEXTEQU`