3. Assembly Language Fundamentals

Thursday, August 15, 2024 12:26 PM

Ch. 3 Assembly Lang. Fundamentals

* Directives - command embedded in source code acted upon by assembler

  Examples: .386, .model, .data, .code, .stack, $\underbrace{\text{dword}}_{\text{data type}}$

  - **stack directive** - specifies size of stack
  - **data directive** - splits program into data segment, where we define variables
  - **code directive** - segment where you place executable instructions

* Instruction - short mnemonics that describe the operation to be performed

  Examples: mov, add, sub, mul, jmp, call

* Operands - value used for input or output of an instruction

  Examples:

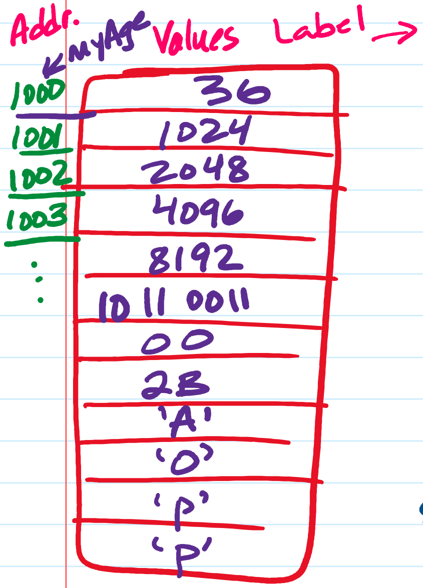
$$\text{mov}\ \underset{\substack{\uparrow\\ \text{destination operand}}}{\underline{\text{eax}}}\ ,\ \underset{\substack{\uparrow\\ \text{source operand}}}{\underline{5}}$$

$$\text{inc}\ \underset{\substack{\uparrow\\ \text{destination operand}}}{\underline{\text{ecx}}}$$

* Labels - used for variable names or to jump to a location in code

* Labels used for variable names or to jump to a location in code

Examples:



data type, initial value

```
.data
myAge        byte   36
myArr        byte   1024, 2048
             byte   4096, 8192
someNum      byte   10110011b
someWord     word   2Bh
someChar     byte   'A'
someString   byte   "oppenheimer", 0
```

Template

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

ExitProcess proto, dwExitCode:DWORD

.data

.code

   main proc


        INVOKE  ExitProcess, 0

   main endp

END main
```

.386 directive – 32-bit program that can access 32-bit registers & addresses

- .386 directive – 32-bit program that can access 32-bit registers & addresses

- .model directive – programs memory model
  * stdcall – calling convention for procedures
  * Windows 32-bit services require the stdcall convention to be used.

- .stack directive – sets aside 4096 bytes for storage of runtime stack

Line 5 – Declares a function prototype for the ExitProcess function, which is a standard Windows service

Defining Data

In .data segment

comment ↓

```
value1    byte    'A'    ; 8-bit storage
value2    byte    ?      ; uninitialized
value3    sbyte   -100   ; signed byte
```

* Multiple Initializers

```
list    byte    10, 20, 30, 40
        byte    50, 60, 70, 80
```

address ↖

offset	Value
0000	10
0001	20
0002	30
0003	40

* strings (Array of bytes)

greeting byte "Good morning", 0

*null-terminated string

*DUP operator

*allocate memory for multiple items

```
label1  byte  20 DUP (0)       ; 20 bytes
                                 intialized to 0
label2  byte  30 DUP (?)
label3  byte  4 DUP("STACK")
```

$\hookrightarrow$ "STACKSTACKSTACKSTACK"

(2 bytes)

* WORD creates storage for 16-bit integers

```
word1   word   65535
mylist  word   1, 2, 3, 4, 5
```

offset	value
0000	1
0002	2
0004	3
0006	4
0008	5

(4 bytes)

*DWORD creates storage for 32-bit integers

```
Val1    dword   12345678h
Val2    dword   20 DUP (?)
mylist  dword   10, 20, 30, 40, 50
```

	offset	value
hex →	0000	10

hex →

offset	value
0000	10
0004	20
0008	30
000C	40
0010	50

* Floating point types REAL4, creates 4-bytes of storage for single-precision floating point variable

```
Val1      real4   -1.2
shortArr  real4   20 DUP(0.0)
```

* Little-Endian Order
  * x86 processor stores & retreives data using little-endian order. Least significant byte is stored @ first memory address.

```
someVar dword   12345678h
```

MSB ... LSB

offset	value
0000	78
0001	56
0002	34
0003	12

* Symbolic Constants

* Use to associate an identifier (symbol) with an integer expression or some text
* Do not reserve storage!! (Not variables)
* Replaced @ assembly
* Cannot change @ runtime.

```
COUNT = 50

.code

    mov eax, COUNT
```

equivalent:

```
    mov eax, 50
```

* Current location counter ($)

```
list  byte  10,20,30,40

listSize = ($ - list)
```

	offset	value
list →	0000	10
	0001	20
	0002	30
	0003	40
$ →	0004	

```
List  word  10h,20h,30h,40h
listSize = ($ - List)/2

List  dword  30h, 50h, 20h, 80h
ListSize = ($ - List)/4
```