

Forum:[Hack Forums](#)

Thread: [Trying to learn assembly language , DOS , Linux or Windows ?](#)

Witchcraft33 - April 28, 2018, 6:23 am

Which one has more examples available on the internet ?

My first try ended up in different hello worlds , that is all

Any book suggestions ?

Thanks

Donkey Death Jam - April 28, 2018, 3:20 pm

x86 Assembly has the most resources I believe. There was a nice read I had a while back 'Practical Malware Analysis' which had a chapter/crash-course on assembly, it doesn't go too deep into it but it explains the basics. Regardless it's a great book and was addicting to read which says a lot for someone who hates reading. 'Art of Intel x86 Assembly' is worth the read too. In general though, books will be your main source of learning, and looking at other people's source code. YouTube & other e-learning platforms don't have much on this, and if they do, it generally isn't correct & makes a hard thing to learn ten times harder. Good luck & shoot me a PM if you get stuck!

.kr. - April 28, 2018, 3:49 pm

I write assembly only on microcontroller like Atmel Atmega8. The Documentation are great you'll find all you need on the Atmel website.

Think its a good startpoint to learn the basics

Witchcraft33 - April 28, 2018, 10:22 pm

Thanks for the replies .

OK , I have few options .

I can focus on MS DOS Assembly , Linux Assembly or Windows MASM32 Assembly right now

I collected a few notes from here and there , now it looks like this .

Quote:label mnemonic operand(s) ; comment

- 1.Labels - In an assembly language program, a label is simply a name for an address
 - 2.Data Declarations - DB DW DD DQ DT
 - 3 Opcode - MOV, PUSH , POP , ADD , SUB , OR , AND , XOR , DEC , INC
 - 4.Registers - EAX EBX ECX EDX CS DS SS ES FS GS ESI EDI EIP ESP EBP cf pf af zf sf
tf if df of iopl nt rf vm ac vif vip id
 - 5.Arithmetic operators - add , sub , mul , div , and , or , xor
 - 6.Control flow operators - call ,test , cmp , jmp ,je , jne ,jge ,ja ,jae , jl , jle , jbe , jo , jno je , jg , jl ,
jz , jnz , js , jns
 - 7.Pointers - mov eax, dword [ebx] , mov ax, word [bx] ,mov al, byte [0]
- Quote:General purpose registers

Accumulator - Input/Output , Operations
Base register - Holds address of data
Counter register - Counts , used in loops
Data register - Holds data for output

Segment registers

Code Segment - Holds address of code segment
Data Segment - Holds address of data segment
Stack segment - Holds address of stack segment
Extra segment - Holds address of data segment

Index registers

Source Index - Points the source operand
Destination Index -Points the destination operand

Special purpose registers

Instruction Pointer - Holds the next instruction
Stack Pointer - Points current top of the stack
Base pointer - Base of the top of the stack

Flag registers - Holds current status of the program

Quote:The three basic modes of addressing are –

Register addressing
Immediate addressing
Memory addressing

Register Addressing

In this addressing mode, a register contains the operand. Depending upon the instruction, the register may be the first operand, the second operand or both.

For example,

MOV DX, TAX_RATE ; Register in first operand
MOV COUNT, CX ; Register in second operand
MOV EAX, EBX ; Both the operands are in registers

Immediate Addressing

An immediate operand has a constant value or an expression. When an instruction with two operands uses immediate addressing, the first operand may be a register or memory location, and

the second operand is an immediate constant. The first operand defines the length of the data.

For example,

```
BYTE_VALUE DB 150 ; A byte value is defined
WORD_VALUE DW 300 ; A word value is defined
ADD BYTE_VALUE, 65 ; An immediate operand 65 is added
MOV AX, 45H ; Immediate constant 45H is transferred to AX
```

Direct Memory Addressing

When operands are specified in memory addressing mode, direct access to main memory, usually to the data segment, is required. This way of addressing results in slower processing of data. To locate the exact location of data in memory, we need the segment start address, which is typically found in the DS register and an offset value. This offset value is also called effective address.

In direct addressing mode, the offset value is specified directly as part of the instruction, usually indicated by the variable name. The assembler calculates the offset value and maintains a symbol table, which stores the offset values of all the variables used in the program.

In direct memory addressing, one of the operands refers to a memory location and the other operand references a register.

For example,

```
ADD BYTE_VALUE, DL ; Adds the register in the memory location
MOV BX, WORD_VALUE ; Operand from the memory is added to register
```

Direct-Offset Addressing

This addressing mode uses the arithmetic operators to modify an address. For example, look at the following definitions that define tables of data –

```
BYTE_TABLE DB 14, 15, 22, 45 ; Tables of bytes
WORD_TABLE DW 134, 345, 564, 123 ; Tables of words
```

The following operations access data from the tables in the memory into registers –

```
MOV CL, BYTE_TABLE[2] ; Gets the 3rd element of the BYTE_TABLE
MOV CL, BYTE_TABLE + 2 ; Gets the 3rd element of the BYTE_TABLE
MOV CX, WORD_TABLE[3] ; Gets the 4th element of the WORD_TABLE
MOV CX, WORD_TABLE + 3 ; Gets the 4th element of the WORD_TABLE
```

Indirect Memory Addressing

This addressing mode utilizes the computer's ability of Segment:Offset addressing. Generally, the base registers EBX, EBP (or BX, BP) and the index registers (DI, SI), coded within square brackets for memory references, are used for this purpose.

Indirect addressing is generally used for variables containing several elements like, arrays. Starting address of the array is stored in, say, the EBX register.

The following code snippet shows how to access different elements of the variable.

```
MY_TABLE TIMES 10 DW 0 ; Allocates 10 words (2 bytes) each initialized to 0
MOV EBX, [MY_TABLE]    ; Effective Address of MY_TABLE in EBX
MOV [EBX], 110          ; MY_TABLE[0] = 110
ADD EBX, 2              ; EBX = EBX + 2
MOV [EBX], 123          ; MY_TABLE[1] = 123
```

MS DOS

Code .model small

```
.data
max_temp DB 92h
min_temp DB 52h
avg_temp DB ?
.code
.startup
```

```
mov ax,@data
mov ds,ax
mov al, max_temp
add al, min_temp
mov ah, 00h
adc ah, 00h
mov bl, 02h
div bl
mov avg_temp, al
.exit
end
Linux
```

Code section .text

global _start ;must be declared for using gcc

_start: ;tell linker entry point

```
mov    eax, '3'
sub    eax, '0'
mov    ebx, '4'
sub    ebx, '0'
add    eax, ebx
add    eax, '0'

mov    [sum], eax

mov    ecx, msg
mov    edx, len
mov    ebx, 1      ;file descriptor (stdout)
mov    eax, 4      ;system call number (sys_write)
```

```

int    0x80        ;call kernel

mov    ecx, sum

mov    edx, 1
mov    ebx, 1        ;file descriptor (stdout)
mov    eax, 4        ;system call number (sys_write)
int    0x80        ;call kernel

mov    eax, 1        ;system call number (sys_exit)
int    0x80        ;call kernel

```

section .data

```
msg    db          "The sum is:", 0xA,0xD
```

```
len equ $ - msg
```

```
segment .bss
```

```
sum resb 1
```

MASM32

Code .386

.model flat ,stdcall

OPTION CASEMAP:NONE

include masm32.inc

include kernel32.inc

includelib kernel32.lib

includelib masm32.lib

.DATA

```
MasgName db "Enter the NUmber  ",00ah,00dh
```

.DATA?

```
num db 2 dup (?)
```

.code

start:

```
invoke StdOut ,addr MasgName
```

```
; Get 1st number
```

```
invoke StdIn ,addr num,10
```

```
; eax = length
```

```
mov [num+ eax-2 ], 0 ; Set terminator
```

```
invoke StripLF,ADDR num
```

```
; Now convert ASCIIz number to decimal
```

```
invoke atodw , addr num
```

```
mov ebx, eax ; Move number from eax to ebx
```

```

;
;_____
invoke StdOut ,addr MasgName
; Get 2nd number
invoke StdIn ,addr num,10
; eax = length
mov [num+ eax-2 ], 0 ; Set terminator
invoke StripLF,ADDR num

; Now convert ASCIIz number to decimal
invoke atodw , addr num
; eax is 2nd number

add eax,ebx

invoke dwtoa, eax,addr num
invoke StdOut ,addr num
invoke StdIn ,addr num,10
invoke ExitProcess,0
end start
Not exactly sure where to go from here ?

```

enmafia2 - June 2, 2018, 11:43 am

Start learning what functions do, then try to make little pieces of software, then if you're into it you can use dissassembler and reverse engineer some code :P

KingArthurBoss - September 18, 2018, 4:41 pm

(Apr 28, 2018 03:20 PM)Donkey Death Jam Wrote: [\[CITE\]](#) Good luck & shoot me a PM if you get stuck!

Hey Donkey Death Jam can you list some resources for learning assembly? I see you've already listed The art of intel assembly ? ?

Masm4ever - September 19, 2018, 12:55 am

Install [the Masm32 SDK](#), then study \Masm32\Examples
[David Evans has a short guide that is worth reading.](#)

MrCracker_ - October 7, 2018, 3:39 pm

Learn NASM for linux platform. if you need an Ebook as guide buy or get a "Assembly Language Step by Step".

If you need any suggestion or if you stuck then PM me. I will happy to help you.

eliasta - October 26, 2018, 5:10 am

You should check virtual masm

macheti228 - November 20, 2018, 7:51 am

good information for me men

realsunjester - November 20, 2018, 7:55 am

(Apr 28, 2018 03:20 PM)Donkey Death Jam Wrote: [\[CITE\]](#) x86 Assembly has the most resources I believe. There was a nice read I had a while back 'Practical Malware Analysis' which had a chapter/crash-course on assembly, it doesn't go too deep into it but it explains the basics. Regardless it's a great book and was addicting to read which says a lot for someone who hates reading. 'Art of Intel x86 Assembly' is worth the read too. In general though, books will be your main

source of learning, and looking at other people's source code. YouTube & other e-learning platforms don't have much on this, and if they do, it generally isn't correct & makes a hard thing to learn ten times harder. Good luck & shoot me a PM if you get stuck!

both of that books are badass, i also suggest them.

there are also plenty of undergrads that publish assembly papers/books online, cant go wrong with yale and vatech papers.

TGillb94 - July 13, 2019, 4:39 pm

(Apr 28, 2018 03:20 PM)Donkey Death Jam Wrote: [\[CITE\]](#) x86 Assembly has the most resources I believe. There was a nice read I had a while back 'Practical Malware Analysis' which had a chapter/crash-course on assembly, it doesn't go too deep into it but it explains the basics. Regardless it's a great book and was addicting to read which says a lot for someone who hates reading. 'Art of Intel x86 Assembly' is worth the read too. In general though, books will be your main source of learning, and looking at other people's source code. YouTube & other e-learning platforms don't have much on this, and if they do, it generally isn't correct & makes a hard thing to learn ten times harder. Good luck & shoot me a PM if you get stuck!

Hey, I'm looking to get into Assembly. Would you recommend a certain learning path? I'm comfortable with JS and currently completed a C certification. Loads of information about there, sometimes difficult to parse. Always good to ask a person with more experience :) Let me know

Son of Morning - September 6, 2019, 12:09 am

<https://www.youtube.com/watch?v=VQAKkuLL31g&t=1s> - Has the best tutorial that I've seen. Although it's Linux x64 you can easily convert the code to x86 as he explains later in the tutorial (replacing r with e)

null_byte - September 15, 2019, 1:22 pm

(Sep 6, 2019 12:09 AM)Defrag Wrote: [\[CITE\]](#)
<https://www.youtube.com/watch?v=VQAKkuLL31g&t=1s> - Has the best tutorial that I've seen. Although it's Linux x64 you can easily convert the code to x86 as he explains later in the tutorial (replacing r with e)

Very interesting set of resources, thanks for sharing!