Babystep3

From OSDev Wiki

A look at machine code (opcodes, prefix, etc)

Babystep3: A look at machine code

Tutorial

Previous	Next
Babystep2	Babystep4

```
; nasmw encode.asm -f k

mov cx, 0xFF
times 510-($-$$) db 0
db 0x55
db 0xAA
```

Don't partycopy to disk. Just open this in DEBUG (for Windows, Hexdump will be nice for Linux users)

```
C:\osdev\debug encode.bin
```

Type in 'd' after the '-' to see the binary file. ('?' will give you help, 'q' will quit). You will see something like this:

```
0AE3:0100 B9 FF 00 00 00 00 etc...
```

Look up the opcode for MOV here: http://www.baldwin.cx/386htm/MOV.htm See Section "17.2.2.1 Opcode" here: http://www.baldwin.cx/386htm/s17_02.htm

In other words, there is a unique register number (CX=1) added to the base opcode value 'B8' to give 'B9', which you see in the dump.

But watch what happens when you replace CX with ECX:

```
mov ecx, 0xFF
times 510-($-$$) db 0
db 0x55
db 0xAA
```

```
0AE3:0100 66 B9 FF 00 00 00 00 etc...
```

The '66' is an Operand Size Override Prefix generated by the assembler when there is a discrepancy with the default mode, which when NASM assembles binary files, it is 16-bit. The same thing happens if you use the BITS directive to change the mode, but it differs from the size of the operand:

```
[BITS 32]
   mov cx, 0xFF
   times 510-($-$$) db 0
   db 0x55
   db 0xAA
```

This doesn't actually change the mode of the processor, but it does help it interpret the subsequent bytes.

Addresses

Address encoding is a bit more complicated

```
 mov cx, [temp]

temp db 0x99
   times 510-($-$$) db 0
   db 0x55
   db 0xAA
```

```
0AE3:0100 8B 0E 04 00 99 00 00 00 etc...
```

- '8B' is the opcode
- '0E' is a ModR/M byte which help the opcode interpretation

See Section "17.2.1 ModR/M and SIB Bytes" here: http://www.baldwin.cx/386htm/s17_02.htm

The rules for interpreting this byte, which contains different fields (see Fig. 17-2), but fortunately Table 17-2 makes it easier. Look up '0E' and you will see at the left it says "disp16" which means that the operand will be interpreted as a 16-bit offset.

'04 00' is the 16-bit offset. If you are confused why 0x0004 is backwards, it's because the Intel processor is "little endian". The "little" end of the number comes first.

'99' is of course the value of the byte at 0x0004 (8B is at 0x0000)

Be aware of another prefix called the Address size Override Prefix '67' which the assembler generates when there is a discrepancy just like with '66' above.

This stuff matters for a bunch of reasons, but since we will be making the switch from 16-bit real mode to 32-bit protected mode, our code is going to also change. And being aware of what a dump looks like can prevent a lot of grief.

- This page has been accessed 42,850 times.