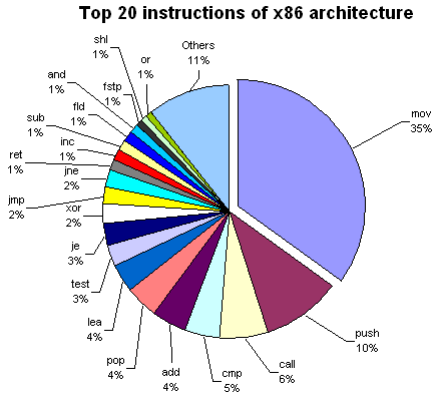
# x86 Machine Code Statistics

Which instruction is the most common one in your code? In this test, **three popular open-source applications** were disassembled and analysed with a Basic script:

- 7-Zip archiver (version 2.30 beta 28, file 7za.exe),
- LAME encoder (version 3.92 MMX, file lame.exe), and
- NSIS installer (version 2.0, file makensis.exe).

All programs were developed with Microsoft Visual C++ 6.0.

## Most frequent instructions




**The most popular instruction** is *MOV* (35% of all instructions). Note that *PUSH* is twice more common than *POP*. These instructions are used in pairs for preserving EBP, ESI, EDI, and EDX registers across function calls, and *PUSH* is also used for passing arguments to functions; that's why it is more frequent. *CALL*s to functions are also very popular.

More than 50% of all code is dedicated to moving things between registers and memory (*MOV*), passing arguments, saving registers (*PUSH, POP*), and calling functions (*CALL*). Only 4th instruction (*CMP*) and the following ones (*ADD, LEA, TEST, XOR*) do actual calculations.

From conditional jumps, *JE* and *JNE* (equal and not equal) are the most popular. *CMP* and *TEST* are commonly used to check conditions. The percentage of the *LEA* instruction is surprisingly high, because MS VC++ compiler generates it for multiplications by constant (e.g., `LEA eax, [eax*4+eax]`) and for additions and subtractions when the result should be saved to another register, e.g.:

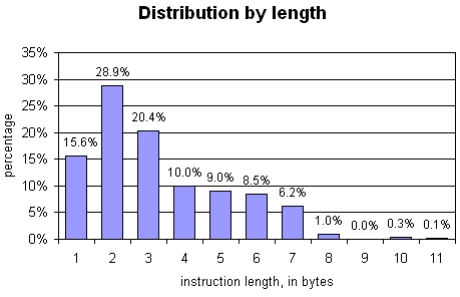
```
LEA eax, [ecx+04]
LEA eax, [ecx+ecx]
```

The compiler also pads the code with harmless forms of LEA (for example, the padding may be `LEA edi, [edi]`). As is easy to see, the top 20 instructions include all logical operations (*AND, XOR, OR*) except *NOT*.

Though LAME encoder uses **MMX** technology instructions, their share in the whole code of the program is very low. Two **FPU instructions** (*FLD* and *FSTP*) appears in the top 20.

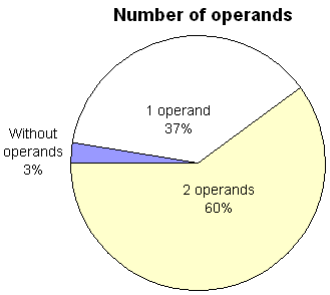
But what about other instructions? It turns out that **multiplication and division are very rare**: *IMUL* takes 0.13%, *IDIV* takes 0.04%, and both *MUL* and *DIV* do 0.02%. Even string operations such as *REPZ SCASB* or *REPZ MOVSB* are more common (0.32%) than all *IMUL*s and *IDIV*s. On the contrary, *FMUL* is more common than *FADD* (0.71% versus 0.27%).

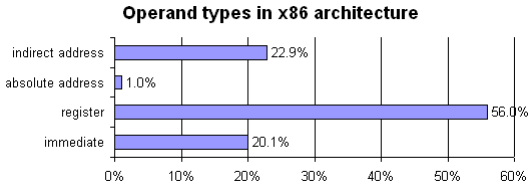
## Average instruction length




75% of x86 instructions are shorter than 4 bytes. But if you multiply the percentage by length, you will find that these short instructions take only 53% of the code size. So another **half of a typical executable file consists of instructions with 32-bit immediate values**, which are 5 bytes or longer.

## The number and type of operands



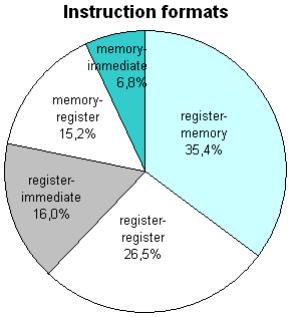




Here are some examples of **operand types**:

- immediate: `00000008, 00401024`;
- register: `eax, esp`;
- absolute address: `dword[00401024], byte[00401024]`;
- indirect address: `dword[esp + 10], dword[00401024 + eax * 4]`;

The parser is fairly limited and operands of the *JMP* and *CALL* instructions are counted as immediate, while in fact they are absolute addresses. Still you can see that **most operands are registers**. Global variables are rare in modern programs.

## Instruction formats




Examples of these instructions:

- register-memory: `ADD eax, [esp + 10]; MOV eax, [00401024]`
- register-register: `ADD eax, ecx`
- register-immediate: `CMP eax, 10`
- memory-register: `MOV [esp + 10], eax; MOV [esi + ecx * 4], eax`
- memory-immediate: `MOV [esp + 10], 0`

## Conclusion

Certainly, some observations are true only for MSVC++ compiler. Other compilers will use other instructions; for example, some of them can't do the trick with *LEA* instruction, and they will use *IMUL* or *MOV/ADD* instead. But you can see several **general trends**: most instructions have 2 operands; memory-register format is less frequent than register-memory; *MOV* is the most popular instruction and so on.

**Download source code (Basic, AWK) and Excel sheet with all data (19 Kb)**

Created 10 years ago by *Peter Kankowski*
Last changed 10 years ago

## 9 comments

### Peter Kankowski, 13 years ago
Here are more comments about this post:
http://board.flatassembler.net/topic.php?t=5249

### Moraaz Code Blog » x86 Machine Code Statistics, 12 years ago
[...] [via] [...]

### Raymond Delord, 9 years ago
I don't think those measures are really relevant.

Because, even if add, and other calculus instructions had really little number of use, they are often used in kinds of loops 'for' or 'while'.

And by using your code only on encoding and installing programs, your results will obviously show the "mov" instructions as the more used.

The relevant measure would be to make statistics on instant code.

### Peter Kankowski, 9 years ago
Thanks for the interesting idea. Gathering statistics at run-time would require much more work. As I can imagine, an x86 emulator would be needed, because performance counters don't provide a counter for each instruction.

### Joe Smirnoff, 8 years ago

Well, Peter, you could just use Valgrind. It's basically a x86 run-time emulator.

### Vladimir Sedach, 7 years ago
Peter, run-time stats is close to your data. See

"Analysis of x86 Instruction Set Usage for DOS/Windows Applications..." by
Ing-Jer Huang and Tzu-Chin Peng

at

http://esl.cse.nsysu.edu.tw/publications/paper/conference/Analysis%20of%20x86%20Instruction%20Set%20Usage%20for%20DOS%20Windows%20Applications%20and%20Its%20Implication%20or

### Peter Kankowski, 7 years ago
Vladimir, many thanks for the link!

### G.Nitz, 6 years ago
That was a very interesting thing I never read about before. Thanks a lot for it.

Mr. G.

### Sirmabus, 6 years ago
I went searching for instruction statistics and luckily your page came out at the top.

I wasn't surprised when I saw it was from your site that I've visited often.

Very useful information, presented in a way easy to grasp with your graphs, etc.

Raymond, your post was years ago but anyhow, your sort of comparing "comparing apples to oranges" here.

Better yet like comparing "Red" to "Gravenstein" (different) apples, as he says says here "..most common one in your code". Which implies a context of the instructions a compiler generates, dynamic as in measuring the qualities of running could is IMHO a distinctly different context.

And incidentally would be hard, if not impossible to trace dynamically in real time. You are talking with an emulator or tracer an exponential slow down that could alter the code flow of the target (as it compensates for the slow down, order of it's message flows are different, etc). Unless perhaps using some sort of ICE setup.

Spasiba moi droog,

Your name:

Comment:

Submit


## Featured pages

Discussion: the first language
Which programming language to learn first?

Hash functions: An empirical comparison
Benchmark program for hash tables and comparison of 15 popular hash functions.

Recommended books and sites
The minimal reading list to become a good programmer.

Software interface design tips
How to design easy-to-use interfaces between modules of your program.

Implementing strcmp, strlen, and strstr using SSE 4.2 instructions
Using new Intel Core i7 instructions to speed up string manipulation.

Table-driven approach
How to make you code shorter and easier to maintain by using arrays.

x86 Machine Code Statistics
Which instructions and addressing modes are used most often. What is the average instruction length.

## Recent comments