Architecture 1001: x86-64 Assembly GotoExample.c

Xeno Kovah – 2021
xeno@darkmentor.com

All materials is licensed under a Creative Commons “Share Alike” license.

- http://creativecommons.org/licenses/by-sa/4.0/

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work
"Is derived from Xeno Kovah's 'Architecture 1001: x86-64 Assembly’ class, available at https://ost.fyi"

Control Flow

- Two forms of control flow
  - Conditional - go somewhere if a condition is met. Think “if”s, switches, loops
  - Unconditional - always go somewhere. Function calls, goto, exceptions, interrupts
- We’ve already seen function calls manifest themselves as call/ret, let’s see how goto manifests itself in asm

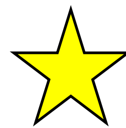
GotoExample.c

```c
#include <stdio.h>
int main(){
    goto mylabel;
    printf("skipped\n");
mylabel:
    printf("goto ftw!\n");
    return 0xb01dface;
}
```

```
main:
0000000140001000  sub   rsp,28h
0000000140001004  jmp   0000000140001012
0000000140001006  lea   rcx,[000000014000600h]
000000014000100D  call  0000000140001090
$mylabel:
0000000140001012  lea   rcx,[000000014000601h]
0000000140001019  call  0000000140001090
000000014000101E  mov   eax,0B01DFACEh
0000000140001023  add   rsp,28h
0000000140001027  ret
```

GotoExample.c

```c
#include <stdio.h>
int main(){
    goto mylabel;
    printf("skipped\n");
mylabel:
    printf("goto ftw!\n");
    return 0xb01dface;
}
```



```
main:
0000000140001000  sub   rsp,28h
0000000140001004  jmp   0000000140001012
0000000140001006  lea   rcx,[000000014006000h]
000000014000100D  call  0000000140001090
$mylabel:
0000000140001012  lea   rcx,[000000014006010h]
0000000140001019  call  0000000140001090
000000014000101E  mov   eax,0B01DFACEh
0000000140001023  add   rsp,28h
0000000140001027  ret
```

JMP - Jump

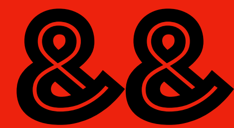
- Unconditionally change RIP to given address
- Ways to specify the address:
  - *Short, relative* (RIP = RIP of next instruction + 1 byte sign-extended-to-64-bits displacement)
    - Frequently used in small loops
    - Some disassemblers will indicate this with a mnemonic by writing it as “jmp short”
    - jmp -2 == infinite loop for short relative jmp :)
    - “jmp 0000000140001012” doesn’t have the number 0000000140001012 anywhere in it, it’s really “jmp 0x0C bytes forward”
  - *Far, absolute indirect* - We'll discuss in future class

JMP - Jump

- Ways to specify the address:
  - *Near, relative* (RIP = RIP of next instruction + 4 byte sign-extended-to-64-bits displacement)
  - *Near, absolute indirect* (address calculated with r/m64)

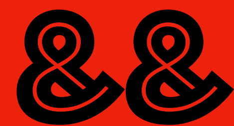
Book page 91

STOP



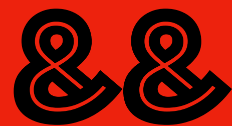
STEP THROUGH

THE ASSEMBLY

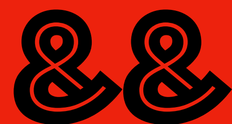


CHECK YOUR UNDERSTANDING

STOP



STEP THROUGH THE ASSEMBLY



CHECK YOUR UNDERSTANDING

GotoExample.c takeaways

- goto == jmp in asm :)

```c
#include <stdio.h>
int main(){
    goto mylabel;
    printf("skipped\n");
mylabel:
    printf("goto ftw!\n");
    return 0xb01dface;
}
```

```
main:
0000000140001000  sub  rsp,28h
0000000140001004  jmp  0000000140001012
0000000140001006  lea  rcx,[0000000140006000h]
000000014000100D  call 0000000140001090
$mylabel:
0000000140001012  lea  rcx,[0000000140006010h]
0000000140001019  call 0000000140001090
000000014000101E  mov  eax,0B01DFACEh
0000000140001023  add  rsp,28h
0000000140001027  ret
```

Instructions we now know (13)

- NOP
- PUSH/POP
- CALL/RET
- MOV
- ADD/SUB
- IMUL
- MOVZX/MOVSX
- LEA
- **JMP**

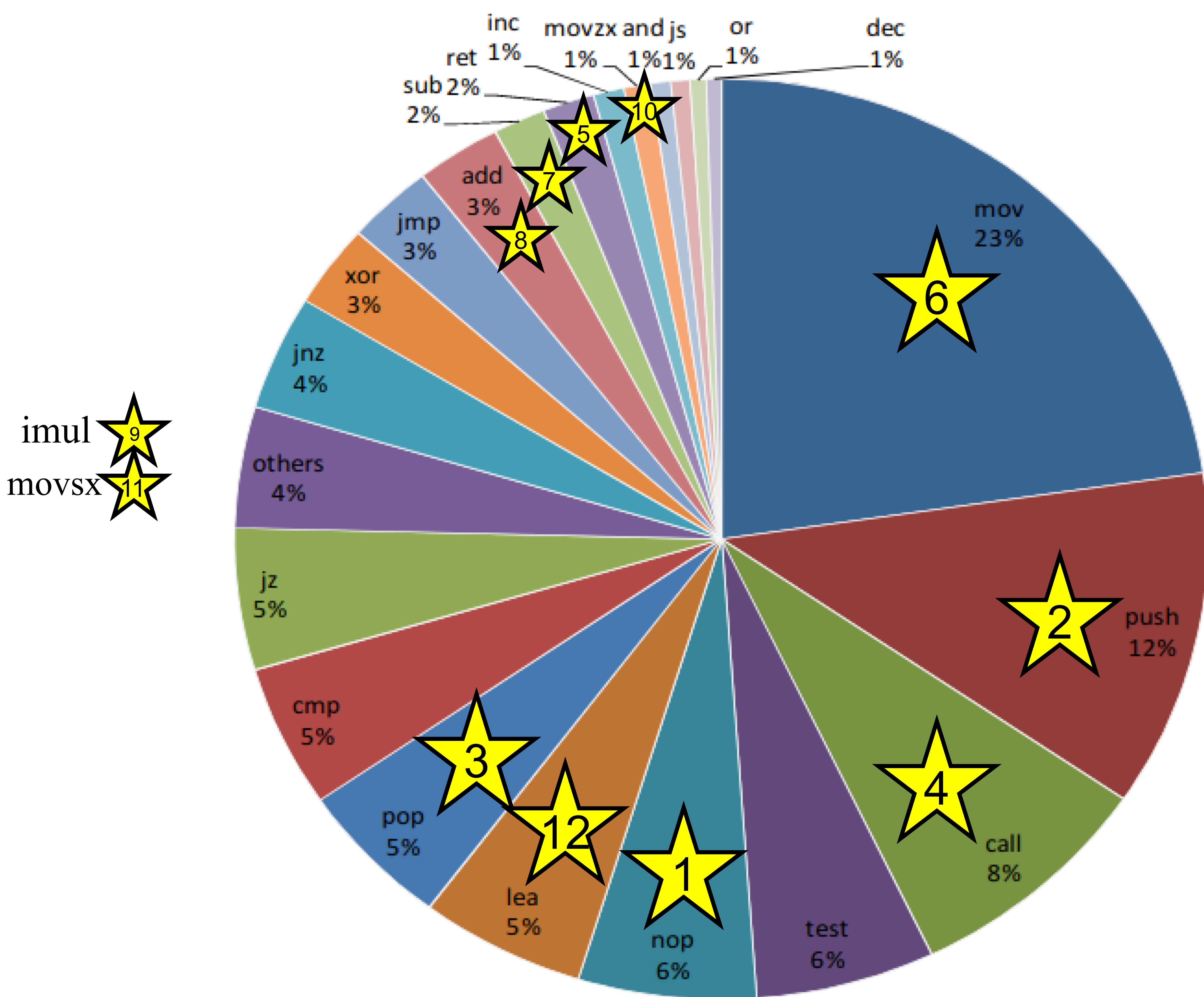


Fig. 2 Web browsers' instruction frequencies

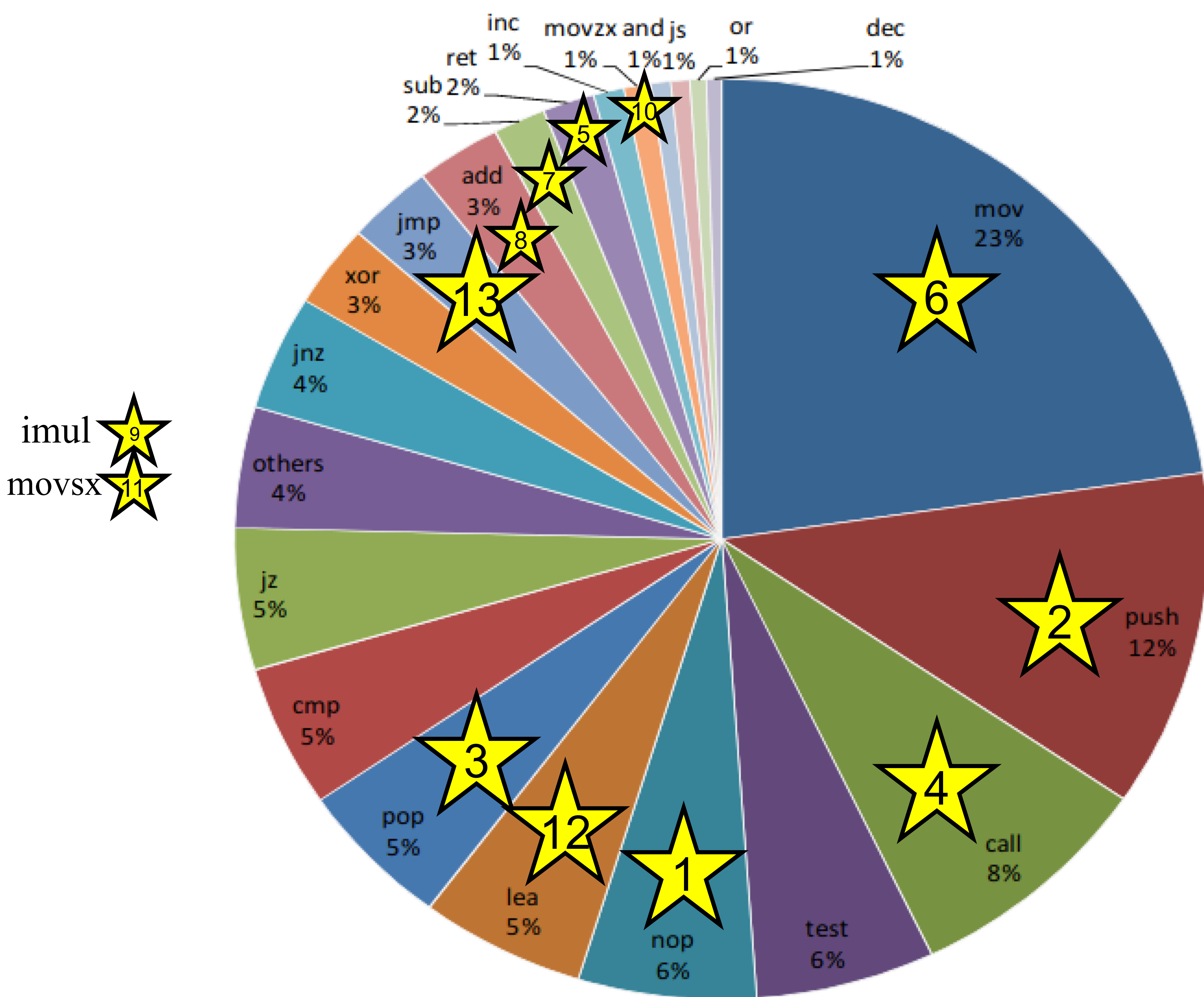


Fig. 2 Web browsers' instruction frequencies