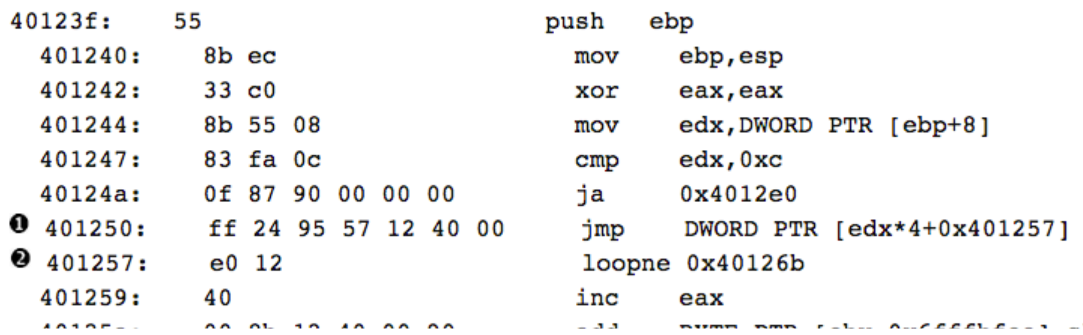
Ben Actis

Linear sweep is the easier out of the two approaches The Hardest decision is where to begin decompiling code. The algorithm simply reads each instruction at t time. For non fixed length architectures (x86), it is also responsible for calculating the length of each instruction. All code sections will be disassembled completely. (Eagle)

The downside of linear sweep is that it can't handle data that is intermixed with instructions. This will produce inaccurate results because each byte is handled, as it was part of an instruction. The example from Ida Pro book highlights this problem perfectly. The bytes at 2 (memory address 401250) should not be interrupted as code. Additionally, linear sweep cannot understand program control flow in non-linear branches. (Eagle)

Example 1-1. Linear sweep disassembly



```
40123f:    55                           push   ebp
  401240:    8b ec                        mov    ebp,esp
  401242:    33 c0                        xor    eax,eax
  401244:    8b 55 08                     mov    edx,DWORD PTR [ebp+8]
  401247:    83 fa 0c                     cmp    edx,0xc
  40124a:    0f 87 90 00 00 00            ja     0x4012e0
❶ 401250:    ff 24 95 57 12 40 00         jmp    DWORD PTR [edx*4+0x401257]
❷ 401257:    e0 12                        loopne 0x40126b
  401259:    40                           inc    eax
```

Recursive decent is the more difficult out of the two approaches, and is the approach that IDA pro uses heavily. After spending weeks on the x86 disassembler assignment, this student understands very well why IDA pro is thousands of dollars. Instructions referenced by another instruction are handled by a linear sweep approach such as sequential flow instructions: mov, push, pop etc. For conditional instructions (jnz), if the result is false linear sweep continues, but if it is true than the branch is taken. (Eagle)

Recursive decedent shines when there are unconditional branching instructions. In the Example 1-1 above, recursive decent would try to calculate the destination not the unconditional jump, and start disassembly there at a later point. Function calls follows this same approach.

This also points to one of recursive decent problems; if the destination of a jump instruction utilizes a runtime value it is not always possible to calculate the correct destination. Similar problems can effect returns from function calls, such as manipulating the return address. (Eagle)

Works Cited

Eagle, C. *The IDA Pro Book.* No Starch Press.