

# Re: General structure of disassembler

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.programming/2006-01/msg01027.html>

---

- *From:* "[jongware]" <jongware@xxxxxxxxxxxxxxxxxxxx>
  - *Date:* Tue, 31 Jan 2006 02:12:57 +0100
- 

"moi" <avk@localhost> wrote in message  
[news:joadneatRcqWOkPenZ2dnUVZ8qKdnZ2d@xxxxxxxxxxxxx](mailto:news:joadneatRcqWOkPenZ2dnUVZ8qKdnZ2d@xxxxxxxxxxxxx)  
> [Jongware] wrote:  
>> Last night I wrote ...

[a lot of random mutterings, then:]

>>>I have experimented both with flags connected directly to the source  
>>>bytes,  
>>>and dumping the whole source into a linked list structure. The pro of  
the  
>>>first option is that display-to-offset is 1-1 -- the con being that (a)  
>>>either it is impossible to insert additional lines, or (b) with a flag  
>>>'additional lines available', each source byte has to be scanned for  
this  
>>>value whenever the cursor moves up or down a few lines. The pro of the  
>  
> Ahh, you are talking user-interface now. The basic problem would be to  
> synchronise 'cursors' (into {code,annotation,display\*N} )  
> I would probably (for sanity!) just keep them as separate 'addressable'  
> domains, and navigate through them separately. (but keep the cursors  
> tied together, or at least 'within their box'.

Spot on. I've implemented a simple viewer and am already running into the  
source bytes<-->meta lines dichotomy.  
The boxes you're talking about would be the smallest data chunk whilst still  
holding a fair amount of bytes and lines.

[...]

> I don't understand what you mean by 'line insertions'. If it is only  
> an annotation thing, it would not be a big deal: either you add 'text'  
> to an existing 'anchor', or you'll have to create an anchor.

That would be a comment line not immediately attached to a source byte.  
E.g., 'regular' comments on source should display to the right of the  
relevant byte, a line insertion would appear \*above\* the line containing the  
source byte, so it doesn't go 1:1 anymore.

[some lines omitted for brevity]

## Re: General structure of disassembler

- > What is this line-thing, again?
- > Is this just about displaying it, than ?

It's the difference between a single line containing one or more source bytes (which should be counted a 'line' in the text editor sense but maps 1:1 to the source) and a comment line (which does NOT map to source bytes). The 'line' as can be seen on screen MAY point to source bytes and MAY have additional text. Note that this causes the discrepancy between 'mapping to source' (eg. go to an absolute offset in the \*source\* code) and 'mapping to screen' (i.e. go to a text line).

Example? Sure.

Raw data bytes, starting at offset 0, are 0xcd, 0x21, 0x00, 0x02, 0x68, 0x69, 0x21, 0x00

User sees 'db 0xcd' on the first line, 'db 0x21' on the next and so on for 8 lines. He can move the cursor up and down and left and right in this text representation. If he moves the cursor to line #0 it comes at the top of the screen. So does it when he moves to absolute source offset #0. Data matches display.

User thinks 'hey, waidaminnut! that's code!' and presses some key. Data bytes on his current line are parsed and create a line such as "INT 21, 0x0002"

.... thereby concatenating 4 bytes into a single line. The display is now  
(ln 0) INT 21, 0x0002

(ln 1) db 0x68

(ln 2) db 0x69 (etc. for the remainder)

and the total number of lines decreases to 5. Line number does not match source offset. But still any line number matches AN address in the source so in this situation it's still a simple map.

It gets worse when the user thinks 'hey, 0x0002 in this newly invented code points to text' and presses a key to convert this to an offset-to-text. The computer rattles, and produces this output:

(ln 0) INT 21, offset txt\_0002

(ln 1) txt\_0002:

(ln 2) ds "hi!", 0 ; offset at 0x0000

There are now only 3 lines: Line #0 still points to source bytes (4 bytes of it), line #1 does not, line #2 again points to 4 source bytes, and to an (ideally autogenerated) comment string.

What I DON'T want is to convert an entire file to text immediately -- I can do that already, and it's boring. The strength (and fun) of manual disassembly comes from the manual part of it; if the user wants to rename the text label to 'string\_Hi' it should also reflect this in all references to it. A good way to do this is on-the-fly creation of the text to be displayed, and for that I'll need an internally consistent data representation, from line number (on screen) to user-added text and/or data bytes and back.

I'm working now on the "text editor" idea in which any line \_may\_ point to a length of data bytes (which may well be 0), and as soon as the sequence of linked text lines gets too long for comfort I'm gonna split it into binary chunks, propagating BOTH the number of lines AND number of data bytes up

## Re: General structure of disassembler

each changed leaf to the root. In this scenario the root leaf ALWAYS holds the total number of lines (great for scrolling to the end), and also ALWAYS holds the total number of source bytes (which should not change). I can also insert consistency checks for debugging, I guess when the leafs don't add up anymore I made a mistake somewhere.

Thanks for the comments so far!

[jongware]

---

### • *References:*

- ◆ *General structure of disassembler*
    - ◇ *From:* [jongware]
  - ◆ *Re: General structure of disassembler*
    - ◇ *From:* [Jongware]
  - ◆ *Re: General structure of disassembler*
    - ◇ *From:* moi
- 
- Prev by Date: *Re: Please use wxWidgets*
  - Next by Date: *Re: Retired programmer wants to learn programming*
  - Previous by thread: *Re: General structure of disassembler*
  - Next by thread: *Source viewer for C++*
  - Index(es):
    - ◆ *Date*
    - ◆ *Thread*