flat assembler

Documentation and tutorials.

flat assembler

Freqently Asked Questions

Here you can find the answers for some of the most common questions about the flat assembler, the ones that were really frequently asked. If you've got some question or problem which is not discussed here, you can look for more help on the message board.

Why do I get `out of memory` error even though I have more than enough memory in my system?

In multitasking environments flat assembler avoids allocating all the possible memory, and thus the memory usage is by default limited to some smaller quantity - in Linux the default limit is 16 MB, in Windows it is all available physical memory and half of the available swap space. If you want to change this limit, use the `-m` option in command line, followed by the number of kilobytes.

Does flat assembler have some directive like `incbin`?

Yes, it is called `file`, and you can use it like any other data definition directives (that means it can be preceded with label without a colon). It also allows you to specify the offset in file and count of bytes you want to include. For more information look into the section 1.2.2 of the manual.

Why the instruction `mov eax,'ABCD'` is assembled into `mov eax,44434241h`? Shouldn't it be reversed?

Altough the most of other assemblers interpret quoted values treating the first character as the most significant, I've decided to use this different approach, just because it's more handy in the most situations. That's because for x86 architecture the least significant byte is the first byte in memory, so if you want to check whether there is `'ABCD'` string at `ebx` address, you can just write `cmp dword [ebx],'ABCD'`.

I'm trying to conditionally define some constant with `equ` directive by putting it in the `if` block, but it seems that even when this condition is false, the constant gets defined. Why?

That's because all symbolic constants and macroinstruction (that means every symbol you define with `equ`, `macro`, or `struc` directive) are processed at the preprocessor stage, while directives like `if` or `repeat` are processed at assembly stage, when all macroinstructions and symbolic constants have already been replaced with corresponding values (generally structures which you have to end with the `end` directive followed by the name of structure are processed at assembly stage). On the other hand, the numerical constants (which you define with `=` symbol) are of the same kind as labels, and therefore are processed at assembly stage, so you can define and use them conditionally.

In COFF and PE formats I can mark section as containing initialized data with the `data` flag in the section declaration. How can I mark section as containing uninitialized data?

Flat assembler marks the section with flag of uninitialized data automatically when the section contains uninitialized data only (this is the data declared with reservation directives, or with `?` values gives to data declarations). When you create such section, you don't have to declare `data` or `code` flag, as it will be marked as uninitialized data automatically.

If I put an `extrn` directive in my code, flat assembler emits the external reference to the object file even if the code doesn't reference the symbol, how can I avoid it?

You can redefine `extrn` as macroinstruction, which will emit the reference only if the symbol is used somewhere, for example:

```
macro extrn symbol
 {
  if used symbol
   extrn symbol
  end if
 }
```

You can also use the `global` macro, which automatically detects whether symbol has to be declared as public or external and this way allows to use the common headers for all the object files of project. It looks like:

```
macro global [symbol]
 {
  local isextrn,isglobal
  if defined symbol & ~ defined isextrn
    public symbol
  else if used symbol & defined isglobal
    extrn symbol
    isextrn = 1
  end if
  isglobal = 1
 }
```

Can I use the compiled resource file instead of macros to build the resource section when creating PE format?

Yes, flat assembler has such feature since the 1.50 release. You can create the resource section from resource file made by any resource compiler or editor, just declare it this way:

```
section '.rsrc' data readable resource from 'my.res'
```

And you don't need to put anything more in such section. In case you don't want the separate section for resource, you can put the resources into any other section with `data` directive:

```
data resource from 'my.res'
end data
```

How to define an array of structures?

Since structure macroinstruction needs a label for each its instance, the structure cannot be simply repeated, since in each repeat structure needs an unique label, otherwise you would get the `symbol already defined` error. One way to overcome this is to

actually create the unique label for each copy of structure, like:

```
StrucArray:
 rept 100
  {
    local s
    s MYSTRUC
  }
```

The other solution can be used when array doesn't have to be initialized and your data structure has the fixed size - like the ones defined with the `struct` macro (the `struct` defines the constant holding the size of structure with the name created by attaching `sizeof.` before the name of structure). Then you can easily calculate how much space the array needs and just reserve this amount of bytes:

```
StrucArray rb 100*sizeof.MYSTRUC
```

Why do I get `invalid use of symbol` error when assembling PE or object file?

This error happens when value of some relocatable label is used in context, in which assembler cannot guarantee that the value encoded in instruction will be correct after relocating the code. The labels are considered relocatable only with output formats that generate relocation information, which are the object formats and PE (but only in case when the fixups data is included into executable by programmer).

Can I link object files or static libraries into executable with flat assembler?

No - this is the job of a linker, not assembler. Though flat assembler is able to output some types of executables directly, it has nothing to do with linking - it just formats the output code to be an executable file instead of object or flat binary, with the general rule that output code is always generated in the same order and way as it is defined by source. But if you prefer to work with many modules and libraries, you should use the object output of flat assembler, and then link it into final executable with some linker.

My anti-virus software reports that package downloaded from this website contains viruses. Are you infecting my computer?

The modern anti-virus programs use sophisticated heuristicts to detect suspicious files, and some of the small programs provided as an examples, that come with flat assembler, are sometimes considered risky by them, or even reported as trojans or viruses. However this is most likely a false alarm. It is recommended that in case of doubt you use some service like Virus Total to check a suspicious file with many different anti-virus engines.