Assembler directives

Assembler directives are all instructions to the assembler that are not actual instructions. Ophis's set of directives follow.

- `.outfile` *filename*: Sets the filename for the output binary if one has not already been set. If no name is ever set, the output will be written to `ophis.bin`.

- `.advance` *address* [, *filler*]: Forces the program counter to be *address*. Unlike the `.org` directive, `.advance` outputs bytes (the value of *filler*, or zeroes if it is unspecified) until the program counter reaches a specified address. Attempting to `.advance` to a point behind the current program counter is an assemble-time error.

- `.alias` *label value*: The .alias directive assigns an arbitrary value to a label. This value may be an arbitrary argument, but cannot reference any label that has not already been defined (this prevents recursive label dependencies).

- `.byte` *arg* [, *arg*, ...]: Specifies a series of arguments, which are evaluated, and strings, which are included as raw ASCII data. The final results of these arguments must be one byte in size. Seperate constants are seperated by comments.

- `.cbmfloat` *string* [, *string*, ...]: Specifies a series of strings, which are interpreted as floating point constants, and then included in the 5-byte floating point format used by the Commodore BASICs. This format is 8 bits of exponent, followed by a sign bit and a 31-bit big-endian mantissa fraction. (The 1 in front of the binary point is presumed to be present.) An exponent of 0 specifies a constant of 0, and the exponent is shifted up by 129 before being stored.

  Because IEEE-754 doesn't perfectly match the Commodore's system, if you wish to precisely replicate individual constants that cannot be represented exactly you may have better luck with the following program, which will run on both the Commodore 64 and VIC-20:

```
10 CLR:V=0:PV=PEEK(45)+256*PEEK(46)+2
20 INPUT "NUMBER (0 TO QUIT)";V
30 IF V=0 THEN END
40 PRINT ".BYTE";
50 FOR I=0 TO 4
60 IF I>0 THEN PRINT CHR$(157);",";
70 PRINT PEEK(PV+I);:NEXT I:PRINT:GOTO 20
```

  This program will print out a `.byte` directive for you to include in your program to represent that number.

- `.checkpc` *address*: Ensures that the program counter is less than or equal to the address specified, and emits an assemble-time error if it is not. *This produces no code in the final binary - it is there to ensure that linking a large amount of data together does not overstep memory boundaries.*

- `.data` *[label]*: Sets the segment to the segment name specified and disallows output. If no label is given, switches to the default data segment.

- `.incbin` *filename* [, *offset* [, *length*]]: Inserts the contents of the file specified as binary data. Use it to include graphics information, precompiled code, or other non-assembler data. You may also optionally specify an index to start including from, or a length to only include a subset.

- `.include` *filename*: Includes the entirety of the file specified at that point in the program. Use this to order your final sources, if you aren't doing it via the command line.

- `.org` *address*: Sets the program counter to the address specified. *This does not emit any code in and of itself, nor does it overwrite anything that previously existed*. If you wish to jump ahead in memory, use `.advance`.

- `.require` *filename*: Includes the entirety of the file specified at that point in the program. Unlike `.include`, however, code included with `.require` will only be inserted once. The `.require` directive is useful for ensuring that certain code libraries are somewhere in the final binary. They are also very useful for guaranteeing that macro libraries are available.

- `.space` *label size*: This directive is used to organize global variables. It defines the label specified to be at the current location of the program counter, and then advances the program counter *size* steps ahead. No actual code is produced. This is equivalent to `label: .org ^+size`.

- `.text` *[label]*: Sets the segment to the segment name specified and allows output. If no label is given, switches to the default text segment.

- `.word` *arg* [, *arg*, ...]: Like `.byte`, but values are all treated as two-byte values and stored low-end first (as is the 6502's wont). Use this to create jump tables (an unadorned label will evaluate to that label's location) or otherwise store 16-bit data.

- `.dword` *arg* [, *arg*, ...]: Like `.word`, but for 32-bit values.

- `.wordbe` *arg* [, *arg*, ...]: Like `.word`, but stores the value in a big-endian format (high byte first).

- `.dwordbe` *arg* [, *arg*, ...]: Like `.dword`, but stores the value high byte first.

- `.scope`: Starts a new scope block. Labels that begin with an underscore are only reachable from within their innermost enclosing `.scope` statement.

- `.scend`: Ends a scope block. Makes the temporary labels defined since the last `.scope` statement unreachable, and permits them to be redefined in a new scope.

- `.macro` *name*: Begins a macro definition block. This is a scope block that can be inlined at arbitrary points with `.invoke`. Arguments to the macro will be bound to temporary labels with names like `_1`, `_2`, etc.

- `.macend`: Ends a macro definition block.

- `.invoke` *label* [*argument* [, *argument* ...]]: invokes (inlines) the specified macro, binding the values of the arguments to the ones the macro definition intends to read. A shorthand for `.invoke` is the name of the macro to invoke, backquoted.

[<<< Previous](#)
Macros

[Home](#)
[Up](#)