Axe 1.0.0

A Brief Overview Of Changes/Features

Drawing

Axe has tons of new support for drawing with many new commands in this area. The most major change is the ability to use almost all of the drawing commands on arbitrary buffers using the last argument. This also comes with the elimination of the previous store arrow notation for arbitrary sprite buffers. The following are now available:

Pt-On(X,Y,Pic,Buff)
Pt-Off(X,Y,Pic,Buff)
Pt-Change(X,Y,Pic,Buff)
Pxl-On(X,Y,Buff)
Pxl-Off(X,Y,Buff)
Pxl-Change(X,Y,Buff)
pxl-Test(X,Y,Buff)
Line(X1,Y1,X2,Y2,Buff)
Rect(X,Y,W,H,Buff)
RectI(X,Y,W,H,Buff)

Another new drawing feature is the new Pt-Mask()^{r}. It functions almost exactly the same as the regular Pt-Mask() but the pixel combination that used to represent gray now represents “Invert” and it only uses a single buffer to draw the sprite instead of two. Also, since clearing both buffers is a common task, I added a useful shorthand for that which is ClrDraw^{rr}.

In addition to drawing commands, all the DispGraph commands can now be used with any buffer combination. ClrDraw optimizations work on grayscale now too, which clears both buffers at the same time as drawing using the wasted cycles that wait for the LCD port.

DispGraph(Buff)
DispGraph(Buff1,Buff2)^{r}
DispGraph(Buff1,Buff2)^{rr}
DispGraphClrDraw(Buff)
DispGraphClrDraw(Buff1,Buff2)^{r}
DispGraphClrDraw(Buff1,Buff2)^{rr}

Named Variables

The symbol name limit has now been extended to 5 characters and lowercase is now allowed for all but the first letter. The symbol limit has also increased to about 2000. You can also create new custom named variables! To do so, you first set the variable's address using:

ConstantExpression→ᵒMyVar

You can then refer to the variable's address later in the code as ᵒMyVar or Use the 16-bit value directly with a simple MyVar.

Labels are dereferenceable too. You can get a label's address with ʟMyLbl.

In addition to the new naming. All program data such as sprites and strings can now be forward declared. However, *constants* must still be declared before they're used. I enforce this restriction because it optimizes your code a lot more optimized.

Functional Programming

The most major of changes comes here. First and foremost, calling subroutines in Axe now has a convenient new syntax. The following are now identical:

sub(LBL,Arg1,Arg2)
LBL(Arg1,Arg2)

Secondly, Axe now allows true functional programming by making functions first class data types. Since labels dereference, you can now call and goto subroutines who's label is a variable. You simply put the label in parenthesis using the new notation, and it works from any left hand parenthesis token too, not just plain ones. sub(A)() will call subroutine A, and then call its return value for example. These two are nearly identical:

LBL(A)
(ʟLBL)(A)

I say nearly because the “HL” value passed to the subroutine when it's variable is going to be the label instead of the last argument.

Thirdly, the awesome new lambda feature is now very useful. It basically allows you to create simple, unnamed, expressions that you can pass to anything that takes a label. The syntax is simply: λ(Expression). The lambda can be accessed with [Log]. For instance:

λ(r_1+r_2)(5,6)

This creates a subroutine that returns the sum of its 2 arguments then calls that subroutine with 5 and 6 as the arguments. This example is useless, but it demonstrates the syntax. Suppose you wrote Map:

Lbl Map
$r_1 \rightarrow r_4$
For(A,0,r_3-1)
(r_4)({r_2+A})$\rightarrow${r_2+A}
End
Return r_2

Map takes 3 arguments; A subroutine, a data pointer, and the size of the data. It will map the expression to each byte in the data. Lambda expressions are really useful here:

Map(λ(r_1+1),Data(1,2,3),3) Increment each element
Map(λ(r_1*2),Data(1,2,3),3) Double each element
Map(λ(r_1^2),Data(1,2,3),3) Square each element
etc.

Colons have also changed to be inline operators in order to make lambdas more powerful, so you can chain a bunch of expressions together. In addition, the ternary operator is finally complete to allow inline if statements. The syntax is a Expression?TrueResult,FalseResult. Surrounding parenthesis are not needed, but recommended. Here is an expression that computes a maximum:

λ(r_1<r_2?r_2,r_1)

Parser Features

The parser now gives a warning at the end of parsing when the program had code (not data) past the $C000 boundary. This can be ignored if you are using Crabcake or similar. In addition, all data is put at the end of the program instead of being mixed with subroutines, so many programs previously over the limit may now work.

All error messages are now slightly longer and slightly more descriptive to make debugging easier.

Axioms can now do replacements with offset addresses. Similar to how normal offsets prefix with “ld a,a” the new offset replacements prefix with “ld b,b \ .db Offset”. Offset is unsigned so you can pick any address within the first 255 bytes of a routine or data.

Optimizations

There are huge new optimizations. I lost track of how many routines I optimized, but besides actual routines, there are 2 major new ones.

The first is that all jumps “upward” within range will automatically convert to relative jumps instead of absolute ones. This affects goto's and all loop structures, but does not affect if statements.

The second is the much cooler peephole opts. A lot of times individual commands cannot get optimized any further, but 2 specific commands that come right after each other can combine into a single more optimized version. One example is the statement “If {A}”. Normally, it would get the byte at a into hl and then check if the value is zero. However, we already know that the high byte H is zero so only L needs to be checked.

The third optimization I've added is that using a referenced constant pointer for any math operation will properly optimize as if it were a variable. Even the single byte and big endian forms optimize here as well.

Combined, these new optimizations reduce the size of all code by about 2% on average, which is quite significant. Unfortunately parsing speed is maybe 30% slower, but I feel its defiantly worth it.

Another optimization I should mention is that I've auto-opted the 8.8 multiplication with constants.

Miscellaneous

There is a new notation for Fill (the old one still exists). It has 3 arguments instead of 2 and takes the following form:

Fill(Data,Size,Value)

Size here is the actual size of the data block unlike the 2 argument version.

For loops now have a super optimized form when you need a piece of code to execute a fixed number of times. The syntax is just For with one argument; the number of times to execute the code block. One other cool thing about it is that results can pass in and out of the loop block unaltered since it doesn't use HL. Also, goto's are prohibited in this structure.

Another new command is cumSum(Data,Size) which finds the simple 16-bit arithmetic check-sum of a given block data.

Returnʳ is also new, it allows programs to instant-quit by using the error handler's return point.

Pi “π” is now the binary prefix instead of “b”

You can type in fixed point numbers directly using standard decimal notation. For instance, 12.25 becomes ᴇ0C40 which is the fixed point representation. You must have at least one digit in the one's place or else it will parse as a comment: .5 is a comment, 0.5 is a fixed point number. You can only have up to 3 digits after the decimal, but that's more than the maximum precision anyway, so that shouldn't be a big deal. Hopefully, this will make fixed point numbers more readable.