**High Level Assembler**

This document describes a High Level Assembler implementation for the Spectrum ZXNext.

Generally, case is ignored, except in quoted strings, and comments are double forward slashes which cause everything after that to be ignored on the same line.

Expressions

Expressions are made up of an alternating sequence of terms and operators. Everything is 16 bit unsigned. (Negative used later means depends on bit 15)

A term is either:

1. A numeric constant in decimal - 3764
2. A string, which is a constant which is the address of an ASCIIZ implementation of the string.
3. An identifier. Identifiers begin with $ or a-z, and contain the characters a-z and 0-9. Identifiers beginning with $ are globals.
4. In two special cases, an identifier followed by a term in square brackets (e.g. fred[4]). The meaning of this is the memory location whose address is the value in fred + 4. This identifier can be used as the *first* term in an expression or the target of the assignment operator ‘>’

Operators are:

+ - \* / 4 standard operators

% Modulus

& | ^ Bitwise And,Or,Xor.

> Assignment operator

So the Pascal equivalent of a := b + 4, for example is b+4>a.

The > does not necessarily end the expression, you can write 0>a>b>c if you wish.

> causes creation by assignment, so ><identifier> will cause a global or local variable to be created according to the name. This does not apply to >g[term].

Language elements

These elements are loosely checked with a structure stack. There can be multiple elements on a line separated by colons.

*proc <identifier>(p1,p2,p3,p4)*

Define a procedure with up to four parameters. These are received (in order) in HL, DE, BC, IX.

*endproc*

Returns from the procedure. There can be multiple exit points if you really want. Values are returned by convention in $return

*if (<condition>) ….. endif*

Conditional execution. A condition is an expression ending in either #0,=0 or <0 which is used to determine the test.

*while (condition) …. endwhile*

Conditional loop, executes while contents are true.

*for (<expression>) …. next*

Repeats the code <expression> times. If a variable “index” exists then the loop value is copied into it at the *top* of the loop.

*<identifier>(term,term,term,term)*

Invokes a procedure, putting term parameters into HL, DE, BC and IX. The parameters are not checked. A global $return is available for return values.

*<expression>*

An expression is a statement in its own right, and is normally used for assignment via the > operator.

Scope

Global variables scope is the whole application space. Local variables scope is the procedure. Local procedures scope is the local source file.

Note that procedure parameters and locals are all *static* so you cannot recurse in this language using the local variables at present.

Processing

* Remove all comments, tabs, strip spaces
* Remove all quoted strings, replace by constant addresses.
* Convert all hexadecimal constants to decimal constants.
* Remove all spaces, join together with :~:
* Split by procedure.
* For each procedure …..
* Allocate a local for each parameter in the header.
* Scan for globals and locals in the body, and replace appropriately with @<constant>, allocating memory as appropriate.
* Compile the procedure header saving HL DE BC IX as appropriate
* Compile the procedure body.

Issues

In the dictionary, procedures are stored with a following (, as a way of determining the scope changes. Scope is driven by the $ prefix but it is treated differently.

Add hexadecimal constants.

Static memory allocation ?

Passing variable addresses (@xxxx, when @@address it’s just address ?)