Tunguska Manual 0.5

Viktor Lofgren <vlofgren@gmail.com>

http://www.acc.umu.se/~achtt315/tunguska/

2008-11-14

Contents

Chapter 1

Introduction

1.1 The fine print

1.2 Foreword

This is a multi-part document covering the use of the Tunguska Emulator, the Tunguska Assembler, the Tunguska C compiler 3CC, a brief introduction to Ternary Computing, and the Specifications of Tunguska.

Tunguska is by no means affiliated with any organization, religion, political movement, nationality, sports team, text editor or pizza topping; mentioned, insinuated, or otherwise referenced. It is non-profit, non-affiliated, and largely only exists for the fun of it.

If you do find some horrible crime against mathematics or grammar in here, first make sure this is the latest version of the document by visiting the Tunguska website, and if it indeed is the latest version you've found the error in, feel free to send me an email with a suggested correction.

Part I

The software

Chapter 2

The emulator

2.1 Changes

From 0.0.3.x to 0.4, the most important change is the new assembler directive %INC. What this means is that instead of telling the assembler to assemble *all files*, you just tell it to assemble ram.asm, and it figures out where to include what automatically.

From 0.4.x to 0.5, the main change is the inclusion of 3CC, the (non-standard) C compiler for tunguska.

2.2 Getting started

To compile and install Tunguska, simply do the following

```
> cd tunguska-version
> ./configure
> make
> make install
```

To be of any sort of use, Tunguska needs a memory image to load. Starting tunguska without one would be the equiavalent of booting your computer without any sort of hard drive. Don't worry, you don't need to illegally download some proprietary image off some murky backwater website. All you need to get started ships with Tunguska. After compiling and installing Tunguska itself, you need to assemble an image, and this is how you do it.

2.2.1 The old ASM image

This memory image is old and proven, dating back to ye olden days of Tunguska's creation. It is however written entirely in assembly, and is less accessible than the C based image described in the next section. The ASM image has more features though, and as such, serves as a nice demo of Tunguska's functionality.

```
> tar zxvf tunguska-version/share/memory_image_asm.gz[1]
> cd memory_image_asm
> tg_assembler -o image.ternobj ram.asm
```

You're now ready to load the image with Tunguska

```
> tunguska image.ternobj
```

You can also assemble disk images, and load them with tunguska. To do this, you need to find the source code for some such image, I suggest you download the pong clone from Tunguska's website *http://www.acc.umu.se/~achtt315/tunguska/* and put it in the same directory as the other .asm-files. You're now ready to assemble

```
> tg_assembler -o pong.ternobj pong.asm
```

[1] This archive is also installed in /usr/local/share/tunguska (or someplace similar, if you changed the prefix.)

And to run it in tunguska, start with the system image

```
> tunguska image.ternobj
```

From the tunguska prompt type

```
> LOAD
ENTER IMAGE TO LOAD=pong.ternobj
DISK LOADED
```

Now, from the tunguska prompt type

```
> RUN
```

2.2.2 The new 3CC image

The 0.5 tunguska release and later also comes with a smaller memory image written in C, here is how to compile and run that:

```
> tar zxvf tunguska-version/share/memory_image_3cc.gz
> cd memory_image_3cc
> make
```

You load it in tunguska by running

```
> tunguska image.ternobj
```

And you're done! That's all there is to it. Now you'll probably want to dive in and make your own programs. Good for you.

The most helpful tip on that (except reading this document) is to look up a 6502-assembly reference or guide on the internet. Tunguska is compatible enough for most of the stuff you'll find there to be applicable. If something doesn't work as expected, head over to **Appendix A** and check so the instruction is still there (some instructions have changed names, some have been dropped).

2.3 Command line arguments

Tunguska takes the following command line arguments:

-h	Display help message
-f	Full screen
-F	Specify floppy image
-Z	Don't attempt to actually load the floppy, assume it's zero.

2.4 The interface

The border around the window indicates the size of the raster window (bright) and the text/vector window(dark).

2.4.1 Special keys

Esc	Exit
F9	Enable trace mode, spew the location of the program counter to standard output.
F10	Debug, print all registers to standard output.
F11	Step instruction (when in pause mode).
F12	Send interrupt
Pause	Toggle pause

2.4.2 Indicator lights

Top-left of the Tunguska window is three indicator ligths, labelled R, G, AG. They correspond to

R	Running when lit, paused when not
G	Graphics mode (see specifications)
AG	Auxiliary graphics mode (see specifications)

Chapter 3

The C compiler, 3CC

3CC is tunguska's c-compiler. It is not compatible with any standard of C, and never will be. It is also significantly less mature than tunguska in general, since it's parallel development started much later than tunguska did. Caveat user.

3.1 Standard

The following is a list of portions of 3CC that differ from standard C. Some of these deviations are bugs that will be fixed in future versions, some are permanent features.

3.1.1 Datatypes

3CC has only two basic datatypes: **char** and **int**, and they are both signed. In fact, unsigned datatypes do not exist due to the base of tunguska. There are no **long** or **short** types or modifiers. Only integral types are supported, there is no **float** or **double** types or modifiers. **struct**s are in, and work as should be expected. **Arrays** almost work properly, though the array code is somewhat unstable[bug]. Array declarations mostly work as expected in regular C. Global variables can be **extern**, but not functions. The whole notion is redundant since there is no intermediate object stage. In 3CC, compiling multiple files is indistinguishable from tacking all of the code together into a single file. The following construct **does not work**:

Type	Size	Max	Min
char	1	364	-364
int	2	265,720	-265,720
type*	2	265,720	-265,720

Table 3.2: List of datatypes

```
type a,b[,c,...];
```

3.1.2 Constants

Analogous to the 0x-notation in binary C, 3CC allows you to specify blanced nonary number with 0n. Furthermore, it allows ternary numbers to be specified with 0t.

```
int a = 0nABD410; // This is a full 12 trit integer
char b = 0tPPNPP0; // This is a 6 trit character
char c = 0n412; // This is also 6 trit character
```

It should be mentioned that some constant expressions do not fold properly in the current version of 3CC, and are hence evaluated at runtime.

3.1.3 Dynamic vs. static variables

3CC uses a 729 tryte variable stack. This imposes significant limits on how many dynamic variables can be allocated, before data is corrupted due to stack overflow. In general, it is a bad idea to allocate arrays and structs dynamically. Deeply recursive functions are also dangerous. If the number of recursions ever

risk comig close to 729 / (stack frame size[1]), you should seriously consider finding another approach. 3CC currently does not support tail recursion.

Speed is another good reason reason to consider using static variables. They assign faster, evaluate faster, increment faster, pretty much anything about them is faster. The only downside is that recursion is out the window, as is calling the same functions from the interrupt service routine[2] and userland.

3.1.4 Arithmetics and operators

3CC has most of C's standard operators. It does not have a digit-shift operator (<< or >>), however a nice thing to keep in mind is that due to the base Tunguska operates in, there is no difference between arithmetic shifts and logical shifts – so you might as well use multiplication and division by powers of 3.

Pointer arithmetic does not work as expected (it works like integer arithmetic)[bug].

3.1.4.1 N-value operator <<N>>

3CC has the ability to take advantage of ternary logic's many different operators, and gives the programmer the ability to use any given commutative ternary operator (tritwise) on two ints or chars. The name of this operator is absolutely terrifying: "arbitrary commutative ternary tritwise operator operator", so let's just call it N-value operator instead. It has the following syntax:

```
a <<N>> b
```

This specific command will execute the operator with FFUUTT-code **N** with parameters **a** and **b**. See section 6.2.3 for the meaning of FFUUTT codes and arbitrary ternary operators. Needless to say, this operator is very powerful once mastered.

3.1.4.2 The sign operator ˜

Since logical inversion is identical to arithmetic inversion, the ˜-operator has been given the new function of returning the sign of a value in a single trit. Use the minus operator - to invert a number / flip it's trits. There are several useful tricks involving ˜:

- ˜X is the value of the most significant trit of X.
- X*˜X = abs(X)

3.1.4.3 Comparisons

Some of the comparing operators of 3CC–greater than and less than–behave slightly different from regular C. If the values are differet, they return true or false, but if they are the same, they return 0.

$$a<b=\begin{cases}1 & a<b\\ 0 & a=b\\ -1 & a>b\end{cases}\qquad a>b=\begin{cases}-1 & a<b\\ 0 & a=b\\ 1 & a>b\end{cases}\qquad a\leq b=\begin{cases}1 & a<b\\ 1 & a=b\\ -1 & a>b\end{cases}\qquad a\geq b=\begin{cases}-1 & a<b\\ 1 & a=b\\ 1 & a>b\end{cases}$$

3.1.5 Operator precedence

Most operators in 3CC work as expected, but for reference, this is how they precede:

1. Incrementers, decrementers
2. Tritwise operations and the sign operator
3. Multiplication, division and modulo
4. Addition and subtraction
5. Comparisons (e.g. >)
6. Logical operations (i.e. && and ||)
7. Assignments

[1] Total size of all dynamic variables of the function, including arguments.
[2] See the **By Example** section below.

3.1.6 Missing features

Among other features that are missing are

- Gotos*
- The ternary operator (i.e. foo?bar:baz;)†
- Function pointers*
- Variadic functions*
- Switch-case statements
- Do-while statements – Use **for** (;;) { loop_code_here; **if** (!condition) **break**; } for now.
- Unions†
- Enums†
- Typedefs†

† Will be implemented in the future. * Will probably not be implemented in the forseeable future.

3.1.7 #pragma directives

3CC has a few pragma directives, all relating to where in memory the code is being written.

SCL **S**tore **C**urrent **L**ocation.

RCL **R**estore **C**urrent **L**ocation.

ORIGIN Set surrent location.

To understand these instructions you need to understand what a ternobj file is. It is essentially a memory snapshot of the entire system memory, that is injected directly into the memory of tunguska. Naturally, writing code for all of the memory in sequence would be very hard, so what SCL, RCL and ORIGIN does is allow you to jump between memory locations, and back. There are various reasons why you might want to do this, but the most useful one is defining global variables in the same location Tunguska has it's memory registers. Example:

```
#pragma SCL // Store current location
#pragma ORIGIN 0nDDDDDD // Jump to 0nDDDDDD
struct {
  char number;
  char data;
} irq;
#pragma RCL // Jump back to the previous location
```

This can be inserted virtually anywhere outside of a function, and will allow you to access IRQ data like any old global variable! See the tunguska specifications for a map of reserved memory addresses and their functionality.

3.2 Running

Running 3CC is simple, though you probably want to manually preprocess your files using some third party tool (GNU CPP works fine) before you pass them to 3cc, since that allows you to use preprocesor macros to include headers arious useful metaprogramming. It is somewhat important to specify origin (-O parameter to 3cc), but 3CC will warn if you choose a disasterous value. A simple build process might look like the following:

1. Preprocess all individual files, save the output to *filename*.ppc
2. Pass all .ppc-files to 3cc in one go: `3cc -o myprogram.asm *.ppc`

3. Assemble the source file: `tg_assembler -O someplace[3] -o myprogram.ternobj myprogram.asm`
4. Run the built program: `tunguska myprogram.ternobj`

A working and easy to modify makefile (for GNU make) is supplied:

```
1  # GNU style makefile for building sources with
2  # GNU CPP, 3CC and tg_assembler.
3  #
4  # This code is public domain.
5
6  SOURCES=file.c anotherfile.c yetanotherfile.c  # These are the sources you
       are building
7  OUTPUT=myprogram.ternobj  # This is the output file
8
9  # If you store your tunguska assembler or 3cc in some
10 #   funky place, you may want to alter these:
11 ASSEMBLER=tg_assembler
12 TRICC=3cc
13 CPP=cpp
14
15 ORIGIN=0n100000 # Specify origin
16
17 # Don't modify any code beneath this line
18
19
20 # Make fileN.c into fileN.ppc and store it in PP_SOURCES
21 PP_SOURCES=$(SOURCES:.c=.ppc)
22
23 # Main build sequence
24 all: image clean_mini
25
26 # Compile image
27 image: $(PP_SOURCES)
28        $(TRICC) -oout.asm $(PP_SOURCES)
29        $(ASSEMBLER) -O$(ORIGIN) -o$(OUTPUT) out.asm
30
31 # Remove preprocessed sources
32 clean_mini:
33        rm -rf *.ppc
34
35 # Remove all output
36 clean: clean_mini
37        rm -rf out.asm $(OUTPUT)
38
39 # Tell make to use suffix rules for targets ending in .ppc
40 .SUFFIXES: .ppc
41
42 # Tell make how to preprocess sources
43 .c.ppc:    $(CPP) $< -o $@
```

3.3 By example

The following section is dedicated to some examples of code illustrating key to essential functionality of Tunguska. They are code that require a lot of low level direct memory manipulation, inline assembly

[3] *0n400000 is a good choice, just make sure you write a bootstrap function at 0n000000.*

or other arcane tricks that need a deep level of understanding of the underlying architecture I can't ask people to get right away.

3.3.1 Interrupt Service Routine (ISR)

This is how to build an interrupt service routine, to handle input and various exceptions. The default ISR is simply the instruction to return back to the code – leaving the interrupt unhandled, but dropped from the queue. If you want access to peripherals like they keyboard or the mouse in your program, you will need an ISR.

```
1  #pragma SCL /* Save current location, we're going to define some code at magic locations
2  #pragma ORIGIN 0nDDDDDD
3  struct {  /* The IRQ registers are located at 0nDDDDDD and 0nDDDDDC */
4    char number;
5    char data;
6  } irq;
7  #pragma ORIGIN 0 /* Make sure the bootstrap ends up at origin, it must
8                    * be the first code run */
9  void bootstrap() {
10   asm("SEI");                    /* Stop interrupts */
11   *(void**)(0n444442) = isr;     /* Set interrupt handler */
12   *(char*)(0n444441) = 0n444;    /* Set clock interval (number of instructions) */
13   asm("CLI");                    /* Resume interrupts */
14   main();                        /* Run code */
15   asm("PAUSE");                  /* Pause if main returns, to avoid
16                                     screenfulls of garbled nonsense */
17 }
18 #pragma RCL /* Restore previous location in memory */
19 void main() { /* Your code goes here */ }
20
21 void clock_cycle(char data) { /* Something like this: */
22   static int counter;
23
24   set_random(data);
25   if(counter ++ > 1000) repaint(); /* See the next example on text output */
26 }
27
28 void isr() {
29   /* It is paramount that you save the tmp variable
30    * before evaluating any sort of code, otherwise
31    * you will break concurrently running code. */
32   asm("PSH tmp", "PSH tmp+1");
33
34   if(irq.number == 0) /* Key interrupt */
35         got_key_input(irq.data);
36   else if(irq.number == 1) /* Clock interrupt */
37         clock_cycle(irq.data);
38   else if(irq.number == 5) /* Mouse motion */
39         mouse_motion(irq.data / 27,
40                      irq.data ^ 0t000NNN);
41   else if(irq.number == 6) /* Mouse click */
42         mouse_click(irq.data);
43   else unhandled_interrupt();
44
45   /* End of handler, restore tmp (note the reversed order) */
46
47   asm("PLL tmp+1", "PLL tmp");
48
49 /* The following is especially important in an ISR,
```

```
50   * the compiler will attempt to return with the RST
51   * instruction, but that causes catastrophical failures
52   * in an interrupt handler, so it is necessary to
53   * manually inject an RTI (ReTurn from Interrupt)
54   * instruction */
55
56    asm("RTI");
57  }
```

3.3.2 Text output

This section outlines how to put characters on the screen. It is fairly simple to expand it to a simple typewriter style console.

```
1  #pragma SCL
2  #pragma ORIGIN 0nDDDDDB
3  char graphics_register;
4  #pragma ORIGIN 0nDDBDDD
5  char text_buffer[27][54];
6  #pragma RCL
7
8  /* Force a screen redraw */
9  void repaint() {
10   /* Use the TSH operator to increase the last trit of
11    * the graphics register by 1 without rolling over. */
12   graphics_buffer = graphics_buffer<<T>>0t00000P;
13 }
14
15 /* Put a character on screen, on a given row and column */
16 void putcxy(char c, char row, char col) {
17   text_buffer[row][col] = c;
18
19   /* A call to repaint here makes the screen refresh immediately */
20   repaint();
21 }
```

Chapter 4

The assembler

4.1 Conventions

The following document uses the following conventions, anything inside hard brackets [like this] is optional. Anything inside braces and separated by pipes {like|this} is a situation where you must select one of the options (in the example, either like or this). 'n' means any numeral, 's' means any string, 'a' means any address or label name, 'l' means explicitly any label name.

4.2 Command line arguments

The assembler takes the following command line arguments:

-v Verbose mode

-o Specify output

-h Display help message

4.3 Numerical constants

There are two main numeral classes in the Tunguska assembler , trytes and words. Trytes have trit-width 6, and words have trit-width 12. A tryte can generally be used in place of a word, but a word can not be used in place of a tryte. There are two special operators, 'LOW' and 'HIGH' that allow you to extract the individual trytes of a word, and use that as a tryte.

In the strictest sense, it is sometimes possible to use a word in place of a tryte without the assembler complaining, but it isn't recommended.

There are two allowed numeral bases, decimal and balanced nonary. Balanced nonary allows numbers in the range -4,...,4; but since there is no symbols for negative numbers, the letters A,...,D are used to symbolize -1,...,-4. Balanced nonary is to ternary roughly what octal is to binary.

31 Regular decimal. No prefix required. Since it is within +-364, both a word and a tryte.

1000 Regular decimal. No prefix required. Only a word, not a tryte.

%0DA Nonary triplet. Always a tryte long.

%0DA114 Nonary sextet. Always a word long.

%000000 Nonary sextet. Always a word long, even if it is smaller than 364.

LOW %0DA114 Lower tryte of nonary sextet. Equivalent to %114.

4.4 Instructions and addressing modes

In general, there is no forbidden instructions in Tunguska. Even if you try to pass an unexpected addressing mode to an operator, the whole machine shouldn't crash and burn. There can be unexpected behavior though, so don't do it on purpose. For a complete list of Tunguska instructions, see the machine specifications.

The tunguska assembler expects all instructions to be uppercase, any lower case instructions will be interpeted as labels or variables, and the assembler will complain.

4.4.1 Addressing modes

OP	Implicit addressing. No argument.
OP A	Accumulator. Whatever operation is done with the accumulator as argument. Strictly speaking, this is the same as implicit addressing.
OP #n	Immediate addressing. Whatever operation is done with the directly specified numeral as argument.
OP a	Absolute addressing. Whatever operation is done on the memory at address a.
OP a,X	Absolute addressing with X offset. Whatever operation is done on the memory at address a+X.
OP a,Y	Absolute addressing with Y offset. Whatever operation is done on the memory at address a+Y.
OP (a)	Indirect addressing. Whatever operation is done on the memory pointed to by the memory at address a.
OP (a,X)	Indirect addressing with X offset. Whatever operation is done on the memory pointed to by the memory at address (a+X).
OP (a),Y	Indirect addressing with Y offset. Whatever operation is done on (the memory pointed to by the memory at address a) + Y.
OP X,Y	XY-addressing. Whatever operation is done on the memory pointed to by X:Y.

4.5 Assembler macros

The Tunguska assembler supports a series of pseudo-instructions, or macros that do not affect the machinecode itself, but allows the assembler to enter non-generated data, or perform other operations.

As a rule of thumb, all macros begin with an **@**, and are all uppercase.

@DT {n|s}[, {n|s}, ...]

Define tryte. Accepts a comma separated list of numerals and strings. These are entered into the assembled memory output as-is.

@DW {n|s|a}[, {n|s|a}, ...]

Define word. Accepts a comma separated list of numerals, strings or memory addresses (labels).

@REST {n|s} [{n|s} = 0]

Reserve argument1 number of trytes into memory, set them to argument2.

@EQU l {n|s|a}

Set variable argument1 to argument2. For most intents and purposes, this is identical to jumping to argument 2 and declaring a label there.

@ORG {n|a}

Jumps instruction counter to address or label specified. Interpret as "this is where I want the following code to go into memory."

@INC {s}

Include file argument1 at this position.

4.6 Inline arithmetics

The Tunguska assembler has support of inline arithmetics, that is, it can calculate pretty much any algebraic function based on values available to it at assembly-time. A magic **$$** token is available, resolving to the address of this (the current) memory position. It works pretty much like you'd expect it to, with regular infix syntax. The only quirk is that you can't use paranthesis for precedence override, instead you must use braces.

This works: {1+2} * 3 - 5 + $$ - somelabel*2

This doesn't: (1+2) * 3 - 5 + $$ - somelabel*2

4.7 Labels and assembler variables

While labels and assembler variables are in many ways interchangable, there are a few differences. Labels can have local child labels and child variables that, from within the label are accessible through .localname and from outside the label through label.localname, but a variable can not.

A label is declared through **labelname:** in the beginning of a line, and accessed through substituting labelname where-ever an address is requested.

A variable is declared through **@EQU variable value**, and is accessible in much the same way a label is. Futhermore, it is possible to store not only words, but trytes in variables, which can be accessed for an instance in immediate addressing mode like this: **OP #variable**.

A very powerful combination is the **$$** -token and variables. For an instance, if you want to determine the length of a string automatically for use later, you can use a construct like this:

```
mystring:  @DT     'Hello world!', 2, 'How are you doing?'
           @EQU    .length         $$ - mystring
```

The length of **mystring** (which reads: 'Hello world![new line]How are you doing?') will be stored in **mystring.length** at no cost of machine memory.

Part II

Introduction to Ternary Computing

Notes

This is more or less original research, and on some parts it might be downright erroneous. See it as a crash course into the relevant parts of Ternary computing to Tunguska. It's unfortunately a bit esoteric in it's fairly heavy use of mathematical and logical symbols and conventions.

Non-mathematical introduction

Before I go into the mathematical stuff, I'm going to list a few of the benefits of balanced ternary numeral systems and ternary logic.

There are many neat features of this base, for an example, it is very simple to round off. What you do to round a balanced number is to drop the least significant digits (as many as you want), and replace them with zeros; and that's all there is to it, you'll never get an error in the same way rounding the decimal number 199 to 100 will.

Another benefit is the representation of negative numbers. In a binary computer, you need to keep a separate bit to keep track of the sign of your number. This is redundant in a computer with a balanced base. There is no possibility of "minus zero", and no need for ugly fixes like two's complement.

Negative numbers in general make a lot more sense in a balanced base. A lot of everyday mathematics regarding negative numbers is downright strange, from the separate minus operation (which is just adding a negative number), to the minus prefix to negative numbers. It resembles some sort of cargo cult, struggling to keep a hold of the confused conventions of the ancestors without really considering what they do.

More resources

There is a multitude of resources on Ternary Computing available on the Internet, though they can be difficult to find. Here are some of them:

http://xyzzy.freeshell.org/trinary Trinary Computer Systems. Very long and complete document about ternary/trinary computers.

http://jeff.tk/wiki/Trinary Trinary - Jeff.tk (Some sort of project to buld a ternary computer? Has a lot of useful links, not very clear on what the purpose of the page is.)

http://www.ternary.info/ Ternary.info - special interest group on balanced ternary numeral system and trinary logic (Ternary site mostly in Russian, has helpful people in it's forum.)

http://en.wikipedia.org/wiki/ Lots of information under they keywords "**Balaned ternary**", "**Ternary logic**", "**Ternary numeral system**", ...

http://www.trinary.cc/ Mostly hardware-oriented trinary computer site.

Chapter 5

Numerical representation

5.1 Conventions

I will use subscript to indicate numeral base. n_{10} is decimal, n_3 is ternary, n_{3b} is balanced ternary, n_{9b} is balanced nonary. When nothing else is indicated, assume decimal.

I will use vector form to represent a number, in this fashion:

$$n =< a_{N-1}, a_{N-2}, ..., a_1, a_0 >$$

n will mean arbitrary number, **N** the number of digits in a number, **a** will mean a digit in given number.

5.2 Ternary numeral base

The **regular ternary** numeral system uses base 3, that is, it has three digits: 0, 1 and 2. The value of a ternary number is formally defined as

$$V = \sum_{i=0}^{N} 3^i a_i \tag{5.1}$$

Where a_i is the i:th digit (and a_0 is the least significant digit). More practically, this means that the ternary number 201, in decimal is

$$210_3 = 2 \cdot 3^2 + 0 \cdot 3 + 1 = 19_{10}$$

There really isn't anything special or spectacular with this numeral base, it works much like you'd expect any old base to do. Much like any conventional base, it's range is

$$0 \leq X < 3^N \tag{5.2}$$

where N is the number of digits. At a fixed word size, addition with consideration to spillover is defined by

$$T_0 + T_1 = (T_0 + T_1) \ \boldsymbol{mod}\ 3^N \tag{5.3}$$

5.2.1 Balanced ternary

The **balanced ternary** numeral system is a non-standard base. It still follows equation (4.1), but the digits are shifted one step down. That is, it's digits are -1, 0, 1; but since there is no symbol for the digit -1, it's conventional to call the digits N, 0 and P. Again, to supply an example:

$$PN0_{3b} = 1 \cdot 3^2 + (-1) \cdot 3 + 0 = 6_{10}$$

An important thing to note is it's range. Note the extreme difference between the range of regular ternary in equation (4.2), and the range of balanced ternary:

$$-\lfloor \frac{3^N}{2} \rfloor \leq X \leq \lfloor \frac{3^N}{2} \rfloor \tag{5.4}$$

Addition with consideration to spillover is defined by

$$T_0 + T_1 = (T_0 + T_1 + \lfloor \frac{3^N}{2} \rfloor) \; \boldsymbol{mod} \; 3^N - \lfloor \frac{3^N}{2} \rfloor \tag{5.5}$$

Where T_1 is positive. Do note the difference between equations (4.5) and (4.3).

The biggest difference between balanced bases and non-balanced "regular" bases is how negative numbers are handled. There is no neat way of handling negative numbers in non-balanced bases, so you have to invent work-arounds. These work-arounds are generally eyesores, either because they allow illegal states (negative zero in one's complement), or because they generate asymmetric ranges (two's complement).

This issue is non-existent in balanced bases, since negative numbers are inherently supported. All you need to do to invert a balanced number is invert all the digits (4.6).

$$-n = - < a_{N-1}, a_{N-2}, ..., a_1, a_0 > = < -a_{N-1}, -a_N, ..., -a_1, -a_0 > \tag{5.6}$$

An important thing to note is that *the sign of a balanced number is the sign of the most significant digit.* While there is nothing wrong with holding on to the minus prefix commonly used in balanced numbers, it really is quite redundant.

5.2.2 Compact representation

There are two major ways of representing ternary numbers in a more manageable way: **Nonary** (Base 9) and **Septemvigesimal** (Base 27). Septemvigesimal is very unpractical, since it requires symbols for 17 additional digits beyond our decimal 0...9.

Nonary is much more manageable. It is very close to decimal in data density and therefore easy to "intuit" the value of. Nonary should be easier to wrap your mind around than say hexadecinal which is more alien to decimal, data-density-wise.

In this document, we're interested specifically **balanced nonary**. It has nine digits, -4, -3, -2, -1, 0, 1, 2, 3, 4. Again, we lack symbols for negative digits, and for this reason, I will introduce the letters A...-D to represent -1,...-4.

It's value is governed by the following formula:

$$V = \sum_{i=0}^{N} 9^i a_i \tag{5.7}$$

Example:

$$3AD_{9b} = 9^2 \cdot 3 + 9 \cdot (-1) + (-4) = 230_{10}$$

Each balanced nonary digit represents two balanced nonary digits.

5.3 Ternary representation on binary computers

There is no such binary word size b that it's possible to represent a ternary word of size t without superfluous states. This follows from the prime uniqueness theorem, which simplified to fit this case states that

$$3^b \neq 2^t \forall b, t \neq 0$$

So the quest is not to find a lossless representation of ternary values on binary computer, but minimizing loss while maximizing usability and speed. Assume ternary word size 6, 729 states. The smallest number of binary bits that can represent that many states is 10, 1024 states. That's 295 meaningless states. Furthermore, there is no real obvious connection between individual bits in the binary representation of the ternary word, and the digits of the ternary word.

Obviously, this is not a good representation. It is small, but quite useless. Instead, if one represents a ternary digit with two bits. This representation squanders states pretty carelessly, but it is by far the most easily used one. There is a clear tie between bits and ternary digits, and it is quite fast.

It is also worth mentioning that it is possible to implement ternary logic with memory pointers, where the value is represented by a pointer to a memory location containing a true-false value, but this

memory location can also be some invalid value (typically NULL), meaning the third state. However, this is even worse from a state point of view. On a 32 bit computer, $2^{32} = 4,294,967,296$ states (the size of a memory address) are used to store one state, and the other two states are stored in hopefully one bit, but more likely 8. So, worst case scenario, that's using $2^{32} + 2^8 = 4,294,967,552$ states to store 3 states. Needless to say, this is not something you'll want to use for speed or size. No matter how you twist and turn this, it will never be an effective solution[1].

5.3.1 Hexadecimal Packed Balanced Nonary

A convenient trick to outputing balanced nonary data in binary computers is to pack them in hexadecimal numbers. What you do is simply to map 0,...,4 to 0,...,4; and -1,...,-4 to 10,...,14. Conversion is a bit tricky, but the benefits is to be able to use native printf-like functions that understand hexadecimal numbers to print nonary numbers without having to mess around with strings.

[1] For creating a ternary computer, emulated or in hardware. There is probably some cases where this sort of construction can be useful in other fields.

Chapter 6

Ternary logic

6.1 Conventions

In this part, I will abbreviate TRUE to T, UNKNOWN to U, FALSE to F; furthermore, I will use standard logic symbols

- $\bigwedge$= AND
- $\bigvee$= OR
- $\bigoplus$= XOR
- ...

6.2 True, Unknown, False

Boolean logic, TRUE or FALSE-logic, while immensely powerful, is limited due to the fact that it only deals in absolutes. Answer the question "Was it raining in western Beijing 4:39 PM, March 13 1839?" in boolean logic. Unless you're an archeometeorologist, are exceptionally old, or own a time machine; you won't know the answer, and therefore, you will be unable to answer either true or false.

But, note well how your head is not spinning and sparking and you how you are not shrieking "DOES NOT COMPUTE!" repeatedly, because unlike robots in science fiction films from the 1950's, you understand ternary logic.

Ternary logic isn't something new and scary, it's something old and mundane. It's, just as boolean logic, a formalization of human logic. Only difference is that ternary logic introduces a third state. There are multiple ideas of what this third state should be; unknown, both, neither, a zen non-answer. None of them are inherently *wrong*, they are all ternary logic systems. I will be primarily describing true-unknown-false logic.

It's important to realize that this is just one possible interpretation of ternary logic. Even though I will be using "proof"-like devices to to derive the output of the functions, this is merely so that the system of logic formulated is self-consistent, and compatible with booelan logic. This also means the proofs can be more lax, and call on common sense more often than otherwise merited.

6.2.1 Logical conditionals

What does this mean for logical conditionals? Let's take a look at them.

6.2.1.1 Material conditional

Material conditionals don't need that much modification. Assume you have a rule "If P, then Q", or in logical symbols $P \rightarrow Q$, then since modifying existing boolean rules is undesirable, it is necessary to make a rule for the case P is Unknown. It is not hard to convince yourself that if P is unknown, and P implies Q, then Q must also be unknown. We can also protect ourselves against the logical fallacy of affirming the consequent by introducing a rule that states that if P implies Q, and Q is true, then P is unknown.

6.2.1.2 Logical biconditional

Logical biconditionals are also pretty much the same. Assume you hae the rule "P if and only if Q", or in logical symbols $P \leftrightarrow Q$, then the case of concern is when P is unknown. Since all cases involving true or false, for both P and Q, are already taken by boolean logic, then a logical biconditional with either P or Q unknown must imply that the other is also unknown.

6.2.2 Ternary operators

Boolean logic isn't wrong, it works and it arrives at the correct results, so whatever extensions are made to it must preserve backward compatibility with boolean logic, so all standard boolean operators with exclusively true or false values must result in the same values you would expect them to with conventional boolean logic.

6.2.2.1 $\bigwedge$ (AND)

The AND operator can pretty much be intuited from what we know of it's boolean counterpart. We know AND is commutative, so the only cases we need to deal with are:

- T $\bigwedge$ U = U; Since $T \bigwedge F \neq T \bigwedge T$, the result must be U.
- F $\bigwedge$ U = F; Since $F \bigwedge F = F \bigwedge T = F$, the result must be F.
- U $\bigwedge$ U = U; Since both arguments are unknown, nothing is known about the result.

6.2.2.2 $\bigvee$ (OR)

The same logic used to deduce the results of the AND operator can be used to deduce the results of the OR operator. It also is commutative, so the cases are the same:

- T $\bigvee$ U = T; Since $T \bigvee F = T \bigvee T = T$, the result must be T.
- F $\bigvee$ U = U; Since $F \bigvee F \neq F \bigvee T$, the result must be U.
- U $\bigvee$ U = U; Since both arguments are unknown, nothing is known about the result.

6.2.2.3 $\bigoplus$ (XOR)

The same logic used to deduce the results of AND and OR also works for XOR. XOR is also commutative, so these are the cases we must consider:

- T$\bigoplus$U = U; Since T $\bigoplus$ F $\neq$ T$\bigoplus$ T, T$\bigoplus$U=U.
- F$\bigoplus$U = U; Since F $\bigoplus$ F $\neq$ F$\bigoplus$ T, T$\bigoplus$U=U.
- U$\bigoplus$U = U; Both arguments are unknown, nothing can be said about the result.

A small point of interest, if ternary logic is represented with balanced ternary, so that T=1, U=0, F=-1, then $P \bigoplus Q = -(P \cdot Q)$, which makes $\bigoplus$ very useful in ternary computing, since it both does masking and inversion.

6.2.2.4 ˜ (NOT)

If nothing is known about a variable, then nothing can be known about the inverted variable. Therefore, ˜U = U.

6.2.3 Formalization with Set Theory

It is possilble to generalize the method used above to go from boolean logic to ternary true-unknown-false logic with set theory.

First define a the set $R = \{True, False\}$, furthermore, the subsets $T = \{True\}$, $F = \{False\}$, $U = \{True, False\}$.

Furthermore, define the following axioms

$$O(T,U) = O(T,T) \cup O(T,F) \tag{6.1}$$

$$O(F,U) = O(F,T) \cup O(F,F) \tag{6.2}$$

$$O(U,T) = O(T,T) \cup O(F,T) \tag{6.3}$$

$$O(F,U) = O(T,F) \cup O(F,F) \tag{6.4}$$

$$O(U,U) = O(U,T) \cup O(U,F) \cup O(T,U) \cup O(F,U) \tag{6.5}$$

Given you know the truth values for any given boolean value, you can use these five axioms to determine the remaining truth values in the true-unknown-false logic described herein. This is a method of deriving a ternary extension of a boolean function, so it can't be proven or disproven (indeed, it can not be true or false).

An interesting extension to this formalism is the introduction of Neither $N = \{\}$ as the empty set, allowing for a description of a quaternary true-unknown-false-neither system. A possible rule set for this could look something along the line of

$$O(X,N) = \{T,F\} \backslash (O(X,T) \cup O(X,F))$$

...

6.2.4 Non-boolean operations and FFUUTT-notation

There are 19,683 possible ternary operations (stemming from the 9 degrees of freedom available). Most of these are headache inducing, non-commutative and just plain unpleasant to work with. Now, sifting through just south of 20,000 operations is a lot of work, so it would be nice to find some sort of subset that is guaranteed to be more easy to work with. I've already mentioned commutativity. A op B should, in a nice operation, be the same as B op A.
Since commutative operators removes three degrees of freedom, there are 729 of them. Unlike boolean logic, which has 16 operations (all with a name), no one can be expected to remember 729 names. Therefore, I introduce FFUUTT-numbers. Each operation is given a number from -364 through 364, that describes it's operation in a quite eloquent way.
The number is simply the result of the operation when the arguments are arranged in a specific, and easy to remember order. Just insert the results into a table like the one to the right, And then interpret the list of numbers as digits in a balanced nonary number N, which is the FFUUTT-code for the given operation.

A	B	A *OP* B
F	F	X_5
F	U	X_4
U	U	X_3
U	T	X_2
T	T	X_1
T	F	X_0

Table 6.1: FFUUTT-table

$$N_{OP} = 3^5 X_5 + 3^4 X_4 + 3^3 X_3 + 3^2 X_2 + 3X_1 + X_0$$

6.2.4.1 Shift (N=168)

The shift operator is an operator of convenience. It doesn't correspond to any particular logical relationship. What it does is essentially addition with wrap around. With balanced ternary, the function is defined as

$$S(P,Q) = \begin{cases} 1 & P+Q=-2 \\ -1 & P+Q=-1 \\ 0 & P+Q=0 \\ 1 & P+Q=1 \\ -1 & P+Q=2 \end{cases}$$

6.2.4.2 Shift w/o rollover (N=-312)

The shift without rollover operator (called permute in Tunguska, from it's ability to permute values) is like shift, without the wrap-around.

$$S|(P,Q) = \begin{cases} -1 & P+Q<0 \\ 0 & P+Q=0 \\ 1 & P+Q>0 \end{cases}$$

6.2.4.3 BUT (N=-241)

The BUT operator is a complement to AND and OR. It does not correspond to any semantic operator. It is part of the same family of operators as AND and OR, with a 5:3:1 distribution of values. AND has 5F:3U:1T, OR has 5T:3U:1F; while BUT has 5U:3F:1T.

There are many other 5:3:1-operators, truth to be told, BUT is only implemented in Tunguska because it is in TriINTERCAL, so not to hang any TriINTERCAL-users out to dry without their favorite operator.

6.3 Other systems

Other systems I'm not going to spend any more paper (or screen) deducing, but roughly outline because of their importance is true/neither/false (TNF) and boolean-conservative (BC).

True/neither/false is another human-logic translation where the third state is set to mean neither instead of unknown.

Boolean-conservative is another step towards that direction, it treats the third state as false. As an outcome of this, it's return values are always compatible with boolean logic.

The C++ library "Boost" implements yet another ternary logical system in "tribool"[1], that is not consistent with regular boolean logic.

I'm bringing them up because they are good food for thought, and pondering them, and even more exotic, non-boolean systems are a great way of getting a more intimate understanding of ternary logic.

For an example, consider the XOR operation, it is notable that this is not the only meaningful extension of XOR. A pattern that emerges in binary XOR is that the operation answers the question "are these values different?" – but the above interpetation answers it in a more abstract way than one might like: Looking at the meaning of the symbols, U and U represent all possible permutations of true and false, but looking at the symbols themselves, they are the same and only the same.

So, which one is XOR? Both are. The only real requirement to *be XOR* is to be compatible with boolean XOR. The true/unknown/false XOR I started out with is one variant of XOR, the symbol-level XOR is another. For any given commutative boolean operation, there are three degrees of freedom, and consequently 27 ternary siblings (ranging from sensible to nonsensical.)

6.4 Truth tables

Here are truth tables for the hitherto mentioned logical operations.

[1] See 'http://www.boost.org/doc/libs/1_36_0/doc/html/tribool.html'

A	T	T	T	U	U	U	F	F	F	**N**
B	T	U	F	T	U	F	T	U	F	
A $\bigwedge$ **B**	T	U	F	U	U	F	F	F	F	-322
A $\bigvee$ **B**	T	T	T	T	U	U	T	U	F	-320
A $\bigoplus$ **B**	F	U	T	U	U	F	T	U	F	241
A BUT **B**	T	U	F	U	U	U	F	U	F	-241
A shift **B**	F	T	U	T	U	F	U	T	F	168
A shift w/o rollover **B**	T	T	U	T	U	F	U	F	F	-312

Table 6.3: Truth tables

Appendix A

Tunguska specifications

Tunguska uses balanced ternary arithmetic, True-Unknown-False logic, tryte width 6. Address width and word size 12. Standard notation balanced base-9 (nonary). Accessible address range DDD:DDD$_{b9}$ to 444:444$_{b9}$. Total accessible memory 531$_{10}$ ktrytes. Big endian address storage.

A.1 Registers

Tunguska has relatively few general purpose registers, but the speed of memory access is more or less identical to register access, so this is not that big of a deal. Directly user-accessible registers are indicated by bold.

Name	Purpose
A	Accumulator
X	Address register
Y	Address register
PC	Program counter
S	Stack index
CL	Clock register
P	Processor status

A.1.1 Processor status register specification

The processor status register (P) has the following tritfields, and it is set by most operations.

MST					**LST**
PR	V	B	I	G	C

More elaborately, the flags have the following names and functionality

Flag	Full name	Description
C	Carry	Carry of the last operation.
G	Comparison	The sign of the last operation.
I	No-Interrupt	Put interrupts on hold.
B	Break in progress	Set during interrupts.
V	Overflow	Set by CMP, much like G. Used in jumps.
PR	Parity	The sign of the trit sum of the last operation.

When the No-Interrupt flag is set, all interrupts will be put on hold until it is cleared. The only limiting factor in how many interrupts can be kept waiting is the host system memory. The break flag is set when an interrupt is in progress. It is cleared when the interrupt is over. It also acts like the I-flag, blocking interrupts. The I and B flags may merge in the future to make better use of ternary capabilities.

A.2 Op-code specifications

The tunguska operation codes are following as following, where A is addressing mode and C is operation. In general, there are no forbidden combinations of addressing modes and operations, specifying an unsupported mode may be unpredictable, but will not crash the system.

MST					**LST**
A	A	C	C	C	C

A.2.1 Addressing modes

DEC	B9	Symbol	Size	Description
-4	D	ABS	3	Value at memory position directly specified
-3	C	IMM	2	Value immediately following op-code
-2	B	AX	3	ABS with offset X
-1	A	AY	3	ABS with offset Y
0	0	ACC	1	Accumulator
0	0	IMP	1	Implicit – no argument
1	1	INDX	3	Value at (Memory position + X)
2	2	INDY	3	Value at (Memory position)+Y
3	3	INDIRECT	3	Value pointed to by the memory position directly specified
4	4	X:Y	1	Memory at page X, index Y

A.2.2 Operations

DEC	B9	Symbol	Valid addressing mods	Comment
-40	DD	CLV	IMP	Clear overflow
-39	DC	BRK	IMP	Trigger interrupt
-38	DB	RTI	IMP	Return from interrupt
-37	DA	LDA	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Set accumulator to memory value
-36	D0	LDX	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Set Y register to memory value
-35	D1	LDY	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Set X register to memory value
-34	D2	STA	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Store accumulator's value in memory
-33	D3	STX	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Store X register's value in memory
-32	D4	STY	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Store Y register's value in memory
-31	CD	TAX	IMP	Transfer A to X
-30	CC	TAY	IMP	Transfer A to Y
-29	CB	TXA	IMP	Transfer X to A
-28	CA	TYA	IMP	Transfer Y to A
-27	C0	TSX	IMP	Transfer Stack index to X
-26	C1	TXS	IMP	Tarnsfer X to Stack index
-25	C2	PHA	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Push to stack
-24	C3	PHP	IMP	Push processor status to stack
-23	C4	PLA	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Pull from stack
-22	BD	PLP	IMP	Pull processor status from stack
-21	BC	AND	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A=A∧memory
-20	BB	EOR	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A=A⊕memory
-19	BA	ORA	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A=A∨memory
-18	B0	BIT	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Set status flag as though AND
-17	B1	ADD	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A=A+memory
-16	B2	CMP	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Set status flag as though A-memory
-15	B3	INC	ACC, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Increase value by 1
-14	B4	INX	IMP	Increase X register by 1
-13	AD	INY	IMP	Increase Y register by 1
DEC	**B9**	**Symbol**	**Valid addressing mods**	**Comment**

DEC	B9	Symbol	Valid addressing mods	Comment
-12	AC	DEC	ACC,ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Decrease value by 1
-11	AB	DEX	IMP	Decrease X register by 1
-10	AA	DEY	IMP	Decrease Y register by 1
-9	A0	ASL	ACC,ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Shift value left, spillover in carry
-8	A1	LSR	ACC,ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Shift value right, spillover in carry
-7	A2	ROL	ACC,ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Rotate left
-6	A3	ROR	ACC,ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Rotate right
-5	A4	JMP	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump
-4	0D	JSR	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump to subroutine
-3	0C	RST	IMP	Return from subroutine
-2	0B	CLC	IMP	Clear carry
-1	0A	CLI	IMP	Clear interrupt flag
0	00	NOP	IMP	No operation
1	01	SEC	IMP	Set carry
2	02	SEI	IMP	Set interrupt flag
3	03	MLH	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A=high tryte(A*memory)
4	04	MLL	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A=low tryte(A*memory)
5	1D	DIV	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A = integer($\frac{A}{memory}$)
6	1C	MOD	ACC, IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A = remainder($\frac{A}{memory}$)
7	1B	PLX	IMP	Pull X from stack
8	1A	PLY	IMP	Pull Y from stack
9	10	PHX	IMP	Push X to stack
10	11	PHY	IMP	Push Y to stack
11	12	JCC	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if carry is clear
12	13	JCT	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if carry is positive
13	14	JCF	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if carry is negative
14	2D	JEQ	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if comparison flag is clear
15	2C	JNE	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if comparison flag isn't clear
16	2B	JLT	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if comparison flag is negative
17	2A	JGT	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if comparison flag is positive
18	20	JVC	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if overflow is clear
19	21	JVS	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Jump if overflow is set
20	22	IVC	IMP	Invert carry flag
21	23	PRM	IMM, ACC, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Permutate value (tritwise add 1)
22	24	TSH	IMM, ACC, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	PRM without roll over
23	3D	BUT	IMM, ACC, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A BUT Value
24	3C	LAD	ABS, ABSX, ABSY, IND, INDX, INDY	Load ADdress into X:Y
25	3B	PGT	IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Copy contents of page A to page mem
26	3A	PGS	IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Set contents of page A to mem
27	30	CAD	ABS, ABSX, ABSY, IND, INDX, INDY	Compare X:Y with argument address
28	31	XAM	IMM, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	Exchange A with memory value
29	32	XAX	IMP	Exchange A with X
30	33	XAY	IMP	Exchange A with Y
31	34	XYX	IMP	Exchange X with Y
32	4D	PLC	IMM, ACC, ABS, ABSX, ABSY, IND, INDX, INDY, X:Y	A/M Commutative operation from stack
33	4C	ADW	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y | STACK	12 trit addition
34	4B	MLW	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y | STACK	12 trit multiplication
35	4A	DVW	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y | STACK	12 trit division
36	40	MDW	ABS, ABSX, ABSY, IND, INDX, INDY, X:Y | STACK	12 trit modulo
37	41	RESV		
38	42	RESV		
39	43	PAUSE	PAUSE	PAUSE
40	44	DEBUG	DEBUG	DEBUG
DEC	**B9**	**Symbol**	**Valid addressing mods**	**Comment**

A.3 Reserved addresses

The following memory addresses have special functionality, and should not be used for general purpose code or data storage.

Address range	Purpose
DDD:DDD/C	IRQ, IRQ data
DDD:DDB	Screen mode
DDD:DDA	Disk I/O
DDD:DD0	AGDP Operation
DDD:DD1/2	AGDP Register 1
DDD:DD3/4	AGDP Register 2
DDD:DCD/C	AGDP Register 3
DDC:DDD-DDC:444	Stack
DDB:DDD-DDA444	Screen buffer, text mode
DDB:DDD-DDB:444	Screen buffer, vector mode
DDB:DDD-CAB:444	Screen buffer, raster mode
444:441	Clock interrupt interval[1]
444:442/3	Interrupt handler vector

A.3.1 Screen

The Screen Mode tryte (DDD:DDB) has the following function

MST					**LST**
RES	RES	RES	Aux graphics mode	Graphics mode	Redraw

Screen redraws when Redraw is set to 1, and allowed graphics mode are raster(-1), text (0) and vector (1).

A.3.1.1 Vector mode

Vector mode only uses the first page of the screen buffer (that is, DDB:DDD/444), and stores vector information in sets of three trytes (allowing for a total of 121 interconnected lines in 729 colors). They are drawn in succession, and black lines are not painted (there won't be black spots where black lines intersect visible ones.)

	MST					**LST**
First tryte	MRed	LRed	MGreen	LGreen	MBlue	LBlue
Second tryte	UNUSED	X -	Coo-	-rdi-	-na-	-te
Third tryte	UNUSED	Y -	Coo-	-rdi-	-na-	-te

A.3.1.2 Raster mode

Raster mode resolution is 324x243[2] and uses a total of 108 memory pages, with two color depths:

- One pixel per tryte, allowing for a total of 729 colors. They are encoded like in the first tryte in vector mode above, addressed according to (1).

$$T(x, y) = DDBDDD_{9b} + 324y + x \tag{A.1}$$

- One trit per pixel, allowing for a total of three gray light intensities, memory addressed according to (2), color intensity specified by (3):

$$T(x, y) = DDBDDD_{9b} + 54y + \lfloor \frac{x}{6} \rfloor \tag{A.2}$$

[1] Don't set it to zero or less, unpredictable behavior will ensue
[2] Observant people will notice that this is 400_{b9}x300_{b9}

$$R(x) = x \textbf{ mod } 6 \tag{A.3}$$

A.3.1.3 Text mode

Text mode resolution is 27 rows by 54 columns (2 memory pages). Visible and control characters are in the range 0...99, and arranged in the following fashion:

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9
0x	**NUL**	**BS**	**NL**	**RES**	**RES**	**RES**	**RES**	**RES**	**RES**	**RES**
1x	A	B	C	D	E	F	G	H	I	J
2x	K	L	M	N	O	P	Q	R	S	T
3x	U	V	W	X	Y	Z	0	1	2	3
4x	4	5	6	7	8	9	.	,	!	?
5x	a	b	c	d	e	f	g	h	i	j
6x	k	l	m	n	o	p	q	r	s	t
7x	u	v	w	x	y	z	**BB**	**BB**	**BB**	**BB**
8x	**BB**	**BB**	**BB**	**BB**	**BB**	**BB**	=	-	*	/
9x	%	<	>	≤	≥	(	)	$	+	#

Symbol	Description
NUL	Null
BS	Back space
NL	New line
RES	Reserved
BB	Graphical block

A.4 Interrupts

IRQ	**Function**	**IRQ data**
0	Keypress interrupt	Key
1	Clock interrupt	Random value
2	Keyboard break sent	undef.
3	Arithmetic error	undef.
4	Soft interrupt	undef.
5	Mouse motion	relative X : relative Y
6	Mouse event	click = 1, release = -1

Interrupts cause the system to push all registers onto stack, and jump to the Interrupt Service Routine (ISR), whose location is stored in 444:442/3. Arguments to the ISR are stored in two memory locations, IRQ and IRQDATA, DDD:DDD and DDD:DDC respectively. Interrupts are queued if the I-flag is set, or there already is an interrupt being processed.

An important interrupt is the clock interrupt, it is raised whenever the CL register (which increments with every cycle), exceeds the value stored at 444:441. The RTI instruction is used to properly return from an ISR.

A.5 Disk I/O

Tunguska features a floppydisk-like virtual disk drive. It is controlled through the Disk I/O tryte (DDD:DDA). It is the same size as the main memory (729^2 trytes). All data transfer is on the block level, and block size is 729 trytes. The following operations are allowed:

Name	Number	Function
NOOP	0	Idle
READ	1	Read from disk to memory
WRITE	2	Write from memory to disk
SYNC	3	Write from virtual disk to physical file
SEEK	4	Set disk position to page
GETPOS	5	Get disk position

SEEK and GETPOS use the accumulator as input. Data is read to and from memory page specified by the Y register.

The disk image format is 729^2 consecutive 16 bit *short integer*s, each storing a single memory cell. To save space, the images are compressed with ***zlib.***

A.6 Auxiliary General Data Processor (AGDP)

The AGDP offers block operations and floating point arithmetics. It has three 12 trit registers, AGDP R1-R3; and one memory position that determines it's operation.

Operation	Number	Function	Arguments
NOOP	0	Nothing	-
ITOF	1	Convert Integer to Float	R1 -> R1
FTOI	2	Convert Float to Integer	R1 -> R1
FADD	3	Add two float numbers	R1, R2 -> R1
FMUL	4	Multiply two float numbers	R1, R2 -> R1
FDIV	5	Divide two float numbers	R1, R2 -> R1
FEXP	6	Exponent of R1	R1 -> R1
FLOG	7	Natural logarithm of R1	R1 -> R1
FCOS	8	Cosine of R1	R1 -> R1
FSIN	9	Sine of R1	R1 -> R1
BLT	10	Block transfer R1 to R2	Length R3
BLS	11	Block set R1 to val R2+1	R2+1->R1:(R1+R3)
BLA*	12	Blockwise AND R1 vs. R2	Length R3
BLX*	13	Blockwise XOR R1 vs. R2	Length R3
BLO*	14	Blockwise OR R1 vs. R2	Length R3
BSH*	15	Blockwise "TSH" R1 vs. R2	Length R3
BLP*	16	Blockwise "PRM" R1 vs. R2	Length R3
WHEN	17	First six fields of struct_tm is inserted in R1:R3	-

* Overlap undefined.

A.7 Floating point

Tunguska has support for the equivalent of half-precision floating point numbers, with base 3, a [-40,40] range exponent, and [-3280, 3280] range significand. Where the numbers are packed in two trits in the following fashion.

MST					**LST**	**MST**					**LST**
E	E	E	E	S	S	S	S	S	S	S	S

The mathematics are more or less identical to binary floating point, the only significant difference is

that there is no need for bias or a sign bit (sign is determined by the significand S). The mathematics behind it is as follows.

$$F = S \cdot 3^E \tag{A.4}$$

$$S = F \cdot 3^{-E} \tag{A.5}$$

$$E = \lfloor \frac{\ln |F|}{\ln 3} \rfloor \tag{A.6}$$

A.8 Notes for 6502 programmers

There are some major differences between the 6502 and this machine. Beyond the obvious differences in endianness, and processor status flag, some instructions have changed. **PHA** is replaced by a general purpose **PSH** ("*PHA A*" will replicate the behavior of **PHA**), that will push any memory value onto the stack. **PLL** is it's respective pull operation. **ADC** has changed name to **ADD** with no change in functionality. **RTS** has changed name to **RST** with no change in functionality.

X:Y addressing is a new mode that uses both the X and Y register to address a memory location. The **LAD** operator is a quick way to transfer a complex memory location onto X:Y (compare with the x86 **"**LEA**"** operation). Tunguska has an address comparison operation **CAD** that compares the argument with X:Y, and sets flags accordingly.

Branching has been abandoned, use jumps instead. The same behavior can be acheived with the $$-token.

There is also a series of convenience operations **XAM**, **XAX**, **XAY**, **XYX** that serve to swap the contents of registers or memory in one atomic instruction.

A.9 Debugging

Tunguska operates some built-in debugging facilities, mainly the **DEBUG** instruction which prints the contents of all registers to stdout, and the **PAUSE** instruction which acts like a breakpoint and halts execution at a specific point in the code, to allow for instruction stepping with the F11 key (execution is resumed by pressing the Pause key.)

Appendix B

The GNU Free Documentation License

GNU Free Documentation License Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of

the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same

name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version

published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.