

# Chapter 1

## Specification

SPAL simple compile to OpenCL.

### 1.1 stages of compilation

1. natural language text (perhaps)
2. analytic language text
3. SPAL language text
4. SPAL encoded tiles
5. (OpenCL) C with SPAL names
6. (OpenCL) C with natural names (perhaps)

### 1.2 Method for implementation

In theory can use any language for implementation. Though ideally would be a version of C which is similar to the above, so it could then be recoded in SPAL.

### 1.3 answer verification

The agree debug library is OpenCL and holy ceremony (pure function) compatible.

Ideally would have a way of listing many inputs and their corresponding outputs. If this could be fed to an OpenCL kernel that would be delicious.

The agree debug library can be the “testing framework” for SPAL programs. So each agree statement adds a line to the newspaper, after the program is complete it can list the statements in the newspaper, saying those are the tests that failed. Additionally could have a list of the number that have passed.

Pyash	SPAL	C	file
kratta krathnimna li	cardinal _top cardinal name _nom _rea	int main () {	cardinal_name.c
swicta hnimna li	social _top name _nom _rea	void name () {	cardinal_name.c
hmasta hnimna li	mind _top name _nom _rea	inline void name (); inline void name () {	library_cardinal_name.h library_cardinal_name.c
krathmasta hnimna li	cardinal mind _top name _nom _rea	kernel void name () {	cardinal_name.cl
htipdoyu txikka hciccu	ten _num _ins indexFinger _acc down _con	if (i < 0xA) {	
zrundofi	0 _num _return	return 0;	
fe	_finally	}	
hnimna tyindo cyah	name _nom three _num _cop	name = 3;	
txikna zrondo cyah	indexFinger _nom zero _num _cop	i = 0;	
htipdoyu txikka hciccu hyikdoyu plosliwa htekhromli	ten _num _ins indexFinger _acc down _con indexFinger _acc one _num _ins plus _rea _and library program _rea	for {;i < 0xA; ++i}{ library_program ();}	

I'm thinking can save both the line number, and the amount that have passed in the first line of the newspaper. It can be an actual sentence, with two 16bit spaces for the values.

gzat na hnuc do lweh hnuc do mwah slak fa li

A newspaper until number with number succeeded.

A newspaper should be at least 16 sentences long, which is one page or 512 bytes, and less than or equal to 512 sentences, (32 pages), since that is the most that could fit in L1 memory with other processes.

## 1.4 Memory

There is no dynamic allocation of memory, only static, until further notice.

This is because historically dynamic allocation of memory has led to many memory leaks and other problems.

## 1.5 Control Flow

Instead of the traditional for loop that relies on variables defined outside it's bounds, the for loops in SPAL only contain a function, and a listener to see if it should break early.

That way based on the size of the for loop, the compiler can assign it to run on single thread, multi thread CPU, or GPU.

## 1.6 translate all independentClauses to C

Any independent-clause can be turned into C.

can be of the form:

---

```
sort1_case1_sort2_case2_verb_mood (sort1 name, sort2 name);
```

---

### 1.6.1 C Name Composition

For C, will need to include the types of the names in order to properly call functions, otherwise would have to have extra searching to locate which function is being referred to.

This will make it a bit like Navajo or Swahili, where the noun class will be mandatory. So we should have easy grammar words for them,

for names of things:

**plu** paucal-number 8bit

**do** number 16bit

**pu** plural-number 32bit

**ml6h** multal-number 64bit

**ml6hhsosve** multal-number sixteen vector, vector of 16 64bit values.

**fe** referrential, pointer

**crih** letter, char

**crihfe** letter referrential, char \*

It seems I would only need a hash table lookup for operating on the GPU,  
seems like most of the other stuff can be done with a few conditionals.

## Chapter 2

# Pyash Encoding

The virtual machine uses variable-length-instruction-word (VLIW), loosely inspired by head and tails instruction format (HTF). HTF uses VLIW's which are 128 or 256 bits long, however there can be multiple instructions per major instruction word.

### 2.1 VLIW's Head Index

The head is really a parse index, to show the phrase boundaries. In TroshLyash each bit represents a word, each of which is 16bits, when a phrase boundary is met then the bits flip from 1 to 0 or vice-versa at the phrase boundary word. index takes up the first 16bits of the VLIW. This would lead to 256bit (32Byte) VLIW's. The real advantage of the indexing occurs when there either multiple sentences per VLIW, or when there are complex sentences in the VLIW's. Having the VLIW's broken up into 32Byte chunks, makes it easier to address function entry points, which can be placed at the beginning of a VLIW. Can fit 16 VLIWS in a POSIX page, 128 VLIW's in a Linux page, so would only need 1 byte (8bits) for addressing functions that are within 1 page distance.

### 2.2 Word Compression

Now for the slightly more interesting issue of packing as many as 5 glyphs into a mere 16 bits. Why this is particularly interesting is that there is an alphabet of 32 glyphs, which would typically required 5 bits each, and thus 25bits in total. However the 16 bit compression is mostly possible due to the rather strict phonotactics of TroshLyash, as only certain classes of letters can occur in any exact place. The encoding supports 4 kinds of words, 2 grammar word classes and 2 root word classes. Where C is a consonant, T is a tone and V is a vowel, they are CVT, CCVT, and CVTC, CCVTC respectively.

### 2.2.1 CCVTC or CSVTF

I'll start with explaining the simplest case of the CCVTC word pattern. To make it easier to understand the word classes can call is the CSVTF pattern, where S stands for Second consonant, and F stands for Final Consonant. The first C represents 22 consonants, so there needs to be at least 5 bits to represent them. Here are the various classes

**“C”** : “p”, “t”, “k”, “f”, “s”, “c”, “x”, “b”, “d”, “g”, “v”, “z”, “j”, “n”, “m”, “q”, “r”, “l”, “y”, “w”,

**“S”** “f”, “s”, “c”, “y”, “r”, “w”, “l”, “x”, “z”, “j”, “v”,

**“V”** “i”, “a”, “e”, “o”, “u”, “6”,

**“T”** “7”, “\_”,

**“F”** “p”, “t”, “k”, “f”, “s”, “c”, “n”, “m”

, (can check the phonology page for pronunciation) C needs 5 bits, S would need 4 bits, however the phonotactics means that if the initial C is voiced, then the S must be voiced, thus “c” would turn into “j”, “s” into “z” and “f” into “v”, also none of the ambiguously voiced phonemes (l, m, n, q, y, w, r) can come before a fricative because they have a higher sonority, thus must be closer to the vowel. So S only needs 3 bits. V needs 3 bits T needs 2 bits and F needs 3 bits which is a total of 16 bits.  $5+3+3+2+3 = 16$  However there are other kinds of words also. we'll see how those work.

### 2.2.2 HCVTF

So here we have to realize that CVC or CVTC is actually HCVTF due to alignment. So what we do is make a three bit trigger from the first word, the trigger is 0, which can be three binary 0's,  $0b000$   $3+5+3+2+3 = 16$  H+C+V+T+C this does mean that now  $0b1000$ ,  $0b10000$  and  $0b11000$  is no longer useable consonant representation, however since there are only 22 consonants, and only 2 of those are purely for syntax so aren't necessary, so that's okay, simply can skip the assignment of 8, 16 and 24.

### 2.2.3 CSVT

This is similar to the above, except we use  $0b111$  as the trigger, meaning have to also skip assignment of 15, 23 and 31.  $3 + 5 + 3 + 3 + 2 = 16? + C + S + V + T$

### 2.2.4 CVT

For this one can actually simply have a special number, such as 30, which indicates that the word represents a 2 letter word.  $5+5+3+2+1$   $F+C+V+T+P$  what is PF P can be a parity-bit for the phrase, or simply unassigned.

## 2.3 Quotes

Now with VM encodings, it is also necessary to make reference to binary numbers and things like that. The nice thing with this encoding is that we can represent several different things. Currently with the above words, we have 1 number undefined in the initial 5 bits. 29 can be an initial dot or the final one, can call the the quote-denote (QD), depending on if parser works forwards or backwards. Though for consistency it is best that it is kept as a suffix (final one), as most other things are suffixes.  $5+3+8 = 16$  Q+L+B QD has a 3 bit argument of Length. The Length is the number of 16bit fields which are quoted, if the length is 0, then the B is used as a raw byte of binary. Otherwise the B represents the encoding of the quoted bytes, mostly so that it is easier to display when debugging. The type information is external to the quotes themselves, being expressed via the available TroshLyash words. So in theory it would be possible to have a number that is encoded in UTF-8, or a string that is encoded as a floating-point-number. Though if the VM interpreter is smart then it will make sure the encoding is compatible with the type Lyash type, and throw an error otherwise.

## 2.4 Extension

This encoding already can represent over 17,000 words, which if they were all assigned would take 15bits, so it is a fairly efficient encoding. However the amount of words can be extended by increasing number of vowels, as well as tones. And it may even be possible to add an initial consonant if only one or two of the quote types is necessary. However this extension isn't likely to be necessary anytime in the near future, because adult vocabulary goes up to around 17,000 words, which includes a large number of synonyms. For instance the Lyash core words were generated by combining several different word-lists, which were all meant to be orthogonal, yet it turns out about half were internationally synonyms, so were cut down from around eight thousand to around four thousand words. It will be possible to flesh out the vocabulary with compound words and more technical words later on. Also it might make sense to supplant or remove some words like proper-names of countries.

## 2.5 Encoding Tidbit Overview

0	2	4	6	8	10	12	14	16	
C			S	V		T	F		
SRD		C			V	T	F		
LGD		C			S	V		T	
SGD			C			V		T	P
QD			QS						

Legend C Initial Consonant  
 S Secondary Consonant  
 V Vowel  
 T Tone  
 F Final Consonant  
 SRD Short Root Denote  
 LGD Long Grammar Denote  
 SGD Short Grammar Denote  
 P (optional) Phrase Parity Check tidbit  
 QD Quote Denote  
 QS Quote Sort 2.7



## 2.6 Table of Values

#	C	S	V	T	F
width	5	3	3	2	3
0	SRD	y /j/	i /i/	E	m /m/
1	m /m/	w /w/	a /ä/	MT /+//	k /k/
2	k /k/	s z /s z/	u /u/	7 /+//	p /p/
3	y /j/	l /l/	e /e/	- /+//	n /n/
4	p /p/	f v /f v/	o /o/		s /s/
5	w /w/	c j /f ʒ/	6 /ə/		t /t/
6	n /n/	r /r/	4 /i/ (U)		f /f/
7	LGD	x /x ʒ/	3 /æ/ (U)		c /f/
8	SRO				
9	s /s/				
10	t /t/				
11	l /l/				
12	f /f/				
13	c /f/				
14	r /r/				
15	LGO				
16	SRO				
17	b /b/				
18	g /g/				
19	d /d/				
20	z /z/				
21	j /ʒ/				
22	v /v/				
23	LGO				
24	SRO				
25	q /ŋ/				
26	x /ʒ/				
27	1 /l/				
28	8 /ll/				
29	QD				
30	SGD				
31	LGO				

	blank means out of bounds
E	Error signal
U	unused
MT	middle tone, no marking
QD	quote denote
SGD	short grammar word denote
SRD	short root word denote
LGD	long grammar word denote
SRO	short root word denote overflow
LGO	long grammar word denote overflow

## 2.7 Quote Sort

0	5	6	8	11	13	15
	QS					
QD	NL	R	VT	ST	SD	

### 2.7.1 definitions

**QS** quote sort

**QD** quote denote

**NL** name or literal bit

**R** region

**VT** vector thick

**ST** scalar thick

**SD** sort denote

16 tidbit					
5 tidbit	1 tidbit	2 tidbit	3 tidbit	2 tidbit	3 tidbit
quote denote	literal or name	region	vector thick	scalar thick	sort denote
definitions					
0	literal	private	1	1 byte, 8 tidbit	letter (s)
1	name	worldwide	2	2 byte, 16 tidbit	word (s)
2		preordained	4	4 byte, 32 tidbit	sentence (s)
3		coworker	8	8 byte, 64 tidbit	database
4			16		unsigned integer
5			U		signed integer
6			U		floating point number
7			3		function

The quote denote is 5 bits long, leaving 11 bits. the next 2 bits is used to indicate bit thickness of quote scalar (s), the following 3 bits is used to indicate the magnitude of the vector (s), 1 bit for name or literal

**letter** 1 \_letter

**word** word \_word

**phrase** word \_acc \_phrase

**sentence** word \_acc \_rea \_independent\_clause

**text**

**function**

**datastructure**

**named data type**

**unsigned integer** one two three \_number (291)

0 base	0 source-case	1 way-case	2 destination-case	3 location-case
1 space-context (x)	nominative-case	instrumental-case	dative-case	accusative-case
2 genitive-case	ablative-case	prosecutive-case	allative-case	locative-case
3 discourse-context	possessive-case	descriptive-case	possessed-case	relational-case
4 social-context	initiative-case	topic-case	terminative-case	vocative-case
5 surface-context (y)	causal-case	evidential-case	benefactive-case	comitative-case
6 interior-context (z)	delative-case	vialis-case	superlative-case	superessive-case
7 time-context (t)	elative-case	perlative-case	illative-case	inessive-case
	initial-time	during-time	final-time	temporal-case

Table 2.1: grammatical-case number system

**signed integer** one two three \_negatory\_quantifier \_num (-291)

**floating\_point\_number** two four \_floating\_point\_num ten \_bas one \_neg \_exponential  
\_num (2.4)

**fixed\_point\_number** two \_flo one \_num (2.1)

**rational** one \_rational three \_num (1/3)

**decimal number** ten \_bas one one \_num (11)

**hexadecimal number** sixteen \_bas eleven \_num (11)

**vector** world \_word \_and \_voc \_word two sixteen word \_vector (vector of 16  
unsigned shorts each short containing a word, intialized to repeating se-  
quence of “hello \_vocative\_case”)

In the case of a refferential, or variable name, the name can be (up to) four  
words long, that way it fits in a 64bit area — similar to a 64bit address.

## 2.8 Independent-Clause Code Name

Decided to make the independant-clause code name actually a universal hash,  
based on the sorts, cases, aspects and mood of the sentence. It’s easier that  
way.

The grammatical cases can have a table to make it easy to identify them.