colorForth and Object Oriented Programming

2023 Sep 03

Howerd Oakford www.inventio.co.uk

Abstract

Is colorForth Object Oriented? Yes and no. But first some history.

I first came across Object Oriented Programming (OOP) in the late 1990's, then around 2003 I was asked to convert a program for a medical pump to use a type of OOP called "Dependency Inversion", where the device wakes up not knowing anything about what it is or should be doing, then progressively discovers its hardware, and even later its software, and finally its purpose in life. This was an interesting experiment, but the program took around two minutes to start, instead of less than a second – this was not acceptable for a medical device (a power glitch could easily kill the patient), so the idea was dropped.
I was used to maximising the performance in embedded systems, so I never really paid much attention to this kind of OOP.

Back in 1995, the book "Design Patterns: Elements of Reusable Object-Oriented Software" defined 23 ways to design a software architecture, and talked about "MVC" Model View Controller as an architectural model – MVC was taken up by Microsoft and heralded as the way to write software, for a while. But MVC does not deal with events, and Microsoft Windows is an event-based operating system, so Microsoft dropped the idea, and seem to have removed all evidence from the internet.

Moving on to 2023, I notice that, in Germany at least, OOP is becoming popular again, but obviously not the Dependency Inversion type of OOP – this is too slow to be practical for embedded devices with limited resources. With the arrival of the internet search engines and ChatGPT there seems to be a consensus of what OOP actually means, from the OOP Wiki page https://en.wikipedia.org/wiki/Object-oriented_programming :
"Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code. "
and
"Alan Kay has differentiated his notion of OO from the more conventional abstract data type notion of object, and has implied that the computer science establishment did not adopt his notion."

An educational website https://www.educative.io/blog/object-oriented-programming says :
*"Object-Oriented Programming (OOP) is a programming paradigm in computer science that relies on the concept of **classes** and **objects**. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects."*

Diving in a bit deeper, the Udemy course https://www.udemy.com/course/der-komplettkurs-zur-modernen-c-programmierung/ lists the six key points of OOP, but because this information is in the form of an excellent German video (and behind a paywall), here is roughly the same list in English: https://medium.com/knowledge-pills/six-key-concepts-of-object-oriented-programming-oop-248f2412c736 :

Class: A class is a blueprint or template for creating objects. It defines the attributes (data) and methods (behavior) that objects of the class will have.
Object: An object is an instance of a class. It represents a real-world entity and has its own state (data) and behavior (methods).
Encapsulation: Encapsulation is the principle of hiding the internal details of an object and exposing only the necessary information through well-defined interfaces. This helps to ensure that the object's state is protected and can only be accessed or modified in a controlled way.

Inheritance: Inheritance is the mechanism by which a class can inherit attributes and methods from another class. This allows for the creation of hierarchies of classes, where subclasses can add or modify the behavior of their parent classes.
Polymorphism: Polymorphism is the ability of objects of different classes to respond to the same message (method) in different ways. This allows for more flexible and extensible code, as new classes can be added without breaking existing code.
Abstraction: Abstraction is the process of identifying the essential features of an object and ignoring the irrelevant details. This helps to simplify the design and implementation of complex systems, by focusing on the key concepts and relationships between objects.

To sum up : the precise meaning of OOP depends on who you ask, and when you ask. OOP is clearly about Objects, but exactly what these Objects are and how you Orient your program around them varies with time and author.

Going back to Alan Kay's Abstract Data Type of OOP, https://en.wikipedia.org/wiki/Abstract_data_type : *"an ADT may be defined as a 'class of objects whose logical behavior is defined by a set of values and a set of operations'".* Each colorForth token (Class) has a value (its name) and one of a set of abstract operations (its colour). I don't know if that is what Alan Kay meant by an Abstract Data Type, but it seems to fit to me.

So how does colorForth fit into the concept of OOP? Using the obvious get-out clause, that I can define Objects and how they are Oriented to be anything I like, a colorForth object is simply the 32-bit colorForth token.
The classes of Objects (seen as Abstract Data Types) are defined by the colours of the tokens, and the operations are Creation and Display of the Object by the Editor, and Execution of the Object by the compiler.
Stretching the terminology a bit, I could say that colorForth follows the Model View Controller model, with the "4-bit colour + 28-bit name" as the Model, the editor as View and the compiler as Controller.

Following the six key points of OOP, colorForth :

1. Class is the abstract colour of the Object, chosen in the editor at edit time – this creates an
2. Object, an instance of this class of Object with varying data content.
3. Encapsulation – the Object is completely encapsulated in the 32-bit colorForth token.
4. Inheritance – a green Class Object inherits the properties from its red Class Object
5. Polymorphism – at first I thought that this could only apply to typed languages, but then I realised that, in C++ at least, a Class has very similar properties to a Struct, so I could say that colorForth is actually a strongly typed language (even though you cannot define new Classes/Structs easily from within the language). This means that a word "foo" acts in one way when it is an Object of Class "Red", and another way if it is defined as an Object of Class "Green" – one name has two actions.
6. Abstraction is another reason to like OOP, even if certain implementations are inefficient, and certain definitions of OOP make it difficult to understand what is meant by "abstraction". I compare this to my favourite mathematical joke : 2 + 2 = 5 , for large values of 2 or small values of 5. Of course, we all want to be able to describe our programs in an abstract way, but this is not exclusive to OOP. With the extra 4 bits for the colorForth "colour" I intend to add structure to the system, at the fundamental Object level. This will allow a programmer to define structure in their program : sentences, paragraphs and chapters, or a Holon-style Module, Groups and Word database. This will allow a flexible level of abstraction.

But seriously…

There is actually a point to all of this word-play. In my last PDF “colorForth: The Next Generation” that I published a month ago, I described a new Data Model for colorForth that adds structure to the code. This came about from the need to allow Magenta variables to have names that include extension tokens. I’ve left that PDF content at the end of this new document, in case you missed it.

In a different context I have been studying Object Oriented techniques at work, in an effort to understand legacy C++ OOP code, and this has paid off – I now see more than five decades of evolution of the meaning of OOP, and I have got a glimpse of what some of these terms mean.

Add to the mix the Forth concept of each Forth word having its own code routine that describes the action of that Class of words, and I realised something about colorForth: for Magenta variables @ and ! are broken.

Most Forths have an implicit memory model that considers memory as a contiguous block of bytes, with an address on the stack used to fetch (@) and store (!) a data value into that address. This is a wonderfully simple concept, and it has worked well for many decades, but it is not really Object Oriented, in that the action of @ and ! are not encapsulated in the colorForth token Object. Think of having just one getter and one setter defined universally for all classes…

It is the Magenta variable again that is making life difficult. In order to save a 32 bit value in the source code we must use two tokens containing 16 bits each. Just putting any random 32 bit value inline in the code breaks the encapsulation of the colorForth token. You can no longer tell if a 32 bit “token” is really a token, or just a random data value.
The solution is easy : Magenta variables must have their own versions of @ and ! . I will think of a name in due course, or maybe use polymorphism, if that turns out to be the best way (which I doubt). This requires that literal addresses have their own different versions of @ and ! (probably exactly the same as the current ones).

This will make the colorForth system closer to the ideals of Object Oriented Programming.

Summary

For certain definitions of Object, Orientation, Class and Type, colorForth can be described as a strongly-typed Object-Oriented language.
For at least one of the definitions of OOP, the implication is that the simple memory addressing scheme of @ and ! are no longer appropriate for Magenta variables.

colorForth: The Next Generation

2023 Aug 12

Howerd Oakford www.inventio.co.uk

Abstract

colorForth : What is it for? Why am I still developing it after more than 20 years?
Because it is interesting - here are some new ideas that I intend to implement :

I need more colours!

All colorForths, from Chuck's original in 2001 to cf2023, use a token format with 4 bits for the "colour" and 28 bits for a Shannon-Fano compressed name. The "colours" are :

```
actionColourTable:     ;     * = number
    dd colour_blank    ; 0     extension token, remove space, do not change the colour
    dd colour_yellow   ; 1     yellow "immediate" word
    dd colour_yellow   ; 2  *  yellow "immediate" 32 bit number in the following pre-parsed cell
    dd colour_red      ; 3     red forth wordlist "colon" word
    dd colour_green    ; 4     green compiled cf2023
    dd colour_green    ; 5  *  green compiled 32 bit number in the following pre-parsed cell
    dd colour_green    ; 6  *  green compiled 27 bit number in the high bits of the token
    dd colour_cyan     ; 7     cyan macro wordlist "colon" word
    dd colour_yellow   ; 8  *  yellow "immediate" 27 bit number in the high bits of the token
    dd colour_white    ; 9     white lower-case comment
    dd colour_white    ; A     first letter capital comment
    dd colour_white    ; B     white upper-case comment
    dd colour_magenta  ; C     magenta variable
    dd colour_silver   ; D
    dd colour_blue     ; E     editor formatting commands
    dd colour_black    ; F
```

I need at least one more bit, so that Magenta variables can have more than 28 bits of Shannon-Fano compressed name.

The current colorForth token format :

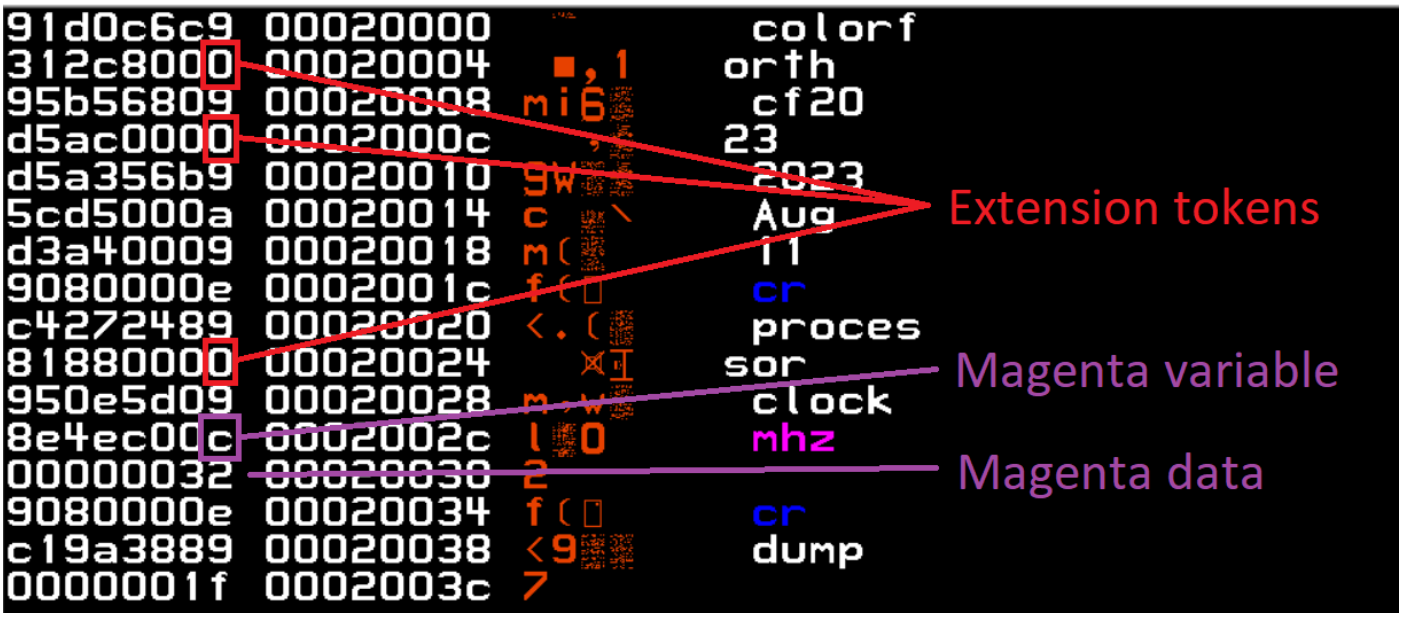
Colour			**Name**	
4 bits			28 bits Shannon-Fano / UTF-8	

Maybe it is time for an update…

Extension Tokens

Longer names are supported by the Extension “colour”, they backspace to remove the space that was added when displaying the previous token, then continue with the same colour as the previous token.

Typing ’64 block dump’ shows the start block, and how the white lowercase (colour 9) comment “colorforth” requires two tokens, “colorf” and “orth”.
The Magenta variable (colour hex c) uses the next address in the block to store its data.
It is impossible to know if the Magenta data is really data, or an extension token for the Magenta variable’s name, so Magenta variables are currently implemented without extension tokens.
If you define a Magenta variable with a name that does not fit into one token, it will not work.



Attempting to define a Magenta variable called “longname” gets a far as “longna”, then the extension token containing “me” gets interpreted as the Magenta data.
($8a000000 = -1979711488). Just don’t do it!

Encapsulation and Structure

This problem is actually an example of a larger set of problems to do with encapsulation – how do you define the beginning and end of a sequence of tokens?
For example, a string expressed as a sequence of tokens.

The solution is to add an Extension bit to the colour part of each token.

This means that you lose a bit from the name, leaving only 27 bits instead of 28.
So “rshift” no longer fits into one token, the digit ‘1’ is the ‘t’, so we could rename it “rshif”.

But why stop there?
By taking another bit from the name we could have 2 bits for “structure” :

0 0	on	carry on
0 1	dn	drop down a level
1 0	up	jump up a level
1 1	end	end of sequence

This would allow structured data such as JSON and XML files to be defined.

And, by taking another two bits we can add Version Control:

0 0	both versions
0 1	my version
1 0	the other version
1 1	deleted token (displayed as blank)

This leads to the new colorForth token format :

Colour	Structure	Version	Name	
4 bits	2 bits	2 bits	24 bits Shannon-Fano / UTF-8	

This also conveniently maps to one byte for the “colour” and meta-data and 3 for the name.

Reducing the number of bits for the name means that more tokens will require extension tokens, and this will increase the code size.

Currently, red colorForth tokens, that define a new word, only use the first token to compare with names in the wordlists. This means that names such as “myvalue1” and “myvalue2” clash – they appear to have the same name, even though their extension tokens in the source block are different.

Adding an Extension bit would make it easier to implement a comparison using more than one token. This would increase the size of the wordists in memory, and slow down the search for a word in the wordlists – an interesting tradeoff.

Structure

Each colorForth source block is an unstructured array of 32 bit tokens.
Following one of the key principles underlying colorForth, everything focuses on the token :
The editor reads it and displays it, the compiler reads it and compiles it, the interpreter reads it and interprets it, all depending on its “colour”.
Adding two bits for “Structure” allows the array of tokens to be structured :

0 0	on	carry on
0 1	dn	drop down a level
1 0	up	jump up a level
1 1	end	end of sequence

For example, for storing a JSON file (From : https://www.guru99.com/json-tutorial-example.html) :

```
{
  "student": [

     {
        "id":"01",
        "name": "Tom",
        "lastname": "Price"
     },

     {
        "id":"02",
        "name": "Nick",
        "lastname": "Thameson"
     }
  ]
}
```

Using a notation for each colorForth token [structure | **name**]

```
[on|stude] [end|nt]
    [dn|id]
        [up|01]
    [dn|name]
        [up|Tom]
    [on|lastn] [dn|ame]
        [up|Price]
    [up|]
    [dn|id]
        [up|01]
    [dn|name]
        [up|Nick]
    [on|lastn] [dn|ame]
        [on|Thame] [up|son]
    [up|]
```

Each new indentation in the JSON file is represented by a dn (down) token. Names that require more than one token use the “on” structure until the name is finished, then they can go up, down, or end this part of the structure.
A string would be a “dn” token, some “on” tokens and an “up” token.

Version Control

Again, following one of the key principles underlying colorForth, everything focuses on the token :

0 0 both versions
0 1 my version
1 0 the other version
1 1 deleted token (displayed as blank)

Bob and I want to work on our latest colorForth project.
To implement colorForth version control, tokens are marked as either “my” or “other” tokens.
Say “my” tokens are in blocks 256 to 511, and Bob’s “other” tokens are in blocks 1256 to 1511, copied from Bob’s computer.

Editing

We start off with the same code, and I edit the name of a token from “rshift” to “rshif”.
I mark the original token as “other”, and insert my edited token marked with “my”.

Pressing the F2 button toggles the Editor between displaying “my” an the “other” tokens, only one or the other is displayed, so I can see the change flashing.

Inserting

When I insert a new token, I mark it as “my” token, and also add the same token marked as “deleted”. This means that pressing F2 displays either my new token, or a blank space of the same width, so that the rest of the tokens in the block, after my edit, are not shifted around.

Deleting

Similarly to inserting a new token, deleting a token means marking it as “deleted”, again so the rest of the tokens are not shifted around, and repeatedly pressing F2 highlights only the important differences.

Pushing to Bob

At some point I will want to show Bob what I have done, and vice-versa, so “my” and the “other” code can be compared. When the version we want to keep has been selected, the Version Control fields can be replaced by “both”, and tokens marked as “deleted” can be actually deleted, and the blocks can be exported to Bob and anyone else who is interested.

Please note that both the Structure and Version Control ideas are as yet unproven.

Watch this space!

Cheers,

Howerd howerd@inventio.co.uk 2023 Aug 12

Captains Log

Added debug code to choose the number of “colour” bits used

```
%define NUM_SF_BITS 0x18            ; number of bits in each token's Shannon-Fano name field, was 0x1C,
changed to 0x18 for Next Generation 24 bit names

bits_:
    db NUM_SF_BITS

lj0:
    mov cl, [ bits_ ]
    add cl, 0x20 - NUM_SF_BITS ; 0x04
    shl dword [ esi ],cl
    ret

lj:
    call lj0
    _DROP_
    ret

full:
    call lj0
    inc dword [ v_words ]
    mov byte [ bits_ ], NUM_SF_BITS
    sub [ bits_ ], ch
    mov _TOS_, edx
    _DUP_
    Ret

word_:
    call right
    mov dword [ v_words ], 0x01
    mov dword [ chars ], 0x01
    _DUP_
    mov dword [ esi ], 0x00
    mov byte [ bits_ ], NUM_SF_BITS
```

Added debug code to show the extension tokens in a different colour

```
%define SHOW_EXTENSION_TOKENS   1   ; set true to display the extension token parts of each word

extension:  ; display an extension token, do not change the colour
    %if SHOW_EXTENSION_TOKENS
    call setSilver
    %endif

    mov _SCRATCH_, [ v_10000_iconw ]
    sub dword [ v_gr_xy ], _SCRATCH_   ; move iconw horizontal pixels back, to remove the space at the
end of the last word
    test dword [ ( edi * 4 ) - 0x04 ], 0xFFFFFFF0
    jnz showSF_EDI_
    dec edi
    mov [ v_lcad ], edi
    call space_
    call show_cursor                ; show the PacMan-like cursor
    pop edx                         ; EXIT from calling word
    _DROP_                          ; the ret below will return to the word that called  extension
    ret                             ; so it looks like it never happened
```

Printed out the Shannon-Fano “pack” code

to make it easier to follow. Sometimes a paper printout is useful to get a better overview.

```
; ********************************************************************************************************
; Shannon-Fano compression
; ********************************************************************************************************

bits_:
    db 0x1C

lj0:
    mov cl, [ bits_ ]
    add cl, 0x04
    shl dword [ esi ],cl
    ret

lj:
    call lj0
    _DROP_
    ret

full:
    call lj0
    inc dword [ v_words ]
    mov byte [ bits_ ], 0x1C
    sub [ bits_ ], ch
    mov _TOS_, edx
    _DUP_
    ret

pack0:
    add _TOS_, byte  0x50
    mov cl, 0x07
    jmp short pack1

pack_:
    cmp al, 0x10
    jnc pack0
    mov cl, 0x04
    test al, 0x08
    jz pack1
    inc ecx
    xor al, 0x18
pack1:
    mov edx, _TOS_
    mov ch,cl
.back:
    cmp [ bits_ ], cl
    jnc .forward
    shr al,1
    jc full
    dec cl
    jmp short .back
.forward:
    shl dword [ esi ],cl
    xor [ esi ], _TOS_
    sub [ bits_ ], cl
    ret

exit_:      ; exit to the quit loop
    call right
    mov _TOS_, [ v_words ]
    lea esi, [ esi + (_TOS_ * 4 ) ]
    _DROP_
    jmp quit_

word_:
    call right
    mov dword [ v_words ], 0x01
    mov dword [ chars ], 0x01
    _DUP_
    mov dword [ esi ], 0x00
    mov byte [ bits_ ], 0x1C
word1:
    call letter
    jns .forward
    mov edx, [ shiftAction ]
    jmp dword [edx+_TOS_*4]
.forward:
    test al,al
    jz word0
    _DUP_
    call echo_
    mov al, [ _TOS_ + ASCII_to_SF_table ]
    call pack_
    inc dword [ chars ]
word0:
    _DROP_
    call get_key_
    jmp short word1

right:
    _DUP_
    mov ecx, 11
    lea edi, [history]
    xor _TOS_, _TOS_
    rep stosb
    _DROP_
    ret

; ignore 0 to 3, NOP, N, spacebar, AltGr
letter:
    cmp al, 0x04
    js .forward
    mov edx, [ currentKeypadIcons ]
    mov al, [ _TOS_ + edx ]
.forward:
    ret
```

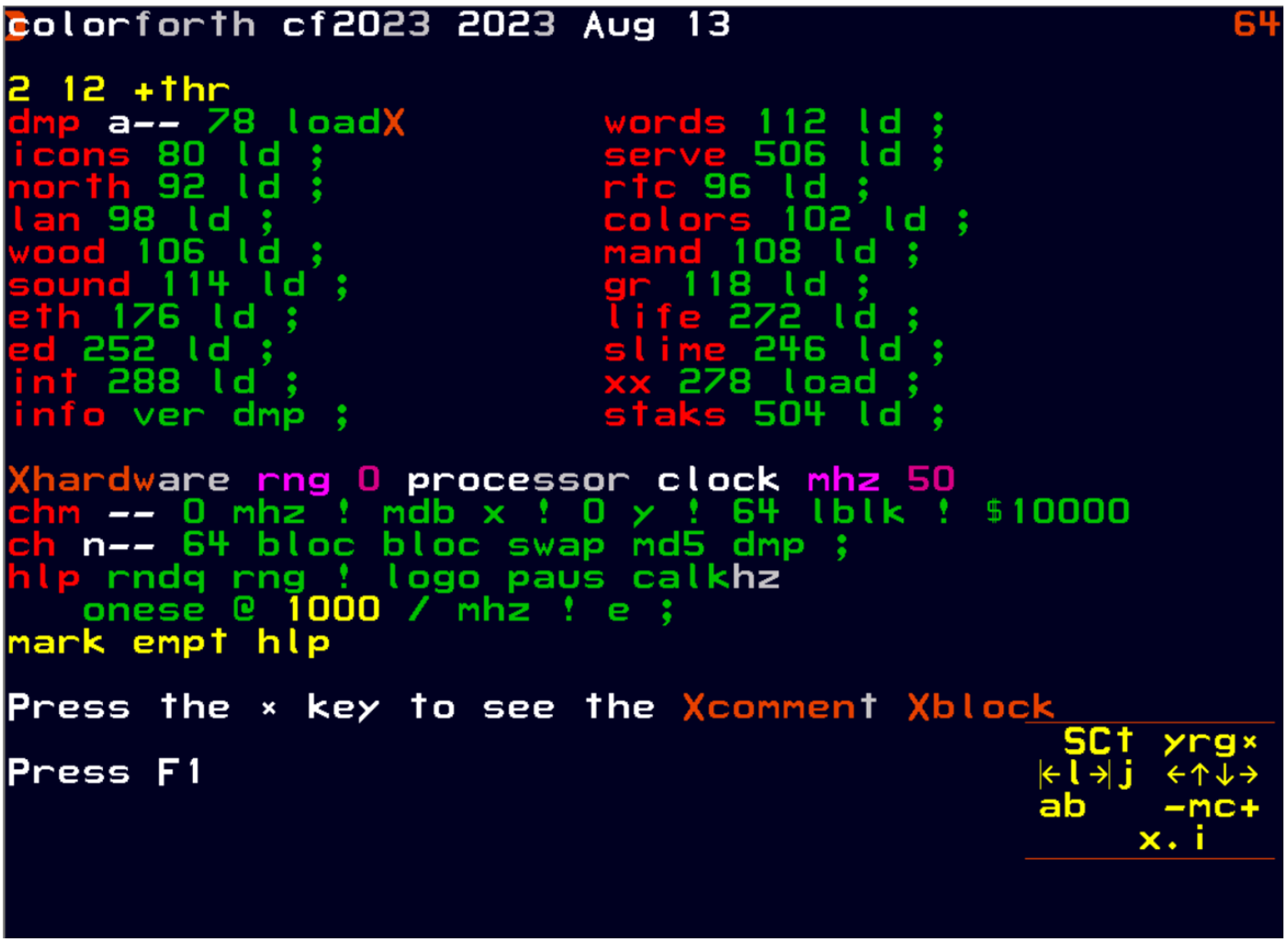
cf2023 Shannon-Fano Compression 2023 Aug 12

```
; ********************************************************************************
; keypad jump tables
; actions for the three editor state change keys : N spacebar AltGr
; ********************************************************************************

graph0:
    dd nul0, nul0, nul0, alph0
    db '  a ' ; _ _ a _ '  a ' ;

graph1:
    dd word0, exit_, lj, alph
    db 'x.a ' ;

alpha1:
    dd word0, exit_, lj, graph
    db 'x.* ' ;
```

Caught some badly formatted tokens

Something happened while editing block 64 that made “dmp” show “words”. The orange ‘X’ marks the spot.
This can happen because the old token format uses 28 bits, but the new format only 24, so any token that has a name that uses the last 4 bits is not found in the dictionary, is not visible in the editor (without this extra code), and causes mysterious “invisible” bugs. In the case of “dmp” it lost the ‘;’ and dropped through to “words”.
It also made it impossible to use the editor to delete the token!

```
showShannonFano:    ; ( token -- ) \ display the Shannon-Fano encoded token on TOS
    mov _SCRATCH_, _TOS_            ; save the token value
    and _SCRATCH_, 0x000000F0       ; mask out only the extra 4 "colour" bits
    cmp _SCRATCH_, 0x00000000
    jz .forward0
        call setOrange
        EMIT_IMM('X')   ; warn the user that we have hit problem token
    .forward0:
```



Dump of “dmp” showing the badly formatted token $000009c6
I had to use “78 load” instead of typing “dmp” !