Please consider a binary integer register named ui, of unspecified length.

Please consider a binary integer register named base that can contain either the value 10 or the value 16.

Please consider the following (here suggested, not implemented at this time) commands encoded as if Unicode characters: where X and Y are each used to represent a herein unspecified hexadecimal character that would need to be suggested in a formal application for encoding of the characters in regular Unicode.

```
U+XY000 ui:=ui*base;
U+XY001 ui:=ui*base + 1;
U+XY002 ui:=ui*base + 2;
U+XY003 ui:=ui*base + 3;
U+XY004 ui:=ui*base + 4;
U+XY005 ui:=ui*base + 5;
U+XY006 ui:=ui*base + 6;
U+XY007 ui:=ui*base + 7;
U+XY008 ui:=ui*base + 8;
U+XY009 ui:=ui*base + 9;
U+XY00A ui:=ui*base + 10;
U+XY00B ui:=ui*base + 11;
U+XY00C ui:=ui*base + 12;
U+XY00D ui:=ui*base + 13;
U+XY00E ui:=ui*base + 14;
U+XY00F ui:=ui*base + 15;

U+XY010 ui:=0;
U+XY011 base:=10;
U+XY012 base:=16;
```

Thus a sequence of the character codes may be used to enter any non-negative integer number into the ui register, provided that the register is long enough to contain the number. The first character of the sequence needs to set the value of the base being used, unless the value of the base has been set previously and there is no scope for ambiguity. The sequence needs to set the value of ui to 0 before any digits are entered..

TO DO Formal names for the characters may be needed.

Please consider a binary integer register named ai, of unspecified length.

Please consider the following commands encoded as if Unicode characters, in the same manner as before.

```
U+XY013 ai:=0;
U+XY014 ai:=ui;
U+XY015 ui:=ai;
U+XY016 ai:=ai + ui;
U+XY017 ai:=ai - ui;
U+XY018 ai:=ai * ui;
U+XY019 ai:=ai / ui;
U+XY01A ai:=ai % ui;
```

Thus a sequence of the character codes may be used to carry out integer arithmetic operations.

Please consider a binary integer array named mi, each element of the array being a binary integer of unspecified length.

Please consider a binary integer register named j, of unspecified length.

Please consider the following commands encoded as if Unicode characters, in the same manner as before.

```
U+XY01B j:=ai;
U+XY01C j:=ui;
U+XY01D ai:=j;
U+XY01E ui:=j;

U+XY01F mi[j]:=ai;
U+XY020 ui:=mi[j];
```

Please note how storage into memory is from ai and loading from memory is into ui.

* * *

2.0

2.1 The link flag

Please consider a Boolean register named link that can have the value false or the value true.

Please consider the following commands encoded as if Unicode characters, in the same manner as before.

```
U+XY021 link:=false;
U+XY022 link:=true;
U+XY023 link:=not link;
U+XY024 link:=ai > 0;
U+XY025 link:=ai = 0;
U+XY026 link:=ai < 0;
U+XY027 link:=ai > ui;
U+XY028 link:=ai = ui;
U+XY029 link:=ai < ui;
```

2.2 The if statement

```
U+XY02A if
U+XY02B then
U+XY02C elsif
U+XY02D else
U+XY02E endif;
```

The five commands if, then, elsif, else, endif; are used in the following manner, where c stands for one or more commands computing a value of link and o stands for one or more commands performing processing.

if c then o elsif c then o elsif c then o else o endif;

2.2.1 The character if

The character if does nothing. It is used for computing relative_depth, which is described later in the description of the character then.

2.2.2 The character then

The character then has different effects depending upon the state of link when the character then is obeyed.

If the value of link is true it does nothing. If the value of link is false, the value of an internal variable named relative_depth is set to zero and then a forward search is to be made. The forward search is for the next occurrence of any of the characters elsif, else or endif; when the value of relative_depth is zero. Computation then continues from that character. However, during the forward search, whenever the character if is encountered the value of relative_depth is increased by 1 and whenever the character endif; is encountered and the value of relative_depth is above zero the value of depth is decreased by 1. In this manner, multiple if structures may be nested. The character then is not a marker.

2.2.3 The character elsif

The character elsif does nothing. It is a marker for a forward search from the character then.

2.2.4 The character else

The character else does nothing. It is a marker for a forward search from the character then.

2.2.5 The character endif;

The character endif; does nothing. It is a marker for a forward search from the character then. When encountered in a forward search from a character then and the value of relative_depth is above zero then the value of relative_depth is decreased by 1.

* * *

```
U+XY02F unused at present
```

2.3 The while statement

```
U+XY030 while
U+XY031 do
U+XY032 endwhite;
```

2.3.1 The character while

The character while does nothing. It is a marker for a reverse search from the character endwhile;. When encountered in a reverse search from the character endwhile; and the value of relative_depth is above zero, the value of relative_depth is decreased by 1.

MORE NEEDED. EXPLAIN THAT relative_depth is first set to zero for a reverse search from the character endwhile;.

2.3.2 The character do

MORE NEEDED

2.3.3 The character endwhile;

MORE NEEDED

2.4 The repeat statement

```
U+XY033 repeat
U+XY034 until
U+XY035 endrepeat;
```

2.4.1 The character repeat

MORE NEEDED

2.4.2 The character until

MORE NEEDED

2.4.3 The character endrepeat;

MORE NEEDED

* * *

3.0 Program structure

```
U+XY036 start;
U+XY037 halt;
U+XY038 finish;
```

A block of object code would start with a U+XY036 character. There would then be software until a U+XY037 character. After the U+XY037 character there may, but need not, be function definitions. A U+XY038 character completes the block of software.

In use, the object code, encoded within a text file, would first be stored in a software buffer and then executed once the U+XY038 character had been received. When executing, if the U+XY037 character is encountered, the program ends and processing of the input stream continues with the next text character after the U+XY038 character.

* * *

4.0 Output to screen

A basic output to the screen is provided using the putpixel(x,y,z,r,g,b); character.

```
U+XY039 x:=ai;
U+XY03A y:=ai;
U+XY03B z:=ai;
U+XY03C r:=ai;
U+XY03D g:=ai;
U+XY03E b:=ai;
U+XY03F putpixel(x,y,z,r,g,b);
```

* * *

5.0 Floating point

af, uf
Memory array mf[1..1023].

The registers af and uf and the memory array are all of type floating point.

```
U+XY040 af:=0.0;
U+XY041 af:=af + uf;
U+XY042 af:=af - uf;
U+XY043 af:=af * uf;
U+XY044 af:=af / uf;

U+XY045 uf:=0.0;

U+XY046 mf[j]:=af;
U+XY047 uf:=mf[j];

U+XY048 link:=af > 0.0;
U+XY049 link:=af < 0.0;
```

U+XY04A through to U+XY04D are not used at present.

conversion from integer to floating point

```
U+XY04E af:=float(ai);
```

conversion from floating point to integer

```
U+XY04F ai:=entier(af);
```

U+XY050 is not used at present.

floating point functions

```
U+XY051 af:=sin(af);
U+XY052 af:=cos(af);
U+XY053 af:=tan(af);
U+XY054 af:=arcsin(af);
U+XY055 af:=arccos(af);
U+XY056 af:=arctan(af);
U+XY057 af:=arctan2(af,uf); * Compute arctan(af/uf) with regard to the
signs of both af and uf.
U+XY058 af:=exp(af);
U+XY059 af:=log(af); * The natural logarithm
U+XY05A af:=sqrt(af);
U+XY05B af:=abs(af);
U+XY05C af:=sinh(af);
U+XY05D af:=cosh(af);
U+XY05E af:=arcsinh(af);
U+XY05F af:=arccosh(af);
```

* * *

6.0 Complex numbers

Registers

az, uz

Memory array mz[1..1023]

The registers az and uz and the memory array mz are all of type complex. A program that interprets the command codes should model each complex number by using two floating point numbers.

```
U+XY060 az:=0.0;
U+XY061 az:=az + uz;
U+XY062 az:=az - uz;
U+XY063 az:=az * uz;
U+XY064 az:=az / uz;

U+XY066 az:=az * af; * this produces multiplication of a complex number
by a real number
U+XY067 az:=az / af; * this produces division of a complex number by a
real number

U+XY068 az.r:=af;
U+XY069 az.i:=af;
U+XY06A af:=az.r;
U+XY06B af:=az.i;
U+XY06C af:=radius(az); * af takes the r part of the (r,theta)
representation of az
U+XY06D af:=theta(az); * af takes the theta part of the (r,theta)
representation of az
U+XY06E mz[j]:=az;
U+XY06F uz:=mz[j];

U+XY070 az:=conj(az); * The complex conjugate
U+XY071 az:=sin(az);
U+XY072 az:=cos(az);
U+XY073 az:=tan(az);
U+XY074 az:=arcsin(az);
U+XY075 az:=arccos(az);
U+XY076 az:=arctan(az);
U+XY077 az:=arctan2(az,uz); * Compute arctan(az/uz) with regard to the
signs of both az and uz.
U+XY078 az:=exp(az);
U+XY079 az:=log(az); *The natural logarithm
U+XY07A az:=sqrt(az);
U+XY07B az:=abs(az);
U+XY07C az:=sinh(az);
U+XY07D az:=cosh(az);
U+XY07E az:=arcsinh(az);
U+XY07F az:=arccosh(az);
```

* * *

7.0 Quaternions

Some of the functions of a quaternion are quite exotic. However, at least two of them, logarithm and exponential, have a very practical application.

Quaternions are very useful for three-dimensional design packages. Both positions and rotations can each be expressed as quaternions. For example, Serif ImpactPlus uses quaternions internally.

If one has a rotation and one wishes to break it down into a number of stages, so that, for example, a rotation can be shown taking place in an animation; then taking the logarithm of a quaternion that

represents a rotation, dividing by the number of steps desired and then taking the exponential of that result, produces the rotation quaternion for one step of the animation.

Registers

aq, uq

Memory array mq[1..1023]

The registers aq and uq and the memory array mq are all of type quaternion. A program that interprets the command codes should model each quaternion by using four floating point numbers.

The following commands are added.

```
U+XY080 aq:=0.0;
U+XY081 aq:=aq + uq;
U+XY082 aq:=aq - uq;
U+XY083 aq:=aq * uq;
U+XY084 aq:=aq / uq; * this produces division with the conjugate /
conjugate multiplication term placed on the right hand side.
U+XY085 aq:=aq | uq; * this produces division with the conjugate /
conjugate multiplication term placed on the left hand side.
U+XY086 aq:=aq * af; * this produces multiplication of a complex number
by a real number
U+XY087 aq:=aq / af; * this produces division of a complex number by a
real number

U+XY08E mq[j]:=aq;
U+XY08F uq:=mq[j];

U+XY098 aq.r:=af;
U+XY099 aq.i:=af;
U+XY09A aq.j:=af;
U+XY09B aq.k:=af;
U+XY09C af:=aq.r;
U+XY09D af:=aq.i;
U+XY09E af:=aq.j;
U+XY09F af:=aq.k;
```

Some more may be needed to produce quaternion versions of modulus radius and theta.

The following are research allocations. Some are well defined, yet, for example, tan(aq); may perhaps not be clearly defined.

However, the list is included as some, such as aq:=log(aq); and aq:=exp(aq); could be useful for computing various stages on the route of a rotation path between a starting point and a finishing point.

```
U+XY0A0 aq:=conj(aq); * The quaternion conjugate
U+XY0A1 aq:=sin(aq);
U+XY0A2 aq:=cos(aq);
U+XY0A3 aq:=tan(aq);
U+XY0A4 aq:=arcsin(aq);
U+XY0A5 aq:=arccos(aq);
U+XY0A6 aq:=arctan(aq);
```

```
U+XY0A7 aq:=arctan2(aq,uq); * Compute arctan(aq/uq) with regard to the
signs of both aq and uq.
U+XY0A8 aq:=exp(aq);
U+XY0A9 aq:=log(aq); *The natural logarithm
U+XY0AA aq:=sqrt(aq);
U+XY0AB aq:=abs(aq);
U+XY0AC aq:=sinh(aq);
U+XY0AD aq:=cosh(aq);
U+XY0AE aq:=arcsinh(aq);
U+XY0AF aq:=arccosh(aq);
```

* * *

9.0 Characters

MORE NEEDED

Characters

Registers ah, uh
Memory array mh[1..1023].
The registers ah and uh and the memory array are all of type character.

```
U+XY0B0 ah:=uh;
U+XY0B1 uh:=(char) ai;
U+XY0B2 ui:=(integer) ah;
U+XY0B4 place the next character in uh

U+XY0B6 mh[j]:=ah;
U+XY0B7 uh:=mh[j];
```

* * *

10.0 Strings

MORE NEEDED

Text Strings

Please use a Unicode non-character for internally denoting the end of a text string in software implementing these codes so that the possibility that U+0000 can be included in a text string if desired is kept open.

Registers at, ht
Memory array mt[1..1023].
The registers at and ut and the memory array are all of type string.

```
U+XY0B8 at:=ut;
U+XY0B9 at:=concatenate(at,ut)
U+XY0BA ut:=(string) ah;
U+XY0BB uh:=(char) at.j; *The first character of a string is found by
using 1 as the value of j.

U+XY0BC start of text string
U+XY0BD end of text string and place text string in ut;
```

```
U+XY0BE mt[j]:=at;
U+XY0BF ut:=mt[j];
```

This section is in a handwritten typeface at this stage so as to emphasise that more research is needed on the matter of using this portable interpretable object code to display text upon a screen using fonts. Putting text on the screen is a complicated issue in that this system is intended to have a small footprint and yet also to be capable of expansion. For initial development experiments, text output to the screen could be with textsize set at 32 and font set at 0. The 32 corresponds to 24 point on the Windows platform.

U+F7F99 xtext:=ai;
U+F7F9A ytext:=ai;
U+F7F9B textsize:=ai;
U+F7F9C font:=ai;

Here font is passed from ai, an integer, maybe it should be passed from a text string?

U+F7FD1 drawtext(xtext,ytext,r,g,b,textsize,font,at);

My thinking is that using U+F7FD1 when xtext is set at 200 and ytext is set at 100 and textsize is set at 32 would have the baseline of the text start at (200,132). Starting a second line of text with xtext set at 200 and y text set at 150 and textsize set at 32 would give clearance for descenders such as in g, j, p, q and y when using most fonts.

Using U+F7FD1 does not update the values of xtext or ytext as this is a function for displaying text without referring to the font metrics.

Yet the whole point of the portable interpretable object code is to be portable, so specifying particular fonts would seem to be important. If the portable interpretable object code were used within some future version of a pdf document, then maybe linking to the font embedding mechanism of a pdf could be used. For plain text usage of the portable interpretable object code then maybe there could be a set of standardized fonts made publicly available so that intended displays would be produced. However, would that unnecessarily restrict the possibilities of using the wide range of fonts that are available today and any designed in the future? This is a big problem and needs discussion so as to arrive at the best possible solution.

Perhaps the portable interpretable object code needs a special type of variable that consists of an integer together with a text string. If the integer has a positive non-zero value then that would designate a font from a standardized set and if the integer has a value of zero then the name of the font is found in the text string. Indeed a -1 could mean to use the name and also set italics, with -2 for bold and -3 for bold italics.

* * *

11.0 Functions and calls to functions

A function is defined as either an internal function or an external function.

11.1 Internal functions

An internal function is defined by using a single define character followed by the software of the function followed by an enddefine character.

Commands to define a function.

```
U+XY0D1 define{1}
U+XY0D2 define{2}
U+XY0D3 define{3}
U+XY0D4 define{4}
U+XY0D5 define{5}
U+XY0D6 define{6}
U+XY0D7 define{7}
U+XY0D8 define{8}
U+XY0D9 define{9}
U+XY0DA define{10}
U+XY0DB define{11}
U+XY0DC define{12}
U+XY0DD define{13}
U+XY0DE define{14}
U+XY0DF define{15}

U+XY0E1 enddefine{1};
U+XY0E2 enddefine{2};
U+XY0E3 enddefine{3};
U+XY0E4 enddefine{4};
U+XY0E5 enddefine{5};
U+XY0E6 enddefine{6};
U+XY0E7 enddefine{7};
U+XY0E8 enddefine{8};
U+XY0E9 enddefine{9};
U+XY0EA enddefine{10};
U+XY0EB enddefine{11};
U+XY0EC enddefine{12};
U+XY0ED enddefine{13};
U+XY0EE enddefine{14};
U+XY0EF enddefine{15};
```

Suppose that a program is to call a function. The function is defined after the halt; command of the program yet before the finish; command. Suppose that it is desired to define function 1. This is done by using U+XY0D1 define{1} followed by whatever commands one wants in the function followed by U+XY0E1 enddefine{1};. The U+XY0E1 command is both a marker of the end of the definition and a return; command combined. A separate return; command is also provided so that, for example, a return can be made if some condition is met while obeying the function.

In order to call the function, the appropriate call command is used.

```
U+XY0C1 call[1];
U+XY0C2 call[2];
U+XY0C3 call[3];
```

```
U+XY0C4 call[4];
U+XY0C5 call[5];
U+XY0C6 call[6];
U+XY0C7 call[7];
U+XY0C8 call[8];
U+XY0C9 call[9];
U+XY0CA call[10];
U+XY0CB call[11];
U+XY0CC call[12];
U+XY0CD call[13];
U+XY0CE call[14];
U+XY0CF call[15];
```

In an application program using the portable interpretable object code, a call would be simulated in software within the sandbox of the interpreting program. There would be no access to the call and return stack of the computer.

In order to return without reaching an enddefine command there is the return; command.

```
U+XY0F7 return;
```

11.2 External functions

It is thought that it would be useful to define a function or functions within a block of software and to later use that or those functions from another program.

This would all take place within the sandbox of the system.

```
U+XY0F1 start of text string of name for defining an external function
U+XY0F2 end of text string of name for defining an external function;
U+XY0F3 enddefine[external];
U+XY0F4 start of text string of name for calling an external function
U+XY0F5 end of text string of name for calling an external function and
then call that function;
U+XY0F6 call(at); * The external call is to the function whose name is
in the at register.
```

MORE NEEDED

* * *

12.0 Pause command

The pause command is added so that the facility exists to build up a display gradually so that the build up of the display can be observed.

```
U+XY0FF pause(ai); * ai is in milliseconds.
```

* * *

13.0 Additional commands relating to the value of j.

```
U+XY100 j:=j + 1;
U+XY101 j:=j - 1;
```

```
U+XY102 link:=j > 0;
U+XY103 link:=j = 0;
U+XY104 link:=j < 0;
U+XY105 link:=j > ui;
U+XY106 link:=j = ui;
U+XY107 link:=j < ui;
U+XY108 link:=j > mi[0];
U+XY109 link:=j = mi[0];
U+XY10A link:=j < mi[0];
```

* * *

14.0 Defining a program as interrupt driven

Previously, a program has been defined as being between start; and finish; commands.

When the finish; command is reached, the program is executed.

However, when an interrupt; command is placed before the finish; command the program is flagged as an interrupt-driven program.

The original idea of an interrupt command was so that the portable interpretable object code could be used to produce illustrations in documents. However, it may be possible to extend the interrupt facility so as to make the portable interpretable object code of wider applicability, yet still within a sandbox environment. So although U+XY200 satisfies the original idea, U+XY201 through to U+XY20F are added so that the idea of using interrupt commands can be explored in research discussions.

```
U+XY200 interrupt;
U+XY201 interrupt[1];
U+XY202 interrupt[2];
U+XY203 interrupt[3];
U+XY204 interrupt[4];
U+XY205 interrupt[5];
U+XY206 interrupt[6];
U+XY207 interrupt[7];
U+XY208 interrupt[8];
U+XY209 interrupt[9];
U+XY20A interrupt[10];
U+XY20B interrupt[11];
U+XY20C interrupt[12];
U+XY20D interrupt[13];
U+XY20E interrupt[14];
U+XY20F interrupt[15];
```

* * *