ZMachine Template B/zmt

Purpose

To provide routines handling low-level Z-machine facilities.

---

§**1. Summary.** This segment closely parallels "Glulx.i6t", which provides exactly equivalent functionality (indeed, usually the same-named functions and in the same order) for the Glulx VM. This is intended to make the rest of the template code independent of the choice of VM, although that is more of an ideal than a reality, because there are so many fiddly differences in some of the grammar and dictionary tables that it is not really practical for the parser (for instance) to call VM-neutral routines to get the data it wants out of these arrays.

§2. Variables and Arrays.

```
Global top_object; ! largest valid number of any tree object
Global xcommsdir; ! true if command recording is on
Global transcript_mode; ! true if game scripting is on

Constant INPUT_BUFFER_LEN = 120; ! Length of buffer array

Array  buffer    -> 123;            ! Buffer for parsing main line of input
Array  buffer2   -> 123;            ! Buffers for supplementary questions
Array  buffer3   -> 123;            ! Buffer retaining input for "again"
Array  parse     buffer 63;         ! Parse table mirroring it
Array  parse2    buffer 63;         !

Global dict_start;
Global dict_entry_size;
Global dict_end;
```

§**3. Starting Up.** `VM_Initialise()` is almost the first routine called, except that the "starting the virtual machine" activity is allowed to go first.

```
[ VM_Initialise i;
    standard_interpreter = HDR_TERPSTANDARD-->0;
    transcript_mode = ((HDR_GAMEFLAGS-->0) & 1);
    dict_start = HDR_DICTIONARY-->0;
    dict_entry_size = dict_start->(dict_start->0 + 1);
    dict_start = dict_start + dict_start->0 + 4;
    dict_end = dict_start + ((dict_start - 2)-->0) * dict_entry_size;
    buffer->0  = INPUT_BUFFER_LEN;
    buffer2->0 = INPUT_BUFFER_LEN;
    buffer3->0 = INPUT_BUFFER_LEN;
    parse->0   = 15;
    parse2->0  = 15;
    top_object = #largest_object-255;
    #ifdef FIX_RNG;
    @random 10000 -> i;
    i = -i-2000;
    print "[Random number generator seed is ", i, "]^";
    @random i -> i;
    #endif; ! FIX_RNG
];
```

§**4. Enable Acceleration.** This rule enables use of March 2009 extension to Glulx which optimises the speed of Inform-compiled story files, so for the Z-machine it has no effect.

```
[ ENABLE_GLULX_ACCEL_R;
    rfalse;
];
```

§**5. Release Number.** Like all software, IF story files have release numbers to mark revised versions being circulated: unlike most software, and partly for traditional reasons, the version number is recorded not in some print statement or variable but is branded on, so to speak, in a specific memory location of the story file header.

`VM_Describe_Release()` describes the release and is used as part of the "banner", IF's equivalent to a title page.

```
[ VM_Describe_Release i;
    print "Release ", (HDR_GAMERELEASE-->0) & $03ff, " / Serial number ";
    for (i=0 : i<6 : i++) print (char) HDR_GAMESERIAL->i;
];
```

§**6. Keyboard Input.** The VM must provide three routines for keyboard input:

(a) `VM_KeyChar()` waits for a key to be pressed and then returns the character chosen as a ZSCII character.
(b) `VM_KeyDelay(N)` waits up to $N/10$ seconds for a key to be pressed, returning the ZSCII character if so, or 0 if not.
(c) `VM_ReadKeyboard(b, t)` reads a whole newline-terminated command into the buffer `b`, then parses it into a word stream in the table `t`.

There are elaborations to due with mouse clicks, but this isn't the place to document all of that.

```
[ VM_KeyChar win  key;
    if (win) @set_window win;
    @read_char 1 -> key;
    return key;
];
[ VM_KeyDelay tenths  key;
    @read_char 1 tenths VM_KeyDelay_Interrupt -> key;
    return key;
];
[ VM_KeyDelay_Interrupt; rtrue; ];
[ VM_ReadKeyboard  a_buffer a_table i;
    read a_buffer a_table;
    #ifdef ECHO_COMMANDS;
    print "** ";
    for (i=2: i<=(a_buffer->1)+1: i++) print (char) a_buffer->i;
    print "^";
    #ifnot;
    i=0;  ! suppress compiler warning
    #endif;
    #Iftrue (#version_number == 6);
    @output_stream -1;
    @loadb a_buffer 1 -> sp;
    @add a_buffer 2 -> sp;
    @print_table sp sp;
    new_line;
    @output_stream 1;
    #Endif;
];
```

§**7. Buffer Functions.** A "buffer", in this sense, is an array containing a stream of characters typed from the keyboard; a "parse buffer" is an array which resolves this into individual words, pointing to the relevant entries in the dictionary structure. Because each VM has its own format for each of these arrays (not to mention the dictionary), we have to provide some standard operations needed by the rest of the template as routines for each VM.

The Z-machine buffer and parse buffer formats are documented in the DM4.

`VM_CopyBuffer(to, from)` copies one buffer into another.

`VM_Tokenise(buff, parse_buff)` takes the text in the buffer `buff` and produces the corresponding data in the parse buffer `parse_buff` – this is called tokenisation since the characters are divided into words: in traditional computing jargon, such clumps of characters treated syntactically as units are called tokens.

`LTI_Insert` is documented in the DM4 and the `LTI` prefix stands for "Language To Informese": it's used only by translations into non-English languages of play, and is not called in the template.

```
[ VM_CopyBuffer bto bfrom i;
    for (i=0: i<INPUT_BUFFER_LEN: i++) bto->i = bfrom->i;
];
[ VM_PrintToBuffer buf len a b c;
    @output_stream 3 buf;
    switch (metaclass(a)) {
        String: print (string) a;
        Routine: a(b, c);
        Object, Class: if (b) PrintOrRun(a, b, true); else print (name) a;
    }
    @output_stream -3;
    if (buf-->0 > len) print "Error: Overflow in VM_PrintToBuffer.^";
    return buf-->0;
];
[ VM_Tokenise b p; b->(2 + b->1) = 0; @tokenise b p; ];
[ LTI_Insert i ch  b y;
    ! Protect us from strict mode, as this isn't an array in quite the
    ! sense it expects
    b = buffer;
    ! Insert character ch into buffer at point i.
    ! Being careful not to let the buffer possibly overflow:
    y = b->1;
    if (y > b->0) y = b->0;
    ! Move the subsequent text along one character:
    for (y=y+2 : y>i : y--) b->y = b->(y-1);
    b->i = ch;
    ! And the text is now one character longer:
    if (b->1 < b->0) (b->1)++;
];
```

§**8. Dictionary Functions.** Again, the dictionary structure is differently arranged on the different VMs. This is a data structure containing, in compressed form, the text of all the words to be recognised by tokenisation (above). In I6 for Z, a dictionary word value is represented at run-time by its record number in the dictionary, 0, 1, 2, ..., in alphabetical order.

`VM_InvalidDictionaryAddress(A)` tests whether `A` is a valid record address in the dictionary data structure.

`VM_DictionaryAddressToNumber(A)` and `VM_NumberToDictionaryAddress(N)` convert between record numbers and dictionary addresses.

```
[ VM_InvalidDictionaryAddress addr;
    if ((UnsignedCompare(addr, dict_start) < 0) ||
        (UnsignedCompare(addr, dict_end) >= 0) ||
        ((addr - dict_start) % dict_entry_size ~= 0)) rtrue;
    rfalse;
];
[ VM_DictionaryAddressToNumber w; return (w-(HDR_DICTIONARY-->0 + 7))/9; ];
[ VM_NumberToDictionaryAddress n; return HDR_DICTIONARY-->0 + 7 + 9*n; ];
```

§**9. Command Tables.** The VM is also generated containing a data structure for the grammar produced by I6's `Verb` and `Extend` directives: this is essentially a list of command verbs such as DROP or PUSH, together with a list of synonyms, and then the grammar for the subsequent commands to be recognised by the parser.

```
[ VM_CommandTableAddress i;
    return (HDR_STATICMEMORY-->0)-->i;
];
[ VM_PrintCommandWords i da j;
    da = HDR_DICTIONARY-->0;
    for (j=0 : j<(da+5)-->0 : j++)
        if (da->(j*9 + 14) == $ff-i)
            print "'", (address) VM_NumberToDictionaryAddress(j), "' ";
];
```

§**10. SHOWVERB support.** Further VM-specific tables cover actions and attributes, and these are used by the SHOWVERB testing command.

```
#Ifdef DEBUG;
[ DebugAction a anames;
    if (a >= 4096) { print "<fake action ", a-4096, ">"; return; }
    anames = #identifiers_table;
    anames = anames + 2*(anames-->0) + 2*48;
    print (string) anames-->a;
];
[ DebugAttribute a anames;
    if (a < 0 || a >= 48) print "<invalid attribute ", a, ">";
    else {
        anames = #identifiers_table; anames = anames + 2*(anames-->0);
        print (string) anames-->a;
    }
];
#Endif;
```

§**11. RNG.** No routine is needed for extracting a random number, since I6's built-in `random` function does that, but it's useful to abstract the process of seeding the RNG so that it produces a repeatable sequence of "random" numbers from here on: the necessary opcodes are different for the two VMs.

```
[ VM_Seed_RNG n;
    if (n > 0) n = -n;
    @random n -> n;
];
```

§**12. Memory Allocation.** This is dynamic memory allocation: something which is never practicable in the Z-machine, because the whole address range is already claimed, but which is viable on recent revisions of Glulx.

```
[ VM_AllocateMemory amount;
    return 0;
];
[ VM_FreeMemory address;
];
```

§**13. Audiovisual Resources.** The Z-machine only barely supports figures and sound effects. I7 only allows us to use them for Version 6 of the Z-machine, even though sound effects have a longer pedigree and Infocom used them on some version 5 and even some version 3 works: really, though, from an I7 point of view we would prefer that anyone needing figures and sounds use Glulx instead.

```
[ VM_Picture resource_ID;
    #IFTRUE #version_number == 6; ! Z-machine version 6
    @draw_picture resource_ID;
    #ENDIF;
];
[ VM_SoundEffect resource_ID;
    #IFTRUE #version_number == 6; ! Z-machine version 6
    @sound_effect resource_ID;
    #ENDIF;
];
```

§**14. Typography.** Relatively few typographic effects are available on the Z-machine, so that many of the semantic markups for text which would be distinguishable on Glulx are indistinguishable here.

```
[ VM_Style sty;
    switch (sty) {
        NORMAL_VMSTY, NOTE_VMSTY: style roman;
        HEADER_VMSTY, SUBHEADER_VMSTY, ALERT_VMSTY: style bold;
    }
];
```

§**15. Character Casing.** The following are the equivalent of `tolower` and `toupper`, the traditional C library functions for forcing letters into lower and upper case form, for the ZSCII character set.

```
[ VM_UpperToLowerCase c;
switch (c) {
    'A' to 'Z': c = c + 32;
    202, 204, 212, 214, 221: c--;
    217, 218: c = c - 2;
    158 to 160, 167 to 169, 208 to 210: c = c - 3;
    186 to 190, 196 to 200: c = c - 5 ;
    175 to 180: c = c - 6;
}
return c;
];
[ VM_LowerToUpperCase c;
switch (c) {
    'a' to 'z': c = c - 32;
    201, 203, 211, 213, 220: c++;
    215, 216: c = c + 2;
    155 to 157, 164 to 166, 205 to 207: c = c + 3;
    181 to 185, 191 to 195: c = c + 5 ;
    169 to 174: c = c + 6;
}
return c;
];
```

§**16. The Screen.** Our generic screen model is that the screen is made up of windows: we tend to refer only to two of these, the main window and the status line, but others may also exist from time to time. Windows have unique ID numbers: the special window ID -1 means "all windows" or "the entire screen", which usually amounts to the same thing.

Screen height and width are measured in characters, with respect to the fixed-pitch font used for the status line. The main window normally contains variable-pitch text which may even have been kerned, and character dimensions make little sense there.

Clearing all windows (`WIN_ALL` here) has the side-effect of collapsing the status line, so we need to ensure that `statuswin_cursize` is reduced to 0, in order to keep it accurate.

```
[ VM_ClearScreen window;
    switch (window) {
        WIN_ALL:    @erase_window -1; statuswin_cursize = 0;
        WIN_STATUS: @erase_window 1;
        WIN_MAIN:   @erase_window 0;
    }
];
#Iftrue (#version_number == 6);
[ VM_ScreenWidth  width charw;
    @get_wind_prop 1 3 -> width;
    @get_wind_prop 1 13 -> charw;
    charw = charw & $FF;
    return (width+charw-1) / charw;
];
#Ifnot;
[ VM_ScreenWidth; return (HDR_SCREENWCHARS->0); ];
```

```
#Endif;
[ VM_ScreenHeight; return (HDR_SCREENHLINES->0); ];
```

§**17. Window Colours.** Each window can have its own foreground and background colours.

The colour of individual letters or words of type is not controllable in Glulx, to the frustration of many, and so the template layer of I7 has no framework for handling this (even though it is controllable on the Z-machine, which is greatly superior in this respect).

```
[ VM_SetWindowColours f b window;
    if (clr_on && f && b) {
        if (window == 0) {  ! if setting both together, set reverse
            clr_fgstatus = b;
            clr_bgstatus = f;
            }
        if (window == 1) {
            clr_fgstatus = f;
            clr_bgstatus = b;
        }
        if (window == 0 or 2) {
            clr_fg = f;
            clr_bg = b;
        }
        if (statuswin_current)
            @set_colour clr_fgstatus clr_bgstatus;
        else
            @set_colour clr_fg clr_bg;
    }
];
[ VM_RestoreWindowColours; ! compare I6 library patch L61007
    if (clr_on) { ! check colour has been used
        VM_SetWindowColours(clr_fg, clr_bg, 2); ! make sure both sets of variables are restored
        VM_SetWindowColours(clr_fgstatus, clr_bgstatus, 1, true);
        VM_ClearScreen();
    }
    #Iftrue (#version_number == 6); ! request screen update
    (0-->8) = (0-->8) | $$00000100;
    #Endif;
];
```

§**18. Main Window.** The part of the screen on which commands and responses are printed, which ordinarily occupies almost all of the screen area.

`VM_MainWindow()` switches printing back from another window, usually the status line, to the main window. Note that the Z-machine implementation emulates the Glulx model of window rather than text colours.

```
[ VM_MainWindow;
    if (statuswin_current) {
        if (clr_on && clr_bgstatus > 1) @set_colour clr_fg clr_bg;
        else style roman;
        @set_window 0;
    }
    statuswin_current = false;
];
```

§**19. Status Line.** Despite the name, the status line need not be a single line at the top of the screen: that's only the conventional default arrangement. It can expand to become the equivalent of an old-fashioned VT220 terminal, with menus and grids and mazes displayed lovingly in character graphics, or it can close up to invisibility.

`VM_StatusLineHeight(n)` sets the status line to have a height of `n` lines of type. (The width of the status line is always the width of the whole screen, and the position is always at the top, so the height is the only controllable aspect.) The $n = 0$ case makes the status line disappear.

`VM_MoveCursorInStatusLine(x, y)` switches printing to the status line, positioning the "cursor" – the position at which printing will begin – at the given character grid position (x, y). Line 1 represents the top line; line 2 is underneath, and so on; columns are similarly numbered from 1 at the left.

```
[ VM_MoveCursorInStatusLine line column; ! 1-based position on text grid
    if (~~statuswin_current) {
        @set_window 1;
        if (clr_on && clr_bgstatus > 1) @set_colour clr_fgstatus clr_bgstatus;
        else                            style reverse;
    }
    if (line == 0) {
        line = 1;
        column = 1;
    }
    #Iftrue (#version_number == 6);
    Z6_MoveCursor(line, column);
    #Ifnot;
    @set_cursor line column;
    #Endif;
    statuswin_current = true;
];
#Iftrue (#version_number == 6);
[ Z6_MoveCursor line column  charw charh; ! 1-based position on text grid
    @get_wind_prop 1 13 -> charw; ! font size
    @log_shift charw $FFF8 -> charh;
    charw = charw / $100;
    line = 1 + charh*(line-1);
    column = 1 + charw*(column-1);
    @set_cursor line column;
];
#Endif;
```

```
#Iftrue (#version_number == 6);
[ VM_StatusLineHeight height  wx wy x y charh;
    ! Split the window. Standard 1.0 interpreters should keep the window 0
    ! cursor in the same absolute position, but older interpreters,
    ! including Infocom's don't - they keep the window 0 cursor in the
    ! same position relative to its origin. We therefore compensate
    ! manually.
    @get_wind_prop 0 0 -> wy; @get_wind_prop 0 1 -> wx;
    @get_wind_prop 0 13 -> charh; @log_shift charh $FFF8 -> charh;
    @get_wind_prop 0 4 -> y; @get_wind_prop 0 5 -> x;
    height = height * charh;
    @split_window height;
    y = y - height + wy - 1;
    if (y < 1) y = 1;
    x = x + wx - 1;
    @set_cursor y x 0;
    statuswin_cursize = height;
];
#Ifnot;
[ VM_StatusLineHeight height;
    if (statuswin_cursize ~= height)
        @split_window height;
    statuswin_cursize = height;
];
#Endif;

#Iftrue (#version_number == 6);
[ Z6_DrawStatusLine width x charw scw;
    (0-->8) = (0-->8) &~ $$00000100;
    @push say__p; @push say__pc;
    BeginActivity(CONSTRUCTING_STATUS_LINE_ACT);
    VM_StatusLineHeight(statuswin_size);
    ! Now clear the window. This isn't totally trivial. Our approach is to select the
    ! fixed space font, measure its width, and print an appropriate
    ! number of spaces. We round up if the screen isn't a whole number
    ! of characters wide, and rely on window 1 being set to clip by default.
    VM_MoveCursorInStatusLine(1, 1);
    @set_font 4 -> x;
    width = VM_ScreenWidth();
    spaces width;
    ClearParagraphing();
    if (ForActivity(CONSTRUCTING_STATUS_LINE_ACT) == false) {
        ! Back to standard font for the display. We use output_stream 3 to
        ! measure the space required, the aim being to get 50 characters
        ! worth of space for the location name.
        VM_MoveCursorInStatusLine(1, 2);
        @set_font 1 -> x;
        switch (metaclass(left_hand_status_line)) {
            String: print (string) left_hand_status_line;
            Routine: left_hand_status_line();
        }
        @get_wind_prop 1 3 -> width;
        @get_wind_prop 1 13 -> charw;
        charw = charw & $FF;
```

```
        @output_stream 3 StorageForShortName;
        print (PrintText) right_hand_status_line;
        @output_stream -3; scw = HDR_PIXELSTO3-->0 + charw;
        x = 1+width-scw;
        @set_cursor 1 x; print (PrintText) right_hand_status_line;
    }
    ! Reselect roman, as Infocom's interpreters go funny if reverse is selected twice.
    VM_MainWindow();
    ClearParagraphing();
    EndActivity(CONSTRUCTING_STATUS_LINE_ACT);
    @pull say__pc; @pull say__p;
];
#Endif;
```

§**20. Quotation Boxes.** No routine is needed to produce quotation boxes: the I6 `box` statement generates the necessary Z-machine opcodes all by itself.

§**21. Undo.** These simply wrap the relevant opcodes.

```
[ VM_Undo result_code;
    @restore_undo result_code;
    return result_code;
];
[ VM_Save_Undo result_code;
    @save_undo result_code;
    return result_code;
];
```

§**22. Quit The Game Rule.**

```
[ QUIT_THE_GAME_R; if (actor ~= player) rfalse;
    GL__M(##Quit,2); if (YesOrNo()~=0) quit; ];
```

§**23. Restart The Game Rule.**

```
[ RESTART_THE_GAME_R;
    if (actor ~= player) rfalse;
    GL__M(##Restart,1);
    if (YesOrNo()~=0) { @restart; GL__M(##Restart,2); }
];
```

§**24. Restore The Game Rule.**

```
[ RESTORE_THE_GAME_R;
    if (actor ~= player) rfalse;
    restore Rmaybe;
    return GL__M(##Restore,1);
    .RMaybe; GL__M(##Restore,2);
];
```

§25. Save The Game Rule.

```
[ SAVE_THE_GAME_R flag;
    if (actor ~= player) rfalse;
    #IFV5;
    @save -> flag;
    switch (flag) {
        0: GL__M(##Save,1);
        1: GL__M(##Save,2);
        2: GL__M(##Restore,2);
    }
    #IFNOT;
    save Smaybe;
    return GL__M(##Save,1);
    .SMaybe; GL__M(##Save,2);
    #ENDIF;
];
```

§26. Verify The Story File Rule. This is a fossil now, really, but in the days of Infocom, the 110K story file occupying an entire disc was a huge data set: floppy discs were by no means a reliable medium, and cheap hardware often used hit-and-miss components, as on the notorious Commodore 64 disc controller. If somebody experienced an apparent bug in play, it could easily be that he had a corrupt disc or was unable to read data of that density. So the VERIFY command, which took up to ten minutes on some early computers, would chug through the entire story file and compute a checksum, compare it against a known result in the header, and determine that the story file could or could not properly be read. The Z-machine provided this service as an opcode, and so Glulx followed suit.

```
[ VERIFY_THE_STORY_FILE_R;
    if (actor ~= player) rfalse;
    @verify ?Vmaybe;
    jump Vwrong;
    .Vmaybe; return GL__M(##Verify,1);
    .Vwrong;
    GL__M(##Verify,2);
];
```

§27. Switch Transcript On Rule.

```
[ SWITCH_TRANSCRIPT_ON_R;
    if (actor ~= player) rfalse;
    transcript_mode = ((0-->8) & 1);
    if (transcript_mode) return GL__M(##ScriptOn,1);
    @output_stream 2;
    if (((0-->8) & 1) == 0) return GL__M(##ScriptOn,3);
    GL__M(##ScriptOn,2); VersionSub();
    transcript_mode = true;
];
```

§28. Switch Transcript Off Rule.

```
[ SWITCH_TRANSCRIPT_OFF_R;
    if (actor ~= player) rfalse;
    transcript_mode = ((0-->8) & 1);
    if (transcript_mode == false) return GL__M(##ScriptOff,1);
    GL__M(##ScriptOff,2);
    @output_stream -2;
    if ((0-->8) & 1) return GL__M(##ScriptOff,3);
    transcript_mode = false;
];
```

§29. Announce Story File Version Rule.

```
[ ANNOUNCE_STORY_FILE_VERSION_R ix;
    if (actor ~= player) rfalse;
    Banner();
    print "Identification number: ";
    for (ix=6: ix <= UUID_ARRAY->0: ix++) print (char) UUID_ARRAY->ix;
    print "^";
    ix = 0; ! shut up compiler warning
    if (standard_interpreter > 0) {
        print "Standard interpreter ",
            standard_interpreter/256, ".", standard_interpreter%256,
            " (", HDR_TERPNUMBER->0;
        #Iftrue (#version_number == 6);
        print (char) '.', HDR_TERPVERSION->0;
        #Ifnot;
        print (char) HDR_TERPVERSION->0;
        #Endif;
        print ") / ";
    } else {
        print "Interpreter ", HDR_TERPNUMBER->0, " Version ";
        #Iftrue (#version_number == 6);
        print HDR_TERPVERSION->0;
        #Ifnot;
        print (char) HDR_TERPVERSION->0;
        #Endif;
        print " / ";
    }
    print "Library serial number ", (string) LibSerial, "^";
    #Ifdef LanguageVersion;
    print (string) LanguageVersion, "^";
    #Endif; ! LanguageVersion
    #ifdef ShowExtensionVersions;
    ShowExtensionVersions();
    #endif;
    say__p = 1;
];
```

§**30. Descend To Specific Action Rule.** There are 100 or so actions, typically, and this rule is for efficiency's sake: rather than perform 100 or so comparisons to see which routine to call, we indirect through a jump table. The routines called are the `-Sub` routines: thus, for instance, if `action` is `##Wait` then `WaitSub` is called. It is essential that this routine not be called for fake actions: in I7 use this is guaranteed, since fake actions are not allowed into the action machinery at all.

Strangely, Glulx's action routines table is numbered in an off-by-one way compared to the Z-machine's: hence the `+1`.

```
[ DESCEND_TO_SPECIFIC_ACTION_R;
    indirect(#actions_table-->action);
    rtrue;
];
```

§**31. Veneer.**

```
[ OhLookItsReal; ];
[ OhLookItsRoom; ];
[ OhLookItsThing; ];
[ OC__Cl obj cla j a n objflag;
! if (cla > 4) OhLookItsReal();
! if (cla == K1_room) OhLookItsRoom();
! if (cla == K2_thing) OhLookItsThing();
    @jl obj 1 ?NotObj;
    @jg obj max_z_object ?NotObj;
    @inc objflag;
    @je cla K1_room ?~NotRoom;
    @test_attr obj mark_as_room ?rtrue;
    @rfalse;
    .NotRoom;
    @je cla K2_thing ?~NotObj;
    @test_attr obj mark_as_thing ?rtrue;
    @rfalse;
    .NotObj;
    @je cla Object Class ?ObjOrClass;
    @je cla Routine String ?RoutOrStr;
    @jin cla 1 ?~Mistake;
    @jz objflag ?rfalse;
    @get_prop_addr obj 2 -> a;
    @jz a ?rfalse;
    @get_prop_len a -> n;
    @div n 2 -> n;
    .Loop;
    @loadw a j -> sp;
    @je sp cla ?rtrue;
    @inc j;
    @jl j n ?Loop;
    @rfalse;
    .ObjOrClass;
    @jz objflag ?rfalse;
    @je cla Object ?JustObj;
```

```
    ! So now cla is Class
    @jg obj String ?~rtrue;
    @jin obj Class ?rtrue;
    @rfalse;
    .JustObj;
    ! So now cla is Object
    @jg obj String ?~rfalse;
    @jin obj Class ?rfalse;
    @rtrue;
    .RoutOrStr;
    @jz objflag ?~rfalse;
    @call_2s Z__Region obj -> sp;
    @inc sp;
    @je sp cla ?rtrue;
    @rfalse;
    .Mistake;
    RT__Err("apply 'ofclass' for", cla, -1);
    rfalse;
];
[ Unsigned__Compare x y u v;
    @je x y ?rfalse; ! i.e., return 0
    @jl x 0 ?XNegative;
    ! So here x >= 0 and x ~= y
    @jl y 0 ?XPosYNeg;
    ! Here x >=0, y >= 0, x ~= y
    @jg x y ?rtrue; ! i.e., return 1
    @ret -1;
    .XPosYNeg;
    ! Here x >= 0, y < 0, x ~= y
    @ret -1;
    .XNegative;
    @jl y 0 ?~rtrue; ! if x < 0, y >= 0, return 1
    ! Here x < 0, y < 0, x ~= y
    @jg x y ?rtrue;
    @ret -1;
];
[ RT__ChLDW base offset;
    @loadw base offset -> sp;
    @ret sp;
];
```