NoteTaker Smalltalk Conventions

This is a working document. Send protests to: Ted Kaehler
Filed on: [IVY]< kaehler > NoteTaker.Conventions
This is version 5 on March 6, 1979
(Changes are underlined)

Object Pointer Space

A pointer to an object is called an OOP (ordinary object pointer). The 16 bits of an oop are apportioned as follows:

oop = xxxx xxxx xxxx xx00 The oop is a pointer to an object. The oop is the actual displacement in the OT of the entry of this object. The first oop is 0 and the last actual oop may be as big as 0EFFC hex.

oop = xxxx xxxx xxxx xxx1 This is the oop of an integer. The value of the integer is oop/2.

oop = xxxx xxxx xxxx xx10 (Not yet specified)

Physical Memory

The bottom few K of physical memory are local to the processor. The entire space is divided as follows (all addresses are byte addresses in hex):

```
00000   (CS points here)
        Local memory for interpreter, storage management,
        interrupt vector.
02000     (Local memory ends at 02000 assuming 8K bytes of it).
        OT is up to 15K entries of 4 bytes apiece.
10000
        Data area. (actual smalltalk objects reside here).
3FFFF   (end of physical memory 256K bytes).
FFFF0
        Bootstrap locations for 8086
FFFFF   (end of virtual memory 1M bytes).
```

Object Table (OT)

The object table contains a two word entry for each object and is indexed by the oop (plus OTbase) off the code segment register. The bit format follows (Note that I am showing the bits as high:low but in memory the bytes are reversed):

```
rrrr rrrr uuut tttv     (bytes 1,0)
ssss ssss ssss ssss     (bytes 3,2)
```

where r is REF the reference count, u are unused bits (possibly for garbage collector status), t is TOFF the core address offset, v is CLL (core longer than length, for inflated allocation), and s is TSEG the core address segment origin. We will use the LDS instruction to load the first OT word into register SI and the second word into segment register DS. Subsequent MOV instructions used to access the data fields of the object compute the effective address as DS*16 +SI. Notice that the v bit is anded out to give a zero bit there. Thus the TOFF field overlaps the TSEG field by one bit. TOFF and TSEG are normalized so that negative offsets of up to 8 bytes are allowed on SI,DS. After anding out the REF, u, and v fields of SI, the core address of the field ends up being ssss ssss ssss ssss 0000 + t ttt0 + offset.

Normalization applies to both objects and free blocks, thus the normalized core address in an OT entry will survive many allocations and frees. A normalization is only done during an allocation which must carve the core off a big block (this is also when a new OT entry filled in). Objects are renormalized during compaction.

The OT must lie below the data space. OT entries may point below the data space at the OT (for empty entries the freelist). They point into the data space (for objects and free blocks). They may not point above the data space. The end of the OT is marked by a fake entry beyond the end. It has REF=0. A fake piece of core just beyond the end of the data space contains its oop (ILLOOP). (The compacter uses this to know when to stop).

Segment Registers

CS always points to base of current 8086 code (00000). CS is used to address the OT just above the code.
SS Points to the current context during interpretation. It is switched to the top of the stack space (which grows down) while large chunks of 8086 code run.
DS changes continually and points to the segment of the current object (instance, message dict, class, literal, or object reference).
ES points to segment of current method (the byte codes).

Remember to say 'seg cs' before accessing any variables in the code segment!

Mapping

Given an OOP in BX, the code to find the object in the OT and load field 3 looks like this:

```
    SEG  CS                      ;using code segment
    LDS  SI,OTbase!BX     ;load DS,SI with oop in BX off CS + OTbase
    AND  SI,#TOFFMSK              ;AND out non-offset bits (TOFFMSK = 01EH)
    MOV  AX,3!SI          ;load AX with field 3 of object (using DS)
```

Format of an Object

The 0th word of an object is pointed at by the OT. It contains the high 10 bits of the oop of the class of the object. (The low 6 bits of the oop of a class must be zero). The low 6 bits of the zeroth word are LC, the length code. The fields follow in the next words. Objects must begin on even byte addresses.

Lengths of Objects

It is always possible to tell if an object is in use or on a freelist by looking at the REF field of the OT. A zero in the REF field means that the object is on a freelist. If so, its total length in bytes is found in its zeroth core word. (There is also a freelist of empty OT entries with no core. They each point at the next free OT entry and not at core and thus have length zero). Else REF is not zero and the object is in use. If the LC field in the zeroth core word contains a zero, the object is an octave object and the total length in bytes minus 2 is found in the word before the zeroth word. (The OT still points at the word containing the class). If LC is not zero, it contains the total number of bytes in the object (up to 63). The maximum size of an object is 64K-1 bytes. The minimum size is 4 bytes (zero field objects are octave and thus have a 4 byte header). Note that the length of any object is available without reference to any other object.

Classes

Of the 15K possible objects, 1K of them may be classes. The oop of a class has 0 in the low 6 bits. A special allocation call dispenses these valuable oops from the end of the OT empty entry freelist where they were stashed in primordial times. If a non-class would suddenly like to become a class, it must be renamed. Object renaming will be treated as a special primitive (like allInstances) and will involve a scan of all fields in the system (~1 sec). Allocation of oops will be from the bottom up with NIL being 0, false being 1, true being 2. All newly allocated object will have 0 in every field. Thus all pointer fields are initialized to NIL and all non-pointer fields are initialized to 0 bits.

New Objects

A field called Instspec in each class carries information about the instances of that class. All instances are allowed to have two kinds of fields. Named fields are fields that every instance of this class has (they known by the compiler and come first in the instance). The LONG field in Instspec encodes how many named fields instances of this class have. Extra fields are held only by this instance (they are at the end of the instance). The object creation routine adds the number of named fields to the number of extra fields to compute the length of the new instance. The call to the creation routine supplies the number of extra fields. (Many objects will have 0 extra fields. Strings and Vectors have no named fields).

An object is created with REF=0. As soon as it is pushed, it acquires the positive reference count associated with an object in use. Beware that it is in limbo between being created and being refi'd. No compaction is allowed while an object has REF=0 and is not on a freelist.

Storage Allocation

Every piece of user core is pointed at by the OT at all times. All free core and empty OT entries hang around on freelists. Lengths in words of (empty entry),2,3,4,5,6,7,8,9,10,11,12,13,14,15,16, and BIG get separate free lists. A freelist head contains an oop, which points at core. The first core location contains the byte length and the second the oop of the next entry on the list. (For the (empty entry) list, TOFF has 0FH, an illegal value, and TSEG contains the oop of the next entry). Free object must have REF=0 and CLL=0. (The deallocation routine makes sure this is true). The (empty entry) list must always have something on it. Other lists may be empty. No merging of adjacent free blocks is done. Instead compaction unifies core when fragmentation sets in.

Objects may carry more core than they use. If the core longer than length bit (CLL) is on in the OT, then the true core length is found above the top of the object (above HEADER or OLENGTH). The idea behind this length inflation is so that 'one size fits all' for small objects (avoids needlessly chopping up big blocks when a slightly too big small block is around) and so big strings may grow without many allocations. (The first system will make minimal use of CLL). The compactor shrinks all inflated objects.

Compaction

A compaction occurs between the low water mark (LWM) and the end of the data space. The low water mark is found by searching the lists of free blocks for the lowest one. For objects in the OT with core above LWM, we stuff the contents of their 0th or -1th core word into TSEG. In the core word we put their oop. In the low bit of the TOFF field (TOL) we put a 1 if they had a -1th word and a 0 if they didn't (this information would be destroyed otherwise). During the core sweep, free blocks are distinguished from objects by a zero in the REF field. The length of any object or block may still be computed. After an object is moved, the new TSEG and TOFF are computed by normalizing the current core location. No compactions are allowed while a recursive free is in progress, since some state resides in blocks that are marked free.

Locking of Objects

Each piece of machine code that wants to lock an object must reserve within itself a locking block. The block contains places for an oop, an offset, and a segment. At assembly time the routine informs the core compactor of the location of its locking block. To lock an object, the routine finds out the current core address of the object and enters both it and the oop in the block. If any object creation or storage allocation occurs between two successive uses of the core address, that address must be reloaded from the block into the machine registers. (As you might guess, whenever a compaction occurs, a cleanup routine goes through all the locking blocks and maps the oops to get the new core addresses). Objects are always free to move during a compaction. Since objects are not actually locked, they do not need to be unlocked. Format of a locking block:

    oop
    offset (low bits of core address)
    segment (high 16 bits)

Reading the Data Structure Address Space Diagrams

Enclosed are diagrams that describe all the data structures and data paths in this vmem. To help you understand them, I will walk through the first page. Find Fig 1b, OT Format and mapping to fields. The format of an OT entry is in the upper left-hand corner. The two OT words appear with the fields labeled (a label is the name of a field, not its contents). We will follow TSEG and TOFF through a memory reference to the core of this object. The index of a field within the object is added to TOFF and TSEG in the proper allignment. The result is TADR, a 20 bit quantity. The large square in the center converts the 20 address entering at the top to a selection of one out of 2↑20 possibilities. Since the bottom address bit is 0 and since only some of the values are never used (objects don't start every 2 bytes), the words 'some 2' appear in the square. Since we only have 256K out of 1M, three quarters of the values are unused. The values that are used are expanded to the left. The large open arrow with the word 'byte' in it means that the values are used as byte addresses to a memory. The words 'memstart' name the starting location of this table in the memory. The braces on the left name two parts of the table. Individual examples of places referenced in the memory are carried down to the lower part of the page. Three types of objects are shown with various fields labeled.

In general, boxes are data structure formats except squares with diagonal lines which are 'bits to choice' conversions. Braces and curved lines are correspondences between the same field in one place and another. Words in boxes are field names. Numbers beside curved lines are numbers of bits. Numbers beside vertical lines are maximums and minumums of value ranges. Numbers in boxes are actual bit values. Text outside boxes are comments.

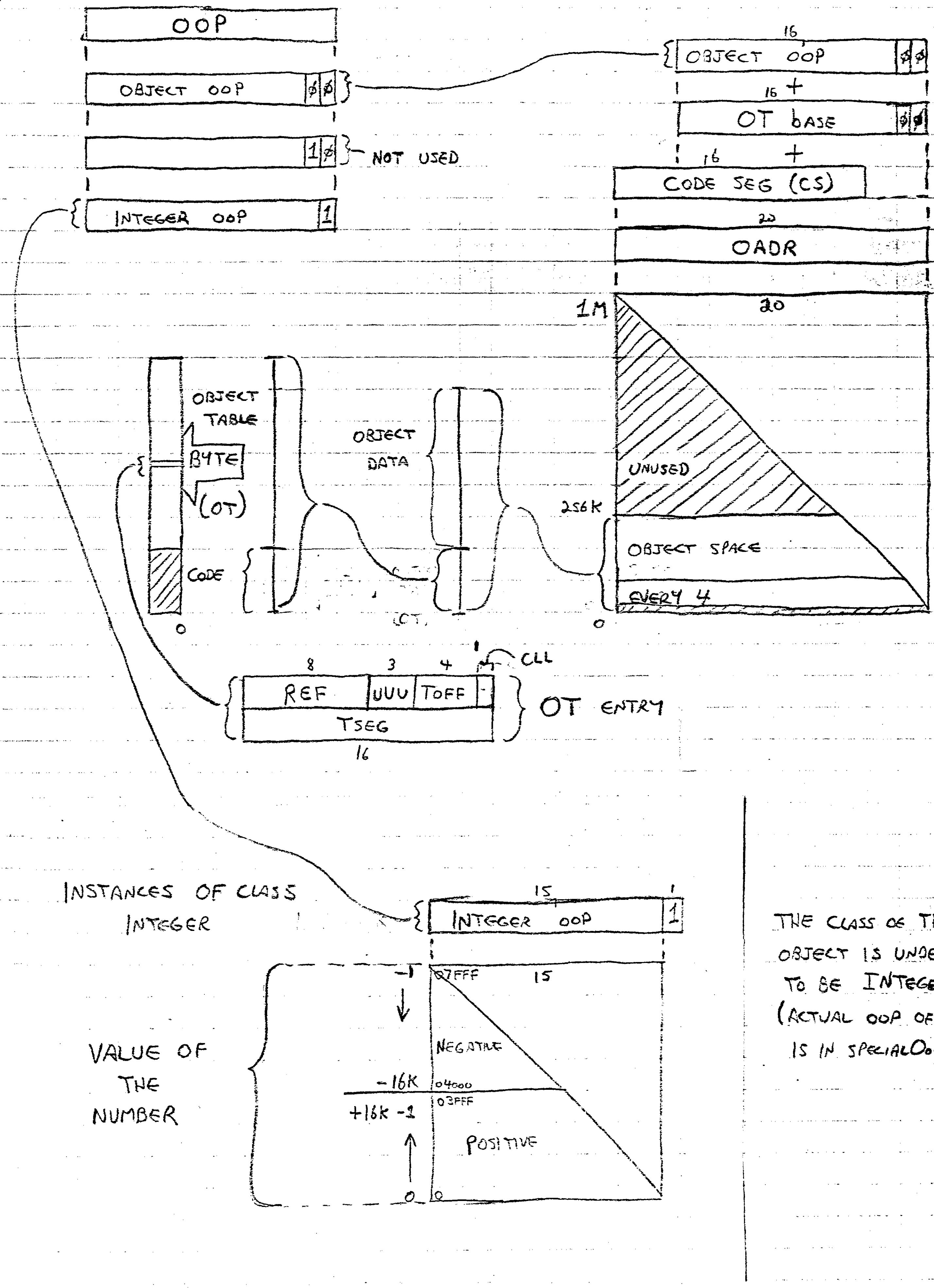
Hex Helper

```
(hex  decimal  octal  english)
(01  1  01 one)                     (02  2  02  two)
(04  4  04  four)                   (08  8  010  eight)
(010  16  020  sixteen)             (020  32  040  thirty-two)
(040  64  0100  sixty-four)         (080  128  0200 one-twenty-eight)
(0100  256  0400  two-fifty-six)    (0200  512  01000  five-twelve)
(0400  1024  02000  1K)             (0800  2048  04000  2K)
(01000  4096  010000  4K)           (02000  8192  020000  8K)
(04000 2↑14 040000  16K)            (08000  2↑15  0100000  32K)
(010000  2↑16  0200000  64K)        (020000  2↑17  0400000  128K)
(040000  2↑18  01000000  256K)      (080000  2↑19  02000000  512K)
(0100000  2↑20  04000000  1M)
```

I will distribute this memo whenever there are major additions. Ingalls Robson Fairbairn Kay

Tesler Horn Kaehler Merry McCall Krasner Deutsch (11)

FIG 1a NOTETAKER OOP FORMAT AND OT MAPPING



THE CLASS OF THIS OBJECT IS UNDERSTOOD TO BE INTEGER (ACTUAL OOP OF INTEGER IS IN SPECIALOops)

FIG. 1b NOTETAKER OT FORMAT & MAPPING TO FIELDS

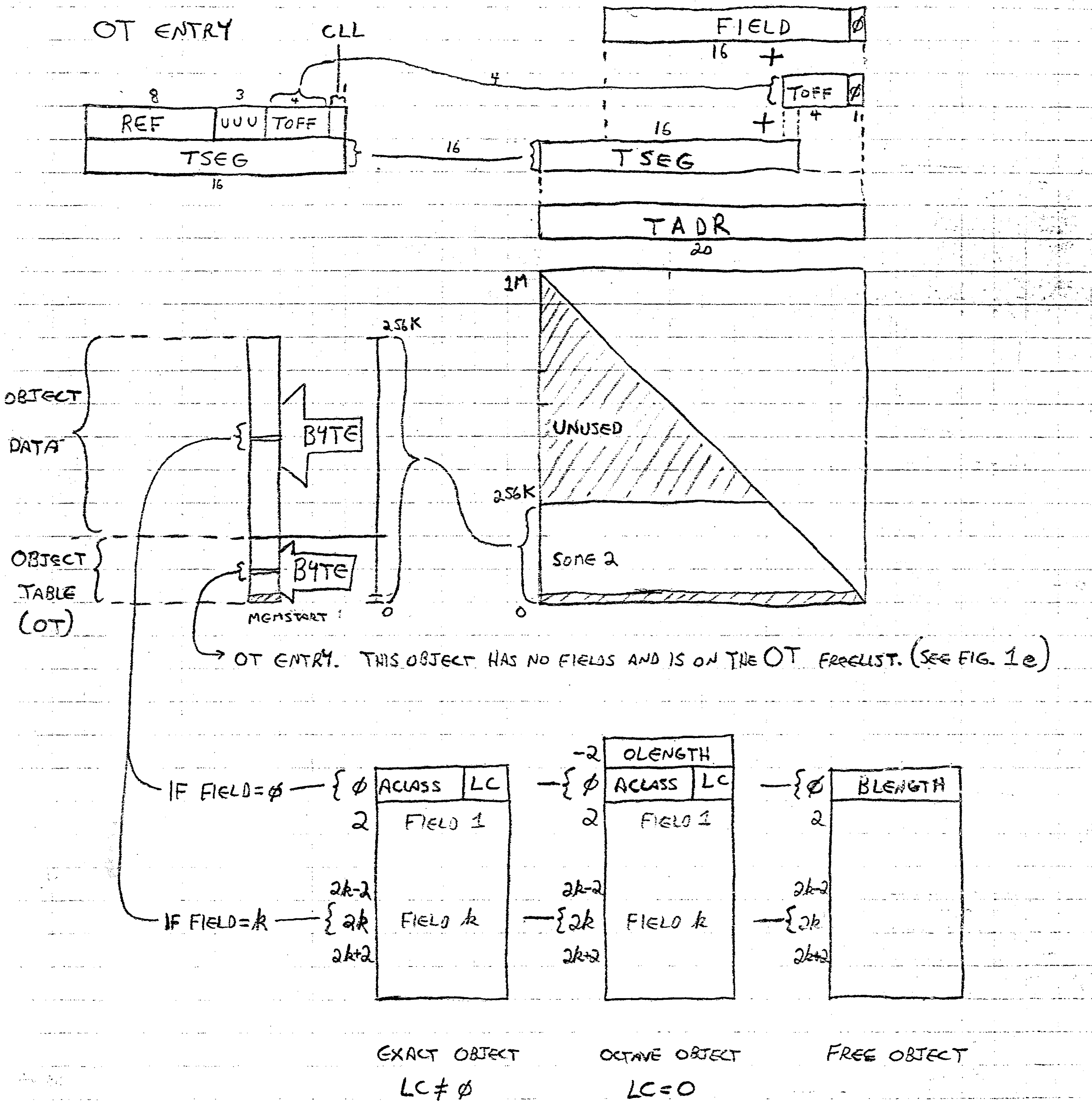
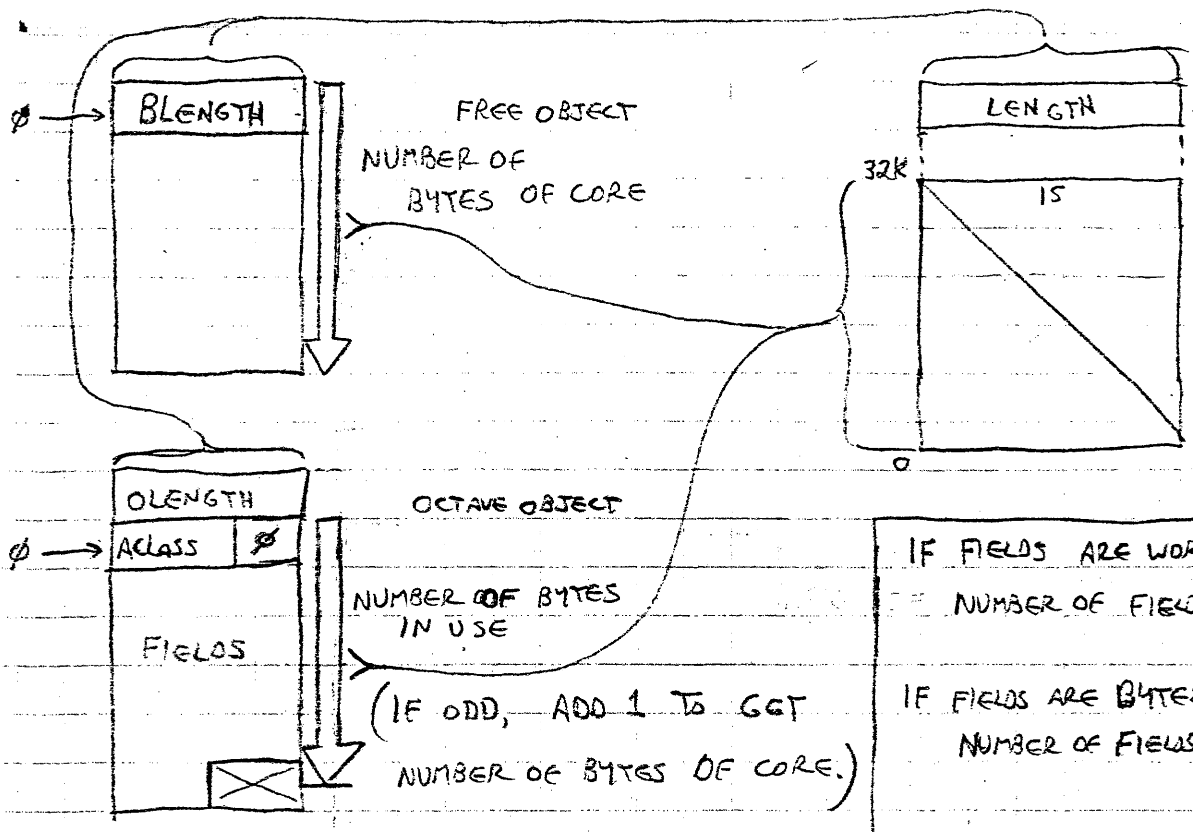


FIG. 1c

LENGTH FORMATS



IF FIELDS ARE WORDS,

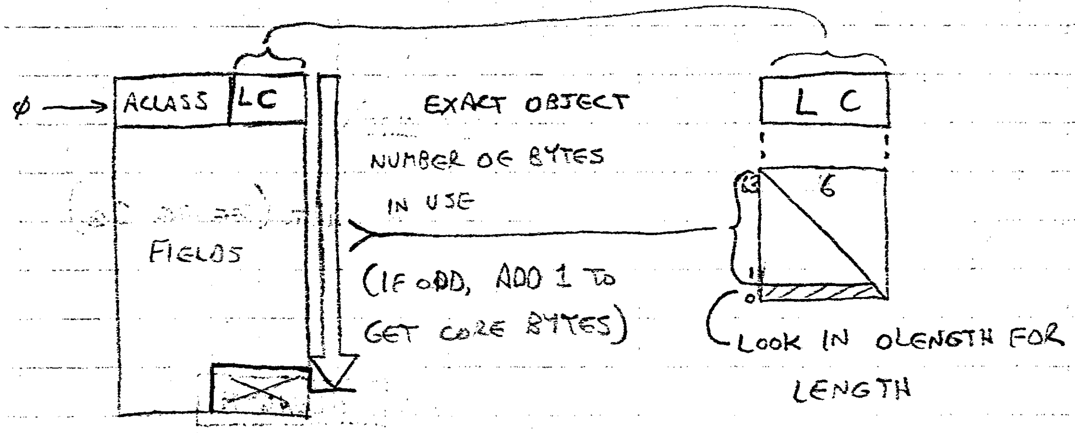
NUMBER OF FIELDS = $(\text{OLENGTH})/2 - 1$

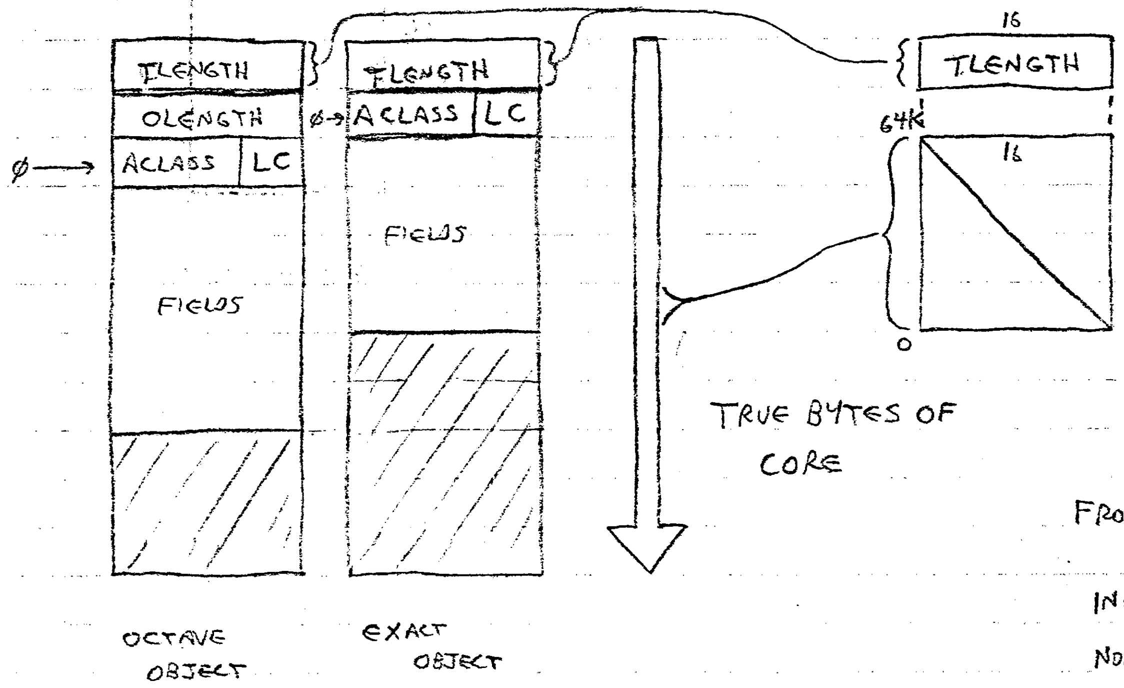
OR $(\text{LC})/2 - 1$

IF FIELDS ARE BYTES,

NUMBER OF FIELDS = $(\text{OLENGTH}) - 2$

OR $(\text{LC}) - 2$





INFLATED ALLOCATION.

IF CLL = 1, THEN

TRUE BYTES OF CORE

FOUND IN TLENGTH, ABOVE

OBJECT

FROM OT ENTRY

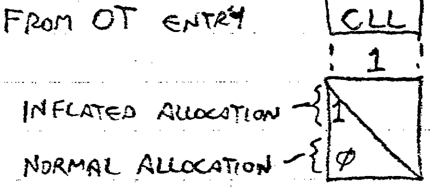


FIG 1d OOP OF CLASS AND DATA IN A CLASS

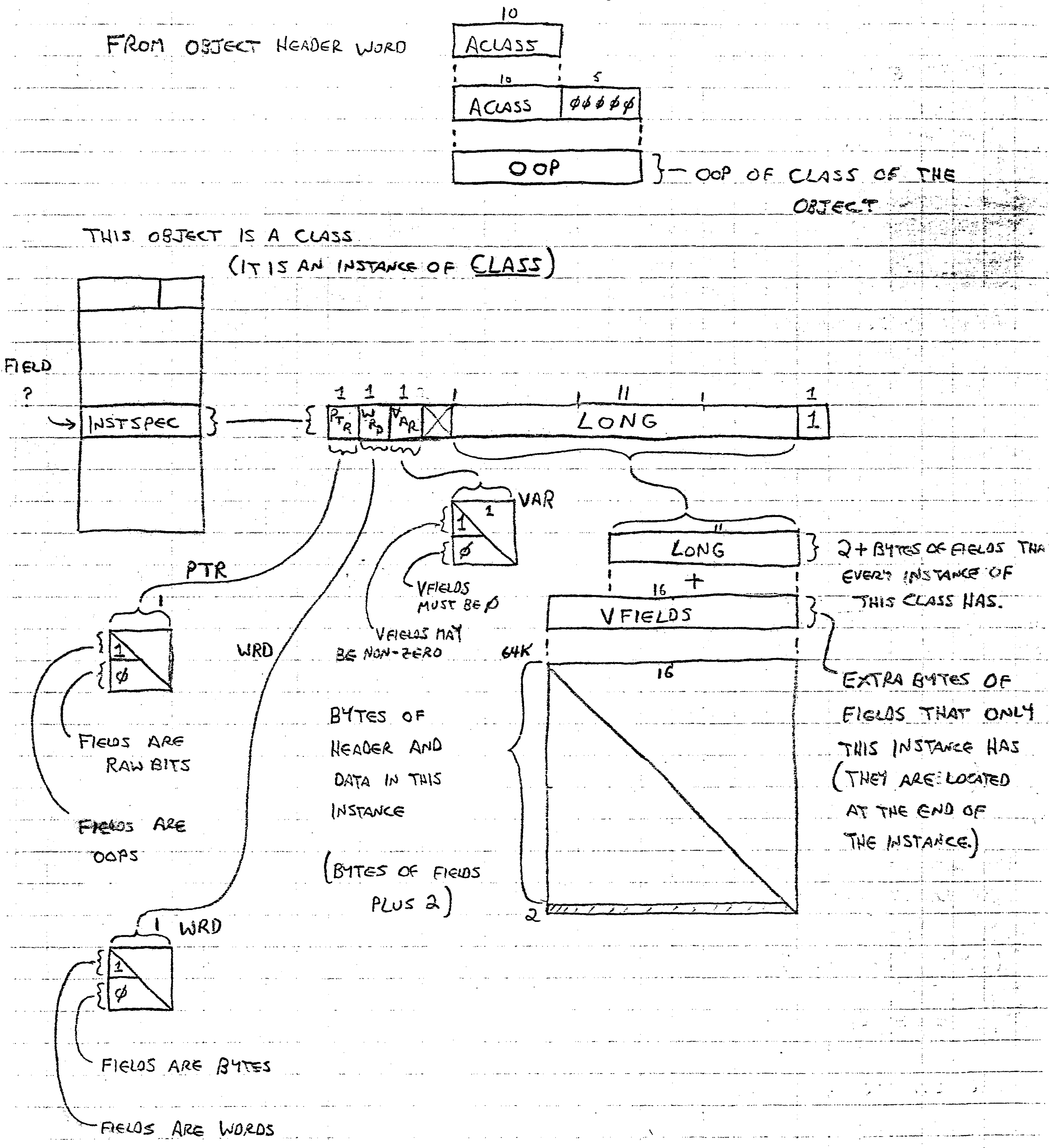
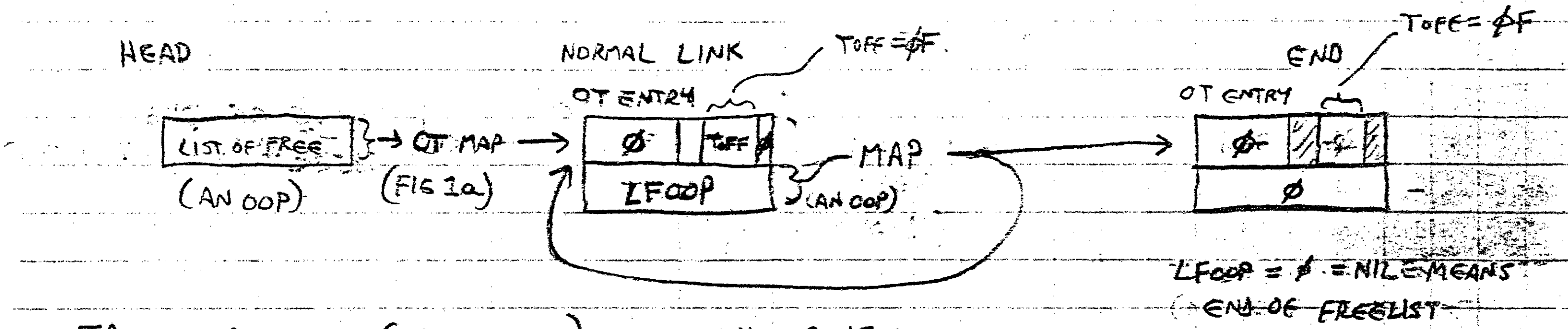
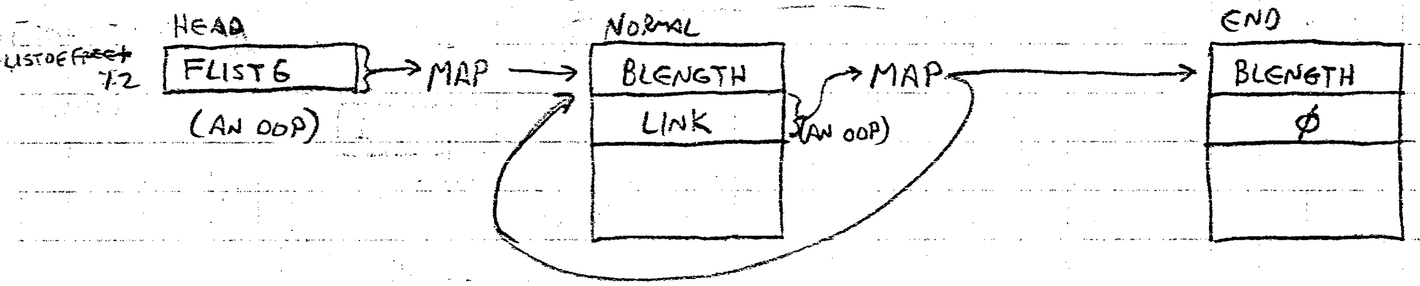


FIG 1e OBJECTS ON FREELISTS.
("MAP" TRANSFORMS AN OOP TO ITS FIELDS IN CORE, THAT IS, THE COMBINATION OF FIG 1a AND 1b.)

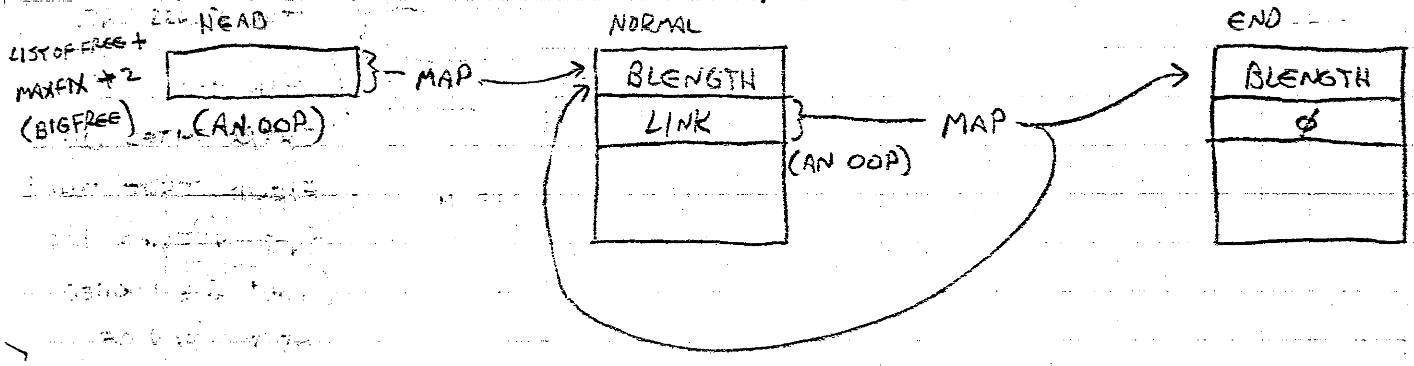
EMPTY ENTRIES IN Object TABLE. (OT)



FREE OBJECT (WITH CORE) LENGTH 2-17 WORDS



BIG FREE OBJECT LENGTH >17 WORDS



NO CONSTRAINTS ON THE ORDER OF ANY FREELIST

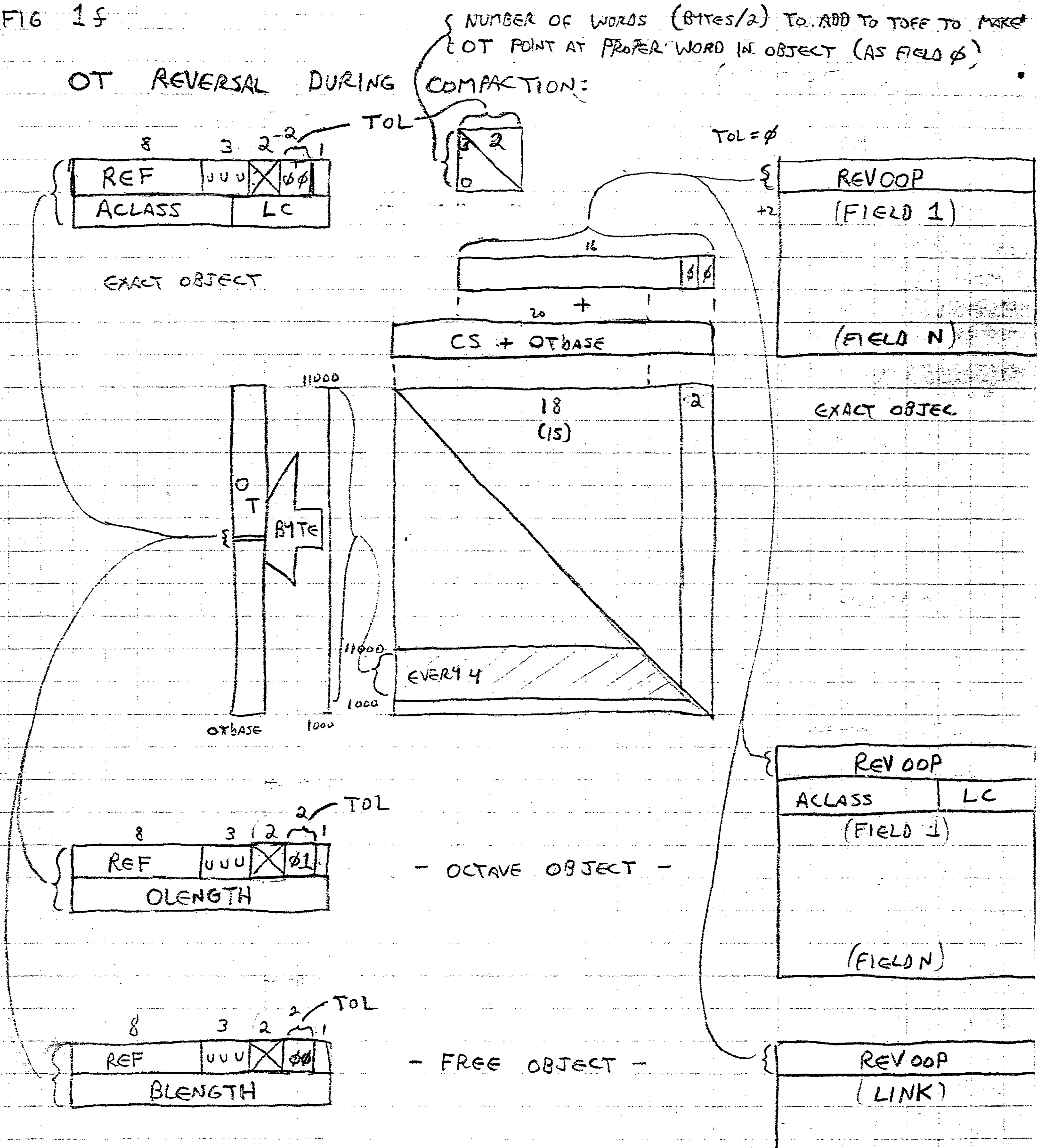
(EMPTY ENTRIES HAS FREE CLASS ENTRIES ON THE END)

NO FREE BLOCK MAY HAVE BLENGTH $\geq (64K - 32)$ BYTES

FOR ANY FREE OBJECT, REF = Ø, CLL = Ø

FIG 1f

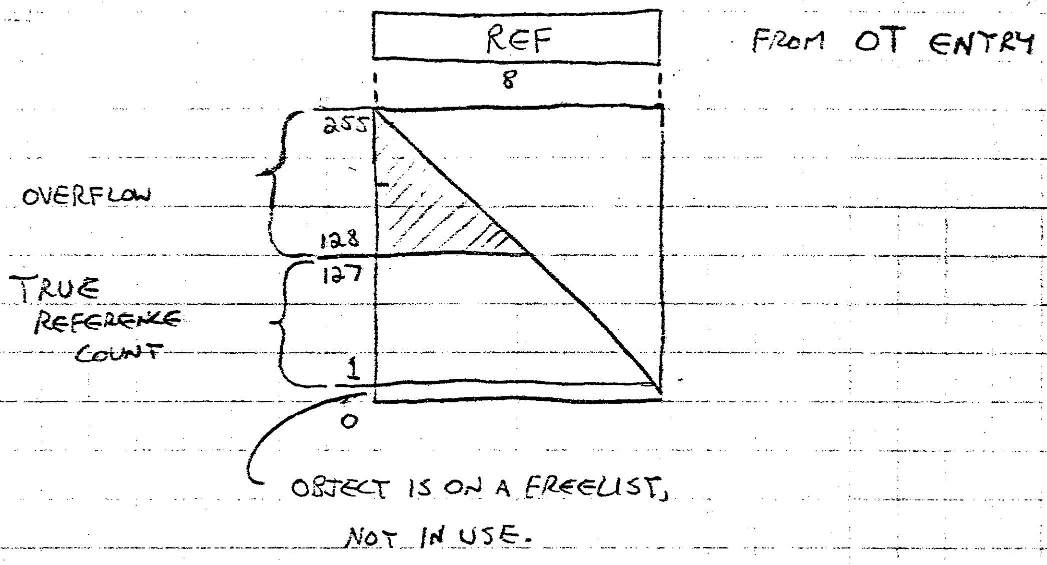
OT REVERSAL DURING COMPACTION:



FOR INFLATED ALLOCATION OBJECTS,
SAME AS FOR NORMAL OBJECTS.
EXCEPT CCL=1, OT TENT ← TLENGTH

FIG. 1g REFERENCE COUNT

REFERENCE COUNT



OVERFLOW: THESE OBJECTS ARE STUCK WITH OVERFLOW REFCT. THEY WILL NEVER BE FREED. OVERFLOW VALUE IS RANDOM. WHEN IT GETS OUT OF RANGE, IT IS RESET TO MIDDLE.

TRUE REFCT: ACTUAL NUMBER OF OBJECTS POINTING AT THIS OBJECT.

FIG 1h TOFF VALUES

TOFF IS NORMALIZED TO ALLOW OFFSETS OF ± 8 BYTES WITHOUT OVERFLOW OR UNDERFLOW.

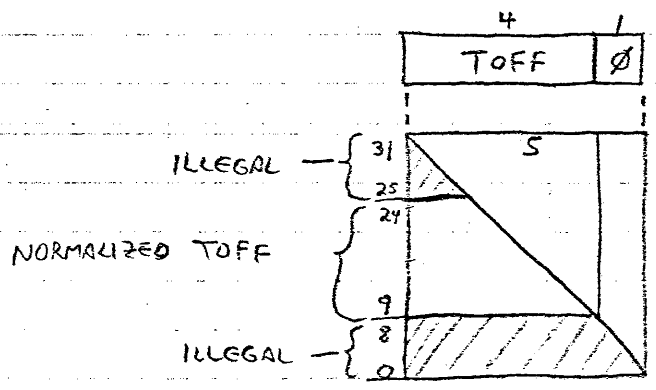


FIG 1i OBJECT FORMAT DURING A RECURSIVE FREE.

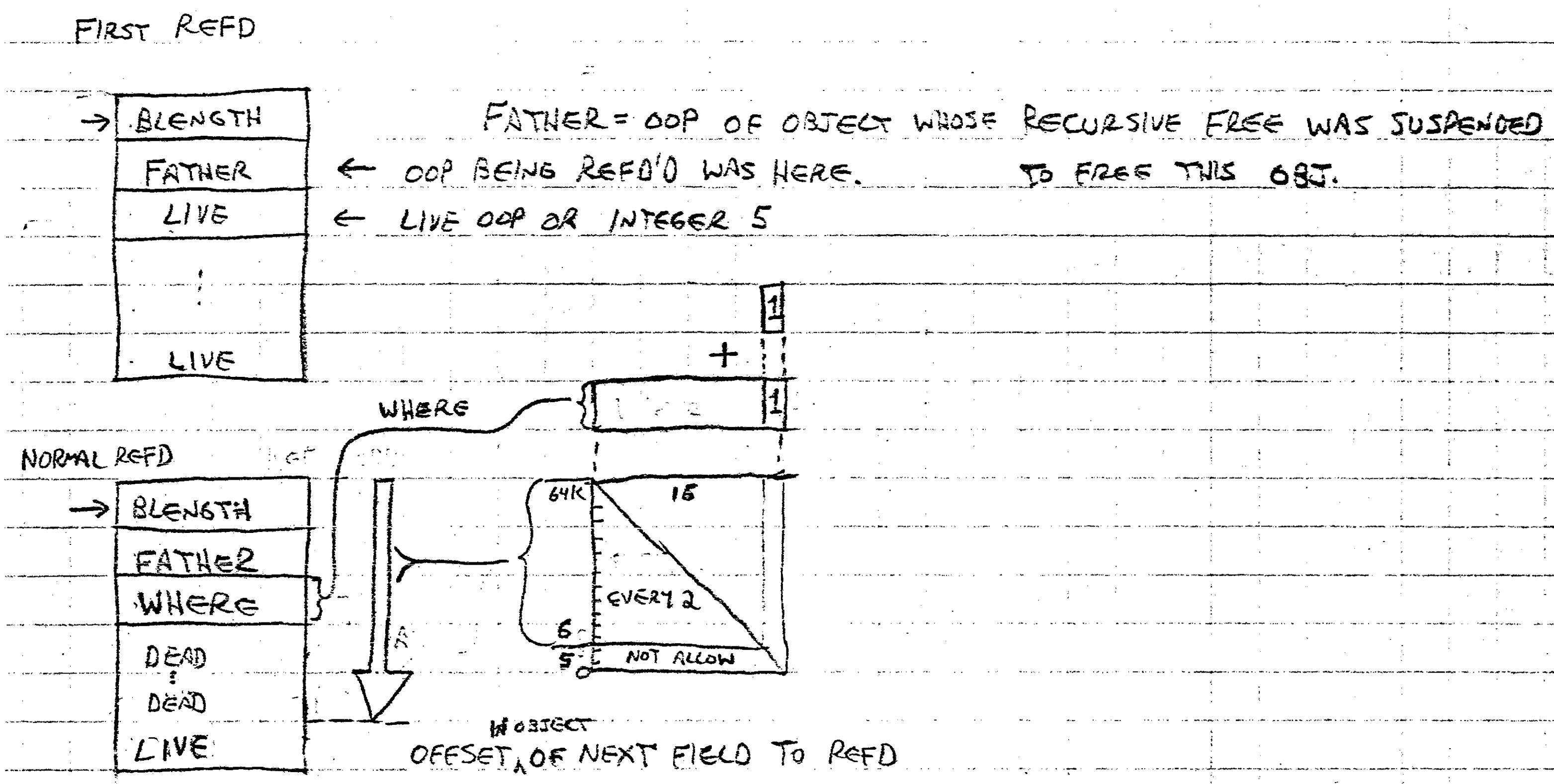
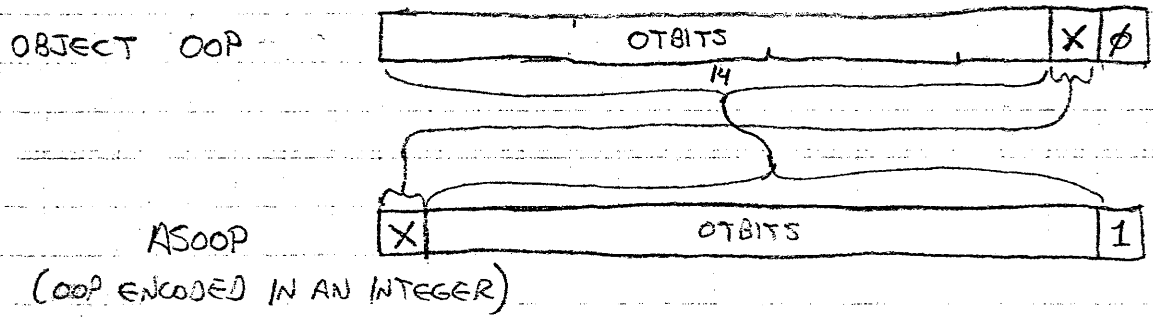


FIG 1j ENCODING OF AN OOP IN AN INTEGER



OBJECT OOP
OTBITS
X
ϕ
14
ASOOP
(OOP ENCODED IN AN INTEGER)
X
OTBITS
1