M. Levin
7 July 1965

INFORMATION INTERNATIONAL INC.

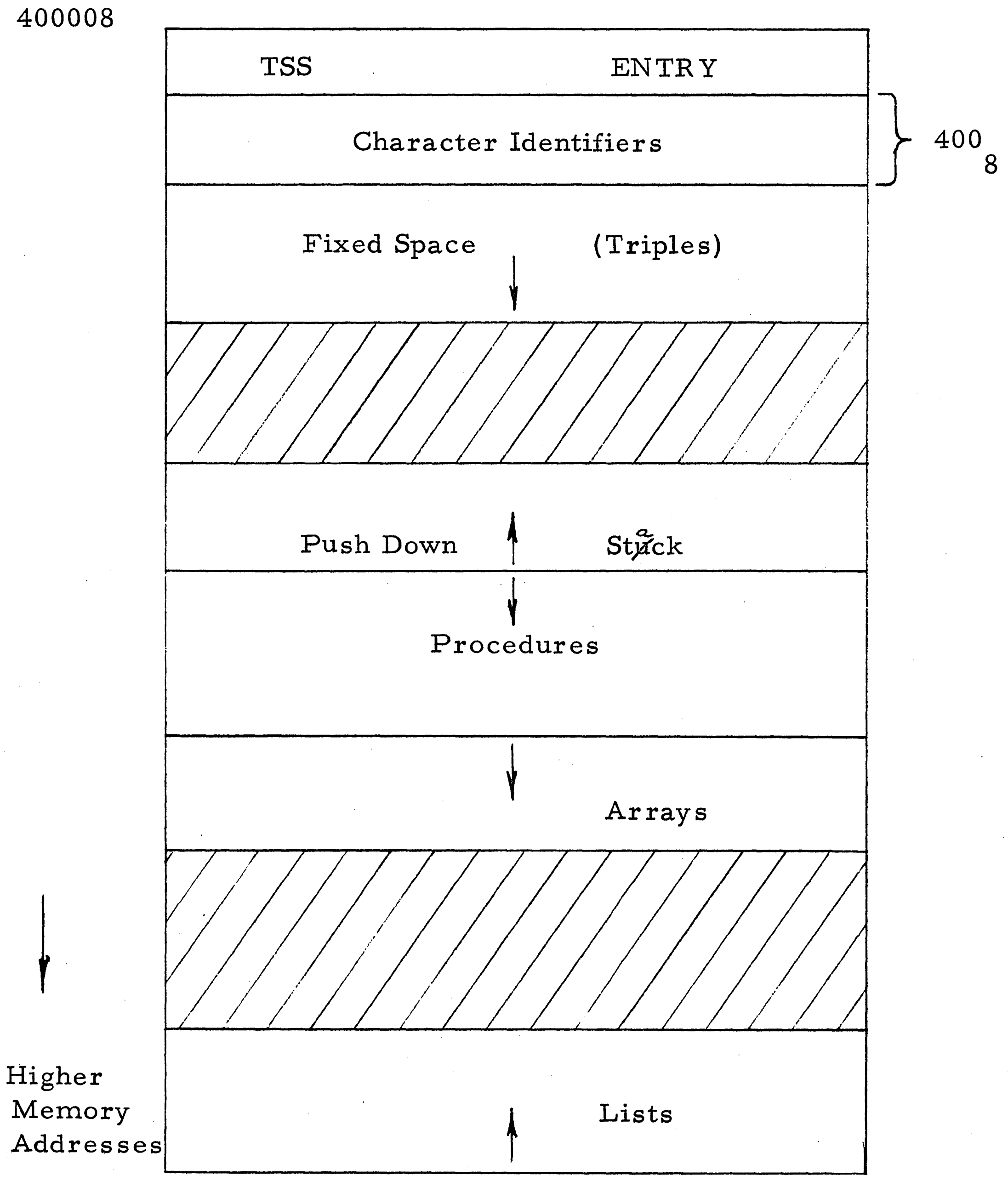
LISP II MEMO #9

Internal Storage Conventions for LISP II

I TYPE TABLE

0			
1			
2	Boolean (22 by location)	42	Boolean Array (62 by location)
3	Octal (23 by location)	43	Octal Array (63 by location)
4	Integer (24 by location)	44	Integer Array (64 by location)
5	Real (25 by location)	45	Real Array (65 by location)
6	String		
7	Identifier		
10	Symbol (30 by location)	50	Symbol Array (70 by location)
11	Formal (31 by location)	51	Formal Array (71 by location)
12	Fluid Cell	75	Function Sub-specified
13	Quote Cell	76	Tables
14	Function Descriptor	77	Table Entry Types

100-7777	Table Entry Types
10000-37777	Virtual Types


400008
TSS
ENTRY
Character Identifiers
400
8
Fixed Space
(Triples)
Push Down
St~~y~~ack
a
Procedures
Arrays
Higher
Memory
Addresses
Lists

II FIXED SPACE

Fig. 1 Character Identifier

07	VF-List	T	P-List

T=22 for characters A through Z

T=32 for other characters

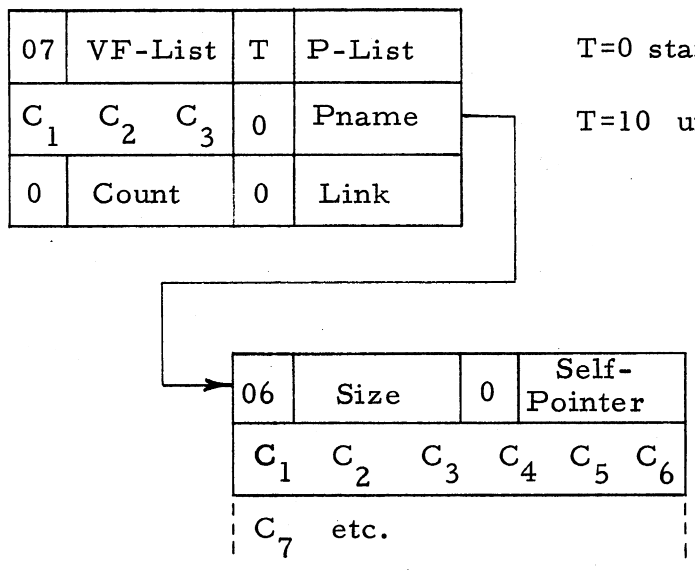
Fig. 2 Identifier - less than 7 characters in name

07	VF-List	T	P-List
C_1 C_2	C_3	C_4	C_5 C_6
0	Count	0	Link

T=04 standard spelling

T=14 unusual spelling

Fig. 3 Identifier - 7 or more characters in name



T=0 standard spelling

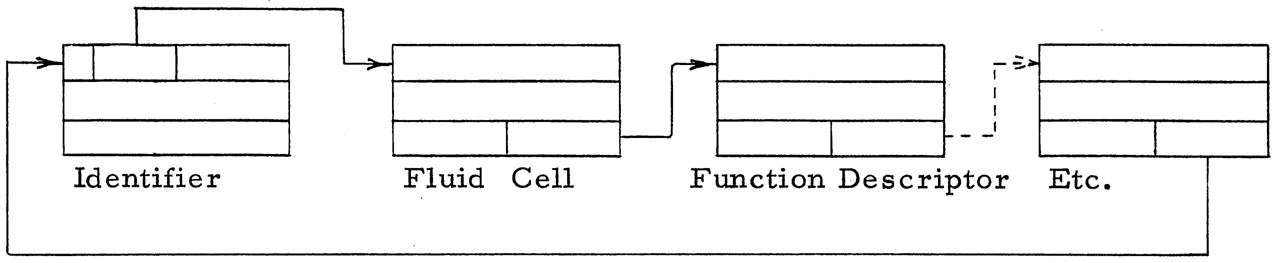
T=10 unusual spelling

This is a string located in array space.

07 is the prefix that specifies identifiers.

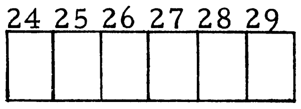
The VF-list (variable and function list) is a threaded list of all fluid cells and function descriptors of this identifier. Note its peculiar structure.

Fig. 4



The T (tag) field of an identifier has bits with the following significance.

Fig. 5



Bit 24. Used by the garbage collector. It is 0 during normal execution.

Bit 25. This bit is 1 for character identifiers, 0 for other identifiers.

Bit 26. This bit is 1 for identifiers with non-standard spelling. An identifier has normal spelling if and only if the first character is a letter, and all other characters are letters, digits, or the period.

Bit 27. This bit is to 1 when the pname of the identifier consists of 6 or less characters stored in the identifier itself.

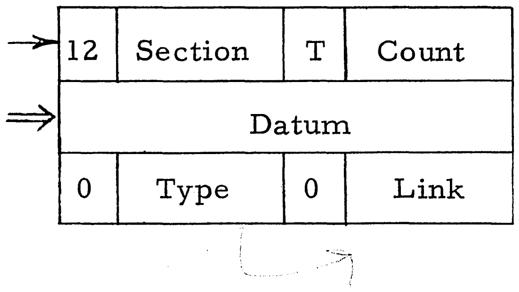
Bit 28. This bit is 1 when the identifier is not to be collected as garbage at any time.

When an identifier has more than 6 characters in its pname, then the pname is stored as a string.

Excess spaces in pnames are filled with the illegal code 376_8.

The link field is used to string identifiers together on a bucket of the oblist.

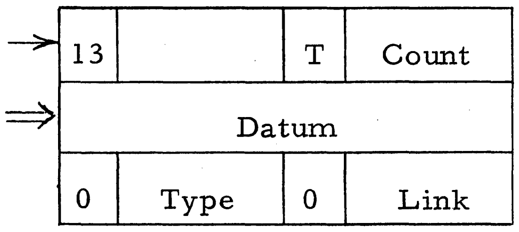
Fig. 6 Fluid Cell



T=0

FORMAL CELL LIKE FUNCTION CELL BUT CODE 16 AND DATUM LIKE FLUID CELL

Fig. 7 Quote Cell



T=0 ?

Fig. 8 Function Descriptor

This is an octal array located in array space.

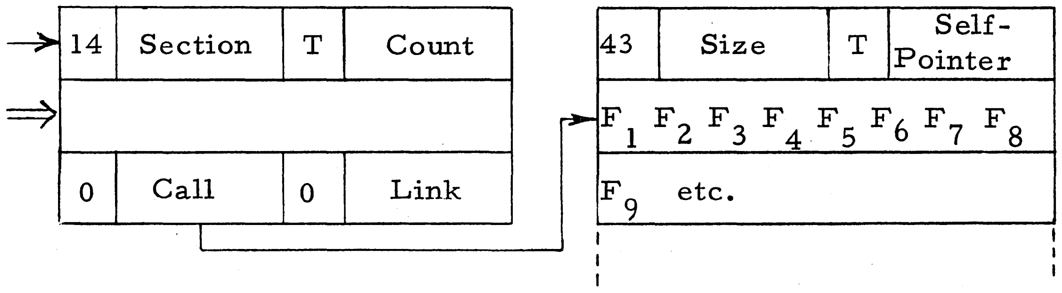
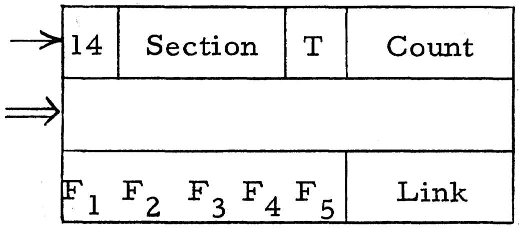


Fig. 9 Function Descriptor (3 or less arguments)

5 or less bytes to descriptor



The section specifies the section to which the variable or function belongs, or else is 0, indicating a global variable or function.

The count field specifies how many references to the cell exist in code. This is incremented when a procedure is loaded, and decremented when a procedure is excised.

The link field strings the function and fluid cells of any identifier into a threaded list as per Figure 4.

Pointers from pure procedure into fixed space are indicated as =⟩ . Pointers of other kinds are indicated as → .

The type of a fluid or quote cell is as per the type chart.

The datum in these cells may be absolute one-word types, or pointers.

The call of a function descriptor gives type conversion information. The first field specifies the number of arguments. The second field specifies the type of the value of the function. Succeeding fields specify the types of the parameters in order.

The six bit codes used for all but the first field are those on the type list.

If five or less fields are required, then the call is not remote, but occurs in the function descriptor itself. In this case, bit 25 of the tag will be set to 1.

Bit 24 of the tag is used by the garbage collector and is normally zero.

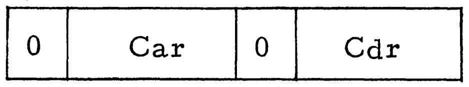
The middle word of the function descriptor normally contains a pointer in the address to the starting location of code. If it points to the middle word of another function descriptor, then the indirect bit, (bit 25) is set to 1.

If the function is not in core memory, if it is undefined, if it is being traced, or for several other reasons, the address may be to some trap location. The left half of the word may then contain other information. In these situations, the 26 bit of the <u>first</u> word of the function descriptor is set to 1.

Bit 27 is set to 1 when the second word contains an instruction to be executed. This is done only in unusual situations.

III LIST SPACE

Fig. 10



LISP node

IV ARRAY SPACE

Fig. 11 String

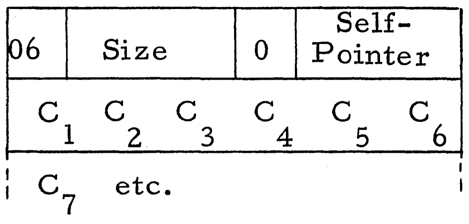
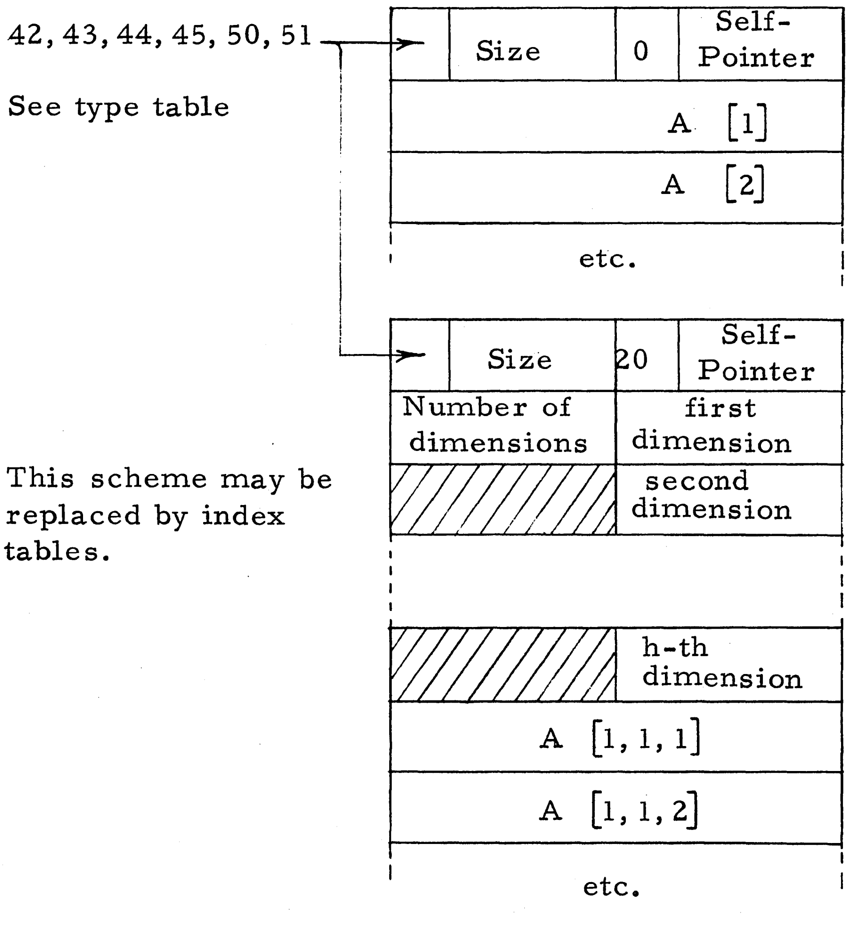


Fig. 12 Arrays of 1-word elements



See type table

1 dimensional array

Multi-dimensional array

This scheme may be replaced by index tables.

Strings contain 6 8-bit ASCII characters per word. Unused bytes in a word are filled with the illegal character 376_8.

The size field specifies the total number of words in the string or array.

The self pointer is contained in the same position in all arrays. It is used by the garbage collector.

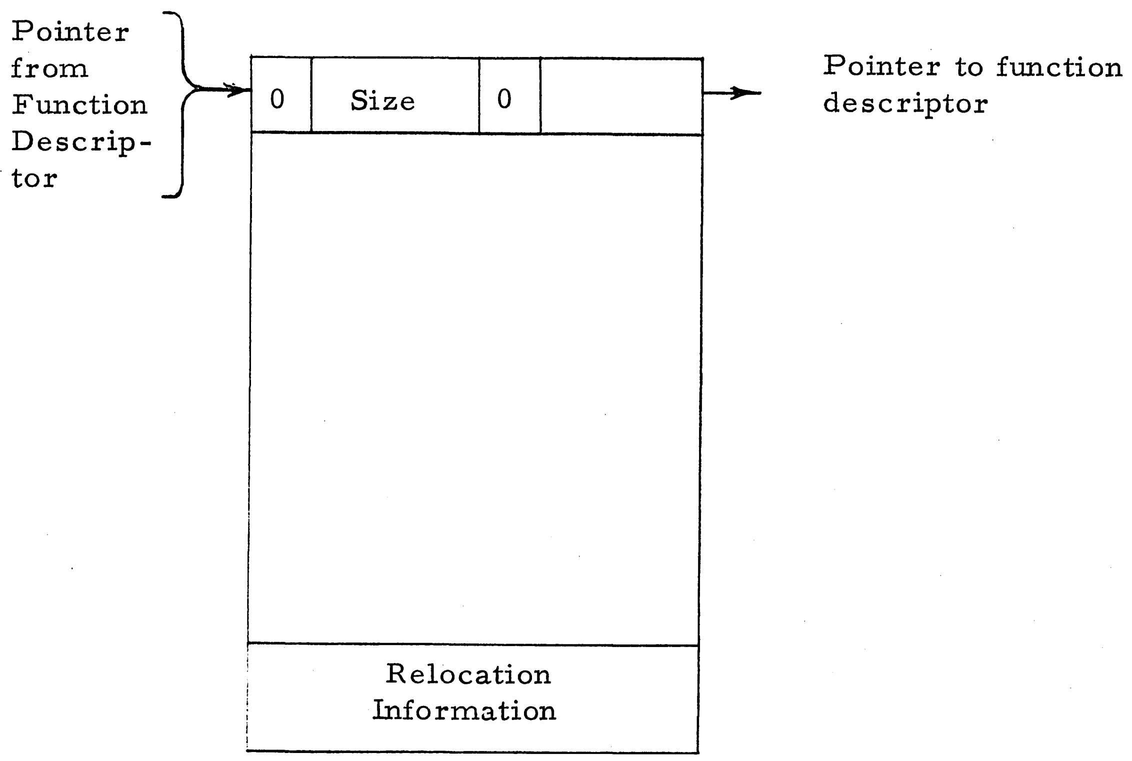
Symbol arrays and symbol variables contain pointers. If these point to the heads of arrays, lisp nodes, or first words of triples in fixed space, then the pointer is in the address, and the rest of the word is empty.

Pointers to the second word of function descriptors have the indirect bit set. Pointers into arrays have a pointer to the head of the array in the left portion of the word.

Formal variables, and formal array elements only have pointers to the middle words of function descriptors. It is legitimate to branch indirect to such a variable.

V PROCEDURE SPACE

Fig. 13 Procedure


Pointer from Function Descrip- tor
0
Size
0
Pointer to function descriptor
Relocation Information

Relocation information (2 bits per word) is packed from left to right starting with the last word and working backwards.

0 - means no relocation or count.

1 - if address is local then it is relocatable. If address is to fixed space, then increment count when loading, and decrement count when excising.