B.3

(95)

DATA LAYOUTS IN ENPL FOR THE GE636

M. D. McIlroy
May 12, 1965

These data layouts will be adhered to for argument passing to external procedures by both ENPL and all later versions of NPL for the GE636 system.

SPECIFIERS

Some data in NPL will be referenced directly by the object code. Some will be referenced indirectly through specifiers that give information about the layout of the data as well as its location. Specifiers are used for

- label data and entry names
- string data
- arrays
- structures
- and probably for files

Specifiers always consist of an ITS pair pointing to an address for the data plus additional information which we shall call dope. The dope for different sorts of data may be different.

SCALAR DATA

Single and double precision floating point numbers will be stored in the natural forms. Double precision floating point quantities will sit at even locations. Single or double precision will be assigned to cover the declared precision.

Binary fixed point numbers will be stored right justified. Widths less than 36 will occupy a single word, widths less than 72 will occupy an even-odd pair. The limiting width of fixed binary arithmetic will be 71 bits; fixed overflow will result from bits outside 71 that appear in intermediate answers in arithmetic. Default fixed point precision is 17 bit binary integers.

Decimal fixed point numbers will be kept as right-justified binary integers with implied decimal scaling. Fixed overflow will be the same as binary fixed overflow. Truncation that occurs in storing decimal data will be to

the encompassing binary size.

Label data and procedure parameter specifiers will have the form

0 (mod 2)	ITS	to program point
2	ITS	stack pointer when label was first assigned.

String specifiers will have the form

0 (mod 2)	ITS	to data addressing origin
2	LMD	length, max, and offset

The quantity known as LMD is a two-word item in the following layout. No distinction is made between bit and character strings.

0 (mod 2)	0-17	L	length in bits, less than 2**18
0	18-35	M	maximum length in bits
1	0-35	D	offset of first bit of string from addressing origin, counted in bits (mod 36*2**18)

Varying strings will be dynamically allocated and will only occupy the number of words required by their current length.

ARRAYS

Array specifiers consist of a pointer to the addressing origin of the array together with (the dope) a pointer to a dope vector, which gives the dimensioning information for the array. Arrays of strings may have further information specifying the strings in the dope. The addressing origin is the position that would be occupied by the 0,0,...,0 element of the array if it existed.

Dope vectors have the form.

0 (mod 2)	n	number of dimensions
1		unused
2	lb1	lower bound of first dim
3	hb1	upper bound of first dim

	...	
2n	lbn	lower bound of n-th dim
2n+1	hbn	upper bound of n-th dim
2n+2	m1	multiplier for first dim
	..	
3n+1	mn	multiplier for n-th dim

The multipliers are used in mapping from subscript values to addresses relative to the addressing origin. For packed arrays of nonvarying strings the multipliers read in bits, for all other arrays the multipliers read in words.

The address of an array element A(s1,s2,...,sn) is calculated by the formula

address origin + s1*m1 + ... + sn*mn (mod k)

where k=2**18 if the multipliers are in words and k=36*2**18 if the multipliers are in bits. In the basic case of arrays that are neither cross-sections nor DEFINED, mn will be the size of a data item, and the other multipliers will be given by

mj = (hbi - lbi + 1)*mi j=i-1

The specifier for an array of scalars not strings will have the form

0 (mod 2)	ITS	addressing origin
2	ITS	to dope vector

The specifier for an array of nonvarying strings will have the form

0 (mod 2)	ITS	addressing origin
2	LMD	of addressing origin
4	ITS	to dope vector

The specifier for an array of varying strings will have the form

0 (mod 2)	ITS	addressing origin for array of LMD pairs

2	ITS	to dope vector for array of LMD pairs
4	ITS	addressing origin of strings

There will probably be an aligned-vs-packed indicator in the dope vector for an array of strings. This could go well in the "unused" word of the dope vector.

STRUCTURES

Specifiers for major structures will have the form

0 (mod 2)	ITS	addressing origin
2	ITS	to structure dope vector
4	ITS	addressing origin of varying strings

A dope vector for a structure or array of structures will have the form

0 (mod 2)	p0	to array dope vector
1	p1	to first substructure
	..	
k	pk	to k-th substructure

Pointer p0 is 0 for structures without dimension. The substructure pointers pi have several interpretations depending on the substructure:

1. Elementary substructure, except non-varying string. Pointer pi points, in the data segment relative to the origin given in word 0 of the major structure specifier, to the addressing origin for this data item (to specifier if the item is a label; to LMD pair if the item is varying string).

2. Non-varying string. Pointer pi points, in the dope vector segment relative to the origin given in word 2 of the major structure specifier, to an LMD pair for the addressing origin of this data item.

3. Non-elementary substructure. Pointer pi points, in the dope vector segment relative to the origin given in word 2 of the major structure specifier, to the substructure dope vector.

If an array of structures contains one or more substructures of types 2 or 3, its gross dope vector consists of the structure dope vector, followed immediately (mod 2) by the array dope vector then the gross dope vectors or LMD pairs for these substructures taken in order.

ARGUMENTS

Argument lists, which will be pointed to by ap in the standard calling sequence, will be a vector of ITS pairs pointing to data or specifiers, whichever is appropriate.