TO: MSPM Distribution
FROM: N. I. Morris
SUBJECT: BY.14.00
DATE: 12/13/67

A new entry point, <u>ptr$addrel</u>, has been added to <u>ptr</u>. It is used to add to the offset of an existing segment pointer.

Published: 12/13/67
(Supersedes: BY.14, 01/06/67)

Identification

Relative Pointer Manipulation Procedures (PTR)
N. I. Morris

Purpose

Data bases exist which may be used by many different processes. These data bases may contain pointer information. Since the EPL pointer type variable is a standard `its' pair, pointer variables will contain a segment number. This can lead to disastrous results if different processes refer to the same data base by a different segment number. For the above reason, it is desirable to provide a pointer variable which is independent of segment number. This pointer is called a `relative pointer'; it contains no segment number, only an offset. In generating a relative pointer from an `its' pair, the segment number is discarded, leaving only the offset portion of the `its' pair. This relative pointer is contained in a bit string of length 18.

It is the purpose of the PTR procedures to convert `its' pairs to relative pointers in order to store them in common data bases, and to convert relative pointers back to `its' pairs in order to use them.

Implementation

Due to the consideration of execution time and the ease of coding, the PTR procedures have been machine-language-coded in EPLBSA. At some time in the future, they should probably be included in the EPL compiler as built-in functions.

Usage

1. ptr$rel

To generate a relative pointer from an `its' pair:

```
     relative_pointer = ptr$rel (its_pointer);
```

`its_pointer' is an EPL variable of type ptr.
`relative_pointer' is a bit string of length 18.
`relative_pointer' will contain the offset portion of the `its' pair contained in `its_pointer'.

2. ptr$ptr

To generate an `its' pair from a relative pointer:

```
its_pointer = ptr$ptr (segment_pointer,
relative_pointer);
```

`segment_pointer' is any pointer to the same segment to which `its_pointer' will point. `its_pointer' will contain the contents of `segment_pointer' with the offset part replaced by the contents of `relative_pointer'.

3. ptr$baseno

To extract the segment number from an `its' pair:

```
`segment_number' = ptr$baseno (its_pointer);
```

`segment_number' is a bit string of length 18. It will contain the segment number portion of the contents of `its_pointer'.

4. ptr$baseptr

To generate an `its' pair from a segment number:

```
its_pointer = ptr$baseptr (segment_number);
```

`its_pointer' will contain an `its' pair pointing to the segment specified by the contents of `segment_number'. The offset of the `its' pair in `its_pointer' will be zero.

5. ptr$addrel

To add to the offset of an existing pointer:

```
its_pointer = ptr$addrel (segment_pointer, relative_pointer);
```

`its_pointer' will contain on `its' pair derived from `segment_pointer' by a logical addition of the offset portion of `segment_pointer' and `relative_pointer'. The segment number of `its_pointer' is identical to the segment number in `segment_ pointer'.