TO: C. T. Clingen
F. J. Corbató
R. J. Feiertag
J. W. Gintell
N. I. Morris
M. D. Schroeder
J. H. Saltzer
S. H. Webber
B. L. Wolman
V. L. Voydock

FROM: Richard Gumpertz

DATE: June 14, 1971

SUBJECT: Use of itb pointers in argument lists

One of the inefficiencies in the current Multics PL/I Compiler is that it must build argument lists at run-time. This is, in many modular programs, an extreme drawback because it makes the "call" statement an expensive statement to use.

It would be nice if the argument list could be created at compile time rather than at run-time. Unfortunately, the location of most arguments is not known of compile time for two reasons. The first, which is the most obvious, is that the segment number of the segment containing the argument is not known at compile time. Another problem, however, is that the offset within the segment is not always known, as in the case of items in a program's stack frame.

Arguments to procedures may be classified into 6 general classes:

1) automatic — in the caller's stack frame at a known offset from the beginning of the frame.

2) internal static — in the caller's linkage section at a known offset from the beginning of the linkage section.

How can we translate this information into argument lists of pointers to be set up in an argument list? The current compiler does an "eapbp" for each of its arguments and then an "stpbp" into the stack to set up the argument list. I would like to propose that this method be used only if many arguments of type 6 are used in a calling sequence. In all other cases an argument list may be constructed in the text section of the procedure at compile time. (Actually it might have to be moved for execute only procedures.) Arguments of type 6 could be converted to type 5 at run-time by an "eapbp; stpbp sp|temp_ptr" sequence. The argument addressing by the callee would be very similar to that used above, with one exception; the "sp" and "lp" are changed while running the callee. Therefore, two new base registers must be reserved, called "old_sp" and "old_lp" respectively. These would be reserved to point **wherever "sp" and** "lp", respectively, pointed to just before the call statement. That is, "old_sp" points to the previous stack frame and "old_lp" points to the previous linkage frame. The argument list double word pointers used in each of the 5 classes would then be:

1)	automatic	itb old_sp, offset
2)	internal_static	itb old_lp, offset
3)	constants	arg offset; arg 0
4)	external static	itb old_lp, offset, *
5)	based (internal pointer) (with 0 offset)	itb old_sp, offset, * or itb old_lp, offset, *

```
          eapbp                    descr_for_d-*, ic
          stpbp                    sp|arglist+18
          eapap                    sp|arglist
          "rest of call operator
```

Under the proposed system the code would be:

```
          eapap                    arglist_*, ic
          "rest of call operator
            .
            .
            .
arglist:  zero                     2*4, 4
          zero                     2*4, 0
          itb                      old_sp, a
          itb                      old_lp, b
          itb                      old_lp, c_ptr, *
          itb                      old_sp, p, *
          arg                      descr_for_a; zero
          arg                      descr_for_b; zero
          arg                      descr_for_c; zero
          arg                      descr_for_d; zero
```

The PL/I call (or entry) operator would have to be expended by two instructions:

```
          eapold_sp                sp|0
          eapold_lp                lp|0
```

The extra indirection is clearly cheaper than the two instruction sequence "eapbp; stpbp" now performed by the caller. If the arguments are referenced frequently the indirection can be saved by copying this argument pointer once at entry to the callee.

The location of a procedure's linkage and stack frames are well-defined and can be found in the dump. These pointers can be added easily to the offset specified in the "itb" pair.

5) This is clearly just a matter of changing the specifications. It must be done soon, however.

I would appreciate any comments you have on the above proposal, and any suggested improvements will be welcomed.