From Barry Wolman /

DRAFT

3/4/69

PL/I Macro Expander

This note describes the PL/I macro expander as currently implemented. Several proposed changes will be discussed later.

The macro expander EXPMAC is called by various modules of the code generator to expand a macro using 0 to 7 substitutable arguments supplied by the user. The result of a call to EXPMAC is one or more binary words appended to the current output text and relocation buffer segments. EXPMAC maintains a parallel relocation segment; the relocation code in the segment are packed together when the symbol segment is created.

The external static points OUTPUT_PT and RELOC_PT point to the text and relocation buffers. They are updated by EXPMAC to point to the next available locater.

Macro arguments are passed to EXPMAC via points to RAND and ADDRESS nodes. A RAND node is used to record various attributes of an operand, one is filled in for each operand of the current PL/I operator being compiled. When the ADDRESS field of a RAND node is NULL, the operand is not <u>addressable</u>. When the addressability module is called for the operand, and ADDRESS node is filled in and the ADDRESS field of the RAND node is set to point at it. (some entries to EXPMAC will do this).

The macro argument is contained in the ADDRESS node as a 36 bit word having the same structure as a 645 machine word. The ADDRESS node also records relocation bits for the left and right halves of the address word and whether or not the address is guaranteed to be on an even boundary. The KIND field of the address node is encoded as

0	no special action required
1	reference to an as yet undefined location in the text segment
2	reference to a previously defined location in the text segment

A macro is an integer value which is used to index a table of macro bodies provided by the user. The definition corresponding to a macro is found in the data segment <MACRO_TABLE> via the structure.

```
dcl  1  macro_table$macro_table(1000)ext static,
        2 rel_ptr        bit(18),
        2 op_code        bit(9),
        2 size           bit(9);
```

If the SIZE field is zero, the macro body consists of a single 645 instruction whose operation code is given by the field OP_CODE. In this case, the macro may be used with either 0 or 1 arguments, depending on which entry of EXPMAC is called.

If the SIZE field is non-zero, REL_PTR is the offset in <MACRO_TABLE> of a macro body consisting of SIZE words. Each word of the macro has the structure

```
dcl   1    macro_word   based(p),
           2 dummy      bit(3),
           2 number     bit(3),
           2 increment  bit(12),
           2 rest       bit(18);
```

If the NUMBER field is zero, no argument is used for this word of the macro. EXPMAC copies the word into the output segment with absolute relocation used for both halves of the word.

If the NUMBER field is non-zero, argument NUMBER is combined with the 36 bit address corresponding to macro NUMBER as follows:

1) The macro word is copied into the output segment
2) Bits 0-17 of the word in the output segment are cleared
3) the 36 bit address is OR'ed into the output segment
4) the INCREMENT field is added to the offset field of the word resulting from step 3.

If the macro argument represents a forward reference to an undefined location in the text segment (KIND=1), EXPMAC will maintain a usage chain which can be filled in later. In this case the SYMBOL field of the ADDRESS node points at a LABEL_ATTRIBUTE block whose LAST_USAGE field bolds the offset, in the text segment, of the last use of this label. EXPMAC sets the offset part of the current output word to LAST_USAGE and LAST_USAGE to the offset of the current word.

If the macro argument represents a reference to an already determined location in the text segment (KIND=2), EXPMAC converts the absolute text offset contained in the macro argument into a self-relative offset so that IC modification may be used.

An error message is printed if too few arguments are supplied. Extra arguments are ignored.

The following entries to EXOMAC are currently implemented

1)
```
call  expmac (macro,arg_pt);
dcl   macro   fixed,
      arg_pt  ptr;
```

This expands a macro with a single argument. ARG_PT may point at either a RAND or ADDRESS node. If the ADDRESS field of a RAND node is null, M-A will be called.

2)
```
call  expmac$one(macro,arg_pt,double);
dcl   double  fixed;
```

Same as EXPMAC except DOUBLE is non-zero if the macro argument specifies a double length datum; in this case MACRO+2 is expanded if the address is guaranteed to be even and MACRO+1 is expanded otherwise.

3)
```
call  expmac$zero(macro);
```
No arguments are expected

4)
```
call  expmac$many(macro,arg_pt, arg_cut);
dcl   arg_cut   fixed,
   1 arg(arg_cut) ptr band(p);
```

ARG_PT points at an array of points to RAND or ADDRESS nodes. If the ADDRESS field of a RAND node is null, an error will occur.

5) call expmac$abs(blk_pt, blk_cut);

```
dcl  blk_pt  ptr,
     blk_cut  fixed;
```

The block of words pointed at by BLK_PT is copied into the text segment. Absolute relocation is used.

6) call expmac/cur_loc(org_val),

```
dcl  org_val  fixed;
```

ORG_VAL is set to the current position in the text segment

7) call expmac$origin(org_val)

Sets the current position in the text segment to ORG_VAL.

8) call expmac$fill_usage(val,last_use);

```
dcl  (val,last_use) fixed bin(15);
```

The usage string starting at location LAST_USE in the text segment is filled in with VAL.

The macro expander has the faults to produce an EPLBSA-like listing by disassembling the binary text it generates. Because of forward references to symbols not having a value assigned, it is not to generate the PL/I assembly listing as the macros are expanded. The listing is actually done after all macros have been generated and all assigned.

In the near future EXPMAC will be changed to use an external static pointer and a structure to access the macro definition table. This will facilitate the use of different tables.

It would be nice if the macro definition could indicate how the macro affected the execution environment, i.e., that the 2nd argument should be loaded (if necessary) into XR7 or that the EAQ, XR2 and XR3 are des destroyed by the code resulting from the macro. These faults will not be available for quite some time.