DATE: June 16, 1978 PE-1-364, Rev. 1

FROM: Bradford E. Hampson

TO: All R & D Personnel

SUBJECT: The Formatted Output Package "ioa$".

The entry points ioa$ and ioa$rs perform output conversion and formatting services, and are callable from any user program. Programs running in 64V mode must use the V-mode library version VIOALB; programs running in any other mode use the R-mode version IOALIB (temporarily, X.IOALIB). The calling sequence for ioa$ and ioa$rs seen at the user level is the same for either version.

The formal documentation for ioa$ and ioa$rs is presented below. By way of introduction, however, the following comments are in order. Basically, ioa$ allows the caller to specify a character string called a control_string, as well as up to 99 optional arguments. The control string specifies literal text to be output, but intermixed within the literal text can be escape sequences called conversion requests. These control the conversion of the data represented by the optional arguments to character-string form, and specify how the results of the conversions are to be edited into the final output string. Thus, conversion requests specify both the data type of an optional argument and the format of the resulting conversion.

The principal reasons for using ioa$ instead of FORTRAN I/O (with FORMAT statements) are the following. First, ioa$ is a smaller total package than FORTRAN I/O by a factor of about 2.5. Second, the conversion specifiers available in ioa$ are more flexible than those provided by FORTRAN (for example, ioa$ permits control over justification, zerofilling, and so forth). And finally, the format of an ioa$ control string is particularly convenient in terms of ease of concatenation of conversions with literal text, and of a possible need to specify a conversion format at runtime.

Loading the ioa$ Package

To load the V-mode version, use the loader command "library vioalb". To load the R-mode version, use "library (x.)ioalib".

OR PMA routine: SEG
DYNT IOA$
END

". for V-mode.

------ Subroutine Call 06-14-78 --------
ioa$ | | ioa$ |
------ Library VICALB and IOALIB --------

try: ioa$

rpose:

subroutine ioa$ is called to generate formatted output text and put it to the user's terminal. Conversions of several data types, o various formats, can be easily interspersed with literal text in a tural and flexible way.

present version of ioa$ does not support floating-point corversions any type, nor does it support automatic conversion of entire arrays.

ge:

```
declare ioa$ entry;

call ioa$ (control, conlen [, arg1, ..., arg99]);
```

trol is a nonvarying character string (Fortran: array containing packed characters) whose length is given by conlen. Conlen may be larger than the actual length of the string provided that the string ends with either "x$" or "x." as described below. This string specifies both literal output text and the conversions to be performed on the arg1, if ary. (Input)

len is the length of control, in characters. Conlen may be larger than the actual length of control, as explained above. (Note: a previous version of ioa$ allowed negative values of conlen to specify "indefinite length". Use of this feature is not recommended, as support for it will be withdrawn at some point.) (Input)

1, ..., arg99
are optional arguments of any datatype representing values required by conversion specifiers in the string <control>. The datatype ascribed by ioa$ to each arg1 is determined by the particular specifier in the control string causing the argument to be referenced. Up to 59 optional arguments can be specified. (Input, optional)

Control String

only special character in a control string is the character "x". other text is output literally. The control string ends either (1) er conlen characters have been read; (2) when "x$" or "x." is ountered; or (3) when an optional argument is needed but none ains. The "x" character begins a conversion specifier, the general mat of which is:

x[<fw>] [:<prec>] [z] [r] <type>

Note that any alphabetic character in a conversion specifier may be upper- or lower-case, equivalently. In the following discussions, the phrase "next argument" refers to the next optional argument in sequence from the list arg1, ..., arg99. If no more arguments exist when one is needed, formatting is terminated. A given argument is used only once, unless the reposition request ("y" conversion) is used. Example: in the specifier 'x27:1zd', <fw> is 27, <prec> is 1, "z" is present, "r" is omitted, and <type> is "d".

<fw> specifies the field width, or in some cases a repeat count. If omitted or zero, fields will default to the exact number of positions required to represent the value. An omitted repeat count defaults to 1, while a zero repeat count is honored as such (i.e. as a no-operation). A nonzero <fw> specifies the number of character positions to be occupied by the converted datum, filled with blanks or zeroes as needed. A datum can be left- or right-justified in its field. A constant field width is specified by providing a decimal integer for <fw>. If the field size is to be computed at execution time, a variable field width is specified by providing the character "#" for <fw>. The next argument will then be used as a single precision integer specifying the field width. A negative variable <fw> is processed as if <fw> were omitted.

<prec> is the argument precision. Three different precisions are allowed at present: 0 for fixed binary(16,0) or "single precision unsigned"; 1 for fixed binary(15,0) or "single precision signed"; and 2 for fixed binary(31,0) or "double precision signed". These correspond to the FORTRAN datatypes INTEGER*2 (taken as unsigned), INTEGER*2 and INTEGER*4, respectively. If <prec> is numeric but greater than 2, 2 is assumed; if it is non-numeric, the precision specifier is ignored. Thus, 'x:2:A' will result in a precision of 2. If <prec> is omitted, 1 is assumed.

z if specified, means that a numeric field is to be zero-filled instead of blank-filled. Note that "z" will have no visible effect if padding is not actually needed (<fw> omitted or too small to contain the value). "z" has no effect on ascii or pointer conversions. The "x" and "l" conversions use "z" in a special way; see below.

r if specified, means that the default justification sense for the conversion will be reversed. See the individual conversion request descriptions for specifics.

<type> specifies the control operation, or the conversion to be performed (and hence the datatype of the input argument, if one is used). Implemented <type>s are:

x literal 'x'. "z", "r", <prec> and <fw> are ignored.

decimal integer. The next argument is the integer, of precision specified by <prec>. "z" and <fw> are honored, except that <fw> is ignored if it is too small to contain the value. "r" is honored; the default justification sense is right.

is like "q", except conversion is done in octal radix.

is like "q", except conversion is done in hexadecimal radix.

generate newline characters. <Fw> is a repeat count; <fw> newlines will be generated. "z" and "r" are ignored, as is <prec>. Due to a peculiarity of the operating system routine that performs terminal I/O, the character pair CR followed by LF is currently used to represent 'newline'.

generate formfeed characters. <Fw> is a repeat count; <fw> formfeeds will be generated. "z", "r" and <prec> are ignored. The ASCII formfeed character (octal 214) is presently used to represent formfeeds.

generate filler. <Fw> is a repeat count; <fw> filler characters are gererated. If the "z" flag is present, the filler character used will be '0'; else blanks are used. "r" and <prec> are ignored.

reposition in argument list. <Fw> specifies the argument number; <fw>=1 will position to arg1 (the first optional argument). If <fw> is less than 1, 1 is assumed; if greater than 99, 99 is assumed. Hence, just '%y' repositions to the first argument. "z", "r" and <prec> are ignored.

terminate control string. "z", "r", <prec> and <fw> are ignored.

terminate control string and append single newline to output. "z", "r", <prec> and <fw> are ignored.

logical. <Prec> is ignored; the next argument is a single 16-bit word representing PL/1 datatype "bit(16) aligned" (Fortran: LOGICAL). The datum is considered "true" if any bit is 1; else "false". The result of the conversion is one of the letters "T" or "F", unless the "z" flag is present, in which case "TRUE" and "FALSE" are used instead. The default justification sense is right.

word. Equivalent to :02o (unsigned, single precision zerofilled octal). "z" and <prec> are ignored. Justification is handled as for the "o" conversion.

ASCII. The next argument is a nonvarying character string (Fortran: array containing packed characters); the following argument is a precision 1 integer containing the

length of the string in characters. A negative or zero string length is interpreted as representing the null string. The string length is adjusted downward by stripping off all trailing blanks (i.e. all blanks to the right of the rightmost nonblank character). The default justification sense is left. "z" and <prec> are ignored.

c ASCII. Like "a", except that trailing blanks are not stripped; that is, if the string length is N, exactly N characters will be output.

v Varying character string. The next argument is a varying character string (Fortran: array whose first element contains the string length, and remaining elements contain packed characters). No length argument is needed, as the string is self-describing. The string is converted as for the "c" conversion above.

p pointer. In R-mode, the next argument is a a single-word address value. The address value is converted to a non-zerofilled unsigned octal integer. In V-mode, the next argument is a pointer and may be either 2 or 3 words long, depending on the bit extension flag of the pointer. If the pointer has its fault bit set, the result of the conversion has the format '*FAULT/WWWWWW', where W..W is the first word of the pointer in octal. Otherwise, the format used is 'SSSS(R)/WWWWWW(BB)', where R is the ring number, S..S the segment number, W..W the word number and (BB) the bit offset (if any). All values octal except BB, which is in decimal if present. The default justification sense is left.

(start repeat group. <Fw> is the repeat count, and must be nonzero. If <fw> is omitted, 1 is assumed. All text and conversions between the opening %(and the closing %) (see below) are repeated <fw> times, on succeeding optional arguments if any is used. If no closing %) is encountered, then the repeats will not be performed. Nested repeat groups are not presently allowed; if a %(is encountered inside a repeat group, the result of the inner %(conversion will be two question marks. "z", "r" and <prec> are ignored.

) end repeat group. The group is repeated until the repeat count specified in the opening %(is exhausted. <Fw>, "z", "r" and <prec> are ignored.

Notes:

If an illegal <type> is found, the result of the conversion is the two characters '??'. If the output buffer overflows, or if the input argument list is exhausted when a new argument is needed, conversion is terminated (without a newline appended).

... has an internal working buffer of capacity 400 characters. Thus,

400 characters may be gererated by a single call to ioa$.

[ex: (Fortran)

```
CALL IOA$('VALUE= %:2D.%.',100,4556809)
ces
VALUE= 4556809.

CALL IOA$('LOCATION XP CONTAINS %6W.%.',100,LCC(I),I)
 produce
LOCATION 4002(3)/2707 CONTAINS 103022.

CALL IOA$('X12A IS %5"z"L.%.',15,'VERITAS',7,
     .TRUE.)
ces
VERITAS      IS  TRUE.
```

[ex: [PL/I]

```
declare ioa$ entry,
        a char(14) var, b char(32) aligned,
        c ptr, d fixed bin, e fixed bin(31),
        f bit(16) aligned;

a = 'hello';
b = 'x';
c = addr (a);
d = 16;  e = -d;
f = (d = 15);
call ioa$ ('%6ra, %5zc,%vi%p,%2d. %5zl%.',
    100, b, 32, d, a, c, e, f);

 produce:

|     x, 00016,hello|6002(3)/2401,-16. FALSE<CR><LF>|
```

[ex: ioa$rs

[ex:

The entry point ioa$rs provides the same kind of output conversion ces as ioa$, except that the formatted text is returned to the r in a memory buffer, instead of being output to the user's nal. The buffer is proviced by the caller, and hence the capacity e buffer is determined by the caller. Note that ioa$ has an mentation restriction on output volume, because its internal r is only 400 characters large.

[i

```
declare ioa$rs entry;

call ioa$rs (buffer, bufsize, buflen, control,
```



```
     conlen [, arg1, ..., arg99])];
```

buffer is the nonvarying character string (Fortran: array of packed characters) into which ioa$rs will write the formatted text. (Output)

bufsize is the capacity of buffer, in characters; that is, buffer must be able to hold <bufsize> characters or more. Note that buffer is blank-padded to its stated capacity if the length of the generated text is less than bufsize. (Input)

buflen is the number of characters of text generated by the conversion operations. That is, the PL/I expression "substr (buffer, 1, buflen)" represents the complete returned string. (Output)

control is a control string, with exactly the same format and meaning as for ioa$ above. (Input)

conlen is the length in characters of control, interpreted in the same way as for ioa$, above. (Input)

arg1, ..., arg99
are the optional arguments, as for ioa$. Up to 99 optional arguments can be specified. (Input, optional)

Example:

```
     declare buf char(80) aligned,
             len fixed bin,
             ioa$rs entry;

     call ioa$rs (buf, 80, len, '%6zd%$', 6, -123);
```

would set <len> to 6 and <buf> to '-00123' (blank padded).

(End)