WULF71
Wulf,W.A.;
BLISS: A Language for Systems Programming;
Communications of the ACM (December 1971) pp. 780-791.
BLISS is one of a small number of systems languages designed to achieve the space/time economy of assemble while retaining the programming/ maintenance ease of a structure high level language. To achieve this blend, BLISS reflects the underlying hardware design of the host computer in both overt and subtle ways.

The criteria used in designing BLISS included (in descending order of importance):

- space/time economy
- access to all relevant hardware features
- control over representation of data structures
- a flexible range of controls (including co-routines)
- modularization
- overall good language design
- machine independence

BLISS is an "ALGOL" like language with block structure, block variables, and similar conditional and looping constructs. This design decision was supported by the following two arguements. First, that a higher level language's redability and structure facilitates maintenance, redesign and recoding, which is of paramount importance. Second, that program efficency is "determined by overall program design and not locally tricky, 'bit-picking' coding practices". Further efficiency was gained by specifically targeting the language to the host machine, a PDP-10. Proposed language constructs were evaluated in terms of the machine code that could be generated. While some constructs where introduced to improve readability and ease programming, it was also necessary that the addition information allow better code generation.

The article describes in some detail the language, its underlying structure and implementation. Presented here is an overview of some of the more interesting language features.

In BLISS programs all data are fields of bits. The interpretation of these bits (as integer, real, ...), and its consequent transformation, is an intrinsic prperty of the operator and not of the operands. This "typless" manipulation allows the flexible data handling required by systems programmers.

The value of an identifier is a pointer to a value, and not the actual data value itself as is common in most languages. The dot "." notation is used to address the actual value (eg. ".x"). Pointers are a fundamental concept in BLISS and are useful in record structure processing and dynamic programming.

BLISS contains the standard if..then..else, looping, case-select, and function calls. Parameters are passed as "call-by-value" but call by reference is implicit in the fact that pointer values can be passed. The "goto" construct was replaced by structured programming and "escape" expressions. These escapes (eg. "exit-block", "exitloop",...) are a type of highly structured forward branch which are awkward to implement using control expressions only. Co-routines (or asynchronous processes) were implemented using the "cre-ate" expressions which sets up run-time data constructs. Execution control is sequentiallly exchanged between processes using the "exchj" expression.

One of the major features of BLISS is the mechanism which permits the definition of data structures in terms of an access algorithm for each element independent of the data manipulation algorithm. The access is defined via the "structure" expression and mapped onto the actual data storage area via the "map" expres-sion. For example:

```
structure arry[i,j]=(.arry + .i*10 + .j)
own       x[100],y[100]
```

```
map     arry x;y
```

maps a 10x10 array structure onto data areas x and y. This allows reference of the for x[.i,.j]. This powerful feature allows either the accessing algorithm or manipulation code to be modified without affecting the other.

While BLISS was shown to be a viable language, and is still used to write drivers and other software on PDP11 and VAX11 computers, it has not found wide acceptance. This may be due to the popularity of the "C" language. However the design guidelines and many of the language constructs are well worth consideration when designing system language alternatives to assembler or as underlying concepts to be embodied in general purpose languages.