To: Distribution

From: Richard Bratt

Subject: MSPL PL/I Extension Report

Date: December 7, 1976

This document reports the recommendations of the Multics System Programming Language (MSPL) committee for extensions to Multics PL/I. The committee, which was quite conservative, observed the following guidelines in making its recommendations.

o no recommendation may conflict with ANSI PL/I; only features which superset ANSI PL/I may be considered.

o no recommendation may be made which has a reasonable probability, in the committee's opinion, of conflicting with foreseeable alterations in ANSI PL/I.

o no recommendation may seriously violate the spirit of ANSI PL/I.

o no recommendation may be made which does not answer an identifiable need of the Multics community in general and the maintainers of the Multics system in particular.

The committee arrived at its recommendations by considering many individual proposals to extend Multics PL/I. The text of this report is taken, for the most part, directly from those proposals which were accepted by the committee. Although an attempt has been made to avoid major stylistic differences between the various individual recommendations which comprise this report, no attempt has been made to rewrite each recommendation in a single canonical form. As a result, the reader may notice inevitable variances in the style, language, and level of detail among the recommendations which follow.

Most recommendations are presented in the form of a prototype description suitable for inclusion in AG94 with only minor editing. In some cases, where a recommendation requires more extensive integration with AG94, the committee has chosen to provide a description of the desired feature and to leave the appropriate manual changes to the implementors of these recommendations.

The remainder of this report is divided into three sections. The first section is composed of recommended data attributes. The second section is composed of recommended builtin functions. The final section is composed of recommendations which fall into neither of the previous classes. The section describing recommended builtin functions is further subdivided into three groups. Group one is composed of descriptions of builtin functions which deal with strings. Group two is composed of descriptions of builtin functions which deal with the environment external to PL/I. Finally, group three is composed of the descriptions of those recommended builtin functions which deal with internal PL/I data representations.

One appendix has been included in this report. This appendix documents the feelings of the committee toward several proposed changes in the format of the listing produced by the Multics PL/I compiler. This section was not included in the main body of the report since the committee felt that these enhancements, which do not represent language changes, can be adequately addressed through the normal Multics Change Review (MCR) mechanism.

ATTRIBUTE SECTION

abnormal Attribute

Syntax: <abnormal attribute> ::= abnormal

A name declared with the <abnormal attribute> is a variable that may be accessed asychronously by more than one Multics <program>. To obtain the value of an abnormal variable, the <program> always accesses storage; expressions depending on such variables are never commoned.

The <abnormal attribute> should be used with care as its use may degrade the performance of the compiled code. The <abnormal attribute> is a nonstandard <attribute>.

unsigned Attribute

Syntax: <unsigned attribute> ::= unsigned | uns

An item declared with the <unsigned attribute> represents a nonnegative value. The use of unsigned is nonstandard and only compatible with the real mode attribute, the fixed type attribute, the binary base attribute, and a scale factor of zero.

The <unsigned attribute> only affects the stored representation of unaligned items. Whereas unaligned items of precision n are normally packed in n+1 bits, unaligned items declared with the <unsigned attribute> are packed in n bits.

The size condition, if enabled, occurs when a negative value or a value not representable within the declared precision is assigned to an item declared with the <unsigned attribute>.

BUILTIN SECTION

String Builtins

bitrel Builtin

Example: bitrel (P, C)

bitrel is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

P must be a scalar pointer value. C is converted to a fixed-point, binary, real value C' of precision (24, 0). The program is in error if C' is negative.

The result is a pointer to the C'+1st element of an unaligned array of single bits located by P.

charrel Builtin

Example: charrel (P, C)

charrel is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

P must be a scalar pointer value. C is converted to a fixed-point, binary, real value C' of precision (21, 0). The program is in error if C' is negative.

The result is a pointer to the C'+1st element of an unaligned array of single characters located by P.

ltrim Builtin

Example: ltrim (S, C) or ltrim (S)

ltrim is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

S and C are converted to the character-strings S' and C'. If C is omitted, the value of C' is a single blank character. The result R is a character-string.

If n is zero then R is the null character-string. Otherwise, for k=1,2,...,n the kth character of S', S'k, is tested to see if it occurs in C'. Let m be the first value of k for which the test fails; or if the test succeeds for all value of k, m=n+1.

The length of the result R is l=n-m+1. For k=1,2,...l Rk=S'k+m-1.

rtrim Builtin

Example: rtrim (S, C) or rtrim (S)

rtrim is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

S and C are converted to the character-strings S' and C'. If C is omitted, the value of C' is a single blank character. The result R is a character-string.

To determine the value of R, let n be the length of S'. For k=n,n-1,...,1 the kth character of S', S'k, is tested to see if it occurs in C'. Let m be the first value of k for which the test fails: or if the test succeeds for all values of k, m=0.

The length of the result R is l=m. For k=1,2,...m Rk = S'k.

← .ne

9-bit String Builtins

It is proposed that a new character set be defined which includes all possible 9-bit bytes. This new character set, the Multics Extended Character Set, contains the standard Multics ASCII

Character set as a proper subset in the natural way. To facilitate use of the Multics Extended Character Set, the Multics PL/I builtins search, translate, and verify should be extended to operate compatably on the extended character set. In addition, the builtins collate9 and high9 should be added to define the Multics Extended Character Set.

The use of high9 or collate9 is nonstandard and makes programs dependent on Multics PL/I.

Environment Builtins

clock Builtin

Example: clock () or clock

clock is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

The result R is a fixed-point, unscaled, binary, real value of precision 52. The value of R is the number of microseconds since 0000 hours Greenwich mean time January 1, 1901.

oncode Builtin

Example: oncode() or oncode

The value returned by this function is a fixed point, binary, real number of precision (35, 0). This value indicates the reason why the condition was signalled. This value is a standard Multics status code (see "Status Codes" in the Multics Programmer's Manual). Because the run-time routines that support the execution of PL/I programs are subject to modification and improvement, the list of error codes which can be returned is subject to change, and is not published in this document. If a program is expected to run on other implementations of PL/I, the program logic must not depend upon the value of this builtin function.

Note: An efficient method must be provided for translating the new values returned by oncode into the equivalent old values as some parts of the system (incorrectly) depend upon accidental properties of these old values.

stack_frame_ptr <u>Builtin</u>

Example: stack_frame_ptr () or stack_frame_ptr

stack_frame_ptr is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

stack_frame_ptr returns a pointer to the stack frame of the current block.

stack_base_ptr <u>Builtin</u>

Example: stack_base_ptr () or stack_base_ptr

stack_base_ptr is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

stack_base_ptr () returns a pointer to the base of the current block's stack segment.

stacq <u>Builtin</u>

Example: stacq (X, Y, Z)

stacq is a nonstandard built-in function and its use makes programs depend on the Multics hardware.

X must be a scalar pointer value. Y and Z are converted to bit-strings Y' and Z' of length 36.

If the 36 bit word addressed by the pointer X contains the bit-string value Y', the value of Z' is assigned to that word; otherwise, no assignment is made.

The result R is a bit-string of length 1. If the assignment of Z' to the location denoted by X was made, the value of R is "1"b; otherwise, it is "0"b.

The testing of the word addressed by X and the assignment of Z' to the word addressed by X is an indivisible operation of the

Multics hardware.

vclock Builtin

Example: vclock () or vclock

vclock is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

The result R is a fixed-point, unscaled, binary, real value of precision 52. The value of R is the total number of microseconds of virtual CPU time used by the calling process.

Note: It is recommended that this builtin not be implemented until such time as it can be implemented efficiently.

Internal Representation Builtins

allocated_size Builtin

Example: allocated_size (X)

allocated_size is a nonstandard built-in function and its use makes programs depend on the internal representation of data in Multics PL/I.

X must be an unsubscripted <reference> to a level-one variable.

The result is a fixed-point, binary, real number of precision (19, 0) whose value is the number of 36-bit words occupied by the generation of storage obtained by evaluating the reference X. Note that when X is a reference to a based variable with <refer options>s, this function returns a value that depends on the <reference> contaired in the <refer option>, not on the <expression> in the <extent expression>.

code_ptr Builtin

Example: code_ptr (V)

code_ptr is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

V must be an entry, label or format value. The result R is a pointer value. If V is an entry value then the entry pointer of the entry value is the result. If V is a label value then the code pointer of the label value is the result. If V is a format value then the format pointer of the format value is the result.

environment_ptr Builtin

Example: environment_ptr (V)

environment_ptr is a nonstandard built-in function and its use makes programs dependent on Multics PL/I.

V is an entry, label or format value. The result of this builtin is a pointer value. The result is the environment pointer of V.

CONSTANT SECTION

Octal, hex constants

It is proposed that Multics PL/I be extended to include the ANSI standard format for bit strings. In particular, the "...."bn, syntax should be supported where n is 1, 2, 3, or 4. This allows octal numbers to be input as "377777000140"b3, etc.

LISTING SECTION

Comment close checking

The PL/I lexer will check for the string "/*" within a comment string and issue a warning if found.

APPENDIX

The following list of proposed changes to the listing file produced by the Multics PL/I compiler was generated from many informal discussions. The committee considered these proposals and voted on each separately. The numbers in parenthesis following each proposal indicate the number of committee members voting for, voting against, and abstaining from voting respectively. The results of this poll do not represent an official recommendation by the committee.

1. Change the section of the listing for "NEVER REFERENCED" variables to include a list of useless (unused by the compiler) declarations. The list will include only level 1 names of structures and unstructured variables. The list will also include unreferenced labels, builtins and parameters. Note that it appears difficult to have the compiler indicate which include files are not needed because this is a "lex time" feature that does not permit easy transformation of the necessary information. (4, 0, 1)

2. Add an optional section entitled "OFFSETS OF AUTOMATIC VARIABLES" consisting of a list of all (referenced) automatic variables sorted by offset in the stack frame. Structured variables will have items within the structure listed separately and indented. There might be three or four columns of such information. (5, 0, 0)

3. Add an optional section entitled "OFFSETS OF INTERNAL STATIC VARIABLES" analogous to the section described in 2 above. This section would not include static "variables" allocated in the text. (5, 0, 0)

4. Change the source listing to include a "*" in column 10 (currently left blank) for all comment lines continued from the previous line. This is to catch missing comment close sequences. (4, 0, 1)

5. Change the source listing so that the colon following a label is immediately followed by a curly-bracketted list of line numbers on which the label was referenced. (4, 2, 1)

6. Change the numbers in the variable field of the assembly listing statements to be octal to conform with debug. (2, 1, 2)

7. Arrange the list of variables referenced (at the end of the source) by internal procedure, i.e.,

   a.) Have a separate cross-reference for each internal procedure. (0, 7, 0) b.) Record with each variable the name of the internal procedure in which it was declared. (6, 0, 1)

8. Optionally record in the left margin the block indentation level. The level is incremented for PROC, BEGIN, DO, etc. (4, 3, 0)

9. Reformat the list of internal procedures to facilitate the determination of those that are quick and those that are not quick. For those that are not, indicate why not. (7, 0, 0)

10. Include the pathname of the source segment, the installation ID, and the date time modified of the source and include files in the listing file. (7, 0, 0)