To: Distribution

From: Paul Green

Date: 11/28/77

Subject: Plans for MR7.0 PL/I Compiler

This MTB describes the enhancements that will be available in the MR7.0 PL/I compiler. Some desirable features that may or may not be available, depending on time constraints, are also listed. Comments on this MTB should be sent to Green.Multics on the Phoenix or MIT Multics systems.

New Features

The following features are listed in order of descending priority.

1. packed decimal data (unsigned or leading sign, 4 bits per digit, digit aligned)

2. unsigned fixed binary

3. various enhancements to the listing segment: automatic storage map, static storage map, name of block a variable was declared in

4. line-oriented input in get edit (ability to read an entire input line, similar to read-to-stream-file change made in MR6.0)

5. radix-factor bit strings in get list

6. a subroutine interface to the compiler (for create_data_segment and display_pl1_structure and the private tools create_include_file and structure_xref)

7. eliminate the EXL long_profile command by combining it with the profile command

8. stop statement for use in run units

9. the substraddr, stackframeptr, stackbaseptr, codeptr, and environmentptr builtins

Internal Changes

The following changes are completely internal to the compiler, and will be done for MR7.0 if time permits. These changes will provide a cleaner base for implementation of new features.

1. reformat opcode_info database to get an extra 4 bits per opcode so that the optimizer can use it to determine the jump, signal, irreducible, and set_operand1 flags, instead of computing these flags every time.

2. reformat macro_table database to hold odd pointer registers. Eliminate direct calls to state_man$erase_regs for odd bases.

Suggested Changes

The following suggestions have been made for PL/I, and are likely to be implemented if time permits. They are listed in order of descending priority.

1. put size of structures (and variables?) in listing

2. add compiler option to control listing of unreferenced variables and structure members

3. put fully qualified names (or some variation) in symbol map

4. optimize setting several bits in same word to the same state

5. better versions of search and verify builtins that return number of characters scanned, instead of index of stopping character.

6. warning for multiple closure of end statements

7. new source map to allow source segments to be greater than 64K words

8. new source_id field to allow source segments to contain more than 16K lines

Completed Changes

The following changes have already been made to the compiler, and are available in >exl>o.

1. error if number of initial elements for a static array is not the same as the number of elements in the array

The remainder of this MTB describes the packed decimal/unsigned project in more detail.

Purpose

The purpose of this project is to support unsigned fixed binary values, and 4-bit decimal values in Multics PL/I. These new decimal values can be real or complex, fixed or float, signed or unsigned (the sign, if present, will be a leading sign). The PL/I implementation of packed decimal will be compatible with the existing COBOL implementation of COMP-8 packed decimal values. COBOL COMP-8 packed decimal values are digit-aligned, with an optional leading sign. COBOL COMP-5 packed decimal values are byte-aligned, with an optional trailing sign. There will be no way to specify a byte-aligned, trailing sign value in PL/I, so COMP-5 will not be compatible with PL/I. Even though PL/I will not support COMP-5, other system routines, such as any_to_any_, will support it.

Scope

The PL/I compiler, runtime I/O, runtime conversion, and debugging tools will be changed to support packed decimal. The standard system debuggers (debug and probe) will be changed. Other standard subroutines (formline_, decode_descriptor_) will also be changed. No unbundled software will be changed directly--it is up to the maintainers of each unbundled product to change their own programs.

Timetable

The cutoff date for installing the compiler at MIT and Phoenix is July 31, 1978. We plan to install this compiler a lot sooner than that, but it is too early to tell.

Declaring packed decimal values

In the present PL/I compiler (release 23 and earlier), decimal use a signed, 9-bits-per-digit format. Decimal values declared with the aligned attribute begin on a word boundary. Decimal values declared with the unaligned attribute begin on a byte boundary. We want to have packed decimal values begin on a digit boundary. To add packed decimal values to our PL/I language, we either have to add a new arithmetic base (binary, decimal, packed_decimal), or a new alignment attribute (aligned, unaligned, digit_aligned), or redefine the meaning of the current alignment attributes for decimal values. There are no other ways to add packed decimal; these are the alternatives.

The first alternative (a new base) is out of the question. It would require extensive changes to the compiler; every place we now test symbol.decimal we would have to test both symbol.decimal and symbol.packed_decimal. This would require hundreds of changes to the compiler (and manual). The second alternative (a new alignement attribute) has a similar problem. Every place the compiler uses the aligned or unaligned attribute would have to be changed to understand the digit_aligned attribute. The manual would have to be changed to restrict defining and overlaying for digit_aligned, just as it now restricts defining and overlaying for unaligned.

The fact is that the PL/I language has only one way of specifying different internal representations for the same data type: the aligned/unaligned attributes. We have no choice but to use the unaligned attribute to mean "packed" for decimal values, if we wish to maintain a consistent PL/I language. Thus, in the new compiler decimal values declared with the unaligned attribute will begin on a digit boundary, and will occupy 4 bits per digit.

Compatibility considerations

By changing unaligned decimal values from byte-aligned to digit-aligned, and from 9-bit to 4-bit, we are creating a compatibility problem for programs written and compiled before the new compiler. If we are not careful, users who recompile old programs with release 24 will be unable to access data bases that contain (old) unaligned decimal values. Programs compiled with release 24 will be unable to call programs compiled with older releases, and vice versa. If we could identify old programs, we could issue warnings when they were recompiled with release 24, and give any user using unaligned decimal the chance to recompile all of the programs at the same time. Conversion of data bases may well require a program to read the old data base (compiled with an old compiler), and a program to write a converted data base (compiled with a new compiler).

This is an incompatible change, and will affect every user now using unaligned decimal data. The following steps will be taken by the compiler to aid in the identification and conversion of old programs.

1. identify old programs by the absence of a new option for the main procedure, "options(packed_decimal)".

2. issue a severity 2 error for every unaligned decimal variable in old programs.

3. whether the program is old or new, compile unaligned decimal values as packed decimal.

4. ship release 23b in MR7.0 as "pl1_r23", so that users can continue to compile old programs with an old compiler, as an aid to conversion.