From: W. Olin Sibert

To: MTB Distribution

Date: June 9, 1981

Subject: Generating Unique Bits for the ADP

This MTB discusses the way Multics will deal with the non-unique clock values returned by the ADP Calendar Clock, detailing the changes and incompatibilities which result. Please direct any comments or questions to the author:

By Multics mail at MIT or System-M to:

Sibert.Multics

Or by mail to:

W. Olin Sibert
Cambridge Information Systems Laboratory
HED MA22

HVN 8*261-9353

Introduction:

On the ADP, the Calendar Clock (in the CIU, the ADP equivalent of the SCU) is not guaranteed to return a different value each time it is read. This is a change from the clock behaviour of the Level 68 SCU clocks, which do return a unique value. This operation of the clock on the Level 68 is depended on for the generation of unique identifiers in several contexts, and this capability must be available on the ADP, as well.

The uses of the calendar clock in Multics fall into two major categories: (1) measurement of real time intervals, and (b) generation of unique values. The non-unique property of the clock reading on the ADP will have no effect on its utility for calculating time intervals, since it is supposed to always monotonically increase, and therefore all intervals will be either positive or zero.

Since the format of the clock reading (produced by the RCCL instruction) does not change from its present fixed bin (71) count of microseconds format, no program changes will be required to reformat the value. The only effect the different implementation has is in the generation of unique values.

The Situation Today:

Some programs invalidly depend on the PL/I "clock" builtin to return a unique value. The ADP clock may cause these programs to break, but this is rather unlikely. In order for any problem to occur, an actual collision of values must occur, and this is a very improbable event, since it requires that two such "unique" values have been requested within the same one microsecond interval. Furthermore, this "feature" of the Level 68 clock is not documented, and is not widely known in the field. Therefore, this change to the "clock" builtin should not be a problem.

For generating unique bit strings, there is already a defined interface: the unique_bits_ subroutine. This subroutine will be modified for the ADP to detect that it is running on an ADP system, and call a special gate entry which will get a unique clock reading for it, and return. This will satisfy the requirement for generating unique values. The unique_bits_ subroutine is already documented to return a value which corresponds to a clock reading.

To avoid any further problems, the documentation for the PL/I clock builtin (AG94, AM83), and the clock_ subroutine (AG93) will be updated to say:

> "The clock builtin (or clock_ subroutine) is not guaranteed to return a clock reading which is necessarily different from any other clock reading taken on this system. To get a truly unique value based on the clock reading, the unique_bits_ subroutine should be used."

An SRB notice will also be distributed with MR9.0, to warn of this problem, and suggest that programs be converted:

> "The practice of using the return value from the clock builtin function or the clock_ subroutine for generating a unique value is not guaranteed to work. Two simultaneous invocations of the procedure may result in the same value being returned twice. To avoid any problems resulting from this behaviour, the unique_bits_ subroutine should be used whenever a unique clock reading is required. Future hardware improvements may make this problem more serious, so any programs which are affected should be modiified."

Implementation:

The mechanism for generating unique bits will be implemented entirely within the unique_bits_ subroutine. If it finds that it is being called on a Level 68 system, it will simply use the clock builtin, and return exactly as it does today. Otherwise, it will check to see if it is running in ring zero, and, if not, will call the new gate entry, hcs_$unique_bits, to get its unique value.

The hcs_$unique_bits gate is just another way to call unique_bits_; when unique_bits_ discovers it is, in fact, being run in ring zero, it will go through a little routine which reads the clock (with the clock builtin), extracts the low order word, checks to see that it is different from the value in scs$last_unique_value, and, if so, uses the stacq builtin to update scs$last_unique_value. If the low half of the clock is not already different, it will loop, reading the clock, until it is. If this event does not occur within a reasonable number of tries (a few milliseconds), it will assume that all is lost, and crash the system because the clock is stopped. Similarly, if the stacq fails, it will try again until it succeeds.

Only the low order word of the clock is saved, because stacq can only be used to indivisibly update a word at a time. In any case, the absolute value of the unique clock reading is not interesting; only its uniqueness is. Even if two successive calls were made exactly 2**36 microseconds apart, so the low order word would be the same, but the high order word different, it would only occasion a microseconds delay, since it would appear the same as if two successive calls resulted in the same clock reading.

An additional modification must be made to init_pvt, which today checks the clock reading for a variety of malfunctions (running backwards, running slow, stopped, etc.) and crashes the system when it detects them, to avoid crashing the system because it appears that the ADP clock is stopped because it returned the same value twice in a row.

These changes should prove adequate for support of the ADP calendar clock. Since no data formats have changed (contrary to initial notice), ALM programs can continue to read the clock and use the value for time calculations just as they do today.