Timing Margin Requirements for AGC Programs

A recently reported problem with the revised LM H2 program, whereby anomalous throttle commands can occur in P66 for a TLOSS as low as 8% ("TLOSS" is a computer slow-down factor, which in Apollo 11 was 15% due to RR CDU inputs), has caused attention to be directed again at what a meaningful flight-qualification value for this parameter should be. There is obviously some maximum value beyond which no program can be expected to work, and on the other hand "negative" values clearly would not be acceptable.

One reasonable question to ask is whether there is any requirement for the program to function at non-zero values of TLOSS. The Apollo 11 problem, of course, demonstrates the desirability of some sort of safety margin for the program, but it might be argued that the RR counter problem is now "understood" and "fixed", and no other input counters can reasonably run away and still make it prudent to stay with the PGNCS (an obvious example is a bad IMU CDU input counter). The two manned flights previous to the G mission in the LM didn't encounter the problem, so probably one could have made the statement before the G flight that input counters were "understood" too.

In addition to possible noise effects (spacecraft vibration, for example), the major reason for having a positive TLOSS value should be that preflight computations can not duplicate exactly the environment to be encountered by the flight software. For example, although the machine language orders are independent of the particular bit patterns used, the interpretive language orders, in some cases, are not. In particular, the interpreter divide algorithm (also used for other purposes such as unit vector formation), in addition to various alternate paths depending on gross magnitude differences and sign agreement, requires an additional 20 MCT (about 0.23 ms) for each place the divisor must be shifted left (to normalize it). Other effects, of lesser magnitude, also occur in the interpretive language (checks for "legal" overflow, for example).

Besides the software "noise" execution, the software also is required to do a variable amount of work during a two-second Average-G cycle. One way in which this variability can arise, of course, is by crew DSKY manipulations, and as a result there is a general desire to avoid using the DSKY during descent. Another way the variability can arise is due to phasing of the channel 13 "stall" requirements (such as for telemetry) with respect to the radar readings (this can provide a delay of about 0 - 5 ms depending on the particular phasing involved). A third way is due to gyro compensation, where if more than 2 pulses are required (and at least one per gyro), a job is established and four tasks called (adding at least 5 ms more to the execution time).

Finally, there is the effect of crew-initiated or spacecraft-initiated inputs that influence the computer. One of the most dramatic of these is the RCS thruster fail switches, each of which causes the comparatively long "1/ACCS" routine to be executed (hence if crew disables a complete quad, two such jobs, 480 ms apart, will become established on top of the normal load). In addition, the DAP controls its own switching into and out of GTS control, although numerical information on the effect of such switching is not available. The gimbal trim computations seem more elaborate than the RCS ones, however, suggesting that another crew technique to save execution time would be to disable the trim system by the appropriate spacecraft switch.

As a consequence, it seems desirable that a program have a positive TLOSS capability before it is used on a flight. A specific number is not proposed at

this time, but it is suggested that the figure be sufficient to handle:

1. 1% for interpretive and other path-length variations.
2. ½% to handle "C13STALL" effects.
3. gyro compensation on this cycle.
4. RCS failure monitor (a 1/ACCS job) on this cycle (one).

together with the "normal full" load on the computer of R10, GTS, and the like.

In addition to the "handle" capability, it seems appropriate that the TLOSS procedure continue to be used as a method for stress testing the software, in order to discover areas of program performance which break down, and in order to determine if this breakdown is more sensitive than necessary. For example, it was noticed after the G mission landing that an extra job register set was required (briefly) in the descent/ascent programs because of the selection of display interface routine that had been made. As a result, for H1 the display interface routine selection was changed, presumably with no impact on performance and with a saving of one core set at this particular point in flight (previously the program established a separate display job, then did its own end-of-job). The general design objective should be that the performance of the program achieve a "graceful" degradation, rather than being subject to abrupt cliffs. A reasonable bound for such testing probably would be a factor that would allow the Apollo 11 problem plus the ones listed above (to handle the case of a failed RR channel status bit, for example).

Study efforts associated with TLOSS should not be restricted to "making a run and see what happens". Instead, a set of timing information on the performance of the various areas of the coding should be accumulated and subjected to manual analysis, since only by this means can the various phasing possibilities be evaluated economically. Such an analysis, for example, apparently <u>was</u> done for the original P66 (pre PCR 988) coding, although it apparently has only recently been considered for the PCR 988 version of the H2 program.