Massachusetts Institute of Technology
C.S. Draper Laboratory
Cambridge, Massachusetts

LUMINARY Memo #152

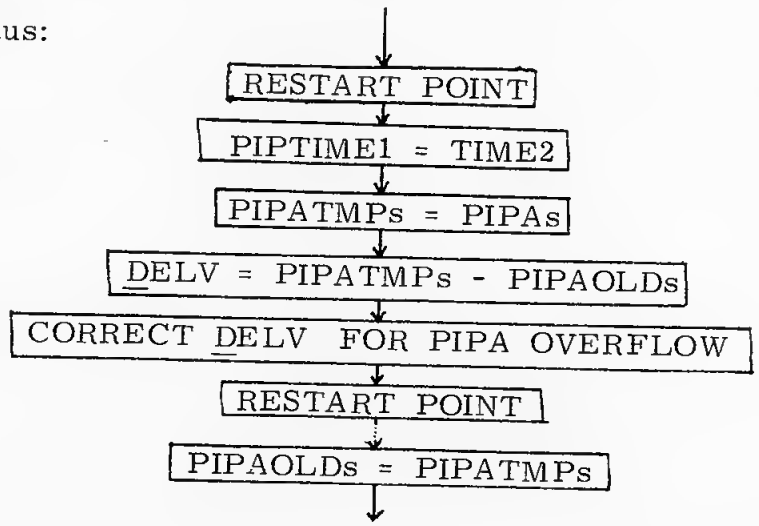
To: Distribution
From: D. Eyles
Date: 21 May 1970
Subject: Cyclical PIPA Reader

A PIPA reading philosophy has been developed which significantly simplifies powered-flight restart protection and facilitates asynchronous accelerometer reads outside of Servicer. This scheme, called the Cyclical PIPA Reader, is particularly suited to a Variable Guidance Period Servicer and is being used in version ZERLINA.

The philosophy is this: never zero the PIPAs (except in PREREAD) and compute DELV as the difference between each and the previous PIPA reading, suitably correcting if a PIPA has overflowed between readings.

In other words: after a preliminary restart point, time is stored in PIPTIME1 and the contents of the PIPAs are transferred into the PIPATMP registers. DELV is then computed component by component as the difference between the PIPATMP contents and the contents of the PIPAOLD registers. Next the components of DELV are corrected for the chance that between this and the previous reading the corresponding PIPA has overflowed, i.e. passed POSMAX (or NEGMAX) and started counting up (or down) again from zero. Finally, after another restart point the contents of the PIPATMPs are moved to the PIPAOLDs.

Thus:



This renders unnecessary the complicated and unlovely restart protection logic which currently takes up most of the subroutine PIPASR. Note that a restart anywhere between the first and second restart points will cause control to be transferred to the first restart point. The clocks and PIPAs will then be read again and DELV recomputed. The second reading may not see the same values in the PIPAs and clocks, but this doesn't matter. A restart after the second restart point merely causes the PIPATMPs to be retransferred into the PIPAOLDs.

For asynchronous PIPA reads the problem is simplified because they do not have to reckon with a READACCS juggernaut which periodically comes around and destroys values in the PIPAs. This destruction turns out to be wanton. Now an asynchronous read can preserve its own PIPAOLD type registers and compute its DELV just as Servicer does, with no mutual interference.

If Servicer's copying of the PIPATMPs into the PIPAOLDs is not done immediately after the PIPA reading but is done at COPYCYCL where also R1S, V1S, PIPTIME1 and G1 (which in ZERLINA replaces GDT1/2) are under INHINT copied into R, V, PIPTIME and G, then this fact follows: that at any time whatsoever (T_1) PIPA - PIPAOLD is the thrust delta-v needed to compute current position and velocity from $\underline{R}$ and $\underline{V}$. This computation is always:

$$\Delta T = T_1 - \text{PIPTIME}$$

$$\underline{R}_1 = \Delta T \left(\underline{V} + \frac{(\text{PIPA} - \text{PIPAOLD}) + \Delta T\, \underline{G}}{2}\right)$$

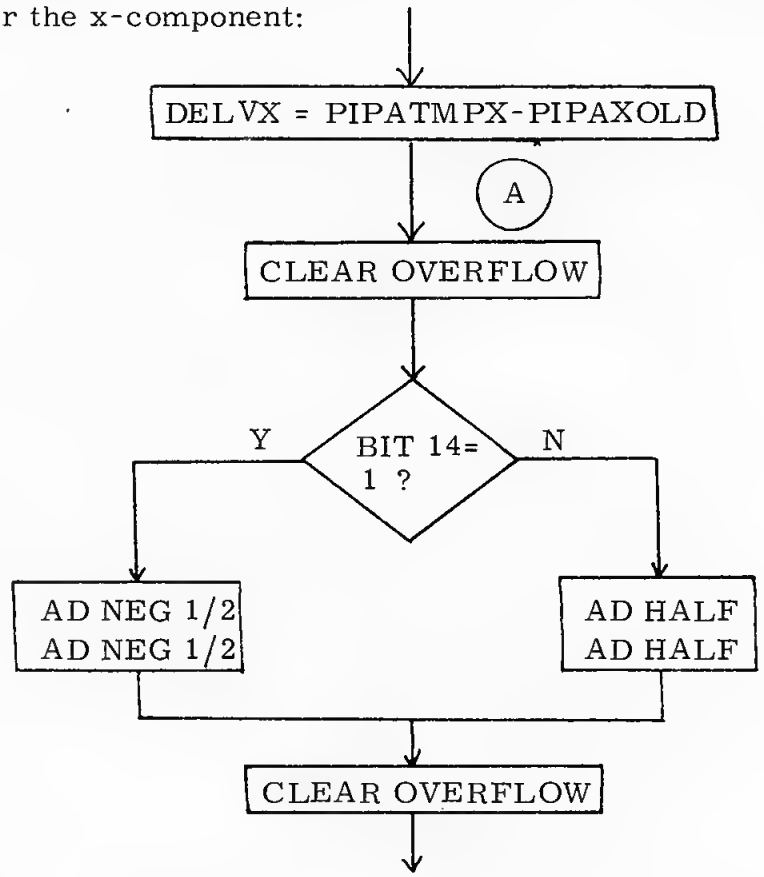
$$\underline{G}_1 = -\frac{M}{|\underline{R}_1|^2}\ \text{UNIT}(\underline{R}_1)$$

$$\underline{V}_1 = \underline{V} + (\text{PIPA} - \text{PIPAOLD}) + \frac{\Delta T\, \underline{G} + \Delta T \underline{G}_1}{2}$$

In other words, the PIPAOLDs make a consistent set with R, V, PIPTIME and G. This consistency, and the consequent uniformity of computations, simplify the Landing Analog Displays, P66 ROD, and the Landing Radar Update routine, all of which in ZERLINA make such an extrapolation.

I should explain that as implemented in ZERLINA the registers I have called the PIPATMPs are PIPATMPX, PIPATMPY and PIPATMPZ. The so-called PIPAOLDs are PIPAXOLD, PIPAYOLD, and PIPAZOLD.

As the reader may have guessed the corrections of DELV for PIPA overflow is the key to the Cyclical PIPA Reader. The method used is as follows for the x-component:



Although it may not be instantly apparent, the assumption that makes this logic work is that the difference between successive PIPA readings can never amount to half a register or more unless PIPA overflow has occurred. In other words, that delta-v can never exceed 81.91 m/s over one PIPA interval. The worst conceivable LM case is when both APS and DPS tanks are (about) empty and the DPS engine is thrusting at full throttle. Even in this case the PIPA interval, i.e. guidance period, would have to exceed 8 seconds for the assumption to be violated. It should nevertheless be explicitly recognized that an assumption is being made.

That the logic charted above is valid for all cases can most easily be seen by considering certain ranges of value taken on by the now PIPA difference, call it ΔV, before any correction is applied (at point A in the flow chart):

(a) ΔV positive, $|\Delta V| < 81.92$ m/s, 00000 to 17777
(b) ΔV negative, $|\Delta V| < 81.92$ m/s, 77777 to 60000
(c) ΔV positive, $|\Delta V| \geq 81.92$ m/s, 20000 to 37777
(d) ΔV negative, $|\Delta V| \geq 81.92$ m/s, 57777 to 40000
(e) ΔV positive with overflow
(f) ΔV negative with overflow

Cases (a) and (b) are the most common ones. The logic dictates that (a) POSMAX + 1 (or HALF twice) be added to the positive ΔV and (b) NEGMAX - 1 (or NEG1/2 twice) to the negative ΔV. These additions are in fact vacuous in these cases because once overflow is cleared the difference ΔV is unaffected. DELV = ΔV in other words — the final value equals the raw value. Examples of processes which lead to cases (a) and (b) are shown next, with the PIPA counts written above the arrows and all numbers in octal:

$$30567 \xrightarrow{+15} 30604 \Longrightarrow \Delta V=00015 \Longrightarrow DELV=00015$$
$$57432 \xrightarrow{+123} 57555 \Longrightarrow \Delta V=00123 \Longrightarrow DELV=00123$$
$$77700 \xrightarrow{-20} 77660 \Longrightarrow \Delta V=77757 \Longrightarrow DELV=77757$$

Cases (c) and (d) result when the PIPA in question has overflowed during the PIPA interval. For instances:

$$37765 \xrightarrow{+20} 00005 \Longrightarrow \Delta V=40017 \Longrightarrow DELV=00020$$
$$40021 \xrightarrow{-40} 77761 \Longrightarrow \Delta V=37740 \Longrightarrow DELV=77737$$
$$37770 \xrightarrow{+12} 00002 \xrightarrow{-4} 77775 \Longrightarrow \Delta V=40005 \Longrightarrow DELV=00006$$

The logic causes twice HALF to be added to the big negative ΔV, or twice NEG1/2 to the big positive ΔV. This procedure yields the right result: when a PIPA has overflowed, to correct the difference simply add back in a full register of counts of the right sign.

Cases (e) and (f) will be extremely rare, but can result from a PIPA history similar to the last example given above, in which the PIPA

counts reverse direction during a PIPA interval. For instances:

$$40017 \xrightarrow{-22} 77773 \xrightarrow{+30} 00026 \Longrightarrow \Delta V = {}^{0}40006 \Longrightarrow \text{DELV=00006} \quad \text{(e)}$$

$$37770 \xrightarrow{+12} 00002 \xrightarrow{-14} 77765 \Longrightarrow \Delta V = {}^{1}37775 \Longrightarrow \text{DELV=77775} \quad \text{(f)}$$

(When sign bits disagree the dominant one is shown at the upper left.) These cases are the reason for including the initial clearing of overflow, for it turns out that clearing overflow reduces cases (e) and (f) to cases (a) and (b), and leads to the correct answer.

In the current ZERLINA this correction process is performed by a subroutine PIPNORM whose input is the raw difference ΔV and output the corrected difference. When (and if) Variable Servicer is put into LUMINARY the routine PIPNORM will replace VACRLEAS (not needed in ZERLINA) in fixed-fixed memory, making it available to P66ROD and the Landing Analog Displays which need to perform the correction but occupy fixed banks remote from Servicer.

The reader may have realized that there is a lesson for the future in this involved explanation of the correction process. This is that PIPAs, like CDUs, should work in 2's complement. If this were the case a simple modular subtract would suffice to compute DELV in all cases, and the engine in question would not have to oblige by providing less than half-a-register in PIPA counts over each PIPA interval.