

TI-59 Printer Interrupt: Stopping a Printed Line Halfway

What the printer interrupt is

The PC-100 printer prints a line as 7 rows of dots and needs about 0.2 s for it. It works on its own: the calculator sends the line and goes on with the program. TI-59 users found a program step that stops the printer while it is still printing. The line then appears with only its first rows of dots. Programs use this for graphics with a finer vertical resolution than whole characters ([texas-print.ti59](#), the TI logo) or to show truncated characters ([printer-quirks.ti59](#)).

[printer-interrupt.ti59](#) is the smallest example. It prints "A0" once in full and once cut off after 2 rows.

The hack

The interrupt is a program step with the keycode 1F. Keycodes are two decimal digits, so no key can record it. It has to be manufactured with a firmware bug that leaves a hexadecimal digit in program memory.

When the program reaches 1F, the firmware turns its two digits into a jump address without checking them. The jump lands three instructions before the signal that tells the printer "line finished, advance the paper". The printer is still busy with the line before, and the early signal stops it.

Rules for using it

- Put it right after the print: `0 0p 05 1F`. The number of rows printed depends on the time between the two. With nothing in between, 2 of 7 rows are printed; slower steps in between give more.
- The display must be 0, with no EE mode and no error, when 1F runs.
- Step 000 must hold `INV SBR`, and the step after 1F must be RTN. After the interrupt the program continues at step 000. `INV SBR` there returns to the RTN, and the RTN returns from the subroutine that holds 1F.
- Call that subroutine with a label (A in the examples). All labels must come before 1F.
- 1F can only be made at a step divisible by 8, from 008 to at most 376.
- Cards recorded after 1F was made do not keep it. Record the program before making the code.

Making the code

Both methods use LRN and Ins on a corrupted step buffer (see "How the code is made" below). They differ in how they corrupt it.

- **The article's sequence** (PPX Exchange, March/April 1982; [texas-print.ti59](#) documents every step). Key `lnx (` at step 008, then
`GT0 008 9 0p 17 Pgm 12 SBR 444 R/S P/R LRN Ins LRN CLR RST`. It needs the Master Library module and the partition 9 or 10 Op 17, and goes through four separate firmware bugs.

- **The fast-mode hack** ([printer-interrupt.ti59](#)). Put St flg on step $8k+4$, a 0 on $8k+6$ and R/S on $8k+7$, where $8k$ is the step for 1F. For 1F at 008: key `lnx (` at 008, St flg at 012 and R/S at 015, then key `1 + 48 EE 12 +/- = GT0 012 SST 0 ST0 LRN Ins LRN CLR RST`. No module and no special partition are needed.

The keys at $8k$ and $8k+1$ must suit the ROM data used for that location. The article's table lists them for 008–104; both methods use the same table. For 008 they are `lnx (`, which become 1F and RTN.

Ins also moves every step after the location up by one, up to the end of the partition. The last step of the partition is lost. The 6 steps after the RTN get meaningless codes.

How the interrupt works

Dispatch. For every program step, the firmware splits the keycode into its two digits and builds a jump address from them (ROM `0F53`). Legal digits are 0–9. Nothing checks this, so the digit F builds an address too. For 1F it is `0EF1`.

The landing. `0EF1` is in the middle of the routine that prints a number. Three instructions later that routine sends PRT.GO, the signal that ends a printed line. With the display at 0, positive and without EE, the branches in between skip everything else and go straight to PRT.GO. Other display contents take other branches, which is why the article warns of hangs.

The printer. Op 05 sent the line a moment before. The printer is still printing its dot rows. PRT.GO while it is busy makes it stop. In the example, 1020 of the 2808 instruction cycles of a line have passed, so 2 of the 7 rows are on paper.

The way back. After PRT.GO, execution continues at step 000, and the step after 1F is left as a return point, as after a subroutine call. The trace shows the steps 000, 009 and then the caller: `INV SBR` at 000 returns to 009, and the RTN there returns from the label subroutine.

How the code is made

You need three facts about the calculator's internals.

1. **The step buffer.** The firmware reads 8 program steps at a time into a buffer and works through it; [TI-59 Fast Mode](#) explains the details. The buffer is normally a copy of the RAM register the step counter points to.
2. **The program source.** A digit in the status register says where program steps come from. Its value 8 means the ROM's built-in keycode programs ($\Sigma+$, $P \rightarrow R$, ...). The buffer then reads from a ROM table row instead of RAM.
3. **Hexadecimal digit 0.** The CPU's arithmetic is decimal in digits 1–15 of a register, but hexadecimal in digit 0. Digit 0 of a program register is the units digit of its first step, $8k$.

Ins trusts the buffer. Ins computes each register as $(\text{RAM} - \text{buffer}) + (\text{buffer shifted up one step})$. With an honest buffer this is just the shift. With a wrong one, the difference is garbage, and in digit 0 a borrow gives a hex digit instead of a decimal one.

Corrupting the buffer. Both methods leave the program source at 8 while the calculator is stopped just before step 8k. LRN then moves to 8k, a new block, and loads the buffer from ROM row $16 + k$ instead of RAM. For 008 the row starts with 04 65. Ins at 008 then gives:

- step 008: $23 - 04 = 1F$ (units $3 - 4 = F$ with a borrow, tens $2 - 0 - 1 = 1$);
- step 009: $53 - 65 - 1 + 04 = 92$, the RTN.

Setting the source to 8. The article's sequence gets there through a failed module call (Pgm 12 SBR 444), whose cleanup is incomplete. R/S then runs bytes of the module header. P→R finally writes source 8 and then gives up because its own step lies outside the small partition. The fast-mode hack writes the source directly: $X = 1.000000000048$ holds source 8 and step 004. A non-zero source starts a run. The buffer still holds the block with St flg, so steps 005–007 run the three steps after St flg: the first is swallowed, the 0 makes X displayable, and the R/S stops the run at step 007 with the source still 8.

Limits

Everything here was observed on Calc-U-59, which runs the original TI-59 ROM unmodified, and not on a real TI-59. The number of rows printed depends on the timing of the calculator and the printer, and real machines differ, as the PPX article notes.