IME 86S Calculator
Theory of Operation

bhilpert / madrona.ca
2024 Sep
Rendition: 2025-Jan-31

Contents

Introduction

The IME 86S is a desktop calculator from the mid-1960s. A digitised copy of the original manufacturer schematic was obtained from the DoPECC website [https://dopecc.net]. This original IME schematic is limited to a representation of the discrete electronics. The IME schematic has been re-interpreted to produce a logic schematic with block diagram. This commentary has been produced from analysis of the schematics.

[reference]: Text within square brackets is a reference to some page, figure or logic element in the new logic schematic. E.g. [logic.15] [17.1PL].

Architectural Overview

The IME 86S is a digit-serial design. At a functional architectural level, data processing involves 7 registers, each holding a 16-digit signed decimal number comprised of 4-bit BCD-encoded digits, and several decade counters. Arithmetic is performed in the counters via counting algorithms, not binary arithmetic.

The digits of the 7 registers are implemented in a magnetic core memory while the signs are held in a bank of flip-flops.

The control and sequencing of the higher-level arithmetic and entry operations is implemented with assorted flags and two sequence counters which execute micro-programs.

Electronic Implementation of Logic

The IME 86S is implemented in discrete Ge solid-state electronics. Active components are Ge transistors, primarily PNP. As a PNP-based design the main power supply for the logic is positive-ground, with Vcc = –12V. A base-bias supply of +6V for the logic is also present to ensure transistors are pulled into cutoff, to compensate for the forward voltage drop from diodes in the logic. The transistors are used for flip-flops, inverters, a few buffers, and in inverting 'pulsers'.

The internal construction of logic elements is presented on [logic.22::24].

Logic Nomenclature

• Logic 0 = 0V

• Logic 1 = –V

• 0-edge = transition from logic 1 to 0, positive-going voltage edge

• 1-edge = transition from logic 0 to 1, negative-going voltage edge

Gates

Gates are constructed with Ge diodes, in a form which may be called "Diode-AND-OR-Logic". This form allows a limited degree of cascading of gates without active devices in-between. 'Wired' gates are also used.

Flip-Flops

Flip-flops were expensive to implement in discrete form, giving rise to optimisations that appear quite odd in the post-discrete era of integrated circuity.

Capacitor triggering was common. The capacitive storage and RC time constants provide a storage element prior to the active transistor flip-flop, in a limited sense functioning as a master stage to an active slave stage. The capacitive storage allows a zero-to-negative hold time for the flip-flop. At the time, this was more economic than full M-S flip-flops.

IME 86S flip-flops are formed starting with a standard base-portion of two transistors and associated loopback and bias components. A range of input constructs may then be applied to the base-portion, in single or in multiple.

Flip-flop construction is presented on [logic.24].

Pulsers & Asynchronism

Producing a fully synchronous design can be costly, with doubled flip-flops and more gates. In avoiding this, the IME-86S design - as with other discrete-electronic logic designs - has significant asynchronous aspects. A readily apparent example of this is the counters all being of ripple form.

One of the techniques used in asynchronous designs is the introduction of a circuit element which will be called here 'pulsers'. Pulsers produce a pulse at their output when an edge

occurs at their input. These may be used for objectives such as:

- To delay an edge briefly to bridge over a period in which glitches may be occurring, with the delayed edge then triggering some action.
- Producing a brief pulse to quickly perform some action. For example, at the beginning of a subroutine, QBEN is asserted. The 1-edge of that assertion triggers [8.8PP] to produce a quick 0-pulse to clear the TC Counter in preparation for the subroutine.
- To disambiguate actions triggerred by the same event.

• Pulser types — There are several types of pulsers to provide different edge response and pulse polarity. Some pulsers also have a gate input which must be enabled first for a pulse to be propagated out when an edge occurs.

• Tuning — Pulsers do not all produce pulses of the same period. There was some amount of 'tuning' performed in the design to achieve the desired objective of each pulser, as evidenced by the range of RC values used [logic.23].

Semiconductor Replacement

The Ge diodes and transistors of the implementation may be difficult or inconvenient to obtain more than 50 years after the production of the design. This isn't really much of an issue in effecting repairs as Si tends to be fine for replacements.

• Diodes — The higher forward junction voltage of Si versus Ge might give rise to a concern around threshold voltages.

The concern is that for AND-form gates if the Vf drop were too high, a 0 wouldn’t be able to shut off the transistor the gate eventually feeds into. This cascades where an AND gate directly feeds another AND gate, with an accumulating voltage drop from diodes in series. 'AND gate' here includes many of the multiple inputs of flip-flops which are essentially negative-logic OR gates (AND-form).

However, the bases of most transistors are pulled down well below conduction by a bias resistor to V+6. This provides some headroom before increased Vf drop would leave a transistor conducting.

Simple silicon switching diodes such as 1N914 or 1N4148 are likely acceptable for most or many gate diodes.

Schottky diodes with their low Vf similar to Ge are the other option. The BAT46 is one suggestion, and was used in repairs of one unit.

• Transistors — The following standard Si transistors were used for replacements in repairs of one unit:

- 2N3906: logic transistors (flip-flops, inverters, etc.)
- 2N2907: core drivers
- MPSA92: Nixie cathode drivers

Timing & Data Elements

Timing

A timing diagram for the major clock signals is presented on [logic.21].

- Master clock — Timing begins with an astable flip-flop operating at ~ 89 KHz generating signal ØM.
- Digit period — The master clock triggers a 4-bit binary ripple counter: the ØC Counter [FC1,2,4,8]. The 16-count cycle produced by this counter constitutes a digit cycle or period (ØD) during which a digit of a number is processed.

  The 16 ØM pulses occurring in a digit cycle are used to trigger various actions. The primary aspect of the digit cycle is to read a digit of each of two operands from core, optionally perform arithmetic with them, and restore them to core:

  - ØR: Two pulses in ØR read a digit of each operand from core.
  - ØA: A span of 10 ØM pulses used for the decade counting arithmetic.
  - ØW: Two pulses restore the two operand digits to core, one of which may have been modified by arithmetic.

  ØGS alternates in-between the two pulses of ØR and ØW to distinguish the selection of the two operand registers.
- Digit-of-number — The 0-edge of FC8 represents the end of one digit and the beginning of another. This edge clocks another 4-bit binary ripple counter: the ØD Counter [FD1,2,4,8]. The 16 states of this counter define the digits of a number, from ØD0 (LSD) to ØD15 (MSD) to constitute a number cycle.
- Up/down — The ØD Counter can count up or down, so can scan or process a number from LSD to MSD or MSD to LSD. The default counting direction is up, LSD to MSD [29.10Nv].
- ØN — A 5th flip-flop [FØN] in the ØD Counter pauses the clocking of the counter for one digit period (ØN) at the end of number cycles. The ØD Counter is 0 during ØN.

Digit Memory (Core)

The digits of the numbers held by the 7 registers are stored in a small core memory.

- Encoding — The digits are encoded in standard 4-bit 1248 BCD code.
- Physical — The core memory is physically organised in a 2D planar array of 16 • (8 • 4).
- Logical — Logically, the core memory is a 3D array of 8 registers, by 16 digits/register, by 4 bits/digit. However, of the 8 register planes, only 7

are used - the 8th is spare.

• Electrical

Electrically, the form is what was referred to in the era of core memory as "2-1/2 D".

In the common 3-Dimensional form of core memory, 2D half-current addressing is used, with the 3rd dimension of bit-arrays sharing 2 sets of address drivers for the wires on both axes of the bit-arrays. Each bit-array has a 3rd inhibit wire to distinguish between bits, and the sense function is performed with a fourth wire for each bit-array, or in later designs inhibit and sense share a 3rd wire.

In the 2-1/2D form, 2D half-current addressing is used with a set of address drivers/wires shared on only one axis. On the other axis each bit-array has an independant set of drivers/wires with the drivers of each set having a common gating input to distinguish control of the bits.

This 2-12D form requires more drivers than 3D but simplifies the threading of the cores.

• Shared sense

With the 2-1/2D form, a separate sense wire may be avoided. Rather, sensing can be accomplished using the address wires of the independant axis. This is made feasible by differentiating the timing of the address pulses on the two axes.

In the IME86, the register/row wires are used for both addressing and sensing. During a Read of core, read current on the register/row wires is enabled for period ØRL which is longer than the period of ØR used for the digit/column wires. The magnetic flip which will produce the sense pulse is triggered by ØR on the digit/column wires. These wires are orthogonal to the register/row wires which are observed for the sense pulses, so the magnetic field of the triggering pulse has minimal inductive influence on the register wires and the small sense pulses remain distinguishable from the triggering pulse.

• Double winding

The row and column wires are threaded through the cores twice, having been looped back underneath the core mat. This permits halving the current requirement of the drivers in achieving the needed magnetic influence, as well as doubling the sense pulse level into the sense amplifiers.

• Drive supply

Core requires bidirectional current drive through the cores, one direction for clear/read, one for set/write. This is accomplished in the IME 86 in a novel manner with the single –12V supply. The address wires all have one end connected to a common point held at –6V. The drivers then can switch the other end to – 12V for one direction (clear/read) and to ground for the other direction (set/write).

The –6V however, is derived not from a typical dropping of the –12V supply, but by what amounts to a charge pump driven by the core accesses.

In the calculator application, the core accesses follow a persistent and regular pattern. A capacitor [1000µF on V–6] between the –6V common

point and GND serves as the store for the charge pump. During the clear/read accesses the driver end of address wires is taken to V–12 and current flows in one direction through the wires to charge the capacitor. During the set/write accesses the driver end is taken to GND and current flows in the other direction, discharging the capacitor. A write always promptly follows a read so there is a near balance between charge and discharge of the capacitor. Imbalance however, results from:

- Set/write does not drive current in the register/row wires for bits=0.
- Clear/read has a longer ØRL pulse on the register/row wires.

Both of these present an imbalance in favor of charging. To compensate, a shunt regulator [TP.7] controlled by differential pair [TP.5,6] is applied across the capacitor to bleed off excess charge and maintain the core common point at –6V.

Registers

The 7 registers are each comprised of its digits in core memory and a sign flag.

• Code

Registers are identified internally by a 3-bit code:

Register	Code	GG~	BB~	AA~
R1	0	0	0	0
R3	1	0	0	1
R2	2	0	1	0
RT	3	0	1	1
R4	4	1	0	0
RA	5	1	0	1
RB	6	1	1	0
-	7	1	1	1

Note the coding here uses the 0-assert values. Nearly all the register-code signalling is done with these values.

The coding was chosen such that the values with only one zero bit are applied to the A, B & T registers. These registers are frequently referenced in the micro-programs, the single zero bit reduces the number of diodes needed for their referencing.

• Access

At any given instant, one register can be accessed. Accessing a register means a digit of that register will be read or written during a core memory operation (single pulse of ØR or ØW). There are two sources for selection of the register to be accessed:

- the GAA~,BB~,GG~ signals (collectively GGBA),
- the 3 current-register flags FAA,BB,GG (collectively FGBA)

[logic.17]. The current-register is the register being displayed.

By default, the register selected for access is the current-register (FGBA). The selection is switched to GGBA by assertion of GSEL [19.15Nv] which is subject to a variety of conditions.

- GGBA — The GGBA signals are generated on demand by the procedure logic to access a specific register.
- FGBA — The FGBA current-register flags can be loaded from either:
  - the keyboard when the user selects a register for display,
  - procedure logic such as for automatic selection of the T register for display at the completion of an operation.
- Decoding — The 3-bit register code is partially decoded to a major-minor 1-of-2 & 1-of-4 code [GMx]. These signals are used to select both the register digits in core and the register sign flag. Further decoding to the individual register is done in the core drivers and sign flags.

- **Shifting** — The digits of a register need to be shifted up or down in several procedures. A shared facility performs this, involving the FGZ flag and the X & Z Decades.
- Enable — Shifting is enabled by the shift signals GSL (left), GSR (right), and FESH for numeral-entry. These assert GSH [3.4Nv].

  FGZ by default is held in reset, and released from reset by GSH=1. Once thus enabled, FGZ toggles at ØA1 which occurs in the digit cycle between the read and write of core.

  In the first digit period, X is loaded with a digit value from core. Z may be zero or it may have a non-0 value from activity prior to the shift enable.

  FGZ toggles to 1 on the first ØA1 after being enabled, asserting GZ and deasserting GX, so while X was loaded with the current digit from core, Z is written to that digit in core.
- Leap-frog — In the next digit period, Z will be loaded from core, FGZ toggled to 0, and X written to core. This leap-frogging with X & Z shifts the digit contents, continuing till GSH is deasserted.
- Direction — The shift direction is determined by the count direction of the ØD Counter. ØD counts up by default, producing a left shift. GSR assertion changes the ØD count direction to down for right shift.
- Fill — Shift-Right may either zero-fill the MSD (GSRz) or recirculate the LSD into the MSD (GSRc). When shifting right, the first digit period is ØD0, and ØDC then decrements to ØD15. For GSRc, this moves the LSD into the MSD. For GSRz, the write pulse in ØD15 is suppressed [29.9NG]. The last digit write of moving digit 1 to digit 0 takes place during the ØN period when ØDC=0.

• **Transfer** Some procedures require copying the contents of the current register to another register. This is performed by asserting GTR for a number cycle:

- GSEL will switch the accessed register.
- The FGBA register will be accessed when ØGS=0.
- The GGBA register will be accessed when ØGS=1.
- Sense pulses are disabled when ØGS=1.
- Decade switching is not in effect, so X is the only decade used in core accesses.

In consequence, during a digit cycle:

- On the 1st ØR pulse, X will be loaded with an FGBA digit.
- On the 2nd ØR pulse, the GGBA digit is read-accessed, clearing the cores, but any sense pulses do not affect X.
- On the 1st ØW pulse, X is restored to the FGBA digit.
- On the 2nd ØW pulse, X is written to the GGBA digit.

For sign transfer, FSQ is loaded with the sign of the FGBA register (ØGS=0). With GTR asserted, the FSQ outputs are gated to SETN and RESETN when ØGS=1, loading the GGBA sign with the FSQ value.

Sign Flags

The sign flags for the registers are provided by a bank of 7 flag flip-flops [FSA,B,T,1,2,3,4].

• Addressing The sign flags are addressed by the same GMx signals used for addressing the register's digits in core memory. The addressed sign flag becomes the target for modification by the SETN & RESETN signals, and becomes the source for the signal SN.

• Operand signs For operations involving multiple registers - where the register selection changes over the course of the operation - two further flags are used to hold the signs of the registers at issue [FSP,FSQ].

For signed arithmetic operations, FSP & FSQ are XOR'd to see if they differ. If they do differ, GASUB is asserted [4.6Nv] and subtraction is performed rather than addition.

• Change sign The FCS flag detects and holds whether the result of a subtraction produced a negative result. At the end of a subtract (GASUB~=0) number cycle, if a borrow from the last digit is present (FYC=0), the 0-edge of ØN sets FCS=1.

FCS is also set by a pressing of the Negative key.

With FCS=1, during ØN, FSP and FSP~ are gated to signals RESETN & SETN respectively - note the inversion. FSP was earlier loaded with the sign of the current register. During the ØW pulse(s) of ØN, these signals are gated through to set or reset the sign of the current register, thus

flipping the sign.

Digit Decades X,Y,Z

Three 4-bit flip-flop registers are used for the processing of digits.

- X & Y — During add/subtract number cycles, X and Y hold a digit from each of the two operand registers. Both X & Y are wired as decade counters to serve this end. X is also used for shifting register digits and the multiplex display scanning.
- Z — The Z Decade is actually wired simply as a binary counter. It is used for shifting register digits, holding a numeral entered from the keyboard, and as a counter in resolving new digits during the MP/DV/SR procedures.
- Selection — At any instant one of the three Decades is selected for loading-from and writing-to core memory. By default, the X Decade is selected [GX]. The assertion of either GY or GZ deselects X and switches the selection to the according Y or Z. Z is primarily selected for register-shift operations.
- 2 operands — Y is selected in 2-operand arithmetic operations. As with the Registers where ØGS is used to switch the accessed Register, so is ØGS used to switch the selected Decade. Assertion of GXY enables the switching of the selection. X is selected during the 1st of the ØR and ØW pulses, while Y is selected during the 2nd ØR and ØW pulses.

Arithmetic & the Digit Cycle

The arithmetic logic performs an addition or subtraction with the two operand digits in the X and Y Decades. The operations performed are:

GASUB=0: Y = Y + X

GASUB=1: Y = Y – X

This is accomplished with counting techniques, not binary-addition logic. Performing the operation is tied tightly to the interior timing of the digit cycle.

- Reading — A digit of the two operand registers must first be loaded from core into the X & Y Decades.

  At the beginning of the digit period, X & Y are cleared by the brief pulse of ØPCL [30.12PH]. The 1st pulse of ØR then triggers a core read. Sense pulses from the bits of the digit being read 1-set the flip-flops of X. The 2nd pulse of ØR triggers a core read from the other operand register to 1-set the flip-flops of Y.

  The selection between the two operand registers being read, and X versus Y being loaded, is controlled by ØGS and other control signals. ØGS toggles state between the two ØR pulses to change the selection.
- Add — By example, given X=7 and Y=5. An operation condition enables the arithmetic logic [5.12Nv]. When ØA begins, FXR is released from reset.

  During ØA, gate XINC is enabled to allow 10 pulses of ØM through to

increment X. X will roll over after 3 pulses, the rollover triggers FXR=1.

With FXR=1 and GASUB=0 (add), the gate YINC is enabled to allow ØM pulses to increment Y. ØA ends after another 7 pulses, setting FXR=0 and disabling incrementing of Y.

The net result when ØA ends is:

- X returns to its initial value of 7 having been incremented 10 times.
- Y has been incremented 7 times from 5, rolling over and ending at 2.
- The rollover of Y has set the Carry Flag FYC=1.

• Carry-in

If there is a carry-in (FYC=1) from the previous digit, Y is incremented during ØA1. This does not conflict with the FXR-gated increments because FXR will be enabled at the earliest at the 2nd pulse of ØA (X=9).

FYC is cleared at the end of ØA1, so is ready for a carry generation from the current digit, and again, prior to occasion for conflict.

• Subtract

For subtraction, GASUB=1. During ØA, the behaviour of X is the same.

However, incrementing of Y is now enabled while FXR=0 with the exception of ØA1 [5.11Nd] - thus from the 2nd pulse of ØA till the X rollover. The exception is tied up with the borrow strategy.

• Borrow

A borrow necessitates *one less* increment of Y. The ØA1 exception produces this borrow by default. As the carry flag produces an increment of Y, if the carry flag is set such that its assertion indicates the *lack* of need for borrow, the carry increment will compensate for the default borrow.

When subtracting, rollover of Y indicates *no* borrow is necessary. For example, for (5–2): Y=5,X=2, the 5 will be incremented 8 times producing Y=3 and FYC=1. FYC is as desired for providing the compensating increment to the next digit. For (5-6): Y=5,X=6, the 5 will be incremented 4 times producing Y=9 and FYC=0. With FYC=0, the default borrow is effective in the next digit.

An initial increment of Y is generated for the first digit, this compensates for the missing increment in that digit.

• Writing

After ØA completes, the two pulses of ØW restore the X digit to one operand register - necessary due to the destructive read of core - and writes the modified Y digit to the other operand register. ØGS again alternates between the two pulses to change the register and Decade accessed.

DP Counter

The DP Counter (DPC) is a 4-bit binary up-down ripple counter. It maintains the decimal point position as selected by the user with the shift-left and shift-right keys. The counter value is modified only by the pressing of the shift keys, its value is otherwise static. See the DP-Shift

procedure for more detail.

- Value — The value in DPC corresponds to the tenths digit, not the units digit. If the DP position is set to one fractional digit, DPC=0, as the LSD digit is processed and displayed during ØD0. If the display is all digits integral, DPC=15.
- Comparator — The value of DPC is continuously compared to the ØD Counter. When they are equal, signal DP=ØD is asserted. This is used, in part, to turn on the appropriate DP lamp in the display as the display is being multiplexed.

Tracking Counter

The Tracking Counter (TC) is another 4-bit binary up-down ripple counter. It serves for two uses:

- Tracking the digit position during the Numeral-Entry procedure after the DP key has been pressed.
- As a loop counter to track progress during the MP/DV/SR procedures.

Procedure-specific operation of TC is covered in more depth in the according procedure description.

- Direction — The default count direction is down. The MP/DV/SR procedures may alter the direction to up.
- Clear — TC is cleared to 0 by a brief pulse [8.8PP] when the Subroutine Sequence Counter QB is enabled.
- Comparator — Like DPC, the value of TC is continuously compared to the ØD Counter. When they are equal, signal TC=ØD is asserted.
- TC=DP — A further comparison signal of TC and DPC is generated. During a number cycle, if TC, DP and ØD all become equal, FSC3 is set 1 asserting TC=DP. This occurs at the end of the digit period when they were all equal. TC=DP remains asserted until the beginning of the next number cycle when FSC3 is cleared 0.

Execution Elements

Procedures & Programs

A procedure is a sequence of actions taken to perform some function. Procedures may be categorised into:

- **Simple**: Executed under the control of flags or keypress signals.
- **Main Program**: More-complex functions are performed by execution of a micro-program. Main Programs are sequenced by the QM Counter.
- **Sub Program**: A main program may invoke a subroutine, sequenced by the QB Counter.

This leads to the following table of procedures:

Procedure	Simple	Programmed		Select Signal
		Main	Sub	
Idle	•			not EXEC
Display Select	•			KMEM
Clear General	•			FRS=1
DP Shift Left	•			FRS=3
DP Shift Right	•			FRS=2
Numeral Entry	•			FEEN
Negate	•			KNEG
AddA		•		PEQ•N•PEQp
AddC		•		PEQ•(N~)•PEQp
Multiply/Divide Queue		•		PMD•N
Multiply		•		PEQ•N•PMPp
Divide		•		PEQ•N•PDVp
Multiply/Divide Chain		•		PMD•N•PMPp, •PDVp
Raise Exponent 1		•		PMM•(N~)•(Nn~)
Raise Exponent 2		•		PMM•(N~)•Nn
Raise Complete		•		PEQ•(N~)•Nn
Square Root		•		PSR
?PostRoot		•		PEQ•N•PRRp
Multiply			•	BMP
Divide			•	BDV
Square Root			•	BSR

The procedure logic can be broadly broken into three parts:

- procedure selection logic,
- procedure enable logic,
- procedure execution logic.

Procedure Selection

Which procedure is selected for execution is indicated by a set of flags, or for a few simple procedures a key state. See the List of Procedures.

• FRS

The keypresses for Clear-General and DP-Shifting are encoded into 2 bits and recorded in the FRS1 & FRS2 flags. These procedures share the requirement of processing a set of registers. See the according procedure section.

FRS2,1	Procedure
00	–
01	Clear-General
10	DP-Shift-Right
11	DP-Shift-Left

• FPa1,a2,b1,b2,N

Selecting the procedure for the more-complex arithmetic functions involves recognising certain key sequences of up to 3 steps.

- FPa1,a2 record the most-recent operation key pressed.
- FPb1,b2 record the preceding operation key pressed.
- FPN records whether a number preceded the most recent operation keypress (between the a & b operation keypresses). The number may be a keyboard entry or a register display.

These flags are decoded to a set of procedure-select signals. The signals names are framed in the keypress order with most recent leftmost, e.g.:

PEQ•N•PDVp	DV was pressed, then a Number, then EQ.
PEQ•(N~)•PEQp	EQ was pressed twice in a row.

'EQ' means any of the 5 Equals keys.

Procedure Enable & Execution

• Idle vs. Execution

The idle state of the calculator is distinguished from execution by the signal EXEC. EXEC is asserted initially by the flag FEX after a key is pressed. FEX synchronises the keypress to the number cycle and ensures that EXEC is asserted for an integral count of number cycles. FEX (and EXEC) remain asserted while a key is pressed.

The Simple procedures are short enough to be completed within the period of FEX. EXEC remains asserted while FEX is asserted.

For the Programmed operations, the QM sequence counter is invoked to leave its idle-0 state. QM being non-0 sustains assertion of EXEC till the procedure is complete (QM returns to 0) if it takes longer than the keypress.

• Procedure enable — Procedure execution is initiated by a keypress. The Set-Gate input of FEX is normally 1 (disabled) while the Reset-Gate input is normally 0 (enabled), so FEX is not asserted. The pressing of a key flips the state of both gate inputs, enabling FEX to assert on the 0-edge of the next ØN. This situation persists while the key is pressed.

When the key is released the 5µF capacitor at FEX.SG~ begins to charge, eventually disabling the Set-Gate input (1). The reverse-biased BE junction of T3.13 acting as a zener presents a higher threshold to the inverter T3.1, thus there is a further delay before the Reset-Gate input flips back to 0 (enable). Once it does so, FEX deasserts on the 1-edge of the next ØN.

• State transition — State transitions or steps in a procedure are triggerred by the 0-edge of ØN. The ØN period then, comes at the end of a state cycle, and state cycles are synchronous to the number cycle. A register operation is an atomic event at this level.

Numerous actions are triggered by or in consequence of the 0-edge of ØN: the EXEC flag, the procedure selection flags, the QM & QB Counters, the F31 & F32 phase flags, the Tracking Counter, etc.

Program Sequence Counters

The QM and QB Counters control sequencing of the Programmed procedures.

As counters, they follow a fixed sequence. The decoded output steps of these counters are gated with the procedure-select and other signals to determine the actions performed at a given step. This contrasts with a more-canonical state-machine design with next-state logic producing a variable sequence of states.

• QM - Main — QM is a 4-bit binary counter providing 16 steps or states. The 16 states are decoded first from binary to two 1-of-4 sequences, a Major (QMA::D), and a minor (QMa::d). These major-minor sequences are then further decoded as necessary to arrive at a unique 1-of-16 state (QM00::15), an example being the QM=0 AND gate at the gating input of [16.4PL].

QM is the main control for arithmetic procedures. QM normally sits in its idle 0 state. When EXEC asserts for a programmed procedure, QM is incremented to 1 [16.2PL]. QM≠0 enables QM to be incremented on each number cycle [16.4PL].

The QM sequence does not loop - for main programs the sequence is traversed only once. The sequence ends when QM rolls over to 0.

• QB - Subroutine — QB is a 3-bit binary counter. QB is decoded in a Major-minor 1-of-n scheme similar to that of QM. QB is used to execute a sub-program in the course of the main program. There are 3 sub-programs: BMP, BDV & BSR.

• QB - BMP/BDV — The BMP & BDV subroutines are short enough that they do not require the 3rd QB bit. Only the QBa,b,c,d signals are used, providing 4 steps,

referred to as QB0::3.

There is however, a 5th state that occurs when the counter is held in reset and QBa is suppressed from asserting. This state will be referred to as QBN. QBN does not have an explicit signal, it is distinguished by none of the QB-minor signals asserting.

- QB - BSR — The BSR subroutine requires 8 steps, the 3 bits of QB are decoded to 8 steps QB0::7. The QBN state is not used.
- QB - ancillary use — QB is also used to generate a register-selection sequence during the Clear-General and DP-Shift procedures.
- QB activity — During MP/DV/SR, one of the operation signals PMP,PDV,PSR is asserted. At step QM6 in the main sequence, gating enables one of the sub-procedure signals BMP,BDV,BSR to assert [16.10,11,12Nv].

  This latter assertion has two effects:

  - QM is halted by disabling the ØN pulses to [16.4PL].
  - QBEN is asserted [8.7Nv]. This enables QB, though it may not yet be active if being held in the QBN state. When activated, the counter is released from reset and increments on each number cycle.

  Incrementing of QB can be stopped at some step by the assertion of QBLP. Number cycles continue to be performed so execution loops at that step till QBLP is deasserted.

  The QBN state is used in the BMP/BDV subroutines for preparatory and/or cleanup activity [8.9NG,8.10NG]. The QB sequence may be terminated early by taking it to the QBN state.
- QB seq. looping — During MP/DV/SR, the QB sequence loops (repeatedly passes through QB0) until flag F32 deasserts to produce a pulse [16.3PL] to increment QM from 6 to 7. QM≠6 deasserts the sub-procedure signals, deasserting QBEN and resetting and disabling QB, and re-enables ØN incrementing of QM, allowing the main program to proceed.

Programming Logic

The micro-code for the programs is implemented with diodes forming NORN and OR gates. This can be construed into matrices as the presentation in the logic schematic illuminates to some degree, though there is no such matrix readily visible in the implementation.

Simple Procedures

Idle

• Display

During idle the default state of register selection is in effect [19.14Nd], this being the register indicated by the Current Register flags FGBA.

The default state of core memory read and write is also in effect, with GX asserted [logic.14/32A-2]. GX being asserted, during each digit period, the X Decade is loaded from core and restored back to core. The contents of X are presented to the Nixie numeral decoder for display.

The default counting of the ØD counter is also in effect, counting up.

Display Select

Display-Select is executed to set which register is displayed in the Nixies.

• Start

The register-select keypress is encoded to KGG,BB,AA (collectively KGBA) and KMEM asserted [logic.6]. KMEM assertion triggers a FEX/EXEC cycle.

• Load FGBA

EXEC assertion generates a pulse to clear FGBA [19.1PL]. With FEX asserted, the 0-edge of ØN pulses after the clear generate pulse(s) to set the FGBA bits from KGBA.

Clear General

Pressing the general C key clears the registers A, B & T. Accomplishing this requires stepping through the digits of the 3 registers in sequence while clearing them, as well as clearing the sign flags for those registers.

• Select

When EXEC is asserted, FRS is loaded with 1, selecting the Clear-General procedure for execution. FRS≠0 asserts QBEN.

• Sequence

QBEN being asserted releases QB from reset, enabling it to increment on every number cycle. QB proceeds through its 8 steps. As it does so, its sequence signals QBxx are encoded into a register-selection code:

QB	G~GBA	Register
Aa	111	-
Ab	110	B
Ac	101	A
Ad	011	T
Ba	111	-
Bb	110	B
Bc	101	A
Bd	011	T

Thus, during the number cycles of the QB sequence these registers are

selected for access.

• Clearing — FRS=1 also asserts GCL~.

GCL~ asserts XYCL~, this holds X and Y in reset, so during the QB number-cycles the digits of the selected register are written back to core as 0.

GCL~ also asserts reset to the sign flags, so during the QB sequence, as the sign flag of the selected register is accessed it is cleared.

The QM Sequence Counter is reset in sync with ØN [16.1PL].

• End — As QB returns to 0, the 0-edge of QBB resets FRS=0, terminating the Clear-General procedure.

DP Shift

Shifting the decimal point position requires the DP Counter be incremented or decremented, but as well, all 7 registers must be shifted up or down.

• DP Counter — The pressed shift-left or shift-right key enables the DP counter to count in the according direction, left:up, right:down. On the assert edge of EXEC, the DP counter is clocked.

• Shift sequencing — The shifting of the registers is sequenced using the same facilities as the Clear-General procedure: the FRS flags and the QB Sequence Counter. FRS is set to a state indicating shift-left (FRS=3) or shift-right (FRS=2). The sequence of register selection differs from Clear-General:

QB	G~GBA	Register
Aa	111	-
Ab	110	B
Ac	101	A
Ad	100	IV
Ba	011	T
Bb	010	II
Bc	001	III
Bd	000	I

The FRS state asserts GSL~ for shift-left and GSRz~ for shift-right. For shift-right, GSRz~ sets the ØD Counter to count down rather than up.

• Shifting — The shifting is performed as described in the Registers section using the FGZ flag and X,Z leap-frogging.

Number Entry

Due to the fixed (but manually-set) decimal point, entry of a number must select between two sub-procedures when entering a numeral: one for integral digits and one for fractional digits.

• Start — The pressed numeral key is encoded into BCD signals KN1,2,4,8. KNM is asserted. KNM enables flag FEX to set at the beginning of an upcoming number cycle. KNM also enables FEEN (Flag Entry Enable)

to be set by EXEC. With FEEN set, flag FESH is released from reset to respond to it's assorted Set inputs. FESH will be used to control the shifting up of integral digits and placement of the new numeral in the correct digit position.

- Register — A new number is always entered into Register B. If the current register FGBA is not B, when EXEC asserts, flag FECL is set 1 for one number cycle. FECL asserts GGBA=B and loads FGBA=(GGBA=B). GCL is asserted to clear RB during the number cycle.
- — ? How is GCL distinguished in time from entry action?
- FKP flag — The FKP flag distinguishes entry of integral versus fractional digits. Pressing of the DP key results in assertion of FKP. FKP is cleared when some non-number key is pressed.

- **Integral entry** — If the DP key has not been pressed, the existing digits to the left of the DP position (>DP) must be shifted up and the new numeral entered at digit DP+1.
- New numeral — When EXEC is asserted, the Z Decade is loaded with the new numeral, though it is not yet in core.
- Shifting — As the digit scanning proceeds up, X is being loaded from and written to core. With FKP=0, when ØD increments to DP+1, the 0-edge of DP=ØD triggers FESH=1 [FESH.S2].

  FESH assertion enables the shifting facility described in the Registers commentary. The first numeral written to core is the new numeral in Z and the shift leap-frogging proceeds for the remainder of the number cycle.

  FESH is cleared at the end of the number cycle via clearing of FEEN [FEEN.R1,FESH.R4].

- **Fractional entry** — For fractional digits, there is no need for shifting of digits, the new numeral must merely be appended to the end of the existing fractional digits. The Tracking Counter will be used to maintain an indication of where this end is.
- Aligning TC — TC must start at the DP position, that is, TC must be set equal to DP. When FEX is asserted due to pressing of the DP key, and FKP now 1 but FEEN=0 (not a numeral key), if TC≠DP a burst of ØM pulses is produced to clock TC [3.2PL], the burst being terminated when TC=DP.
- Adjusting TC — The default counting state of TC is down [1.9Nd]. As fractional digits are entered, the 0-edge of FEEN triggers 3.1PL to decrement TC, and so track the end of the fractional digits.
- Entering new — As with integral entry, the keypress numeral is loaded into Z. As the digit scanning proceeds up, with FKP~=0, when TC=ØD, FESH is asserted [FESH.S1], enabling the shift facility, and the new numeral in Z will be

written to memory.

In contrast to integral entry, FESH is promptly cleared at the end of the same digit period [FESH.R1], disabling the shift facility and the shifting up of subsequent digits. The clearing of FESH also clears FEEN.

- ? Why are there inputs FESH.S3 & FESH.S4 ?

Negate

The Negate procedure is initiated by pressing of the Negative key. The objective is simply to flip the sign flag of the current register.

- Flipping — KNEG is asserted and a FEX/EXEC cycle initiated. With KNEG~=0, the 0-edge of EXEC~ triggers FCS=1. The sign flag of the current register is then flipped as described in the Sign Flags commentary.

  FCS is cleared via the gated-reset input at the end of the number cycle. Though KNEG~ may continue to enable the SG~ input there are no further trigger edges from EXEC~.

Programs

Nomenclature:

=	Action performed by edge or pulse trigger.
<=	Action performed by number cycle.
±	Signed arithmetic cycle.
F	Current register FGBA = { A, B, T, 1, 2, 3, 4 }

By example:

F = T	FGBA is loaded with code for Register T.
RB <= R(F)	Register referenced by FGBA is copied to Register B.
R(F) <+= RT	Register T is added to register referenced by FGBA.
RT<<	Register T is shifted left one digit.
F32<0	0-edge of F32 triggers action.

Notes:

• The program tables may not represent the finer timing relationships of triggerred actions.

• In program steps performing an arithmetic cycle there are occasions where the addend register is the same as the sum register. This would suggest the register will be added to itself and its value doubled, however a doubling does not in fact occur. During the digit cycles as X & Y are loaded from the same register, the 2nd read from core reads 0 because the core has been cleared and not yet restored. Thus, rather than R+R the operation is effectively R+0 and leaves the register value unchanged.

• See also the IME-86S subroutine emulations (simple Python program).

Program: AddA

	Select: PEQ•N•PEQp
QM	**Action**
1	RB <= R(F)
2	F = KeypressReg (via GFK~)
7	R(F) <±= RB
9	F = B
10	RA <= R(F)
11	RB <= 0

Add Addend. Add a value to the EQ-keypress register. For the EQ-general key, the target register is RT.

1: The addend in the current register is copied to RB. If a new number was just entered and the current register is RB, this is effectively a null operation.
2,7: RB is added to the target register indicated by the keypress.
10: The addend now in RB is saved in RA.
9,11: RB is cleared and the current register set to RB.

Program: AddC

	Select: PEQ•(N~)•PEQp
QM	**Action**
1	RB <= R(F)
2	F = KeypressReg (via GFK~)
7	R(F) <±= RB
10	F = T

Add Complete. A 2nd sequential keypress of EQ key(s) adds the current register to the EQ-keypress register and the current register is set to T.

Program: MP/DV Queue

	Select: PMD•N
QM	**Action**
9	RA <= RB
10	F = B

As the design uses infix key-sequencing, pressing MP or DV must be recorded so a later EQ keypress knows what is to be done. The recording (queuing) is done in the FPb flags. Pressing the MP/DV keys also does some operand preparation:

9: The entered number N is copied to RA, to be the multiplier or divisor.
10: The current register is set to RB in readiness for entry of the next operand.

Program: Multiply

	Select: PEQ•N•PMPp
QM	**Action**
1	RB <= R(F)
2	F = KeyboardReg (via GFK~)
3	RT <= 0
6	BMP: RT <= RB•RA
7	R(F) <±= RT
9	R4 <+= RA :if ModeS
10	F = T

With MP queued, pressing an EQ key executes this program to perform the multiplication.

1: The current register is copied to RB to be the multiplicand.
2: The current register is set to the EQ-keypress register.
3,6: T is cleared and the multiply performed.
7: If an EQ-Mx key is pressed the product in RT is added to the Mx register. If the EQ-general key is pressed R(F) will be RT, but RT is not altered (see no-double comment in Program Notes).
9: If the item-count mode is enabled, the multiplier in RA is added to R4.
10: T becomes the current register.

Program: Divide

	Select: PEQ•N•PDVp
QM	**Action**
1	RB <= R(F)
2	F = KeyboardReg (via GFK~)
3	RT <= 0
6	BDV: RT <= RB/RA
7	R(F) <±= RT
10	F = T

The program for EQ for a queued DV is nearly identical to that of MP. There is no item-count.

Program: MP/DV Chain

	Select: PMD•N•PMPp, PMD•N•PDVp
QM	**Action**
1	RB <= R(F)
2	F = T
3	RT <= 0
6	BMP / BDV
9	RA <= RB
10	F = B
11	RB <= 0

?TODO

Program: Raise Exponent 1,2,C

	Select: PMM•(N~)•(Nn~)	**Raise Exponent 1**
QM	**Action**	
	FNn = 0	
1	F = A	
2	RB <= R(F)	
3	RT <= 0	
5	R4 <= 0	
6	BMP: RT <= RA•RB	
7	R4 <+= 1	
9	R4 <+= 1	
11	F = 4	
	FNn = 1	

	Select: PMM•(N~)•Nn	**Raise Exponent 2**
QM	**Action**	
1	F = T	
2	RB <= R(F)	
3	RT <= 0	
6	BMP: RT <= RA•RB	
9	R4 <+= 1	
11	F = 4	

	Select: PEQ•(N~)•Nn	**Raise Complete**
QM	**Action**	
1	RB <= R(F)	
2	F = KeyboardReg (via GFK~)	
5	R4 <= 0	
7	R(F) <±= RT	
9	F = T	

The Raise programs are executed in combination to perform the Exponentiation operation.

- FNn — Flag FNn is used to initiate the sequence of multiplications by distinguishing between the first (FNn=0) and subsequent multiplications (FNn=1).

• Prior — The Raise process actually begins when the first press of MP prepared for a multiplication (PMD•N). This left the base in A and cleared FNn=0.

• 1 - First — The 2nd sequential keypress of the MP key asserts PMM•(N~). Note PMP is also asserted. With FNn=0, initialisation actions and the first multiplication are performed:

- 3,5: RT & R4 are cleared. The result will build in RT. R4 will be the exponent indicator.
- 6: At QM6, the first multiply is performed, leaving the square of the base in RT.
- 7,9: R4 is incremented to 2, indicating the squaring.
- 11: The 0-edge of QMC (end of QM11) sets FNn=1, indicating the Raise operation has been initialised.

• 2 - Subsequent — On following presses of the MP key with FNn=1, a modified program is executed:

- 2: The result-so-far in RT is copied to RB to be the multiplicand. The original base remains in RA as the multiplier.
- 6: Another multiply is performed.
- 9: The exponent in R4 is incremented.

• C - Completion — The exponent in R4 is copied to RB. The result in RT is added or copied to the keypress register and the current register set to RT.

PMP is asserted during this program but BMP invokation at QM6 is suppressed [16.13.Nd].

The press of EQ or a non-MP operation key deasserts PMM•(N~), and FNn is cleared 0.

Program: Square Root

	Select: PSR•(N)
QM	**Action**
1	R4 <= R(F)
2	RB <= 0
3	RT <= 0
5	RA <= 0
6	BSR: RT <= SQRT(R4)
9	F = T

The current register is copied to R4, other required registers cleared and the square root subroutine executed. On return, the root is in RT. The current register is set to RT.

Program: ?PostRoot

	Select: PEQ•N•PRRp
QM	**Action**
1	RB <= R(F)
2	F = KeyboardReg (via GFK~)
3	RT <= 0
7	R(F) <±= RB
9	F = B
10	RA <= R(F)
11	RB <= 0

Why?

Subroutine: Multiply

	Select: BMP	
	RA: multiplier	
	RB: multiplicand	
	RT: product	
QB	**Condition**	**Action**
QBEN		TC = 0, F31 = 0, F32 = 0
*	if F32=0 • DP=ØD•ØD15	F31 = 1, F32 = 1
*	if TC ROLL	F31 = 0
*	if F31=0 • F32=1	QB = N
*	if F31=0 • F32=1 • TC=DP	F32 = 0
0		Z = RA[0]
	if TC=DP	F31 = 1
1	loop while Z>0	RT <+= RB, – –Z
	if Z=0 • F31=1 • F32=0	RT[0] <+= 5 ?rounding
2	if F32=0	RT>>
	if F32=1	RB<<
3		RA>>, ++TC
	if F31=1 @EON	F32 = 1
N		RB<<, ++TC
return	F32<–0	++QM - continue Main

The multiply subroutine is executed at step 6 of the main sequence if PMP is asserted.

- Operands — RA contains the multiplier, RB contains the multiplicand, the product will accumulate in RT.
- QB looping — The QB sequence will loop, but the MSB of QB will be ignored, so it will be seen as (two iterations of) a 4-step loop. In each iteration, the LSD of RA (multiplier) will be processed and RA rotated right.
- Start — The 1-edge of QBEN clears TC to 0 [8.8PP]. TC will be used a loop counter to count 16 iterations through the QB loop. F31 & F32 are released from reset.
- QB0 - Get LSD — The LSD of RA (multiplier) is loaded into Z. When TC=DP, F31 is set 1. F31 distinguishes the processing of fractional digits versus integral digits.
- QB1 - Accumulate — QB is halted and the step loops. On each iteration (number cycle), RB (multiplicand) is added to RT (product) and Z decremented. When Z becomes 0, this sub-looping stops and the QB loop resumes.

- QB2 - Shift — If fractional digits are being processed (F31=0), the product is being reduced in magnitude, so RT is shifted down. If integer digits are being processed (F31=1), the product is being increased, so the multiplicand RB is shifted up so it will accumulate in higher digits of the product.
- QB3 - Shift RA — The multiplier is rotated right so the next higher digit becomes the LSD. The loop counter TC is incremented. If TC<=15, the QB loop repeats.
- QB loop exit — When 16 iterations are complete, TC rolls from 15 to 0. The 0-edge of TC8 clears F31=0. With F32=1 and F31=0, the QB sequence is terminated by inhibiting and resetting QB [8.9NG].
- DP re-align — However, QM is not yet re-enabled as F32 is still 1. If RB has been shifted up, it is no longer aligned to the DP position. In the following number cycles, RB continues to be rotated up (with wrap to the LSD), and TC incremented until TC=DP. This returns RB to its original DP alignment position. TC=DP terminates this interim activity by clearing F32=0.
- Subroutine return — The 0-edge of F32 kicks QM from 6 to 7, completing the subroutine and re-enabling the Main program.

Subroutine: Divide

	Select: BDV	
	RA: dividend	
	RB: divisor	
	RT: quotient	
QB	**Condition**	**Action**
QBEN		TC = 0, F31 = 0, F32 = 0
N	loop while F31=0	
	if F32=0	RB<<
	if F32=0 • ØD15 • X≠0	F31 = 1
	if F32=0 @EON	++TC
0		
1	loop while FCS=0	RA <–= RB, if FYC=1 (no borrow): ++Z
	when FCS=1	RA <+= RB
2		RT<<, RT[0] = Z
	if F32=1 • TC=DP	F31 = 0, QB = N
	if ØD15 • DP=ØD	F32 = 0
3	if F32=0	RB>>z
	if F32=1	RA<<
	if F31=F32	++TC
	if F31≠F32	– –TC
	if F31=1 • TC>15	F32 = 1
N	loop while F31=0	
	if F32=1	RA>>z
	if F32=1 @EON	– –TC
	if TC<0	F32 = 0
return	F32<–0	++QM - continue Main

• Phases

The division algorithm is better understood as proceeding through 4 phases as elaborated in the table following. These phases are defined by the F31 & F32 flags, and denoted here as a bit pair, e.g.:

Phase 01 == F31=0, F32=1

For the 1st and 4th of these phases, QB - though enabled by QBEN - is held in reset by F31=0 [8.10NG], execution thus loops at QBN. These are a preparation and a cleanup phase. The quotient is derived in the

2nd and 3rd phases. The point in distinguishing the 2nd & 3rd phases is which operand will be shifted, due to the fixed DP position of the design.

Phase 00		Phase 10		Phase 11		Phase 01	
QB	TC up	**QB**	TC down	**QB**	TC up	**QB**	TC down
N loop	RB<<, ++TC till MSB≠0, then 1=>F31.	0	-	0	-	N loop	RA>>, – –TC till TC rolls from 0.
		1 loop	RA ±= RB Resolve new digit in Z.	1 loop	RA ±= RB Resolve new digit in Z.		
		2	RT<< + Z	2	RT<< + Z		
		3	RB>>, – –TC	3	RA<<, ++TC		

• Phase 00 — The dividend is shifted up to eliminate MSD zeroes. Execution loops at this step shifting the divisor up until a non-zero digit entering the MSD sets F31=1, finally activating QB. For each shift, TC is incremented.

• Phase 10 & 11 — When F31=1, QB can proceed through its sequence. In these phases the QB sequence will loop, each loop producing a digit of the quotient. The sequence goes through 0 but with F31=1 no operation is performed.

The transition from phase 10 to 11 is marked by the setting of F32=1. This occurs when the decrementing TC rolls from 0 to 15 [F32.SK.1].

• Phase 10 — TC counted up from 0 while divisor RB was shifted up in phase 00. During phase 10, TC counts back down to 0 while shifting RB down, returning RB to its original position.

• Phase 11 — TC now counts up again and as new quotient digits are produced, dividend RA is shifted up. When TC=DP all that can be produced has been and the QB sequence loop is terminated by setting F31=0, transitioning to phase 01.

This transition occurs at the end of step QB2 [F31.RK.2]. F31=0 resets QB and takes it to the QBN state, so the final QB3 step does not occur.

• QB1 - Resolve — QBLP is asserted, so execution loops at this step. During each number-cycle iteration, the divisor is subtracted from the dividend by assertion of GASUB & GAAB [10.10NG]. If there is no borrow out of the subtraction (FYC=1), Z is incremented [8.2PL].

A borrow (FYC=0) indicates the divisor went negative. The new quotient digit has been resolved but the overdraft must be corrected. FYC=0 sets FCS=1 which holds the borrow indication over the next number-cycle. With FCS=1, GAAB is asserted but GASUB is not [10.9NG], and so the divisor is added back to the dividend.

FCS=1 also deasserted QBLP, so QB proceeds to step 2 after the add-back is complete.

• QB2 - Quotient — Shift-left is enabled for the quotient in RT [10.6NG]. In the shift X-Z

leap-frog, Z is the first digit written to core, so the newly-resolved digit in Z is injected at the LSD of the quotient. Z is cleared 0 at the end of shift number cycles [3.3PL].

- QB3 - Shift op — The divisor RB is shifted down or the dividend RA is shifted up depending on the phase.
- Phase 01 — With F32=1, rather than shifting RB up as in phase 00, the dividend RA is shifted down to bring it to back down to DP alignment. TC is switched again to counting down and the phase is complete when TC rolls over from 0 to 15. The rollover clears F32=0 [F32.RK.1].
- Subroutine return — The 0-edge of F32 kicks QM from 6 to 7, completing the subroutine and re-enabling the Main program.

Subroutine: Square Root

	Select: BSR	
	R4: N	
	RA: reducing N dividend	
	RB: 2-increment divisor	
	RT: root	
QB	**Condition**	**Action**
QBEN		TC = 0, F31 = 1, F32 = 0
*	if DP=15:	F32 = 1
*	if DP=ØD•ØD=ODD:	F31 = 0
*	if TC ROLL:	F32 = 0
0		RA<<
1		R4<<, RA[0] = R4[15]
2	if F31=0:	RA<<
3	if F31=0:	R4<<, RA[0] = R4[15]
4		RB<<, RB[0] = 1, Fsr = 0
5	loop while FSC=0:	
	if Fsr=0:	RA <–= RB, Fsr = 1
	if Fsr=1:	RB <+= 2, ++Z, Fsr = 0
	if FCS = 1:	RA <+= B
6		RT<<, RT[0]=Z
	if F32=1:	++TC
7		RB <–= 1
	if TC=DP:	F32 = 1
		++TC
		F31 = 0
0 return	F32<0	++QM - continue Main

• QB0::3 - Get pair — Digits of N are processed in pairs from the MSD of N down. N is in R4, the pair processing will be done from RA. To transfer digits from R4 to RA, RA is shifted up first (QB0,2) leaving the LSD empty. This is followed by shifting up R4 (QB1,3). As this number cycle ends, Z has been loaded with the MSD to be rotated down to the LSD. At this point (ØN), the single register-code-bit distinguishing R4 from RA is deasserted to 1 [9.2NG], switching from R4 to RA [9.6NG], and Z is written to the LSD of RA.

• F31 — The pairing boundary must be aligned to the DP, that is, the DP must not

be in the middle of a pair. If the DPC value is even then the DP and register pairs are 'out-of-phase'. To bring them into phase the first pair transfer will be just one digit from R4.

F31=0 will indicate the pairing is in-phase. At the beginning of the process, F31 is initialised to 1. The DP is then tested and if odd (in-phase) F31 is cleared 0. At steps QB2,3, if F31=1 the shift-and-transfer are suppressed [9.10Nv]. Later, in step QB7, F31 is set 0, so in all subsequent sequence iterations two digits are transferred.

- QB4 - Shift op — RB is shifted up and 1 added [12.4NG]. The +1 is achieved by incrementing Z to 1 at the beginning of the number cycle [12.3Nv], which will then be loaded into the RB LSD at the beginning of the Shift-Left.

- QB5 - Resolve — This step loops to resolve a new digit of the root - QBLP is asserted. In each iteration RB will be subtracted from RA [12.7NG]. If the result is positive (FCS=0), 2 is added to RB, Z incremented and execution loops [12.9NG]. This can require two number cycles for each loop iteration. Flag Fsr alternates in each iteration to distinguish the two number cycles, Fsr=0: subtract, Fsr=1: add & increment.

  If RA went negative in the subtraction [FCS=1], the loop is terminated [12.10NG]. A final number cycle in QB5 is executed to correct the overdraft by adding RB back to RA.

- QB6 - Shift root — The building root RT is shifted up with the new digit in Z injected into the LSD [9.4NG].

  TC is also incremented if processing the integer portion of the radicand (F32=1). The produces a double-increment of TC for integral digits, accounting for the halved magnitude of the root relative to the radicand.

- QB7 — 1 is subtracted from RB. TC is incremented. If TC is passing the DP position (processing is moving to the integral digits), F32 is set 1.

- Subroutine return — When TC rolls over from 15 to 0, due to the increment in either QB6 or QB7, F32 is cleared 0. The 0-edge of F32 kicks QM from 6 to 7, completing the subroutine and re-enabling the Main program.

- (?) — The QBa input to 12.3Nv appears superfluous.

Document Log

- 2024 Sep Initial writing.

—— EOD ——