IME 84 Calculator
Theory of Operation

bhilpert / madrona.ca

2025 Oct

Rendition: 2026-Mar-29

Contents

Introduction

The IME-84 is a desktop calculator from the early 1960s, implemented in discrete-component electronics. A schematic - with interpretation to logic and block & timing diagrams - has been produced based on three sources:

- A schematic from a reverse-engineering effort in 1986/7 by Klemens Krause. This schematic is limited to a representation of the discrete electronics. A few errors were uncovered in the process of creating the new logic schematic.
- PCB photos and examination of IME-84 unit #1.8400903 by Jef Ongena.
- PCB photos of an IME-84 unit obtained via the DoPECC website. [https://dopecc.net].

This commentary has been produced from analysis of the new logic schematic.

[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-84 is a digit-serial design. At a functional architectural level, data processing involves 4 registers, each holding a 16-digit 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 4 registers are implemented in a magnetic core memory.

The control and sequencing of the higher-level arithmetic and entry operations is implemented with a monostable-based sequencer, assorted flags, counters and ad-hoc logic.

Relation to Mechanical Calculators & the IME 86

As one of the very earliest electronic desktop calculators, the IME-84 took many design cues from the mechanical calculators of the day. Beyond the mentioned counter-based arithmetic the most noticeable inheritance is in aspects of the user interface.

Mechanical calculators had significant shortcomings around matters such as decimal point management and handling of negative numbers. Additional complexity would have been required to deal with these in a more thorough manner, in already mechanically complex machines. While the IME-84 made some improvements on these matters, it retained many of the shortcomings.

The IME-84 preceded the IME-86 in the market by around 2 years. The two models appear quite similar as seen from the exterior and in the internal construction. The 86 however, is not simply a few minor improvements on the 84. The 86 would take significant steps away from the mechanical heritage:

- Keyboard — The 84 mimics the keyboard interface and register structure of mechanical calculators, as seen for instance in the two "=" keys - one for multiplication, one for division - and effectively a third in the "*" key. The "*" is a direct inheritance from mechanical adding machines where it meant "Print Total", not the modern computing-language use as a symbol for multiplication.

The 86 steps away from these anachronisms, towards a simpler, more flexible and more intuitive keyboard interface.

• Decimal Points — Decimal point management in the 84 was limited. There are two DP position counters: one for the keyboard entry and one for the result. Values in registers could thus have different DP positions but inconsistencies can arise when transferring values between registers.

DP management in the 86 is much improved. The DP position is the same for all registers, the position may be moved by the user in the course of operations, with consistency being maintained between all the registers.

• Signs — In the 84, there is a single sign flag - applicable only to the result of an operation. The internal 10's-complement arithmetic is exposed to the user at times and the proper presentation of a negative value is not performed until the "*" key is pressed. Negative values will not be dealt with properly in multiplication and division.

In the 86, each register has a sign flag and proper handling and presentation of negative values is maintained through all operations.

The electronic implementation of logic is largely the same in the 84 and 86. Architecturally though, there are again significant differences between them, some notable ones being:

• Digit Cycle — In the 84, the digit cycle is variable, being extended for 10 pulses of the master clock for arithmetic counting.

In the 86, the digit cycle is fixed, the arithmetic counting period is always included in the digit cycle.

• Core Power — In the 84, the core drive supply is a typical linear dropping regulator with a resistive divider for V–c.

In the 86, the core drive supply comes directly from the main logic supply with V–c derived from the drive currents with a charge pump and shunt regulator.

• Core Drive — Both models use a 2-wire arrangement for the core, with drive and sense shared on the wires of one axis.

The 84 employs a 2D drive scheme: 1/2-current XY addressing is used for write, but unit-current addressing is used for read.

The 86 employs a 2-1/2D drive scheme, 1/2-current XY addressing is used for both read and write.

• Core Access — In the 84, each of the 4 registers has an associated decade counter, and the registers are all being scanned simultaneously with digits continually transferred between core and the register's decade.

In the 86, the 7 registers are only scanned and loaded into a decade when it is necessary to do so, with just 3 decades shared between the registers on a need basis.

• Execution

The procedure execution facilities are significantly different.

The 84 has an odd unsynchronised state machine built with pulsers and monostables for some procedures, and multiple counters and ad-hoc logic for others.

The 86 is more regularised with two sequencing counters controlling most procedures. Some carry-over similarity can be seen in the PQ Counter of the 84 and the QB Counter of the 86: both sequence the primary loops of multiplication and division in 4 states.

Electronic Implementation of Logic

The IME-84 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 +4.5V 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.20::22].

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-84 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.22].

Pulsers & Asynchronism

Producing a fully synchronous design can be costly, with doubled flip-flops and more gates. In avoiding this, the IME-84 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 'soak up' glitches by maintaining assertion for a brief period, so a glitch will not appear as a distinguished edge.

- To delay an edge briefly to bridge over a period in which glitches may be occurring, with the delayed edge then triggering some action.
- To produce a brief pulse to quickly perform some action.
- To disambiguate actions triggered 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.21,22].

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+4.5. This provides some headroom before increased Vf drop would leave a transistor conducting.

Simple silicon switching diodes such as 1N914 or 1N4148 may work for most or many gate diodes, however the higher Vf may reduce the engineered tolerance margins.

Schottky diodes with their low Vf similar to Ge are the other option. The BAT46 is one suggestion, and has been used in repairs of both an IME-84 & IME-86S.

• Transistors

The following standard Si transistors were used for replacements in repairs of an IME-84 and IME-86S:

• Logic transistors:	2N3906
• Core drivers:	2N2907
• Nixie anode drivers:	MPSA92
• Nixie cathode drivers (IME84):	MPSA92
• Nixie cathode drivers (IME86):	MPSA42 (suggested, not tried)

Timing & Data Elements

Timing

A timing diagram for the major clock signals is presented on [logic.18].

- Master clock: Timing begins with an astable flip-flop operating at ~ 20 KHz generating signal ØM.
- Digit period: By default, the master clock triggers a 2-bit binary ripple counter: the Cycle Counter (ØCC) [FØC1,2]. The 4-stage cycle produced by this counter constitutes a digit cycle or period (delimited by the 0-edge of FØC2=ØD) during which digits of numbers in the main registers are processed.

  During the ØCC stages several timing signals are produced:

  - ØDp: A pulse to clear the Register Decades prior to core sense.
  - ØRp: A digit of the each of the 4 main registers is retrieved from core memory into a Decade counter/latch.
  - ØWp: The digit values - perhaps modified - are written back / restored to core.
  - ØX: A selected register digit is enabled for display in one of the Nixie tubes.
- Arithmetic: A digit period may be extended, for performing arithmetic on digit values. The extension is 10 cycles of the master clock inserted in the middle of the digit period, between the read and write.

  A 0-edge on A_START~ sets flag ØA=1. This halts ØCC, and enables counting of 4-bit decade counter ØAC. ØAC counts out 10 cycles of the master clock - producing signal ØA10 - and then clears flag ØA, inhibiting itself, and restarting the digit cycle.
- Digit-of-number: The 0-edge of FØC2 represents the end of one digit and the beginning of another. This edge clocks the ØD Counter [FØD1,2,4,8], a 4-bit binary ripple counter. The 16 states of this counter define the digits of a number, from ØD0 (LSD) to ØD15 (MSD).
- 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.
- Number cycle: Flip-flop [FØN] is 1-set at the beginning of a number cycle [25.1Nd]. This pauses the clocking of ØDC for one digit period (ØN).

  A full number cycle is thus actually 17 digit periods in time. ØDC=0 during ØN regardless of counting direction.

Digit Memory (Core)

The digits of the numbers held by the 4 registers are stored in a small core memory.

• Physical	The core memory is physically organised as a 2D planar array of 16 • (4 • 4).
• Logical	Logically, the core memory is a 3D array of 4 registers, by 16 digits per register, by 4 bits per digit. The 16 wires on the digits/register axis will be referred to as digit-lines, the 4 wires on the bits/digit axis as the register-bit-lines.
• Encoding	The digits are encoded in standard 4-bit 1248 BCD code.
• Electrical	Electrically, the form is what was referred to in the era of core memory as "2D 2-wire". All the cores on one axis will be accessed during a read cycle with a linear address applied on the other axis. The 16•16 physical array is electrically being treated as 16 words of 16 bits, the 16 bits of a word then being logically treated as 4-bit digits of 4 registers.
• Drive policy	Core requires bidirectional current drive through the cores, one direction for clear-read, one for set-write. For the 2D form, the core drive policy differs from the usual X/Y 1/2-current drive of more-common 3D core implementations.

In the IME-84, during clear-read, all 16 cores on a digit-line are to be cleared-read simultaneously - to clear a digit of all 4 registers and load the Decade(s) of all 4 registers. As such, there is no need to involve register-bit-line selection during the clear-read. The clear-read can be accomplished by driving a full unit-current pulse through the digit-line rather than 1/2 currents on each axis (a unit-current being the current level required to flip the magnetic state of a core).

For writing, there is a need to distinguish register-bit-lines, as cores on different register-bit-lines will of course typically have different intended states. A digit-line will now carry a 1/2-current (in the opposite direction to clear-read) while the register-bit-lines carry a coincident 1/2-current for the cores which are to be set 1. [Core Cycle Scoping TOP.28]

This drive policy is summarised in the following table:

Core Drive Relations			
	Digit-Line	**Register-Bit-Line**	**Pulse**
Clear-read	+1	sense	ØRp
Write 0	–1/2	0	ØWp
Write 1	–1/2	–1/2	ØWp

One benefit of this scheme is the register drivers [MR] only need to drive current in one direction, making them much simpler.

For the bidirectional drive of the digit-lines the common end of the wires is connected to a supply of ~ –2.5V while the drivers are capable of switching the other end to either –4.5V or 0V [MD].

• Shared sense

With the 2D form, a separate sense wire may be avoided. Rather, sensing can be accomplished using the drive wires of the independant axis, here the register-bit-lines.

The magnetic flip which will produce a sense pulse is triggered by ØRp on the digit-lines. As these lines are orthogonal to the register-bit-lines which are observed for the sense pulses, the magnetic field of the triggering pulse has minimal inductive influence on the register-bit-lines and the small sense pulses remain distinguishable from the triggering pulse.

The drive policy provides a second benefit here in that there is no need to drive current on the sensing wire during the sensing process, further reducing potential drive-sense conflict.

• Sense Amplifer

The 16 sense amplifiers are formed from two transistors each. The 1st transistor is directly coupled to a grounded register-bit wire. An emitter resistor to V+4.5 biases it into the linear region with negative feedback. The collector is AC-coupled to the 2nd transistor.

The 2nd transistor has a bias network switched by the strobe input [diode into the bias resistors]. The inhibit state for the strobe input is logical 0. In this state the diode conducts, lowering the bias at the transistor base to some volts below cutoff, its collector thus logical 1.

When the strobe input is asserted 1 during the 1st half of the digit cycle, the bias for the 2nd transistor is raised much nearer to conduction. After the clear-read pulse (ØRp) at the beginning of the digit cycle, a positive-going sense pulse on the register-bit wire is amplified to produce a negative-going pulse at the base of the 2nd transistor, pushing it into conduction, and thus a 0-going pulse at the collector. This pulse is AC-coupled into an SK~ edge-set type input for parallel-load of a register Decade. [Core Sense Scoping TOP.29]

There are two (known) versions of the sense amplifier, the primary difference being the bias network of the 2nd transistor. The precise voltage levels at the base for the two strobe states thus differ between the two versions.

Number Registers

The 4 number registers are each comprised of its 16 digits in core memory and one or two 'Decades', where a Decade is either a 4-bit decade counter with parallel load or a 4-bit latch.

The 4 registers are:

Number Registers	
RK	Keyboard entry. During multiplication, this is the multiplicand. During division, this is the divisor.
RA	Accumulator. During multiplication, the product will accumulate in RA. For division, the first-entered operand is transferred to RA to become the dividend.
RB	For multiplication, the first-entered operand is transferred to RB to become the multiplier. During division, the quotient will build in RB.
RM	User memory.

• Decades	The digit contents of registers are continually & sequentially read from core into a decade counter (R*r*D) associated with each register. While in a counter the value may be altered via counter arithmetic, or cleared. The value is then written back to core to restore or update the digit value.
• RK 2nd decade	RK has an associated 2nd decade, RKX. This decade is a latch only and does not have counter functionality. It does include a 2nd parallel-load input for loading numerals from the keyboard.
• RB 2nd decade	RB also has an associated 2nd decade, RBX.

• **Transfer** One might anticipate that copying one register to another would be accomplished in the common manner of using the arithmetic facilities to effectively perform *D*=*S*+0. It is not clear why this sort of method was not used but, in the IME-84 as it is, only the RA and RM registers are able to receive a sum.

Instead, copying the contents of one register to another is accomplished with 4 transfer gates. Each transfer gate gates the sense outputs and decade parallel-load inputs of a register onto a single shared 4-bit transfer bus.

• Construction Logically, a transfer gate is constructed from two gated pulsers on each bit line - one for input onto the bus and one for output off the bus; both are enabled when the transfer gate is enabled. Electrically, the transfer gates establish a low-impedance AC connection between a register sense-decade circuit and the transfer bus. The AC connection suffices for transporting sense pulses.

• Operation — In copying one register *S* to another register *D*, the first action is to clear *D*. The transfer gates of both registers are then enabled. This effectively places two sense outputs and up to four decade inputs on each bit-line of the transfer bus. During the subsequent digit cycles, sense pulses from register *S* trigger-set those decades of both *S* & *D* which have their parallel-load enabled. Register *D* having been cleared prior will generate no sense pulses to confuse matters on the transfer bus.

• **Shifting** — The operations of Numeral Entry, Multiplication & Division require the ability to shift appropriate registers. Shifting is performed by a mechanism formed from flip-flop FSHLF and logic controlling the core read and write of the decades for RK and RB. Only RK & RB are available for shifting, selected by SH0B1K.

• Enable — Shifting is enabled when FPWORK=1 and SHIFT=1. SHIFT=1 when:

- Add/Subtract is not being performed,
- and either:
    - the PQ Counter is in state 2 or 3 during M/D,
    - or, M/D is otherwise NOT being performed.

• Action — With shifting enabled, FSHLF (Flag Shift Leap Frog) is toggled on the 1-edge of ØD and thus operates in quadrature with digit cycles which are synchronous to the 0-edge of ØD. The 4-state sequence thus produced is used to leap-frog a digit value into an adjacent digit using two Decades as the leaping frogs of interim storage:

- step 1: load digit (d) from core into one decade (a)
- step 2: write the digit from another decade (b)
- step 3: load digit (d±1) into decade (b)
- step 4: write digit (d±1) from decade (a)

• Direction — The shift direction (d+1 versus d–1) is determined by the count direction of the ØD Counter. ØDC counts up by default, producing an up/left shift. When PQC=2, ØDC counts down for a down/right shift.

• Rotate — The shift operation is end-around, it actually produces a rotation of the digits: during ØN at the end of a shift-up number cycle, the MSD which was loaded into one of the decades is written into the LSD, as ØDC=0 during ØN. During a shift-down, the LSD loaded during ØN at the beginning is written into the MSD as ØDC=15 at the first digit step.

This is used for benefit in some arithmetic operations, e.g. during Multiply, RB is being shifted down, at the end of the multiply RB digits have been restored to their proper position, and the value thus retained for further use as an operand.

Arithmetic & the Digit Cycle

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

FSUB=0: RAD += RKD

FSUB=1: RAD –= RKD

This is accomplished with counting techniques, not binary-addition logic.

Following is an explanation of the add and subtract process, by example. Given RKD=7 and RAD=5:

• Reading — An operation condition sets A_START~=1. In the 1st half of the digit cycle, the RKD and RAD decades are loaded as usual.

• Add — At the middle of the digit period the 0-edge of ØD~ sends A_START~=0, setting ØA=1 and FRKR=0.

During the ØA period, the 10 pulses of ØA10 will clock RKD to perform a full rotation [RKD.INC~]. RKD will roll over after 10-7=3 pulses, the rollover triggers FRKR=1.

With FSUB=0 (add) and FRKR=1, ØA10 pulses are enabled to increment RAD. ØA and the ØA10 pulses end after another 7 pulses.

The 10th pulse of ØA10 completes a ØAC cycle: on the 0-edge of FØA8, FØA is cleared 0. The net result when ØA ends is:

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

• Writing — The normal 2nd half of a digit cycle now proceeds and at ØWp the RKD digit is restored to core (necessary due to the destructive read of core) and the RAD digit modified with the sum written to core.

• Carry-in — FCYIN is set 1 during the 1st pulse period of ØA10. If there is a carry-in (FRACY=1) from the previous digit, the 1st pulse of ØA10 increments RAD.

At the end of the FCYIN period, FRACY is cleared 0, so is ready for a carry generation from the current digit.

• Subtract — For subtraction, FSUB=1. During ØA, the behaviour of RKD is the same as for addition. However, incrementing of RAD is now enabled while FRKR=0 with the exception of the FCYIN period - thus from the 2nd pulse of ØA10 till the RKD rollover. The exception is tied up with the borrow strategy.

• Borrow — A borrow necessitates *one less* increment of RAD. The FCYIN exception produces this borrow by default. As the carry flag produces an increment of RAD, 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 RAD indicates *no* borrow is necessary. For example, for (5–2): RAD=5,RKD=2, the 5 will be incremented 8 times producing RAD=3 and FRACY=1. FRACY is as desired for providing the compensating increment to the next digit. For (5-6): RAD=5,RKD=6, the 5 will be incremented 4 times producing RAD=9 and FRACY=0. With FRACY=0, the default borrow is effective in the next digit.

An initial increment of RAD is generated for the first digit [ØD0 to AND gate], this compensates for the missing increment in that digit.

• GT Mode — With the Memory Mode switch set to the GT position (Grand Total), all arithmetic increment pulses directed to RA are also directed to RM, thus all additions & subtractions performed on RA are also performed on RM.

DP Counters

There are two 4-bit hex counters present for decimal-point management. DPKC stores the digit position of the decimal point for the number in the RK register. DPRC stores the DP for operation results.

• Enable — Pressing of the DP key sets FDPKE=1, enabling DPKC & DPRC to count numeral entries.

• DPKC — Once enabled, DPKC will increment on each entry of a numeral,

• DPRC — DPRC will similarly count on each numeral entry, however it may either increment or decrement. The count direction is determined by the operation about to be performed:

  - • Multiplication: DPRC continues counting up as the numerals of the 2nd operand are entered, so the result DP will be the sum of those of the two operands.
  - • Division: DPRC counts down, subtracting the DP position of the divisor from that of the dividend.

Negative Flag

The Negative flag [FNEG] drives the Negative lamp to indicate the result of an operation is less than 0.

• Strategy — The Negative flag detects whether a borrow occurred out of digit 15 during a subtract operation.

Recall from the Arithmetic commentary that a borrow from a digit subtraction is indicated by the *lack* of rollover of RAD, alternatively stated as the rollover of RAD indicates no borrow.

If a subtraction arithmetic number cycle is being performed, at the beginning of the arithmetic period of digit 15, FNEG is set 1 [18.9Nv]. If a rollover of RAD occurs during the digit 15 arithmetic, FNEG is cleared

0 [18.15Bf].

• Already negative — If a subtraction is being performed from an already negative number, the Negative-Hold flag [FNEGH] is asserted, which then holds FNEG at set-1, to inhibit it from being cleared.

Overflow Flag

The Overflow flag [FOVF] drives the Overflow lamp to indicate the result of an addition or multiplication has overflowed the register capacity.

• Strategy — A carry out of digit 15 is detected: FOVF is enabled to set during ØA of digit 15 as long as FNEG has not been set and the procedure is not SUBTRACT or DIVIDE. If RAD rolls over FOVF is set 1.

Procedures

A procedure is a sequence of actions taken to perform some function. IME-84 procedures can be characterised into 3 categories in regard to their mechanism of execution, as presented in the table below. The categories and the procedures in them are described in following sections.

List of Procedures

Procedure	**Keypress**	**Sequencer**	**Sync**	**Description**
CLEAR-ALL	•			Clear state
CLEAR-RK	•			Clear entry
CLEAR-RM	•			Clear user memory
REG-SELECT	•			Select register for display
DECIMAL-POINT	•			Enter decimal point
PREPARE-DIVIDE	•			Prepare for division
PREPARE-MULTIPLY		•		Prepare for multiplication
TOTAL		•		Present sum, negate if needed
RECALL		•		Copy result to entry
Nn		•		Raise exponent
IM		•		Store in user memory
RM		•		Copy user memory to entry
NUMERAL-ENTRY			•	Enter a numeral
ADD/SUBTRACT			•	Calculate add or subtract
MULTIPLY			•	Calculate multiply
DIVIDE			•	Calculate divide

Keypress Procedures

Keypress procedures are comparitively simple procedures that - while they may perform multiple actions - do not require state-memory beyond the indication of the procedure. Keypress procedures are not synchronised to the number cycle.

Procedure: CLEAR-ALL, CLEAR-RK, CLEAR-RM

The CLEAR procedures clear some subset of state in the calculator.

• CLEAR-ALL	Pressing the C key clears registers RK, RA & RB and assorted other state-holding entities such as the control counters, DP Counters, etc. to bring the machine to a determined state. It does not clear RM.
• CLEAR-RK	Pressing the Clear-entry (CT) key clears RK.
• CLEAR-RM	Pressing the Clear-mem (CM) key clears RM.
• Register clear	Clearing a register is accomplished simply by asserting Rrd_CLR~=0 for the Register Decade(s) for long enough for all the register digits in core to be written.
	Note this does not require synchronising to the number cycle. Clearing can begin at any digit, as long as the clearing persists for 17 or more digit periods. The operation is reliant on the minimum period of a keypress being longer than 17 digit periods.
• Variation	A modification was introduced around 1965 whereby the C key would clear only RK & RB, and the CT key would clear both RK & RA.

Procedure: REG-SELECT

A keypress of one of the 4 Register-Select keys directly sets the state of the RS Latch formed from RS1,2 with a 2-bit encoding of the register. The RS Latch state is decoded to 1-of-4 and used to enable data from the according Register Decade to be applied to the display numeral decoder via the Display Selector.

Procedure: DECIMAL-POINT

Pressing of the DP key sets FDPKE=1, which enables counting for DPKC and DPRC on each entry of a numeral.

• Anomaly	DP management does not work properly for addition & subtraction, entering sequential operands with decimal points simply results in a continually incrementing DP position. See MOD2025.11 [page 31].

Procedure: PREPARE-DIVIDE

Pressing of the Divide key (÷) enables the RK and RA transfer gates to copy RK to RA, where it will become the dividend in the upcoming divide procedure. When the Divide key is released, RK is cleared by a pulse long enough to clear a register [T19.6,7,8,et al].

• Anomaly	RA is not cleared prior to the transfer. If RA is not zero it must be manually cleared before pressing the Divide key. See MOD2025.12 [page 31].

Sequencer Procedures

Sequencer procedures are performed by a state machine which will be referred to as the Sequencer. The Sequencer does as its name implies: sequences in time a set of actions. The Sequencer is constructed from a large set of pulsers and 10 monostable flip-flops. The pulsers along with NORN gates in essence form next-state logic, while the monostables are the state-holding entities.

In operation, a keypress triggers a pulser to initiate a sequence. The pulser triggers one of the state monostables. When the period of that monostable expires, it triggers another pulser which triggers another state monostable, and so on for as long as the sequence requires. Most of the pulsers have a gate input to conditionally direct the sequence.

The notable aspect of this design is that a sequence is entirely dependant on RC time-constants and is unsynchronised to the master-clock-derived number cycle. This places inherent restrictions on what can be accomplished with the Sequencer. The actions performed by the Sequencer are primarily clearing and transferring registers, both of these are idempotent - the result is the same as long as the state action period is at least 17 digit periods. These actions can begin in the middle of a number cycle and can include a fractional portion of a number cycle. Other actions performed are simple triggered actions, and triggering a keypress at the end of the sequence.

In the Sequencer procedure descriptions following, a transfer of one register to another means both registers are placed on the transfer bus (register transfer-gate enabled). The indicated direction of transfer is derived from which register was cleared prior.

Procedure: PREPARE-MULTIPLY

The Multiply key (X) invokes a sequenced procedure to copy the multiplier entered into RK to RB.

PREPARE-MULTIPLY	
State	**Action**
-	Press X key
S3.3	0 => RB
S5.1	RK => RB
S8.4	0 => RK
(S2.2)	0 => RA

• Variation — In some units, RA is cleared in preparation for the eventual summing of the product. However, the pulser components that produce S2.2 are absent in some units, thus RA is not automatically cleared and must be manually pre-cleared if necessary. Without a pre-clearing, the product will simply accumulate into RA with its prior contents. This allows sum-of-product operations to be performed without requiring or using an additional register.

Procedure: TOTAL

The Total key (*) presents the result of a series of additions and subtractions by switching the display to RA. If a sum is less than 0 it is natively in 10s-complement form, the Total key (un)complements it by performing a subtraction so it displays as the expected negative value.

TOTAL			
State	**Action**	**State**	**Action**
-	Press T key		
if RA current:		if RK current:	
*	select RK	*	select RA
if NEG=1:			
S8.1	0 => RK		
S1.1	RA => RK		
S2.1	0 => RA		
S7.1	trigger KSUB		

• Subtotal	A 2nd press of the Total key (while RA is displayed) switches the display back to RK in preparation for another entry, and complements a negative sum back to 10s-complement form so further entries will accumulate properly into the sum.
	A double press then, permits a subtotal to be observed in the course of a summation sequence while properly accounting for negative sums.
• Suppress count	If the Item-Count (N) mode is enabled, a Subtract operation increments the item-count in RM. This increment needs to be suppressed for complementing SUBTRACT procedures instigated by the TOTAL procedure for negative values. The suppression is performed by an AND gate inhibiting RM_INC~ during the FSM7 period [logic.11].

Procedure: RECALL

The Recall key (R) retrieves another register for re-use as entry.

- If RA is current, copy RA to RK.
- If RB is current, copy RB to RK.
- Clear RA & RB, and select RK
- If neither RA or RB was selected, RK is cleared with no other actions taken.

RECALL			
State	**Action**	**State**	**Action**
-	Press R key		
S8.6	0 => RK		
H1	-		
if RA current:		if RB current:	
S1.3	RA => RK	S5.3	RB => RK
S2.1	0 => RA	S2.3	0 => RA
S3.1	0=>RB select RK		

Note KR~ is asserted during CLEAR-ALL, and thus the RECALL sequence is also performed by CLEAR-ALL.

Procedure: Nn

The Exponentiation key (Nn) raises a base N entered into RK to some power n, where n is determined by how many times the Nn key is pressed. This is accomplished with some register copying followed by triggering a multiply operation.

Nn			
State	**Action**	**State**	**Action**
-	Press Nn key		
if RK current:		if RA current:	
S3.2	0 => RB	S8.5	0 => RK
S5.2	RK => RB	H2	
S4.1	RB => RA	S1.2	RA => RK
S8.3	0 => RK	S2.1	0 => RA
H2			
S6.2	trigger KME	S6.1	trigger KME

• Power 1	If RK is the register selected for display, a press of the Nn key will initiate the exponentiation process. The base entered in RK is copied to RB and RB to RA (RA is presumed to have been cleared prior), RK is cleared, and the multiplication procedure triggerred. The multiplication multiplies RB by 0, so nothing is added into RA. The multiplication displays RA at completion. RA has been preloaded with the base, so the display is effectively base^1.
• Power 2+	On each subsequent Nn keypress with RA being the displayed register, the product in RA is copied to RK to enter it as the next multiplicand, RA cleared, and another multiplication triggered.
• Counting powers	If the Memory Mode switch has been set to N, each multiplication increments RM by 1: The assertion of FPWORK at the beginning of the multiply sets FNINC=1. This enables MF_N to assert during FCYIN of ØD0, which enables an increment pulse to be applied to RM. As MF_N deasserts, FNINC is cleared 0, so only a single increment is performed for each assertion of FPWORK.
• Anomaly	RA is not cleared on the 1st press of the Nn key. If RA is not zero it must be manually cleared before starting an exponentiation operation. See MOD2025.12 [page 31].

Procedure: IM

The Store Memory key (IM) copies RK to RM and clears RK.

IM	
State	**Action**
-	Press IM key
if RK current:	
S9.1	0 => RM
S10.1	RK => RM
S8.2	0 => RK

Procedure: RM

The Recall Memory key (RM) copies RM to RK.

RM	
State	**Action**
-	Press RM key
if RK current:	
S10.2	RK <=> RM

• Anomaly RK is not cleared prior to the RK<=>RM transfer. If RK is not zero it must be manually cleared before pressing the RM key. See MOD2026.02 [page 32].

Synchronised Procedures

Synchronised procedures modify a number value in a register and consequently must be synchronised to the number cycle. For example, a (serial) add must begin at the LSD and not continue after the MSD. There is a small sequence counter involved for some of these procedures but there is no formal state machine governing the overall execution of a procedure, rather, there is some amount of procedure-specific ad-hoc logic.

- Debounce — Pressing of a numeral key or an arithmetic operation key is debounced with one of two Schmitt triggers [T24.1,2] [TK.1,2,3].
- Syncing — Synchronised procedures are initiated by de-assertion of KNML or the assertion of KOP~. The 0-edge of either signal sets FPSYNC=1. This is then synchronised to the number cycle: on the next instance of a 0-edge of ØN, a sync pulse clears FPSYNC. The consequent 0-edge of FPSYNC sets FPWORK=1 and the procedure may now proceed.

  During execution of the procedure, sync pulses may be suppressed [20.9Nv].

  As the procedure ends, a sync pulse is permitted through to FPWORK and it now clears.
- PL Counter — Multiplication and Division both involve a primary loop and a secondary loop. The primary loop is the 'outer' or controlling loop in the algorithm, and in the IME-84 is always executed 16 times. The PL Counter (PLC) is a 4-bit hex counter tracking iterations of the primary loop.
- PQ Counter — The Primary Loop Sequence Counter (PQC) is a 2-bit counter producing a sequence of 4 steps to sequence actions within an iteration of the primary loop. PQC increments on number cycles but self-stops at 0 to idle. PQC is triggered into action during Multiply or Divide by setting FPQ1=1, upon which it proceeds through the sequence 1, 2, 3 over 3 number cycles till returning to 0 idle.

PQC States

PQC	PROCEDURE	ACTION
0	NML-ENT	Shift RK UP
	ADD/SUB	RA ±= RK
	MUL	Secondary loop: partial-product additions
	DIV	Secondary loop: construct quotient digit
1	MUL	-
	DIV	Overdraft correction: RA += RK
2	MUL	Shift RB DOWN
	DIV	Shift RK DOWN
3	MUL	Shift RK UP
	DIV	Shift RB UP

Procedure: NUMERAL-ENTRY

Numeral entry proceeds largely through default actions once FPWORK has been asserted.

• Keypress	When a numeral key is pressed, the numeral value is encoded to BCD [KN1,2,4,8] and applied to parallel-load Set inputs of latch RKX. RKX is not being cleared and loaded from core by default, so now holds the keypress numeral value.
• Start entry	Releasing of the numeral key sets FPSYNC=1. This is synchronised to ØN per the sync process, setting FPWORK=1.
• Shifting	With FPWORK asserted, the default assertion of SHIFT=1 begins a shift operation at the beginning of digit-0. In the middle of digit-0, FSHLF toggles to 1, enabling the new numeral in RKX to be written to core. The shift operation then proceeds through the number cycle. PQC is at its idle 0 during this number cycle, the shift operation performed is thus RK-up.
• Finish	With FPSYNC=0, the 0-edge of ØN clears FPWORK.
• Decimal point	If the DP key was pressed prior during entry of this number, FDPKE=1, and pressing of a numeral key increments counter DPKC.
• Anomaly	Due to the end-around rotation of the MSD to the LSD during shift-up, entering excessive digits will result in the MSD wrapping to the LSD where it will be OR'd with the new numeral and may produce digit values outside the BCD encoding range.

Procedure: ADD/SUBTRACT

Executing Add & Subtract is similar to Numeral-Entry in being performed in a single number cycle, the main difference is that shifting is suppressed, which consequently enables arithmetic.

• Select function	The keypress of + or – immediately sets FSUB appropriately to determine whether addition or subtraction will be performed. The keypress also sets FPAS=1.
• Start	The keypress sets FPSYNC=1 and synchronisation to the number cycle proceeds per the sync process.
• Execution	Shifting is suppressed due to the assertion of FPAS.
	With FPWORK asserted and shifting suppressed, arithmetic is enabled for the upcoming number cycle to perform RA ±= RK.
• Finish	As with Numeral-Entry, FPWORK is cleared at the 2nd ØN.
	When the + or – key is released, a pulse long enough to clear a register is generated [T19.6,7,8,et al] to clear RK in preparation for another entry.

Procedure: MULTIPLY

The Multiply procedure is performed when the "=" key is pressed.

• Operands	RK contains the multiplicand, RB contains the multiplier, the product will accumulate in RA.
• PLC	During each iteration of the Primary Loop, one digit of the multiplier RB is processed.
• MSC	Multiplication Partial-Product Counter.
	In each primary loop iteration, MSC will count out 10 iterations of a secondary loop, where each iteration is a number cycle during which multiplicand RK may be added to product RA. The additions are mediated by flag FMBXR (RBX has Rolled to zero).
• Other facilities	PQC, SHLF, RBX, FBXLD, FMBXR.

The process is described in the following commentary:

• Prior	Prior to starting the Multiply, RBX has been loaded with the LSD of multiplier RB due to default activity during idle: FBXLD is set 1 at the end of ØN and cleared at the end of ØD0. Also PLC=0, MSC=0, SHC=0 and FMBXR=0.
• Start	Pressing of the Multiply-Equal key asserts KOPME~=0 and KOP~=0. The 0-edge of these signals respectively set FPML=1 and FPSYNC=1. On the 1st subsequent ØN, FPSYNC sets FPWORK=1.
	The 0-edge of KME~ also sets the register-select flags [FRS1,2] to RA so the product in RA will be displayed at the end of the multiplication.
	Processing is now in the first iteration of the primary loop and the first iteration of the secondary loop. Note further ØN sync pulses are suppressed [20.9Nv].
• P-P Additions	At the beginning of each iteration of the secondary loop, both MSC and RBX are incremented [20.4Nv]. When RBX rolls to 0, FMBXR is set 1, disabling partial-product additions via MPPAE=0.
	When FMBXR=0, addition of RK to RA is enabled by MPPAE=1 (Multiply Partial-Product Accumulate Enabled). MPPAE controls both the incrementing of RA and the initiation of arithmetic cycles. With addition enabled, the secondary-loop number cycle adds the multiplicand RK to the product RA.
	At the end of 10 iterations of the secondary loop, MSC increments PQC to 1. Secondary loop iteration is now inhibited [20.4Nv].
• Next Digit	Now back in the primary loop, with PQC enabled to cycle, the number cycle of PQC=1 is a null cycle: shifting is suppressed [21.8Nv] as well as arithmetic [MPPAE].
	During PQC=2, RB is shifted down so RBX will load with the next higher digit of the multiplier. RBX is loaded after the end of the RB shift-down

[FBXLD.SE].

During PQC=3, RK is shifted up so subsequent partial-products secondary loop iterations will add the multiplicand*10.

PLC is incremented [N20.Z] to track the primary loop iterations.

• Finish — When 16 iterations of the primary loop have transpired, FPML is cleared [FPML.RE]. The consequent 0-edge of FPML clears FPWORK.

Procedure: DIVIDE

The division procedure is performed when the "÷=" key is pressed.

• Operands — RK contains the divisor, RA contains the dividend, the quotient will build in RB.

• FPDV1 — Flag indicating 1st stage of Division. During stage 1, the divisor will be shifted to the upper end of RK and PLC pre-incremented to limit the primary-loop iterations in stage 2 for the available resolution.

• FPDV2 — Flag indicating 2nd stage of Division. During stage 2, primary-loop iterations will be performed to reduce the dividend in RA while building the quotient in RB.

• D1C — Division stage 1 Counter. D1C will count out 15 number cycles to constitute stage 1.

• FDK15Z — Flag Division RK digit 15 is Zero. During stage 1 as the divisor is being shifted up in RK, this flag will indicate when a non-zero digit has entered the MSD of RK.

• PLC — PLC will first be adjusted in stage 1 for the resolution available, in stage 2 the remaining counts will control primary loop iterations.

• PQC — Used during stage 2 to sequence actions within primary-loop iterations.

• FDA15R — Flag Division RA digit 15 has Rolled over. Used during stage 2 to detect overdraft from the dividend.

In the following description of the process, "[e*n*]" references are events enumerated on the Division Timing Graph [logic.19].

• Start — Depressing the Division-Equals key asserts KDE~, KOPDE~ and KOP~ [e0]. KDE~ assertion sets FRS1,2 to select RB for display (once the division is complete) and sets FSUB=1 so upcoming arithmetic operations will subtract. The 0-edge of KOPDE~ sets FDK15Z=1. The 0-edge of KOP~ initiates the sync process, resulting in FPWORK asserted [e1].

• Stage 1a — The 0-edge of FPWORK~ sets FPDV1=1 [e2]. This increments counter D1C to 1 and enables D1C to be incremented on each number cycle. During these number cycles RK is shifted up (PQC=0) until a non-zero digit enters digit 15: when FDK15Z goes 0 [e3], SYNC~ pulses are re-enabled, the 1st one clearing FPWORK=0 [e4].

• Stage 1b — With FPWORK deasserted, shifting of RK is terminated while incrementing of PLC is enabled. FPDV1 remains asserted, so D1C continues incrementing. When DSC rolls from 15 to 0, FPDV1 is cleared [e5], disabling incrementing of D1C & PLC. The 0-edge of FPDV1 sets FPDV2=1.

At the beginning of stage 1b, D1C=1+16–#dd (where #dd=number of digits in the divisor). At the beginning of stage 2, PLC=#dd–1, which will allow for (16–#dd+1) primary-loop iterations in stage 2.

• Stage 2

FPDV2 assertion suppresses further sync pulses and sets and holds FPWORK in assertion. With FPWORK and FPDV2 asserted, and shifting suppressed [PQC=0,21.8Nv], division secondary-loop iterations of arithmetic number cycles are performed. The dividend is reduced by subtracting the divisor RK from RA. If the result of the subtraction is positive, FDA15R sets 1, indicating there is no borrow. The upcoming ØN pulse clears FDA15R, the transition increments quotient digit RBX.

• Next Digit

If the result of the reduction subtraction is negative an overdraft from the dividend occurred. PQC is incremented to 1, exiting the reduction secondary loop. This clears FSUB [e6], switching arithmetic to addition. Shifting is suppressed [21.8Nv] for this number cycle, so an arithmetic number cycle is performed, adding RK back to RA to correct the overdraft.

PQC is incremented to 2, which consequently increments PLC [e7]. During the next two number cycles of the PQC cycle (PQC=2 & 3), RK is shifted down and RB shifted up. At the beginning of the RB shift-up, the new quotient digit built in RBX is written to the LSD.

At the end of the PQC cycle, FSUB is set back to 1 [e8]. With PQC back at 0, processing proceeds with another secondary loop of reducing the dividend and construction of the next quotient digit.

• Finish

If PLC rolled over from 15 to 0 when it was decremented, the reduction of the dividend is finished and FPDV2 is cleared 0 [e9]. WIth FPDV2=0, FPWORK is released from forced-set, but remains 1 for the time being. The PQC cycle proceeds through PQC=2, but both the shift down of RK [PQC=2=>20.8Nv] and the sync pulse [PQC=2=>20.9Nv] are suppressed. Sync pulses thereafter are enabled, thus the sync pulse at the end of PQC=3 finally clears FPWORK=0 [e10].

• GT Mode

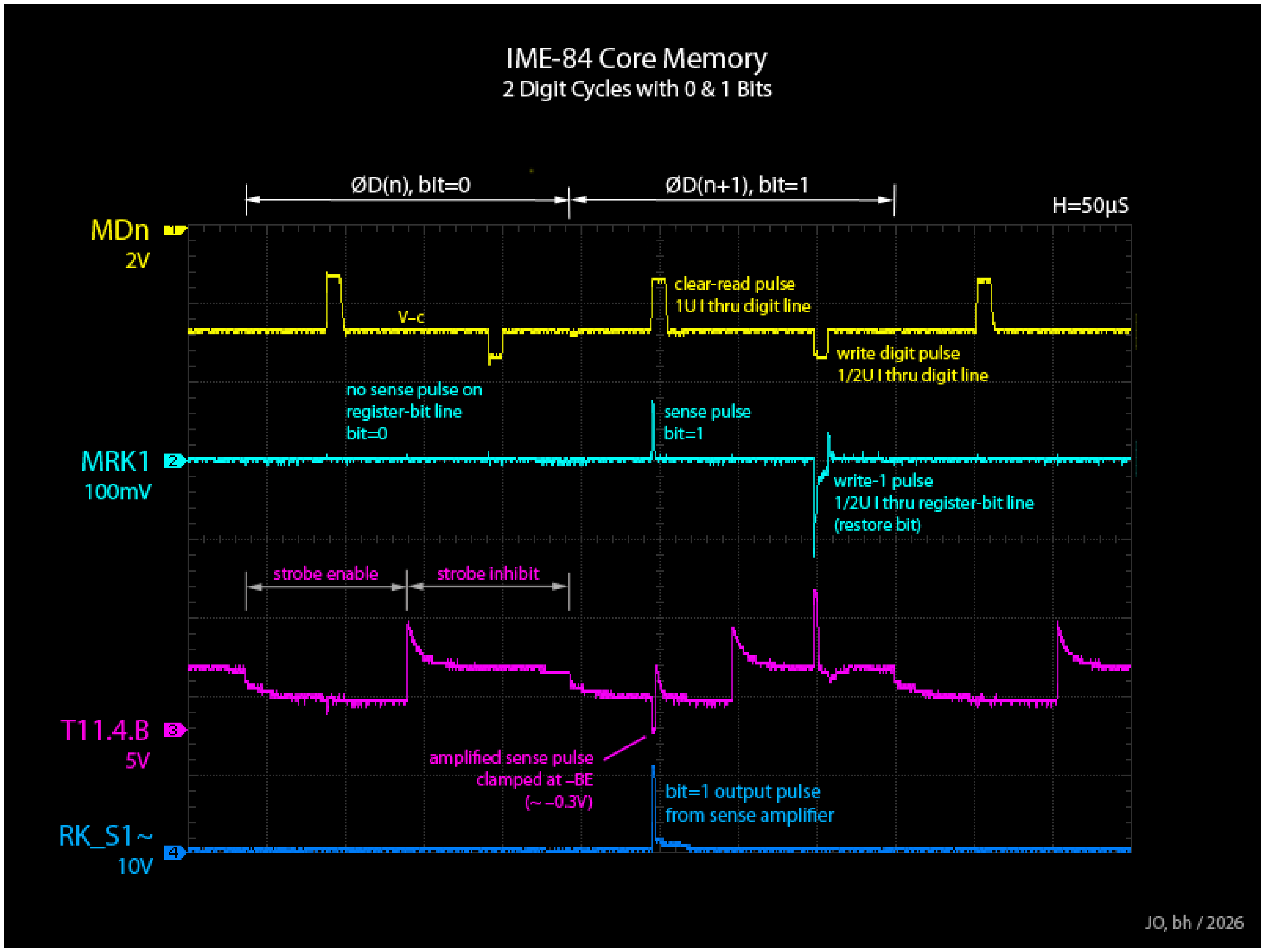
With the Memory Mode switch set to the GT position, the subtractions and add-backs performed on the dividend in RA are also performed on RM. The net effect is:

RM(new) = RM(prior) – (divisor*quotient)

Appendices

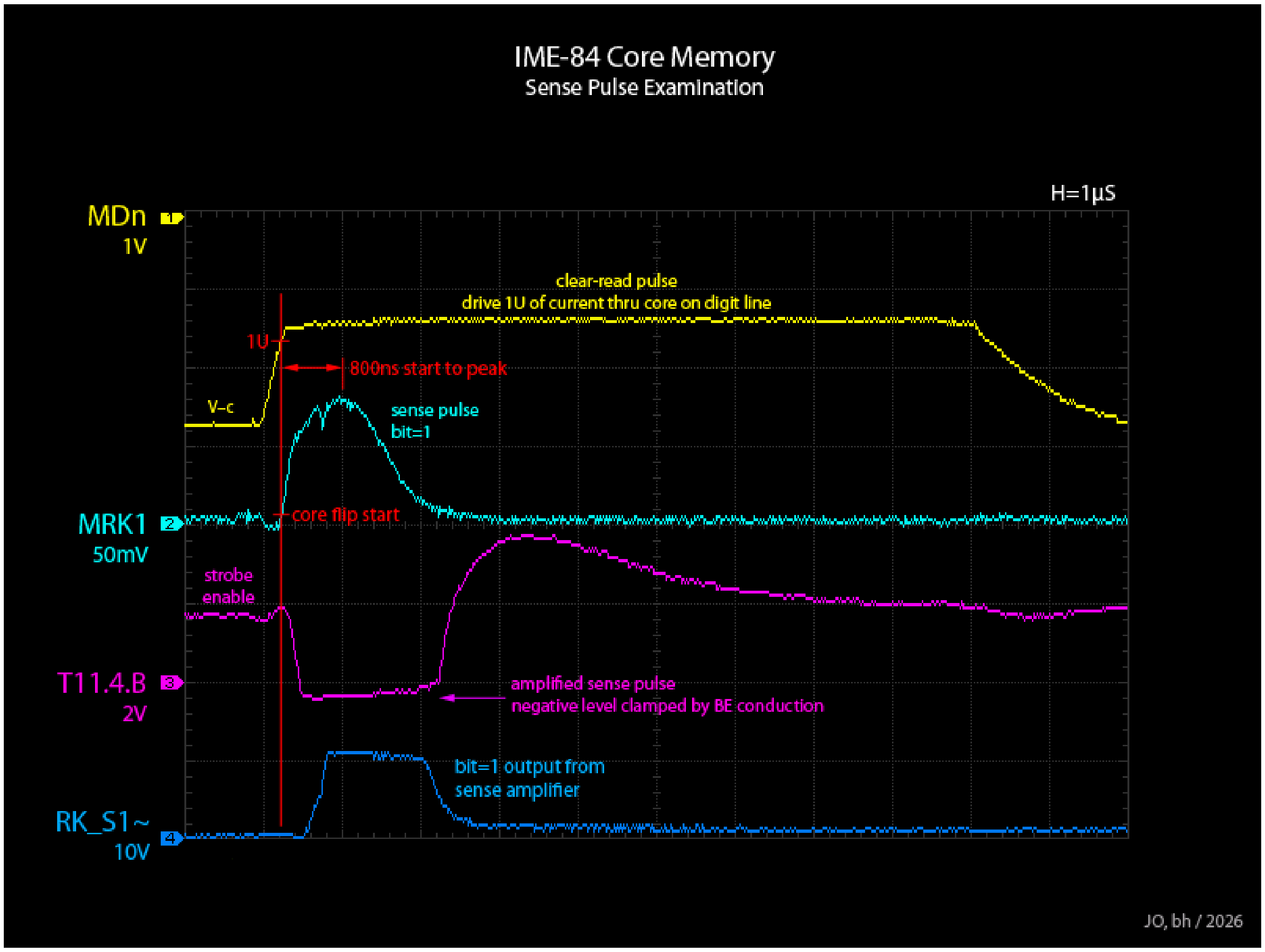
Appendix 1: Core Cycle Scoping

A scope capture of two core memory digit cycles. The first digit cycle reads a 0-bit, the 2nd reads and restores a 1-bit.



Appendix 2: Core Sense Scoping

A closer examination of core memory signals showing the sense pulse.



Appendix 3: Engineering Modifications

Some modifications to correct or improve circuit reliability.

MOD196X.A: Modification to Stabilise FSM7

• Problem	The Rm+Cm pair for the FSM7 monostable presents a quite high impedance for the conducting-state base drive of the Q transistor (T27.3). The low drive may leave T27.3 in or near linear operation where it amplifies noise which in turn is presented to the base of T27.4, mis-triggerring the monostable.
• Solution	Rm+Cm for FSM7 are changed from 100k+220n to 47K+470n.
• History	The problem was encountered and traced in 2025 during repairs of a unit which had the original higher-impedance values. Another unit observed had these components modified to the values described, using 1960s-era components. Apparently then, the problem was recognised and a solution devised sometime in the mid-60s.

MOD2026.01: Modification to KNn Pulser Margins

• Problem	When the Nn key is pressed, the trigger pulses from the 3.2 and 8.5 pulsers may be of high enough amplitude that the corresponding FSM monostable is triggerred even if the pulser is inhibited.
• Solution	The idle-state voltage level of the KNn~ line is lowered, thus reducing the trigger pulse magnitude. However, lowering the KNn~ idle-state voltage also reduces the clearance margin for inhibiting pulses in the pulsers gated by KNn (4.1, 5.2, etc.), so a balance of values must be found.
• Implementation	A 68K resistor to GND is added to the KNn~ line, lowering the idle level from –12V to ~ –9V.

Appendix 4: User-Interface Modifications

A few modifications which remove some anomalous behaviour or improve the operational experience for the user.

MOD2025.11: Modification to Improve the DP Display during Summation

• Problem	There is no effective DP management when performing a sequence of additions and subtractions. Pressing the DP key at any point will result in the DP position moving left with all subsequent numeric keypresses, including both integral and fractional digits of further operands. This is due to the FDPKE flag not being reset after an operand has been fully entered.
• Improvement	A trivial modification to the electronics can be made which will allow a DP position to be set at entry of the first operand. The DP position then remains fixed in the display during entry of further operands, so long as the DP key is not again pressed. It is up to the operator to enter enough 0 keypresses for a given operand as necessary to obtain the intended alignment. The DP position is reset to 0 (LSD) when Clear-All is pressed, as usual. While far from a complete solution to DP issues for the IME-84, it is a small improvement providing for a consistent display of the DP during summations.
• Implementation	See [logic.16]. A single diode is added to provide a Reset-Collector input to the FDPKE flag. The objective is to simply reset FDPKE when an addition or subtraction operation is performed, thus inhibiting the DP counters from further incrementing.
• Why not?	The modification is so simple that one wonders why it wasn't included in the original design. This is perhaps explained by the small value of the integral Italian lire currency in the period which meant that for simple currency accounting summations, decimal points were not needed.

MOD2025.12: Modification to Clear RA prior to Divide & Exponentiation

• Problem	RA is not automatically cleared prior to transferring the dividend to RA when the Divide key is pressed, nor prior to transferring the base to RA when starting an exponentiation on the 1st press of the Nn key. In consequence, an entered dividend or base is OR'd with the current contents of RA. In contrast to the removal of clearing RA prior to a multiply, where the behaviour has the benefit of enabling sum-of-products, the lack of clear for divide and exponentiation does not have any apparent benefit.
• Improvement	A new action to clear RA is interjected at the beginning of the PREPARE-DIVIDE and KNn(K) procedures.

• Implementation

See [logic.6]. For PREPARE-DIVIDE, a break is made in the circuit between the Divide key and the KDIV~ signal distribution by removing the jumper between points 65 and 25 on the keyboard main circuit board.

Two monostables with triggerring pulsers are inserted in this break. The 1st monostable performs the RA clear, the 2nd in essence simulates the pressing of the Divide key to perform the original PREPARE-DIVIDE procedure.

For KNn(K), the 1st monostable is triggerred by the Nn key only if RK is currently selected. The trigger pulser for the 2nd monostable is gated so the monostable is not triggerred during KNn(K).

The new components can all be assembled on a small board, with several connections to the keyboard boards.

• Ideal Solution

For greater consistency with the principles of the IME-84 design, the PREPARE-DIVIDE procedure would be changed from a Keypress procedure to a Sequencer procedure. This would involve implementing a 2 or 3-step sequence of KDIV=>FSM2=>FSM1=>FSM8 (0=>RA,RK<=>RA,0=>RK), requiring the addition of 2 or 3 pulsers on board 26 [logic.8], as well as a gate or inverter to prevent a loop forming from the ungated trigger of FSM2 from FSM1. This necessitates tacking a dozen or more components onto board 26.

The original PREPARE-DIVIDE mechanism which enables the RK=>RA transfer must also be disconnected. This involves opening two circuits.

Also, there are unfortunately no spare pins on board 26 to feed KDIV~ onto the board, so a wire connection onto the board is necessary.

While this solution could readily have been implemented as part of the original design, it is not now suggested as a modification due to it's disruptiveness.

• Alternative

Another solution considered is to suppress the sense pulses from RA during the RK<=>RA transfer, essentially clearing RA as the transfer takes place. However, this solution presents a probability of failure during the last digit transfer as the key is released, due in part to the period between the core-clear and sense-pulse.

MOD2026.02: Modification to Clear RK for Recall-Memory

• Problem

When the RM key is pressed, RK is not cleared prior to the transfer. If RK is not already 0, both RK & RM will receive the OR of the values in the two registers.

• Improvement

A new action to clear RA is interjected at the beginning of the RM procedure.

• Implementation

See [logic.8]. The 10.2 pulser is redirected to trigger FSM8 (0=>RK) and a gated pulser added to trigger FSM10 (RK<=>RM) at the end of FSM8. This can all be performed on board 27 without permanent alteration to the board.

Document Log

• 2025 Oct	Initial writing.

—— EOD ——