Repairing HP9800 Desktop Calculators – Part 3

Tony Duell (ard@p850ug1.demon.co.uk)

Memory System Theory of Operation – Continued

4 The HP9830 Memory System

The HP9830 memory system contains the same basic sections as the memory systems of the other machines. However, the memory timing system is a very different design, and is not easy to understand. Let's begin with some of the simpler parts. Perhaps I should add at this point that the first 200 HP9830 machines made had a totally different design of memory system, electrically more like that of the HP9810. Since I have never seen such a machine (and it is possible that none still exist, since they would be converted to the system described here if any part of the memory system failed), I cannot describe such machines in any detail. The description below will cover the type of HP9830 that you are likely to have to repair.

4.1 M register and address decoder

The 16 bit M register consists of U8, U9, U10 and U11 on the **Memory Address** PCB, and follows the same basic design as that in the other machines. It is serially loaded from the processor ALU output (**ALU(0)**) when μ**(27)** is low. This enables the clock to the M register via U26d and U27c. Since the mode input to the register is low, the register shifts towards the LSB, loading the output of the ALU into the MSB position.

To read the registers, the S microcode field contains 1. This causes the output of U9d on the **Memory Data** PCB to go high, This enables the clock to the M register via U27b on the **Memory Address** PCB. This causes the register to parallel-load with its own contents rotated one bit towards the LBS end – the register recirculates are required. The LSB of the register is read into the ALU via the second gate in the AND-OR-invert data selector U8 on the **Memory Data** PCB, enabled by U9d.

Of course these 2 operations can occur simultaneously. Then, the register is loaded as for a simple load operation, but the LSB is read into the ALU via the data selector as described for the read operation.

As usual, the parallel outputs of the M register form the address bus for the memory devices.

4.1.1 ROM addressing

A memory cycle occurs whenever the microcode R field contains either 3 or 6. This is detected by by U4c on the **Memory Address** PCB which asserts **MemEn.**

The first 8K of the 32K word processor address space is decoded into 512-word blocks by U32. Of the 16 outputs, 13 are fed to the main ROM board where they are converted to the required 12V levels and used to enabled pairs of ROM devices. The last 2 outputs are used to select ROMs on an extension ROM PCB that does not include the buffer circuitry, so these signals are converted to 12V on the **Memory Address** PCB by U31b and U31d

As in the HP9820, ROM 1 is split between the first half of block 1 and the first 256 words of the RAM space. The RAM that would have occupied the latter part of the address space is mapped to the second half of block 1. ROM 1 is therefore enabled (by **ROMCS(1)** going low) whenever either **RawCS(1)** and **M(8)** are low or **RawRAMCS(0)**, **M(8)** and **M(9)** are all low. These conditions are detected by U6a, U30b and U6b.

The low 8 bits of the M registers are gated to the ROM address lines by, e.g. U34c whenever a read cycle occurs. Address bit 8 is combined with **RawRAMCS(0)** so that one half of the ROM appears in block 1 and the other half just before the start of the RAM. In both of these areas, **M(8)** is low, so the output of U19f is high. In block 1, **RawRAMCS(0)** is high, so the output of U14c is high and that of U30a is low. In the first RAM area, **RawRAMCS(0)** is low, thus forcing the output of U14c low and that of U309a high, selecting the other half of the ROM. The outputs of these AND gates are inverted by buffers on the **ROM** PCB or **Ext ROM Selector** PCB before being fed to the ROM chips, so that both the A8 and A8/ inputs to each ROM are high when a memory cycle is not occurring, disabling both halves of the ROM.

The second 8K words of the memory map contains the 8 optional ROM modules (3 on internal PCBs, 5 plugged into the **ROM Backplane**). This part of the memory map is decoded by U6 on the **Ext ROM Selector** PCB. The 16 outputs of this decoder are buffered to 12V levels, for example by U5a and its associated components, and then fed to the ROMs in the optional modules.

The **Ext ROM Selector** PCB also contains buffers, such as U1d, for the address lines to these ROMS (and also to the last 2 blocks of standard ROM, which are contained on a similar PCB). As described above, the 2 buffers associated with address bit 8, U3e and U3f, are inverters

4.1.2 RAM Addressing

The third 8K of the processor address space is mostly occupied by RAM. An address in this area is detected by U5b on the **Memory Address** PCB which enables the decoder U17. The outputs of this decoder select individual 1K blocks of RAM. The higher 7 outputs of this decoder are logically ORed with one of two refresh signals, **RfOdd** or **RfEven**, since the RAM is refreshed one half at a time, and then fed to the RAM PCBs.

As you might expect, the lowest block of RAM (enabled by **RAMSel(0)**) is a little more complicated, since part of this RAM appears in ROM block 1. The output of U18a is low if either area is selected and the ROM in that area is not enabled. The output of that gate is logically ORed with **RfEven** by U29c and used to enabled the appropriate RAMs.

The output buffers on the first **RAM** PCB are enabled by **DOE(0)** from U16b when any RAM on that board is accessed. Similarly, U16a asserts **DOE(1)** to enable the buffers on the second **RAM** PCB when its RAMs are accessed.

The RAM address lines fall into 3 groups. **M(0)...M(4)** are fed, along with the outputs of the refresh address counter, into AND-OR-invert data selectors such as U15b, the outputs of which are connected to the address buffers on the **RAM** PCBs. When the **Rfsh** signal is not asserted, the outputs of the M register are fed to the RAM, whereas when **Rfsh** is asserted (during a refresh cycle), the RAM is addressed by the refresh address counter.

The next 3 address bits, **M(5)...M(7)** are simply fed to the address buffers on the **RAM** PCBs.

The last 2 address bits, **M(8)** and **M(9)** are ANDed with the **RawCS(10)** signal by U30c and U30d before going to the RAM buffers. This causes the first quarter of a 1K RAM block to be used in block 1.

Note that the RAM address inputs are 'scrambled', that is M register bit n is not necessarily linked to the RAM address input n. But since this applies both to reading and writing, a given address in the M register will always access the same RAM location, which is all that is important.

4.2 T register

As in the other HP9800 machines, the T register is a 16 bit shift register which communicates serially with the processor. The parallel inputs and outputs of the register are connected to the memory data buses.

The T register is located on the **Memory Data** PCB, and consists of U3, U4, U5 and U6 cascaded in the usual way. The serial input to the T register comes from the AND-OR gate U2.

To load the T register, microcode bit μ**(26)** is low, This enables the third gate of U2 via U1c, and thus the output of the ALU is fed to the T register. **Bitclk** is passed to the register via U9c, enabled by U9a and U7d. Thus the T register is serially loaded from the ALU.

To read the T register, the S microcode field contains 2. This causes the output of U9b (**Ten**) to go high, enabling the second gate in U2. This gate feeds the LSB of the T register back to the serial input, so the register recirculates. The clock is enabled by the output of U7b going low when the S field contains 2. The fourth gate in the AND-OR-invert data selector U8 is enabled by **TEn**, so the LSB is gated to **ALUSerIn** and read into the ALU.

Simultaneous reading and writing of the T register is possible. The clock is enabled as before, and since **Ten** is high, the fourth gate in U8 is enabled and the LSB is read into the ALU. However, since $\mu(26)$ is low, the second gate in U2 is disabled (preventing recirculation of the T register) while the third gate is enabled by U1c. So the MSB of th T register is loaded from the ALU output.

The first gate in U2 causes the T register to recirculate during DMA read operations, while the fourth gate allows it to be loaded from the external DMA device for DMA write operations. Since I have never investigated a DMA device, no more can be said about these functions.

The first gate in the **ALUSerIn** selector , U8, is enabled by the I/O control circuitry to allow the I/O shift register to be read into the ALU. And the third gate forces a 1 into the ALU input when the S field contains 3. This is similar to the other machines in the HP9800 family.

The parallel outputs of the T register are buffered and inverted, e.g. by U12a and fed to the **RAM** PCB inputs. Similarly, the memory PCB data outputs are buffered and inverted, e.g. by U15f and applied to the parallel inputs of the T register.

The mode input to the T register is made high during memory read cycles by U1a, the **TMode** signal is only used for DMA operations and is normally low. Thus for read cycles, **TLd** will copy the data from the selected memory location into the T register. The timing of the T register parallel load will be described in the next section.

4.3 Memory timing

Unlike the other HP9800 machines, the memory timing in the HP9830 is controlled by a state machine rather than a counter and decoder. This circuit is located on the **Memory Address** PCB.

There are 4 state flip-flops. U1a (JK) provides the **RAMCE** signal, U1b (JK) provides **WrTime**, U2a (JK) is the source of **IncRfsh/** and U3b (D type) is an internal state variable. All flip-flop outputs change on the falling edge of **MClk**. The flip-flop control inputs are driven by a 'sea of gates' which takes as its inputs the outputs from the 4 flip-flops and 2 control signals, **Rfsh** which is asserted during a refresh cycle and **R(3)** which, of course, goes low during memory write cycles. These control inputs are kept in a constant state throughout a particular memory cycle.

The flip-flops are held in the reset state by U13d until a memory or refresh cycle occurs. When **MemEn** goes high or **Rfsh/** goes low, the reset pins of the flip-flops are released and the state machine runs.

The **RAMEn** signal, which enables the selection buffers on the **RAM** PCBs is produced by combining 3 of the flip-flop outputs by U24c and U24d.

It is, I think, pointless to attempt to describe the operation of that 'sea of gates' in detail. Practically, if the state machine fails to work correctly – that is, if the observed sequence of states is not the expected one – a logic analyser can be used to see what the inputs to the flip-flops are just before the sequence goes astray, and from that, the faulty gate can be quickly identified.

For example, suppose U1b doesn't toggle when it's expected to in a normal memory cycle. If the output of U3b is 0, then the K input of U1b has to be 1 (if not, then perhaps U3b is defective after all). The J input will be 1 (thus causing the toggle) due to U14b if U3b is 0 (which has just been checked) and if **Rfsh/** is 1 (which we're assuming, since it's not a refresh cycle, but which should be checked anyway) and if the output of U15a is 1. This is effectively an equivalence circuit on U1a and U1b, so if the outputs of those flip-flops should be checked. Therefore, given the correct and incorrect signals, it's clear where the fault must lie.

Therefore I intend to give the sequence of states for the read and write cycles, along with a description of how they affect the rest of the memory system.

4.3.1 Read cycle

The state machine goes through the following sequence :

RAMCE: 0 WrTime: 1 IncRfsh/: 0 U3b: 1 RAMEn: 0

The **WrTime** signal has no effect here, since **RAMCE** is 0, inhibiting U12d.

RAMCE: 0 WrTime: 1 IncRfsh/: 1 U3b: 1 RAMEn: 0

RAMCE: 0 WrTime: 1 IncRfsh/: 1 U3b: 0 RAMEn: 0

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 0

RAMCE: 0 WrTime: 0 IncRfsh/: 0 U3b: 1 RAMEn: 1

The falling edge of **IncRfsh/** has no effect, since the refresh address counter is held reset except when a refresh cycle is occurring. **RAMEn** goes to 1, enabling the control buffers if this a RAM cycle, and deassserting the CE input to ROMs.

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

The **RAMCE** signal goes to a 1, starting the RAM cycle (controlled by delays on the **RAM** PCB).

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 1

RAMCE: 1 WrTime: 1 IncRfsh/: 1 U3b: 0 RAMEn: 1

The output of U12d now pulses high, since **WrTime** and **RAMCE** are both 1. This strobes the data from the selected memory location into the T register.

RAMCE: 1 WrTime: 0 IncRfsh/: 0 U3b: 0 RAMEn: 1

RAMCE: 0 WrTime: 0 IncRfsh/: 0 U3b: 0 RAMEn: 0

The memory devices are disabled and the state machine returns to the initial state.

4.3.2 Write cycle

The **R(3)** input is now low, and the state machine performs the following sequence.

RAMCE: 0 WrTime: 1 IncRfsh/: 0 U3b: 1 RAMEn: 0

RAMCE: 0 WrTime: 1 IncRfsh/: 1 U3b: 1 RAMEn: 0

RAMCE: 0 WrTime: 1 IncRfsh/: 1 U3b: 0 RAMEn: 0

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 0

RAMCE: 0 WrTime: 0 IncRfsh/: 0 U3b: 1 RAMEn: 1

As in the read cycle, the falling edge of **IncRfsh/** has no effect. **RAMEn** goes high, enabling the control buffers for the selected RAM devices.

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

RAMCE: 1 WrTime: 0 IncRfsh/: 0 U3b: 1 RAMEn: 1

The **RAMCE** signal goes high, starting a RAM cycle. This is timed by delays on the **RAM** PCB.

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 1

RAMCE: 1 WrTime: 1 IncRfsh/: 1 U3b: 0 RAMEn: 1

The **WrTime** signal pulses high here. This has no effect on the T register, since the mode input is low. However, U27a is enabled by U26c on a write cycle, so the **RAMWr** signal is asserted, writing the contents of the T register to the selected RAM location. Note that there's one more clock cycle (state) between the RAM cycle starting and data transfer on a write cycle than on a read cycle.

RAMCE: 1 WrTime: 0 IncRfsh/: 0 U3b: 0 RAMEn: 1

RAMCE: 0 WrTime: 0 IncRfsh/: 0 U3b: 0 RAMEn: 0

End of cycle, all flip-flops are cleared.

4.4 RAM refresh

A RAM refresh cycle begins when U7 on the **Memory Address** PCB times out, setting **RfOut** high. This is linked to **RfIn** on the **Main Backplane** which causes the reset signal to be removed from U20a. At the end of the current microinstruction, U20a is set, asserting **Rfsh** and **Wait/** (via U31e). Thus the CPU is halted while the refresh operation takes palace.

The **Rfsh** signal switches the RAM address data selectors so that the lowest 5 RAM address lines are now driven by the refresh address counter. The latter consists of a 4 bit counter U21 and a JK flip-flop U20b wired as a 5 bit synchronous counter. When the **Rfsh/** signal goes low at the start of the refresh cycle, U3a is toggled and either **RfOdd** or **RfEven** is asserted by U4a or U4b. This enables half of the RAM devices in the machine for refreshing, as described above.

The memory timing state machine is enabled by **Rfsh/** forcing the output of U13d high and thus removing the reset signal form the state machine flip-flops. The state machine then repeatedly performs the following sequence (2 cycles are given) :

RAMCE: 0 WrTime: 0 IncRfsh/: 0 U3b: 1 RAMEn: 1

On the very first cycle, although **IncRfsh/** is low it has not *gone* low, so the address is not incremented (it is in all subsequent cycles). **RAMEn** is 1, so the RAM control buffers are enabled

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

Now **RAMCE** goes high, starting a RAM cycle. This is what actually does the refresh.

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 1

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 0

RAMCE: 0 WrTime: 0 IncRfsh/: 0 U3b: 1 RAMEn: 1

Pulse **IncRfsh/** low for one clock cycle. This increments the refresh address counter.

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 1 RAMEn: 1

Refresh the next RAM location

RAMCE: 1 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 1

RAMCE: 0 WrTime: 0 IncRfsh/: 1 U3b: 0 RAMEn: 0

Round again, increment the address, and refresh some more

Since **Rfsh/** is low, U14b is inhibited and thus the state machine flip-flop U1b can never be set, so **WrTime** remains low (this can also be seen from the state table above). Thus neither **TLd** or **RAMWr** are ever asserted, so no data transfers take place during the refresh cycle. It is the action of accessing RAM the RAMs that refreshes them.

At the end of the refresh cycle, the 5 bit address counter rolls over from 11111 to 00000. The falling edge of the output of U20b triggers the monostable U7, which clears U20a, ending the refresh cycle. The machine runs normally until U7 times out again, starting another refresh cycle. Of course in the next refresh cycle U3a will be in the other state, so the other half of the RAM is refreshed.

4.5 DMA flip-flop

This flip-flop, U2b on the **Memory Address** PCB operates in a similar way to the corresponding flip-flop in the HP9820 memory system. When a memory cycle occues, the reset input of U2b is de-asserted, and since both inputs of U4d are high, **DisμClk** goes low, inhibiting the microcode clock.

If a DMA cycle is not occuring, **EMB/** is high. Therefore, when U1a is set during the memory cycle, the output of U27d goes high, setting U2b. This re-eenables the microcode clock before the start of the next microinstruction.

If a DMA device pulls **EMB/** low, U2b is not set, the mcirocode colock remains inhibited. Thus no further microinstructions can be exectued until the DMA operation ends and **EMB/** goes high again.

4.6 Memory PCBs

The HP9830 memory PCBs are a little more complicated than those used in the HP9810 or HP9820 since they contain buffer circuitry as well as the memory

devices themselves.

4.6.1 Main ROM PCB

The **ROM** PCB contains 7K words of firmware stored in 28 512*8 ROM devices. The address inputs to the board are buffered, e.g. by U5d, and as explained about the buffers for **ROMAddr(8)** and **ROMAddr(8)/**, U3e and U3f are inverters. The **RAMEn** memory timing signal is inverted by U3d and converted to 12V levels, and fed to the CE input of the ROMs (which is thus deasserted par way through the memory cycle.

Selection signals from the decoder circuitry on the **Memory address** PCB are converted to 12V levels by buffers on the **ROM** PCB, such as U1d. They are then applied to pairs of ROM devices.

The data outputs from the ROMs are buffered by open-collector buffers such as U10c, and then fed to the inputs of the **Memory Data** PCB, and thus into the T register.

4.7 Internal extension ROM PCBs

Up to 3 **Internal Ext ROM** PCBs can be fitted to the HP9830, while the last part of the standard BASIC firmware is stored in ROMs on a PCB that is electrically identical. These PCBs contain up to 2 pairs of 512*8 ROMs along with open-collector data buffers such as U1d. The address and control signals for these ROM are mostly buffered on the **Ext ROM Selector** PCB, although the 12V level conversion for the selection inputs to the last part of the standard firmware is performed on the **Memory Address** PCB.

These ROM PCBs contain open-collector data buffers such as U1d between the outputs of the ROMs and the inputs of the **Memory Data** PCB.

4.7.1 ROM modules

The **ROM Backplane**, accessible via a door on the left hand side of the machine, accepts up to 5 **ROM Modules**. These modules simply contain 2 pairs of 512*8 ROM chips, all the necessary buffering being built into the HP9830 itself.

The address lines are buffered by the same buffers on the **Ext ROM Selector** PCB that are used for the **Internal Ext ROM** boards. Similarly, the 12V-level selection signals are generated by circuitry on the **Ext ROM Selector** in the usual way.

The data outputs from the **ROM Modules** are buffered by open-collector buffers such as U2d on the **ROM Backplane** before being fed to the **Memory Data** PCB.

4.7.2 RAM PCBs

The HP9830, like the other HP9800 machines, used 1103 1K*1 DRAM devices. Unlike the other machines in the family, the conversion between 5V TTL levels and the 16V logic levels of these PMOS RAMs is performed on the **RAM PCB** itself. Each RAM PCB contains up to four 1K word blocks of RAM, (a total of 64 RAM devices), the machine has backplane slots for 2 boards.

The 10 incoming address lines from the **Memory Address** PCB are converted to 16V levels by inverter/transistor circuits, such as that consisting of U5f and Q1. The outputs of the **Memory Data** PCB are inverted and converted to the PMOS logic levels by open-collector buffers such as U10a, pulled up to 16V. The inversion in these buffers compensates for the fact that the RAM device data output is inverted with respect to its input.

The conversion of the RAM data outputs to TTL levels is performed by special sense amplifier devices, type 3208 (U11 U9 and U7). The outputs of these are enabled by the **DOE/** signal from the **Memory Address** PCB whenever any RAMs on the PCB are accessed.

The control inputs to the RAMs are provided by the 3207 driver ICs, U1, U2 and U3. Half of such an IC is used to provide the **CE/** and **Precharge** inputs to a given set of 16 RAM devices (storing 1K words of data). I shall describe the circuitry that is controlled by **RAMSel(0)**, the other 3 circuits operate identically.

When a memory cycle accessing this area of RAM occurs **RAMSel(0)** is asserted by the address decoder circuitry and **RAMEn** is asserted by the memory timing state machine. Together, these 2 signals enable U1a and U1b. When **RAMCE** is set high by the memory timing state machine, the **CE0/** select input to that set of RAMs is asserted (made low, the 3207 being an inverting buffer).

The transistor Q3 is normally saturated, thus charging C1. When either **CE0/** or **CE1/** goes low, Q3 is cut off by the appropriate diode. C1 thus discharges, asserting the precharge signal via U1b shortly after **CE0/** is asserted. This meets the timing requirements of the 1103 DRAM.

When a RAM write cycle occurs, **RAMWr** is asserted by U27a on the **Memory address** PCB. This enables U1c, U1d, U3a and U3b on the **RAM**

PCB. The buffer corresponding to the selected RAM area thus asserts the **Wr** signal to those RAMs.

4.8 Test connectors

The H9830 memory system has test connectors on the **Memory Address**, **Memory Data** and **RAM** PCBs. The connector on the top of **Memory Address** PCB carries the outputs of the M register, the **MemEn** and **RfEven/** signals and the connections for the timing capacitor of the refresh start monostable. The connector on the top of **Memory Data** PCB carries the outputs of the T register. These connectors can be used in a similar way to the test connector on the HP9810 or HP9820 **Memory Backplane**

The connector on the side of the **RAM** PCB carries the RAM address and control signals (at 16V PMOS levels) and the reference voltage for the RAM output sense amplifiers. But since this connector is inaccessible when the PCB is fitted into the machine, it is of very limited use.

4.9 How the memory timing state machine was figured out, or 'CAH'

This section is optional. You don't need to understand this to be able to repair the HP9830. However, you may find it interesting.

It is relatively easy – in theory – to work out the sequence of a state machine such as the HP9830 memory timing circuit, where all the flip-flops change at the same time and where the control inputs are kept in the same state throughout the period of interest (in this case a memory cycle). All you have to do is :

1. Start with the flip-flops in the known starting state (in this case, all cleared)
2. Work out the states of the outputs of all the gates in the 'sea of gate' and thus determine the states of the inputs to the flip-flops
3. Work out the new states of all the flip-flops
4. Go round again, and repeat until you've worked out enough states

Notice that I said 'in theory'. In practice, it's very easy to make a mistake when doing this by hand. And therefore I did something I call 'Calculator Aided Hackery' (or *CAH*).

I used an HP handheld calculator, an HP49G, to keep track of all the logic states and effectively to simulate the state machine. The HP49G is ideal for this sort of work (OK, an HP48 or an HP50G would probably do as well) because :

- It's interactive, I can try things and see the effect
- It is easy to program
- It has a good range of boolean functions
- It's small enough to go to the problem, it will fit on my workbench along with the HP9830 and electronic test instruments.
- The large ('infinite') stack means i can keep as many logic states on the stack as I want to without worrying
- I can creates lists of tagged objects giving the new states, I don;t have to remember what I've stored in a numbered register or similar
- I can connect it to my desktop computer and upload the results, avoiding typographical errors.

The program is relatively simple. A set of named variables holds the states of the flip-flops and the control inputs :

```
U1a : 0
U1b : 0
U2a : 0
U3b : 0
NWr : 1
RFSH : 0
```

There's a program to initialise the system by clearing all the 'flip-flops' and setting the control inputs appropriately. The latter can be changed manually to investigate all types of RAM cycles.

```
INIT :  \<< 0 'U1a' STO 0 'U1b' STO 0 'U2a' STO 0 'U3b' STO
0 'RFSH' STO 1 'NWr' STO \>>
```

And one to display the current states as a list of tagged objects.

```
DSP :  \<< U1a "RAMCE" \->TAG U1b "WrTime" \->TAG
U2a "IncRfsh/" \->TAG U3b "U3b" \->TAG
RAMEn "RAMEn" \->TAG 5 \->LIST \>>
```

The **RAMEn** signal is calculated in the obvious way from the states of the flip-flops :

```
RAMEn : \<< U3b U1b NOT AND NOT U1a NOT AND NOT \>>
```

The process of simulating the state machine is quite simple. What we do is to calculate the new states of the flip-flops and store them on the calculator stack. Then when all have been calculated, we store the new states into the flip-flop variables. That storing corresponds to all the flip-flops changing on the falling edge of **MClk**.

A D-type flip-flop is easy to simulate. We just calculate the state of the D input, and to update it we store that value in the flip-flop variable. A JK is a little harder, here's a program which takes the states of J, K, and the old value of the output from the stack and returns the new value of the output :

```
JK : \<< \-> j k q
\<< j k XOR DUP j AND SWAP NOT j q XOR AND OR
\>> \>>
```

This is perhaps a little hard to understand. What it does is say that if the J and K input are in opposite states (corresponding to the 'set' or 'reset' functions), the new output is given by the J input. If they're in the same state, then the new output is either unchanged, or the opposite of the current output, which can be calculated by XORing the old output with either J or K (J is used here).

Then we need a set of programs to calculate the flip-flop inputs, basically the program equivalent of the 'sea of gates' :

```
U1aJ: \<< U1b NOT U3b AND U2a AND \>>

U1aK : \<< U2a NOT RFSH OR U3b NOT AND \>>

U1bJ : \<< RFSH NOT U3b NOT AND U2a U1a XOR NOT AND \>>

U1bK : \<< U3b NOT \>>

U2aJ : \<< U3b \>>

U2aK : \<< U1a U1b OR NWr RFSH OR U3b AND OR U1b U1a AND NOT
AND NOT \>>

U3bD : \<< U3b U2a NOT AND U1a NOT U1b NOT AND OR \>>
```

Putting it all together, here's a program which first calculates the new states of the flip-flops, then stores them, and produces a list of the new states on the stack :

```
NXT : \<< U1aJ U1aK U1a JK U1bJ U1bK U1b JK U2aJ U2aK U2a JK
U3bD 'U3b' STO 'U2a' STO 'U1b' STO 'U1a' STO DSP \>>
```

And since memory cycles are 12 clock cycles long (due to the microinstructions that activate them), working out the next state 12 times in succession is useful too :

```
CYCLE : \<< 1 12 START NXT NEXT 12 \->LIST \>>
```

And that's basically it. Initialise the system, set the control inputs appropriately, and execute **CYCLE**.

Of course these programs have other uses too. If you have an HP9830 memory system that is going astray, store the states of the flip-flops just before it goes wrong in the variables. The 'sea of gates' programs will then tell you what the flip-flops inputs should be. And this can be compared to what your logic analyser is telling you that they *are*. Of course this makes it relatively easy to find the fault.

Relax...The worst is over. The remaining sections of the machine are much easier to understand.